Scalable Motion Planning Using Lazy SMT-Based Solving

Yasser Shoukry[†§] Pierluigi Nuzzo[†] Indranil Saha[**]
Alberto L. Sangiovanni-Vincentelli[†] Sanjit A. Seshia[†] George J. Pappas[*] Paulo Tabuada[§]

***Abstract*—We present a scalable robot motion planning algorithm for reach-avoid problems. We assume a discrete-time, linear model of the robot dynamics and a description of the workspace in terms of a set of obstacles and a target region, where both the obstacles and the region are polyhedra. Our goal is to construct a trajectory, and the associated control strategy, that steers the robot from its initial point to the target while avoiding obstacles. Differently from previous approaches, based on the discretization of the continuous state space or uniform discretization of the workspace, we propose a lazy satisfiability modulo theory approach to decompose the planning problem into smaller subproblems, which can be efficiently solved using specialized solvers. At each iteration, we use a coarse, obstacle-based discretization of the workspace to obtain candidate high-level, discrete plans that solve a set of Boolean constraints, while completely abstracting the low-level continuous dynamics. The feasibility of the proposed plans is then checked via a convex program, under constraints on both the system dynamics and the control inputs, and new candidate plans are generated until a feasible one is found. To achieve scalability, we show how to generate succinct explanations for the infeasibility of a discrete plan by exploiting a relaxation of the convex program that allows detecting the earliest possible occurrence of an infeasible transition between workspace regions. Simulation results show that our algorithm favorably compares with state-of-the-art techniques and scales well for complex systems, including robot dynamics with up to 50 continuous states.**

I. Introduction

An increasing number of safety-critical robotics applications (e.g., in navigation, rescue missions, manipulation, and surgery) as well as autonomous systems (e.g., unmanned aircraft and self-driving cars) require efficient techniques that can rigorously reason about hybrid system behaviors and guarantee the correctness of a controller implementation. Algorithmic control synthesis from formal specifications captured by a logic formalism, such as Linear Temporal Logic (LTL) [1], holds considerable promise for encompassing a rich set of task specifications, while providing implementations that are correct by construction [2]–[8]. However, the complexity of today's robotics and autonomous systems poses a set of unprecedented challenges to synthesis techniques.

A major difficulty stems from the growing need to reason about the tight integration of discrete abstractions (as in *task planning*) with continuous motions (*motion planning*) [9]. This integration can soon become daunting for complex, high-dimensional systems, since a vast hybrid, discrete/continuous space must be searched while accounting for complex geometries, motion dynamics, collision avoidance, and temporal goals. In this paper, we address this challenge by focusing on an essential problem in motion planning, which is embedded in almost all robotics applications, i.e., the reach-avoid problem. However, our approach can be directly extended to motion planning from generic LTL specifications by leveraging the bounded model checking encoding technique for LTL model checking by Biere et al. [10] to encode the discrete planning problem.

Given the robot dynamics, described as a discrete-time linear system, an initial state, a description of the workspace, and a target region, we aim at planning a collision-free and dynamically-feasible motion trajectory that steers the robot from its initial point to the target region. While a growing body of work has focused, over the years, on the synthesis of reactive controllers to perform high-level tasks, a set of computational difficulties in this context still comes from the interplay between motion trajectories and task constraints. The task planner needs to generate motions that the robot can execute in the physical world, i.e., which satisfy the robot dynamics constraints (e.g., bounds on the velocity, directions of motions) and are collision-free. However, in complex systems, effective discrete planning techniques may produce solutions that are not realizable due to constraints imposed by dynamics; on the other hand, effective methods for generating collision-free and dynamically-feasible trajectories may end up with violating the constraints imposed by the task planner.

A first category of techniques for controller synthesis in the context of motion planning utilizes a discrete abstraction of the system, often obtained by partitioning the continuous state space into polytopes, and an automata theoretic approach to synthesize the controller [2]–[6]. However, these approaches are subject to the *curse of dimensionality* and become usually impractical for systems with more than five continuous states [11]. Moreover, some of these approaches assume the availability of low-level feedback controllers that are capable of generating collision-free and dynamically-feasible trajectories that are compatible with each automaton action, which may not always be the case for complex robotic systems. A second category of approaches attempts at synthesizing the high-level planner together with the associated low-level controller, by either leveraging mixed integer linear programming (MILP) encodings of task specifications [12], [13] or sampling-based methods [14], [15]. MILP-based planners can leverage the empirical performance of state-of-the-art solvers to solve for both the discrete and continuous constraints at the same time; however, they still tend to be impractical when the problem size grows. On the other hand, sampling-based techniques tend to perform poorly on the obstacle avoidance problem in the presence of narrow passages [16], and do not have, in general, control over the number of hops of the generated trajectory.

In this paper, we propose a scalable solution for the integration of task planning and robot motion planning for reach-avoid problems. Differently from previous approaches, based on the discretization of the continuous state space or uniform discretization of the workspace, we adopt a *lazy*

Affiliations: [†]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA, [§]Department of Electrical Engineering, University of California, Los Angeles, CA, [*]Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA, [**]Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India

This work was partially sponsored by TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA and by the NSF project ExCAPE: Expeditions in Computer Augmented Program Engineering.

Satisfiability Modulo Theory (SMT) approach [17] to decompose the planning problem into smaller subproblems, which can be efficiently solved using specialized solvers. An SMT solver takes as input a logical formula in first-order logic that can involve combinations of background theories, and either provides an assignment to the variables that satisfies all the constraints in the formula, or says that none exists. To do so, the lazy SMT paradigm combines a SAT solver with a theory solver. The SAT solver efficiently reasons about combinations of Boolean constraints to suggest possible assignments, while the theory solver checks the consistency of the given assignments, and provide the reason for the conflict, a counterexample, whenever inconsistencies are found.

We build on the above SMT paradigm to construct our algorithm. At each iteration, we use a coarse, obstacle-based discretization of the workspace to obtain candidate high-level, discrete plans that solve a set of Boolean constraints, while completely abstracting the low-level continuous dynamics. The feasibility of the proposed plans is then checked via a convex program, under constraints on both the system dynamics and the control inputs, and new candidate plans are generated until a feasible one is found. To achieve scalability, we show how to generate succinct explanations (counterexamples) for the infeasibility of a discrete plan by exploiting a relaxation of the convex program that allows detecting the earliest possible occurrence of an infeasible transition between workspace regions.

Our methodology differs from classical approaches to reach-avoid problems [18]–[20], e.g., based on the solution of a Hamilton-Jacobi-Isaacs equation. Rather than formulating a complete, general optimization problem, which may be computationally challenging, we focus on solving a special case accurately and efficiently. We then aim at leveraging this result as a building block to solve more general problems, e.g., by supporting complex LTL specifications, through abstraction and refinement techniques. In this respect, our methods are also inspired by the counterexample-based control approaches [21]. Simulation results show that our algorithm favorably compares with other state-of-the-art techniques.

II. Problem Formulation

We consider a robot that moves in a workspace $\mathcal{W} \subset \mathbb{R}^w$ where w can be 2 or 3, corresponding, respectively, to a 2-dimensional or 3-dimensional workspace. We use $\|a\|$ to denote the infinity norm of a and formulate the reach-avoid motion planning problem as follows.

A. Robot Model

We assume the robot dynamics is described by a discrete-time, input-constrained, linear system of the form:

$$x_{t+1} = Ax_t + Bu_t, \tag{II.1}$$

$$x_0 = \overline{x}, \qquad \|u_t\| \leq \overline{u} \quad \forall t \in \mathbb{N} \tag{II.2}$$

where $x_t \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state of the robot at time $t \in \mathbb{N}$, $u_t \in \mathcal{U} \subseteq \mathbb{R}^m$ is the robot input, $\overline{x}$ is the robot initial state and $\overline{u}$ is the input constraint. The matrices A and B represent the robot dynamics and have appropriate dimensions. For a robot with nonlinear dynamics that is either differentially flat or feedback linearizable, the state space model (II.1) corresponds to its feedback linearized dynamics.

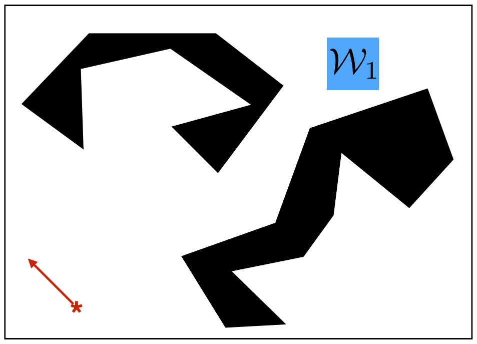

Fig. 1. Pictorial representation of the workspace, obstacles, and target for a reach-avoid problem. The initial state of the robot, including both position and angle, is represented by the red star and arrow.

B. Workspace

We assume the robot must avoid a set of *obstacles* $\mathcal{O} = \{\mathcal{O}_1, \dots, \mathcal{O}_o\}$, with $\mathcal{O}_i \subset \mathbb{R}^w$, and represent the *workspace* as $\mathcal{W} = \mathcal{W}_0 \cup \mathcal{W}_G$, where $\mathcal{W}_G$ is a *region* of interest (the target region or *Goal*) and $\mathcal{W}_0$ is the *free space*, characterized by the absence of both obstacles and target. As pictorially represented in Fig. 1, both the target region and the obstacles are assumed to be polygons.

To better describe the interplay between discrete planner and continuous planner in our algorithm, it is also useful to uniquely associate to the target region and the free space defined above an atomic proposition in the set $\Pi = \{\pi_0, \pi_G\}$. We then denote by $h_{\mathcal{W}\to\Pi} : \mathcal{W} \to \Pi$ the map from each point $w \in \mathcal{W}$ to the atomic proposition $\pi_i \in \Pi$ that evaluates to one (true) at w. Moreover, a subset of the robot state variables, describing its position (coordinates), is also used to describe $\mathcal{W}$. Therefore, we denote as $h_{\mathcal{X}\to\mathcal{W}} : \mathcal{X} \to \mathcal{W}$ the natural projection of the state x onto the workspace $\mathcal{W}$, and by $h_{\mathcal{X}\to\Pi}$ the map from the robot state to the set of atomic propositions, obtained after projecting the state onto the workspace, i.e., $h_{\mathcal{X}\to\Pi}(x) = h_{\mathcal{W}\to\Pi}(h_{\mathcal{X}\to\mathcal{W}}(x))$.

Finally, given the set $\overline{\mathcal{W}} = \{\mathcal{W}_0, \mathcal{W}_G\}$, we introduce an *adjacency function* $Adj : \overline{\mathcal{W}} \times \overline{\mathcal{W}} \to \mathbb{B}$ over the pairs of non-overlapping elements in $\overline{\mathcal{W}}$ such that $Adj(\mathcal{W}_i, \mathcal{W}_j) = 1$ if $\mathcal{W}_i$ and $\mathcal{W}_j$ are adjacent and 0 otherwise[1]. Because of the one-to-one correspondence between elements in $\overline{\mathcal{W}}$ and atomic propositions in Π, we also write $Adj(\pi_i, \pi_j) = 1$ if π_i and π_j are associated with adjacent regions in $\overline{\mathcal{W}}$ and 0 otherwise. Moreover, for all i, $Adj(\pi_i, \pi_i) = 1$ holds.

C. Problem Definition

Definition 2.1 (Input Problem Instance): An input problem instance is defined as the tuple $\mathcal{P} = \langle \mathcal{W}, \Pi, Adj, (A, B), \overline{x}, \overline{u} \rangle$, where:

- $\mathcal{W}$ is the workspace,
- Π is the set of atomic propositions corresponding to the target and the free space,
- Adj is the adjacency function defining the connectivity of the different regions in the workspace,
- (A, B) is the robot dynamics,
- $\overline{x}$ is the initial state of the robot,
- $\overline{u}$ is the bound on the robot inputs.

Definition 2.2 (Trajectory): A *trajectory* of a robot for an input problem instance $\mathcal{P} = \langle \mathcal{W}, \Pi, Adj, (A, B), \overline{x}, \overline{u} \rangle$ is

[1]Two polyhedra in $\mathbb{R}^w$ are adjacent if they share a facet of dimension $w-1$.

defined as a pair of finite sequences (x, ρ) where $x = x_0 x_1 x_2 \ldots x_{L+1}$, with $x_i \in \mathcal{X}$, is a sequence of states and $\rho = \rho_0 \rho_1 \rho_2 \ldots \rho_{L+1}$, with $\rho_i \in \Pi$, is a sequence of propositions associated with the workspace regions, and such that $h_{\mathcal{X} \to \Pi}(x_i) = \rho_i$ for any $0 \leq i \leq L+1$. Because of the one-to-one correspondence between workspace regions and atomic propositions, we also use $\rho^{\mathcal{W}}$ to denote the sequence of regions associated with ρ, and call ρ (or $\rho^{\mathcal{W}}$) the *region trajectory*.

Definition 2.3 (Valid trajectory): For an input problem instance $\mathcal{P} = \langle \mathcal{W}, \Pi, Adj, (A, B), \overline{x}, \overline{u} \rangle$, a trajectory (x, ρ) is called a *valid trajectory*, if the following holds:

- **Initial state constraint**: $x_0 = \overline{x}$,
- **Dynamics and input constraints**: there exists u_i such that $x_{i+1} = Ax_i + Bu_i$ and $\|u_i\| \leq \overline{u}$,
- **Workspace constraints**: $Adj(\rho_i, \rho_{i+1}) = 1 \quad \forall\, i: \; 0 \leq i \leq L+1$,
- **Final state constraints**: $\rho_{L+1} = \pi_G$.

We now formally define the motion planning problem that we solve in this paper.

Problem 2.4 (Motion Planning Problem): Given an input problem instance $\mathcal{P} = \langle \mathcal{W}, \Pi, Adj, (A, B), \overline{x}, \overline{u} \rangle$, synthesize a valid trajectory for the robot.

III. SMT-Based Solution

The problem of synthesizing a robot motion plan under constraints on the continuous dynamics is traditionally solved using expensive discretizations of the state space, which typically lead to state explosion as the number of continuous states and the number of obstacles increases. Our strategy aims, instead, at creating coarser abstractions of both the state space and the workspace, thus effectively decoupling the problem of *generating an obstacle-free path* from the one of *checking its physical realizability*. By leveraging the *lazy satisfiability modulo theory* (SMT) paradigm [17], we then partition the planning problem into two smaller subproblems involving reasoning, respectively, on sets of discrete and continuous variables from the original problem. These subproblems can be efficiently solved using specialized techniques, following a similar approach as in the CalCS [22] and Imhotep-SMT solvers [23]–[25].

As illustrated in Algorithm 1 and Fig. 2(a), we start by computing a coarse, multi-resolution, discretization of the free space $\mathcal{W}_0$ based on the obstacles and the target. Unlike grid-based methods, where the workspace is discretized using a grid (or mesh) of (small) uniform resolution, the coarse abstraction used by our method avoids state explosion. Our decomposition procedure is similar to the ones previously proposed in the literature for triangular [26] or polygonal [18] representations. As a result of the discretization step, we generate a new set Π^* of atomic propositions, corresponding to the new abstraction, along with the corresponding adjacency function, denoted by Adj^*.

Based on the above discretization, our solution follows an iterative approach combining a SAT solver (Dis-Plan) and a theory solver (Con-Plan). At each iteration, we start by generating a candidate high-level path ρ that satisfies the set of constraints of the reach-avoid problem, encoded using a Boolean formula ξ. Since this path is only defined over the set of Boolean propositions Π^*, its computation will ignore the robot dynamics and the input constraints. Because of the coarse abstraction of the workspace, the number of

Algorithm 1 SMT-Based Motion Planner

```
1: $(\Pi^*, \mathcal{W}^*, Adj^*) :=$ Wksp.Abstract$(\Pi, \mathcal{W}, Adj)$;
2: Initialize the horizon:        $L := 1$;
3: while Trajectory is not found do
4:     $($dStatus$, \rho) :=$ Dis-Plan$(\Pi^*, Adj^*, \xi)$;
5:     if dStatus == UNSAT then
6:         Increase horizon:        $L := L + 1$;
7:     else
8:         (cStatus, $x$, $u$) := Con-Plan.Check$(\overline{x}, \overline{u}, \rho)$;
9:         if cStatus == Infeasible then
10:            $\phi_{\text{ce}} :=$ Con-Plan.CounterExample$(\overline{x}, \overline{u}, \rho)$;
11:            $\xi := \xi \wedge \phi_{\text{ce}}$;
12: return $(\rho, x, u)$;
```

atomic propositions in Π^* is only determined by the obstacle configuration in the workspace and does not depend of the state dimension of the continuous dynamics. This step can then be performed efficiently using off-the-shelf SAT solvers.

We can then check the *feasibility* of the generated path ρ with respect to the system dynamics (A, B), the control inputs $\overline{u}$, and the robot initial state $\overline{x}$, by casting it as a *convex* optimization problem. In fact, while generating a path that satisfies both the robot dynamics and the constraints imposed by the reach-avoid problem may be, in general, non-convex, our SMT-based architecture is able to reason about this complex problem by decomposing it into a sequence of smaller subproblems, each combining a Boolean satisfiability problem, defined only over Π^*, and a convex optimization problem, defined only over the real-valued variables x (state) and u (input). If both the Boolean and the real-valued constraints are satisfied, we return a valid trajectory consisting of the proposed plan and the corresponding state and control input trajectories. Otherwise, the proposed sequence ρ is marked as infeasible and new candidate plans are generated, such as the ones in Fig. 2(b)-(c), until a feasible one is found.

In this iterative scheme, learning "succinct explanations" that can capture the root causes for the infeasibility of a plan, and rule out the largest possible number of invalid plans per iteration, is instrumental to achieve fast convergence and scalability. To do so, we exploit convex programming to check the feasibility of a plan in terms of a conjunction of continuous constraints while minimizing a certain cost. We then use a relaxation of the continuous trajectory feasibility problem with slack variables to detect the earliest possible occurrence of an infeasible transition between two workspace regions and suggest the generation of plans that can avoid such a transition.

In what follows, we provide details on both the discrete and continuous plan generation mechanisms, including the implementation of Dis-Plan and Con-Plan, as well as on the generation of succinct infeasibility proofs.

IV. Generation of the High-Level Discrete Plan

As represented in Fig. 1, a classic reach-avoid specification defines an initial point, a *Goal* (target) region ($\mathcal{W}_G = \mathcal{W}_1$), and a set of obstacles to avoid. Given an input problem instance $\mathcal{P} = \langle \mathcal{W}, \Pi, Adj, (A, B), \overline{x}, \overline{u} \rangle$ for this problem, Dis-Plan performs a multi-resolution discretization of the free space and generates a formula that represents any valid trajectory (x, ρ) of the robot. The decision variables for the formula are given by the sequence of atomic propositions associated with the regions to be occupied by the robot. Given the new set Π^*

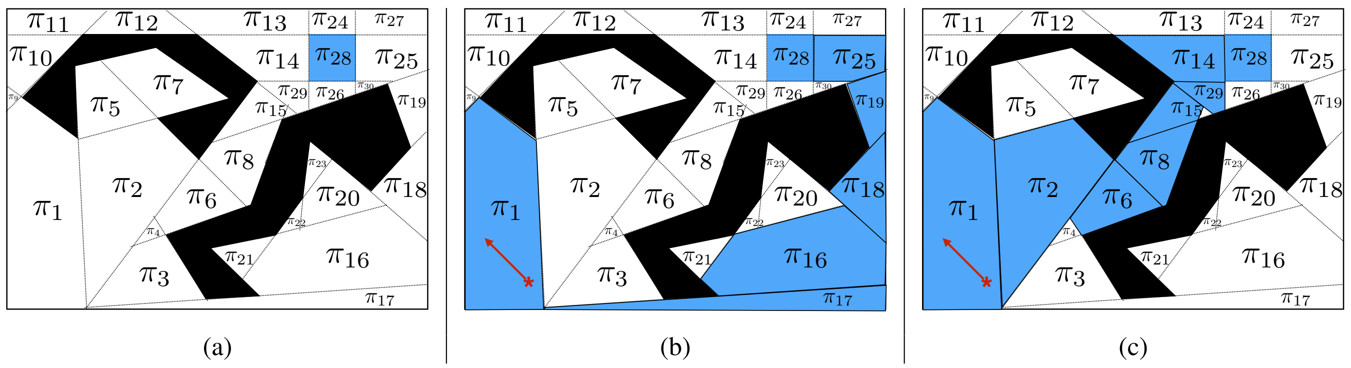

Fig. 2. (a) Coarse discretization of the free space for the workspace in Fig. 1, for the same configuration of obstacles and target region. (b) A candidate path to satisfy the reach-avoid specification with $\pi_G = \pi_{28}$ is highlighted in blue. (c) Another candidate path to the *Goal* (π_{28}) proposed as an alternative to the one in (b).

of atomic propositions associated to the workspace regions after discretization of the free space, and the corresponding adjacency function Adj^*, we represent the region trajectory ρ as $\rho = (\rho_0\rho_1 \ldots \rho_{L+1})$, where $\rho_0 = \overline{\rho} = h_{\mathcal{X}\to\Pi^*}(\overline{x})$ is the atomic proposition associated with the initial state of the system (π_1 in Fig. 2), and ρ_{L+1} is the *Goal* region (π_{28} in Fig. 2). For instance, for the scenario in Fig. 2, we obtain $\Pi^* = \{\pi_1, \ldots, \pi_{30}\}$.

The set of constraints for the workspace can be captured by the following formula:

$$\xi \equiv (\rho_0 = \overline{\rho}) \wedge (\rho_{L+1} = Goal) \wedge \bigwedge_{t=1}^{L+1} \rho_t \in \mathcal{N}(\rho_{t-1}),$$

where $\mathcal{N}(\rho_i) = \{\pi_j \in \Pi^* \ |Adj^*(\pi_i, \pi_j) = 1\}$ denotes the set of regions that are adjacent (neighbors) to ρ_i. The above formula enforces that the trajectory starts with the initial region, associated with $\overline{\rho}$, and proceeds to the *Goal* region while only visiting regions that are adjacent. We observe that obstacle avoidance is implicitly encoded by the fact that Π^* and Adj^* are defined only over the regions of interest and the free space. To support generic LTL specifications, DIS-PLAN can use the bounded model checking encoding technique for LTL model checking [10] to generate high-level plans that satisfy the specifications. The Boolean formula ξ can then be solved using a SAT solver to generate a model ρ.

V. GENERATION OF THE CONTINUOUS TRAJECTORY

Given a region trajectory ρ specifying $(L+1)$ polyhedra to be visited by the robot, the continuous planner CON-PLAN performs two tasks. First, it checks whether ρ is feasible given the constraints on the robot dynamics (CON-PLAN.CHECK). Second, if ρ is not feasible, CON-PLAN generates counterexample formulas describing the "minimal" set of inconsistent constraints (CON-PLAN.COUNTEREXAMPLE). Both tasks are detailed below.

A. Checking Feasibility of a Discrete Plan

To check whether a given region trajectory ρ is feasible, i.e., whether it satisfies the robot initial state, dynamics, and input constraints, we can formulate and solve the following feasibility problem:

*Problem 5.1 (*CON-PLAN.CHECK*):*

$$\min_{\substack{u_0,\ldots,u_L \in \mathbb{R}^m \\ x_1,\ldots,x_{L+1} \in \mathbb{R}^n}} \quad 1$$

$$x_0 = \overline{x}, \tag{V.1}$$

$$h_{\mathcal{X}\to\mathcal{W}}(x_i) \in \rho_i^{\mathcal{W}}, \quad i = 1, \ldots, L+1 \tag{V.2}$$

$$x_{i+1} = Ax_i + Bu_i \quad i = 0, \ldots, L \tag{V.3}$$

$$\|u_i\| \leq \overline{u}, \quad i = 0, \ldots, L \tag{V.4}$$

where (V.1) is the initial condition; (V.2) is the trajectory constraint, ensuring that region $\rho_i^{\mathcal{W}}$, as specified by the high-level plan, is visited at time i; (V.3) captures the robot dynamics; and (V.4) expresses the input constraints.

Because all the regions are polyhedra, constraints (V.2) can all be encoded as a set of linear inequalities of the form $A_{\mathcal{W}_i} x_i \leq b_{\mathcal{W}_i}$, where $(A_{\mathcal{W}_i}, b_{\mathcal{W}_i})$ represents the facets of polyhedron $\mathcal{W}_i$. Therefore, Problem 5.1 turns into a linear program (LP) and can be efficiently solved using off-the-shelf tools. Because the continuous solver is only used to assess the feasibility of a conjunction of constraints, our approach avoids dealing with the combinatorial complexity arising from the enumeration of all the possible paths in the presence of the obstacles in a continuous domain.

B. Generating Counterexamples

If Problem 5.1 is successfully solved, we return a feasible state and control input trajectory. On the other hand, if Problem 5.1 is `Infeasible`, we have a *counterexample*, i.e., an infeasible trajectory that can be used to augment the original SAT encoding with additional Boolean constraints that rule out the current assignment. To do so, a trivial formula can be generated as follows:

$$\phi_{\text{triv-ce}} := \bigvee_{i=0}^{L+1} \neg\rho_i, \tag{V.5}$$

meaning that at least one of the regions proposed by the SAT solver cannot be visited at the given time. However, by excluding only one Boolean assignment at each iteration, such a formula can lead, in the worst case, to the daunting complexity of exhaustively searching over all the possible high-level region trajectories.

As anticipated in Sec. III, we aim, instead, at generating succinct infeasibility proofs that can precisely detect the origin of inconsistency and rule out a broader class of assignments to Boolean variables. It is possible to generate a more compact formula by detecting the earliest possible occurrence of an infeasible transition between two regions in ρ. This search can be performed by repeatedly querying the theory solver to check the feasibility of a shorter trajectory, i.e., a truncated version (or a *prefix*) of ρ, using the same formulation as in

Problem 5.1. While such a method leads to the solution of L additional optimization problems in the worst case, we discuss next how to retrieve the same information by formulating just one optimization problem.

C. Succinct Counterexample Generation

It is possible to detect *the earliest occurrence* of an infeasible transition between two regions by casting a relaxed version of Problem 5.1 via the addition of slack variables to both the input and dynamic constraints. If a nonzero slack is first found as being necessary at time $k < L$, then a reason of infeasibility for the continuous dynamics comes from the transition from region $\rho_k^{\mathcal{W}}$ to region $\rho_{k+1}^{\mathcal{W}}$. This leads to generating a counterexample formula similar to (V.5) but of length $k+2$. Therefore, generating a compact formula amounts to finding the earliest occurrence of a nonzero slack. However, we also need to make sure that nonzero slacks are added only *when necessary* to make Problem 5.1 feasible. This is equivalent to looking for the earliest occurrence of a nonzero slack variable after the longest possible sequence of zero slack variables, i.e., the longest feasible region trajectory for the given dynamics and initial conditions.

To formalize the above objective, we proceed as follows. Given a constant $\epsilon \in \mathbb{R}_{>0}$, we define the function $\textsc{zeroPrefix}_\epsilon : \mathbb{R}_{\geq 0}^{L+1} \to \mathbb{N}$ as:

$$\textsc{zeroPrefix}_\epsilon(s_0, s_1, \dots, s_L) = \min k \text{ s.t. } \sum_{i=0}^{k} s_i > \epsilon.$$

Intuitively, for small ϵ, $\textsc{zeroPrefix}_\epsilon$ returns the number of zero elements at the beginning of a sequence $s = s_0, \dots, s_L$, i.e., the length of its "zero prefix." Using this function, we can then look for sequences of slack variables that maximize the number of initial elements set to zero and introduce nonzero elements only when necessary, by casting the following optimization problem:

Problem 5.2:

$$\max_{\substack{u_0,\dots,u_L \in \mathbb{R}^m \\ v_0,\dots,v_L \in \mathbb{R}^m \\ s_0^u,\dots,s_L^u \in \mathbb{R} \\ s_0^v,\dots,s_L^v \in \mathbb{R} \\ x_1,\dots,x_{L+1} \in \mathbb{R}^n}} \textsc{zeroPrefix}_\epsilon((s_0^u + s_0^v), \dots, (s_L^u + s_L^v))$$

subject to

$$\begin{array}{ll} x_0 = \overline{x}, & \\ h_{\mathcal{X}\to\mathcal{W}}(x_i) \in \rho_i^{\mathcal{W}}, & i = 1, \dots, L+1 \\ x_{i+1} = Ax_i + Bu_i + B'v_i & i = 0, \dots, L \\ \|u_i\| \leq \overline{u} + s_i^u, & i = 0, \dots, L \\ \|v_i\| \leq s_i^v, & i = 0, \dots, L \\ 0 \leq s_i^u, \quad 0 \leq s_i^v & i = 0, \dots, L \end{array}$$

Problem 5.2 is a relaxed version of Problem 5.1, where input constraints are modified, at each time, by adding a slack variable $s_i^u \geq 0$, and dynamics constraints are relaxed by adding an additional control input v_i, bounded by another slack variable $s_i^v \geq 0$. B' is chosen such that the matrix $[B \quad B']$ is surjective. By looking at the longest prefix of zero slack variables, Problem 5.2 is then able to find the longest region trajectory that is feasible. If the optimum length is k^*, then k^*+1 will be the length of the resulting counterexample formula. A remaining drawback is the possible intractability of Problem 5.2, whose objective function, basically counting the number of zero elements in the prefix of a sequence, is nonconvex. It is, however, possible to still find the optimum k^* of Problem 5.2 using a convex (or even linear) program. To state this result, at the heart of our efficient decision procedure, we first provide the following definition.

Definition 5.3: Let $\overline{s}$ be defined as the least upper bound on the value of the slack variable $s = s^u + s^v$ such that the following constraints:

$$\begin{array}{ll} \multicolumn{2}{c}{x = Ax' + Bu + B'v} \\ h_{\mathcal{X}\to\mathcal{W}}(x) \in \rho^{\mathcal{W}} & h_{\mathcal{X}\to\mathcal{W}}(x') \in \rho^{\mathcal{W}'} \\ \|u\| \leq \overline{u} + s^u & \|v\| \leq s^v \\ 0 \leq s^u, & 0 \leq s^v \end{array}$$

are feasible for any two adjacent regions $\mathcal{W}, \mathcal{W}'$ and for any two states $x \in \mathcal{W}$ and $x' \in \mathcal{W}'$.
The value of $\overline{s}$ can be easily pre-computed offline for a given workspace and obstacle configuration, and a given discretization associated with the set Π^* of atomic propositions. Then, for $\epsilon \in \mathbb{R}_{>0}$ and $\overline{s} \geq \epsilon$, we can use the following problem to find the maximum length of a feasible region trajectory:

Problem 5.4:

$$\min_{\substack{u_0,\dots,u_L \in \mathbb{R}^m \\ v_0,\dots,v_L \in \mathbb{R}^m \\ s_0^u,\dots,s_L^u \in \mathbb{R} \\ s_0^v,\dots,s_L^v \in \mathbb{R} \\ x_1,\dots,x_{L+1} \in \mathbb{R}^n}} \sum_{i=0}^{L} s_i^u + s_i^v$$

subject to

$$\begin{array}{ll} x_0 = \overline{x}, & \\ h_{\mathcal{X}\to\mathcal{W}}(x_i) \in \rho_i^{\mathcal{W}}, & i = 1, \dots, L+1 \\ x_{i+1} = Ax_i + Bu_i + B'v_i & i = 0, \dots, L \\ \|u_i\| \leq \overline{u} + s_i^u, & i = 0, \dots, L \\ \|v_i\| \leq s_i^v, & i = 0, \dots, L \\ 0 \leq s_i^u, \quad 0 \leq s_i^v & i = 0, \dots, L \\ \dfrac{\overline{s}}{\epsilon}\left(\displaystyle\sum_{k=0}^{i-1} s_k^u + s_k^v\right) \leq s_i^u + s_i^v & i = 1, \dots, L \end{array}$$

which is, again, an LP that can be efficiently solved. We can now establish the following result stating that solving Problem 5.4 results into providing the smallest set of region constraints that leads to infeasibility. The proof is provided in the appendix (Sec. A).

Theorem 5.5: Let $\overline{s} \in \mathbb{R}_{>0}$, as in Definition 5.3, and $\epsilon \in \mathbb{R}_{>0}$ satisfy $\overline{s} \geq \epsilon$. Then, any solution of Problem 5.4 is also a solution of Problem 5.2.

Moreover, we observe that Problem 5.4 subsumes the original feasibility check in Problem 5.1. That is, once a region trajectory ρ is generated by the SAT solver, we can just solve one instance of Problem 5.4 to both check the feasibility of ρ and generate a counterexample when needed. This result is summarized in Algorithm 2. By leveraging the results of Problem 5.1 and Problem 5.4, we can state the following guarantees of Algorithm 1, whose proof follows from the construction of the proposed motion planner as described above.

Algorithm 2 $(\text{cStatus}, x, u, \phi_{\text{ce}}) = \text{Con-Plan}(\rho)$

```
1:  Solve Problem 5.4;
2:  $\forall\ i:\ s_i^* = s_i^u + s_i^v$;
3:  if $\sum_{i=0}^{L} s_i^* = 0$ then
4:      $\text{cStatus} = \texttt{feasible}$;
5:      return $(\text{cStatus}, x^*, u^*, 1)$
6:  else
7:      $\text{cStatus} = \texttt{infeasible}$;
8:      Let $k^* := \text{zeroPrefix}_\epsilon(s_0^* \ldots s_{L+1}^*)$;
9:      $\phi_{\text{ce}} := \bigvee_{i=0}^{k^*+1} \neg\rho_i$;
10:     return $(\text{cStatus}, x^*, u^*, \phi_{\text{ce}})$;
```

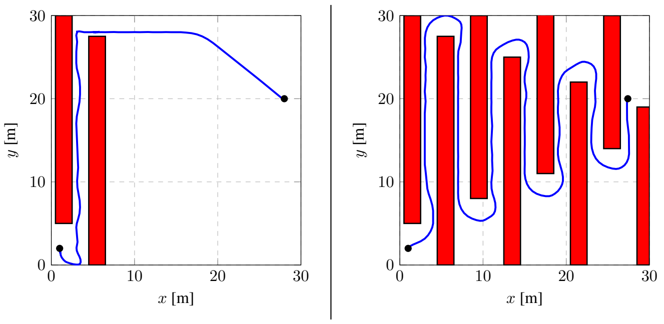

Fig. 3. Trajectories generated by the SMT-based motion planner for a maze-like workspace with different numbers of passages.

Theorem 5.6: The SMT-based motion planning Algorithm 1 generates a valid trajectory (x, ρ) for the reach-avoid motion planning problem $\mathcal{P} = \langle \mathcal{W}, \Pi, Adj, (A, B), \overline{x}, \overline{u} \rangle$.

VI. Results

We developed our theory solver in Matlab and interfaced it with the SAT solver SAT4J [27], to generate the discrete plans, and Cplex, to solve the LPs and generate the counterexamples. All the experiments were executed on an Intel Core i7 3.4-GHz processor with 8 GB of memory.

A. Case Study 1: Dubin's Vehicle

We demonstrate the effectiveness of our motion planning algorithm by applying it to a reach-avoid problem for a Dubin's vehicle (also known as differential drive robot). The kinematics of this robot can be transformed into a linear chain of integrators using dynamic feedback linearization. A discrete-time linear model is then computed from the feedback linearized model. As shown in Table I, we consider a 30m $\times$ 30m maze-like workspace with increasing number of passages, ranging from one to four. Since automata theoretic approaches to control synthesis are known to be subject to state explosion, we directly compare the performance of our SMT-based motion planner against the rapidly exploring random tree (RRT) algorithm with dynamics and input constraints [28] and the LTL OPT toolbox [13], which implements a one-shot MILP encoding of the specification. LTL OPT is configured to use CPLEX as in our tool. We run each experiment 10 times and report the average execution time of each of the three algorithms. Fig. 3 shows some of the generated trajectories.

As shown in Table I, Algorithm 1 scales better than other techniques. We observe that most of the execution time is spent in the generation of the workspace discretization. Solving multiple instances of SAT and LPs is efficiently performed thanks to the proposed solver architecture and its underlying

TABLE I
Comparison of the run time performance of Algorithm 1 with respect to the RRT algorithm [28] and the LTL OPT toolbox [13] for a maze-like workspace (e.g., see Fig. 3).

Number of passages	**SMT-Based Motion Planner [s]**			**RRT [s]**	**LTL OPT [s]**
	Discrete abstraction	Dis-Plan	Con-Plan		
1	1.9975	0.1360	0.2542	10.4381	> 7200
2	7.1461	1.1290	0.9294	122.3017	time out
3	19.3267	3.6495	1.0053	423.6957	time out
4	43.0985	4.0913	1.9204	1002.4193	time out

abstractions. On our benchmarks, Algorithm 1 is at least an order of magnitude faster than the RRT method, which is known to significantly slow down in the presence of narrow passages. Specifically, in the maze with one passage, RRT explored 58397 samples to find a feasible trajectory, i.e., two orders of magnitude more samples than the length of the final trajectory. On the contrary, each passage is compactly represented just as a polyhedron in our setup.

The execution time of LTL OPT exceeded the time-out threshold (4 hours) in all the cases except for the 1-passage maze. The degradation in performance may be due to the very large number of variables (several thousands) of the resulting MILP, the number of variables depending on the length of the trajectory. Thanks to the separation between real-valued and Boolean variable reasoning, Algorithm 1 requires, instead, solving a set of very efficient LPs.

B. Case Study 2: Scalability Results

Curse of dimensionality is known to be a major concern when applying formal methods to control synthesis and robotic applications. In this case study, we assess the effectiveness of the algorithms introduced in Sec. III-V in terms of scalability. We consider again the maze-like workspace with increasing number of passages as in Sec. VI-A, and we investigate the execution time as we increase the number of states n. For each test case, we randomly generate the matrices A and B and average the computation time over 10 runs of the same experiment. Albeit not offering a statistical significant sample, our results are representative of the several simulations performed while testing our solver.

Fig. 4 shows, on a logarithmic scale, the execution time of the proposed motion planner as n increases. In all tests, we report the cumulative time due to both the discrete and continuous planners. The time to compute the discrete abstractions is equal to the one reported in Tab. I. Thanks to the proposed architecture, the dimension of the continuous state space only affects the number of variables of the linear programs in Sec. V. Even for systems with up to 50 state variables, the execution time ranges from 1.64 s (maze with one passage) to 152 s (maze with 4 passages), showing the potential of our approach to be deployed on complex robotic systems.

VII. Conclusions

We presented a scalable algorithm for the reach-avoid robot motion planning problem. We assumed a discrete-time, linear model of the robot dynamics and a description of the workspace in terms of a set of obstacles and a target region that are polyhedra. Our algorithm combines a coarse, obstacle-based discretization of the workspace with a lazy SMT-based approach to efficiently generate a discrete plan that

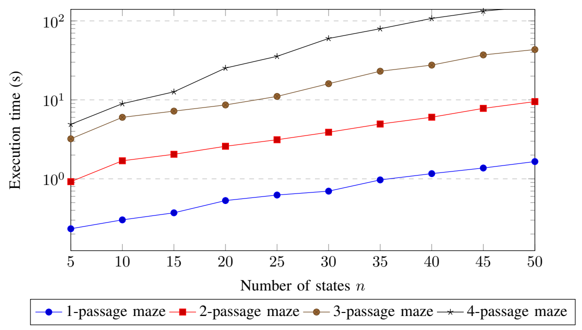

Fig. 4. Execution time of the motion planner for different maze-like workspace configurations as a function of the number of continuous states.

satisfies a set of Boolean constraints together with a continuous state and control trajectory that is physically realizable. The proposed algorithm scales well for high-dimensional systems including robot dynamics with up to 50 continuous states. Moreover, it can be directly extended to motion planning from generic LTL specifications by leveraging the bounded model checking encoding technique for LTL model checking by Biere et al. [10] to encode the discrete planning problem. Future work also includes extending the proposed techniques to multi-robot motion planning and planning in the presence of uncertainties in the dynamics and bounded disturbances.

References

[1] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977, pp. 46–57.

[2] P. Tabuada and G. J. Pappas, "Linear time logic control of discrete-time linear systems," *IEEE Trans. Automatic Control*, vol. 51, no. 12, pp. 1862–1877, 2006.

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

[4] 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, 2009.

[5] 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, 2009.

[6] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Automatic Control*, vol. 57, no. 11, pp. 2817–2830, 2012.

[7] P. Nuzzo, H. Xu, N. Ozay, J. Finn, A. Sangiovanni-Vincentelli, R. Murray, A. Donze, and S. Seshia, "A contract-based methodology for aircraft electric power system design," *IEEE Access*, vol. 2, pp. 1–25, 2014.

[8] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *Int. Conf. Intelligent Robots and Systems*, 2014, pp. 1525–1532.

[9] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI Communications*, no. Preprint, pp. 1–12.

[10] A. Biere, K. Heljanko, T. Junttila, T. Iatvala, and V. Schuppan, "Linear encoding of bounded LTL model checking," *Logical Methods in Computer Science*, vol. 2, no. 5:5, pp. 1–64, 2006.

[11] M. Rungger, M. Mazo Jr, and P. Tabuada, "Specification-guided controller synthesis for linear systems and safe linear-time temporal logic," in *Proc. Int. Conf. Hybrid Systems: Computation and Control*. ACM, 2013, pp. 333–342.

[12] S. Karaman and E. Frazzoli, "Linear temporal logic vehicle routing with applications to multi-UAV mission planning," *Int. J. Robust and Nonlinear Control*, vol. 21, no. 12, pp. 1372–1395, 2011.

[13] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *IEEE Int. Conf. Robotics and Automation*. IEEE, 2014, pp. 5319–5325.

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

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

[16] L. Zhang and D. Manocha, "An efficient retraction-based rrt planner," in *IEEE Int. Conf. Robotics and Automation*. IEEE, 2008, pp. 3743–3750.

[17] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, *Satisfiability Modulo Theories, Chapter in Handbook of Satisfiability*. IOS Press, 2009.

[18] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Automatic deployment of robotic teams," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 75–86, 2011.

[19] K. Margellos and J. Lygeros, "Hamilton–jacobi formulation for reach–avoid differential games," *IEEE Trans. Automatic Control*, vol. 56, no. 8, pp. 1849–1861, 2011.

[20] Z. Zhou, R. Takei, H. Huang, and C. J. Tomlin, "A general, open-loop formulation for reach-avoid games," in *Proc. Conf. Decision and Control*. IEEE, 2012, pp. 6501–6506.

[21] T. A. Henzinger, R. Jhala, and R. Majumdar, *Proc. Int. Colloquium on Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ch. Counterexample-Guided Control, pp. 886–902.

[22] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. Sangiovanni-Vincentelli, "CalCS: SMT solving for non-linear convex constraints," in *Int. Conf. Formal Methods in Computer-Aided Design*, Oct 2010, pp. 71–79.

[23] Y. Shoukry, P. Nuzzo, A. Puggelli, A. L. Sangiovanni-Vincentelli, S. A. Seshia, and P. Tabuada, "Secure State Estimation Under Sensor Attacks: A Satisfiability Modulo Theory Approach," *ArXiv e-prints*, Dec. 2014, [online] http://adsabs.harvard.edu/abs/2014arXiv1412.4324S.

[24] Y. Shoukry, P. Nuzzo, A. Puggelli, A. L. Sangiovanni-Vincentelli, S. A. Seshia, M. Srivastava, and P. Tabuada, "IMHOTEP-SMT: A Satisfiability Modulo Theory Solver for Secure State Estimation," in *Proc. Int. Workshop on Satisfiability Modulo Theories*, July 2015.

[25] Y. Shoukry, P. Nuzzo, N. Bezzo, A. L. Sangiovanni-Vincentelli, S. A. Seshia, and P. Tabuada, "Secure state reconstruction in differentially flat systems under sensor attacks using satisfiability modulo theory solving," in *Conf. Decision and Control*, 2015, pp. 3804–3809.

[26] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas, "Symbolic planning and control of robot motion [grand challenges of robotics]," *IEEE Robotics & Automation Magazine*, vol. 14, no. 1, pp. 61–70, 2007.

[27] D. L. Berre and A. Parrain, "The Sat4j library, release 2.2," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, pp. 59–64, 2010.

[28] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

Appendix

A. Proof of Theorem 5.5

We start by discussing the following intermediate result.

Proposition A.1: Consider the motion planning problem defined in Problem 2.4 based on the formulation in Sec. II. For any set of polyhedra $\rho_1, \dots, \rho_{L+1}$, generated by DIS-PLAN as described in Sec. IV, let the variables $s_0 = s_0^u + s_0^v, \dots, s_L = s_L^u + s_L^v$ be selected such that the following constraints are satisfied:

$$h_{\mathcal{X}\to\mathcal{W}}(x_i) \in \rho_i, \qquad i = 1, \dots, L+1 \qquad \text{(A.1)}$$

$$x_{i+1} = Ax_i + Bu_i + B'v_i \qquad i = 0, \dots, L \qquad \text{(A.2)}$$

$$\|u_i\| \leq \overline{u} + s_i^u, \qquad i = 0, \dots, L \qquad \text{(A.3)}$$

$$\|v_i\| \leq s_i^v, \qquad i = 0, \dots, L \qquad \text{(A.4)}$$

$$0 \leq s_i^u, \quad 0 \leq s_i^v \qquad i = 0, \dots, L \qquad \text{(A.5)}$$

$$\frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{i-1} s_k^u + s_k^v\right) \leq s_i^u + s_i^v \qquad i = 1, \dots, h-1, \qquad \text{(A.6)}$$

where $h_{\mathcal{X}\to\mathcal{W}}$, B', $\epsilon \in \mathbb{R}_{>0}$, and $\overline{s}$ are defined as in Sec. II and Sec. V, and $\overline{s} \geq \epsilon$ holds. Assume $s_j = \overline{s}$ for some $j \in \{0, \dots, L\}$. Then, for all $j' \in \{0, \dots, L\}$, $j' > j$, we obtain:

$$s_{j'} = O_{\overline{s},\epsilon}(j' - j) + \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j-1} s_k\right),$$

where $O_{\overline{s},\epsilon}(j' - j)$ is a constant that depends only on $j' - j$.

Proof: For all $j' > j$ we obtain

$$s_j = \overline{s} \Rightarrow \frac{1}{\epsilon}\left(\sum_{k=0}^{j'-1} s_k\right) \overset{(a)}{\geq} \frac{1}{\epsilon} s_j = \frac{1}{\epsilon}\overline{s} \overset{(b)}{\geq} 1, \tag{A.7}$$

where (a) follows from the fact that all the slack variables are positive and (b) follows from the assumption that $\overline{s} \geq \epsilon$. By definition of $\overline{s}$ as the least upper bound over the optimal slack variables satisfying constraints (A.1)-(A.5), we also obtain $s_i \leq \overline{s}$ for all i in the absence of (A.6). By combining this fact with (A.7), we then conclude

$$s_j = \overline{s} \Rightarrow s_{j'} \leq \overline{s} \leq \frac{1}{\epsilon}\left(\sum_{k=0}^{j'-1} s_k\right)\overline{s} \qquad \forall j' > j. \tag{A.8}$$

We now observe that the upper bound to $s_{j'}$ in (A.8) is equal to the lower bound given by (A.6). Therefore, we can finally write

$$\begin{aligned} s_j = \overline{s} \Rightarrow s_{j'} &= \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j'-1} s_k\right) = \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j-1} s_k\right) + \frac{\overline{s}}{\epsilon}\left(\sum_{k=j}^{j'-1} s_k\right) \\ &\overset{(a)}{=} \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j-1} s_k\right) + \left(\frac{\overline{s}}{\epsilon}\right)^{(j'-j)} + \sum_{k=0}^{j'-j-1} k\left(\frac{\overline{s}}{\epsilon}\right)^k \\ &\overset{(b)}{=} \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j-1} s_k\right) + O_{\overline{s},\epsilon}(j'-j) \end{aligned}$$

where (a) follows after expanding $\frac{\overline{s}}{\epsilon}\left(\sum_{k=j}^{j'-1} s_k\right)$ and considering that $s_j = \overline{s}$ and (b) follows after defining $O_{\overline{s},\epsilon}(j'-j) = \left(\frac{\overline{s}}{\epsilon}\right)^{(j'-j)} + \sum_{k=0}^{j'-j-1} k\left(\frac{\overline{s}}{\epsilon}\right)^k$. ■

We are now ready to prove Theorem 5.5.

Proof of Theorem 5.5: Consider the following optimization problem, in the context of the formulation in Sec. V:

Problem A.2:

$$\max_{\substack{u_0,\ldots,u_L \in \mathbb{R}^m \\ s_0^u,\ldots,s_L^u \in \mathbb{R} \\ s_0^v,\ldots,s_L^v \in \mathbb{R} \\ x_1,\ldots,x_{L+1} \in \mathbb{R}^n \\ v_0,\ldots,v_L \in \mathbb{R}^m}} \textsc{zeroPrefix}_\epsilon((s_0^u + s_0^v),\ldots,(s_L^u + s_L^v))$$

subject to

$$\begin{aligned} & x_0 = \overline{x}, && \\ & h_{\mathcal{X}\to\mathcal{W}}(x_i) \in \rho_i, && i = 1,\ldots,L+1 \\ & x_{i+1} = Ax_i + Bu_i + B'v_i && i = 0,\ldots,L \\ & \|u_i\| \leq \overline{u} + s_i^u, && i = 0,\ldots,L \\ & \|v_i\| \leq s_i^v, && i = 0,\ldots,L \\ & 0 \leq s_i^u, \quad 0 \leq s_i^v && i = 0,\ldots,L \\ & \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{i-1} s_k^u + s_k^v\right) \leq s_i^u + s_i^v && i = 1,\ldots,L \end{aligned} \tag{A.9}$$

Problem A.2 is a constrained version of Problem 5.2, due to the introduction of the constraints (A.9). However, the added constraints are inactive at optimum[2]. Therefore, any solution of Problem A.2 is also a solution of Problem 5.2.

[2]At the optimum k^* for Problem 5.2, the left hand side of the constraints (A.9) evaluates to zero for all $i = 0,\ldots,k^*$, which makes these constraints equivalent to the non-negativity constraints on the slack variables.

To prove our final result, we then need to show that a solution of Problem 5.4 is also a solution of Problem A.2. We proceed by contradiction. Let

$$s^* = (s_0^*,\ldots,s_L^*), \quad s_i^* = s_i^{u*} + s_i^{v*} \quad i = 0,\ldots,L$$

be the solution of Problem 5.4. We then assume that s^* is not a solution of Problem A.2, i.e., there exists $s = (s_0,\ldots,s_L)$ such that:

$$\begin{aligned} \textsc{zeroPrefix}_\epsilon(s_0^*,\ldots,s_L^*) &< \textsc{zeroPrefix}_\epsilon(s_0,\ldots,s_L) \\ &\leq \textsc{zeroPrefix}_\epsilon(s_0,\ldots,s_L) + 1. \end{aligned} \tag{A.10}$$

Given $j = \textsc{zeroPrefix}_\epsilon(s^*)$, by definition of $\textsc{zeroPrefix}_\epsilon$ and by (A.10), we obtain:

$$\sum_{i=0}^{j} s_i^* > \epsilon, \qquad \sum_{i=0}^{j} s_i \leq \epsilon, \tag{A.11}$$

hence:

$$\begin{aligned} \sum_{i=0}^{L} s_i^* &= \sum_{i=0}^{j} s_i^* + s_{j+1}^* + \sum_{i=j+2}^{L} s_i^* \overset{(a)}{>} \epsilon + s_{j+1}^* + \sum_{i=j+2}^{L} s_i^* \\ &\overset{(b)}{>} \epsilon + \overline{s} + \sum_{i=j+2}^{L} s_i^* \\ &\overset{(c)}{=} \epsilon + \overline{s} + \sum_{i=j+2}^{L}\left(O_{\overline{s},\epsilon}(i-j-1) + \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j} s_i^*\right)\right) \\ &\overset{(d)}{>} \epsilon + \overline{s} + \sum_{i=j+2}^{L}\left(O_{\overline{s},\epsilon}(i-j-1) + \overline{s}\right) \end{aligned} \tag{A.12}$$

where (a) follows from (A.11), (b) follows from (A.9), which implies that $s_{j+1}^* \geq \frac{\overline{s}}{\epsilon}\left(\sum_{i=0}^{j} s_i^*\right) > \frac{\overline{s}}{\epsilon}\cdot\epsilon = \overline{s}$, (c) follows from Proposition A.1 and the assumption that $\overline{s} \geq \epsilon$, and (d) follows from (A.11). On the other hand, we also obtain

$$\begin{aligned} \sum_{i=0}^{L} s_i &= \sum_{i=0}^{j} s_i + s_{j+1} + \sum_{i=j+2}^{L} s_i \overset{(e)}{\leq} \epsilon + s_{j+1} + \sum_{i=j+2}^{L} s_i \\ &\overset{(f)}{\leq} \epsilon + \overline{s} + \sum_{i=j+2}^{L} s_i \\ &\overset{(g)}{=} \epsilon + \overline{s} + \sum_{i=j+2}^{L}\left(O_{\overline{s},\epsilon}(i-j-1) + \frac{\overline{s}}{\epsilon}\left(\sum_{k=0}^{j} s_i\right)\right) \\ &\overset{(h)}{\leq} \epsilon + \overline{s} + \sum_{i=j+2}^{L}\left(O_{\overline{s},\epsilon}(i-j-1) + \overline{s}\right) \end{aligned} \tag{A.13}$$

where (e) follows from (A.11), (f) follows from the definition of $\overline{s}$, the fact that $\sum_{i=0}^{h} s_i \leq \epsilon$, and the bound (A.9), leading to $s_{j+1} \geq \frac{\overline{s}}{\epsilon}\sum_{i=0}^{j} s_i \geq \overline{s}$, (g) follows from Proposition A.1 and the assumption that $\overline{s} \geq \epsilon$, and (e) follows from (A.11).

From both (A.13) and (A.12), we finally conclude

$$\sum_{i=0}^{L} s_i^* > \sum_{i=0}^{L} s_i, \tag{A.14}$$

stating that s^* is not a minimal point for Problem 5.4, which is in contradiction with the initial assumption of s^* being a solution to Problem 5.4. ■