# Efficient TCTL Model Checking Algorithm for Timed Actors

E. Khamespanah
University of Tehran, Tehran, Iran
Reykjavik University, Reykjavik, Iceland
e.khamespanah@ut.ac.ir

R. Khosravi
University of Tehran
Tehran, Iran
r.khosravi@ut.ac.ir

M.Sirjani
Reykjavik University
Reykjavik, Iceland
marjan@ru.is

## ABSTRACT

Non-Polynomial time complexity of model checking algorithms for TCTL properties in dense time is one of the obstacles against using model checking for timed systems. Alternatively, polynomial time algorithms are suggested for model checking discrete time models presented as Duration Time Graphs (DTG) versus a subset of TCTL formula (TCTL$_{\leq,\geq}$). While TCTL$_{\leq,\geq}$ can be model checked in polynomial time, the problem of model checking for exact time condition (TCTL$_{=}$) is an NP-Hard problem unless certain conditions hold. In this work we tackle model checking of timed actors using DTG. At the first step, we propose a reduction technique by folding all the instantaneous transitions, resulting folded timed transition system (FTS). At the second step, we show how the FTS of timed actors with discrete time can be mapped to a DTG. Then, we show when the necessary conditions hold for the FTS of timed actors and hence there is an $O(n^2)$ algorithm for model checking of complete TCTL properties (including TCTL$_{\leq,\geq}$ and TCTL$_{=}$) which have small constant time quantifiers. We use a set of case studies to illustrate the impact of using this technique in different application domains.

## Categories and Subject Descriptors

D.2.4 [**SOFTWARE ENGINEERING**]: Software/Program Verification - Assertion checkers, Formal methods, Model checking.

## Keywords

Actor Model, Timed Rebeca, Model Checking, TCTL, Discrete Time Transition System, Duration Transition Graph

## 1. INTRODUCTION

Using timed actors[1] for modeling and analysis of real-time systems with asynchronous message passing is one of the well-known approaches. Although there are some works on verification of timed actors [10, 7], the lack of efficient model checking algorithm has limited the use of model checking for verification of timed actors.

Most of the works on model checking of real-time systems are done for Alur and Dill's Timed Automata [3]. As a result, there exists a deep theoretical knowledge and large number of practical experiences for the systems which can be specified by timed automata. Based on established theoretical works the model checking algorithms of timed automata are at least in PSPACE-hard class for TCTL properties [2]. Timed model checkers like UPPAAL only support a subset of TCTL that can be model checked efficiently [4].

On the contrary, wider range of TCTL properties can be efficiently analyzed for simpler families of timed models. One simplification is done in [8, 6] by assuming that each transition takes exactly one time unit. Later, a small extension is added to this work by allowing existence of instantaneous transition (zero time transitions) in [13]. Finally Timed Transition Graph (TTG) [5] and Duration Transition Graph (DTG) [12] extended the former works by associating discrete time duration to the transitions. Although TTG and DTG are less expressive than timed automata, there are efficient model checking algorithms for them. DTG is expressive enough to being used as the semantics of discrete timed actors, as we used here.

In this work we use the algorithm of [12] for model checking of timed actors. To this aim, in Section 4, we show that the semantics of discrete timed actors, shown as Timed Transition Systems (TTS), can be captured as a DTG and hence the algorithm of [12] can be used for model checking of it. Although the proposed model checking algorithm efficiently works for TCTL$_{\leq,\geq}$ properties, model checking against TCTL$_{=}$ properties remains NP-complete. We propose a new approach based on [15] for model checking of TCTL$_{=}$ properties for Timed Rebeca models. Timed Rebeca is an extension of Rebeca [19, 18] with time features for modeling and verification of time-critical systems. Rebeca is an actor-based language for modeling concurrent and reactive systems with asynchronous message passing. An introduction to Timed Rebeca and its formal semantics is presented in Section 2 and TCTL model checking of DTGs is presented in Section 3. The results of applying the approach of this work is depicted in Section 5 to illustrate the impact of the approach in verification of timed actors.

The current work will add an efficient tool for model checking TCTL properties of Timed Rebeca models to the existing rich toolset of Rebeca. A short overview of the previous ap-

[1] We use "timed actor" and "discrete timed actor" in this paper interchangeably.

*AGERE* '14 Portland, USA

proaches are presented in the following as related works.

**Related Work.** A tool is developed for model checking Timed Rebeca by transforming Timed Rebeca models to timed automata. The resulted timed automata are model checked against TCTL properties using UPPAAL toolset. Timed automata are used as back-end timed model of different high-level modeling languages and UPPAAL is used as a successful back-end model checker. But in case of Timed Rebeca, because of the inefficiency of modeling asynchronous communication among actors by synchronized communication of timed automata, model checking results in state space explosion even for middle-sized case studies [11].

In another work, Floating Time Transition System (FTTS) is defined as an event-based semantics for Timed Rebeca models in [11]. Focusing on the analysis of Timed Rebeca based on the key features of actors, being event-driven and isolated, results in a significant amount of state space reduction in FTTS. However, FTTS cannot be used for model checking state-based TCTL formulas. States in FTTS contain the local times of each rebec, in addition to values of their state variables and the bag of their received messages. The local times of rebecs in a state can be different from each other, and there is no unique value for time in each state. This is only admissible where we are not interested in the state of all the rebecs at a specific point of time, e.g. checking for deadlock freedom and deadline misses, or any other event-based property.

Another tool is developed for mapping Timed Rebeca to Real-Time Maude. This enables a formal model-based methodology which combines the convenience of intuitive modeling in Timed Rebeca with formal verification in Real-Time Maude. Real-Time Maude is supported by a high-performance toolset providing a spectrum of analysis methods, including simulation through timed rewriting, reachability analysis, and (untimed) linear temporal logic (LTL) model checking as well as timed CTL model checking. As described in [16], a reduction technique is applied to the generated Real-Time Maude models to avoid state space explosion. Mainly, a number of statements (which are related to the instantaneous statements of Timed Rebeca except sending message) are "group together" and execute them in one "atomic" rewrite step. This approach significantly improves the performance of the model checking in comparison with the "standard" approaches (i.e. each action is performed by a rewrite step). While Real-time Maude provides us with a wide range of analysis tools, our current work only covers the TCTL model checking. The experimental results show that our new tool outperform Real-Time Maude in TCTL model checking. In the reduction technique of our current work, we group together instantaneous statements including sending messages. This results in more reduction on the size of state spaces.

There are also analysis tools for Timed Rebeca models using simulation techniques. In [14], the simulation engine of Erlang is used to generate a number of traces and verify them. Using this approach, state space explosion is avoided; however, it does not guarantee the correctness of model.

**Contributions.** In a nutshell, the contributions of this paper can be summarized to proposing a technique for efficient TCTL model checking of timed actors by:

- Proposing Folded Timed transition System by applying a reduction technique on Timed transition System used as the semantics of timed actors
- Proving that the Folded Timed Transition System is a DTG which is used for model checking against TCTL$_{\leq,\geq}$ properties
- Using a modified version of pseudo-polynomial time algorithm of finding the Exact Path Length in weighted graphs for model checking against TCTL$_{=}$ properties

## 2. TIMED REBECA

Timed Rebeca is an extension of Rebeca [19] with time features for modeling and verification of time-critical systems. We illustrate Timed Rebeca language constructs using a simplified version of ticket service example in Figure 1.

A Timed Rebeca model consists of a number of *reactive classes*, each describing the type of a certain number of *actors* (called *rebecs* in Timed Rebeca. In this paper we use rebec and actor interchangeably). In the ticket service model, we have three reactive classes `TicketService`, `Agent`, and `Customer`. Each reactive class declares a set of *state variables*. The local state of each actor is defined by the content of its message bag and values of its state variables. Following the actor model, the communication in the model takes place by asynchronous messages passing among actors. Each actor has a set of *known rebecs* to which it can send messages to. For example, an actor of type `TicketService` knows an actor of type `Agent` (line 2), to which it can send messages (line 12). Reactive classes declare the messages to which they can respond. The way an actor responds to a message is specified in a *message server*. An actor can change its state variables through assignment statements (e.g., line 13), make decisions through conditional statements (not appearing in our example), and communicate with other actors by sending messages (e.g., line 12). The periodic behavior is modeled by actors sending messages to themselves (e.g., line 38). Since the communication is asynchronous, each actor has a *message bag* from which it takes the next incoming message. The ordering of message in a message bag is based on the arrival times of messages. An actor takes the first message from its message bag, executes the corresponding message server in an isolated environment, and then takes the next message (or waits for the next message to arrive) and so on. The message server may have *nondeterministic assignment* statement which is used to model nondeterminism in the behavior of a message server.

Finally, the `main` block is used to instantiate the actors of the system. In the ticket service model, three actors are created receiving their known rebecs and the arguments to their constructor upon instantiation (lines 42-44).

Timed Rebeca adds three primitives to Rebeca to address timing issues: *delay*, *deadline* and *after*. A *delay* statement models passing of time for an actor during execution of a message server (line 14). Note that all other statements are assumed to execute instantaneously. The keywords *after* and *deadline* can be used in conjunction with a method call. The term `after` *n* indicates that it takes *n* units of time for the message to be delivered to its receiver. For example, the periodic task of requesting a new ticket is modeled in line 45 by the customer sending a `try` message to itself and letting the receiver (itself) to take it from its message bag only after 30 units of time. The term `deadline` *n* shows that if the message is not taken in *n* units of time, it will be purged from the

```
1  reactiveclass TicketService {
2    knownrebecs {Agent a;}
3    statevars {
4      int issueDelay, nextId;
5    }
6    TicketService(int myDelay) {
7      issueDelay = myDelay;
8      nextId = 0;
9    }
10   msgsrv requestTicket() {
11     delay(issueDelay);
12     a.ticketIssued(nextId);
13     nextId = nextId + 1;
14   }
15 }
16 reactiveclass Agent {
17   knownrebecs {
18     TicketService ts;
19     Customer c;
20   }
21   msgsrv requestTicket() {
22     ts.requestTicket()
23         deadline(5);
24   }
25   msgsrv ticketIssued(byte id) {
26     c.ticketIssued(id);
27   }
28 }
29 reactiveclass Customer {
30   knownrebecs {Agent a;}
31   Customer() {
32     self.try();
33   }
34   msgsrv try() {
35     a.requestTicket();
36   }
37   msgsrv ticketIssued(byte id) {
38     self.try() after(30);
39   }
40 }
41 main {
42   Agent a(ts, c):();
43   TicketService ts(a):(3);
44   Customer c(a):();
45 }
```

**Figure 1: The Timed Rebeca model of ticket service system.**

receiver's message bag automatically. For example, lines 26-27 indicates a `requestTicket` message to the ticket service must be started to execute before five units from sending the message.

## 2.1 Semantics of Timed Rebeca In Timed Transition System

At the first step of presenting the semantics of Timed Rebeca, we formalize the definition of a number of primitive concepts in Timed Rebeca. A rebec $r_i$ with the unique identifier $i$ is defined as the tuple $(\mathcal{V}_i, \mathcal{M}_i, \mathcal{K}_i)$ where $\mathcal{V}_i$ is the set of its state variables, $\mathcal{M}_i$ is the set of its message servers, and $\mathcal{K}_i$ is the set of its known rebecs. The set of all the values of the state variables of $r_i$ is denoted by $Vals_i$. For a Timed Rebeca model $\mathcal{M}$, there is a universal set $\mathcal{I}$ which contains identifiers of all the rebecs of $\mathcal{M}$.

A (timed) message is defined as $tmsg = ((sid, rid, mid), ar, dl)$, where rebec $r_{sid}$ sends the message $m_{mid} \in \mathcal{M}_{rid}$ to rebec $r_{rid}$. This message is delivered to the rebec $r_{rid}$ at $ar \in \mathbb{N}_0$ as its arrival time and the message should be served before $dl \in \mathbb{N}_0$ as its deadline. For the sake of simplicity, we ignore the parameters of the messages here.

Each rebec $r_i$ has a message bag $\mathcal{B}_i$ which can be defined as a multiset of timed messages. $\mathcal{B}_i$ stores the timed messages which are sent to $r_i$. The set of possible states of $\mathcal{B}_i$ is denoted by $Bags_i$.

Timed transition system is generally the standard semantic framework for discrete timed systems, and we define the TTS of Timed Rebeca in this section.

Timed Transition System of the Timed Rebeca model $\mathcal{M}$ is a tuple of $TTS = (S, s_0, Act, \rightarrow, AP, L)$ where $S$ is the set of states, $s_0$ is the initial state, $Act$ is the set of action, and $\rightarrow$ is the transition relation. $AP$ is the set of atomic propositions and $L$ is a labeling function in form of $L : S \rightarrow 2^{AP}$.

**States.** A state $s \in S$ consists of the local states of the rebecs, together with the current time of the state. The local state of rebec $r_i$ in state $s$ is defined as the tuple $(V_{s,i}, B_{s,i}, pc_{s,i}, res_{s,i})$, where

- $V_{s,i} \in Vals_i$ is the values of the state variables of $r_i$
- $B_{s,i} \in Bags_i$ is the message bag of $r_i$
- $pc_{s,i} \in \{null\} \cup (\mathcal{M}_i \times \mathbb{N})$ is the program counter, tracking the execution of the current message server ($null$ if $r_i$ is idle in $s$)
- $res_{s,i} \in \mathbb{N}_0$ is the resuming time, if $r_i$ is executing a `delay` in $s$

So, state $s \in S$ can be defined as the following where $now_s \in \mathbb{N}$ is the current time of $s$.

$$\left( \prod_{i \in \mathcal{I}} (V_{s,i}, B_{s,i}, pc_{s,i}, res_{s,i}), now_s \right)$$

**Initial State.** $s_0$ is the initial state of the Timed Rebeca model $\mathcal{M}$ where the state variables of the rebecs are set to their initial values (according to their types), the `initial` message is put in the bag of all rebecs having such a message server, the program counters of all rebecs are set to $null$, and the time of the state is set to zero.

**Actions.** There are three possible types of actions: sending a message $tmsg = ((sid, rid, mid), ar, dl)$ (as defined in Section 2.1), executing a statement by an actor (which we consider as an internal transition $\tau$), and progress of $n \in \mathbb{N}$ units of time. Hence, the set of actions is defined as

$$Act = \bigcup_{i \in \mathcal{I}} ((\mathcal{I} \times i \times \mathcal{M}_i) \times \mathbb{N} \times \mathbb{N}) \cup \{\tau\} \cup \mathbb{N}$$

**Transition Relations.** Before defining the transition relation, we introduce the notation $E_{s,i}$ which denotes the set of *enabled messages* of rebec $r_i$ in state $s$ which contains the messages which their arrival time is less than or equal to $now_s$. The transition relation $\rightarrow \subset S \times Act \times S$ is defined such that $(s, act, t) \in \rightarrow$ if and only if one of the following conditions holds.

1. **(Taking a message for execution)** In state s, there exists $r_i$ such that $pc_{s,i} = null$ and there exists $tmsg \in E_{s,i}$. Here, we have a transition of the form $s \xrightarrow{tmsg} t$. This transition results in extracting $tmsg$ from the message bag of $r_i$, setting $pc_{t,i}$ to the first statement of the message server corresponding to $tmsg$, and setting $res_{t,i}$ to $now_t$ (which is the same as $now_s$). Note that $V_{t,i}$ remains the same as $V_{s,i}$. These transitions are called *taking-event transitions*.

2. **(Internal action)** In state $s$, there exist $r_i$ such that $pc_{s,i} \neq null$ and $res_{s,i} = now_s$. The statement of message server of $r_i$ specified by $pc_{s,i}$ is executed and one of the following cases occurs based on the type of the statement. Here, we have a transition of the form $s \xrightarrow{\tau} t$.

(a) Non-delay statements: the execution of the statement may change the value of a state variable of rebec $r_i$ or sending a message to other rebecs. Here, $pc_{t,i}$ is set to the next statement (or *null* if there is no more statements). All other elements of $t$ is the same as those of $s$.

(b) Delay statement with parameter $d \in \mathbb{N}$: the execution of delay statement sets $res_{t,i}$ to $now_s + d$. All other elements of the state remain unchanged. Particularly, $pc_{t,i} = pc_{s,i}$ because the execution of delay statement is not yet complete. The value of the program counter will set to the next statement after completing the execution of delay (as will be shown in the third case).

These transitions are called *internal transitions*.

3. **(Progress of time)** If in state $s$ none of the conditions in cases 1 and 2 hold, meaning that $\nexists r_i \cdot ((pc_{s,i} = null \wedge E_{s,i} \neq \emptyset) \vee (pc_{s,i} \neq null \ \wedge \ res_{s,i} = now_s))$, the only possible transition is via progress of time. In this case, $now_t$ is set to $now_s + d$ where $d \in \mathbb{N}$ is the minimum value which makes one of the aforementioned conditions become true. The transition is of the form $s \xrightarrow{d} t$. For any rebec $r_i$, if $pc_{s,i} \neq null$ and $res_{s,i} = now_t$ (the current value of $pc_{s,i}$ points to a delay statement), $pc_{t,i}$ is set to the next statement (or to *null* if there are no more statements). These transitions are called *time transitions*. Note that when such a transition exists, there is no other outgoing transition from $s$ and it called *progress-of-time state*.

**Atomic Propositions.** In General, atomic propositions are used to formalize temporal characteristics of states. In case of Timed Rebeca, atomic propositions intuitively express simple known facts about the value of state variables of rebecs in states of the model under consideration.

**Labeling Function.** Function $L : S \rightarrow 2^{AP}$ relates a set of atomic propositions to each state, shown by $L(s)$ for a given state $s$. $L(s)$ intuitively relates the atomic propositions which are satisfied in state $s$.

## 3. TIMED MODEL CHECKING FOR DISCRETE TIME SYSTEMS

As mentioned in Section 1, there are many timed models for modeling of discrete time systems which can be model checked efficiently (in polynomial-time). Usually, these timed models are based on Kripke Structures (KS). Using KS, the elapsing of time is handled by events. Duration transition graph (DTG) is defined as an extension of KS for handling real-time aspects of systems in a way that model checking remains efficient (polynomial-time) [12]. A duration transition graph is a transition system which assigns duration to each transition. A duration is shown by an interval between two natural numbers. Figure 2 is an example of DTG to model the academic activities of a researcher.

Definition 1 (Duration Transition Graph). *A duration transition graph is a tuple of $DTG = (S, s_0, Act, \rightarrow, AP, L)$ where $S$ is the set of states, $s_0$ is the initial state, $Act$ is the set of action, and $\rightarrow \subseteq S \times Act \times \rho \times S$ is the transition relation such that $\rho$ is a finite ($\rho = [n, m] \cdot n, m \in \mathbb{N}$) or right-open infinite ($\rho = [n, \infty) \cdot n \in \mathbb{N}$) interval. $AP$ is the set of atomic propositions and $L$ is a labeling function in form of $L : S \rightarrow 2^{AP}$.* □

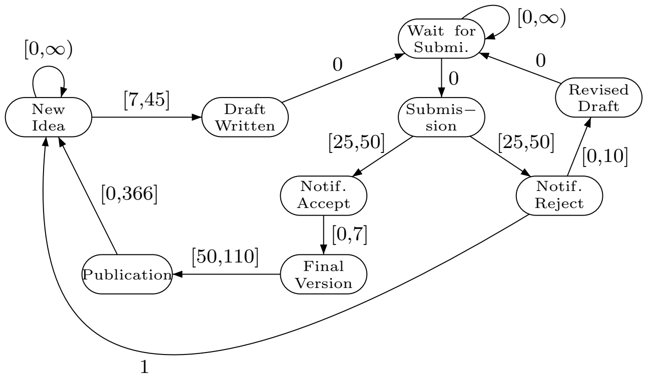

**Figure 2: A DTG modeling publications by one researcher (time in days) [12].**

In the above definition, the meaning of transition between two state can be interpreted in two different ways, called *jump semantics* and *continuous semantics*. For a given transition $(s, act, [n, m], s')$ the mentioned semantics are interpreted as the following.

**Jump Semantics** using this semantics, moving from state $s$ to $s'$ takes an integer time $d \in [n, m]$. Hence, if the system is in state $s$ at time $t$, then it is in state $s'$ at time $t + d$ and there is no position for times $t + 1, t + 2, \cdots, t + d - 1$. The idea of this semantics is the same as the semantics of Timed Transition Graph [5]

**Continuous Semantics** using this semantics, the system waits for $d - 1$ units of time ($d \in [n, m]$) in state $s$ before performing action *act*. The idea of this semantics is the same as the semantics of timed automata of Alur [3] and the semantics of Timed Rebeca as described in Section 2.

Using DTG, there is a polynomial-time model checking algorithm for TCTL properties; however, model checking of TLTL or TCLT* properties remains PSPACE-complete.

Timed CTL (TCTL) is a real-time variant of CTL aimed to express properties of timed systems. In TCTL, the until modality is equipped with a time constraint such that the TCTL formula $\Phi \, \mathrm{U}^{\rho} \, \Psi$ holds for state $s$ if and only if $\Psi$ holds in state $s'$ while $\Phi$ holds in all states from $s$ to $s'$ and time difference between $s$ and $s'$ satisfies condition $\rho$. In the following, its interpretation for DTGs is presented [12].

Definition 2 (Syntax of TCTL). *A TCTL formula is formed according to the following grammar:*

$$\Phi ::= p \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \exists\,\varphi \mid \forall\,\varphi$$

*where $p$ is an atomic proposition and $\varphi$ is a path formula. A path formula in TCTL is formed according the the following grammar:*

$$\varphi ::= \mathrm{X}\,\Phi \mid \Phi \, \mathrm{U}^{\sim c} \, \Phi$$

*where $c$ is a natural number and $\sim \in \{<, \leq, =, \geq, >\}$.* □

Definition 3 (Semantics of TCTL). *The following clauses show that when a given TCTL formula $\Phi$ holds for state $s$ of $DTG = (S, s_0, Act, \rightarrow, AP, L)$. Here, we assumed that $Path(DTG, s_0)$ represents a set of timed path of DTG from the state $s_0$ in form of $\pi \in Path(DTG, s_0) \wedge \pi = s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} \cdots$.*

- $s \models \exists \mathbf{X} \Phi \Leftrightarrow \exists\, \pi \in Path(DTG, s) \cdot s_1 \models \Phi$
- $s \models \forall \mathbf{X} \Phi \Leftrightarrow \forall\, \pi \in Path(DTG, s) \cdot s_1 \models \Phi$
- $s \models \exists \Phi_1 \mathbf{U}^{\sim c} \Phi_2 \Leftrightarrow \exists\, \pi \in Path(DTG, s) \wedge \exists\, n \geq 0 \cdot (s_n \models \Phi_2 \wedge \sum_{i \in [0,n)} d_i$ *satisfies condition* $\sim c \wedge (\forall\, 0 \leq j < n \cdot s_j \models \Phi_1)$
- $s \models \forall \Phi_1 \mathbf{U}^{\sim c} \Phi_2 \Leftrightarrow \forall\, \pi \in Path(DTG, s) \wedge \exists\, n \geq 0 \cdot (s_n \models \Phi_2 \wedge \sum_{i \in [0,n)} d_i$ *satisfies condition* $\sim c \wedge (\forall\, 0 \leq j < n \cdot s_j \models \Phi_1)$

□

In the following we introduce the model checking algorithm of DTGs against $\text{TCTL}_{\leq,\geq}$ properties according to [12]. The modified version of this algorithm is used for model checking of Timed Rebeca models.

Let $DTG_{\mathcal{M}} = (S, s_0, Act, \rightarrow, AP, L)$ be a DTG. The extended version of standard CTL model checking algorithm is used to support $\exists \Phi_1 \mathbf{U}^{\sim c} \Phi_2$ and $\forall \Phi_1 \mathbf{U}^{\sim c} \Phi_2$ subformulas. The following two cases show that how the extension works for timed subformula $\xi = \exists \Phi_1 \mathbf{U}^{\sim c} \Phi_2$.

- $\xi = \exists \Phi_1 \mathbf{U}^{\leq c} \Phi_2$: Assume that $DTG_{\mathcal{M}}$ is reduced to subgraph $DTG^{sub}_{\mathcal{M}} = (S', s'_0, Act, \rightarrow', AP', L')$ where only states satisfying $\exists \Phi_1 \mathbf{U} \Phi_2$ are kept. In addition, the value of lower bound of duration intervals are assumed as the weight of each transition of $DTG^{sub}_{\mathcal{M}}$. This way, any state $s \in S'$ is in $s \models \xi$ relation iff running *single source shortest path* algorithm from state $s \in S'$ results in finding a path from $s$ to $s'$ where $s' \models \Phi_2$ and the weight of the path is not bigger than $c$.
- $\xi = \exists \Phi_1 \mathbf{U}^{\geq c} \Phi_2$: In this case, we assumed that we have $DTG^{sub}_{\mathcal{M}}$ the same as the case one. Here $s \in S'$ is in $s \models \xi$ relation iff one of the following condition holds.
  - there is a longest acyclic path from $s$ to a state which satisfies $\Phi_2$
  - there is a path with cycle inside path from $s$ to a state which satisfies $\Phi_2$

Both of these cases can be checked in polynomial time.

Formulas in form of $\forall \Phi_1 \mathbf{U}^{\sim c} \Phi_2$ can be transformed to their equivalent formulas with $\exists \mathbf{U}^{\sim c}$ operator. Therefore, the above cases can be used to verify them.

# 4. TCTL MODEL CHECKING OF TIMED REBECA

For TCTL model checking of Timed Rebeca models, at the first step, we propose a reduction technique to reduce the size of timed transition systems, called "folding instantaneous transitions". The model checking algorithms of this paper works for reduced timed transition systems (folded timed transition system or FTS in short). We model check $\text{TCTL}_{\leq,\geq}$ properties using the algorithms of [12]. To this aim, we prove that FTS of a Timed Rebeca model is a DTG. Therefore, we can use polynomial-time algorithms for model checking of Timed Rebeca models against $\text{TCTL}_{\leq,\geq}$. Then, we show that the approach of [15] can be used to have an efficient algorithm for model checking of $\text{TCTL}_{=}$ properties in FTS. Finally, we discuss that using FTS, the model checking algorithm of Timed Rebeca models against TCTL properties reduced to an $O(n^2)$ problem for a wide range of TCTL properties.

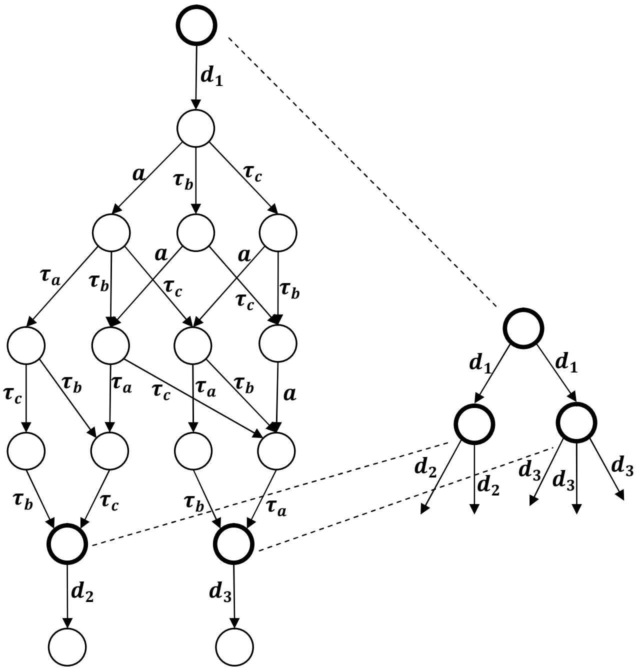

**Figure 3: Example of how the folding instantaneous transitions reduction works.**

## 4.1 Folding Instantaneous Transitions

Folding instantaneous transitions is a reduction technique that eliminates instantaneous transitions from the state space. Applying this reduction technique results in a transition system (FTS) which has only progress-of-time transitions. The main idea behind FTS is the fact that in timed actor systems with continuous semantics the residual time in states with instantaneous transitions are zero. Therefore, if we are looking for correctness of properties which are defined by state propositions (not action propositions) only the progress-of-time states can be considered. It means that the environment observes the value of state variables in the state before starting the first instantaneous transition and the state after the last instantaneous transition.

Regarding to the continuous semantics of Timed Rebeca and as we are looking for model checking against TCTL properties with state propositions, folding instantaneous transition reduction is applicable for TTS of Timed Rebeca models. In Timed Rebeca, taking-event and internal transitions are instantaneous and sequence of taking-event and internal transitions are surrounded by progress-of-time transitions (it is assumed that the system does not have Zeno behavior). Figure 3 shows a FTS (in the right side) corresponding to the timed transition system (in the left side) of a Timed Rebeca model. In the figure, the states with bold borders are progress-of-time states. We assumed that after progress of time by $d_1$ time units, one actor starts doing a new action (shown by taking-event transition $a$) and two other actors continue their previously started actions (shown by internal transitions $\tau_b$ and $\tau_c$).

For formal definition of FTS of Timed Rebeca models, at the first step, we need to define a function which finds the nearest progress-of-time states from a given state. For a given timed transition system $TTS_{\mathcal{M}} = (S, s_0, Act, \rightarrow, AP, L)$ a set of nearest progress-of-time states from a given state $s \in S$ is shown by $npts(s)$. Each state $s' \in npts(s)$ is a progress-of-time state to which there is a path $\pi = s, s_1, s_2, \cdots, s_n, s'$ where none of $s_1, s_2, \cdots, s_n$ are progress-of-time state. Algorithm 10 shows the details of finding *npts*. As we assumed that the Timed Rebeca models are Zeno free, all the cycles of $TTS_{\mathcal{M}}$ have at least one progress-of-time state. Therefore, the `while` loop in line 3 of Algorithm 10 terminates in polynomial time. Based on Zeno freedom assumption, in Algorithm 10 each transition of the system's transition system is traversed at most once in `while` loop; hence, the order of the algorithm is $O(n^2)$ where $n$ is the number of states.

**Algorithm 1:** $npts(s)$ finds the nearest progress-of-time states of a given state $s$

```
   Input: State $s$ from a timed transition system
   Output: Set of nearest progress-of-time states
1  $\Pi \leftarrow \emptyset$
2  $\Gamma \leftarrow \{s\}$
3  while $\Gamma \neq \emptyset$ do
4      remove $s'$ from $\Gamma$
5      if $IS\_PROGRESS\_OF\_TIME(s')$ then
6          $\Pi \leftarrow \Pi \cup \{s'\}$
7      else
8          $\Gamma \leftarrow \Gamma \cup SUCCESSORS(s')$
9  return $\Pi$
10 using Breadth-First Search
```

Using the definition of *npts*, the FTS of a Timed Rebeca model is defined based on TTS of that model as the following.

Definition 4 (Folded Timed Transition System). *For a given timed transition system $TTS_{\mathcal{M}} = (S, s_0, Act, \rightarrow, AP, L)$, folded timed transition system is defined as a tuple of $FTS_{\mathcal{M}} = (S', s'_0, Act', \hookrightarrow, AP', L')$ where:*

- *$S' \subseteq S$, $s'_0 = s_0$, $AP' = AP$, and $L' = L$*
- *All the members of $S'$ are progress-of-time states except the initial state.*
- *For two states $s'_1, s'_2 \in S'$ there is $(s'_1, act', s'_2) \in \hookrightarrow$ if and only if $s'_2 \in npts(s'_1)$. The value of $act'$ is the same as the outgoing progress-of-time transition of $s'_1$. In case of $s'_1 = s'_0$, $act'$ is set to a progress-of-time transition with duration zero.*

□

For creating a FTS of a given Timed Rebeca model, at the first step, the timed transition system of the Timed Rebeca model is created. Then, using a modified version of Breadth-First Search algorithm the FTS is created, as shown in Algorithm 2. The algorithm starts from the initial state of TTS and includes its nearest progress-of-time states as the successors of the initial states. Then it continues in the same way to find the successors of the newly added states.

Lemma 1. *The FTS of a given TTS can be created by an algorithm with time complexity of $O(n^2)$ where n is the number of the states of TTS.*

Proof. The loop in line 10 traverses progress-of-time transitions and the other transitions are traversed by function *npts*

**Algorithm 2:** $FTS(TTS_{\mathcal{M}})$ creates FTS from a given TTS

```
   Input: Timed transition system
          $TTS_{\mathcal{M}} = (S, s_0, Act, \rightarrow, AP, L)$
   Output: Folded timed transition system of model $\mathcal{M}$
1  $S' \leftarrow \{s_0\}$
2  $Act' \leftarrow \emptyset$
3  $\hookrightarrow \leftarrow \emptyset$
4  $AP' \leftarrow AP$
5  $L' \leftarrow L$
6  $\Pi \leftarrow \{s_0\}$
7  while $\Pi \neq \emptyset$ do
8      remove $s$ from $\Pi$
9      $NPTS \leftarrow npts(s)$
10     foreach element $s'$ of $NPTS$ do
11         $\hookrightarrow \leftarrow \hookrightarrow \cup \{(s, act', s')\}$
12         if $s' \notin S'$ then
13             $S' \leftarrow S' \cup \{s'\}$
14             $\Pi \leftarrow \Pi \cup \{s'\}$
15 return $(S', s_0, Act', \hookrightarrow, AP, L)$
```

in line 9 of the algorithm. Therefore, each transition of the TTS is traversed once. This way, the order of the algorithm is the same as the number of the transitions of the TTS which is $O(n^2)$. □

## 4.2 Model Checking for TCTL$_{\leq,\geq}$ Properties

A general overview of the model checking algorithm for DTGs against TCTL$_{\leq,\geq}$ properties is presented in Section 3. We can use the proposed algorithm for model checking of Timed Rebeca models because of the fact that FTSs of Timed Rebeca models are DTG.

Lemma 2. *The FTSs of Timed Rebeca models are DTG.*

Proof. For a given Timed Rebeca model $\mathcal{M}$, its FTS is defined as $FTS_{\mathcal{M}} = (S', s'_0, Act', \hookrightarrow, AP', L')$. The only difference between DTG and FTS is in the semantics of their transition relations. Each transition of DTG has an action label and time duration; however, transition relations of FTS are only *progress-of-time*. So, $FTS_{\mathcal{M}}$ can be assumed as a DTG where the actions of transitions are $\tau$ transitions. □

This way, we use the polynomial-time algorithm of [12] for model checking of Timed Rebeca models against TCTL$_{\leq,\geq}$ properties.

Corollary 1. *There is an $O(n^2)$ algorithm for model checking of Timed Rebeca models against TCTL$_{\leq,\geq}$ properties.*

## 4.3 Model Checking for TCTL$_{=}$ Properties

As known in graph theory, there is no polynomial algorithm for finding exact path length (EPL) between two nodes of a graph and finding EPL is NP-Hard (using reduction from finding the EPL between two states to subset-sum problem [9]). Using the same approach, the authors in [12] showed that the problem of model checking for exact time condition is a NP-Hard problem. Therefore, there is no polynomial-time algorithm for model checking of TCTL; however, its TCTL$_{\leq,\geq}$ subset can be model checked in polynomial-time.

But, as discussed in [15] there is a pseudo-polynomial algorithm for finding the EPL between two nodes of weighted graphs. The order of the algorithm is $O(W^2n^3 + |k| \min(|k|, W)\, n^2)$, where $n$ is the number of nodes, $W$ is the biggest absolute value of any edge weight, and $k$ is the target weight. The algorithm works in the following two phases.

- **Preprocessing:** In this phase, graphs are processed with a relaxation algorithm. As a result, the weights edges of different paths are set to values with the same signs. Note that, the proposed algorithm works for graphs with positive, negative, and zero weighted edges. The complexity of this phase is $O(W^2n^3)$.
- **Finding-Path:** At the second phase, the EPL between two nodes is found in the relaxed graph. The complexity of this phase is $O(|k| \min(|k|, W)n^2)$.

In case of finding the EPL in FTS of Timed Rebeca models, $W$ is the biggest time duration in the FTS. Hence, based on the semantics of Timed Rebeca, the value of $W$ is limited to the maximum value which is used as the parameter of `delay` and `after`. The value of $k$ is the same as the time quantifier of the given TCTL$_=$ formula (e.g. for TCTL$_=$ formula $\exists\, \Phi_1 \mathbf{U}^{=5} \Phi_2$ $k$ equals to five). Finding EPL is a polynomial-time algorithm if $W$ and $k$ have upper bound constant values. There is no limitation on the value of $W$ as it can be dynamically set by timed statements of Timed Rebeca. However, for wide range of TCTL properties, the time quantifier of TCTL$_=$ formulas are small constant values (in comparison to the size of the transition system). Based on this fact, there is a polynomial time model checking algorithm for Timed Rebeca models against TCTL$_=$ formulas with small constant time quantifiers as we will show in the following lemma.

Lemma 3. *There is an $O(n^2)$ algorithm for model checking of Timed Rebeca models against TCTL$_=$ properties with small constant time quantifiers.*

Proof. As the reduced FTS of Timed Rebeca models has only progress-of-time transitions, the weight of all the transitions are positive natural numbers and there is no need for relaxation algorithm with cost of $O(W^2n^3)$. Therefore, the complexity of the model checking algorithm is reduced to $O(|k| \min(|k|, W)\, n^2)$.

On the other hand, the time quantifiers assumed to be small constant integer values. Therefore, the algorithm looks for exact path between two states with constant value. Hence, the value of $k$ is constant in its corresponding finding EPL problem. Having constant value for $k$, the value of $\min(|k|, W)$ is at most $k$. As a result, the time complexity of finding exact path length in the state space is reduced from $O(|k| \min(|k|, W)\, n^2)$ to $O(n^2)$. □

Theorem 1. *Model checking of Timed Rebeca models against TCTL properties with small constant time quantifiers is an $O(n^2)$ problem.*

Proof. This follows directly from Lemma 1, Corollary 1, and Lemma 3. □

## 5. EXPERIMENTAL RESULTS

We provided five different case studies in different sizes to illustrate how efficiently the reduction technique works. The host computer of model checking toolset was a desktop computer with 1 CPU (2 cores) and 8GB of RAM storage, running Mavericks OS X 10.9.4. The selected case studies are *Ticket Service*, simplified version of *802.11 Wireless Protocol*, *Wireless Sensor and Actuator Networks (WSAN)*, simplified version of *Scheduler of Hadoop*, and model of NoC system with 64 cores. The Timed Rebeca code of the case studies and the model checking toolset are accessible from Rebeca homepage [1].

Details of *Ticket Service* case study is explained in Section 2. Catching the deadline of issuing the ticket is the main property of this model. We created different sizes of ticket service model by varying the number of customers, which results in three to ten rebecs in the model. In case of simplified version of *802.11 Wireless Protocol*, we modeled three wireless nodes which are communicating via a medium. The medium sets random back-off time when more than one node starts to send data, to resolve data collision in the medium. Deadlock avoidance is the main property of this model. In the third case study, a WSAN is modeled as a collection of actors for sensing, radio communication, data processing, and actuation. In WSAN applications, scheduling challenge is difficult as the wireless sensor platforms —which typically use event-driven operating systems— do not provide real-time scheduling guarantees. Deadline hit is verified as the main property of this model. As the forth case study, we modeled a simplified version of the behavior of MapReduce of Hadoop system, called YARN. We modeled one client which submits jobs to YARN resource manager. The resource manager distributes the submitted job among application masters and application masters split the job into some tasks and distribute tasks among some nodes. This model has 32 rebecs and verified to ensure that all the jobs are services before their deadlines. Finally, we used model of the traffic of packets in NoC systems. This model was published in [17] and we used one of its versions with mesh of 8x8 cores.

Table 1 shows the results of model checking of the four case studies. As shown in the table, the state space size of the FTS is at least two times smaller than the TTS in all the case studies. However, avoiding interleaving of concurrent instantaneous actions in *8x8 NoC* results in up to 92% reduction in the state space size.

## 6. CONCLUSION

In this paper we proposed a new reduction technique, folding instantaneous transitions, which significantly reduces the state space of timed actor models. Beside reducing the size of the state space, applying the reduction technique enables efficient TCTL model checking of the models. Formerly, we had to translate timed actor models to Real-Time Maude for TCTL model checking. Using the work of this paper, we apply model checking technique on a reduced state space which results in supporting bigger transition system. In addition, the work of this paper outperforms time consumption of TCTL model checking in comparison to using Real-Time Maude.

Experimental evidence supports our theoretical observation that the reduction technique results in smaller state space in general. In case of models with many concurrently executing actors, the reduced state space is up to 92% smaller than its original timed transition system. Therefore, we can efficiently model check more complicated models against complete TCTL properties under certain conditions. In addition, our technique and the proofs are applicable for model checking of any discrete time real-time model with continuous semantics against TCTL properties with propositions on the values of state variables.

| Problem | Size | State Space Size | Reduced State Space Size | Percentage of Reduction |
|---|---|---|---|---|
| Ticket Service | **2 customers** | 77 | 10 | 87% |
| | **3 customers** | 360 | 39 | 89% |
| | **4 customers** | 1825 | 184 | 90% |
| | **5 customers** | 10708 | 1045 | 90% |
| | **6 customers** | 73461 | 6996 | 90% |
| | **7 customers** | 581962 | 54019 | 91% |
| WSAN | - | 1920 | 818 | 57% |
| Yarn | - | 533 | 172 | 68% |
| 8x8 NoC | - | 74192 | 6068 | 92% |

**Table 1: The size of state spaces in different case studes.**

## Acknowledgments

The work on this paper has been partially supported by the project "Timed Asynchronous Reactive Objects in Distributed Systems: TARO" (nr. 110020021) of the Icelandic Research Fund.

## 7. REFERENCES

[1] *Rebeca Home Page*. http://www.rebeca-lang.org.

[2] L. Aceto and F. Laroussinie. Is your model checker on time? on the complexity of model checking for timed modal logics. *Journal of Logic and Algebraic Programming*, 52-53:7–51, 2002.

[3] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[4] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In R. Alur, T. A. Henzinger, and E. D. Sontag, editors, *Hybrid Systems*, volume 1066 of *Lecture Notes in Computer Science*, pages 232–243. Springer, 1995.

[5] S. V. Campos and E. M. Clarke. Theories and experiences for real-time system development. chapter Real-time Symbolic Model Checking for Discrete Time Models, pages 129–145. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1994.

[6] S. V. A. Campos, E. M. Clarke, W. R. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *RTSS*, pages 266–270. IEEE Computer Society, 1994.

[7] F. S. de Boer, M. M. Jaghoori, C. Laneve, and G. Zavattaro. Decidability problems for actor systems. In M. Koutny and I. Ulidowski, editors, *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 562–577. Springer, 2012.

[8] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. volume 4, pages 331–352, 1992.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[10] M. Geilen, S. Tripakis, and M. Wiggers. The earlier the better: a theory of timed actor interfaces. In M. Caccamo, E. Frazzoli, and R. Grosu, editors, *HSCC*, pages 23–32. ACM, 2011.

[11] E. Khamespanah, Z. S. Kaviani, R. Khosravi, M. Sirjani, and M.-J. Izadi. Timed-Rebeca Schedulability and Deadlock-Freedom Analysis Using Floating-Time Transition System. In *SPLASH*. ACM, 2012.

[12] F. Laroussinie, N. Markey, and P. Schnoebelen. Efficient timed model checking for discrete-time systems. *Theoretical Computer Science*, 353(1-3):249–271, 2006.

[13] F. Laroussinie, P. Schnoebelen, and M. Turuani. On the expressivity and complexity of quantitative branching-time temporal logics. *Theoretical Computer Science*, 297(1-3):297–315, 2003.

[14] B. Magnusson. Simulation-Based Analysis of Timed Rebeca Using TeProp and SQL. Master's thesis, Reykjavï£¡k University, School of Computer Science, Iceland, 2012. http://rebeca.cs.ru.is/files/MasterThesisBrynjarMagnusson2012.pdf.

[15] M. Nykänen and E. Ukkonen. The exact path length problem. *J. Algorithms*, 42(1):41–53, 2002.

[16] Z. Sabahi-Kaviani, R. Khosravi, M. Sirjani, P. C. Ölveczky, and E. Khamespanah. Formal semantics and analysis of timed rebeca in real-time maude. In C. Artho and P. C. Ölveczky, editors, *FTSCS*, volume 419 of *Communications in Computer and Information Science*, pages 178–194. Springer, 2013.

[17] Z. Sharifi, M. Mosaffa, S. Mohammadi, and M. Sirjani. Functional and performance analysis of network-on-chips using actor-based modeling and formal verification. *ECEASST*, 66, 2013.

[18] M. Sirjani and M. M. Jaghoori. Ten Years of Analyzing Actors: Rebeca Experience. In G. Agha, O. Danvy, and J. Meseguer, editors, *Formal Modeling: Actors, Open Systems, Biological Systems*, volume 7000 of *Lecture Notes in Computer Science*, pages 20–56. Springer, 2011.

[19] M. Sirjani, A. Movaghar, A. Shali, and F. S. de Boer. Modeling and Verification of Reactive Systems using Rebeca. *Fundam. Inform.*, 63(4):385–410, 2004.