An Efficient Algorithm for Monitoring Practical TPTL Specifications

Adel Dokhanchi, Bardh Hoxha, Cumhur Erkan Tuncali, and Georgios Fainekos
Arizona State University, Tempe, AZ, U.S.A.
Email: {adokhanc,bhoxha,etuncali,fainekos}@asu.edu

***Abstract*—We provide a dynamic programming algorithm for the monitoring of a fragment of Timed Propositional Temporal Logic (TPTL) specifications. This fragment of TPTL, which is more expressive than Metric Temporal Logic, is characterized by independent time variables which enable the elicitation of complex real-time requirements. For this fragment, we provide an efficient polynomial time algorithm for off-line monitoring of finite traces. Finally, we provide experimental results on a prototype implementation of our tool in order to demonstrate the feasibility of using our tool in practical applications.**

I. Introduction

In Cyber-Physical Systems (CPS), many safety critical components of the system are controlled by embedded computers which interact with the physical environment. Due to the safety-critical nature of these applications, it is important to verify their correctness during system development stages. However, the verification problem for CPS with respect to safety requirements is undecidable, in general [1]. An alternative to formal verification is semi-formal model-based testing and monitoring of CPS. We utilize formal logic, in order to formally specify real-time requirements.

Metric Temporal Logic (MTL) was introduced to provide the formalization of real-time specifications [16]. Since its introduction, MTL and its variants have been used in the verification of real-time systems [20]. Several tools, such as S-TaLiRo [3] and Breach [7], have been developed by the academic community for the purpose of semi-formal verification of MTL specifications. These tools use off-line and on-line monitoring algorithms to check whether the execution trace of a CPS satisfies/falsifies an MTL formula. In off-line monitoring, the execution trace is finite and generated by running the system for a bounded amount of time. Then, the off-line monitor checks whether the execution trace satisfies the specification. On the other hand, an on-line monitor runs simultaneously with the system. In this paper, we consider off-line monitoring of TPTL specifications.

The time complexity of off-line monitoring for MTL is linear to the size of a finite system trace and linear to the size of MTL formula. Several algorithms using dynamic programming [10] or sliding windows [8] have been proposed for MTL monitoring of CPS. In this paper, we consider TPTL specifications which are more expressive than MTL specifications [4]. TPTL is an extension of Linear Temporal Logic (LTL) with freeze quantifiers represented as "$x.$". A freeze quantifier $x.$ assigns to time variable x the "current" time stamp when the corresponding subformula $x.\varphi(x)$ is evaluated [2]. Then, the time value (stored in x) can be evaluated inside time constraints which are linear inequalities over the time variables.

Since its introduction, two semantics where considered for TPTL [2], [4]. Alur's semantics [2] allows two time variables in time constraints (for example $x + 1 \leq y + 4$). In contrast, Raskin's semantics allows only one time variable in the time constraint ($x \leq 4$) and implicitly considers the current time as the second time variable [4], [21]. Since the latter semantics was first considered by Jean-Franois Raskin in [21], we will refer to it as "Raskin's TPTL semantics" in this paper. Raskin's TPTL semantics was mentioned with alternative terms such as "Timed LTL" in [17]. In another line of work, in [6], the authors augmented Alur's time constraints with more complex temporal-special predicates to define the closeness property of two different CPS trajectories. However, the authors in [6] did not provide a TPTL monitoring algorithm.

Since TPTL subsumes MTL, it is expected that the monitoring problem of TPTL is computationally more complex [11]. It has been proven that monitoring of a finite trace with respect to Alur's TPTL specification is PSPACE-hard [18]. In [18], the authors transform a Quantified Boolean Formula (QBF), which is PSPACE-hard, into a TPTL formula with real value time variables. A similar complexity result (PSPACE-hard) for Raskin's TPTL semantics is obtained for integer time variables in [11]. It is mentioned in [11] that in order to obtain a polynomial time algorithm for TPTL monitoring (path checking), we need to fix the number of time variables. In other words, if the number of time variables is bounded then the finite trace monitoring will be polynomial to the size of the TPTL formula. However, in [11], the authors did not provide any applicable algorithm for TPTL monitoring and they focused only on the complexity class.

In this work, we move one step further from [11], and allow the number of time variables to be arbitrary, but they must be independent to each other[1]. For this fragment of TPTL, we provide an efficient TPTL monitoring algorithm which has time complexity quadratic in the length of the finite trace. In addition, the runtime of the algorithm is proportional to the number of time variables in TPTL.

In terms of related work, a rewriting based algorithm for TPTL has been provided in [5]. In [5], the authors did not evaluate the time complexity of their proposed algorithm. The rewriting technique was used for on-line monitoring of TPTL specifications in [13]. The authors used the relativization of TPTL formula with respect to the sequence of observed states [13], and it was reported that the time complexity is exponential to the size of TPTL formula [13]. To the best of our knowledge, our paper is the first work where an efficient

[1]In Section II-B, Definition 5, we introduce independent time variables.

and practical TPTL off-line monitoring algorithm is provided.

II. Preliminaries

We assume a sampled representation of system behavior with a discrete trace as the input to the monitoring algorithm. We utilize the notion of Timed State Sequences (TSS) [2] to represent the sampled behavior of a system using a digital clock. We interpret TPTL formulas over TSS. Assume $AP = \{a, b, \cdots\}$ is a set of atomic propositions, $\mathbb{R}_+$ is the set of non-negative real numbers, and $\mathbb{N}$ denotes non-negative integers.

Definition 1 (State and Time Sequences [2]): A state sequence $\sigma = \sigma_0\sigma_1\sigma_2\cdots$ is an infinite sequence of states $\sigma_i \subseteq AP$, where $i \in \mathbb{N}$. A (sampled) time sequence $\tau = \tau_0\tau_1\tau_2\ldots$ is an infinite sequence of time stamps $\tau_i \in \mathbb{R}_+$, where $i \in \mathbb{N}$.

We assume that the time sequence τ is:

1) **Initialized**, which means that the start up time is zero ($\tau_0 = 0$).
2) **Monotonic**, which means that $\tau_i \leq \tau_{i+1}$ for all $i \in \mathbb{N}$.
3) **Progressive**, which means that for all $t \in \mathbb{R}_+$ there is some $i \in \mathbb{N}$ such that $\tau_i > t$.

Definition 2 (Timed State Sequence (TSS) [2]): A timed state sequence $\rho = (\sigma, \tau)$ is a pair consisting of a state sequence σ and a time sequence τ where $\rho_0\rho_1\rho_2\cdots = (\sigma_0, \tau_0)(\sigma_1, \tau_1)(\sigma_2, \tau_2)\cdots$.

Given an infinite TSS ρ, we consider a finite prefix of ρ as a finite TSS. The symbol $\hat{\rho} = (\hat{\sigma}, \hat{\tau})$ is used to denote a finite TSS with the size of $|\hat{\rho}| = |\hat{\sigma}| = |\hat{\tau}|$. In this paper, we consider the monitoring of finite TSS with the size of $|\hat{\rho}|$ which is equal to the number of simulation/execution samples.

A. TPTL Syntax and Semantics

To prevent any confusion in the presentation, we consider Raskin's TPTL semantics [21], [4][2]. TPTL is an extension of LTL that enables the formalization of real-time properties by including time variables and a freeze time quantifier [2].

Definition 3 (Syntax for TPTL): The set of TPTL formulas φ over a finite set of atomic propositions (AP) and a finite set of time variables (V) is inductively defined according to the following grammar:

$$\varphi ::= \top \mid a \mid x \sim r \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 U \varphi_2 \mid x.\varphi$$

where $x \in V$, $r \in \mathbb{R}_+$, $a \in AP$, and $\sim \in \{\leq, <, =, >, \geq\}$, and $\top$ is the symbol for "True".
The time constraints of TPTL are represented in the form of $x \sim r$. The freeze quantifier $x.$ assigns the current time of the formula's evaluation (at each sampled time τ_i) to the time variable x. A TPTL formula is *closed* if every occurrence of a time variable is within the scope of a freeze quantifier [2]. In TPTL specifications, we always deal with closed formulas.

We note that "False" is represented as $\bot \equiv \neg\top$ and "Implication" is represented as $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$. For all formulas ψ, ϕ, $\Diamond\psi \equiv \top U\psi$ (Eventually ψ), $\Box\psi \equiv \neg\Diamond\neg\psi$ (Always ψ), and $\psi R\phi \equiv \neg(\neg\psi U\neg\phi)$ (ψ Releases ϕ) are defined in the conventional way. Since we focus on off-line monitoring, we only consider the TPTL semantics for finite traces.

[2]We will explain in Section II-B why we chose Raskin's semantics.

Definition 4 (Discrete-Time Semantics for TPTL): Let $\hat{\rho} = (\hat{\sigma}, \hat{\tau})$ be a finite TSS and $i \in \mathbb{N}$ where $i < |\hat{\rho}|$ is the index of the current sample, $a \in AP$, $\varphi \in TPTL$, and an environment $\varepsilon : V \rightarrow \mathbb{R}_+$. The satisfaction relation $(\hat{\rho}, i, \varepsilon) \models \varphi$ is defined recursively as follows:

$(\hat{\rho}, i, \varepsilon) \models \top$
$(\hat{\rho}, i, \varepsilon) \models a$ iff $a \in \sigma_i$
$(\hat{\rho}, i, \varepsilon) \models \neg\varphi$ iff $(\hat{\rho}, i, \varepsilon) \not\models \varphi$
$(\hat{\rho}, i, \varepsilon) \models \varphi_1 \wedge \varphi_2$ iff $(\hat{\rho}, i, \varepsilon) \models \varphi_1$ and $(\hat{\rho}, i, \varepsilon) \models \varphi_2$
$(\hat{\rho}, i, \varepsilon) \models \varphi_1 \vee \varphi_2$ iff $(\hat{\rho}, i, \varepsilon) \models \varphi_1$ or $(\hat{\rho}, i, \varepsilon) \models \varphi_2$
$(\hat{\rho}, i, \varepsilon) \models \bigcirc\varphi$ iff $(\hat{\rho}, i+1, \varepsilon) \models \varphi$ and $i < (|\hat{\rho}| - 1)$
$(\hat{\rho}, i, \varepsilon) \models \varphi_1 U \varphi_2$ iff $\exists j, i \leq j < |\hat{\rho}|$ s.t. $(\hat{\rho}, j, \varepsilon) \models \varphi_2$ and $\forall k, i \leq k < j$ it holds that $(\hat{\rho}, k, \varepsilon) \models \varphi_1$
$(\hat{\rho}, i, \varepsilon) \models x \sim r$ iff $(\tau_i - \varepsilon(x)) \sim r$ i.e. (current time stamp) $- \varepsilon(x) \sim r$
$(\hat{\rho}, i, \varepsilon) \models x.\varphi$ iff $(\hat{\rho}, i, \varepsilon[x := \tau_i]) \models \varphi$

The semantics of TPTL are defined over an evaluation function $\varepsilon : V \rightarrow \mathbb{R}_+$ which is an environment for the time variables. Assume $x = r$ where $x \in V$, and $r \in \mathbb{R}_+$, then we have $\varepsilon(x) = r$. Given a variable $x \in V$ and a real number $q \in \mathbb{R}_+$, we denote the environment with $\varepsilon' = \varepsilon[x := q]$ which is equivalent to the environment ε on all time variables in V except variable x. The assignment operation $x := q$ changes the environment ε to the new environment ε'. Formally, $\varepsilon'(y) = \varepsilon(y)$ for all $y \neq x$ and $\varepsilon'(x) = q$. We write $\mathbf{0}$ for the (**zero**) environment such that $\mathbf{0}(x) = 0$ for all $x \in V$. We say that $\hat{\rho}$ satisfies φ ($\hat{\rho} \models \varphi$) iff $(\hat{\rho}, 0, \mathbf{0}) \models \varphi$. A variable "$x$" that is bounded by a corresponding *freeze quantifier* "$x.$" saves the local temporal context τ_i (now) in "x". Assume $\varphi(x)$ is a formula with a free variable x. The TSS $\hat{\rho}$ satisfies $x.\varphi(x)$ if it satisfies $\varphi(\tau_0 = 0)$, where $\varphi(0)$ is obtained from $\varphi(x)$ by replacing all the free occurrences of the variable x with constant 0 [2].

B. TPTL Fragments

In this section, we introduce a TPTL fragment for which we have developed a monitoring algorithm. This restriction is crucial for obtaining the polynomial runtime of the algorithm.

Definition 5 (Independent Time Variable): A time variable x is independent if it is in the scope of only one freeze quantifier $x.$ and no other time variable is in the scope of the corresponding freeze quantifier ($x.$).

For example in $x.(\psi(x) \vee \Diamond y.\varphi(x, y))$, neither x nor y is independent. This is because x is within the scope of the freeze time quantifiers $x.$ in $x.(\psi(x) \vee \Diamond y.\varphi(x, y))$ and $y.$ in $y.\varphi(x, y)$. Similarly, y is not the only time variable that is within the scope of $y.$ in $y.\varphi(x, y)$. However, both x and y are independent in $x.(\psi(x) \vee \Diamond y.\varphi(y))$.

Now we explain why we focus on Raskin's semantics in our monitoring algorithm. In Raskin's semantics, each time constraint contains a single time variable (see Definition 3). However, in Alur's semantics each time constraint contains two time variables [2]. In Alur's semantics, time variables in the same constraint are dependent to each other. As a result, in order to benefit from independent time variables, we should consider Raskin's semantics.

Definition 6 (Encapsulated TPTL formula): Encapsulated TPTL formulas are TPTL formulas where all the time variables are independent.

In other words, an encapsulated formula is a closed formula in which every sub-formula has at most one free time variable.

Definition 7 (Frozen Subformula): Given an encapsulated TPTL formula Φ, a frozen subformula ϕ of Φ is a subformula which is bounded by a freeze quantifier corresponding to (an independent) time variable.

In encapsulated formulas, all the closed subformulas are frozen. For example the formula $x.(\psi(x) \vee \Diamond y.\varphi(x,y))$ is not an "encapsulated" formula because $y.\varphi(x,y)$ is not frozen since x, y are not independent. Here are two TPTL formulas φ_1,φ_2 that look similar but only one of them is encapsulated.

- $\varphi_1 = \Box x.\Diamond(a \wedge x \leq 10 \wedge y.\Box(\mathbf{y} \leq \mathbf{2} \wedge y \geq 1 \wedge b))$
- $\varphi_2 = \Box x.\Diamond(a \wedge x \leq 10 \wedge y.\Box(\mathbf{x} \leq \mathbf{2} \wedge y \geq 1 \wedge b))$

In the above, φ_1 is encapsulated, but φ_2 is not encapsulated since $y.\Box(x \leq 2 \wedge y \geq 1 \wedge b)$ where $x \leq 2$ is inside the scope of "$y.$".

Lemma 1: Any MTL formula can be represented by an "encapsulated" TPTL formula.

Proof: Each time interval of an MTL temporal operator can be represented with a unique time variable which is independent of the rest of time variables. The syntactic modification works as follows: every MTL formula of the form $\varphi = \psi U_{[l,u]}\phi$ can be recursively represented as the following TPTL formula $\varphi = x.(\psi U(x \geq l \wedge x \leq u \wedge \phi))$. The resulting TPTL formula is encapsulated. ■

Lemma 2: MTL is less expressive than "encapsulated" TPTL formulas.

Proof: It is proven in [4] that the following TPTL formula, which is evidently encapsulated, cannot be expressed by any MTL formula [4]: $\psi = x.\Diamond(a \wedge x \leq 1 \wedge \Box(x \leq 1 \rightarrow \neg b))$ ■

In the rest of the paper, we focus on the following problem:

Problem 1: Given a finite TSS $\hat{\rho}$ and an "encapsulated" TPTL formula φ, check whether $\hat{\rho}$ satisfies φ ($\hat{\rho} \models \varphi$).

III. Monitoring Encapsulated TPTL Formulas

A. TPTL Representation

In the following, we will describe the data structure that will be utilized to capture the solution for the TPTL monitoring problem. We store each TPTL formula in a binary tree data structure. Consider the following example:

Example 1: Assume $AP = \{a, b\}$ and let
$\phi = \Box x.\Diamond((x \leq 1 \rightarrow a) \wedge y.\Diamond(y \leq 1 \rightarrow \neg b))$
$\phi \equiv \Box x.\Diamond((x \leq 1 \rightarrow a) \wedge y.\psi_1(y)) \equiv \Box x.\psi_2(x)$
where we use ψ_1 and ψ_2 to simplify the presentation:
$\psi_1(y) \equiv \Diamond(y \leq 1 \rightarrow \neg b)$
$\psi_2(x) \equiv \Diamond((x \leq 1 \rightarrow a) \wedge y.\psi_1(y))$

In this example, we have two independent time variables x and y. The binary tree of Example 1 is depicted in Fig. 1. There, the thirteen nodes correspond to thirteen subformulas.

In Fig. 1, each subformula φ_i has a node corresponding to the highest operator for φ_i. In addition, for each subformula φ_i we assign an index i. The order of indexes is generated according to a topological sort where parents have lower index

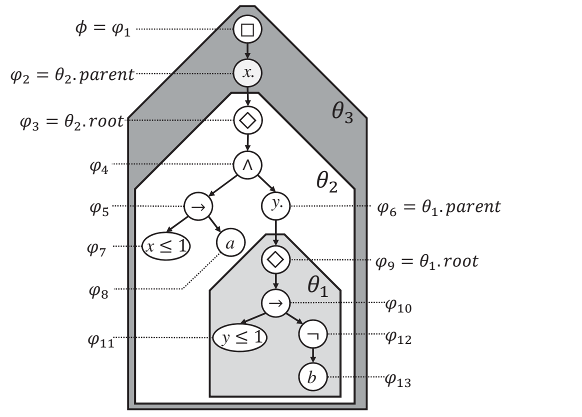

Fig. 1. Binary tree of Example 1 (ϕ) with three subtrees corresponding to sets of subformulas $\theta_1, \theta_2, \theta_3$.

values than children. Therefore, the original subformula ϕ obtains the index 1 because it is the first visited. To evaluate each node's $\top/\bot$ value we need to evaluate its children's $\top/\bot$ value before, this is because of the TPTL recursive semantics (see Definition 4). If we evaluate the nodes in the decreasing order of indexes, we would be able to evaluate all the children before their parents.

Now, we must partition the formula tree into subtrees rooted by the freeze time operators. Since in Example 1, we have two independent time variables, we created 2+1 subtrees (two for time variables and one for the original formula). Each subtree contains a set of subformulas. These subformulas and their corresponding subtrees $\theta_1, \theta_2, \theta_3$ are shown in Fig. 1 with different colors:

The set θ_1 contains subformulas rooted at node φ_9 represented in the **light-gray** subtree. The set θ_1 contains the subformulas of $y.\psi_1(y)$ as follows $\theta_1 = \{\Diamond(y \leq 1 \rightarrow \neg b), y \leq 1 \rightarrow \neg b, y \leq 1, \neg b, b\} = \{\varphi_9, \varphi_{10}, \varphi_{11}, \varphi_{12}, \varphi_{13}\}$.

The set θ_2 contains subformulas rooted at node φ_3 represented in the **white** subtree. The set θ_2 contains the subformulas of $x.\psi_2(x)$ as follows $\theta_2 = \{\Diamond((x \leq 1 \rightarrow a) \wedge y.\psi_1(y)), (x \leq 1 \rightarrow a) \wedge y.\psi_1(y), (x \leq 1 \rightarrow a), y.\psi_1(y), x \leq 1, a\} = \{\varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7, \varphi_8\}$.

The set θ_3 contains subformulas rooted at node φ_1 represented in **dark-gray** subtree. The set θ_3 contains the subformulas of $\theta_3 = \{\Box x.\psi_2(x)\ ,\ x.\psi_2(x)\} = \{\varphi_1, \varphi_2\}$.

Each of the subtrees θ_1 and θ_2 have distinguished fields referencing to (the index of) *parent* and *root* nodes which are represented in Fig. 1 as follows:
1) $\theta_1.parent = 6$ and $\theta_1.root = 9$.
2) $\theta_2.parent = 2$ and $\theta_2.root = 3$.

Note that θ_1 is subformula of θ_2, and θ_2 is subformula of θ_3. This ordering is very important for our algorithm. We created these subtrees because each frozen subformula can be separately evaluated. Therefore, we can guarantee the polynomial runtime. The method will be described in details in Section IV.

B. Monitoring Table

We assume that the sampled system output is mapped (projected) on a *finite* TSS $\hat{\rho}$; therefore, we can evaluate the system output using our off-line monitor. If the specification does not have a freeze time operator, then the formula is an LTL formula for which the existing monitoring algorithms will be utilized [22]. If the specification has a freeze time operator, we first "instantiate" the time variable with the time label of the current sample before formula evaluation. Then, we compute $\bot/\top$ values of the corresponding time constraints. When time constraints are evaluated, they will be resolved to $\bot/\top$, and then, the frozen subformula $(x.\varphi(x))$ is converted into an LTL formula. Hence, we can apply dynamic programming method [22] to compute the Boolean value of the frozen subformula.

For each frozen subformula $(x.\varphi(x))$ at each time instance τ_i, we must first precompute the Boolean $(\bot/\top)$ value of the corresponding time constraints to transform this frozen subformula into an LTL. A two-dimensional matrix $M_{|\phi|\times|\hat{\rho}|}$ with height (number of rows) $|\phi|$, and width (number of columns) $|\hat{\rho}|$ is created. Here $|\phi|$ denotes the number of subformulas in ϕ, and $|\hat{\rho}|$ is the number of samples. Note that row indexing starts from 1 $(\phi \equiv \varphi_1)$ up to $|\phi|$ and column indexing starts from 0 (ρ_0) up to $|\hat{\rho}| - 1$.

The monitoring table of Example 1 is presented in Table I. At the beginning, the system outputs corresponding to atomic propositions $(AP = \{a, b\})$ are stored in the rows which belong to the propositions a (row φ_8) and b (row φ_{13}) in Table I. In Fig. 1, the subformula $\psi_2(x)$ is depicted inside the **white** subtree and $\psi_1(y)$ is depicted inside the **light-gray** subtree. In the following, we explain the other rows of Table I and provide a high level overview of the monitoring of ϕ:

1st Run) We first instantiate time variable y at each sample i with the corresponding timed instance τ_i to evaluate the Boolean values for the corresponding time constraint $y \leq 1$ (row φ_{11}). The instantiation transforms $y.\psi_1(y)$ into an LTL formula. Then we compute the Boolean values of $\psi_1(\tau_0)$, $\psi_1(\tau_1)$, $\psi_1(\tau_2)$, $\ldots$, $\psi_1(\tau_6)$ from left to right. Now the Boolean value of $y.\psi_1(y)$ for each time stamp τ_i is available for the higher level subtree of the Table I. Therefore, the Boolean values should be copied from row φ_9 to row φ_6.

2nd Run) Given the $\bot/\top$ values of $y.\psi_1(y)$, we can instantiate x at each time stamp τ_i and modify formula $x.\psi_2(x)$ into an LTL formula. Then we compute the Boolean values of $\psi_2(\tau_0)$, $\psi_2(\tau_1)$, $\psi_2(\tau_2)$, $\ldots$, $\psi_2(\tau_6)$ from left to right. Now the Boolean values of $x.\psi_2(x)$ are available for each time stamp τ_i for the higher subtree. As a result, the $\bot/\top$ values should be copied from row φ_3 to row φ_2.

3rd Run) The Boolean value of $\Box x.\psi_2(x)$ is computed given the Boolean values of $\psi_2(\tau_i)$ according to the semantics of Always $(\Box)$ operator:

$$\phi \equiv \bigwedge_{i=0}^{6} \psi_2(\tau_i)$$

IV. TPTL Monitoring Algorithm

The algorithms has the main following steps.

1) For each time variable (frozen subformula) and for each time stamp.
2) Resolve the time constraints into $\bot/\top$ values (This step converts the corresponding frozen subformula into an LTL formula).

TABLE I. The Monitoring Table of formula ϕ of Example 1 (Fig. 1)

φ_i(OP)	τ_0	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
$\varphi_1(\Box)$	$\{\bot/\top\}$						
$\varphi_2(x.)$	$\psi_2(0)$	$\psi_2(\tau_1)$	$\psi_2(\tau_2)$	$\psi_2(\tau_3)$	$\psi_2(\tau_4)$	$\psi_2(\tau_5)$	$\psi_2(\tau_6)$
$\varphi_3(\Diamond)$	$\psi_2(0)$	$\psi_2(\tau_1)$	$\psi_2(\tau_2)$	$\psi_2(\tau_3)$	$\psi_2(\tau_4)$	$\psi_2(\tau_5)$	$\psi_2(\tau_6)$
$\varphi_4(\wedge)$							
$\varphi_5(\rightarrow)$							
$\varphi_6(y.)$	$\psi_1(0)$	$\psi_1(\tau_1)$	$\psi_1(\tau_2)$	$\psi_1(\tau_3)$	$\psi_1(\tau_4)$	$\psi_1(\tau_5)$	$\psi_1(\tau_6)$
$\varphi_7(x \leq 1)$							
$\varphi_8(a)$							
$\varphi_9(\Diamond)$	$\psi_1(0)$	$\psi_1(\tau_1)$	$\psi_1(\tau_2)$	$\psi_1(\tau_3)$	$\psi_1(\tau_4)$	$\psi_1(\tau_5)$	$\psi_1(\tau_6)$
$\varphi_{10}(\rightarrow)$							
$\varphi_{11}(y \leq 1)$							
$\varphi_{12}(\neg)$							
$\varphi_{13}(b)$							

3) Compute $\bot/\top$ value of the resulting LTL formula using the dynamic programming algorithm.
4) These $\bot/\top$ values of frozen subformula are used to evaluate the higher level subformulas.

In the following, a detailed description and pseudo code of the proposed algorithm for TPTL monitoring will be explained.

A. TPTL to LTL Transformation

The pseudo code of the monitoring algorithm is provided in Algorithm 1 and its main loop has $|V| + 1$ iterations where $|V|$ is the number of freeze time variables. Algorithm 1 calls Algorithm 2 for computing the Boolean value of LTL subformulas. The first line of Algorithm 1 sets the monitoring table entries of the corresponding atomic propositions, namely the Boolean value of each $p \in AP$ is extracted from the finite state sequence $\hat{\sigma}$. In addition, Line 1 sets the monitoring table entries for constant boolean values $\bot/\top$. For each time variable v_k (in Line 2), we need to compute the $\bot/\top$ value of the subtree θ_k. The order of k is in such away that the inner most subtree (θ_1) is evaluated first then θ_2, and finally, θ_3 (See Fig 1 for Example 1). This order is crucial for the correctness of the algorithm, because higher level subformulas consider the lower level frozen subformulas as $\bot/\top$.

To transform the frozen formula into LTL for each sample time t between 0 to $|\hat{\rho}| - 1$ (see Line 3), we must first instantiate the time variable v_k to the corresponding time stamp τ_t, then compute the Boolean value of the corresponding time constraint $v_k \sim r$. The instantiation evaluates the whole constraint row into $\bot/\top$ in Lines 4-13 of Algorithm 1. The environment is updated based on the time stamp τ_t and the formula translated into an LTL formula. Now we use a dynamic programming algorithm based on [22] to compute the $\bot/\top$ value of the frozen subformula in Lines 14-18. In Line 15 of Algorithm 1, $\theta_k.max$ $(\theta_k.min)$ is the maximum (minimum) index of subformulas in the subtree θ_k. In Example 1:
1) $\theta_1.min = 9$ and $\theta_1.max = 13$
2) $\theta_2.min = 3$ and $\theta_2.max = 8$

When the Boolean value of the frozen subformula of $v_k.\psi(v_k)$ $(\theta_k.root)$ at time stamp $v_k = \tau_t$ is resolved, this Boolean value is copied to the parent of θ_k $(\theta_k.parent)$ to be used by higher level subformulas (see Line 19 of Algorithm 1). The loop of Line 3-20 continues for the other time stamps $(\tau_1 \ldots \tau_{|\hat{\rho}|-1})$ and computes the $\bot/\top$ value of the frozen subformula for each instantiation of v_k to the time stamps $\tau_1 \ldots \tau_{|\hat{\rho}|-1}$ in this order. Now we resolved the $\bot/\top$ value of the frozen subformula of $v_k.\psi(v_k)$ for all time stamps. We continue this process for other time variables (Lines 2-21).

Algorithm 1 TPTL Monitor

Input: φ, $\hat{\rho} = (\sigma_0, \tau_0)(\sigma_1, \tau_1)\cdots(\sigma_T, \tau_T)$; **Global variables:** $M_{|\varphi|\times|\hat{\rho}|}$; **Output**: $M[1,0]$.

```
procedure TPTLMONITOR($\varphi, \hat{\rho}$)
1: Initialize all rows in $M_{|\varphi|\times|\hat{\rho}|}$ corresponding to predicates
   $\varphi_j \equiv p \in AP$ with $\top/\bot$ value according to $\hat{\sigma}$.
2: for $k \leftarrow 1$ to $|V|$ do
3:     for $t \leftarrow 0$ to $|\hat{\rho}| - 1$ do
4:         for $u \leftarrow t$ to $|\hat{\rho}| - 1$ do
5:             for each $\varphi_j \equiv v_k \sim r \in \theta_k$ where
6:                 $j$ is the index of $v_k \sim r$ in $M$ do
7:                 if $(\tau_u - \tau_t) \sim r$ then
8:                     $M[j,u] \leftarrow \top$
9:                 else
10:                    $M[j,u] \leftarrow \bot$
11:                end if
12:            end for
13:        end for
14:        for $u \leftarrow |\hat{\rho}| - 1$ down to $t$ do
15:            for $j \leftarrow \theta_k.max$ down to $\theta_k.min$ do
16:                $M[j,u] \leftarrow ComputeLTL(\varphi_j, u, M_{|\varphi|\times|\hat{\rho}|})$
17:            end for
18:        end for
19:        $M[\theta_k.parent, t] \leftarrow M[\theta_k.root, t]$
20:    end for
21: end for
22: for $u \leftarrow |\hat{\rho}| - 1$ down to 0 do
23:     for $j \leftarrow \theta_{|V|+1}.max$ down to $\theta_{|V|+1}.min$ do
24:         $M[j,u] \leftarrow ComputeLTL(\varphi_j, u, M_{|\varphi|\times|\hat{\rho}|})$
25:     end for
26: end for
27: return $M[1,0]$ // Return the value of the first cell/row in
    $M_{|\varphi|\times|\hat{\rho}|}$ table
end procedure
```

When the Boolean values of the frozen subformulas are resolved for each time variable $v_1 \ldots v_k \ldots v_{|V|}$ in this order, we have an LTL formula for the highest level subformula where it corresponds to subtree $\theta_{|V|+1}$. To compute the $\bot/\top$ value of the highest set of subformulas we run Lines 22-26 of Algorithm 1. Note that Lines 22-26 are almost identical to Lines 14-18 because the highest set of subformulas is in LTL. The final value that corresponds to the monitoring trace is stored in table entry $M[1,0]$ and it will be returned to the user. The table entry $M[1,0]$ contains the Boolean value of the TPTL specification (φ_1) at sampled index 0.

B. LTL Monitoring

Now we explain how to compute the Boolean values of the LTL subtree. Algorithm 2 is based on [22], and follows Definition 4. Algorithm 1 calls Algorithm 2 at each sample u. Algorithm 2 has the following 5 cases to compute the Boolean values of the corresponding LTL operators:

1) Lines 1-2 for the NOT operation ($\neg$).
2) Lines 3-4 for the AND operation ($\wedge$).
3) Lines 5-6 for the OR operation ($\vee$).
4) Lines 7-12 for the NEXT operation ($\bigcirc$).
5) Lines 13-19 for the UNTIL operation (U).

Algorithm 2 LTL Monitor

Input: $\varphi_j, u, M_{|\varphi|\times|\hat{\rho}|}$; **Output**: $M[j,u]$.

```
procedure COMPUTELTL($\varphi_j, u, M_{|\varphi|\times|\hat{\rho}|}$)
1: if $\varphi_j \equiv \neg\varphi_m$ then
2:     return $\neg M[m,u]$
3: else if $\varphi_j \equiv \varphi_m \wedge \varphi_n$ then
4:     return $M[m,u] \wedge M[n,u]$
5: else if $\varphi_j \equiv \varphi_m \vee \varphi_n$ then
6:     return $M[m,u] \vee M[n,u]$
7: else if $\varphi_j \equiv \bigcirc\varphi_m$ then
8:     if $u = |\hat{\rho}| - 1$ then
9:         return $\bot$
10:    else
11:        return $M[m, u+1]$
12:    end if
13: else if $\varphi_j \equiv \varphi_m U \varphi_n$ then
14:    if $u = |\hat{\rho}| - 1$ then
15:        return $M[n,u]$
16:    else
17:        return $M[n,u] \vee (M[m,u] \wedge M[j,u+1])$
18:    end if
19: end if
end procedure
```

Note that Algorithm 2 (ComputeLTL) is $O(1)$ complexity. Since we can evaluate each frozen subformula $(x.\varphi(x))$ separately because of independent time variables, the time complexity of the algorithm is proportional to the number of time variables and the size of the subformula. On the other hand, for each time sample we instantiate each time variable to convert the TPTL subformula into an LTL subformula in $O(|\hat{\rho}|)$ then run the LTL monitoring algorithm in $O(|\hat{\rho}|)$. As a result, the upper bound on the time complexity of Algorithm 1 is $O(|V| \times |\varphi| \times |\hat{\rho}|^2)$, where $|V|$ is the number of time variables, $|\varphi|$ is the number of subformulas, and $|\hat{\rho}|$ is the number of TSS samples. Both algorithms' correctness proofs are provided in Section VII.

C. Running example

In this section, we utilize our monitoring algorithm to compute the solution for Example 1. First step of the algorithm is the $\top/\bot$ computation of the frozen subformula $y.\psi_1(y)$ which corresponds to subtree θ_1 and is represented in **light-gray** rows of Tables I and II. In Table II, when the time value of y is instantiated to 0, then the value of the time constraint $y \leq 1$ will be resolved for all the samples of i between 0 to 6 according to the following inequality $\tau_i - 0 \leq 1$. Now $\psi_1(0)$ is transformed into LTL and $\psi_1(0)$ is evaluated, i.e., $\psi_1(0) \equiv \top$ (see row φ_9 column τ_0). Then, the time value of y is instantiated to $\tau_1 = 0.3$ and the value of the time constraint $y \leq 1$ will be resolved for all the samples of i between 1 to 6 according to the following inequality $\tau_i - 0.3 \leq 1$. Similarly, $\psi_1(0.3)$ is transformed into LTL and $\psi_1(0.3)$ can be computed, i.e., $\psi_1(0.3) \equiv \top$ (see row φ_9 column τ_1). We continue the computation of $\psi_1(\tau_i)$ with the following instantiation $\tau_2 = 0.7, \ldots, \tau_6 = 1.9$ similar to τ_0. Now $\bot/\top$ values of the frozen subformula $y.\psi_1(y)$ for each time stamp τ_i are available in row φ_9 of Table II.

The Boolean values of subtree θ_1 should be available for higher level subformulas. Therefore, the row φ_9 will be copied to row φ_6 (in Table II both rows have the same color). Now we can continue the second run of the algorithm. The $\top/\bot$ computation of the frozen subformula $x.\psi_2(x)$ which corresponds to subtree θ_2 is represented in **white** rows of Table I and II. In Table II, the time value of x is instantiated to 0, then the value of $\psi_2(0)$ is computed, i.e., $\psi_2(0) \equiv \top$ (see row φ_3 column τ_0). Now, the time value of x is instantiated to $\tau_1 = 0.3$ and the value of $\psi_2(0.3)$ is computed $\psi_2(0.3) \equiv \top$ (see row φ_3 column τ_1). We continue the computation of $\psi_2(\tau_i)$ similarly with $\tau_2 = 0.7 \ldots \tau_6 = 1.9$. Now the $\bot/\top$ values of the frozen subformula $x.\psi_2(x)$ for each time stamp τ_i are available in row φ_3 of Table II. Since the Boolean values of subtree θ_2 should be available for higher level subformulas, the row φ_3 is copied to row φ_2. Finally, we compute $\phi = \Box x.\psi_2(x)$ using Lines 22-26 of Algorithm 1 which corresponds to following: $\phi = \bigwedge_{i=0}^{6} \psi_2(\tau_i) \equiv \bot$

V. Experiments

An implementation of our TPTL monitoring algorithm is provided in the S-TaLiRo testing framework [15]. S-TaLiRo is a Matlab toolbox that uses stochastic techniques to find initial states and inputs to Simulink models which result in trajectories that falsify MTL formulas. With our TPTL off-line monitoring algorithm, S-TaLiRo can evaluate specifications that are more expressive than MTL.

A. Runtime Analysis

We measured the runtime of our TPTL monitoring algorithm using the S-TaLiRo toolbox. The system under test was the Automatic Transmission (AT) model provided by Mathworks as a Simulink demo [19]. We introduced a few modifications to the model to make it compatible with the S-TaLiRo framework, which are explained in [14]. AT has two inputs of Throttle and Brake. The outputs contain two real-valued traces: the rotational speed of the engine ω and the speed of the vehicle v. In addition, the outputs contain one discrete-valued trace *gear* with four possible values.

To provide TPTL specifications, we defined four atomic propositions corresponding to the following predicates:
1) $a_1 \equiv (\omega \geq 4500)$: "rotational speed of the engine $\geq$ 4500"
2) $a_2 \equiv (\omega \leq 1500)$: "rotational speed of the engine $\leq$ 1500"
3) $a_3 \equiv (v \geq 40)$: "speed of the vehicle $\geq$ 40"
4) $a_4 \equiv (v \leq 120)$: "speed of the vehicle $\leq$ 120"
Note that these predicates are chosen to be non-trivial and have meaning in the CPS context. The TPTL formulas are generated based on typical safety reactive response specifications. We generated these TPTL formula patterns to check the runtime with respect to: 1) Size of system trace 2) Number of temporal operators 3) Number of time variables.

We created 18 TPTL formulas that cannot be expressed in MTL. All the specifications have the reactive response pattern: $\Box(a_1 \rightarrow \psi)$ where ψ is categorized in two groups:

1) EA group (ψ_{EA}): contains Eventually/Always specifications with 2, 4 and 8 temporal operators.
2) UR group (ψ_{UR}): contains Until/Release specifications with 2, 4 and 8 temporal operators.

We first chose a ψ specification in LTL from Table III column (LTL template). In Table III, column (#) represents the number of temporal operators for each LTL template. Then, we added time variables to create a TPTL specification. The last column in Table III represents the number of TPTL formulas that we created by adding time constraints on ψ. The time variables that we add to ψ correspond to individual temporal operators. In this case, for ψ_{EA2} we create two TPTL formulas with one and two time variables respectively given as ϕ_1 and ϕ_2:

EA $\quad \phi_1 = \Box(a_1 \rightarrow x.\Diamond(a_2 \wedge \Box(a_3 \vee a_4 \wedge C_x)))$
EA $\quad \phi_2 = \Box(a_1 \rightarrow x.\Diamond(a_2 \wedge C_x \wedge y.\Box(a_3 \vee a_4 \wedge C_y)))$

where C_x and C_y are the corresponding time constraints for x and y. Similarly for ψ_{UR2} we created two TPTL formulas with one and two time variables respectively given as ϕ_1 and ϕ_2:

UR $\quad \phi_1 = \Box(a_1 \rightarrow x.(a_2 U(a_3 R(a_4 \wedge C_x))))$
UR $\quad \phi_2 = \Box(a_1 \rightarrow x.(a_2 U a_4 \wedge C_x \wedge y.(a_3 R(a_4 \wedge C_y)))$

We used a similar method to generate ϕ_3 with one time variable, ϕ_4 with two time variables, and ϕ_5 with four time variables based on ψ_{EA4} and ψ_{UR4} with the total number of six TPTL formulas. Finally, we create eight TPTL formulas based on ψ_{EA8} and ψ_{UR8}. These formulas are ϕ_6, ϕ_7, ϕ_8, ϕ_9 and they are represented in Table IV. Our experiments were conducted on a 64-bit Intel Xeon CPU (2.5GHz) with 64-GB RAM and Windows Server 2012. We used Matlab 2015a and Microsoft Visual C++ 2013 Professional to compile our algorithms' code (in C) using the Matlab mex compiler.

The runtime is provided in Table IV. Each row considers two TPTL formulas in EA or UR configuration. For example, the first column ϕ_1 represents $\Box(a_1 \rightarrow x.\Diamond(a_2 \wedge \Box(a_3 \vee a_4 \wedge C_x)))$ and $\Box(a_1 \rightarrow x.(a_2 U(a_3 R(a_4 \wedge C_x))))$ in EA and UR configurations, respectively. In Table IV the second column (#) represents the number of temporal operators in the corresponding frozen subformula, namely, the number of of temporal operators in $\psi_{EA\#}$ or $\psi_{UR\#}$. The third column ($|V|$) in Table IV represents the number of time variables in $\psi_{EA\#}$ or $\psi_{UR\#}$.

We tested our algorithm with the execution traces of the length 1000, 2000, and 10000. For each TPTL formula, we tested our algorithm 100 times where the AT's throttle input is provided by random signal generator (without brake). We reported the mean value (in **Bold**) and variance of the algorithm's runtime in Table IV. It can be seen that when the length of the trace doubles from $|\hat{\rho}|$=1,000 to $|\hat{\rho}|$=2,000 , the runtime quadruples (see **Mean** values in Table IV). Similarly, when the length of trace increases ten times from $|\hat{\rho}|$=1,000 to $|\hat{\rho}|$=10,000 the runtime increased 100 times (see **Mean** values in Table IV). Now, consider the mean values of ϕ_1 and ϕ_2. The number of time variables in ϕ_1 is one and in ϕ_2 is two. It can be seen that mean values of ϕ_2 are twice as those of ϕ_1. Similarly, comparing ϕ_3 and ϕ_4 and ϕ_5 shows that the runtime is proportional to the number of time variables. Finally, comparing rows ϕ_1 and ϕ_3 and ϕ_6 shows that the runtime relates to the number of temporal operators. The experimental results indicate that the runtime behaves as expected, considering that our algorithm is in $O(|V| \times |\varphi| \times |\hat{\rho}|^2)$.

B. Case Study

In this section, we consider CPS requirements which are impossible to formalize in MTL [4], but we formalize them in TPTL, very easily. The ultimate goal is to run the testing

TABLE II. Computing the Boolean values for $\phi = \Box x.\psi_2(x)$. Boolean values correspond to the final snapshot of Monitoring Table.

φ_i	subformula	$\tau_0 = 0$	$\tau_1 = 0.3$	$\tau_2 = 0.7$	$\tau_3 = 1.0$	$\tau_4 = 1.1$	$\tau_5 = 1.5$	$\tau_6 = 1.9$
φ_1	$\phi = \Box x.\psi_2(x)$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
φ_2	$x.\psi_2(x) \equiv x.\Diamond((x \le 1 \to a) \wedge y.\psi_1(y))$	$\psi_2(0) \equiv \top$	$\psi_2(\tau_1) \equiv \top$	$\psi_2(\tau_2) \equiv \top$	$\psi_2(\tau_3) \equiv \top$	$\psi_2(\tau_4) \equiv \bot$	$\psi_2(\tau_5) \equiv \bot$	$\psi_2(\tau_6) \equiv \bot$
φ_3	$\Diamond((x \le 1 \to a) \wedge y.\psi_1(y))$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
φ_4	$(x \le 1 \to a) \wedge y.\psi_1(y)$	$\bot$	$\bot$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
φ_5	$x \le 1 \to a$	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$
φ_6	$y.\psi_1(y) \equiv y.\Diamond(y \le 1 \to \neg b)$	$\psi_1(0) \equiv \top$	$\psi_1(\tau_1) \equiv \top$	$\psi_1(\tau_2) \equiv \top$	$\psi_1(\tau_3) \equiv \top$	$\psi_1(\tau_4) \equiv \bot$	$\psi_1(\tau_5) \equiv \bot$	$\psi_1(\tau_6) \equiv \bot$
φ_7	$x \le 1$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
φ_8	a	$\bot$	$\bot$	$\top$	$\top$	$\top$	$\bot$	$\bot$
φ_9	$\Diamond(y \le 1 \to \neg b)$	$\top$	$\top$	$\top$	$\top$	$\bot$	$\bot$	$\bot$
φ_{10}	$y \le 1 \to \neg b$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
φ_{11}	$y \le 1$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$
φ_{12}	$\neg b$	$\top$	$\top$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
φ_{13}	b	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\top$	$\top$

TABLE III. Specifications of ψ before adding time variables.

LTL	#	LTL template	TPTLs
ψ_{EA2}	2	$\Diamond(a_2 \wedge \Box(a_3 \vee a_4)$	2
ψ_{EA4}	4	$\Diamond(a_2 \wedge \Box(a_3 \vee a_4 \wedge \psi_{EA2})$	3
ψ_{EA8}	8	$\Diamond(a_2 \wedge \Box(a_3 \vee a_4 \wedge \Diamond(a_2 \wedge \Box(a_3 \vee a_4 \wedge \psi_{EA4}))))$	4
ψ_{UR2}	2	$a_2 U(a_3 R a_4)$	2
ψ_{UR4}	4	$a_2 U(a_3 R(a_4 \wedge \psi_{UR2}))$	3
ψ_{UR8}	8	$a_2 U(a_3 R(a_4 \wedge (a_2 U(a_3 R(a_4 \wedge \psi_{UR4})))))$	4

algorithm on these requirements. Our TPTL monitoring algorithm is provided as add-on to the S-TaLiRo testing framework. S-TaLiRo searches for counterexamples to MTL properties through global minimization of a robustness metric [9]. The robustness of an MTL formula φ is a value that measures how far is the trace from the satisfaction/falsification of φ. This measure is an extension of Boolean values ($\top/\bot$) for representing satisfaction or falsification. A positive robustness value means that the trace satisfies the property and a negative value means that the property is not satisfied. The stochastic search then returns the simulation trace with the smallest robustness value that was found.

To falsify safety requirements in TPTL which are more expressive than MTL, we should use our proposed TPTL monitor that can handle those specifications. Now let us consider the Automatic Transmission (AT) system. It contains the discrete output *gear* signal with four possible values ($gear = 1$, ..., $gear = 4$) which indicate the current gear in the auto-transmission controller. We use four atomic propositions g_1, g_2, g_3, g_4 for each possible gear value, where $(gear = i) \equiv g_i$. Then we define three up-shifting events as follows:

1) $e_1 = g_1 \wedge \bigcirc g_2$ means shift from gear one to gear two.
2) $e_2 = g_2 \wedge \bigcirc g_3$ means shift from gear two to gear three.
3) $e_2 = g_3 \wedge \bigcirc g_4$ means shift from gear three to gear four.

In CPS, it is possible that we need to specify the safety requirement about three or more events in sequence, but the time difference between the first and last event happening should be of importance. In general, these types of specification are impossible to represent in MTL. We provide two very succinct TPTL specifications that can formalize these challenging requirements.

The first requirement is as follows:
"*Always if e_1 happens, then if e_2 happens in future and if e_3 happens in future after e_2, then the duration between e_1 and e_3 should be equal or more than 8.*"

This specification is formalized in the following formula:

$$\Phi_1 = \Box z.(e_1 \to \Box(e_2 \to \Box(e_3 \to z \ge 8)))$$

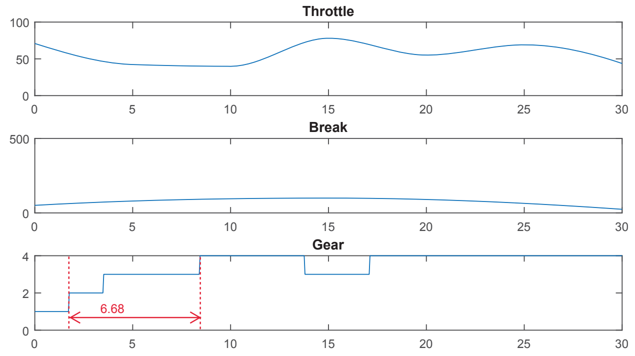

Fig. 2. Falsification of Φ_1 using S-TaLiRo. The duration between e_1 and e_3 is less than 8 seconds.

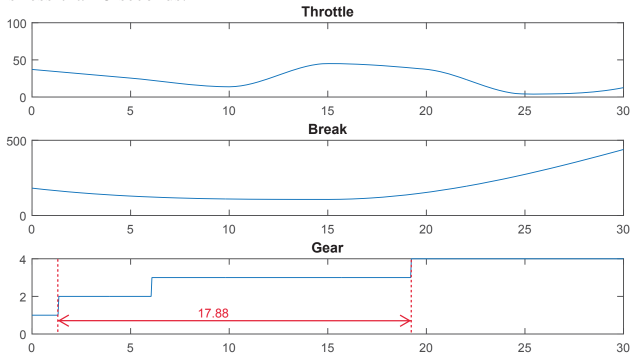

Fig. 3. Falsification of Φ_2 using S-TaLiRo. The duration between e_1 and e_3 is more than 12 seconds.

S-TaLiRo successfully falsified Φ_1 which is represented in Fig. 2. In Fig. 2 the Throttle, Break, and Gear trajectory of the corresponding falsification is presented. It can be seen that the duration between e_1 and e_3 is less that 8. Its actual value is $8.4 - 1.72 = 6.68 < 8$.

The second requirement is as follows:
"*Always if e_1 happens, then e_2 should happen in future, and e_3 should happen in future after e_2, and the duration between e_1 and e_3 should be equal or less than 12.*"

This specification is formalized by the following formula:

$$\Phi_2 = \Box z.(e_1 \to \Diamond(e_2 \wedge \Diamond(e_3 \wedge z \le 12)))$$

In Fig. 3 the Throttle, Break, and Gear trajectories of the falsification of Φ_2 are represented. It can be seen that the duration between e_1 and e_3 is more than 12, its actual value is $19.2 - 1.32 = 17.88 > 12$. This case study shows that S-TaLiRo

TABLE IV. The runtime of Monitoring Algorithm for 18 TPTL formulas. All the values are in seconds.

			$\|\hat{\rho}\|$=1,000				$\|\hat{\rho}\|$=2,000				$\|\hat{\rho}\|$=10,000			
			EA ($\psi_{EA\#}$)		UR ($\psi_{UR\#}$)		EA ($\psi_{EA\#}$)		UR ($\psi_{UR\#}$)		EA ($\psi_{EA\#}$)		UR ($\psi_{UR\#}$)	
ϕ	#	$\|V\|$	**Mean**	Var.	**Mean**	Var.	**Mean**	Var.	**Mean**	Var.	**Mean**	Var.	**Mean**	Var.
ϕ_1	2	1	**0.077**	0.0002	**0.064**	0.000	**0.326**	0.001	**0.250**	0.0013	**8.512**	0.066	**6.427**	0.068
ϕ_2	2	2	**0.151**	0.0005	**0.137**	0.0003	**0.5887**	0.0018	**0.551**	0.002	**14.31**	0.191	**13.67**	0.175
ϕ_3	4	1	**0.142**	0.0003	**0.097**	0.0001	**0.5885**	0.002	**0.382**	0.002	**15.33**	0.232	**10.46**	0.154
ϕ_4	4	2	**0.205**	0.0003	**0.15**	0.0002	**0.871**	0.0032	**0.604**	0.002	**22.9**	0.344	**16.35**	0.24
ϕ_5	4	4	**0.417**	0.0012	**0.38**	0.0004	**1.721**	0.0058	**1.558**	0.007	**46.25**	7.08	**41.2**	1.077
ϕ_6	8	1	**0.227**	0.0001	**0.154**	0.0002	**0.948**	0.005	**0.552**	0.0046	**30.27**	9.708	**17.01**	2.184
ϕ_7	8	2	**0.367**	0.025	**0.235**	0.0011	**1.474**	0.0078	**1.023**	0.0137	**41.59**	2.17	**26.95**	2.204
ϕ_8	8	4	**0.533**	0.0042	**0.437**	0.0013	**2.26**	0.024	**1.751**	0.0115	**66.13**	34.36	**48.95**	8.857
ϕ_9	8	8	**1.145**	0.025	**1.093**	0.0066	**4.9**	0.0391	**4.346**	0.1413	**137**	220	**124.6**	184

can be used for the falsification problem of challenging TPTL requirements. The method we propose in this work opens the possibility for CPS off-line monitoring of very complex specifications in TPTL using an efficient algorithm.

VI. Conclusions and Future works

In this paper, we provide an efficient polynomial time algorithm for a practical subset of TPTL specifications. We show that very complex specifications can be succinctly represented in this TPTL subset. In addition, we can combine full TPTL with a bounded number of time variables with our suggested algorithm to test the specifications that have an arbitrary number of independent time variables and full TPTL with limited number of time variables. Finally, our method can help CPS developers to efficiently test requirements that cannot be expressed in MTL.

Acknowledgments: This research was partially funded by NSF awards CNS-1350420 and CNS-1319560.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] R. Alur and T. A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[3] Y. S. R. Annapureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[4] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. *Inf. Comput.*, 208(2):97–116, 2010.

[5] M. Chai and H. Schlingloff. A rewriting based monitoring algorithm for TPTL. In *Proceedings of the 22nd International Workshop on Concurrency, Specification and Programming, Warsaw, Poland*, pages 61–72, 2013.

[6] J. V. Deshmukh, R. Majumdar, and V. S. Prabhu. Quantifying conformance using the skorokhod metric. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, pages 234–250, 2015.

[7] A. Donze. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[8] A. Donze, T. Ferrre, and O. Maler. Efficient robust monitoring for STL. In *Proceedings of the 25th International Conference on Computer Aided Verification*, CAV, pages 264–279, Berlin, Heidelberg, 2013. Springer-Verlag.

[9] G. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, 2009.

[10] G. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using S-TaLiRo. In *Proceedings of the American Control Conference*, 2012.

[11] S. Feng, M. Lohrey, and K. Quaas. Path checking for MTL and TPTL over data words. In *Developments in Language Theory - 19th International Conference, DLT 2015, Liverpool, UK, July 27-30, 2015, Proceedings.*, pages 326–339, 2015.

[12] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of the Fifteenth IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification XV*, pages 3–18, London, UK, UK, 1996.

[13] J. Håkansson, B. Jonsson, and O. Lundqvist. Generating online test oracles from temporal logic specifications. *STTT*, 4(4):456–471, 2003.

[14] B. Hoxha, H. Abbas, and G. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.

[15] B. Hoxha, H. Bach, H. Abbas, A. Dokhanchi, Y. Kobayashi, and G. Fainekos. Towards formal specification visualization for testing and monitoring of cyber-physical systems. In *Int. Workshop on Design and Implementation of Formal Tools and Systems*. October 2014.

[16] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[17] K. J. Kristoffersen, C. Pedersen, and H. R. Andersen. Runtime verification of timed LTL using disjunctive normalized equation systems. In *Proceedings of the 3rd Workshop on Run-time Verification*, volume 89 of *ENTCS*, pages 1–16, 2003.

[18] N. Markey and J.-F. Raskin. Model checking restricted sets of timed paths. *Theor. Comput. Sci.*, 358(2):273–292, Aug. 2006.

[19] MathWorks. Modeling an automatic transmission controller, available at: http://www.mathworks.com/help/simulink/examples/modeling-an-automatic-transmission-controller.html.

[20] J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, pages 1–13, 2008.

[21] J.-F. Raskin. Logics, automata and classical theories for deciding real-time. *Ph.D. Thesis, University of Namur, Belgium*, 1999.

[22] G. Rosu and K. Havelund. Synthesizing dynamic programming algorithms from linear temporal logic formulae. Technical report, Research Institute for Advanced Computer Science (RIACS), 2001.

VII. Appendix

In this section, we will prove the correctness of Algorithms 1 and 2. Our method first transforms the TPTL formula into LTL formula using Algorithm 1. Then it uses the dynamic programming method for monitoring LTL using Algorithm 2.

A. Proof of the correctness of Algorithm 1

Theorem 1: Given an encapsulated TPTL formula φ, and a finite TSS $\hat{\rho}$, after the execution of Algorithm 1 the returned value is:

$$M[1,0] = \top \text{ iff } (\hat{\rho}, 0, \mathbf{0}) \models \varphi$$

To prove this theorem, we must show that the Boolean value of the subformulas that are computed using Algorithm 1, follows the TPTL semantics in Definition 4. Since Algorithm 1 does not evaluate propositional and temporal operators, their corresponding proof will be provided in Section VII-B.

According to the TPTL semantics in Definition 4, for each freeze time operation $x.\varphi(x)$, and for each time stamp τ_i we must instantiate the time variable x with the value of τ_i. This instantiation enables us to evaluate time constraints and transform TPTL to LTL. The loop of Lines 2-21 is the main loop of Algorithm 1 which instantiates each variable v_k with each time sample τ_t in Line 3.

Lemma 3: The loop invariant of Algorithm 1 is as follows:

$$\forall j, k, t \text{ where } \varphi_j \equiv v_k.\varphi_i, 0 \leq t < |\hat{\rho}| :$$

$$M[j,t] = \top \text{ iff } (\hat{\rho}, t, \varepsilon) \models v_k.\varphi_i$$

We use induction to prove the loop invariant of Algorithm 1.

Base: If $|V| = 0$, then formula is in LTL and algorithm does not enter the to loop of Lines 2-21 (only executes Lines 22-26). The proof of LTL is provided in Section VII-B.

Induction Hypothesis: We assume for all v_l, where $l < k$ the invariant holds. In other words

$$\forall j, l < k, t \text{ where } \varphi_j \equiv v_l.\varphi_i, 0 \leq t < |\hat{\rho}| :$$

$$M[j,t] = M[\theta_l.parent, t] = \top \text{ iff } (\hat{\rho}, t, \varepsilon) \models v_l.\varphi_i$$

Induction Step: To show the correctness for the case of v_k, we prove that Algorithm 1 correctly transform TPTL into LTL. Then we apply the correctness of LTL (See Section VII-B) to establish the correctness of invariant considering v_k. Thus, we consider two cases that instantiate and evaluate v_k and show that Algorithm 1 follows the semantics in Definition 4. According to I.H. and since time variables are independent, we can correctly consider frozen subformulas of φ_i as $\top/\bot$. As a result, we will conclude that φ_i is in LTL.

Case of $v_k.\varphi_i$**:**
Consider the semantics of the freeze operator in Definition 4:

$$(\hat{\rho}, t, \varepsilon) \models v_k.\varphi_i \text{ iff } (\hat{\rho}, t, \varepsilon[v_k := \tau_t]) \models \varphi_i$$

According to this semantics, the freeze operation "v_k." first assigns a new value to the variable ($v_k := \tau_t$). Then the $\top/\bot$ value of $v_k.\varphi_i \equiv \varphi_j$ will be resolved to the same $\top/\bot$ value of φ_i (with the new environment update). Therefore, for each variable assignment ($v_k := \tau_t$), we first update the environment variables (Algorithm 1, Line 3), and then copy the φ_i's $\top/\bot$ value into $v_k.\varphi_i$'s corresponding row (Algorithm 1, Line 19).

Since each time variable v_k is independent, we create the subtree (set) θ_k corresponding to the subformulas of $v_k.\varphi_i(v_k)$ (see Section III-A). To evaluate $v_k.\varphi_i(v_k)$, we must first instantiate variable v_k for each time stamp $\tau_0 \dots \tau_{|\hat{\rho}|-1}$. This instantiation is considered in Line 2 of Algorithm 1 for time variable v_k and for each sample of time $0 \dots (|\hat{\rho}| - 1)$ in Line 3 of Algorithm 1. Now we must copy the resulting $\top/\bot$ value from φ_i back to $v_k.\varphi_i$. The row corresponding to $\theta_k.root$ contains the $\top/\bot$ value of φ_i which is the root of θ_k subtree. This values must be copied to the row $\theta_k.parent$ which is the parent of subtree θ_k and it corresponds to φ_j (Algorithm 1, Line 19).

Case of $v_k \sim r$**:**
Consider the semantics of time constraints in Definition 4:

$$(\hat{\rho}, u, \varepsilon) \models v_k \sim r \text{ iff } (\tau_u - \varepsilon(v_k)) \sim r$$

In the above semantics, $\varepsilon(v_k)$ corresponds to the frozen value of the time variable v_k (environment of v_k). In the previous case for $v_k.\varphi_i$, we mentioned that we should instantiate v_k at each time stamp $\tau_0 \dots \tau_{|\hat{\rho}|-1}$. According to semantics in Definition 4, each freeze operator assigns the environment variable for the current and future samples of time t:

$$(\hat{\rho}, t, \varepsilon) \models v_k.\varphi_i \text{ iff } (\hat{\rho}, t, \varepsilon[v_k := \tau_t]) \models \varphi_i$$

Which means that the environment updates $\varepsilon[x := \tau_t]$ are observable for the current and the future samples ($t \leq u$). Therefore, after we instantiated variable v_k at each time stamp τ_t, the environment update will affect all the samples u between $t \leq u \leq |\hat{\rho}| - 1$. As a result, the time constraint $v_k \sim r$ must be updated for all future samples of $t \leq u \leq |\hat{\rho}| - 1$ for $\varepsilon[v_k := \tau_t]$ instantiation.

Lines 4-13 of Algorithm 1 follow the above discussion. Namely, for time variable v_k, we instantiate each time stamp τ_t (Line 3), the time constraints of current/future samples are evaluated according to the frozen time stamp τ_t. Actual evaluation happens in the Line 7 of Algorithm 1, where $(\tau_u - \tau_t) \sim r$ follows the semantic $(\tau_u - \varepsilon(v_k)) \sim r$ for each environment assignment of $\varepsilon[v_k := \tau_t]$. Lines 14-18 of Algorithm 1 will evaluate the LTL formula $\varphi_i(\tau_t)$.

So far, we transformed TPTL $v_k.\varphi_i(v_k)$ into LTL $\varphi_i(\tau_t)$ for each time stamp τ_t. Now we can prove that the loop invariant of Algorithm 1 holds for v_k.

Proof: We will prove the Induction Step by assuming the correctness of LTL formula φ_i according to Section VII-B:

$$\forall i, t, \varepsilon \text{ where } \varphi_i \subset LTL, 0 \leq t < |\hat{\rho}|$$

$$M[i,t] = \top \text{ iff } (\hat{\rho}, t, \varepsilon) \models \varphi_i$$

Since for each θ_k, $i = \theta_k.root$ is the index of the highest LTL, $M[\theta_k.root, t]$ will also contain the correct $\top/\bot$ value, therefore $M[i,t] = M[\theta_k.root, t] = \top$ iff $(\hat{\rho}, t, \varepsilon) \models \varphi_i(v_k = \tau_t)$ iff $(\hat{\rho}, t, \varepsilon[v_k := \tau_t]) \models \varphi_i$

Since in Line 19 $M[\theta_k.parent, t] \leftarrow M[\theta_k.root, t]$
and $j = \theta_k.parent$ we have
$M[j,t] \leftarrow M[i,t]$, as a result

$$M[j,t] = M[\theta_k.parent, t] = \top \text{ iff } (\hat{\rho}, t, \varepsilon) \models v_k.\varphi_i \equiv \varphi_j$$

■

B. Proof of the correctness of Algorithm 2

LTL formulas consider only propositional and temporal operators; therefore, the time variables' environment (ε) is not affected by Algorithm 2. Since time variables do not change during Algorithm 2, we assume that Algorithm 2 considers time constraints as $\top/\bot$ values since they are already evaluated in Algorithm 1. In this section, we prove that the output of Algorithm 2 corresponds to the correct evaluation of the LTL subformula φ_j at sample instance u based on Definition 4.

In essence, we will prove $M[j,u] = \top$ if $(\hat{\rho}, u, \varepsilon) \models \varphi_j$ and similarly $M[j,u] = \bot$ if $(\hat{\rho}, u, \varepsilon) \not\models \varphi_j$. For the proof of Algorithm 2, we use induction:

Base: In Section IV, we mentioned that in Line 1 of Algorithm 1 the corresponding values for atomic propositions are stored in the monitoring table. In essence, for each $a \in AP$, and for each time stamp τ_u, we save the following values in the monitoring table entry $M[a_{index}, u]$, where a_{index} is the index of atomic proposition a in the monitoring table $M_{|\varphi|\times|\hat{\rho}|}$:

1) $M[a_{index}, u] \leftarrow \top$ if $a \in \sigma_u$ if $(\hat{\rho}, u, \varepsilon) \models a$
2) $M[a_{index}, u] \leftarrow \bot$ if $a \notin \sigma_u$ if $(\hat{\rho}, u, \varepsilon) \not\models a$

Since evaluation of predicates is independent of the time variables' environment (ε) the above cases are always satisfied for all sample instances u and all environments ε. As a result, every table entry corresponding to a predicate, correctly reflects the satisfaction of the predicate with respect to the state trace $\hat{\sigma}$ and the environment ε. Similarly, the table entries for constant Boolean values ($\top/\bot$) are trivially correct.

Induction Hypothesis: Algorithm 1 updates the values of Table from right to left, i.e., for the samples with indexes $|\hat{\rho}| - 1$ down to 0. This is because we resolve temporal operators looking into the future. Namely, if the Boolean value in the next samples of time are resolved, then we can resolve the Boolean evaluation for the current sample of time. For the Induction Hypothesis, we assume the table entries for the proper subformulas of φ_j at the same or future samples contain the correct $\top/\bot$, i.e, we assume that

$$\forall \varphi_k \subset \varphi_j, \forall v \geq u, M[k,v] = \top \text{ iff } (\hat{\rho}, v, \varepsilon) \models \varphi_k$$

And also for the same subformula (φ_j), we assume the table entries for all the future samples contain the correct $\top/\bot$ values as follows:

$$\forall v > u, M[j,v] = \top \text{ iff } (\hat{\rho}, v, \varepsilon) \models \varphi_j$$

Induction Step: For the induction step we consider five cases of φ_j:

Case 1: $\varphi_j \equiv \neg\varphi_m$:
Consider $M[j,u] \leftarrow \neg M[m,u]$ (Algorithm 2, Line 2).
According to Definition 4: $(\hat{\rho}, u, \varepsilon) \models \neg\varphi_m$ iff $(\hat{\rho}, u, \varepsilon) \not\models \varphi_m$
Based on IH: $M[m,u] = \bot$ iff $(\hat{\rho}, u, \varepsilon) \not\models \varphi_m$ iff (based on Def. 4) $(\hat{\rho}, u, \varepsilon) \models \neg\varphi_m \equiv \varphi_j$
Therefore, $M[j,u] = \neg M[m,u] = \neg\bot$ iff $(\hat{\rho}, u, \varepsilon) \not\models \varphi_m$ iff $(\hat{\rho}, u, \varepsilon) \models \neg\varphi_m \equiv \varphi_j$
As a result $M[j,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_j$

Case 2: $\varphi_j \equiv \varphi_m \wedge \varphi_n$:
Consider $M[j,u] \leftarrow M[m,u] \wedge M[n,u]$ (Algorithm 2, Line 4).
According to Definition 4: $(\hat{\rho}, u, \varepsilon) \models \varphi_m \wedge \varphi_n$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ and $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
Based on IH: $M[m,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ and $M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
We know that, $M[m,u] \wedge M[n,u] = \top$ iff $M[m,u] = \top$ and $M[n,u] = \top$
Thus, $M[m,u] \wedge M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ and $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
Therefore, $M[m,u] \wedge M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m \wedge \varphi_n \equiv \varphi_j$
As a result $M[j,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_j$

Case 3: $\varphi_j \equiv \varphi_m \vee \varphi_n$:
Consider $M[j,u] \leftarrow M[m,u] \vee M[n,u]$ (Algorithm 2, Line 6).
According to Definition 4: $(\hat{\rho}, u, \varepsilon) \models \varphi_m \vee \varphi_n$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ or $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
Based on IH: $M[m,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ and $M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
We know that, $M[m,u] \vee M[n,u] = \top$ iff $M[m,u] = \top$ or $M[n,u] = \top$
Thus, $M[m,u] \vee M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ or $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
Therefore, $M[m,u] \vee M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m \vee \varphi_n \equiv \varphi_j$
As a result $M[j,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_j$

Case 4: $\varphi_j \equiv \bigcirc\varphi_m$
Consider $M[j,u] \leftarrow M[m,u+1]$ if $u < |\hat{\rho}| - 1$ (Line 11) and $M[j,u] \leftarrow \bot$ otherwise (Line 9 of Algorithm 2).
According to Definition 4 we have two cases:
Case 4.1) $u < (|\hat{\rho}| - 1)$:
$(\hat{\rho}, u, \varepsilon) \models \bigcirc\varphi_m$ iff $(\hat{\rho}, u+1, \varepsilon) \models \varphi_m$
Based on IH: $M[m,u+1] = \top$ iff $(\hat{\rho}, u+1, \varepsilon) \models \varphi_m$ iff $(\hat{\rho}, u, \varepsilon) \models \bigcirc\varphi_m \equiv \varphi_j$
As a result $M[j,u] = M[m,u+1] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_j$
Case 4.2) $u = |\hat{\rho}| - 1$:
by Definition 4, $(\hat{\rho}, u, \varepsilon) \not\models \bot$
Line 9 of Algorithm 2 similarly assigns $M[j,u] \leftarrow \bot$

Case 5: $\varphi_j \equiv \varphi_m U \varphi_n$
According to [12], Until operation can be simplified according to following equivalence relation:

$$\phi U \psi \equiv \psi \vee (\phi \wedge \bigcirc(\phi U \psi))$$

In other words, we need to consider current value of $\bigcirc(\phi U \psi)$ (future value of $\phi U \psi$ at the next sample) and use the current values of ϕ and ψ to resolve and evaluate $\phi U \psi$ at the current sample using equation $\psi \vee (\phi \wedge \bigcirc(\phi U \psi))$. Algorithm 2 considers two case for $\varphi_j \equiv \varphi_m U \varphi_n \equiv \varphi_n \vee (\varphi_m \wedge \bigcirc(\varphi_m U \varphi_n))$:
Case 5.1) $u < (|\hat{\rho}| - 1)$:
Now consider the update of $M[j,u] \leftarrow M[n,u] \vee (M[m,u] \wedge M[j,u+1])$ according to Line 17 of Algorithm 2.
Based on IH: $M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_n$ and $M[m,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ and
$M[j,u+1] = \top$ iff $(\hat{\rho}, u+1, \varepsilon) \models \varphi_j$ iff $(\hat{\rho}, u, \varepsilon) \models \bigcirc\varphi_j$
According to Case 2 (Conjunction) $M[m,u] \wedge M[j,u+1] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m$ and $(\hat{\rho}, u, \varepsilon) \models \bigcirc\varphi_j$
Therefore, $M[m,u] \wedge M[j,u+1] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_m \wedge \bigcirc\varphi_j$
We know that, $M[j,u] = \top$ iff $M[n,u] = \top$ or $M[m,u] \wedge M[j,u+1] = \top$
According to Case 3 (Disjunction) $M[j,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_n$ or $(\hat{\rho}, u, \varepsilon) \models \varphi_m \wedge \bigcirc\varphi_j$
As a result, $M[j,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_n \vee (\varphi_m \wedge \bigcirc\varphi_j)$
Case 5.2) $u = |\hat{\rho}| - 1$:
According to Case 4.2 for Next operator: $(\hat{\rho}, u, \varepsilon) \not\models \bot$
This implies that $\varphi_j \equiv \varphi_n \vee (\varphi_m \wedge \bot) \equiv \varphi_n \vee \bot \equiv \varphi_n$
Now consider the update of $M[j,u] \leftarrow M[n,u]$ according to Line 15 of Algorithm 2.
Based on IH: $M[n,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_n$
Therefore after the assignment, $M[j,u] = \top$ iff $(\hat{\rho}, u, \varepsilon) \models \varphi_j$