Model-Checking Distributed Real-Time Systems with States, Events, and Multiple Fairness Assumptions*

Farn Wang

Dept. of Electrical Engineering, National Taiwan University
1, Sec. 4, Roosevelt Rd., Taipei, Taiwan 106, ROC
+886-2-23635251 ext. 435; FAX:+886-2-23671909
farn@cc.ee.ntu.edu.tw, http://cc.ee.ntu.edu.tw/~farn

Model-checker/simulator **Red** 5.1 will be available at http://cc.ee.ntu.edu.tw/~val

Abstract. At this moment, there lacks a specification language for distributed real-time system properties involving states and events. There also lacks a language for fairness assumptions in dense-time systems. We have defined a new temporal logic, $TECTL^f$, for the flexible specification of distributed real-time systems with constraints involving events, states, and fairness assumptions. Then we have also designed algorithms for model-checking $TECTL^f$ formulas. Finally, we have endeavored to implement and experiment the ideas in our tool, **Red** 5.1, and shown that the ideas could be used in practice.

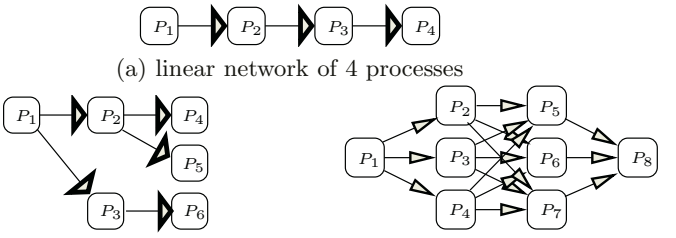
Keywords: Distributed, real-time, model-checking, verification, events, fairness.

1 Introduction

It has long been argued that neither pure state-based nor pure event-based languages quite support the natural expressiveness desirable for the specification of real-world systems [11, 17, 18]. The inadequacy of such pure languages is even more salient in the setting of distributed real-time systems, where multiple clocks do not tick at the same times. For example, to specify that an event must eventually happen using dense-time state-based language *TCTL* (a branching-time temporal logic) [1], one common style is to use an artificial state (or urgent state) that immediately follows this event. But for distributed real-time systems, this style looks cumbersome and unnatural since there is no perfect way to say how long this urgent state should last or whether we allow other transitions to happen in this state. Although there were many works in combining state-based and event-based languages [11, 16–18, 20], they were all developed for untimed

* The work is partially supported by NSC, Taiwan, ROC under grants NSC 92-2213-E-002-103, NSC 92-2213-E-002-104, and by the System Verification Technology Project of Industrial Technology Research Institute, Taiwan, ROC (2004).

C. Rattray et al. (Eds.): AMAST 2004, LNCS 3116, pp. 553–567, 2004.



(a) linear network of 4 processes

(b) binary tree network of 6 processes (c) lattice network of 8 processes

Fig. 1. Network configurations for diffusive computations

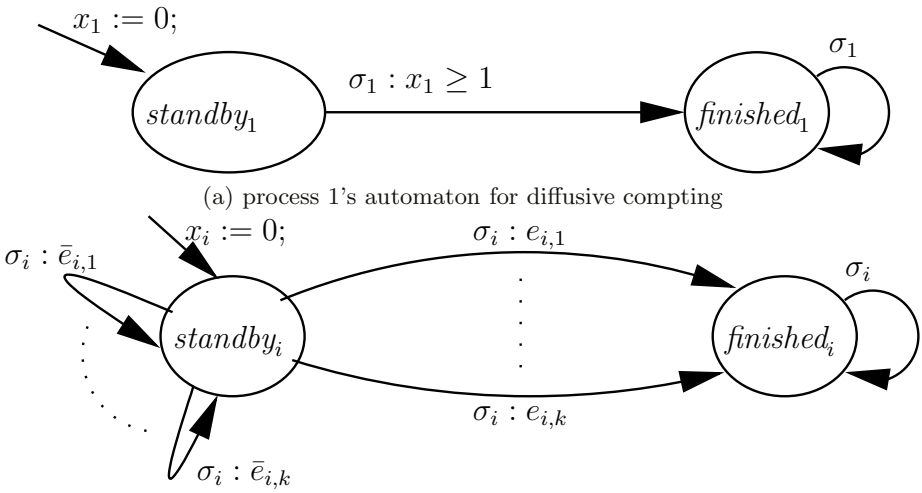
systems. Facing the undecidability of verification problems of discrete-time extensions of linear temporal logics [3, 4], we have chosen to extend TCTL[1] [1] to a specification language for dense-time properties involving both states and events.

Another challenge arises in specifying that "something good will eventually happen" in distributed real-time systems. Such properties are called "liveness" in linear-time logics and "inevitability" [27] in branching-time logics. For example, after turning on their computers, people *"naturally"* expect that operating systems will start correctly. But such a property can be impossible to verify with the models of nondeterministic automata unless we assume that each component of the system has a *fair* share of execution. The concept means that we do not have to worry about those "unnatural" behaviors as long as the systems work fine with those "natural" behaviors. For example, we may have the linear, binary tree, and lattice networks in figures 1(a), (b), and (c) for diffusive computing [10]. An arrow from a process P_i to another P_j means that the finishing of P_i may lead to the finishing of P_j. For convenience, we let $\mathtt{src}(i)$ be the index set of processes with an outgoing arrow to P_i. P_1 will nondeterministically make transition, with event σ_1, to its *finished* state. For each $i > 1$, process P_i will nondeterministically check if any process with index in $\mathtt{src}(i)$ has *finished* and make transition, with event σ_i, to its *finished* state. Note here, we add a dummy cycle transition with events $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ to their respective *finished* states. This is the traditional way to model fairness in finite-execution systems. The automata do not guarantee that all processes will enter their *finished* states. But it will be odd that if some process does not execute infinitely often. Thus with the fairness assumption that "every process executes infinitely often," we can deduce that all processes will enter their *finished* states.

Two commonly accepted concepts of fairness assumptions [13] are:

- **Correctness with strong fairness assumption:** The concept of *strong fairness* means that "something will happen infinitely often." For the networks in figure 1 with four processes, in order to verify the eventual finishing of all processes, we may want to assume that "for each i, transitions with event σ_i execute infinitely often."

[1] TCTL model checking probelm against timed automatas is PSPACE-complete.



(a) process 1's automaton for diffusive compting

Assume $\texttt{src}(i) = \{h_1, \ldots, h_k\}$.
$e_{i,j}$: when $finished_{h_j} \wedge x_i \geq 1$, goto $finished_i$.
$\bar{e}_{i,j}$: when $(\neg finished_{h_j}) \wedge x_i \geq 1$, let $x_i = 0$ and goto $standby_i$.

(b) process i's automaton for diffusive compting

Fig. 2. Diffusive propagations with fairness assumptions

- **Correctness with weak fairness assumption:** The concept of *weak fairness* means that "something will eventually stabilize to a state." In figure 2, one example is that "eventually all processes finish their tasks."

Motivated by the discussion in previous paragraphs, we have extended TCTL [1] with event constraints and fairness assumptions and defined a new specification language *TECTL*^{f} (acronym for *Timed Event CTL with Fairness assumptions*) in section 3. For example, for the systems in figure 2, the goal of diffusive computing is that all processes will eventually enter their *"finished"* states. This specification can be expressed with strong fairness assumptions as

$$\forall^{[\sigma_1 \geq 1, \sigma_2 \geq 1, \sigma_3 \geq 1, \sigma_4 \geq 1]} \Diamond \bigwedge_{0 \leq i \leq m} finished_i \qquad \text{(A)}$$

Here the strong fairness assumptions are written in square brackets as event predicates to the superscript of the universal path quantifier $\forall$. Strong fairness assumption $\sigma_i \geq 1$ means that transitions with one or more events σ_i will happen infinitely often. Such a style of fairness assumptions as event predicates allows for flexible characterizations of sets of transitions.

We shall present algorithms for model-checking *TECTL*^{f} formulas in section 4. We have used our ideas to extend the functionality of our TCTL model-checker **Red** [21–25]. Our implementation, **Red** 5.1, uses CRD (Clock-Restriction Diagram), a BDD-like structure [5, 8], with symbolic manipulation algorithms. We shall report our experiments to check the performance of our techniques in section 5. The tool and benchmarks can be downloaded for free at

`http://cc.ee.ntu.edu.tw/~val`

2 Timed Automata

We use the widely accepted model of *timed automata* [2] to describe the transitions in dense-time state-spaces. A *timed automaton* is a finite-state automaton equipped with a finite set of clocks which can hold nonnegative real-values. At any moment, the timed automaton can stay in only one *mode* (or *control location*). Each mode is labeled with an invariance condition on clocks. In its operation, one of the transitions can be triggered when the corresponding triggering condition is satisfied. Upon being triggered, the automaton instantaneously transits from one mode to another and resets some clocks to zero. In between transitions, all clocks increase their readings at a uniform rate.

For convenience, given a set P of atomic propositions and a set X of clocks, we use $B(P, X)$ as the set of all Boolean combinations of atoms of the forms p and $x \sim c$, where $p \in P$, $x \in X \cup \{0\}$, "$\sim$" is one of $\leq, <, =, >, \geq$, and c is an integer constant.

Definition 1. (timed automata) A timed automaton A is given as a tuple $\langle \Sigma, X, Q, I, \mu, E, \gamma, \lambda, \tau, \pi \rangle$ with the following restrictions. Σ is a finite set of event names. X is a finite set of clocks. Q is a finite set of modes. $I \in B(Q, X)$ is the initial condition. $\mu : Q \mapsto B(\emptyset, X)$ defines the conjunctive invariance condition of each mode. E is the finite set of transitions. $\gamma : E \mapsto (Q \times Q)$ defines the source and destination modes of each transition. $\lambda : (E \times \Sigma) \mapsto \mathcal{N}$ defines the number of instances of an event type that happen on each transition. Such a general scheme allows for the modeling of broadcasting and multicasting of many generic transmission events. $\tau : E \mapsto B(\emptyset, X)$ and $\pi : E \mapsto 2^X$ respectively defines the triggering condition and the clock set to reset of each transition. ■

Definition 2. (states) A state of $A = \langle \Sigma, X, Q, I, \mu, E, \gamma, \lambda, \tau, \pi \rangle$ is a pair like (q, ν) with $q \in Q$ and $\nu : X \mapsto \mathcal{R}^+$, the set of nonnegative real numbers. ■

We say a state (q, ν) satisfies a state predicate $\eta \in B(P, X)$, where P is either $\emptyset$ or Q, iff the following inductive conditions are satisfied.

- $(q, \nu) \models q'$ iff $q' = q$;
- $(q, \nu) \models x \sim c$ iff $\nu(x) \sim c$;
- $(q, \nu) \models \eta_1 \vee \eta_2$ iff $(q, \nu) \models \eta_1$ or $(q, \nu) \models \eta_2$;
- $(q, \nu) \models \neg\eta_1$ iff it is not the case that $(q, \nu) \models \eta_1$. ■

For any $t \in \mathcal{R}^+$, $\nu + t$ is a valuation identical to ν except that for every $x \in X$, $\nu(x) + t = (\nu + t)(x)$. Given $\bar{X} \subseteq X$, $\nu\bar{X}$ is a new valuation identical to ν except that for every $x \in \bar{X}$, $\nu\bar{X}(x) = 0$.

Definition 3. (runs) Given a timed automaton $A = \langle \Sigma, X, Q, I, \mu, E, \gamma, \lambda, \tau, \pi \rangle$, a *run* is an infinite computation of A along which time diverges. Formally speaking, a run is an infinite sequence of state-time pairs $((q_0, \nu_0), t_0)((q_1, \nu_1), t_1) \ldots ((q_k, \nu_k), t_k) \ldots\ldots$ such that

- $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically increasing divergent real-number sequence, i.e., $\forall c \in \mathcal{N}, \exists h > 1, t_h > c$, and
- **Invariance condition:** for all $k \geq 0$, for all $t \in [0, t_{k+1} - t_k]$, $(q_k, \nu_k + t) \models \mu(q_k)$; and

- **Transitions:** for all $k \geq 0$, either
  - **a null transition:** $q_k = q_{k+1}$ and $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$; or
  - **a discrete transition** e**:** denoted $q_k \xrightarrow{e} q_{k+1}$. The constraints is that there is an $e \in E$ such that $\gamma(e) = (q_k, q_{k+1})$, $(q_k, \nu_k + t_{k+1} - t_k) \models \tau(e)$, and $(\nu_k + t_{k+1} - t_k)\pi(e) = \nu_{k+1}$. ■

3 *TECTL*^{f} (Timed Event CTL with Fairness Assumptions)

3.1 Syntax

In our model of timed automata, a transition can be labeled with many instances of an event type. Thus, we devise *event predicates*, to characterize complex event constraints on transitions, with the following syntax.

$$\epsilon ::= \textstyle\sum_i a_i\sigma_i \sim c \mid \neg\epsilon_1 \mid \epsilon_1 \vee \epsilon_2$$

where σ_i is an event name and a_i's and c are integer constants. For example, for figure 1, $\sigma_i \geq 1$ represents the set of transitions with at least one σ_i event.

TECTL^f is an extension to TCTL [1] with the following syntax rules.

$$\phi ::= q \mid x \sim c \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid x.\phi_1 \mid \exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\phi_1\mathcal{U}^\epsilon\phi_2 \mid \exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^\epsilon\phi_1$$

Here $q \in Q$, $x \in X$, $c \in \mathcal{N}$. $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_n$ are either event predicates or *TECTL*^{f} formulas. ϵ is either null (not specified) or an event predicate. ϕ_1 and ϕ_2 are *TECTL*^{f} formulas. When ϵ is not specified in a path formula, then we may simply write $\phi_1\mathcal{U}\phi_2$ and $\Box\phi_1$. 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."
- $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}$ means "there exists a run satisfying *strong fairness assumptions* $\alpha_1, \ldots, \alpha_m$ and *weak fairness assumptions* $\beta_1, \ldots, \beta_n$."
- $\phi_1\mathcal{U}^\epsilon\phi_2$:
  - When ϵ is null, it works as a classical until-formula and means that along a computation, ϕ_1 is true until ϕ_2 happens.
  - When ϵ is specified, it means that along a computation, ϕ_1 is true until a transition, satisfying ϵ, happens and immediately makes ϕ_2 true.
- $\Box^\epsilon\phi_1$:
  - When ϵ is null, it works as a classical always-formula and means that along a computation, ϕ_1 is true in every state.
  - When ϵ is specified, it means that along a computation, whenever a transition satisfying ϵ happens, immediately after the transition, ϕ_1 is true.

Also we adopt the following standard shorthands :

- *true* for $0 = 0$ and *false* for $\neg$*true*;
- $\phi_1 \wedge \phi_2$ for $\neg((\neg\phi_1) \vee (\neg\phi_2))$

- $\phi_1 \rightarrow \phi_2$ for $(\neg\phi_1) \vee \phi_2$
- $\exists^{[\,]}$ and $\exists_{\langle\,\rangle}$ can both be abbreviated as $\exists$.
- $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Diamond^{\epsilon}\phi_1$ for $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle} true\,\mathcal{U}^{\epsilon}\phi_1$,
- $\forall^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^{\epsilon}\phi_1$ for $\neg\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Diamond^{\epsilon}\neg\phi_1$
- $\forall^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\phi_1\mathcal{U}^{\epsilon}\phi_2$ for $\neg((\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}(\neg\phi_2)\mathcal{U}^{\epsilon}\neg(\phi_1\vee\phi_2))\vee(\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^{\epsilon}\neg\phi_2))$
- $\forall^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Diamond^{\epsilon}\phi_1$ for $\forall^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle} true\mathcal{U}^{\epsilon}\phi_1$

Example 1. For the system in figure 2, we have specification (A) already with strong fairness assumptions in page 555. Another specification is *"If process 1 eventually stabilizes to the finished state and process 2 executes infinitely many times, then process 2 will eventually finish."* The specification with strong and weak fairness assumptions is

$$\forall^{[\sigma_2\geq 1]}_{\langle finished_1\rangle}\Diamond finished_2 \tag{B}$$

One last specification is *"Every time after process 2 executes a transition and ends in the* standby *mode, it will stay there for at least one more time units."* In *TECTL*^{f}, this specification is

$$\forall\Box^{\sigma_2\geq 1}(standby_2 \rightarrow x.\Box(x\leq 1 \rightarrow standby_2)) \tag{C}$$

■

3.2 Semantics

Given an event predicate ϵ and a transition $e \in E$, we say e satisfies ϵ, in symbols $e \models \epsilon$, according to the following inductive rules.

- $e \models \sum_i a_i\sigma_i \sim c$ iff $\sum_i a_i\lambda(e,\sigma_i) \sim c$;
- $e \models \neg\epsilon$ iff it is not the case that $e \models \epsilon$; and
- $e \models \epsilon_1 \vee \epsilon_2$ iff either $e \models \epsilon_1$ or $e \models \epsilon_2$.

For convenience of presentation, we have the following definition.

Definition 4. (runs with timed fairness assumption) Given event predicates or *TECTL*^{f} formulas $\alpha_1,\ldots,\alpha_m,\beta_1,\ldots,\beta_n$, a run

$$\rho = ((q_0,\nu_0),t_0)((q_1,\nu_1),t_1)\ldots((q_k,\nu_k),t_k)\ldots\ldots$$

satisfies *strong fairness assumptions* $\alpha_1,\ldots,\alpha_m$ and *weak fairness assumptions* $\beta_1,\ldots,\beta_n$, denoted $\rho \models \overset{\infty}{\Diamond}\{\alpha_1,\ldots,\alpha_m\}\wedge\overset{\infty}{\Box}\{\beta_1,\ldots,\beta_n\}$, *iff along the run,*

- *states or transitions satisfying* $\alpha_1,\ldots,\alpha_m$ *respectively happen infinitely often; and*
- *the system eventually stabilizes to states or transitions satisfying* $\beta_1,\ldots,\beta_n$.

Formally speaking, $\rho \models \overset{\infty}{\Diamond}\{\alpha_1,\ldots,\alpha_m\}\wedge\overset{\infty}{\Box}\{\beta_1,\ldots,\beta_n\}$, iff

- for each $1\leq i\leq m$ and $c\in\mathcal{N}$,
  - if α_i is a *TECTL*^{f} formula, there are a $k>1$ and a $t\in[0,t_{k+1}-t_k]$ such that $t_k+t>c\wedge(q_k,\nu_k+t)\models\alpha_i$;
  - if α_i is an event predicate, there are a $k>1$ and an $e\in E$ such that $t_k>c\wedge q_k\xrightarrow{e}q_{k+1}\wedge e\models\alpha_i$;

- for each $1 \leq j \leq n$, there is a $c \in \mathcal{N}$,
  - if β_j is a $TECTL^f$ formula, then for every $k > 1$ and $t \in [0, t_{k+1} - t_k]$ such that $t_k + t > c$, $(q_k, \nu_k + t) \models \beta_j$.
  - if β_j is an event predicate, then for every $k > 1$ and $e \in E$ such that $t_k > c$ and $q_k \xrightarrow{e} q_{k+1}$, $e \models \beta_j$. ■

Note that we bind the concept of fairness to the divergence of time. For example in the condition of strong fairness, we require that α_i happens at infinitely and divergently many clock readings. This is quite different from the traditional fairness concepts in untimed systems [13].

Definition 5. (Satisfaction of $TECTL^f$ formulas): We write in notations $A, (q_1, \nu_1) \models \phi$ to mean that ϕ is satisfied at state (q_1, ν_1) in timed automaton A. The satisfaction relation is defined inductively as follows.

- When $\epsilon \in B(Q, X)$, $A, (q_1, \nu_1) \models \epsilon$ iff $(q_1, \nu_1) \models \epsilon$, which was previously defined in the beginning of subsection 3.2;
- $A, (q_1, \nu_1) \models \phi_1 \vee \phi_2$ iff either $A, (q_1, \nu_1) \models \phi_1$ or $A, (q_1, \nu_1) \models \phi_2$
- $A, (q_1, \nu_1) \models \neg\phi_1$ iff $A, (q_1, \nu_1) \not\models \phi_1$
- $A, (q_1, \nu_1) \models x.\phi$ iff $A, (q_1, \nu_1\{x\}) \models \phi$, where $\nu_1\{x\}$ is a valuation identical to ν_1 except that $x = 0$.
- $A, (q_1, \nu_1) \models \exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle} \phi_1 \mathcal{U}^\epsilon \phi_2$ iff there exists a run $\rho = ((q_1, \nu_1), t_1)((q_2, \nu_2), t_2)\ldots$, an $i \geq 1$, and a $\delta \in [0, t_{i+1} - t_i]$ such that
  - $\rho \models \overset{\infty}{\Diamond} \{\alpha_1, \ldots, \alpha_m\} \wedge \overset{\infty}{\Box} \{\beta_1, \ldots, \beta_n\}$;
  - if ϵ is null, then $A, (q_i, \nu_i + \delta) \models \phi_2$ and for all j, δ' such that either $(0 \leq j < i) \wedge (\delta' \in [0, t_{j+1} - t_j])$ or $(j = i) \wedge (\delta' \in [0, \delta))$, $A, (q_j, \nu_j + \delta') \models \phi_1$;
  - if ϵ is an event predicate, then there is an $e \in E$ with $q_i \xrightarrow{e} q_{i+1} \wedge e \models \epsilon$ such that $A, (q_{i+1}, \nu_{i+1}) \models \phi_2$ and for all j, δ' with $(0 \leq j \leq i) \wedge (\delta' \in [0, t_{j+1} - t_j])$, $A, (q_j, \nu_j + \delta') \models \phi_1$.
- $A, (q_1, \nu_1) \models \exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle} \Box^\epsilon \phi_1$ iff there exists a run $\rho = ((q_1, \nu_1), t_1)(q_2, \nu_2), t_2)\ldots$ such that
  - $\rho \models \overset{\infty}{\Diamond} \{\alpha_1, \ldots, \alpha_m\} \wedge \overset{\infty}{\Box} \{\beta_1, \ldots, \beta_n\}$;
  - if ϵ is null, then for every $i \geq 1$ and $\delta \in [0, t_{i+1} - t_i]$, $A, (q_i, \nu_i + \delta) \models \phi_2$.
  - if ϵ is an event predicate, then for every $i \geq 1$ and $e \in E$ such that $q_i \xrightarrow{e} q_{i+1} \wedge e \models \epsilon$ implies $A, (q_{i+1}, \nu_{i+1}) \models \phi_2$.

A timed automaton $A = \langle \Sigma, X, Q, I, \mu, E, \gamma, \lambda, \tau, \pi \rangle$ satisfies a $TECTL^f$ formula ϕ, in symbols $A \models \phi$, iff for every state $(q_0, \nu_0) \models I$, $A, (q_0, \nu_0) \models \phi$. ■

4 Model-Checking Algorithm

The key component in the algorithm is for the construction of state-space representations for the following formula:

$$\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle} \Box^\epsilon \phi_1 \qquad \text{(NZF: non-Zeno Fairness)}$$

for any given $\alpha_1, \ldots, \alpha_m, \beta_1, \ldots, \beta_n$, ϵ (specified or not), and ϕ_1. Then in subsection 4.2, we present a symbolic algorithm for the construction of state-space

representations characterized by NZF formulas. In subsection 4.3, we present our symbolic evaluation algorithm for $TECTL^f$ formulas. Finally in subsection 4.4, we discuss an alternative NZF evaluation algorithm to the one in subsection 4.2. We shall compare the performance of these two algorithms in section 5.

4.1 Basic Reachability Procedures

We need two basic procedures, one for the computation of weakest preconditions of discrete transitions and the other for those of backward time-progressions. Details about the two procedures can be found in [15,21–24,26]. Given a state-space representation η and a discrete transition e, the first procedure, $\texttt{xtion\_bck}(\eta, e)$ with $\gamma(e) = (q, q')$, computes the weakest precondition

- in which every state satisfies the invariance condition $\mu(q)$; and
- from which we can transit to states in η through e.

η can be represented as a DBM set [12] or as a BDD-like data-structures [21, 23, 25]. Our algorithms are independent of the representation scheme of η. The second procedure, $\texttt{time\_bck}(\eta)$, computes the space representation of states

- from which we can go to states in η simply by time-passage; and
- every state in the time-passage also satisfies the invariance condition imposed by $\mu()$ for whatever modes the states are in.

With the two basic procedures, we can construct the event version of the symbolic backward reachability procedure, denoted $\texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}(\eta_0, \eta_1, \eta_2)$ for convenience, as in [15, 21–24, 26]. Intuitively, $\texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}(\eta_0, \eta_1, \eta_2)$ characterizes the state-space for $\exists \eta_1 \mathcal{U} \eta_2$ except for the following three differences.

- Only transitions satisfying β, a state predicate, are permitted in the fixpoint calculation; and
- When ϵ is null, all states constructed in the least fixpoint iterations must satisfy η_0; and
- When ϵ is an event predicate, the postcondition immediately after a transition satisfying ϵ must satisfy η_0.

The design of this procedure is for the evaluation of formulas like $\exists \Box^{\epsilon} \phi_1$. Note that the semantics of $\exists \Box^{\epsilon} \phi_1$ says that immediately after a discrete transition satisfying ϵ, the destination state must satisfy ϕ_1; in all other cases, the states are not required to satisfy ϕ_1.

Computationally, when ϵ is null, $\texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}(\eta_0, \eta_1, \eta_2)$ can be defined as the least fixpoint of equation:

$$Y = \eta_2 \vee \Big(\eta_0 \wedge \eta_1 \wedge \texttt{time\_bck}(\eta_0 \wedge \eta_1 \wedge \bigvee_{e \in \mathbf{E}; e \models \beta} \texttt{xtion\_bck}(Y, e))\Big).$$

i.e., $\texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}(\eta_0, \eta_1, \eta_2) \equiv$

$$\texttt{lfp}Y.\ \Big(\eta_2 \vee \Big(\eta_0 \wedge \eta_1 \wedge \texttt{time\_bck}(\eta_0 \wedge \eta_1 \wedge \bigvee_{e \in \mathbf{E}; e \models \beta} \texttt{xtion\_bck}(Y, e))\Big)\Big).$$

The monotonicity of F in fixpoint equation $Y = F(Y)$ ensures the computability of the least fixpoint. Given an event predicate ϵ, we let $\mathbf{E}^{\epsilon}$ be the set of discrete transitions satisfying ϵ while letting $\mathbf{E}^{\neg\epsilon}$ be the set of those discrete transitions not satisfying ϵ. When ϵ is an event predicate, $\texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}(\eta_0, \eta_1, \eta_2) \equiv$

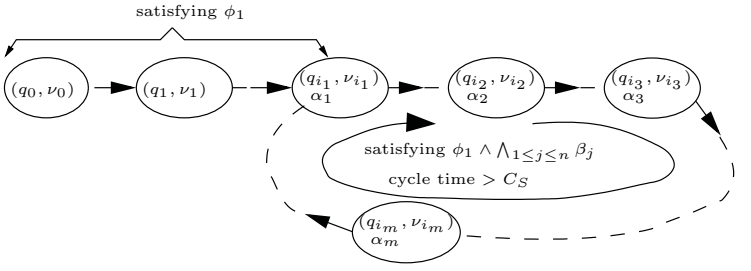


Fig. 3. The run segment for NZF $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^{\epsilon}\phi_1$ and $q_{i_m} = q_k$ and $\nu_{i_m} = \nu_k$

$$\texttt{lfp}Y.\eta_2 \vee \eta_1 \wedge \texttt{time\_bck}\left(\eta_1 \wedge \left(\begin{array}{c} \bigvee_{e\in\mathbf{E}^{\epsilon};e\models\beta} \texttt{xtion\_bck}(Y \wedge \eta_0, e) \\ \vee \bigvee_{e\in\mathbf{E}^{\neg\epsilon};e\models\beta} \texttt{xtion\_bck}(Y, e) \end{array}\right)\right)$$

Note that while calculating the fixpoint, we use

- formula $\bigvee_{e\in\mathbf{E}^{\epsilon};e\models\beta} \texttt{xtion\_bck}(Y \wedge \eta_0, e)$ to enforce that states immediately after transitions satisfying ϵ must satisfy η_0; and
- formula $\bigvee_{e\in\mathbf{E}^{\neg\epsilon};e\models\beta} \texttt{xtion\_bck}(Y, e)$ to enforce that states immediately after transitions not satisfying ϵ have no obligation.

4.2 Evaluation of NZF

To maintain the non-Zeno run quantification of TCTL, it means that when we say something like "a run along which all fairness assumptions are honored," we really mean *"a run along which all fairness assumptions are honored AND TIME DIVERGES."* Considering that time-divergence means *"time increases by at least $c \ge 1$ infinitely often,"* non-Zenoness can actually be considered as a strong fairness condition.

An NZF formula $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^{\epsilon}\phi_1$ is satisfied at a state (q_0, ν_0) with a cycle of states and a path leading from (q_0, ν_0) to the cycle such that

- if ϵ is null, all states along the path and cycle satisfy ϕ_1;
- if ϵ is an event predicate, all states immediately following a transition in E^{ϵ} along the path and cycle must satisfy ϕ_1;
- all states along the cycle satisfy the $TECTL^f$ formulas in $\beta_1, \ldots, \beta_n$;
- all transitions along the cycle satisfy the event predicates in $\beta_1, \ldots, \beta_n$;
- for each of $1 \le i \le m$, α_i is satisfied at least once along the cycle; and
- the cycle time is greater than $c \ge 1$.

In our experience [27], we found that the biggest constant, denoted C_S, used in the timed automaton and the $TECTL^f$ formula is usually a proper choice for c for efficient evaluation. A picture for such a run segment from (q_0, ν_0) is in figure 3. For convenience, we need the following notations. Let $L[\phi_1]$ be the state-predicate characterizing the set of states satisfying ϕ_1 and $\texttt{gfp}X.F(X)$ denote the greatest solution to the the equation of $X = F(X)$. Similarly, the monotonicity of F in fixpoint equation $Y = F(Y)$ ensures the computability of the greatest fixpoint.

Also, let $B^\epsilon_0(X) \equiv x > C_S$ for all X. Finally for conciseness, let β and $\check{\beta}$ be the shorthands for $\bigwedge_{1\le j\le n;\beta_j \text{is an event predicate.}} \beta_j$ and $\bigwedge_{1\le j\le n;\beta_j \text{is an event predicate}} \beta_j$ respectively. The state space satisfying $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^\epsilon\phi_1$ can then be characterized as follows.

$$\texttt{reachable}^\epsilon\texttt{\_bck}_{true}(L[\phi_1], true, \texttt{gfp}X.(x.B^\epsilon_m(X))) \qquad \text{(FX\_NZF)}$$

where for all $1 \le i \le m$,

- if α_i is a *TECTL^f* formula, $B^\epsilon_i(X)$ is defined as
$$\texttt{reachable}^\epsilon\texttt{\_bck}_\beta\left(L[\phi_1], L[\check{\beta}], L[\alpha_i] \wedge B^\epsilon_{i-1}(X)\right)$$
- if ϵ is null and α_i is an event predicate, $B^\epsilon_i(X)$ is defined as
$$\texttt{reachable}^\epsilon\texttt{\_bck}_\beta\left(L[\phi_1], L[\check{\beta}], \bigvee_{e\in E; e\models\alpha_i\wedge\beta} \texttt{xtion\_bck}(B^\epsilon_{i-1}(X), e)\right)$$
- if ϵ and α_i are both event predicates, $B^\epsilon_i(X)$ is defined as
$$\texttt{reachable}^\epsilon\texttt{\_bck}_\beta\left(L[\phi_1], L[\check{\beta}], \begin{pmatrix}\bigvee_{e\in E^\epsilon; e\models\alpha_i\wedge\beta} \texttt{xtion\_bck}(B^\epsilon_{i-1}(X)\wedge L[\phi_1], e)\\ \vee\bigvee_{e\in E^{\neg\epsilon}; e\models\alpha_i\wedge\beta} \texttt{xtion\_bck}(B^\epsilon_{i-1}(X), e)\end{pmatrix}\right)$$

The cycle is embodied through the greatest fixpoint $\texttt{gfp}X.(x.B^\epsilon_m(X))$. Note that the weak fairness assumptions as event predicates are enforced through β as the restriction to transitions. The other weak ones are enforced through $L[\check{\beta}]$ as the second argument to $\texttt{reachable}^\epsilon\texttt{\_bck}_\beta()$. The cycle is composed of m segments, each of which fulfills a strong fairness assumption and is constructed differently according to whether the corresponding strong assumption is for a *TECTL^f* formula or an event predicate. In the 2nd and 3rd items, for an event predicate strong assumption α_i, we start constructing the corresponding segment from the precondition of the event, i.e., $\bigvee_{e\in\ldots; e\models\alpha_i\wedge\beta} \texttt{xtion\_bck}(\ldots, e)$. The fulfillment of the assumptions are checked with the existence of such cycles. The following lemma is for the correctness of this formulation.

Lemma 1. *: formula (FX_NZF) characterizes the set of states which satisfy* $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^\epsilon\phi_1$. ■

Suppose that we have already calculated the state-space representations $L[\phi']$ for all the proper subformulas ϕ' of $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^\epsilon\phi_1$. We can use the following algorithm to compute state-predicates for NZFs.

NZF($[\alpha_1,\ldots,\alpha_m], \langle\beta_1,\ldots,\beta_n\rangle, \epsilon, W$)
/* where W is $L[\phi_1]$, i.e., the state-predicate for states satisfying ϕ_1. */ {
  let $\check{\beta}$ be $\bigwedge_{1\le j\le n;\beta_j \text{is not an event predicate.}} \beta_j$;
  let β be $\bigwedge_{1\le j\le n;\beta_j \text{is an event predicate.}} \beta_j$;
  $R := W \wedge L[\check{\beta}]$; $R' = false$;
  Repeat until $R = R'$, {
    $R' := R$; $R := R \wedge x > C_S$, where x is never used anywhere else. (1)
    for $i := m$ to 1, if α_i is not an event predicate,
      $R := \texttt{reachable}^\epsilon\texttt{\_bck}_\beta(W, L[\check{\beta}], R \wedge L[\alpha_i])$; (2)
    else if ϵ is null,

$R := \texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}(W, L[\check{\beta}], \bigvee_{e\in E;e\models\beta\wedge\alpha_i} \texttt{xtion\_bck}(R, e))$; (3)

else

$R := \texttt{reachable}^{\epsilon}\texttt{\_bck}_{\beta}\left(W, L[\check{\beta}], \left(\begin{array}{l}\bigvee_{e\in E^{\epsilon};e\models\beta\wedge\alpha_i} \texttt{xtion\_bck}(R\wedge L[\phi_1], e)\\ \vee \bigvee_{e\in E^{\neg\epsilon};e\models\beta\wedge\alpha_i} \texttt{xtion\_bck}(R, e)\end{array}\right)\right)$; (4)

$R := R' \wedge \texttt{var\_eliminate}(\texttt{bypass}(x = 0 \wedge R, x), \{x\})$; (5)

}

return $\texttt{reachable}^{\epsilon}\texttt{\_bck}_{true}(W, true, R)$;

}

The repeat-loop in NZF() calculates state-space representation of $\texttt{gfp}X.(x.B_m(X))$. The inner for-loop iterates to check the fulfillment of $\alpha_m, \alpha_{m-1}, \ldots, \alpha_1$ in sequence. Through the backward reachability analysis steps from statements (1) to (4), we expect the cycle to go backward from states with $x > C_S$ (when $C_S \geq 1$) to states with $x = 0$. The cyclic execution time $> C_S$ is enough for non-Zenoness. The return statement uses one final least fixpoint procedure on the result of the repeat-loop to calculate the state-space representatoin for formulation (FX_NZF).

In procedure NZF(), we use procedure $\texttt{var\_eliminate}(\eta, \{x\})$ which partially implements the Fourier-Motzkin elmination [14] and will eliminate all information in state-predicate η related to x. But before the application of this procedure, we first apply procedure $\texttt{bypass}(\eta, x)$ which will add to η any transitivity information deducible from x. For example,

$$\begin{aligned}&\texttt{var\_eliminate}(\texttt{bypass}(x < 3 \wedge y - x \leq -2, x), \{x\})\\ &= \texttt{var\_eliminate}(x < 3 \wedge y - x \leq -2 \wedge y < 1, \{x\})\\ &= y < 1\end{aligned}$$

Note that in the first step, new constraint $y < 1$ is deduced. Details of procedures `var_eliminate`() and `bypass`() can be found in [25].

4.3 Evaluation of $TECTL^f$ Formulas

The following evaluation algorithm uses the procedures presented in the last two subsections as basic blocks to evaluate $TECTL^f$ formulas.

Eval$(A, \bar{\phi})$ {

switch $(\bar{\phi})$ {

case $(false)$:	return $false$;
case (q):	return q;
case $(x \sim c)$:	return $x \sim c$; ;
case $(\phi_1 \vee \phi_2)$:	return Eval$(A, \phi_1) \vee$ Eval(A, ϕ_2);
case $(\neg\phi_1)$:	return $\neg$Eval(A, ϕ_1);
case $(x.\phi_1)$:	return $\texttt{var\_eliminate}(\texttt{bypass}(x = 0 \wedge \text{Eval}(A, \phi_1), x), \{x\})$;
case $(\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\phi_1\mathcal{U}^{\epsilon}\phi_2)$:	$Y_1 :=$ Eval(A, ϕ_1); $Y_2 :=$ Eval$(A, \phi_2) \wedge$ NZF$([\alpha_1, \ldots, \alpha_m], \langle\beta_1, \ldots, \beta_n\rangle, \texttt{NULL}, true)$; if ϵ is an event predicate, $Y_2 := \bigvee_{e\in E^{\epsilon}} \texttt{xtion\_bck}(Y_2, e)$; return $\texttt{reachable\_bck}_{true}(true, Y_1, Y_2)$;
case $(\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^{\epsilon}\phi_1)$:	$W :=$ Eval(A, ϕ_1); return NZF$([\alpha_1, \ldots, \alpha_m], \langle\beta_1, \ldots, \beta_n\rangle, \epsilon, W)$;

}

}

Theorem 1. *Given a timed automaton A, a* TECTL^f *formula ϕ, and a state (q,ν), $(q,\nu) \models \phi$ iff $(q,\nu) \models Eval(A,\phi)$.*

Proof: The framework of the proof is a standard induction on the structure of ϕ. Other than that, we also need to check that instead of evaluating $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\phi_1\mathcal{U}^\epsilon \phi_2)$, we evaluate $\exists\phi_1\mathcal{U}^\epsilon\left(\phi_2 \wedge \exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box true\right)$. The NZF is asserted additionally when the liveness property ϕ_2 is fulfilled. Finally, when ϵ is not null, we start the least fixpoint evaluations for $\exists\mathcal{U}^\epsilon$-formulas from the preconditions of those transitions that satisfying ϵ. This is compatible with the semantics of $\mathcal{U}^\epsilon$. ■

Given a timed automaton A and a *TECTL^f* formula ϕ, A satisfies ϕ iff Eval$(A,\neg\phi)$ is *false*.

4.4 Another Algorithm for NZF Evaluation

There can be other algorithms for the valuation of NZF. We have experimented with an algorithm that uses m auxiliary variables $a_1,\ldots,a_m$ for the bookkeeping of strong fairness assumption fulfillment in NZF. The auxiliary variables serve the purpose to flag whether along the computation, corresponding strong fairness assumptions have been fulfilled. The advantage of this algorithm is that inside the greatest fixpoint evaluation, we only need one iteration of least fixpoint evaluation with the new procedure `reachable_aux_vars`$^\epsilon$`_bck`$_\beta()$ instead of m iterations of least fixpoint evaluation with `reachable`$^\epsilon$`_bck`$_\beta()$. The idea is that initially we start `reachable_aux_vars`$^\epsilon$`_bck`$_\beta()$ from states where $a_1,\ldots,a_m$ are all false. Then in the execution of `reachable_aux_vars`$^\epsilon$`_bck`$_\beta()$, each time we compute a weakest precondition from a basic time-progression (`time_bck`()) or from a discrete transition (`xtion_bck`()), we check whether a strong assumption α_i has just been fulfilled and set the corresponding a_i to true if it has. In the end of this single iteration of least fixpoint evaluation, the states in the cycles that have fulfilled all strong fairness assumptions are those with $a_1 \wedge \ldots \wedge a_m$ true. Then the state space satisfying $\exists^{[\alpha_1,\ldots,\alpha_m]}_{\langle\beta_1,\ldots,\beta_n\rangle}\Box^\epsilon\phi_1$ can also be characterized with the following formulation.

```
reachable^ε_bck_true(
  L[φ1], true,
  gfpX.var_eliminate(
    (x.reachable_aux_vars^ε_bck_β(L[φ1], L[β̃], X ∧ x > C_S ∧ ⋀_{1≤i≤m} ¬a_i)) ∧ ⋀_{1≤i≤m} a_i,
    {a_1, ..., a_m}
  )
)
```

We shall report the performance comparison in the next section.

5 Implementation and Experiment

We have implemented the ideas in our model-checker/simulator, **Red** version 5.1, for communicating timed automata [19]. The events are interpreted as input-output event pairs through communication channels. **Red** uses the new BDD-like data-structure, *CRD* (Clock-Restriction Diagram) [23–25], and supports both

Table 1. Performance data of scalability w.r.t. number of processes

spec.	# proc's	Linear networks		Tree networks		Lattice networks	
		sequential	aux. var.s	sequential	aux. var.s	sequential	aux. var.s
(A)	2	0.03s/5k	0.01s/5k	0.02s/5k	0.01s/5k	0.02s/5k	0.00s/5k
	4	0.10s/14k	0.09s/14k	0.08s/14k	0.13s/14k	0.14s/16k	0.17s/18k
	6	0.53s/28k	0.47s/28k	0.47s/27k	1.00s/35k	0.81s/34k	1.93s/58k
	8	1.86s/46k	1.71s/46k	1.99s/46k	4.82s/70k	5.42s/64k	15.0s/192k
	10	5.16s/70k	4.74s/70k	5.45s/70k	19.1s/145k	17.0s/93k	99.8s/697k
	12	11.7s/100k	11.0s/100k	15.0s/99k	68.7s/301k	64.3s/141k	781s/2452k
	14	23.7s/136k	23.4s/136k	34.4s/135k	239s/635k	219s/198k	6289s/9203k
	16	44.4s/179k	45.4s/179k	82.1s/179k	837s/1315k	1046s/377k	52383s/35356k
	18	79.1s/229k	82.5s/229k	163s/229k	3098s/2801k	2074s/601k	Not
	20	127s/288k	143s/288k	341s/287k	12532s/5961k	5381s/1011k	Available
(B)	2	0.00s/5k	0.00s/5k	0.01s/5k	0.02s/5k	0.00s/5k	0.00s/5k
	4	0.01s/14k	0.02s/14k	0.04s/14k	0.03s/14k	0.01s/16k	0.02s/16k
	6	0.05s/27k	0.08s/27k	0.10s/27k	0.12s/27k	0.18s/33k	0.21s/33k
	8	0.15s/46k	0.17s/46k	0.39s/46k	0.41s/46k	0.85s/64k	0.85s/64k
	10	0.29s/70k	0.36s/70k	1.12s/70k	1.22s/70k	3.11s/93k	3.29s/93k
	12	0.59s/99k	0.59s/99k	2.83s/99k	3.14s/99k	10.7s/141k	10.9s/141k
	14	0.99s/136k	1.05s/136k	6.95s/135k	7.43s/145k	34.1s/198k	34.8s/198k
	16	1.56s/179k	1.69s/179k	15.5s/179k	16.4s/179k	107s/363k	108s/363k
	18	2.38s/230k	2.48s/230k	34.0s/229k	35.3s/229k	280s/570k	287s/570k
	20	3.42s/288k	3.57s/288k	70.5s/288k	73.2s/288k	789s/1010k	792s/1010k
(C)	2	0.02s/5k	Not	0.02s/5k	Not	0.01s/5k	Not
	4	0.19s/14k	Available	0.16s/14k	Available	0.17s/16k	Available
	6	1.14s/28k		0.92s/28k		0.89s/33k	
	8	4.85s/46k		3.70s/46k		4.76s/64k	
	10	15.6s/70k		12.5s/70k		14.2s/93k	
	12	42.1s/99k		37.7s/103k		50.0s/141k	
	14	100s/136k		103s/174k		174s/259k	
	16	212s/178k		282s/306k		604s/629k	
	18	433s/229k		689s/474k		1358s/906k	
	20	791s/288k		1878s/877k		4096s/1630k	

data collected on a Pentium 4 Mobile 1.6GHz with 256MB memory running LINUX;
s: seconds; k: kilobytes of memory in data-structure; N/A: not available;

forward and backward analyses, full $TECTL^f$ model-checking with non-Zeno computations, deadlock detection, and counter-example 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. **Red** 5.1 also accepts $TECTL^f$ formulas with quantifications on process identifiers for succinct specification.

We use the diffusive computing algorithm [10] in figure 2, with the three network configurations in figures 1, to demonstrate the performance of our techniques. The $TECTL^f$ specifications in our experiment include properties (A), (B) and (C) in example 1 in page 558. In table 1, please find the performance data.

Data in the "sequential" columns were collected with the NZF evaluation algorithm in subsection 4.2 while those in the "aux. var.s" columns were with the algorithm with auxiliary variables. Specification (C) does not have fairness assumptions and does not use options for choosing NZF algorithms.

Our techniques seem quite efficient for the benchmarks. When we carefully check the data, we find that **Red** 5.1 can actually handle much higher concurrencies since the memories used at 20 clocks are still not very much in the "sequential" columns. It is also interesting to see that the NZF evaluation algorithm in subsection 4.2 performs better than the alternative in subsection 4.4. Though the latter needs less iterations of least fixpoint evaluations inside the greates fixpoint evaluation, the single iteration of $\texttt{reachable\_aux\_vars}^{\epsilon}\texttt{\_bck}_{\beta}()$ incurs huge memory complexities and usually performs worse than the algorithm in subsection 4.2. This is understandable since the complexity of BDD-based algorithms is usually exponential to the number of variables.

6 Conclusion

We investigate how to add the concepts of events and fairness assumptions to TCTL model-checking and define the new language of $TECTL^f$. Our framework allows for the specification and verification of punctual event properties and multiple strong and weak fairness assumptions. Our implementation and experiments have shown that the ideas could be of practical use. More research are expected to develop useful specification languages for distributed real-time systems.

References

1. R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems, IEEE LICS, 1990.
2. R. Alur, D.L. Dill. Automata for modelling real-time systems. ICALP' 1990, LNCS 443, Springer-Verlag, pp.322-335.
3. R. Alur, T.A. Henzinger. A really temporal logic, 30th IEEE FOCS, pp.164-169, 1989.
4. R. Alur, T.A. Henzinger. Real-Time Logics: Complexity and Expressiveness. Information and Computation **104**, 35-77, 1993.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L.Dill, L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond, IEEE LICS, 1990.
6. M. Bozga, C. Daws. O. Maler. Kronos: A model-checking tool for real-time systems. 10th CAV, June/July 1998, LNCS 1427, Springer-Verlag.
7. J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, Wang Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. Hybrid Control System Symposium, 1996, LNCS, Springer-Verlag.
8. R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., C-35(8), 1986.
9. E. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic, Proceedings of Workshop on Logic of Programs, Lecture Notes in Computer Science 131, Springer-Verlag, 1981.
10. Chandy, Misra. Parallel Program Design - A Foundation, Addison-Wesley, 1988.
11. S. Chaki, E.M. Clarke, J. Ouaknine, N. Sharygina, N. Sinha. State/Event-based Software Model Checking. IFM 2004, LNCS 2999, Springer-Verlag.

12. D.L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. CAV'89, LNCS 407, Springer-Verlag.
13. E.A. Emerson, C.-L. Lei. Modalities for Model Checking: Branching Time Logic Strikes Back, Science of Computer Programming 8 (1987), pp.275-306, Elsevier Science Publishers B.V. (North-Holland).
14. J.B. Fourier. (reported in:) Analyse des travaux de l'Académie Royale des Sciences pendant l'année 1824, Partie Mathématique, 1827.
15. T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems, IEEE LICS 1992.
16. M. Huth, R. Jagadeesan, D. Schmidt. Modal transition systems: A foundation for three-valued program analysis. ESOP 2001, LNCS 2028, Springer Verlag.
17. E. Kindler, T. Vesper. ESTL: A Temporal Logic for Events and States. ATPN 1998, LNCS 1420, Springer-Verlag.
18. D. Kozen. Results on the propositional mu-calculus. Theoretical Computer Science, 27:333-354, 1983.
19. A. Shaw. Communicating Real-Time State Machines. IEEE Transactions on Software Engineering 18(9), September, 1992.
20. R. De Nicola, F. Vaandrager. Three Logics for Branching Bisimulation. Journal of the ACM (JACM), 42(2):458-487, 1995.
21. F. Wang. Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems. TACAS'2000, LNCS 1785, Springer-Verlag.
22. F. Wang. Region Encoding Diagram for Fully Symbolic Verification of Real-Time Systems. the 24th COMPSAC, Oct. 2000, Taipei, Taiwan, ROC, IEEE press.
23. F. Wang. RED: Model-checker for Timed Automata with Clock-Restriction Diagram. Workshop on Real-Time Tools, Aug. 2001, Technical Report 2001-014, ISSN 1404-3203, Dept. of Information Technology, Uppsala University.
24. F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram, Proceedings of FORTE, August 2001, Cheju Island, Korea.
25. F. Wang. Efficient Verification of Timed Automata with BDD-like Data-Structures, to appear in special issue of STTT (Software Tools for Technology Transfer, Springer-Verlag) for VMCAI'2003, LNCS 2575, Springer-Verlag.
26. F. Wang, P.-A. Hsiung. Efficient and User-Friendly Verification. IEEE Transactions on Computers, Jan. 2002.
27. F. Wang, G.-D. Huang, F. Yu. TCTL Inevitability Analysis of Dense-Time Systems. 8th CIAA (Conference on Implementation and Application of Automata), July 2003, Santa Barbara, CA, USA; LNCS 2759, Springer-Verlag.