Translating Temporal Logic to Controller Specifications

Georgios E. Fainekos, Savvas G. Loizou and George J. Pappas

Abstract— **The problem of designing hybrid controllers in order to satisfy safety or liveness specifications has received much attention in the past decade. Much more recently, there is an increased interest in designing hybrid controllers in order to achieve more sophisticated discrete specifications, such as those expressible in temporal logics. A great challenge is how to compose safety and liveness controllers in order to achieve more complex specifications. Existing approaches are predominantly bottom-up, in the sense that the overall control and composition (or switching) logic requires verification of the integrated closed-loop hybrid system. In this paper, we advocate and develop a top-down approach for this problem by synthesizing controllers which satisfy the specification by construction. Given a flat linear temporal logic specification as an input, we develop an algorithm that translates the temporal logic specification into a hybrid automaton where in each discrete mode we impose controller specifications for the continuous dynamics. In addition to achieving the desired specification by construction, our methodology provides a very natural interface between high level logic design and low level control design.**

I. Introduction

In the last decade, there has been an increased interest in the design of hybrid controllers that satisfy safety and/or reachability properties [1]–[6]. Reachability specifications require that the system reaches within a finite time a predetermined region of the state space, while safety specifications call for trajectories that always remain within a set. Note that these two problems are actually dual. That is, one can pose the safety problem as a reachability problem by requiring that no trajectory would ever enter the unsafe set.

The purpose of this paper is to present the basis for a framework for the synthesis of controllers that satisfy specifications beyond the classical safety and reachability properties. Such specifications characterize sequences of events and set invariants. One way to attack the problem is to utilize regular languages for the description of the specifications and abstractions for the discretization of the continuous plant [7] and, thus, to formulate an event-based framework for the supervisory control of hybrid systems [8]. Here, we propose a fragment of the Linear Temporal Logic (LTL) as a formalism that captures an interesting class of user specifications. We believe that LTL is a natural mathematical language that enables control engineers to automate the controller synthesis problem in a way that minimizes errors due to software or human factors.

The Linear Temporal Logic (LTL) over models with underlying time flow the natural numbers has been proven extremely useful in a wide range of applications. In the context of Discrete Event Systems (DES), it was initially proposed as a verification tool [9]. Recently, LTL has also found applications in the synthesis of (non-reactive) hybrid controllers for linear systems both in continuous [10], [11] and discrete time [12]. In a different approach [13], generators of models for LTL formulas (Büchi automata) have been utilized as supervisors of multi-robot navigation functions.

The common characteristic of all the aforementioned approaches is that at the level of the logic they either take an event-based approach or they assume an explicit discretization of the time. In the case of discrete-time, the limitations are apparent especially when one considers multi-agent systems where the interaction between the agents can take place at any time instant. On the other hand, the event based approach might seem adequate for modelling continuous dynamical systems, but that also has its limitations. For one, it cannot distinguish between events that must hold at a particular time instant and events that must hold for a non-zero time interval. Moreover, when event based semantics are used for planning purposes there is no clean way to state at the logical level that a system should remain for all future times (events) within an invariant set (recall that LTL defines infinite behaviors).

In this paper, we take a different approach which is similar to [14]. We define the semantics of LTL over the real time line $\mathbb{R}_0^+$ in an attempt to model the continuous system behavior. We then identify the fragments of flat LTL for which we can built generators of their models in a modular way and in the form of hybrid automata. The modular construction is inspired by a similar construction for partially-ordered deterministic Büchi automata [15] and it relies on the closure properties of the hybrid automata under union and intersection and the semantics of LTL for the valuation of the atomic propositions.

The hybrid automaton that results from the above procedure has only an abstract description of the continuous system (differential inclusions) in each discrete location. Therefore, even though it is a non-deterministic generator for the models of flat LTL formulas, it cannot act as a generator of deterministic continuous trajectories that would satisfy the same temporal specification. For that, we need to design controllers for the continuous system in each discrete location. Given a formula, we develop an algorithm that derives a set of safety and reachability requirements from each discrete location of the hybrid automaton. Using this set of requirements, we can construct feedback control laws employing a number of design methodologies [1]–[6], not necessarily the same in each control location. The resulting hybrid automaton, which is actually now reduced to a sequential composition of controllers, generates continuous trajectories that satisfy the temporal specification by construction.

Research is partially supported by NSF EHS 0311123, NSF ITR 0121431, and the Army Research Office MURI Grant DAAD 19-02-01-0383.

The authors are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104, USA. E-mail: {fainekos+,sloizou+,pappasg+}@grasp.upenn.edu

II. Problem Formulation

We consider an n-dimensional continuous dynamical system Σ which is described by the differential equation

$$\dot{x}(t) = f(x(t), u(t)) \quad x(t) \in X \subseteq \mathbb{R}^n \quad u(t) \in U \subseteq \mathbb{R}^n$$

where $x(t)$ describes the state of the system at time t and $u(t)$ is the control input. The goal of this paper is to construct a closed-loop hybrid controller that generates control inputs $u(t)$ for system Σ so that the resulting continuous trajectory $x(t)$ for a set of initial conditions $X_0 \subseteq X$ satisfies a formula–specification ϕ in the Linear Temporal Logic (LTL).

For the high level planning problem, we consider the existence of a number of regions of interest to the user in the state space X of the system Σ. Let $\Pi = \{\pi_1, \ldots, \pi_n\}$ be a finite set of symbols that label these regions. The denotation $[\![\cdot]\!] : \Pi \to 2^X$ of each symbol in Π represents a subset of X, i.e. for any $\pi \in \Pi$ it is $[\![\pi]\!] \subseteq X$. We make the assumption that for all $\pi \in \Pi$ the set $[\![\pi]\!]$ is connected.

In order to outline the expediency of LTL as a specification language for control problems, we first give an informal description of the temporal operators. The formal syntax and semantics of LTL are presented in Section IV. LTL formulas are built over a set of atomic propositions, the set Π in our case, using combinations of the traditional $(\vee, \wedge, \neg)$ and temporal $(\Diamond, \Box, \mathcal{U}, \mathcal{R})$ operators. Informally, the specification $\Diamond\phi$, which reads as *eventually* ϕ, indicates that the subformula ϕ will become true in the future. Thus, it formalizes reachability specifications. Safety specifications and system invariants are captured by the *always* operator. For example $\Box\pi$ (or $\Box\neg\pi$) specifies that every trajectory of the system should always remain within (or avoid) the region $[\![\pi]\!]$. The formula $\phi_1 \mathcal{U} \phi_2$ intuitively expresses the property that ϕ_1 is true *until* ϕ_2 becomes true. Finally, the *release* operator $\phi_1 \mathcal{R} \phi_2$ denotes that ϕ_2 always holds, a requirement which is released as soon as ϕ_1 becomes true. More complicated specifications can be composed from the basic temporal and Boolean operators (for examples see [10]–[12]). In order to better explain the different steps in our framework, we consider throughout this paper the following example.

Example 1: Consider the system $\dot{x}(t) = u(t)$ with $x \in \mathbb{R}^2$. Assume that there exist four areas of interest denoted by $\pi_0, \pi_1, \pi_2, \pi_3$ such that $[\![\pi_1]\!], [\![\pi_2]\!], [\![\pi_3]\!] \subseteq [\![\pi_0]\!]$ (see Figure 2). The initial conditions of the system are somewhere in the set π_0. The desired specification for the system given in natural language is: "Stay always in π_0 and visit area π_1, then area π_2 and then go to region π_3 while avoiding π_1."

For such spatio-temporal specifications, in this paper we provide a computational solution to the following problem.

Problem 1 (Temporal Logic Controller Synthesis): Given a dynamical system Σ, a set of initial conditions $X_0 \subseteq X$ and a flat LTL formula ϕ over Π, construct a closed-loop system in the form of a hybrid automaton $\mathcal{H}$ such that the resulting system trajectories $x(t)$ starting at some point $x(0) \in X_0$ satisfy the formula ϕ.

III. Hybrid Automata

A hybrid automaton is a mathematical model that captures systems that exhibit both discrete and continuous dynamics [7], [16]. In brief, a *hybrid automaton* is a tuple

$$\mathcal{H} = (X, V, E, Inv, Flow, Init, Guard, F)$$

where X is the state space of the system Σ, V is the set of control locations, $E \subseteq V \times V$ is the set of control switches, $Inv : V \to 2^X$ assigns an invariant set to each location, $Flow : V \times X \to 2^{\mathbb{R}^n}$ constraints the time derivative of the continuous part of the state, $Init(v) : V \to 2^X$ assigns to each control location a set of initial conditions, $Guard : E \to 2^X$ is the guard condition that enables a control switch e and, finally, $F \subseteq V$ is the set of final locations.

Note that in the hybrid automata that we use in this paper all the reset maps [16] are defined to be the identity function. In the following, we assume that the guards and the location invariants are connected sets. Also, we let $H = V \times X$ to denote the state space of the hybrid automaton $\mathcal{H}$.

Informally, the trajectories of the hybrid automaton $\mathcal{H}$ consist of combinations of *continuous flows* and *discrete transitions*. A trajectory of the automaton $\mathcal{H}$ can only start at a location v iff $x(0) \in Init(v)$. When the system is in control location v, then it evolves under the gradient of a vector field whose value is constraint by the set $Flow(v, \cdot)$ while the state x remains always within the invariant set $Inv(v)$. When the continuous part of the trajectory enters a guard set, for example $x \in Guard(v, v')$, the edge (v, v') becomes enabled and the system *instantaneously* switches to the control location v'. The new state of the system will be (v', x). Due to the way that the guard sets are defined, more then one control switches may become enabled when the continuous trajectory enters a guard set. Thus the hybrid automaton $\mathcal{H}$ exhibits non-deterministic behavior. The other source of non-determinism in the hybrid automaton model is the non-uniquely defined flow conditions.

Formally, the semantics of a hybrid automaton are given in terms of generalized or timed transition systems [16]. A generalized transition system is a tuple $\mathcal{T}_{\mathcal{H}} = (H, H_0, \to)$ where H_0 is the set of initial conditions and $\to$ is a transition relation. Due to space limitations, we refer the reader to [7], [16] for the exact definition of the generalized transition system. For the purposes of this paper, we overload the definition of a *trajectory* η of the transition system $\mathcal{T}_{\mathcal{H}}$ and we define it to be a function $\eta : \mathbb{R}_0^+ \to X$. In other words, we only consider the projection of the actual trajectory of $\mathcal{T}_{\mathcal{H}}$ on the continuous state space X. Note that this definition of the trajectory as a function is meaningful since the reset maps of $\mathcal{H}$ are the identity functions. The set of all trajectories η of $\mathcal{T}_{\mathcal{H}}$ starting from a state in H_0 is the language $\mathcal{L}(\mathcal{T}_{\mathcal{H}})$ of the generalised transition system $\mathcal{T}_{\mathcal{H}}$.

In section V, we will need the notions of union and intersection of hybrid automata. Given two hybrid automata with the same continuous state space, we construct a hybrid automaton that has the same language as the union and intersection of the languages of its components. Let $\mathcal{H}_1 = (X, V_1, E_1, Inv_1, Flow_1, Init_1, Guard_1, F_1)$ and $\mathcal{H}_2 = (X, V_2, E_2, Inv_2, Flow_2, Init_2, Guard_2, F_2)$ be two hybrid automata such that the sets V_1 and V_2 are disjoint (this can always be achieved by renaming). The definition of their *union* is straightforward: $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2 = (X, V_1 \cup V_2, E_1 \cup E_2, Inv_1 \cup Inv_2, Flow_1 \cup Flow_2, Init_1 \cup Init_2, Guard_1 \cup Guard_2, F_1 \cup F_2)$. The definition of their *intersection* $\mathcal{H}_1 \cap \mathcal{H}_2$ is a hybrid automaton $\mathcal{H} = (X, V_1 \times V_2, E, Inv, Flow, Init, Guard, F_1 \times F_2)$ where

- $Inv(v_1, v_2) = Inv_1(v_1) \cap Inv_2(v_2)$
- $Flow(v_1, v_2) = Flow_1(v_1) \cap Flow_2(v_2)$
- $Init(v_1, v_2) = Init_1(v_1) \cap Init_2(v_2)$

- $e = ((v_1, v_1'), (v_2, v_2')) \in E$ iff
  1) $e_1 = (v_1, v_1') \in E_1$ and $v_2 = v_2'$. In this case we set $Guard(e) = Guard_1(e_1) \cap Inv_2(v_2)$.
  2) $v_1 = v_1'$ and $e_2 = (v_2, v_2') \in E_2$. In this case we set $Guard(e) = Inv_1(v_1) \cap Guard_2(e_2)$.
  3) $e_1 = (v_1, v_1') \in E_1$ and $e_2 = (v_2, v_2') \in E_2$. In this case $Guard(e) = Guard_1(e_1) \cap Guard_2(e_2)$.

The following proposition follows directly from the definition of the union and the Proposition 12 in [17].

Proposition 1: Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be hybrid automata. If $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$, then $\mathcal{L}(\mathcal{T}_\mathcal{H}) = \mathcal{L}(\mathcal{T}_{\mathcal{H}_1}) \cup \mathcal{L}(\mathcal{T}_{\mathcal{H}_2})$ and if $\mathcal{H} = \mathcal{H}_1 \cap \mathcal{H}_2$, then $\mathcal{L}(\mathcal{T}_\mathcal{H}) = \mathcal{L}(\mathcal{T}_{\mathcal{H}_1}) \cap \mathcal{L}(\mathcal{T}_{\mathcal{H}_2})$.

IV. Flat Linear Temporal Logic

In this section, we review the syntax of the flat Linear Temporal Logic (LTL) [18] in the Normal Negation Form (NNF) and define its semantics with respect to a generalised transition system $\mathcal{T}$. Recall that in NNF the negation can appear only in front of the atomic propositions. If Π is the finite set of atomic propositions, then we denote by $\overline{\Pi}$ the set of all boolean combinations of elements of Π. The definition of the map $[\![\cdot]\!]$ is extended naturally as follows $[\![\pi_1 \wedge \pi_2]\!] = [\![\pi_1]\!] \cap [\![\pi_2]\!]$ and $[\![\neg\pi]\!] = [\![\pi]\!]^c = X \backslash [\![\pi]\!]$.

In the so-called flat fragment of LTL, the left operant of the until ($\mathcal{U}$) and release ($\mathcal{R}$) temporal operators consists only of Boolean combinations of atomic propositions, i.e. elements of the set $\overline{\Pi}$. Therefore, the until and release temporal operators are syntactically restricted to the form $\overline{\pi}\,\mathcal{U}\phi$ and $\overline{\pi}\,\mathcal{R}\phi$ respectively. For clarity of presentation, we denote the flat versions of the until and release operators by $\underline{\mathcal{U}}$ and $\underline{\mathcal{R}}$. For $\overline{\pi} \in \overline{\Pi}$, the syntax of flat LTL in NNF is

$$\phi \ ::= \ \overline{\pi} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \overline{\pi}\,\underline{\mathcal{U}}\phi \mid \overline{\pi}\,\underline{\mathcal{R}}\phi$$

As usual, the boolean constants $\top$ (true) and $\bot$ (false) are defined as $\top = \overline{\pi} \vee \neg\overline{\pi}$ and $\bot = \neg\top$ respectively. The various additional temporal operators are defined as *eventually* $\Diamond\phi = \top\,\mathcal{U}\phi$, *always* $\Box\phi = \bot\,\mathcal{R}\phi$ and *unless* $\overline{\pi}\,\mathcal{W}\phi = \Box\overline{\pi} \vee \overline{\pi}\,\mathcal{U}\phi$. Note that the syntax does not contain the next time operator since it is meaningless with dense time semantics. Also in the following discussion, we employ the notation LTL$(op_1, \ldots, op_k)$ to denote the fragment of LTL in NNF for which the formulas are build using only the boolean and temporal connectives in the list $op_1, \ldots, op_k$. For example, LTL$(\underline{\mathcal{U}}, \wedge)$ is the fragment of flat LTL that uses only the conjunction and the flat until operator. The fragment LTL$(\underline{\mathcal{U}}, \wedge, \vee)$ captures reachability specifications, while the fragment LTL$(\Box, \wedge, \vee)$ safety properties.

We define the semantics of LTL$(\underline{\mathcal{U}}, \underline{\mathcal{R}}, \wedge, \vee)$ formulas with respect to a generalized transition system $\mathcal{T}$. For $\eta \in \mathcal{L}(\mathcal{T})$ and $t \in \mathbb{R}_0^+$, we define the t right shift of η to be $\eta|_t(s) = \eta(t+s)$. Let $\phi \in$ LTL$(\underline{\mathcal{U}}, \underline{\mathcal{R}}, \wedge, \vee)$, $\eta \in \mathcal{L}(\mathcal{T})$ and $t, s \in \mathbb{R}^+$, then the semantics of the formula ϕ are defined recursively as follows

- $\eta \models \overline{\pi}$ iff $\eta(0) \in [\![\overline{\pi}]\!]$
- $\eta \models \phi_1 \wedge \phi_2$ iff $\eta \models \phi_1$ and $\eta \models \phi_2$
- $\eta \models \phi_1 \vee \phi_2$ iff $\eta \models \phi_1$ or $\eta \models \phi_2$
- $\eta \models \overline{\pi}\,\underline{\mathcal{U}}\phi$ iff there exists $t \geq 0$ such that $\eta|_t \models \phi$ and for all s if $0 \leq s \leq t$ then $\eta(s) \in [\![\overline{\pi}]\!]$
- $\eta \models \overline{\pi}\,\underline{\mathcal{R}}\phi$ iff for all $t \geq 0$ it is $\eta|_t \models \phi$ or there exists s such that $0 \leq s \leq t$ and $\eta(s) \in [\![\overline{\pi}]\!]$

When $\eta \models \phi$, we say that the trajectory η is a model of ϕ or that η satisfies the specification ϕ (we write $\eta \not\models \phi$ otherwise). Also, we say that an LTL formula ϕ is satisfiable if there exists a trajectory η such that $\eta \models \phi$. If for all $\eta \in \mathcal{L}(\mathcal{T})$ it is $\eta \models \phi$, then we say that $\mathcal{T}$ satisfies ϕ or that ϕ is true in $\mathcal{T}$. We should point out that when we adopt the normal negation form of LTL we do not loose in expressive power. The loss in the expressive power is due to the "flatness" [18] and the non-standard semantics (closed right bound) of the until operator.

Example 2: Going back to Example (1), we can now formally write the specification using a flat LTL formula:

$$\phi = \Box\pi_0 \wedge \Diamond(\pi_1 \wedge \Diamond(\pi_2 \wedge (\neg\pi_1)\,\mathcal{U}\pi_3)) \qquad (1)$$

V. From Flat LTL to Hybrid Automata

In this section, we describe an algorithmic procedure for deriving a hybrid automaton whose transition system generates the models of certain fragments of LTL. The construction that we develop mimics the modular construction of partially-ordered deterministic Büchi automata that is presented in [15]. First, we derive algorithms for the LTL$(\underline{\mathcal{U}}, \wedge, \vee)$ and LTL$(\Box, \wedge, \vee)$ fragments of the logic. Then, using the properties of closure of the hybrid automata under the operations of union and intersection, we give a solution for any formula ϕ generated using the grammar

$$\phi \ ::= \ \overline{\pi} \mid \phi_\Box \mid \phi \wedge \phi \mid \phi \vee \phi \mid \overline{\pi}\,\underline{\mathcal{U}}\phi$$

where $\phi_\Box \in$ LTL$(\Box, \wedge, \vee)$. We denote the aforementioned fragment of LTL by LTL$^{\underline{\mathcal{U}}}_{\Box}$.

A. The LTL$(\underline{\mathcal{U}}, \wedge, \vee)$ Fragment

The fragment LTL$(\underline{\mathcal{U}}, \wedge, \vee)$ contains the boolean operators of conjunction and disjunction and the flat until temporal operator. For example, the subformula $\Diamond(\pi_1 \wedge \Diamond(\pi_2 \wedge (\neg\pi_1)\,\mathcal{U}\pi_3))$ in Example 2 belongs to this fragment. For notational convenience, we denote the hybrid automaton whose transition system generates models of ϕ by $\mathcal{H}_\phi$ and its transition system by $\mathcal{T}_\phi$ (instead of $\mathcal{T}_{\mathcal{H}_\phi}$). We refer to both of them as generators of the models of ϕ. We proceed to construct such a hybrid automaton in a modular way. The following propositions are immediate from the definitions.

Proposition 2: $\mathcal{H}_{\overline{\pi}} = (X, \{v\}, \emptyset, X, \mathbb{R}^n, [\![\overline{\pi}]\!], \emptyset, \{v\})$ is the resulting hybrid automaton when $\phi = \overline{\pi} \in \overline{\Pi}$.

The next proposition is the basic building block for constructing a generator for the models of any LTL$(\underline{\mathcal{U}}, \wedge, \vee)$ formula.

Proposition 3: Let $\phi = \overline{\pi}\,\underline{\mathcal{U}}\psi$ such that $\psi \in$ LTL$(\underline{\mathcal{U}}, \wedge, \vee)$ and assume that the hybrid automaton $\mathcal{H}_\psi$ = (X, V, E, Inv, $Flow$, $Init$, $Guard$, F) is given, then for $v' \notin V$

$$\mathcal{H}_\phi = (X, V \cup \{v'\}, E', Inv', Flow', Init', Guard', F)$$

where for $V_{in} = \{v \in V \mid Init(v) \cap [\![\overline{\pi}]\!] \neq \emptyset\}$ we have

- $E' = E \cup \{(v', v) \mid v \in V_{in}\}$
- $Inv'(v') = Init'(v') = [\![\overline{\pi}]\!]$ and $Flow(v') = \mathbb{R}^n$
- $Inv'(v) = Inv(v)$ for all $v \in V$
- $Flow'(v) = Flow(v)$ for all $v \in V$
- $Init'(v) = Init(v) \cap [\![\overline{\pi}]\!]$ for all $v \in V$
- $Guard'(v', v) = Init'(v)$ for all $v \in V_{in}$
- $Guard'(e) = Guard(e)$ for all $e \in E$

Algorithm 1 The LTL($\underline{\mathcal{U}}, \wedge, \vee$) Fragment

Input: A formula $\phi \in$ LTL($\underline{\mathcal{U}}, \wedge, \vee$)
Output: The hybrid automaton $\mathcal{H}_\phi$

```
1: procedure LTL_U TOHA($\phi$)
2:     if $\phi = \overline{\pi}$ then
3:         return $\mathcal{H}_{\overline{\pi}}$                          ▷ Proposition 2
4:     else if $\phi = \phi_1 \wedge \phi_2$ then
5:         return LTL_U TOHA($\phi_1$) $\cap$ LTL_U TOHA($\phi_2$)
6:     else if $\phi = \phi_1 \vee \phi_2$ then
7:         return LTL_U TOHA($\phi_1$) $\cup$ LTL_U TOHA($\phi_2$)
8:     else if $\phi = \overline{\pi}\mathcal{U}\psi$ then
9:         $\mathcal{H}_\psi \leftarrow$ LTL_U TOHA($\psi$)
10:        return $\mathcal{H}_\phi$                          ▷ Proposition 3
11:    end if
12: end procedure
```

Note that in the above construction, the time derivative of the continuous state of the hybrid automaton is unconstrained. Therefore, we must impose some fairness requirements in the following sense. If in a control location there exist outgoing edges, then the continuous flow can remain only for a finite time within that location. This is taken care of in Section VI at the level of the atomic controllers.

Informally, Proposition 3 states that if the initial conditions for the execution of the hybrid automaton $\mathcal{H}_\psi$ are satisfied, then the current position must also be in the set $[\![\overline{\pi}]\!]$. The intersection $Init(v) \cap [\![\overline{\pi}]\!]$ in the set $Init'(v)$ is due to the non-standard semantics of the until operator, that is we have allowed a non-strict inequality in the upper bound of the interval of the first operant (invariant condition) of the until. On the other hand, if the execution of $\mathcal{H}_\psi$ cannot start in the current state, then the system must flow under the invariant $[\![\overline{\pi}]\!]$ until such a state is reached.

Algorithm 1 presents the procedure for the synthesis of a hybrid automaton that generates the models of a specification ϕ in LTL($\underline{\mathcal{U}}, \wedge, \vee$). The correctness of the algorithm is immediate from Propositions 1-3. As far as the complexity is concerned, the intersection of the hybrid automata in line 5 as well as the set intersections in Propositions 1 and 3 are the dominating terms. If automaton $\mathcal{H}_{\phi_i}$ has size n_i, then the automaton $\mathcal{H}_1 \cap \mathcal{H}_2$ has size $O(n_1 n_2)$. If we denote by $|\phi|$ the length of the formula ϕ, then the resulting automaton $\mathcal{H}_\phi$ has size $O(\exp(|\phi|))$ [15]. As far as the intersection and the complementation of the sets are concerned, these depend on the representation of the sets and on the dimension of the space.

It is easy to verify that the graph $G = (V, E)$ of the hybrid automaton $\mathcal{H}_\phi$ is acyclic. Notice that initially we mark the locations without outgoing edges as final. After the construction of $\mathcal{H}_\phi$ is completed, several locations v could have $Inv(v) = \emptyset$. Similarly, there could exist edges e for which $Guard(e) = \emptyset$. We must remove such locations and control switches as these characterize non-feasible solutions for our system. Discarding such nodes and edges could convert intermediate control locations, i.e. locations which are not final, to locations without outgoing control switches, we refer to these locations as *deadlock locations*. Deadlock locations and their incoming edges also need to be removed. If at the end of the procedure there do not exist locations marked as final reachable from a location with non-empty initial conditions, then there does not exist a continuous trajectory that would satisfy the specification ϕ. In this case, the language $\mathcal{L}(\mathcal{T}_\phi)$ is empty and we call the respective hybrid automaton $\mathcal{H}_\phi$ as empty. Furthermore, the formula ϕ is unsatisfiable with respect to our model of hybrid automata. To summarize

Proposition 4: Let $\phi \in$ LTL($\underline{\mathcal{U}}, \wedge, \vee$) and let $\mathcal{H}_\phi$ be the output of Algorithm 1, then $\eta \in \mathcal{L}(\mathcal{T}_\phi)$ implies $\eta \models \phi$.

B. The LTL($\Box, \wedge, \vee$) Fragment

First note that due to the syntactic equivalences $\Box\Box\phi \equiv \Box\phi$ and $\Box(\phi_1 \wedge \phi_2) \equiv \Box\phi_1 \wedge \Box\phi_2$ for any formulas $\phi, \phi_1, \phi_2 \in$ LTL, we only need to study the translation of the LTL($\Box, \vee$) fragment of the logic. In more detail, the following can be proven by induction on the depth of the nested always temporal operators.

Proposition 5: Let $\phi \in$ LTL($\Box, \wedge, \vee$), then ϕ can be rewritten in the form $\phi = \bigwedge_{i\in\mathcal{I}} \left(\overline{\pi}_i \vee \bigvee_{j\in\mathcal{J}_i} \Box\psi_j\right)$ where $\mathcal{I}, \mathcal{J}_i \subseteq \mathbb{N}$ and $\psi_j \in$ LTL($\Box, \vee$).

In the following, we give a brief description for the construction of generators for the LTL($\Box, \vee$) fragment.

Proposition 6: $\mathcal{H}_{\Box\overline{\pi}} = (X, \{v\}, \emptyset, [\![\overline{\pi}]\!], \mathbb{R}^n, [\![\overline{\pi}]\!], \emptyset, \{v\})$ is the resulting hybrid automaton when $\phi = \Box\overline{\pi}$.

Proposition 7: Let $\phi = \Box(\overline{\pi} \vee \psi)$ with $\psi \in$ LTL($\Box, \vee$) and assume that the hybrid automaton $\mathcal{H}_\psi$ = (X, V, E, Inv, $Flow$, $Init$, $Guard$, F) is given, then for $v' \notin V$

$$\mathcal{H}_\phi = (X, V\cup\{v'\}, E', Inv', Flow', Init', Guard', F\cup\{v'\})$$

where for $V_{in} = \{v \in V \mid Init(v) \cap [\![\overline{\pi}]\!] \neq \emptyset\}$ we have

- $E' = E \cup \{(v', v) \mid v \in V_{in}\}$
- $Inv'(v') = Init'(v') = [\![\overline{\pi}]\!]$ and $Flow'(v') = \mathbb{R}^n$
- $Inv'(v) = Inv(v)$, $Flow'(v) = Flow(v)$ and $Init'(v)$ $= Init(v)$ for all $v \in V$
- $Guard'(v', v) = Init(v) \cap [\![\overline{\pi}]\!]$ for all $v \in V_{in}$
- $Guard'(e) = Guard(e)$ for all $e \in E$

Let us give the intuition behind the above construction. Note that $\psi = \Box\psi_1 \vee \ldots \vee \Box\psi_n$ since $\overline{\pi}$ includes all the possible boolean combinations of the atomic propositions at the top level. Also, we point out the equivalence $\Box(\Box\phi_1 \vee \Box\phi_2) \equiv \Box\phi_1 \vee \Box\phi_2$. The semantics of $\eta \models \Box(\overline{\pi} \vee \psi)$ are $(\forall t \geq 0).(\eta(t) \in [\![\overline{\pi}]\!] \vee \eta|_t \models \psi)$. Therefore, there exist three nondeterministic choices

1) $\overline{\pi}$ holds now and for all future times (the system remains in the control location v')
2) $\overline{\pi}$ holds for a finite time interval of the form $[t, s)$ and then ψ holds for all future times $s' \geq s$ (the system jumps from location v' to some location v of $\mathcal{H}_\psi$)
3) ψ holds for all future times (the initial conditions of $\mathcal{H}_\psi$ must be satisfied in this case)

Note that in case 2, the semantics of the logic do not enforce continuity conditions, but the semantics of the hybrid automaton do (identity reset maps). Therefore we modify accordingly the $Guard'$ constraints. Moreover, for this fragment of the logic we do not need fairness conditions. A continuous flow can remain in the same control location for an infinite duration.

Proposition 8: Let $\phi \in$ LTL($\Box, \wedge, \vee$) and let $\mathcal{H}_\phi$ be the resulting hybrid automaton, then $\eta \in \mathcal{L}(\mathcal{T}_\phi)$ implies $\eta \models \phi$.

Due to space limitations we do not present the algorithm for this fragment of the logic, but it is similar to

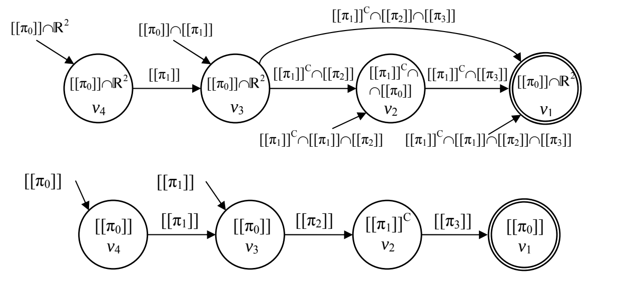

Fig. 1. The hybrid automaton $\mathcal{H}_\phi$ of Example 3. The set $Inv(v_i)$ is displayed inside each circle. The set $Guard(v_i, v_{i-1})$ is displayed on top of each edge (v_i, v_{i-1}). The floating arrows point to control locations with non-empty initial conditions. The double circle denotes the final location. *Above:* Before checking emptiness. *Below:* The finalized hybrid automaton.

Algorithm 1. A similar construction can also be developed for the LTL($\underline{\mathcal{R}}, \vee$) fragment, but the LTL($\Box, \wedge, \vee$) and LTL($\underline{\mathcal{U}}, \wedge, \vee$) seem more suitable for control applications.

C. The LTL$^{\mathcal{U}}_{\Box}$ Fragment

The LTL$^{\mathcal{U}}_{\Box}$ fragment captures combinations of reachability and safety properties. A typical example of a formula that belongs to this fragment is $\Box(\overline{\pi}_1 \vee \Box\overline{\pi}_2) \wedge \Diamond(\overline{\pi}_3 \wedge \Diamond\overline{\pi}_4 \vee \overline{\pi}_5 \,\mathcal{U} \Box\overline{\pi}_6) \wedge \Diamond\overline{\pi}_7$. From the above discussion, the following proposition is immediate.

Proposition 9: Let $\phi \in$ LTL$^{\mathcal{U}}_{\Box}$. If $\eta \in \mathcal{L}(\mathcal{T}_\phi)$, then $\eta \models \phi$.

Example 3: Fig. 1 displays the hybrid automaton $\mathcal{H}_\phi$ that generates the models of the temporal specification (1).

VI. Deriving the Controller Specifications

The hybrid automaton construction that we presented in the previous section can be used to model the closed-loop behavior of system Σ if a finite set of controllers is suitably designed. There exist two issues to be resolved. First, the specifications on the vector fields in each location are too abstract in the sense we have only specified the desired closed loop behavior. Second, there may exist several control switches out of each location which may have overlapping guard sets or whose guard sets might never be reachable. In this section, we resolve these issues by deriving very specific constraints for each control location and by determinizing the hybrid automaton.

Assume the existence of a set $\mathcal{C}$ of triplets $\{A, B, \Gamma\}$, where B and Γ are connected subsets of X such that $A \subseteq B$ and $\Gamma \subseteq B$. Let $\mathcal{G} : \mathcal{C} \to X^U$ be a mapping from the set of triplets to the set of all possible feedback control laws $g : X \to U$ for the system Σ. For some $c \in \mathcal{C}$ and a $g_c \in \mathcal{G}(c)$, the autonomous system becomes $\dot{x}(t) = f_c(x(t)) = f(x(t), g_c(x(t)))$, which we assume that it has a unique solution. The triplet $c = \{A, B, \Gamma\}$ captures the constraints that the trajectory $x(t)$ must satisfy. The semantics for each triplet $c \in \mathcal{C}$ are as follows

- A describes the *initial conditions* where the control law g_c can be applied, i.e. $x(0) \in A$.
- Γ describes the *final conditions* (*goal set*), i.e. there exists some finite time $t_g \in \mathbb{R}^+$ such that $x(t_g) \in \Gamma$.
- B captures the *invariance* that the control law must satisfy, i.e. for all $0 \leq t \leq t_g$ it is $x(t) \in B$.

By slightly abusing terminology, we call the triplet c as an *atomic controller* (instead of the actual control law g_c).

Consider now that given a formula $\phi \in$ LTL$^{\mathcal{U}}_{\Box}$, we derive the hybrid automaton $\mathcal{H}_\phi$ using the procedure outlined in Section V. There are two possible ways to proceed in order to design the continuous control specifications for each control location and, hence, derive a set of controllers that would generate the desired high-level behavior for the closed-loop system. Given a neighborhood of initial conditions $X_0 \subseteq X$ of the system Σ such that $X_0 \subseteq Init(v)$ for some $v \in V$, we can construct the required controllers on-the-fly by traversing the graph G using Breadth or Depth First Search until we reach a final state (both algorithms' running time is $O(V + E)$). If such a sequence of control locations is $v_0, v_1, \ldots, v_m$ such that $X_0 \subseteq Init(v_0)$ and $v_m \in F$, then it is easy to verify that the atomic controllers have the following constraints for $0 < i < m$

- $c_0 = \{Init(v_0), Inv(v_0), Guard(v_0, v_1)\}$
- $c_i = \{Guard(v_{i-1}, v_i), Inv(v_i), Guard(v_i, v_{i+1})\}$
- $c_m = \{Guard(v_{m-1}, v_m), Inv(v_m), \emptyset\}$

Having defined the constraints c for each atomic controller, we proceed to design each feedback control law f_c as it is outlined in Section VII. Note though that it is not always the case that such a feedback controller exists for a set of constraints c. When such a situation occurs, we have to remove the respective edges from the graph and repeat the procedure. As there exists only a finite number of edges, we know that the process terminates in finite time. As before, if a final control location is not reachable from the location v_0 (after cutting out the unrealizable controller specifications), then there does not exist a feasible solution.

The other approach consists of designing off-line controllers for all the possible initial conditions. We use again the Breadth First Search algorithm, but now we start from each final control location and we built a spanning tree by traversing the edges of the graph backward. The resulting directed tree $G_T = (V_T, E_T)$ has the following structure. Each node $v \in V_T$ can have at most one outgoing edge and it may have several incoming edges. We denote the latter set of edges by E_i, while the former (singleton or empty) set by E_o. Therefore, for each control location $v \in V_T$ we can derive the following atomic controller constraints, if $E_o = \{e_o\}$

- $c = \left\{\bigcup_{e \in E_i} Guard(e) \cup Init(v), Inv(v), Guard(e_o)\right\}$

otherwise, if $E_o = \emptyset$

- $c = \left\{\bigcup_{e \in E_i} Guard(e) \cup Init(v), Inv(v), \emptyset\right\}$

As before, by providing this set of constraints to the control engineer along with the system description Σ, we can potentially create a vector field f_c which satisfies the required constraints (Section VII). If no such feedback control law can be generated, then the respective control switch e_o is removed from the initial graph G and a new tree G_T is generated by repeating the above procedure.

Finally, let $\mathcal{H}_\phi = (X, V, E, Inv, Flow, Init, Guard, F)$ and let $G_T = (V_T, E_T) \subseteq G$ be the path or the spanning tree constructed as described above. If we determinize the flow in each control location $v \in V_T$ by setting $Flow'(v, x) = f_c(x)$ for the corresponding atomic controller c, then the new hybrid automaton $\mathcal{H}'_\phi = X, V_T, E_T, Inv, Flow', Init, Guard, F \cap V_T)$ is the desired closed-loop deterministic system whose trajectories $\eta \in \mathcal{L}(\mathcal{T}'_\phi)$ satisfy the specification ϕ by construction.

Example 4: Using any of the two proposed methods on the hybrid automaton of Example 3, we get: $c_1 = \{[\![\pi_3]\!],$

$[\![\pi_0]\!]$, $\emptyset\}$, $c_2 = \{[\![\pi_2]\!], [\![\pi_1]\!]^c, [\![\pi_3]\!]\}$, $c_3 = \{[\![\pi_1]\!], [\![\pi_0]\!], [\![\pi_2]\!]\}$ and $c_4 = \{[\![\pi_0]\!], [\![\pi_0]\!], [\![\pi_1]\!]\}$.

VII. Designing the Controllers

In this section, we demonstrate an approach for designing a control law g_c given a set of constraints in the form of an atomic controller c. As mentioned in the introduction, there exist several methodologies [1]–[6] that can accomplish this task, but here we will focus on Navigation Functions (NV) [5]. Note though that the translation procedure introduced in this paper is a general framework for producing controller specifications and need not be specifically linked to any of the referred controller design methodologies. Even more importantly, different control design methodologies can be applied to different atomic controllers. Therefore, if a design framework fails, we can try another one.

We illustrate the applicability of the proposed methodology using navigation functions [5]. The workspace consists of the regions of interest shown in Fig. 2. Consider the specification (1) and suppose that the underlying system is a fully actuated robot described by the trivial kinematic model $\dot{x} = u$. Navigation function based controllers provide vector fields that can be constructed based on metrics denoting the distance from the good and the bad sets. We will use the following notation to denote a navigation function controller

$$\mathcal{F}\left[\mu_g, \mu_b\right](x) \triangleq -K\nabla\left[\frac{\mu_g}{\left(\mu_g^k + \mu_b\right)^{1/k}}\right](x)$$

where μ_g and μ_b denote metrics from the good sets and the bad sets respectively. For details on the construction and operation of those controllers the interested reader is referred to [5]. Based on the constraints for each atomic controller provided in Example 4 the following controllers are created:

- $c_1 : u = \mathcal{F}\left[1, |\cdot|_{\partial\pi_0}\right](x)$
- $c_2 : u = \mathcal{F}\left[|\cdot|_q, \left(|\cdot|_{\partial\pi_0}\right)\left(|\cdot|_{\partial\pi_1}\right)\right](x)$ with $q \in [\![\pi_3]\!]$
- $c_3 : u = \mathcal{F}\left[|\cdot|_q, \left(|\cdot|_{\partial\pi_0}\right)\right](x)$ with $q \in [\![\pi_2]\!]$
- $c_4 : u = \mathcal{F}\left[|\cdot|_q, \left(|\cdot|_{\partial\pi_0}\right)\right](x)$ with $q \in [\![\pi_1]\!]$

The trajectories of the system under the controllers specified by the translation of formula ϕ defined in (1) are depicted in Fig. 2. As we can see, the system moved from its initial position x to the region defined by π_1 under the influence of controller c_4, then to the region defined by π_2 under the influence of controller c_3 and, then, to the region defined by π_3 while avoiding region $[\![\pi_1]\!]$ under the influence of controller c_2. The last controller c_1, whose task was to maintain the system in the $[\![\pi_0]\!]$ region, had the net effect of moving the system in the center of the workspace. Note that the last controller is not required to avoid region $[\![\pi_1]\!]$. As we can see, the LTL specification is satisfied by the trajectory.

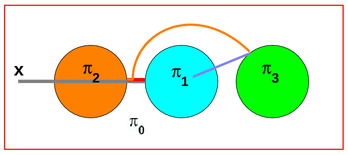

Fig. 2. Trajectory of the system $\dot{x} = u$ with control laws created based on the controller specifications produced by the flat LTL formula (1)

VIII. Conclusions and Future Work

We have presented a new methodology for the controller synthesis problem. Given a flat linear temporal logic specification, we construct a hybrid automaton that generates trajectories that satisfy the specification. The vector fields for each control location can be designed using existing methods from the literature [1]–[6]. This gives us the flexibility to use a variety of methods in control design, even different for each control location of the automaton. The future directions of research are two-fold. First, we are working on an algorithm that can handle the full LTL and, second, we are investigating a distributed version of the presented framework.

References

[1] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli, "Effective synthesis of switching controllers for linear systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1011–1024, 2000.

[2] J. Lygeros, C. Tomlin, , and S. Sastry, "Controllers for reachability specifications for hybrid systems," *Automatica*, vol. 35, no. 3, pp. 349–370, 1999.

[3] L. C. Habets and J. H. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope," *Automatica*, vol. 40, pp. 21–35, 2004.

[4] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[5] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential fields," *IEEE Trans. Robot. Automat.*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[6] D. C. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 4. IEEE, October 2003, pp. 3546– 3551.

[7] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 2, pp. 971–984, 2000.

[8] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon, "Supervisory control of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1026 – 1049, July 2000.

[9] J. G. Thistle and W. M. Wonham, "Control problems in a temporal logic framework," *International Journal of Control*, vol. 44, no. 4, pp. 943–976, 1986.

[10] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec. 2005, pp. 4885 – 4890.

[11] M. Kloetzer and C. Belta, "A fully automated framework for control of linear systems from LTL specifications," in *9th HSCC*, ser. LNCS, vol. 3927. Springer, 2006, pp. 333–347.

[12] P. Tabuada and G. J. Pappas, "Linear time logic control of discrete-time linear systems," accepted for publication in the IEEE Transactions on Automatic Control.

[13] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on LTL specifications," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Dec. 2004.

[14] T. Moor and J. M. Davoren, "Robust controller synthesis for hybrid systems using modal logic," in *Proceedings of the 4th HSCC*, ser. LNCS, vol. 2034. Springer, 2001, pp. 433–446.

[15] R. Alur and S. L. Torre, "Deterministic generators and games for LTL fragments," in *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, 2001, pp. 291–302.

[16] T. A. Henzinger, "The theory of hybrid automata," in *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1996, pp. 278–292.

[17] ——, "Hybrid automata with finite bisimulatioins," in *the 22nd ICALP*, ser. LNCS, vol. 944. Springer, 1995, pp. 324–335.

[18] D. Dams, "Flat fragments of CTL and CTL*: Separating the expressive and distinguishing powers." *Logic Journal of the IGPL*, vol. 7, no. 1, pp. 55–78, 1999.