Theoretical Computer Science 691 (2017) 27–54

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Time window temporal logic ☆

Cristian-Ioan Vasile [a,*,1,2], Derya Aksaray [b,2], Calin Belta [b]

[a] Division of Systems Engineering, Boston University, Brookline, MA 02446, United States
[b] Department of Mechanical Engineering, Boston University, Boston, MA 02215, United States

ARTICLE INFO

Article history:
Received 12 March 2016
Received in revised form 19 June 2017
Accepted 17 July 2017
Available online 24 July 2017
Communicated by J.-F. Raskin

Keywords:
Timed temporal logic
Temporal relaxation
Controller synthesis
Verification
Finite state automata
Unambiguous languages

ABSTRACT

This paper introduces *time window temporal logic* (TWTL), a rich expressive language for describing various time bounded specifications. In particular, the syntax and semantics of TWTL enable the compact representation of serial tasks, which are prevalent in various applications including robotics, sensor systems, and manufacturing systems. This paper also discusses the relaxation of TWTL formulae with respect to the deadlines of the tasks. Efficient automata-based frameworks are presented to solve synthesis, verification and learning problems. The key ingredient to the presented solution is an algorithm to translate a TWTL formula to an annotated finite state automaton that encodes all possible temporal relaxations of the given formula. Some case studies are presented to illustrate the expressivity of the logic and the proposed algorithms.

1. Introduction

Temporal logics provide mathematical formalisms to reason about (concurrent) events in terms of time. Due to their rich expressivity, they have been widely used as specification languages to describe properties related to correctness, termination, mutual exclusion, reachability, or liveness [34]. Recently, there has been great interest in using temporal logic formulae in the analysis and control of dynamical systems. For example, linear temporal logic (LTL) [5] has been extensively used in motion planning and control of robotic systems, e.g., [42,20,1,45,6,44,22,11,27,30].

In some real-world applications, the tasks may involve some time constraints (e.g., [38,36]). For example, consider a robot that is required to achieve the following tasks: every visit to A needs to be immediately followed by visiting B within 5 time units; two consecutive visits to A need to be at least 10 time units apart; or visiting A and visiting B need to be completed within 15 time units. Such tasks cannot be described by LTL formulae since LTL cannot deal with temporal properties with explicit time constraints. Therefore, bounded temporal logics are used to capture the time constraints over the tasks. Examples are bounded linear temporal logic (BLTL) [39,18], metric temporal logic (MTL) [26], and signal temporal logic (STL) [33].

☆ This work was supported in part by NSF grants numbers NRI-1426907, NSF CMMI-1400167, and ONR grant number N00014-14-1-0554.
* Corresponding author.
E-mail addresses: cvasile@mit.edu (C.-I. Vasile), daksaray@mit.edu (D. Aksaray), cbelta@bu.edu (C. Belta).
[1] Present address: Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA.
[2] Present address: MIT Computer Science and Artificial Intelligence Laboratory, The Stata Center, Building 32, 32 Vassar Street, Cambridge, MA 02139, USA.

http://dx.doi.org/10.1016/j.tcs.2017.07.012
0304-3975/

In this paper, we propose a specification language called *time window temporal logic* (TWTL). The semantics of TWTL is rich enough to express a wide variety of time-bounded specifications, e.g., "*monitor A* for 3 time units within the time interval $[0, 5]$ and after that *monitor B* for 2 time units within $[4, 9]$. This logic was defined in our previous conference papers [43,2], and used to specify persistent surveillance tasks for multi-robot systems. Moreover, we define a notion of *temporal relaxation* of a TWTL formula, which is a quantity computed over the time intervals of a given TWTL formula. In this respect, if the temporal relaxation is: *negative*, then the tasks expressed in the formula should be completed before their designated time deadlines (i.e., satisfying the relaxed formula implies the satisfaction of a more strict formula than the original formula); *zero*, then the relaxed formula is exactly the same as the original formula; *positive*, then some tasks expressed in the formula are allowed to be completed after their original time deadlines (i.e., satisfying the relaxed formula may imply the violation of the original formula or the satisfaction of a less strict formula).

In this paper, we present an automata-based framework to solve verification, synthesis, and learning problems that involve TWTL specifications. One property of TWTL specifications we exploit in the proposed solutions is that the associated languages are finite. In the theoretical computer science literature, finite languages and the complexity of constructing their corresponding automata have been extensively studied [32,16,7,12,9]. One of the main benefits of the proposed framework is its capability to efficiently construct the annotated automata that can encode not only the original formula but also all temporal relaxations of the given formula. Such an efficient construction mainly stems from the proposed algorithms that are specifically developed for TWTL formulae.

The proposed language TWTL has several advantages over existing temporal logics. First, in many robotics missions, a desired specification can be represented in a more compact and comprehensible way in TWTL than BLTL, MTL, or STL. For example, deadlines expressed in a TWTL formula indicate the exact time bounds as opposed to an STL formula where the time bounds can be shifted. Consider a specification as "stay at A for 4 time steps within the time window $[0, 10]$", which can be expressed in TWTL as $[H^4 A]^{[0,10]}$. The same specification can be expressed in STL as $F_{[0,10-4]} G_{[0,4]} A$ where the outermost time window needs to be modified with respect to the inner time window. Furthermore, compared to BLTL and MTL, the existence of an explicit concatenation operator results in a more compact representation for serial tasks that are prevalent in various applications including robotics, sensor systems, and manufacturing systems. Under some mild assumptions, we provide a very efficient (linear-time) algorithm to handle concatenation of tasks. In general, the complexity associated with the concatenation operation is exponential in the worst case, even for finite languages [32].

Second, the notion of temporal relaxation enables a generic framework to construct the automaton of all possible relaxations of a TWTL formula. In literature, there are some studies investigating the control synthesis problems for minimal violations of LTL fragments [37,40,41,31,14]. In contrast to existing works, the annotated automaton proposed in this paper can encode all possible temporal relaxations of a given formula. Accordingly, such an automaton can be used in a variety of problems related to synthesis, verification, and learning to satisfy minimally relaxed formulae. Third, we show that the complexity of constructing the automata for a given TWTL formula is independent of the corresponding time bounds. To achieve this property, we exploit the structure of finite languages encoded by TWTL formulae.

We present a set of provably-correct algorithms to construct the automaton of a given TWTL formula (both for the relaxed and unrelaxed cases). We formulate a generic problem in terms of temporal relaxation of a TWTL formula, which can be specialized into problems such as verification, synthesis, and learning. We developed a Python package to solve these three problems, which is available for download from *hyness.bu.edu/twtl*.

2. Preliminaries

In this section, we introduce the notation and briefly review the main concepts from formal languages, automata theory, and formal verification. For a detailed exposition of these topics, the reader is refereed to [5,17] and the references therein.

Given $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$, $n \geq 2$, the relationship $\mathbf{x} \sim \mathbf{x}'$, where $\sim \in \{<, \leq, >, \geq\}$, is true if it holds pairwise for all components. $\mathbf{x} \sim a$ denotes $\mathbf{x} \sim a\mathbf{1}_n$, where $a \in \mathbb{R}$ and $\mathbf{1}_n$ is the n-dimensional vector of all ones. The extended set of real numbers is denoted by $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$.

Let Σ be a finite set. We denote the cardinality and the power set of Σ by $|\Sigma|$ and 2^Σ, respectively. A *word* over Σ is a finite or infinite sequence of elements from Σ. In this context, Σ is also called an *alphabet*. The length of a word w is denoted by $|w|$ (e.g., $|w| = \infty$ if w is an infinite word). Let k, $i \leq j$ be non-negative integers. The k-th element of w is denoted by w_k, and the sub-word $w_i, \ldots, w_j$ is denoted by $w_{i,j}$. A set of words over an alphabet Σ is called a *language* over Σ. The languages of all finite and infinite words over Σ are denoted by Σ^* and Σ^ω, respectively.

Definition 2.1 *(Prefix language).* Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two languages. We say that $\mathcal{L}_1$ is a prefix language of $\mathcal{L}_2$ if and only if every word in $\mathcal{L}_1$ is a prefix of some word in $\mathcal{L}_2$, i.e., for each word $w \in \mathcal{L}_1$ there exists $w' \in \mathcal{L}_2$ such that $w = w'_{0,i}$, where $0 \leq i < |w'|$. The maximal prefix language of a language $\mathcal{L}$ is denoted by $P(\mathcal{L}) = \{w_{0,i} \mid w \in \mathcal{L}, i \in \{0, \ldots, |w| - 1\}\}$.

Definition 2.2 *(Unambiguous language).* A language $\mathcal{L}$ is called unambiguous language if no proper subset L of $\mathcal{L}$ is a prefix language of $\mathcal{L} \setminus L$.

The above definition immediately implies that a word in an unambiguous language can not be the prefix of another word. Moreover, it is easy to show that the converse is also true.