Optimal Mixed Discrete-Continuous Planning for Linear Hybrid Systems

Jingkai Chen
jkchen@csail.mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Brian C. Williams
williams@csail.mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

Chuchu Fan
chuchu@mit.edu
Massachusetts Institute of Technology
Cambridge, MA, USA

ABSTRACT

Planning in hybrid systems with both discrete and continuous control variables is important for dealing with real-world applications such as extra-planetary exploration and multi-vehicle transportation systems. Meanwhile, generating high-quality solutions given certain hybrid planning specifications is crucial to building high-performance hybrid systems. However, since hybrid planning is challenging in general, most methods use greedy search that is guided by various heuristics, which is neither complete nor optimal and often falls into blind search towards an infinite-action plan. In this paper, we present a hybrid automaton planning formalism and propose an optimal approach that encodes this planning problem as a Mixed Integer Linear Program (MILP) by fixing the action number of automaton runs. We also show an extension of our approach for reasoning over temporally concurrent goals. By leveraging an efficient MILP optimizer, our method is able to generate provably optimal solutions for complex mixed discrete-continuous planning problems within a reasonable time. We use several case studies to demonstrate the extraordinary performance of our hybrid planning method and show that it outperforms a state-of-the-art hybrid planner, Scotty, in both efficiency and solution qualities.

CCS CONCEPTS

• **Computing methodologies** → **Robotic planning**.

KEYWORDS

Linear Hybrid Systems, Hybrid Planning, Optimization

ACM Reference Format:
Jingkai Chen, Brian C. Williams, and Chuchu Fan. 2021. Optimal Mixed Discrete-Continuous Planning for Linear Hybrid Systems. In *24th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '21), May 19–21, 2021, Nashville, TN, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3447928.3456654

1 INTRODUCTION

Hybrid systems are a powerful modeling framework to capture both the physical plants and embedded computing devices of complex cyber-physical systems. When planning the desired behaviors of a hybrid system, we have to consider both the discrete actions taken by the computing units and the continuous control inputs for the physical actuators. This poses unique and significant challenges in planning for hybrid systems, as one has to consider the change of dynamics of the continuous flow by the control inputs, the interleaves of continuous flows and the discrete transitions between modes, resets associated with transitions, and concurrently running agents in multi-agent systems.

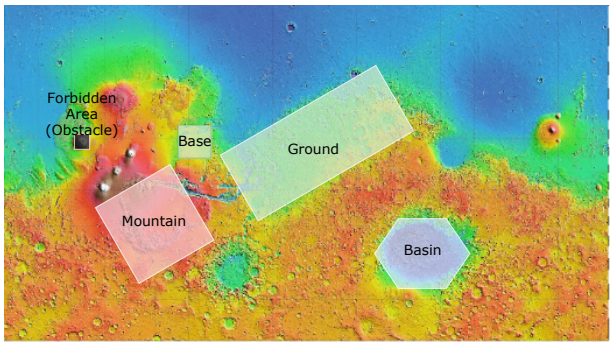


Figure 1: Map of Mars.

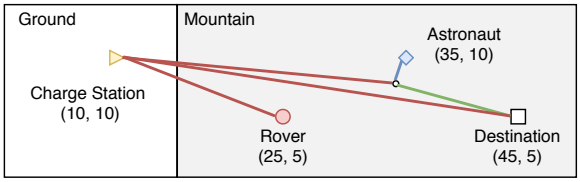


Figure 2: Example of Mars transportation problems: the charge station is marked as ▹, and its position is at (10,10); the rover ○ is at (25,5); the astronaut ⋄ is at (35,10); and the destination □ is at (45,5). In the example solution, the rover path is in red, the astronaut walking path is in blue, and the astronaut taking a ride is in green.

Planning mixed discrete-continuous strategies in hybrid systems is theoretically difficult: on the discrete side, planning with numeric state variables has been proved to be undecidable [26]; on the continuous side, computing the exact unbounded time reachable states for hybrid systems is also a well-known undecidable problem [27]. Nevertheless we are interested in high-quality solutions with the shortest makespan or lowest energy consumption, which is crucial to designing high-performance systems.

Motivating Example. Consider a task where an astronaut should go to an observation location by crossing different terrains (e.g., mountain, ground, and basin) on Mars, and a rover needs to go to the charge station, as shown in Figure 2. The astronaut can either walk or take a rover. The moving speed of the rover is much faster than the walking speed of the astronaut. The rover is powered by a battery. This battery can be charged when the rover is stopped

HSCC '21, May 19–21, 2021, Nashville, TN, USA

ACM ISBN 978-1-4503-8339-4/21/05.
https://doi.org/10.1145/3447928.3456654

in a charge station and should always have the remaining battery outside of the charge station. While the rover and astronaut should not enter the forbidden areas, the rover can move through different terrains with different velocity limits and energy consumption rates. After the rover is manually shut down, it cannot restart within 1 minute. In this mission, plans with shorter makespans are preferred.

A sound solution to this problem is that the astronaut moves directly to the destination, and the rover moves directly to the charge station without picking up the astronaut. However, in this plan, the rover is not used at all. If the rover has enough battery, it can deliver the astronaut to the destination first and then go to the charge station, which saves a lot of time for the astronaut. Unfortunately, energy is always limited in reality. Intuitively, a better solution is that while the astronaut is moving towards the rover, the rover moves to the charge station for charging, then picks up and delivers the astronaut to the destination, and finally returns to the charge station. As the rover moves much faster than the astronaut, this plan requires much less time than letting the astronaut walk to the destination. Another possible faster solution is that the rover picks up and delivers the astronaut somewhere midway to the destination. Then, the astronaut walks to the destination while the rover is moving back to the charge station for a charge.

In this paper, we adopt the hybrid I/O automaton [42] framework to model the mixed discrete-continuous planning problems, which is expressive enough to capture all the mentioned features such as discrete and continuous input and state variables, linear dynamics for continuous flows, and guards and resets for discrete transitions. The crucial technique to make the mixed discrete-continuous optimal planning possible is the introduction of finite-step runs. A step is defined either as a discrete jump or as a set of continuous flows in a mode where we need to compute the dwell time for staying in this mode. We prove that the optimal solution (also called a run) with finite steps will converge to the optimal solution of the original hybrid system when the number of steps goes to infinity. By tailoring the automaton to admit only finite-step runs, we show that finding the optimal input such that the corresponding run has a minimum makespan can be encoded as a Mixed Integer Linear Program (MILP), which can be solved efficiently using off-the-shelf MILP solvers, such as Gurobi [25].

To encode our mixed discrete-continuous planning problem as a MILP, we draw inspiration from the idea of modeling flow tubes [31] as linear programs [14]. We further extend these linear programs to incorporate discrete decisions to model discrete variables and actions. Our complexity analysis shows that the number of MILP variables and constraints at each step increases linearly with the product of the number of linear constraints in each condition and the number of operators and variables. To accelerate MILP solving, we introduce two types of additional constraints about conflicting operators. We also show that our solution approach is able to plan for the temporally concurrent goals known as Qualitative State Plans (QSPs) [30], which describe the desired system behavior over continuous time as tasks with time windows.

To demonstrate the efficiency and solution qualities of our method, we benchmarked against Scotty [14] on three domains: Mars transportation, air refueling, and truck-and-drone delivery. In addition to dealing with different dynamics under a large number of modes, all these three domains require judiciously coordinating heterogeneous agent teams for cooperation and carefully reasoning over resources to decide necessary recharging or refueling. The experimental results show that our approach can find high-quality solutions for all the problems in seconds and provide optimality proof for most examples, while Scotty fails to solve half of the problems within 600 seconds. Moreover, the makespans of our first solutions returned within 1 second are already better than those of Scotty, and our final solutions can significantly improve them.

The remainder of this paper is organized as follows. We start by discussing the related work in both planning and controller synthesis (Section 2). In Section 3, we give the formal definition of our hybrid planning problem as well as a formulation of our motivating example. In Section 4, we introduce a tractable variant of the hybrid planning problem by fixing the action number of automaton runs, which leads to a finite-step linear hybrid planning problem. In Section 5, we present our MILP encoding of this finite-step linear hybrid planning problem. Then, we introduce our extension to deal with temporally concurrent goals in Section 6. Section 7 shows the results of benchmarking our method against the Scotty Planning system on three challenging domains. Finally, concluding remarks are discussed in Section 8.

2 RELATED WORK

Planning [1, 4, 6, 10, 14, 41] and controller synthesis [11, 24, 28, 35, 35, 44, 47, 50–52, 54–56] methods intersect at finding the (optimal) strategies for various system models with respect to different system specifications. In what follows, we briefly mention a couple of representative approaches that are related, without exhaustively listing all approaches in each category.

Discrete abstraction-based synthesis. Discrete abstraction-based synthesis methods first compute a discrete, finite-state abstraction of control systems and then synthesize discrete controllers based on automaton theories or two-player games [24, 35, 35, 44, 50, 55, 56]. Synthesis tools based on abstraction such as CoSyMA [45], Pessoa [48], LTLMop [36, 54], Tulip [15, 56], and SCOTS [49] can support complex nonlinear systems, stochastic uncertainties, or non-deterministic transitions [16, 40, 46, 49, 53, 57], and general GR(1) [57] or Signal Temporal Logic [47] specifications. Our problem may be solved using abstraction-based synthesis using temporal logic specifications. However, none of the above tools can be used directly on our general linear hybrid system with both discrete input/state signals and guards/resets in transitions. Moreover, our approach aims at finding high-quality solutions with low costs or high rewards over long horizons instead of finding all valid solutions. In fact, our planning approach is efficient and effective at finding high-level plans, which is complementary and can be combined with the controller synthesis algorithms for achieving autonomy in complex hybrid systems.

Sampling-based planning. Sampling-based methods such as Probabilistic Roadmaps (PRM) [34], Rapidly-exploring Random Tree (RRT) [37], Fast Marching Tree (FMT) [32], and hybrid automata planner [39] are widely used for searching for plans in nonconvex, high-dimensional, and even partially unknown environments. Researchers also combine the PRM sampling method with classical planning for solving task-and-motion planning problems, which

involves both continuous motions and discrete tasks [21–23, 33, 38]. Compared with the deterministic guarantees provided by controller synthesis methods and our approach, these methods come only with probabilistic guarantees.

Hybrid planning. The planning problems with a subset of the features considered in this paper (i.e., mixed discrete-continuous models, continuous control variables, autonomous transitions, indirect effects, and concurrency) can be of the categories PDDL2.1[17], PDDL-S [14], or PDDL+[18]. Some SMT-based PDDL+ planners such as SMTPlan+ [6] and dReal [5] are complete given a finite number of fixed time steps. However, PDDL+ does not support control variables and these planners solve different problems from ours. As most of the solution approaches to PDDL2.1 and PDDL-S [9, 13, 29] use greedy search methods such as the enforced-hill climbing, they are neither complete nor optimal even with finite steps. Moreover, most of their heuristics belong to the Metric-FF family [29], which are known to suffering from resource persistence or cyclical resource transfer [8] when indirect effects or obstacles are present. Thus, none of these heuristics can handle all of these problem features and often lead the greedy search to be blind in certain domains. These motivate us to develop an effective method that is guaranteed to provide high-quality solutions for mixed discrete-continuous planning problems with various features.

Note that hybrid planning problems are closely related to the formalism of hybrid automata studied in model checking [27, 42], which can be found in [18]. In addition, researchers put many efforts into translating PDDL+ to hybrid automata [2, 3], which leverages the advanced hybrid model checking tools [7, 19, 20] to efficiently prove plan non-existence. Our method directly plans for hybrid automata instead of any PDDL extension. These two representations can be translated to each other since snap actions are basically jumps, and the overall conditions and effects of durative actions are basically flows. By using jumps and flows instead of durative actions, we are able to have a clean MILP encoding for hybrid planning problems.

Among all the hybrid extensions of PDDL, the problems this paper aims at are most relevant to PDDL-S since it is the only planning formalism that supports continuous control variables over time [14]. Kongming [41] is the first planner that is able to solve PDDL-S, and then a more scalable planner Scotty [14] was developed. Scotty is able to efficiently solve complex underwater exploration problems, and it is the current state-of-the-art PDDL-S planner. The reasons for its efficiency are: (1) Scotty encodes the cumulative effect of each control variable as a single variable, which renders a clean convex optimization problem for plan validation, which are called flow tubes [31]; (2) It uses the temporal relaxed planning graph heuristics (i.e., delete relaxation) [9] to guide its greedy search. Our method is inspired by its cumulative effect encoding and extends its optimization problem to handle discrete decisions and nonconvex conditions. By using such an encoding, we do not need to discretize the timeline with a fixed time step as [41] or discretize control parameters as [9]. Meanwhile, our solution approach avoids the incompleteness and suboptimality caused by Scotty's greedy search.

3 PROBLEM FORMULATION

We use a linear hybrid automaton with inputs as the model for our system and then define the hybrid planning problem on this automaton.

Definition 3.1 (Linear Hybrid Automaton). A linear hybrid automaton with inputs is a tuple $\mathcal{H} = \langle V = (Q \cup E), q_{\texttt{Init}}, \texttt{Goal}, J, F \rangle$:

- $Q = L \cup X$ is the set of *internal* variables, which are also called *state* variables. L is the set of discrete state variables, the values of which $\text{val}(L)$ are taken from finite sets called modes. X is the set of continuous state variables, the values of which $\text{val}(X)$ are taken from continuous sets over the reals. We call $\text{val}(L) = \text{val}(L) \times \text{val}(X)$ internal state space.
- E is the set of *external* variables, which are also called *input* variables. External variables could also contain discrete and continuous variables, which are defined analogously to the internal variables.
- $q_{\texttt{Init}} \in \text{val}(L) \times \text{val}(X)$ is an initial state, and `Goal` is a predicate that represents a set of goal states.
- J is the set of *jumps*. A jump $j \in J$ is associated with a *condition cond* and an *effect eff*. The condition *cond* is a *predicate* over V, where a predicate is a computable Boolean-valued function $cond : \text{val}(V) \rightarrow \mathbb{B}$ that maps the values of the variables V to either true or false. The condition is also known as the guard condition or the enabling condition of the jump. An effect $eff : V \rightarrow Q$ specifies how the value of the state variables changes when the jump occurs. It assigns new values to the variables in Q. The variables that are not mentioned in the effect statements are assumed to remain unchanged.
- F is the set of *flows* for the state variables X. $F_k \subseteq F$ is the set of flows for $X_k \subseteq X$, where $\{X_0, X_1, .., X_K\}$ is a set of disjoint continuous variable sets such that $\cup_k X_k = X$. A flow $f \in F_k$ is associated with a differential equation $\dot{X}_k = A_k E + B_k$ and a condition *cond* over V, where A_k, B_k are constant matrices, and *cond* is defined in the same way as in jumps that specifies when a flow f is activated. At each time, multiple flows $f \in F$ can be activated with exactly one flow f_k from each F_k. That is, there will always be a set of flows, which together specify the evolution of the continuous internal variables X as linear differential equations. We call such a set of flows the *flow set* at each time, and it belongs to the power set of F. During the time when a flow is activated, the values of discrete state variables stay the same.

Note that $\text{val}(Q)$ also defines the invariant set of the internal variables, where $\text{val}(X)$ could be nonconvex. Therefore, we can avoid defining the unsafe set separately.

Without loss of generality, we use an integer variable with domain $\{0, 1, .., |\text{val}(v)| - 1\}$ to replace a discrete variable $v \in V$, where $|\text{val}(v)|$ is the number of the elements in $\text{val}(v)$. Thus, we can further assume that all conditions, initial states, and goals are represented as a propositional sentence of liner constraints:

$$\phi ::= \text{true} \mid (GV \geq H) \mid \phi_1 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2, \tag{1}$$

where $G \in \mathbb{R}^{|V|}$ is a $|V|$-vector of real values and $H \in \mathbb{R}$ is a real value. This propositional sentence of linear constraints can represent both convex and nonconvex regions defined by linear inequalities over both integer and continuous variables. Effects can

be also represented by (1) except its linear constraints involve both $V = Q \cup E$, which are the state and input variables before taking the effects, and Q', which is the state variables after taking the effects.

Example 3.2. To formulate our motivating example, we define two discrete internal variables: the astronaut $\mathtt{LA} \in \{0, 1\}$ has modes `Walking(0)` and mode `Riding(1)`; the rover $\mathtt{LR} \in \{0, 1, 2\}$ has `Driving(0)`, `Stopped(1)` and `Charge(2)`.

Internal continuous variables $\mathtt{pA} \in [0, 50] \times [0, 30]$ represent the astronaut's position, and $\mathtt{xR} \in [0, 50] \times [0, 30] \times [0, 30] \times [0, \infty)$ includes the rover's position $\mathtt{pR} \in [0, 50] \times [0, 30]$, battery level $E \in [0, 30]$, and an internal clock $c \in [0, \infty)$. pRx and pRy are the rover's positions over the x-axis and the y-axis, respectively. The initial state is $\mathtt{LA} = 0, \mathtt{LR} = 1, \mathtt{pA} = (35, 10), \mathtt{xR} = (25, 5, 10, 0)$ as shown in Figure 2.

This system also takes commands as input variables, including discrete input variables $\mathtt{cmdA} \in \{0, 1\}$, $\mathtt{cmdR} \in \{0, 1, 2\}$, and continuous input variables $\mathtt{vA} \in [-0.2, 0.2]^2$, $\mathtt{vR} \in [-5, 5]^2$ representing velocities.

Jumps

```
Board   cond (LA=0)∧(cmdA=1)∧(pA=pR)∧(vR=0) eff (LA=1)
Deboard cond (LA=1)∧(cmdA=0)∧(vR=0) eff  (LA=0)
Stop    cond (cmdR=0) eff (LR=1),(c=0)
Drive   cond (LR=1)∧(cmdR=1)∧(c>1) eff (LR=0)
Charge  cond (LR=1)∧(cmdR=2)∧(pR=(10,10)) eff (LR=2)
```

Flows

```
Ride   eq ṗA(t)=vR cond(LA=1)∧(cmdA=1)∧(pA=xA)
Walk   eq ṗA(t)=vA cond (LA=0)∧(cmdA=0)
Ground eq ẋR(t)= [vR,-1,0] cond (LR=0)∧(cmdR=1)∧(pRx<20)
Mount  eq ẋR(t)= [vR,-2,0]  cond (LR=0)∧(cmdR=1)∧(pRx>20)∧(|vR|<2)
Stop   eq ẋR(t)= [0,0,0,1] cond (LR=1)∧(cmdR=0)
Charge eq ẋR(t)= [0,0,0,10] cond (LR=2)∧(cmdR=2)
```

While flow sets can only change continuous state variables, jumps can change both discrete and continuous state variables. The conditions for both jumps and flows would depend on all variables (i.e., including state and input variables). While we call the union of jumps and flows $J \cup F$ as *operators* O, we call both jumps and flow sets $J \cup 2^F$ as *actions* A. We use cond_a to denote the set of states $v \in \mathsf{val}(V)$ such that the condition associated with the action a is true: $\phi(v) = True$.

Given a flow set $a = \cup_k f_k \in 2^F$, we denote the derivative of X as $AE + B$. Such A and B can be easily constructed from the differential equations for each flow $X_k = A_k E + B_k$. If at the beginning of the flow the value of X is x_0, and the elapsed time of such flow is δ, then the X's value would be updated as $x \leftarrow x_0 + A\Delta + B\delta$, where $\Delta = \int_0^\delta E dt$ is the cumulative effects of E during d.

An input signal is a function $e : [0, \infty) \to \mathsf{val}(E)$, which specifies the value of the input variables at any time $t \geq 0$. Once an input signal is fixed, a run of the hybrid automaton is defined as follows:

Definition 3.3. Given a linear hybrid automaton $\mathcal{H} = \langle V = (Q \cup E), q_{\mathtt{Init}}, \mathtt{Goal}, J, F\rangle$ and an input command $e : [0, \infty) \to \mathsf{val}(E)$, a *run* [1] of $\mathcal{H}$ is defined as a sequence of internal states $q_0, \cdots, q_n \in \mathsf{val}(L) \times \mathsf{val}(X)$:

$$\xi_{\mathcal{H},e} = q_0 \xrightarrow{a_0,\delta_0} q_1 \cdots, q_{n-1} \xrightarrow{a_{n-1},\delta_{n-1}} q_n,$$

such that

(1) $q_0 \in q_{\mathtt{Init}}$ and $q_n \in \mathtt{Goal}$.
(2) $a_0, \cdots, a_{n-1}$ are actions. Let $t_i = \sum_{j=0}^{i-1} \delta_j$ be the accumulated time associated with q_i for each $i = 0, \cdots, n$, then: (a) if $a_i \in J$ is a jump, then $\delta_i = 0$, $(q_i, e(t_i)) \in \mathsf{cond}(a_i)$, and $q_{i+1} = \mathsf{eff}(q_i, e(t_i))$; (b) if $a_i \in 2^F$ is a flow set, then $\delta_i \geq 0$, $(q_i, e(t_i)) \in \mathsf{cond}(a_i)$, $x_{i+1} = x_i + A\int_{t_i}^{t_{i+1}} e(\tau)d\tau + B\delta_i$, $\ell_{i+1} = \ell_i$ where $q_i = (\ell_i, x_i)$. Moreover, between the time $t \in [t_i, t_{i+1})$, $(q(t), e(t))$ should always satisfy $\mathsf{cond}(a_i)$.

We also denote the total time $\sum_{i=0}^{n-1} \delta_i$ of a run e as $\xi_{\mathcal{H},e}.\mathtt{TotalTime}$. Note that although e is defined on the infinite time horizon $[0, \infty)$, we do not need to have the value for $e(t)$ when $t > \xi_{\mathcal{H},e}.\mathtt{TotalTime}$

Now given a linear hybrid automaton with inputs, we can define the planning problem as finding an input signal whose run has the minimum makespan.

Definition 3.4. Given a linear hybrid automaton $\mathcal{H} = \langle V = (Q \cup E), q_{\mathtt{Init}}, \mathtt{Goal}, J, F\rangle$, the planning problem is to find a the optimal input e^* signal so the corresponding solution's makespan is minimized:

$$e* = \underset{e}{\operatorname{argmin}}\, \xi_{\mathcal{H},e}.\mathtt{TotalTime}$$

4 FINITE-STEP DECISION PROBLEM

Solving the planning problem (Definition 3.4) to get the optimal input signal e^* needs to reason over all possible $e(t)$. For most hybrid automaton, this is intractable, as the unbounded-time reachability problem is undecidable even for rectangular hybrid automaton [27], which is a simpler hybrid automaton than ours with the right-hand side of the differential equations containing only constants.

Essentially, to solve the optimal e^*, we need to assign values of all input variables for infinitely many t. To make this problem solvable, we fix the number of actions allowed in the run of the hybrid automaton and simplify the original problem by searching for $e(t)$ that corresponds to each action. We introduce a *fixed-step linear hybrid automaton* to capture such an idea.

Definition 4.1. A *finite-step linear hybrid automaton with input* $\mathcal{H}_n$ is a linear hybrid automaton $\mathcal{H}$ (as defined in Definition 3.1) with all runs of $\mathcal{H}$ to have exactly n actions.

In Section 5, we present how to use a MILP encoding to solve the planning problem for fixed-step linear hybrid automaton. Next, we show that for a hybrid automaton $\mathcal{H}$, once we fix the number of actions n and make it $\mathcal{H}_n$, the feasible solution set (the set of input signals e such that $\xi_{\mathcal{H}_n,e}$ is a run of $\mathcal{H}$ with n actions) is non-decreasing as the number of actions n increases.

Lemma 4.2. *Let $\mathcal{H}^n$ be a finite-step linear hybrid automaton for $\mathcal{H}$ with fixed action number n. Let $\Xi_{\mathcal{H}^n}$ and $\mathcal{E}_{\mathcal{H}^n}$ be all the runs of $\mathcal{H}^n$ and their corresponding input signals, respectively. For any $0 < n' < n$, we have $\mathcal{E}_{\mathcal{H}^{n'}} \subseteq \mathcal{E}_{\mathcal{H}^n}$.*

Proof. For any $e \in \mathcal{E}_{\mathcal{H}^n}$, let $q \xrightarrow{a,\delta} q'$ be a segment of its run $\xi_{\mathcal{H},e}$ and $a \in 2^F$ is a flow set. We can replace this segment with $q \xrightarrow{a,\delta} q' \xrightarrow{a,0} q'$, and the new run is still a run of $\mathcal{H}$ given input signal e. As this new run has $n + 1$ actions, we prove that $e \in \mathcal{E}_{\mathcal{H}^{n+1}}$ for any $e \in \mathcal{E}_{\mathcal{H}^n}$. □

[1] We assume that in our run Zeno behaviors are not allowed. That is, we do not allow an infinite number of jumps to occur in a finite time interval.

As the original hybrid automaton $\mathcal{H}$ does not fix the action number, we know $\mathcal{E}_{\mathcal{H}} = \bigvee_{n=0}^{n=\infty} \mathcal{E}_{\mathcal{H}^n} = \lim_{n\to\infty} \mathcal{E}_{\mathcal{H}^n}$, which directly follows from Lemma 4.2.

Let

$$e_n^* = \underset{e\in\mathcal{E}_{\mathcal{H}^n}}{\operatorname{argmin}}\, \xi_{\mathcal{H}_n,e}.\texttt{TotalTime}. \tag{2}$$

As $\mathcal{E}_{\mathcal{H}^{n'}} \subseteq \mathcal{E}_{\mathcal{H}^n}$, it is easy to check that $\xi_{\mathcal{H}_{n'},e_{n'}^*}.\texttt{TotalTime} \geq \xi_{\mathcal{H}_n,e_n^*}.\texttt{TotalTime}$. This gives us the following corollary.

Corollary 4.3. *Following Lemma 4.2, then* $\lim_{n\to\infty} e_n^* = e^*$, *where* e_n^* *is defined as* (2).

5 MIXED INTEGER LINEAR ENCODING

In this section, we describe how to encode a finite-step linear hybrid planning problem as a Mixed Integer Linear Program, in which numbers of variables or constraints at each step increase linearly with the product of the operator number and the number of disjuncts involved in each condition (Section 5.3). We first introduce a method to encode formulas with syntax as in (1) (Section 5.1), and move on to the detailed encoding procedure for the finite-step linear hybrid problem (Section 5.2). Additional constraints about conflicting operators for speeding up MILP solving are discussed in (Section 5.4).

5.1 Encoding Linear Constraint Formulas

Firstly, we introduce the methods to encode constraints with syntax (1) as MILP constraints. We start by encoding a canonicalized form and move on to the general case.

Encoding CNF Linear Constraint Formula. Note that a condition expressed using (1) can be always transformed into a conjunctive normal form (CNF) of linear constraints:

$$\mathsf{cond}(V) \equiv \wedge_r^m \vee_s^{m_r} (\mathsf{cond}_{rs}(V)) \equiv \wedge_r^m \vee_s^{m_r} (G_{rs}V \geq H_{rs}), \tag{3}$$

where $G_{rs} \in \mathbb{R}^{|V|}$ and $H_{rs} \in \mathbb{R}$, m is the number of conjuncts in cond, and m_r is the number of disjuncts in the r^{th} conjunct. For convenience, we also replace $G_{rs}V > H_{rs}$ with $G_{rs}V \geq H_{rs}$ without invalidating the solutions. As disjunctions are present in cond, which result in nonconvex sets in general, we use the Big-M method to handle such disjunctive logic in $\vee_s^{m_r}(G_{rs}V \geq H_{rs})$. We define a m_r-vector of intermediate Boolean variables α_r with domain $\{0,1\}$. While $G_{rs}V \geq H_{rs}$ should hold if $\alpha_{rs} = 1$, we have $\alpha_{rs} = 1$ for at least one α_{rs}. Then, the r^{th} disjunction $\vee_s^{m_r}(G_{rs}V \geq H_{rs})$ is represented as a set of linear constraints:

$$\left(\wedge_s^{m_r} (\alpha_{rs} = 1 \implies G_{rs}V \geq H_{rs})\right) \wedge \left(\sum_s^{m_r} \alpha_{rs} \geq 1\right) \tag{4}$$

Let M be a very large positive number, then each implication is encoded as linear inequalities over both V and indicator variables:

$$G_{rs}V + M(1-\alpha_{rs}) \geq H_{rs} \tag{5}$$

As $\mathsf{cond}(V) = \text{true}$ needs all the conjuncts to hold, we end up with the following constraint:

$$\wedge_r^m \wedge_s^{m_r} \left((G_{rs}V + M(1-\alpha_{rs}) \geq H_{rs}) \wedge \left(\sum_s^{m_r} \alpha_{rs} \geq 1\right)\right). \tag{6}$$

Encoding General Linear Constraint Formula. While some regions are easier to encode by using CNFs, the CNFs of some others take more space. For example, in Figure 3, the blue region is a good candidate to be encoded as CNF $(\wedge_0^3 \phi_{0i}) \wedge (\vee_0^3 \phi_{1i}) \wedge (\vee_0^3 \phi_{3i})$, where ϕ_{ij} is a linear inequality and its direction is given in the figure. However, the green region is more intuitive to be encoded as $(\wedge_0^3 \phi_{2i}) \vee (\wedge_0^3 \phi_{4i})$, which is not CNF.

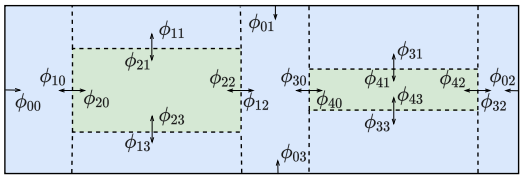


Figure 3: Examples of two regions to encode as linear constraint formulas.

We can use similar methods to encode the general case in (1) in a recursive fashion. Given a set of formulas $\{\phi_0, \phi_1, ..\phi_m\}$, we assume the MILP constraint for ϕ_r is already encoded as $\wedge_s^{m_r}(\mathsf{LHS}_{rs} \geq \mathsf{RHS}_{rs})$, which is a set of linear constraints over V and some indicator variables. We show both the conjunction and disjunction encodings of these formulas. To obtain the MILP constraint of conjunction $\wedge_r^m \phi_r$, we have:

$$\wedge_r^m \wedge_s^{m_r} (\mathsf{LHS}_{rs} \geq \mathsf{RHS}_{rs}). \tag{7}$$

To encode disjunction $\vee_r^m \phi_r$, we introduce an m-vector of Boolean variables α and the disjunction is captured by:

$$\left(\wedge_r^m \wedge_s^{m_r} (\mathsf{LHS}_{rs} + M(1-\alpha_r) \geq \mathsf{RHS}_{rs})\right) \wedge \left(\sum_r \alpha_r^m \geq 1\right). \tag{8}$$

For each linear constraint, if there exist some indicator variable that is not equal to 1, the constraint trivially holds. This set of constraints can be further canonicalized into $\wedge_s^{m_r}(\mathsf{LHS}_{rs} \geq \mathsf{RHS}_{rs})$ over V and indicator variables for other logic operations.

5.2 Encoding Finite-step Hybrid Planning

Now we are ready to encode the entire planning problem as defined in Definition 3.4 for linear hybrid automata with n-step runs.

5.2.1 Variables. To represent the internal states $\{q_0, q_1, .., q_n\}$, we define a set of variables $\{Q_0, Q_1, .., Q_n\}$, and Q_i corresponds to the internal state Q_i right after a_i occurs and right before a_{i+1} occurs. Their domains are copied from Q. To represent the input signal e, we also have $E_i \in \{E_0, E_1, .., E_{n-1}\}$, corresponding to the values of E when a_i occurs, and their domains are copied from E.

To represent the actions that happen at each step, we define a set of binary activation variables $\{P_0, P_1, .., P_{n-1}\}$. P_i is the union of P_i^J and $P_i^F = P_i^{F_0} \cup P_i^{F_1} \cup, .., P_i^{F_K}$, which are the activation variables at step i for jumps J and flows $\{F_1, F_2, .., F_K\}$, respectively. Each $p_i^o \in P_i$ corresponds to an operator o (i.e., a jump or a flow) at step i. If $p_i^o = 1$, operator o is activated at step i; otherwise, o is inactivated. To fully determine the effects of flows, we need to specify the cumulative effects of the input variables and the elapsed time during these flows. Thus, we define d_i with domain $[0, \infty)$ to represent the elapsed time during step i; and real variable Δ denotes $\int_0^{d_i} E_i dt$, the cumulative effects of E_i during step i.

We denote the set of all these MILP variables as $\mathcal{V}$. Let $\Pi : \mathcal{V} \to \mathbb{R}$ be a MILP solution that maps a MILP variable $v \in \mathcal{V}$ to a value $\Pi(v) \in \text{val}(v)$. Then, given a MILP solution Π, we can get the values of the input command e as well as a valid run $\xi_{\mathcal{H},e} = q_0 \xrightarrow{a_0,\delta_0} q_1 \cdots, q_{n-1} \xrightarrow{a_{n-1},\delta_{n-1}} q_n$ as given in Definition 3.3. We extract e over duration $[0, \sum_{i=0}^{(n-1)} \Pi(d_i)]$ as follows:

$$e(t) = \begin{cases} \Pi(E_i), & \text{if } t = (\sum_{j=1}^{i} \Pi(d_j)) \\ \Pi(\Delta_i)/\Pi(d_i), & \text{if } (\sum_{j=0}^{(i-1)} \Pi(d_j)) < t < (\sum_{j=0}^{i} \Pi(d_j)) \end{cases} \tag{9}$$

We extract the run $\xi_{\mathcal{H},e}$ of e from Π as follows:

$$\begin{aligned} q_i &= \Pi(Q_i), \quad \text{for } i \in \{0, 1, .., n\}, \\ \delta_i &= \Pi(d_i), \quad \text{for } i \in \{0, 1, .., n-1\}, \\ a_i = a \text{ if } p \in P_i \text{ and } \Pi(p) = 1, &\quad \text{for } i \in \{0, 1, .., n-1\}. \end{aligned} \tag{10}$$

5.2.2 Objective Function. As specified in Definition 3.3, we aim at finding a run with minimum `TotalTime`, and thus the objective function is $\sum_{i=0}^{n-1} \delta_i$.

5.2.3 Constraints. Next, we introduce the constraints over these variables, which force only one jump or one flow set to be chosen at every time step with their conditions being satisfied and effects being imposed, such that the goal states can be reached from the initial state through these actions. Figure 4 summarizes all constraints C1-C10 to encode the planning problem. We have the following theorem that justify the pair of input command and run given by (9)-(10) is valid:

Theorem 5.1. *Given a n-step hybrid automata $\mathcal{H}$ and a MILP solution Π over the variables $\mathcal{V}$ as defined in Section 5.2.1, the input command e and run $\xi_{\mathcal{H},e}$ that are extracted from Π by* (9)-(10) *are an optimal solution of $\mathcal{H}$ if $\mathcal{V}$ satisfies constraint C1-C10 in Figure 4 and $\sum_{i=0}^{n-1} \delta_i$ is minimized.*

Next, we prove Theorem 5.1 by explaining C1-C10 in detail. As these constraints are the exact translation of Definition 3.3, our solution approach is sound and complete and thus optimal.

Initial and Goal States. First, constraint C1 ensures that runs start from $q_{\texttt{Init}}$, and its final state Q_n satisfies `Goal` such that Definition 3.3(1) is respected. We use (7)-(8) to encode constraint $\texttt{Goal}(Q_N) = \texttt{true}$.

Operator Activation. Constraint C2 can avoid the ambiguity of having multiple jumps or multiple flows for the same internal continuous variables at each step. Recall that $p_i^o = 1$ means the corresponding operator o is activated at step i. Constraint C2 forces either of the following conditions to hold: (1) exactly one jump is active, and all the flows are inactivated (Definition 3.3(2a)); (2) all jumps are inactivated, and there is exactly one flow for each continuous variable set is activated (Definition 3.3(2b)).

Jump Constraint. When a jump is active, its conditions should be satisfied, and their effects should be imposed (Definition 3.3(2a)). For each jump $j \in J$ with condition $cond_j$, when jump j is activate at step i, which is $p_i^j = 1$, condition $cond_j$ should hold, which is captured by C3. Constraint C4 enforces the effect, which is linear constraints $Q_i = \texttt{eff}_j(V_{i-1})$, to happen right after jumps. Note that this constraint also forces the unaffected variables to remain the same after the jumps. In addition, C5 ensures that the elapsed time during jumps is zero, which is activated when some jump is chosen.

Initial and goal states

C1. $(Q_0 = q_{\texttt{Init}}) \wedge (\texttt{Goal}(Q_n) = \texttt{true})$

Operator activation

C2. $\bigwedge_{i=0}^{n-1} \bigwedge_{F_k \subseteq F} (\sum_{p_i^o \in (P_i^J \cup P_i^{F_k})} p_i^o = 1)$

Jump constraint

C3. $\bigwedge_{i=0}^{n-1} \bigwedge_{j \subseteq J} ((p_i^j = 1) \implies \texttt{cond}_j(V_i))$

C4. $\bigwedge_{i=1}^{n} \bigwedge_{j \subseteq J} ((p_i^j = 1) \implies (Q_i = \texttt{eff}_j(V_{i-1})))$

C5. $\bigwedge_{i=0}^{n-1} \bigwedge_{j \in J} ((p_i^j = 1) \implies (d_i = 0))$

Flow constraint

C6. $\bigwedge_{i=0}^{n-1} \bigwedge_{f \subseteq F} (p_i^f = 1) \implies (\bigwedge_r \bigvee_s (cond_{f,rs}^Q(Q_i) \wedge cond_{f,rs}^Q(Q_{i+1})))$

C7. $\bigwedge_{i=0}^{n-1} \bigwedge_{f \subseteq F} (p_i^f = 1) \implies cond_f^E(E_i)$

C8. $\bigwedge_{i=0}^{n-1} \bigwedge_{f \subseteq F} (p_i^f = 1) \implies cond_f^\Delta(\Delta_i, d_i)$

C9. $\bigwedge_{i=0}^{n-1} \bigwedge_{f \subseteq F} ((p_i^f = 1) \implies (X_{k(i+1)} - X_{ki} = A_f \Delta_i - B_f d_i))$

C10. $\bigwedge_{i=0}^{n-1} \bigwedge_{f \in F} ((p_i^F = 1) \implies (L_{i+1} = L_i))$

Figure 4: Constraints in the MILP encoding.

Flow Constraint. While the condition of jumps should only hold right before it happens, the condition of flows should always hold until the next action (Definition 3.3(2b)). Let $cond_f$ be the condition of a flow $f \in F$ and denote the constraints of $cond_f$ over Q and E as $cond_f^Q$ and $cond_f^E$, respectively. Given our solution specification, while the constraint over Q should hold through the execution of f, including the start and end, the constraint over E should also thoroughly hold except at the end of this flow, where the change of E may trigger other jumps or flows.

Since the considered dynamics are linear and all the conditions are sets of disjunctive linear constraints, satisfying $cond_f^Q$ at the start and the end of flow f with respect to the same disjunct in each disjunctive linear constraint implies $cond_f^Q$ holds during f. Constraint C6 captures $cond_f^Q$ at the start and end of flow j being activated. This constraint is sufficient to guarantee the linear trajectory from Q_i ti Q_{i+1} always satisfies $\texttt{cond}_j^Q$, and the reason is as follows: they share the same indicator variables and always satisfy the same disjunct in each disjunction, and thus they are in the same convex region; as the line between two points in a convex region always stay in this region, we know that the trajectory from Q_i to Q_{i+1} satisfy $\texttt{cond}_f^Q$. As a similar encoding for motion planning with polytope obstacles can be seen in [12], our encoding method extends it to handle more general linear constraint formula.

As condition $cond_f^E$ should hold through flow f except at the end, we add constraints over E when f starts and the constraints over Δ, which is the cumulative effects of E during f happening.

While the former constraint is captured in C7, the latter is in C8. For a linear constraint $G^E_{f,rs}E_i \geq H^E_{f,rs}$, we can obtain the equivalent linear constraint over Δ_i and d_i by integrating it over time on both sides and substituting in $\Delta = \int_0^d Edt$:

$$G^E_{f,rs}\Delta_i \leq H^E_{f,rs}d_i. \tag{11}$$

By doing this for each $G^E_{f,rs}E_i \geq H^E_{f,rs}$ in $\mathsf{cond}^E_{f,rs}$, we obtain the condition $\mathsf{cond}^\Delta_{f,rs}$ over (Δ, d).

Then, we determine the evolution of state variables during the flows. Recall that the state variables Q consist of continuous state variables X and discrete state variables L. Let the differential equation of a flow f be $\dot{X}_k = A_f E + B_f$, and then C9 enforces continuous dynamics by adding the effects of Δ and d to X_k. Constraint C10 makes sure that flows do not change discrete variables.

5.3 Complexity Analysis

Now, we discuss the complexity of our MILP encoding. Let n be the number of total steps, K be the total number of disjoint continuous variable sets $\{X_1, X_2, .., X_L\}$, Q be the internal state variables, E be the input variables, J be the jumps, F be the flows, and m and m' be the maximum number of linear inequalities and disjuncts in each condition or effect, respectively. As shown in Section 5.2.1, we know the total number of variables in a MILP is the sum of the following variables: the variables for internal states, input signals, elapsed times, cumulative effects, which are $n(|Q| + 2|E| + 1)$ MILP variables in total; the activation variables for flows and jumps, which is $n(|J| + |F|)$; the additional Boolean variables for indicating activated disjuncts in conditions, which is $2m' + nm'(|J| + 2|F|)$ as shown in Table 1. Thus, we know the number of all the variables is in $\mathcal{O}(n(|V| + m'(|J| + |F|))$, where $\mathcal{O}$ is the asymptotic notation.

Table 1: Numbers of indicator variables in C1-C10.

C1	C2	C3	C4 or C5	C6	C7 or C8	C9 or C10
$2m'$	0	$nm'\|J\|$	0	$nm'\|F\|$	$nm'\|F\|$	0

Table 2: Numbers of linear constraints in C1-C10.

C1	C2	C3 or C4	C5	C6	C7 or C8	C9 or C10
$2m$	nK	$nm\|J\|$	$n\|J\|$	$2nm\|F\|$	$nm\|F\|$	$n\|F\|$

We show the total numbers of constraints in C1-10 in Table 2. The total number of all these constraints is $2m+n(K+(2m+1)|J|+(3m+1)|F|)$. As $K \leq |F|$, we know the number of total constraints is in $\mathcal{O}(nm(|J|+|F|))$. At each time step, the constraint number increase linearly with the product of the number of operators $|J| + |F|$ and the number of disjuncts in a condition m.

5.4 Additional Techniques for Speeding up

To accelerate MILP solving, we add additional constraints to encode conflicting operators that cannot happen together or in sequence. While constraints C3, C4, C6, and C7 already prevent these conflicting operators from happening, additional constraints can help a MILP optimizer to effectively prune state space and thus reduce total runtimes.

One type of additional constraints is about the flows with mutually exclusive conditions. Let $f, f' \in F$ be two flows whose differential equations scope on different continuous variable sets $X_k, X_{k'}$, and cond_f and $\mathsf{cond}_{f'}$ be there conditions. If $\mathsf{cond}_f \wedge \mathsf{cond}_{f'}$ is always false, which means their specified states are totally disjoint, we add constraint $p^f_i + p^{f'}_i \leq 1$ for every $i \in \{0, 1, .., n-1\}$, which ensures at most one of these flows can be activated at every step. The total number of the added constraints is nm_f, where m_f is the number of conflicting flow pairs. Note that these constraints are redundant, which are already encoded by C6, but could help a MILP optimizer to easily identify conflicting operators without further checking the complex constraints in C6.

Another type of additional constraints is about subsequent conflicting operators. Let $o, o' \in J \cup F$ be two operators, and post^Q_o and $\mathsf{pre}^Q_{o'}$ be the possible internal states after taking o, and the possible internal states before o', respectively. We have $\mathsf{pre}^Q_{o'} = \mathsf{cond}^Q_{o'}$ regardless of o' is a jump or a flow, where $\mathsf{cond}^Q_{o'}$ is the condition of o' over internal states Q. On the other hand, post_o can be different given different operator types: if o is a flow, post_o is still cond^Q_o since flow o needs this condition to hold during its happening; if o is a jump, we set $\mathsf{post}_o = \{\mathsf{eff}_o(q,u) \mid \mathsf{cond}^Q_o(q) = \mathsf{true} \text{ and } (q,u) \in V\}$. If $\mathsf{post}_o \wedge \mathsf{pre}_{o'}$ is always false, we know they cannot happen in sequence. Thus, we add constraint $p^o_i + p^{o'}_{i+1} \leq 1$ for every $i \in \{0, 1, .., n-2\}$. The total number of the added constraints is $(n-1)m_o$, where m_o is the number of conflicting operator pairs that cannot happen in sequence.

6 TEMPORALLY CONCURRENT GOALS

In this section, we extend our method to handle a set of temporally concurrent goals by compiling them into our hybrid automata with a certain final goal. There are various types of specifications to represent desired system behaviors over continuous time in both planning and control, such as STL (Signal Temporal Logic) [43], and Qualitative State Plan (QSP) [30]. While the former formalism has a more expressive syntax by using formal logic, QSP is a well-known specification used in planning and is more suitable to the applications we consider in this paper, which specifies a set of tasks to complete as well as the temporal bounds between their starts and ends. We introduce our method to deal with QSP in this section and present related experiments in Section 7.3. We view exploring the methods and applications related to STL as our future work.

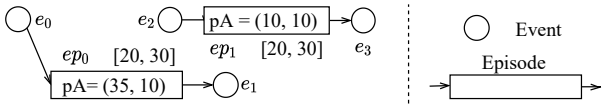


Figure 5: QSP example of four events and two episodes. Episode ep_0 constrain the astronaut to stay at the initial state between 20 and 30 minutes right after the mission begins; Episode ep_1 constrain the astronaut to stay at the charge station between 20 and 30 minutes sometime during the mission.

A *QSP* is a tuple $qsp = \langle EV, EP \rangle$: EV is a set of events, and $e_0 \in EV$ is the initial event that represents the mission begins; EP is a set of episodes. Each episode $ep = \langle e^\vdash e^\dashv, lb, ub, \mathsf{cond} \rangle$ is associated with start and end events $e^\vdash, e^\dashv \in EV$, a duration bound $[lb, ub]$, and a condition cond. For each $e \in EV$, we denote $\{ep \in EP \mid e = ep.e^\vdash\}$ as $\mathtt{starting}(e)$ and $\{ep \in EP \mid e = ep.e^\dashv\}$ as $\mathtt{ending}(e)$. An example of QSP is given in Figure 5.

A schedule $s : EV \to \mathbb{R}^{\geq}$ to qsp is a function that maps $e \in EV$ to a non-negative real value such that (1) $s(e_0) = 0$; and (2) $lb_i \leq s(e_i^{\vdash}) - s(e_i^{\dashv}) \leq ub_i$ for every $ep_i \in EP$. We say a trajectory ξ satisfies qsp if there exists a schedule s such that for every $ep_i \in EP$, $\mathsf{cond}_i(\xi(t)) = \text{true}$ when $(s(e_i^{\vdash}) < t < s(e_i^{\dashv}))$.

Given a hybrid automaton $\mathcal{H} = \langle V = (Q \cup E), q_{\texttt{Init}}, \texttt{Goal}, J, F\rangle$ and a QSP $qsp = \langle EV, EP\rangle$, we compile this QSP into the original automaton as described below, and the runs of the obtained new automaton $\mathcal{H}'$ respect both $\mathcal{H}$ and qsp. We denote this new automaton as $\mathcal{H}' = \langle V' = (Q' \cup E), q'_{\texttt{Init}}, \texttt{Goal}', J', F'\rangle$.

First, we make a clock variable c_{ep} with domain $[-2, \infty)$ for each episode $ep \in EP$. While $c_{ep} = -1$ means ep has not started, $c_{ep} = -2$ means ep has been achieved. When ep is happening, $c_{ep} \geq 0$. Thus, the continuous state variables of $\mathcal{H}'$ is $Q' = Q \cup C$ and $C = \{c_{ep} \in [-2, \infty) \mid ep \in EP\}$. Since all the episodes have not started in the beginning except the episodes started by initial event e_0, the new initial state is $q'_{\texttt{Init}} = q_{\texttt{Init}} \cup \{(c = -1) \mid c \in C/\texttt{starting}(e0)\}\} \cup \{c = 0 \mid ep \in \texttt{starting}(e_0)\}$. As all the episodes should be achieved eventually, the new goal is $\texttt{Goal}' = \texttt{Goal} \cup \{(c = -2) \mid c \in C\}$.

To describe that clock variables reset at events, we add a set of jumps J_{EV}, and $J' = J \cup J_{EV}$. For each event $e \in EV$, there is a jump $j_e \in J_{EV}$ with the following condition: $\{(c_{ep} = -1) \mid ep \in \texttt{starting}(ep)\}$, which ensures event e has not happened before, and $\{(lb(ep) \leq c_{ep} \leq ub(e)) \mid ep \in \texttt{ending}(e)\}$, which shows e should end only when all the ended episodes has lasted for a proper duration with respect to their temporal bounds. The effects $\{(c_{ep} = 0) \mid ep \in \texttt{starting}(e)\}$ and $\{(c_{ep} = -2) \mid ep \in \texttt{ending}(e)\}$ capture the clock variable resets for started episodes and ended episodes, respectively.

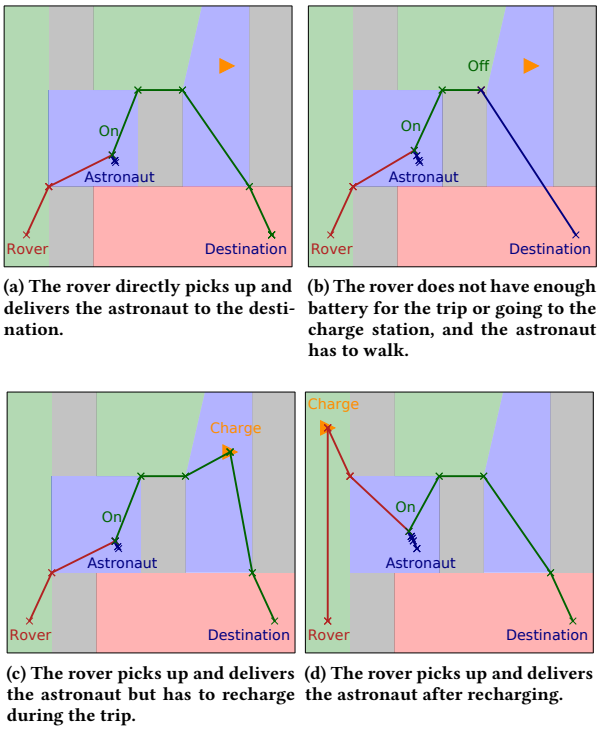
To force the condition is imposed and its clock variable clicks when an episode is happening, we have a flow f^1_{ep} for each clock variable $c \in C$. This flow has differential equation $\dot{c}_{ep} = 1$ and conditions $(c_{ep} \geq 0) \cup \mathsf{cond}_{ep}$. We also have f^0_{ep} with differential equation $\dot{c}_{ep} = 0$ and condition $(c_{ep} \leq -1)$ to represent that episode ep is not happening. Thus, the new flows are $F' = F \cup F_{EP}$ and $F_{EP} = \{f^0_{ep} \mid ep \in EP\} \cup \{f^1_{ep} \mid ep \in EP\}$.

7 EXPERIMENTAL RESULTS

To demonstrate the capabilities of our method, we ran our MILP encoding with Gurobi 9.0.1, which is highly optimized and leverages multiple processor cores, and benchmarked against Scotty [14] on the Mars transportation domains with different initial setups, the air refueling domains with different numbers of UAVs taking photos in different numbers of regions, and the truck-and-drone delivery domains with different numbers of trucks, drones, and packages. All experiments were run on a 3.40GHZ 8-Core Intel Core i7-6700 CPU with 36GB RAM with a runtime limit of 600s. At the end of this section, we also discuss the sizes of these MILP encodings.

7.1 Mars Transportation Domain

The Mars transportation domains involve reasoning over obstacle avoidance and battery consumption under different terrains, such that the astronaut can reach the destination with the help of the rover in the shortest time. A map consists of a set of regions, and each region is a polygon associated with a terrain type. A region can be of the forbidden area, mountain, ground, and basin, which follows the terrains in Figure 1. Driving a rover in different terrains has different velocity limits and energy consumption rates: driving in the mountains should be limited to 10km/h, and the battery consumption rate is 3unit per hour; the velocity limit and the consumption rate in the basin is 30km/h and 2unit/h, and those are 50km/h and 2unit/h for the ground. Walking in these three terrains is 2km/h. On the map, while we fix the initial locations and destinations for astronauts and rovers, we vary the locations of charge stations and the initial battery levels in the four different examples in Figure 6.



(a) The rover directly picks up and delivers the astronaut to the destination.

(b) The rover does not have enough battery for the trip or going to the charge station, and the astronaut has to walk.

(c) The rover picks up and delivers the astronaut but has to recharge during the trip.

(d) The rover picks up and delivers the astronaut after recharging.

Figure 6: Mars transportation examples with different initial battery levels and charge station locations: the charge station is marked as ▹; the forbidden areas, mountain, ground, and basin are in gray, red, green, and blue, respectively; the route of the rover is in red and starts from the bottom left; the route of an astronaut walking is in blue, and its goal destination is at the bottom right; the route of the astronaut taking the rover to the destination is in green.

In Figure 6, the forbidden areas, mountain, ground, and basin are in gray, red, green, and blue, respectively. As we can see, while the rover starts from its initial location at the bottom left and traverses through mountains and basins to arrive at the destination, the astronaut walks towards the rover and joins the ride. It is interesting to notice that the rover chooses the upper route since traversing the lower mountain area costs more time and energy. While the battery is enough for the rover to complete the route in (a), it is insufficient in (b) and (c). In Figure 6(c), the rover carries the astronaut to the charge station and then continues the mission after getting enough battery. In Figure 6(b), the rover battery is too low and even insufficient for the trip to the charge station. Thus, the astronaut gets off and walks from the closest location to the

Table 3: Experimental results of twelve domains. The three numbers after each delivery domain name are the numbers of trucks, drones, and packages, respectively; t: the total runtime in seconds; g: the makespan of the returned solution; t_1: the runtime to find the first solution; g_1: the makespan of the first solution; t_*: the runtime to first find the solution that is finally returned; n: the number of actions; $\#\mathcal{V}_C$, $\#\mathcal{V}_I$, #C: the numbers of continuous variables, integer variables, and constraints in our MILP encoding; $\#\mathcal{V}'_C$, $\#\mathcal{V}'_I$, #C': the numbers of continuous variables, integer variables, and constraints in the presolved MILP models.

Domain	Scotty		MILP Encoding											
	t	g	t	g	t_1	g_1	t_*	n	$\#\mathcal{V}_C$	$\#\mathcal{V}'_C$	$\#\mathcal{V}_I$	$\#\mathcal{V}'_I$	#C	#C'
Mars (a)	<1	35	<1	4.6	<1	35	<1	6	65	54	66	34	1003	527
Mars (b)	<1	35	<1	25.2	<1	35	<1	6	65	54	66	34	1003	527
Mars (c)	<1	35	5.0	5.2	<1	35	<1	9	95	84	99	58	1501	884
Mars (d)	<1	35	3.9	6	<1	35	<1	9	95	84	99	58	1501	915
Air (a)	4	91	<1	43.7	<1	82.7	<1	7	75	58	80	22	897	423
Air (b)	17	163	2.4	55.9	<1	103.9	1.5	12	125	109	185	117	2443	1826
Air (c)	>600	NA	>600	84.3	1.7	177.8	70.9	24	245	232	610	478	11117	9274
Air (d)	>600	NA	>600	40.7	0.6	108.2	154.4	18	278	260	566	450	13692	11181
Delivery (a) (1,1,1)	NA	NA	8.9	36	<1	240	3.3	7	99	91	339	240	6449	3347
Delivery (b) (1,2,2)	NA	NA	13.0	12	<1	120	3.3	7	129	115	402	308	7610	4325
Delivery (c) (2,4,4)	NA	NA	>600	132	<1	780	219.6	11	291	267	1128	999	28538	20804
Delivery (d) (2,4,4)	NA	NA	>600	240	1.9	960	14.5	11	339	319	1172	1029	34130	25583

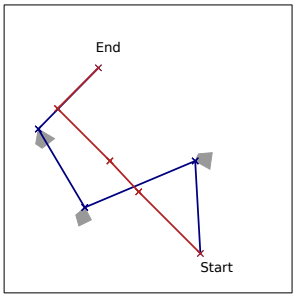
destination after draining the battery. In Figure 6(d), the rover also gets recharged, but it happens before picking up the astronaut due to different charge station locations.

Table 3 shows that our method is able to find the optimal solution and prove its optimality for all these four domains. While Scotty can find a consistent solution $g = 35$ within one second $t < 1$, our method can also find such a solution $g_1 = 35$ within one second $t_1 < 1$. In this solution, the astronaut directly moves to the destination without the help of the rover, which only uses one action but takes a very long time. While Scotty stops after finding this solution, our method keeps searching for better solutions and finds the optimal solution roughly within one second $t_* < 1$. These solutions are then proved to be optimal and returned as Gurobi exhausts the solution space. Thus, our method is able to quickly find a consistent solution and an optimal solution for the Mars transportation domains.

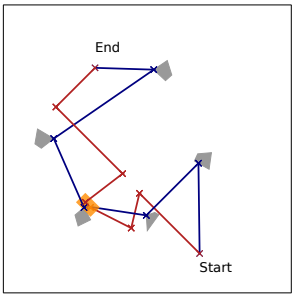
7.2 Air Refueling Domain

In this domain, autonomous Unmanned Aerial Vehicles (UAVs) need to take pictures of several regions before landing at the destination location. Since a UAV has limited fuel, it needs to refuel in-air from a tanker plane. This problem is difficult since it requires reasoning on the optimal ordering of visiting all the regions and also coordinating the UAVs and the tank planes to take necessary refueling. When multiple UAVs are in a mission, we should also effectively dispatch the photo-taking tasks such that the makespan is minimized. The maximum velocity of the tank plane is $20m/s$. While flying, UAVs can fly with the velocity up to $30m/s$, and the fuel decreases at $2\text{unit}/s$. Refueling requires the distances between UAVs and tanks planes to be less than $10m$. When an UAV is refueling, the maximum allowable velocity is $5m/s$, and the fuel increases at $10\text{unit}/s$. While the tank capacity of UAVs is 100 units, we assume the tank plane has enough fuel during missions.

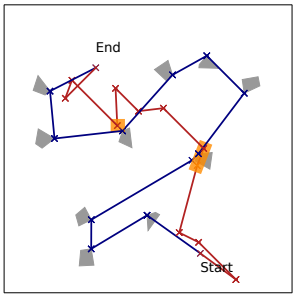
We experimented with this domain on four examples with different numbers of regions and UAVs, as shown in Figure 7. The UAVs and the plane start from the same spot and should arrive at the same destination. While there is only one UAV in the examples (a), (b),



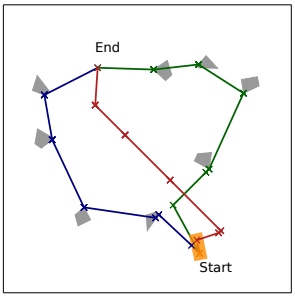
(a) The UAV takes photos for three regions and does not need refueling.



(b) The UAV takes photos for four regions and refuels once along the route.



(c) The UAV takes photos for ten regions and refuels twice along the route.



(d) Two UAVs take photos for eight regions along two different routes. While one UAV does not need refueling, the other one refuels once.

Figure 7: Air refueling examples with different numbers of UAVs and regions to take photos: the regions for taking photos are gray polygons; all the examples consider one UAV (blue) and one tank plane (red) except example (d), which has an additional UAV (green); all the UAVs (blue) are fueled up (i.e., 100 units) in the beginning except the second UAV (green) in domain (d), whose fuel is 10 units. When the plane is refueling, the routes are makred in yellow.

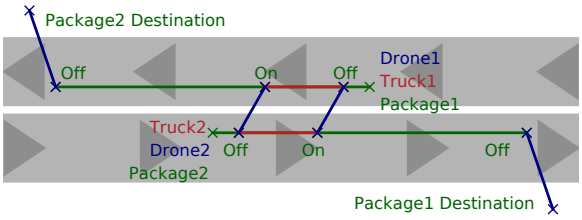


Figure 8: Examples of two trucks and two drones delivering two packages: their initial positions and the package destinations are marked; the truck routes are in red or green and start from the bottom left and the top right, respectively; the routes are in green if the trucks are carrying drones; the routes of drones flying are in blue.

and (d), we add another UAV in example (d). All the examples only have one tank plane. While our method succeeds in finding feasible solutions in two seconds for all the examples, Scotty spends much more time on (a) and (b) and fails to solve the other two examples within 10 minutes, which require more complex coordination on visiting a larger number of regions. It is interesting to note that our first solutions are already better than the Scotty solutions, and the makespans of our final solutions are mostly half of those of Scotty. This is because the delete-relaxation heuristics in Scotty misguided its greedy search when energy resources (i.e., fuel) are in this domain, which prevents Scotty from being effective or efficient in this domain.

7.3 Truck-and-Drone Delivery Domain

In this domain, we consider a fleet of delivery trucks, each equipped with a couple of drones, and the drone and truck both make deliveries to the customers. While trucks can travel between depots through highways or roads, the drone can fly freely in obstacle-free regions or land on trucks to take a ride. When trucks are driving on the road, they should follow the minimum and maximum speed limits as well as the directions, which prevents trucks from violating the traffic rules such as making U-turns on highways. Drones are more flexible, but they are slower, and the travel distance is limited by their battery capacity. In this domain, we look for a plan to deliver all the packages in the shortest time. Figure 8 shows an example of the truck-and-drone delivery domains between two depots, in which the two trucks loaded with packages and drones are driving towards each other on a two-way street. Unfortunately, the package destinations are not on the road ahead, and the trucks cannot turn around. A reasonable plan is that the packages are swapped to the other truck by using the drones to cross the street, and then the truck and drone on the other side continue delivery.

We test on a map with five depots and ten highways between these depots. Each road is straight and around 10km long with a speed limit of 30-60km/h. The drone can fly with a maximum speed of 5km/h. We assume no obstacle for drones in these examples. The experimental results of four truck-and-drone delivery examples are shown in Table 3. While the packages can be delivered at any time in the first three examples, Delivery (d) requires the packages to be delivered within certain time windows specified as a QSP. As we noticed, Scotty does not make progresses to carrying drones to the deport near the drop-off locations and thus fails to solve any of these problems. It can be seen from the t_1 column that our method is able to find the solution very quickly within several seconds. The optimal solutions can also be found in a very short time, t_* for both (a) and (b). In domains (c) and (d), in which we have two trucks, four drones, and four packages, even though it fails to prove the optimality of the incumbent in 10 minutes, their returned solutions largely reduce the makespan of the first returned solutions.

7.4 MILP Model Study

Now, we study the MILP models of the three benchmarked domains. Table 3 shows the numbers of integer variables $\mathcal{V}_I$, continuous variables $\mathcal{V}_C$, and constraints C in our original encoding (Section 5), as well as those (i.e., $\mathcal{V}'_C$, $\mathcal{V}'_I$, C') in the model that has been presolved by Gurobi. Gurobi presolves a MILP model by compiling it into a smaller model with equivalent feasible and optimal solutions as the original. As we can see in Table 3, the presolved models reduce about 20% continuous variables, 20 $\sim$ 40% integer variables, and 30 $\sim$ 50% constraints in most examples. We also observed that presolving takes less than 0.1s in our experiments.

As the Mars domain does not have many discrete state variables or actions, the numbers of its discrete variables and continuous variables are roughly the same. When it comes to the air refueling domain with more than 8 regions to visit or the truck-and-drone delivery domain with a large number of discrete variables to indicate trucks are on a certain highway, we observe the number of discrete variables is 2 $\sim$ 5 times than that of continuous variables. We also note that it is more difficult to find provably optimal solutions in the MILP problems with more integer variables. While Gurobi can find a solution for all the examples in 2s even for Delivery (d), which has 1511 variables and 34130 constraints, the largest problem we can prove optimality within the runtime limit is Delivery (b), which has 537 variables and 7610 constraints.

8 CONCLUSIONS

In this paper, we presented a mixed discrete-continuous planning approach that fixes the action number of the automaton runs and encodes the corresponding finite-step hybrid planning problem as a MILP. Our complexity analysis shows that the number of the MILP variables and constraints at each step increases linearly with the product of the number of linear constraints involved in each condition and the number of operators and variables. By leveraging the state-of-the-art MILP optimizer Gurobi, our method is able to efficiently find provably optimal or high-quality solutions for challenging mixed discrete-continuous planning problems. This was supported by our experimental results against Scotty on the Mars transportation domains, the air refueling domains, and the truck-and-drone delivery domains.

Acknowledgements. The authors acknowledge support from the DARPA Assured Autonomy under contract FA8750-19-C-0089 and the DARPA Creative Problem Solver under contract N16A-T002-0149. The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] Arthur Bit-Monnot, Luca Pulina, and Armando Tacchella. 2019. Cyber-Physical Planning: Deliberation for Hybrid Systems with a Continuous Numeric State. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29. 49–57.

[2] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. 2015. PDDL+ planning with hybrid automata: Foundations of translating must behavior. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

[3] Sergiy Bogomolov, Daniele Magazzeni, Andreas Podelski, and Martin Wehrle. 2014. Planning as model checking in hybrid domains. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.

[4] Daniel Bryce. 2016. A happening-based encoding for nonlinear pddl+ planning. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence*.

[5] Daniel Bryce, Sicun Gao, David J Musliner, and Robert P Goldman. 2015. SMT-Based Nonlinear PDDL+ Planning.. In *AAAI*. 3247–3253.

[6] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. 2016. A compilation of the full PDDL+ language into SMT. (2016).

[7] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. 2000. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer* 2, 4 (2000), 410–425.

[8] Amanda Coles, M Fox, and D Long. 2013. A hybrid LP-RPG heuristic for modelling numeric resource flows in planning. *Journal of Artificial Intelligence Research* 46 (2013), 343–412.

[9] Amanda Jane Coles, Andrew I Coles, Maria Fox, and Derek Long. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44 (2012), 1–96.

[10] Giuseppe Della Penna, Daniele Magazzeni, and Fabio Mercorio. 2012. A universal planning system for hybrid domains. *Applied intelligence* 36, 4 (2012), 932–959.

[11] Chuchu Fan, Umang Mathur, Sayan Mitra, and Mahesh Viswanathan. 2018. Controller synthesis made real: reach-avoid specifications and linear dynamics. In *International Conference on Computer Aided Verification*. Springer, 347–366.

[12] Chuchu Fan, Umang Mathur, Sayan Mitra, and Mahesh Viswanathan. 2018. Controller Synthesis Made Real: Reachavoid Specifications and Linear Dynamics. In *Computer Aided Verification*. Springer International Publishing, 347–366. https://doi.org/10.1007/978-3-319-96145-3_19

[13] Enrique Fernandez-Gonzalez, Erez Karpas, and Brian Williams. 2017. Mixed discrete-continuous planning with convex optimization. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[14] Enrique Fernández-González, Brian Williams, and Erez Karpas. 2018. Scottyactivity: Mixed discrete-continuous planning with convex optimization. *Journal of Artificial Intelligence Research* 62 (2018), 579–664.

[15] Ioannis Filippidis, Sumanth Dathathri, Scott C. Livingston, Necmiye Ozay, and Richard M. Murray. 2016. Control design for hybrid systems with TuLiP: The Temporal Logic Planning toolbox. In *IEEE Conference on Control Applications*. 1030–1041.

[16] Ioannis Filippidis, Sumanth Dathathri, Scott C Livingston, Necmiye Ozay, and Richard M Murray. 2016. Control design for hybrid systems with TuLiP: The temporal logic planning toolbox. In *2016 IEEE Conference on Control Applications (CCA)*. IEEE, 1030–1041.

[17] Maria Fox and Derek Long. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research* 20 (2003), 61–124.

[18] Maria Fox and Derek Long. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27 (2006), 235–297.

[19] Goran Frehse. 2008. PHAVer: algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer* 10, 3 (2008), 263–279.

[20] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 379–395.

[21] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2015. FFRob: An efficient heuristic for task and motion planning. In *Algorithmic Foundations of Robotics XI*. Springer, 179–195.

[22] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2017. Sample-Based Methods for Factored Task and Motion Planning.. In *Robotics: Science and Systems*.

[23] Caelan Reed Garrett, Tomas Lozano-Perez, and Leslie Pack Kaelbling. 2018. FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research* 37, 1 (2018), 104–136.

[24] Antoine Girard. 2012. Controller synthesis for safety and reachability via approximate bisimulation. *Automatica* 48, 5 (2012), 947–953.

[25] Incorporate Gurobi Optimization. 2020. Gurobi optimizer reference manual. *URL http://www. gurobi. com* (2020).

[26] Malte Helmert. 2002. Decidability and Undecidability Results for Planning with Numerical State Variables.. In *AIPS*. 44–53.

[27] Thomas A Henzinger, Peter W Kopke, Anuj Puri, and Pravin Varaiya. 1998. What's decidable about hybrid automata? *Journal of computer and system sciences* 57, 1 (1998), 94–124.

[28] Sylvia L Herbert, Mo Chen, SooJean Han, Somil Bansal, Jaime F Fisac, and Claire J Tomlin. 2017. FaSTrack: A modular framework for fast and guaranteed safe motion planning. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 1517–1522.

[29] Jörg Hoffmann. 2003. The Metric-FF Planning System: Translating"Ignoring Delete Lists"to Numeric State Variables. *Journal of artificial intelligence research* 20 (2003), 291–341.

[30] Jörg Hoffmann and Bernhard Nebel. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14 (2001), 253–302.

[31] Andreas G Hofmann and Brian Charles Williams. 2006. Robust Execution of Temporally Flexible Plans for Bipedal Walking Devices.. In *ICAPS*. 386–389.

[32] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. 2015. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research* 34, 7 (2015), 883–921.

[33] Leslie Pack Kaelbling and Tomás Lozano-Pérez. 2011. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 1470–1477.

[34] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12, 4 (1996), 566–580.

[35] Marius Kloetzer and Calin Belta. 2008. A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications. *IEEE Trans. Automat. Control* 53, 1 (2008), 287–297.

[36] Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. 2009. Temporal Logic based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics* 25, 6 (2009), 1370–1381.

[37] James J Kuffner and Steven M LaValle. 2000. RRT-Connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, Vol. 2. IEEE, 995–1001.

[38] Fabien Lagriffoul, Neil T Dantam, Caelan Garrett, Aliakbar Akbari, Siddharth Srivastava, and Lydia E Kavraki. 2018. Platform-independent benchmarks for task and motion planning. *IEEE Robotics and Automation Letters* 3, 4 (2018), 3765–3772.

[39] Morteza Lahijanian, Lydia E Kavraki, and Moshe Y Vardi. 2014. A sampling-based strategy planner for nondeterministic hybrid systems. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 3005–3012.

[40] Luca Laurenti, Morteza Lahijanian, Alessandro Abate, Luca Cardelli, and Marta Kwiatkowska. 2020. Formal and efficient synthesis for continuous-time linear stochastic hybrid processes. *IEEE Trans. Automat. Control* (2020).

[41] Hui X Li and Brian C Williams. 2008. Generative Planning for Hybrid Systems Based on Flow Tubes.. In *ICAPS*. 206–213.

[42] Nancy Lynch, Roberto Segala, Frits Vaandrager, and Henri B Weinberg. 1995. Hybrid i/o automata. In *International Hybrid Systems Workshop*. Springer, 496–510.

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

[44] Kaushik Mallik, Anne-Kathrin Schmuck, Sadegh Soudjani, and Rupak Majumdar. 2018. Compositional synthesis of finite-state abstractions. *IEEE Trans. Automat. Control* 64, 6 (2018), 2629–2636.

[45] Sebti Mouelhi, Antoine Girard, and Gregor Gössler. 2013. CoSyMA: A Tool for Controller Synthesis Using Multi-scale Abstractions. In *International Conference on Hybrid Systems: Computation and Control*. ACM, 83–88.

[46] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. 2013. Falsification of LTL safety properties in hybrid systems. *International Journal on Software Tools for Technology Transfer* 15, 4 (2013), 305–320.

[47] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: Computation and control*. 239–248.

[48] Pritam Roy, Paulo Tabuada, and Rupak Majumdar. 2011. Pessoa 2.0: A Controller Synthesis Tool for Cyber-physical Systems. In *International Conference on Hybrid Systems: Computation and Control*. ACM, 315–316.

[49] Matthias Rungger and Majid Zamani. 2016. SCOTS: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th international conference on hybrid systems: Computation and control*. 99–104.

[50] Paulo Tabuada. 2009. *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer.

[51] Paulo Tabuada and George J. Pappas. 2006. Linear Time Logic Control of Discrete-Time Linear Systems. *IEEE Trans. Automat. Control* 51, 12 (2006), 1862–1877.

[52] Sean Vaskov, Shreyas Kousik, Hannah Larson, Fan Bu, James Ward, Stewart Worrall, Matthew Johnson-Roberson, and Ram Vasudevan. 2019. Towards provably not-at-fault control of autonomous robots in arbitrary dynamic environments.

arXiv preprint arXiv:1902.02851 (2019).

[53] René Vidal, Shawn Schaffert, Omid Shakernia, John Lygeros, and Shankar Sastry. 2001. Decidable and semi-decidable controller synthesis for classes of discrete time hybrid systems. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No. 01CH37228)*, Vol. 2. IEEE, 1243–1248.

[54] Kai Weng Wong, Cameron Finucane, and Hadas Kress-Gazit. 2013. Provably-correct robot control with LTLMoP, OMPL and ROS. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2073.

[55] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. 2012. Receding Horizon Temporal Logic Planning. *IEEE Trans. Automat. Control* 57, 11 (2012), 2817–2830.

[56] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M. Murray. 2011. TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning. In *International Conference on Hybrid Systems: Computation and Control*. ACM, 313–314.

[57] Tichakorn Wongpiromsarn, Ufuk Topcu, Necmiye Ozay, Huan Xu, and Richard M Murray. 2011. TuLiP: a software toolbox for receding horizon temporal logic planning. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*. 313–314.