2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Motion Planning for Automata-based Objectives using Efficient Gradient-based Methods

Anand Balakrishnan, Merve Atasever, Jyotirmoy V. Deshmukh
University of Southern California, Los Angeles, California, USA
Email: {anandbal, atasever, jdeshmuk}@usc.edu

***Abstract*— In recent years, there has been increasing interest in using formal methods-based techniques to safely achieve temporal tasks, such as timed sequence of goals, or patrolling objectives. Such tasks are often expressed in real-time logics such as Signal Temporal Logic (STL), whereby, the logical specification is encoded into an optimization problem. Such approaches usually involve optimizing over the quantitative semantics, or robustness degree, of the logic over bounded horizons: the semantics can be encoded as mixed-integer linear constraints or into smooth approximations of the robustness degree. A major limitation of this approach is that it faces scalability challenges with respect to temporal complexity: for example, encoding long-term tasks requires storing the entire history of the system. In this paper, we present a quantitative generalization of such tasks in the form of symbolic automata objectives. Specifically, we show that symbolic automata can be expressed as matrix operators that lend themselves to automatic differentiation, allowing for the use of off-the-shelf gradient-based optimizers. We show how this helps solve the need to store arbitrarily long system trajectories, while efficiently leveraging the task structure encoded in the automaton.**

I. Introduction

For autonomous robots operating in highly uncertain or dynamic environments, motion planning can be challenging [1], [2]. A popular class of motion planning algorithms that address such environments is *model predictive control*, which recomputes short-term plans in real-time to adapt to changes in the environment or stochasticity in the environment dynamics [3], [4]. In this design paradigm, the system designer creates a predictive model of the behavior of the robot and its operating environment that is used at runtime to predict future states based on some finite sequence of control actions. The system uses this *model predictive* approach along with an online optimizer to fine the optimal sequence of control actions that minimize some user-specified cost function on the predicted trajectory. The system then applies the first control action, and then replans the sequence for the newly observed state. This paradigm is also referred to as *receding horizon* planning.

In most modern motion planning approaches, the optimization problem is directly or indirectly reduced to solving quadratic cost functions over system states and actions, with the assumption that the desired system behaviors are the ones that minimize such costs [5], [6]. While this is adequate for task objectives such as tracking a set of way-points, or minimizing the energy consumed by the robot, some objectives require the robot to exhibit complex spatio-temporal behavior [7], [8], [9]. There is growing body of literature to specify such tasks using formalisms such as Linear Temporal Logic (LTL) [10] and Signal Temporal Logic (STL) [11] and the use of these logics for robot motion planning.

Broadly speaking, there have been two high-level directions one can take with motion planning with temporal logic specifications [12]: 1) by translating the specification into an automaton, and decomposing the planning problem over it; or 2) by using the *quantitative semantics* of the logic to directly optimize for satisfaction or robustness of the system w.r.t. the specification.

The techniques presented in [13], [8], [14], [15], [16] represent the former of the above approaches. Here, the frameworks deconstruct complex, temporal motion planning over the transitions in an automaton representing the temporal specification. Specifically, for each location in the automaton, a corresponding set is computed in the state space of the system, and the motion plan is computed for each pair of such connected sets. These approaches suffer from the lack of scalability, as the size of the automata can increase exponentially to that of the temporal logic specification.

On the other hand, several proposed works in recent literature directly optimize over the semantics of the temporal logic. Of particular relevance to this paper is STL which allows defining a *robust satisfaction value* or *robustness* that approximates the distance of a given trajectory from the set of trajectories satisfying the formula [17], [18], [19]. The robustness metric is leveraged to encode the motion planning problem for linear (and piecewise-linear) dynamical systems as a mixed integer linear program [20], [21] or by gradient-based optimization of the smooth approximation of robustness [22], [23]. A key limitation of these approaches are that they are not applicable to general nonlinear models, or suffer from intractability with increasing prediction horizon and formula complexity.

Our Contributions: In this paper, we propose a framework to bridge the gap between automata-based techniques and robustness-based optimization. Specifically, we define a *matrix operator* on symbolic automata that

This work was partially supported by the National Science Foundation through the following grants: CAREER award (SHF-2048094), CNS-1932620, CNS-2039087, FMitF-1837131, CCF-SHF-1932620, funding by Toyota RD and Siemens Corporate Research through the USC Center for Autonomy and AI, an Amazon Faculty Research Award, and the Airbus Institute for Engineering Research.

1) translates symbolic automata to weighted matrices given a system state; and
2) builds on abstract matrix operations to enable automatic differentiation.

This matrix operator can be used along with off-the-shelf gradient-based optimization pipelines to compute end-to-end motion plans for complex temporal specifications. We will demonstrate the efficacy of our proposed framework in various motion planning scenarios by comparing against similar tools. We also explore how various notions of *robustness* can be achieved within the same pipeline, and how they manifest in concrete tasks.

Related Work

In automata-based methods, temporal logic specifications are usually translated to finite state automata (for finite-length behavior) or ω-regular automata (for infinite behavior) [24]. The control problem is then reduced to a graph game on the product of the specification automaton and a finite model of the system, such as a transition system or a Markov Decision Process [25], [15]; or as a hierarchical control problem on a hybrid system [7], [13], [26], [27]. The main limitation of automata-based methods is the prohibitive computational complexity, due to the exponential blow-up of finite model abstractions for infinite systems.

The main thrust of optimization-based methods are temporal logics with semantics defined over finite-length signals, namely Signal Temporal Logic [11] and Metric Temporal Logic [28]. Specifically, many optimization-based approaches exploit the quantitative semantics defined for these logics presented in [17] and [18]. These *robustness* metrics are used in optimization pipelines to assign costs that penalize deviation from specified system behavior. For example, [21] and [29] both present methods to translate STL formulas into mixed-integer constraints for convex optimization, based on prior work for Linear Temporal Logic [30]. Likewise, [23] and [22] present smooth approximations of STL quantitative semantics to enable gradient-based optimization directly over the STL formula. Unlike automata-based approaches, such methods require entire signal histories to compute a robustness score, and the $\min$ and $\max$ operations in the quantitative semantics, can lead to various local optima due to vanishing gradients and non-linearity.

Recent developments in *automatic differentiation* algorithms [31] has enabled a boom in the use of gradient-based optimization methods, via off-the-shelf libraries like PyTorch [32], and JAX [33]. The works in [22] and [23] leverage this in when defining their smooth approximations of robustness. Similarly, the library presented in [34] builds on PyTorch to define the quantitative semantics of STL as *compute graphs*. Such gradient-based methods have allowed for the evaluation of temporal logic quantitative semantics to scale well by leveraging the compute pipeline that such libraries build on, but still suffer from dependence on history.

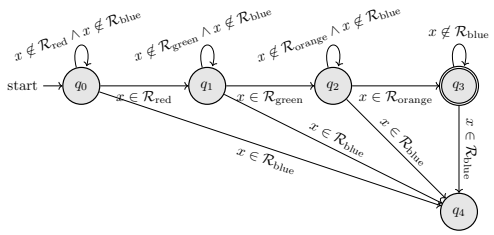

Fig. 1. Example of a symbolic automaton describing the specification "move to region $\mathcal{R}_{\text{red}}$, then region $\mathcal{R}_{\text{green}}$, and then to region $\mathcal{R}_{\text{orange}}$ in order while always avoiding region $\mathcal{R}_{\text{blue}}$. " An *accepting* run in the automaton is a sequence of states $\boldsymbol{x} \in \Sigma^*$ that moves the automaton location from the initial location q_0 to the accepting location q_3.

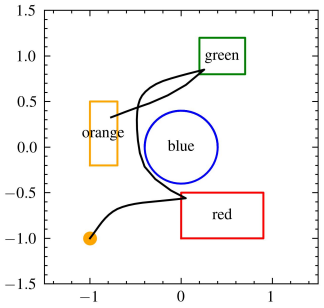

Fig. 2. An example trajectory in a 2D workspace that satisfies the specification in Fig. 1. Here, the state x is a vector in $\mathbb{R}^2$, and the trajectory starts at a point $(-1, -1)$ and completes the specified task.

II. Preliminaries

In this section, we will define some notation and background for our proposed method. Through this paper, we will use S^n to denote the state space of our system, where $S \subseteq \mathbb{R}$. Then, we can define predicates on S^n as Boolean expressions with the recursive grammar:

$$\varphi := \top \mid \bot \mid \mu(x) \geq 0 \mid \varphi \wedge \varphi \mid \varphi \vee \varphi, \tag{1}$$

where

- $\top$ and $\bot$ refer to *true* and *false* values respectively;
- $\mu : S^n \to \mathbb{R}$ is a scalar, differentiable function; and
- $\varphi \wedge \varphi$, and $\varphi \vee \varphi$ refer to Boolean conjunction (*and*), and disjunction (*or*) operations respectively.

Let Φ denote the set of all such predicates over S^n. For some $s \in S^n$ and $\varphi \in \Phi$, we say that s *models* φ (denoted $s \models \varphi$) if s satisfies the Boolean predicate φ.

Remark. The above syntax is similar to that of Signal Temporal Logic [11] without the temporal operators. Moreover, subsequent definitions will leverage this to define some concepts introduced in [35].

Definition 1 (Symbolic Automata [36])**.** A symbolic automaton is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, Q_F, \Delta)$ where

- Σ is an input alphabet;
- Q is a set of *locations* in the automaton, with Q_0 and Q_F denoting the initial and final (or accepting) locations respectively; and
- $\Delta : Q \times Q \to \Phi$ denotes a mapping from transitions in the automaton to a Boolean predicate expression.

In this paper, we restrict ourselves to automata that accept *regular languages*, i.e., automata that are defined on finite length sequences of elements in the input alphabet. We use Σ^* to refer to the set of all finite length sequence of elements $s \in \Sigma$. Given a sequence of input alphabets, $\xi = (s_0, s_1, \ldots, s_l) \in \Sigma^*$, a *run* in the symbolic automaton $\mathcal{A}$ is a sequence of locations $(q_0, q_1, \ldots, q_{l+1})$ such that $s_i \models \Delta(q_i, q_{i+1})$. We use $q_i \xrightarrow{s_i} q_{i+1}$ to denote such a valid transition in the automaton, and $\mathsf{run}_{\mathcal{A}}(\xi)$ to denote the set of runs induced in $\mathcal{A}$ by $\xi \in \Sigma^*$. If single trace ξ can generate multiple runs in $\mathcal{A}$, we say that the automaton is non-deterministic. A run is *accepting* in the automaton if $q_l \in Q_f$ for any induced run $(q_0, q_1, \ldots, q_l) \in \mathsf{run}_{\mathcal{A}}(\xi)$, and we denote this by $\xi \models \mathcal{A}$.

Remark. Note that a run $\mathsf{run}_{\mathcal{A}}(\xi)$ is *rejecting* in $\mathcal{A}$ if the last location in the run is not in Q_F. Thus, for a finite state automaton, the logical negation of the specification can be obtained by checking if a run does not end in Q_F, i.e., $Q'_F = Q \setminus Q_F$.

Example 1. In Fig. 1 and Fig. 2, we see an example of a symbolic automaton with a sequential specification, and a trajectory that satisfies it.

Definition 2 (Semiring [37])**.** A tuple, $\mathbb{K} = (K, \oplus, \otimes, \tilde{0}, \tilde{1})$ is a *semiring* with the underlying set K if $\left(K, \oplus, \tilde{0}\right)$ is a commutative monoid with identity $\tilde{0}$; $\left(K, \otimes, \tilde{1}\right)$ is a monoid with identity element $\tilde{1}$; $\otimes$ distributes over $\oplus$; and $\tilde{0}$ is an annihilator for $\otimes$ (for all $k \in K, k \otimes \tilde{0} = \tilde{0} \otimes k = \tilde{0}$).

Definition 3 (Symbolic Weighted Automata [38], [36], [35])**.** Given a symbolic automaton $\mathcal{A}$ and a semiring $\mathbb{K} = \left(K, \oplus, \otimes, \tilde{0}, \tilde{1}\right)$, a symbolic *weighted* automaton over a semiring $\mathbb{K}$ is a tuple $(\mathcal{A}, w)$, where $w : Q \times \Sigma \times Q \to K$ is a weighting function.

We define the weight of a sequence of inputs $\xi \in \Sigma^*$ in an symbolic weighted automaton $\mathcal{A}$ as the mapping $w_{\mathcal{A}} : \Sigma^* \to K$, where

$$w_{\mathcal{A}}(\xi) = \bigoplus_{r \in \mathsf{run}_{\mathcal{A}}(\xi)} \bigotimes_{i=0}^{|\xi|} w(r_i, \xi_i, r_{i+1})$$

Remark. Note that when using $\mathbb{K} = (\mathbb{N} \cup \{\infty\}, \min, +, 0, \infty)$, this definition of w is equivalent to that of the standard shortest path in a directed graph.

Example 2 (Weights of a path). Looking at the same automata as in Fig. 1, let us define a weighting function as

$$w(q, x, q') = \begin{cases} 0, & \text{if } q' = q_4 \\ 1, & \text{otherwise.} \end{cases}$$

This weighting function assigns 0 weight if the input sequence enters the rejecting sink state q_4. Under different semirings, we can see different effects of $w_{\mathcal{A}}$ as follows (noting that $|\mathsf{run}_{\mathcal{A}}(\xi)| = 1$ for any $\xi \in \Sigma^*$ as $\mathcal{A}$ is *deterministic*):

- Boolean Semiring $\mathbb{K} = (\{0, 1\}, \vee, \wedge, 0, 1)$: Here, any input sequence ξ that does not induce a run that enters q_4 will have $w_{\mathcal{A}}(\xi) = 1$ as all weights will be 1 (which is equivalent to the Boolean true value $\top$). Otherwise, $w_{\mathcal{A}}(\xi) = 0$, as even a single 0 weight transition will render the conjunction $\otimes \equiv \wedge$ false.
- $(\min, \max)$ Semiring $\mathbb{K} = (\{0, 1\}, \max, \min, 0, 1)$: This is semantically equivalent to the Boolean semiring, and thus produced identical results.
- $(\min, +)$ Semiring $\mathbb{K} = (\mathbb{N} \cup \{\infty\}, \min, +, 0, \infty)$: Referred to as a *tropical* semiring, under this semiring, $w_{\mathcal{A}}(\xi)$ outputs the length of the input sequence until it first reaches the rejecting sink q_4 or the length of the input sequence itself $|\xi|$.

Matrix Semirings: If m is a positive integer and K is a semiring, then the set of $m \times m$ matrices with entries in K, denoted $K^{m \times m}$, is also a semiring [39]. Specifically, for matrices $A, B, C \in K^{m \times m}$, we can define the semiring operation as follows:

- Addition $A \oplus B = C$ is defined as element-wise addition $C_{ij} = A_{ij} \oplus B_{ij}$;
- The additive identity matrix is, intuitively, the $m \times m$ matrix with all entries $\tilde{0}$;
- Multiplication $AB = C$ is defined similar to matrix multiplication as $C_{ij} = \bigoplus_{k=0}^{m-1} A_{ik} \otimes B_{kj}$; and
- The multiplicative identity matrix (or simply, identity matrix) is similar to the usual: an $m \times m$ matrix with all diagonal entries equal to $\tilde{1}$, and the rest are $\tilde{0}$.

Remark. Note that the above translates to the usual matrix multiplication in linear algebra for the semiring of reals $(\mathbb{R}, +, \times, 0, 1)$.

From the above definition of matrix semiring arithmetic, one can derive the vector dot product, the vector-matrix product, and various other concepts from linear algebra that show direct equivalences under the abstract algebraic framework [39]. In the rest of this paper, we will use the standard notation for matrix multiplication ($C = AB$), vector dot product ($x_3 = x_1 \cdot x_2 = x_1 x_2$), and vector-matrix multiplication ($x_2 = x_1^T A$), but the operations are defined on semirings unless otherwise specified.

III. Motion Planning with Automaton Matrix Operators

For optimization-based motion planning, we concern ourselves with two general motion planning problems given temporal specifications:

Problem 1 (Open-Loop Motion Planning)**.** Given a discrete-time, dynamical system

$$x_{t+1} = f(x_t, u_t), \tag{2}$$

where $x_t, x_{t+1} \in S^n$, $u_t \in \mathcal{U} \subseteq \mathbb{R}^m$, and $f : S^n \times \mathcal{U} \to S^n$ is a (piecewise) differentiable function. For a task automaton $\mathcal{A}$, a planning horizon $H \in \mathbb{N}$, and some initial state $x_0 \in \Sigma = S^n$, compute a control plan $\boldsymbol{u}^* = \left(u_0^*, \ldots, u_{H-1}^*\right)$, such that, for $t \in 0, \ldots, H-1$:

$$\begin{aligned} \boldsymbol{u}^* &= \operatorname{argmin}_{\boldsymbol{u}} \|\boldsymbol{u}\| \\ \text{s.t.} \quad & (x_0, \ldots, x_H) \models \mathcal{A} \\ & x_{t+1} = f(x_t, u_t). \end{aligned} \tag{3}$$

Problem 2 (Closed-loop Control)**.** Given a discrete-time, dynamical system like in Eq. 2 and a task automaton $\mathcal{A}$, derive a feed-back control law

$$\begin{aligned} u_t^* &= \pi(x_t) \\ \text{s.t.} \quad & (x_0, x_1, \ldots) \models \mathcal{A} \\ & x_{t+1} = f(x_t, u_t^*) \end{aligned} \tag{4}$$

In this section, we will define the *automaton matrix operator* $\mathbb{A}$ for a symbolic automaton $\mathcal{A}$. The matrix operator for an automaton $\mathcal{A}$ is a mapping $\mathbb{A} : \Sigma \to K^{|Q|\times|Q|}$ that, for each input element in the alphabet $x \in \Sigma$ maps to a $|Q| \times |Q|$ matrix with entries in the set K with some semiring associated with it. The goal of this mapping is to capture the structure of the automaton while leveraging matrix semiring algebra [39] to compute weights of system trajectories and, consequently, reason about the acceptance of the given system trace in the automaton.

Before we formally define $\mathbb{A}$, we will define a few weighting functions for it:

Definition 4 (Generalized Weights)**.** For a given predicate $\varphi \in \Phi$, some value $x \in S^n$ and a semiring $\mathbb{K} = (K \subseteq \mathbb{R}, \oplus, \otimes, \tilde{0}, \tilde{1})$, let $\lambda : S^n \times \Phi \to K$ be recursively defined as follows

$$\begin{aligned} &\lambda(x, \top) = \tilde{1}, && \lambda(x, \bot) = \tilde{0} \\ &\lambda(x, \mu(x) \geq 0) &&= \begin{cases} \tilde{0} & \text{if } \mu(x) \geq 0 \\ \mu(x) & \text{if } \mu(x) < 0 \end{cases} \\ &\lambda(x, \varphi_1 \wedge \varphi_2) &&= \lambda(x, \varphi_1) \otimes \lambda(x, \varphi_2) \\ &\lambda(x, \varphi_1 \vee \varphi_2) &&= \lambda(x, \varphi_1) \oplus \lambda(x, \varphi_2). \end{aligned} \tag{5}$$

Let $\alpha, \beta \in K^{|Q|}$ be the initial and final weights respectively for $\mathcal{A}$ such that:

$$\alpha_i = \begin{cases} \tilde{1}, & \text{if } q_i \in Q_0 \\ \tilde{0}, & \text{otherwise,} \end{cases} \quad \beta_i = \begin{cases} \tilde{1}, & \text{if } q_i \in Q_F \\ \tilde{0}, & \text{otherwise.} \end{cases} \tag{6}$$

In the above definitions, λ corresponds to a symbolic weighting function that generates a weight in K for each concrete input $x \in S^n$; and α and β correspond to the initial and final locations in the automaton respectively.

Definition 5 (Automaton Matrix Operator)**.** For a given weighted automaton $(\mathcal{A}, w)$, where $w(q, s, q') = \lambda(s, \Delta(q, q'))$, let $\mathbb{A} : S^n \to K^{|Q|\times|Q|}$ be a matrix semiring operator over the semiring $\mathbb{K} = (K \subseteq \mathbb{R}, \oplus, \times, \tilde{0}, \tilde{1})$ such that:

$$\mathbb{A}(s)_{ij} = \lambda(s, \Delta(q_i, q_j)) \tag{7}$$

Note that the matrix operator $\mathbb{A}(x)$ defines a *weighted transition matrix* for the automaton where each entry $\mathbb{A}(x)_{ij}$ determines the cost of moving from location q_i to q_j when seeing the input x. Thus, given a previous weighted location vector q and an input state x, we can write the next weighted location vector as $q' = q^T\mathbb{A}(x)$. Thus, we can define the weight of a state trajectory $\xi = (x_0, x_1, \ldots, x_l)$ from the set of initial locations Q_0 (encoded in α) to any final location in Q_F (encoded in β) as follows:

$$w_{\mathcal{A}}(\xi) = \alpha^T\mathbb{A}(x_0)\mathbb{A}(x_1)\ldots\mathbb{A}(x_l)\beta. \tag{8}$$

By encoding the automaton as a matrix operator, and defining the semantics of the weighted automaton trough matrix semirings, we are able to leverage state-of-the-art automatic differentiation libraries built on matrix and array operations. Thus, Algorithm 1 shows how we leverage this in a gradient-based pipeline to solve the open-loop control problem.

From the above, we can solve the open-loop plan as the solution of a gradient-based optimization problem using the procedure in Algorithm 1. Specifically, the gradient $\nabla_{\boldsymbol{u}} w_{\mathcal{A}}$ in line 8 can be symbolically computed using off-the-shelf algorithms. Algorithm 2 shows how we can use the open-loop algorithm as a subroutine in computing a receding-horizon control law for satisfying $\mathcal{A}$ by *memoizing* the current weight in $\mathcal{A}$ at time t in the vector q_t, as seen in line 8, as in model predictive control (MPC). However, in general, motion planning is NP-Hard and gradient-based methods may not guarantee convergence to an optimal solution. They require either sufficiently good initial guesses or can be used in combination with sampling-based methods like the cross-entropy method [40].

Algorithm 1 Gradient-based optimization with automaton matrix operator.

1: **procedure** OPEN-LOOP$_{\mathcal{A},\mathbb{K}}(x_{\text{init}}, q_{\text{init}}, H, \gamma, k)$
   $x_{\text{init}} \in S^n$: initial system state
   $q_{\text{init}} \in K^{|Q|}$: an initial weight configuration
   $H \in \mathbb{N}$: planning horizon
   $\gamma > 0$: learning rate for gradient descent
   $k \in \mathbb{N}$: number of optimization epochs
2: Zero-initialize $\boldsymbol{u} = (u_0, \ldots, u_{H-1})$
3: $x_0 \leftarrow x_{\text{init}}$
4: **for** $1, \ldots, k$ epochs **do**
5: $\xi = (x_0, x_1, \ldots, x_H)$ from Eq. 2.
6: Compute $\mathbb{A}(x_i)$ for $i \in 0, \ldots, H$.
7: $w_{\mathcal{A}}(\xi) = q_{\text{init}}^T\mathbb{A}(x_0)\mathbb{A}(x_1)\ldots\mathbb{A}(x_H)\beta$.
8: $\boldsymbol{u} \leftarrow \boldsymbol{u} + \gamma\nabla_{\boldsymbol{u}} w_{\mathcal{A}}(\xi)$.
9: **end for**
10: **return** $\boldsymbol{u}$
11: **end procedure**

IV. Experimental Results

To demonstrate the applicability of our method, we will show examples of open-loop and closed-loop planning with some specifications relevant to autonomous robots in general. In all the experiments, we will compare against two other related works:

- STLCG [34], where finite-length signal traces are evaluated over Signal Temporal Logic (STL) formula encoded as quantitative computation graphs in PyTorch [32]. This, along with our automaton operator method, will be used in a gradient-based optimization pipeline.
- Mixed integer program (MIP) encoding of STL robustness following [21], [12] which will be optimized using off-the-shelf mixed integer convex program solvers [41].

Algorithm 2 Gradient-based MPC with automaton matrix operator.

procedure $\text{MPC}_{\mathcal{A},\mathbb{K}}(x_{\text{init}}, H, \gamma, k)$
  $x_{\text{init}} \in S^n$: initial system state
  $H \in \mathbb{N}$: planning horizon
  $\gamma > 0$: learning rate for gradient descent
  $k \in \mathbb{N}$: number of optimization epochs
  $x_0 \leftarrow x_{\text{init}}$
  $q_0 \leftarrow \alpha$
  for $t = 0, 1, \ldots$ **do**
    $\boldsymbol{u}^* \leftarrow \text{OPEN-LOOP}_{\mathcal{A},\mathbb{K}}(x_t, q_t, H, \gamma, k)$
    Apply u_t^* (first item in $\boldsymbol{u}^*$)
    Update x_{t+1} from environment
    $q_{t+1} \leftarrow q_t^T \mathbb{A}(x_t)$
  end for
end procedure

Note. While the above frameworks are able to compute quantitative costs from logical specifications, one should note that the quantitative semantics of STL necessarily require the entire history of the system (unless restricted to a specific syntactic subset). Thus, this makes them generally unfeasible to use for *all* MPC tasks, but we make best efforts to do so for our experiments. Specifically, closed-loop control for both, φ_1 and φ_2 in the experimental results is unfeasible using the above methods. This restriction is not present in our automaton-based approach.

In the case of open-loop experiments, we will report the total number of optimizer iterations until a satisfying trajectory in the system is found (denoted t^*). Meanwhile, for closed-loop experiments, we will report the final robustness [18] of the system trajectories with respect to the corresponding STL task specifications(denoted ρ). We report the results of our experiments in Table I, with the first column describing the specification under test.

Remark. For the MILP solver, we report the number of simplex iterations performed by Gurobi [42] for the open-loop problems. It should be noted that while Gurobi's simplex optimization iterations are about 10-20 times faster than a gradient-based solution on an Intel Core i7 (1.80GHz) CPU machine with no graphics processor for computation purposes, the MILP solver is unfeasible for even relatively simple, long-horizon specifications.

To aid our presentation, we will informally describe the syntax and semantics of discrete-time STL (DT-STL). In addition to the Boolean predicates defined in Eq. 1, temporal logics add the following temporal operators (relevant to our studies)

- $\mathsf{G}_{[a,b]}\,\varphi$, where $a, b \in \mathbb{N} \cup \{\infty\}$, describes that the formula φ must hold for all time steps $t \in [a, b]$.
- $\mathsf{F}_{[a,b]}\,\varphi$, where $a, b \in \mathbb{N} \cup \{\infty\}$, describes that the formula φ must hold at least once for $t \in [a, b]$.

We refer the readers to [24] for a detailed survey of such specification languages.

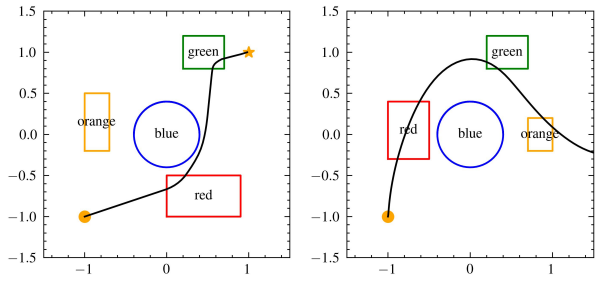

Fig. 3. Example of a trajectory satisfying φ_1 and φ_2. The left-side trajectory was produced by the $(\min, \max)$ automaton in a single integrator dynamics model, while the second one was produced by the $(\max, +)$ automaton for unicycle model.

φ_1: Reach Multiple and Avoid: Here, we control a point mass on a 2D workspace, governed via simple single-integrator dynamics by directly controlling its velocity on the workspace:

$$x_{t+1} = x_t + u_t \Delta t, \tag{9}$$

where the sampling time $\Delta t = 0.1$ seconds. The goal of the controller is defined by the specification

$$\begin{aligned} \varphi_1 := \; & \mathsf{F}\,\mathsf{G}_{[0,5]}(x \in \mathcal{R}_{\text{red}}) \\ & \wedge \mathsf{F}\,\mathsf{G}_{[0,5]}(x \in \mathcal{R}_{\text{green}}) \\ & \wedge \mathsf{F}(|x - x_{\text{goal}}| \leq \delta) \\ & \wedge \mathsf{G}(x \notin \mathcal{R}_{\text{blue}}), \end{aligned} \tag{10}$$

which says that the controlled mass must visit regions $\mathcal{R}_{red}$ and $\mathcal{R}_{\text{green}}$ for at least 5 time steps and visit x_{goal} (in any order) while always avoiding $\mathcal{R}_{\text{blue}}$.

In Fig. 3, we can see that while all the tested methods are able to generate accepting trajectories in the open-loop case, the $(\max, +)$-automaton matrix operator performs ≈ 10 times faster than other gradient-based methods.

Moreover, in the closed-loop case, only the automaton-based methods seem to be able to complete the task. This is because there is no way to encode temporal requirements as above in these other frameworks, making automata-based methods the only viable option for long-horizon, temporal tasks.

φ_2: Sequential Tasks with Avoid: Here, we model the system as a simple unicycle on a 2D workspace, with each state being represented as $p = (x, y, \theta, v, \omega)$: the x- and y-positions, the heading, the linear velocity, and the angular velocity of the unicycle. The system is thus controlled by the second-order linear and angular accelerations $u = (u_a, u_\omega)$:

$$p_{t+1} = p_t + \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \\ u_a \\ u_\omega \end{bmatrix} \Delta t, \tag{11}$$

where the sampling time $\Delta t = 0.1$ seconds. The goal of the controller is to move the unicycle from some initial state to

TABLE I

RESULTS FOR MOTION PLANNING USING AUTOMATA AND TEMPORAL LOGIC OBJECTIVES WITH SINGLE INTEGRATOR DYNAMICS.

Specification	Setting	Performance Num. Iterations t^* (lower better) Robustness ρ (higher better)				H	k	γ
		$(\min, \max)$	$(\max, +)$	STLCG	MILP			
φ_1: Visit the RED region, GREEN region and the STAR in any order, while avoiding unsafe region, BLUE.	Open Loop (t^*)	746	78	1164	42265	50	1400	0.05
	Closed Loop (ρ) (50 time steps)	0.0	0.0	–	–	15	30	0.05
φ_2: Visit 3 regions in sequence RED $\rightarrow$ GREEN $\rightarrow$ ORANGE at least once while avoiding unsafe region BLUE.	Open Loop (t^*)	35	425	234	–	80	1400	0.05
	Closed Loop (ρ) (80 time steps)	0.060	0.048	–	–	40	30	0.05
Adaptive Cruise Control	Closed Loop (ρ) (3000 time steps)	-1.78	0.4	-2.21	0.6	250	30	0.05

the regions $\mathcal{R}_{\text{red}}$, $\mathcal{R}_{\text{green}}$, and $\mathcal{R}_{\text{orange}}$ in that sequence, while avoiding $\mathcal{R}_{\text{blue}}$:

$$\begin{aligned}\varphi_2 := \\ &\mathsf{F}((x \in \mathcal{R}_{\text{red}}) \wedge \mathsf{F}((x \in \mathcal{R}_{\text{green}}) \wedge \mathsf{F}((x \in \mathcal{R}_{\text{orange}})))) \\ &\wedge \mathsf{G}(x \notin \mathcal{R}_{blue})\end{aligned} \tag{12}$$

Note that, similar to φ_1, this specification cannot be encoded as a receding horizon controller in STLCG and the MILP encoding, as the history of what regions have been visited need to be encoded. Moreover, we do not perform the experiment for the open-loop problem with the MILP encoding as the system is inherently non-linear, and linearizing it about discrete θ is not a requirement fot the other frameworks.

Adaptive Cruise Control: In an adaptive cruise control (ACC) scenario, we are designing a controller for the trailing car (called *ego* vehicle) such that it maintains a cruising velocity while also remaining a safe time gap away from the lead car. Here, the controller operates on the state space

$$x = (p_{\text{ego}}, v_{\text{ego}}, d_{\text{lead}}, v_{\text{rel}}),$$

where $p_{\text{ego}}, v_{\text{ego}} \in \mathbb{R}$ are the longitudinal position and longitudinal velocity of the ego vehicle; $d_{\text{lead}} \in \mathbb{R}$ is the distance to the lead vehicle; and $v_{\text{rel}} \in \mathbb{R}$ is the relative velocity of the lead car. While the ego controller receives the actual d_{lead} and v_{rel} form an external source, for the MPC prediction step, we (potentially incorrectly) assume a constant velocity for the lead car. The control input to the system $u \in \mathbb{R}$ is the acceleration of the ego vehicle, thus we can write an approximate *predictive* model of the system as:

$$\begin{aligned} p_{\text{ego},t+1} &= p_{\text{ego},t} + v_{\text{ego},t}\Delta t + \frac{1}{2}u_t\Delta t^2 \\ v_{\text{ego},t+1} &= v_{\text{ego},t} + u_t\Delta t \\ d_{\text{lead},t+1} &= d_{\text{lead},t} + v_{\text{rel},t}\Delta t - \frac{1}{2}u_t\Delta t^2 \\ v_{\text{rel},t+1} &= v_{\text{rel},t} - u_t\Delta t \end{aligned} \tag{13}$$

where the sampling time $\Delta t = 0.01$ seconds.

The requirements for the behavior of an ACC is parameterized by a target cruise speed v_{ref} that the ego must reach if safe to do so; a safety distance d_{safe} that the ego vehicle must not cross when trailing a car; and a *safe time gap* t_{safe}, which is the time threshold to violating d_{safe}. Thus, we can define the requirements by the following:

$$\varphi_{\text{safe}} = \mathsf{G}(d_{\text{lead}} > d_{\text{safe}}) \tag{14}$$

$$\begin{aligned}\varphi_{\text{ref}} = \mathsf{G}((d_{\text{lead}} > d_{\text{follow}}) \\ \Rightarrow \mathsf{F}_{[0,\delta]}(|v_{\text{lead}} - v_{\text{ref}}| < \epsilon \vee d_{\text{lead}} \leq d_{\text{follow}})),\end{aligned} \tag{15}$$

where $d_{\text{follow}} = d_{\text{safe}} + v_{\text{rel}}t_{\text{safe}}$ is a safe following distance that doesn't violate the safe time gap, with the parameters $v_{\text{ref}} = 15\text{m/s}$, $t_{\text{safe}} = 1.4\text{s}$, $d_{\text{safe}} = 5\text{m}$, $\delta = 50$ time steps and $\epsilon = 1\text{m/s}$. The above specification φ_{ref} describes that the controller must reach the target speed if the safe time gap isn't violated.

In this scenario, we can see from Table I that the gradient-based methods that use $(\min, \max)$-based semantics seem to get stuck in local optima, similar to the analysis in [23]. This isn't a problem for the $(\max, +)$ semiring as replacing $\min$ operations with standard addition makes the problem more conducive to gradient-based optimization, while preserving acceptance semantics.

V. CONCLUSION

In this paper, we target encoding automata-based specifications into quantitative objective functions, allowing efficient encoding of signal history, along with quantitative cost functions for motion planning. Specifically, we define a automaton matrix operator that encodes transitions in the automaton, thereby leveraging matrix-based automatic differentiation tools for gradient-based motion planning. To the best of our knowledge, this is the first such framework that combines optimization-based and automata-based methods to leverage modern compute capabilities to directly optimize over automaton specifications efficiently. Moreover, to mitigate issues with local optima and vanishing gradients in the quantitative semantics, we show how using an alternative $(\max, +)$-algebraic semantics (as presented in [35]) allows us to find satisfying trajectories faster.

REFERENCES

[1] S. M. LaValle, *Planning Algorithms*. Cambridge ; New York: Cambridge University Press, 2006.

[2] L. E. Kavraki and S. M. LaValle, "Motion Planning," in *Springer Handbook of Robotics*, ser. Springer Handbooks, B. Siciliano and O. Khatib, Eds. Cham: Springer International Publishing, 2016, pp. 139–162.

[3] A. G. Richards, "Robust constrained model predictive control," Thesis, Massachusetts Institute of Technology, 2005. [Online]. Available: https://dspace.mit.edu/handle/1721.1/28914

[4] E. F. Camacho and C. B. Alba, *Model Predictive Control*. Springer Science & Business Media, Jan. 2013.

[5] S. Katayama, M. Murooka, and Y. Tazaki, "Model predictive control of legged and humanoid robots: Models and algorithms," *Advanced Robotics*, vol. 37, no. 5, pp. 298–315, Mar. 2023.

[6] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, "Model Predictive Control for Micro Aerial Vehicles: A Survey," in *2021 European Control Conference (ECC)*, June 2021, pp. 1556–1563.

[7] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-Logic-Based Reactive Mission and Motion Planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, Dec. 2009.

[8] C. Finucane, Gangyuan Jing, and H. Kress-Gazit, "LTLMoP: Experimenting with language, Temporal Logic and robot control," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2010, pp. 1988–1993.

[9] C. Menghi, C. Tsigkanos, P. Pelliccione, C. Ghezzi, and T. Berger, "Specification Patterns for Robotic Missions," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2208–2224, Oct. 2021.

[10] A. Pnueli, "The Temporal Logic of Programs," in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*. IEEE, 1977, pp. 46–57.

[11] O. Maler and D. Nickovic, "Monitoring Temporal Properties of Continuous Signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, ser. Lecture Notes in Computer Science, Y. Lakhnech and S. Yovine, Eds. Berlin, Heidelberg: Springer, 2004, pp. 152–166.

[12] 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.

[13] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal Logic Motion Planning for Dynamic Robots," *Automatica*, vol. 45, no. 2, pp. 343–352, Feb. 2009.

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

[15] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding Horizon Temporal Logic Planning," *IEEE Transactions on Automatic Control*, vol. 57, no. 11, pp. 2817–2830, Nov. 2012.

[16] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative Temporal Planning in Uncertain Environments With Partial Satisfaction Guarantees," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 583–599, June 2016.

[17] 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, Sept. 2009.

[18] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic over Real-Valued Signals," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, K. Chatterjee and T. A. Henzinger, Eds. Berlin, Heidelberg: Springer, 2010, pp. 92–106.

[19] A. Donzé, T. Ferrère, and O. Maler, "Efficient Robust Monitoring for STL," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer, 2013, pp. 264–279.

[20] S. S. Farahani, V. Raman, and R. M. Murray, "Robust Model Predictive Control for Signal Temporal Logic Synthesis," *IFAC-PapersOnLine*, vol. 48, no. 27, pp. 323–328, Jan. 2015.

[21] V. Raman, A. Donze, 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*. Los Angeles, CA, USA: IEEE, Dec. 2014, pp. 81–87.

[22] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta, "Control from Signal Temporal Logic Specifications with Smooth Cumulative Quantitative Semantics," *arXiv:1904.11611 [cs]*, Apr. 2019.

[23] Y. V. Pant, H. Abbas, and R. Mangharam, "Smooth Operator: Control Using the Smooth Robustness of Temporal Logic," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*, Aug. 2017, pp. 1235–1240.

[24] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications," in *Lectures on Runtime Verification: Introductory and Advanced Topics*, ser. Lecture Notes in Computer Science, E. Bartocci and Y. Falcone, Eds. Cham: Springer International Publishing, 2018, pp. 135–175.

[25] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, Apr. 2008.

[26] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, Dec. 2011.

[27] L. Lindemann and D. V. Dimarogonas, "Efficient Automata-based Planning and Control under Spatio-Temporal Logic Specifications," in *2020 American Control Conference (ACC)*, July 2020, pp. 4707–4714.

[28] R. Koymans, "Specifying Real-Time Properties with Metric Temporal Logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[29] S. Sadraddini and C. Belta, "Formal Synthesis of Control Strategies for Positive Monotone Systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 2, pp. 480–495, Feb. 2019.

[30] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-Based Trajectory Generation with Linear Temporal Logic Specifications," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 5319–5325.

[31] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic Differentiation in Machine Learning: A Survey," *Journal of Machine Learning Research*, vol. 18, no. 153, pp. 1–43, 2018.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[33] R. Frostig, M. J. Johnson, and C. Leary, "Compiling Machine Learning Programs via High-Level Tracing," *Systems for Machine Learning*, vol. 4, no. 9, 2018.

[34] K. Leung, N. Arechiga, and M. Pavone, "Back-Propagation Through Signal Temporal Logic Specifications: Infusing Logical Structure into Gradient-Based Methods," in *Algorithmic Foundations of Robotics XIV*, ser. Springer Proceedings in Advanced Robotics, S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu, Eds. Cham: Springer International Publishing, 2021, pp. 432–449.

[35] S. Jakšić, E. Bartocci, R. Grosu, and D. Ničković, "An Algebraic Framework for Runtime Verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2233–2243, Nov. 2018.

[36] L. D'Antoni and M. Veanes, "The Power of Symbolic Automata and Transducers," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham: Springer International Publishing, 2017, vol. 10426, pp. 47–67.

[37] M. Mohri, "Semiring Frameworks and Algorithms for Shortest-Distance Problems," *Journal of Automata, Languages and Combinatorics*, vol. 7, no. 3, pp. 321–350, Jan. 2002.

[38] M. Droste, W. Kuich, and H. Vogler, Eds., *Handbook of Weighted Automata*, ser. Monographs in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[39] J. S. Golan, *Semirings and Affine Equations over Them: Theory and Applications*. Dordrecht: Springer Netherlands, 2003.

[40] H. Bharadhwaj, K. Xie, and F. Shkurti, "Model-Predictive Control via Cross-Entropy and Gradient-Based Optimization," in *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. PMLR, July 2020, pp. 277–286. [Online]. Available: https://proceedings.mlr.press/v120/bharadhwaj20a.html

[41] V. Kurtz and H. Lin, "Mixed-Integer Programming for Signal Temporal Logic With Fewer Binary Variables," *IEEE Control Systems Letters*, vol. 6, pp. 2635–2640, 2022.

[42] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com