# A Rewriting Based Monitoring Algorithm for TPTL[⋆]

Ming Chai[1], Holger Schlingloff[2]

[1] ming.chai@cms.hu-berlin.de, [2] hs@informatik.hu-berlin.de

**Abstract.** In this paper, we present a rewriting based monitoring algorithm for time propositional temporal logic (TPTL), which is a classic time extension of linear temporal logic (LTL). TPTL has been shown to be more expressive than other real-time extensions of LTL, e.g., metric temporal logic (MTL). We first describe the syntax and semantics of TPTL on finite time-traces. Using Maude, which is an executable environment for various logics, we give rewriting clauses to check whether a finite time-trace satisfies a TPTL formula. We use our algorithm to test a concrete example from the European Train Control System (ETCS), and evaluate it on several benchmarks. The results show the feasibility of our approach.

## 1 Introduction

Runtime verification is proposed for checking whether a run of a system satisfies or violates a given correctness property [1]. It is seen as a lightweight verification technique when compared to model checking and testing. Runtime verification is able to avoid the following problems of model checking: *i)* when checking a high complexity system, model checking could suffer from the so-called state explosion problem; *ii)* when checking a black-box system, a model of the system may not be available for model checking; *iii)* the object of model checking is a model of the system, not the system itself.

Runtime verification is performed by using a *monitor*. This is a device or a piece software that reads a behavior of the system under monitoring and gives a certain verdict (*true* or *false*) as the result. A behavior of the system is presented by its *trace*, which is an observable execution sequence of the system. Unlike model checking, runtime verification does not check all executions of the underlying system, but a finite trace. Hence it does not suffer from the state explosion problem when dealing with a large system. Furthermore, runtime verification does not need a model of the system. Therefore, it is well suited to check black-box systems. Finally, the checking object of runtime verification is the system itself. Thus, the possibility of introducing additional errors in the modeling is excluded.

⋆ This work was supported by the State Key Laboratory of Rail Traffic Control and Safety (Contract No.: RCS2012K001), Beijing Jiaotong University

One of the most interesting problems in runtime verification is how to build a monitor from a high level specification. Havelund et al. [2] propose a formula rewriting based runtime verification approach, constituting part of a project named Java PathExplorer (JPAX). Their work aims at monitoring Java programs and has been used in Mars Exploration Rover missions. Feng et al. propose an MOP framework for software development and analysis, in which the satisfaction/violation of properties can be detected by executing the code. Barringer et al. [3] propose a rule-based system for trace analysis RuleR. They also propose the LOGSCOPE system, which is an extension of RuleR with a simple, user-friendly temporal logic. Gastin et al. [4] propose an LTL to Büchi automata translation, which is able to generate monitors [5].

For checking time-relevant properties, real-time logics have been introduced into runtime verification. Bauer et al. [6] work on TLTL based runtime verification for monitoring real-time properties. They define TLTL by introducing two formulae ($\rhd_a \in I$) and ($\lhd_a \in I$) with $a$ being an event, and $I$ being a time interval. They translate TLTL formulae to event-clock automata for detecting whether a trace is accepted or rejected.

Metric temporal logic (MTL) [7] is a well studied real-time logic. It is obtained by extending standard LTL with a time bounded temporal operator $\mathcal{U}_{[a,\ b]}$, where $a$, $b$ are natural numbers. When dealing with dense time, two different semantics of MTL are considered, depending on whether the trace to be checked consists of discrete events or continuous states. These two different semantics are called pointwise semantics and interval-based semantics, respectively. Several MTL based monitoring approaches have been proposed. Thati et al. [8] propose a formula rewriting based monitoring algorithm for MTL with pointwise semantics. Nickovic et al. propose monitoring algorithms for a restricted version of MTL, named MITL. The time interval of temporal operators in MITL cannot be singular. Basin et al. [9] propose a monitoring algorithm for metric first-order logic with pointwise semantics. Their approach can cope with variables ranging over infinite domains. They also develop algorithms for both pointwise MTL and interval-based MTL [10].

Alur et al. [11] propose a "more temporal" real-time logic, named time propositional temporal logic (TPTL). It is obtained from LTL by introducing a freeze quantifier "$x.$". A TPTL formula can "reset" a formula clock at some point by assigning variables in the formula to the time value when the formula is evaluated. The expressiveness of TPTL and MTL is studied in [12, 13]. It has been proven that TPTL is strictly more expressive than MTL. Although the verification and model checking problem for TPTL has been studied intensely, the number of TPTL based runtime verification approaches is quite limited. One example is Kristoffersen et al. [14], who give a monitoring algorithm for LTL$_t$, which also extends LTL by a freeze quantifier. The difference between TPTL and LTL$_t$ is that the latter needs an extra clock variable $r$ for expressing time.

In this paper, we propose a formula rewriting based runtime verification approach for TPTL. The monitor consists of a TPTL formula and a formula rewriting algorithm, where the formula is generated from a high level specification. The monitor receives a time-trace, which is generated from the underlying system. It detects failures through checking whether this time-trace violates the formula. The process is shown in Fig. 1. Our algorithm is developed directly based on the syntax and semantics of TPTL.

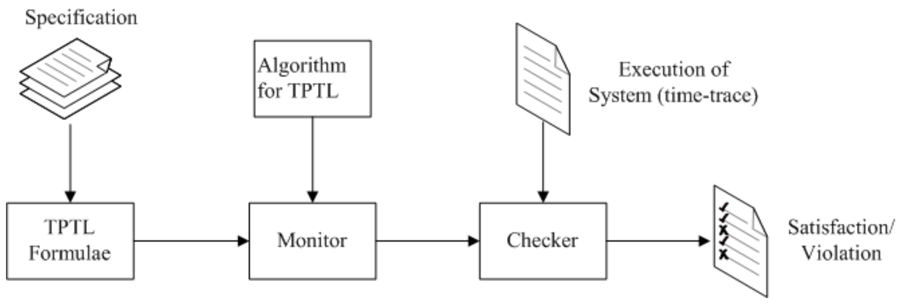

**Fig. 1.** The runtime verification process

Our algorithm is based on Maude [15], which is a high performance system for model checking, theorem proving, and programming. It can be used for runtime verification implementation. We use the Maude rewriting logic, in the style of the LTL rewriting program proposed by Havelund [16]. Additionally, we present a case study of a concrete example in the railway domain. We translate several properties contained in the specifications of a signaling system to TPTL formulae, and abstract some executions of the system to time-traces. Then we monitor these time-traces in Maude. The results show that our approach is an ideal method for monitoring time-traces.

The rest part of the paper is organized as follows. Section 2 introduces the definition of TPTL, including the syntax and semantics. Section 3 presents the Maude-based algorithm for TPTL based monitor. Section 4 shows a case study with a concrete example from the railway domain. Section 5 contains the conclusion and future work.

## 2 Preliminaries

### 2.1 Time-events and Time-traces

Given a (finite) set of atomic propositions $AP$ and a (finite) alphabet $\Sigma = 2^{AP}$, an *event* is defined as any single element of $\Sigma$, i.e. $e = \{p_1, \cdots, p_m\}$ with $p_1, \cdots, p_m \in AP$. If $e$ is a singleton, we omit the curly brackets in the denotation. If we denote the set of natural numbers by $\mathbb{N}_{\geq 0}$ and $t \in \mathbb{N}_{\geq 0}$, then a *time-event* is defined as a pair $te = (e, t)$ from the set $\Sigma \times \mathbb{N}_{\geq 0}$. The natural number $t$ in a time-event $te$ is a discrete *time stamp*, to identify the time of the event emitted by a running real-time system. Given a time-event $te = (e, t)$, we define

$Event(te) \triangleq e$ and $Time(te) \triangleq t$.Based on time-events, a *time-trace* is defined as follows.

**Definition 1.** ***(Time-trace)*** *A time-trace tt is a (possibly infinite) sequence of time-events, i.e.* $tt = ((e_0, t_0), (e_1, t_1), \cdots)$*, where for each* $i \in \mathbb{N}_{\geq 0}$*, it holds that* $t_i < t_{i+1}$ *(strict monotonicity).*

Given a finite time-trace $tt = (te_0, te_1, \cdots, te_n)$, we use the following denotations in the rest part of the paper: the length of $tt$ is denoted by $|tt|$; the $i^{th}$ time-event in $tt$ is denoted by $tt[i]$, i.e. $tt[i] \triangleq te_i$.

## 2.2 Syntax and Semantics of TPTL

LTL is a widely-accepted logic for specifying properties of infinite traces. TPTL is an extension of LTL to express real-time properties. It contains a freeze quantifier "$x.$", which assigns the time value when the formula is evaluated to the variable $x$. A TPTL formula $x.\ \varphi(x)$ is satisfied by a time-trace $tt$ iff $\varphi(time(tt[0]))$ is satisfied by $tt$. For instance, a TPTL formula $(\Box\ x.\ (Request \rightarrow \Diamond\ y.\ (Ack \wedge y < 5 + x)))$ expresses the property "whenever an event *Request* occurs, then the acknowledgment event *Ack* must occur within 5 time units". This formula is satisfied, e.g., by the time-trace $(\cdots, (Request, 7), \cdots, (Ack, 11), \cdots)$, since 11 $< 5 + 7$. More precisely, TPTL is defined as follows.

**Definition 2.** ***(Syntax for TPTL)*** *Given a finite set AP of atomic propositions and a set V of free variables, the terms of* $\pi$ *and formulae* $\varphi$ *of TPTL are inductively formed according to the following grammar, where* $x \in V$*,* $r \in \mathbb{N}_{\geq 0}$*,* $p \in AP$ *and* $\sim \in \{\leq, <, =, >, \geq\}$*:*

$\pi ::= x + r \mid r$

$\varphi ::= \bot \mid p \mid (\varphi_1 \rightarrow \varphi_2\ ) \mid (\varphi_1\ \mathcal{U}\ \varphi_2) \mid \pi_1 \sim \pi_2 \mid x.\ \varphi.$

The following shorthands are used in TPTL as in LTL: $\Diamond\ \varphi$ stands for $\top\ \mathcal{U}\ \varphi$, $\Box\ \varphi$ stands for $\neg\Diamond\ \neg\varphi$, and $\bigcirc\ \varphi$ stands for $\bot\ \mathcal{U}\ \varphi$.

Assume that $\mathcal{E}$ is a function $\mathcal{E}$: $V \rightarrow \mathbb{N}_{\geq 0}$ for assigning free variables in $\mathbb{N}_{\geq 0}$ (time value) such that $\mathcal{E}(x + r) = \mathcal{E}(x) + r$ and $\mathcal{E}(r) = r$. Given a variable $x$ and a natural number $r$, we denote $\mathcal{E}[x := r]$ for the evaluation $\mathcal{E}'$ such that $\mathcal{E}'(x) = r$, and $\mathcal{E}'(y) = \mathcal{E}(y)$ for all $y \in V\backslash\{x\}$. In runtime verification, the time-traces to be checked are finite. Hence, we give TPTL finite semantics as follows.

**Definition 3.** ***(Semantics for TPTL)*** *Let tt be a finite trace with* $i \in \mathbb{N}_{\geq 0}$ *being a position, p a proposition, and* $\varphi_1$ *and* $\varphi_2$ *any TPTL formulae. The satisfaction relation* $(tt, i, \mathcal{E}) \models \varphi$ *is defined inductively as follows:*

$(tt, i, \mathcal{E}) \not\models \bot$;

$(tt, i, \mathcal{E}) \models p$ *iff* $p \in Event(tt[i])$;

$(tt, i, \mathcal{E}) \models (\varphi_1 \rightarrow \varphi_2)$ *iff* $(tt, i, \mathcal{E}) \models \varphi_1$ *implies* $(tt, i, \mathcal{E}) \models \varphi_2$;

$(tt, i, \mathcal{E}) \models (\varphi_1\ \mathcal{U}\ \varphi_2)$ *iff there exists* $i < j < |tt|$ *with* $(tt, j, \mathcal{E}) \models \varphi_2$ *and for all* $i < j' < j$ *it holds that* $(tt, j', \mathcal{E}) \models \varphi_1$;

$(tt, i, \mathcal{E}) \models \pi_1 \sim \pi_2$ *iff* $\mathcal{E}(\pi_1) \sim \mathcal{E}(\pi_2)$;

$(tt, i, \mathcal{E}) \models x.\ \varphi$ *iff* $(tt, i, \mathcal{E}[x := Time(tt[i])]) \models \varphi$.

As is proven in [13], TPTL is strictly more expressive than MTL. The property "whenever an a-event occurs, then a b-event will occur in the future and, later a c-event will occur within 3 time units" can be expressed by a TPTL formula as: $\Box\ x.\ (a \to \Diamond\ (b \wedge \Diamond\ y.\ (c \wedge y < x + 3)))$. This property cannot be expressed in MTL.

## 3 The Rewriting Algorithm for TPTL in Maude

Subsequently, we develop an algorithm for checking whether a finite time-trace satisfies a TPTL formula. More specifically, when checking the satisfaction relation between a finite time-trace and a TPTL formula, the formula is continuously transformed to another formula by consuming the first time-event in the time-trace. This procedure processes iteratively, until the last time-event is consumed. It will output a boolean value in $\mathbb{B} = \{true,\ false\}$. Our algorithm is implemented in Maude, which provides an executable environment for various logics. Here where we informally describe some of Maude's features which are related to the algorithm, more details can be found in the manual [17].

### 3.1 The Algorithm for Basic Rewriting Operators and Logic Connectives

In this algorithm, we use the functional modules following the pattern

fmod <name> is <body> emdfm.

The body of a functional module consists of a collection of declarations, of which we will use sorts (*sort* and *sorts*), subsorts (*subsort* and *subsorts*), operations (*op* and *ops*), variables (*var* and *vars*) and equations (*eq*).

We first need to define all necessary data types involved in the algorithm, including atomic proposition (*Atom*), event (*Event*), time-event ($TimeEvent$), time-trace ($TimeTrace$) and free variable (*FreeV*). Atomic propositions have one sort, with no operations or constrains. It is defined as follows.

fmod ATOM is sort Atom . endfm

As is shown in Fig. 2, the module DATA-TYPE defines all other data types. The statements "protecting ATOM" and "protecting NAT" import the modules ATOM and NAT without changing their initial semantics. NAT is a module of *natural number*s.

In our algorithm, an event consists of a set of atomic propositions; a time-event consists of an event and a natural number; a time-trace consists of a sequence of time-events; and a free variable consists of a natural number (the value of the variable) and an atomic proposition (the name of the variable). The operators "_ _", "_:-_ ", "_,_" and "_ of _" generate an event, a time-event, a time-trace and a free variable, respectively. Every operator has a priority feature, which is declared through "[prec $n$]" with $n \in \mathbb{N}_{\geq 0}$.

```
fmod DATA-TYPE is protecting ATOM . protecting NAT .
    sorts Event TimeEvent TimeTrace .
    subsorts Atom < Event .
    subsorts Nat < Atom .
    subsorts TimeEvent < TimeTrace .
    sort FreeV .
    subsort FreeV < Formula .
    sort Formula .
    subsort Atom < Formula .
    op __ : Atom Event -> Event [prec 23] .
    op _:-_ : Event Nat -> TimeEvent [prec 23] .
    op nil : -> Event .
    op _,_ : TimeEvent TimeTrace -> TimeTrace [prec 25] .
    op _ of _ : Nat Atom -> FreeV [prec 23].
endfm
```

**Fig. 2.** Definition of data types

The statements *subsorts* declare *Atom* to be a subsort of *Event*, *TimeEvent* to be a subsort of *TimeTrace*, and *FreeV* to be a subsort of *Atom*. Furthermore, we define *Nat* to be a subsort of *Atom*.

Based on the syntax and semantics of TPTL described above, we define several operators, "$\_\{\_\}$", "$\_\{\_\}'$" and "$\_\models\_$", for checking whether a time-trace satisfies a formula. The operator "$\_\{\_\}$" receives a formula and an event. It yields the formula $\top/\bot$ depending on whether the event satisfies the formula or not. The operator "$\_\{\_\}'$" is defined on basis of "$\_\{\_\}$" for checking the satisfaction relation between a time-event and a formula. A time-event *te* satisfies a formula $\varphi$ iff $\varphi\{Event(te)\}$ returns $\top$. By extending "$\_\{\_\}'$", the operator "$\_\models\_$" is defined for checking whether a time-trace satisfies a formula. This operator receives a time-trace and a formula, and generates a boolean value in $\mathbb{B}$. Given a formula $\varphi$ and a time-trace (te, tt) consisting of a time-event te and its suffix tt, then (te, tt) $\models \varphi$ returns *true*/*false* iff $\varphi\{\text{te}\}'$ returns $\top/\bot$ as the result.

The calculation rules of logic connectives $\wedge$ (and), $\vee$ (or), ++ (exclusive or), ! (negation), $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence) are declared as usual [16].

In our algorithm, the comparison operators ($\leq$, $<$, $=$, $>$ and $\geq$) and the primitive operator (+) are denoted by $\leq'$, $<'$, $='$, $>'$, $\geq'$ and $+'$ respectively, to distinguish the original definition of these operators in Maude. See $<$ as an example of comparison operators, the declaration for $<'$ is shown in Fig. 3.

```
vars R R' N N' : Nat .
vars A A' : Atom .
op _<'_ : Formula Formula -> Formula [prec 40] .
ceq R <' R' = true if R < R' .
ceq R <' R' = false if R > R' or R == R' .
ceq ( N of A ) <' R = true if N < R .
ceq ( N of A ) <' R = false if N > R or N == R .
ceq ( N of A ) <' ( N' of A' ) = true if N < N' .
ceq ( N of A ) <' ( N' of A' ) = false if N > N' or N == N' .
```

**Fig. 3.** The calculation rule for comparison operator

## 3.2 The Algorithm for Temporal Operators and Freeze Quantifiers

In this part we describe the rewriting algorithm for temporal operators and freeze quantifiers in TPTL. The algorithm for temporal operators ($\Box$, $\Diamond$, $\mathcal{U}$ and $\bigcirc$) is defined in a module LOGICS-LTL, shown in Fig. 4.

```
fmod  LOGICS-LTL is extending PROP-CALC .
    vars  X Y : Formula . var TE : TimeEvent . var TT : TimeTrace .
    op  _U_ : Formula Formula -> Formula [prec 14] .
    op  _U'_ : Formula Formula -> Formula [prece 14] .
    op  []_ : Formula -> Formula [prec 11] .
    op  <>_ : Formula -> Formula [prec 11].
    op  o_ : Formula -> Formula [prec 11] .
    eq  TE |= X U Y = false .
    eq  TE, TT |= X U Y = TT |= X U' Y .
    eq  TE, TT |= X U' Y = TE, TT |= Y or TE, TT |= X and TT |= X U' Y .
    eq  TE |= X U' Y = TE |= Y .
    eq  <> X = (true U X) .
    eq  [] X = (! <> (! X)) .
    eq  o X = (false U X) .
endfm
```

**Fig. 4.** Algorithm for standard LTL operators

In Maude, we denote the formula $x.\ \varphi$ by $(R\ of\ x)$ @ $\varphi$ with $x \in AP$ being the name of the quantifier, $R \in \mathbb{N}_{\geq 0}$ being the value of the quantifier, and $\varphi$ being a TPTL formula. In addition, we define an operator "@@" for assigning free variables in $\varphi$. The rewriting process of $tt \models (R\ of\ x)$ @ $\varphi$ is separated into two steps as follows.

1. The variable $x$ of $x.\ \varphi$ is set to the time when the formula is evaluated. Hence, the formula $(R\ of\ x)$ @ $\varphi$ is rewritten to another formula $((Time(tt[0])$ of $x)$ @@ $\varphi)$, where $(Time(tt[0])$ is the initial time value from the given time-trace;

2. The operator @@ assigns all occurrences of variable $x$ in $\varphi$ to the value $(Time(tt[0]))$, and proceeds with the $tt \models \varphi$ checking process.

The algorithm for this process is defined in a module FREE-QUAN, shown in Fig. 5.

```
fmod FREE-QUAN is extending LTL .
    vars X Y Z Z' : Formula .
    var E : Event . var TE : TimeEvent . var TT : TimeTrace .
    op _@_ : FreeV Formula -> Formula [prec 7] .
    op _@@_ : FreeV Formula -> Formula [prec 6] .
    op _+'_ : FreeV Nat -> Nat [prec 39] .
    /* the value of a freeze quantifier (R of A) equals to T, which is the time of the
        first time-event in the time-trace */
    eq E :- T, TT |= (R of A) @ X = E :- T , TT |= (( T of A ) @@ X) .
    eq E :- T |= (R of A) @ X = E :- T |= ( T of A ) @@ X .
    ceq ( M of A) @@ ( M' of A') = ( M of A' ) if A == A' . // a FreeV (M' of A') is
        assigned to the value of the freeze quantifier ( M of A) if they have the same
        name
    ceq ( M of A) @@ ( M' of A') = ( M' of A' ) if A =/= A' . // a FreeV (M' of
        A') is not assigned to the value of the freeze quantifier ( M of A) if they have
        different names
    /* the value assignment rule for an algebraic formula. */
    ceq ( N of A) @@ (N' of A' +' R) = N + R if A == A' .
    ceq ( N of A) @@ (N' of A' +' R) = (N' of A' +' R) if A =/= A' .
endfm
```

**Fig. 5.** The rewriting algorithm for freeze quantifiers

In addition, we introduce the following equivalences into the algorithm for the operator @@. These equivalences are declared in the module FREE-QUAN, where N, N′, M and M′ are natural numbers; A, A′, B and B′ are atomic propositions; E is an event; and X, Y, true and false are formulae.

- (N of A) @@ ((N' of A') @@ (M of B)) = (N of A) @@ (N' of A') @@ (M of B);
- (N of A) @@ (X $\wedge$ Y) = ((N of A) @@ X) $\wedge$ ((N of A) @@ Y);
- (N of A) @@ (X ++ Y) = ((N of A) @@ X) ++ ((N of A) @@ Y);
- (N of A) @@ E = E;
- (N of A) @@ true = true;
- (N of A) @@ false = false;
- (N of A) @@ (X $<'$ Y) = ((N of A) @@ X) $<'$ ((N of A) @@ Y);
- (N of A) @@ ((N' of A') @ X) = (N' of A') @ ((N of A) @@ X);
- (N of A) @@ ($\Diamond$ X) = $\Diamond$ ((N of A) @@ X);
- (N of A) @@ ($\Box$ X) = $\Box$ ((N of A) @@ X);
- (N of A) @@ (X $\mathcal{U}$ Y) = ((N of A) @@ X) $\mathcal{U}$ ((N of A) @@ Y);

– ( N of A ) @@ ($\bigcirc$ X) = $\bigcirc$ ((N of A) @@ X).

*Example 1.* A time-trace ((a, 1), (b, 4)) and a TPTL formula ($x.$ ($a\,\mathcal{U}\ y.$ ($b \wedge y < x + 5$))) are denoted by ( n of x) @ (b U ((n' of y) @ (a /\ (( n' of y) <' ((n of x) +' 5))))) in Maude, respectively. The rewriting process for checking the satisfaction relation between them is as follows:

1. according to the rewriting rule for the operator @, the formula ($a : -1$, $b : -4 \models (n\ of\ x)$ @ ($a\,\mathcal{U}$ (($n'\ of\ y$) @ ($b \wedge$ (( $n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5)))))) equals to (($a : -1$, $b : -4 \models$ (1 $of\ x$) @@ ($b\,\mathcal{U}$ (($n'\ of\ y$) @ ($a \wedge$ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5))))));
2. according to the equivalences defined above, the formula (1 $of\ x$) @@ ($a\,\mathcal{U}$ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5))))) equals to (((1 $of\ x$) @@ $a$) $\mathcal{U}$ ((1 $of\ x$) @@ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5)))))), where
   (a) ((1$of\ x$) @@ $b$) equals to $b$;
   (b) ((1 $of\ x$) @@ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5))))) equals to (($n'\ of\ y$) @ ((1 $of\ x$) @@ ($b \wedge$ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5))))), which equals to (($n'\ of\ y$) @ (((1 $of\ x$) @@ $b$) $\wedge$ ((1 $of\ x$) @@ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5)))));
   (c) the formula (($n'\ of\ y$) @ (((1 $of\ x$) @@ $b$) $\wedge$ ((1 $of\ x$) @@ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5))))) equals to (($n'\ of\ y$) @ ($b \wedge$ (((1 $of\ x$) @@ ($n'\ of\ y$)) $<'$ ((1 $of\ x$) @@ (($n\ of\ x$) $+'$ 5))))), which equals to (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ 6))), hence
   (d) the formula (((1 $of\ x$) @@ $a$) $\mathcal{U}$ ((1 $of\ x$) @@ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ (($n\ of\ x$) $+'$ 5)))))) equals to ($a\,\mathcal{U}$ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ 6))));
3. according to the rewriting algorithm for the operator $\mathcal{U}$, the formula ($a : -1$, $b : -4 \models$ ($a\,\mathcal{U}$ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ 6))))) equals to ($b : -4 \models$ ($a\,\mathcal{U}'$ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ 6))))), which equals to ($b : -4 \models$ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ 6))));
4. the formula ($b : -4 \models$ (($n'\ of\ y$) @ ($b \wedge$ (($n'\ of\ y$) $<'$ 6)))) equals to ($b : -4 \models$ ((4 $of\ y$) @@ ($b \wedge$ (($n'\ of\ y$) $<'$ 6)))), which equals to ($b : -4 \models$ (((4 $of\ y$) @@ $b$) $\wedge$ ((4 $of\ y$) @@ (($n'\ of\ y$) $<'$ 6))));
5. the formula ($b : -4 \models$ (((4 $of\ y$) @@ $b$) $\wedge$ ((4 $of\ y$) @@ (($n'\ of\ y$) $<'$ 6)))) equals to ($b : -4 \models$ ($b \wedge$ ((4 $of\ y$) @@ ($n'\ of\ y$) $<'$ (4 $of\ y$) @@ 6))), which equals to ($b : -4 \models$ ($b \wedge$ $true$));
6. the formula ($b : -4 \models$ ($b \wedge$ $true$)) equals to (($b : -4 \models b$) $\wedge$ ($b : -4 \models$ $true$)), which equals to $true$;
7. therefore, the time-trace ((a, 1), (b, 4)) satisfies the formula ($x.$ ($a\,\mathcal{U}\ y.$ ($b \wedge y < x + 5$))).

## 4 Case Study: the RBC/RBC Handover Process

In this section, we apply our TPTL runtime verification implementation to a concrete example from the European Train Control System (ETCS). ETCS is a signaling, control and train protection system that is replacing the national, incompatible safety systems within Europe. ETCS consists of the on-board sub-system (composed of ERTMS/ETCS on-board equipment, the on-board part of the GSM-R radio system and specific transmission modules for existing national train control systems), and the track-side sub-system (composed of balise, line-side electronic unit, GSM-R, radio block center (RBC), euroloop and radio infill unit) [18]. In ETCS, the RBC is responsible for providing movement authorities to allow the safe movement of trains. A movement authority is generated by computing messages to be sent to the trains, where the messages are on the basis of information received from external track-side systems and information exchanged with the on-board sub-system. A route is divided into several RBC supervision areas. Here we consider the RBC/RBC handover specification. When a train approaches the border of an RBC supervision area, an RBC/RBC handover process takes place (see Fig. 6). The RBC/RBC handover specification specifies how a train moves from one RBC supervision area to an adjacent one.

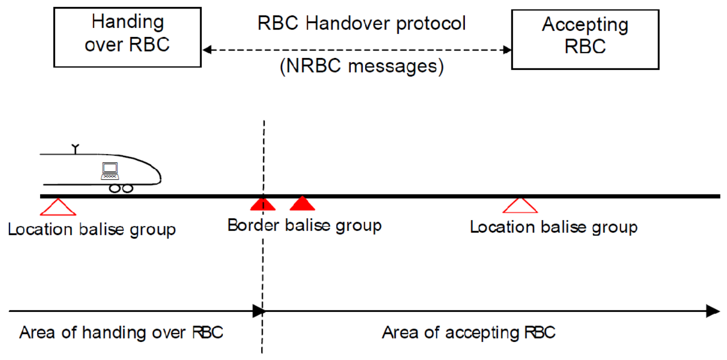

**Fig. 6.** The RBC/RBC handover process

We consider properties on basis of the two different specifications: FIS for the RBC/RBC Handover [19] and RBC-RBC Safe Communication Interface [20]. An execution of the system refers to the following properties in the FIS for the RBC/RBC Handover.

- Property 1: “the handing over RBC is responsible to send information about an approaching train to the accepting RBC area (i.e. pre-announcement)” (4.2.2.1);
- Property 2: “the handing over RBC must send Acknowledgment after receiving route related information” (5.2.2.5);
- Property 3: “if the Acknowledgment for route related information is missing, the accepting RBC must send route related information again” (5.2.3.5).

Based on the specification of the Safe Communication Interface, we assume that the time to take into account an incoming message and produce an answer is between 30 and 60 time units. We also assume that the tolerance window for the messages transition time is between 0 and 50 time units. Table 1 shows the abbreviations used in our case.

**Table 1. Abbreviations in case study**

| Abbreviation | Definition |
|---|---|
| HOVcond | Handover condition detected |
| PreANN | Pre-announcement |
| RRI | Route related information |
| Ackn | Acknowledgment |
| AcknMissing | The Acknowledgement is missed |
| RRIReq | Route related information request |
| MAReq | Movement authority request |
| PosRep | Position report |
| Ann | Announcement |
| TOR | Taking Over Responsibility |
| BPSRE | Position report: "Border passed by safe rear end" |
| BPFE | Position report: "Border passed by max safe front end" |

Let *Mess* be any message. We write "*sendMess*" for the *Mess* which is sent by a component, and "*recvMess*" for the *Mess* which is received by a component. The above properties can be expressed by the following TPTL formulae.

- Property 1: $\varphi_1 = \Box\, x.(\text{sendPreANN} \to \Diamond\, y.\ (\text{recvPreANN} \wedge (y \leq x+50)))$.
- Property 2: $\varphi_2 = \Box\, x.(\text{recvRRI} \to \Diamond\, y.(\text{sendAckn} \wedge (y \geq x+30) \wedge (y \leq x+60)))$.
- Property 3: After an RRI message is sent by the accepting RBC, three time intervals must be considered: the transition time of RRI ($0 < r_1 \leq 50$), the time for producing acknowledgment ($30 \leq r_2 \leq 60$) and the transition time of the message acknowledgment ($0 < r_3 \leq 50$). Hence, if the accepting RBC does not receive the acknowledgment between 30 and 160 ($= 50 + 60 + 50$) time units after sending an RRI, an AcknMissing message should occur. The accepting RBC should resend an RRI after the AcknMissing message occurs, within 50 time units. Now property 3 can be expressed by the TPTL formula $\varphi_3$, :
  - $\varphi_{31} = \Box\, (x.(\text{sendRRI} \to \Diamond\, y.(\text{recvAckn} \wedge (y \leq x+160) \wedge (y \geq x+30))) ++\ x.(\text{sendRRI} \to \Diamond\, y.(\text{AcknMissing} \wedge (y > x+160))))$;
  - $\varphi_{32} = \Box\, x.(\text{AcknMissing} \to \Diamond\, y.(\text{sendRRI} \wedge (y < x+50)))$;
  - $\varphi_3 = \varphi_{31} \wedge \varphi_{32}$.

We assume that the handing over RBC and the accepting RBC have a synchronized clock, beginning at time 0. An example of their executions is given in Fig. 7. A corresponding time-trace is as follows.

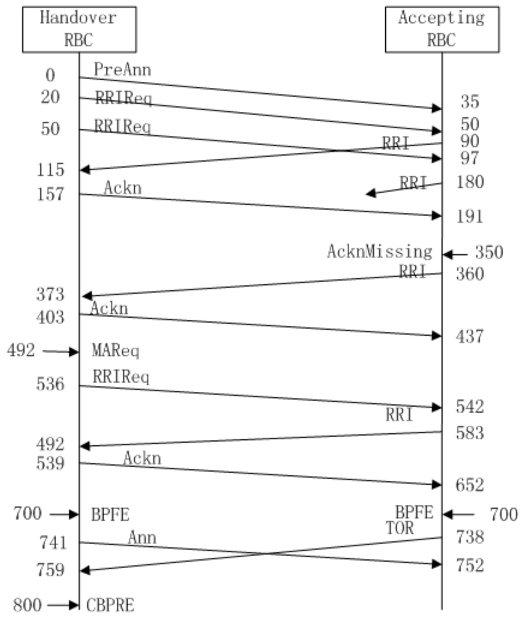

**Fig. 7.** An example of message sequence

$tt_1$ = (sendPreANN, 0), (sendRRIReq, 20), (recvPreANN, 35), ({sendRRIReq, recvRRIReq}, 50), (sendRRI, 90), (recvRRIReq, 97), (recvRRI, 115), (sendAckn, 157), (sendRRI, 180), (recvAckn, 191), (AcknMissing, 350), (sendRRI, 360), (recvRRI, 373), (sendAckn, 403), (recvAckn, 437), (recvMAReq, 492), (sendRRIReq, 536), (recvRRIReq, 542), (sendRRI, 583), (recvRRI, 592), (send Ackn, 639), (recvAckn, 652), (recvBPFE, 700), (sendTOR, 738), (sendAnn, 741), (recvAnn, 752), (recvTOR, 759), (recvCBPRE, 800).

The calculation results of $tt_1 \models \varphi_1$, $tt_1 \models \varphi_2$ and $tt_1 \models \varphi_3$ in Maude are all *true*. It means that this execution satisfies all the three properties.

Time-trace $tt_2$ represents an execution in which some errors occur: *i*) the accepting RBC receives the pre-announcement 60 time units after it is sent; *ii*) the handing over RBC does not send the acknowledgment after reception of an RRI; *iii*) when missing the acknowledgment of an RRI, the accepting RBC does not resend it.

$tt_2$ = (sendPreANN, 0), (sendRRIReq, 20), (recvPreANN, 60), ({sendRRIReq, recvRRIReq}, 65), (sendRRI, 90), (recvRRIReq, 97), (recvRRI, 115), (sendRRI, 180), (recvMAReq, 492), (sendRRIReq, 536), (recvRRIReq, 542), (sendRRI, 583), (recvRRI, 592), (sendAckn, 639), (recvAckn, 652), (recvBPFE, 700), (sendTOR, 738), (sendAnn, 741), (recvAnn, 752), (recvTOR, 759), (recv CBPRE, 800).

The calculation results of $tt_2 \models \varphi_1$, $tt_2 \models \varphi_2$ and $tt_2 \models \varphi_3$ are all *false*, which means that this execution of the system violates the properties.

We repeated similar experiments several times with difference traces. The checking efficiency is shown in Fig. 8. The case study shows that our TPTL based runtime verification implementation is able to detect failures in the executions of a system with an acceptable complexity.

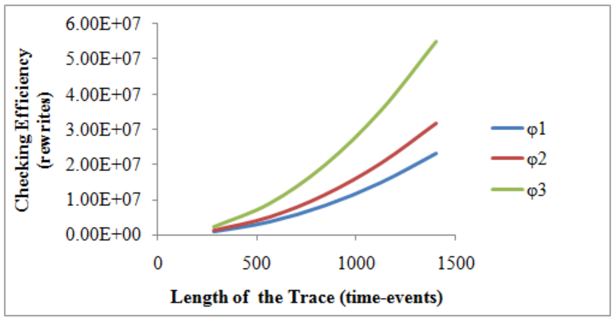

**Fig. 8.** The monitoring efficiency in Maude

## 5 Conclusion

In this paper, we have proposed a runtime verification method for TPTL. We developed a formula rewriting based algorithm, and implemented the algorithm in Maude. This makes it possible to check the satisfaction relation between a long time-trace and a complex TPTL formula automatically. Furthermore, we have presented a case study with a concrete example from the railway domain. The results show the feasibility of our implementation.

There are several interesting topics for future work. Firstly, as is well known, LTL with two truth values gives misleading results when checking finite traces. For this reason, we want to develop a three-valued TPTL, introducing a third truth value "*inconclusive*". This truth value means the satisfaction relation between a time-trace and a TPTL formula is decided by the potential suffix of the given initial fragment of the time-trace. Secondly, the clock reset principle in a TPTL formula $x.\varphi$ is to freeze the variable $x$ in $\varphi$ when the formula is evaluated. This makes TPTL unintuitive in the cases when a property contain a "clock-reset" condition. Hence an extension of TPTL with modifying the freeze quantifier "$x.$" to "$\psi.$" is worth to be studied, where $\psi$ is any formula. Last but not least, to solve the difficulty of writing formal specifications in runtime verification, we are going to study specification techniques. The long-term goal is to develop a methodology to semi-automatically translate system specifications from the railway domain into temporal formulae.

## References

1. Leucker, M., Schallhart, C.: A Brief Account of Runtime Verification. Journal of Logic and Algebraic Programming 78, 293-303 (2009)
2. Havelund, K., Roşu, G.: Monitoring Java Programs with Java PathExplorer. Electronic Notes in Theoretical Computer Science 55, 200-217 (2001)
3. Barringer, H., Havelund, K., Rydeheard, D., Groce, A.: Rule Systems for Runtime Verification: A Short Tutorial. In: Runtime Verification, pp. 1-24. Springer, (2009)
4. Gastin, P., Oddoux, D.: Fast LTL to Büchi Automata Translation. In: Computer Aided Verification, pp. 53-65. Springer, (2001)
5. d'Amorim, M., Roşu, G.: Efficient Monitoring of $\omega$-languages. In: Computer Aided Verification, pp. 364-378. Springer, (2005)
6. Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Transactions on Software Engineering and Methodology (TOSEM) 20, 14 (2011)
7. Koymans, R.: Specifying Real-time Properties with Metric Temporal Logic. Real-time systems 2, 255-299 (1990)
8. Thati, P., Roşu, G.: Monitoring Algorithms for Metric Temporal Logic Specifications. Electronic Notes in Theoretical Computer Science 113, 145-162 (2005)
9. Basin, D., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime Monitoring of Metric First-order Temporal Properties. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, pp. 49-60. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, (2008)
10. Basin, D., Klaedtke, F., Zălinescu, E.: Algorithms for Monitoring Real-time Properties. In: Runtime Verification, pp. 260-275. Springer, (2011)
11. Alur, R., Henzinger, T.A.: A Really Temporal Logic. Journal of the ACM (JACM) 41, 181-203 (1994)
12. Alur, R., Henzinger, T.A.: Real-time Logics: Complexity and Expressiveness. Information and Computation 104, 35-77 (1993)
13. Bouyer, P., Chevalier, F., Markey, N.: On the Expressiveness of TPTL and MTL. Information and Computation 208, 97-116 (2010)
14. Kristoffersen, K.J., Pedersen, C., Andersen, H.R.: Runtime Verification of Timed LTL Using Disjunctive Normalized Equation Systems. Electronic Notes in Theoretical Computer Science 89, 210-225 (2003)
15. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Quesada, J.F.: Maude: Specification and Programming in Rewriting Logic. Theoretical Computer Science 285, 187-243 (2002)
16. Havelund, K., Rosu, G.: Monitoring Programs Using Rewriting. In: Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on, pp. 135-143. IEEE, (2001)
17. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: Maude Manual (version 2.6). University of Illinois, Urbana-Champaign 1, 4.6 (2011)
18. UNISIG: SUBSET-026: System Requirements Specification. (2008)
19. UNISIG: SUBSET-039: FIS for the RBC/RBC Handover. (2005)
20. UNISIG: SUBSET-098: RBC-RBC Safe Communication Interface. (2007)