Timeout and Calendar based Finite State Modeling and Verification of Real-Time Systems

Indranil Saha, Janardan Misra, and Suman Roy

HTS (Honeywell Technology Solutions) Research Lab,
151/1 Doraisanipalya, Bannerghatta Road, Bangalore 560 076, India
Email: {indranil.saha, janardan.misra, suman.roy}@honeywell.com

Abstract. To overcome the complexity of verification of real-time systems with dense time dynamics, Dutertre and Sorea proposed timeout and calender based transition systems to model real-time systems and verify safety properties using k-induction. In this work, we propose a canonical finitary reduction technique, which reduces the infinite state space of timeout and calender based transition systems to a finite state space. The technique is formalized in terms of clockless finite state timeout and calendar based models represented as predicate transition diagrams. Using the proposed reduction, we can verify these systems using finite state model checkers and thus can avoid the complexity of induction based proof methodology. We present examples of Train-Gate Controller and the TTA startup algorithm to demonstrate how such an approach can be efficiently used for verifying safety, liveness, and timeliness properties using the finite state model checker Spin.

1 Introduction

Modeling and verification of timeout based real-time systems with continuous dynamics is an important and hard problem that has evoked a lot of prime research interest with industrial focus for many years in the recent past. The problem of faithfully modeling and consequently formally verifying such timeout based real-time systems is rather difficult because the state space of these systems is essentially infinite owing to the diverging valuation required by the timing and timeout variables. Because of this infiniteness of the state space none of the known formal verification techniques can be applied to completely verify some of the interesting properties, e.g., liveness properties, timing deadlocks etc. Although infinite state model checkers like SAL (Symbolic Analysis Laboratory) [10] have been used with limited success for verifying safety properties. The verification process employed by these tools demands significant additional manual efforts in defining supporting lemmas and abstractions for scaling up the model.

Spin [9] is a tool for automatically verifying finite state distributed systems. There are broadly two attempts for extending Spin with time [4, 5, 16]. Real-time extension of Spin (RT-Spin [16]) is one such work, which provides timed automata (TA) [1] with real-valued clocks as a modeling framework, though is incompatible with the partial order reduction implementation of Spin. Another is the work on DT-Spin [4, 5], which allows one to quantify (discrete) time elapsed between events, by specifying the time slice in which they occur. DT-Spin is compatible with the partial order reduction of Spin and has been used to verify industrial protocols, e.g., AFDX Frame management protocol [13] and TTCAN [14]. Nonetheless, systems with asynchronous communication with bounded delays between components cannot be modeled directly by using the mechanism of asynchronous channels that Spin provides since there is no explicit provision to capture message transmission delays. One possibility is to model each channel as a separate process with delay as a state variable. In [4], the channels in the example of PAR protocol have been implemented in the same way. But for systems with relatively large number of components and dense connectivity among the components, modeling channels in this way is difficult and state space explosion becomes an unavoidable problem. UPPAAL [2], which can model TA, has the same limitation when modeling asynchronous communications with bounded delays - every channel has to be modeled as a separate TA capturing the message transmission delays.

Dutertre and Sorea [6] proposed timeout based modeling of time triggered systems with dense time dynamics, which have been traditionally used as a model of execution in discrete event system simulations. They presented a modeling approach, where expected delivery delays for all undelivered messages can be stored in a global data structure called *calendar* [6, 7]. Formally, a *calendar* is a set of bounded size of the form $C = \{\langle e_1, t_1\rangle, \ldots, \langle e_r, t_r\rangle\}$, where each event e_i is associated with the time point t_i when it is scheduled to occur. The calendar based model along with the timeouts for individual processes has been used to model the TTA startup protocol [7]. Using the infinite bounded model checker of SAL [10], they proved the safety property by k induction. Unfortunately, not all of the safety properties are inductive in nature and therefore may require support of auxiliary lemmas. In [7], proof of the safety property for the TTA startup having just 2 nodes itself required 3 additional lemmas. A verification diagram based abstraction method proposed in [12], was used to prove the invariant property for models having upto 10 nodes. However, liveness properties still remain beyond the scope of this approach. Pike [11] builds on the work of [6] and proposes a new formalism called Synchronizing Timeout Automata (STA) to reduce the induction depth k required for k-induction. STA is defined using shared timeouts such that the resulting transition system does not involve a clock.

Since in timeout and calendar based models, global time and timeouts always increase, such models cannot be directly used for finite state verification. To that end, we propose a finitary reduction technique which effectively reduces the infinite state timeout and calendar based transition systems with discrete dynamics to finite state transition systems. This technique enables us to model a real-time system without considering a clock explicitly. We formalize the timeout and calendar based models as predicate transition diagrams and their behavior in terms of timeout and calendar based transition systems. Such a formal modeling framework provides background to effectively reason about the correctness of the various possible hypotheses for efficiently verifying these models beyond limited experiments. We demonstrate by examples, how such a modeling approach can be efficiently used for verifying safety, liveness, and timeliness properties using the finite state model checker Spin.

The remainder of the paper is organized as follows: section, In Section 2, we describe the finitary reduction technique and formalize it in terms of clockless modeling in Section 3. In Section 4 we discuss models of time and executablitiy conditions for dense time model. Section 5 presents the experimental results followed by concluding discussion in section 6.

2 Finitary Reduction

With reference to the timeout and calendar based modeling presented in [6, 7], notice that although these models can be used to efficiently capture dense time semantics without using a continuously varying clock, it is difficult to use these models for finite state model checking. The difficulty arises because of the fact that the valuations for the global clock t and the timeout variables in $\mathcal{T}$ diverge and thus are not bounded by a finite domain. Unlike TA one cannot reset the global clock or the individual timeouts in these models because straightforward attempts for such resetting results only in incorrect behaviors. One possible solution may be to bound the value of the global clock and the timeouts by appropriate large constants based upon the system specification. But such a upper bound is quite difficult to estimate in case of practical industrial applications and also with such an approach liveness properties cannot be verified.

We propose a finitary reduction technique, which is formalized in terms of clockless modeling and semantics in the next section. This technique effectively reduces the timeout and calendar based transition systems with discrete dynamics into finite state systems, which, in turn, can be expressed and model checked by finite state model checkers.

Informally, the technique can be described as follows: To implement time progress transition, a special process is required to increase the global clock to the minimum of timeouts, when each of the timeout values is strictly greater than the current value of the clock. Other processes wait until their timeouts are equal to the global clock, and when it is so, they take the discrete transitions and updates their timeouts in future. We propose to model the special process which is responsible for time progress transition in such a way that it does not explicitly use the clock variable and prevents the timeout variables to grow infinitely. We call this process *time_progress*. When no discrete transition is possible in the system due to the fact that the discrete transitions for all the systems are scheduled in the future, *time_progress* finds out the minimum of all the timeouts in $\mathcal{T}$ and scales down all these timeouts by the minimum. In this way at least one of the timeouts becomes zero. A process is allowed to take a discrete transition when its timeout becomes zero. When it happens the process updates its timeout and does other necessary jobs.

If the timeouts are always incremented by finite values then it is guaranteed that the value of a timeout will always be in a finite domain. But there are cases when a timeout increment cannot be bounded by finite value. For example, a process may have to wait for an external signal before its next discrete transition. In this case, next discrete transition of the process does not depend on its own timeout, so the timeout of the process is set to the relatively large value, so that it does not affect the next discrete transitions of other processes. In another situation, it may be desired that 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. In that case all what we need is to limit the value of the timeout without omitting any of the possible interleaving of the process steps. To do that we limit the timeout value in $[0, M+1]$, where M is the maximum of all the integer constants that are used to define the upper limit of different timeouts for different processes in the system.

The suggested technique gives rise to a canonical representation of the clock and timeout valuations in any state in the sense that for the timeout and calendar based models considered here, there cannot be any further reduction possible without actually loosing the relative timing delay information. This is because this technique effectively reduces timeout valuations into a canonical partial ordering structure and also simultaneously keeps the information on the actual timeout increments intact. This approach can be seamlessly extended for the calendar based models as well.

It should be added that the finitary reduction considered in this work is effective only under discrete dynamics since with dense modeling such a reduction though reduces an infinite region (e.g., $\mathcal{R}^n$) to a finitely bounded region (e.g., $[0,1]^n$), it would still contain infinitely many points resulting into infinite permissible paths.

Above discussion is formalized in terms of "clockless" modeling and associated semantics in the next section.

3 Timeout and Calendar based Clockless Models

In this section we provide a formalization of timeout and calendar based clockless models as predicate transition diagrams and associated semantics in terms of state transition systems.

3.1 Timeout based Models: Clockless Modeling

Syntax The Timeout based Model (ToM) ([6]) can be represented as

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

Where each process P_i is a sequential non-deterministic process having τ_i as its local timeout and $\mathcal{X}_i$ as a set of local timing variables used for determining the relative delay between events. "$||$" is the parallel composition operator. Formula θ restricts the initial values of variables in

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

where the set of all timeouts is $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, and $\mathcal{X} = \bigcup_i \mathcal{X}_i$. $Var = G \cup L_1 \cup L_2 \cup \ldots \cup L_n$ is the set of other state variables assuming values from finite domains. Variables in G are globally shared among all the processes while L_i contains variables local to process P_i. f^{Var} is the set of computable functions on Var.

Each process P_i is represented using a *predicate transition diagram*, which is a finite directed graph with nodes $Loc_i = \{l^i_0, l^i_1, \ldots, l^i_{m_i}\}$, called *locations*. The entry location is l^i_0. There are two kinds of edges in the graph of a process P_i: *Timeout edges* and *Synchronous Communication edges*. Edge definitions involve an enabling condition or guard ρ, which is a boolean-valued function or a predicate.

Timeout Edges: A timeout edge $(l^i_j, \rho \Rightarrow \langle \tau_i := update_i, \eta, f\rangle, l^i_k)$ in the graph of the process P_i is represented as

$$l^i_j \xrightarrow{\rho \Rightarrow \langle \tau_i := update_i, \eta, f\rangle} l^i_k,$$

where $update_i$ specifies how timeout τ_i is to be updated on taking a transition on the edge when guard ρ evaluates to True. $\eta \subseteq \mathcal{X}_i$ specifies the local timing variables which capture the relative increment in the value of timeout τ_i while taking transition on the edge. $f \in f^{Var}$ manipulates the state variables in $G \cup L_i$.

$update_i$ is defined using the rule: $update_i = k_1 \mid k_2 \mid \infty \mid \max(\mathcal{M})$, where $l - z \prec k_1 \prec' m - z'$, $\prec, \prec' \in \{<, \leq\}$ and $k_2 \succ l - z$, $\succ \in \{>, \geq\}$; $z, z' := w|0$ and $l, m \in \mathcal{N}_0$ are non-negative integer constants. $\mathcal{M}$ is the set of all the integer constants that are used to define the upper limit of different timeouts for different processes in the system. $\max(\mathcal{M})$ returns the maximum of all the integers in $\mathcal{M}$.

Constraints on k_1, k_2 specify how the new value of timeout τ_i should be determined based upon the value of some local timing variable w, which would have captured the increments in the value of timeout τ_i in some earlier transitions. Setting a timeout to ∞ is used to capture the requirement of indefinite waiting for an external signal/event. Setting 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.

Synchronous Communication Edges: As rendezvous communication between a pair of processes (P_s, P_r) is represented by having an edge pair (e_s, e_r) s.t. $e_s \in P_s$ and $e_r \in P_r$:

$$e_s : l^s_j \xrightarrow{\rho \Rightarrow \langle ch!m, \tau_s := update_s, \eta, g\rangle} l^s_k$$

$$e_r : l^r_j \xrightarrow{True \Rightarrow \langle ch?\bar{m}, \tau_i := update_r, \eta', h\rangle} l^r_k$$

where *ch* is the channel name, $m \in L_s$ is the message sent, and $\bar{m} \in L_r$ receives the message; $g, h \in f^{\,Var}$.

Semantics With a given ToM

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

we associate the following transition system $S_P = (\mathcal{V}, \Sigma, \Sigma_0, \Gamma)$, which will be referred to as a *timeout based clockless transition system* :

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 $\perp$ denotes before the start of the process.
2. Σ is the set of states. Every state $\sigma \in \Sigma$ is an interpretation of $\mathcal{V}$ such that, for $x \in \mathcal{V}$, $\sigma(x)$ is 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 and $\sigma_0(\pi_i) = \perp$ for each process P_i.
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 Transitions: Γ_e is the set of entry transitions and contains an *entry transition* ν_e^i for every process P_i. In particular $\forall \sigma_0 \in \Sigma_0$,

$$(\sigma_0, \sigma') \in \nu_e^i \Leftrightarrow \begin{cases} 1.\ \forall x \in \mathcal{U} :\ \sigma'(x) = \sigma_0(x) \\ 2.\ \forall \tau \in \mathcal{T} :\ \sigma'(\tau) \geq 0 \\ 3.\ \sigma_0(\pi_i) = \perp \ \text{ and } \sigma'(\pi_i) = l_0^i \end{cases}$$

Time Progress Transition: The first kind of edges $\nu_+ \in \Gamma_+$ are those where all the timeouts are decremented by the minimum of the current timeout values. In particular,

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

Timeout Increment Transition: If $(l_j^i, \rho \Rightarrow \langle update_i, \eta, f\rangle, l_k^i)$ is an edge in the predicate transition diagram for process P_i, then there is a corresponding edge $\nu_0^i \in \Gamma_0$:

$$(\sigma, \sigma') \in \nu_0^i \Leftrightarrow \begin{cases} 1.\ \rho \text{ holds in } \sigma \\ 2.\ \text{If } \sigma(\tau_i) = 0 \ \text{ then} \\ \quad \sigma'(\tau_i) = update_i > 0 \ \text{ else } \ \sigma'(\tau_i) = \sigma(\tau_i) \\ 3.\ \forall x \in \eta :\ \sigma'(x) = \sigma'(\tau_i) + \sigma(x) \text{ and} \\ \quad \forall x \in \mathcal{X} \setminus \eta :\ \sigma'(x) = \sigma(x) \\ 4.\ \forall v \in G \cup L_i : \sigma'(v) = f(\sigma(v)) \text{ and} \\ \quad \forall v \in Var \setminus (G \cup L_i) :\ \sigma'(v) = \sigma(v) \\ 5.\ \sigma(\pi_i) = \ l_j^i \text{ and } \sigma'(\pi_i) = l_k^i \end{cases}$$

If $update_i = k_1$ s.t. $l - z \prec k_1 \prec m - z'$, $update_i$ nondeterministically selects an integer δ such that $l - \sigma(z) \prec \delta \prec m - \sigma(z')$. If $update_i = k_2$ s.t. $k_2 \succ l - z$, $update_i$ nondeterministically selects an integer δ such that $\delta \succ l - \sigma(z)$, else if $update_i = \infty$, it selects a relatively very large integer value and returns it to account for indefinite waiting. If $update_i = \max(\mathcal{M})$, $update_i$ nondeterministically selects any integer δ in $[0, M+1]$, where M is the maximum of all the integers in $\mathcal{M}$ returned by $\max(\mathcal{M})$.

Synchronous Communication For a pair of processes P_s, P_r having edges (e_s, e_r) as defined before, $\nu^{sr}_{syn\_comm} \in \Gamma_{syn\_comm}$ exists such that:

$$(\sigma, \sigma') \in \nu^{sr}_{syn\_comm} \Leftrightarrow \begin{cases} 1.\ \rho \text{ holds in } \sigma \\ 2.\ \sigma'(\tau_s) = update_s > \sigma(\tau_s) \\ \quad\ \sigma'(\tau_r) = update_r > \sigma(\tau_r) \\ 3.\ \forall x \in (\eta):\ \sigma'(x) = \sigma'(\tau_s) + \sigma(x), \quad \text{and} \\ \quad\ \forall x \in (\eta'):\ \sigma'(x) = \sigma'(\tau_r) + \sigma(x) \quad \text{and} \\ \quad\ \forall x \in \mathcal{X} \setminus (\eta \cup \eta'):\ \sigma'(x) = \sigma(x) \\ 4.\ \sigma'(\bar{m}) = \sigma(m) \\ 5.\ \forall v \in G \cup L_s : \sigma'(v) = g(\sigma(v)), \quad \text{and} \\ \quad\ \forall v \in G \cup L_r : \sigma'(v) = h(\sigma(v)) \quad \text{and} \\ \quad\ \forall v \in Var \setminus (G \cup L_r \cup L_s):\ \sigma'(v) = \sigma(v) \\ 6.\ \sigma(\pi_s) = l^s_j, \sigma(\pi_r) = l^r_j \text{ and} \\ \quad\ \sigma'(\pi_s) = l^s_k, \sigma'(\pi_r) = l^r_k \end{cases}$$

This semantic model defines the set of possible computations of the timeout system P as a set of state sequences (possibly infinite) starting with some initial state in Σ_0 and following edges in Γ.

Example: Train-Gate Controller

We will illustrate the timeout based model as formalized above using the example of the Train-Gate Controller (TGC) (adapted from [1].) The example of TGC demonstrates synchronous communication between system components, since the communications between Train and Controller, and between Controller and Gate are assumed to be synchronous.

TGC is an automatic controller that controls the opening and closing of a Gate at railroad crossing. The system is composed of three components: Train, Gate, and Controller. 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 10 time units and the Gate has to be down within another 10 time units. The Train can enter the crossing at any time after 20 time units since it sent the *approach* signal. While exiting the crossing the Train sends the *exit* signal to the Controller. The requirement is that after sending the *approach* signal the Train must send the *exit* signal within 50 time units. The Controller sends the *raise* signal to the Gate within 10 time units after it receives the *exit* signal. The Gate is required to be up within another

10 time units. All the communications are assumed to be synchronous, that is, there is no message transmission delay.

Figure 1 demonstrates the clockless timeout based model of TGC. The timing requirements are captured by suitably defining the *update* functions on the edges. For example, consider the edge (t_0, t_1) for the train labeled with $(\tau_t = 0) \Rightarrow \langle ch!approach, (\tau_t := k | 20 \leq k \leq 50), x\rangle$. Here $(\tau_t = 0)$ indicates that the system starts when train sends the *approach* signal over the shared channel ch and nondeterministically sets its timeout τ_t to some value k between $[20, 50]$ indicating that after sending the *approach* signal it can enter the crossing any time after 20 time units. Upper limit of 50 is used to indicate that the train cannot enter later than 50 time units because it is required that train has to indeed exit the crossing on or before 50 time units. Having spent k units of time in state t_1, train takes transition on the next timeout to state t_2 and resets its timeout to some value k' between $[0, 50-k]$, which indicates that the train must exit (and send *exit* signal to the controller) from state t_2 no more than before it has spent at most total of 50 units of time in states t_1 and t_2, that is, $0 \leq k + k' \leq 50$. Similarly on taking a transition on edge from g_1 to g_2 for the gate, $\tau_g := \infty$ denotes that the Gate would be waiting for the signal *raise* in state g_2 to be received on channel ch_1 from the Controller.

3.2 Calendar Based Models: Clockless Modeling

Syntax To capture (lossless) asynchronous communication with bounded message transfer delay, timeout based model is extended with a calendar data structure. A calendar is a linear array of bounded size, where each cell contains the following information: {message, sender_id, receiver_id, expected_delivery_time}. Let $\mathcal{C}$ to denote the calendar array, a globally shared object. We have

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

Sending a message is represented in the predicate transition diagram of process P_i using the following edge:

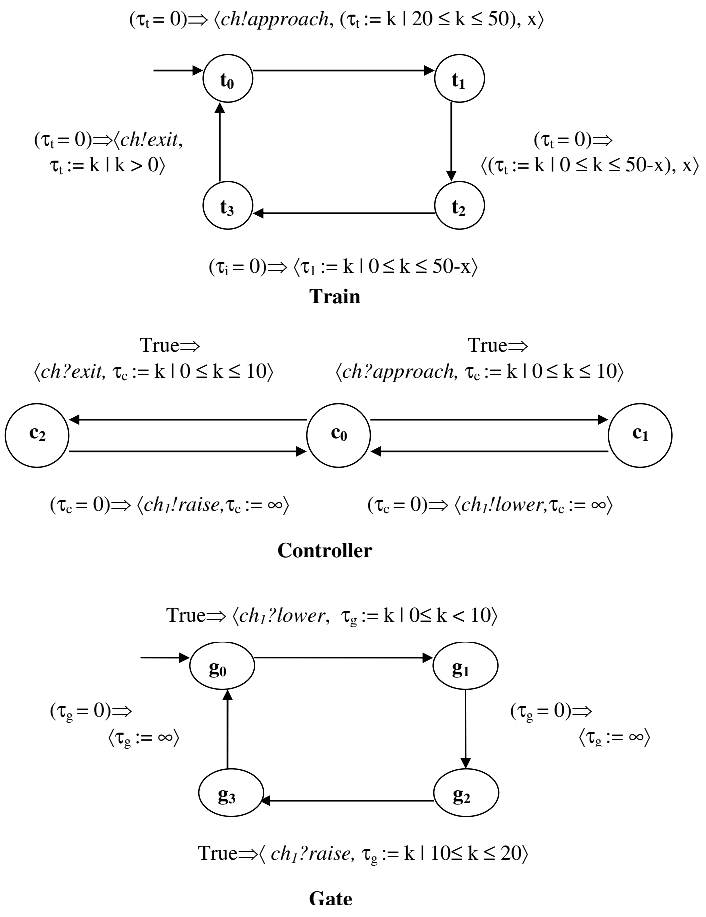
$$l^i_j \xrightarrow{\rho \Rightarrow \langle send(m,i,R,\Lambda), \tau_i := update_i, \eta, f\rangle} l^i_k,$$

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

Corresponding receiving of the message is represented in the predicate transition diagram of each of the processes P_r ($\forall\ r \in R$) using the following edge:

$$l^r_j \xrightarrow{True \Rightarrow \langle receive(m,i,r), \tau_r := update_r, \eta, g\rangle} l^r_k,$$

where $receive(..)$ specifies that a message m sent by process P_i is to be received by the process P_r. When 'time' elapsed in terms of timeout increments approaches some expected delivery time λ_r as specified by the sender process in the calendar $\mathcal{C}$ for entry $e =$ {m, i, r, λ_r}, a transition is taken on this edge and the entry e is deleted from $\mathcal{C}$.



$(\tau_t = 0) \Rightarrow \langle ch!approach, (\tau_t := k \mid 20 \leq k \leq 50), x\rangle$
t_0
t_1
$(\tau_t = 0) \Rightarrow \langle ch!exit, \tau_t := k \mid k > 0\rangle$
$(\tau_t = 0) \Rightarrow \langle(\tau_t := k \mid 0 \leq k \leq 50\text{-}x), x\rangle$
t_3
t_2
$(\tau_i = 0) \Rightarrow \langle \tau_1 := k \mid 0 \leq k \leq 50\text{-}x\rangle$
Train
True$\Rightarrow$ $\langle ch?exit, \tau_c := k \mid 0 \leq k \leq 10\rangle$
True$\Rightarrow$ $\langle ch?approach, \tau_c := k \mid 0 \leq k \leq 10\rangle$
c_2
c_0
c_1
$(\tau_c = 0) \Rightarrow \langle ch_1!raise, \tau_c := \infty\rangle$
$(\tau_c = 0) \Rightarrow \langle ch_1!lower, \tau_c := \infty\rangle$
Controller
True$\Rightarrow$ $\langle ch_1?lower, \tau_g := k \mid 0 \leq k < 10\rangle$
g_0
g_1
$(\tau_g = 0) \Rightarrow \langle \tau_g := \infty\rangle$
$(\tau_g = 0) \Rightarrow \langle \tau_g := \infty\rangle$
g_3
g_2
True$\Rightarrow$ $\langle ch_1?raise, \tau_g := k \mid 10 \leq k \leq 20\rangle$
Gate

Fig. 1. Clockless model for Train-Gate Controller

Semantics Given a calendar $\mathcal{C}$, we assume that the set of delays for all undelivered messages at any state σ can be extracted using function $\Delta : \sigma(\mathcal{C}) \rightarrow 2^{\mathcal{N}}$.

Let $\Gamma = \Gamma_e \cup \Gamma_+ \cup \Gamma_0 \cup \Gamma_{syn\_comm} \cup \Gamma_{asyn\_comm}$ denote the set of transitions in the *calendar based clockless transition system.* Γ_e (the set of Entry Transitions), Γ_{syn_comm} (Synchronous Communications) and Γ_0 (Timeout Increment Transitions) are defined in same way as in the timeout based model. The definition for the Time Progress Transition edges in Γ_+ are modified using calendar object $\mathcal{C}$ as follows:

Time Progress Transition: The edges ν_+ are redefined so that all the timeout and calendar delay entries are decremented by the minimum of all timeouts and the message delays in calendar. Let $\alpha = \min\{\sigma(\mathcal{T}) \cup \Delta(\sigma(\mathcal{C}))\}$,

$$(\sigma, \sigma') \in \nu_+ \Leftrightarrow \begin{cases} 1.\ \alpha > 0 \\ 2.\ \forall \tau \in \mathcal{T} :\ \sigma'(\tau) = \sigma(\tau) - \alpha \\ 3.\ \forall \lambda \in \Delta(\sigma(\mathcal{C})) :\ \sigma'(\lambda) = \sigma(\lambda) - \alpha \\ 4.\ \forall z \in \mathit{Var} :\ \sigma'(z) = \sigma(z) \\ 5.\ \text{If } \exists\{\text{m, i, r, } \lambda_r\} \in \sigma(\mathcal{C}) \text{ such that } \alpha = \lambda_r \\ \qquad \text{then } \forall x \in \mathcal{X}_r :\ \sigma'(x) = \sigma(x) + \alpha \text{ and} \\ \qquad\qquad \forall x \in \mathcal{X} \setminus \mathcal{X}_r :\ \sigma'(x) = \sigma(x) \\ \quad \textbf{else } \forall x \in \mathcal{X} :\ \sigma'(x) = \sigma(x) \\ 6.\ \forall i :\ \sigma'(\pi_i) = \sigma(\pi_i) \end{cases}$$

We additionally define new transitions Γ_{asyn_comm} corresponding to $send()$ and $receive()$ to capture asynchronous communication:

Send Transition: If $(l^i_j, \rho \Rightarrow \langle send(m, i, R, \Lambda), update_i, \eta, f\rangle, l^i_k)$ is an edge in process P_i, then we have a corresponding edge ν^i_{send} which adds $|R|$ cells to the calendar array $\mathcal{C}$:

$$(\sigma, \sigma') \in \nu^i_{send} \Leftrightarrow \begin{cases} 1.\ \rho \text{ holds in } \sigma \\ 2.\ \text{If } \min\{\sigma(\mathcal{T})\} = \sigma(\tau_i) = 0 \\ \qquad \text{then } \sigma'(\tau_i) = update_i > 0 \\ \quad \text{else } \sigma'(\tau_i) = \sigma(\tau_i) \\ 4.\ \forall x \in \eta :\ \sigma'(x) = \sigma'(\tau_i) + \sigma(x) \text{ and} \\ \quad \forall x \in \mathcal{X} \setminus \eta :\ \sigma'(x) = \sigma(x) \\ 5.\ \forall v \in G \cup L_i : \sigma'(v) = f(\sigma(v)) \text{ and} \\ \quad \forall v \in \mathit{Var} \setminus (G \cup L_i) :\ \sigma'(v) = \sigma(v) \\ 6.\ \forall r \in R : \sigma'(\mathcal{C}) := \sigma(\mathcal{C}) + \{\text{m, i, r, } \lambda_r\} \\ 7.\ \sigma(\pi_i) = l^i_j \text{ and } \sigma'(\pi_i) = l^i_k \end{cases}$$

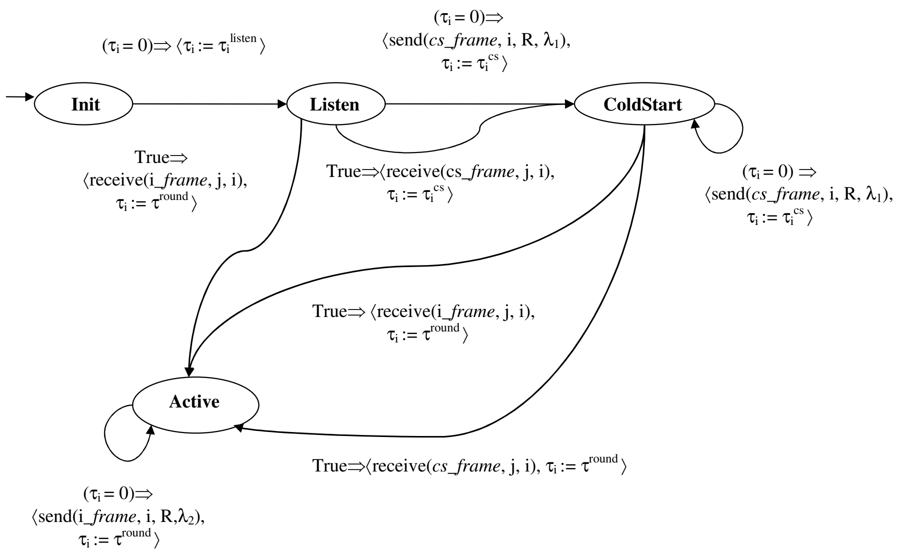
Receive Transition: If $(l^r_j, True \Rightarrow \langle receive(m, i, r), \tau_r := update_r, \eta, g\rangle, l^r_k)$ is an edge in the graph of process P_r, then we have a corresponding edge $\nu^r_{receive} \in \Gamma_{asyn\_comm}$, which deletes the entry $\{m, i, r, \lambda_r\}$ from the calendar array $\mathcal{C}$ when λ_r is 0:

$$(\sigma, \sigma') \in \nu^r_{receive} \Leftrightarrow \begin{cases} 1.\ \exists\{m, i, r, \lambda_r\} \in \sigma(\mathcal{C}) \text{ s.t. } \lambda_r = 0 \\ 2.\ \sigma'(\tau_r) = update_r > 0 \\ 3.\ \forall x \in \eta :\ \sigma'(x) = \sigma'(\tau_r) + \sigma(x) \text{ and} \\ \quad \forall x \in \mathcal{X} \setminus \eta :\ \sigma'(x) = \sigma(x) \\ 4.\ \forall v \in G \cup L_r : \sigma'(v) = f(\sigma(v)) \text{ and} \\ \quad \forall v \in \mathit{Var} \setminus (G \cup L_r) :\ \sigma'(v) = \sigma(v) \\ 5.\ \sigma'(\mathcal{C}) := \sigma(\mathcal{C}) - \{m, i, r, \lambda_r\} \\ 6.\ \sigma(\pi_r) = l^r_j \text{ and } \sigma'(\pi_r) = l^r_k \end{cases}$$

Example: TTA Startup Algorithm

Above formalization of the calendar based model can be illustrated using the TTA startup algorithm. TTA startup executes on a logical bus meant for safety-critical applications in both automotive and aerospace industries. In a normal operation, N computers or nodes share a TTA bus using a TDMA schedule. 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. For detailed understanding of startup protocol, we refer the reader to [15].

Figure 2 depicts the calendar based clockless predicate transition diagram of the i^{th} node. In the TTA startup algorithm, all the communications are asynchronous and message delivery delays, which are finite and specified by the designer, have to be taken into account for correct operation of the protocol. τ_i^{listen} and τ_i^{cs} represent how much time a node spends in the *Listen* state and the *Coldstart* state respectively, if no external signal is received. τ^{round} denotes the time a node spends in the *Active* state before sending its next massage. $R = \{1, \ldots, N\} \setminus \{i\}$ represents that all the nodes except the sender i are required to receive the message in the network. λ_i's denote the message delivery time for the corresponding send events. In the TTA, message delivery times for all the receivers are considered to be the same, and that is why we have considered a single variable λ_i to represent that delay.



$(\tau_i = 0) \Rightarrow \langle \tau_i := \tau_i^{listen} \rangle$
$(\tau_i = 0) \Rightarrow \langle$send(*cs_frame*, i, R, λ_1), $\tau_i := \tau_i^{cs} \rangle$
Init
Listen
ColdStart
True$\Rightarrow \langle$receive(i_*frame*, j, i), $\tau_i := \tau^{round} \rangle$
True$\Rightarrow \langle$receive(cs_*frame*, j, i), $\tau_i := \tau_i^{cs} \rangle$
$(\tau_i = 0) \Rightarrow \langle$send(*cs_frame*, i, R, λ_1), $\tau_i := \tau_i^{cs} \rangle$
True$\Rightarrow \langle$receive(i_*frame*, j, i), $\tau_i := \tau^{round} \rangle$
Active
True$\Rightarrow \langle$receive(*cs_frame*, j, i), $\tau_i := \tau^{round} \rangle$
$(\tau_i = 0) \Rightarrow \langle$send(i_*frame*, i, R,$\lambda_2$), $\tau_i := \tau^{round} \rangle$

Fig. 2. Clockless model for the i^{th} processor in TTA Startup algorithm.

4 Models for Time

It remains unspecified as to what the underlying model of time is (for clock, timeouts, timing variables etc.) while defining the clockless semantics of the timeout and calendar based models.

The choice of dense ($\mathcal{R}^+$) domain versus discrete ($\mathcal{N}_0$) domain critically affects the size of the state space of the model. Indeed, with the dense time model, we need to add the following nonzenoness condition to ensure effective progress in the model: *There must not be infinitely many time progress (or timeout increment) transitions effectively within a finite interval.* Formally,

Nonzenoness: Clockless Semantics:

$$\neg[\exists\ \sigma_0\sigma_1\ldots \text{ s.t. } \exists\ \delta \in \mathcal{R}^+ \text{ and } \Sigma_{i=0}^{\infty} \min\{\sigma_i(\mathcal{T})\} \leq \delta]$$

Another point to note is that clockless semantics reduces infinite state transition system to a finite state transition system only in case of the choice of discrete domain for the clock and timeout variables. This is because for the dense domain, clockless semantics can only limit unbounded set $\mathcal{R}^+$ to a bounded interval. Nonetheless, verification of a real-time system in a dense domain is equivalent to verifying the system in the discrete domain if the behavior of the system captured by the model and the properties considered are digitizable [8]. It can be shown that if we restrict the update function to *weakly constrained* intervals (e.g., $update_i = k_1 \mid k_2 \mid \infty \mid \max(\mathcal{M})$, where $k_1 \in [l, m]$ and $k_2 \geq l$) then similar to the timed transition system of [8] (refer theorem 2), transition systems for timeout and calendar based models also give rise to digitizable behaviors (computations). Also for qualitative properties like the safety and liveness properties, their verification in the discrete domain is equivalent to verifying these properties in dense domain (refer to proposition 1 in [8]).

5 Experimental Results

In this section, we report experimental results of verification of TGC and the TTA startup algorithm using the model checker Spin. We carry out our experiments on an Intel (R) P4 machine with 2.60 GHz speed and 1 GB RAM, and running Windows 2000.

Train-Gate Controller

For the TGC example as discussed before, we consider safety and timeliness properties for verification.

The safety property considered is: "When the Train crosses the line, the Gate should be down". The property can be expressed in LTL as follows:

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

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

The timeliness property considered states that the time between two states in execution will by bounded by a particular value. We can find many timeliness properties in this example. We mention one of them here: "The time between the transmission of the *approach* signal by the Train and when the Gate is down 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 to *true*. When the second event happens, $flag_2$ is set to *true* and $flag_1$ is reset to $false$. Also, the proctype time_progress is modeled as follows:

```
proctype time_progress () {
  do
  :: timeout ->
  atomic {
      Find out the minimum of all the timeout values
      Subtract the minimum value from all the timeouts
      if
      :: flag1 == true ->
            time_diff = time_diff + min_timeout;
      :: flag2 == true ->
            flag2 = false;
            time_diff = 0;
      :: else ->
      fi;
  }
  od
}
```

A global variable *time_diff* (initially set to 0) captures the time difference between the instants when these two flags are set. During every discrete transition between the two discrete transitions of interest, minimum timeout value is added to *time_diff*. The property is specified as:

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

Table 1 illustrates computational resources and time required to prove the safety and the timeliness property for TGC. Both the properties have been proved by exhaustive verification keeping the option of partial order reduction turned on.

TTA Startup Algorithm

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". For

Properties	States stored	States matched	Transitions usage	Total actual memory (in MB)	Time (in seconds)
Safety	246236	422596	668832	19.531	6
Timeliness	253500	415484	668984	21.988	6

Table 1. Computational resources and time required for verification of the Train-Gate Controller

two nodes participating in the startup process, the corresponding LTL property is given below:

$$\Box((p_0 \wedge p_1) \wedge (q_0 \wedge q_1) \ \rightarrow \ \Diamond(r \wedge s))$$

where, $p_0 \equiv (pc[0] = state\_active)$, $p_1 \equiv (pc[1] = state\_active)$, $q_0 \equiv (time\_out[0] > 0)$, $q_1 \equiv (time\_out[1] > 0)$, $r \equiv (time\_out[0] = time\_out[1])$, and $s \equiv (slot[0] = slot[1])$. $pc[i]$ 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 Safety property ensures that when the nodes are in the *active* state, then they are indeed synchronized. But it does not answer the question whether all the nodes will eventually be synchronized or not. To ensure that all the nodes will eventually be synchronized, it has to be specified in the form of a liveness property: "Eventually all the nodes will be in the *active* state and remain so". This 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 the TTA startup we used clockless modeling together with the options of exhaustive verification and bitstate hashing technique offered by Spin, in both the cases keeping the the option of partial order reduction turned on. By exhaustive verification technique, the safety property can be verified for the TTA models with upto 5 nodes and liveness property can be verified upto 4 nodes. Bitstate hashing enables us to verify both the properties for models with upto 9 nodes. For 10 nodes, the verification does not terminate even in 4 hours.

Table 2 describes the computational resources and time required to prove the safety and liveness properties for the TTA Startup protocol using bitstate hashing technique.

Experiments with dense time modeling with clockless reduction using SAL were also carried out on the TGC model presented in [6]. The safety property has been verified at depth 14 as done in [6]. Nonetheless, applying the clockless reduction in SAL models do not scale up the existing results further, primarily because the clockless reduction even though reduces the unbounded set $\mathcal{R}^+$ to a bounded interval, such an interval still will contain uncountably many points giving rise to infinite many possible execution paths of finite lengths.

Properties	No of nodes	States stored	States matched	Transitions	Total actual memory usage (in MB)	Time
Safety	2	487	143	630	8.914	6 sec
	3	6142	6490	12632	8.914	7 sec
	4	123452	253057	376509	8.914	7 sec
	5	3.31158e+06	1.03436e+07	1.36552e+07	8.914	47 sec
	6	1.59195e+07	5.93261e+07	7.52457e+07	9.016	3 min
	7	3.44457e+07	1.29191e+08	1.63636e+08	9.016	8 min
	8	4.01727e+07	2.43036e+08	2.83209e+08	9.016	16 min
	9	4.10158e+07	1.29835e+09	1.33936e+09	9.016	97 min
Liveness	2	1445	1036	2481	8.914	7 sec
	3	16582	21980	38562	8.914	7 sec
	4	305893	677235	983128	8.914	8 sec
	5	7.39657e+06	2.38099e+07	3.12064e+07	8.914	1 min
	6	2.73472e+07	1.09554e+08	1.36901e+08	8.914	8 min
	7	3.83552e+07	2.0233e+08	2.40685e+08	9.016	14 min
	8	4.07954e+07	3.47211e+08	3.88006e+08	9.016	24 min
	9	4.10157e+07	2.30705e+09	2.34807e+09	9.016	163 min

Table 2. Computational resources and time required to verify safety and liveness property by bitstate hashing technique in Spin for the TTA Startup

6 Conclusion

In this work we proposed a canonical finitary reduction technique formalized in terms of clockless modeling and associated semantics, which renders timeout and calendar based models of real-time systems amenable to finite state verification. There exists an equivalence between the corresponding discrete and the dense domain verifications for the qualitative safety and liveness properties considered in this work on the weakly constrained models assuming discrete dynamics. Verification of safety properties for the TTA start up protocol consisting of upto 9 nodes demonstrates the effectiveness of the reduction technique as compared to the dense time modeling and verification results reported in literature [7], which rely on additional efforts to find out appropriate supporting lemmas and abstractions to scale up the model.

Dynamic rescheduling of timeouts to deal with interrupts can further extend the current framework in order to model hardware-software co designs and preemptive scheduling type of scenarios. Work can also be further extended by considering shared timing variables, deadlines to capture urgencies [3], and by considering the timeout update rules, which include the possibility of interactive updation to deal with game theoretic properties.

References

1. R. Alur and D. L. Dill. "A Theory of Timed Automata". In *Theoretical Computer Science*, Vol. 126, No. 2, pp. 183-236, 1994.

2. G. Behrmann, A. David and K. G. Larsen. “A Tutorial on UPPAAL”. In *4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM-RT'04)*, LNCS 3185, Springer-Verlag, pp. 200–236, 2004.
3. S. Bornot, J. Sifakis, and S. Tripakis. “Modeling Urgency in Timed Systems”. In *Compositionality: the Significant Difference (COMPOS'97)*, LNCS 1536, Springer-Verlag, pp. 103-129, 1997.
4. D. Bošnački and D. Dams. “Integrating Real Time into Spin: A Prototype Implementation”. In *Proceedings of the Formal Description Techniques and Protocol Specification, Testing and Verification (FORTE/PSTV)*, Kluwer, pp. 423-439, 1998.
5. D. Bošnački and D. Dams. “Discrete-Time PROMELA and Spin”. In *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98)*, LNCS 1486, Springer, pp. 307-310, 1998.
6. B. Dutertre, M. Sorea. “Timed Systems in SAL”. Technical Report, Computer Science Laboratory, SRI Internationalhhe, 2004.
7. B. Dutertre and M. Sorea. “Modeling and Verification of a Fault-Tolerant Real-Time Startup Protocol using Calendar Automata”. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04)*, LNCS 3253, Springer-Verlag, pp. 199-214, 2004.
8. T. A. Henzingerz, Z. Manna, and A. Pnueli. “What Good Are Digital Clocks?”. In *Proceedings of the 19th International Colloquium on Automata, Languages, and Programming (ICALP'92)*, LNCS 623, Springer-Verlag, pp. 545-558, 1992.
9. G. J. Holzmann. “The SPIN Model Checker, Primer and Reference Manual”. Addison-Wesley, 2003.
10. L. M. Moura, S. Owre, H. Rue, J. M. Rushby, N. Shankar, M. Sorea, and A. Tiwari. “Sal 2”. In *Proceedings of International Conference on Computer-Aided Verification (CAV'04)*, LNCS 3114, Springer, pp. 496-500, 2004.
11. L. Pike. “Real-Time System Verification by k-Induction”. Technical report, NASA Langley Research Center. TM-2005-213751, 2005. Available at http://www.cs.indiana.edu/ lepike/pub_pages/reint.html.
12. J. Rushby. “Verification Diagrams Revisited: Disjunctive Invariants for Easy Verification”. In *Proceedings of International Conference on Computer-Aided Verification (CAV'00)*, LNCS 1855, Springer-Verlag, pp. 508-520, 2000.
13. I. Saha and S. Roy. “A Finite State Modeling of AFDX Frame Management using Spin”. In *Proceedings of the 11th International Workshop on Formal Methods for Industrial Critical Systems (FMICS'06)*, LNCS 4346, Springer-Verlag, pp. 227-233, 2006.
14. I. Saha and S. Roy. “A Finite State Analysis of Time-triggered CAN (TTCAN) Protocol using Spin”. In *Proceedings of the International Conference on Computing: Theory and Application (ICCTA'07)*, IEEE Computer Society, pp. 77-81, 2007.
15. W. Steiner and M. Paulitsch. “The Transition from Asynchronous to Synchronous System Operation: An Approach for Distributed Fault- Tolerant System”. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, IEEE Computer Society, pp. 329-336, 2002.
16. S. Tripakis and C. Courcoubetis. “Extending PROMELA and Spin for Real Time”. In *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'96)*, LNCS 1055, Springer-Verlag, pp. 329-348, 1996.