Linear-time Temporal Logic with Event Freezing Functions*

Stefano Tonetta
FBK-irst
tonettas@fbk.eu

Formal properties represent a cornerstone of the system-correctness proofs based on formal verification techniques such as model checking. Formalizing requirements into temporal properties may be very complex and error prone, due not only to the ambiguity of the textual requirements but also to the complexity of the formal language. Finding a property specification language that balances simplicity, expressiveness, and tool support remains an open problem in many real-world contexts.

In this paper, we propose a new temporal logic, which extends First-Order Linear-time Temporal Logic with Past adding two operators "at next" and "at last", which take in input a term and a formula and represent the value of the term at the next state in the future or last state in the past in which the formula holds. We consider different models of time (including discrete, dense, and super-dense time) and Satisfiability Modulo Theories (SMT) of the first-order formulas. The "at next" and "at last" functions can be seen as a generalization of Event-Clock operators and can encode some Metric Temporal operators also with counting. They are useful to formalize properties of component-based models because they allow to express constraints on the data exchanged with messages at different instants of time. We provide a simple encoding into equisatisfiable formulas without the extra functional symbols. We implement a prototype tool support based on SMT-based model checking.

1 Introduction

The specification of properties is a fundamental step in the formal verification process. System requirements must be captured by formal properties, typically using logic formulas. However, this is often a complex activity and may become a blocking factor for an industrial adoption of the formal techniques. The informal requirements are quite ambiguous but also the complexity of the target logic may be the source of errors in the specification. Finding a property specification language that balances simplicity, expressiveness, and analysis tool support remains an open problem in many real-world contexts.

One of the most popular logics used in computer science to specify properties for formal verification is Linear-time Temporal Logic (LTL) [27]. The model of time is typically discrete and models are discrete, linear sequences of states. We consider First-Order LTL [25] with future as well as past operators [24]. Thus, the system state is described by individual variables and first-order functions/predicates can describe their relationship. In the spirit of Satisfiability Modulo Theories (SMT) [5], the formulas are interpreted modulo a background first-order theory as in [19] (here, we restrict to a quantifier-free fragment with all signature's symbols rigid). Efficient SMT-based model checking techniques can be used to verify temporal properties on systems described with first-order formulas (see, e.g, [12, 16]).

In the case of real-time systems, LTL is interpreted over a dense model of time or super-dense (dense time with possible sequences of instantaneous events, as needed for example for asynchronous real-time systems). When considering real models of time, it becomes natural to have constraints on the

*This work has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No. 700665 (project CITADEL).

P. Bouyer, A. Orlandini & P. San Pietro (Eds.): 8th Symposium on
Games, Automata, Logics and Formal Verification (GandALF'17)
EPTCS 256, 2017, pp. 195–209, doi:10.4204/EPTCS.256.14

time elapsing between different events. Therefore, LTL has been extended either with clocks/freezing operators as in TPTL [4] or with metric operators as in [22, 1, 30]. Again, these extensions can be combined with first-order logic (e.g., to represent message passing [23] or for monitoring specification in [6]).

When a system or component is seen as a black box, the properties must be specified in terms of the observable variables or messages exchanged with the system environment. This is for example the case of properties of monitors (which trigger alarms based on some condition on the observed variables/messages) or contract-based specifications (which formalize the assumptions/guarantees of components independently of the implementation). In these cases, the properties must capture the relationship between the observable variables at different points of time, without referring to internal variables that store the corresponding values. It is therefore necessary to have suitable mechanisms to refer to the value of variables at different points of time. Instead of enriching the specification language with registers as in register automata [17] to explicitly store the value of variables in an operational-style specification, we adopt a more declarative style with functions that directly return the value of variables at the next or last state in which a formula will be/was true.

More specifically, we extend the quantifier-free fragment of First-Order Linear-Time Temporal Logic with Past operators adding "at next" $u@\tilde{F}(\phi)$ and "at last" $u@\tilde{P}(\phi)$ functional symbols, which are used to represent the value of a term u at the next state in the future or at the last state in the past in which a formula ϕ holds. For example, the formula $G(alarm \leftrightarrow x@\tilde{P}(read) = x@\tilde{P}^2(read))$ says that *alarm* is true iff in the last two points in which *read* was true the variable x had the same value. We consider different models of time, including discrete, dense, and super-dense time. In the dense time setting, the definition has to take into account that a minimum time point may not exist because a formula may be true on open intervals. The "at next" and "at last" functions can be seen as a generalization of Event-Clock Temporal Logic (ECTL) operators [29, 20, 30] (which, on turn, are the logical counterpart of event clocks [2]) and can encode some Metric Temporal Logic (MTL) operators [22] also with counting [21, 26]. They are useful to formalize properties of component-based models because they allow us to express constraints on the data exchanged with messages at different instants of time. We provide a simple encoding of the formulas with these extra functional symbols into equisatisfiable formulas without them. We implemented a prototype tool support based on SMT-based model checking.

The natural alternative to the logic we proposed would be to use registers and freezing quantifiers as in [17] and TPTL. Despite freezing quantifiers provide a higher expressiveness (also with respect to MTL [8]), they are not so common in industrial applications (at least compared to LTL and MTL), either because they are less intuitive to use or they lack tool support.

The main contributions of the paper are the following. First, we identify an extension of LTL that can express interesting properties relating variables at different points of time. Second, we define the new operators in a very rich setting that includes first-order constraints, past operators, dense and super-dense semantics; this gives also a uniform treatment of LTL satisfiability modulo theories in the case of real time models. Third, we provide a prototype tool support that effectively proves interesting properties, while many logics in the real-time setting lack of tool support.

The rest of the paper is organized as follows: Section 2 introduces the considered time models, LTL satisfiability modulo theories, and its extension with metric operators; Section 3 defines the extension with the new event freezing functions; Section 4 describes the satisfiability procedure; Section 5 presents some preliminary experimental results; finally, Section 6 concludes the paper and draws directions for future work.

2 Background

2.1 Time models

$\mathbb{R}_0^+$ is the set of non-negative real numbers. A time interval is a convex subset of $\mathbb{R}_0^+$. The left endpoint of an interval I is denoted by $l(I)$, while the right endpoint by $r(I)$. Two intervals I and I' are almost adjacent iff $r(I) = l(I)$ (so they may overlap in at most one point). A singular interval is an interval in the form $[a,a]$ for some $a \in \mathbb{R}_0^+$. A time interval sequence is a sequence $I_0, I_1, I_2, \ldots$ of time intervals such that, for all $i \geq 0$, I_i and I_{i+1} are almost adjacent and $\bigcup_{i \geq 0} I_i = \mathbb{R}_0^+$.

We consider different models of time [3, 4]. A time model is a structure $\tau = \langle T, <, \mathbf{0}, v \rangle$ with a domain T, a total order $<$ over T, a minimum element $\mathbf{0} \in T$, and a function $v : T \to \mathbb{R}_0^+$ that represents the real time of a time point in T. The v function is used instead of a distance (e.g., as in [22]) to treat the weakly-monotonic case in a more uniform way. A *time point* is an element of T. In particular, we consider the following models:

- discrete time models where $T = \mathbb{N}$, $\mathbf{0}$ and $<$ are the standard zero and order over natural numbers, $v(0) = 0$ and $v(0), v(1), v(2), \ldots$ is a non-decreasing divergent sequence (this is also called the pointwise semantics; we use these models also for discrete-time LTL ignoring these real-time timestamps);
- dense (strictly-monotonic) time model where $T = \mathbb{R}_0^+$, $\mathbf{0}$ and $<$ are the standard zero and order over the real numbers, and v is the identity function;
- super-dense (weakly-monotonic) time models where 1) $T \subset \mathbb{N} \times \mathbb{R}_0^+$ such that the sequence of sets $I_0, I_1, I_2, \ldots$ where, for all $i \geq 0$, the set $I_i := \{t \mid \langle i,t \rangle \in T\}$, is a time interval sequence (thus subsequent intervals can overlap in at most one point), 2) $\langle i,t \rangle < \langle i',t' \rangle$ iff $i < i'$ or $i = i'$ and $t < t'$, 3) $\mathbf{0} = \langle 0,0 \rangle \in \mathbb{N} \times \mathbb{R}_0^+$, and 4) $v(\langle i,t \rangle) = t$.

2.2 First-Order Linear-time Temporal Logic

We consider First-Order Linear-time Temporal Logic with Past Operators, which we refer to for simplicity as LTL.

Given a first-order signature Σ and a set V of variables, we define the syntax of Σ-formulas as follows:

$$\phi := p(u, \ldots, u) \mid \phi \wedge \phi \mid \neg\phi \mid \phi \tilde{U} \phi \mid \phi \tilde{S} \phi$$
$$u := c \mid x \mid f(u, \ldots, u)$$

where p is a predicate symbol of Σ, u is a term, f is a functional symbol of Σ, c is a constant symbol of Σ, and x is a variable in V.

Σ-formulas are interpreted by a first-order structure interpreting the symbols in Σ and assignments to variables that vary along time. More specifically, a state $s = \langle M, \mu \rangle$ is given by a first-order structure M and an assignment μ of variables of V into the domain of M. Given a state $s = \langle M, \mu \rangle$ and a symbol c of Σ or variable $x \in V$ we use $s(c)$ to denote the interpretation of c in M and $s(x)$ to denote the value $\mu(x)$ assigned by μ to x. Given M, let V^M be the set of states with first-order structure M. A trace $\sigma = \langle M, \tau, \overline{\mu} \rangle$ is given by a first-order structure M, a time model τ, and a mapping $\overline{\mu}$ from the domain of τ into V^M. Given a trace $\sigma = \langle M, \tau, \overline{\mu} \rangle$ and $t \in \tau$, we denote by $\sigma(t)$ the state $\langle M, \overline{\mu}(t) \rangle$.

We assume to be given a Σ first-order theory $\mathscr{T}$. Given a Σ first-order structure M, an assignment μ to variables of V, and a Σ first-order formula ϕ over V, we use the standard notion of $\langle M, \mu \rangle \models_{\mathscr{T}} \phi$. In the rest of the paper, we omit the first-order signature Σ and theory $\mathscr{T}$ for simplicity.

In our definition of trace, the first-order structure M is shared by all time points, meaning that the interpretation of the symbols in the signature Σ is rigid, does not vary with time. However, note that the interpretation of symbols may not be "fixed" by the background theory. These"uninterpreted" symbols are also called *parameters*. For example, the signature Σ can include the symbols of the theory of reals (including the constants 0 and 1) and an additional constant symbol p, whose value is not determined by the theory but does not vary with time (thus, p is a parameter).

Given a trace $\sigma = \langle M, \tau, \overline{\mu} \rangle$, a time point t of τ, and a Σ formula ϕ, we define $\sigma, t \models \phi$ recursively on the structure of ϕ.

$$\sigma, t \models p \text{ iff } \sigma(t) \models p$$
$$\sigma, t \models \phi_1 \wedge \phi_2 \text{ iff } \sigma, t \models \phi_1 \text{ and } \sigma, t \models \phi_2$$
$$\sigma, t \models \neg\phi \text{ iff } \sigma, t \not\models \phi$$
$$\sigma, t \models \phi_1 \tilde{U} \phi_2 \text{ iff there exists } t' > t, \sigma, t' \models \phi_2 \text{ and for all } t'', t < t'' < t', \sigma, t'' \models \phi_1$$
$$\sigma, t \models \phi_1 \tilde{S} \phi_2 \text{ iff there exists } t' < t, \sigma, t' \models \phi_2 \text{ and for all } t'', t' < t'' < t, \sigma, t'' \models \phi_1$$

Note that we are using the strict version of the "until" and "since" operators, where both arguments are required to hold in points strictly greater or less than the current time.

Finally, $\sigma \models \phi$ iff $\sigma, \mathbf{0} \models \phi$. We say that ϕ is satisfiable iff there exists σ such that $\sigma \models \phi$. We say that ϕ is valid iff, for all σ, $\sigma \models \phi$.

We use the following standard abbreviations:

$$\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2) \qquad \top := p \vee \neg p$$
$$\bot := \neg\top \qquad \phi_1 U \phi_2 := \phi_2 \vee (\phi_1 \wedge \phi_1 \tilde{U} \phi_2)$$
$$F\phi := \top U \phi \qquad G\phi := \neg(F\neg\phi)$$
$$\phi_1 S \phi_2 := \phi_2 \vee (\phi_1 \wedge \phi_1 \tilde{S} \phi_2) \qquad P\phi := \top S \phi$$
$$H\phi := \neg(P\neg\phi)$$

As usual in many works on real-time temporal logics (e.g., [1, 28]), we assume the "finite variability" of traces, i.e., that the evaluation of predicates by a trace changes from true to false or vice versa only finitely often in any finite interval of time. This can be lifted to temporal formulas in the sense that temporal operators preserve the finite variability property (as proved for example in [1]). Formally, we say that a trace σ is *fine* for ϕ in a time interval I iff for all $t, t' \in I$, $\sigma, t \models \phi$ iff $\sigma, t' \models \phi$. A trace σ has the *finite variability* property iff for every formula ϕ there exists a sequence of points $t_0, t_1, t_2 \ldots$ of σ such that σ is *fine* for ϕ in every interval (t_i, t_{i+1}), for $i \geq 0$. In the following, we assume that traces have the finite variability property.

2.3 Next Operator and Function

Since $\tilde{U}$ is the strict version of the "until" operator, we can write the standard X as abbreviation:

$$X\phi := \bot \tilde{U} \phi$$

X is well defined in the different time models, also in the case of dense or super-dense time. In the case of weakly-monotonic time, $X\phi$ can be true only on a discrete step (i.e., in $\langle n, t \rangle$ if $\langle n+1, t \rangle$ is also in T). In the case of strictly-monotonic time, $X\phi$ is always false.

With the strict until, we can also define a continuous counterpart of the X operator:

$$\tilde{X}\phi := \phi \tilde{U} \top \wedge \neg X \top$$

Note that $X\top$ is true in all and only in discrete steps. Thus, $\tilde{X}\phi$ is always false in the case of discrete time, while in the case of dense time it is true in the time points with a right neighborhood satisfying ϕ. In the super-dense time case, $\tilde{X}\phi$ is false in the discrete steps, while in other time points it is true if a right neighborhood satisfies ϕ. Note that this is a variant of the more standard $\phi\tilde{U}\top$ formula, which has been studied for example in [18]. However, it was considered only in the dense time case. Here, we added $\neg X\top$, because it will be more convenient in the super-dense time case.

Similarly, we define the "yesterday" operators $Y\phi := \bot\tilde{S}\phi$ and $\tilde{Y}\phi := \phi\tilde{S}\top \wedge \neg Y\top$. We also define the weaker version of "yesterday" that is true in the initial state: $Z\phi := (Y\top \vee \tilde{Y}\top) \rightarrow Y\phi$ and $\tilde{Z}\phi := (Y\top \vee \tilde{Y}\top) \rightarrow \tilde{Y}\phi$.

In the discrete-time setting, we often use also the functional counterpart of X, here denoted by *next* [25]. Given a term u, the interpretation of $next(u)$ in a trace σ at the time point t is equal to the value of u assigned by σ at the time point $t+1$. "next" does not typically have a counterpart in the dense time case. Let LTL-next be the extension of LTL with the *next* function (with discrete time).

2.4 Metric Temporal Operators

In this section, we define some extensions of LTL that use metric operators to constrain the time interval between two or more points. We give a general version in the first-order setting that include also weakly-monotonic time and parametric intervals.

Metric Temporal Logic (MTL) formulas are built with the following grammar:

$$\begin{aligned}
&\phi := p(u,\ldots,u) \mid \phi \wedge \phi \mid \neg\phi \mid \phi\tilde{U}_I\phi \mid \phi\tilde{S}_I\phi\\
&I := [cu,cu] \mid (cu,cu] \mid [cu,cu) \mid (cu,cu) \mid [cu,\infty) \mid (cu,\infty)\\
&cu := c \mid f(cu,\ldots,cu)
\end{aligned}$$

where the terms u are defined as before and cu are terms that do not contain variables. Thus, the bounds of intervals used in MTL (as well in the other logics defined below) are rigid and may contain parameters. We assume here that the background first-order theory contains the theory of reals and that the terms cu have real type.

The abbreviations $\tilde{F}_I, \tilde{G}_I, \tilde{P}_I, \tilde{H}_I$ and their non-strict versions are defined in the usual way. Moreover, for all logics defined in this section, we abbreviate the intervals $[0,a]$, $[0,a)$, $[a,\infty)$, (a,∞), $[a,a]$, by respectively $\leq a$, $< a$, $\geq a$, $> a$, $= a$. Thus, for example, $\tilde{F}_{=p}b$ is an abbreviation of $\tilde{F}_{[p,p]}b$.

Let $\sigma = \langle M, \tau, \overline{\mu}\rangle$. We give the semantics just for the metric operators:

$\sigma,t \models \phi_1\tilde{U}_I\phi_2$ iff there exists $t' > t, v(t') - v(t) \in M(I), \sigma, t' \models \phi_2$ and for all $t'', t < t'' < t', \sigma, t'' \models \phi_1$

$\sigma,t \models \phi_1\tilde{S}_I\phi_2$ iff there exists $t' > t, v(t) - v(t') \in M(I), \sigma, t' \models \phi_2$ and for all $t'', t' < t'' < t, \sigma, t'' \models \phi_1$

where $M(I)$ is the set obtained from I by substituting the terms at the endpoints with their interpretation (thus it may be also an empty set).

MTL_0^∞ is the subset of MTL where the intervals in metric operators are in the form $[0,a]$, $(0,a]$, $[0,a)$, $(0,a)$, $[a,\infty)$, (a,∞).

Event-Clock Temporal Logic (ECTL) is instead defined with the following grammar:

$$\phi := p(u,\ldots,u) \mid \phi \wedge \phi \mid \neg\phi \mid \phi\tilde{U}\phi \mid \phi\tilde{S}\phi \mid \rhd_I\phi \mid \lhd_I\phi$$

where u and I are defined as above.

We just give the semantics for the new symbols:

$\sigma,t \models \rhd_I \phi$ iff there exists $t' > t, v(t') - v(t) \in M(I), \sigma,t' \models \phi$ and for all $t'', t < t'' < t', \sigma,t'' \not\models \phi$

$\sigma,t \models \lhd_I \phi$ iff there exists $t' > t, v(t) - v(t') \in M(I), \sigma,t' \models \phi$ and for all $t'', t' < t'' < t, \sigma,t'' \not\models \phi$

Finally, we define the Temporal Logic with Counting (TLC) with the following grammar:

$$\phi := p(u,\ldots,u) \mid \phi \wedge \phi \mid \neg\phi \mid \phi \tilde{U} \phi \mid \phi \tilde{S} \phi \mid \overrightarrow{C}^k_{<cu}\phi \mid \overleftarrow{C}^k_{<cu}\phi$$

where u and cu are defined as above.

We just give the semantics for the new symbols:

$\sigma,t \models \overrightarrow{C}^k_{<cu}(\phi)$ iff there exist $t_1,\ldots,t_k, t < t_1 < t_2 < \ldots < t_k, v(t_k) - v(t) < M(cu)$ such that for all $i \in [1,k], \sigma,t_i \models \phi$

$\sigma,t \models \overleftarrow{C}^k_{<cu}(\phi)$ iff there exist $t_1,\ldots,t_k, t_k < t_{k-1} < \ldots < t_1 < t, v(t) - v(t_k) < M(cu)$ such that for all $i \in [1,k], \sigma,t_i \models \phi$

3 LTL with Event Freezing Functions

3.1 Until next occurrence

Before introducing the new operators, we observe some subtleties of the dense-time semantics. In the discrete-time setting, $F\phi$ and $(\neg\phi)U\phi$ are equivalent. In other words, if ϕ is true in the future, there exists a first point in which it is true, while ϕ is false in all preceding points. This is not the case in the dense-time setting, since for example the third trace of Figure 1 satisfies $F\phi$ but not $(\neg\phi)U\phi$: for every time in which ϕ holds, there exists a left open interval in which ϕ holds as well.

We can instead use another variant of the until operator defined as:

$$\phi_1 U_C \phi_2 := \phi_1 U (\phi_2 \vee (\phi_1 \wedge \tilde{X}\phi_2))$$

Thus, with U_C we are requiring that ϕ_2 holds in a point or in every point of a right interval. In this case, we are guaranteed that there exists a minimum point that satisfies such condition. In fact, since we are assuming finite variability, $F\phi$ is equivalent to $(\neg\phi)U_C\phi = (\neg\phi)U(\phi \vee \tilde{X}\phi)$. In the next sections, we will use this condition to characterize the next point in the future that satisfies ϕ. In particular, when we say "the next point in the future in which ϕ holds", we actually mean ϕ holds in that point or in a right left-open interval (see also Figure 1). Similarly, for the past case.

Note that this is related to the issue of U in the dense time setting raised first by Bouajjani and Lakhnech in [7] and later by Raskin and Schobbens in [29], namely that $\phi_1 U \phi_2$ is satisfied only if the time interval in which ϕ_2 holds is left-closed. In [7], this is solved by considering $(\phi_1 \vee \phi_2)U\phi_2$. However, this does not solve our issue of characterizing the first point in which ϕ_2 holds. In [29], the issue was solved at the semantic level by defining the U on timed state sequence that are fine for the subformulas and quantifying over the time intervals of the sequence instead of over the points of the time domain. We instead chose a more classical approach to define the semantics which seems to clarify better what we mean for "the next point in which ϕ holds". This is more similar to the semantics defined in [20] for event clocks in Event-Clock Timed Automata and for the corresponding quantifiers in the equally expressive monadic logic. However, in [20], a nonstandard real number is used in case ϕ holds in a left-open interval.

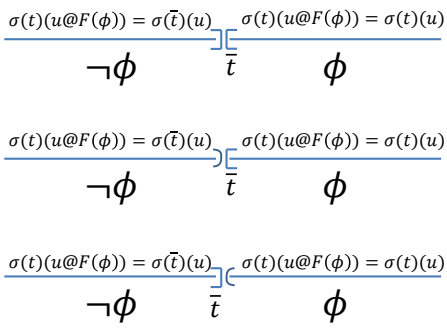

Figure 1: Graphical view of different cases in which ϕ holds in the future. $\bar{t}$ represents "the next point in the future in which ϕ holds".

3.2 Event Freezing Functions

We extend the logic with two binary operators, "at next" $u@\tilde{F}(\phi)$ and "at last" $u@\tilde{P}(\phi)$, which take in input a term u and a formula ψ and represent the value of u at the next point in the future, respectively at the last point in the past, in which ψ holds. If such point does not exist we consider a default value represented by a constant $def_{u@\tilde{F}(\psi)}$ or $def_{u@\tilde{P}(\psi)}$. As in SMT, we also use an if-then-else operator, extended to the temporal case.

The set of LTL with Event Freezing Functions (LTL-EF) formulas is therefore defined as follows:

$$\phi := p(u,\ldots,u) \mid \phi \wedge \phi \mid \neg\phi \mid \phi \tilde{U} \phi \mid \phi \tilde{S} \phi$$
$$u := c \mid x \mid f(u,\ldots,u) \mid u@\tilde{F}(\phi) \mid u@\tilde{P}(\phi) \mid ite(\phi,u,u)$$

A formula ϕ is interpreted on a trace $\sigma = \langle M, \tau, \overline{\mu} \rangle$, where M is a first-order structure over the signature extended with the constant symbols def_u for every event freezing term u in ϕ. The semantics of LTL is thus extended as follows:

- $\sigma(t)(u@\tilde{F}(\phi)) = \sigma(t')(u)$ if there exists $t' > t$ such that, for all t'', $t < t'' < t'$, $\sigma, t'' \not\models \phi$ and $\sigma, t' \models \phi$; $\sigma(t)(u@\tilde{F}(\phi)) = \sigma(t')(u)$ if there exists $t' \geq t$ such that, for all t'', $t < t'' \leq t'$, $\sigma, t'' \not\models \phi$ and $\sigma, t' \models \tilde{X}\phi$; otherwise, $\sigma(t)(u@\tilde{F}(\phi)) = M(def_{u@\tilde{F}(\phi)})$
- $\sigma(t)(u@\tilde{P}(\phi)) = \sigma(t')(u)$ if there exists $t' < t$ such that, for all t'', $t' < t'' < t$, $\sigma, t'' \not\models \phi$ and $\sigma, t' \models \phi$; $\sigma(t)(u@\tilde{P}(\phi)) = \sigma(t')(u)$ if there exists $t' \leq t$ such that, for all t'', $t' \leq t'' < t$, $\sigma, t'' \not\models \phi$ and $\sigma, t' \models \tilde{Y}\phi$; otherwise, $\sigma(t)(u@\tilde{P}(\phi)) = M(def_{u@\tilde{P}(\phi)})$
- $\sigma(t)(ite(\phi,u_1,u_2)) = \sigma(t)(u_1)$ if $\sigma, t \models \phi$, else $\sigma(t)(ite(\phi,u_1,u_2)) = \sigma(t)(u_2)$

The "if-then-else" operator ite can be used to define the non-strict version:

$$u@F(\phi) := ite(\phi, u, u@\tilde{F}(\phi))$$
$$u@P(\phi) := ite(\phi, u, u@\tilde{P}(\phi))$$

We define the following abbreviations:

$$u@\tilde{F}^1(\phi) := u@\tilde{F}(\phi) \qquad u@\tilde{F}^{i+1}(\phi) := (u@\tilde{F}(\phi))@\tilde{F}^i(\phi) \text{ for } i \geq 1$$
$$u@\tilde{P}^1(\phi) := u@\tilde{P}(\phi) \qquad u@\tilde{P}^{i+1}(\phi) := (u@\tilde{P}(\phi))@\tilde{P}^i(\phi) \text{ for } i \geq 1$$

3.3 Extension with Explicit Time

In this section, we extend the language with an explicit notion of time that can be constrained using the event freezing functions defined above. In particular, we introduce an explicit symbol *time*, which represents the time elapsed from the initial state. We allow *time* to be compared with constant terms.

The new set of LTL-EF formulas with explicit time (XLTL-EF) is defined as follows:

$$\phi := p(u,\ldots,u) \mid tu \bowtie cu \mid \phi \wedge \phi \mid \neg\phi \mid \phi\tilde{U}\phi \mid \phi\tilde{S}\phi$$
$$u := c \mid x \mid f(u,\ldots,u) \mid u@\tilde{F}(\phi) \mid u@\tilde{P}(\phi) \mid ite(\phi,u,u)$$
$$tu := time \mid tu@\tilde{F}(\phi) \mid tu@\tilde{P}(\phi)$$
$$cu := c \mid f(cu,\ldots,cu)$$
$$\bowtie := < \mid > \mid \leq \mid \geq$$

The semantics of LTL-EF is extended as follows: $\sigma(t)(time) := v(t)$

Note that we assume that the signature Σ contains the real arithmetic operators and that the underlying theory contains the theory of reals.

3.4 Coverage of Metric Operators

These operators can be seen as a generalization of the ECTL operators as below:

$$\rhd_I \phi := time@\tilde{F}(\phi) - time \in I \wedge \neg\phi\tilde{U}\phi$$
$$\lhd_I \phi := time - time@\tilde{P}(\phi) \in I \wedge \neg\phi\tilde{S}\phi$$

We can encode similarly MTL_0^∞ operators. As proved in [20], in case of non-singular intervals with real constant bounds, MTL operators can be expressed in ECTL (and thus in XLTL-EF).

We can also express TLC properties as follows:

$$\overrightarrow{C}^k_{<cu}(\phi) := time@\tilde{F}^k(\phi) - time < cu \wedge \tilde{F}^k(\phi)$$
$$\overleftarrow{C}^k_{<cu}(\phi) := time - time@\tilde{P}^k(\phi) < cu \wedge \tilde{P}^k(\phi)$$

3.5 Sensor Example

Consider a sensor with input y and output x and a Boolean flag *correct* that represents whether or not the value reported by the sensor is correct. Let us specify that the output x is always equal to the last correct input value with $G(x = y@P(correct))$. We assume that a failure is permanent: $G(\neg correct \rightarrow G\neg correct)$. Consider also a Boolean variable *read* that represents the event of reading the variable x. Let us say that the reading happens periodically with period p: $p > 0 \wedge read \wedge G(read \rightarrow \rhd_{=p} read)$. Finally, let us say that an alarm a is true if and only if the last two read values are the same: $G(a \leftrightarrow x@\tilde{P}(read) = x@\tilde{P}^2(read))$.

We would like to prove that, given the above scenario, every point in which the sensor is not correct is followed within $2 * p$ by an alarm:

$$(G(x = y@P(correct)) \wedge G(\neg correct \rightarrow G\neg correct) \wedge$$
$$p > 0 \wedge read \wedge G(read \rightarrow \rhd_{=p} read) \wedge G(a \leftrightarrow x@\tilde{P}(read) = x@\tilde{P}^2(read)))$$
$$\rightarrow G(\neg correct \rightarrow F_{\leq 2*p} a)$$

In the following, we show that this kind of problems can be indeed solved automatically with SMT-based techniques.

4 Satisfiability Procedure

4.1 Overview of the Procedure

The satisfiability problem for (first-order) LTL with discrete time, and thus also for (X)LTL-EF which is an extension thereof, is in general undecidable (see for example [19]). However, we can reduce it to SMT-based model checking, which although undecidable has effective and mature tool support (also in the case of LTL-next). We thus propose to reduce the satisfiability for (X)LTL-EF to the one for LTL-next. The approach consists of the following steps: 1) the formula is translated into an equisatisfiable one with discrete-time model, 2) the event freezing functions are removed generating an equisatisfiable LTL-next formula. Since LTL-EF is a subset of XLTL-EF, in the following we consider just XLTL-EF formulas.

4.2 Discretization

Given an XLTL-EF formula with dense or super-dense time, we create an equisatisfiable one with discrete time. We will use the non-strict version of the temporal operators as these are those typically supported by tools for LTL. The discretization approach is similar to the one described in [14]. The idea is to split the time evolution is a sequence of singular or open intervals in such a way that the trace is fine for the input formula on such intervals. To this purpose we introduce three variables that encode a sequence of time intervals used to sample the value of variables: ι is a Boolean variable that encodes if the interval is singular or open; δ is a real variable that encodes the time elapsed between two samplings; ζ is a real variable that accumulates arbitrary sums of δ. A constraint ψ_ι ensures that the value of these additional variables represent a valid time interval sequence (e.g., after an open interval there must be a singular interval and ζ is infinitely often greater than 1 and reset). Another constraint ψ_{time} ensures that the variable *time* is equal to the accumulation of δ and that the evaluation of predicates in the formula is uniform in open intervals.

Given these extra variables, we can define the translation. Given a formula ϕ over V, we rewrite ϕ into ϕ_D over $V \cup \{\iota, \delta, \zeta\}$ defined as:

$$\phi_D := \mathscr{D}(\phi) \wedge \psi_\iota \wedge \psi_{time}$$

where $\mathscr{D}$, ψ_ι, and ψ_{time} are defined as follows.

$\mathscr{D}(\phi)$ is defined recursively on the structure of ϕ and rewrites the temporal operators splitting between the case in which the current interval is singular (ι) or open ($\neg\iota$).

Let us consider first $\phi_1 \tilde{U} \phi_2$; intuitively, to hold in a time point t, if t belongs to an open interval (fine for ϕ_1), then ϕ_1 must hold in t; similarly, if $\phi_1 \tilde{U} \phi_2$ because ϕ_2 holds in $t' > t$ and t' is part of an open

interval, also ϕ_1 must hold in t'. Thus, $\phi_1 \tilde{U} \phi_2$ is translated as follows: either the current interval is open ($\neg\iota$), ϕ_2 holds in a future singular interval and ϕ_1 holds now and until that interval; or the current interval is open, ϕ_2 holds in a future open interval and ϕ_1 holds now and until that interval included; or the current interval is singular, ϕ_2 holds in a future singular interval and ϕ_1 holds (strictly) until that interval; or the current interval is singular, ϕ_2 holds in a future open interval and ϕ_1 holds (strictly) until that interval included. Similarly for the past case. Overall:

$$\begin{aligned}\mathscr{D}(\phi_1 \tilde{U} \phi_2) :=& (\neg\iota \wedge \mathscr{D}(\phi_1) \wedge (\mathscr{D}(\phi_1) U ((\iota \wedge \mathscr{D}(\phi_2)) \vee (\mathscr{D}(\phi_1) \wedge \mathscr{D}(\phi_2))))) \vee \\ & (\iota \wedge X(\mathscr{D}(\phi_1) U ((\iota \wedge \mathscr{D}(\phi_2)) \vee (\mathscr{D}(\phi_1) \wedge \mathscr{D}(\phi_2))))) \\ \mathscr{D}(\phi_1 \tilde{S} \phi_2) :=& (\neg\iota \wedge \mathscr{D}(\phi_1) \wedge \mathscr{D}(\phi_1) S ((\iota \wedge \mathscr{D}(\phi_2)) \vee (\mathscr{D}(\phi_1) \wedge \mathscr{D}(\phi_2)))) \vee \\ & (\iota \wedge Y(\mathscr{D}(\phi_1) S ((\iota \wedge \mathscr{D}(\phi_2)) \vee (\mathscr{D}(\phi_1) \wedge \mathscr{D}(\phi_2)))))\end{aligned}$$

Let us consider now $u@\tilde{F}(\phi)$. We first define $\mathscr{D}(u@F(\phi))$, which is used as intermediate step to define $\mathscr{D}(u@\tilde{F}(\phi))$. The discretization of $u@F(\phi)$ is translated into the value of u at the first state in the future such that ϕ holds in that state or the following state corresponds to an open interval in which ϕ holds:

$$\begin{aligned}\mathscr{D}(u@F(\phi)) :=& \mathscr{D}(u)@F(\mathscr{D}(\phi) \vee X(\neg\iota \wedge \mathscr{D}(\phi))) \\ \mathscr{D}(u@P(\phi)) :=& \mathscr{D}(u)@P(\mathscr{D}(\phi) \vee Y(\neg\iota \wedge \mathscr{D}(\phi)))\end{aligned}$$

If the current interval is open and ϕ holds in all points of the interval, then $u@\tilde{F}(\phi) = u = u@F(\phi)$. Similarly, if the current interval is singular and is followed by an open interval in which ϕ holds, then $u@\tilde{F}(\phi) = u = u@F(\phi)$. If the current interval is open and ϕ does not hold in the interval, then again $u@\tilde{F}(\phi) = u@F(\phi)$. Finally, if the interval is singular and is not followed by an open interval in which ϕ holds, $u@\tilde{F}(\phi)$ is equal to the value of $u@F(\phi)$ in the next interval. The overall translation is the following:

$$\begin{aligned}\mathscr{D}(u@\tilde{F}(\phi)) :=& ite(\iota \wedge X(\iota \vee \neg\mathscr{D}(\phi)), next(\mathscr{D}(u@F(\phi))), \mathscr{D}(u@F(\phi))) \\ \mathscr{D}(u@\tilde{P}(\phi)) :=& ite(\iota \wedge Z(\iota \vee \neg\mathscr{D}(\phi)), prev(\mathscr{D}(u@P(\phi))), \mathscr{D}(u@P(\phi)))\end{aligned}$$

where we use *prev* for simplicity with the following semantics: $\sigma(0)(prev(u@P(\phi))) = def_{u@P(\phi)}$ and $\sigma(i+1)(prev(u@P(\phi))) = \sigma(i)(u@P(\phi))$. In practice, this is rewritten in terms of *next* and an extra monitor variable in the usual way.

The following completes the definition with the trivial cases:

$$\begin{aligned}\mathscr{D}(\phi_1 \wedge \phi_2) :=& \mathscr{D}(\phi_1) \wedge \mathscr{D}(\phi_2) & \mathscr{D}(\neg\phi_1) :=& \neg\mathscr{D}(\phi_1) \\ \mathscr{D}(p(u_1,\ldots,u_n)) :=& p(\mathscr{D}(u_1),\ldots,\mathscr{D}(u_n)) & \mathscr{D}(tu \bowtie cu) :=& \mathscr{D}(tu) \bowtie \mathscr{D}(cu) \\ \mathscr{D}(f(u_1,\ldots,u_n)) :=& f(\mathscr{D}(u_1),\ldots,\mathscr{D}(u_n)) & \mathscr{D}(time) :=& time \\ \mathscr{D}(c) :=& c & \mathscr{D}(v) :=& v\end{aligned}$$

ψ_ι encodes the structure of the time model (to enforce for example that after an open interval there must be a singular one and that in a discrete step time does not elapse):

$$\begin{aligned}\psi_\iota :=& \iota \wedge G((\iota \wedge \delta = 0 \wedge X(\iota)) \vee (\iota \wedge \delta > 0 \wedge X(\neg\iota)) \vee (\neg\iota \wedge \delta > 0 \wedge X(\iota))) \wedge \\ & G((next(\zeta) - \zeta = \delta) \vee (\zeta \geq 1 \wedge next(\zeta) = 0)) \wedge GF(\zeta \geq 1 \wedge next(\zeta) = 0)\end{aligned}$$

Finally, ψ_{time} encodes the value of *time* and forces the uniformity of predicates over *time* in open intervals:

$$\psi_{time} := time = 0 \wedge G(next(time) - time = \delta) \wedge \bigwedge_{tu \bowtie cu \in Sub(\phi)} G(\neg\iota \rightarrow ((\mathscr{D}(tu \leq cu) \rightarrow X\mathscr{D}(tu \leq cu)) \wedge (\mathscr{D}(tu \geq cu) \rightarrow Y\mathscr{D}(tu \geq cu))))$$

where $Sub(\phi)$ denotes the set of subformulas of ϕ.

Note in particular that we require to split the time intervals in such a way that for every constant *cu* occurring in a time constraint, $[cu, cu]$ is a time interval in the sequence. Note that *cu* can be in general a term built with the signature symbols that are interpreted rigidly.

Written as above the discretization clearly produces a formula whose size is exponential in the input. However, since we are interested in equisatisfiability we can always use extra variables (one for subformula) to obtain a linear-size formula.

We now prove that the translation is correct, i.e., that the new formula is equisatisfiable.

Theorem 1 *ϕ and ϕ_D are equisatisfiable.*

Proof. Given a trace $\sigma = \langle M, \tau, \overline{\mu} \rangle$ satisfying ϕ we can build a trace σ_D with a discrete time model satisfying ϕ_D as follows. Let $I_0, I_1, I_2, \ldots$ be a sequence of time intervals such that 1) σ is fine for all subformulas of ϕ in each interval I_i, 2) each interval I_i in the sequence is singular or open, and 3) in case of super-dense time, for all $i \geq 0$, there exists an integer n_i such that $\langle n_i, t \rangle \in \tau$ for all $t \in I_i$. We build an assignment to ι, δ and ζ based on such sequence. The values of ι, δ, and ζ are determined by the sequence of intervals in order to satisfy ψ_ι.

Let us define the value assigned by σ_D to ι, δ, ζ as follows:

- $\sigma_D(i)(\iota) = \top$ iff I_i is singular;
- $\sigma_D(i)(\delta) = (r(I_{i+1}) - l(I_{i+1}))/2$ if I_i is singular
  otherwise $\sigma_D(i)(\delta) = (r(I_i) - l(I_i))/2$;
- $\sigma_D(0)(\zeta) = 0$
  $\sigma_D(i+1)(\zeta) = \sigma_D(i)(\zeta) + \sigma_D(i+1)(\delta)$ if $\sigma_D(i+1)(\zeta) \leq 1$
  otherwise $\sigma_D(i+1)(\zeta) = 0$

Thus, $\sigma_D \models \psi_\iota$. Notice in particular, that if $I_i = I_{i+1}$ then $\delta = 0$, if I_i is singular and I_{i+1} is not then δ is equal to half of the length of I_{i+1} and if I_i is not singular then δ is equal to half of the length of I_i.

Let $\delta_i := \sigma(i)(\delta)$ and $t_i = \sum_{0 \leq h < i} \delta_h$ for all $i \geq 0$. Notice that for all $i \geq 0$, $t_i \in I_i$. We complete the time model of σ_D by defining $v(i) := t_i$ for all $i \geq 0$.

Let $\mathbf{t}_i = t_i$ in case σ has a dense time and $\mathbf{t}_i = \langle n_i, t_i \rangle$ in case of super dense time. Let us complete the definition of σ_D by saying that for all $i \geq 0$, $\sigma_D(i)(x) := \sigma(\mathbf{t}_i)(x)$.

We now prove that, for all $i \geq 0$, for all subformulas ψ of ϕ, $\sigma, \mathbf{t}_i \models \psi$ iff $\sigma_D, i \models \mathscr{D}(\psi)$ and for all terms u in ϕ, $\sigma(\mathbf{t}_i)(u) = \sigma_D(i)(\mathscr{D}(u))$.

By definition of $@F$, $\sigma(\mathbf{t}_i)(u@F(\psi))$ is the value of u at the next point $t \geq \mathbf{t}_i$ such that $\sigma, t \models \psi \vee \tilde{X}\psi$. Since σ is fine for ψ, t must belong to a singular interval I_j with $j \geq i$ (so $t = \mathbf{t}_j$). By inductive hypothesis, $\sigma, \mathbf{t}_j \models \psi$ iff $\sigma_D, j \models \mathscr{D}(\psi)$ and $\sigma, \mathbf{t}_{j+1} \models \psi$ iff $\sigma_D, j+1 \models \mathscr{D}(\psi)$. Thus, $\sigma, \mathbf{t}_j \models \psi \vee \tilde{X}\psi$ iff $\sigma_D, j \models \mathscr{D}(\psi) \vee X(\neg\iota \wedge \mathscr{D}(\psi))$. Moreover, still by inductive hypothesis, for all $i < h < j$, $\sigma, \mathbf{t}_h \models \psi$ iff $\sigma_D, h \models \mathscr{D}(\psi)$ and $\sigma(\mathbf{t}_j)(u) = \sigma_D(j)(\mathscr{D}(u))$. Thus, $\sigma(\mathbf{t}_i)(u@F(\psi)) = \sigma_D(i)(\mathscr{D}(u@F(\psi)))$.

It is routine to prove the other cases and we can conclude that $\sigma_D \models \mathscr{D}(\phi)$.

Finally, $\sigma_D \models \psi_{time}$: in fact, $\sigma_D \models time = 0 \wedge G(next(time) - time = \delta)$ by definition of σ_D; the rest of ψ_{time} is trivially satisfied because σ is fine for ϕ.

Vice versa, suppose that there exists σ with discrete time such that $\sigma \models \mathscr{D}(\phi)$. Then we can build a σ_C with super-dense time such that $\sigma_C \models \phi$ as follows. Let $t_i = \sum_{0 \leq h < i} \delta_h$. $I_i := [t_i, t_i]$ if $\sigma, i \models \iota$; otherwise $I_i := (t_{i-1}, t_{i+1})$. Let $\sigma(t)(v) = \sigma(i)(v)$ for every $t \in I_i$.

It is routine to prove that $\sigma_C \models \phi$. ◇

4.3 Removing Event Freezing Functions

In the following, we assume that satisfiability is restricted to traces with discrete time and we use the non-strict version of temporal operators. If the term $u@F(\phi)$ occurs in a formula ψ, we can obtain a formula $\mathscr{R}(\psi, u@F(\phi))$ equisatisfiable to ψ where the term $u@F(\phi)$ has been replaced with a fresh variable $p_{u@F(\phi)}$. More specifically,

$$\begin{aligned}\mathscr{R}(\psi, u@F(\phi)) := & \psi[p_{u@F(\phi)}/u@F(\phi)] \wedge \\ & G(F\phi \rightarrow (\neg\phi \wedge next(p_{u@F(\phi)}) = p_{u@F(\phi)}) U (\phi \wedge p_{u@F(\phi)} = u)) \wedge \\ & G(G\neg\phi \rightarrow p_{u@F(\phi)} = def_{u@F(\phi)})\end{aligned}$$

$\mathscr{R}(\psi, u@F(\phi))$ is a formula on an extended set of variables. Namely, if ϕ is a formula over variables V, then $\mathscr{R}(\psi, u@F(\phi))$ is a formula over $V \cup \{p_{u@F(\phi)}\}$, where $p_{u@F(\phi)}$ does not occur in ψ. However, the value of $p_{u@F(\phi)}$ is uniquely determined by a trace over V. In other words, given a trace σ over V, we can define a trace $\mathscr{R}(\sigma, u@F(\phi))$ over $V \cup \{p_{u@F(\phi)}\}$ such that $\sigma \models \phi$ iff $\mathscr{R}(\sigma, u@F(\phi)) \models \mathscr{R}(\psi, u@F(\phi))$. $\mathscr{R}(\sigma, u@F(\phi))$ is simply defined as follows:

$$\begin{aligned}&\mathscr{R}(\sigma, u@F(\phi))(t)(x) = \sigma(t)(x), x \in V \\ &\mathscr{R}(\sigma, u@F(\phi))(t)(p_{u@F(\phi)}) = \sigma(t)(u@F(\phi))\end{aligned}$$

Theorem 2 *If $\sigma \models \phi$ then $\mathscr{R}(\sigma, u@F(\phi)) \models \mathscr{R}(\psi, u@F(\phi))$. If $\sigma \models \mathscr{R}(\psi, u@F(\phi))$, then $\sigma \models \phi$. Thus, ψ and $\mathscr{R}(\psi, u@F(\phi))$ are equisatisfiable.*

Proof. Let us assume that $\sigma \models \phi$.

Given the definition of $\mathscr{R}(\sigma, u@F(\phi))$, the prophecy variable $p_{u@F(\phi)}$ is given the value of the term $u@F(\phi)$, and thus $\mathscr{R}(\sigma, u@F(\phi)) \models \psi[p_{u@F(\phi)}/u@F(\phi)]$.

For every t, if $\sigma, t \models F(\phi)$, then there exists $t' \geq t$ such that $\sigma, t' \models \phi$ and for all t'', $t \leq t'' < t'$, $\sigma, t'' \not\models \phi$. Thus, $\sigma(t')(u@F(\phi)) = \sigma(t'')(u@F(\phi)) = \sigma(t')(u)$. Thus $\sigma, t \models G(F(\phi) \rightarrow (\neg\phi \wedge next(p_{u@F(\phi)}) = p_{u@F(\phi)}) U (\phi \wedge p_{u@F(\phi)} = u))$.

For every t, if $\sigma, t \models G(\neg\phi)$, then $\sigma(t)(u@F(\phi)) = \sigma(t)(def_{u@F(\phi)})$. Thus, $\sigma, t \models G(G(\neg\phi) \rightarrow p_{u@F(\phi)} = def_{u@F(\phi)})$.

Vice versa, let us assume that $\sigma \models \mathscr{R}(\psi, u@F(\phi))$. It is sufficient to prove that $\sigma(t)(p_{u@F(\phi)}) = \sigma(t)(u@F(\phi))$.

Let us assume that there exists $t' \geq t$ such that, for all t'', $t \leq t'' < t'$, $\sigma, t'' \not\models \phi$ and $\sigma, t' \models \phi$; thus, $\sigma(t)(u@F(\phi)) = \sigma(t')(u)$. Since $\sigma, t \models F\phi \rightarrow (\neg\phi \wedge next(p_{u@F(\phi)}) = p_{u@F(\phi)}) U (\phi \wedge p_{u@F(\phi)} = u)$, then $\sigma(t)(p_{u@F(\phi)}) = \sigma(t')(u)$ as well.

If such t' does not exists, then $\sigma(t)(u@F(\phi)) = \sigma(t)(def_{u@F(\phi)})$. Also, $\sigma, t \not\models F(\phi)$. Since $\sigma, t \models G\neg\phi \rightarrow p_{u@F(\phi)} = def_{u@F(\phi)}$, then $\sigma(t)(p_{u@F(\phi)}) = \sigma(t)(def_{u@F(\phi)})$ as well. This concludes the proof. ◇

Formula	Valid	Time in sec.
$(G(x = y@P(correct)) \wedge G(\neg correct \rightarrow G\neg correct) \wedge$ $p > 0 \wedge read \wedge G(read \rightarrow \rhd_{=p} read) \wedge G(a \leftrightarrow x@\tilde{P}(read) = x@\tilde{P}^2(read)))$ $\rightarrow G(\neg correct \rightarrow F_{\leq 2*p} a)$	Yes	16
$G(b \rightarrow (x@F(b) = x))$	Yes	0
$G(\tilde{X}(b) \rightarrow (x@F(b) = x))$	Yes	0
$Fb \rightarrow (\neg bU(b \vee (\tilde{X}b)))$	Yes	0
$(G(a \rightarrow F_{\leq 1} b) \wedge G(b \rightarrow F_{\leq 1} c)) \rightarrow G(a \rightarrow F_{\leq 2} c)$	Yes	5
$(G(a \rightarrow F_{\leq p} b) \wedge G(b \rightarrow F_{\leq p} c)) \rightarrow G(a \rightarrow F_{\leq 2*p} c)$	Yes	8
$(\rhd_{=q} \rhd_{=p} b) \rightarrow (\rhd_{=p+q} b \vee \rhd_{\leq q} b)$	Yes	5
$Fb \rightarrow (\neg bUb)$	No	0
$\neg(x = y \wedge F_{\geq 3} x > y)$	No	0
$(G(a \rightarrow F_{\leq 3} b) \wedge G(b \rightarrow F_{\leq 3} c)) \rightarrow G(a \rightarrow F_{\leq 3} c)$	No	2
$(\rhd_{=p} \rhd_{=p} b) \rightarrow (\rhd_{=2*p} b)$	No	4

Table 1: Some examples and their verification results

We similarly remove past event freezing operator $@P$ with the following rule:

$$\begin{aligned}\mathscr{R}(\psi, u@P(\phi)) := & \psi[p_{u@P(\phi)}/u@P(\phi)] \wedge \\ & G(P\phi \rightarrow (\neg\phi \wedge Z(next(p_{u@P(\phi)}) = p_{u@P(\phi)}))S(\phi \wedge p_{u@P(\phi)} = u)) \wedge \\ & G(H\neg\phi \rightarrow p_{u@P(\phi)} = def_{u@P(\phi)})\end{aligned}$$

5 Experimental Evaluation

The satisfiability procedure presented in the previous section has been implemented in a simple prototype written in C that takes in input a XLTL-EF formula and performs the described transformations. Then, we use nuXmv [9] to check LTL satisfiability modulo theories, in particular we use the algorithm that combines IC3IA [11] with k-liveness [15]. Actually, we create an universal model and we apply model checking to check that the formula is valid. We checked the validity of different formulas using a machine with Intel(R) Core(TM) i7-3720QM CPU at 2.60GHz with 4GB of memory. All results are available at `https://es.fbk.eu/people/tonetta/papers/gandalf17/`.

In Table 1, we report the time needed to solve some example formulas. The sensor example described above was proved by nuXmv in 16s. This result is promising considering that we do not implement any optimization neither in the translation nor in the engine. The reported time includes the time for translation, verification, and counterexample generation in case of not valid formulas.

As proof of concept, we also verify the validity of some MTL_0^∞ (e.g. $(G(a \rightarrow F_{\leq p} b) \wedge G(b \rightarrow F_{\leq p} c)) \rightarrow G(a \rightarrow F_{\leq 2*p} c)$) and ECTL (e.g. $(\rhd_{=q} \rhd_{=p} b) \rightarrow (\rhd_{=p+q} b \vee \rhd_{\leq q} b)$) formulas, also including parameters. To the best of our knowledge, this is the first tool that is able to automatically prove the validity of this kind of formulas.

6 Conclusions and Future Work

In this paper, we considered an extension of first-order linear-time temporal logic with two new event freezing functional symbols, which represent the value of a term at the next state in the future or last state in the past in which a formula holds. We defined the semantics in different time models considering discrete time with real timestamps, dense and super-dense (weakly-monotonic) time. We precisely characterized what we mean for "next point in the future in which ϕ holds" so that, assuming finite variability, such point always exists also in the dense time setting. Using an explicit variable that represents time, standard metric operators can be encoded in the new logic. We provided a reduction to equisatisfiable discrete-time formulas without event freezing functions and we solve the satisfiability of the latter by SMT-based model checking. A prototype implementation of the technique shows that the approach can analyze interesting properties in an automated way despite the expressiveness of the logic.

The directions for future works are manifold. We want to integrate this techniques in mature tools such as nuXmv [9] and in OCRA [10] for contract-based reasoning; we want to extend it to encompass variables with continuous function evolution and constraints on their derivatives as in HRELTL [14] (HyCOMP [13] is actually already supporting $time\_until(\phi)$ and $time\_since(\phi)$, which are a restricted version of $time@F(\phi)$ and $time@P(\phi)$, where ϕ must represent a discrete change); finally, we want to apply the new logic in industrial use cases within the CITADEL project (`http://citadel-project.org/`) to specify complex properties of monitoring components.

References

[1] R. Alur, T. Feder & T.A. Henzinger (1996): *The Benefits of Relaxing Punctuality*. *J. ACM* 43(1), pp. 116–146, doi:10.1145/227595.227602.

[2] R. Alur, L. Fix & T.A. Henzinger (1999): *Event-Clock Automata: A Determinizable Class of Timed Automata*. *Theor. Comput. Sci.* 211(1-2), pp. 253–273, doi:10.1016/S0304-3975(97)00173-4.

[3] R. Alur & T.A. Henzinger (1991): *Logics and Models of Real Time: A Survey*. In: *REX Workshop*, pp. 74–106, doi:10.1007/BFb0031988.

[4] R. Alur & T.A. Henzinger (1993): *Real-Time Logics: Complexity and Expressiveness*. *Inf. Comput.* 104(1), pp. 35–77, doi:10.1006/inco.1993.1025.

[5] C.W. Barrett, R. Sebastiani, S.A. Seshia & C. Tinelli (2009): *Satisfiability Modulo Theories*. In: *Handbook of Satisfiability*, pp. 825–885, doi:10.3233/978-1-58603-929-5-825.

[6] D.A. Basin, F. Klaedtke & S. Müller (2010): *Policy Monitoring in First-Order Temporal Logic*. In: *CAV*, pp. 1–18, doi:10.1007/978-3-642-14295-6_1.

[7] A. Bouajjani & Y. Lakhnech (1995): *Temporal Logic + Timed Automata: Expressiveness and Decidability*. In: *CONCUR*, pp. 531–545, doi:10.1007/3-540-60218-6_40.

[8] P. Bouyer, F. Chevalier & N. Markey (2010): *On the expressiveness of TPTL and MTL*. *Inf. Comput.* 208(2), pp. 97–116, doi:10.1016/j.ic.2009.10.004.

[9] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri & S. Tonetta (2014): *The nuXmv Symbolic Model Checker*. In: *CAV*, pp. 334–342, doi:10.1007/978-3-319-08867-9_22.

[10] A. Cimatti, M. Dorigatti & S. Tonetta (2013): *OCRA: A tool for checking the refinement of temporal contracts*. In: *ASE*, pp. 702–705, doi:10.1109/ASE.2013.6693137.

[11] A. Cimatti, A. Griggio, S. Mover & S. Tonetta (2014): *IC3 Modulo Theories via Implicit Predicate Abstraction*. In: *TACAS*, *LNCS* 8413, Springer, pp. 46–61, doi:10.1007/978-3-642-54862-8_4.

[12] A. Cimatti, A. Griggio, S. Mover & S. Tonetta (2014): *Verifying LTL Properties of Hybrid Systems with K-Liveness*. In: *CAV*, *LNCS* 8559, Springer, pp. 424–440, doi:10.1007/978-3-319-08867-9_28.

[13] A. Cimatti, A. Griggio, S. Mover & S. Tonetta (2015): *HyComp: An SMT-Based Model Checker for Hybrid Systems*. In: *TACAS*, pp. 52–67, doi:10.1007/978-3-662-46681-0_4.

[14] A. Cimatti, M. Roveri & S. Tonetta (2009): *Requirements Validation for Hybrid Systems*. In: *CAV*, pp. 188–203, doi:10.1007/978-3-642-02658-4_17.

[15] K. Claessen & N. Sörensson (2012): *A Liveness Checking Algorithm that Counts*. In: *FMCAD*, IEEE, pp. 52–59.

[16] J. Daniel, A. Cimatti, A. Griggio, S. Tonetta & S. Mover (2016): *Infinite-State Liveness-to-Safety via Implicit Abstraction and Well-Founded Relations*. In: *CAV*, pp. 271–291, doi:10.1007/978-3-319-41528-4_15.

[17] S. Demri & R. Lazic (2009): *LTL with the freeze quantifier and register automata*. *ACM Trans. Comput. Log.* 10(3), pp. 16:1–16:30, doi:10.1145/1507244.1507246.

[18] C.A. Furia & M. Rossi (2007): *On the Expressiveness of MTL Variants over Dense Time*. In: *FORMATS*, pp. 163–178, doi:10.1007/978-3-540-75454-1_13.

[19] S. Ghilardi, E. Nicolini, S. Ranise & D. Zucchelli (2007): *Combination Methods for Satisfiability and Model-Checking of Infinite-State Systems*. In: *CADE*, pp. 362–378, doi:10.1007/978-3-540-73595-3_25.

[20] T. A. Henzinger, J.-F. Raskin & P.-Y. Schobbens (1998): *The Regular Real-Time Languages*. In: *ICALP*, pp. 580–591, doi:10.1007/BFb0055086.

[21] Y. Hirshfeld & A.M. Rabinovich (2006): *An Expressive Temporal Logic for Real Time*. In: *MFCS*, pp. 492–504, doi:10.1007/11821069_43.

[22] R. Koymans (1990): *Specifying Real-Time Properties with Metric Temporal Logic*. *Real-Time Systems* 2(4), pp. 255–299, doi:10.1007/BF01995674.

[23] R. Koymans (1992): *Specifying Message Passing and Time-Critical Systems with Temporal Logic*. *Lecture Notes in Computer Science* 651, Springer, doi:10.1007/3-540-56283-4.

[24] O. Lichtenstein, A. Pnueli & L.D. Zuck (1985): *The Glory of the Past*. In: *Logics of Programs*, pp. 196–218, doi:10.1007/3-540-15648-8_16.

[25] Z. Manna & A. Pnueli (1992): *The temporal logic of reactive and concurrent systems - specification*. Springer, doi:10.1007/978-1-4612-0931-7.

[26] J.J. Ortiz, A. Legay & P.-Y. Schobbens (2010): *Memory Event Clocks*. In: *FORMATS*, pp. 198–212, doi:10.1007/978-3-642-15297-9_16.

[27] A. Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*, pp. 46–57, doi:10.1109/SFCS.1977.32.

[28] A.M. Rabinovich (1998): *On the Decidability of Continuous Time Specification Formalisms*. *J. Log. Comput.* 8(5), pp. 669–678, doi:10.1093/logcom/8.5.669.

[29] J.-F. Raskin & P.-Y. Schobbens (1997): *State Clock Logic: A Decidable Real-Time Logic*. In: *HART*, pp. 33–47, doi:10.1007/BFb0014711.

[30] J.-F. Raskin & P.-Y. Schobbens (1999): *The Logic of Event Clocks - Decidability, Complexity and Expressiveness*. *Journal of Automata, Languages and Combinatorics* 4(3), pp. 247–286.