Labeling finite abstractions of autonomous systems in real-time

Brian Paden, Peng Liu, and Schuyler Cullen

Advanced Technology Group, Samsung Smart Machines
brian.paden@samsung.com, peng.liu@samsung.com

Abstract. Temporal logic provides a

1 Introduction

In the context of autonomous systems and robotics, linear temporal logic provides an expressive language for defining desired properties of an autonomous agent. LTL extends predicate logic with operators that allow constraints to be placed on the ordering of events. The techniques that have been developed for planning motions satisfying specifications given as LTL formulae are well suited to systems requiring guaranteed satisfaction of functional safety requirements and traceability of failures.

The most common approach to generating motion plans satisfying complex task specifications is to construct a finite state, discrete time transition system approximating the continuous state, continuous time physical system with transitions associated to a feasible motion between two configurations. Similarly, relevant features of an environment are abstracted from the scene as sets of logical predicates or atomic propositions labeling the transitions between discrete states of the transition system. A path or trace through the transition system generates a string of predicates which may or may not satisfy the LTL formulae defining the desired behavior of the system. Early work on [1]... This was later [2]. The probabilistic roadmap was the seminal work on the class of sampling based planners. LTL was applied to PRMs in [3].

While the logical precision of formal methods is attractive in task and motion planning applications, in practice, engineers must face the curse of dimensionality when constructing the required finite approx

In many of these techniques, systems with continuous state spaces evolving in continuous time must be approximated by finite state systems

2 Receding Horizon Planning Formulation

The mobility of an autonomous agent is well modeled by a controlled dynamical system with $x(t) \in \mathbb{R}^n$ and $u(t) \in \mathbb{R}^m$ denoting the state and control respectively at time $t \in \mathbb{R}$. Dynamic models derived from first principles typically take the form

$$\frac{d}{dt}x(t) = f(x(t), u(t)), \tag{1}$$

where f is a Lipschitz continuous function from $\mathbb{R}^n \times \mathbb{R}^m$ into $\mathbb{R}^n$. For each Lebesgue integrable input signal $u : [t_0, t_0 + t_h] \to \mathbb{R}^m$, there is a unique trajectory $x : [t_0, t_0 + t_h] \to \mathbb{R}^n$ satisfying (1) through a measured initial state $x(t_0) = x_0$.

Informally, receding horizon planning strategy generates a control signal u over a finite time horizon which meets various specifications on the resulting motion through the present state x_0, and minimized a cost functional.

A finite set of atomic propositions Π, rich enough to provide the task planning specification, are given as part of the problem data. For example, in the context of advanced driver assistance systems, these predicates might include terms such as `DrivableSurface`, `LegalSurface`, `NominalLane`, `DashedWhiteLine`, etc. Each predicate represents a subset of the system's state space, specific to the planning scenario, where that predicate is interpreted as $\top$ at each point in that subset. These regions of the state space are usually defined implicitly by states where a physical object intersects a region of interest. To make this more precise, we will consider the *workspace*, as a region in $\mathbb{R}^k$, and assume that there is a map $F : X \to 2^W$ identifying the space in W occupied by the autonomous agent for each system state in X. Analogously, predicates in the state space are implicitly defined by states of the system x such that $F(x)$ has nonempty intersection with a subset S of W occupying a region associated to a particular predicate.

Each state x can then be labeled with an element of $\{\top, \bot\}^{|\Pi|}$ identifying the interpretation of each predicate at that particular state. Strings of elements from $\{\top, \bot\}^{|\Pi|}$ will be used to form regular and ω-regular languages in which the specifications of the system can be made. The map $\mathcal{L} : \mathbb{R}^n \to \{\top, \bot\}^{|\Pi|}$ defining the interpretation of the predicates at each state is called the *labeling function*. If $\lambda = \mathcal{L}(x)$ and $\lambda_i = \top$, then the i^{th} predicate, with respect to an indexing convention, is considered true at state x. The finite duration $[t_0, t_0 + t_h]$ of a trajectory x over the planning horizon is partitioned into uniform intervals $[t_0, t_1] \cup [t_1, t_2] \cup ... \cup [t_{n-1}, t_n]$, and segments of trajectory along each interval $t \in [t_i, t_{i+1}]$ are labeled with $\lambda \in \{\top, \bot\}^{|\Pi|}$ where $\lambda_j = \top$ if there exists $t \in [t_i, t_{i+1}]$ such that $\mathcal{L}(x(t))_j = \top$. Using this labeling rule, each trajectory over a finite time horizon is associated with a finite string $w = w_0 w_1 w_2 ... w_n$ of subsets of Π corresponding to the predicates intersected in each time interval. This string will be viewed as a prefix to an infinite string or ω-word over the subsets of Π. This is illustrated in Figure ...

Since many task planning problems have moving areas of interest, the state of the system is augmented with the state $\tau \in \mathbb{R}$ which encodes the current time into the state. The evolution of τ is simply,

$$\dot{\tau} = 1, \tag{2}$$

which augments the system dynamics in (1). A time-varying predicate in the original state $\mathbb{R}^n$ becomes a static object in the augmented state space $\mathbb{R}^{n+1}$ and trajectories generate words over $\{\top, \bot\}^{|\Pi|}$ as the state passes through dynamic predicates in the same way as as for static scenarios. This is illustrated in Figure ...

2.1 Linear Temporal Logic as a Specification Language

Linear temporal logic (LTL) has become one of the predominant specification languages for motion and task planning for autonomous systems. The syntax and semantics of LTL are described in this section for reference.

Linear temporal logic (LTL) which consists of the usual logical operators $\neg$ (not) and $\vee$ (or) together with the temporal operators $\mathcal{U}$ (until) and $\bigcirc$ (next). The set of LTL formulae are defined recursively as follows:

1. Each subset of Π is a formula.
2. If ϕ is a formula, then $\neg\phi$ is a formula.
3. If ϕ_1 and ϕ_2 are formulae, then $\phi_1 \vee \phi_2$ is a formulae.
4. If ϕ_1 and ϕ_2 are formulae, then $\phi_1 \mathcal{U} \phi_2$ is a formulae.

Let $w = w_0 w_1 w_2 ...$ be an ω-regular word over 2^{Π}. In the context of motion and task planning, each w_i is a subset of Π representing the atomic propositions which are true at time interval i.

The semantics of LTL define which words w satisfy ϕ, in which case we write the relation $w \models \phi$. A pair (w, ϕ) in the complement of the satisfaction relation is denoted $w \not\models \phi$. The satisfaction relation for LTL is define recursively as follows:

1. For $p \subset \Pi$, $w \models p$ if $p \in w_0$.
2. $w \models \neg\phi$ if $w \not\models \phi$.
3. $w \models \phi_1 \vee \phi_2$ if $w \models \phi_1$ or $w \models \phi_2$.
4. $w \models \bigcirc\phi$ if $w_1 w_2 ... \models \phi$.
5. $w \models \phi_1 \mathcal{U} \phi_2$ if there exists i such that $w_i w_{i+1} ... \models \phi_2$ and for each $j < i$, $w_j w_{j+1} ... \models \phi_1$.

Useful constructs derived from these operators are the following: $\phi_1 \wedge \phi_2 \coloneqq \neg(\neg\phi_1 \vee \neg\phi_2)$, $\phi_1 \Rightarrow \phi_2 \coloneqq \neg\phi_1 \vee \phi_2$, $\phi_1 \Leftrightarrow \phi_2 \coloneqq \phi_1 \Rightarrow \phi_2 \wedge \phi_2 \Rightarrow \phi_1$, $\top \coloneqq \phi \wedge \neg\phi$, $\bot \coloneqq \neg\top$, $\Diamond\phi \coloneqq \top\mathcal{U}\phi$, and $\Box\phi \coloneqq \neg\Diamond\neg\phi$.

2.2 Buchi Automata and Runtime Monitors

The set of ω-regular words that satisfy a formula ϕ forms an ω-regular language, and each ω-regular language can be represented as a Buchi automaton that accepts words satisfying ϕ and rejects them otherwise.

3 Finite State and Discrete Time Approximations

3.1 Synchronous Product with Runtime-monitors

4 Real-time Labeling of Transition Systems

4.1 Partitioning the Workspace

4.2 Indexing the partition

4.3 GPU kernels for binary matrix vector multiplication

5 Empirical Results

References

1. Fainekos, G.E., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for mobile robots. In: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, IEEE (2005) 2020–2025

2. Kress-Gazit, H., Fainekos, G.E., Pappas, G.J.: Temporal-logic-based reactive mission and motion planning. IEEE transactions on robotics **25**(6) (2009) 1370–1381
3. Plaku, E.: Path planning with probabilistic roadmaps and co-safe linear temporal logic. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE (2012) 2269–2275