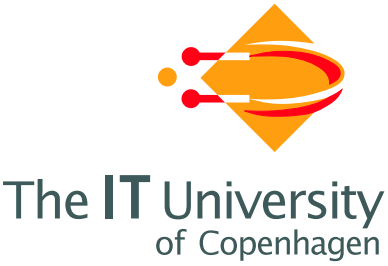

Event-Based Runtime Checking of Timed LTL

Kåre J. Kristoffersen
Christian Pedersen
Henrik R. Andersen

IT University Technical Report Series **TR-2003-37**
ISSN 1600–6100 **November 2003**

ISSN 1600–6100

ISBN 87-7949-050-6

Copies may be obtained by contacting:

IT University of Copenhagen
Glentevej 67
DK-2400 Copenhagen NV
Denmark

Telephone: +45 38 16 88 88
Telefax: +45 38 16 88 99
Web `www.it-c.dk`

Abstract. In this paper we describe an event-based algorithm for runtime verification of timed linear temporal logic. The algorithm is based on a rewriting of the formula expressing a desired or undesired property of a timed system. Rewriting takes place, at discrete points in time, but only when there is a relevant state-change taking place in the timed system, or a deadline, determined by the formula, has been passed. By limiting the rewriting to only points in time where an event occurs, and not at all discrete time-points, makes the algorithm useful in situations where there are large data sets and large differences in the relevant time scales (ranging perhaps from milliseconds to months as in business software).

The algorithm works by rewriting, for each event, the timed LTL formula into a residual formula that takes into account the time and system state at the occurence of the event. The residual formula will be the requirement for the timed system in the future, to be further rewritten at the occurrence of the next event.

Keywords: Timed LTL, Disjunctive Normalized Equation Systems, Residual formula, Smallest Interesting Timepoint, Timed Based Fixed Point Reduction, Charaterization of Runtime Verification.

1 Introduction

Runtime verification is a branch of verification in which a running program is supervised by a concurrently running *verifier*. In this paper we shall look at *timed runtime verification*, in

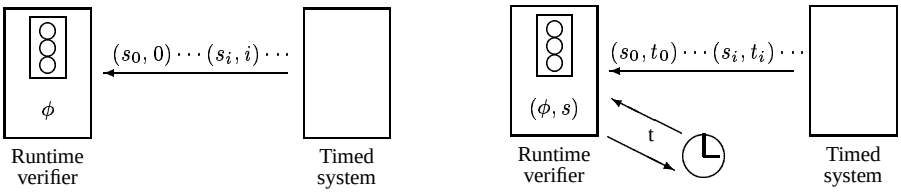

Fig. 1. A runtime verifier supervising a timed system. In the figure to the left, state information is transmitted from the timed system at each discrete time point $0, 1, 2, \ldots$. The runtime verifier computes a new residual formula ϕ at each step, resulting in one of three situations, indicated by the three-coloured traffic light. "Red" corresponds to the situation where an error has been detected, i.e., the formula has reduced to false. "Green" corresponds to acceptance, i.e., the formula has reduced to true and the verifier can stop. "Yellow" corresponds to the situation where so far no error has been found but the verifier cannot yet stop. In the figure to the right the verifier only computes when an event happens, at the time points $t_0 < t_1 < \cdots$. An event is either a *state change* in the timed system or a timeout. Timeouts are set by the verifier and computed as what we shall refer to as the *smallest interesting timepoint* as given by the formula ϕ.

which time will be an important parameter in the task of the verifier. Our verifier will use a discrete notion of time but it is developed such that there will not be a discrete sampling of state at the choosen resolution but only at "relevant" points in time. The relevant points are given by events coming from state changes in the non-timed part of the timed system, and timeout defined by the formula under verification. The verifier is capable of dealing with time even when the time scale spans from milliseconds to days or months. This is crucial for getting a useful verifier in systems with large amounts of data and events timed at milliseconds but with consequences that might be in days or months. This will be important in application areas such as business software (also sometimes commercially called Enterprise Resource Planning systems), where transactions occur at the millisecond level but for instance paying conditions might span weeks or months. Figure 1 illustrates the situation comparing a state-based "polling scheme" with an event-driven timed verifier.

Properties are formulated in a real-time logic called LTL_t. The logic LTL_t is an extension of LTL with real time constructs embodied by a freeze quantifier together with atomic clock constraints making it possible to express real time logical properties of a system (following [7]). This logic is suitable for expressing timed safety, liveness and fairness properties. An example of a *bounded* liveness property for a bank account could be: *If Balance is negative then within 10 days the Balance should be positive or zero*. In LTL_t this could be written as follows (assuming the resolution and scale of time is days):

$$\Box(Balance < 0 \Rightarrow x.t \leq x + 10\, U\ Balance \geq 0)$$

In the above expression the variable t is the global time, or more precisely the time of the observation trace generated by the system being monitored. In this paper we use states with discrete time. This corresponds to time stamps with real time at a given accuracy (for instance, milliseconds). The semantics of the *freeze* formula $x.\phi$ is that when evaluated, the value of x is replaced by the current time.

We introduce a normal form for formulae in LTL_t and show that all formulae may be written in this normal form. A formula on normal form consists of a collection of mutually dependent formula identifiers each stating as a disjunctive normal form:

1. **What should hold *now* (i.e. a *propositional* part).**
2. **What identifiers should hold in the *next* state (i.e. a *temporal* part).**

Initially the property to be verified is a distinct idenfier (the *top* identifier) of the equation system corresponding to the original formula. As the verifier proceeds this will be expanded to a disjunctive normal form of identiers referred to as a residual formula. The number of equations in the normal form will be linear in the number of temporal operators of the original property. However, the total size might be exponential in the worst-case due to conversion of some sub-formulae into disjunctive-normal forms.

Related work

The traditional field of application for formal methods and especially verification is within safety critical and embedded systems for which correctness is of vital importance. Errors are either fatal or they are costly. Therefore a lot of energy has been put into developing tools for checking such systems at design time, that is, prior to execution i.e. SPIN [4]. As an alternative, attempts have been made to check a running java program in Java PathExplorer [1]. Common for these efforts is that what is verified is a *program*, and typically a program taking input from a very restricted set of possibilities. This implies that the program under investigation can be regarded as a closed system, which may be checked alone.

Financial applications and business software (ERP Systems) as well as traditional back end databases are a class of applications that have attracted very little attention from the formal methods community. Such a system can be given a semantics which resemples that of timed traces, namely a sequence of states where each state consists of predicates true in this state and then a *time stamp* explaining *when* the state is measured. One reason that these systems have not gained very much interest from the verification communuty is that they are typically not of a safety–critical nature and hence correctness is of less importance compared with traffic control or production systems. At the same time these systems differ from the beforementioned in that they can (and should be able to) consume unboundedly many different input data. As such they suffer severely from the well–known ”State Explosion Problem” and consequently exhaustive verification is not feasible.

There could be several angles of attack on solving the timed run-time verification problem. One line could be to translate the timed LTL-formula into a timed automaton. The main challenge here is how to deal with the freeze operator. The logical clocks should correspond to clocks of the automaton, however, there is no simple bound of the number of needed clocks. Consider for instance a *punctuality* property like

$$\Box(p \Rightarrow x.\Diamond(q \wedge t = x + 1000)),$$

expressing that always, when p holds, then after exactly 1000 time units, q should hold. If p change as often as possible then it might change 500 times before the first time q should be verified to be true. All these time points must be remembered indicating that the automaton will need up to at least 500 clocks. It means that it would be impractical to try to compute a full automaton in advance and an on-the-fly construction employing a variation of alternating automaton seems to be a way to find a solution. In [11] this problem is avoided by limiting the expressive power of the timed logic to upper-bounds. They avoid having any acceptance condition and there is no blow-up in the number of neeeded clocks. We work with the full logic.

The approach taken here is that of computing a residual formula, taking into account the states seen sofar, and expressing the property to hold for the timed system in the future. This

is a bit like computing "weakest pre-conditions" but in a forwards manner: a "strongest post-condition", as it has been classically done for program verification in the pioneering work of Dijkstra and Hoare.

The current approach also has a relationship to [16] and [15] in terms of rewriting a specification with respect to state information. In [16] it is shown how to build a (branching time) transition system into a modal logic specification by constructing a *quotient*. This method is used as an efficient verification procedure for *parallel* systems. In [15] the same exercise is repeated for real time systems. In both these works the quotient consists of a new (and *larger*) set of formulae whose number and size are sought to be kept small by applying a number of heuristics for minimization. Our work differs significantly from that of [16] and [15] in two substantial ways. First, by rewriting the specification to a normal form *prior* to verification we obtain a specification whose size is *fixed* throughout verification. Second, at runtime (corresponding to verification time in their work) we need only to maintain a formula which is a disjunctive normal form of identifiers that will need to hold in the next state.

Outline

The paper is organized as follows. In Section 2 we present the logic LTL_t as well as its interpretation in timed traces. In Section 3 we present our normal form for formulae in LTL_t and show that any formula may be written in this normal form. In Section 4 we present our algorithm and state that our method is sound and relatively complete with respect to satisfiability. In Section 5 we present the notion of smallest interesting timepoint and in Section 6 we present a notion of time based fixed point reductoins which improves the efficiency of our method. Finally we conclude and give directions for further work.

2 Temporal Logics for Real Time

Let $AP = \{p, q, r, \ldots\}$ be a set of *atomic propositions* and let $t \in \mathbb{N}$ be a discrete time. We denote a *timed state* by a pair (s, t) where $s \subset AP$ whose meaning is that the propositions in s, and only these, are true at time t. The time component can be thought of as a time-stamp on a snapshot of a system's state.

Definition 1. ***A* (discretely) timed trace *over* AP *is an infinite sequence of states***

$$\sigma = \sigma_0\sigma_1 \cdots \sigma_i \cdots,$$

where each* $\sigma_i = (s_i, t_i)$ *is a timed state. We require that* $t_i < t_{i+1}$ *for all* i*. A timed trace is* complete *if* $t_0 = 0$ *and for all* i*:* $t_{i+1} = t_i + 1$*. We denote by* Σ *the set of all timed traces and we denote by* $\hat{\Sigma}$ *the set of all complete timed traces.

We use the notations $s(\sigma_i)$ and $t(\sigma_i)$ for the propositional respectively timed part of a timed state. We use superscripting with i for the sub-sequence σ^i, which starts at σ_i, i.e., the sequence $\sigma^i = \sigma_i\sigma_{i+1}\cdots$. An incomplete trace can always be completed and in [5] the algorithmic framework rested on an assumption that this was done. In this paper we will relax this condition, and allow a more natural notion of timed traces with the possibility of taking timesteps a size larger than one time unit. More precisely we will show that under certain (and quite commonly appearing) circumstances it is possible to skip the processing of some of the states in a timed trace. That is, given that the propositional part is unchanged and that

none of the timing constraints in the specification changes value a new state does not need to be considered, since the residual formula would not change anyway.

Definition 2. Timed LTL, LTL_t, *is given by the following abstract syntax*

$$\phi ::= p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid t \sim x + c \mid x.\phi \mid \phi_1 U \phi_2 \mid \phi_1 V \phi_2 \mid \bigcirc \phi$$

where $p \in AP$, t ***refers to the "current time" in a timed trace,*** x ***is a*** **discrete formula clock,** $c \in \mathbb{N}$ ***and*** $\sim \in \{<, \leq, =, >, \geq\}$**.**

The syntactic elements are: atomic propositions and negations of them, logical connectives, *time constraints*, the *freeze operator* which "records" the current time in the clock variable x, and the temporal operators *until*, *release* (the dual of until) and *next*. We shall use standard abbreviations such as $\Box\phi = falseV\phi$ for the always operator, $\Diamond\phi = trueU\phi$ for the eventually operator, and implication etc. as logical connectives.

The purpose of the standard temporal operators until (U) and release (V) is that they capture real-time properties: $\phi_1 U \phi_2$ holds if there is some time in the future t where ϕ_2 holds and on all time points up to (not including) this one, ϕ_1 holds. When working with discrete time and completed traces, this corresponds quite closely to the existence of an index i, such that σ^i satisfy ϕ_2 and for all smaller indices j, ϕ_1 holds. With discrete time and completed traces, the next operator also has a natural semantics: "next" refers to the next point in time, which is the time associated with the next timed state of the trace.

The resulting logic can be viewed as an extension of the normal untimed linear time temporal logic extended with the notion of clocks, which are capable of recording time (=positions) in the trace using the freeze operator and relate them to other times (=positions) in the trace using time constraints.

In the logic we only have negation at the level of propositions, but assume that the user can still freely use it since a negation may be pushed inside to the propositions using the standard procedure interchanging conjunction with disjunction and until with release (the operator V).

The specification language for our runtime verification will be that of closed LTL_t expressions. Neither we shall use the $\bigcirc$–operator nor the V–operator. However, the reason to include them in the logic is that they are needed when we rewrite specifications to *normal form*, see Section 3.

Given an environment for formula clocks ϵ as a partial mapping from the set of clocks to $\mathbb{N}$, we can define the semantics $[\![\phi]\!]\epsilon \subseteq \hat{\Sigma}$ of a formula ϕ with free clock variables in the domain of an enviroment ϵ, inductively as follows:

$$\begin{aligned}
[\![p]\!]\epsilon &= \{\sigma \in \hat{\Sigma} \mid p \in s(\sigma_0)\} \\
[\![\neg p]\!]\epsilon &= \{\sigma \in \hat{\Sigma} \mid p \notin s(\sigma_0)\} \\
[\![\phi_1 \wedge \phi_2]\!]\epsilon &= [\![\phi_1]\!]\epsilon \cap [\![\phi_2]\!]\epsilon \\
[\![\phi_1 \vee \phi_2]\!]\epsilon &= [\![\phi_1]\!]\epsilon \cup [\![\phi_2]\!]\epsilon \\
[\![t \sim x + c]\!]\epsilon &= \{\sigma \in \hat{\Sigma} \mid t(\sigma_0) \sim \epsilon(x) + c\} \\
[\![x.\phi]\!]\epsilon &= \{\sigma \in \hat{\Sigma} \mid \sigma \in [\![\phi]\!]\epsilon[t(\sigma_0)/x]\} \\
[\![\phi_1 U \phi_2]\!]\epsilon &= \{\sigma \in \hat{\Sigma} \mid \exists i.\ (\sigma^i \in [\![\phi_2]\!]\epsilon \text{ and } \forall j, 0 \leq j < i.\ \sigma^j \in [\![\phi_1]\!]\epsilon)\} \\
[\![\phi_1 V \phi_2]\!]\epsilon &= \{\sigma \in \hat{\Sigma} \mid \forall i.\ \sigma^i \in [\![\phi_2]\!]\epsilon \text{ or} \\
&\qquad (\exists j.\ \sigma^j \in [\![\phi_1]\!]\epsilon \text{ and } \forall i, 0 \leq i \leq j.\ \sigma^i \in [\![\phi_2]\!]\epsilon)\} \\
[\![\bigcirc \phi]\!]\epsilon &= [\![\bigcirc]\!]([\![\phi]\!]\epsilon),
\end{aligned}$$

where $[\![\bigcirc]\!](S) = \{\sigma \in \hat{\Sigma} \mid \sigma^1 \in S\}$.

A complete timed trace σ is now said to satisfy a closed formula ϕ if $\sigma \in [\![\phi]\!]\epsilon$ for an environment ϵ where all clocks have value zero written as $\sigma \models \phi$. A timed trace σ satisfy a formula if the completion of it, $\hat{\sigma}$, satisfy the formula.

Using the next-operator, we can find an alternative characterization of the until and release operators in terms of fixed-points. We use the lattice $2^{\hat{\Sigma}}$ of subsets of $\hat{\Sigma}$ ordered by set-inclusion. Then for any monotonic function $F : 2^{\hat{\Sigma}} \to 2^{\hat{\Sigma}}$ there will be a minimum fixed-point $\mu F \subseteq \Sigma$ and a maximum fixed-point $\nu F \subseteq \Sigma$ [17]. Observe, that the following functions are monotonic:

$$F_{\phi_1 U \phi_2, \epsilon}(S) = ([\![\phi_1]\!]\epsilon \cap [\![\bigcirc]\!]S) \cup [\![\phi_2]\!]\epsilon$$
$$F_{\phi_1 V \phi_2, \epsilon}(S) = ([\![\phi_1]\!]\epsilon \cap [\![\phi_2]\!]\epsilon) \cup ([\![\bigcirc]\!]S \cap [\![\phi_2]\!]\epsilon)$$

The until and release operators are now given by a minimum and a maximum fixed-point:

Lemma 3. ***For any formulae ϕ_1, ϕ_2 with free variables in the domain of ϵ, the following holds:***

$$[\![\phi_1 U \phi_2]\!]\epsilon = \mu F_{\phi_1 U \phi_2, \epsilon}$$
$$[\![\phi_1 V \phi_2]\!]\epsilon = \nu F_{\phi_1 V \phi_2, \epsilon}$$

For convenience, fixed-points can be written as equations. We thus write for instance, $X =_\mu F(X)$ for an equation system with the solution $X = \mu F$, see for instance [16] for an example of how this is done in the modal μ-calculus.

Using the lemma it is straightforward to write down a formula as an equivalent set of equations using only the temporal next-operator leaving out the until and the release operators. For convenience, the algorithm will work with equation systems.

3 Disjunctive Normalized Equation Systems

In this Section we present our notion of Disjunctive Normalized Equation Systems and argue why it forms a suitable basis for runtime verification of Timed LTL.

Definition 4. ***A Normal Form Equation System $\mathcal{D}$ over formula identifiers $\{X_1, \ldots, X_n\}$ and formula clocks $V = \{x_1, \ldots, x_m\}$ is a set of defining equations***

$$X_1(\boldsymbol{x}_1) = \phi_1$$
$$\vdots$$
$$X_n(\boldsymbol{x}_n) = \phi_n$$

where for all $i \in \{1, \ldots, n\}$, $\boldsymbol{x}_i$ is a (possibly empty) vector of variables from V, such that $\boldsymbol{x}_i$ includes all the free variables in ϕ_i. The right-hand sides ϕ_i are each on the following form :

$$\phi_i ::= \bigvee_{j \in J_i} \boldsymbol{x}.(\psi_{ij} \wedge \bigcirc \bigwedge_{l \in L_{ij}} X_l(\boldsymbol{x}_l))$$

where J_i is an index set, $L_{ij} \subseteq \{1, \ldots, n\}$ is a subset of indices 1 to n, ψ_{ij} is a non-temporal formula composed of the propositional part of LTL_t: atomic propositions, clock constraints and Boolean connectives, and finally $\boldsymbol{x}$ is a (possibly empty) vector of formula clocks from V.

Notice, that the actual arguments to the formula identifiers are always exactly the same as the declared formal parameters. Some of the variables might be bound by the freeze operator and their values might be restricted by constraints in the propositional part, ϕ_i.

Intuitively the normal form means the following: There is a set of possibilities each of which describes:

1. **What should hold *now* (i.e. a *propositional* part) and**
2. **Which identifiers should hold in the *next* state (i.e. a *temporal* part).**

This disjunctive normal form is particularily suitable for performing runtime verification, since here we are at all times faced with the job of 'processing' one single state and this job really implies two things: First, checking whether the current state is ok and second keeping track of what should hold for the future, i. e. what the residual formula should be. The normal form introduces the $\bigcirc$–operator as suggested by the functions $F_{\phi_1 U \phi_2, \epsilon}$ and $F_{\phi_1 V \phi_2, \epsilon}$ in Section 2. We refer to [5] for a translation procedure from Timed LTL to Disjunctive Normal Form.

Example 5. **Consider the closed LTL_t expression which states that q becomes true before the elapse of 5 time units and that p holds until then.**

$$\phi = x.((p \wedge t < x + 5)\, U\, q)$$

Rewriting to disjunctive normal form using the procedure sketched above we obtain the following equation system $\mathcal{D}_\phi$:

$$\begin{aligned} X_0() &= x.((p \wedge t < x + 5 \wedge \bigcirc X_1(x))) \vee q \\ X_1(x) &= (p \wedge t < x + 5 \wedge \bigcirc X_1(x)) \vee q \end{aligned}$$

As can be seen we use numbers rather than formulae themselves as indexes on formula identifiers. This is more convenient. For a formula ϕ we shall be using X_0 instead of X_ϕ. Sometimes we shall be refer to X_0 as the top identifier. In the example above the equations for X_0 and X_1 are not the only ones generated, in fact also equations for e.g. the propostions p and q will be constructed. However, since these are not referred to either directly or indirectly by X_0 we shall omit these for clarity reasons. Further notice that the right-hand side of X_1 is repeated in the right-hand side for X_0 due to the rewriting into normal form.

Theorem 6. ***Let ϕ be a formula in LTL_t. Then ϕ may be rewritten to a Normal Form Equation System $\mathcal{D}(\phi)$ such that for all (σ, t) it holds that***

$$(\sigma, t) \models \phi \iff (\sigma, t) \models \mathcal{D}_\phi(X_\phi)$$

The number of identifiers in a normalform equation system will b equal to the number of temporal oprators in the formula plus 1.

4 Verification using Normal Form Equations

Our runtime verification procedure consists of a residual formula construction '/' defined below. We use $\sigma_i \vdash \psi$ for denoting whether the state σ_i fulfills the non-temporal formula ψ. It will thus be either true or false.

Definition 7. ***(Residual Formula) Let ϕ be an LTL_t formula, let $\mathcal{D}_\phi$ be the Normal Form Equation System for ϕ and let σ_0 be a timed state. Then the residual of ϕ with respect to σ_0 is obtained using the following transformations***

$$\left(\bigvee_{j\in J_i} \boldsymbol{x}.(\psi_{ij} \wedge \bigcirc \bigwedge_{l\in L_{ij}} X_l(\boldsymbol{x_l})\right)/\sigma_0 = \bigvee_{j\in J_i} (\sigma_0 \vdash \psi_{ij}[t_0/\boldsymbol{x}] \wedge \bigwedge_{l\in L_ij} X_l(\boldsymbol{x_l})[t_o/\boldsymbol{x}])$$

$$\left(\bigvee_{j\in J_i} \bigwedge_{l\in L_ij} X_l(\boldsymbol{a})\right)/\sigma_0 = \bigvee_{j\in J_i} \bigwedge_{l\in L_{ij}} X_l(\boldsymbol{a})/\sigma_0$$

$$X_l(\boldsymbol{a})/\sigma_0 = \mathcal{D}\phi(X_l(\boldsymbol{x}))[\boldsymbol{a}/\boldsymbol{x}]/\sigma_0$$

In the definition above $\boldsymbol{x}$ denotes the formal parameters $x_1, \ldots, x_n$ (formula clocks) while $\boldsymbol{a}$ denotes the corresponding actual parameters. Initially we will be using $X_0(0, \ldots, 0)$ as the top identifier, which is all right since any use of a formula clock x_i will be preceeded by a freeze of this clock.

Regarding correctness, if $\phi/\sigma_0 = true$ then it holds that for all suffixes σ^1 that $\sigma_0\sigma^1 \models \phi$. And if $\phi/\sigma_0 = false$ then for no suffix σ^1 it holds that $\sigma_0\sigma^1 \models \phi$. If ϕ/σ_0 gives a result ϕ' other than the constants $true$ and $false$ then for any suffix σ^1 it holds that $\sigma^1 \models \phi'$ if and only if $\sigma_0\sigma^1 \models \phi$. In other words, our property transformer is sound. Now ideally, a tautology ϕ' should be reduced to the constant value $true$, and similarly, an unsatisfiable formulae should preferably be reduced to the constant value $false$. But this demands the incorporation of a SAT solver which would seriously hamper the efficiency of our method. Thus, our method can be said to be relative complete. Moreover, formulae which are not tautologies or falsities may be bigger than necessary. However, by application of heuristics in the style of [9, 16, 15] it is possible to apply a range of formula reductions towards a smaller specification (possibly $true$ or $false$).

In the following we shall be using the notation $\phi/^i\sigma$ to denote the residual obtained from of the the i first timed states from the complete trace σ with respect to the formula ϕ. For complete timed traces this devises an algorithm for runtime verifiation, see the following example. In the next section we shall see how to reduce the number of residual constructions when taking into account the smallest interesting timepoint.

Definition 8. ***Let $\sigma = \sigma_0\sigma_1, \ldots \sigma_i\sigma_{i+1} \ldots$ be a timed trace. Then we define***

$$\phi/^0\sigma = \phi/\sigma_0$$
$$\phi/^{i+1}\sigma = (\phi/^i\sigma)/\sigma_{i+1}$$

Example 9. **Let $\sigma = (\{p\}, 0)(\{p\}, 1)(\{p, q\}, 2)\sigma^3$ be a timed trace, and let ϕ be the formula $x.((p \wedge t < x + 5)\, U\, q)$ that states that q should hold before 5 time units and p should hold until then, see example 5. Clearly σ satisfies ϕ regardless of σ^3 and we shall now apply the residual construction to see this. Therefore we first compute the residual of $X_0() = x.((p \wedge t <$**

$x + 5) \wedge \bigcirc X_1(x)) \vee q$ **with respect to** $(\{p\}, 0)$**.**

$$
\begin{aligned}
& X_0()/(\{p\}, 0) \\
= & \mathcal{D}\phi(X_0())/(\{p\}, 0) \\
= & x.((p \wedge t < x + 5 \wedge \bigcirc X_1(x)) \vee q)/(\{p\}, 0) \\
= & ((\{p\}, 0) \vdash p \wedge t < 0 + 5) \wedge X_1(0)) \vee (\{p\}, 0) \vdash q \\
= & (true \wedge X_1(0)) \vee false \\
= & X_1(0)
\end{aligned}
$$

This means that in order for σ **to satisfy** $X_0()$ **then** σ^1 **should satisfy** $X_1(0)$**. Consequently, we therefore compute the residual of** $X_1(0)$ **and the first state of** σ^1 **namely** $(\{p\}, 1)$**.**

$$
\begin{aligned}
& X_1(0)/(\{p\}, 1) \\
= & \mathcal{D}\phi(X_1(0))/(\{p\}, 1) \\
= & (p \wedge t < 0 + 5 \wedge \bigcirc X_1(0) \vee q)/(\{p\}, 1) \\
= & (((\{p\}, 1) \vdash p \wedge t < 5) \wedge X_1(0)) \vee (\{p\}, 1) \vdash q \\
= & (true \wedge x_1(0)) \vee false \\
= & X_1(0)
\end{aligned}
$$

Still, no final conclusion may be drawn, so, we compute the residual of $X_1(0)$ **with respect to the first state of** σ^2**, i.e.** $(\{p, q\}, 2)$

$$
\begin{aligned}
& X_1(0)/(\{p, q\}, 2) \\
= & \mathcal{D}\phi(X_1(0))/(\{p, q\}, 2) \\
= & (p \wedge t < 0 + 5 \wedge \bigcirc X_1(0)) \vee q/(\{p, q\}, 2) \\
= & (((\{p, q\}, 2) \vdash p \wedge t < 5) \wedge X_1(0)) \vee (\{p, q\}, 2) \vdash q \\
= & (true \wedge X_1(0)) \vee true \\
= & true
\end{aligned}
$$

The result $true$ **means that no matter what** σ^3 **might be, then we may now once and for all conclude that** σ **satisfies** ϕ**.**

5 Smallest Interesting Timepoint

Having time constraints in the logic implies that the elapse of time alone may cause a formula to change truth value. Consider the time constraint $t < c$**, where** t **is a discrete clock, which is moving forward, and** c **is a constant. Before time** c **this constraint is true. However, at time** c **it becomes false, and moreover, it will remain false. Therefore, if we for a moment assume that time is the only changing parameter in a timed trace, we may say that** c **is the *smallest interesting timepoint* for the time constraint** $t < c$**. Similarly, the time constraint** $t \geq c$ **has the property that it is false until time becomes c, after which it becomes true, and remain so in all future. Consequently we shall say that** c **is the smallest interesting timepoint for the constraint** $t \geq c$**. As a special case we have** $t = c$ **which is false all the time except in the timepoint where the time is exactly** c**, therefore when the time is** $c + 1$ **we can conclude, that** $t = c$**, will never be true again. Henceforth,** $c+1$ **is the smallest interesting timepoint for** $c+1$**. Common for all time propositions on the form** $t \sim c$ **is that at time** c **(or** $c + 1$**) we are able to conclude that their future truth value will remain unchanged.**

The above considerations may be employed to avoid the tedious task of considering a complete timed trace. Assume a timed trace like $(\{p\}, 0)(\{p\}, 1)(\{p\}, 2)\ (\{p\}, 3)(\{p\}, 4)(\{p\}, 5)(\{p\}, 6)...$

i.e. a trace where the propositional part does not change, and assume we are interested in checking a property like $\Box(p \wedge t < 10)$. In this case it is enough to check that proposition p holds at time 0 (which it does) and then jump directly to time 10, which is the smallest interesting timepoint for the formula, since it is here where $t < 10$ changes truth value

The smallest interesting timepoint (SIT) is defined in Definition 10 on the structure of formulae.

Definition 10. ***(Smallest Interesting Timepoint) Assume a normal form equation system over n identifiers $X_1, \ldots, X_n$. Let $J, L_j \in \mathbb{N}$ and let f be a function mapping any pair (l, j) to a number in $\{1 \ldots n\}$.***

$$
\begin{aligned}
SIT(\bigvee_{j=1}^{J} \bigwedge_{l=1}^{L_j} X_{f(j,l)}(\boldsymbol{a}_{jl}), (s,t)) &= min\{SIT(D(X_{f(j,l)}(\boldsymbol{x}_{f(j,l)}))[\boldsymbol{a}_{jl}/\boldsymbol{x}_{f(j,l)}], (s,t)) \\
&\quad |1 \leq j \leq J, 1 \leq l \leq L_j\} \\
SIT(\bigvee_{j=1}^{J} \phi_j, (s,t)) &= min\{SIT(\phi_j, (s,t))\} \\
SIT(\boldsymbol{x}.(\psi \wedge \bigcirc \bigwedge_{i \in I} X_i(x)), (s,t)) &= \begin{cases} t+1 & \textbf{\textit{if }} (s,t) \vdash \psi[t/\boldsymbol{x}] \textbf{\textit{ and } } \boldsymbol{x} \textbf{\textit{ not empty}} \\ \infty & \textbf{\textit{if }} (s,t) \nvdash \psi[t/\boldsymbol{x}] \textbf{\textit{ and } } \boldsymbol{x} \textbf{\textit{ not empty}} \\ SIT(\psi, (s,t) & \textbf{\textit{if } } \boldsymbol{x} \textbf{\textit{ is empty}} \end{cases} \\
SIT(\phi_1 \wedge \phi_2, (s,t)) &= SIT(\phi_1 \vee \phi_2, (s,t)) \\
&= min\{SIT(\phi_1, (s,t)), SIT(\phi_2, (s,t))\} \\
SIT(p, (s,t)) &= SIT(\neg p, (s,t)) = \infty \\
SIT(t > c, (s,t)) &= SIT(t \leq c, (s,t)) = c + 1 \\
SIT(t = c, (s,t)) &= SIT(t < c, (s,t)) = SIT(t \geq c, (s,t)) = c
\end{aligned}
$$

The only case where the smallest interesting timepoint will be as small as $t+1$, i.e. the time in the next state, is when (1) new clock variables are bound and (2) the propositional formula ψ is satisfied, since in this case new identifiers (previously hidden behind the $\bigcirc$–operator) are now instantiated in the residual formula.

The following theorem states that the residual computation need not take place for a timed state if it has the same propositional content as its predecessor and further that its time (i.e. the current time) is below the smallest interesting timepoint for the current residual.

Theorem 11. ***Let $\sigma = \sigma_0\sigma_1\sigma^2$ be a timed trace $(\sigma_0 = (s_0, t_0)$ and $\sigma_1 = (s_1, t_1))$ and let ϕ be a LTL_t residual formula, such that $t_1 < SIT(\phi/\sigma_0)$***
then $\sigma_1, \sigma^2 \models \phi/\sigma_0$ if and only if $\sigma^2 \models \phi/\sigma_0$.

Example 12. **Let us consider the punctuality property $\Box(p \Rightarrow x.(\Diamond(t = x + 10 \wedge q)))$ [7]. After translating to disjunctive normal form we get the following equation system:**

$$
\begin{aligned}
X_0 &= \neg p \wedge \bigcirc X_0 \\
&\quad \vee x.(p \wedge true \wedge \bigcirc(X_1(x) \wedge X_0)) \\
&\quad \vee x.(p \wedge q \wedge t = x + 10 \wedge \bigcirc X_0)
\end{aligned}
$$

$$
\begin{aligned}
X_1(x) &= \bigcirc X_1(x) \\
&\quad \vee q \wedge t = x + 10
\end{aligned}
$$

Checking the trace $\sigma = (\{p\}, 0)(\{\}, 1)(\{\}, 2)(\{\}, 3) \ldots (\{\}, 8)(\{\}, 9)(\{q\}, 10) \ldots$ we get that σ^1 must satisfy the residual formula $X_1(0) \wedge X_0$. Now, using Definition 10 we have that $SIT(X_1(0) \wedge X_0, (\{p\}, 0))$ is 1. This means that we also have to compute the residual for

the next state $(\{\}, 1)$. The residual thus obtained is $X_1(0) \wedge X_0$ and as a smallest interesting timepoint we get 10 since $SIT(x_1(0), (\{\}, 1)) = 10$ and $SIT(x_0, (\{\}, 1)) = \infty$. And now, since the propositions do not change from σ_2 through σ_9 we do not have to compute the residual since it will be not change, see Theorem 11. At time 10 we will have to compute the residual since this is the next interesting timepoint and the residual now becomes X_0 because proposition q is true here. Had q not been true at time 10 the whole residual would reduce to $false$.

The example above suggests that we really need to rewrite any timed trace to its complete version, but in practice this will not happen. If the program being monitored only dumps information when the state is *changed* then the runtime checker will at any timepoint know that "no news" means that the propostional part of the state is unchanged.

6 Time Based Fixed Point Reductions

In exhaustive verification a property $\Diamond p$ (eventually p) will be concluded to be not satisfied if the state graph of the system being checked has an infinite p–free execution trace, i.e. a loop for which no states satisfies predicate p. In Runtime Verification, unfortunately, cycles go undetected and hence we are not able to reach such a conclusion. We are, however, able to do something else of great value, namely to exploit the passage of *time* as a basis of strengthening the completenes of our method. Consider for instance a formula $\Diamond(p \wedge t < 5)$ (p becomes true before time 5), which in Disjunctive Normal Form becomes the minimal fixed point equation $X =_\mu \bigcirc X \vee (p \wedge t < 5)$. Obviously, if for a given trace, p is not satisfied at neither time $0, 1, 2, 3$ or 4 then the property is not satisfied no matter what the continuation might be. Unfortunately, our OK transformer does not quite capture this. For all timed states in which p does not hold, i.e. $(\{\}, i), i \in I\!N$ we get that $X/(\{\}, i) = X$ which means that after time 4 there is still a "hope", namely if X can be satisfied (but this is really a false hope!). Therefore, at time 5 we do need to *rewrite* the defining equation of X to $X =_\mu \bigcirc X \vee false$, which again is equivalent to $false$ since for this definition of X no finite number of unfoldings exists. In very much the same fashion we may conclude that time constraints with a lower bound (i.e. $t > c, t \geq c$) from certain timepoints (which in this particular case are $c + 1$ and c) becomes true and utilize this fact to rewrite maximal fixed point equations.

In the following we therefore introduce two things. First, a formula transformer tR (At Time t Reduction) whose purpose it is, given a (possibly closed) LTL_t formula ϕ and a timed state σ_0, to attempt to determine future truthhood (or falsehood) of timing constraints inside ϕ. And second, a fixed point reduction technique which attempts to use these rewritten equations to draw conclusions.

Definition 13. ***(At Time t Reduction)***

Let $\sigma = \sigma_0\sigma_1 \ldots$ be a timed trace, let X_0 be the top identifier in an equation system on disjunctive normal form and let $\phi = \bigvee_i \bigwedge_j X_{ij}(a_{ij})$ be the residual formula obtained by computing $X_0/^{i-1}\sigma$. Then the 'At time t_i reduction of ϕ', denoted $tR(\phi, t_i)$ is defined as follows

$$
\begin{aligned}
\mathbf{t}R(\psi_1 \vee \psi_2, t_i) &= \mathbf{t}R(\psi_1, t_i) \vee \mathbf{t}R(\psi_2, t_i) \\
\mathbf{t}R(\psi_1 \wedge \psi_2, t_i) &= \mathbf{t}R(\psi_1, t_i) \wedge \mathbf{t}R(\psi_2, t_i) \\
\mathbf{t}R(\bigcirc X, t_i) &= \bigcirc X \\
\mathbf{t}R(x.\phi, t_i) &= x.\phi \\
\mathbf{t}R(p, t_i) &= p \\
\mathbf{t}R(\neg p, t_i) &= \neg p \\
\mathbf{t}R(t < c, t_i) &= \begin{cases} \textbf{\textit{false}} & \textbf{\textit{if}}\ t_i \geq c \\ t < c & \textbf{\textit{otherwise}} \end{cases} \\
\mathbf{t}R(t \leq c, t_i) &= \begin{cases} \textbf{\textit{false}} & \textbf{\textit{if}}\ t_i > c \\ t \leq c & \textbf{\textit{otherwise}} \end{cases} \\
\mathbf{t}R(t = c, t_i) &= \begin{cases} \textbf{\textit{false}} & \textbf{\textit{if}}\ t_i > c \\ t = c & \textbf{\textit{otherwise}} \end{cases} \\
\mathbf{t}R(t > c, t_i) &= \begin{cases} \textbf{\textit{true}} & \textbf{\textit{if}}\ t_i > c \\ t > c & \textbf{\textit{otherwise}} \end{cases} \\
\mathbf{t}R(t \geq c, t_i) &= \begin{cases} \textbf{\textit{true}} & \textbf{\textit{if}}\ t_i \geq c \\ t \geq c & \textbf{\textit{otherwise}} \end{cases}
\end{aligned}
$$

Returning to the example with the minimal fixed point equation $X =_\mu \bigcirc X \vee (p \wedge t < 5)$ we have that $\mathbf{t}R(\bigcirc X \vee (p \wedge t < 5), i) = \bigcirc X \vee (p \wedge t < 5)$ for $i = 0, \ldots, 4$ which nicely resembles the fact that up to time 4 there is still a hope to satisfy the property. But $\mathbf{t}R(\bigcirc X \vee (p \wedge t < 5), 5) = \bigcirc X \vee (p \wedge false) = \bigcirc X \vee false$ and thus the definition of X reduces to $X =_\mu \bigcirc X \vee false$ which does not have a solution.

In general we can say that if the righthandside of a minimal fixed point equation, $X =_\mu F(X)$, has no next–free disjuncts which are different from the constant $false$ then X is itself equivalent to false. And in our tool TimeChecker employ this to reduce the residual formula. A dual reduction for maximal fixed point equations exists: If the righthandside contains a disjunct for which the propositional requirement is not present the solution is all timed traces because the solution to $Y =_\nu \bigcirc Y$ is all timed traces.

The entire framework for runtime verification presented in this paper, is implemented in a tool called TIMECHECKER. This includes a translation from a LTL_t formula to a disjunctive normalform together with full implementation of the resisual construction with smallest interesting time point, and time based fixed point reduction. At the current stage we have checked the validity of the method on a set of smaller examples. In the future we are going to apply the method in a real environment.

7 Conclusion and Future Work

In this paper we have improved our runtime verifier for timed LTL [5] from being state-based to become event-based, making the algorithm feasible for systems with big differences in the magnitudes of the relevant time points. We have expanded our previously presented framework for runtime verification of timed LTL, in such a way that we have improved both completness and performance. Completness has been improved since we are able to detect timeouts in time constraints which is used in context of detecting if a minimial fixpoint formula can or cannot be satisfied within a limited number of formula unfoldings. The introduction of smallest interesting timepoint is a big performance improvement since we are

now able to skip processing a lot of the statesequences, from the complete trace. Therefore it is also a big step towards a logic with dense time instead of discrete time.

In [11] an automaton-based solution is presented for the limitation of having only temporal operators with upper-bounds. It would be interesting to investigate to what extent restricting our logic to only upper bounds could allow for further improvements in the algorithm. It is for instance not difficult to see that if the residual formula is for instance $X_1(3) \wedge X_1(4) \wedge X_1(5)$ and there are only upper bounds in the equations, then this will be equivalent to $X_1(3)$ (because the two last will be implied by the first).

Another type of improvement to search for, could be in a better computation of the smallest interesting timepoint in the case of freeze-operators. In the current version this will sometimes simply be the next timepoint, which is the smallest possible step to take. This might happen with formulae as $\Box(p \Rightarrow x.\phi)$, where, whenever p holds, a new freeze must be performed. If p holds for a longer period, e.g., from timepoint 0 to 1000, many freeze-operations, being reflected as identifier-instantiations with different parameter values, will be performed. Here, it might be beneficial to instead use change-propositions expressing the change of a proposition like p. These would be $p\uparrow$ for p goes "high" and $p\downarrow$ for p goes "low" and express the properties with these instead. This would make the logic more like an interval logic (as e.g. duration calculus [12]) and the timing properties could be thought more of as properties of timing diagrams known from hardware design and analysis.

In general, timed LTL is undecidable ([7]), which means that it will not be possible to guarantee completeness in the sense that the verifier should in each step always be able to reduce the residual formula to false (if it is unsatisfiable) respectively true (if it is a tautology). However, for the fragments of timed LTL that are decidable, it should be possible to improve the verifier, by using the appropriate decision procedure.

Though our framework can be used in several contexts it is constructed with ERP-systems in mind. Therefore a future development will be to expand TIMECHECKER **in such a way, that it is able to concurrently monitor several processes each having their own residual formula. In addition to this, we shall begin to address the important problem of how to *automatically instrument* an ERP system (or any other running program) with the ability to present its current timed state in a form which can be used by the** TIMECHECKER**.**

References

1. **K. Havelund, G. Ruso. Monitoring Java Programs with Java PathExplorer. First Workshop on Runtime Verification (RV'01), Paris, France, 23 July 2001. Electronic Notes in Theoretical Computer Science, Volume 55, Number 2, 2001**
2. **D. Giannakopoulou, K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. Automated Software Engineering 2001 (ASE'01), San Diego, California, 26-29 November 2001, IEEE Computer Society.**
3. **R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple onthe -fly automatic verification of linear temporal logic. Proc. 15th International Symposium on Protocol Specification, Testing and Verification (PSTV XV), pp 318, Chapman and Hall, 1995.**
4. **G. Holzmann. The Model Checker SPIN. IEEE Trans. on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.**
5. **K. Kristoffersen, C. Pedersen and H. R. Andersen. Runtime Verification of Timed LTL using Disjunctive Normalized Equation Systems. Appears in Preliminary Proceedings of Third International Workshop on Runtime Verification RV '03, Boulder, Colorado, USA. July 13, 2003. (Technical Report MS-CIS-03-21, University of Pensylvania). To appear in Issue 89.2 of Electronic Notes in Theoretical Computer Science.**
6. **Time-Rover Corp. Temporal Business Solutions. The DB Rover. 2000.**
7. **R. Alur, T. Henzinger. Logics and Models of Real Time: A Survey. Real Time: Theory in Practice, Lecture Notes in Computer Science 600, Springer-Verlag, 1992, pp. 74-106.**
8. **T. Henzinger. It's About Time: Real-Time Logics Reviewed. Proceedings of the Ninth International Conference on Concurrency Theory (CONCUR), Lecture Notes in Computer Science 1466, Springer-Verlag, 1998, pp. 439-454.**

9. **K. Havelund, G. Ruso. Monitoring Programs using Rewriting. Automated Software Engineering 2001 (ASE'01), San Diego, California, 26-29 November 2001, IEEE Computer Society.**
10. **M.C.W. Geilen, D.R. Dams. An on-the-Fly Tableau Construction for a Real-Time Temporal Logic. Proceedings of the Sixth International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT2000, 20-22 September 2000, Lecture Notes in Computer Science Vol.1926 Pune, India, Ed. M. Joseph, pp. 276-290, ©Springer-Verlag, Berlin, 2000.**
11. **M.C.W. Geilen. An improved on-the-fly tableau construction for a real-time temporal logic. Computer Aided Verification, CAV 2003, Proceedings. Boulder, Colorado, USA, July 8 – 12, 2003.**
12. **Z. Chaochen, C.A.R. Hoare, A.P. Ravn: A Calculus of Durations, Information Processing Letter, 40, 5, pp. 269-276, 1991.**
13. **E. M. Clarke, O. Grumberg, D. A. Peled. Model Checking. The MIT Press 2001.**
14. **J. S. David. Three Events that Defines an REA Methodology for Systems Analysis, Design and Implementation. 2001.**
15. **K. G. Larsen, P. Pettersson and W. Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. Proceedings of the 16th IEEE Real-Time Systems Symposium, Pisa, Italy, 5-7 December, 1995.**
16. **H. R. Andersen. Partial Model Checking (Extended Abstract). Proceedings of LICS'95. La Jolla, San Diego, June 26-29, 1995, IEEE Computer Society Press, pp. 398-407.**
17. **A. Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications, Pacific. J. Math., 55 (1955), pp. 285-309.**