TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning

Tichakorn Wongpiromsarn*, Ufuk Topcu**, Necmiye Ozay**, Huan Xu**, and Richard M. Murray**
* Singapore-MIT Alliance for Research and Technology, Singapore
** California Institute of Technology, Pasadena, CA
{nok, utopcu, necmiye, mumu, murray}@cds.caltech.edu

ABSTRACT

This paper describes `TuLiP`, a Python-based software toolbox for the synthesis of embedded control software that is provably correct with respect to an expressive subset of linear temporal logic (LTL) specifications. `TuLiP` combines routines for (1) finite state abstraction of control systems, (2) digital design synthesis from LTL specifications, and (3) receding horizon planning. The underlying digital design synthesis routine treats the environment as adversary; hence, the resulting controller is guaranteed to be correct for any admissible environment profile. `TuLiP` applies the receding horizon framework, allowing the synthesis problem to be broken into a set of smaller problems, and consequently alleviating the computational complexity of the synthesis procedure, while preserving the correctness guarantee.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods*; D.2.10 [**Software Engineering**]: Design—*Methodologies*

General Terms

Design, Verification

Keywords

Linear temporal logic, receding horizon control

1. INTRODUCTION

To achieve higher levels of autonomy, modern embedded control systems need to reason about complex, uncertain environments and make decisions that enable complex missions to be accomplished safely and efficiently. To this end, linear temporal logic (LTL) is widely used as a specification language to precisely define system correctness properties. The embedded control software needs to be able to integrate discrete and continuous decision-making and provide correctness guarantee with respect to a given specification. Furthermore, since the environment may be dynamic and unknown a priori, it is important that the controller ensures proper response to all the admissible environment profiles.

A common approach to embedded control software synthesis is to construct a finite transition system that serves as an abstract model of the physical system and synthesize a strategy, represented by a finite state automaton, satisfying the given properties based on this model. Software packages based on this procedure include LTLCon [4], conPAS2 [11], Pessoa [5] and LTLMoP [1]. LTLCon and conPAS2 handle affine systems and piecewise affine systems, respectively, and arbitrary LTL specifications. Pessoa admits nonlinear and switched dynamics but only a very limited class of LTL specifications. However, these three tools do not handle the adversarial nature of the environment. Hence, the controller is only provably correct with respect to an a priori known and fixed environment. In contrast, LTLMoP accounts for adversaries but only considers fully actuated systems operating in the Euclidean plane. To keep the synthesis problem tractable, LTLMoP only admits the GR[1] fragment of LTL. A sampling-based method has been proposed for μ-calculus specifications in [3] but does not provide a correctness guarantee for all the admissible environments.

This paper introduces `TuLiP`, a Python-based toolbox for embedded control software synthesis. Similar to LTLMoP, `TuLiP` models the environment as an adversary and only considers GR[1] formulas. This often leads to the state explosion problem since all the admissible environment profiles need to be taken into consideration in the synthesis process. `TuLiP` alleviates this complexity by integrating a receding horizon framework [9]. Additionally, it admits general affine dynamics with bounded disturbances.

2. TuLiP FEATURES

We now summarize two key features of `TuLiP` (available at `http://www.cds.caltech.edu/tulip`).

2.1 Embedded Control Software Synthesis

`TuLiP` deals with systems that comprise the plant, i.e., the physical component regulated by the controller, and its potentially dynamic and a priori unknown environment. Note that the environment does not only include the factors that are external to the plant but it also includes the factors over which the system does not have control, e.g., hardware failure. The plant may contain both continuous (e.g. physical) and discrete (e.g. computational) components. `TuLiP` models the embedded control software synthesis problem as a game between the plant and the environment. Given the model of the plant and system specification φ in LTL, `TuLiP` provides a function that automatically synthesizes a controller that ensures system correctness with respect to φ for any admissible environment, if such a controller exists (i.e., φ is realizable). If φ is unrealizable, `TuLiP` also provides counter examples, i.e., initial states starting from which the environment can falsify φ regardless of controller's actions.

The synthesis feature relies on (1) generating a proposition preserving partition of the continuous state space, (2) continuous state space discretization based on the evolution of the continuous state [8], and (3) digital design synthesis. JTLV [6] is used as the underlying synthesis routine.

Currently, `TuLiP` handles the case where the continuous state of the plant evolves according to discrete-time linear time-invariant dynamics: for $t \in \{0, 1, 2, \ldots\}$, $s[t+1] = As[t] + Bu[t] + Ed[t]$, $u[t] \in \mathcal{U}$, $d[t] \in \mathcal{D}$, $s[0] \in \mathcal{S}$, where $\mathcal{S} \in \mathbb{R}^n$ is the continuous state space, $\mathcal{U} \in \mathbb{R}^m$ and $\mathcal{D} \in \mathbb{R}^p$ are the sets of admissible control inputs and exogenous disturbances, $s[t], u[t], d[t]$ are the continuous state, the control signal and the exogenous disturbance, respectively, at time t. $\mathcal{U}, \mathcal{D}, \mathcal{S}$ are assumed to be bounded polytopes.

The specification φ is assumed to be of the form

$$\varphi = \big(\psi_{init} \wedge \Box\psi_e \wedge \bigwedge_{i \in I_f} \Box\Diamond\psi_{f,i}\big) \implies \big(\Box\psi_s \wedge \bigwedge_{i \in I_g} \Box\Diamond\psi_{g,i}\big),$$

known as GR[1]. Here $\psi_{init}, \psi_e, \psi_{f,i}, i \in I_f, \psi_s$ and $\psi_{g,i}, i \in I_g$ are propositional formulas. ψ_{init}, ψ_e and $\psi_{f,i}, i \in I_f$ essentially describe the assumptions on the initial state of the system and the environment. ψ_s and $\psi_{g,i}, i \in I_g$ describe the desired behavior of the system. See [9] for more details.

2.2 Receding Horizon Framework

For systems with a certain structure, the computational complexity of the planner synthesis can be alleviated by solving the planning problems in a receding horizon fashion, i.e., compute the plan or strategy over a "shorter" horizon, starting from the current state, implement the initial portion of the plan, and recompute the plan. This approach essentially reduces the problem into a set of smaller problems. Certain sufficient conditions ensure that this "receding horizon" strategy preserves the desired system-level properties.

Given a specification in the form of φ above, `TuLiP` first constructs a finite state abstraction of the physical system. Then, for each $i \in I_g$, we organize the system states into a partially ordered set $\mathcal{P}^i = (\{\mathcal{W}^i_j\}, \preceq_{\psi_{g,i}})$ where $\mathcal{W}^i_0$ are the set of states satisfying $\psi_{g,i}$. For each j, we define a short-horizon specification Ψ^i_j associated with $\mathcal{W}^i_j$ as

$$\begin{aligned}\Psi^i_j \;=\; & \big((\nu \in \mathcal{W}^i_j) \wedge \Phi \wedge \Box\psi_e \wedge \textstyle\bigwedge_{k \in I_f} \Box\Diamond\psi_{f,k}\big) \\ & \implies \big(\Box\psi_s \wedge \Box\Diamond(\nu \in \mathcal{F}^i(\mathcal{W}^i_j)) \wedge \Box\Phi\big).\end{aligned}$$

Here Φ is a propositional formula that describes receding horizon invariants and $\mathcal{F}^i : \{\mathcal{W}^i_j\} \to \{\mathcal{W}^i_j\}$ defines the intermediate goal for starting in $\mathcal{W}^i_j$. Let $\mathcal{V}$ be the entire state space of the system. As described in [9], a sufficient condition for the receding horizon strategy to lead to correct execution with respect to φ is that for all $i \in I_g$, (1) $\mathcal{W}^i_0 \cup \mathcal{W}^i_1 \cup \ldots \cup \mathcal{W}^i_p = \mathcal{V}$, (2) $\mathcal{W}^i_0 \prec_{\psi_{g,i}} \mathcal{W}^i_j, \forall j \neq 0$, (3) $\mathcal{F}^i(\mathcal{W}^i_j) \prec_{\psi_{g,i}} \mathcal{W}^i_j, \forall j \neq 0$, (4) $\psi_{init} \implies \Phi$ is a tautology, and (5) Ψ^i_j is realizable $\forall j$.

Given the plant model, φ, $\{\mathcal{W}^i_j\}$, $\mathcal{F}^i$ and Φ, `TuLiP` automatically constructs the short horizon specification Ψ^i_j for each i, j. It includes functions for verifying that there exists a partial order $\preceq_{\psi_{g,i}}$ and that the sufficient condition above is satisfied; and automatically computing the receding horizon invariant Φ if one exists or report an error otherwise.

3. APPLICATIONS AND DISCUSSIONS

We have demonstrated the successful applications of `TuLiP` in multiple applications, including autonomous driving [9], vehicle management systems in avionics [10] and multi-target tracking. Other simpler examples are included in the current release of the toolbox. For the autonomous driving problem, the receding horizon framework needs to be applied for the car to be able to drive a reasonable distance. Due to the state explosion problem, `TuLiP` cannot automatically find the receding horizon invariant Φ for this specific application. Nevertheless, it provides useful guidelines for the user to easily manually construct Φ. Once Φ is constructed, `TuLiP` successfully checks that the sufficient condition for applying the receding horizon strategy is satisfied.

`TuLiP` constructs Φ roughly by starting $\Phi = \textit{True}$ and iterating between (1) checking the realizability of each Ψ^i_j, and (2) updating Φ to be the conjunction of current Φ and the negation of the counter examples of unrealizable Ψ^i_j (if any). This process stops when $\psi_{init} \implies \Phi$ is no longer a tautology or all the Ψ^i_j are realizable. Since the counter examples are given as the enumeration of all the infeasible initial states, the size of Φ quickly increases. An extension of the current version of `TuLiP` is to implement a procedure for reducing the counter examples into a small formula. We also plan to integrate various existing software packages into `TuLiP` including a user-friendly simulation environment such as Player/Stage [2] and a state space discretization procedure that admits a more general class of systems (e.g. one based on approximate simulations and bisimulations as discussed in [7] and implemented in [5]).

4. REFERENCES

[1] C. Finucane, G. Jing, and H. Kress-Gazit. LTLMoP. `http://code.google.com/p/ltlmop/`.

[2] B. Gerkey, R. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Conf. on Advanced Robotics*, 2003.

[3] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE CDC*, 2009.

[4] M. Kloetzer and C. Belta. LTLCon. `http://iasi.bu.edu/~software/LTL-control.htm`.

[5] M. Mazo, A. Davitian, and P. Tabuada. Pessoa: A tool for embedded controller synthesis. In T. Touili, B. Cook, and P. Jackson, editors, *CAV*, volume 6174 of *LNCS*, pages 566–569. Springer, 2010.

[6] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *Verification, Model Checking and Abstract Interpretation*, volume 3855 of *LNCS*, pages 364 – 380. Springer, 2006. `http://jtlv.sourceforge.net/`.

[7] P. Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach.* Springer, 2009.

[8] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Automatic synthesis of robust embedded control software. In *AAAI SS on Embedded Reasoning: Intelligence in Emb'd Systems*, pages 104–111, 2010.

[9] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In *HSCC*, pages 101–110, 2010.

[10] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Formal synthesis of embedded control software: Application to vehicle management systems. In *AIAA Infotech@Aerospace*, 2011. submitted.

[11] B. Yordanov and C. Belta. conPAS2. `http://hyness.bu.edu/conPAS2.html`.