Proof of Theorem 1

We begin by establishing a fundamental property of the search order.

Lemma 1. *The sequence of extracted rule-apex-realization pairs has monotonically non-decreasing f_1-values.*

Proof. The rule-apex-realization pair extracted from OPEN has the smallest f_1-value due to the lexicographic ordering. Also, any newly generated pair has an f_1-value that is greater than or equal to that of its parent pair, and the resulting pair of a merge can not have an f_1-value smaller than both of its merged pairs (their rule apexes). Thus, the sequence of extracted pairs has monotonically non-decreasing f_1-values.

The next two lemmas justify the correctness of pruning based on previously expanded nodes.

Lemma 2. *If there exists a rule-apex-realization pair $\mathcal{RP}'$ in $G^{=}_{\mathrm{cl}}(s(\mathcal{RP}))$ the truncated $\mathbf{f}$-value of which weakly rule-dominates the truncated $\mathbf{f}$-value of some realization-pair $\mathcal{RP}$ on Line 7 of the Algorithm 5, then the $\mathbf{f}$-value of $\mathcal{RP}'$ weakly rule-dominates the $\mathbf{f}$-value of $\mathcal{RP}$.*

Proof. The rule-apex-realization pair $\mathcal{RP}'$ has been expanded and added to $G^{=}_{\mathrm{cl}}(s(\mathcal{RP}))$ before $\mathcal{RP}$ was extracted from OPEN. Let

$$\mathcal{RP}_0 = \mathcal{RP}', \mathcal{RP}_1, \mathcal{RP}_2, \ldots \mathcal{RP}_k = \mathcal{RP}$$

be the sequence of rule-apex-realization pairs extracted since then. Because $\mathcal{RP}'$ is still in $G^{=}_{\mathrm{cl}}(s(\mathcal{RP}))$, the Line 1 of the Algorithm 5 never held. According to Lemma 1 and since the heuristic function is consistent, it follows that $f_1(\mathcal{RP}_i) = f_1(\mathcal{RP}_{i+1})$ for all $i < k$. Thus, $f_1(\mathcal{RP}') = f_1(\mathcal{RP})$ which along with the premise in the lemma implies $\mathbf{f}(\mathcal{RP}') \lesssim_{\mathcal{R}} \mathbf{f}(\mathcal{RP})$.

Lemma 3. *If there exists a rule-apex-realization pair $\mathcal{RP}'$ in $G^{<}_{\mathrm{cl}}(s(\mathcal{RP}))$ the residual $\mathbf{f}$-value of which weakly rule-dominates the residual $\mathbf{f}$-value of some rule-apex-realization pair $\mathcal{RP}$ on Line 9 of the Algorithm 5, then it also holds that the $\mathbf{f}$-value of $\mathcal{RP}'$ weakly rule-dominates the $\mathbf{f}$-value of $\mathcal{RP}$.*

Proof. Since $\mathcal{RP}'$ was previously expanded and added to $G^{<}_{\mathrm{cl}}(s(\mathcal{RP}))$, the condition on Line 1 of the Algorithm 5 held at least once. According to Lemma 1 and because the heuristic function is consistent, it holds that $f_1(\mathcal{RP}') < f_1(\mathcal{RP})$. Thus, the $\mathbf{f}$-value of $\mathcal{RP}'$ weakly rule-dominates the one of $\mathcal{RP}$.

We then turn to the correctness of the solution-set maintenance.

Lemma 4. *Let $\mathcal{RP} = \langle \mathbf{r}, x \rangle$ be a rule-apex-realization pair and y be any realization such that $\mathbf{r} \lesssim_{\mathcal{R}} \mathbf{c}_{\mathcal{R}}(y)$. If we merge $\mathcal{RP}$ with another rule-apex-realization pair then the rule apex of the resulting rule-apex-realization pair will still weakly rule-dominate ($\lesssim_{\mathcal{R}}$) y.*

Proof. The rule apex of the resulting rule-apex-realization pair is the componentwise minimum of the two merged rule apexes.

The next lemma shows that $\boldsymbol{\varepsilon}$-rule-dominance satisfies one-sided transitivity with respect to exact rule-dominance. In other words, approximate dominance is preserved when composed with exact dominance. This property allows RA*pex to reason about sets of realizations using rule-apex-realization pairs and is crucial for establishing the algorithm's approximation guarantees. Specifically, for a given ε-bounded rule-apex-realization pair $\mathcal{RP} = \langle \mathbf{r}, x \rangle$, it allows the representative realization x to $\boldsymbol{\varepsilon}$-rule-dominate all the realizations that the pair represents.

Lemma 5. *Let x, y, z be realizations such that $x \lesssim^{\boldsymbol{\varepsilon}}_{\mathcal{R}} y$ and $y \lesssim_{\mathcal{R}} z$, then $x \lesssim^{\boldsymbol{\varepsilon}}_{\mathcal{R}} z$.*

Proof. Let $\mathcal{R} = \langle R, \leq \rangle$ be a rulebook with $R = \{r_1, \ldots, r_N\}$. Assume $x \lesssim^{\boldsymbol{\varepsilon}}_{\mathcal{R}} y$ and $y \lesssim_{\mathcal{R}} z$. To show that $x \lesssim^{\boldsymbol{\varepsilon}}_{\mathbb{R}} z$, suppose there is a rule $r_i \in R$ such that $r_i(x) > (1 + \varepsilon_i) r_i(z)$. We need to show that there exists a more important rule $r_j > r_i$ such that $r_j(x) < (1 + \varepsilon_j) r_j(z)$. Consider two possibilities.

- $r_i(x) \neq (1 + \varepsilon_i) r_i(y)$. If $r_i(x) > (1 + \varepsilon_i) r_i(y)$, then $x \lesssim^{\varepsilon}_{\mathcal{R}} y$ guarantees that there exists a rule $r_k > r_i$ such that $r_k(x) < (1 + \varepsilon_k) r_k(y)$. If instead $r_i(x) < (1 + \varepsilon_i) r_i(y)$, then we can set $r_k = r_i$. Thus, in either case, there exists $r_k \geq r_i$ such that $r_k(x) < (1 + \varepsilon_k) r_k(y)$. We are done if $r_k(y) \leq r_k(z)$, since we can take $r_j = r_k$. Otherwise, $r_k(y) > r_k(z)$ and $y \lesssim_{\mathcal{R}} z$ imply that there exists $r_l > r_k$ such that $r_l(y) < r_l(z)$. Again, we are done if $r_l(x) \leq (1 + \varepsilon_l) r_l(y)$, and if not, there has to be some rule $r_m > r_l$ such that $r_m(x) < (1 + \varepsilon_m) r_m(y)$. Repeating this argument construncts an increasing chain $r_i \leq r_k < r_l < r_m \ldots$. Since R is finite, the chain has to stop, which is only possible if there exists $n \in \{1, \ldots, N\}$ such that $r_n(x) < (1 + \varepsilon_n) r_n(z)$.
- $r_i(x) = (1 + \varepsilon_i) r_i(y)$. Then $r_i(x) > (1 + \varepsilon_i) r_i(z)$ implies that $r_i(y) > r_i(z)$. Because $y \lesssim_{\mathcal{R}} z$, there must exist a rule $r_k > r_i$ such that $r_k(y) < r_k(z)$. We are done if $r_k(x) \leq (1 + \varepsilon_k) r_k(y)$, since one can take $r_j = r_k$. Otherwise, there must exist a rule $r_l > r_k$ such that $r_l(x) < (1 + \varepsilon_l) r_l(y)$. Applying the same reasoning as in the previous case, we again obtain an increasing chain $r_i \leq r_k < r_l \ldots$ which must terminate at some r_n with $r_n(x) < (1 + \varepsilon_n) r_n(z)$.

Using the above results, we now relate the behavior of RA*pex to arbitrary optimal solutions.

Lemma 6. *For any prefix $x_l = [s_1, s_2 \ldots s_l]$ of any solution*

$$x = [s_1 (= s_{start}), s_2 \ldots s_L (= s_{goal})]$$

*with $1 \leq l \leq L$, when the algorithm terminates, there exists either (*Case 1:*) an expanded rule-apex-realization pair $\mathcal{RP}$ (that is, one that reaches Line 9 of the Alg 1) that ends at the state s_l and whose rule apex weakly rule-dominates the $\mathbf{g}$-value of the realization x_l or (*Case 2:*) a rule-apex-realization pair $\mathcal{RP}$ in the solution set such that the $\mathbf{f}$-value of its representative realization $\boldsymbol{\varepsilon}$-rule-dominates the $\mathbf{f}$-value of the realization x_l.*

Proof. The proof is done via induction. The lemma holds for $l = 1$ and any solution since the realization pair $\mathcal{RP} = \langle \mathbf{0}, [s_{\text{start}}] \rangle$ gets expanded and has the properties required for Case 1. Now assume that the lemma holds for some $l < L$ and any solution. Then, we prove that it is also true for $l + 1$ and this solution.

Assume that Case 1 holds for l and consider both $\mathcal{RP}$ - the realization-pair mentioned there and its potential child realization pair $\mathcal{RP}'$ created on Line 14 of the Algorithm 1 for $s' = s_{l+1}$. The realization pair $\mathcal{RP}'$ ends at the state s_{l+1} and its rule apex weakly rule-dominates the **g**-value of the realization x_{l+1}, which implies that its **f**-value weakly rule-dominates the **f**-value of the realization x_{l+1}. We distinguish three cases:

1. First, the condition on Line 11 in Alg 5 (or Line 1 in the Alg 2) holds for some realization pair in the solution set, meaning, the **f**-value of the representative path of this realization pair $\boldsymbol{\varepsilon}$-rule-dominates the **f**-value of the realization pair $\mathcal{RP}'$. The algorithm replaces this realization pair with a new realization-pair $\mathcal{RP}''$ in the solution set on Line 13 in the Alg 5 (or Line 3 in the Alg 2). The pair $\mathcal{RP}''$ stays in the solution set but the algorithm might merge it several times with other rule-apex-realization pairs on Line 2 of Algorithm 4 before it terminates. The rule apex of the pair $\mathcal{RP}''$ weakly rule-dominates the **f**-value of the realization x_{l+1} (since this rule apex is the component-wise minimum of the **f**-value of the pair $\mathcal{RP}'$ and another rule apex, and hence weakly rule-dominates the **f**-value of the pair $\mathcal{RP}'$, which in turn weakly rule-dominates the **f**-value of the realization x_{l+1}) and merging it with other realization-pairs does not change this property according to Lemma 4. Since the pair $\mathcal{RP}''$ also remains $\boldsymbol{\varepsilon}$-bounded, the **f**-value of its representative path always $\boldsymbol{\varepsilon}$-rule-dominates the **f**-value of itself, which equals its rule-apex. Put together, according to Lemma 5 the **f**-value of its representative path $\boldsymbol{\varepsilon}$-rule-dominates the **f**-value of the realization x_{l+1}. Thus, the merged realization pair satisfies Case 2 for $l + 1$.
2. Second, the condition on Line 7 or Line 9 of the Alg 5 (or Line 5 in the Alg 2) holds , meaning, there exists a rule-apex-realization realization pair $\mathcal{RP}^{=}$ in $G^{=}_{\text{cl}}(s(\mathcal{RP}'))$ the truncated **f**-value of which weakly rule-dominates the truncated **f**-value of the rule-apex-realization pair $\mathcal{RP}'$ or there exists a rule-apex-realization pair $\mathcal{RP}^{<}$ in $G^{<}_{rmcl}(s(\mathcal{RP}'))$ the residual **f**-value of which weakly rule-dominates that of the pair $\mathcal{RP}'$. In either case, the expanded pairs $\mathcal{RP}^{=}$ and $\mathcal{RP}^{<}$ ending at the state s_{l+1} have **f**-values that weakly rule-dominate the **f**-value of the pair $\mathcal{RP}'$ according to Lemmas 2 and 3 respectively. Thus, their rule apex weakly rule-dominate the rule apex of the pair $\mathcal{RP}'$. Thus, either $\mathcal{RP}^{=}$ or $\mathcal{RP}^{<}$ satisfies Case 1 for $l + 1$ since the rule apex of the pair $\mathcal{RP}'$ in turn weakly rule-dominates the **g**-value of the realization x_{l+1}.
3. Otherwise, the algorithm executes Line 17 of the Alg 1 for the rule-apex-realization pair $\mathcal{RP}'$, where the pair is inserted into OPEN, perhaps after having been merged with another realization-pair in the Algorithm 4. The algorithm might merge it several more times before finally extracting it. Its rule apex weakly rule-dominates the **g**-value of the realization x_{l+1} and

merging it with other realization pairs does not change this property according to Lemma 4. Thus, if this realization pair is expanded, it satisfies Case 1 for $l+1$. If it is extracted but not expanded, then the pruning conditions in the Alg 5 (or the Alg 2) hold, and as we have already proved, Case 1 or Case 2 holds.

Assume that Case 2 holds for l and consider the rule-apex-realization pair mentioned there. The **f**-value of the representative realization of this pair $\boldsymbol{\varepsilon}$-rule-dominates the **f**-value of the realization x_l. Since the heuristic function is consistent, the **f**-value of the realization x_l in turn weakly rule-dominates the **f**-value of the realization x_{l+1}. Thus, according to Lemma 5, this realization-pair satisfies Case 2 for $l+1$.

Finally, we combine these results to prove Theorem 1.

Proof of Theorem 1. Lemma 6 holds for prefix $x_L = x$ of any solution x. In case its Case 2 holds, the theorem holds by definition for realization x since the **f**-values of the solutions are equal to their costs. In case its Case 1 holds, consider the rule-apex-realization pair as mentioned. This pair ends at the the goal state, and the algorithm thus executed the Line 11 in Alg 1 for it, where the pair was inserted into the solution set, maybe after having been merged with another rule-apex-realization pair on Line 2 of the Algorithm 4. The pair stays in the solution set, and might be merged several more times with other rule-apex-realization pairs before the algorithm terminates. The rule apex of the pair weakly rule-dominates the **g**-value of the path x according to Lemma 6 and merging it with other pairs does not change this fact according to Lemma 4. Since the pair also remains $\boldsymbol{\varepsilon}$-bounded (due to the conditions on Line 3 in Alg 4, Line 11 in Alg 5, Lemma 1, and the consistency of the heuristic function), the **f**-value of its representative realization always $\boldsymbol{\varepsilon}$-rule-dominates the **f**-value of itself, which equals its rule apex. Putting it all together, the **f**-value of its representative realization $\boldsymbol{\varepsilon}$-rule-dominates the **g**-value of the realization x according to Lemma 5. Thus, the theorem holds by definition for the realization x since the **g**- and **f**-values of the solutions are equal to their costs.

Formal Specification and Control Synthesis of Autonomous Robots Using Rulebooks

Tichakorn Wongpiromsarn[1], Konstantin Slutsky[2], and Emilio Frazzoli[3]

***Abstract*—This paper presents a formal specification framework for planning and control of autonomous robots, focusing on the challenge of managing complex trade-offs among multiple, potentially conflicting objectives. These include hierarchical relationships and non-comparable objectives, some of which may be too complex to be captured by standard additive cost functions. We leverage the *rulebook* formalism to represent such objectives and their relationships and formulate two control synthesis problems: single-strategy synthesis, which seeks one optimal strategy, and complete synthesis, which computes the full set of optimal strategies with respect to a rulebook, analogous to the Pareto front in multi-objective planning. We show that our formulation generalizes existing temporal logic-based and optimization-based planning and control, providing a unifying framework across robotics, formal methods, control theory, and operations research. For single-strategy, we identify tractable subclasses and present a polynomial-time algorithm that accommodates richer combinations of objectives than prior work. For complete synthesis, we introduce an algorithm to compute all optimal solutions and analyze its computational complexity. In both cases, we present case studies that include complex multi-objective planning problems and demonstrate the practical effectiveness of our approach compared to existing methods.**

***Index Terms*—Formal methods in robotics and automation, motion and path planning, graph search-based path planning**

I. Introduction

Autonomous robots are often required to satisfy or optimize multiple complex objectives simultaneously. For example, self-driving cars and unmanned aerial vehicles must avoid collisions, comply with regulatory requirements (e.g., staying within designated lanes or airspace and maintaining a sufficient clearance from obstacles), and optimize for performance metrics such as travel time and energy efficiency [1]–[3]. In many situations, however, these objectives may conflict and cannot be satisfied simultaneously, e.g., in the trolley problem, where the system must choose between undesirable outcomes. Unlike semi-autonomous robots that operate under close human supervision and can rely on human intervention in such complex situations, a fully autonomous robot must make timely decisions and take action on its own, often before human intervention is possible, to minimize harm. This need is explicitly acknowledged in regulations such as §107.21 in Title 14 of the Code of Federal Regulations [4]. Applications such as agricultural spraying with toxic chemicals further illustrate the challenge where robots are required to complete tasks that inherently pose safety risks. These challenges necessitate the development of a principled framework that can formally specify such diverse and potentially conflicting objectives and guide planning strategies to prioritize critical objectives while managing unavoidable trade-offs.

Variants of temporal logics provide formalisms for precisely describing complex objectives or specifications of autonomous robots due to their expressiveness [5], [6]. Correct-by-construction synthesis of control strategies with respect to temporal logic specifications has been extensively studied across various settings, including deterministic [7]–[13], reactive [14]–[16], and probabilistic [17]–[22]. Recent languages such as Metric Temporal Logic [23], robust Linear-time Temporal Logic [24], and Signal Temporal Logic (STL) [25]–[33] include robust semantics and allow quantitative assessment of the robustness of satisfaction. Various robustness metrics have been introduced [34]–[37]. These formalisms, however, do not capture the unequal importance of different objectives. Since all objectives are combined into a single formula, they are implicitly treated as equally important. As a result, these formalisms cannot distinguish between violations of high-priority objectives (e.g., collision avoidance) and those of lower-priority (e.g., signaling before changing lanes).

Traditional approaches to handle multiple objectives often aggregate all objectives into a single scalar function using a weighted sum [38]–[41]. Each objective is assigned a weight reflecting its relative importance and the robot's task is to optimize this aggregate. While this formulation enables the use of standard optimization techniques and can address a wide range of problems, it becomes problematic in scenarios where objectives follow a strict hierarchy, e.g., prioritizing safety over rule compliance and rule compliance over performance optimization. To ensure that a higher-priority objectives are never compromised in favor of lower-priority ones, a lexicographic ordering needs to be imposed such that lower-priority objectives are only taken into account when all higher-priority ones are optimized. Such strict prioritization cannot be represented by any finite set of weights, as an infinitesimal violation of a high-priority objective could otherwise be outweighed by large improvements in lower-priority ones.

A common approach to address this limitation is to treat critical objectives as hard constraints and the remaining objectives as soft constraints or costs to be minimized. This leads to a constrained optimal control problem, where the solution represents the robot's optimal strategy [42], [43]. Model predictive control (MPC) is a widely used approach for solving such problems in real time [44]. Control Lyapunov functions (CLFs) and control barrier functions (CBFs) have been introduced to encode stability and hard constraints that can be formulated as the superlevel set of a smooth

Received 11 August 2025; revised 26 November 2025; accepted 26 January, 2025. This work was supported in part by NSF under Grant CNS-1446151, CNS-2141153, and DMS-2153981. This article was recommended for publication by Editor J. O'Kane upon evaluation of the reviewers' comments. (Corresponding author: T. Wongpiromsarn)

T. Wongpiromsarn is with the Department of Computer Science, Iowa State University, Ames, IA 50011, nok@iastate.edu

K. Slutsky is with the Department of Mathematics, Iowa State University, Ames, IA 50011, kslutsky@iastate.edu

E. Frazzoli is with the Institute for Dynamic Systems and Control, ETH Zürich, Zürich, Switzerland, efrazzoli@ethz.ch

function. Together, they provide a unified optimization-based framework that integrates both stability and safety requirement and reduce the control synthesis problem to a quadratic program (QP) that can be solved efficiently at each time step [45]–[47].

A key limitation of constrained optimal control formulations is their restriction to a two-level priority structure, with hard constraints at the higher level and soft constraints at the lower level. Multi-objective MPC has been investigated. For example, [48] provides conditions under which Pareto optimal solutions can be selected to ensure closed-loop stability, while [49] considers lexicographic multi-objective formulations. However, these formulations are typically limited to additive objectives (i.e., the sum of stage costs over the horizon plus terminal costs), which can limit their ability to capture non-additive objectives, such as the maximum stage cost.

The multi-objective shortest path (MOSP) problem has also been studied extensively in operational research and computer networks [50], [51]. Here, the objectives are treated independently and the goal is to compute Pareto optimal solutions that represent trade-offs among conflicting objectives. Unlike optimal control problems, which typically involve continuous-state systems described by differential equations, MOSP focuses on discrete-state systems modeled as graphs. Both exact and approximate algorithms have been developed to solve the MOSP problem [51]–[58].

Autonomous robots operating in complex real-world environments often have to handle complex trade-offs and relationships among objectives. Some objectives may be too complex to be captured by simple additive cost functions. Moreover, while certain objectives naturally form a strict hierarchy (e.g., prioritizing collision avoidance over lane keeping), others (e.g., obstacle clearance and lane keeping) may be important but not directly comparable in terms of priority.

To handle complex objectives and their diverse relationships, we employ the *rulebook* formalism introduced in [59], which provides a principled way to specify objectives and their relative priorities. This formalism was originally developed to compare robot behaviors based on externally observable outcomes. At the core of this framework is the concept of ordering rules based on their relative importance, where each rule captures a distinct objective. Its semantics imposes a preference ordering over possible robot behaviors.

This paper extends the application of rulebooks beyond behavior comparison, showing that rulebooks can serve as a general specification language for expressing objectives and constraints in robot planning and control. While [59] focuses on ranking behaviors, we leverage rulebook semantics to formally define acceptable behaviors and provide algorithms to synthesize control strategies that satisfy these specifications. Despite its potential, the use of rulebooks in this context remains largely unexplored. To our knowledge, the only prior work in this direction are [60], [61], both of which consider restricted forms of rulebooks with the primary focus on computing a single optimal strategy. Specifically, [60] assumes a total order over rules, resulting in a strict lexicographic hierarchy among objectives and further requires each rule to be a differentiable function. Under these assumptions, the rulebook is reduced to a single utility function via a weighted sum and the optimal solution with respect to this function converges to the rulebook-optimal solution as the ratios between successive weights approach infinity. On the other hand, [61] assumes a total preorder over rules and restricts rules to Boolean formulas that must hold at all times. Our work departs from prior literature by focusing on establishing rulebooks as a general and expressive specification language for planning and control, rather than tailoring them to specific optimization settings. Our goal is to support a broad class of objectives and constraints along with rich prioritization structures. To achieve this, we consider the most general form of rulebooks defined as arbitrary preorders, allowing both hierarchical and non-comparable relationships among rules. Building on this foundation, we introduce weak assumptions on the rules under which computing an optimal solution remains tractable. Furthermore, we go beyond extracting a single optimal behavior and develop a complete synthesis procedure that characterizes the entire set of rulebook-consistent behaviors. This enables us to address the verification problem, i.e., determining whether a given behavior is consistent with a rulebook, which is fundamental for using rulebooks as a formal specification language but has not been handled in prior work.

Our primary contributions are as follows.

(1) In Section III, we formulate control synthesis with rulebooks, considering two variants: single-strategy control synthesis, which computes one optimal control strategy, and complete control synthesis, which computes the full set of optimal strategies (analogous to the Pareto front in MOSP). Then, in Section IV, we show that these formulations subsume temporal-logic-based and optimization-based planning and control as special cases. As a result, they provide a unified framework that bridges theoretical foundations from diverse fields, including formal methods, control theory, optimization, and operations research, to define and compute optimal robot's strategies. Moreover, the explicit structure and semantics of rulebooks offer a principled language for specifying and reasoning about robot behavior, independent of the underlying robot model, making it well-suited for communication among diverse stakeholders, including system designers, engineers, and regulatory bodies.

(2) In Section V, we identify conditions on the rules that allow the single-strategy control synthesis problem to be solved efficiently. This follows a common practice in the literature of introducing tractable subclasses such as limiting temporal logic to GR(1) fragments in reactive synthesis [62], [63] or limiting cost functions to specific forms like quadratic costs in control [45]–[47]. However, instead of reducing rulebooks to existing formulations, we directly formulate assumptions on the rules themselves and provide a polynomial-time algorithm under these assumptions. This approach supports a broader class of objectives, including combinations of non-additive, non-differentiable objectives and temporal logic specifications, which has not been addressed in prior work, as demonstrated in the case study in Section V-D.

(3) In Section VI, we introduce an algorithm to solve complete control synthesis, which is essential to answer the verification question. Unlike traditional formal methods that offer a binary correctness criterion, rulebooks define a preference structure without explicitly labeling acceptable behaviors. As a result, determining whether a given robot behavior is consistent with the rulebook requires comparing its objective values to those of optimal ones.

While the control synthesis algorithms we build on are

grounded in existing techniques, our key contribution lies in how we develop and integrate them under a single, unified framework based on rulebooks, which not only generalizes and unifies several prior approaches, but also supports novel combinations of specifications and objectives that are difficult to express or solve using existing formulations. We use the term "control" in a broad sense that also encompasses what is often referred to as "planning" in literature, namely, the generation of control inputs or actions that guide a robot toward desired behaviors specified by given objectives. We review related work in Section IV and discuss how our formulation generalizes and unifies several existing formulations. All experiments in this paper were conducted using the toolbox available at https://github.com/tichakornw/planning, on a Dell XPS 15 (Intel Core i7-10750H, 2.60 GHz).

II. Rulebooks

This section presents formal definitions and theoretical foundations of the *rulebook* formalism. Throughout the paper, we let $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, $\mathbb{R}_{>0}$, and $\mathbb{N}$ denote the set of real, non-negative real, positive real, and natural numbers, respectively. Additionally, for any sequence $s = \langle s_1,...,s_m\rangle$, we write $s_{i:j}$ to denote the subsequence $\langle s_i,...,s_j\rangle$.

A. Formal Definitions of Rulebooks

A rulebook is evaluated over a set of possible outcomes, referred to as the set of *realizations* and denoted by Σ. In the context of autonomous robots, a realization corresponds to a world trajectory that captures the behavior of the robot, other agents, and relevant objects in the environment.

A rulebook consists of two main components: a set of rules, each representing a distinct objective, and a preorder that captures the relative importance among them. A rule is defined as a function that takes a realization as input and returns a non-negative real number representing the cost associated with that objective (e.g., the amount of rule violation).

Definition 1 (Rule, [59])**.** A *rule* is a function $r : \Sigma \to \mathbb{R}_{\geq 0}$ that measures the degree of violation of its argument.

If $r(x) < r(y)$, then realization y violates the rule r to a greater extent than does x. On the other hand, $r(x) = 0$ indicates that realization x is fully compliant with the rule. While we refer to this function as a "rule", it is not limited to a regulatory constraint but can also represent performance-related criteria like comfort, efficiency, or user preferences that are desirable but not strictly required. Additionally, a rule does not need to be analytical and can be learned from data.

Remark 1. In [59], the set of realizations, Σ, is restricted to externally observable actions and outcomes in the world. In contrast, because we will use the rulebook framework to specify all objectives for control synthesis, rather than for evaluating robot behaviors from the perspective of an external observer, we do not impose this restriction and allow Σ to include any information relevant to the objectives, including internal states or unobservable variables.

A preorder is used to specify the relative importance of these rules, allowing us to express both hierarchical relationships and the notion of non-comparability among objectives.

Definition 2 (Preorder, [64], [65])**.** A *preorder* on a set S is a binary relation $\lesssim$ that is reflexive ($s \lesssim s$ for all $s \in S$), and transitive ($s_1 \lesssim s_2$ and $s_2 \lesssim s_3$ imply $s_1 \lesssim s_3$ for all $s_1,s_2,s_3 \in S$).

Remark 2. A preorder is a generalization of partial order. While both are reflexive and transitive, a preorder does not require antisymmetry, i.e., it is possible for both $s_1 \lesssim s_2$ and $s_2 \lesssim s_1$ to hold for some $s_1 \neq s_2$. As a result, every partial order is a preorder but a preorder that is not antisymmetric is not a partial order. This lack of antisymmetry makes preorders particularly useful in the rulebook formalism as they support the notion of *rules of the same rank*, i.e., distinct rules treated as having equal priority. As we will see later, such rules can be aggregated using monotonic functions (with the weighted sum being a common example) through a refinement operation, while still preserving the original preferences encoded by the unaggregated rulebook. In this way, rules of the same rank provide a natural foundation for capturing the weighted-sum formulations commonly used in the literature.

Given a preorder $\lesssim$ on a set S, we define the associated strict preorder $<$ on S by $s_1 < s_2$ if and only if $s_1 \lesssim s_2$ and $s_2 \not\lesssim s_1$. Furthermore, the equivalence relation $\sim$ induced by $\lesssim$ is defined by $s_1 \sim s_2$ if and only if $s_1 \lesssim s_2$ and $s_2 \lesssim s_1$. For each $s \in S$, the equivalence class of s under $\sim$ is denoted by $[s] = \{s' \in S \mid s' \sim s\}$. The quotient set of S under $\sim$, denoted $S/\!\sim$, is the set of all such equivalence classes, i.e., $S/\!\sim = \{[s] \mid s \in S\}$. The preorder $\lesssim$ on S induces a partial order $\preceq$ on $S/\!\sim$, defined by $[s] \preceq [s']$ if and only if $s_1 \lesssim s_2$ for some (equivalently, all) $s_1 \in [s]$ and $s_2 \in [s']$.

A *total preorder* is a special case of a preorder where every pair of elements is comparable, i.e., for any $s_1,s_2 \in S$, either $s_1 \lesssim s_2$ or $s_2 \lesssim s_1$. A *total order* is a total preorder that is also antisymmetric, meaning that if $s_1 \lesssim s_2$ and $s_2 \lesssim s_1$, then $s_1 = s_2$. Equivalently, in a total order, for any distinct elements $s_1,s_2 \in S$, either $s_1 < s_2$ or $s_2 < s_1$ holds.

Definition 3 (Rulebook, [59])**.** A *rulebook* is defined as a tuple $\mathcal{R} = \langle R, \lesssim\rangle$, where R is the set of rules and $\lesssim$ is a preorder on R, specifying their relative importance.

Throughout the paper, we assume that R is finite. Since the ordering of the rules forms a preorder, comparing any two rules $r_1,r_2 \in R$ yields one of the three possibilities:

(1) One rule has *strictly higher priority* than the other. We write $r_1 > r_2$ to indicate that r_1 has strictly higher priority than r_2. This typically occurs when r_1 is safety-related and r_2 is not, e.g., r_1 enforces collision avoidance, while r_2 enforces correct use of turn signals.

(2) They are *incomparable*, i.e., $r_1 \not\lesssim r_2$ and $r_2 \not\lesssim r_1$. This typically occurs when they address different concerns, e.g., r_1 addresses safety of property while r_2 addresses safety of animals.

(3) They are *of the same rank*, denoted by $r_1 \sim r_2$. This typically occurs when they address related concerns, e.g., r_1 and r_2 enforce clearance to different types of objects.

B. Graph Representation of Rulebooks

Since a preorder $\lesssim$ on R can be viewed as a partial order $\preceq$ on the quotient set $R/\!\sim$, we can represent a rulebook $\mathcal{R} = \langle R, \lesssim\rangle$ using a Hasse diagram, i.e., a graph $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ of the partially ordered set $\langle R/\!\sim, \preceq\rangle$. Each vertex $v \in V_{\mathcal{R}}$

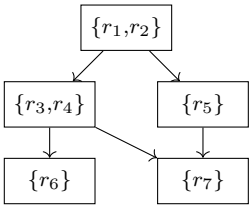


Fig. 1: Hasse diagram $G_{\mathcal{R}}$ representing the rulebook $\mathcal{R}$ in Example 1. Each node v corresponds to a set $[v] \subseteq R$ of rules that are of the same rank, i.e., mutually equivalent under $\lesssim$. A directed edge from v to v' indicates that every rule in $[v]$ has strictly higher priority than every rule in $[v']$.

represents an equivalence class $[r] \in R/\sim$ of rules with the same rank. We write $[v] \subseteq R$ for the set of rules associated with v. An edge $(v,v') \in E_{\mathcal{R}}$, denoted by $v \to v'$, indicates that $[v'] \prec [v]$, meaning that every rule in $[v]$ has strictly higher priority than every rule in $[v']$ under the original preorder $\lesssim$.[1]

Example 1. Consider a rulebook $\mathcal{R} = \langle R, \lesssim \rangle$, where $R = \{r_1,...,r_7\}$ and $\lesssim$ is given as the reflexive and transitive closure of the following base relations:

- $r_1 \lesssim r_2$, $r_2 \lesssim r_1$ (so r_1 and r_2 are of the same rank),
- $r_3 \lesssim r_4$, $r_4 \lesssim r_3$ (so r_3 and r_4 are of the same rank)
- $r_3 \lesssim r_1$, $r_3 \lesssim r_2$, $r_4 \lesssim r_1$, $r_4 \lesssim r_2$, $r_5 \lesssim r_1$, $r_5 \lesssim r_2$ (so r_1 and r_2 have strictly higher priority than r_3, r_4, r_5),
- $r_6 \lesssim r_3$, $r_6 \lesssim r_4$, $r_7 \lesssim r_3$, $r_7 \lesssim r_4$ (so r_3 and r_4 have strictly higher priority over r_6 and r_7),
- $r_7 \lesssim r_5$ (so r_5 has strictly higher priority over r_7).

The reflexive and transitive closures of these base relations ensure that $\lesssim$ is a preorder, i.e., $\lesssim$ is both reflexive and transitive. Specifically, the reflexive closure adds the relations $r_i \lesssim r_i, \forall i$, ensuring reflexivity. The transitive closure adds the relations $r_i \lesssim r_k$ for any i,k if there exists j such that $r_i \lesssim r_j$ and $r_j \lesssim r_k$, ensuring transitivity. For example, since $r_6 \lesssim r_3$ and $r_3 \lesssim r_1$, the transitive closure adds $r_6 \lesssim r_1$.

The resulting set of equivalence classes is $R/\sim = \{[r_1],[r_3],[r_5],[r_6],[r_7]\}$, where $[r_1] = [r_2] = \{r_1,r_2\}$, $[r_3] = [r_4] = \{r_3,r_4\}$, and $[r_i] = \{r_i\}$ for all $i \in \{5,6,7\}$. The preorder $\lesssim$ on R induces a partial order $\preceq$ on the quotient set $R/\sim$ with the following relations: $[r_3] \prec [r_1]$, $[r_5] \prec [r_1]$, $[r_6] \prec [r_3]$, $[r_7] \prec [r_3]$, and $[r_7] \prec [r_5]$. The Hasse diagram of $\langle R/\sim, \preceq \rangle$ is shown in Fig. 1.

C. Induced Preorder over Realizations

A rulebook $\mathcal{R}$ induces a preference relation $\lesssim_{\mathcal{R}}$ over realizations Σ by comparing the degree to which each realization violates the rules. Intuitively, a realization x is considered *at least as good as* another realization y, denoted $x \lesssim_{\mathcal{R}} y$, if x does not perform worse than y on any rule without compensating by performing better on a higher-priority rule.

Definition 4 (Induced Preorder, [59])**.** Let $\mathcal{R} = \langle R, \lesssim \rangle$ be a rulebook over realizations Σ. The *induced preorder* $\lesssim_{\mathcal{R}} \subseteq \Sigma \times \Sigma$ is defined as $x \lesssim_{\mathcal{R}} y$ if and only if for every rule $r \in R$ such that $r(x) > r(y)$, there exists a higher priority rule $r' > r$ such that $r'(x) < r'(y)$. In other words, any rule that prefers y over x must be of lower priority than some rule that prefers x over y.

If at least one of $x \lesssim_{\mathcal{R}} y$ or $y \lesssim_{\mathcal{R}} x$ holds, we say that x and y are *comparable*; otherwise, they are *incomparable*. The following proposition establishes that the induced relation $\lesssim_{\mathcal{R}}$ is a preorder on Σ, which ensure that $\lesssim_{\mathcal{R}}$ is logically consistent and does not admit cyclic preferences among realizations. A complete proof can be found in [59].

Proposition 1 ([59])**.** Let $\mathcal{R} = \langle R, \lesssim \rangle$ be a rulebook over a set of realizations Σ. The induced preorder $\lesssim_{\mathcal{R}} \subseteq \Sigma \times \Sigma$ as defined in Definition 4 is reflecxive and transitive.

The induced preorder $\lesssim_{\mathcal{R}}$ also defines the corresponding strict preference relation $<_{\mathcal{R}}$ over Σ. In particular, $x <_{\mathcal{R}} y$ indicates that x is *strictly better than* y.

Definition 5 (Induced Strict Preorder, adapted from [59])**.** Let $\lesssim_{\mathcal{R}} \subseteq \Sigma \times \Sigma$ be the induced preorder of $\mathcal{R}$. The *induced strict preorder* $<_{\mathcal{R}} \subseteq \Sigma \times \Sigma$ is defined as $x <_{\mathcal{R}} y$ if and only if $x \lesssim_{\mathcal{R}} y$ and $y \not\lesssim_{\mathcal{R}} x$. In other words, $x <_{\mathcal{R}} y$ means that x is at least as good as y but y is not at least as good as x.

Realizations that are equally preferred according to the rulebook are considered equivalent. This notion of equivalence is formalized by the induced equivalence relation $\sim_{\mathcal{R}}$.

Definition 6 (Induced Equivalence Relation, adapted from [59])**.** Let $\lesssim_{\mathcal{R}} \subseteq \Sigma \times \Sigma$ be the induced preorder of a rulebook $\mathcal{R}$. The *induced equivalence relation* $\sim_{\mathcal{R}} \subseteq \Sigma \times \Sigma$ is defined as $x \sim_{\mathcal{R}} y$ if and only if $x \lesssim_{\mathcal{R}} y$ and $y \lesssim_{\mathcal{R}} x$.

The following proposition characterizes this equivalence in terms of rule evaluations. A full proof can be found in [59].

Proposition 2 ([59])**.** Let $\mathcal{R} = \langle R, \lesssim \rangle$ be a rulebook over a set of realizations Σ and let $x,y \in \Sigma$. Then, $x \sim_{\mathcal{R}} y$ if and only if $r(x) = r(y)$ for all rules $r \in R$.

Example 2. Consider a rulebook with $R = \{r_1,r_2\}$ and two realizations x and y, where $r_1(x) = 1, r_2(x) = 2, r_1(y) = 2, r_2(y) = 1$. Suppose r_1 and r_2 are of the same rank, i.e., $r_1 \sim r_2$. Since $r_2(x) > r_2(y)$ and there is no rule $r' > r_2$ such that $r'(x) < r'(y)$, we can conclude that $x \not\lesssim_{\mathcal{R}} y$. Similarly, since $r_1(y) > r_1(x)$ and there is no rule $r' > r_1$ such that $r'(y) < r'(x)$, we can also conclude that $y \not\lesssim_{\mathcal{R}} x$. As a result, x and y incomparable. The same conclusion will be reached if r_1 and r_2 are incomparable. On the other hand, if $r_1 > r_2$, we can conclude that $x <_{\mathcal{R}} y$, whereas if $r_2 > r_1$, then $y <_{\mathcal{R}} x$.

D. Refining Rulebooks

To tailor robots to specific contexts, it is often useful to refine a rulebook to resolve ambiguities or add more detailed preferences while preserving the original intent. For example, we may have a general-purpose or base rulebook that reflects broad societal values or general regulations, while a refined version incorporates specific legislation, user preferences, or task-specific priorities. Refinement is also essential in single-strategy control synthesis (Section V) to incorporate all preferences, ensuring the extracted strategy is the most desirable among all optimal solutions defined by the base rulebook. The key requirement is that the refined

[1] We reverse the direction of the edges compared to a traditional Hasse diagram to simplify topological sorting of equivalent rules in Section VI.

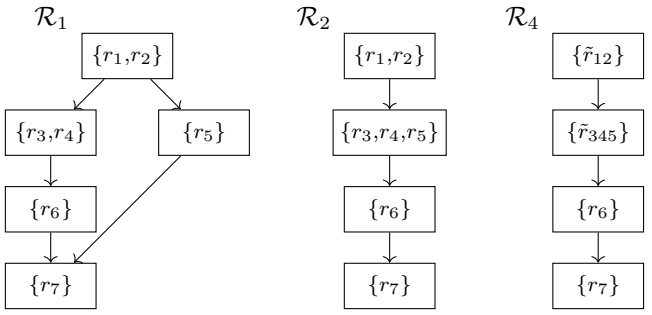


Fig. 2: Hasse diagram of the rulebooks $\mathcal{R}_1$ and $\mathcal{R}_2$ (Example 3) and $\mathcal{R}_4$ (Example 4), each being a refinement of $\mathcal{R}$.

rulebook should remain consistent with the original in the sense that no preference from the base rulebook should be contradicted.

Definition 7. Let $\mathcal{R}_1 = \langle R_1, \lesssim_1 \rangle$ and $\mathcal{R}_2 = \langle R_2, \lesssim_2 \rangle$ be two rulebooks over the same set of realizations Σ. We say that $\mathcal{R}_2$ is a *refinement* of $\mathcal{R}_1$ if for all $x,y \in \Sigma$,

$$x \lesssim_{\mathcal{R}_1} y \Longrightarrow x \lesssim_{\mathcal{R}_2} y \quad \text{and} \quad x <_{\mathcal{R}_1} \Longrightarrow x <_{\mathcal{R}_2} y. \tag{1}$$

Two basic operations that can be used to construct a rulebook that is a refinement of the original rulebook according to Definition 7 are *priority refinement* and *rule aggregation*. Intuitively, *priority refinement* introduces a new priority relationship between two previously incomparable rules and *rule aggregation* combines two rules of the same rank into a single rule while preserving their shared priority. We now formalize each of these operations. The following definitions are equivalent to those in [59] but are reformulated to explicitly specify the conditions under which each operation can be applied and how the resulting rulebook is constructed to offer a more application-oriented formulation.

Definition 8 (Priority Refinement)**.** Let $\mathcal{R}_1 = \langle R_1, \lesssim_1 \rangle$ be a rulebook and $r,r' \in R_1$ be two incomparable rules, i.e., $r \not\lesssim_1 r'$ and $r' \not\lesssim_1 r$. A *priority refinement* operation produces a new rulebook $\mathcal{R}_2 = \langle R_2, \lesssim_2 \rangle$ such that $R_2 = R_1$ and $\lesssim_2$ is the smallest preorder satisfying $\lesssim_1 \subseteq \lesssim_2$ and either $r \lesssim_2 r'$ or $r' \lesssim_2 r$ or both. In other words, $\lesssim_2$ extends $\lesssim_1$ by introducing a new priority relation between two incomparable rules or declaring them to be of the same rank, followed by taking its transitive closure to ensure that $\lesssim_2$ remains a preorder.

Example 3. Consider the rulebook $\mathcal{R} = \langle R, \lesssim \rangle$ in Example 1. The incomparability between r_6 and r_7 can be resolved through a *priority refinement* operation to produce a refined rulebook $\mathcal{R}_1 = \langle R, \lesssim_1 \rangle$, where $\lesssim_1$ is the smallest preorder containing $\lesssim$ and satisfying $r_7 \lesssim_1 r_6$. Intuitively, this refinement declares r_6 to have strictly higher priority than r_7.

We can further refine $\mathcal{R}_1$ by applying another *priority refinement* operation to produce $\mathcal{R}_2 = \langle R, \lesssim_2 \rangle$, where r_3 and r_5 are of the same rank by adding the relations $r_5 \lesssim_2 r_3$ and $r_3 \lesssim_2 r_5$. To ensure that $\lesssim_2$ remains a preorder, we take its transitive closure, which requires including $r_4 \lesssim_2 r_5$, $r_5 \lesssim_2 r_4$, and $r_6 \lesssim r_5$. As a result, $\mathcal{R}_2$ is a refinement of $\mathcal{R}_1$ with r_5 being in the same rank as r_3 and r_4.

Both $\mathcal{R}_1$ and $\mathcal{R}_2$ are illustrated in Fig. 2. Note that $\lesssim_2$ forms a total preorder, though it is not a total order because some distinct rules remain equivalent.

Definition 9 (Rule Aggregation)**.** Let $\mathcal{R}_1 = \langle R_1, \lesssim_1 \rangle$ be a rulebook and $r,r' \in R_1$ be two rules of the same rank, i.e., $r \sim_1 r'$. A *rule aggregation* operation produces a new rulebook $\mathcal{R}_2 = \langle R_2, \lesssim_2 \rangle$ such that (1) $R_2 = (R_1 \setminus \{r,r'\}) \cup \{\tilde{r}\}$, where $\tilde{r}(x) = \alpha(r(x), r'(x))$ and α is a function that is strictly monotone in both arguments; and (2) for any $r_1,r_2 \in R_2 \setminus \{\tilde{r}\}$, $r_1 \lesssim_2 r_2$ if and only if $r_1 \lesssim_1 r_2$; $r_1 \lesssim_2 \tilde{r}$ if and only if $r_1 \lesssim_1 r$ and $r_1 \lesssim_1 r'$; and $\tilde{r} \lesssim_2 r_1$ if and only if $r \lesssim_1 r_1$ and $r' \lesssim_1 r_1$. In other words, this operation merges rules r and r' into a single rule $\tilde{r} = \alpha(r,r')$ that inherits their joint priority relations. A typical choice of α is a positive weighted sum, i.e., $\alpha(r(x),r'(x)) = ar(x) + br'(x)$ where $a,b \in \mathbb{R}_{>0}$.

Remark 3. Definition 9 can be naturally expanded to accommodate aggregation of multiple rules $r_1, \ldots, r_K$ with $r_i \sim_1 r_j$ for all i,j by setting $\tilde{r}(x) = \alpha(r_1(x),\ldots,r_K(x))$, where $\alpha : \mathbb{R}^K_{\geq 0} \to \mathbb{R}_{\geq 0}$ is strictly monotone in all arguments.

Example 4. Consider the rulebook $\mathcal{R}_2 = \langle R, \lesssim_2 \rangle$ in Example 3. We can refine $\mathcal{R}_2$ by applying a *rule aggregation* operation to combine r_1 and r_2 into a single rule $\tilde{r}_{12} = ar_1 + br_2$ for some $a,b \in \mathbb{R}_{>0}$. This results in a rulebook $\mathcal{R}_3 = \langle R_3, \lesssim_3 \rangle$, where $R_3 = (R \setminus \{r_1,r_2\}) \cup \{\tilde{r}_{12}\}$. The aggregated rule $\tilde{r}_{12}$ inherits the joint priority relations of r_1 and r_2 in the sense that any rule that is ranked below both r_1 and r_2 in $\mathcal{R}_2$ is also ranked below $\tilde{r}_{12}$ in $\mathcal{R}_3$ and any rule that is ranked above both r_1 and r_2 in $\mathcal{R}_2$ is also ranked above $\tilde{r}_{12}$ in $\mathcal{R}_3$.

We can further refine $\mathcal{R}_3$ by applying two additional *rule aggregation* operations to combine r_3,r_4,r_5 into a single rule $\tilde{r}_{345}$. The resulting rulebook $\mathcal{R}_4 = \{R_4, \lesssim_4\}$ is shown in Fig. 2. Note that $\lesssim_4$ forms a total order.

The following proposition states that applying any combination of the two basic operations above results in a rulebook that is a refinement of the original rulebook. A complete proof can be found in [59].

Proposition 3 ([59])**.** Applying any combination of the two basic operations (*priority refinement* and *rule aggregation*) to a rulebook $\mathcal{R}_1$ produces a rulebook $\mathcal{R}_2$ that is a refinement of $\mathcal{R}_1$ according to Definition 7

Examples 3 and 4 illustrate that a rulebook can be refined into one where the rules are totally ordered. The following proposition formalizes this observation.

Proposition 4. Given a rulebook $\mathcal{R} = \langle R, \lesssim \rangle$, there exists a finite sequence of *priority refinement* and *rule aggregation* operations that produces a rulebook $\mathcal{R}_T = \langle R_T, \lesssim_T \rangle$ such that $\lesssim_T$ is a total order on R_T and $\mathcal{R}_T$ is a refinement of $\mathcal{R}$ according to Definition 7.

Proof. Let $\mathcal{R}_0 = \langle R_0, \lesssim_0 \rangle$ be the original rulebook, where $R_0 = R$ and $\lesssim_0 = \lesssim$. We construct a sequence of rulebooks $\mathcal{R}_{i+1} = \langle R_{i+1}, \lesssim_{i+1} \rangle$ for each iteration $i \geq 0$ until $\lesssim_i$ becomes a total order on R_i. At each iteration,

- Initialize $R_{i+1} = R_i$ and $\lesssim_{i+1} = \lesssim_i$.
- If there exist incomparable rules $r, r' \in R_i$, apply a *priority refinement* operation by adding either $r \lesssim_{i+1} r'$ or $r' \lesssim_{i+1} r$ or both. Then, take the transitive closure to maintain the preorder.

- Otherwise, if there exist distinct rules $r,r' \in R_i$ with $r \sim_i r'$, apply a *rule aggregation* operation to combine them into a single rule and update R_{i+1} and $\lesssim_{i+1}$ accordingly.

Each iteration strictly decreases either the number of incomparable pairs or the number of rules in the same rank. Since R is finite, this process terminates after a finite number of steps. Let $\mathcal{R}_T = \langle R_T, \lesssim_T \rangle$ be the rulebook obtained at termination. By construction, $\lesssim_T$ is a total order on R_T because no pairs of rules are incomparable or equivalent. Additionally, by Proposition 3, $\mathcal{R}_T$ is a refinement of $\mathcal{R}$. □

Remark 4. The commonly used weighted sum formulation can be viewed as a special case of the rulebook formalism, where all rules are initially of the same rank. Applying a sequence of rule aggregation operations then combines them into a single weighted sum as demonstrated in Example 4.

Remark 5. Two different relations between r and r', namely when they are of the same rank ($r \sim r'$) and when they are incomparable ($r \not\lesssim r'$ and $r' \not\lesssim r$), result in the same induced ordering over realizations. However, they differ in how the rulebook may be refined, as defined in Definition 7. If $r \sim r'$, the rulebook can be refined through *rule aggregation*, which combines r and r' into a new rule $\tilde{r}$ as defined in Definition 9. In contrast, if r and r' are incomparable, the rulebook can be refined through *priority refinement*, which introduces a new priority relation $r \lesssim r'$ or $r' \lesssim r$ or both, allowing for a strict priority like $r < r'$ or $r' < r$. However, if $r \sim r'$, introducing a strict priority between them in a refinement is not allowed as it would contradict their initial equivalence in rank. Thus, refinement operations serve as a mechanism for controlling how a rulebook can be modified while ensuring that any changes respect the original design semantics and preserve the intended priority structure.

Another operation that allows one to resolve preferences between realizations previously considered equivalent is *rule augmentation*. This operation introduces a new rule that has strictly lower priority than all the existing rules. However, as shown in [59], rule augmentation satisfies only the second condition of (1) but, in general, not the first. Specifically, realizations x,y that are equivalent under the original rulebook $\mathcal{R}_1$, i.e., $x \lesssim_{\mathcal{R}_1} y$ and $y \lesssim_{\mathcal{R}_1} x$ may no longer be equivalent under the refined rulebook $\mathcal{R}_2$, which may now prefer one realization over the other. Although this violates the requirement that refinements preserve equivalence classes, rule augmentation can still be useful in applications where such distinctions are allowed or desirable. For example, one may introduce an efficiency-related objective to break ties among otherwise equivalent realizations.

Definition 10 (Rule Augmentation)**.** Let $\mathcal{R}_1 = \langle R_1, \lesssim_1 \rangle$ be a rulebook. A *rule augmentation* operation constructs a new rulebook $\mathcal{R}_2 = \langle R_2, \lesssim_2 \rangle$ such that $R_2 = R_1 \cup \{r\}$ and $r <_2 r'$ for all $r' \in R_1$.

III. Control Synthesis with Rulebooks

This section formulates the control synthesis problem with respect to rulebooks. We adopt the standard formal methods approach, which separates the robot model from its specification. Specifically, the robot *model* describes all the physically possible behaviors of the robot, regardless of whether they are desirable, while the *specification* describes all the desirable behaviors, regardless of whether they are physically achievable. However, in traditional formal methods, a specification is typically Boolean, i.e., a trajectory either satisfies it or not. This allows one to determine whether a given robot trajectory satisfies a given specification by examining the trajectory alone, without having to consider alternatives. In contrast, our specification encodes a set of objectives and their relative priorities, rather than a binary notion of satisfaction. As a result, it is not straightforward to determine whether a given trajectory is consistent with the specification in the sense that it is optimal just by examining it in isolation. To address this, in Section VI, we show how to compute the complete set of optimal robot trajectories with respect to a given rulebook. This set enables us to determine whether a given trajectory is consistent with the rulebook by comparing it against all optimal alternatives.

Robot model: Let $X \subset \mathbb{R}^d$, where $d \in \mathbb{N}$, be a compact set of states and U be a compact set of control inputs (or actions) available to the robot. The initial state is denoted by $x_I \in X$ and the set of goal states by $X_G \subseteq X$. The robot's dynamics are described by either a *discrete-time* model

$$x_{t+1} = f(x_t, u_t), x_0 = x_I, t \in \mathbb{N}, \tag{2}$$

or a *continuous-time* model

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \mathbf{x}(0) = x_I, t \in \mathbb{R}_{\geq 0}. \tag{3}$$

While both models are relevant, for brevity, we provide full formal details only for the discrete-time case. At each state $x \in X$, the available actions are $U(x) \subseteq U$. A *control strategy* is a finite sequence of actions $\mathbf{u} = \langle u_0, u_1, \ldots, u_T \rangle$ for some $T \in \mathbb{N}$. A strategy is *admissible* if $u_t \in U(x_t)$ for all t, where the associated state sequence satisfies (2). The set of admissible control strategies is then defined as $\mathcal{U} = \{\langle u_0, u_1, \ldots, u_T \rangle \mid T \in \mathbb{N}, x_{t+1} = f(x_t, u_t), x_0 = x_I, u_t \in U(x_t), \forall t \in \{0, \ldots, T\}\}$. Given a control strategy $\mathbf{u} = \langle u_0, u_1, \ldots, u_T \rangle \in \mathcal{U}$, we denote the corresponding robot's trajectory by $\mathbf{x}_\mathbf{u}(t) = \Phi(t, x_I, \mathbf{u})$, where $\Phi : \{1, \ldots, T+1\} \times X \times \mathcal{U} \to X$ is the discrete-time flow map that returns the state at time t given the initial state x_I and control strategy $\mathbf{u}$. The set of robot trajectories induced by $\mathcal{U}$ is given by $\mathcal{X} = \{\Phi(\cdot, x_I, \mathbf{u}) \mid \mathbf{u} \in \mathcal{U}\}$. Finally, the set of goal-reaching control strategies is defined as $\mathcal{U}_G = \{\mathbf{u} \in \mathcal{U} \mid \Phi(T+1, x_I, \mathbf{u}) \in X_G\}$. For the continuous-time model, a *control strategy* is a function $\mathbf{u} : [0, T] \to U$ defined over some time horizon $T \in \mathbb{R}_{\geq 0}$. The sets $\mathcal{U}$, $\mathcal{X}$ and $\mathcal{U}_G$ as well as the flow map Φ are defined analogously to the discrete-time case, with admissible control strategies represented by Lebesgue measurable functions instead of sequences. We refer the reader to [66]–[68] for technical assumptions to ensure existence and uniqueness of solutions of (3).

Robot specification: We use the rulebook formalism described in Section II to specify the objectives of the robot. The set of realizations is defined as $\Sigma = \mathcal{X} \times \mathcal{U}$. Under this definition, a rule r takes as input a robot's trajectory $\mathbf{x} \in \mathcal{X}$ and a control strategy $\mathbf{u} \in \mathcal{U}$ and returns a non-negative number that quantifies the degree of violation or cost associated with $\mathbf{x}$ and $\mathbf{u}$ with respect to r. Recall that for the discrete-time case, $\mathbf{x}$ and $\mathbf{u}$ are sequences, whereas for the continuous-time case, $\mathbf{x}$ and $\mathbf{u}$ are functions.

Since the robot's trajectory is uniquely determined for any control strategy $\mathbf{u} \in \mathcal{U}$ under both models (2) and (3), we slightly abuse notation and write $\mathbf{u} \lesssim_\mathcal{R} \tilde{\mathbf{u}}$ for any admissible control strategies $\mathbf{u}, \tilde{\mathbf{u}} \in \mathcal{U}$ to mean $(\mathbf{x}_\mathbf{u}, \mathbf{u}) \lesssim_\mathcal{R} (\mathbf{x}_{\tilde{\mathbf{u}}}, \tilde{\mathbf{u}})$, where $\mathbf{x}_\mathbf{u}(t) =$

$\Phi(t,x_I,\mathbf{u})$ and $\mathbf{x}_{\tilde{\mathbf{u}}}(t)=\Phi(t,x_I,\tilde{\mathbf{u}})$ are the trajectories induced by $\mathbf{u}$ and $\tilde{\mathbf{u}}$, respectively. We are particularly interested in control strategies in $\mathcal{U}_G$, i.e., those that drive the robot to the goal region. We say that control strategy $\mathbf{u}^* \in \mathcal{U}_G$ is *optimal* or *non-dominated* with respect to $\mathcal{R}$ if there is no other $\mathbf{u} \in \mathcal{U}_G$ such that $\mathbf{u} <_{\mathcal{R}} \mathbf{u}^*$. Note that this restriction to $\mathcal{U}_G$ is without loss of generality since one can simply set $\mathcal{U}_G = \mathcal{U}$ for applications without a goal region.

We now formalize two related control synthesis problems, one computes a single optimal strategy for a given rulebook, and the other computes the complete set of optimal strategies. Access to this complete set supports formal verification and enables humans or learning algorithms to choose among optimal options without explicitly deriving rules for complex criteria such as natural driving behavior.

Problem 1 (Single-strategy control synthesis with rulebooks)**.** Given a robot model (2) or (3) and a rulebook $\mathcal{R} = \langle R, \lesssim \rangle$, compute an optimal control strategy $\mathbf{u}^* \in \mathcal{U}_G$.

Problem 2 (Complete control synthesis with rulebooks)**.** Given a discrete-time, discrete-state robot model (2) and a rulebook $\mathcal{R} = \langle R, \lesssim \rangle$, compute the (maximum) complete set $\mathcal{U}_G^* \subseteq \mathcal{U}_G$ of all optimal control strategies such that for all $\mathbf{u}^* \in \mathcal{U}_G^*$ and $\mathbf{u} \in \mathcal{U}_G$, $\mathbf{u} \not<_{\mathcal{R}} \mathbf{u}^*$ and for each $\mathbf{u}' \in \mathcal{U}_G \setminus \mathcal{U}_G^*$, there exists $\mathbf{u}^* \in \mathcal{U}_G^*$ such that $\mathbf{u}^* <_{\mathcal{R}} \mathbf{u}'$.

Note that Problem 2 considers the case where X and U are finite to ensure the finiteness of $\mathcal{U}_G^*$.

Relevance to robotics applications: Problem 1 and Problem 2 serve as a unifying framework for a broad range of control synthesis formulations commonly found in robotics literature, as will be shown in Section IV. In particular, the rulebook formalism provides a systematic and expressive way to specify and reason about competing objectives or constraints, making it particularly relevant for autonomous robots tasked with complex missions that require them to make decisions under trade-offs. A key application where the rulebook formalism has demonstrated significant potential is autonomous driving, where it has been used to formalize traffic rules and their relative priorities. This is crucial because real-world traffic rules may conflict and strictly adhering to all of them may not be possible as acknowledged by regulatory bodies [69].

Formalizing traffic rules has also been an active area of research in the robotics community [70]–[72]. However, as shown in [73], such formalization is not straightforward because humans do not always agree on the best way to drive, especially in complex situations that may require breaking some rules (e.g., the trolley problem or "Boston Left"). To investigate this challenge, [73] proposed a specific rulebook that encodes traffic rules and introduced a dataset of 92 complex driving scenarios, each with multiple possible behaviors of an autonomous vehicle. Experienced human drivers were asked to compare pairs of behaviors and indicate their preferrences. The results showed that the proposed rulebook achieved accuracy comparable to that of state-of-the-art machine learning models, while also offering the critical benefit of interpretability, a key requirement for earning the trust of regulators and the public. Unlike black-box models, the rulebook formalism enables explicit justification of decisions based on clearly defined rules and their priorities, offering a transparent framework for decision-making in safety-critical domains. Moreover, the generality of the rulebook formalism and its independence from the underlying robot model make it particularly well-suited for communication among diverse stakeholders, including system designers, engineers, and regulatory bodies.

IV. Unifying Planning and Control Formulations

In this section, we demonstrate how a wide range of existing planning and control formulations, whether already widely used in robotics or showing strong potential for future applications, can be unified within our framework as instances of either Problem 1 or Problem 2. We exclusively consider deterministic, time-invariant systems, leaving extensions to adversarial or probabilistic settings as future work.

A. Control Synthesis with Temporal-Logic Specifications

Boolean satisfaction semantics: Earlier work in this domain has focused on computing a robot's control strategy $\mathbf{u}$ to ensure that the resulting trajectory $\mathbf{x}_\mathbf{u}$ satisfies a given temporal logic formula φ, denoted by $\mathbf{x}_\mathbf{u} \models \varphi$. Here, φ typically encodes the robot's high-level complex tasks. Satisfaction in this context is binary, i.e., a trajectory either satisfies or violates the specification, with no quantification of the extent of satisfaction or violation. Different variants of temporal logics have been explored, along with both continuous-time and discrete-time models of the robot. For example, references [7]–[10], [28], [29] consider a continuous-time model, [11] considers a discrete-time, discrete-state model, and [12], [27] consider a discrete-time, continuous-state model. In terms of specifications, [7] considers propositional temporal logic over the reals (RTL), a continuous-time version of linear temporal logic (LTL); [8] considers LTL without the "next" operator (LTL_X); [9] considers syntactically co-safe LTL; [10], [11] consider LTL; [12] considers deterministic μ-calculus; and [27]–[29] consider signal temporal logic (STL) specifications.

Quantitative satisfaction semantics: More recent work leverages variants of temporal logic that include quantitative semantics, allowing for a measure of how well a specification is satisfied. One of the most widely used formalisms is STL, introduced in [25]. The STL robustness degree, denoted by $\rho(\mathbf{x}, \varphi, t)$, quantifies how robustly a signal suffix $(\mathbf{x}, t)$ satisfies an STL formula φ. This quantitative semantics enables optimization-based formulation. For example, references [30], [31] consider a discrete-time model, an STL specification φ over predicates in X or U, and cost function J. The goal is to compute an optimal trajectory that minimizes J, while satisfying φ, if such a trajectory exists. Otherwise, the formulation seeks a trajectory that minimizes the degree of violation of φ, quantified using STL's robustness metrics. On the other hand, [32] considers a continuous-time model and proposes an RRT*-based algorithm that aims to maximize the robustness degree with respect to an STL specification φ. To ensure asymptotic optimality under this complex objective, the RRT* algorithm is modified to bias sample selection toward maximal satisfaction of φ.

Formulating as an instance of Problem 1: We begin by defining a rule r_φ that captures the specification φ.

- Under boolean semantics, r_φ is defined as $r_\varphi(\mathbf{x},\mathbf{u}) = 0$ if $\mathbf{x} \models \varphi$ and $r_\varphi(\mathbf{x},\mathbf{u}) = 1$ otherwise.

- Under quantitative semantics, r_φ is defined as $r_\varphi(\mathbf{x}, \mathbf{u}) = \max\{0, -\rho(\mathbf{x}, \varphi, 0)\}$, where $\rho(\mathbf{x}, \varphi, t)$ is the STL robustness degree of $(\mathbf{x},t)$ with respect to φ.

If the only objective is to ensure that the robot satisfies φ (as in, e.g., [7]–[10], [12]), the rule set is simply $R=\{r_\varphi\}$. If minimization of a cost function $J:\mathcal{X}\times\mathcal{U}\to\mathbb{R}_{\geq 0}$ is also considered, but only after ensuring satisfaction of φ (as in [11], [27], [30]), we define an additional rule $r_J=J$, resulting in $R=\{r_\varphi,r_J\}$ with $r_\varphi>r_J$, meaning that r_φ has priority over r_J. We consider two possible cases.

(1) φ is satisfiable. Regardless of whether the cost function J is included, the rule priority $r_\varphi > r_J$ ensures that any optimal control strategy $\mathbf{u}^*$ (as defined in Problem 1) results in $r_\varphi(\mathbf{x}_{\mathbf{u}^*},\mathbf{u}^*)=0$. Based on the definition of r_φ, this implies that the robot satisfies the specification, i.e., $\mathbf{x}_{\mathbf{u}^*}\models\varphi$ under Boolean semantics and $\rho(\mathbf{x}_{\mathbf{u}^*},\varphi,0)\geq 0$ under quantitative semantics. Additionally, since r_φ is lower-bounded by 0, it does not reward higher degree of satisfaction of φ. As a result, if J is included in the rulebook, an optimal control strategy $\mathbf{u}^*$ minimizes J among all the strategies that satisfy φ.

(2) φ is not satisfiable. Under Boolean semantics, any optimal control strategy $\mathbf{u}^*$ (as defined in Problem 1) results in $r_\varphi(\mathbf{x}_{\mathbf{u}^*},\mathbf{u}^*)>0$, indicating that φ is not satisfiable; thus, $\mathbf{u}^*$ is not a valid solution. Under quantitative semantics, since $r_\varphi>r_J$, an optimal control strategy $\mathbf{u}^*$ prioritizes minimizing the degree of violation $-\rho(\mathbf{x}_{\mathbf{u}^*},\varphi,0)$.

In both cases, the definition of optimal control strategies in Problem 1 is consistent with standard approaches in the literature on control synthesis with temporal logic specifications.

B. Minimum-Violation Planning

Minimum-violation planning addresses scenarios where a robot is subject to a set $\Phi=\{\varphi_1,...,\varphi_N\}$ of specifications and may not be able to satisfy all of them. Thus, the goal shifts from full satisfaction to minimizing the degree of violation. This problem has been investigated under both discrete-time, discrete-state [74]–[78] and continuous-time [67], [68], [79], [80] models, using various metrics proposed to quantify the extent to which each specification is violated.

Maximizing task completion: In [74], each specification φ_i corresponds to a task, specified by an LTL formula and associated with a reward $rew(\varphi_i)\in\mathbb{N}$. The control synthesis problem is to maximize the total reward $\sum_{\{\varphi_i \mid \mathbf{x}\models\varphi_i\}} rew(\varphi_i)$. Equivalently, this can be framed as a penalty minimization problem by introducing a penalty function $c_i:\mathcal{X}\to\mathbb{N}$ for each φ_i such that $c_i(\mathbf{x})=0$ if $\mathbf{x}\models\varphi_i$ and $c_i(\mathbf{x})=rew(\varphi_i)$ otherwise. The control synthesis problem then becomes computing a control strategy that minimizes the total cost $\sum_i c_i(\mathbf{x}_\mathbf{u})$ among all strategies $\mathbf{u}\in\mathcal{U}$.

Minimizing safety violation: In [67], [68], [78], each specification φ_i corresponds to a safety requirement, expressed using finite LTL (FLTL), and the set $\Phi=\{\varphi_1,...,\varphi_N\}$ is equipped with a total preorder. The degree to which a trajectory $\mathbf{x}$ violates a specification φ_i is quantified by a penalty function $c_i:\mathcal{X}\to\mathbb{R}_{\geq 0}$, where $c_i(\mathbf{x})$ is the minimum amount of time that must be removed from $\mathbf{x}$ for it to satisfy φ_i. The total preorder over Φ imposes a priority structure, which partitions Φ into equivalence classes $\Psi_1,...,\Psi_M$, where each $\Psi_i\subseteq\Phi$ groups specifications of equal priority. These classes are ordered such that specifications in Ψ_i have higher priority than those in Ψ_j for any $j>i$. We refer to the overall priority structure as $\Psi=(\Psi_1,...,\Psi_M)$. To combine the specifications within each group Ψ_j, a weight w_i is assigned to each specification φ_i and the total violation for Ψ_j is computed as the weighted sum of the the individual violations. Finally, a cost function $J:\mathcal{X}\to\mathbb{R}_{\geq 0}$ is defined to capture performance objectives (e.g., minimizing time to reach the goal region). The total cost of a trajectory $\mathbf{x}$ can be represented as a vector $c(\mathbf{x})=(c_1,...,c_{M+1})$, where each c_j for $j\in\{1,...,M\}$ denotes the total weighted violation of specifications in group Ψ_j and $c_{M+1}=J(\mathbf{x})$. These cost vectors are compared using standard lexicograhic ordering, where vectors are compared element-wise from left to right and the first differing element determines the ordering. The control synthesis problem here is to compute a control strategy that minimizes the cost $c(\mathbf{x}_\mathbf{u})$ among all goal reaching strategies $\mathbf{u}\in\mathcal{U}_G$.

Other tasks and violation metrics: Other types of specifications and violation metrics have also been explored [75]–[77], [79]–[81]. While the specific metrics vary, a common theme across these formulations is the association of each specification φ_i with a penalty function c_i such that $c_i(\mathbf{x})$ quantifies the extent to which trajectory $\mathbf{x}$ violates φ_i. These penalty functions are sufficiently flexible to accommodate a variety of violation metrics. For example, in formulations that only care whether φ_i is satisfied (e.g., when φ_i represents a hard constraint), the penalty can be defined as $c_i(\mathbf{x})=0$ if $\mathbf{x}\models\varphi_i$; otherwise, $c_i(\mathbf{x})$ assigns a fixed penalty for the violation. The priority structure Ψ as described in the previous paragraph also offers a unified structure across these formulations. For example, formulations that aggregate penalties using a weighted sum can be viewed as a special case with $M=1$. Formulations with a combination of hard and soft constraints correspond to a hierarchy with $M=2$, with hard constraints in Ψ_1 and soft constraints in Ψ_2. In all cases, the control synthesis problem can be posed as computing a strategy $\mathbf{u}\in\mathcal{U}_G$ that minimizes the overall cost vector $c(\mathbf{x}_\mathbf{u})$ in the lexicographic sense induced by the priority structure Ψ.

Formulating as an instance of Problem 1: The functions c_i can be viewed as rules r_i by defining $r_i(\mathbf{x},\mathbf{u})=c_i(\mathbf{x})$. This allows minimum-violation planning with various task specifications and violation metrics to be naturally represented using a totally preordered rulebook. If reaching a goal state is not required, we simply let $X_G=X$. Under this construction, Problem 1 directly corresponds to the minimum-violation planning problem and an optimal strategy defined in Problem 1 minimizes the penalty $c(\mathbf{x}_\mathbf{u})$ in the lexicographic sense induced by the ordering of Ψ.

C. Constrained Optimal Control

In general, a finite-horizon constrained optimal control problem for continuous-time, time-invariant system can be formulated as [44], [47], [82]

$$\begin{aligned} \operatorname{argmin}_{\mathbf{u}\in\mathcal{U}} \quad & \int_0^T L(\mathbf{x}(t),\mathbf{u}(t))dt+\Phi(\mathbf{x}(T)) \\ \text{s.t.} \quad & \dot{\mathbf{x}}(t)=f(\mathbf{x}(t),\mathbf{u}(t)),\mathbf{x}(0)=x_I \\ & u(t)\in U, x(t)\in X \\ & g(\mathbf{x}(t),\mathbf{u}(t))\leq 0,\ \forall t\in[0,T]. \end{aligned} \tag{4}$$

For computational purposes, a discrete-time formulation is often used, where the integral cost is replaced by a finite sum and the continuous-time dynamics are approximated by a discrete-time model. In this setup, L and Φ represent the stage cost

and terminal cost, respectively. The constraint $g(\mathbf{x}(t),\mathbf{u}(t))\leq 0$ captures various state and control constraints, including bounds, linear inequalities, and set membership constraints. The stability and performance of the system have been extensively studied, especially for linear dynamics $f(x,u) = Ax + Bu$, where A, B are constant matrices, and quadratic cost functions $L(x,u)=x^TQx+u^TRu$, where Q,R are positive semi-definite weight matrices. We refer the reader to [44], [82] for more details.

Control barrier functions (CBFs) have been introduced as a method to ensure that a nonlinear affine control system remains within a safe set defined by $\{x\in X\subset\mathbb{R}^d \mid b(x)\geq 0\}$, where $b:\mathbb{R}^d\to\mathbb{R}$ is a continuously differentiable function [45]–[47]. This corresponds to the constrained optimal control problem in (4) where the system dynamics are of the form $f(\mathbf{x},\mathbf{u})=A(\mathbf{x})+B(\mathbf{x})\mathbf{u}$, with A and B locally Lipschtiz, and the safety constraint is expressed as $g(\mathbf{x}(t),\mathbf{u}(t))=-b(\mathbf{x}(t))$. **Formulating as an instance of Problem 1:** First, we define goal region to be the entire state space, i.e., $X_G = X$. We define a rule to represent the cost function as $r_J(\mathbf{x},\mathbf{u}) = \int_0^T L(\mathbf{x}(t),\mathbf{u}(t))dt+\Phi(\mathbf{x}(T))$ for the continuous-time setting and $r_J(\mathbf{x},\mathbf{u})=\sum_0^{T-1}L(\mathbf{x}(t),\mathbf{u}(t))+\Phi(\mathbf{x}(T))$ for the discrete-time setting. To capture the constraint $g(\mathbf{x}(t),\mathbf{u}(t))\leq 0,\forall t\in[0,T]$, we define $r_g(\mathbf{x},\mathbf{u})=\max\{\max_t g(\mathbf{x}(t),\mathbf{u}(t)),0\}$, which is 0 if and only if the constraint is satisfied at all time and is strictly positive otherwise. We then define the rulebook as $R=\{r_g,r_J\}$ with $r_g>r_J$, indicating that constraint satisfaction takes precedence over cost minimization. Suppose there exists a feasible strategy $\mathbf{u}$ such that $g(\mathbf{x}(t),\mathbf{u}(t))\leq 0,\forall t\in[0,T]$. Since $r_g>r_J$ and the minimum value of $r_g(\mathbf{x},\mathbf{u})$ is 0, any optimal control strategy $\mathbf{u}^*$ defined in Problem 1 must satisfy $r_g(\mathbf{x}_{\mathbf{u}^*},\mathbf{u}^*)=0$ and minimize $r_J(\mathbf{x}_\mathbf{u},\mathbf{u})$ among all feasible strategies $\mathbf{u}$ for which $g(\mathbf{x}(t),\mathbf{u}(t))\leq 0,\forall t\in[0,T]$.

D. Multi-Objective Shortest Path

The multi-objective shortest path (MOSP) problem is concerned with computing optimal paths in a graph $G=(V,E)$, where V is the set of vertices and $E\subseteq V\times V$ is the set of edges. Such graphs are often used to represent networks (e.g., in communication or transportation domains). In our context, the graph represents a discrete-time, discrete state robot model by setting $V=X$ and $E=\{(x,x') \mid x,x'\in V, x'=f(x,u)$ for some $u\in U(x)\}$. Unlike the classical shortest path problem, which involves a single objective and aims to compute a single optimal path, MOSP considers multiple objectives, represented by edge weight functions $w_1,w_2,...,w_N:E\to\mathbb{R}_{\geq 0}$, and aims to compute the set of non-dominated (i.e., Pareto optimal) paths from a source vertex x_I to a goal vertex in X_G. In our context, a path corresponds to a trajectory $\mathbf{x}=\langle x_0,x_1,...,x_{T+1}\rangle$. We denote the weight of $\mathbf{x}$ with respect to the i^{th} objective by $w_i(\mathbf{x})$.

Two main types of objectives have been considered. The first and most common is the sum formulation, where the weight of a path with respect to the i^{th} objective is the sum of the edge weights along the path, i.e., $w_i(\mathbf{x})=\sum_{i=0}^T w_i(x_i,x_{i+1})$. The second is the min-max formulation, which is equivalent to the max-min formulation, also known as the bottleneck formulation [55]. In this case, the weight of a path with respect to the i^{th} objective is the maximum edge weight along the path, i.e., $w_i(\mathbf{x})=\max_{i=0}^T w_i(x_i,x_{i+1})$. This formulation is useful when the objective is to avoid paths with any particularly bad segments. For example, in communication networks, one may want to minimize the worst-case latency along a path or maximize the minimum bandwidth. Similarly, in robotic navigation, the min-max formulation can help avoid the narrowest or most hazardous segments. Unlike the sum formulation, which may favor briefly passing through a highly unsafe region, the min-max formulation prefers long exposure to moderate risk over brief exposure to extreme risk.

A path $\mathbf{x}$ is Pareto optimal if there is no other path $\mathbf{x}'$ that improves at least one objective without worsening others. Extensions of Dijkstra's and A* algorithms have been developed to solve MOSP [51]–[57]. However, the number of Pareto optimal paths can grow exponentially with the number of vertices in the worst case. As a result, the time complexity for computing the full Pareto front is also exponential in the number of vertices in the worst case. For this reason, time complexity is typically expressed in terms of the size of the solution set [51]. A faster and more practical approach is to compute an approximate solution set using the concept of ϵ-approximate domination [51], [58]. **Formulating as an instance of Problem 2:** MOSP can be naturally expressed using a rulebook where every pair of rules is incomparable. Define a rule r_i corresponding to the i^{th} objective as $r_i(\mathbf{x},\mathbf{u})=\sum_{i=0}^T w_i(x_i,x_{i+1})$ for the sum formulation and $r_i(\mathbf{x},\mathbf{u})=\max_{i=0}^T w_i(x_i,x_{i+1})$ for the min-max formulation, where $\mathbf{x}=\langle x_0,\dots,x_{T+1}\rangle$. The rulebook is then defined as $R=\{r_1,...,r_N\}$ with the priority relation satisfying $r_i\not\lesssim r_j$ for all $i\neq j$. Under this construction, Problem 2 exactly recovers the standard MOSP problem.

V. Single-Strategy Control Synthesis

According to Proposition 4, given a rulebook $\mathcal{R}=\langle R,\lesssim\rangle$, we can construct its refinement $\mathcal{R}_T=\langle R_T,\lesssim_T\rangle$, where $\lesssim_T$ is a total order by repeatedly applying *priority refinement* to pairs of incomparable rules and *rule aggregation* to pairs of rules of the same rank. Intuitively, for any pair of incomparable rules, there are two refinement options: assign one rule higher priority than the other, or place them at the same rank. Rules of the same rank must be aggregated. In practice, the choice of added priorities or aggregation functions is guided by relevant regulations, user preferences, or task-specific considerations and implemented accordingly, similar to weight-tuning in weighted-sum approaches. For example, in autonomous driving, a base rulebook may specify that lane-keeping and obstacle-clearance rules are initially incomparable. An implementation can refine this rulebook according to societal norms or design objectives by either prioritizing one rule over the other or placing them at same rank with an appropriate aggregation, analogous to tuning weights in a weighted-sum optimization. Since $\mathcal{R}_T$ is a refinement of $\mathcal{R}$, Definition 7 ensures that incorporating such preferences does not contradict the original rulebook $\mathcal{R}$, i.e., if $\mathbf{u}<_\mathcal{R}\tilde{\mathbf{u}}$, then $\mathbf{u}<_{\mathcal{R}_T}\tilde{\mathbf{u}}$ also holds for any $\mathbf{u},\tilde{\mathbf{u}}\in\mathcal{U}$. Additionally, since the definition of optimality depends only on strict preferences between realizations, one can also apply *rule augmentation* to resolve ties among equivalent realizations.

For single-strategy synthesis, we are only interested in identifying one optimal strategy. As a result, any refinement of $\mathcal{R}$ that resolves ambiguities among equally preferred or incomparable strategies will yield a valid solution. Therefore,

we will work with the refined rulebook $\mathcal{R}_T$, which represents a specific instantiation of $\mathcal{R}$. Without loss of generality, we let $R_T = \{r_1,...,r_N\}$ and $r_i >_T r_{i+1}$ for all $i \in \{1,...,N-1\}$.

A. Approach

First, consider discrete-time, discrete-state systems in (2), which can be represented by a directed graph $G_f = (V_f, E_f)$, where the vertex set $V_f = X$ corresponds to the set of system states and the edge set $E_f \subseteq X \times U \times X$ encodes feasible transitions, with each edge labeled by a control action. Specifically, $(x,u,x') \in E_f$ iff $u \in U(x)$ and $x' = f(x,u)$.

We define a *path* on G_f as a sequence of edge labels $\mathbf{u} = \langle u_0, u_1, \dots, u_T \rangle$ that results in a state sequence $\mathbf{x_u} = \langle x_0,...,x_{T+1} \rangle$ where $x_0 = x_I$ and $(x_t,u_t,x_{t+1}) \in E_f$, $\forall t$. By defining a path in terms of edge labels, rather than the usual sequence of states, a path directly corresponds to a control strategy. This formulation also allows for multiple distinct actions from a given state to result in the same successor state.

Assumptions on rulebooks: We impose structural requirements on how rules are evaluated over trajectories. Specifically, we assume that each rule can be incrementally evaluated by combining its evaluations over sub-trajectories using a binary operation $*$. This operation must also satisfy algebraic properties that ensure consistency and compatibility with ordering. These requirements are formalized below.

Assumption 1. For each rule $r_i \in R_T$, there exists an associated binary operation $*$ that satisfies the following properties.

(A1.1) For any control strategy $\mathbf{u} = \langle u_0, u_1, \dots, u_T \rangle \in \mathcal{U}$ and robot trajectory $\mathbf{x} = \langle x_0, x_1, \dots, x_{T+1} \rangle$, $r_i(\mathbf{x}, \mathbf{u}) = r_i(\mathbf{x}_{0:t+1}, \mathbf{u}_{0:t}) * r_i(\mathbf{x}_{t+1:T+1}, \mathbf{u}_{t+1:T})$, $\forall t \in \{0,...,T-1\}$.

(A1.2) 0 is the identity element, i.e., for all $a \in \mathbb{R}_{\geq 0}$, $0 * a = a * 0 = a$.

(A1.3) The operation $*$ is *isotone*, i.e., for all $a,b,c, \in \mathbb{R}_{\geq 0}$, if $a \leq b$, then $a * c \leq b * c$ and $c * a \leq c * b$.

(A1.4) For all $a,b,c,d \in \mathbb{R}_{\geq 0}$, if $a * c = a * d = b * c$, then $b * d \leq a * c$.

Assumption (A1.1) allows each rule to be encoded as an edge weight on G_f. Specifically, for each edge $(x,u,x') \in E$, the weight with respect to rule r_i is given by $r_i(\langle x,x' \rangle, \langle u \rangle)$. The weight of any path on G_f with respect to r_i can then be computed by aggregating the weights of individual edges along the path using the binary operation $*$, i.e., $r_i(\mathbf{x}, \mathbf{u}) = r_i(\mathbf{x}_{0:1}, \mathbf{u}_{0:0}) * r_i(\mathbf{x}_{1:2}, \mathbf{u}_{1:1}) * \cdots * r_i(\mathbf{x}_{T:T+1}, \mathbf{u}_{T:T})$. In other words, the operation $*$ defines how the edge weights are aggregated under rule r_i and may differ across rules. Common choices of $*$ that satisfy Assumption 1 include addition and max operations. If $*$ is the addition operation, the total weight of a path is simply the sum of the individual edge weights (as in the sum formulation in MOSP). If $*$ is the max operation, the total weight of a path is the maximum edge weight along the path (as in the min-max formulation in MOSP).

Let $*_i$ denote the binary operation associated with rule r_i that satisfies Assumption 1. Then, Assumptions (A1.2) and (A1.3) ensure that $(\mathbb{R}_{\geq 0}, *_i, 0, \leq)$ forms a *cost monoid* and Assumption (A1.4) further ensures that this cost monoid is *regular* as defined in [83]. Since $\lesssim_T$ is a total order on R_T, the rulebook $\mathcal{R}_T$ can be represented as a *regular multicost*, i.e., a prioritized product of regular cost monoids, equipped with coordinate-wise aggregation (using the corresponding $*_i$ operations) and lexicographic comparison. As a result, Problem 1 reduces to computing an optimal path on G_f, where each edge $e = (x,u,x') \in E_F$ is assigned an N-dimensional weight vector $\mathbf{r}(e) = [r_1(\langle x,x' \rangle, \langle u \rangle), ..., r_N(\langle x,x' \rangle, \langle u \rangle)]$. These edge weights are aggregated along a path using coordinate-wise operation of the corresponding $*_i$ and are compared using standard lexicograhic ordering.

Efficient solution to Problem 1: Given well-defined edge weights and aggregation and comparison operations, one may consider applying standard graph search algorithms such as Dijkstra's or A* [38], [84]–[86]. However, as shown in [83], these traditional algorithms do not necessarily return an optimal path when edge weights are elements of a regular multicost because such weights do not necessarily inherit the properties of a cost monoid, which are required for dynamic programming principles to apply (i.e., Bellman's principle of optimality). For example, let $\mathbf{R}_+ = (\mathbb{R}_{\geq 0}, +, 0, \leq)$ denote the standard additive cost monoid and $\mathbf{R}_{\max} = (\mathbb{R}_{\geq 0}, \max, 0, \leq)$ denote the max-based cost monoid. Then, the product $\mathbf{R}_+ \times \mathbf{R}_{\max}$ forms a cost monoid but $\mathbf{R}_{\max} \times \mathbf{R}_+$ does not. As a result, if a rule r_i aggregates edge weights via the max operation and a subsequent rule r_{i+1} uses addition, then the resulting edge weights may not satisfy the axioms of a cost monoid, leading to the failure of traditional graph search algorithms to return an optimal path.

To address this issue, we adopt the iterative algorithm proposed in [83], which is specifically designed to handle graphs whose edge weights are elements of a regular multicost. Algorithm 1 presents an adaptation of this algorithm for our setting. In each iteration $i \in \{1, \dots, N\}$, the algorithm extracts the optimal subgraph G_i of the previous graph G_{i-1}, considering only the single rule r_i and its associated aggregation operation $*_i$. This is done using the procedure $\texttt{OptimalSubgraph}(G_{i-1}, x_I, X_G, r_i, *_i)$, which returns the subgraph of G_{i-1} that contains only those directed edges that lie on some optimal path (with respect to r_i) from the initial vertex x_I to a goal vertex in X_G. Formally, let E_{i-1} be the set of edges of G_{i-1}. An edge $e = (x,u,x') \in E_{i-1}$ belongs to the optimal subgraph of G_{i-1} if and only if $x.come * r_i(e) * x'.go \leq c$, where c is the minimum cost from x_I to X_G in G_{i-1} under scalar edge weights defined by r_i and aggregator $*_i$. Here, $x.come$ denotes the optimal cost-to-come from x_I to x and $x'.go$ denotes the optimal cost-to-go from x' to X_G, both computed using scalar edge weights defined by r_i and aggregator $*_i$. These costs can be computed efficiently using Dijkstra's algorithm, one forward search from x_I to compute $x.come$ and one backward search on the reverse graph from X_G to compute $x'.go$, each using only r_i and $*_i$. Note that this computation ignores all rules except r_i, focusing only on optimizing with respect to a single rule at a time.

After N iterations, the algorithm produces the subgraph G_N. Assumption (A1.4) ensures that every path in the optimal subgraph is itself optimal (cf. Fig. 4). Therefore, any path from x_I to a goal state in X_G within this subgraph is guaranteed to be optimal with respect to $\mathcal{R}_T$. An optimal path $\mathbf{u}^*$ can then be extracted using any graph traversal method such as breadth-first search as in $\texttt{BFS}(G_N, x_I, X_G)$ or depth-first search.

Algorithm 1: Single-strategy control synthesis for discrete-time, discrete-state systems

Require: Graph $G_f = (V_f, E_f)$ representing system in (2), rules $r_1,...,r_N$ with $r_i >_T r_{i+1}, \forall i$ and their associated binary operators $*_1,...,*_N$

Ensure: Optimal strategy $\mathbf{u}^* \in \mathcal{U}$ of (2) with respect to $\mathcal{R}_T$

$G_0 = G_f$
for $i \in \{1,...,N\}$ **do**
  $G_i = \texttt{OptimalSubgraph}(G_{i-1}, x_I, X_G, r_i, *_i)$
end for
$\mathbf{u}^* = \texttt{BFS}(G_N, x_I, X_G)$
return $\mathbf{u}^*$

B. Analysis

Complexity: Algorithm 1 computes an optimal path in time polynomial in the size of the vertex set V_f and the number of objectives N. Specifically, each iteration i consists of two calls to Dijkstra's algorithm using scalar edge weights corresponding to the rule r_i, followed by a linear pass over the edges to extract the optimal subgraph. The algorithm performs up to N such iterations, one for each rule in the rulebook. As a result, its overall asymptotic time complexity is $O(NT_D)$, where T_D is the time complexity of running Dijkstra's algorithm on G_f using standard scalar-valued edge weights (without incorporating rulebook-based prioritization). For example, when using a Fibonacci heap, $T_D \in O(|E_f| + |V_f| \log |V_f|)$ We refer the reader to [83] for the full details of the `OptimalSubgraph` procedure and time complexity analysis of Algorithm 1.

To simplify the presentation, we describe Algorithm 1 as performing N iterations, one for each rule in the rulebook. In practice, the number of iterations can be reduced by grouping together consecutive rules, provided that the product of their associated cost monoids remains a cost monoid. Such grouped rules can be processed together in a single iteration. As a result, the number of iterations needed is the number of these rule groups. For example, both the products $\mathbf{R}_+ \times \cdots \times \mathbf{R}_+$ and $\mathbf{R}_+ \times \cdots \times \mathbf{R}_+ \times \mathbf{R}_{\max}$ form a valid cost monoid regardless of the number of $\mathbf{R}_+$ terms. Thus, if the rulebook contains a mix of addition and max aggregators, the number of iterations is at most the number of rules using the max operation.

Correctness: Assumption 1 is introduced to ensure that the edge weights on G_f with respect to each rule r_i form a regular cost monoid. This, in turn, ensures that the edge weights under a rulebook, represented as an N-dimensional vector $\mathbf{r}(e) = [r_1(\langle x,x' \rangle, \langle u \rangle),...,r_N(\langle x,x' \rangle, \langle u \rangle)]$ for each edge $e = (x,u,x') \in E_f$, form a regular multicost. This property allows us to apply the result from [83], which establishes that Algorithm 1 returns an optimal strategy when edge weights form a regular multicost.

Proposition 5. Suppose a rulebook $\mathcal{R}$ satisfies Assumption 1. Then, Algorithm 1 returns an optimal control strategy as defined in Problem 1.

C. Extensions

Handling temporal logic specifications: Many variants of temporal logic formulas can be algorithmically translated into equivalent automata representations [5]. For example, a formula in LTL or LTL$_X$ can be translated into a Büchi automaton that accepts precisely the infinite words that satisfy the formula. A formula in FLTL, FLTL$_X$ or syntactically co-safe LTL can be translated into a finite automaton. Following minimum-violation planning approaches, e.g., in [67], [78], we translate a formula φ into an automaton $\mathcal{A}_\varphi$ and augment it into a weighted automaton $\overline{\mathcal{A}}_\varphi$. This weighted automaton is constructed such that it accepts all input words, with the weight of the shortest accepting run on a word ω represeting the degree of violation of φ. Given $\overline{\mathcal{A}}_\varphi$ and a finite transition system $\mathcal{T}$ representing the robot model in (2), we construct the product automaton $\mathcal{P} = \mathcal{T} \otimes \overline{\mathcal{A}}_\varphi$. A shortest run in $\mathcal{P}$ from an initial state to an accepting state corresponds to a robot trajectory that minimizes the violation of φ. As a result, to handle rulebooks that include temporal logic specifications, we can apply Algorithm 1 on the graph induced by $\mathcal{P}$ instead of G_f, treating the states and transitions of the product automaton as the underlying graph structure. We demonstrate this approach through a case study in Section V-D.

Handling continuous-state systems: Handling continuous-state systems is generally challenging and existing approaches often rely on restrictive assumptions on the robot model or relatively simple objectives. Among the most promising approaches for handling complex objectives and robot models are sampling-based motion planning algorithms [9], [10], [32], [66]–[68]. In summary, these algorithms incrementally build a graph G_f (a tree in the case of RRT* or a general graph in RRG or PRM*) to approximate the reachable state space or its product with a finite automaton representing a temporal logic specification. G_f is initialized with a single vertex corresponding to x_I. At each iteration, a new state is randomly sampled from the continuous state space X and connections are attempted between the sampled state and a set of nearby vertices in G_f, using a steering function that respects the robot model. It has been shown that under certain conditions, the algorithm is asymptotically optimal, meaning that it converges to the true optimal solution with probability 1 as the number of samples approaches infinity. To ensure this convergence in our setting, additional assumptions on the rules are required beyond Assumption 1 as stated below.

Assumption 2. For each rule $r_i \in R_T$, there exists a constant k_i such that $r_i(\mathbf{x_u}, \mathbf{u}) \leq k_i \mathrm{TV}(\mathbf{x_u})$ for all $\mathbf{u} \in \mathcal{U}$, where $\mathrm{TV}(\mathbf{x_u})$ denotes the total variation (i.e., length) of $\mathbf{x_u}$. Intuitively, this means that the total rule violation along a trajectory grows at most linearly with its length, which is a standard assumption used to guarantee asymptotic optimality of sampling-based algorithms when connection and rewiring decisions are made using a distance metric [66].

With Assumptions 1 and 2, standard sampling-based methods such as RRG or RRT* [66] can be applied to incrementally construct a graph G_f that serves as a finite approximation of the robot model. An optimal policy can then be extracted using Algorithm 1. Following the proof in [67], [68], the resulting trajectory converges almost surely to the true optimal trajectory $\mathbf{x_{u^*}}$, where $\mathbf{u}^*$ is the optimal strategy defined in Problem 1, as the number of vertices in G_f approaches infinity. This is formalized in the following proposition.

Proposition 6. Let $\mathbf{u}^*$ be a solution of Problem 1 and $\mathbf{x}^*$ be the corresponding trajectory. Let $\mathbf{u}_n$ be an optimal solution obtained from graph G_f after n iterations and let $\mathbf{x}_n$ be the

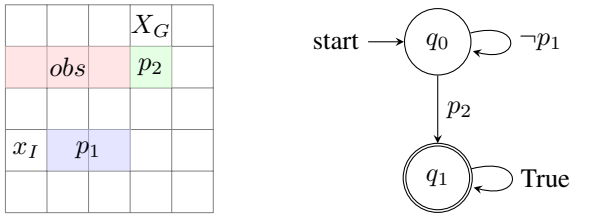


Fig. 3: [Left] A 5×5 grid world showing the robot's initial state x_I, goal region X_G, obstacle region obs (red), and regions satisfying propositions p_1 (blue) and p_2 (green). [Right] A finite-state automaton $\mathcal{A}_\varphi$ for the FLTL formula $\varphi=\neg p_1\ \mathbf{U}\ p_2$.

corresponding trajectory. Then, under Assumptions 1 and 2, the sequence $\mathbf{x}_n$ converges almost surely to $\mathbf{x}^*$ in the bounded variation norm: $\mathbb{P}\Big(\lim_{n\to\infty}\|\mathbf{x}_n-\mathbf{x}^*\|_{\mathrm{BV}}=0\Big)=1$.

D. Case Studies: Discrete-State System

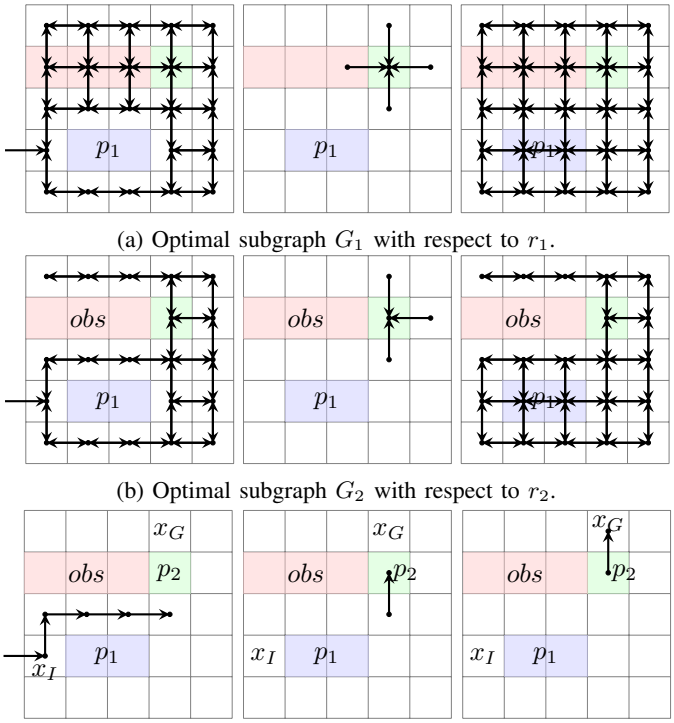
To illustrate the expressive power of our formulation and facilitate a step-by-step walkthrough of the algorithm, we consider a simple example that emphasizes the diversity of objectives rather than the complexity of the robot model. Specifically, the example combines temporal logic specifications, min-max formulations, and standard additive cost functions, a mix that, to the best of our knowledge, has not been addressed in prior work. We show how these heterogeneous objectives can be handled efficiently within a unified framework, and how traditional planning approaches may fail in this setting.

Setup: We consider a robot navigating a 5 x 5 grid world shown in Fig. 3. The set of states and control actions are $X=\{(i,j)\mid i,j\in\{1,...5\}\}$ and $U=\{(0,1),(0,-1),(1,0),(-1,0)\}$, corresponding to up, down, right, and left movements, respectively. The initial state and goal regions are $x_I=(1,2)$ and $X_G=\{x_G\}$, where $x_G=(4,5)$. For each state $x\in X$, the set of available control actions is $U(x)=\{u\in U\mid f(x,u)\in X\}$, where the state transition function f is given by $f(x,u)=(x_x+u_x,x_y+u_y)$, $\forall x=(x_x,x_y)\in X$, $u=(u_x,u_y)\in U$.

The red region labeled obs denotes cells occupied by an obstacle. Additionally, the grid includes labeled regions where atomic propositions p_1 and p_2 hold. We define a rulebook with $R=\{r_1,r_2,r_3\}$ that captures a mix of objective types:

- r_1 encodes the FLTL formula $\varphi=\neg p_1\ \mathbf{U}\ p_2$, requiring the robot to visit p_2 while avoiding p_1 until then. The rule $r_1(\mathbf{x},\mathbf{u})$ quantifies the violation of φ, measured as the minimum number of time steps that must be removed from $\mathbf{x}$ to satisfy φ, based on the definition in [67], [78].
- r_2 enforces a clearance of at least 2 grid steps from obstacles, using a min-max formulation: $r_2(\mathbf{x},\mathbf{u})=\max_{x\in\mathbf{x}}\max(0,\,2-d(x,X_{obs}))$, where $d(x,X_{obs})$ is the minimum number of moves (in the grid graph) required to reach the obstacle region X_{obs} from state x. In other words, the rule penalizes the worst-case violation of a 2-cell clearance requirement along the trajectory.
- r_3 encodes a path efficiency as a standard additive cost: $r_3(\mathbf{x},\mathbf{u})=\|\mathbf{u}\|$, which counts the number of control actions taken, or equivalently, the length of the trajectory.

Results: Let $AP=\{p_1,p_2\}$ be the set of atomic propositions. To handle the FLTL formula φ in r_1, we translate φ into a finite automaton $\mathcal{A}_\varphi$ with the set of states $Q=\{q_0,q_1\}$, initial state q_0, input alphabet 2^{AP}, set of accepting states $F=\{q_1\}$, and the non-deterministic transition relation shown in Fig. 3. The finite transition system representing the robot model is defined as $\mathcal{T}=(X,x_I,R,AP,L)$, where $R=\{(x,f(x,u))\mid x\in X,u\in U(x)\}\subseteq X\times X$ is the transition relation, and $L:X\to2^{AP}$ is a labeling function such that $p_1\in L(x)$ if and only if x lies in the region labeled p_1 and similarly for p_2. The product automaton is given by $\mathcal{P}=\mathcal{T}\otimes\overline{\mathcal{A}}_\varphi=(Q_\mathcal{P},q_{init,\mathcal{P}},\delta_\mathcal{P},F_\mathcal{P})$, where $Q_\mathcal{P}=(X\cup\{init\})\times Q$, $q_{init,\mathcal{P}}=(init,q_0)$, $F_\mathcal{P}=X_G\times F=\{(x_G,q_1)\}$, and the transition relation $\delta_\mathcal{P}$ includes (1) $((x,q),(x',q))$ for all $(x,x')\in R$, $q\in Q$; (2) $((x,q_0),(x',q_1))$ for all $(x,x')\in R$ if $p_2\in L(x')$; and (3) $((init,q),(x_I,q))$ for all $q\in Q$.



(a) Optimal subgraph G_1 with respect to r_1.

(b) Optimal subgraph G_2 with respect to r_2.

(c) Optimal subgraph G_3 with respect to r_3.

Fig. 4: The optimal subgraphs over iterations, split into 3 components based on the type of transition in the product automaton: [Left] Transitions of the form $(x,q_0)\to(x',q_0)$; [Middle] Transitions of the form $(x,q_0)\to(x',q_1)$; [Right] Transitions of the form $(x,q_1)\to(x',q_1)$, for all $x,x'\in X$. The arrow originating from outside of the grid denotes the initial state $q_{init,\mathcal{P}}=(init,q_0)$.

The product automaton $\mathcal{P}$ induces a directed graph $G_\mathcal{P}$, where vertices are $Q_\mathcal{P}$, the initial vertex is $q_{init,\mathcal{P}}$, the goal region is $F_\mathcal{P}=\{(x_G,q_1)\}$, and the edges are defined by $\delta_\mathcal{P}$. The rule r_1 assigns weights to the edges in $G_\mathcal{P}$ as follows

$$
\begin{aligned}
r_1(\langle(x,q_0),(x',q_0)\rangle) &= \begin{cases}0 & \text{if } p_1\notin L(x')\\ 1 & \text{otherwise}\end{cases}\\
r_1(\langle(x,q_1),(x',q_1)\rangle) &= 0\\
r_1(\langle(x,q_0),(x',q_1)\rangle) &= 0 \text{ if } p_2\in L(x')\\
r_1(\langle(init,q),(x_I,q)\rangle) &= 0,\forall q\in Q,
\end{aligned}
$$

for all $(x,x')\in R$. We then apply Algorithm 1 to $G_\mathcal{P}$. Fig. 4 shows the optimal subgraph in each iteration. To aid visualization, we partition each subgraph into 3 components

based on the structure of $\delta_{\mathcal{P}}$: $(x,q_0)\to(x',q_0)$, $(x,q_0)\to(x',q_1)$, and $(x,q_1)\to(x',q_1)$ for all $x,x'\in X\cup\{init\}$. The first component ($(x,q_0)\to(x',q_0)$, left figure) corresponds to transitions before the automaton recognizes that p_2 has been satisfied. To avoid violating φ, the robot must satisfy $\neg p_1$. Thus, G_1 excludes any transitions that would enter the p_1 region while in automaton state q_0. The second component ($(x,q_0)\to(x',q_1)$, middle figure) captures transitions that enter the p_2 region. The third component ($(x,q_1)\to(x',q_1)$, right figure) captures transitions after p_2 has already been satisfied. At this stage, the robot is free to visit any region without incurring additional cost under r_1 and G_1 includes all such transitions. In summary, the subgraph, G_1 includes only edges with zero cost under the highest priority rule r_1, ensuring that the robot satisfies φ.

The optimal subgraph G_2 incurs a penalty of 1 with respect to r_2. This is because there is no path from the initial state $q_{init,\mathcal{P}}=(init,q_0)$ to the goal state (x_G,q_1) that maintains a clearance of at least 2 grid steps from the *obs* region. To reach the goal state, the automaton must transition from q_0 to q_1, which requires the robot to visit the p_2 region. However, this region lies only one move away from the obstacle, making it impossible to satisfy the clearance requirement. As a result G_2 consists of all edges from G_1 whose source and target states are at least 1 grid step away from *obs*, representing the minimal violation of r_2 necessary to satisfy r_1.

Finally, G_3 is the subgraph of G_2 containing only the edges that lie along a shortest path, i.e., the path with the fewest number of moves from $(init,q_0)$ to (x_G,q_1). In this example, the optimal control strategy consists of 6 moves: $\mathbf{u}^*=\langle(0,1),(1,0),(1,0),(1,0),(0,1),(0,1)\rangle$, with trajectory $\mathbf{x}_{\mathbf{u}^*}=\langle(1,2),(1,3),(2,3),(3,3),(4,3),(4,4),(4,5)\rangle$.

Comparison with Dijkstra's algorithm: We compare our approach with Dijkstra's algorithm because it serves as the core search routine in most state-of-the-art minimum-violation planning approaches, which differ primarily in how the product automaton $\mathcal{P}$ is constructed, but ultimately rely on Dijkstra's algorithm to extract optimal paths in $\mathcal{P}$. Since our method also constructs $\mathcal{P}$, this comparison isolates the main difference in how optimal paths are computed. The paths returned by Algorithm 1 and Dijkstra's algorithm are shown in Fig. 5. Both paths satisfy φ, incurring 0 cost with respect to r_1 but incur a penalty of 1 with respect to r_2. However, the path returned by Dijkstra's algorithm has a higher cost under r_3. This occurs because the edge weight structure here does not form a cost monoid. As a result, the Bellman optimality principle does not hold and greedy algorithms like Dijkstra's can fail to produce an optimal solution under this rulebook. For example, consider the subpath from the initial state to the cell immediately below the p_2 region. Along this subpath, Dijkstra's path incurs 0 cost under both r_1 and r_2, while the corresponding subpath of the optimal path incurs a penalty of 1 under r_2. However, to ultimately reach the goal, the robot must pass through the p_2 region, which is 1 cell away from the *obs* region. Therefore, the final portion of the path will necessarily incur a penalty of 1 under r_2, regardless of the earlier subpath. Since Dijkstra's algorithm greedily optimizes the cost at each step, it fails to anticipate this unavoidable penalty and ends up selecting a longer path that appears better locally but performs worse globally with respect to the rulebook.

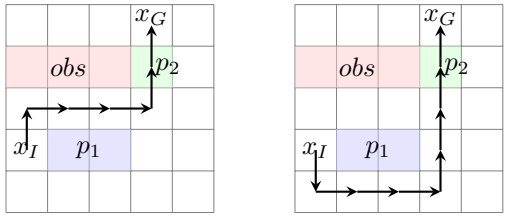


Fig. 5: Failure of Dijkstra's algorithm to find an optimal path. [Left] Optimal path computed by Algorithm 1 [Right] Optimal path computed by Dijkstra's algorithm. In both cases, the paths are projected onto the X-component of the state.

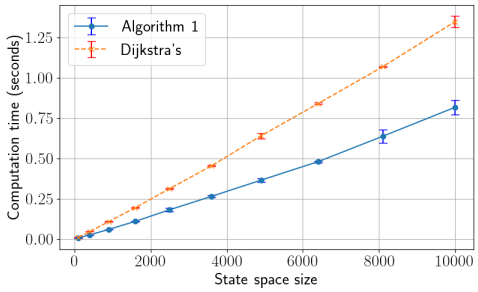


Fig. 6: Computation time for Algorithm 1 vs. Dijkstra's.

Fig. 6 shows how the computation time for both Algorithm 1 and Dijkstra's algorithm increases almost linearly with the number of vertices in the graph. However, Algorithm 1 consistently runs faster. This is because, when applying Dijkstra's algorithm with edge weights derived from the rulebook, each edge weight is an N-dimensional vector that must be compared lexicographically to manage the priority queue. In addition, combining edge weights along a path requires vector operations. In contrast, Algorithm 1 runs up to N iterations of the same Dijkstra's algorithm, but each iteration only handles scalar edge weights corresponding to a single rule. This simplifies both the comparison and accumulation operations, resulting in lower computational overhead per iteration and improved overall performance.

E. Case Studies: Continuous-State System

Setup: We consider a warehouse robot navigating a 2D workspace subject to multiple prioritized rules. The robot must avoid obstacles (r_1), stay out of a busy region to prevent interference with human operators (r_2), maintain a specified clearance from obstacles (r_3), and follow a right-hand rule (r_4), a common strategy in multi-robot systems to mitigate deadlocks by requiring the robots to pass obstacles and each other on the right [87], [88]. Specifically, the right-hand rule requires that the ray extending orthogonally to the robot's right does not intersect any obstacle.

The violation of each rule is measured as the total path length over which the rule is violated, i.e., $r_i(\mathbf{x},\mathbf{u})=\int_0^T\phi_i(\mathbf{x}(t))\|\dot{\mathbf{x}}(t)\|dt$, where ϕ_i is the violation indicator for rule r_i, i.e., $\phi_i(x)=1$ if state x violates rule r_i and $\phi_i(x)=0$ otherwise. We define these indicators as $\phi_1(x)=1$ iff $x\in\mathcal{O}$ and $\phi_2(x)=1$ iff $x\in\mathcal{B}$, where $\mathcal{O},\mathcal{B}\subseteq\mathbb{R}^2$ denote the obstacle and busy regions, respectively. $\phi_3(x)=1$ iff the Euclidean distance from x to $\mathcal{O}$ is below the required clearance. $\phi_4(x)=1$ iff the ray $\{x+\lambda n_r\mid\lambda\geq 0\}$ intersects $\mathcal{O}$, where n_r is the unit vector obtained by rotating the path tangent clockwise 90°. Because violation measure grows proportionally with the

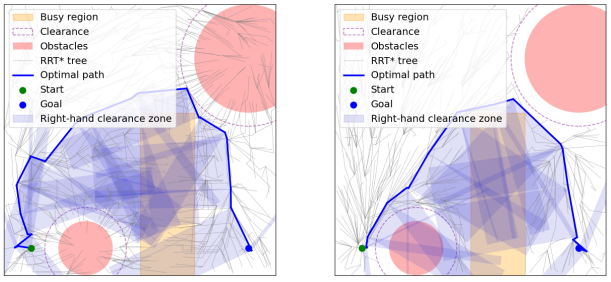


Fig. 7: RRT* planning results with 1,000 iterations. Grey lines show the explored tree. The blue line indicates the final path from start (green) to goal (blue). Circular (red) and rectangular (yellow) regions denote obstacles and busy region. Dashed (red) circles represent clearance constraints. The shaded (blue) regions on the right-hand side of the path indicate areas that should remain free of obstacles according to the right-hand rule. [Left] Our rulebook-based RRT* produces a path satisfying all requirements ($r_1 = r_2 = r_3 = r_4 = 0$) with length $r_5 = 18.13$. [Right] Classical RRT* treats obstacles, busy regions, and clearance as collisions, yielding a shorter path ($r_5 = 15.19$) but violates the right-hand rule, as the shaded zone intersects obstacles.

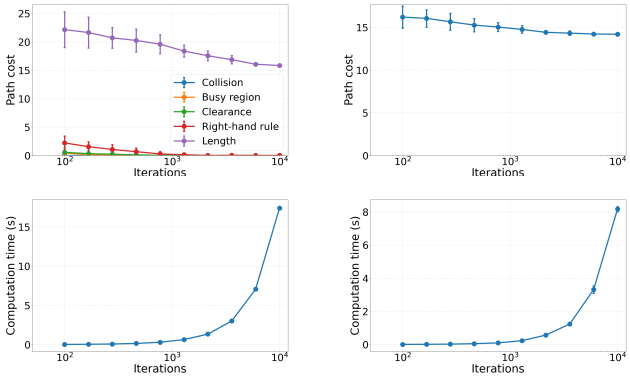


Fig. 8: Path cost and computation time vs. the number of iterations. [Left] Our approach. [Right] Classical RRT*.

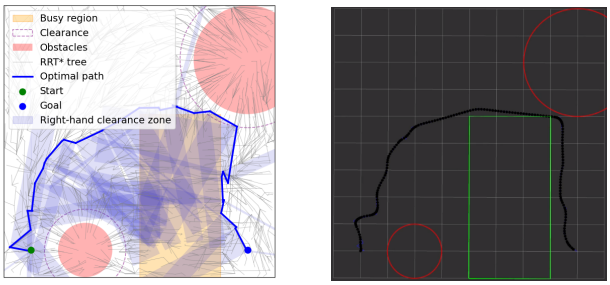


Fig. 9: [Left] When requirements conflict, our rulebook-based RRT* finds a path with small violations, while the classical RRT* fails to find a valid solution. [Right] Turtlebot 3 navigating this scenario in Gazebo using a plan generated by the rulebook-based RRT*. Note that the path differs from the left due to the inherent randomness of RRT*.

trajectory length, the definition of $r_i(\mathbf{x},\mathbf{u})$ satisfies Assumption 2. Finally, the robot should minimize the path length (r_5). The relationship between the rules is defined as $r_1 > r_2 > r_3 > r_4 > r_5$.

Results: Fig. 7 compares the paths generated by our rulebook-based RRT* and the classical RRT*, both using 1,000 samples. Our approach generates a path that satisfies all requirements, while the classical RRT* violates the right-hand rule despite yielding a shorter path. Note the more intricate structure in the early portion of the rulebook-based path, where the algorithm deliberately bends the trajectory to keep the right-hand ray clear of the obstacle. On the other hand, RRT* selects a more direct path but violates the right-hand constraint.

The average computation time for our approach is 444.75 ms, compared to 295 ms for the classical RRT*. Fig. 8 illustrates the convergence of path cost for both methods. While the computation time of the rulebook-based RRT* follows a similar growth trend to that of the classical RRT*, it is roughly twice as large for the same number of samples due to additional cost evaluations and the need to compare multi-dimensional cost vectors (five elements for the rulebook-based approach versus a single scalar for the classical RRT*).

Fig. 9 presents a modified scenario where the busy region overlaps with the clearance zone around an obstacle, making it impossible to satisfy both r_2 and r_3 simultaneously. In this case, the rulebook-based RRT* still finds a feasible path that does not violate r_1 and r_4, while incurring small violations of $r_2 = 0.24$ and $r_3 = 1.36$. We also execute a plan generated by our rulebook-based RRT* in a high-fidelity Gazebo simulation using TurtleBot 3 and a pure-pursuit controller, demonstrating that the robot can successfully navigate the scenario.

VI. Complete Control Synthesis

This section focuses on computing the (maximum) complete set of all optimal control strategies for discrete-time, discrete-state systems as defined in Problem 2. Such a set is particularly useful for verification purposes, e.g., to check whether a given control strategy $\mathbf{u}$ is optimal with respect to a specified rulebook $\mathcal{R}$. As a concrete example, a regulatory authority may want to determine whether an observed behavior of an autonomous vehicle complies with the applicable rulebook. Since this type of analysis is typically performed offline, the computational constraints are less strict than those for the single-strategy synthesis problem.

A. Approach

We represent the system by a directed graph $G_f = (V_f, E_f)$ as in Section V-A and impose Assumption 1 with the additional restriction that the operation $*$ associated with each rule is the addition operation.

Assumption 3. For each rule $r_i \in R_T$, the total cost of a trajectory under policy $\mathbf{u}$ is the sum of the stage-wise costs: $r_i(\mathbf{x},\mathbf{u}) = \sum_{t=0}^{T} r_i(\mathbf{x}_{t:t+1}, \mathbf{u}_{t:t})$.

Furthermore, to avoid degenerate cases where the robot can loop indefinitely without incurring any penalty, we assume the absence of zero-cost loops in G_f. In particular, we require that for any nontrivial cycle in G_f, at least one rule (typically low-priority, efficiency-related rules such as path length, traversal time, or energy consumption) assigns a nonzero cost.

Assumption 4. Given any state $x \in X$, suppose there exists a control strategy $\mathbf{u} = \langle u_0, u_1, \ldots, u_T \rangle$ with $T \geq 0$ that produces a trajectory $\mathbf{x}_\mathbf{u} = \langle x_0, \ldots, x_{T+1} \rangle$ with $x_0 = x_{T+1} = x$. Then, there must exist a rule r_i such that $r_i(\mathbf{x}_\mathbf{u}, \mathbf{u}) > 0$.

Algorithm 2 extends the label-setting algorithm for solving MOSP [51], [52] to compute the set of optimal strategies $\mathcal{U}_G^*$. It can also be seen as a generalization of Dijkstra's algorithm to handle multiple optimal paths. The main idea is to build a search tree $T=(V_T,E_T)$ that represents the set of optimal strategies identified so far from x_I to each visited state. Each vertex $v\in V_T$ corresponds to some state $x\in X$. The root of the tree, denoted by v_1, corresponds to the initial state x_I. Given a state $x\in X$, there may be multiple optimal strategies to reach x; thus, there may be multiple vertices in V_T corresponding to the same state x. To track the association between vertices in V_T and states in X, we use a function $\mathbf{x}:V_T\to X$, i.e., $\mathbf{x}(v)\in X$ is the state associated with vertex $v\in V_T$. Note that Dijkstra's algorithm also constructs such T by keeping track of the best parent for each state. However, since Dijkstra's algorithm tracks at most one optimal path to each state, $\mathbf{x}(v)$ is one-to-one, i.e., each vertex in T corresponds to a unique state in X.

An edge in T is labeled by an action $u\in U$ and $(v,u,v')\in E_T$ only if $u\in U(\mathbf{x}(v))$ and $f(\mathbf{x}(v),u)=\mathbf{x}(v')$, i.e., applying action u at the state associated with v transitions the robot to the state associated with v'. For each $v\in V_T$, let $path(v)$ denote the path from v_0 (the root of T) to v, represented by a sequence of labels of the corresponding edges. Thus, $path(v)$ represents a valid strategy for x_I to reach the state $\mathbf{x}(v)$.

The priority queue Q holds the leaf vertices of T that may have successors. We use $<_{\mathcal{R}}$ on $path(v)$ as the priority function to order Q. Specifically, $path(v)<_{\mathcal{R}} path(v')$ implies that vertex v has higher priority than vertex v' in Q. For each state $x\in X$, we use the variable $\mathcal{U}_x$ to maintain the set of optimal strategies known so far from x_I to x. Each strategy $\mathbf{u}\in\mathcal{U}_x$ corresponds to a unique vertex $v\in V_T$ with $\mathbf{x}(v)=x$. To track this association for each $x\in X$, we use a function $\mathbf{v}_x:\mathcal{U}_x\to V_T$. Specifically, $\mathbf{v}_x(\mathbf{u})$ is the vertex $v\in V_T$ associated with the strategy $\mathbf{u}$.

Lines 1-5 of Algorithm 2 initialize T with a single vertex v_0, which serves as the root of T and corresponds to the initial state x_I. At this point, the only optimal path from v_0 to itself is the empty path $\langle\rangle$. Initially, Q contains only the root vertex v_0. The algorithm then enters a while loop that continues until Q is empty. Assumption 4 ensures that vertices popped from Q correspond to paths that do not pass through the same state more than once; hence the algorithm does terminate eventually.

In each iteration of the while loop, the algorithm removes the vertex v with the highest priority from Q. Based on the priority function to sort Q, v corresponds to a strategy $\mathbf{u}$ such that no other strategy $\mathbf{u}'$ associated with vertices in Q satisfies $\mathbf{u}'<_{\mathcal{R}}\mathbf{u}$. The algorithm then considers each possible action $u\in U(\mathbf{x}(v))$, which leads to the next state $x'=f(x,u)$, and constructs a candidate strategy $\mathbf{u}=path(v)\oplus\langle u\rangle$, which is obtained by appending u to $path(v)$. The function $\mathcal{U}_{x'}.insert(\mathbf{u})$, defined in Algorithm 3, evaluates whether the candidate strategy $\mathbf{u}$ is an optimal strategy known so far to reach x' and updates $\mathcal{U}_{x'}$ accordingly. Specifically, if $\mathbf{u}$ is optimal, it inserts $\mathbf{u}$ into $\mathcal{U}_{x'}$ and removes and returns the set $\mathcal{U}_{sub}\subseteq\mathcal{U}_{x'}$ of strategies that are no longer optimal due to the discovery of $\mathbf{u}$, i.e., those strategies $\mathbf{u}'\in\mathcal{U}_{x'}$ for which $\mathbf{u}<_{\mathcal{R}}\mathbf{u}'$ holds. Line 13 of Algorithm 2 then removes the vertices in Q corresponding to these suboptimal strategies. To keep T small, we can optionally remove $\mathbf{v}_{x'}(\mathbf{u}')$ from T but this does not affect the correctness of the algorithm.

Algorithm 2: Compute the set $\mathcal{U}_G^*$ of optimal plans

Require: The set X of states, the initial state $x_I\in X$, the set $X_G\subseteq X$ of goal states, the set $U(x)\subseteq U$ of available actions of the robot at each state $x\in X$, and the state transition function $f:X\times U\to X$

Ensure: The set $\mathcal{U}_G^*$ of optimal plans

1: Initialize $\mathcal{U}_x=\begin{cases}\{\langle\rangle\}, & \text{if } x=x_I\\ \emptyset & \text{otherwise}\end{cases}$ for each $x\in X$
2: $n=0$
3: $T=(\{v_n\},\emptyset)$
4: $\mathbf{x}(v_n)=x_I$; $\mathbf{v}_{x_I}(\langle\rangle)=v_n$; $Q.insert(v_n)$
5: **while** Q not empty **do**
6: $\quad v=Q.pop()$; $x=\mathbf{x}(v)$
7: $\quad$ **for** each $u\in U(x)$ **do**
8: $\quad\quad x'=f(x,u)$; $\mathbf{u}=path(v)\oplus\langle u\rangle$
9: $\quad\quad\mathcal{U}_{sub}=\mathcal{U}_{x'}.insert(\mathbf{u})$
10: $\quad\quad$ Remove $\mathbf{v}_{x'}(\mathbf{u}')$ from Q for each $\mathbf{u}'\in\mathcal{U}_{sub}$
11: $\quad\quad$ **if** $\mathbf{u}\in\mathcal{U}_{x'}$ **then**
12: $\quad\quad\quad n=n+1$
13: $\quad\quad\quad$ Add vertex v_n and edge $v\xrightarrow{u}v_n$ to T
14: $\quad\quad\quad\mathbf{x}(v_n)=x'$; $\mathbf{v}_{x'}(\mathbf{u})=v_n$
15: $\quad\quad\quad Q.insert(v_n)$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: **end while**
19: **return** $\bigcup_{x\in X_G}\mathcal{U}_x$

Algorithm 3: $\mathcal{U}_{x'}.insert(\mathbf{u})$

Require: The set $\mathcal{U}_{x'}$ of optimal plans identified so far and a candidate plan $\mathbf{u}$ that ends in x'

Ensure: $\mathcal{U}_{x'}$ that includes $\mathbf{u}$ if and only if $\mathbf{u}$ is optimal and the set $\mathcal{U}_{sub}$ of plans in $\mathcal{U}_{x'}$ that are no longer optimal due to the discovery of $\mathbf{u}$

1: $\mathcal{U}_{sub}=\emptyset$
2: **for** each $\mathbf{u}'\in\mathcal{U}_{x'}$ **do**
3: $\quad$ **if** $\mathbf{u}<_{\mathcal{R}}\mathbf{u}'$ **then**
4: $\quad\quad$ Add $\mathbf{u}'$ to $\mathcal{U}_{sub}$
5: $\quad\quad$ Remove $\mathbf{u}'$ from $\mathcal{U}_{x'}$
6: $\quad$ **else if** $\mathbf{u}'<_{\mathcal{R}}\mathbf{u}$ **then**
7: $\quad\quad$ **return** $\emptyset$
8: $\quad$ **end if**
9: **end for**
10: Add $\mathbf{u}$ to $\mathcal{U}_{x'}$
11: **return** $\mathcal{U}_{sub}$

Finally, if the candidate strategy $\mathbf{u}$ is indeed optimal (as indicated by the condition $\mathbf{u}\in\mathcal{U}_{x'}$), a new vertex v_n corresponding to $\mathbf{u}$ is added to T with v as its parent. The functions $\mathbf{x}$ and $\mathbf{v}_{x'}$ are updated accordingly, and v_n is inserted into Q.

Sorting Q in Algorithm 2 and $\mathcal{U}_{x'}.insert(\mathbf{u})$ shown in Algorithm 3 require comparing strategies $\mathbf{u},\mathbf{u}'\in\mathcal{U}$ using $\lesssim_{\mathcal{R}}$. Algorithm 4 performs this comparison. Since a rulebook $\mathcal{R}$ is a partial order on $R/\sim$, we can represent $\mathcal{R}$ by a Hasse diagram, i.e., a graph $G_{\mathcal{R}}=(V_{\mathcal{R}},E_{\mathcal{R}})$ as discussed in Section II-B. From the transitivity property of a preorder, it follows that $G_{\mathcal{R}}$ is a directed acyclic graph. Thus, there exists a topological sorting [86], which

Algorithm 4: Determine whether $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}'$ for any plans $\mathbf{u}, \mathbf{u}'$

Require: Plans $\mathbf{u}, \mathbf{u}' \in \mathcal{U}$ and a topological sort $\langle v_1^{\mathcal{R}} v_2^{\mathcal{R}} ... v_M^{\mathcal{R}} \rangle$ of $V_{\mathcal{R}}$
Ensure: Whether $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}'$
1: $Q^{\mathcal{R}} = \langle v_1^{\mathcal{R}}, v_2^{\mathcal{R}}, ..., v_M^{\mathcal{R}} \rangle$
2: **while** $Q^{\mathcal{R}}$ not empty **do**
3: $v = Q^{\mathcal{R}}.pop()$
4: **if** $\exists r \in [v]$ such that $r(\mathbf{u}) > r(\mathbf{u}')$ **then**
5: **return** $False$
6: **else if** $\exists r \in [v]$ such that $r(\mathbf{u}) < r(\mathbf{u}')$ **then**
7: Remove all successors of v from $Q^{\mathcal{R}}$
8: **end if**
9: **end while**
10: **return** $True$

is a linear ordering of vertices in $V_{\mathcal{R}}$ such that for every directed edge $v_i^{\mathcal{R}} \to v_j^{\mathcal{R}}$, vertex $v_i^{\mathcal{R}}$ comes before $v_j^{\mathcal{R}}$ in the ordering. This topological sorting can be computed using depth-first search with a time complexity of $O(|V_{\mathcal{R}}| + |E_{\mathcal{R}}|)$ and it can be performed offline. Let $\langle v_1^{\mathcal{R}}, v_2^{\mathcal{R}}, ..., v_M^{\mathcal{R}} \rangle$ denote a topological sort of $V_{\mathcal{R}}$, where $M \leq N$ is the number of equivalent classes in R.

Algorithm 4 evaluates two strategies, $\mathbf{u}$ and $\mathbf{u}'$, with respect to the rules in the order $v_1^{\mathcal{R}}, v_2^{\mathcal{R}}, ..., v_M^{\mathcal{R}}$ to determine if $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}'$. To check whether $\mathbf{u} <_{\mathcal{R}} \mathbf{u}'$, we can first run Algorithm 4 to see if $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}'$. If not, we can conclude that $\mathbf{u} \not<_{\mathcal{R}} \mathbf{u}'$. If $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}'$, we then check whether $\mathbf{u}' \lesssim_{\mathcal{R}} \mathbf{u}$ using the same algorithm. If $\mathbf{u}' \lesssim_{\mathcal{R}} \mathbf{u}$, then $\mathbf{u} \not<_{\mathcal{R}} \mathbf{u}'$. Otherwise, $\mathbf{u} <_{\mathcal{R}} \mathbf{u}'$.

B. Analysis

Complexity: For each $x \in X$, let $\mathcal{U}_x^*$ denote the set of optimal strategies from x_I to x. In other words, $\mathcal{U}_x^*$ is $\mathcal{U}_x$ once the while loop in Algorithm 2 terminates. Let $\omega = \max_{x \in X} |\mathcal{U}_x^*|$ represent the size of the largest set of such optimal strategies among all the states $x \in X$.

As in Dijkstra's algorithm, once a vertex v is removed from Q, we know that $path(v)$ is necessarily optimal, i.e., $path(v) \in \mathcal{U}_{\mathbf{x}(v)}^*$. As a result, the while loop runs at most $\omega |X|$ times, with each iteration adding up to $|U|$ states to Q. Thus, the size of Q is upper-bounded by $\omega |X||U|$. Note that the while loop may run fewer times than the largest size Q may attain, because line 13 of Algorithm 2 may remove some elements from Q. On the other hand, $|\mathcal{U}_{x'}|$ is upper-bounded by $\omega |U|$ because x' has at most $|U|$ immediate predecessors. For any element in $\mathcal{U}_{x'}$, the sub-strategy to the immediate predecessor of x', call it x, is optimal, and hence, an element of $\mathcal{U}_x^*$, which has size at most ω.

Assuming that Q is a Fibonacci Heap, removing and adding an element from Q requires $O(\log |Q|)$ and $O(1)$ comparisons, respectively. Using Algorithm 4, comparing any two elements take $O(N)$ time. As a result, the total complexity of adding and removing an element to Q is $O(N)$ and $O(N \log |Q|)$, respectively. With this, line 7 takes time $O(N \log |Q|)$. With appropriate data structures (e.g., using a dictionary to store $\mathbf{x}$), lines 8, 10, 14, 15, 16 and 17 take $O(1)$ time, and as we will see later, they will be dominated by other operations in the complexity analysis, so we can neglect them.

For each iteration of the while loop, lines 10-18 run at most $|U|$ times. Suppose the cost of $path(v)$ is stored for all $v \in V_T$ and the cost of $\mathbf{u}$ is computed and stored when $\mathbf{u}$ is constructed on line 11. Then, line 11 takes $O(N)$ time. $\mathcal{U}_{x'}.insert(\mathbf{u})$ on line 12 takes $O(N|\mathcal{U}_{x'}|)$, which is $O(\omega N |U|)$ time as $|\mathcal{U}_{x'}| \leq \omega |U|$. Line 13 requires removing at most $|\mathcal{U}_{x'}|$ out of Q, each removal taking $O(N \log |Q|)$ time. Thus, line 13 takes $O(\omega N |U| \log |Q|)$ time. Finally, line 18 takes $O(N)$ time.

Combining these together, the computational complexity of Algorithm 2 is $O\big(\omega^2 N |X| |U|^2 \log(\omega |X| |U|)\big)$. Note that the complexity depends on the output size ω, which is common in problems that require computing the complete set of all optimal solutions [51], [56].

Correctness: The correctness of Algorithm 2 is based on the principle of optimality [89]. The following lemma extends this principle to our setup, which involves multiple objectives and a preorder relation among them. For notational simplicity, we focus on initial segments of strategies as this is all that is needed for the proof of Proposition 7, though the result can be extended to arbitrary segments.

Lemma 1. Let $\mathbf{u}^* = \langle u_0^*, u_1^*, ..., u_T^* \rangle \in \mathcal{U}_G^*$ be an optimal strategy and $\langle x_0^*, x_1^*, ..., x_{T+1}^* \rangle$ be the sequence of states obtained from applying $\mathbf{u}^*$, starting from $x_0^* = x_I$. Consider an arbitrary t such that $0 \leq t \leq T+1$. Let $\tilde{\mathbf{u}}^* = \langle u_0^*, u_1^*, ..., u_{t-1}^* \rangle$ denote the sub-strategy of $\mathbf{u}^*$ to reach x_t^* and $\mathcal{U}_t = \{ \langle u_0, u_1, ..., u_{T'} \rangle \in \mathcal{U} \mid T' \in \mathbb{N}, x_{T'+1} = x_t^* \}$ denote the set of all valid strategies for x_I that end at x_t^*. Then, there does not exist $\tilde{\mathbf{u}} \in \mathcal{U}_t$ such that $\tilde{\mathbf{u}} <_{\mathcal{R}} \tilde{\mathbf{u}}^*$.

Proof. Suppose there exists $\tilde{\mathbf{u}} = \langle \tilde{u}_0, ..., \tilde{u}_{T'} \rangle \in \mathcal{U}_t$ with $\tilde{\mathbf{u}} <_{\mathcal{R}} \tilde{\mathbf{u}}^*$. Let $\mathbf{u} = \langle \tilde{u}_0, ..., \tilde{u}_{T'}, u_t^*, u_{t+1}^*, ..., u_T^* \rangle$ be the strategy formed by replacing $\tilde{\mathbf{u}}^*$ in $\mathbf{u}^*$ with $\tilde{\mathbf{u}}$.

Since $\tilde{\mathbf{u}} \lesssim_{\mathcal{R}} \tilde{\mathbf{u}}^*$, by definition, for any rule r such that $r(\tilde{\mathbf{u}}) > r(\tilde{\mathbf{u}}^*)$, there exists $r' > r$ such that $r'(\tilde{\mathbf{u}}) < r'(\tilde{\mathbf{u}}^*)$. Since each rule is additive, it must be the case that for any r such that $r(\mathbf{u}) > r(\mathbf{u}^*)$, there exists $r' > r$ such that $r'(\mathbf{u}) < r'(\mathbf{u}^*)$. As a result, we can conclude that $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}^*$.

Furthermore, since $\tilde{\mathbf{u}}^* \not\lesssim_{\mathcal{R}} \tilde{\mathbf{u}}$, by definition, there exists a rule r such that $r(\tilde{\mathbf{u}}) < r(\tilde{\mathbf{u}}^*)$ and for all $r' > r$, $r'(\tilde{\mathbf{u}}) \leq r'(\tilde{\mathbf{u}}^*)$. Since each rule is additive, it must be the case that $r(\mathbf{u}) < r(\mathbf{u}^*)$ and for all $r' > r$, $r'(\mathbf{u}) \leq r'(\mathbf{u}^*)$. As a result $\mathbf{u}^* \not\lesssim_{\mathcal{R}} \mathbf{u}$. Since $\mathbf{u} \lesssim_{\mathcal{R}} \mathbf{u}^*$ but $\mathbf{u}^* \not\lesssim_{\mathcal{R}} \mathbf{u}$, we can conclude that $\mathbf{u} <_{\mathcal{R}} \mathbf{u}^*$, which contradicts the assumption that $\mathbf{u}^* \in \mathcal{U}_G^*$. □

Since the principle of optimality holds, we can prove the correctness of Algorithm 2. The proof is similar to the proof of correctness of the label-setting algorithm for solving MOSP [51], [52] and is omitted for brevity.

Proposition 7. Suppose that a rulebook $\mathcal{R}$ satisfies Assumptions 1, 3, and 4. Then, Algorithm 2 correctly returns the maximum complete set $\mathcal{U}_G^*$ of optimal control strategies as defined in Problem 2.

C. Case Studies

Obstacle avoidance with different rulebooks: Consider a scenario in which an autonomous vehicle encounters an obstacle, as illustrated in Example 11 of [59] and shown in Fig. 10. Let x_I denote the state at the lower left corner and X_G include the state at the lower right corner. The rulebook consists of four rules: r_1 (no collision), r_2 (lane keeping), r_3 (obstacle clearance), and r_4 (path length). Let $R = \{r_1, r_2, r_3, r_4\}$. Each rule r_i is defined based on

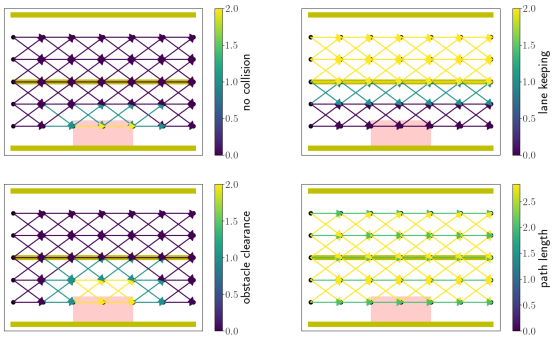


Fig. 10: An autonomous vehicle traveling from left to right encounters an obstacle (red region). Points are states and arrows are transitions. [Top, left] Transition cost for r_1 (no collision); [Top, right] r_2 (lane keeping); [Bottom, left] r_3 (obstacle clearance); and [Bottom, right] r_4 (path length).

the transition cost $r_i(\langle x,x'\rangle,u)$ from a state $x=(x_x,x_y)\in X$ to the next state $x'=(x'_x,x'_y)=f(x,u)$, under an action $u\in U(x)$. The cost functions are defined as follows: $r_1(\langle x,x'\rangle,u)=\mathbf{1}(x\in Obs)+\mathbf{1}(x'\in Obs)$; $r_2(\langle x,x'\rangle,u)=\mathbf{1}(x_2>L)+\mathbf{1}(x'_2>L)$; $r_3(\langle x,x'\rangle,u)=\mathbf{1}(d(x,Obs)<C)+\mathbf{1}(d(x',Obs)<C)$; and $r_4(\langle x,x'\rangle,u)=\sqrt{(x'_2-x_2)^2+(x'_1-x_1)^2}$. Here, $\mathbf{1}(\cdot)$ is the indicator function, which returns 1 if the condition holds and 0 otherwise, $Obs\subset\mathbb{R}^2$ is the obstacle region, $d(x,Obs)$ is the Euclidean distance from state x to Obs, and $C\in\mathbb{R}$ is the required clearance. The constant $L\in\mathbb{R}$ represents the threshold for the y-coordinate beyond which the vehicle is no longer considered to be fully within the lane, Fig. 10 illustrates these transition costs.

Consider the base rulebook $\mathcal{R}_1=\langle R,\lesssim_1\rangle$, where the relationships between the rules are defined as follows: $r_1>_1 r_2$, $r_1>_1 r_3$, $r_2>_1 r_4$, and $r_3>_1 r_4$. This means the no collision rule (r_1) has the highest priority. The lane keeping rule (r_2) and the obstacle clearance rule (r_3) have the second highest priority but are not comparable to each other. Finally, the path length rule (r_4) has the lowest priority.

By applying Algorithm 2 with rulebook $\mathcal{R}_1$, we obtain a set $\mathcal{U}^*_G$ of 13 optimal strategies, as shown in Fig. 11. The average time to compute this set is 3.4 milliseconds. For each optimal strategy $\mathbf{u}^*\in\mathcal{U}^*_G$, the resulting states of the vehicle do not collide with the obstacle, i.e., $r_1(\mathbf{u}^*)=0$. This is because r_1 (no collision) is the highest-priority rule, and as a result, its violation must be avoided at all costs, even if this leads to infinite violations of other rules. Additionally, the strategies in $\mathcal{U}^*_G$ represent different trade-offs between the violations of r_2 (lane keeping) and r_3 (obstacle clearance), since these two rules are incomparable. Finally, no optimal strategy in $\mathcal{U}^*_G$ causes the vehicle to unnecessarily enter the opposite lane since doing so will incur more path length without reducing the violation of the other rules.

Impact of priority refinement: We now explore two different refinements of the base rulebook $\mathcal{R}_1$ by adjusting the priority between r_2 and r_3. First, consider a rulebook, $\mathcal{R}_2=\langle R,\lesssim_2\rangle$, which is a refinement of $\mathcal{R}_1$ in that $\lesssim_2$ includes all the relationships from $\lesssim_1$ and also adds the priority relation $r_2>_2 r_3$. In other words, the lane keeping rule (r_2) is given higher priority than the obstacle clearance rule (r_3) in $\mathcal{R}_2$. Applying Algorithm 2 to $\mathcal{R}_2$ yields a set of 4 optimal strategies, as shown in Fig. 11. The average time to compute this set is 0.9 milliseconds. Note that

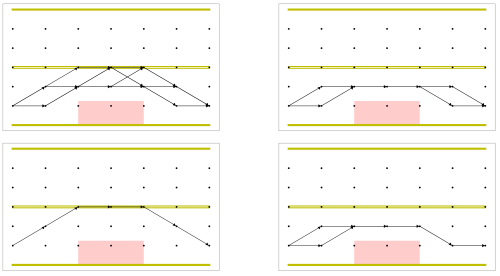


Fig. 11: The set $\mathcal{U}^*_G$ of optimal strategies for [Top, left] $\mathcal{R}_1$, [Top, right] $\mathcal{R}_2$, [Bottom, left] $\mathcal{R}_3$, [Bottom, right] $\mathcal{R}_4$. Multiple optimal strategies may share edges. These strategies either have identical rule-violation costs (e.g., in [top, right] and [bottom, right]) or represent distinct trade-offs among rules (e.g., in [Top, left]) under the rulebook.

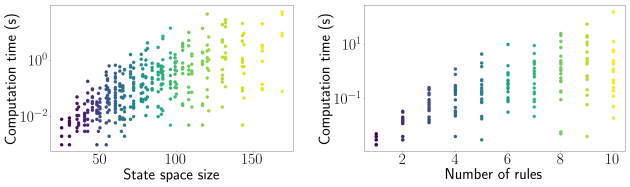


Fig. 12: Exponential growth of computation time under two settings. [Left] Varying state space size with 5 rules and random priorities. [Right] Varying number of rules with random priorities on a 100-state grid.

the set of optimal strategies for $\mathcal{R}_2$ is a subset of those for $\mathcal{R}_1$, as expected since $\mathcal{R}_2$ is a refinement of $\mathcal{R}_1$, and includes only strategies that do not lead to a violation of the lane keeping rule.

Next, consider the opposite refinement $\mathcal{R}_3=\langle R,\lesssim_3\rangle$ of $\mathcal{R}_1$, where $\lesssim_3$ includes all the relationships from $\lesssim_1$ and also adds $r_3>_3 r_2$. This means that in $\mathcal{R}_3$, obstacle clearance (r_3) has higher priority than lane keeping (r_2). In this case, Algorithm 2 returns a set of 1 optimal strategy, as shown in Fig. 11. The average time to compute this set is 0.6 milliseconds. The set of optimal strategies is a subset of those for $\mathcal{R}_1$ and includes only strategies that do not lead to a violation of the obstacle clearance rule.

Comparison with the weighted sum approach: The weighted sum approach can be viewed as a special case of our formulation by defining a rulebook $\mathcal{R}_4$ with a single rule r, given by the weighted sum of r_1,r_2,r_3,r_4, i.e., $r=\sum_{i=1}^4 c_i r_i$, where $c_i\in\mathbb{R}$ is the weight for rule r_i. Applying Algorithm 2 to $\mathcal{R}_4$ yields a set of 2 optimal strategies when $c_i=1,\forall i$, as shown in Fig. 11. The average time to compute this set is 1.7 milliseconds. However, the complete set of optimal strategies should include 4 solutions, similar to those obtained for $\mathcal{R}_2$. The missing optimal strategies result from numerical precision issues, where costs that are theoretically identical may be treated as different due to computational limitations. These issues are more pronounced in the weighted sum approach, as it involves summing weighted values, which can introduce small numerical discrepancies. As a result, strategies with theoretically identical costs may be incorrectly treated as different. In contrast, $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$ evaluate violations with respect to individual rules, making them less susceptible to such precision errors.

Computation time analysis for varying state spaces and rulebooks: First, we analyze grid-based problems by varying

the number of states. For this analysis, we use a rulebook $\mathcal{R}$ containing 5 rules with randomly assigned priority relations $\lesssim$. As shown in Fig. 12, the computation time grows exponentially with the size of the state space. This exponential growth is due to the exponential increase in the total number of optimal strategies (denoted with ω in Section V-B) as the size of the state space grows.

Building on this, we next investigate how the computation time changes as we vary the number of rules in a fixed state space. Specifically, we use a grid-based state space with 100 states and consider rulebooks with a varying number of rules, where the priority relations among the rules are also assigned randomly. Fig. 12 illustrates that the computation time also increases exponentially as the number of rules grows.

Practical considerations: While the example assumes a discrete state space with a coarse discretization, this abstraction is often sufficient for verification and regulatory purposes. In these settings, the interest typically lies in assessing whether a given robot's strategy $\mathbf{u}$ is *approximately* optimal rather than exactly optimal. There are two primary ways to assess this. The first is to use Algorithm 4 to check that for each $\mathbf{u}^* \in \mathcal{U}_G^*$, $\mathbf{u}^* \not\leq_{\mathcal{R}} \mathbf{u}$. The second is to verify that that the resulting trajectory $\mathbf{x}_{\mathbf{u}}$ of the robot remains within an acceptable distance from an optimal trajectory $\mathbf{x}_{\mathbf{u}^*}$ for some $\mathbf{u}^* \in \mathcal{U}_G^*$. For either approach, a coarse discretization of the state space is sufficient to draw meaningful conclusions. More importantly, it is crucial that $\mathcal{U}_G^*$ is complete to avoid incorrectly labeling a $\mathbf{u}$ as suboptimal simply because some optimal strategies were missed, which is a common limitation of weighted-sum approaches.

VII. Conclusion and Future Work

We employed the rulebook formalism as a specification language for planning and control of autonomous robots. Based on this formalism, we formulated both single-strategy and complete control synthesis problems, showing that they generalize existing temporal logic-based and optimization-based planning and control. For single-strategy synthesis, we identified structural assumptions on rules that enable polynomial-time synthesis. This allows our algorithm to handle combinations of objectives that have not been considered in prior work, as illustrated in a case study with temporal logic constraints, min-max safety requirements, and standard efficiency criteria, even when these objectives differ in structure and interpretation. For complete synthesis, we proposed an algorithm to compute all optimal strategies, which provide insights into the different trade-offs among competing objectives. Simulation results demonstrated the practical utility of our approach compared to existing approaches. Overall, the rulebook formalism provides a principled and unifying foundation for specification, control, and verification in autonomous robots.

Future work will focus on extending the framework to adversarial or probabilistic systems to support planning and control under uncertainty and in dynamic or unpredictable environments. For single-strategy synthesis, we aim to identify additional tractable subclasses of rulebooks, especially for continuous-state systems. This includes discovering rule structures or system dynamics that allow efficient synthesis while preserving the expressiveness of the specification. We also plan to investigate receding-horizon planning under rulebooks, where a control strategy is computed over a finite horizon and updated online, enabling real-time operation in dynamic environments while respecting prioritized rules.

For complete synthesis, we will explore approximate and scalable algorithms to handle more complex systems. In particular, we will extend the concept of ϵ-approximate domination from [58], which is currently defined only for incomparable objectives, to rulebooks with hierarchical structure. This extension requires defining ϵ-approximate domination in a way that respects the underlying preorder. We also plan to investigate heuristic strategies to guide the search efficiently.

References

[1] I. Hasuo, "Responsibility-sensitive safety: an introduction with an eye to logical foundations and formalization," 2022.

[2] J. Tůmová, L. I. Reyes Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violation LTL planning with conflicting specifications," in *2013 American Control Conference*, pp. 200–205, 2013.

[3] S. Bharadwaj, T. Wongpiromsarn, N. Neogi, J. Muffoletto, and U. Topcu, "Minimum-violation traffic management for urban air mobility," in *NASA Formal Methods: 13th International Symposium, NFM 2021, Virtual Event, May 24–28, 2021, Proceedings*, (Berlin, Heidelberg), p. 37–52, Springer-Verlag, 2021.

[4] "Code of federal regulations," 2020.

[5] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[6] N. Aréchiga, "Specifying safety of autonomous vehicles in signal temporal logic," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 58–63, 2019.

[7] G. Fainekos, H. Kress-Gazit, and G. Pappas, "Temporal logic motion planning for mobile robots," in *IEEE International Conference on Robotics and Automation*, pp. 2020–2025, 2005.

[8] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," *IEEE Transactions on Automatic Control*, vol. 53, no. 1, pp. 287–297, 2008.

[9] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Sampling-based motion planning with temporal goals," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2689–2696, 2010.

[10] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4817–4822, 2013.

[11] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints*," *Int. J. Rob. Res.*, vol. 30, p. 1695–1708, Dec. 2011.

[12] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *Proc. of IEEE Conference on Decision and Control*, pp. 2222–2229, 2009.

[13] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. Pappas, "Symbolic planning and control of robot motion," *Robotics Automation Magazine, IEEE*, vol. 14, pp. 61–70, 2007.

[14] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *Proc. of IEEE International Conference on Robotics and Automation*, pp. 3116–3121, 2007.

[15] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, reactive, high-level robot control," *IEEE Robotics Automation Magazine*, vol. 18, no. 3, pp. 65–74, 2011.

[16] V. Raman, C. Finucane, and H. Kress-Gazit, "Temporal logic robot mission planning for slow and fast actions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 251–256, Oct 2012.

[17] M. Lahijanian, J. Wasniewski, S. B. Andersson, and C. Belta, "Motion planning and control from temporal logic specifications with probabilistic satisfaction guarantees," in *IEEE International Conference on Robotics and Automation*, (Anchorage, AK), pp. 3227–3232, 2010.

[18] X. C. Ding, S. L. Smith, C. Belta, and D. Rus, "LTL control in uncertain environments with probabilistic satisfaction guarantees," in *IFAC World Congress*, 2011.

[19] A. I. Medina Ayala, S. B. Andersson, and C. Belta, "Temporal logic control in dynamic environments with probabilistic satisfaction guarantees," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007*, pp. 3108–3113, 2011.

[20] R. Sharan and J. Burdick, "Finite state control of POMDPs with LTL specifications," in *2014 American Control Conference*, pp. 501–508, June 2014.

[21] S. Haesaert, P. Nilsson, C. Vasile, R. Thakker, A. Agha-mohammadi, A. Ames, and R. Murray, "Temporal logic control of pomdps via label-based stochastic simulation relations," *IFAC-PapersOnLine*, vol. 51, no. 16, pp. 271–276, 2018. 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018.

[22] A. M. Wells, M. Lahijanian, L. E. Kavraki, and M. Y. Vardi, "LTLf synthesis on probabilistic systems," *Electronic Proceedings in Theoretical Computer Science*, vol. 326, pp. 166–181, Sept. 2020.

[23] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[24] T. Anevlavis, M. Philippe, D. Neider, and P. Tabuada, "Being correct is not enough: efficient verification using robust linear temporal logic," *arXiv preprint arXiv:2102.11991*, 2021.

[25] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems* (Y. Lakhnech and S. Yovine, eds.), (Berlin, Heidelberg), pp. 152–166, Springer Berlin Heidelberg, 2004.

[26] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Formal Modeling and Analysis of Timed Systems* (K. Chatterjee and T. A. Henzinger, eds.), (Berlin, Heidelberg), pp. 92–106, Springer Berlin Heidelberg, 2010.

[27] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *53rd IEEE Conference on Decision and Control*, pp. 81–87, 2014.

[28] L. Lindemann and D. V. Dimarogonas, "Control barrier functions for signal temporal logic tasks," *IEEE Control Systems Letters*, vol. 3, no. 1, pp. 96–101, 2019.

[29] A. Zehfroosh and H. G. Tanner, "Non-smooth control barrier navigation functions for stl motion planning," *Frontiers in Robotics and AI*, vol. Volume 9 - 2022, 2022.

[30] C. Belta and S. Sadraddini, "Formal methods for control synthesis: An optimization perspective," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 2, no. 1, pp. 115–140, 2019.

[31] P. Halder, H. Homburger, L. Kiltz, J. Reuter, and M. Althoff, "Trajectory planning with signal temporal logic costs using deterministic path integral optimization," 2025.

[32] C.-I. Vasile, V. Raman, and S. Karaman, "Sampling-based synthesis of maximally-satisfying controllers for temporal logic specifications," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3840–3847, 2017.

[33] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, "Reactive synthesis from signal temporal logic specifications," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, HSCC '15, (New York, NY, USA), p. 239–248, Association for Computing Machinery, 2015.

[34] T. Akazaki and I. Hasuo, "Time robustness in mtl and expressivity in hybrid system falsification," in *Computer Aided Verification* (D. Kroening and C. S. Păsăreanu, eds.), (Cham), pp. 356–374, Springer International Publishing, 2015.

[35] L. Lindemann and D. V. Dimarogonas, "Robust motion planning employing signal temporal logic," in *2017 American Control Conference (ACC)*, pp. 2950–2955, 2017.

[36] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta, "Control from signal temporal logic specifications with smooth cumulative quantitative semantics," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4361–4366, 2019.

[37] N. Mehdipour, C.-I. Vasile, and C. Belta, "Arithmetic-geometric mean robustness for control from signal temporal logic specifications," in *2019 American Control Conference (ACC)*, pp. 1690–1695, 2019.

[38] S. M. LaValle, *Planning Algorithms*. USA: Cambridge University Press, 2006.

[39] F. Seccamonte, J. Kabzan, and E. Frazzoli, "On maximizing lateral clearance of an autonomous vehicle in urban environments," in *IEEE Intelligent Transportation Systems Conference*, pp. 1819–1825, 2019.

[40] Z. Zhang, L. Zheng, Y. Li, Y. Yu, and X. Qiao, "Model predictive control for path following of autonomous vehicle considering model parameter uncertainties," in *2021 6th IEEE International Conference on Advanced Robotics and Mechatronics (ICARM)*, pp. 207–212, 2021.

[41] T. Wongpiromsarn, M. Kallmann, and A. Kolling, "Locally homotopic paths: Ensuring consistent paths in hierarchical path planning," *IEEE Robotics and Automation Letters*, 2024. to appear.

[42] D. P. Bertsekas, *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996.

[43] T. M. Howard, R. A. Knepper, and A. Kelly, *Constrained Optimization Path Following of Wheeled Robots in Natural Terrain*, pp. 343–352. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[44] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. USA: Cambridge University Press, 1st ed., 2017.

[45] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2017.

[46] L. Wang, A. D. Ames, and M. Egerstedt, "Safety barrier certificates for collisions-free multirobot systems," *IEEE Transactions on Robotics*, vol. 33, no. 3, pp. 661–674, 2017.

[47] W. Xiao, C. G. Cassandras, and C. Belta, *Safe Autonomy with Control Barrier Functions: Theory and Applications*. Synthesis Lectures on Computer Science, Springer International Publishing, 2024.

[48] A. Bemporad and D. Muñoz de la Peña, "Multiobjective model predictive control," *Automatica*, vol. 45, no. 12, pp. 2823–2830, 2009.

[49] D. He, H. Li, and H. Du, "Lexicographic multi-objective mpc for constrained nonlinear systems with changing objective prioritization," *Automatica*, vol. 125, p. 109433, 2021.

[50] A. Chinchuluun and P. M. Pardalos, "A survey of recent developments in multiobjective optimization," *Annals of Operations Research*, vol. 154, pp. 29–50, 2007.

[51] R. G. Garroppo, S. Giordano, and L. Tavanti, "A survey on multi-constrained optimal path computation: Exact and approximate algorithms," *Computer Networks*, vol. 54, no. 17, pp. 3081–3107, 2010.

[52] E. Martins, "On a multicriteria shortest path problem," *European Journal of Operational Research*, vol. 16, pp. 236–245, 1984.

[53] E. Martins and J. Santos, "The labeling algorithm for the multiobjective shortest path problem," tech. rep., Pré-publicações do Departamento de Matemática, Departamento de Matemática da Universidade de Coimbra, November 1999.

[54] J. L. P. De la Cruz, J. L. P. De la Cruz, L. Mandow, and L. Mandow, "A new approach to multiobjective A* search," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, p. 218–223, Morgan Kaufmann Publishers Inc., 2005.

[55] X. Gandibleux, F. Beugnies, and S. Randriamasy, "Martins' algorithm revisited for multi-objective shortest path problems with a maxmin cost function," *4OR*, vol. 4, pp. 47–59, 2006.

[56] P. Maristany de las Casas, A. Sedeño-Noda, and R. Borndörfer, "An improved multiobjective shortest path algorithm," *Computers & Operations Research*, vol. 135, p. 105424, 2021.

[57] O. Salzman, A. Felner, C. Hernandez, H. Zhang, S.-H. Chan, and S. Koenig, "Heuristic-search approaches for the multi-objective shortest-path problem: progress and research opportunities," in *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI '23, 2023.

[58] H. Zhang, O. Salzman, T. K. S. Kumar, A. Felner, C. Hernández Ulloa, and S. Koenig, "A*pex: Efficient approximate multi-objective search on graphs," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, pp. 394–403, Jun. 2022.

[59] A. Censi, K. Slutsky, T. Wongpiromsarn, D. Yershov, S. Pendleton, J. Fu, and E. Frazzoli, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8536–8542, 2019.

[60] M. Penlington, A. Zanardi, and E. Frazzoli, "Optimization of rulebooks via asymptotically representing lexicographic hierarchies for autonomous vehicles," 2024.

[61] W. Xiao, N. Mehdipour, A. Collin, A. Y. Bin-Nun, E. Frazzoli, R. D. Tebbens, and C. Belta, "Rule-based optimal control for autonomous driving," in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, ICCPS '21, (New York, NY, USA), p. 143–154, Association for Computing Machinery, 2021.

[62] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, (New York, NY, USA), p. 179–190, Association for Computing Machinery, 1989.

[63] N. Piterman, A. Pnueli, and Y. Sa'ar, *Synthesis of Reactive(1) Designs*, vol. 3855 of *Lecture Notes in Computer Science*, ch. 24, pp. 364–380. Springer Berlin Heidelberg, 2005.

[64] E. Schechter, "Chapter 3 - relations and orderings," in *Handbook of Analysis and Its Foundations* (E. Schechter, ed.), pp. 49–77, San Diego: Academic Press, 1997.

[65] P. Eklund and W. Gähler, "Generalized cauchy spaces," *Mathematische Nachrichten*, vol. 147, no. 1, pp. 219–233, 1990.

[66] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[67] L. I. R. Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *52nd IEEE Conference on Decision and Control*, pp. 3217–3224, Dec 2013.

[68] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, and U. Topcu, "Minimum-violation planning for autonomous systems: Theoretical and practical considerations," in *2021 American Control Conference (ACC)*, pp. 4866–4872, 2021.

[69] A. I. G. Guerra, T. S. Shiuan, P. Hibbard, Y. J. Yew, and Y. T. Beng, "Behaviour description database for avs in singapore," 2023.

[70] J. Karlsson, S. van Waveren, C. Pek, I. Torre, I. Leite, and J. Tumova, "Encoding human driving styles in motion planning for autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1050–1056, 2021.

[71] F. Toledo, T. Woodlief, S. Elbaum, and M. B. Dwyer, "Specifying and monitoring safe driving properties with scene graphs," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15577–15584, 2024.

[72] T. Wongpiromsarn, S. Karaman, and E. Frazzoli, "Synthesis of provably correct controllers for autonomous vehicles in urban environments," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1168–1173, 2011.

[73] B. Helou, A. Dusi, A. Collin, N. Mehdipour, Z. Chen, C. Lizarazo, C. Belta, T. Wongpiromsarn, R. D. Tebbens, and O. Beijbom, "The reasonable crowd: Towards evidence-based and interpretable models of driving behavior," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6708–6715, 2021.

[74] J. Tumova, L. I. R. Castro, S. Karaman, E. Frazzoli, and D. Rus, "Minimum-violation LTL planning with conflicting specifications," in *2013 American Control Conference*, pp. 200–205, June 2013.

[75] J. Tumova, A. Marzinotto, D. V. Dimarogonas, and D. Kragic, "Maximally satisfying LTL action planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1503–1510, 2014.

[76] J. Tumova, S. Karaman, C. Belta, and D. Rus, "Least-violating planning in road networks from temporal logic specifications," in *2016 ACM/IEEE 7th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 1–9, 2016.

[77] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the minimal revision problem of specification automata," *Int. J. Rob. Res.*, vol. 34, p. 1515–1535, Oct. 2015.

[78] J. Tumova, G. C. Hall, S. Karaman, E. Frazzoli, and D. Rus, "Least-violating control strategy synthesis with safety rules," in *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, pp. 1–10, 2013.

[79] C.-I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scLTL motion planning for mobility-on-demand," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1481–1488, 2017.

[80] M. Lahijanian, S. Almagor, D. Fried, L. E. Kavraki, and M. Y. Vardi, "This time the robot settles for a cost: A quantitative approach to temporal logic planning with partial satisfaction," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, p. 3664–3671, AAAI Press, 2015.

[81] F. Penedo, C.-I. Vasile, and C. Belta, "Language-guided sampling-based planning using temporal relaxation," in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics* (K. Goldberg, P. Abbeel, K. Bekris, and L. Miller, eds.), pp. 128–143, Cham: Springer International Publishing, 2020.

[82] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017.

[83] K. Slutsky, D. Yershov, T. Wongpiromsarn, and E. Frazzoli, "Hierarchical multiobjective shortest path problems," in *Algorithmic Foundations of Robotics XIV*, pp. 261–276, Springer International Publishing, 2021.

[84] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

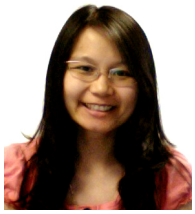
[85] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[86] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 4th ed., 2022.

[87] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

[88] R. A. Knepper and D. Rus, "Pedestrian-inspired sampling-based multi-robot collision avoidance," in *2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication*, pp. 94–100, 2012.

[89] R. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton University Press, 1957.



Tichakorn Wongpiromsarn (Senior Member, IEEE) received the B.S. degree in mechanical engineering from Cornell University and the M.S. and Ph.D. degrees in mechanical engineering from California Institute of Technology. She is currently an Associate Professor with the Department of Computer Science at Iowa State University, Ames, IA, USA. Her research interests lie in the broad area of computer science, control theory, and optimization, with a particular focus on the design and analysis of autonomous systems.

Dr. Wongpiromsarn was the recipient of the NSF CAREER Award.

Konstantin Slutsky received the B.S. degree in mathematics from Kharkiv National University and the M.S. and Ph.D. degrees in mathematics from University of Illinois at Urbana-Champaign. He is currently an Assistant Professor with the Department of Mathematics at Iowa State University, Ames, IA, USA. He studies descriptive set theory, ergodic theory, topological dynamics, and motion planning.

Emilio Frazzoli (Fellow, IEEE) is a professor of Dynamic Systems and Control at ETH Zürich. Until March 2021, he was Chief Scientist of Motional, the latest embodiment of nuTonomy, the startup he founded with Karl Iagnemma in 2013. His main research interest are in robotics, autonomous systems, and intelligent mobility. In acknowledgement of his work in these fields, Emilio has received several awards, including the 2015 IEEE George S. Axelby Award, the 2017 IEEE Kiyo Tomiyasu Award, the 2022 IEEE George N. Saridis Award, and has been named an IEEE Fellow in 2019. In 2022, he received the RSS Test of Time award for his papers on RRT*, with Sertac Karaman. A former full professor at MIT, he directed the research group that first demonstrated an autonomous mobility ("robotaxi") service to the public, and performed the first analysis of the social and economic impact of such a service, based on real transportation data. He holds a Laurea Degree in Aeronautical Engineering from the Sapienza University of Rome, and a PhD in Aeronautics and Astronautics from MIT.