SOFTWARE TESTING, VERIFICATION AND RELIABILITY

Softw. Test. Verif. Reliab. ; **00**:1–41

Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

A Simplification of a Real-Time Verification Problem

Suman Roy[1*] , Janardan Misra[2*] and Indranil Saha[3†]

[3]*Department of EECS, Uni. of California, Berkeley, CA 94720-1770, USA.*

[2]*Accenture Technology (AT) Labs, Bangalore, India.*

[1]*Infosys Labs, Infosys Ltd., Bangalore, India.*

SUMMARY

We revisit the problem of real-time verification with dense time dynamics using timeout and calendar based models, and simplify this to a finite state verification problem. We introduce a specification formalism for these models and capture their behavior in terms of semantics of Timed Transition Systems. We discuss a technique, which reduces the problem of verification of qualitative temporal properties on infinite state space of (a large fragment of) these timeout and calender based transition systems into that on clockless finite state models through a two-step process comprising of digitization and finitary reduction. This technique enables us to verify `safety` invariants for real-time systems using finite state model-checking avoiding the complexity of infinite state (bounded) model checking and scale up models without applying techniques from induction based proof methodology. Moreover, we can verify `liveness` and timeliness properties for real-time systems, which cannot be easily performed by using induction with infinite state model checkers.

Received . . .

KEY WORDS: Real-Time Systems; Model Checking; Timeout and Calendar Model; Digitization; Finite State Verification; Temporal Logics

*E-mail: suman_roy@infosys.com

†E-mail: saha.indra@gmail.com

*Correspondence to: AT Labs, Bangalore, India. E-mail: janardanm@acm.org

1. INTRODUCTION

Real-time systems are an important class of mission critical systems, which have been well studied for their design, implementation, performance and verification. Modeling and verification of real-time systems in dense time domain is an important problem area that evoked significant research interest in the recent past [1, 2, 3, 4, 5, 6]. Because of the fact that the state space of real-time systems with continuous dynamics is uncountable, modeling and verification of them is rather difficult. Using explicit model checkers [2] one usually encoutners the problem of the state explosion, in which the number of system states grows exponentially with the number of system components thus, limiting the applicability of such techniques in most of the industrial applications. In order to overcome such complexity of verification of real-time systems with continuous time domain, Dutertre and Sorea proposed timeout and calendar based transition models, [4, 7], originally from discrete event simulation, to represent the behavior of timed triggered systems. These models are amenable to be used in general-purpose verification environments, like SAL [8] in which state machines and their compositions can be specified. While primarily `safety` properties can be verified on these models with dense time, discrete time modeling of the same can help verify `liveness` and `timeliness` properties, and also help scale up proofs for `safety` properties.

It turns out that verification of a real-time system in dense domain is equivalent to verifying the system in discrete domain if both the behavior of the system captured by the model and the properties considered are digitizable [9]. Also verification of qualitative properties like `safety` and `liveness`, in discrete time domain is equivalent to verifying these properties in dense time domain (refer to Proposition 1 in [9]). In this work, we aim to carry out a finite state modeling and verification on timeout and calendar models without continuously varying clocks. As there are drawbacks of those models earlier proposed [4, 7, 10] from the point of view of design considerations like, the absence of formally defined syntactic models and associated semantics, we slightly deviate from this approach. We consider the specification framework of timed transition diagrams and extend it to formalize timeout and calendar based models as timeout and calendar

Prepared using stvrauth.cls
DOI: 10.1002/stvr

based transition diagrams and their behavior in terms of semantics of transition systems. The benefits that we derive from using this formalization are many-fold. Our framework of timeout transition diagrams (a graphical representation of timed transitions system a la [11]) inherits most of the properties of classical timed transition system (TTS) introduced in [11] including the digitizability from therein. This can be also used to model time-triggered systems and reason about them. Further, we use this formal modeling framework to reduce continuous time verification problem to discrete time finite state verification, albeit under the restriction of delimitation of the timeout values. Towards that, we use a two step technique comprising of digitization and finitary reduction (a schematic diagram of this technique is shown in Fig. 1). We show that the computations of timeout and calendar models are digitizable provided the timeout increments are not allowed to occur within $(0, 1)$-interval. As LTL properties are qualitative and hence, are digitizable, verification of LTL properties on timeout and calendar models in dense time is equivalent to that in discrete time. The next step is to reduce this problem into an equivalent finite state verification problem. We propose a finitary reduction technique which effectively reduces the infinite state timeout and calendar based transition systems with discrete dynamics into a finite state transition system. We achieve this by using a clockless modeling technique which effectively strips the model of the global clock and keeps track of the relative update of timeouts, and restricts the values of variables/timeout updates to bounded domains. We demonstrate by examples, how such a modeling approach can be efficiently used for verifying safety, liveness, and timeliness properties using finite state model checkers, SAL-smc and Spin. We also highlight the scalability of such models for verification purposes by comparing the performance of such models under dense time and finite state modeling. Some of these results appeared in [12].

Related work: Lamport also uses timer variables similar to timeout variables, to express timing requirements in an explicit time approach [13]. In this approach the timing variable x representing global time is incremented by Tick action. While Tick increments x by any real number in a continuous time specification, it increments x by 1 time unit in discrete time specification. The author uses three variables to specify timing bounds on actions, a countdown timer which is

Prepared using stvrauth.cls
DOI: 10.1002/stvr

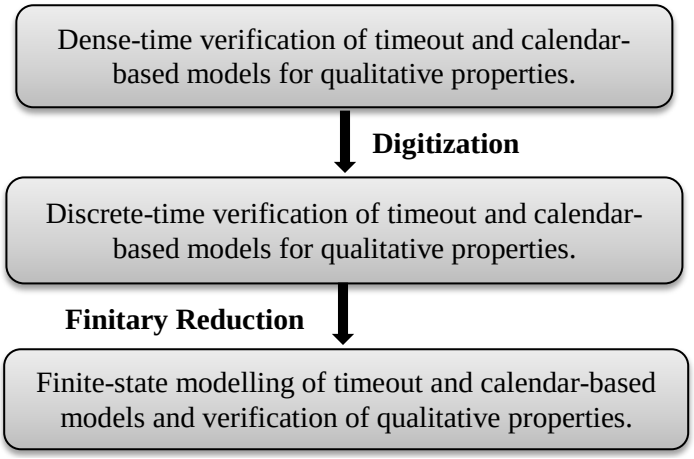

Figure 1: A two-step verification process

decremented by the Tick action, a count-up timer which is incremented by Tick, and an expiration timer that is left unchanged by Tick. When the value of countdown or count-up timer reaches a specified value it expires. An expiration timer expires when the its difference with timing variable x reaches some specified value. It is interesting to compare this approach with timeout-based models. Timeouts are like countdown timers, once the timing variable x reaches the minimum of them, the minimum-valued timeout is set to zero and is assigned to a new value. Also the expiration timer expires in a manner similar to timeouts in clockless models in our approach where we check the difference of timeout values and x and pick up the timeout value for resetting for which the difference becomes zero.

The concept of clockless modeling has been introduced around the same time in [10]. In that Pike builds on the work of [4] and proposes a new formalism called Synchronizing Timeout Automata (STA) to reduce the induction depth k required for k-induction. He introduces a clockless semantics for STA so that the resulting transition system does not involve a clock. STA in effect, describes the overall system architecture in terms of timeout transition system introduced in [4]. However, it turns out that STA cannot be used for modeling timeout models in general. For example, a closer analysis of the SAL model for the example of the Train-Gate Controller presented in [10], reveals that the model in consideration is not deadlock free. This is because the model fails to specify the

Prepared using stvrauth.cls
DOI: 10.1002/stvr

timeout update rules precisely for the transitions leading to a waiting state. These kinds of modeling errors could possibly be eliminated with a modeling framework such as the one proposed in this paper. Another work on clockless modeling is taken up in [14] by converting Timed Automata to an untimed version of Timed Automata. Building upon this, in [15], the authors discuss that by applying discretization on the continuous component of real-time systems, these models could be further translated into Promela models for verification by Spin Model Checker [16]. Spin has been used to finitely model TTA startup algorithm [12] using a clock-less calendar based model. In terms of scalability, finite state verification of TTA in Spin is almost comparable to the verification of TTA based on verification diagram oriented abstraction method [7]. In this paper, we extend the idea presented in [12] by presenting the complete framework and theoretical results on the soundness of the approach.

Organization of the paper The remainder of the paper is organized as follows. In Section 2, we present the formalization of these models in terms of timeout transition diagrams and their behavior in terms of the semantics of transition systems. We discuss the technique of digitization in Section 3 and present our first step of reduction of dense-time verification problem to integral time verification problem. In Section 4, we describe the finitary reduction technique and subsequently, formalize it in terms of clock-less modeling. We present experimental results in Section 5 demonstrating the effectiveness of the suggested approach. Section 6 concludes the paper.

2. FORMALIZATION OF TIMEOUT AND CALENDAR BASED MODELS

Timed automata (TA) is one of the most popular formalisms for specifying real-time systems. However, for systems with asynchronous communication having bounded delays between components, TA based modeling offers no efficient means of specification. Two possible alternatives have been proposed in the literature. The first one is to use state variables for encoding the behavior of asynchronous channels without any explicit provision for capturing message transmission delays. The second option is to model each channel as a separate TA with delay as a state variable. However with relatively large number of components and high degree of connectivity among them, modeling

Prepared using stvrauth.cls
DOI: 10.1002/stvr

channels in this way turns out to be difficult, and state space explosion becomes unavoidable. One faces the same problem with UPPAAL [17], when it is used to model asynchronous communications with bounded delays, - every channel has to be modeled as a separate TA capturing the message transmission delays.

Timeout based modeling approach is an effective alternative to model systems where the processes communicate via shared variables or, the communication between the processes is a rendezvous one. An extension involving a global data structure called *calendar* could be used to model even the inter-process communication delays during message transfers [7]. Currently, these timeout models are specified in terms of state transition systems (STS) [7]. This may offer advantages for performing verification, but STS are not well suited for system design purposes because they describe the behavior of the combined system without explicitly specifying the design of the modular components. More importantly, there is no adequate semantics associated with these timeout models. Therefore, there is a need to introduce formal design models with sound and complete semantics in order to be able to verify design as well use the verified design with minimal manual efforts for implementing actual systems. In [11] an abstract model of timed transition diagram was proposed to syntactically represent real-time concurrent processes. We extend it with a view to represent timeout and calendar based structural elements. Further we describe their associated semantics in terms of timed transition systems.

As an example of timeout based model, let us consider an example of Train-Gate Controller (TGC) adapted from [18] (see Figure 2). TGC is an automatic controller that controls the opening and closing of the Gate at railroad crossing. The system is composed of three components: Train, Gate, and Controller. The communications between the Train and the Controller, and between the Controller and the Gate are assumed to be synchronous. Before entering the railroad crossing the Train sends the signal *approach*. The Controller on receiving this signal is supposed to send the signal *lower* to the Gate within 1 time units and the Gate has to be lowered within another 1 time units. Thus the Train could enter the crossing at any time after 2 time units after it sent the *approach* signal. When exiting the crossing the Train sends the *exit* signal to the Controller. Also after sending

Prepared using stvrauth.cls
DOI: 10.1002/stvr

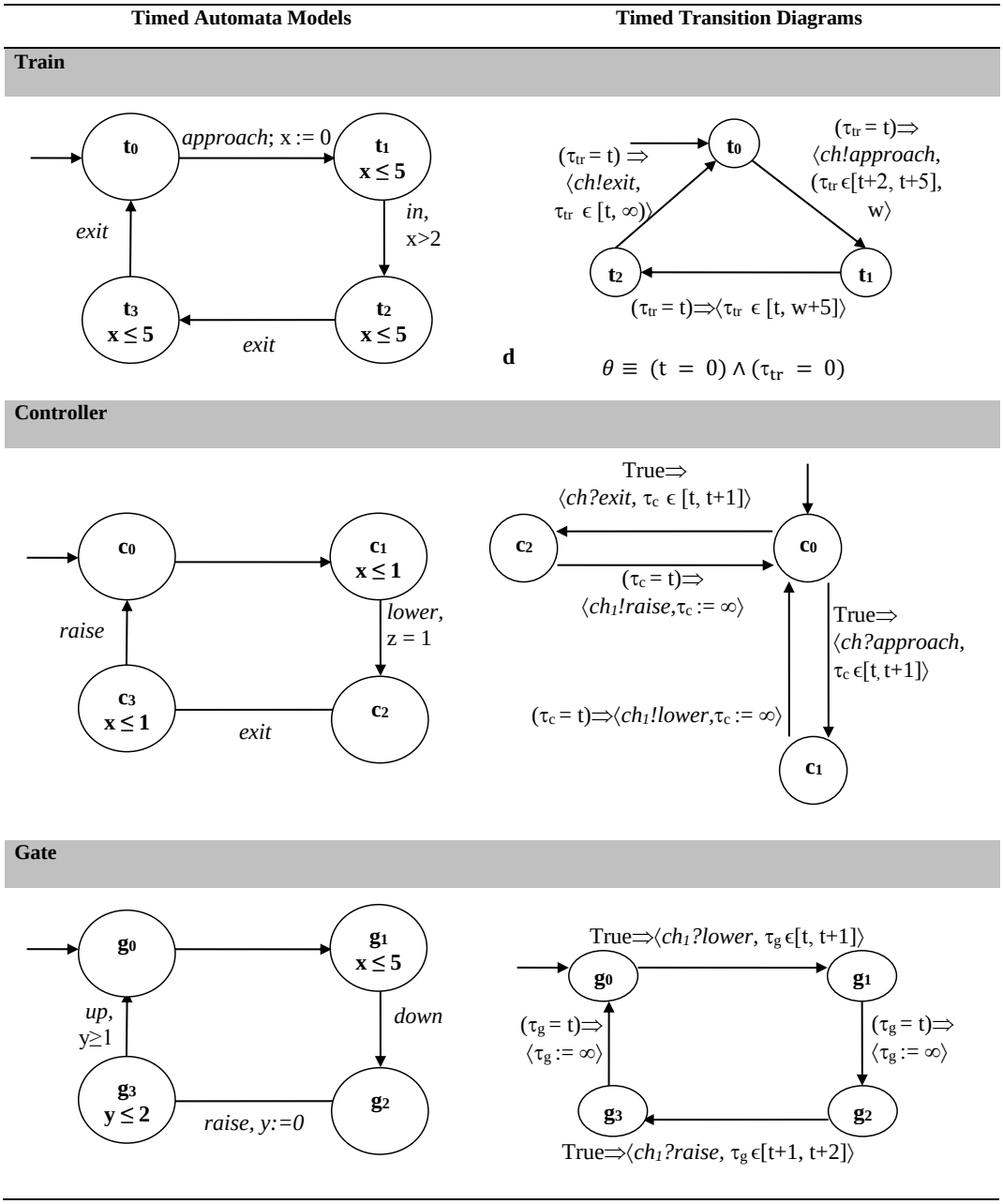

Figure 2: Timed automata and timeout models for Train Gate Controller (TGC)

the *approach* signal the Train must send the *exit* signal within 5 time units; the Controller sends the *raise* signal to the Gate within 1 time units after it receives the *exit* signal; and the Gate is required to be up within another 1 time units. As stated before, all the communications are assumed to be synchronous, that is, there is no message transmission delay.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Both the timed automata-based model (realizing a STS) and the timeout-based model of TGC (to be formally introduced later) (realizing a TTS) are shown in Figure 2. Notice that for the former model of TGC it is very difficult to figure an appropriate semantics in terms of timed execution sequences. Also a detailed design of each individual component is missing in these models. While the time transition diagram of TGC allows one specify these systems in terms of timed execution sequences and guides one to chalk out the design of each modular component in more detail. Moreover in the latter specification, the timing requirements are captured by suitably defining the $update$s on the edges. Setting a timeout to ∞ on an edge is used to denote the requirement of indefinite waiting for an external signal, for example, on taking a transition on edge from g_1 to g_2, $\tau_g := \infty$ denotes that the Gate is going to wait indefinitely for the signal $raise$ to be received on channel ch_1 from the Controller.

2.1. Timeout based Timed Transition Model

Syntax

A shared variable Timeout based Model (ToM) is represented as

$$P: \{\theta\}[P_1||P_2||\ldots||P_n],$$

where each P_i is a sequential non-deterministic process having τ_i as its local timeout and $\mathcal{X}_i$ as the set of local timing variables. Local timing variables are used for determining the relative delay between events. The shared variable t represents the global clock. The other variables are the set G of variables globally shared among the processes and the set L of variables local to processes P_i. The operator "||" denotes parallel composition. The formula θ, called the *data pre-condition* of P, restricts the initial values of variables in

$$\mathcal{U} = \{t\} \cup \mathcal{T} \cup \mathcal{X} \cup Var,$$

where the set of all timeouts is $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, the set of all local timing variables is $\mathcal{X} = \bigcup_i \mathcal{X}_i$, and the set of all global and local variables (excluding global clock) is $Var = (G \cup L_1 \cup L_2 \cup \ldots \cup L_n)$.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Each process P_i is represented using a *timeout transition diagram* (TTD), which is a finite directed graph with a set of nodes $Loc_i = \{l_0^i, l_1^i, \ldots, l_{m_i}^i\}$, called *locations*. The entry location is l_0^i, where the process P_i starts, and the variables in $\mathcal{U}$ are initialized to satisfy θ. The component processes may not start synchronously. Each edge (called timeout edge) is associated with an enabling condition or guard ρ, and a timeout increment rule. A guard ρ is expressed as a conjunction of the linear arithmetic constraints over variables in $\mathcal{U}$. The default condition on the guard is denoted as `true`. Note that if a location l_i^j has no outgoing edge then the process P_i has terminated. Also if each of the guards associated with the outgoing edges of location l_i^j is false, then the process P_i is deadlocked. Thus, a process P_i may remain stationary at some location l_i^j forever.

Timeout Edges: A timeout edge in the graph of the process P_i is represented as

$$l_j^i \xrightarrow[\mathcal{K}]{\rho \Rightarrow \langle \tau_i = update_i(\mathcal{K},\Theta), \Lambda, \bar{x} := \bar{f}(\bar{x}) \rangle} l_k^i,$$

where $update_i(\mathcal{K}, \Theta)$ assigns a value to timeout τ_i, which is to be updated on taking a transition on the edge when the guard ρ evaluates to true depending on the values of timeout increment constraint set $\mathcal{K}$, a subset of local timing variables $\Theta \subset \Lambda \cup \{t\}$. Elaborating, an edge in the graph is labeled by a timeout increment constraint set $\mathcal{K}$ of cardinality at most 2, such that if $l \in \mathcal{K}$ then $l \in \mathbb{N} \cup \{\infty\}$; these constants delimit the bounds for timeout increment. A subset of local timing variables $\Lambda \subseteq \mathcal{X}_i$ capture the value of the clock t while taking transition on the edge. This value may be used during future transitions while estimating the relative delay w.r.t. the transition taken. More specifically, $update_i : 2^{\mathbb{N}\cup\{\infty\}}_{\leq 2} \times 2^{\Lambda \cup \{t\}} \rightarrow \{\tau_i\}$ is partial function, where $2^{\mathbb{N}\cup\{\infty\}}_{\leq 2}$ is the set of subsets over $\mathbb{N} \cup \{\infty\}$ containing at most 2 elements. τ_i is assigned the value: $\tau_i = update_i(\mathcal{K}, \Theta)$, where $\Theta = \{z, z'\}$ or, $\{z\}$ or, $\emptyset$, and $\sim \in \{\leq, <\}$ such that,

$$\begin{cases} l + z \sim \tau_i \sim u + z' & \text{when} \quad \mathcal{K} = \{l, u\} \\ l + z \sim \tau_i < \infty & \text{when} \quad \mathcal{K} = \{l, \infty\} \\ \tau_i \sim u + z' & \text{when} \quad \mathcal{K} = \{u\} \\ \tau_i = \infty & \text{when} \quad \mathcal{K} = \{\infty\} \\ \tau_i = \max(\mathcal{M}) & \text{when} \quad \mathcal{K} \text{ is undefined} \end{cases}$$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Further, $z, z' := t|w|0$ where $w \in \mathcal{X}_i$ is a local timing variable. The variables z, z' make such a range of timeout values relative to the occurrence of specific events. The set $\mathcal{M}$ of integer constants is used to define the upper limit of different timeouts for different processes in the system. The value $\max(\mathcal{M})$ returns the maximum of all the integers in $\mathcal{M}$. In the assignment $\overline{x} := \bar{f}(\bar{x}), x \in Var$, $\bar{f}$ is a vector-valued function, where a component function can be an identity function or a simple computable function.

Moreover, the function $update_i$ specifies how the new value of timeout τ_i should be determined based upon the current value of the clock t and/or w, where w would have captured the value of clock t in some earlier transition. When a timeout is set to ∞ it indicates the capture of the requirement of indefinite waiting for an external signal/event. The selection of the timeout value using $\max(\mathcal{M})$ is used to capture the situation where the next discrete transition of a process may happen at any time in the future, for example, the process may be in a sleeping mode and can wake up at any future point of time. Note $update_i$ forces τ_i to satisfy a linear inequality over local timing variables. When such a linear inequality is clear from the context we shall use the interval notation for it. For example, if $update_i$ enforces $l + z \leq \tau_i \leq u + z'$ then we write $\tau_i \in [l + z, u + z']$, if it is the case $l + z \leq \tau_i < \infty$ then we write $\tau_i \in [l + z, \infty)$ and for $\tau_i < u + z'$ it is $\tau_i \in [0, u + z')$ etc.

To illustrate this, we take up the example of TTD (for Train) shown in Figure 2. Consider the edge $\mathbf{t_1} \xrightarrow{(\tau_{tr}=t) \Rightarrow \langle \tau_{tr} \in [t, w+5] \rangle} \mathbf{t_2}$, in which $\tau_{tr} \in [t, w + 5]$ specifying that when the process **Train** reaches state $\mathbf{t_2}$, the next transition to state $\mathbf{t_0}$ can occur (i.e., train may *exit* the 'gate') anytime before delay from an earlier transition from state $\mathbf{t_0}$ to state $\mathbf{t_1}$ (i.e., when train *approached* the 'gate') is not more than 5 time units.

In case of *synchronized communication* between a pair of processes the component processes would start on a handshake/rendezvous communication. A rendezvous communication between a pair of processes (P_s, P_r) is represented using an edge pair (E_s, E_r) s.t. $E_s \in P_s$ and $E_r \in P_r$ and

$$E_s : l_j^s \xrightarrow[\mathcal{K}_s]{\rho \Rightarrow \langle \alpha!m, \tau_s = update_s(\mathcal{K}_s, \Theta_s), \Lambda_s, \bar{x^s} := \bar{f}(\bar{x^s}) \rangle} l_k^s$$

$$E_r : l_j^r \xrightarrow[\mathcal{K}_r]{\text{True} \Rightarrow \langle \alpha?\tilde{m}, \tau_r = update_r(\mathcal{K}_r, \Theta_r), \Lambda_r, \bar{x^r} := \bar{g}(\bar{x^r}) \rangle} l_k^r$$

where α is the channel name, $m \in L_i$ is the message sent, and $\tilde{m} \in L_r$ the message received.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Semantics

With a given ToM P, we associate the following transition system $S_P = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$, referred to as *timeout based clocked transition system* (TCTS) where,

1. $\mathcal{V} = \mathcal{U} \cup \{\pi_1, \ldots, \pi_n\}$. Each *control variable* π_i ranges over the set $Loc_i \cup \{\perp\}$. The value of π_i indicates the location of the control for the process P_i and it is $\perp$ (undefined) before the start of the process.
2. Σ is the set of states. Every state $\sigma \in \Sigma$ provides an interpretation of $\mathcal{V}$, that is, it assigns values to all the variables in $\mathcal{V}$. For $x \in \mathcal{V}$, let $\sigma(x)$ denote its value in state σ.
3. $\Sigma_0 \subseteq \Sigma$ is the set of initial states such that for every $\sigma_0 \in \Sigma_0$, θ is true in σ_0 (i.e., variables in $\mathcal{U}$ are initialized to satisfy θ) and $\sigma_0(\pi_i) = \perp$ for each process P_i. Note that for each different initialization of variables in $\mathcal{U}$ satisfying θ, we have a different initial state in Σ_0.
4. $\Gamma = \Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm}$ is the set of transitions. Every transition $\nu \in \Gamma$ is a binary relation on Σ defined further as follows:

Entry Transition: Γ_e contains an *entry transition* ν_e^i for every process P_i. As noted before, all the variables in $\mathcal{U}$ are initialized to satisfy the data precondition $theta$. In particular, $\forall \sigma_0 \in \Sigma_0$,

$$\nu_e^i \equiv (\sigma_0, \sigma') \in \Gamma_e \Leftrightarrow \begin{cases} 1. \quad \forall x \in \mathcal{U}.\ \sigma'(x) = \sigma_0(x) \\ 2. \quad \forall \tau \in \mathcal{T}.\ \sigma'(t) \leq \sigma'(\tau) \\ 3. \quad \sigma_0(\pi_i) = \perp \text{ and } \sigma'(\pi_i) = l_0^i \end{cases}$$

Time Progress Transition: The first kind of edges $\nu_+ \in \Gamma_+$ are those where the global clock is increased to the minimum of all timeouts. In particular,

$$\nu_+^i \equiv (\sigma, \sigma') \in \Gamma_+ \Leftrightarrow \begin{cases} 1. \quad \sigma(t) < \min\{\sigma(\mathcal{T})\} \\ 2. \quad \forall \tau \in \mathcal{T}.\ \sigma'(\tau) = \sigma(\tau) \\ 3. \quad \forall x \in \mathcal{X} \cup var.\ \sigma'(x) = \sigma(x) \\ 4. \quad \forall i.\ \sigma'(\pi_i) = \sigma(\pi_i) \\ 5. \quad \sigma'(t) = \min\{\sigma(\mathcal{T})\} \end{cases}$$

Timeout Increment Transition: For the second kind of edges $\nu_0^i \in \Gamma_0$ the global clock equals the minimum of timeouts. Also if there is an edge in the TTD for process P_i as

Prepared using stvrauth.cls
DOI: 10.1002/stvr

$l^i_j \xrightarrow[\mathcal{K}]{\rho \Rightarrow \langle \tau_i = update_i(\mathcal{K},\Theta),\Lambda,\bar{x}:=\bar{f}(\bar{x})\rangle} l^i_k$, then

$$\nu^i_0 \equiv (\sigma,\sigma') \in \Gamma_0 \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \textbf{If } \sigma(\tau_i) = \sigma(t) \\ & \quad \textbf{then } (\sigma'(\tau_i) = update_i(\mathcal{K},\Theta)) \wedge (\sigma'(\tau_i) > \sigma(\tau_i)) \\ & \quad \textbf{else } \sigma'(\tau_i) = \sigma(\tau_i) \\ 4. & \forall x \in \Lambda.\ \sigma'(x) = \sigma(t) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \Lambda.\ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in Var.\ \sigma'(v) = f(\sigma(v)) \\ 6. & \sigma(\pi_i) = l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

A remark about how τ_i is updated using the above is in order. If $l + z \leq \tau_i \leq u + z'$, $update_i$ function non-deterministically selects a value δ such that $(l + \sigma(z) \leq \delta \leq u + \sigma(z')) \wedge (\delta > \sigma(\tau_i))$ and returns δ. Similarly for other cases. If $update_i = \infty$, $update_i$ returns the largest possible constant defined as per the design of the system. If $update_i = \max(\mathcal{M})$, $update_i$ non-deterministically selects any integer δ in $[0, M+1]$, where M is the maximum of all the integers in $\mathcal{M}$ returned by $\max(\mathcal{M})$. The local timing variables in $\Lambda \subseteq \mathcal{X}_i$ for process P_i are assigned the current value of global clock on timeout increment transition, while the other local timing variables in the system retain their old values before this transition. The variables in Λ are thus used to capture the delay between events. The variables in Var either remain unchanged or are assigned a new value according to the computable function f.

Synchronous Transition: For a pair of processes P_s, P_r having synchronous communication edges (E_s, E_r) we need to define an additional synchronous communication transition $\nu^{sr}_{syn\_comm} \in$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Γ_{syn_comm} as

$$\nu^{sr}_{syn\_comm} \equiv (\sigma, \sigma') \in \Gamma_{syn\_comm} \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \sigma'(\tau_s) = update_s(\mathcal{K}_s, \Theta_s) > \sigma(\tau_s) \text{ and} \\ & \sigma'(\tau_r) = update_r(\mathcal{K}_r, \Theta_r) > \sigma(\tau_r) \\ 4. & \forall x \in (\Lambda_s \cup \Lambda_r).\ \sigma'(x) = \sigma(t) \text{ and} \\ & \forall x \in \mathcal{X} \setminus (\Lambda_s \cup \Lambda_r).\ \sigma'(x) = \sigma(x) \\ 5. & \sigma'(\tilde{m}) = \sigma(m) \\ 6. & \forall v_i \in Var_s.\ \sigma'(v_i) = f(\sigma(v_i)) \\ & \forall v_j \in Var_r.\ \sigma'(v_j) = g(\sigma(v_j)) \\ 7. & \sigma(\pi_s) = l^s_j, \sigma(\pi_r) = l^r_j \text{ and} \\ & \sigma'(\pi_s) = l^s_k, \sigma'(\pi_r) = l^r_k \end{cases}$$

The semantic model defines the set of possible computations of the ToM P as a (possibly infinite) set of state sequences $\xi : \sigma_0 \rightarrow \sigma_1 \rightarrow \ldots$, which starts with some initial state σ_0 in Σ_0 and follows with successive transitions in Γ, *i.e.*, $\forall i.(\sigma_i, \sigma_{i+1}) \in \Gamma$. Let $[\![S_P]\!]$ be the set of all these computations of a ToM P as defined by its TCTS S_P.

The Train-Gate Controller (TGC) can be modeled as timeout-based model with synchronous transition which can associated with the semantics of a TTS, as depicted in the right hand side of Figure 2).

2.2. Calendar Based Timed Transition Model

Syntax

Next we capture bounded message transfer delay associated with an asynchronous communication. Towards that the ToM is extended with a calendar data structure. A calendar is a linear list of bounded size, where each element of the list contains the following information, message, sender_id, receiver_id, and expected_delivery_time. Assuming $\mathcal{C}$ to denote the calendar array, a

Prepared using stvrauth.cls
DOI: 10.1002/stvr

globally shared object, we set

$$\mathcal{U} = \{t\} \cup \mathcal{T} \cup \mathcal{X} \cup \mathit{Var} \cup \mathcal{C}$$

Sending a message in a TTD by process P_i is represented using the following edge:

$$l^i_j \xrightarrow[\mathcal{K}_i]{\rho \Rightarrow \langle send(m,i,\Omega), \tau_i = update_i(\mathcal{K}_i, \Theta_i), \Lambda_i, \bar{x^i} := \bar{f}(\bar{x^i})\rangle} l^i_k,$$

where $\Omega \subseteq R \times \Lambda$, $R \subseteq \{1, 2, \ldots n\}$ is the index set for the processes and Λ is the set of expected message delays. $send(m, i, \{(r, \lambda_r), \ldots\})$ specifies that a message m is to be sent by process P_i to each of the processes P_r with the expected delivery time of λ_r. On taking a transition on this edge an entry $\{$ m, i, r, $\lambda_r\}$ is added to $\mathcal{C}$ for each $(r, \lambda_r) \in \Omega$.

The reception of the corresponding message is represented in the TTD for each of the processes $P_r, r \in R$ using the following edge:

$$l^r_j \xrightarrow[\mathcal{K}_r]{\texttt{true} \Rightarrow \langle receive(m,i,r), \tau_r = update_r(\mathcal{K}_r, \Theta_r), \Lambda_r, \bar{x^r} := \bar{g}(\bar{x}^r)\rangle} l^r_k,$$

where $receive(m, i, r)$ specifies that a message m sent by the process P_i is to be received by the process P_r. On taking a transition on this edge, the entry $\{$m, i, r, $\lambda_r\}$ is deleted from $\mathcal{C}$.

Semantics

Given a calendar $\mathcal{C}$, we assume that the set of delays for all undelivered messages at any state σ can be found using the function $\Delta : \sigma(\mathcal{C}) \rightarrow \mathbb{N}$, such a value can be found from the design of the system. Further, we can associate *Calendar-based Clocked Transition System* (CCTS) with a given transition system with a calendar.

The set of transitions in the (CCTS) is $\Gamma = \Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm} \cup \Gamma_{asyn\_comm}$. Both Γ_e (set of Entry Transitions) and Γ_{syn_comm} (set of Synchronous Communications) are same as in TCTS defined earlier. The definitions for the edges in Time Progress Transition (Γ_+) and those for Timeout Increment Transition (Γ_0) are modified using calendar $\mathcal{C}$ as follows:

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Time Progress Transition: $\nu_+ \equiv (\sigma, \sigma') \in \Gamma_+$

$$\Leftrightarrow \begin{cases} 1. & \sigma(t) < \min\{\sigma(\mathcal{T}) \cup \Delta(\sigma(\mathcal{C}))\} \\ 2. & \forall \tau \in \mathcal{T}.\ \sigma'(\tau) = \sigma(\tau) \\ 3. & \forall x \in \mathcal{X} \cup \mathit{Var}.\ \sigma'(x) = \sigma(x) \\ 4. & \forall i.\ \sigma'(\pi_i) = \sigma(\pi_i) \\ 5. & \sigma'(t) = \min\{\sigma(\mathcal{T}) \cup \Delta(\sigma(\mathcal{C}))\} \end{cases}$$

Timeout Increment Transition: If there is a timeout edge in the TTD of process P_i as $l^i_j \xrightarrow[\mathcal{K}]{\rho \Rightarrow \langle \tau_i = update_i(\mathcal{K},\Theta),\Lambda,\bar{x}:=\bar{f}(\bar{x})\rangle} l^i_k$, , then

$$\nu^i_0 \equiv (\sigma, \sigma') \in \Gamma_0 \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \textbf{If } [\sigma(t) = \min\{\sigma(\mathcal{T})\}] \wedge [\sigma(\tau_i) = \sigma(t)] \\ & \quad \textbf{then } \sigma'(\tau_i) = update_i > \sigma(\tau_i) \\ & \quad \textbf{else } \sigma'(\tau_i) = \sigma(\tau_i) \\ 4. & \forall x \in \Lambda.\ \sigma'(x) = \sigma(t) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \Lambda.\ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in \mathit{Var}.\ \sigma'(v) = f(\sigma(v)) \\ 6. & \sigma(\pi_i) = l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

We additionally define new transitions corresponding to $send()$ and $receive()$ to capture asynchronous communication.

Send Transition: If there is a send edge in process P_i, we have the corresponding edge $\nu^i_{send} \in$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Γ_{asyn_comm}, that adds $|\Omega|$ cells to the calendar array $\mathcal{C}$:

$$\nu^i_{send} \equiv (\sigma, \sigma') \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \sigma'(\tau_i) = update_i(\mathcal{K}_i, \Theta_i) > \sigma(\tau_i) \\ 4. & \forall x \in \Lambda_i.\ \sigma'(x) = \sigma(t) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \Lambda_i.\ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_i.\ \sigma'(v) = f(\sigma(v)) \text{ where } f \text{ is not an identity function and} \\ & \forall v \in Var \setminus (G \cup L_i).\ \sigma'(v) = \sigma(v) \\ 6. & \text{for each tuple}(r, \lambda_r) \in \Omega.\ \sigma'(\mathcal{C}) := \sigma(\mathcal{C}) \cup \{\mathsf{m}, \mathsf{i}, \mathsf{r}, \lambda_r\} \\ 7. & \sigma(\pi_i) = l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

Receive Transition: If there is a receive edge in process P_r, we have the corresponding edge $\nu^r_{receive} \in \Gamma_{asyn\_comm}$, which deletes the entry $\{m, i, r, \lambda_r\}$ from the calendar array $\mathcal{C}$ when the clock t reaches λ_r:

$$\nu^r_{receive} \equiv (\sigma, \sigma') \Leftrightarrow \begin{cases} 1. & \text{there exists} \{m, i, r, \lambda_r\} \in \sigma(\mathcal{C}) \text{ s.t. } \sigma(t) = \lambda_r \\ 2. & \sigma'(t) = \sigma(t) \\ 3. & \sigma'(\tau_r) = update_r(\mathcal{K}_r, \Theta_r) > \sigma(\tau_r) \\ 4. & \forall x \in \Lambda_r.\ \sigma'(x) = \sigma(t) \text{ and} \\ & \forall x \in \mathcal{X} \setminus \Lambda_r.\ \sigma'(x) = \sigma(x) \\ 5. & \forall v \in G \cup L_r.\ \sigma'(v) = g(\sigma(v)) \text{ where } g \text{ is not an identity function and} \\ & \forall v \in Var \setminus (G \cup L_r).\ \sigma'(v) = \sigma(v) \\ 6. & \sigma'(\mathcal{C}) := \sigma(\mathcal{C}) \setminus \{m, i, r, \lambda_r\} \\ 7. & \sigma(\pi_r) = l^r_j \text{ and } \sigma'(\pi_r) = l^r_k \end{cases}$$

Similar to the case of TCTS, this semantic model also defines the set of possible computations of the calendar based ToM as a (possibly infinite) set of state sequences starting with some initial state in Σ_0 and following with successive transitions in Γ. We denote $[\![S_P]\!]$ to be the set of all these computations of a calendar based ToM P as defined by its CCTS S_P.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

There are two natural choices for time as the underlying model for clock, the set of non-negative integers $\mathbb{N}$ (discrete time) or the set of non-negative reals $\mathbb{R}$ (dense time). Given the model of time as $TIME$, let $[S_P]_{TIME}$ be the set of all the computations of a ToM (or calendar based ToM) P as defined by its TCTS (or CCTS) S_P. When we consider the underlying model of time as $\mathbb{R}$, we need to add the following non-zenoness condition to ensure time progress in the model: *there must not be infinitely many time progress (or timeout increment) transitions effective within a finite interval.*

3. VERIFICATION RESULTS FOR DIGITIZATION

Following [19], a real time system S can be defined as a tuple $(\Sigma, \mathcal{P}, \mu, \mathcal{H})$, where Σ is a set of states, $\mathcal{P}$ is a set of observable events or observable propositions in Σ, $\mu : \Sigma \to \mathcal{P}$ is a labeling function and H is a set of fusion-closed time state sequences which also satisfies finite variability conditions.

For a given implementation I and specification S, let the implementation language $\mathcal{L}_I$ describe systems and behavior over time while the specification language $\mathcal{L}_S$ describes the timing requirements of the system. Assuming C and T to be mathematical models of computation and time respectively, the real-time verification problem parameterized by $(C, T, \mathcal{L}_I, \mathcal{L}_S)$ is stated as follows: does the implementation of the system I, given as an expression of $\mathcal{L}_I$ meet the specification ϕ given as an expression of $\mathcal{L}_S$, with respect to the semantic assumption (C, T), written as $I \models^{?}_{(C,T)} \phi$ or, equivalently $\mathcal{L}_I \underset{(C,T)}{\subseteq^{?}} \mathcal{L}_S$. In particular, we would consider two important instances of the real-time verification problem - one with a *dense* model of time and another one with an *integral* (discrete) model of time. In the following, we assume TTS as the implementation language and linear time temporal logic (LTL) as the specification formalism.

3.1. Timed Sequences

We shall adopt discrete trace model (using the terminology from [9, 20]) as a mathematical model of computation. By *discrete trace* model one can capture the behavior of a system as an infinite sequence of snapshots of the global system state at certain times. We assume our time domain $TIME$ has a total ordering $\leq$ defined on it. We define an *observation* to be a pair (σ_i, T_i), where σ_i

Prepared using stvrauth.cls
DOI: 10.1002/stvr

is a state and $T_i \in TIME$ is a time stamp. A *timed state sequence* (TSS) $\eta = (\sigma, T)$ is an infinite sequence $\eta : \ (\sigma_0, T_0) \rightarrow (\sigma_1, T_1) \rightarrow (\sigma_2, T_2) \rightarrow \cdots$ of observations[§]. Further, the infinite sequence $T_i \in T$ of time stamps in η satisfy (i) *monotonicity*: time does not decrease, $T_i \leq T_{i+1}$ for all $i \geq 0$, and (ii) *progress:* time progresses, for all $T \in TIME$, $T_i \geq T$ for some $i \geq 0$.

Now onwards, we shall work with dense-time models when $TIME = \mathbb{R}$[¶] and integral-time models when $TIME = \mathbb{N}$[‖]. A TSS under dense-time model will be referred to as *precisely timed* and under integral-time model as *digitally timed*.

Let us denote the set of all TSSs over the $TIME$ domain as TSS_{TIME}. A *real-time* property is a subset of TSS_{TIME}. Every real-time system S defines a real-time property, denoted as $[\![S]\!]$, which is the set of all TSSs of S. Also, every real-time specification ϕ defines a real-time property $[\![\phi]\!]$, the set of real-time sequences that satisfy ϕ.

Now we formulate the real-time verification problem. We say a real-time system S satisfies the specification ϕ, written as

$$S \models_{TIME} \phi \quad \text{ if and only if } \quad [\![S]\!]_{TIME} \subseteq [\![\phi]\!]_{TIME}$$

Consider a dense-time property $\Pi_{\mathbb{R}} \subseteq TSS_{\mathbb{R}}$, a set of of TSSs over $\mathbb{R}$. Its *clock-independent semantics* $\mathbb{N}(\Pi_{\mathbb{R}})$ is the subset of digitally TSSs in $\Pi_{\mathbb{R}}$, *i.e.,* $\mathbb{N}(\Pi_{\mathbb{R}}) = \Pi_{\mathbb{R}} \cap TSS_{\mathbb{N}}$. In [9], it is shown that clock-independent semantics is not very adequate for reasoning about dense time. As a remedy of this, another approximate semantics was introduced, which was called *digitization*.

For any TSS $\eta = (\sigma, T)$, we introduce it untime operation η^- as its state component σ. Also, $\eta^i = (\sigma^i, T^i)$, for $i \geq 0$, denotes the TSS that results from η by deleting the first i observations (note, $\eta^0 = \eta$).

[§]Note that any $\xi \in [\![S_P]\!]$ (previously defined) essentially defines a TSS. This is because, states in ξ have implicit representation for time stamps as $\sigma_0(t), \sigma_1(t), \ldots$, which are otherwise explicitly present in the definition of η as $T_0, T_1, \ldots$

[¶]the set of reals

[‖]the set of natural numbers

Prepared using stvrauth.cls
DOI: 10.1002/stvr

3.2. Digitization

Given $x \in \mathbb{R}$ and $\epsilon \in (0,1]$, we define $[x]_\epsilon = \lfloor x \rfloor$ if $x \leq \lfloor x \rfloor + \epsilon$, otherwise $[x]_\epsilon = \lceil x \rceil$**. Given a precisely timed sequence $\eta = (\sigma, T)$ and $\epsilon \in (0,1]$, we define the *ϵ-digitization* $[\eta]_\epsilon = (\sigma, [T]_\epsilon)$ of η be the digitally timed sequence $(\sigma_0, [T_0]_\epsilon) \rightarrow (\sigma_1, [T_1]_\epsilon) \rightarrow \cdots$

For any dense-time property Π (a set of timed sequences over dense time) let $[\Pi] = \{[\eta]_\epsilon \mid \eta \in \Pi \text{ and } \epsilon \in (0,1]\}$, be a digitization of Π. We write $[\eta]$ instead of $[\{\eta\}]$.

We state some concepts from [9]. Let Π be a dense-time property. Π is *closed under digitization* iff for all $\eta \in TSS_{\mathbb{R}}, \eta \in \Pi$ implies $[\eta] \subseteq \Pi$. Π is *closed under inverse digitization* iff $[\eta] \subseteq \Pi$ implies $\eta \in \Pi$, for all $\eta \in TSS_{\mathbb{R}}$. Finally, Π is *digitizable* iff it is closed under both digitization and inverse digitization, *i.e.,* $\eta \in \Pi$ iff $[\eta] \subseteq \Pi$ for all $\eta \in TSS_{\mathbb{R}}$. We quote the following important results from [9].

Fact 3.1

Assume a real-time system S whose dense-time semantics $[\![S]\!]_{\mathbb{R}}$ is closed under digitization, and a specification ϕ whose dense-time semantics $\phi_{\mathbb{R}}$ is closed under inverse digitization. Then the following is true: $S \models_{\mathbb{R}} \phi$ if and only if $S \models_{\mathbb{N}} \phi$.

A dense-time property Π is said to be *qualitative* if $\eta \in \Pi$ implies $\eta' \in \Pi$ for all precisely timed sequences η and η' with identical state components (*i.e.,* $\eta^- = \eta'^-$).

Fact 3.2

A qualitative property is digitizable.

Digitization of Timeout and Calendar based Transition Systems: Recall that a TCTS is $S = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$ (assume the ToM P to be implicit). We show that the computations for this transition system are digitizable under certain conditions.

A *run* of S over a TSS $\eta : (\sigma_0, T_0) \rightarrow (\sigma_1, T_1) \rightarrow \cdots$ is a sequence of pairs of S of the form $\zeta : (\sigma_0, \nu_0) \stackrel{T_1}{\rightarrow} (\sigma_1, \nu_1) \stackrel{T_2}{\rightarrow} \cdots$ where σ_i denotes the state and ν_i the mapping of variables from $\mathcal{U}$ in state σ_i to appropriate domains and further, it satisfies the following conditions:

** where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the floor and ceiling rounding operations on real numbers respectively

Prepared using stvrauth.cls
DOI: 10.1002/stvr

1. (initiation:) $\sigma_0 \in \Sigma_0$ and $\nu_0(t) = T_0, \ \ \forall i \geq 0. \nu_0(\pi_i) = \perp, \ \ t \in \mathcal{V}, \pi_i \in \mathcal{V}$.
2. (consecution:) for $i \geq 1$ there is an edge $(\sigma_{i-1}, \sigma_i) \in \Gamma (= \Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm})$ such that the following hold:
   - if $(\sigma_0, \sigma_1) \in \Gamma_e$ then $T_0 = \nu_0(t) \leq \nu_1(t) = T_1$ and $\forall \tau \in \mathcal{T}. \sigma_1(\tau) \geq T_1$.
   - if $(\sigma_{i-1}, \sigma_i) \in \Gamma_+$ then $T_{i-1} = \nu_{i-1}(t) < \min\{\sigma_{i-1}(\mathcal{T})\} = \nu_i(t) = T_i$.
   - if $(\sigma_{i-1}, \sigma_i) \in \Gamma_0$ then $T_{i-1} = \nu_{i-1}(t) = \nu_i(t) = T_i$.
   - if $(\sigma_{i-1}, \sigma_i) \in \Gamma_{syn\_comm}$ then $T_{i-1} = \nu_{i-1}(t) = \nu_i(t) = T_i$.
3. (time progress:) for any real number T there exists an $i \geq 0$ such that $T_i > T$.

We say that $\eta \in TSS_{\text{TIME}}$ is *time-consistent* (for S) if S has a run over it. In the sequel we consider only time-consistent behaviors $\eta \in [S]_{\text{TIME}}$ of S, *i.e.,* $\eta \in [S]_{\text{TIME}}$ iff there is a run over η. If $\text{TIME} = \mathbb{N}$ then we get integral behavior of TCTS. Now it is obvious that time at state $j \geq 1$ in a given run, is given by $\nu_j(t) = T_j$. We denote ϵ-digitization of the mapping ν_j as $\langle \nu_j(x) \rangle_\epsilon = [\nu(x)]_\epsilon$ for any variable $x \in \mathcal{U} \subseteq \mathcal{V}$.

Given a computation $\zeta : (\sigma_0, \nu_0) \xrightarrow{T_1} (\sigma_1, \nu_1) \xrightarrow{T_2} \cdots$ its ϵ-digitization is the computation $[\zeta]_\epsilon : (\sigma_0, \langle \nu_0 \rangle)_\epsilon) \xrightarrow{[T_1]_\epsilon} (\sigma_1, \langle \nu_1 \rangle)_\epsilon \xrightarrow{[T_2]_\epsilon} \cdots$, where $\langle \nu_j \rangle_\epsilon$ for $j \geq 1$ are defined above, and $\langle \nu_0(t) \rangle_\epsilon = [T_0]_\epsilon$.

Now we need to analyze the extent to which the set of dense-time computations of a TCTS are closed under digitization. We prove the following result:

Theorem 3.3

The set of dense-time computations of a TCTS are closed under digitization if and only if all timeout increments are at least 1 time unit.

Proof

Suppose $\zeta : (\sigma_0, \nu_0) \xrightarrow{T_1} (\sigma_1, \nu_1) \xrightarrow{T_2} \cdots$ is a run of S over η. For digitization purpose, $[\zeta]_\epsilon$ would be a run of S over $[\eta]_\epsilon$. We have $\langle \nu_0(t) \rangle_\epsilon = [T_0]_\epsilon$. Observe if $T_{i-1} = T_i$ then $[T_{i-1}]_\epsilon = [T_i]_\epsilon$. When $T_{i-1} < T_i$, except for the case of $0 < (T_i - T_{i-1}) < 1$, we have $[T_{i-1}]_\epsilon < [T_i]_\epsilon$. So, if there is an edge $(\sigma_{i-1}, \sigma_i) \in \Gamma$ and $(T_i = T_{i-1}) \vee (T_i \geq T_{i-1} + 1)$, there would be an edge $(\langle \sigma_{i-1} \rangle_\epsilon, \langle \sigma_i \rangle_\epsilon)$ in Γ under $[\zeta]_\epsilon$. Also we can ensure time progress for $[\zeta]_\epsilon$. □

Prepared using stvrauth.cls
DOI: 10.1002/stvr

The result above indicates a precise characterization for the digitization for a TCTS. All timeout increments in an interval of $(0, 1)$ time units result into a TCTS, which are not closed under digitization and therefore cannot be model checked for all LTL properties under discrete time dynamics. This is a technical condition, however it is fair to assume that timeout increments will lie outside the interval $(0, 1)$ in most of the cases. For example, when $u + z \leq \tau_i$, any integral value of u will ensure that $\tau_i \geq 1$. When $\tau_i = \max(\mathcal{M})$, in all likelihood $\max(\mathcal{M})$ will return an integral value. When $l + z \leq \tau_i \leq m + z'$, for $0 < \tau_i < 1$ to be true it has to be the case $l + z > 0$ and $m + z' < 1$ which imply $l > -z, m < 1 - z'$. Such a value of m can be ruled out.

A similar argument can be used to show that the dense computations of a (digitizable) CCTS are also closed under digitization.

3.3. Digitizability of Linear Temporal Logic

The formulas of LTL [21] are commonly interpreted over untimed state sequences. However we can extend their usual interpretation to TSSs as well. Let $\eta = (\sigma, T)$ be a TSS with $\sigma_i \in \Sigma$ for $i \geq 0$. The satisfaction relation $\eta \models \phi$ is defined by extending the one over untimed models [21] inductively, for example:

$$\begin{aligned} \eta \models \bigcirc\phi \quad & \text{iff} \quad \eta^1 \models \phi \text{ and } T_1 \geq T_0, \\ \eta \models \phi_1 \mathcal{U} \phi_2 \quad & \text{iff} \quad \exists i \geq 0. \exists \alpha \in \mathbb{N}. \eta^i \models \phi_2, \text{ where} \\ & \qquad T_i \geq T_0 + \alpha, \text{ and } \forall j. 0 \leq j < i. \eta^j \models \phi_1. \end{aligned}$$

For a LTL-formula ϕ, let the set $[\phi]_{\mathbb{R}} \subseteq TSS_{\mathbb{R}}$ contain all TSSs η over the time domain $\mathbb{R}$ such that $\eta \models \phi$.

Theorem 3.4

For any specification ϕ expressed in LTL, $[\phi]_{\mathbb{R}}$ is closed under inverse digitization.

Proof

To see this, consider two timed sequences η and η' with identical state components. Suppose $\eta \models \phi$, *i.e.*, $\eta \in [\phi]_{\mathbb{R}}$. Now the proof is by induction on the structure of ϕ. At the induction stage, we only consider the case $\phi = \phi_1 \mathcal{U} \phi_2$. Now $\eta \models \phi_1 \mathcal{U} \phi_2$ iff for some $i \geq 0, \alpha \in \mathbb{N}, \eta^i \models \phi_2,$ where $T_i \geq$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

$T_0 + \alpha,$ and $\eta^j \models \phi_1$ for all $0 \leq j < i$. By induction hypothesis, we have $\eta'^i \models \phi_2$ and $\eta'^j \models \phi_1$. Since, $T'_i \geq T'_0$, there exists some $\alpha' \in \mathbb{N}$ such that $T'_i \geq T'_0 + \alpha'$. Therefore $\eta' \models \phi$ and hence $\eta' \in [\phi]_{\mathbb{R}}$. □

An Integral Verification Problem: Given a TCTS or CCTS S, corresponding to a timeout-based or a calender-based model and a specification formula ϕ in LTL we can check $S \models_{\mathbb{R}} \phi$ by verifying whether $S \models_{\mathbb{N}} \phi$. In the next section we will further simplify this problem.

4. CLOCK-LESS MODELING

In this section we propose a finitary reduction technique, which is formalized in terms of clock-less modeling and semantics. This technique effectively reduces the timeout and calendar based transition systems with discrete dynamics into finite state systems, which, in turn, can be specified and model checked by finite state model checkers. The assumption of discrete time as the underlying model is particularly relevant to cases where we are left with integral verification problem exploiting digitization results.

For presenting clock-less modelling and semantics, we will only consider timeout based models having synchronous communication edges for illustration purpose. The case of calendar based modeling will be a similar exercise.

4.1. Timeout based Models: clock-less Modeling

clock-less Syntax

In order to capture the effect of finite state reduction in a timeout model, we restrict the set $\mathcal{U}$ by dropping the global clock variable t, *i.e.*, $\mathcal{U} = \mathcal{T} \cup \mathcal{X} \cup Var$, timeout is updated as $\tau_i =$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

$update_i(\mathcal{K}, \Theta)$ whence $\mathcal{K}$ is a timeout increment constraint set and $\Theta = \{z, z'\}$:

$$\begin{cases} l - z \sim \tau_i \sim u - z' & \text{when} \quad \mathcal{K} = \{l, u\} \\ l - z \sim \tau_i < \infty & \text{when} \quad \mathcal{K} = \{l, \infty\} \\ \tau_i \sim u - z' & \text{when} \quad \mathcal{K} = \{u\} \\ \tau_i = \infty & \text{when} \quad \mathcal{K} = \{\infty\} \\ \tau_i = \max(\mathcal{M}) & \text{when} \quad \mathcal{K} \text{ is undefined} \end{cases}$$

where $z, z' := w|0$, $m \in \mathbb{N} \cup \{\infty\}$ and $w \in \Xi_i$ is a local variable. For any $z \in \mathcal{U}$ let $\sigma_i^-(z)$ stand for the value of the variable z in (clock-less) state σ_i^-.

clock-less Semantics

With clock-less modeling of timeout based models we associate a transition system $S_P^- = (\mathcal{V}^-, \Sigma^-, \Sigma_0^-, \Gamma^-)$, where $\mathcal{V}^- = \mathcal{V} \setminus \{t\}$, Σ^- a set of clock-less states, $\Sigma_0^- \subseteq \Sigma^-$ a set of initial clock-less states (defined in an analogous manner as for clocked transition systems) and Γ^- a set of clock-less transitions. We remark that given a timeout based model, the set of states Σ for clocked transition system and the set of states Σ^- for clock-less transition system are exactly identical modulo the assignment of the global clock variable t. Let $\Gamma^- = \Gamma_e^- \cup \Gamma_+^- \cup \Gamma_0^- \cup \Gamma_{syn\_comm}^-$. While Γ_e^- is same as Γ_e, we will redefine Γ_+^-, and Γ_0^- while $\Gamma_{syn\_comm}^-$ can be modified in the same manner for the TCTS.

Time Progress Transition: The edges ν_+ are redefined such that all the timeout are decremented by the minimum of the current timeout values. In particular,

$$\nu_+ \equiv (\sigma^-, \sigma'^-) \in \Gamma_+^- \Leftrightarrow \begin{cases} 1. \quad \min\{\sigma^-(\mathcal{T})\} > 0 \\ 2. \quad \forall \tau \in \mathcal{T}.\ \sigma'^-(\tau) = \sigma^-(\tau) - \min\{\sigma^-(\mathcal{T})\} \\ 3. \quad \forall x \in \mathcal{X} \cup \mathit{Var}.\ \sigma'^-(x) = \sigma^-(x) \\ 4. \quad \forall i.\ \sigma'^-(\pi_i) = \ \sigma^-(\pi_i) \end{cases}$$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Timeout Increment Transition: For the edges ν_0^i, if there is an edge in the TTD of process P_i as $l_j^i \xrightarrow[\mathcal{K}]{\rho \Rightarrow \langle \tau_i = update_i(\mathcal{K},\Theta),\Lambda,\bar{x}:=\bar{f}(\bar{x})\rangle} l_k^i$, , then

$$\nu_0^i \equiv (\sigma,\sigma') \in \Gamma_0 \Leftrightarrow \begin{cases} 1. & \rho \text{ holds in } \sigma^- \\ 2. & \textbf{If } \sigma^-(\tau_i) = 0 \textbf{ then} \\ & \sigma'^-(\tau_i) = update_i^-(\mathcal{K},\Theta) > 0 \textbf{ else } \sigma'^-(\tau_i) = \sigma^-(\tau_i) \\ 3. & \forall x \in \Lambda.\ \sigma'^-(x) = \sigma'^-(\tau_i) + \sigma^-(x) \text{and} \\ & \forall x \in \mathcal{X} \setminus \Lambda.\ \sigma'^-(x) = \sigma^-(x) \\ 4. & \forall v \in Var. \sigma'^-(v) = f(\sigma^-(v)) \\ 5. & \sigma^-(\pi_i) = l_j^i \text{ and } \sigma'^-(\pi_i) = l_k^i \end{cases}$$

The function $update_i^-$ is a slight modification of $update_i$, for example, if $l - z \leq \tau_i \leq u - z'$ then $update_i^-$ non-deterministically selects a value δ such that $l - \sigma^-(z) \leq \delta \leq u - \sigma^-(z')$. Note that, unlike the local timing variables appearing in Λ in a (clocked) ToM, these timing variables incrementally capture the value of next timeout in a clock-less ToM. An observant reader can see that the relative delay captured by these local timing variables between events are same in both those models. A clocklss model of the Train-Gate Controller (TGC) is shown in Figure 4, notice that is model does not any clock variable.

4.2. LTL formulas for clock-less Models

The formulas will be built using finitely many atomic propositions (constraints), which may be defined in terms of variables in $\mathcal{U}$ (as defined for clock-less timeout and calender models before). A model for a LTL formula would consist of a sequence of states of the form $\sigma_0, \sigma_1, \cdots$, such that each state σ_i gives a boolean interpretation to the propositions, and non-negative integer valued interpretation to the timeout variables in $\mathcal{T}$, timing variables in $\mathcal{X}$, and state variables in Var, all of which are bounded above by some positive integer constant. In a state σ_i, let us assume $\sigma_i(v)$ to be the value of $v \in \mathcal{U}$. Based on this LTL formulas can be appropriately interpreted on each state.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

In terms of these LTL formulas, using clock-less ToM, one can essentially verify the qualitative properties of the associated real-time system, which are otherwise difficult to do using the clocked ToM models. This is because clock-less models preserve the qualitative behavior of the clocked models. As the valuations of the variables in the clock-less models are bounded, the clock-less models effectively give rise to finite state behaviors.

4.3. Clock-less Models simulate Clock Models

In this section we will show that clock-less models simulate clock models with respect to LTL formulas. Let us consider a ToM P and its TCTS $S_P = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$ and also the clock-less ToM P^- and corresponding timeout based clock-less TCTS $S_{P^-} = (\mathcal{V}^-, \Sigma^-, \Sigma_0^-, \Gamma^-)$; both of them modeling the same system. Given a computation $\xi : \sigma_0 \rightarrow \sigma_1 \rightarrow \cdots$ over S_P let us generate a clock-less computation as a sequence of states $\sigma_0^-, \sigma_1^- \cdots$ over S_{P^-} as follows:

- Initial states correspond:

$$\begin{cases} \forall \tau \in \mathcal{T}.\ \sigma_0^-(\tau) = \sigma_0(\tau), \\ \forall x \in \mathcal{X}.\ \sigma_0^-(x) = \sigma_0(x), \\ \sigma_0^-(\pi_i) = \sigma_0(\pi_i) = \perp\ . \end{cases}$$

- Entry transition: if $(\sigma_0, \sigma_1) \in \Gamma_e$:

$$\begin{cases} 1. \quad \forall \tau \in \mathcal{T}. \sigma_1^-(\tau) = \sigma_1(\tau), \\ 2. \quad \forall x \in \mathcal{X}. \sigma_1^-(x) = \sigma_1(x), \\ 3. \quad \sigma_1^-(\pi_i) = \sigma_1(\pi_i) = l_0^i. \end{cases}$$

- Time progress transition: if $(\sigma_{i-1}, \sigma_i) \in \Gamma_+$:

$$\begin{cases} 1. \quad \forall \tau \in \mathcal{T}. \sigma_i^-(\tau) = \sigma_i(\tau) - \min\{\sigma_{i-1}(\mathcal{T})\}, \\ 2. \quad \forall x \in \mathcal{X}. \sigma_i^-(x) = \sigma_i(x), \\ 3. \quad \forall i. \sigma_i^-(\pi_i) = \sigma_i(\pi_i). \end{cases}$$

Prepared using stvrauth.cls
DOI: 10.1002/stvr

- Timeout increment transition: if $(\sigma_{i-1}, \sigma_i) \in \Gamma_e$:

$$\begin{cases} 1. \quad \textbf{if } \sigma^-_{i-1}(\tau_i) = 0 \textbf{ then } \sigma^-_i(\tau_i) = update^-_i(\mathcal{K}, \Theta), \\ \qquad \textbf{else } \sigma^-_i(\tau_j) = \sigma_i(\tau_j), \\ 2. \quad \forall x \in \mathcal{X}.\ \sigma^-_i(x) = \sigma_i(\tau_i) + \sigma_{i-1}(x), \\ 3. \quad \forall i.\ \sigma^-_i(\pi_i) = \sigma_i(\pi_i). \end{cases}$$

  Where $update^-_i$ is defined in P^-.

- Synchronous communication: if $(\sigma_{i-1}, \sigma_i) \in \Gamma_{syn\_comm}$ then

$$\begin{cases} 1. \quad \sigma^-_i(\tau_s) = \sigma_i(\tau_s) \text{ and } \sigma^-_i(\tau_r) = \sigma_i(\tau_r), \\ 2. \quad \forall x \in \mathcal{X}.\ \sigma^-_i(x) = \sigma_i(x), \\ 3. \quad \sigma^-_i(\bar{m}) = \sigma_i(\bar{m}) \text{ and } \sigma^-_{i-1}(m) = \sigma_{i-1}(m), \\ 4. \quad \forall v \in G \cup L_s : \sigma^-_i(v) = \sigma_i(v), \\ \qquad \text{and } \forall v \in G \cup L_r . \sigma^-_i(v) = \sigma_i(v), \\ \qquad \forall v \in \mathit{Var} \setminus (G \cup L_s \cup L_r).\ \sigma^-_i(v) = \sigma_i(v), \\ 5. \quad \sigma^-_{i-1}(\pi_s) = \sigma_{i-1}(\pi_s) = l^s_j, \\ \qquad \sigma^-_{i-1}(\pi_r) = \sigma_{i-1}(\pi_r) = l^r_j \text{ and} \\ \qquad \sigma^-_i(\pi_s) = \sigma^-_i(\pi_s) = l^s_k, \\ \qquad \sigma^-_i(\pi_r) = \sigma_i(\pi_r) = l^r_k. \end{cases}$$

Check that $\sigma^-_0 \in \Sigma^-_0$ and for each i, $(\sigma^-_{i-1}, \sigma^-_i) \in \Gamma^-$. It is clear $\xi^- = \sigma^-_0 \rightarrow \sigma^-_1 \rightarrow \ldots$ forms a clock-less computation over S_{P^-}. We can associate a mapping $\mathrm{Tr} : \Sigma \times \Sigma \rightarrow \Sigma^-$ parameterized by an entry transition as follows. Fix two states, $\sigma_0 \in \Sigma_0, \sigma_1 \in \Sigma$, such that $(\sigma_0, \sigma_1) \in \Gamma_e$. Call $\gamma = (\sigma_0, \sigma_1)$. Then define $\mathrm{Tr}_\gamma(\sigma_0, \sigma_0) = \sigma^-_0, \mathrm{Tr}_\gamma(\sigma_i, \sigma_{i-1}) = \sigma^-_i$, $\forall i \geq 1$.

We say that computations $\xi : \sigma_0 \rightarrow \sigma_1 \rightarrow \ldots$ in S_P and $\xi^- : \sigma^-_0 \rightarrow \sigma^-_1 \rightarrow \ldots$ in S_{P^-} correspond if and only if there exists $\mathrm{Tr}_\gamma : \Sigma \times \Sigma \rightarrow \Sigma^-$ such that $\sigma^-_0 = \mathrm{Tr}_\gamma(\sigma_0, \sigma_0)$ and for every $i \geq 0$, $\sigma^-_i = \mathrm{Tr}_\gamma(\sigma_i, \sigma_{i-1})$, where $\gamma = (\sigma_0, \sigma_1)$. Let $\sigma \in \Sigma$ and $\sigma^- \in \Sigma^-$ be two states and there be a computation ζ in S_P which starts in σ. Then it is easy to see that there exists a corresponding computation ζ' in S_{P^-} beginning with σ^- [21].

Prepared using stvrauth.cls
DOI: 10.1002/stvr

We consider LTL formulas consisting of propositions and variables appearing in clock-less transition system of S_{P^-}. Assume $\sigma \in \Sigma$ and $\sigma^- \in \Sigma^-$ are two states such that $\mathrm{Tr}_\gamma(\sigma, \sigma') = \sigma^-$ for some $\sigma' \in \Sigma$ and some entry transition γ. Using the above information and by an induction on the structure of a LTL formula ϕ we can prove the following.

Theorem 4.1

$\sigma^- \models \phi$ implies $\sigma \models \phi$ (using the classical semantics of LTL formulas [21]). Hence, $S_{P^-} \models \phi$ implies $S_P \models \phi$.

That is to say, S_{P^-} simulates S_P [21]. Thus it is enough to verify properties on the clock-less transition system S_{P^-} instead of on S_P. Similar results can be established for CCTS also.

5. EXPERIMENTAL RESULTS

We measure the efficacy of our technique on three real-time systems: Fischer's mutual exclusion protocol, Train Gate Controller (TGC) and TTA startup protocol. Without loss of generality we assume the timeout increments of these protocols are more than one time unit. A timeout-based model of TGC is already produced in the right half of Figure 2, timeout and calendar-based models of other systems can be found in [22]. Following the steps in Figure 1 we apply the digitization technique on these models to generate discrete models, from which clock-less models are obtained through a finitary reduction proposed by us. We carry out our experiments on these clock-less models on a machine with an Intel(R) Pentium(R) M processor with 1.60 GHz speed and 1 GB RAM and running on Microsoft Windows 7.

[Example 1] Fischer's Mutual Exclusion Protocol

Fischer's protocol is designed to ensure mutual exclusion among real time concurrent processes. In a real time system assume there are n processes $P_1, \ldots, P_n$ trying to access shared resources in a real-time fashion. A process P_i is initially idle (*Sleeping* state), but at any time, may begin executing the protocol provided the value of a global variable $lock$ is 0 and then move to *Wait* state.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

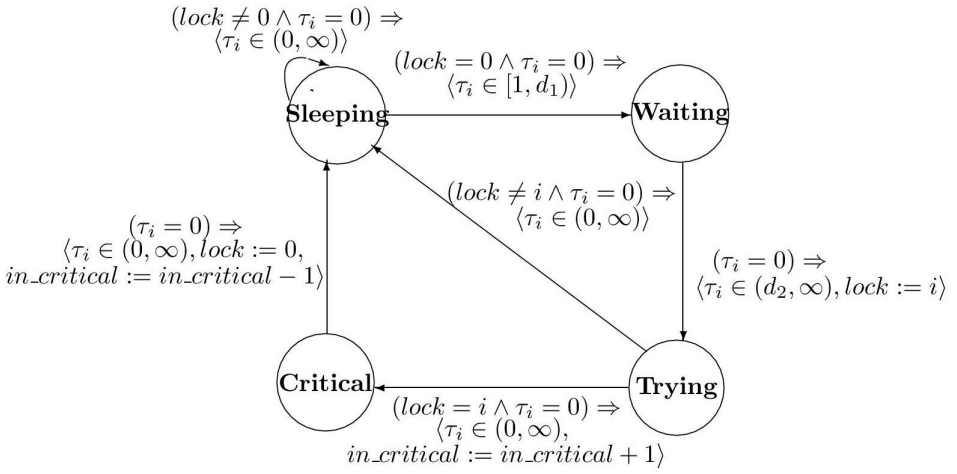

Figure 3: Clockless TTD To compare the for the i^{th} processor in the Fischer's Protocol

There it can wait up to maximum of $d_1 \geq 1$ time units before assigning the value i to $lock$ and moving to *Trying* state. It may enter the *Critical* section after a delay of at least of $d_2 \geq 1$ time units provided the value of $lock$ is still i. Otherwise it has to move to *Sleeping* state. Upon leaving the *Critical* section, it re-initializes $lock$ to 0. There is another global variable, $in\_critical$, used to keep count of the number of processes in the critical section. This variable is auto-incremented (auto-decremented) before a process enters the *Critical* section (leaves the *Critical* section). Mutual exclusion is ensured if $d_1 < d_2$. A clock-less timeout-based TTD of the i^{th} process P_i executing Fischer's protocol is shown in Figure 3.

We model Fischer's protocol in SAL language and verify the following mutual exclusion property:

```
mutual_exclusion: THEOREM
    system |- G(FORALL (i, j: IDENTITY):
    (pc[i] = critical AND pc[j] = critical) => i = j);
```

Table I presents the number of states and time required to prove the mutual exclusion property using SAL-smc. The Fisher's protocol has been verified under dense time in [4] for the same mutual exclusion property. Using bounded model checking the property was proved by k-induction at a

Prepared using stvrauth.cls
DOI: 10.1002/stvr

No of Nodes	States Explored	Total Time (s)	Nodes Nodes	States Explored	Total Time (s)
2	468	0.971	10	1.189e12	148.253
3	7968	1.492	11	1.697e13	282.225
4	124760	3.365	12	2.417e14	632.959
5	1.876e6	6.229	13	3.438e15	1344.082
6	2.760e7	13.860	14	4.885e16	1804.764
7	4.010e8	14.120	15	6.935e17	3932.504
8	5.786e9	45.796	16	9.839e18	7460.948
9	8.306e10	44.604	17	–	–

Table I: Verifying mutual-exclusion property for Fischer's protocol by SAL-smc

depth of 9 for 2 nodes. The same proof fails for 3 nodes. However, using a sequence of lemma it was possible to prove the property by induction at depth 1 for upto 20 nodes.

We also verified the mutual exclusion property for Fischer's protocol using Spin [12] where we used *exhaustive verification* and *bitstate hashing* technique available in the tool, in both the cases keeping the the option of *partial order reduction* turned on. By *exhaustive verification* technique, we could verify models containing only up to 4 nodes. By enabling *Bitstate hashing* we were able to verify the same property for models with upto 6 nodes. Table II describes the computational resources and time required to prove the mutual exclusion property for Fischer's protocol using *bitstate hashing* technique.

To compare the performance and scalability of our verification approach with UPPAAL, we verified Fischer's mutual exclusion protocol available with UPPAAL distribution. The modeling in UPPAAL is done using the framework of TA. The mutual exclusion property could be verified successfully for up to 12 nodes. For 13 nodes, the verification process did not stop even after 7 hours. For verification in UPPAAL, TAs are reduced to Zone Automata which are finite representations of infinite state systems. Although both our clock-less verification and UPPAAL's Zone Automata

Prepared using stvrauth.cls
DOI: 10.1002/stvr

No of Nodes	# States Stored	# States Matched	# Transitions	Memory (MB)	Time (sec)
2	563	463	1026	8.501	0.1
3	18220	29625	47845	8.598	0.11
4	667995	1716011	2384003	44.383	5.01
5	21373206	75073507	96446713	395.366	203.09
6	36720364	1.4129329e+08	1.7801365e+08	1722.014	908.56

Table II: Computational resources required for verification of Fischer's Protocol by Spin

based verification are based on abstracting an infinite system to a finite one, this experimental result shows that our technique is more scalable than UPPAAL, while using SAL-smc model checker.

In [3], Beyar et al. present a TA model for the Fisher's protocol for verification purposes using Rabbit tool. Using a reduction technique based on Zone Automata in combination with an efficient variable ordering for the BDD based representation of the transition relation, this tool could verify reachability properties for 128 nodes in 5200 seconds. Obviously, this Rabbit tool performs much better than our analysis in terms of scalability.

[Example 2] Train gate Controller

Let us again consider Train-Gate Controller (TGC), a description of which has been provided in Section 2 (further see Figure 2 which depicts the timeout-based synchronous model of TGC). A clock-less model of TGC is depicted in Figure 4. For the TGC example, we consider `safety` and `timeliness` properties for verification.

The `safety` property is as follows: *when the Train crosses the line, the Gate should be down*. The property is expressed in LTL as:

$$\Box((t\_state = t_2) \Rightarrow (g\_state = g_2))$$

where, t_state denotes different states of Train, and it is at t_2, when it comes into the crossing. Moreover, g_state denotes different states of Gate, and is at g_2, when the Gate is down.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

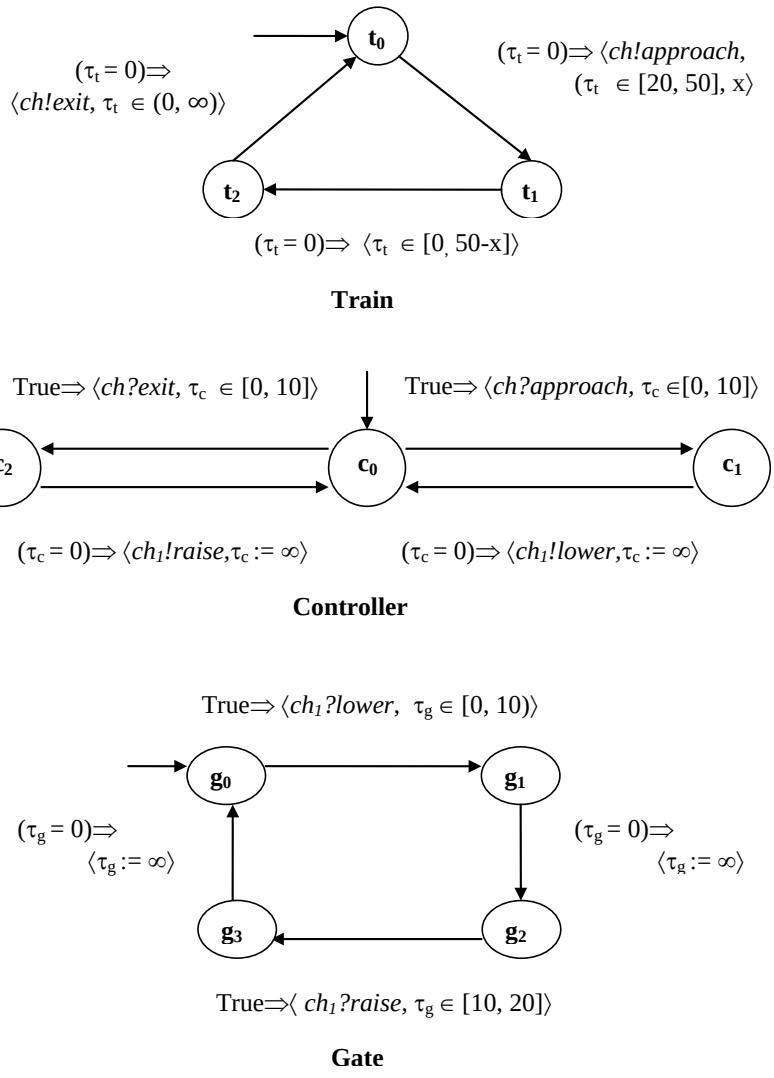

Figure 4: Clock-less model for Train-Gate Controller

`Timeliness` property, in general ensures that the time between two states will by bounded by a particular value. We can find many timeliness properties for this example. We select the following, *"the difference in time between the transmission of the* $approach$ *signal by the Train and the lowering of the Gate should not be more than* 20 *time units"*. To verify this property we use two auxiliary flags, $flag_1$ and $flag_2$ in our model. When the first event occurs $flag_1$ is set as $true$. When the second event happens, $flag_2$ is set as $true$ and $flag_1$ is reset to $false$.

A global variable $time\_diff$ is initially set to 0, which captures the time between the instants when two flags are set. During every discrete transition (on which timeouts are updated) occurring

Prepared using stvrauth.cls
DOI: 10.1002/stvr

Properties	# States Explored	Time (sec)
`Safety`	1.123e6	5.24
`Timeliness`	4.807e5	2.41

Table III: Computational resources required for verification of TGC by SAL-smc

Properties	# States Stored	# States Matched	# Transitions	Memory (MB)	Time (sec)
`Safety`	246236	422596	668832	47.947	1.50
`Timeliness`	253500	415484	668984	50.389	1.58

Table IV: Computational resources and time required for verification of TGC by Spin

between the two discrete transitions of interest, minimum timeout value is added to $time\_diff$. The `timeliness` property is then specified as follows, *"the value of* $time\_diff$ *never goes beyond 20"*. This is expressed in LTL as,

$$\Box(time\_diff \ \leq \ 20)$$

We verify the `safety` and `timeliness` properties for TGC by SAL-smc, and the result is shown in Table III. It might be noted that dense time verification of the `safety` property for TGC took 46.15 seconds [4]. This was proved by k-induction at depth 14 using SAL-inf-bmc.

In Table IV, we illustrate computational resources and time required to prove the `safety` and the `timeliness` properties for TGC by Spin model checker [12]. Both the properties have been proved by *exhaustive verification* keeping the the option of *partial order reduction* turned on.

[Example3] TTA Startup Protocol

TTA startup process executes on a logical bus meant for safety-critical applications in both automotive and aerospace industries. In a normal operation, $N > 1$ nodes share a TTA bus on which they broadcast messages. Figure 5 depicts the calendar-based clock-less timeout transition diagram of the $i^{th}, 1 \leq i \leq N$ node. Edges are labeled with the guard,

Prepared using stvrauth.cls
DOI: 10.1002/stvr

guard_predicate $\Rightarrow \langle$[send/receive](message), timeout_update, [record_time_var]$\rangle$, where guard_predicate denotes the constraint on a transition edge, the satisfaction of which triggers the update of timeouts and occurrence of events like, send(message)/receive(message). Moreover, the (optional) variable record_time_var records the time when a transition occurs on the edge. Each node i has a local timeout τ_i. When a node is powered-on, it performs some internal initialization, and transits to the *Listen* state. In this state it listens for the duration of c_i^{lis} time units to determine if there is a synchronous set of nodes communicating on the bus medium. The nodes which are in the *Active* state are already synchronized, and periodically transmit i-frames. If a node in the *Listen* state receives such an i-frame, it gets synchronized with the set of already synchronous nodes and transits to the *Active* state. If the above does not happen, there are two possibilities. The node waits for a message containing cs-frame* from another node which is not in the *Active* state. When a node completes the reception of a cs-frame, it resets its timeout and enters the *Coldstart* state. Each node that receives neither an i-frame nor a cs-frame during the *Listen* phase enters the *Coldstart* state on its listen timeout and broadcasts a cs-frame after resetting its timeout. Each node in the *Coldstart* state waits for reception of another i-frame until the clock reaches the value of its timeout. If it receives such a frame it synchronizes on its contents and enters the *Active* state; if not, it again broadcasts a cs-frame and loops into the *Coldstart* state. The goal of the startup algorithm is to bring the system from the power-up state, in which all processors are unsynchronized, to the normal operation mode in which all processors are synchronized and follow the same TDMA schedule [23]. For an elaborate description of startup protocol, we refer the reader to [23].

For TTA startup algorithm, we consider the following *safety* property: "whenever two nodes are in their *active* states, the nodes agree on the slot time". This property can be specified in SAL as follows (with the usual meaning for the symbols used below):

```
synchro: THEOREM
    tta |- G(FORALL (i, j: IDENTITY):
```

*formats of the i-frame and the cs-frame are almost same, only their contents differ.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

```
    pc[i] = active AND pc[j] = active AND
    time_out[i] > 0 AND time_out[j] > 0 =>
        time_out[i] = time_out[j] AND slot[i] = slot[j]);
```

The `safety` property ensures that when the nodes are in $active$ state, they are indeed synchronized. But it does not address the question whether all the nodes will be eventually synchronized. To ensure this happens, it is specified in the form of the following `liveness` property, *"eventually all the nodes will be in $active$ state and continue to do so"*. This *liveness property* can be specified in SAL as follows:

```
liveness: LEMMA
    tta |- F(FORALL(i: IDENTITY): G(pc[i] = active));
```

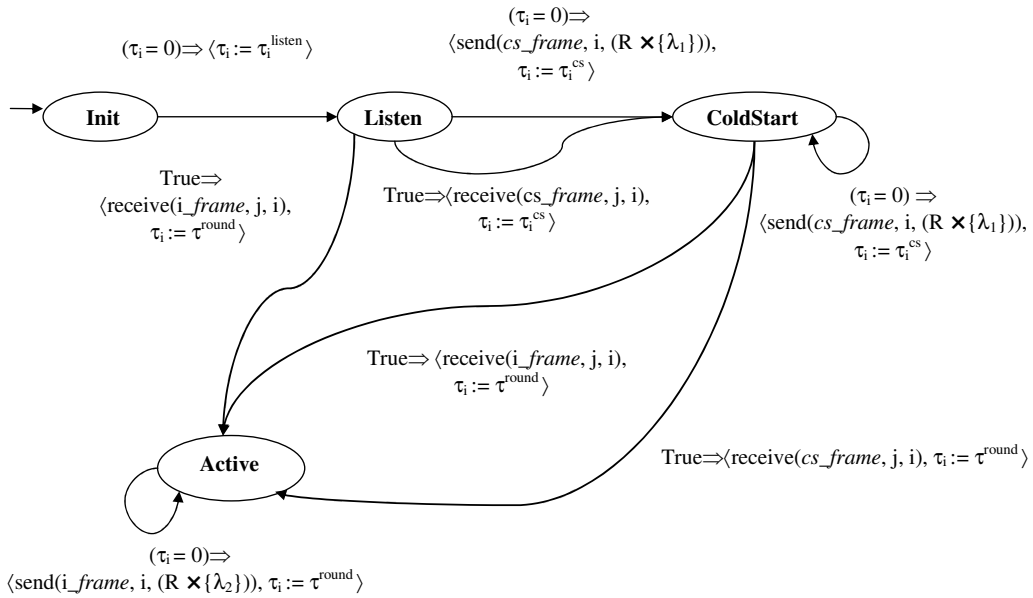

Figure 5: clock-less model for the i^{th} processor in TTA Startup algorithm.

Table V describes the number of states and time required to prove the safety and liveness properties for the TTA Startup protocol using SAL-smab. Let us compare our verification effort with the dense time modeling and verification of the same protocol reported earlier. In [4] and [7], using bounded model checking the same safety property was proved for only 2 nodes by k-induction at depth 9, that too using 3 auxiliary lemmas (the proof failed for 3 nodes). However the invariant

Prepared using stvrauth.cls
DOI: 10.1002/stvr

can be strengthened by constructing an abstraction of the transition systems using a verification diagram-based approach [24], subsequently the property was verified for upto 10 nodes.

No of Nodes	States	Time to Prove Safety Property	Time to Prove Liveness property
2	68	2.874	7.250
3	485	4.536	15.963
4	5297	9.794	65.163
5	76345	60.507	147.512
6	1331650	135.443	513.798
7	26872795	1479.757	1134.731

Table V: Verifying safety and liveness properties by SAL-smc for the TTA Startup.

We also verified these properties using Spin, for which we specified the properties using LTL [12]. For two nodes participating in the startup process, the `safety` property is encoded in LTL as below:

$$\Box((p_1 \wedge p_2) \wedge (q_1 \wedge q_2) \ \Rightarrow \ \Diamond(r \wedge s)),$$

where $p_1 \equiv (pc[0] = state\_active), p_2 \equiv (pc[1] = state\_active)$, $q_1 \equiv (time\_out[0] > 0)$, $q_2 \equiv (time\_out[1] > 0)$, $r \equiv (time\_out[0] = time\_out[1])$, $s \equiv (slot[0] = slot[1])$. Also, $pc[i]$ denotes the current state of the i^{th} node, $time\_out[i]$ denotes the timeout of the i^{th} node, and $slot[i]$ denotes the current time slot viewed by the i^{th} node. $state\_active$ characterizes the $synchronized$ state of a node.

The `liveness` property for two nodes can be specified in LTL as follows:

$$\Diamond\Box((pc[0] = state\_active) \wedge (pc[1] = state\_active))$$

To verify the `safety` and the `liveness` property for TTA startup in Spin, we use both *exhaustive verification* and *bitstate hashing* techniques with *partial order reduction* availed. By *exhaustive verification* technique, the `safety` property can be verified for TTA models containing upto 5 nodes, and the `liveness` property can be verified upto 4 nodes. *Bitstate hashing* enables us to

Prepared using stvrauth.cls
DOI: 10.1002/stvr

verify both the properties for models with upto 9 nodes. For 10 nodes, the verification does not terminate even in 4 hours. Table VI shows the computational resources and time required to prove the `safety` and `liveness` properties for TTA Startup protocol using *bitstate hashing* technique.

Properties	No of Nodes N	# States Stored	# States Matched	# Transitions	Memory (MB)	Time (sec)
Safety	2	487	143	630	8.501	0.01
	3	6142	6490	12632	8.501	0.05
	4	217852	483497	701349	8.501	1.46
	5	4126813	13188075	17314888	8.501	34.72
	6	16508262	62593403	79101665	8.501	165.46
	7	34442659	1.2702415e+08	1.6146681e+08	8.501	364.99
	8	40175448	2.4473144e+08	2.8490689e+08	8.598	665.63
	9	41008029	1.2976237e+09	1.3386317e+09	8.598	4390.17
Liveness	2	725	1036	2481	8.501	0.04
	3	8305	21980	38562	8.501	0.12
	4	249439	1149753	1648373	8.501	3.75
	5	4339737	28293352	36972211	8.501	83.32
	6	12678951	1.1096373e+08	1.3851011e+08	8.501	314.08
	7	20128894	2.0273546e+08	2.4108713e+08	8.501	531.80
	8	25361336	3.4848047e+08	3.8927174e+08	8.598	936.05
	9	40305514	2.307274e+09	2.3482827e+09	8.598	7039.02

Table VI: Computational resources and time required to verify TTA startup by Spin

Let us compare our verification effort with the dense time modeling and verification of the same protocol reported. In contrast, in [4, 7], using bounded model checking the same safety property was proved for only 2 nodes by k-induction at depth 9, that too using 3 auxiliary lemmas (the proof failed for 3 nodes). However the invariant can be strengthened by constructing an abstraction of the

Prepared using stvrauth.cls
DOI: 10.1002/stvr

transition systems using a verification diagram-based approach [24], and subsequently the property was verified for upto 10 nodes.

A fair comparison with similar verification efforts Our experimental results bring out the advantages gained by this technique over infinite state modeling and verification. Experiments on Fischer's protocol and TTA startup protocol highlight that the verification technique scales reasonably well. Though in [4], it has been reported that verification of Fischer's protocol can be scaled up to 53 nodes, the verification process involved finding out auxiliary lemmas manually, which is a non-trivial process. On the other hand our finite state verification, though could not be scaled to this extent, is nonetheless simple and straight-forward. The verification effort involves only modeling the protocols faithfully.

The tangible benefit achieved using our technique is reasonable scalability obtained using finite state model checker. Also this approach enables one to verify liveness properties in a simple manner. It may be noted that liveness properties cannot be easily verified using induction-based proof on SAL. While proving a liveness property one needs to find a proof by examining all finite sequences of states of length k starting from initial states [25]. Using a transformation mentioned in [25] it may be possible to establish a proof of liveness property (involving AG operators in CTL) in SAL, however the proof may get quite complex.

Like SAL-smc, Rabbit uses BDD-based symbolic model checking too. However, it uses some heuristics to determine the ordering of BDD variables, which helps scaling up the verification to large number of processes for Fischer's protocol. We use the default variable ordering of SAL-smc, which may not be the best ordering for verifying Fischer's protocol. On the other hand, Rabbit cannot deal with liveness properties. One could probably implement our algorithm on top of Rabbit and obtain the scalability of Rabbit, in the process also verify liveness properties.

6. CONCLUSION

In this work we have considered the well-known problem of real-time verification with dense time dynamics using timeout and calendar based models and proposed a technique to simplify

Prepared using stvrauth.cls
DOI: 10.1002/stvr

this to a finite state verification problem. Towards this, we define a specification formalism for these models as timeout transition diagrams with associated transition system semantics. Next, we proposed a two-step reduction technique for rendering these models amenable to finite state verification under discrete dynamics. Moreover, one can use any finite state verification engine of choice using our framework. We restricted our attention to the qualitative temporal properties that exclusively corresponds to LTL formulas. However, the proposed reduction technique is amenable to any specification logic which is closed under inverse digitization including branching time temporal logics such as, CTL or CTL*.

Similar techniques like digitization has been studied before to reduce dense time verification to discrete time verification problem. One popular technique is called *discretization* which is based on the notion of *sampling invariance* of a specification formula proposed in [26]. In this, a temporal formula is said to be sampling invariant when its discrete time model coincides with the sampling of its continuous time models. By observing a continuous time model at periodic instants of time one can sample it as a discrete model. Using sampling invariant specification it is possible to combine continuous time parts and discrete time parts which facilitates discrete time verification using automated techniques. Using these ideas, in [27] the authors have verified MTL specification formulas for dense time by reducing the validity problem from dense time to discrete time. As an undecidable problem has been reduced to a decidable problem the technique is necessarily incomplete as it fails to provide conclusive answers on some problem instances. The verification can be performed using discrete-time bounded validity checking. Let us contrast the verification techniques of this work and ours. While this verification technique is not complete our technique is able to provide conclusive answers in all the cases where the technical condition for digitization of timeout models hold true (we restrict ourselves to decidable verification problems). Moreover, the former verification involves lot of intricacies as it uses bounded model checking; whereas we reduce discrete time verification problem to finite state model checking through finitary reduction which benefits from efficient automated approaches.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

The effectiveness of the proposed reduction can be further scaled up by integrating it with additional abstraction techniques used for verifying parametric systems, with arbitrary but finite number of identical processes. For such purpose one can consider algorithm proposed by Saïdi and Lesens in [28] for automatically constructing abstraction for parametric systems in order to verify `safety` properties.

The proposed formalism can be further optimized by considering timeouts as shared variables among processes, so that timeout update rules could be based upon shared timeouts of other processes in the system. This optimization would increase the level of synchronization between component processes and would hopefully scale up the models. It would be interesting our technique on other symbolic model checkers wherein BDD variables could provide necessary optimization which we leave for future work.

ACKNOWLEDGMENT

Most of this work were done when the authors were with HTS Research, Bangalore, India.

REFERENCES

1. Alur R, Courcoubetis C, Dill D. Model-checking in dense real-time. *Information and Computation* 1993; **104**:2–34.
2. Clarke EM, Biere A, Raimi R, Zhu Y. Bounded model checking using satisfiability solving. *Formal Methods in System Design* 2001; **19**(1):7–34.
3. Beyer D, Lewerentz C, Noack A. Rabbit: A tool for BDD-based verification of real-time systems. *Computer Aided Verification*, Springer, 2003; 122–125.
4. Dutertre B, Sorea M. Timed Systems in SAL. *Technical Report*, Computer Science Laboratory, SRI International 2004.
5. Olderog ER, Dierks H. *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge University Press, 2008.
6. Clarke EM, Emerson EA, Sifakis J. Model checking: algorithmic verification and debugging. *Commun. ACM* 2009; **52**(11):74–84.
7. Dutertre B, Sorea M. Modeling and Verification of a Fault-Tolerant Real-time Startup Protocol using Calendar Automata. *Proc. of FORMATS/FTRTFT*, *LNCS*, vol. 3253, Springer, 2004; 199–214.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

8. de Moura L, Owre S, Rueß H, Rushby J, Alur R, Peled D. SAL 2. *Proc. of International Conference on Computer-Aided Verification (CAV'04)*, *LNCS*, vol. 3114, Springer-Verlag, 2004; 496–500.
9. Henzinger TA, Manna Z, Pnueli A. What Good Are Digital Clocks? *Proc. of the 19th International Colloquium on Automata, Languages and Programming (ICALP'92)*, Springer-Verlag, 1992; 545–558.
10. Pike L. Real-time system verification by k-induction. *Technical Report TM-2005-213751*, NASA Technical Memorandum 2005.
11. Henzinger TA, Manna Z, Pnueli A. Timed Transition Systems. *Proc. of the Real-Time: Theory in Practice, REX Workshop*, Springer-Verlag: London, UK, UK, 1992; 226–251.
12. Saha I, Misra J, Roy S. Timeout and Calendar Based Finite State Modeling and Verification of Real-Time Systems. *Proc. of 5th International Symposium on Automated Technology for Verification and Analysis (ATVA'07)*, *LNCS*, vol. 4762, Springer, 2007; 284–299.
13. Lamport L. Real-time model checking is really simple. *Proceeding, Correct Hardware Design and Verification Methods, 13th IFIP WG 10.5 Advanced Research Working Conference, CHARME*, *Lecture Notes in Computer Science*, vol. 3725, Springer, 2005; 162–175.
14. Alur R, Dill L D. A Theory of Timed Automata. *Theoretical computer science* 1994; **126**(2):183–235.
15. Chun KY, Van Hung D. Verifying real-time systems using untimed model checking tools. *The United Nations University. Tech. Report UNU-IIST* 2004; **302**.
16. Holzmann G. *The SPIN Model Checker: Primer and Reference Manual*. 1st edn., Addison-Wesley Professional, 2003.
17. Behrmann G, David A, Larsen KG. A Tutorial on UPPAAL. *Proc. of the 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*, *LNCS*, vol. 3185, Springer-Verlag, New York, 2004; 200–236.
18. Alur R. Timed automata. *Proceedings of International Conference on Computer-Aided Verification (CAV'99)*, *LNCS*, vol. 1633, Springer-Verlag, 1999; 2–8.
19. Alur R, Henzinger T. Logics and Models of Real-time: A Survey. *Proc. of the Real-Time: Theory in Practice, REX Workshop*, *LNCS*, vol. 600, Springer, 1992; 74–106.
20. Bošnacki D. Digitization of timed automata. *Proc. of FMICS*, vol. 99, Springer, 1999; 283–302.
21. Clarke EM, Grumberg O, Peled D. *Model checking*. MIT press, 2000.
22. Saha I, Misra J, Roy S. A Simplification of a Real-Time Verification Problem. *CoRR* 2010; **abs/1008.1417**. URL `http://arxiv.org/abs/1008.1417`.
23. Steiner W, Paulitsch M. The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault- Tolerant System. *Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, vol. 22, IEEE Computer Society, 2002; 329–336.
24. Rushby J. Verification diagrams revisited: Disjunctive invariants for easy verification. *Computer Aided Verification*, vol. LNCS 1855, Springer, 2000; 508–520.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

25. Biere A, Cimatti A, Clarke EM, Strichman O, Zhu Y. Bounded model checking. *Advances in Computers* 2003; **58**:117–148.

26. Furia CA, Rossi M. Integrating discrete- and continuous-time metric temporal logics through sampling. *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS, Proceedings*, *Lecture Notes in Computer Science*, vol. 4202, Springer, 2006; 215–229.

27. Furia CA, Pradella M, Rossi M. Automated verification of dense-time MTL specifications via discrete-time approximation. *15th International Symposium on Formal Methods (FM'08), Proceedings*, *Lecture Notes in Computer Science*, vol. 5014, Springer, 2008; 132–147.

28. Lesens D, Saidi H. Automatic verification of parameterized networks of processes by abstraction. *Electronic Notes of Theoretical Computer Science (ENTCS)* 1997; .

DOI: 10.1002/stvr