2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)

Optimized Step Semantics Encoding for Bounded Model Checking of Timed Automata

Zuxi Chen
School of Computer Science and Technology, Huaqiao University
Fujian, China
zuxichen@hotmail.com

Huixing Fang
School of Information Engineering
Yangzhou University
Jiangsu, China
fanghuixing@gmail.com

Xiangyu Luo
School of Computer Science and Technology, Huaqiao University
Fujian, China
luoxy@hqu.edu.cn

***Abstract*—To BMC of timed automata network, we present a novel time stamp semantics model for timed automata network with synchronization and shared variables, which allows not only mutually independent transitions but also dependent transitions to be compressed together between two states in succession. A key ingredient of our BMC encoding is the use of time stamp variables for shared variable accesses, which are overlooked in previous approaches. The proposed semantics represents the timed automata network in a significantly more compact way than previous step semantics, which allows maximally compressed steps of transitions and therefore is in this sense optimal. A preliminary experimental evaluation shows a significant performance improvement in the number of unrolling of BMC steps and run times as well.**

***Keywords*-Bounded model checking, timed automata network, timed stamp semantics, concurrent real-time systems**

I. Introduction

Timed automata (TA) [1] are automata augmented with continuous variables, clocks, to specify temporal behaviour. While decision procedures for model checking and reachability have been known from the beginning, it took a lot of research in practical algorithms to obtain usable tools, notably [2]. Even so, often these tools will fail to provide an answer due to complexity reasons.

Consequently, research has considered adapting the highly popular bounded model checking (BMC) approach [3] to the timed automata network (TAN), see e.g. [4]–[7] for some early approaches. These approaches are based on the coding of discrete systems with the additional need to deal with the coding of continuous variables, clocks, achieved in different ways. From a methodological point of view, BMC separates model checking into two separate tasks, coding a problem for a solver, e.g. SAT or SMT (SAT modulo theory) and solving it with the solver. The basic idea of BMC is to code execution sequences of automata as an alternation of states and transitions by distinct variables for every position in the sequence and to code logically the relations between these variables (e.g. the transition relation). A solution can then be interpreted in the semantics of the modeled system.

The coding can produce more or less difficult problems for the solver and the difficulty can unfortunately not simply be measured in the size of the formula. In [8], the idea of using step semantics for improved coding was developed: the coding allows several transitions to be executed in parallel, provided they are independent (any possible order leads to the same state), thus effectively compressing the sequence. Apart from reducing the size of the formula, this also makes it easier to be solved, since it remove the artificial choice of the execution order of independent actions.

Independence of transitions for TAN has been studied beyond the context of BMC, see e.g. [9]–[11]. In various ways, these works relax the impact of timing constraints on the order of otherwise independent actions. E.g. [10] allows sequences with time running sometimes backwards between independent actions, since it is ensured that these sequences can be rearranged into a temporally correct order.

It was natural to try to adapt step semantics to this setting, but it is not obvious how to code the occurrence of multiple independent transitions "in parallel" when they are restricted by time. While [5] limited this possibility to the case of transitions occurring at the same time, [12] proposes (in a SAT setting, rather than an SMT setting) to relax this limitation with alternate codings with mixed results: for the most relaxed coding, sequences do get shorter, but the size of the coding exploses and is difficult to solve.

In this work, we propose an efficient SMT encoding for reachability of TAN with local clocks and communication both via synchronization and shared variables. It combines several optimizations of previous codings, e.g. it combines transition occurrences with time passage as in [13] and, for a single process, it follows the coding in [4]. Based on the previous methods, we make the following several novel contributions: 1). A novel time stamp semantics is given to eliminate the implicit clock synchronization and to allow the resulting composed transitions to be executed in the same way as transitions of untimed systems; 2). Most importantly, in the time stamp semantics, time stamps for last read and last write accesses to shared (global) variables are introduced to keep track on the order relations between dependent transitions, and hence the correct execution order of transitions is guarantied; 3). We presented an optimal

978-1-7281-3342-3/19/$31.00 ©2019 IEEE
DOI 10.1109/TASE.2019.00020

encoding for BMC of TAN, the timing constraints of transitions accessing the same shared variables are encoded by the bias of these time stamps. Going further, several accesses to a shared variable in parallel are possible as long as time stamps coherent with that access can be attributed. In this sense, the coding provides an optimal compression.

Time stamps for shared, discrete variables are a non obvious choice, but their merit becomes clear when considering the alternative coding in [12] based on time stamps for transition occurrences: to code dependency between these transitions, a quadratic explosion of constraints occurs, whereas the time stamps of the entities causing the dependency, the shared variables, allow for a concise coding.

The rest of this paper is organized as follows: The formal definition of TAN with its novel time stamp semantics are given in Section II. The BMC problem of TAN in terms of the time stamp semantics is coded in Section III. The experimental results are presented in Section IV.

II. Timed Automata Networks

A. Timed Automaton

A TA [1] is state transition graph augmented with a finite set of clock variables that can be used to constrain the time between certain transitions. This is achieved using a set of positively valued variables called clocks $C = \{x_1, \ldots, x_n\}$, where atomic clock constraint is any expression in the form $x \sim c$ with $\sim \in \{\geq, >, =, <, \leq\}$, $x \in C$ being clock variable and $c \in \mathbb{Z}$ being a constant, we call the set of conjunction of atomic clock constraints as $B(C)$. The subset of $B(C)$ where $\sim \in \{<, \leq\}$ is denoted by $I(C)$.

Definition 1: A TA is a tuple (L, l_0, C, E, Inv), where L is a finite set of locations; $l_0 \in L$ is an initial location; C is a finite set of clocks; $Inv : L \to I(C)$ assigns an invariant to each location; $E \subseteq L \times B(C) \times 2^C \times L$ is a set of transitions; Every transition $e = (l, g, R, l')$ contains source and target location $l, l' \in L$, a clock constraint $g \in B(C)$, and a set R of clock reset condition in the form of $x := 0$ ($x \in C$).

The dynamic behaviours of TA is given by means of a transition system TS. A configuration of TS is a tuple $c = (l, r, t)$ where $l \in L$ is a location, $r : C \mapsto \mathbf{R}^+$ is a clock reset function over C s.t. $r(x)$ denotes the last reset time of clock x, and t is called timed stamp, which represents the time elapsed from the beginning of the path being analyzed. Throughout this paper, we attribute a clock function $v_r^t : C \mapsto \mathbf{R}^+$ to a reset function r and a given time stamp t such that $v_r^t(x) = t - r(x)$. As a result, $v_r^t(x)$ denotes the value of clock x w.r.t. time stamp t. We write $v \models \varphi$ when v satisfies $\varphi \in B(C)$. For a function r, the modified function $r[R = t]$ denotes a function r' such that $r'(x) = t$ for $x \in R$ and $r'(x) = r(x)$ for $x \notin R$.

Definition 2: The time stamp semantics of a TA $TA \doteq (L, l_0, C, E, Inv)$ is defined by a transition system $TS \doteq (S, c_0, \to)$ where $S = L \times \mathbf{R}^{+C} \times \mathbf{R}^+$ is the set of configurations, $c_0 = (l_0, r_0, 0)$ is the initial configuration, $r_0(x) = 0$ for all $x \in C$, and $\to \subseteq S \times S$ is the set of transitions defined as follows: 1). $(l, r, t) \xrightarrow{e} (l', r', t)$ for $e = (l, g, R, l')$ such that $v_r^t \models g$, $r' = r[R = t]$ and $v_{r'}^t \models Inv(l')$; 2). $(l, r, t) \xrightarrow{d} (l, r, t + d)$ for $d \in \mathbf{R}^+$ such that $v_r^{t+d} \models Inv(l)$.

Intuitively, 1) represents an action leaving the current location can be fired if the current clock values satisfy its guard and, after resetting, the invariant of target location holds; 2) denotes time elapse does not switch the current location, but it is allowed only when the location invariant does not violated. In our encoding we represent the paths in a unified way, as interleaving sequences of time steps and actions as $(c_0) \xrightarrow{d_0} (c_1) \xrightarrow{e_1, d_1} (c_2) \ldots \xrightarrow{e_{n-1}, d_{n-1}} (c_n)$.

B. Timed Automaton Network

In this work, TAN consists of a set of TAs that may synchronize on 1) synchronization actions, 2) shared bounded integer variables. For the sake of presentation, we limit ourselves to a set $\mathcal{V}$ of bounded integer shared variables and a set $\mathcal{G}(\mathcal{V})$ of constraints in the form of $\nu \sim c$, where $\nu \in \mathcal{V}$, c is an integer. $\mathcal{S}(\mathcal{V})$ is used to represent the set of assignments in the form of $\nu' := a\nu + c$, where ν, ν' are variables in $\mathcal{V}$ and a, c are integers.

Definition 3 (Timed Automata Network): A TAN is a tuple $TAN = (\mathcal{V}, \Sigma, TAS)$ where: 1). $\mathcal{V} = \{\upsilon_0, \upsilon_1, \ldots \upsilon_m\}$ is a finite set of shared integer variables, each variable $\upsilon_i \in \mathcal{V}$ has a finite value domain $[L_{\upsilon i}, U_{\upsilon i}]$; 2). $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$ is a set of synchronization actions with Σ_i is the subset that appears in TA_i; 3). $TAS \doteq \{TA_1, TA_2, \ldots, TA_n\}$ is a set of TAs with synchronization actions and shared variables. Each $TA_i = (L_i, l_{0i}, \Sigma_i, C_i, E_i, Inv_i)$ with $i \in \{1, \ldots, n\}$ is as defined in Definition 1 except E_i. Here, $E_i \subseteq L_i \times B(C_i) \times 2^{C_i} \times \Sigma_i \cup \bot \times 2^{\mathcal{G}(\mathcal{V})} \times 2^{\mathcal{S}(\mathcal{V})} \times L_i$.

Besides the clock guard and clock resetting, a transition of a TA in a network may optionally have a synchronization action in $\Sigma \cup \bot$, a set of shared variable guards in $\mathcal{G}(\mathcal{V})$ and a set of shared variable assignments in $\mathcal{S}(\mathcal{V})$.

A TAN $TAN = (\mathcal{V}, \Sigma, TAS)$ evolves as the TAs in TAS evolve. A global configuration of the network is a triples $(\bar{l}, s, r, \bar{t}, \tau^w, \tau^r)$, where $\bar{l}$ is a vector of locations, function s denotes the valuation of shared variables in $\mathcal{V}$, function r assigns each clock x in $\bigcup_{i=1}^{n} C_i$ to its last reset time $r(x)$, $\bar{t}$ is a vector of time stamp, function τ^w assigns each ν in $\mathcal{V}$ to its last write time $\tau^w(\nu)$ and function τ^r assigns each ν in $\mathcal{V}$ to its last read time $\tau^r(\nu)$. Given a global configuration $(\bar{l}, s, r, \bar{t}, \tau^w, \tau^r)$, $\bar{l}_i$ is used to represent the local location of $\bar{l}$ w.r.t. TA_i and each TA_i has its own time stamp $\bar{t}_i$.

Definition 4: The time stamp semantics of an TAN $TAN = (\mathcal{V}, \Sigma, TAS)$ is a triple $TS \doteq (S, c_0, \to)$ where $S = L \times \mathbb{Z}^{\mathcal{V}} \times \mathbf{R}^{+\bigcup_i C_i} \times \mathbf{R}^{+TAS} \times \mathbf{R}^{+\mathcal{V}} \times \mathbf{R}^{+\mathcal{V}}$ is the set of configurations, $c_0 = (\bar{l}_0, s_0, r_0, \bar{t}_0, \tau_0^w, \tau_0^r)$ is the initial configuration, $r_0(x) = 0$ for all $x \in \bigcup_i C_i$, $\bar{t}_0(i) = 0$ for each TA_i, $\tau_0^w(\nu) = \tau_0^r(\nu) = 0$ for any $\nu \in \mathcal{V}$, and $\to \subseteq S \times S$ is

the set of transitions with the following three types: **1) Local time passing**: $e \doteq (\bar{l}, s, r, \bar{t}, \tau^w, \tau^r) \xrightarrow{d}_i (\bar{l}, s, r, \bar{t}', \tau^w, \tau^r)$ for $\bar{t'}_i = \bar{t}_i + d$ and $\bar{t'}_j = \bar{t}_j$ for $i \neq j$ (time advances for TA_i and $v_r^{\bar{t'}_i}$ satisfies the invariant of $\bar{l}_i$); **2) Local transitions**: $e \doteq (\bar{l}, s, r, \bar{t}, \tau^w, \tau^r) \xrightarrow{\perp} (\bar{l}', s', r', \bar{t}, \tau^{w\prime}, \tau^{r\prime})$ where $e_i = (l, g, R, \perp, s_\nu, g_\nu, l')$ be a transition of TA_i, such that (a) $\bar{l}_i = l$, $\bar{l'}_i = l'$, and $\bar{l'}_j = \bar{l}_j$ for $j \neq i$; (b) $v_r^{\bar{t}_i}$ satisfies the guard g and $r' = r[R = \bar{t}_i]$; (c) s satisfies the variable guard g_ν and $s' = s[s_\nu]$; (d) if $\nu \in var(s_\nu)$ then $\tau^r(\nu) \leq \bar{t}_i$ and $\tau^w(\nu) \leq \bar{t}_i$; for any $\nu \in \mathcal{V}$, if $\nu \in var(g_\nu)$ then $\tau^w(\nu) \leq \bar{t}_i$. Here, $var(exp)$ denotes the set of variables that appear in expression exp.(e) $\tau^{r\prime}(\nu) = \max(\bar{t}_i, \tau^r(\nu))$ if $\nu \in var(g_\nu)$, and otherwise $\tau^{r\prime}(\nu) = \tau^r(\nu)$; Moreover, $\tau^{w\prime} = \tau^w[var(s_\nu) = \bar{t}_i]$; **3) Synchronized transitions**: $e \doteq (\bar{l}, s, r, t, \tau^w, \tau^r) \xrightarrow{a} (\bar{l}', s', r', t, \tau^{w\prime}, \tau^{r\prime})$, for each i with $a \in \Sigma_i$, there must be an edge $e_i = (l_i, g_i, r_i, a, s_{\nu_i}, g_{\nu_i}, l'_i) \in TA_i$ such that (a) $\bar{l}_i = l_i$ and $\bar{l}'_i = l'_i$ and the general conditions for local transitions are also satisfied; (b) the participating automata are synchronized, i.e. $\bar{t}_i = \bar{t}_j$ with $\Sigma_i \bigcap \Sigma_j = a$.

Intuitively, the rule 1) says that a component may advance its local time as long as the local invariant holds. The rule 2) is the standard interleaving rule for discrete transition. The usual cases of read-write and write-write dependencies appear when a variable is shared for communication by several automata of the network. To ensure the correct semantics, we have to ensure dependent transitions are serialized with respect to their occurrence in time. This is specified by the condition d) and e) of the rule 2). When several components need to be synchronized, each transition of the participating automata must satisfies condition (a)-(e) of rule 2) and must also be checked if they have executed for the same amount of time by condition (b) of the rule 3).

In [12], mutually independent transitions are allowed to occur together between two consecutive configurations, thus allowing compressed "step" executions $(s_0) \xrightarrow{MT_1} (s_1) \ldots \xrightarrow{MT_n} (s_n)$ where $MT = \{(e_1), \ldots, (e_n)\}$ is a set of pairwise independent events. This allows BMC to decrease the bound required to find counter-examples. As a result, it can be quite beneficial to recognize more pair of transitions as independent transitions. In general, two transitions from different automata are called independent if neither disables nor enables the execution of the other, and the same state is reached by executing them in either order. To be practically useful, we present a sufficient checkable syntactic conditions for independence of two transitions for the timed stamp semantics of definition 4.

Theorem 1: Two transition $e_1, e_2 \in \rightarrow$ that satisfies the following properties are independent: 1). The set of automata participate in e_1 is disjoint from the set of automata participate in e_2 ; and 2). The set of shared variables that are accessed by e_1 is disjoint from the set of shared variables that are accessed by e_2, ***or*** both e_1 and e_2 are read to the same shared variable, ***or*** both e_1 and e_2 write to each shared variable the same value, ***or*** both e_1 and e_2 enable in configure c and the value of the shared variable write by e_2 equals to the value of the shared variable in configuration c.

Here, more pair of transitions are recognized as independent than previous work [12], especially the pair of transitions that access to the same shared variable with at least one of transitions write to the shared variable, as long as the two transitions write or read to the shared variable with the same value. As a result, we achieve a significant reduction of execution steps for BMC.

III. Encoding of Timed Automata Network

This section presents how to code a given TAN under time stamp semantics as a difference logic formula for BMC. To this end, a symbolic representation $M = (\mathbf{V}, I, R)$ of TAN is defined, where $\mathbf{V}$ denotes the set of state (configuration) variables, $\mathbf{V}' = \{v' \mid v \in \mathbf{V}\}$ and $\mathbf{V}_i = \{v_i \mid v \in \mathbf{V}\}$ refer to the next state and the i-th step state variables, respectively; $R(\mathbf{V}, \mathbf{V}')$ is a transition relation which relates value of variables in the current step to their value in the next step; I is a formula that represents the set of initial states. In particular, when P is a safety property, whether P can be violated in k steps is reduced to checking satisfiability of the formula as follows.

$$I(\mathbf{V}_0) \wedge R(\mathbf{V}_0, \mathbf{V}_1) \wedge \ldots \wedge R(\mathbf{V}_{k-1}, \mathbf{V}_k) \wedge \neg P(\mathbf{V}_k) \quad (1)$$

A. Encoding Single Timed Automaton

In order to code $TA \doteq (L, l_0, C, E, Inv)$ according to its time stamp semantics $TS \doteq (S, c_0, \rightarrow)$, we define the symbolic representation $M_{TA} = (\mathbf{V}, I, R)$ of TA with $\mathbf{V} = OC \cup loc$. Here, OC contains a real variable ox for each clock variable $x \in C$ and a special real variable τ, the values of variable ox and variable τ correspond to the valuation of $r(x)$ and time stamp t of a configuration (l, r, t) in TS, respectively. loc is a finite domain variable with domain L, we use $loc = l$ denotes the TA is resided in location l.

For a TA, the initial constraint $I(\mathbf{V})$ is coded as following

$$I(\mathbf{V}) \doteq loc = l_0 \wedge \bigwedge_{x \in C}^{m} ox = 0 \wedge \tau \geq 0 \wedge inv_{l_0}(\mathbf{V}) \quad (2)$$

The formula denotes the initial state followed by an arbitrary amount of delay without violating the invariant of l_0.

To code transition realtion R, we begin with keeping track of the evolvement of clocks while an edge is switched (followed by a timed delay) as follows:

$$a_{ce}(\mathbf{V}, \mathbf{V}') \doteq \bigwedge_{x \in r} ox' = \tau \wedge \bigwedge_{x \notin r} ox' = ox \wedge \tau' - \tau \geq 0 \quad (3)$$

Hence, the edge $e \doteq (l_0, g, r, l_1) \in E$ evolves from a location l_0 to l_1, thereafter followed by an arbitrary amount

of time elapses while the corresponding automata remaining in location l_1 can be represented as follows:

$$T_e(\mathbf{V},\mathbf{V}') \doteq loc = l_0 \wedge g_e(\mathbf{V}) \wedge a_{ce}(\mathbf{V},\mathbf{V}') \wedge loc' = l_1 \quad (4)$$

The above constraint $g_e(\mathbf{V})$ which is obtained from guard g by replacing each constraint of the form $x \bowtie c$ and $x-y \bowtie c$ respectively by $\tau - ox \bowtie c$ and $oy - ox \bowtie c$. In addition, for any time elapse we also has to ensure the location's invariant to be held. This is represented by

$$INV(\mathbf{V}) \doteq \bigwedge_{l \in L} loc = l \Rightarrow inv_l(\mathbf{V}) \quad (5)$$

Here, $inv_l(\mathbf{V})$ is obtained from the invariant inv of location l in the same way as $g_e(\mathbf{V})$ is obtained from its corresponding guard g. Finally, the transition relation $R(\mathbf{V},\mathbf{V}')$ that characterize a discrete step execution followed by a time elapse is represented as:

$$R(\mathbf{V},\mathbf{V}') \doteq \bigvee_{e \in E} T_e(\mathbf{V},\mathbf{V}') \wedge INV(\mathbf{V}') \quad (6)$$

Thanks to the composed transition coding of T_e, we can construct the transition relation formula (6) as a disjunction of T_e of all the edge $e \in E$ in TA.

B. Encode TAN using Time Stamp Semantics

Given a network $TAN = \{\Sigma, \mathcal{V}, TAS\}$, where each $TA_i = (L_i, l_{0i}, \Sigma_i, C_i, E_i, Inv_i) \in TAS$ is associated with a symbolic representation $TS_i = (\mathbf{V}_i, I_i, T_i)$. In order to respect the parallel composition requirement in the Definition 4, the symbolic representation $TSN = (\mathbf{V}, \Sigma, I, T)$ of the networks is obtained by setting $\mathbf{V} = \bigcup_{TA_i \in TAS} \mathbf{V}_i \cup \mathcal{V}$, where the variable set $\mathcal{V}$ and Σ is used to code the shared variables and synchronization actions, respectively.

The main idea of the time stamp semantics based coding scheme is to introduce a local time scale for each $TA_i \in TAS$ and then rebuild the order relation between dependent transitions from different automata. For each TA_i, the local time scale variable τ_i indicate the absolute time since the beginning of run of TA_i. As a result, the initial constraints formula for TAN can be easily rewritten as follow:

$$\begin{aligned} I(V) \doteq & \bigwedge_{i=1}^{n} (loc_i = l_{i0} \wedge \bigwedge_{x \in C_i} ox = 0 \wedge \tau_i \geq 0 \\ & \wedge \bigwedge_{x \in C} ox = 0 \wedge inv_{l_{i0}}(V)) \wedge \bigwedge_{v \in \mathcal{V}} v = v_{init} \end{aligned} \quad (7)$$

Here, v_{init} denotes the initial value of shared variable $v \in \mathcal{V}$.

To allow several independent actions from different automata which happen at different instant to be executed in a single step, the transition formula $\hat{T}_e$ associate with edge $e = (l, g, r, a, s_\nu, g_\nu, l')$ of TA_i of the network NTA is coded similar to formula (4) except that the timed scale τ is replaced by the corresponding τ_i and three additional constraints ψ^e_{syn} and ψ^e_s are constructed to ensure that not only each automaton appropriately communicates with each other, but also the dependent transitions are serialized in the order of their occurrence in time which is destroyed by the local time representation. Hence, the corresponding transition is encoded as $\hat{T}_e \doteq T_e \wedge \psi^e_{syn} \wedge \psi^e_s$.

$$\begin{aligned} T_e \doteq & loc_i = l \wedge g_e(\mathbf{V}) \wedge \bigwedge_{x \in r} ox' = \tau_i \\ & \wedge \bigwedge_{x \notin r} ox' = ox \wedge \tau'_i - \tau_i \geq 0 \wedge loc'_i = l' \end{aligned} \quad (8)$$

For the coded formula T_e, the enabling of edge e affects only the clocks of that automaton in its local-time scale, and the clocks of other automata are unaffected, the execution order of transitions is relaxed. However, in order to explore the pair of interacting actions, we must also "rebuild" the necessary order relations of the participating automata. To serve this purpose, for each synchronization between TA_i and TA_j, we must add the condition that the reference clocks (time stamps) of TA_i and TA_j are equal as follows.

$$\psi^e_{syn} \doteq a \wedge \bigwedge_{a' \in \Sigma_i \setminus a} \neg a' \wedge \bigwedge_{j \neq i, a \in \Sigma_j} \tau_i = \tau_j \quad (9)$$

In general, if a variable is shared by several components of the network, the usual cases of read-write and write-write dependencies appear. To ensure that write to the shared variable is serialized with respect to both reads and other writes to the same variable in the order of their occurrence in time, i.e., if transition e_i and e_j in an execution $\ldots, (e_i, \tau_i), \ldots, (e_j, \tau_j), \ldots$ are in conflict with respect to some variable ν, then $i < j$ iff $\tau_i < \tau_j$, the following formula ψ^e_s is added

$$\begin{aligned} \psi^e_s \doteq & \bigwedge_{\nu \in var(s_\nu)} \{w_{\nu,i} \wedge (\nu' = c_\nu) \wedge (\tau_i > w\tau_\nu) \wedge (\tau_i > r\tau'_\nu)\} \\ \wedge & \bigwedge_{\nu \notin var(s_\nu)} \neg w_{\nu,i} \wedge \bigwedge_{\nu \in var(g_\nu)} \{r_{\nu,i} \wedge (\nu = c_\nu) \wedge (\tau_i > w\tau_\nu)\} \\ & \wedge \bigwedge_{\nu \notin var(g_\nu)} \neg r_{\nu,i} \wedge \bigwedge_{i=1}^{n} \bigwedge_{j=1, j \neq i}^{n} \{(w_{\nu,i} \wedge r_{\nu,j}) \Rightarrow \tau_i \geq \tau_j\} \end{aligned} \quad (10)$$

In the above formula (10), for each shared variable ν, we first introduce two new variables $w\tau_\nu$ and $r\tau_\nu$, denoting, respectively, the timepoints of the last read and write to variable ν are executed. Then, we also introduce two boolean variables $w_{\nu,i}$ and $r_{\nu,i}$ to represent respectively wether a shared variable ν is written and read by some transition of TA_i or not. To make sure the automata in the TAN communicate with each other normally, the following constraints is added: 1) if a transition has write to a shared variable ν, the first row of the formula (10) ensure the boolean variable $w_{\nu,i}$ is true, the shared variable ν obtain the new value c_ν, the timepoint when the transition is executed is greater than the timepoints of the last write and read to the shared variable ν are executed; otherwise, boolean variable $w_{\nu,i}$ is false. 2) if

a transition has read to a shared variable ν, the second row of the formula (10) make sure the boolean variable $r_{\nu,i}$ is true, the transition satisfies the shared variable guard, the timepoint when the transition is executed is greater than the timepoint of the last write to the shared variable ν is executed; otherwise, boolean variable $r_{\nu,i}$ is false. 3) if a transition performs a write and another transition perform a read to the shared variable ν from a different automata at the same step, the third row of the formula (10) guarantee the timepoint when the write is executed is greater than that the read is executed (Since read and write to a shared variable ν at the same step are operations on the current state variables and next state variables, this implies read and write the same shared variable at the same step are not mutual exclusion).

When no edge is executed in TA_i, the formula 11 guarantees its local state remain unchanged, and the its read and write to shared variable is not performed.

$$T_{fix} \doteq (loc' = loc) \wedge \bigwedge_{\nu \in A_e} \neg w_{\nu,i} \wedge \bigwedge_{\nu \in C_e} \neg r_{\nu,i} \wedge \bigwedge_{\alpha \in \Sigma_i} \neg\alpha \wedge \bigwedge_{x \in C} (ox' = ox) \wedge (\tau_i' = \tau_i) \quad (11)$$

In the following formula 12, the first row states the rule of update the value of variable $w\tau_\nu$ when multiple transitions from different automata write to a shared variable ν (Since multiple transitions write the same value to a shared variable are not mutual exclusion). When none of the automata writes access to a shared variable ν, the second row constraint guarantees that shared variable ν remains unchanged, and the timepoint of last write access to shared variable ν also remains unchanged.

$$\begin{aligned} ass \quad \doteq \bigwedge_{i=1}^{n} \{((w_{\nu,i} \wedge (\tau_i > w\tau_\nu)) \wedge \bigwedge_{j=1, j\neq i}^{n} (w_{\nu,j} \\ \Rightarrow \tau_i > \tau_j)) \Rightarrow (w\tau_\nu' = \tau_i)\} \wedge \bigwedge_{\nu \in V} \{(\bigwedge_{i=1}^{n} \neg w_{\nu,i}) \\ \Rightarrow ((\nu' = \nu) \wedge (w\tau_\nu' = w\tau_\nu))\} \end{aligned} \quad (12)$$

Furthermore, the first row of the following formula 13 ensures the timepoint of last read to shared variable ν is the max timepoint that the read transitions are executed at the current step. When none of the automata perform a read operator or the read is performed at timepoint before the last read is executed for a shared variable $\nu \in V$, the second row constraint guarantees that the timepoint of last read to shared variable ν remains unchanged.

$$\begin{aligned} com \quad \doteq \bigwedge_{i=1}^{n} \{((r_{\nu,i} \wedge (\tau_i > r\tau_\nu)) \wedge \bigwedge_{j=1, j\neq i}^{n} (r_{\nu,j} \Rightarrow \tau_i > \\ \tau_j)) \Rightarrow (r\tau_\nu' = \tau_i)\} \wedge \bigwedge_{\nu \in V} \{(\bigwedge_{i=1}^{n} \neg(r_{\nu,i} \wedge (\tau_i > \\ r\tau_\nu)) \Rightarrow (r\tau_\nu' = r\tau_\nu)\} \end{aligned} \quad (13)$$

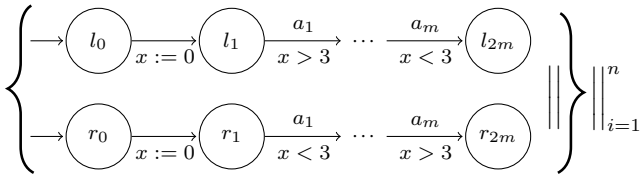

Figure 1. Networks of Chain-like Timed Automata

Again, for any time elapse we have to ensure the location's invariant to be held. This is represented by

$$INV(\mathbf{V}) \doteq \bigwedge_{TA_i \in TAS} \{ \bigwedge_{l \in L_i} loc = l \Rightarrow inv_l(\mathbf{V})\} \quad (14)$$

Finally, the transition realtion R is encoded as follows:

$$\begin{aligned} R(\mathbf{V}, \mathbf{V}') \quad \doteq \bigwedge_{j=1}^{n} \{ \bigvee_{e \in E_j} \hat{T_e}(\mathbf{V}, \mathbf{V}') \vee T_{fix}^{j}(\mathbf{V}, \mathbf{V}') \wedge \\ ass(\mathbf{V}, \mathbf{V}')\} \wedge com(\mathbf{V}, \mathbf{V}') \end{aligned} \quad (15)$$

To an external observer, an interesting state of a network will be those where all the time satmp take the same value. Therefore, for reachability formula (1), the following additional constraint $SYN(\mathbf{V}^k) \doteq \bigwedge_{j=2}^{n} \tau_1^k = \tau_j^k$ is needed.

IV. Experiments

Our current prototype implementation of the bounded model checking tool is implemented in C++ using Yacc+Lex to read the benchmarks and Mathsat 4 as back-end engine. All experiments were run under Ubuntu 14.04 on desktop with an Intel XEON Processor E3-1230 v3 @ 3.3 GHz and 8 GB of memory. Table I summarizes our experimental results for $Fischer's\ protocol$ and $Networks\ of\ Chain-like\ Timed\ Automata$ respectively. For $Fischer's\ protocol$, we check "is there a state in which all the processes enter in the critical section?" [5]. If $\Delta \geq \delta$, then the property is violated and the bound is growing fast with the increasing of number of processes. For $Networks\ of\ Chain-like\ Timed\ Automata(NCTA)$, as shown in Figure 1, is adapted from [11]. The benchmark consists of parallel composition of n independent synchronization pair of chains, each being a parallel composition of two synchronized sequences of length n. We verify if it is possible that all automata reside in its last location at the same time, formalized as property $\varphi = EF \bigwedge_{i=1}^{n} l_{2n}^i \wedge r_{2n}^i$. Table I consist of the following data: the first column shows the number of processes; the second one shows the length of counterexample produced by original encoding scheme, i.e., encodes discrete event and time elapse separately; the next two columns, respectively, show the time and memory consumed by the Single-Global (single transition + global time semantics given by the previous work [5] [6]) encoding; The latest three columns display the same information for the Composed-Step (composed transitions +step semantics)

Table I
EXPERIMENTAL RESULTS FOR PROPERTY ψ(TIME:SEC,MEMORY:KB)

Mod	Single-Global			Composed-Global			Composed-Step		
	k	Time	Mem	k	Time	Mem	k	Time	Mem
F2	7	0.03	84	4	0.01	88	4	0.02	4196
F3	15	0.31	8236	8	0.08	88	5	0.05	40432
F4	23	14.11	31012	12	2.02	10292	6	0.12	42116
F5	31	2296.28	328464	16	403.58	64484	7	0.22	45348
F6	-	-	-	-	-	-	8	0.70	49292
F7	-	-	-	-	-	-	9	0.80	54440
F8	-	-	-	-	-	-	10	1.68	61052
F9	-	-	-	-	-	-	11	5.42	75036
F10	-	-	-	-	-	-	12	11.26	174540
F11	-	-	-	-	-	-	13	124.81	159296
N2	17	0.50	8864	12	0.50	6724	8	0.50	5368
N4	33	6.81	36276	24	9.01	25468	16	9.51	29964
N6	49	29.93	104308	36	33.44	91396	24	102.43	155968
N8	65	70.88	229472	48	121.54	192384	32	111.03	161800
N10	81	158.27	426784	60	267.60	340084	40	247.58	262888
N12	97	302.42	696624	72	455.11	563944	48	263.702	282888
N14	113	1567.34	965544	84	1484.28	870888	56	1171.70	719384
N16	129	-	-	96	-	-	64	2024.89	1056156

encoding. The time limit was fixed to 1 hour for each run, '-' denotes that the system reached the time limit.

V. CONCLUSION

BMC of TAN can be accelerated by exploiting step semantics, several independent actions can be compressed into a single step to alow these actions are executed in parallel, in the hope of achieving faster coverage of reachable state space. However, the coding of the step semantics turned out to be more costly in the ways of coding these step semantics than the gain by compressed action sequences [12]. In this paper, we have presented a novel time stamp semantics for TAN, which is well suited for coding of step semantics since it does not significantly increase transition formula size than interleaving semantics. Moreover, the time stamp semantics also allows more actions to be executed simultaneously within a step than the previous related works for step semantics [12]. As a result, our coding for step semantics can achieve a significantly performance gain compared to the previous works. Experimental results indicate that our coding do indeed substantially speed up bounded model checking of TAN.

ACKNOWLEDGMENT

The authors would like to thank Peter Niebert for fruitful discussions and comments on earlier drafts of this paper. This work was supported by the Young Teachers Education and Scientific Research Projects of Fujian under Grant JAT170041, the Scientific Research Funds of Huaqiao University under Grant 16BS708, the Natural Science Foundation of Higher Education Institutions of Jiangsu Province under Grant 18KJB520052, and the National Natural Science Fund of China under Grant 61802134, 61170028 and 61733006, the Program for New Century Excellent Talents in Fujian Province Universities under Grant 2013FJ-NCET-ZR03, the Natural Science Foundation of Fujian Province under Grant 2015J01255.

REFERENCES

[1] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[2] G. Behrmann, A. David, K. G. Larsen, P. Pettersson, and W. Yi, "Developing UPPAAL over 15 years," *Softw., Pract. Exper.*, vol. 41, no. 2, pp. 133–142, 2011.

[3] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.

[4] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, O. Maler, and N. Jain, "Verification of timed automata via satisfiability checking," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, ser. LNCS, vol. 2469. Springer, 2002, pp. 225–243.

[5] G. Audemard, A. Cimatti, A. Kornilowicz, and R. Sebastiani, "Bounded model checking for timed systems," in *Formal Techniques for Networked and Distributed Systems*, ser. LNCS, vol. 2469. Springer, 2002, pp. 243–259.

[6] M. Sorea, "Bounded model checking for timed automata," in *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002, p. 2002.

[7] W. Penczek, B. Wozna, and A. Zbrzezny, "Towards bounded model checking for the universal fragment of TCTL," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, ser. LNCS, vol. 2469. Springer, 2002, pp. 265–290.

[8] K. Heljanko, "Bounded reachability checking with process semantics," in *In Proceedings of the 12th International Conference on Concurrency Theory (Concur 2001*. Springer, 2001, pp. 218–232.

[9] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi, "Partial order reductions for timed systems," in *International Conference on Concurrency Theory*, ser. LNCS, vol. 1466. Springer, 1998, pp. 485–500.

[10] D. Lugiez, P. Niebert, and S. Zennou, "A partial order semantics approach to the clock explosion problem of timed automata," *Theor. Comput. Sci.*, vol. 345, no. 1, pp. 27–59, 2005.

[11] R. Salah, M. Bozga, and O. Maler, "On interleaving in timed automata," in *CONCUR*, ser. LNCS, vol. 4137. Springer, 2006, pp. 465–476.

[12] J. Malinowski and P. Niebert, "SAT based bounded model checking with partial order semantics for timed automata," in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS. Springer, 2010, vol. 6015, pp. 405–419.

[13] S. Mukhopadhyay and A. Podelski, "Beyond region graphs: Symbolic forward analysis of timed automata," in *Foundations of Software Technology and Theoretical Computer Science, Volume 1738 of LNCS*. Springer, 1999, pp. 233–245.