TCTL Inevitability Analysis of Dense-Time Systems: *From Theory to Engineering*

Farn Wang, *Member*, *IEEE Computer Society*, Geng-Dian Huang, and Fang Yu

Abstract—Inevitability properties in branching temporal logics are of the syntax $\forall\Diamond\phi$, where ϕ is an arbitrary (timed) CTL (Computation Tree Logic) formula. Such inevitability properties in dense-time logics can be analyzed with the greatest fixpoint calculation. We present algorithms to model-check inevitability properties. We discuss a technique for early decision on greatest fixpoint calculation which has shown promising performance against several benchmarks. We have experimented with various issues which may affect the performance of TCTL inevitability analysis. Specifically, our algorithms come with a parameter for the measurement of time-progress. We report the performance of our implementation with regard to various parameter values and with or without the non-Zeno computation requirement in the evaluation of greatest fixpoints. We have also experimented with safe abstraction techniques for model-checking TCTL inevitability properties. The experiment results help us in deducing rules for setting the parameter for verification performance. Finally, we summarize suggestions for configurations of efficient TCTL inevitability evaluation procedure.

Index Terms—TCTL, real-time systems, inevitability, non-Zeno, model-checking, greatest fixpoint, abstraction.

1 Introduction

Since the model-checking theory for *timed automata* were invented roughly a decade ago [1], [17], many theoretical workpieces have been reported and experimental tools have been implemented [4], [6], [12], [13], [18], [19], [20], [21], [24], [26], [27], [28], [29], [30], [31], [32], [34], [36]. However, to apply these research results to engineering projects, hardwork is still needed in order to understand how to configure the proposed algorithms for efficient specification evaluation. The goals of this work are to design new efficient techniques for evaluating modal formulas like $\forall\Diamond\phi$, to use extensive experiments to observe how the evaluation procedure performs, in practice, and to come up with summarized suggestions for the configuration of the procedure for efficient inevitability analysis.

In verification research, two types of specification properties attract the most interest from academia and industry. The first type specifies that "*bad things will never happen*," while the second specifies that "*good things will happen*" [3]. In the branching temporal logics of (timed) *CTL (Computation Tree Logic)* [1], [10], these two concepts can be mapped to modal operators $\forall\Box$ and $\forall\Diamond$, respectively. $\forall\Box$ properties are called *safety* properties while $\forall\Diamond$ properties are usually called *inevitability* properties, [14], [22]. In the domain of dense-time system verification, people have focused on the efficient analysis of safety properties [13], [18], [20], [24], [28], [29], [30], [31], [32], [36]. Inevitability properties in *Timed CTL (TCTL)* [1], [17] are comparatively more difficult to analyze due to the following reason: To analyze an inevitability property, say $\forall\Diamond\phi$, we actually compute the set of states that satisfy the negation of the inevitability, i.e., $\exists\Box\neg\phi$, in symbols $[[\exists\Box\neg\phi]]$. We then determine the inevitability by checking the intersection emptiness between $[[\exists\Box\neg\phi]]$ and the set of the initial states. The difficulty arises if property $\exists\Box\neg\phi$ is violated by Zeno computations, which are those counterintuitive infinite computations whose execution times converge to a finite value [17]. For example, a specification such as

> "Along all computations, eventually a bus collision will happen in three time units,"

can be violated by a Zeno computation whose execution time converges to a finite timepoint, e.g., 2.9 time units. In order to respect the TCTL semantics [1], [17], we have to impose a non-Zeno requirement on computations that may incur extra complexity to the evaluation of inevitability properties. We present a symbolic TCTL model-checking algorithm that can handle non-Zeno requirements in *greatest fixpoint* evaluations. This algorithm involves nested reachability analysis and demands a much higher complexity than simple safety analysis.

To analyze TCTL inevitability properties in industrial projects, it is important to integrate various techniques for a performance solution. We investigate three approaches for such a purpose. Our first approach is a speed-up technique called *Early Decision on the Greatest Fixpoint (EDGF)*. In practice, inevitability usually happens with a precondition. For example, we may want to specify $\forall\Box(\texttt{collision} \rightarrow y.\forall\Diamond(y < 26 \wedge \texttt{idle}))$, meaning that, if the precondition of `collision` is observed, the system will inevitably enter the idle state within 26 time-units. After negation for model-checking, we instead analyze the reachability of $\texttt{collision} \wedge y.\exists\Box(y \geq 26 \vee \neg\texttt{idle})$. In evaluating this negated formula, we want to see if the greatest fixpoint for the $\exists\Box$-formula intersects with the `collision`

- *F. Wang and G.-D. Huang are with the Department of Electrical Engineering, National Taiwan University, BL 616, Nr. 1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, ROC. E-mail: farn@cc.ee.ntu.edu.tw, gdhuang@ntu.edu.tw.*
- *F. Yu is with the Department of Computer Science, College of Engineering, University of California, Santa Barbara, CA 93106. E-mail: yuf@cs.ucsb.edu.*

Manuscript received 9 Mar. 2005; revised 5 June 2006; accepted 13 June 2006; published online xxxxxx.
Recommended for acceptance by M. Chechik.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0057-0305.

state space. We do not actually have to compute the greatest fixpoint to know if the intersection is empty. Greatest fixpoint evaluation procedures usually start with a set, say Y, of states and iteratively delete states from Y until no more deletion can be made. Since the value of Y shrinks iteratively, we can check if the intersection between Y and the precondition state space has become empty at each iteration of the greatest fixpoint construction. If, at an iteration, we find the intersection already empty, further iterations for the greatest fixpoint are unnecessary and we can immediately return the current value of Y (or $false$) without affecting the model-checking results. One nice feature of EDGF is that it does not sacrifice the precision of model-checking. As reported in Section 8, significant performance improvement is shown against several benchmarks.

In the second approach, we investigate how to adjust a parameter value in our greatest fixpoint evaluation algorithms for better performance. The parameter is used for measuring the time-progress in the non-Zeno requirement. After experimenting with several benchmarks, we find that the parameter value may significantly affect the verification performances. To verify a given system and property, we may enhance the verification performance more than 300 times by carefully changing the parameter value. Another interesting observation is that, for all our benchmarks, the performance curves with regard to the parameter values are of similar shapes, independent of the number of processes. In other words, we may predict a good parameter value for parameterized systems with high concurrency by calculating good ones for systems with low concurrency.

Our third approach is using abstraction techniques [9], [35]. The just-mentioned second approach works fine with parameterized systems. For general asymmetric systems, we experimented with using abstraction techniques to predict good parameter values. For many benchmarks, the performance curves (with regard to the parameter values) for abstract evaluation have shapes similar to those for exact evaluation. We focus on the TCTL subclass $\mathrm{TCTL}^{\forall}$, in which every formula can be analyzed with safe abstraction if overapproximation is used in the evaluation of its negation.

One challenge in designing safe abstraction techniques in model-checking is making them precise enough to discern true properties while still allowing us to enhance verification performance. In previous research, people have designed many abstraction techniques for reachability analysis [4], [24], [34], [35], [36]. For model-checking formulas in $\mathrm{TCTL}^{\forall}$, abstraction precision can be a big issue because abstraction-induced imprecision can potentially be magnified when we use imprecise evaluation results of nested modal subformulas to evaluate nesting modal subformulas. In our experiments, we have checked the precision of three previously published abstraction techniques in the evaluation of TCTL inevitabilities.

In this paper, we also discuss abstract evaluation of greatest fixpoints by omitting the requirement for non-Zeno computations in TCTL semantics. As reported in Section 8, many benchmarks can still be verified even without exclusion of Zeno computations.

We have implemented these ideas in our model-checker/simulator **RED** 4.1 [28], [29], [30]. We report here extensive experiments designed to observe the effects of our proposed techniques on inevitability analysis. We have also compared our implementation with Kronos 5.1 [36], a model-checker for full TCTL. The performance data not only shows good promise for our techniques but also provides hints for the configurations of efficient TCTL model-checking algorithm for complex systems. We also experimented with the L2CAP of Bluetooth [16] to see how our techniques perform against industrial projects. Finally, in Section 10, we summarize our experiment report and make suggestions for the configuration of the efficient inevitability evaluation procedure.

2 Related work

To model real-time systems, we may assume that time is either discrete or dense [8]. In other words, the clock readings are either nonnegative integers or nonnegative reals. The discrete real-time model is appropriate for synchronous systems, i.e., systems with a single global clock that ticks every time unit. The state transaction can only happen at ticks. In [7], a method for specifying and verifying discrete real-time systems is proposed.

The timed automata model with dense-time clocks was first presented in [2]. This model is natural for asynchronous systems, i.e., systems with multiple clocks. In contrast to the discrete real-time model, the state transaction can happen at any time moment. In [1], Alur et al. showed that the TCTL model-checking problem is in PSPACE-complexity and gave an algorithm of TCTL model-checking. The algorithm is based on the region graph, whose size is bounded by $|X|! \cdot 2^{|X|} \cdot \amalg_{x\in X}(2c_x + 2)$, where X is the set of clocks and c_x is the largest constant in the constraints for clock x.

The modeling of real-time systems is the first step to applying model checking techniques. In [23], Ober et al. proposed a timed Unified Modeling Language (UML) for modeling real-time systems. They showed how to translate timed UML into timed automata for analysis. In [33], Wang and Yu proposed a timed C, a C-like language for designing real-timed systems. They also showed how to translate timed C into timed automata.

In [17], Henzinger et al. proposed an efficient symbolic model-checking algorithm for TCTL. So far, several verification tools for timed automata have been devised and implemented [18], [20], [24], [28], [29], [30], [31], [32], [36]. UPPAAL [24] is a popular tool based on DBM[1] technology. Recently, Moller has applied abstraction techniques in UPPAAL to analyze restricted inevitability properties without using modal-formula nesting [21]. The idea is to make model augmentations speed up the verification performance. Moller also shows how to extend the idea to analyze TCTL with only universal quantifications. However, no experiment has been reported on the verification of nested modal-formulas. In [5], Behrmann et al. proposed an abstraction technique which is sound and complete with regard to

1. DBM is proposed in [12] for the representation of convex state spaces of timed automata. A DBM is a two-dimensional matrix that records the difference upper bounds between clock pairs up to a certain constant.

reachability analysis, based on distinquish maximal lower and upper bounds. They report an experiment with regard to reachability analysis in UPPAAL and no experiment has been reported on the verification of nested modal-formulas.

Kronos [36] is a full DBM-based TCTL model-checker that incorporates both forward and backward reasoning capabilities. Experiments with Kronos on TCTL *bounded inevitability* (inevitabilities with specified deadline) properties are reported in [36]. No report has been made on how to enhance the performance of the general inevitability analysis. In comparison, our proposed techniques (e.g., EDGF and abstractions) handle both bounded and unbounded inevitabilities.

Our tool **RED** (version 4.1) [30] is a full TCTL model-checker/simulator with a BDD-like data structure, called *CRD (Clock-Restriction diagram)* [28], [29], [30]. Previous research with **RED** was focused on enhancing the performance of safety analysis [26], [27], [28], [29], [30].

Abstraction techniques for safety analysis have been studied in great depth since the pioneering work of Cousot and Cousot [9]. For timed automata, convex-hull overapproximation [35] has been a popular choice for DBM technology. Many overapproximation techniques for timed automata have also been reported in [4] for BDD-like data structures and in [34] specifically for CRD.

Relationships between abstraction techniques and subclasses of CTL with only universal (or existential, respectively) path quantifiers have been studied in [11]. As mentioned, the corresponding framework in TCTL is noted in [21].

3 TCTL Model-Checking

We use *TCTL model-checking* as our verification framework in which we are given a timed automaton [2] as the behavior description and a TCTL formula [1] as the specification and aim to check whether the behavior description satisfies the specification.

3.1 Timed Automata

A timed automaton is a finite-state automaton equipped with a finite set of clocks that can hold nonnegative real-values. It is structured as a directed graph whose nodes are *modes (control locations)* and whose arcs are *transitions*. The modes are labeled with *invariance conditions* while the transitions are labeled with *triggering conditions* and a set of clocks to be reset during the transitions. The invariance conditions and triggering conditions are Boolean combinations of inequalities comparing a clock with an integer. At any moment, a timed automaton can stay in only one *mode* (or *control location*). In its operation, one transition can be triggered when a corresponding triggering condition is satisfied. Upon being triggered, the automata instantaneously transits from one mode to another and resets some clocks to zero. Between transitions, all clocks increase readings at a uniform rate.

For convenience, given a set Q of modes and a set X of clocks, we use $B(Q,X)$ as the set of all Boolean combinations of atoms of the forms q and $x \sim c$, where $q \in Q$, $x \in X \cup 0$, "$\sim$" is either $<$, $\leq$, $=$, $>$, or $\geq$, and c is an integer constant. Every elements in $B(Q,X)$ is called a *state predicate* that represents a set of states (i.e., a state space). A *valuation* of a set is a mapping from the set to another set. $\mathcal{R}^+$ denotes the set of nonnegative reals.

Definition 1. *A timed automaton A is given as a tuple $\langle X, Q, I, \mu, T, \tau, \pi\rangle$ with the following restrictions: X is the set of clocks. Q is the set of modes. $I \in B(Q,X)$ is the initial condition. $\mu : Q \mapsto B(\emptyset, X)$ defines the invariance condition of each mode. $T \subseteq Q \times Q$ is the set of transitions. $\tau : T \mapsto B(\emptyset, X)$ and $\pi : T \mapsto 2^X$, respectively, define the triggering condition and the clock set to reset of each transition.*

Definition 2. *Given an $\eta \in B(Q,X)$ and a valuation ν of X, we say ν* satisfies *η, in symbols $\nu \models \eta$, iff it is the case that, when the variables in η are interpreted according to ν, η will be evaluated as* true.

Definition 3. *A state ν of $A = \langle X, Q, I, \mu, T, \tau, \pi\rangle$ is a valuation of $X \cup Q$ such that*

- *there is a unique $q \in Q$ such that $\nu(q) = true$ and, for all $q' \neq q$, $\nu(q') = false$;*
- *$\forall x \in X$, $\nu(x) \in \mathcal{R}^+$, and $\forall q \in Q, \nu(q) \Rightarrow \nu \models \mu(q)$.*

Given state ν and $q \in Q$ such that $\nu(q) = true$, we call q the mode of ν, in symbols ν^Q. For any $t \in \mathcal{R}^+$, $\nu + t$ is a state identical to ν except that, for every clock $x \in X$, $\nu(x) + t = (\nu + t)(x)$. Given $\bar{X} \subseteq X$, $\nu\bar{X}$ is a new state identical to ν except that, for every $x \in \bar{X}$, $\nu\bar{X}(x) = 0$.

Definition 4. *Given a timed automaton $A = \langle X, Q, I, \mu, T, \tau, \pi\rangle$, a* run *is an infinite sequence of state-time pairs, (ν_0, t_0) $(\nu_1, t_1)\ldots(\nu_k, t_k)\ldots\ldots$, such that $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically increasing real-number (time) divergent sequence and, for all $k \geq 0$,*

- *invariance conditions are preserved in each interval, that is, for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \mu(\nu_k^Q)$, and*
- *either no transition happens at time t_k, that is, $\nu_k^Q = \nu_{k+1}^Q$ and $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$, or a transition happens at t_k, that is,*
  - *there is such a transition, i.e., $(\nu_k^Q, \nu_{k+1}^Q) \in T$; and*
  - *the transition is satisfied, i.e., $\nu_k + (t_{k+1} - t_k) \models \tau(\nu_k^Q, \nu_{k+1}^Q)$ and*
  - *the clocks are reset accordingly, i.e., $(\nu_k + (t_{k+1} - t_k))\pi(\nu_k^Q, \nu_{k+1}^Q) = \nu_{k+1}$.*

3.2 Timed Computation Tree Logic (TCTL)

TCTL [1], [17] is a branching temporal logic for the specification of dense-time systems. The formal semantics of TCTL model-checking could be defined as follows:

Definition 5. *A TCTL formula ϕ has the following syntax rules.*

$$\phi ::= \eta \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid x.\phi_1 \mid \exists\phi_1 \mathcal{U} \phi_2 \mid \exists\Box\phi_1.$$

Here, $\eta \in B(Q,X)$ and ϕ_1, ϕ_2 are TCTL formulas.

The modal operators are intuitively explained in the following:

- $x.\phi$ means that "if there is a clock x with reading zero now, then ϕ is satisfied."
- $\exists$ means "there exists a run."
- $\phi_1 \mathcal{U} \phi_2$ means that, along a computation, ϕ_1 is true until ϕ_2 becomes true.

- $\Box\phi_1$ means that, along a computation, ϕ_1 is always true.

Besides the standard shorthand of temporal logics [1], [17], we adopt the following for TCTL: $\exists\Diamond\phi_1$ for $\exists true\,\mathcal{U}\phi_1$, $\forall\Box\phi_1$ for $\neg\exists\Diamond\neg\phi_1$, $\forall\phi_1\,\mathcal{U}\phi_2$ for $\neg((\exists(\neg\phi_2)\,\mathcal{U}\neg(\phi_1 \vee \phi_2)) \vee (\exists\Box\neg\phi_2))$, and $\forall\Diamond\phi_1$ for $\forall true\,\mathcal{U}\phi_1$.

Two subclasses of TCTL, including TCTL$^\forall$ and TCTL$^\exists$, will be used in Section 7. TCTL$^\forall$ [21] is the universal fragment of TCTL such that only universal path quantifications are used and negations only appear before atoms. Similarly, TCTL$^\exists$ is the existential fragment of TCTL such that only existential path quantifications are used and negations only appear before atoms. Note that the negations of formulas in TCTL$^\forall$ fall correctly in TCTL$^\exists$. A TCTL$^\exists$ formula can express a specification that can be witnessed by a run.

Definition 6. *We write in notations $A, \nu \models \phi$ to mean that ϕ is satisfied at state ν in a timed automaton A. The satisfaction relation is defined inductively as follows:*

- *When $\phi_1 \in B(Q, X)$, $A, \nu \models \phi_1$ according to Definition 2.*
- *$A, \nu \models \phi_1 \vee \phi_2$ iff either $A, \nu \models \phi_1$ or $A, \nu \models \phi_2$.*
- *$A, \nu \models \neg\phi_1$ iff $A, \nu \not\models \phi_1$.*
- *$A, \nu \models x.\phi_1$ iff $A, \nu\{x\} \models \phi_1$. We introduce an additional clock x in A, which is reset to 0 at present.*
- *$A, \nu \models \exists\phi_1\mathcal{U}\phi_2$ iff there exists a run $(\nu_1, t_1)(\nu_2, t_2)\ldots$ such that $\nu_1 = \nu$ in A and there exist an $i \geq 1$ and a $\delta \in [0, t_{i+1} - t_i]$, s.t.*
  - *$A, \nu_i + \delta \models \phi_2$,*
  - *for all j, δ', if either $(1 \leq j < i) \wedge (\delta' \in [0, t_{j+1} - t_j])$ or $(j = i) \wedge (\delta' \in [0, \delta))$, then $A, \nu_j + \delta' \models \phi_1$.*

  In other words, ν satisfies $\exists\phi_1\mathcal{U}\phi_2$ iff there exists a run from ν such that along the run, ϕ_1 is true until ϕ_2 is true.
- *$A, \nu \models \exists\Box\phi_1$ iff there exists a run $(\nu_1, t_1)(\nu_2, t_2)\ldots$ such that $\nu_1 = \nu$ in A, and for every $i \geq 1$ and $\delta \in [0, t_{i+1} - t_i]$, $A, \nu_i + \delta \models \phi_1$. In other words, ν satisfies $\exists\Box\phi_1$ iff there exists a run from ν such that ϕ_1 is always true.*

A timed automaton A satisfies a TCTL formula ϕ, in symbols $A \models \phi$, iff for every state $\nu_0 \models I$, $A, \nu_0 \models \phi$.

4 Model-Checking Algorithm with Non-Zeno Requirements

Our TCTL model-checking algorithm uses backward reasoning. We need two basic procedures, `xtion_bck()` for the computation of the weakest precondition of transitions and `time_bck()` for that of backward time-progression. These two procedures are important in the symbolic construction of backward reachable state space representations. Various presentations of the two procedures can be found in [17], [26], [27], [28], [29], [30], [32]. Given a state space representation η and a transition e, the first procedure, $\texttt{xtion\_bck}(\eta, e)$, computes the weakest precondition

- in which every state satisfies the invariance condition imposed by $\mu()$ and
- from which we can transit to states in $[[\eta]]$ through e.

Note that $[[\eta]]$ is the set of states that satisfy η. The second procedure, $\texttt{time\_bck}(\eta_1, \eta_2)$, computes the space representation of states

- from which we can go to states in $[[\eta_2]]$ simply by time-passage and
- every state in the time-passage satisfies η_1 and also satisfies the invariance condition imposed by $\mu()$.

We have implemented the symbolic characterization of $\texttt{time\_bck}(\eta_1, \eta_2)$ presented in [17] as follows:

$$\texttt{time\_bck}(\eta_1, \eta_2) =$$
$$\{\nu | \exists\delta \in \mathcal{R}^+(\nu + \delta \models \eta_2 \wedge \forall 0 \leq \delta' \leq \delta(\nu + \delta' \models \eta_1 \wedge \mu(\nu^Q)))\}.$$

Note that the inner quantification rules out the case of path condition discontinuity in time-passage. With these two basic procedures, we can construct a backward reachability procedure $\texttt{rch-bck}(\eta_1, \eta_2)$, as shown in [17], [26], [27], [28], [29], [30], [32], which characterizes the backwardly reachable state space from states in $[[\eta_2]]$ through runs along which all states satisfy η_1. $\texttt{rch-bck}(\eta_1, \eta_2)$ can be defined as the least fixpoint of the equation

$$Y = \eta_2 \vee (\texttt{time\_bck}(\eta_1, Y \vee (\eta_1 \wedge \bigvee_{e \in T} \texttt{xtion\_bck}(Y, e)))),$$

i.e.,

$$\texttt{rch-bck}(\eta_1, \eta_2) \equiv \texttt{lfp}Y.(\eta_2 \vee (\texttt{time\_bck}(\eta_1, Y \vee (\eta_1 \wedge \bigvee_{e \in T} \texttt{xtion\_bck}(Y, e))))).$$

Our model-checking algorithm is extended from the classical model-checking algorithm for TCTL [17]. The design of our greatest fixpoint evaluation algorithm with consideration of non-Zeno requirement is based on the following lemma.

Lemma 7. *Given $d \geq 1$, $A, \nu \models \exists\Box\eta$ iff there exists a set Y of states, that all satisfy η such that $\nu \in Y$ and, for all $\nu' \in Y$, there is a finite run segment from ν' of duration $\geq d$ ending in Y.*

Proof. $(\Rightarrow)$ We first assume that $A, \nu \models \exists\Box\eta$ is true. According to the semantics of TCTL, there is an infinite and divergent run Π from ν along which η is always true. Let Y be the set of states in the run Π. Clearly, $\nu \in Y$ and for all $\nu' \in Y$ satisfies η. Considering a state $\nu' \in Y$ in the run Π, there is a finite run segment from ν' of duration $\geq d$ ending in a state $\nu'' \in Y$ in the run Π. Thus, this direction of the lemma is proven.

$(\Leftarrow)$ We next assume that there is a set Y of states that all satisfy η such that $\nu \in Y$ and, for all $\nu' \in Y$, there is a finite run segment from ν' of duration $\geq d$ ending in Y. Since $\nu \in Y$, there are infinitely many finite run segments of duration $\geq d \geq 1$ that can be concatenated together to form an infinite and divergent run starting at ν. Moreover, all states in the infinite and divergent run satisfy η. We can see that $A, \nu \models \exists\Box\eta$. □

With Lemma 7, $\exists\Box\eta$ can be defined with the following greatest fixpoint:

$$\exists\Box\eta \equiv \texttt{gfp}Y.(\texttt{ZC}.\ \texttt{rch-bck}(\eta, Y \wedge \texttt{ZC} \geq d)).$$

Here, clock ZC is an auxiliary clock variable specifically used to measure the non-Zeno requirement. Note that d is a parameter for measuring the time-progress. The following procedure can construct the greatest fixpoint satisfying $\exists\Box\eta$ with a non-Zeno requirement.

```
gfp(η) /* d is a static parameter for measuring
        time-progress */ {
    $Y := \eta$; $Y' := true$;
    repeat until $Y = Y'$, {$Y' := Y$; $Y :=$ clk_elim(
      $\mathtt{ZC} = 0 \wedge$ rch-bck($\eta, Y \wedge \mathtt{ZC} \geq d$), ZC); }          (1)
    return $Y$;
}
clk_elim(η, x) {
    for each $x_1 - x \sim c$ and $x - x_2 \sim' c'$, if $\eta \wedge$
    $x_1 - x \sim c \wedge x - x_2 \sim' c'$ is not empty, {
      $\eta_1 := \eta \wedge x_1 - x \sim c \wedge x - x_2 \sim' c$;
      $\eta := \eta \wedge \neg\eta_1$; $\eta := \eta \vee (\eta_1 \wedge x_1 - x_2$new_ub$(\sim, c, \sim', c'))$;
    }
    return $\eta$;
}
new_ub(~, c, ~', c') {
  if $c = \infty \vee c' = \infty$, return "$< \infty$"
  else if $c = -\infty$, { if $c' \leq 0$, return "$< -\infty$"; else return
      "$< -C_{A:\phi} + c$"; }
  else if $c' = -\infty$, { if $c \leq 0$, return "$< -\infty$"; else return
      "$< -C_{A:\phi} + c$"; }
  $c_r := c + c'$;
  if $\sim$ or $\sim'$ is "$<$", $\sim_r$ is assigned "$<$", else $\sim_r$
    is assigned "$\leq$"
  if $c_r > C_{A:\phi} \vee (\sim_r = "<" \vee c_r = C_{A:\phi})$, return "$< \infty$";
  else if $c_r < -C_{A:\phi} \vee (\sim_r = "<" \vee c_r = -C_{A:\phi})$,
  return "$< -\infty$";
  else return "$\sim_r c_r$";
}
```

Here, clk_elim() applies Fourier-Motzkin elimination [15] to remove a clock from a state predicate without losing information on other clocks. We assume that $\sim$ and $\sim'$ represent operations $<$ or $<=$. Other operations, i.e., $=$, $>$, $>=$, can be handled similarly. c and c' are integers in $\{-\infty, -C_{A:\phi}, \dots, C_{A:\phi}, \infty\}$, where $C_{A:\phi}$ is the maximum constant in A and ϕ. new_ub() computes the new upperbound as the result of adding two. When the new upperbound exceeds $C_{A:\phi}$, we treat it as ∞. For example, with $C_{A:\phi} = 5$, new_ub$(<, 2, \leq, 3) = "< \infty"$ and new_ub$(\leq, 2, \leq, 1) = "\leq 3"$. gfp() iteratively eliminates subspaces which cannot go to a state in Y through finite runs of duration over d. In Section 8, we show that, in many cases, d-values significantly influence verification performances.

The following model-check(A, ϕ), which uses gfp() in the labeling algorithm in [1], [17] to replace the evaluation of $\exists\Box$-formulas, stands for the complete model-checking algorithm with non-Zeno requirement. Eval(A, χ, ϕ) computes a state predicate representing the set of states that satisfy ϕ. Correctness follows from Lemma 7.

```
model-check(A, φ) { if Eval$(A, \emptyset, \neg\phi) \wedge I$ is $false$, return $true$;
    else return $false$. }
Eval$(A, \chi, \bar{\phi})$ /* $\chi$ is the set of clocks in the scope of $\bar{\phi}$ */ {
    switch ($\bar{\phi}$) {
      case ($false$): return $false$;
      case ($p$): return $p \wedge \bigwedge_{x \in \chi} x \geq 0$;
      case($x - y \sim c$): return $x - y \sim c \wedge_{x \in \chi} x \geq 0$;
      case($\phi_1 \vee \phi_2$): return Eval$(A, \chi, \phi_1) \vee$ Eval$(A, \chi, \phi_2)$;
      case ($\phi_1 \wedge \phi_2$): return Eval$(A, \chi, \phi_1) \wedge$ Eval$(A, \chi, \phi_2)$;
      case ($\neg\phi_1$): return $\neg$Eval$(A, \chi, \phi_1)$;
      case ($x.\phi_1$): return
        clk_elim$(x = 0 \wedge$ Eval$(A, \chi \cup \{x\}, \phi_1 \wedge x \geq 0), x)$;
      case ($\exists\phi_1 \mathcal{U} \phi_2$): return
        rch-bck(Eval$(A, \chi, \phi_1)$, Eval$(A, \chi, \phi_2)$);
      case ($\exists\Box\phi_1$): return gfp(Eval$(A, \chi, \phi_1)$);
}}
```

5 Early Decision on Greatest Fixpoint Evaluation

As mentioned in Section 1, inevitability properties usually appear together with preconditions, like $\forall\Box(p \rightarrow \forall\Diamond q)$. After negating for model-checking, we end up with the reachability of $p \wedge \exists\Box\neg q$ and the greatest fixpoint evaluation can stop when the intersection between p and $\exists\Box\neg q$ becomes empty. Based on this idea, we have developed the speed-up technique of *Early Decision on the Greatest Fixpoint (EDGF)*. We first need to define several terms for the presentation of the technique. A *positive Boolean tree predicate (PBTP)* with n arguments is an expression constructed from Boolean conjunctions, disjunctions, and arguments $\alpha_1, \dots, \alpha_n$. We can construct a parsing tree for a PBTP. It can be shown that every interior node is either a conjunction or a disjunction and each argument in a PBTP corresponds to exactly one leaf in the parsing tree. A nice property of PBTP is that it is monotonic. The following lemma establishes this monotonicity.

Lemma 8. *Given a PBTP $p(\alpha_1, \dots, \alpha_n)$ and state predicates $\eta_1, \dots, \eta_n, \eta'_1, \dots, \eta'_n$, if $\eta_1 \subseteq \eta'_1, \dots,$ and $\eta_n \subseteq \eta'_n$, then $p(\eta_1, \dots, \eta_n) \subseteq p(\eta'_1, , \eta'_n)$.*

Proof. This lemma can be proven with structural induction on $p()$. In the base case, there is only one argument and the lemma is true according to the assumption. For the induction, we assume that the lemma is true for all PBTPs with no more than k arguments. Now, we are going to prove that the lemma is true for a $(k+1)$-arguments PBTP $p(\eta_1, \dots, \eta_{k+1})$. If the root of the parsing tree for $p(\eta_1, \dots, \eta_{k+1})$ is a conjunction, $p(\eta_1, \dots, \eta_{k+1})$ must be of the form $p_1(\alpha_1, \dots, \alpha_h) \wedge p_2(\alpha_{h+1}, \dots, \alpha_{k+1})$, where $h \leq k$. According to the assumption, $p_1(\eta_1, \dots, \eta_h) \subseteq p_1(\eta'_1, \dots, \eta'_h)$ and $p_2(\eta_{h+1}, \dots, \eta_{k+1}) \subseteq p_2(\eta'_{h+1}, \dots, \eta'_{k+1})$. According to the definition of conjunction, we then infer that

$$p_1(\eta_1, \dots, \eta_h) \wedge p_2(\eta_{h+1}, \dots, \eta_{k+1}) \subseteq p_1(\eta'_1, \dots, \eta'_h) \wedge p_2(\eta'_{h+1}, \dots, \eta'_{k+1}).$$

Thus, the lemma is proven in this case. The proof when the root is a disjunction is similar to the conjunction case. □

Suppose we have the parsing tree Γ of a PBTP $p(\alpha_1, \dots, \alpha_n)$. A path $\kappa_1\kappa_2 \dots \kappa_m$ in Γ is called a *filtering path* for α_i, $1 \leq i \leq n$, iff κ_1 is the root of Γ and κ_m is the parent of α_i. A subformula ϕ is called a *filtering conjunct* of α_i iff the root of its parsing tree is a sibling of one of

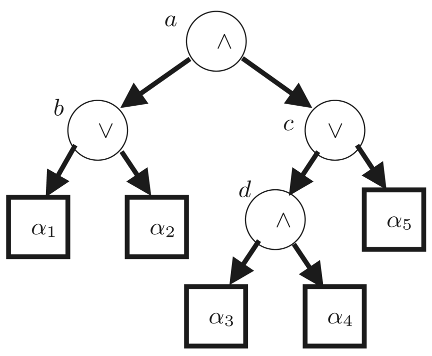

Fig. 1. The parsing tree of PBTP $(\alpha_1 \vee \alpha_2) \wedge ((\alpha_3 \wedge \alpha_4) \vee \alpha_5)$.

$\kappa_2, \ldots, \kappa_n$ and the parent of the root is a conjunction. If $\phi_1, \ldots, \phi_k$ are some filtering conjuncts of α_i, then $\bigwedge_{1 \le i \le k} \phi_i$ is called a *filtering subconjunction* of α_i. We have an example PBTP $(\alpha_1 \vee \alpha_2) \wedge ((\alpha_3 \wedge \alpha_4) \vee \alpha_5)$ in Fig. 1. Path acd is a filtering path for α_3. $\alpha_1 \vee \alpha_2$ and α_4 are both filtering conjuncts of α_3. Thus, $\alpha_1 \vee \alpha_2$, α_4, and $(\alpha_1 \vee \alpha_2) \wedge \alpha_4$ are all filtering subconjunctions of α_3.

For convenience of presentation, in the following, we shall assume that we are interested in the filtering path of α_n in a PBTP $p(\alpha_1, \ldots, \alpha_n)$. But, our lemmas and proofs can be adapted to the general case by argument renaming.

Lemma 9. *Suppose we are given a PBTP $p(\alpha_1, \ldots, \alpha_n)$ such that $\beta = f(\alpha_1, \ldots, \alpha_m)$, $m < n$, is a filtering subconjunction of α_n. Suppose that the greatest fixpoint evaluation of $\exists\Box\phi_1$ yields $\gamma_1\gamma_2\ldots\gamma_g$ (γ_g is the greatest fixpoint) in successive iterations and $\eta_1, \ldots, \eta_{n-1}$ are state predicates. If $f(\eta_1, \ldots, \eta_m) \wedge \gamma_k = false$ for some $1 \le k \le g$, then $p(\eta_1, \ldots, \eta_{n-1}, \gamma_g) \equiv p(\eta_1, \ldots, \eta_{n-1}, false)$.*

Proof. If we rewrite $p(\alpha_1, \ldots, \alpha_n)$ in DNF, according to the distribution law of Boolean algebra, $f(\alpha_1, \ldots, \alpha_m)$ will occur in every conjunction in which α_n occurs. According to the idempotence of conjunctions, we infer that $p(\alpha_1, \ldots, \alpha_n) \equiv p(\alpha_1, \ldots, \alpha_n \wedge f(\alpha_1, \ldots, \alpha_m))$. Since the evaluation of $\exists\Box\phi_1$ shrinks in each iteration, $p(\eta_1, \ldots, \eta_{n-1}, \gamma_g) \subseteq p(\eta_1, \ldots, \eta_{n-1}, \gamma_k)$ according to Lemma 8. If $f(\eta_1, \ldots, \eta_m) \wedge \gamma_k = false$, we can infer that

$$\begin{aligned} p(\eta_1, \ldots, \eta_{n-1}, \gamma_g) &\subseteq p(\eta_1, \ldots, \eta_{n-1}, \gamma_k) \\ &\equiv p(\eta_1, \ldots, \eta_{n-1}, \gamma_k \wedge f(\eta_1, \ldots, \eta_m)) \\ &\equiv p(\eta_1, \ldots, \eta_{n-1}, false). \end{aligned}$$

Since $p()$ has no negations, we also infer that $p(\eta_1, \ldots, \eta_{n-1}, false) \subseteq p(\eta_1, \ldots, \eta_{n-1}, \gamma_g)$. This implies that $p(\eta_1, \ldots, \eta_{n-1}, \gamma_g) \equiv p(\eta_1, \ldots, \eta_{n-1}, false)$. □

In the following, we take $\bar{\phi} = (a \vee b) \wedge ((\exists\Box(p \wedge \exists\Box\neg q) \wedge r) \vee s)$ as an example to illustrate the EDGF technique. The space representation of states that satisfy $\bar{\phi}$ can be evaluated by

$$\begin{aligned} &\texttt{Eval}(A, \emptyset, \bar{\phi}) = \\ &(a \vee b) \wedge ((\texttt{Eval}(A, \emptyset, \exists\Box(p \wedge \exists\Box\neg q)) \wedge r) \vee s). \end{aligned}$$

Remember that $\texttt{Eval}(A, \emptyset, a)$, $\texttt{Eval}(A, \emptyset, b)$, $\texttt{Eval}(A, \emptyset, r)$, and $\texttt{Eval}(A, \emptyset, s)$ equal a, b, r, and s, respectively. The evaluation of $\bar{\phi}$ can be expressed as the PBTP in Fig. 1 by substituting a, b, $\texttt{Eval}(A, \emptyset, \exists\Box(p \wedge \exists\Box\neg q))$, r, and s for $\alpha_1, \ldots, \alpha_5$, respectively. Since $((a \vee b) \wedge r)$ is a filtering subconjunction of the greatest fixpoint evaluation $\texttt{Eval}(A, \emptyset, \exists\Box(p \wedge \exists\Box\neg q))$, we can stop the greatest fixpoint evaluation whenever their intersection becomes empty based on Lemma 9. We rewrite procedure $\texttt{Eval}()$ by introducing a new parameter β to carry the information of the filtering subconjunction. The new procedures are in the following:

```
Eval-EDGF($A, \chi, \beta, \bar{\phi}$) {
  switch ($\bar{\phi}$) {
  case ($false$): return $false$;
  case ($p$): return $p \wedge \bigwedge_{x \in \chi} x \ge 0$;
  case ($x - y \sim c$): return $x - y \sim c \wedge_{x \in \chi} x \ge 0$;
  case ($\phi_1 \vee \phi_2$):
      return Eval-EDGF($A, \chi, \beta, \phi_1$) $\vee$ Eval-EDGF($A, \chi, \beta, \phi_2$);
  case ($\phi_1 \wedge \phi_2$):
      if $\phi_2$ does not contain modal operator,{
          $\eta_2$ := Eval-EDGF($A, \chi, \beta, \phi_2$);
          return $\eta_2 \wedge$ Eval-EDGF($A, \chi, \beta \wedge \eta_2, \phi_1$); }      (2)
      else {
          $\eta_1$ := Eval-EDGF($A, \chi, \beta, \phi_1$);
          return $\eta_1 \wedge$ Eval-EDGF($A, \chi, \beta \wedge \eta_1, \phi_2$); }      (3)
  case ($\neg\phi_1$): return $\neg$Eval-EDGF($A, \chi, true, \phi_1$);
  case ($x.\phi_1$): return clk_elim($x = 0 \wedge$ Eval-EDGF($A$,
      $\chi \cup \{x\}, true, \phi_1 \wedge x \ge 0$), $x$);
  case ($\exists\phi_1\mathcal{U}\phi_2$): return rch-bck(Eval-EDGF($A, \chi, true, \phi_1$),
    Eval-EDGF($A, \chi, true, \phi_2$));
  case ($\exists\Box\phi_1$):
    return gfp_EDGF(Eval-EDGF($A, \chi, true, \phi_1$), $\beta$);      (4)
} }
gfp_EDGF($\eta, \beta$) {
    $Y := \eta$; $Y' := true$;
    repeat until $Y = Y'$ or $(Y \wedge \beta) = false$, { $Y' := Y$;      (5)
      $Y$ := clk_elim(ZC = 0 $\wedge$ rch-bck($\eta, Y \wedge$ ZC $\ge d$), ZC); }
    return $Y \wedge \beta$;      (6)
}
```

For the case $(\phi_1 \wedge \phi_2)$, we strengthen β depending on the evaluation orders of the two conjuncts in (2) and (3). Remember that a PBTP is an expression constructed from Boolean conjunctions and disjunctions. The evaluations of case $(\neg\phi_1)$, $(x.\phi_1)$, $(\exists\phi_1\mathcal{U}\phi_2)$, and $(\exists\Box\phi_1)$ must be the arguments of a PBTP and the evaluations of subformula ϕ_1 and ϕ_2 shall be expressed by other PBTPs. Consequently, when evaluating ϕ_1 and ϕ_2, β is reset to $true$. For example, as mentioned above, $\texttt{Eval-EDGF}(A, \emptyset, (a \vee b) \wedge r, \exists\Box(p \wedge \exists\Box\neg q))$ is an argument of the PBTP for the evaluation of $(a \vee b) \wedge ((\exists\Box(p \wedge \exists\Box\neg q) \wedge r) \vee s)$. When evaluating the subformula $(p \wedge \exists\Box\neg q)$, we reset β to $true$ in (4). Evaluation $\texttt{Eval-EDGF}(A, \emptyset, true, p \wedge \exists\Box\neg q)$ is expressed by another PBTP $p(\alpha_1, \alpha_2) = \alpha_1 \wedge \alpha_2$ with $\alpha_1 = p$ and $\alpha_2 = \texttt{Eval-EDGF}(A, \emptyset, p, \exists\Box\neg q)$. In procedure $\texttt{gfp\_EDGF}()$, we check $(Y \wedge \beta) = false$ to decide for early termination in (5). In (6), we return $Y \wedge \beta$, since $p(\alpha_1, \ldots, \gamma_g) \equiv p(\alpha_1, \ldots, \gamma_g \wedge f(\alpha_1, \ldots, \alpha_m))$ according to the proof of Lemma 9. The following lemma helps establish the correctness of our implementation, i.e., β does carry the information of the filtering subconjunction.

Lemma 10. *Assume that there is a formula* $\bar{\phi}$ *and its evaluation* $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ *constructs a PBTP* $p(\alpha_1, \ldots, \alpha_n)$*, i.e.,* $\alpha_i = \texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i)$ *and* $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ *leads to a call* $\texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i)$ *for all* $1 \leq i \leq n$*. Then,* β_i *is a filtering subconjunction of* α_i*.*

Proof. We prove that β_i is a filtering subconjunction of α_i by structural induction on the PBTP $p(\alpha_1, \ldots, \alpha_n)$. In the base case, there is only one argument. According to procedure $\texttt{Eval-EDGF}()$, $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ is the argument. Since *true* is a filtering subconjunction of every subformula, the lemma automatically holds.

The inductive hypothesis is that this lemma holds for $p()$ with k arguments. Now, we have to prove that, if $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ constructs a PBTP $p(\alpha_1, \ldots, \alpha_{k+1})$, where $\alpha_i = \texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i)$ and $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ leads to a call $\texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i)$ for all $1 \leq i \leq k+1$, then β_i is a filtering subconjunction of α_i. Assume that $\kappa_1\kappa_2 \ldots \kappa_l$ is the filtering path of α_i and α_j is the sibling of α_i. According to procedure $\texttt{Eval-EDGF}()$, we have the following two cases to analyze.

- κ_l *is a disjunction*: According to $\texttt{Eval-EDGF}()$, we know

  $$\texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_j \vee \phi_i) = \texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_j) \vee \texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i).$$

  We also know that $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ constructs a PBTP $p'()$ with k arguments, which is the same as $p()$ except the subtree rooted in κ_l in $p()$ is a leaf $\texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_j \vee \phi_i)$ in $p'()$. By induction hypothesis, β_i is a filtering subconjunction for $\texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_j \vee \phi_i)$. According to the definition, β_i is a filtering subconjunction of $\texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i)$, i.e., α_i.
- κ_l *is a conjunction*: Let $\eta = \texttt{Eval-EDGF}(A, \emptyset, \beta_j, \phi_j)$. There are two subcases. First, if ϕ_j is evaluated before ϕ_i,

  $$\texttt{Eval-EDGF}(A, \emptyset, \beta', \phi_j \wedge \phi_i) = \eta \wedge \texttt{Eval-EDGF}(A, \emptyset, \beta_i, \phi_i),$$

  where $\beta_i = \beta' \wedge \eta$ according to $\texttt{Eval-EDGF}()$. We know that $\texttt{Eval-EDGF}(A, \emptyset, true, \bar{\phi})$ constructs a PBTP $p'()$ with k arguments, which is the same as $p()$ except that the subtree rooted in κ_l in $p()$ is a leaf $\texttt{Eval-EDGF}(A, \emptyset, \beta', \phi_j \wedge \phi_i)$ in $p'()$. By induction hypothesis, β' is a filtering subconjunction for $\texttt{Eval-EDGF}(A, \emptyset, \beta', \phi_j \wedge \phi_i)$. By definition, β_i is a filtering subconjunction of α_i, since κ_l is a conjunction and $\beta_i = \beta' \wedge \eta$. The case where ϕ_i is evaluated before ϕ_j is similar. □

6 Greatest Fixpoint Computation by Tolerating Zenoness

In practice, the greatest fixpoint computation procedures presented in the last two sections can be costly in terms of computing resources due to their characterizations having a least fixpoint nested in a greatest fixpoint. Such characterizations are necessary to guarantee that only non-Zeno computations are considered. In reality, systems with well-designed behaviors may satisfy certain inevitability properties for both Zeno and non-Zeno computations. In such cases, we can benefit from the following less expensive procedure for computing the greatest fixpoint:

$$\exists\Box\eta \equiv \texttt{gfp}Y.(\texttt{time\_bck}(\eta, \eta \wedge \bigvee_{e \in T} \texttt{xtion\_bck}(Y, e))).$$

We can also combine such an overapproximation with EDGF as follows:

```
gfp_Zeno_EDGF($\eta, \beta$) {
  $Y := \eta; Y' := true;$
  repeat until $Y = Y'$ or $(Y \wedge \beta) = false$, { $Y' := Y;$
      $Y := Y \wedge \texttt{time\_bck}(\eta, \eta \wedge \bigvee_{e \in T} \texttt{xtion\_bck}(Y, e));$}
  return $Y \wedge \beta$;
}
```

7 Abstract Model-Checking with $\mathrm{TCTL}^{\forall}$

In the application of abstraction techniques, it is important to make them *safe* [35]. That is to say, when the safe abstraction analyzer says a property is true, the property is indeed true. (But, when it says false, we do not know whether the property is true.) There are two types of abstractions: *overapproximation*, which means that the abstract state space is a superset of the concrete state space, and *underapproximation*, which means that the abstract state space is a subset of the concrete state space. To make an abstraction safe, we should overapproximate when evaluating $\exists\Box\neg\phi$ (the negation of the inevitability). However, negations deeply nested in formulas can turn overapproximations into underapproximations and, thus, make abstraction unsafe.

To guarantee safe abstraction in model-checking, people focused on subclass $\mathrm{TCTL}^{\forall}$ of TCTL [11], [21]. For example, we may write a $\mathrm{TCTL}^{\forall}$ formula $\forall\Box(\texttt{request} \rightarrow \forall\Box(\texttt{service} \rightarrow \forall\Diamond\texttt{request}))$. This formula says that if a request is responded to by a service, then a request will follow the service. The negation of the specification is a $\mathrm{TCTL}^{\exists}$ formula $\exists\Diamond(\texttt{request} \wedge \exists\Diamond(\texttt{service} \wedge \exists\Box\neg\texttt{request}))$. The following lemma shows that overapproximation techniques with $\mathrm{TCTL}^{\exists}$ formulas always yield overapproximation.

Lemma 11. *Given a* $\mathrm{TCTL}^{\exists}$ *formula* ϕ*, if we evaluate each modal-subformula in* ϕ *with overapproximation, then we still get an overapproximation of the state set satisfying* ϕ*.*

Proof. This can be done by an inductive analysis on the structure of ϕ. If ϕ is a literal expression of the forms p or $\neg p$, then the evaluation does not involve any approximation. If ϕ is like $\phi_1 \vee \phi_2$ or $\phi_1 \wedge \phi_2$, then the evaluation of ϕ still yields overapproximation with the inductive hypothesis that the evaluations of ϕ_1 and ϕ_2 are both overapproximations. If ϕ is like $\exists\phi_1\mathcal{U}\phi_2$, then, since the modal-formula is to be evaluated with overapproximation, with the inductive hypothesis, we know that ϕ is evaluated with overapproximation. The case for $\exists\Box\phi_1$ is similar. Thus, this lemma is proven. □

While restricting our specifications to TCTL$^\forall$, we can extend $\texttt{rch-bck}()$ with overapproximations as follows:

$$\texttt{rch-bck}^O(\eta_1,\eta_2) \equiv$$
$$\texttt{lfp}Y.\texttt{abs}(\eta_2 \vee (\texttt{time\_bck}(\eta_1, Y \vee (\eta_1 \wedge \bigvee_{e\in T} \texttt{xtion\_bck}(Y,e))))).$$

Here, $\texttt{abs}()$ means a generic overapproximation procedure. Procedure $\texttt{rch-bck}^O()$ can be used in place of $\texttt{rch-bck}()$ in procedures $\texttt{gfp}()$, $\texttt{Eval-EDGF}()$, and $\texttt{gfp\_EDGF}()$. In our tool **RED** 4.1, we have implemented a series of game-based abstraction procedures suitable for BDD-like data structures and concurrent systems [34]. We use the term "*game*" because we envision the concurrent system operation as a game. Those processes, which we want to verify, are treated as *players*, while the other processes are treated as *opponents*. More precisely, a process is a *player* iff its local variables appear in the TCTL$^\forall$ specification. In the game, the players try to win (maintain the specification property) with the worst (i.e., minimal) assumption on their opponents. According to the well-observed discipline of modular programming [25], the behavioral correctness of a functional module should be based on minimal assumption on the environment. These *game-based abstraction* procedures omit opponents' state-information to make abstractions.

- *Game-abstraction*: The game abstraction procedure will eliminate the state information of the opponents from its argument state-predicate.
- *Game-discrete-abstraction*: This abstraction procedure will eliminate all clock constraints for the opponents in the argument state-predicate.
- *Game-magnitude-abstraction*: A clock constraint like $x - x' \sim c$ is called a *magnitude constraint* iff either x or x' is zero itself (i.e., the constraint is either $x \sim c$ or $-x' \sim c$). This abstraction procedure will erase all nonmagnitude constraints of the opponents in the argument state-predicate.

More details of the abstraction techniques can be found in [34].

8 Implementation and Experiments

We have implemented the ideas in our model-checker/simulator, **RED** version 4.1, for timed automata. **RED** uses the new BDD-like data structure *CRD (Clock-Restriction Diagram)* [28], [29], [30] and supports both forward and backward analysis, full TCTL model-checking with non-Zeno computations, deadlock detection, and counterexample generation. Users can also declare global and local (to each process) variables of type clock, integer, and pointer (to identifier of processes). Boolean conditions on variables can be tested and variable values can be assigned. The TCTL formulas in **RED** also allow quantification on process identifiers for succinct specification. Interested readers can download **RED** for free from http://cc.ee.ntu.edu.tw/~val/. We designed our experiments in two ways. First, we ran **RED** 4.1 with various options and benchmarks to test if our ideas could indeed improve the verification performance of inevitability properties. Second, we compare **RED** 4.1 with Kronos 5.2 to check whether our implementation is competitive.

In the following sections, we shall first discuss the design of our benchmarks and then report our experiments. Data is collected on a Pentium 4 1.7GHz with 256MB memory running LINUX. Execution times are collected for Kronos, while times and memory (for data-structure) are collected for **RED**. "s" means seconds of CPU time, "k" means kilobytes for memory space for data-structures, and "O/M" means "out-of-memory."

8.1 Benchmarks

We used the following benchmarks.

- *Bounded termination detection (TD)* [30]: We have a network of communicating processes. One of the processes will finish execution with a deadline. The other processes will periodically check if their neighbors have finished. They will finish execution if one of their neighbors has finished. Three network configurations, linear lists, binary trees, and lattices, are used in the experiment. The unbounded inevitability property we want to check is that "Inevitably, all processes will finish," i.e., $\forall\Diamond\forall i, \texttt{finished}_i$. The biggest timing constant used is 10.
- *PATHOS real-time operating system scheduling* [4]: In the system, each process runs with a distinct priority in a period equal to the number of processes. The unbounded inevitability property we want to evaluate is that "if the process with lowest priority is in the pending state, then, inevitably, it will enter the running state thereafter." For a system with three processes, this property is $\forall\Box(\texttt{pending}_3 \rightarrow \forall\Diamond\texttt{running}_3)$. The nesting depth of the modal-operators is one.
- *Leader election* [30]: Each process has a local pointer `parent` and a local clock. All processes initially come with `parent` = NULL. Then, a process with its `parent` = NULL may broadcast its request to be adopted by a parent. Another process with its `parent` = NULL may respond. The process with the smaller identifier will become the parent of the other process in the requester-responder pair. The biggest timing constant used is 2. The unbounded inevitability we want to verify is that, eventually, the algorithm will finish with a unique leader elected, i.e., $\forall\Diamond(\texttt{parent}_1 = \texttt{NULL} \wedge \forall i : i \neq 1, (\texttt{parent}_i \neq \texttt{NULL} \wedge \texttt{parent}_i < i))$. There are no nested modal-operators. To guarantee the inevitability, we assume that a process with `parent` = NULL will finish an iteration in two time units.
- *CSMA/CD protocol* [28], [29], [36]: Basically, this is the ethernet bus arbitration protocol with collision-and-retry. The timing constants used are 26, 52, and 808. The following three inevitability specifications have been checked.

(A) When two processes are simultaneously in the transmission mode, then, in 26 time units, the bus will inevitably go back to the idle state. That is,

$$\forall\Box((\texttt{transm}_1 \wedge \texttt{transm}_2) \rightarrow x.\forall\Diamond(x < 26 \wedge \texttt{bus\_idle})).$$

This experiment allows us to observe how our techniques perform with bounded inevitability.

TABLE 1
Performance of the Algorithm Using Non-Zeno Requirement and EDGF

	concurrency	no non-Zeno requirement		non-Zeno requirement	
		EDGF	no EDGF	EDGF	no EDGF
		time/space/answer	time/space/answer	time/space/answer	time/space/answer
TD (Linear)	2 proc.s	0.2s/4k/true	0.2s/4k/true	0.1s/8k/true	0.2s/8k/true
	3 proc.s	0.25s/17k/true	0.28s/17k/true	0.20s/34k/true	0.22s/34k/true
	4 proc.s	5.42s/177k/true	5.41s/177k/true	3.25s/170k/true	3.26s/170k/true
	5 proc.s	2030.53s/62720k/true	2069.80s/62720k/true	59.16s/3569k/true	59.0s/3569k/true
TD (Lattice)	2 proc.s	0.1s/4k/true	0.2s/4k/true	0.2s/8k/true	0.2s/8k/true
	3 proc.s	0.14s/14k/true	0.16s/14k/true	0.12s/31k/true	0.12s/31k/true
	4 proc.s	5.71s/345k/true	5.73s/345k/true	3.24s/296k/true	3.27s/296k/true
	5 proc.s	1048.26s/57301k/true	1050.77s/57301k/true	43.55s/5964k/true	43.19s/5964k/true
TD (Tree)	2 proc.s	0.1s/4k/true	0.2s/4k/true	0.2s/8k/true	0.3s/8k/true
	3 proc.s	0.16s/14k/true	0.16s/14k/true	0.11s/31k/true	0.12s/31k/true
	4 proc.s	3.56s/191k/true	3.58s/191k/true	2.25s/271k/true	2.25s/271k/true
	5 proc.s	1287.7s/75135k/true	1286.93s/75135k/true	30.76s/4380k/true	30.61s/4380k/true
pathos	2 proc.s	0.02s/7k/true	0.02s/7k/true	0.03s/7k/true	0.03s/7k/true
	3 proc.s	0.09s/18k/true	0.1s/18k/true	0.08s/17k/true	0.09s/17k/true
	4 proc.s	0.63s/74k/true	0.66s/74k/true	0.31s/42k/true	0.31s/42k/true
	5 proc.s	6.52s/857k/true	6.65s/859k/true	1.17s/114k/true	1.28s/114k/true
	6 proc.s	161s/15087k/true	162s/15090k/true	5.22s/314k/true	5.37s/314k/true
	7 proc.s	O/M	O/M	30.71s/942k/true	31.16s/941k/true
leader election	2 proc.s	0.04s/10k/true	0.03s/10k/true	0.04s/16k/true	0.04s/16k/true
	3 proc.s	0.28s/33k/true	0.28s/33k/true	0.25s/84k/true	0.24s/84k/true
	4 proc.s	1.96s/84k/true	1.98s/84k/true	1.54s/338k/true	1.53s/338k/true
	5 proc.s	10.01s/23/true	10.07s/234k/true	11.23s/1164k/true	11.17s/1164k/true
	6 proc.s	52.63s/635k/true	48.28s/635k/true	110.9s/7992k/true	110.2s/7992k/true
	7 proc.s	206.7s/1693k/true	205.7s/1693k/true	860.5s/42062k/true	859.7s/42062k/true
CSMA/CD (A)	bus+2 senders	0.07s/25k/true	0.15s/25k/true	0.33s/42k/true	9.29s/90k/true
	bus+3 senders	0.24s/49k/true	0.66s/63k/true	3.09s/191k/true	98.33s/191k/true
	bus+4 senders	0.78s/131k/true	2.38s/201k/true	26.23s/936k/true	867.5s/1578k/true
	bus+5 senders	2.39s/378k/true	8.47s/625k/true	195.14s/4501k/true	6021s/7036k
CSMA/CD (B)	bus+2 senders	0.16s/25k/maybe	0.16s/25k/maybe	1.92s/37k/true	2.3s/37k/true
	bus+3 senders	1.52s/62k/maybe	1.52s/62k/maybe	28.67s/151k/true	34.88s/151k/true
	bus+4 senders	10.94s/239k/maybe	11.58s/239k/maybe	235.48s/765k/true	283s/766k/true
CSMA/CD (C)	bus+2 senders	0.05s/25k/true	0.06s/25k/true	0.06s/25k/true	0.72s/36k/true
	bus+3 senders	0.14s/49k/true	0.21s/49k/true	0.29s/79k/true	5.51s/183k/true
	bus+4 senders	0.43s/97k/true	0.67s/97k/true	1.36s/298k/true	30.99s/752k/true
	bus+5 senders	1.32s/286k/true	2.44s/285k/true	6.73s/1045k/true	173.82s/2724k/true
	bus+6 senders	4.57s/833k/true	8.68s/835k/true	33.53s/3436k/true	907.41s/9031k/true
	bus+7 senders	16.32s/2364k/true	32.84s/2367k/true	166.14s/10652k/true	4558s/27993k/true

(B) If sender 1 is in its `transmission` mode for no less than 52 time units, then it will inevitably enter the `wait` mode, i.e.,

$$\forall\Box((\mathtt{transm}_1 \wedge x_1 \geq 52) \rightarrow \forall\Diamond\mathtt{wait}_1).$$

Especially, this specification can be verified by quantifying only on non-Zeno computations.

(C) If the bus is in the `idle` mode and later enters the `collision` mode, then it will inevitably go back to the `idle` mode, i.e.,

$$\forall\Box(\mathtt{bus\_idle} \\ \rightarrow \forall\Box(\mathtt{bus\_collision} \rightarrow \forall\Diamond\mathtt{bus\_idle})).$$

This property is special in that the nesting depth of the modal-operator is two and can give us some insight into how our abstraction techniques scale to the inductive structure of specifications.

The nesting depth of the modal operators is zero for termination detection and leader-election, one for PATHOS, CSMA/CD (A), and CSMA/CD (B), and two for CSMA/CD (C). The specifications for the benchmarks all fall in $\mathrm{TCTL}^{\forall}$.

8.2 Performance of the Algorithm Using Non-Zeno Requirement and EDGF

In this experiment, we observed the performance of our algorithm with the non-Zeno requirement and the EDGF technique. Performance data is shown in Table 1. The data shows that the EDGF could be useful in practice. When the technique fails, it only incurs a small overhead. When it succeeds, it significantly improves performance two to three fold. Thus, we suggests that EDGF should always be used in inevitability evaluations.

As for the non-Zeno requirement, we find that, with or without the EDGF, a non-Zeno requirement does add more complexity to the evaluation of the inevitability properties against many benchmarks. For the three specifications of the CSMA/CD model, exponential blow-ups were observed.

On the contrary, for the PATHOS benchmark, the sessions with the non-Zeno requirement incur much less complexity than without it. We have looked into the execution of our algorithms for the explanation of these complexity patterns. Procedures `gfp()` and `gfp_EDGF()` both are constructed with an inner loop for the least fixpoint evaluation of `rch-bck()` and an outer loop for the greatest fixpoint evaluation. After we had carefully traced the execution of our model-checker, we found that this benchmark incurs very few iterations of outer loop with non-Zeno requirement, although each iteration can be costly to run. On the other hand, it incurs a significant number of iterations of inner loop without non-Zeno requirement, although each iteration is not so costly. The cumulative effect of the loop iterations results in a performance that contradicts our expectation. This benchmark shows that the

TABLE 2
Performance of the Algorithm Using Different d-Values
(TD, Leader, and Pathos Benchmarks)

	$\sim d$	2 procs	3 procs	4 procs	5 procs
		time/space	time/space	time/space	time/space
TD (Linear)	>=1	0.11s/8k	2.1s/52k	43.30s/1190k	O/M
	>1	0.10s/11k	1.61s/38k	28.49s/402k	496.8s/7996k
	>2	0.6s/10k	0.96s/48k	21.51s/686k	646.78s/29010k
	>3	0.3s/10k	0.58s/41k	12.22s/795k	297.62s/20445k
	>4	0.3s/10k	0.48s/44k	8.84s/466k	240.29s/19799k
	>5	0.4s/10k	0.37s/41k	6.48s/370k	169.89s/16124k
	>6	0.2s/10k	0.33s/40k	6.8s/514k	204.7s/26179k
	>7	0.2s/10k	0.29s/41k	5.35s/438k	130.58s/17608k
	>8	0.3s/10k	0.25s/41k	4.69s/424k	124.86s/18730k
	>9	0.3s/10k	0.27s/42k	4.83s/423k	100.74s/11315k
	>10	0.1s/8k	0.20s/34k	3.25s/170k	60.41s/3569k
TD (Latice)	>=1	0.12s/8k	1.40s/53k	49.54s/1701k	O/M
	>1	0.9s/11k	1.1s/47k	31.13s/689k	791.95s/46037k
	>2	0.6s/10k	0.58s/53k	24.27s/1553k	O/M
	>3	0.5s/10k	0.39s/46k	12.39s/1012k	O/M
	>4	0.3s/10k	0.31s/51k	10.55s/1648k	O/M
	>5	0.3s/10k	0.21s/51k	7.34s/701k	233.59s/46922k
	>6	0.3s/10k	0.25s/51k	6.32s/480k	138.11s/22447k
	>7	0.3s/10k	0.18s/51k	5.12s/326k	88.84s/11235k
	>8	0.1s/10k	0.18s/51k	4.75s/394k	93.31s/14303k
	>9	0.2s/10k	0.18s/46k	5.10s/406k	101.15s/15397k
	>10	0.2s/8k	0.12s/31k	3.24s/296k	44.11s/5964k
TD (Tree)	>=1	0.12s/8k	1.42s/53k	37.64s/1110k	O/M
	>1	0.11s/11k	1.0s/47k	21.96s/745k	486.77s/32802k
	>2	0.5s/10k	0.58s/53k	19.14s/1371k	863.68s/67080k
	>3	0.3s/10k	0.38s/46k	9.60s/982k	565.73s/63593k
	>4	0.3s/10k	0.30s/51k	7.84s/981k	291.60s/41810k
	>5	0.3s/10k	0.21s/51k	5.17s/460k	136.59s/17959k
	>6	0.2s/10k	0.23s/51k	4.48s/358k	88.60s/11206k
	>7	0.2s/10k	0.18s/51k	3.90s/335k	69.76s/9187k
	>8	0.4s/10k	0.18s/51k	3.63s/335k	68.66s/11205k
	>9	0.1s/10k	0.18s/46k	3.77s/436k	81.46s/12720k
	>10	0.2s/8k	0.11s/31k	2.25s/271k	30.81s/4380k
leader	>=1	0.05s/16k	0.53s/88k	3.41s/451k	30.52s/1554k
	>1	0.04s/16k	0.42s/89k	3.55s/482k	28.15s/1612k
	>2	0.03s/16k	0.24s/84k	1.58s/338k	11.625s/1164k
pathos	>=1	0.04s/7k	0.16s/26k	0.65s/76k	3.2s/245k
	>1	0.02s/7k	0.11s/21k	0.55s/76k	3.16s/244k
	>2	0.03s/7k	0.08s/17k	0.37s/51k	1.83s/221k
	>3		0.09s/17k	0.31s/41k	1.41s/119k
	>4			0.32s/42k	1.17s/109k
	>5	not available			1.19s/114k

efficiency of inevitability evaluations depends on many factors.

Finally, benchmark CSMA/CD(B) shows that some inevitability properties can only be verified with the non-Zeno requirement.

8.3 Performance of the Algorithm Using the Different d-Values for Measuring Time-Progress

In (1) of procedure `gfp()`, we use inequality `ZC` $\sim d$ to check time-progress in non-Zeno computations where "$\sim$ " is either $>$ or $\geq$ and d is a parameter ≥ 1. In our experiments, various choices of "$\sim d$" were used, ranging from 1 to beyond the biggest timing constants used in the system models. We found that the choices of "$\sim d$" may greatly affect the verification performance. It is our purpose to come up with suggestions to make good choices through carefully observing the performance patterns. Experiment results are shown in Tables 2 and 3. We have also drawn charts to show time-complexity and memory-complexity with regard to various choices in Fig. 2 and Fig. 3, respectively.

As can be seen from the performance curves, our algorithms may respond to various model structures and specifications with different complexity performance patterns. For benchmarks termination-detection, leader-election, and PATHOS, there is a vague pattern that the bigger the d-value is, the better the performance that follows. For

TABLE 3
Performance of the Algorithm Using Different d-Values
(CSMA/CD Benchmarks)

	$\sim d$	2 procs	3 procs	4 procs
		time/space	time/space	time/space
CSMA/CD (A)	>=1	12.18s/270k	144.14s/1480k	1031.14s/5587k
	>26	0.25s/42k	2.35s/191k	18.48s/731k
	>=39	0.28s/42k	2.53s/191k	20.72s/936k
	>51	0.26s/42k	2.54s/191k	20.77s/936k
	>=64	0.36s/42k	4.30s/259k	39.93s/1230k
	>76	0.36s/42k	4.31s/259k	39.37s/1230k
	>=89	0.36s/42k	5.35s/308k	47.68s/1483k
	>101	0.37s/42k	5.33s/308k	47.13s/1483k
	>127	0.40s/42k	6.78s/392k	78.76s/1947k
	>151	0.40s/42k	7.95s/412k	91.94s/2091k
	>202	0.44s/42k	10.77s/537k	144.99s/2678k
	>303	0.52s/43k	17.20s/735k	285.99s/3872k
	>404	0.60s/44k	24.75s/877k	476.62s/5336k
	>606	0.73s/44k	43.51s/1156k	1035.36s/8103k
	>808	0.91s/45k	67.44s/1418k	1843.21s/10936k
CSMA/CD (B)	>=1	184.57s/214k	1663.24s/667k	10181.75s/1951k
	>26	3.22s/48k	34.15s/226k	251.85s/802k
	>=39	2.33s/59k	24.41s/238k	180.11s/837k
	>51	1.74s/58k	18.29s/238k	134.78s/837k
	>=64	1.75s/37k	22.46s/130k	186.3s/631k
	>76	1.47s/37k	18.71s/129k	155.41s/631k
	>=89	1.35s/37k	21.8s/151k	166.1s/765k
	>101	1.21s/37k	18.70s/151k	146.57s/765k
	>127	1.4s/37k	16.95s/210k	174.99s/897k
	>151	1.12s/37k	20.92s/226k	199.84s/1078k
	>202	0.85s/37k	17.84s/284k	202.96s/1348k
	>303	0.84s/37k	21.33s/397k	301.9s/1966k
	>404	0.64s/37k	19.79s/450k	326.95s/2457k
	>606	1.9s/59k	34.52s/649k	730.54s/4025k
	>808	0.55s/35k	25.6s/719k	649.21s/5198k
CSMA/CD (C)	>=1	2.59s/178k	27.96s/1001k	174.2s/3892k
	>26	0.5s/25k	0.25s/79k	1.9s/298k
	>=39	0.6s/25k	0.25s/79k	1.12s/298k
	>51	0.6s/25k	0.25s/79k	1.13s/298k
	>=64	0.6s/25k	0.32s/80k	1.71s/299k
	>76	0.6s/25k	0.32s/80k	1.72s/299k
	>=89	0.6s/25k	0.32s/80k	1.72s/299k
	>101	0.6s/25k	0.32s/80k	1.73s/299k
	>127	0.6s/25k	0.40s/80k	2.33s/299k
	>151	0.6s/25k	0.41s/80k	2.34s/299k
	>202	0.7s/25k	0.49s/80k	2.94s/299k
	>303	0.9s/25k	0.63s/80k	4.14s/299k
	>404	0.11s/25k	0.81s/80k	5.33s/299k
	>606	0.13s/25k	1.13s/80k	7.70s/299k
	>808	0.16s/25k	1.45s/80k	10.8s/300k

the three CSMA/CD benchmarks, the best performance happens when we choose d around 25 to 50.

Again, we looked into the execution of our algorithms for the explanation of these complexity patterns. We found that the outer loop converges faster with bigger d-values, while the inner loop converges slower. With bigger d-values, we may need fewer iterations of the outer-loop and, at the same time, more iterations of the inner loop to compute greatest fixpoints. The complexity shapes in the figures are thus superposition of the complexities of the outer loop and the inner loop.

It is hard to establish a general relation between the choices of "$\sim d$" and performances. After all, the relation may depend on the systems themselves. However, we were still capable of finding the following observations true for all experiments in this work.

- *Observation 1*: The choice of d-values may greatly affect the inevitability analysis performances.
- *Observation 2*: Usually, "$>= 1$" (i.e., "≥ 1") gives the worst performance and should be avoided.

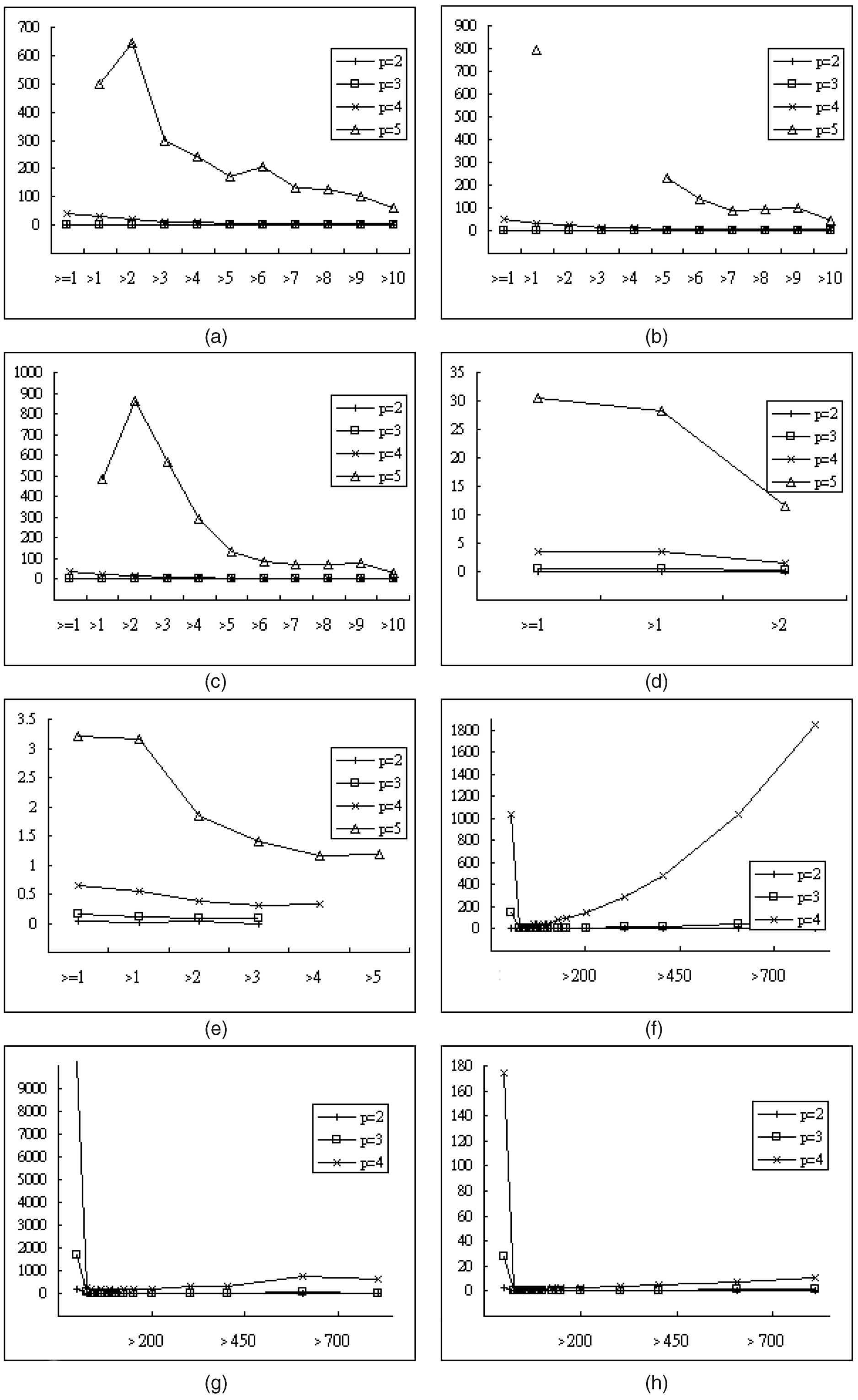

Fig. 2. Time-complexity charts with regard to choices of "$\sim d$" (Data collected with option EDGF). The Y-axis is with "time in second," while the X-axis is with "$\sim d$" used in "ZC $\sim d$." (a) TD(Linear). (b) TD(Latice). (c) TD(Tree). (d) Leader-election. (e) PATHOS. (f) CSMA/CD(A). (g) CSMA/CD(B). (h) CSMA/CD(C).

- *Observation 3*: For a parameterized system, curves for different various choices of "$\sim d$" are of a similar shape for different sizes of concurrency.

Observation 1 implies that the choice of "$\sim d$" can significantly reduce both time and memory complexities. To clarify the influences, we define the B/W-ratio (the Best complexity over the Worst complexity) among the different choices. In case the worst case runs out of memory, we denote the B/W-ratio as a big "O." In Table 4, we show these B/W-ratios of all benchmarks. The smaller the B/W-ratio is, the more we

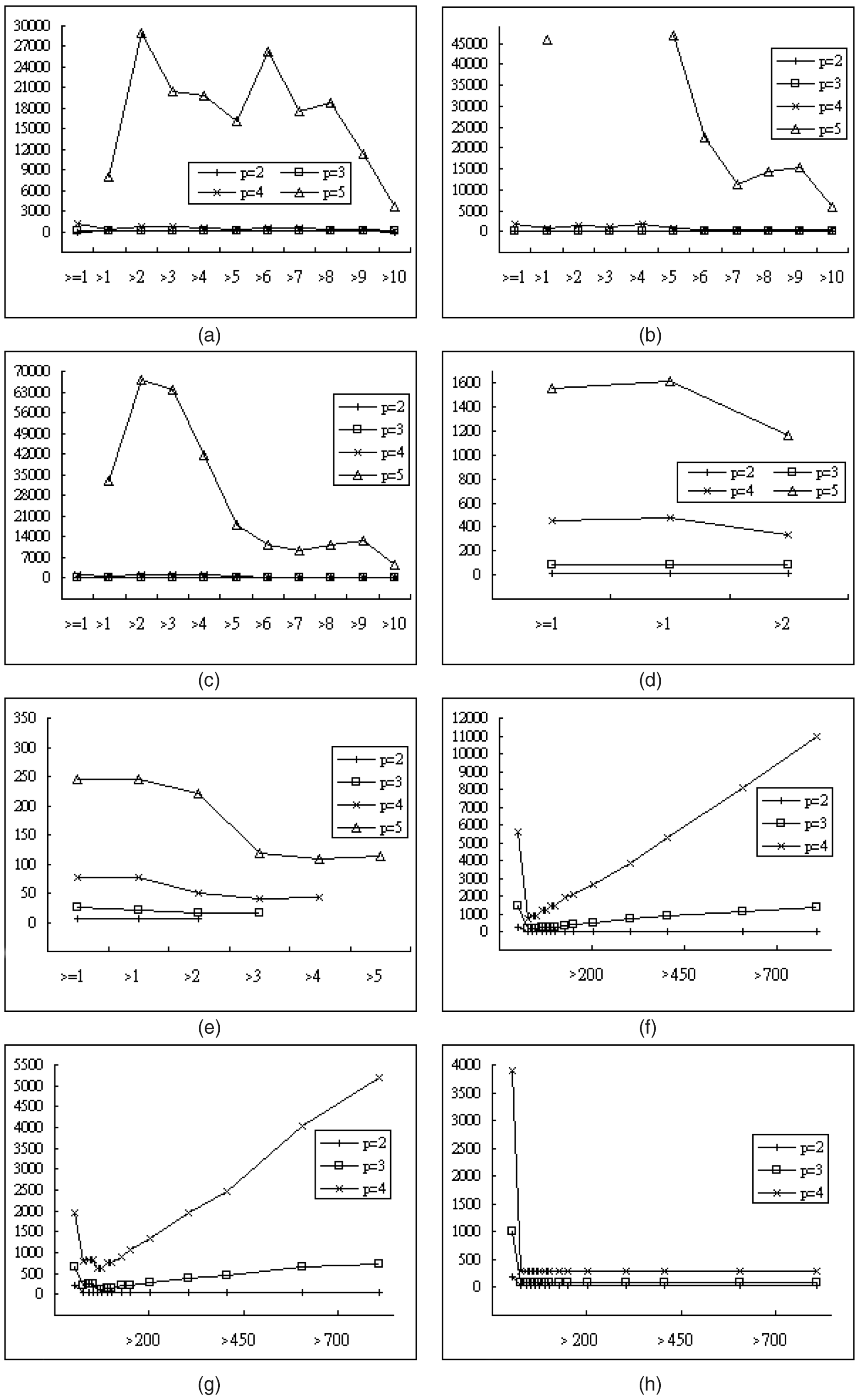

Fig. 3. Memory-complexity charts with regard to choices of "$\sim d$" (data collected with option EDGF). The Y-axis is with "memory space in kb," while the X-axis is with "$\sim d$" used in "$\mathtt{ZC} \sim d$." (a) TD(Linear). (b) TD(Latice). (c) TD(Tree). (d) Leader-election. (e) PATHOS. (f) CSMA/CD(A). (g) CSMA/CD(B). (h) CSMA/CD(C).

can reduce cost by picking up good choices. For example, for specification CSMA/CD(B) with two processes, the B/W-ratio for time complexity is 0.3 percent and implies a performance enhancement of over 300 times just by changing from a bad choice to a good one.

Observation 2 implies that the choice of "≥ 1" may lead to bad, if not the worst, performance. Interestingly, before we had discovered this fact, our tool, **RED**, used this bad choice by default. By changing this default setting, the performance of our tool has been greatly enhanced in the evaluation of many inevitability specifications.

TABLE 4
The B/W Ratios

	resource	2 procs	3 procs	4 procs	5 procs
TD (Linear)	time	16.7%	9.5%	7.5%	O
	space	72.7%	65.4%	14.3%	O
TD (Latice)	time	11.1%	8.6%	6.5%	O
	space	72.7%	58.5%	18.0%	O
TD (Tree)	time	20.0%	7.7%	6.0%	O
	space	72.7%	58.5%	20.6%	O
leader	time	60.0%	45.3%	46.3%	37.7%
	space	100%	94.4%	70.1%	72.2%
pathos	time	50.0%	50.0%	47.7%	37.2%
	space	100%	65.4%	53.9%	44.5%
CSMA/CD (A)	time	2.1%	1.6%	1.8%	
	space	16.7%	12.9%	13.1%	
CSMA/CD (B)	time	0.3%	1.1%	1.3%	
	space	16.4%	17.9%	15.4%	
CSMA/CD (C)	time	19.3%	0.9%	0.6%	
	space	14.0%	7.9%	7.7%	

Observation 3 implies that we can fairly accurately predict good choices of "$\sim d$" for high concurrency parameterized systems by calculating the good ones for low concurrency parameterized systems. In the termination detection benchmarks, choosing $ZC > 10$ leads all experiments to the best performances. In the leader election benchmarks, the best choice is $ZC > 2$, independent of the process numbers. The observation also holds in the CAMA/CD benchmark, where the best choice of the system involving three processes is exactly the best choice of four. This implication could be valuable since the verification complexity is usually at least exponential in the concurrency sizes of the target systems.

From now on, we shall use the best choice of "$\sim d$" reported in this subsection for the experiments yet to be reported.

8.4 Performance of the Algorithm Using Abstraction Techniques

In Table 5, we report the performance data of our **RED** 4.1 with respect to our three abstraction techniques. In general, the abstraction techniques enhance the verification performance. Notably, the game-discrete and game-magnitude abstractions seem to have enough precision to discern true properties.

It is somewhat surprising that the game-magnitude abstraction incurs excessive complexity for the PATHOS benchmark. After carefully examining the traces generated by **RED**, we found that, because nonmagnitude constraints were eliminated, some of the inconsistent convex state spaces in the representation became consistent. These spurious convex state spaces represented many more paths in our CRD and greatly burdened our greatest fixpoint calculation. For instance, the outer loop of procedure `gfp()` takes two iterations to reach the fixpoint with the game-magnitude abstraction. It only takes one iteration to do so without the abstraction. In our previous experience, this abstraction technique has worked efficiently with reachability analysis. It seems that the performance of abstraction techniques for greatest fixpoint evaluation can be subtle.

Another interesting issue is whether we can apply these techniques to predict good choices of "$\sim d$." Since the abstract techniques have significantly influenced the performance for the CSMA/CD benchmarks, we report the experiments in Table 6 with regard to d-values and our three abstraction techniques. All benchmarks using "bus + 4senders" run over with non-Zeno requirement and EDGF on. For both specifications CSMA/CD(A) and CSMA/CD(C), the abstraction techniques significantly reduced the complexity. Moreover, the performance curves for abstract evaluation have shapes similar to those for exact evaluation. The experiment data

TABLE 5
Performance of the Algorithm Using Abstraction Techniques

	concurrency	no abstraction	Game	Game-discrete	Game-mag.
		time/space/answer	time/space/answer	time/space/answer	time/space/answer
pathos	2 procs.s	0.03s/7k/true	0.01s/7k/true	0.03s/7k/true	0.03s/7k/true
	3 procs.s	0.08s/17k/true	0.11s/17k/maybe	0.09s/17k/true	0.1s/22k/true
	4 procs.s	0.31s/42k/true	0.36s/36k/maybe	0.37s/36k/true	0.78s/100k/true
	5 procs.s	1.17s/114k/true	1.16s/71k/maybe	1.2s/71k/true	8.55s/674k/true
	6 procs.s	5.22s/314k/true	2.83s/114k/maybe	3.39s/114k/true	191.1s/6074k/true
	7 procs.s	30.71s/942k/true	6.66s/175k/maybe	8.62s/175k/true	6890s/62321k/true
leader election	2 procs.s	0.04s/16k/true	0.03s/16k/true	0.03s/16k/true	0.02s/16k/true
	3 procs.s	0.25s/84k/true	0.25s/84k/true	0.23s/84k/true	0.25s/84k/true
	4 procs.s	1.54s/338k/true	1.53s/338k/true	1.54s/338k/true	1.52s/338k/true
	5 procs.s	11.23s/1164k/true	11.71s/1164k/true	11.38s/1164k/true	11.39s/1164k/true
	6 procs.s	110.9s/7992k/true	111.2s/7993k/true	110.8s/7993k/true	110.2s/7993k/true
	7 procs.s	860.6s/42062k/true	854.8s/42123k/true	861.5s/42123k/true	867.7s/42123k/true
CSMA/CD (A)	bus+2 senders	0.33s/42k/true	0.33s/42k/true	0.29s/42k/true	0.33s/42k/true
	bus+3 senders	3.09s/191k/true	3.35s/191k/maybe	1.33s/191k/true	3.35s/191k/maybe
	bus+4 senders	26.23s/936k/true	9.57s/731k/maybe	4.79s/731k/true	9.57s/731k/maybe
	bus+5 senders	195.14s/4501k/true	29.89s/2529k/maybe	16.96s/2529k/true	29.8s/2529k/maybe
CSMA/CD (B)	bus+2 senders	1.92s/37k/true	0.58s/25k/true	0.76s/25k/true	0.58s/25k/true
	bus+3 senders	28.67s/151k/true	2.73s/88k/true	3.91s/85k/true	2.73s/88k/true
	bus+4 senders	235.48s/765k/true	9.54s/290k/true	14.72s/281k/true	9.54s/290/true
CSMA/CD (C)	bus+2 senders	0.06s/25k/true	0.06s/25k/true	0.05s/25k/true	0.06s/25k/true
	bus+3 senders	0.29s/79k/true	0.19s/79k/true	0.18s/79k/true	0.19s/79k/true
	bus+4 senders	1.36s/298k/true	0.71s/298k/true	0.73s/298k/true	0.71s/298k/true
	bus+5 senders	6.73s/1045k/true	2.85s/1045k/true	2.90s/1045k/true	2.85s/1045k/true
	bus+6 senders	33.53s/3436k/true	11.84s/3436k/true	11.77s/3436k/true	11.84s/3436k/true
	bus+7 senders	166.14s/10652k/true	47.64s/10652k/true	47.84s/10652k/true	47.64s/10652k/true

All benchmarks run with non-Zeno requirement and EDGF on.

TABLE 6
Performance of the Algorithm Using Different d-Values and Abstraction Techniques

	$\sim d$	no abstraction	Game	Game-discrete	Game-mag.
		time/space	time/space	time/space	time/space
CSMA/CD (A)	>=1	1031.14s/5587k	2642.94s/731k	3076.51s/802k	2982.59s/1231k
	>26	18.48s/731k	7.91s/731k	3.82s/731k	3.71s/731k
	>=39	20.72s/936k	7.85s/731k	3.84s/731k	3.72s/731k
	>51	20.77s/936k	7.86s/731k	3.87s/731k	3.75s/731k
	>=64	39.93s/1230k	12.63s/742k	6.28s/742k	6.17s/742k
	>76	39.37s/1230k	12.64s/742k	6.26s/742k	6.18s/742k
	>=89	47.68s/1483k	13.71s/742k	6.35s/742k	6.23s/742k
	>101	47.13s/1483k	13.41s/742k	6.38s/742k	6.27s/742k
	>127	78.76s/1947k	14.57s/742k	9.10s/742k	9.6s/742k
	>151	91.94s/2091k	14.65s/742k	9.31s/742k	9.12s/742k
	>202	144.99s/2678k	15.67s/742k	11.96s/742k	12.50s/742k
	>303	285.99s/3872k	17.86s/742k	17.31s/742k	19.35s/742k
	>404	476.62s/5336k	20.1s/742k	18.89s/742k	28.3s/780k
	>606	1035.36s/8103k	24.18s/742k	22.47s/742k	48.65s/1151k
	>808	1843.21s/10936k	29.23s/742k	26.33s/742k	72.69s/1487k
CSMA/CD (B)	>=1	10181.75s/1951k	530.48s/288k	549.15s/289k	521.40s/321k
	>26	251.85s/802k	21.11s/288k	22.66s/279k	20.21s/267k
	>=39	180.11s/837k	14.14s/288k	16.10s/279k	14.5s/267k
	>51	134.78s/837k	10.62s/288k	11.87s/279k	10.62s/267k
	>=64	186.3s/631k	9.24s/290k	13.67s/281k	12.81s/269k
	>76	155.41s/631k	7.75s/290k	11.38s/281k	10.45s/269k
	>=89	166.1s/765k	7.3s/290k	10.91s/281k	9.85s/269k
	>101	146.57s/765k	6.35s/290k	10.3s/281k	8.78s/269k
	>127	174.99s/897k	4.73s/290k	9.32s/281k	8.67s/269k
	>151	199.84s/1078k	5.36s/290k	10.6s/281k	9.9s/269k
	>202	202.96s/1348k	3.17s/289k	8.22s/281k	7.55s/269k
	>303	301.9s/1966k	2.96s/289k	8.97s/280k	8.14s/269k
	>404	326.95s/2457k	1.66s/289k	6.59s/281k	6.91s/269k
	>606	730.54s/4025k	4.38s/289k	9.56s/281k	10.18s/269k
	>808	649.21s/5198k	0.89s/269k	3.84s/269k	6.57s/269k
CSMA/CD (C)	>=1	174.2s/3892k	9.35s/298k	9.24s/298k	9.47s/298k
	>26	1.9s/298k	0.58s/298k	0.61s/298k	0.62s/298k
	>=39	1.12s/298k	0.63s/298k	0.61s/298k	0.64s/298k
	>51	1.13s/298k	0.62s/298k	0.60s/298k	0.64s/298k
	>=64	1.71s/299k	0.60s/299k	0.76s/299k	0.78s/299k
	>76	1.72s/299k	0.58s/299k	0.76s/299k	0.77s/299k
	>=89	1.72s/299k	0.58s/299k	0.76s/299k	0.77s/299k
	>101	1.73s/299k	0.58s/299k	0.76s/299k	0.74s/299k
	>127	2.33s/299k	0.58s/299k	0.91s/299k	0.90s/299k
	>151	2.34s/299k	0.60s/299k	0.91s/299k	0.92s/299k
	>202	2.94s/299k	0.58s/299k	1.4s/299k	1.8s/299k
	>303	4.14s/299k	0.58s/299k	1.36s/299k	1.38s/299k
	>404	5.33s/299k	0.60s/299k	1.62s/299k	1.70s/299k
	>606	7.70s/299k	0.58s/299k	2.22s/299k	2.33s/299k
	>808	10.8s/300k	0.72s/300k	2.84s/300k	2.96s/300k

All benchmarks using “bus+4senders” run over with non-Zeno requirement and EDGF on.

suggested that we may be able to make a good choice by experimenting with the less expensive abstract inevitability evaluation for a given verification task. This suggestion could be valuable for guessing good choices of “$\sim d$” when the target systems are not parameterized.

8.5 Performance Comparison with Kronos

In Table 7, we report the performance of Kronos 5.2 with regard to the five benchmarks. Here, we do not adopt termination detection benchmarks since it’s hard to model complex pointer relationships using low-level data-variables provided by Kronos. For PATHOS and leader election, Kronos did not succeed in constructing the quotient automata. But, our **RED** seems to have no problem in this regard with its on-the-fly exploration of the state space. Of course, the lack of high-level data-variables in Kronos’ modeling language may also exacerbate the problem. As for benchmark CSMA/CD(A), Kronos performs very well. We believe this is because this benchmark uses a bounded inevitability specification. Such properties have already been studied in the literature on Kronos [36].

On the other hand, benchmarks CSMA/CD(B) and (C) use unbounded inevitability specifications with modal-subformula nesting depths 1 and 2, respectively. Kronos does not scale up to the complexity of concurrency for these two benchmarks. Our **RED** prevails in these two benchmarks.

9 A Case Study: L2CAP of Bluetooth

To check our techniques against industrial verification tasks, we have modeled and verified the *Logical Link Control and Adaptation Layer Protocol (L2CAP)* of Bluetooth specification [16]. The wireless communication standard of Bluetooth has been widely discussed and adopted in many appliances since it was published. L2CAP is layered over the Baseband Protocol and resides in the data link layer of Bluetooth. This protocol supports message multiplexing, packet segmentation and reassembly, and conveying quality of service information to the upper protocol layer. The protocol regulates the behavior between a master device and a slave device.

TABLE 7
Performance of Kronos in Comparison

	concurrency	no abstraction	extrapolation	inclusion	convex-hull
		time/space/answer	time/space/answer	time/space/answer	time/space/answer
pathos	2 procs	0.0s/true	0.0s/true	0.0s/true	0.0s/true
	3 procs	0.01s/true	0.01s/true	0.02s/true	0.02s/true
	4 procs	Q/N/C	Q/N/C	Q/N/C	Q/N/C
leader election	2 procs	0.0s/true	0.0s/true	0.0s/true	0.0s/true
	3 procs	0.01s/true	0.01s/true	0.01s/true	0.01s/true
	4 procs	0.05s/true	0.06s/true	0.04s/true	0.04s/true
	5 procs	Q/N/C	Q/N/C	Q/N/C	Q/N/C
CSMA/CD (A)	bus+2 senders	0.0s/true	0.01s/true	0.0s/true	0.01s/true
	bus+3 senders	0.01s/true	0.01s/true	0.01s/true	0.01s/true
	bus+4 senders	0.06s/true	0.06s/true	0.06s/true	0.06s/true
	bus+5 senders	0.31s/true	0.31s/true	0.32s/true	0.32s/true
CSMA/CD (B)	bus+2 senders	8.67s/true	8.68s/true	8.65s/true	8.71s/true
	bus+3 senders	O/M	O/M	O/M	O/M
CSMA/CD (C)	bus+2 senders	2.69s/true	2.70s/true	2.72s/true	2.69s/true
	bus+3 senders	O/M	O/M	O/M	O/M

Q/N/C means that Kronos cannot construct the quotient automata.

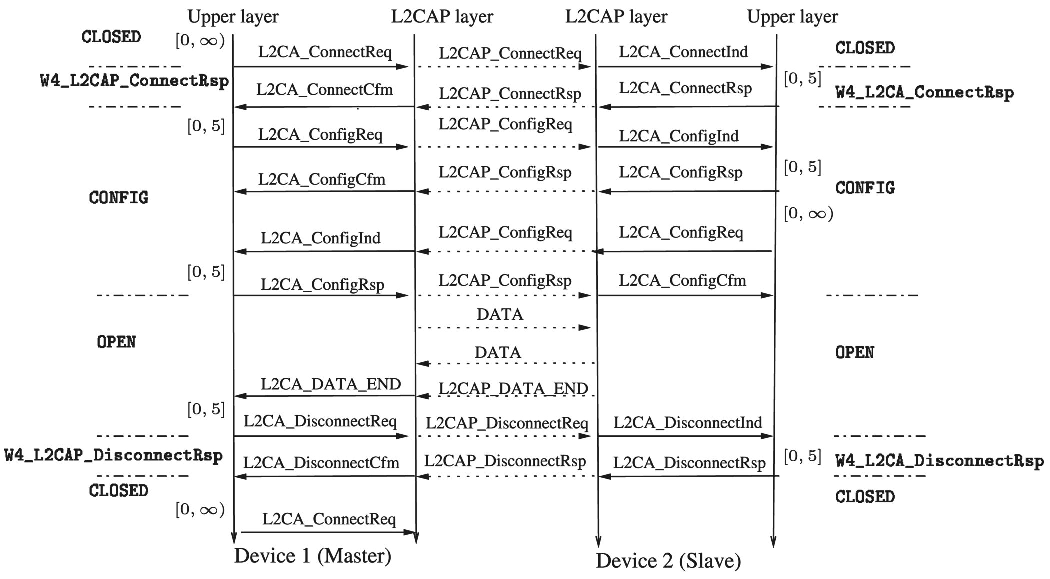

Fig. 4. A message sequence chart of L2CAP.

9.1 Modeling L2CAP

The L2CAP defines the actions performed by a master and a slave. A master is a device issuing a request, while a slave is the one responding to the master's request. A message sequence chart (MSC) that may better illustrate a typical scenario of event sequence in L2CAP can be found in Fig. 4. The two inner vertical lines represent the computation of L2CAP layers in the master and the slave devices. The scenario starts when the master's upper layer issues an `L2CA_ConnectReq` (Connection Request) through the L2CA interface. Upon receiving the request, the master communicates the request through the unreliable network to the slave (with an `L2CAP_ConnectReq`), which will then convey the request to the slave's upper layer (with an `L2CA_ConnectInd`).

The protocol goes on with messages bouncing back and forth until the master sends an `L2CAP_ConfigRsp` message to the slave. Then, both parties can start exchanging data. Finally, the master's upper layer issues message `L2CA_DisconnectReq` to close the connection and the slave confirms the disconnection.

We used nine processes to model the entire activity in L2CAP. They were the master's upper layer, master's L2CAP layer, master's L2CAP time-out process, master's L2CAP extended time-out process, slave's upper layer, slave's L2CAP layer, slave's L2CAP time-out process, slave's L2CAP extended time-out process, and the unreliable network. More details can be found in [33]. We checked the inevitability property that once the network receives the master's data and stays in the TRANSMITTED state, the slave's upper layer will eventually receive the data. This property can be written as: $\forall\Box(\mathtt{transmitted}_n \rightarrow \forall\Diamond\mathtt{received}_s)$.

9.2 Experiments with EDGF and Non-Zeno Requirement

We first check the performance of our implementation with regard to the non-Zeno requirement and the EDGF policy. The performance data is shown in Table 8. In general, we find the EDGF policy can reduce the verification cost in this case study, especially when non-Zenoness is required. Also, the inevitability property still holds without the non-Zeno

TABLE 8
L2CAP with Regard to Non-Zeno Requirements and the EDGF Technique

no non-Zeno requirement		non-Zeno requirement	
EDGF	no EDGF	EDGF	no EDGF
time/space/answer	time/space/answer	time/space/answer	time/space/answer
24.38s/378k/true	26.53s/379k/true	29.17s/210k/true	65.74s/864k/true

requirement. This may imply that for, well-designed industrial products, dropping the non-Zeno requirement could still lead to fast verification with enough precision.

9.3 Experiments with the Abstraction Techniques

In Table 9, we show how various choices of "$\sim d$" may affect the verification performance. A locally optimal inequality is "> 59." Assuming this is the best choice for performance, the B/W-ratio is then 36.33 percent for time complexity and 38.71 percent for memory complexity. We applied our abstraction techniques against different choices of "$\sim d$." Table 9 shows that the abstraction techniques indeed reduce both time and memory cost. Moreover, the good choices of "$\sim d$" for the abstract evaluations also help us accurately predict the good choices for precise evaluaton.

10 Summary of Suggestions for Efficient Configurations

To promote model-checking technology in industrial projects, we need not only elegant theory but also suggestions for the configuration of the efficient model-checking algorithms. In this work, we report extensive experiments to observe how the model-checking algorithms perform in the evaluation of inevitability properties against dense-time systems. In summary, we have the following suggestions.

- EDGF is a good speed-up technique and does not sacrifice the precision of TCTL model-checking. Thus, it should always be used in TCTL model-checking.
- For well-designed industrial designs, it is worthwhile to first evaluate inevitabilities with no non-Zeno requirement.
- For the inevitability analysis of parameterized systems, we can use the good choices of "$\sim d$" for low concurrency systems to predict the good ones for their high concurrency counterparts.
- For nonparameterized systems, we can also use abstraction techniques to help us predict the good choices of "$\sim d$" for exact evaluation.

Acknowledgments

A preliminary version of this manuscript appears in the *Proceedings of the Eighth International Conference on Implementation and Application of Automata, (CIAA '03)*. The work is partially supported by NSC, Taiwan, ROC under grants NSC 90-2213-E-002-131, NSC 90-2213-E-002-132, and by the Internet protocol verification project of the Institute of Applied Science & Engineering Research, Academia Sinica, 2001.

TABLE 9
L2CAP with Regard to the Abstraction Techniques

$\sim d$	no abstraction	Game	Game-discrete	Game-mag.
	time/space	time/space	time/space	time/space
$>=1$	80.28s/1500k	23.61s/697k	30.88s/677k	31.87s/678k
>1	54.90s/1576k	20.61s/694k	25.73s/683k	25.53s/683k
$>=2$	50.72s/1564	20.37s/689k	25.87s/675k	25.30s/676k
>2	28.43s/610k	17.37s/610k	19.87s/610k	19.54s/610k
$>=10$	29.88s/610k	17.25s/610k	19.95s/610k	19.89s/610k
$>=15$	29.49s/610k	17.33s/610k	19.75s/610k	19.78s/610k
$>=20$	30.72s/610k	17.29s/610k	19.51s/610k	20.90s/610k
$>=25$	30.28s/610k	17.37s/610k	19.61s/610k	20.28s/610k
$>=30$	29.42s/610k	17.20s/610k	19.62s/610k	19.61s/610k
$>=35$	31.22s/610k	17.27s/610k	19.42s/610k	20.93s/610k
$>=40$	30.65s/610k	17.19s/610k	19.54s/610k	20.15s/610k
$>=50$	30.6s/610k	17.19s/610k	19.44s/610k	20.48s/610k
>59	29.17s/610k	17.20s/610k	19.45s/610k	19.48s/610k
$>=60$	29.24s/610k	17.45s/610k	19.87s/610k	19.74s/610k
>60	32.16s/644k	17.78s/609k	22.0s/609k	22.42s/609k
$>=75$	32.47s/644k	17.27s/609k	22.46s/609k	23.34s/609k
$>=100$	32.81s/644k	17.32s/609k	23.2s/609k	24.58s/609k
$>=150$	32.55s/643k	17.81s/609k	24.58s/609k	24.10s/609k
$>=200$	32.39s/643k	17.32s/609k	24.64s/609k	24.78s/609k
$>=250$	33.2s/643k	17.33s/609k	24.74s/609k	26.39s/609k
>299	32.22s/643k	18.2s/609k	24.19s/609k	24.28s/609k
$>=300$	31.67s/643k	17.35s/609k	24.32s/609k	24.91s/609k
>300	31.71s/643k	17.39s/609k	24.43s/609k	24.12s/609k

References

[1] R. Alur, C. Courcoubetis, and D.L. Dill, "Model Checking for Real-Time Systems," *Proc. IEEE Fifth Symp. Logic in Computer Science*, pp. 414-425, 1990.
[2] R. Alur and D.L. Dill, "Automata for Modeling Real-Time Systems," *Proc. Int'l Colloquium Automata, Languages, and Programming*, pp. 322-335, 1990.
[3] B. Alpern and F.B. Schneider, "Defining Liveness," *Information Processing Letters*, vol. 21, no. 4, pp. 181-185, 1985.
[4] F. Balarin, "Approximate Reachability Analysis of Timed Automata," *Proc. IEEE Real-Time Systems Symp.*, pp. 52-61, 1996.
[5] G. Behrmann, P. Bouyer, K.G. Larsen, and R. Pelanek, "Lower and Upper Bounds in Zone Based Abstractions of Timed Automata," *Proc. 10th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems*, pp. 312-326, 2004.
[6] G. Behrmann, K.G. Larsen, J. Pearson, C. Weise, and W. Yi, "Efficient Timed Reachability Analysis Using Clock Difference Diagrams," *Proc. 11th Int'l Conf. Computer-Aided Verification*, pp. 341-353, 1999.
[7] S.V. Campos, "A Quantitative Approach to the Formal Verification of Real-Time Systems," PhD Thesis, Carnegie Mellon Univ., 1996.
[8] E.M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 2000.
[9] P. Cousot and R. Cousot, "Abstract Interpretation and Application to Logic Programs," *J. Logic Programming*, vol. 13, no. 2-3, pp. 103-179, 1992.
[10] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal-Logic Specifications," *ACM Trans. Programming Languages and Systems*, vol. 8, no. 2, pp. 244-263, 1986.
[11] E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-Guided Abstraction Refinement," *Proc. 12th Int'l Conf. Computer-Aided Verification*, pp. 154-169, 2000.
[12] D.L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *Proc. Int'l Workshop Automatic Verification Methods for Finite State Systems*, pp. 197-212, 1989.
[13] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The Tool KRONOS," *Proc. Third Hybrid Systems*, pp. 208-219, 1996.
[14] E.A. Emerson, "Uniform Inevitability Is Tree Automation Ineffable," *Information Processing Letters*, vol. 24, no. 2, pp. 77-79, 1987.
[15] B.C. Eaves and U.G. Rothblum, "Dines-Fourier-Motzkin Quantifier Elimination and an Application of Corresponding Transfer Principles over Ordered Fields," *Math. Programming*, vol. 53, no. 3, pp. 307-321, 1992.

[16] J. Haartsen, "Bluetooth Baseband Specification, version 1.0," http://www.bluetooth.com/, 2006.

[17] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic Model Checking for Real-Time Systems, " *Proc. IEEE Seventh Symp. Logic in Computer Science,* pp. 394-406, 1992.

[18] P.-A. Hsiung and F. Wang, "User-Friendly Verification," *Proc. IFIP Joint Int'l Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification,* pp. 279-294, 1999.

[19] F. Laroussinie and K.G. Larsen, "CMC: A Tool for Compositional Model-Checking of Real-Time Systems," *Proc. IFIP Joint Int'l Conf. Formal Description Techniques & Protocol Specification, Testing, and Verification,* pp. 439-456, 1998.

[20] J. Moller, J. Lichtenberg, H.R. Andersen, and H. Hulgaard, "Difference Decision Diagrams," *Proc. Ann. Conf. European Assoc. for Computer Science Logic,* pp. 111-125, 1999.

[21] M.O. Moller, "Parking Can Get You There Faster—Model Augmentation to Speed Up Real-Time Model Checking," *Electronic Notes in Theoretical Computer Science,* vol. 65, no. 6, 2002.

[22] A.W. Mazurkiewicz, E. Ochmanski, and W. Penczek, "Concurrent Systems and Inevitability," *Theoretical Computer Science,* vol. 64, no. 3, pp. 281-304, 1989.

[23] I. Ober, S. Graf, and I. Ober, "Validation of UML Models via a Mapping to Communicating Extended Timed Automata," *Proc. 11th Int'l SPIN Workshop Model Checking of Software,* pp. 127-145, 2004.

[24] P. Pettersson and K.G. Larsen, "UPPAAL2k," *Bull. European Assoc. Theoretical Computer Science,* vol. 70, pp. 40-44, 2000.

[25] R.S. Pressman, *Software Engineering, a Practitioner's Approach.* McGraw-Hill, 1982.

[26] F. Wang, "Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems," *Proc. Sixth Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems,* pp. 157-171, 2000.

[27] F. Wang, "Region Encoding Diagram for Fully Symbolic Verification of Real-Time Systems," *Proc. 24th IEEE Computer Software and Applications Conf.,* pp. 509-515, 2000.

[28] F. Wang, "RED: Model-Checker for Timed Automata with Clock-Restriction Diagram," *Proc. Workshop Real-Time Tools,* Technical Report 2001-014, Dept. of Information Technology, Uppsala Univ., Aug. 2001.

[29] F. Wang, "Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram," *Proc. 21st IFIP Int'l Conf. Formal Techniques for Networked and Distributed Systems,* pp. 235-250, 2001.

[30] F. Wang, "Efficient Verification of Timed Automata with BDD-Like Data-Structures," *Proc. Fourth Int'l Conf. Verification, Model Checking, and Abstract Interpretation,* pp. 189-205, 2003.

[31] F. Wang and P.-A. Hsiung, "Automatic Verification on the Large," *Proc. Third IEEE Int'l Symp. High-Assurance Systems Eng.,* pp. 134-141, 1998.

[32] F. Wang and P.-A. Hsiung, "Efficient and User-Friendly Verification," *IEEE Trans. Computers,* vol. 51, no. 1, pp. 61-83, Jan. 2002.

[33] F. Wang and F. Yu, "OVL Assertion-Checking of Embedded Software with Dense-Time Semantics," *Proc. Ninth Int'l Conf. Real-Time and Embedded Computing Systems and Applications,* pp. 254-278, 2003.

[34] F. Wang, G.-D. Hwang, and F. Yu, "Symbolic Simulation of Real-Time Concurrent Systems," *Proc. Ninth Int'l Conf. Real-Time and Embedded Computing Systems and Applications,* pp. 595-617, 2003.

[35] H. Wong-Toi, "Symbolic Approximations for Verifying Real-Time Systems," PhD thesis, Stanford Univ., 1995.

[36] S. Yovine, "Kronos: A Verification Tool for Real-Time Systems," *Int'l J. Software Tools for Technology Transfer,* vol. 1, no. 1-2, 1997.

Farn Wang received the BS degree in electrical engineering from National Taiwan University in 1982, the MS degree in computer engineering from National Chiao-Tung University in 1984, and the PhD degree in computer sciences from the University of Texas at Austin in 1993. From September 1986 to May 1987, he was a research assistant at the Telecommunication Laboratories, Ministry of Communications, Republic of China. From August 1993 to October 1997, he was an assistant research fellow at the Institute of Information Science (IIS), Academia Sinica, Taiwan, Republic of China. From October 1997 to July 2002, he was an associate research fellow at IIS. In August 2002, he became an associate professor in the Department of Electrical Engineering, National Taiwan University. In August 2005, he became a professor in the Department of Electrical Engineering, National Taiwan University. He is interested in automating human verification experiences to develop verification tools with high abstractness and efficiency. He architected and implemented several tools for the verification of timed and hybrid systems. The tools include RED, a model-checker for timed and hybrid systems, and SGM, an efficient and user-friendly verification tool for timed systems. He is a member of the IEEE Computer Society.

Geng-Dian Huang received the BS degree in information management in 2000 and the MS degree in information management in 2002 from National Taiwan University. He has been a research assistant in the Verification Automata Laboratory at the Institute of Information Science, Academia Sinica, for military service since 2003. In August 2004, he became a PhD student in the Department of Electrical Engineering, National Taiwan University. His research interest is formal verification.

Fang Yu received the bachelor's and master's degrees from the Department of Information Management at National Taiwan University in 1998 and 2000, respectively. In January 2001, he joined the Formal Verification Lab (VAL) at the Institute of Information Science, Academia Sinica, under the supervision of Dr. Farn Wang. Since August 2005, he has been at the University of California, Santa Barbara to pursue the PhD degree is computer science. His research interests include formal methods, software/hardware verification, real-time systems, and membrane computing.