Finite Interval-Time Transition System for Real-Time Actors

Shaghayegh Tavassoli, Ramtin Khosravi, Ehsan Khamespanah

Finite Interval-Time Transition System for Real-Time Actors

Shaghayegh Tavassoli[1], Ramtin Khosravi[1], and Ehsan Khamespanah[1]

School of Electrical and Computer Engineering, University of Tehran - Iran

Abstract. Real-time computer systems are software or hardware systems which have to perform their tasks according to a time schedule. Formal verification is a widely used technique to make sure if a real-time system has correct time behavior. Using formal methods requires providing support for non-deterministic specification for time constraints which is realized by time intervals. Timed-Rebeca is an actor-based modeling language which is equipped with a verification tool. The values of timing features in this language are positive integer numbers and zero (discrete values). In this paper, Timed-Rebeca is extended to support modeling timed actor systems with time intervals. The semantics of this extension is defined in terms of Interval-Time Transition System (ITTS) which is developed based on the standard semantics of Timed-Rebeca. In ITTS, instead of integer values, time intervals are associated with system states and the notion of shift equivalence relation in ITTS is used to make the transition system finite. As there is a bisimulation relation between the states of ITTS and finite ITTS, it can be used for verification against branching-time properties.

Keywords: Actor Model, Timed Rebeca, Interval-Time Transition System, Bisimulation Relation

1 Introduction

Real-time computer systems, are hardware or software systems which work on the basis of a time schedule [15]. Controller of car engines, networks of wireless sensors and actuators, and multimedia data streaming applications are examples of real-time systems. The correct behavior of real-time systems is achieved by correctness of calculated values as well as the time those values were produced [14, 22]. So, verification of a real-time system requires considering both the functional and time behavior of the systems [15]. In the modeling of real-time systems, presenting nondeterministic time behavior may be required. For example, the best response time of drivers when braking (with a high probability of signal prediction) was reported as a value in the range of 0.7 to 0.75 seconds [8]. Using time intervals is a widely used notion to model such behavior. Supporting time intervals for nondeterministic time behavior raises challenges in the schedulability analysis of real-time systems [16].

Examples of nondeterministic time behavior in real-time systems include the network delay in communication systems, driver's reaction time when braking a car, and the execution time of programs on processors. In such cases, determining the exact duration of the tasks is not always possible. Instead, a time interval can be used to specify that the time value will be within a specific range. It is well-known that Worst Case Execution Time (WCET) analysis does not necessarily show the worst time behavior [7]. Hence, using the upper bounds for the actions does not help when analyzing time-sensitive safety properties. In Timed Rebeca, the values of timing primitives are discrete values [12, 13]. Therefore, it is useful to propose an extension on Timed Rebeca for modeling and analyzing real-time systems with time intervals.

Using formal methods, in general, and model checking [6] in particular, is a verification technique which guarantees correctness of systems. Model checking tools for real-time models exhaustively explore state spaces of systems to make sure that given properties hold in all possible executions of the system and specified time constraints are satisfied. *Timed Automata* and *UPPAAL* are widely used for modeling and model checking of real-time systems [4]. Timed Automata are automata in which clock constraints (i.e. time interval constraints associated with clocks) can be associated with both transition guards and location invariants [3].

In the context of distributed systems, *Actor* is used to model systems composed of a number of distributed components communicating via message passing [2, 10]. There are some extensions on the Actor model for modeling real-time systems, e.g. *Timed Rebeca* [17] and *Creol* [5], which provide model checking for timed properties such as schedulability and deadlock freedom. Creol models are transformed to timed automata for the model checking purpose which suffers from support for simple expressions for time constrains and state space explosion for middle-sized models. In contrast, Timed Rebeca provides direct model checking toolset; but, only for models with discrete timing primitives [11, 12] as described in detail in Section 2.

In this paper, we extend Timed Rebeca to support modeling and analysis of real-time systems with time intervals (Section 3). To this end, the notion of Interval-Time Transition System (ITTS) is introduced, which can serve as the basis for the semantic description of timed actor systems. Here, we formally describe the semantics of Timed Rebeca based on ITTS in Section 4. In the second step, we illustrate how the notion of *shift equivalence relation* for ITTS is used to make the transition system finite, if possible. Using bisimulation method, in Section 5, it is proved that shift equivalence relation can be used for detecting equivalent system states in ITTS.

2 Timed Actors

A well-established paradigm for modeling concurrent and distributed systems is the *Actor* model. Actor is introduced by Hewitt [10] as an agent-based language and is later developed as a model of concurrent computation by Agha [2] . *Actors*

are seen as the universal primitives of concurrent computation, such that each actor provides some services which can be requested by other actors by sending messages to it. Execution of a service of an actor may result in changing the state of the actor and sending messages to some other actors. In [1], Timed Rebeca is introduced as an extension on the actor model for modeling of real-time systems.

2.1 Timed Rebeca

Timed Rebeca is introduced in [1] as the real-time extension of the Rebeca modeling language [18, 19, 21]. We explain Timed Rebeca using the example of a simple Ping-Pong model (taken from [20] with slight modifications). In this model, the ping actor sends *pong* message to the pong actor and the pong actor sends *ping* message to the ping actor. A Timed Rebeca model consists of a number of *reactive classes*, each describing the type of a number of *actors* (also called *rebecs*)[1]. There are two reactive classes `PingClass` and `PongClass` in the Ping-Pong model (Listing 1 lines 1 and 16). Each reactive class declares a set of *state variables* (e.g. lines 5-7). The local state of each actor is defined by the values of its state variables and the contents of its message bag. Communication in Timed Rebeca models takes place by asynchronous message passing among actors. Each actor has a set of *known rebecs* to which it can send messages. For example, an actor of type `PingClass` knows an actor of type `PongClass` (line 3), to which it can send `pong` message (line 12). Reactive classes may have some constructors that have the same name as the declaring reactive class and do not have a return value (lines 8-10 for `PingClass`). They may initialize actor's state variables and put the initially needed messages in the *message bag* of that actor (line 9). The way of responds to a message is specified in a *message server* which are methods defined in reactive classes. An actor can change its state variables through assignment statements, makes decisions through conditional statements, communicates with other actors by sending messages (e.g., line 12), and performs periodic behavior by sending messages to itself. Since communication is asynchronous, each actor has a *message bag* from which it takes the next incoming message. The ordering of the messages in a message bag is based on the arrival times of messages. An actor takes the first message from its message bag, executes its corresponding message server in an isolated environment, takes the next message (or waits for the next message to arrive) and so on.

Finally, the `main` block is used to instantiate the actors of the model. In the Ping-Pong model, two actors are created receiving their known rebecs and the parameter values of their constructors upon instantiation (lines 26-27).

```
1  reactiveclass PingClass(3) {
2      knownrebecs {
3          PongClass pong1;
4      }
5      statevars {
6          //e.g. int var1, var2;
7      }
8      PingClass() {
9          self.ping();
10     }
```

[1] In this paper we use rebec and actor interchangeably.

```
11      msgsrv ping() {
12          pong1.pong() after(1);
13          delay(2);
14      }
15  }
16  reactiveclass PongClass(3) {
17      knownrebecs {
18          PingClass ping1;
19      }
20      msgsrv pong() {
21          ping1.ping() after(1);
22          delay(1);
23      }
24  }
25  main {
26       PingClass pi(po) : ();
27       PongClass po(pi) : ();
28  }
```

Listing. 1. The Timed Rebeca model of the Ping-Pong model

Timed Rebeca adds three primitives to Rebeca to address timing issues: *delay*, *deadline* and *after*. A *delay* statement models the passage of time for an actor during execution of a message server (line 13). Note that all other statements of Timed Rebeca are assumed to execute instantaneously. The keywords *after* and *deadline* are used in conjunction with a method call. The term `after(n)` indicates that it takes n units of time for a message to be delivered to its receiver (lines 12 and 21). The term `deadline(n)` expresses that if the message is not taken in n units of time, it will be purged from the receiver's message bag automatically [1].

2.2 Timed Transition Systems

One way of modeling real-time systems is using timed transition systems [9]. The Semantics of Timed Rebeca is defined in terms of timed transition systems in [11]. In this semantics, the global state of a Timed Rebeca model is represented by a function that maps actors' ids to tuples. A tuple contains, the state variables valuations, the content of message bags, local time, the program-counter which shows the position of the statements which will be executed to finish the service to the message currently being processed, and the time when the actor resumes execution of the remained statements.

Transitions between states occur as the results of actors' activities including: taking a message from the mailbox, continuing the execution of statements, and progress of time. In timed transition syatem of Timed Rebeca, progress of time is only enabled when the other actions are disabled for all of the actors. This rule performs the minimum required progress of time to make one of the other rules enabled. As a result, the model of progress of time in the timed transition system of Timed Rebeca is deterministic. The detailed SOS rules of transition relations are defined in [11].

Figure 1 shows the beginning part of the timed transition system of the Ping-Pong example. In this figure, σ denotes the next statement in the body of the message server which is being processed. The pair of $\sigma = \langle ping, 1\rangle$ in the second state of Figure 1 shows that `pi` executed the statements of its `ping` message server up to the statement in line 1. In each state, r is the time when the actor can resume the execution of its remained statements. The first enabled actor of the model is `pi` (as its corresponding reactive class `PingClass` has a constructor which puts message `ping` in its bag, line 9), so, the first possible

transition is taking message `ping`. As shown in the detailed contents of the second state (the gray block), taking the message `ping` results in setting the values of σ and r for the actor `pi`. The next transition results in executing the first statement of the message server `ping` and results in putting the message `pong` in the bag of the actor `po` with release time 1 (because of the value of `after` in line 7). Note that the deadline for mesages in this model is ∞ as no specific value is set as the deadline for these messages. As the next statement of the message server `ping` is a delay statement, `pi` cannot continue the execution. The actor `po` cannot cause a transition too. So, the only possible transition is progress of time which is by 1 unit from the third to the fourth state.

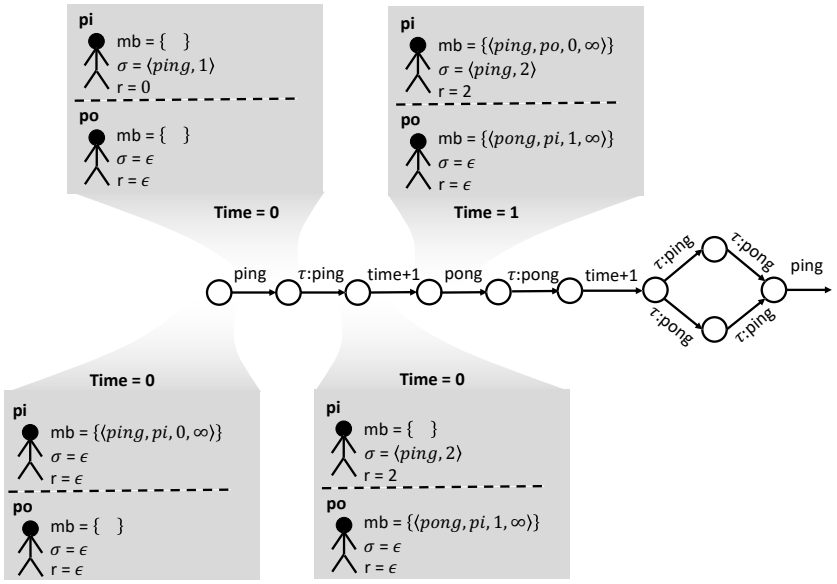

Fig. 1. The beginning part of the transition system of the Ping-Pong example [20].

3 Timed Rebeca with Intervals

The time interval extension to Timed Rebeca enables the use of time intervals with the timing directive `after`. In this model, a time interval is associated with each state of the transition system. To simplify the presentation of this paper, the time features `delay` and `deadline` are omitted. The modified version of Listing 1 which contains time interval for `after` directive is presented in Listing 2. Note that in Line 9, `after([4,8))` means that message `pong` arrives its destination during $[4, 8)$ time units after it has been sent. This value models the nondeterministic delay of the network in message delivery.

```
1  reactiveclass PingClass(3) {
2      knownrebecs {
3          PongClass po;
4      }
5      PingClass() {
6          self.ping();
7      }
8      msgsrv ping() {
9          po.pong() after([4,8));
10     }
11 }
12 reactiveclass PongClass(3) {
13     knownrebecs {
14         PingClass pi;
15     }
16     PongClass() {
17         self.pong();
18     }
19     msgsrv pong() {
20         pi.ping() after([4,8));
21     }
22 }
23 main {
24     PingClass pi(po) : ();
25     PongClass po(pi) : ();
26 }
```

Listing. 2. The Timed Rebeca model of the Ping-Pong model with time intervals

4 Semantics of Timed Rebeca with Intervals

We define the semantics of a Timed Rebeca with Intervals model as an ITTS which is based on the usual transition system semantics for actor systems [11]. Although in timed transition systems (TTS) time intervals can be used as timing constraints on transitions [9], describing the semantics of Timed Rebeca with intervals using TTS makes the semantics complicated, as it does not fit the timing model of TTS. Also, using such semantics as a basis for state space generation and model analysis may result in performance overheads. Thus, it is necessary to define a semantics for Timed Rebeca with intervals which naturally reflects its timing model.

The states in ITTS are composed of the local states of the actor in the system. A key idea behind ITTS is to associate with each state a time interval during which the local states of the actors do not change. The transitions are of two types, namely *message processing* and *time progress*. The former includes taking or processing of a message by an actor (which changes the local state of the actor). Time progress transition changes the state of the system by increasing the left end-point of the time interval of the state.

To keep the semantics description focused on timed behavior, we assume the messages do not have parameters, as they do not affect the time behavior of the model. Otherwise, the semantics rules will be cluttered with actual parameters evaluation and scope management.

Before we describe the formal semantics of ITTS, we introduce a few notations that are used throughout the paper.

4.1 Notation and Basic Definitions

We use the notation *TInterval* (ranged over by α, β, and γ) to denote the set of all time intervals in $R_{\geq 0}$ which are left-closed right-open or left-closed right-closed. Such intervals are written as $[t_1, t_2)$ and $[t_1, t_2]$ respectively. Time interval $[t_1, t_1]$

corresponds to time value t_1 of Timed Rebeca. For $\alpha \in \mathit{TInterval}$, α_ℓ and α_r denotes the left and the right endpoints of α respectively (regardless of being right-open or right-closed). The notation $\alpha_{\ell \leftarrow x}$ (resp. $\alpha_{r \leftarrow x}$) denote the interval obtained from replacing the left (resp. right) endpoint of α with x, e.g., $[t_1, t_2)_{r \leftarrow x}$ is $[t_1, x)$.

For a function $f : X \to Y$, we use the notation $f[x \mapsto y]$ to denote the function $\{(a, b) \in f | a \neq x\} \cup \{(x, y)\}$. We also use the notation $x \mapsto y$ as an alternative to (x, y).

The following notations is used for working with sequences. Given a set A, the set A^* is the set of all finite sequences over elements of A, ranged over by σ and σ'. For a sequence $\sigma \in A^*$ of length n, the symbol a_i denotes the i^{th} element of the sequence, where $1 \leq i \leq n$. In this case, we may also write σ as $\langle a_1, a_2, \ldots, a_n \rangle$. The empty sequence is represented by ϵ, and $\langle h | T \rangle$ denotes a sequence whose first element is $h \in A$ and $T \in A^*$ is the sequence comprising the elements in the rest of the sequence. For $x \in A$ and $\sigma, \sigma' \in A^*$, the expressions $x \oplus \sigma$ and $\sigma \oplus x$ denote a sequence obtained from prepending and appending x to σ respectively. Also, $\sigma \oplus \sigma'$ is the sequence obtained by concatenating σ' to the end of σ. The *deletion* operator is defined such that $\sigma \ominus x$ is the sequence obtained by removing the first occurrence of x in σ. Finally, we define the membership operator for sequences as $x \in \langle a_1, a_2, \ldots, a_n \rangle \stackrel{def}{=} \exists 1 \leq i \leq n \cdot a_i = x$.

VName is defined as the set of all variable names in the model, and *Value* as the set of all values the state variables can take. We do not address typing issues in this paper to keep the focus on time behavior of the actors. Let *Stat* denote the set of all statements appearing in the actors' message handlers, *MName* denote the set of all message names, and *ActorID* denote the set of all unique identities of the actors. The function $\mathit{body} : \mathit{ActorID} \times \mathit{MName} \to \mathit{Stat}$ is defined such that $\mathit{body}(x, m)$ returns the sequence of statements in the body of the message handler of m in the actor with ID x. The set *Act* is defined as $\mathit{MName} \cup \{\tau\} \cup \{\mathit{TP}\}$.

4.2 Messages and Message Bags

A message in ITTS is a tuple $(\mathit{sID}, \mathit{rID}, m, \alpha)$, where *rID* and *sID* denote the receiver ID and the sender ID respectively, m denotes the message name, while α denotes the "after" interval. It means that this message arrives its destination during α time units after it has been sent. The set of all messages is defined as *Msg*. In ITTS, each actor has an associated message bag that holds all its received messages. The set of all message bags is defined as $\mathit{Bag}(\mathit{Msg}) = \mathit{Msg}^*$. For the notation to be simpler, message bags are written in the form of sequences. But it is not necessary for message bags to be sequences, as they can be defined as multisets in general.

4.3 States in ITTS

The system state is composed of the local states of the actors in the system. The local state of an actor with ID x, is defined as the triple (v_x, mb_x, σ_x),

where v_x is the valuation function of the state variables of the actor, mb_x is the message bag of the actor, and σ_x denotes the sequence of statements the actor is going to execute in order to finish the processing of the message being handled. The set of all local states of the actors is denoted as $\mathit{ActorState} = (\mathit{VName} \rightarrow \mathit{Value}) \times \mathit{Bag}(\mathit{Msg}) \times \mathit{Stat}^*$.

A (global) system state in ITTS is a tuple (s, α), where $s \in \mathit{ActorID} \rightarrow \mathit{ActorState}$ is a function mapping each actor ID to its local state, and $\alpha \in \mathit{TInterval}$ is the time interval associated with the system state. It is assumed that S is the set of all possible system states, ranged over by gs (short for *global state*). In the initial system state, we let the time interval of the system state to be $[0, 0]$.

4.4 Order of Events in ITTS

As stated before, to define *time progress* transitions, we must determine the earliest time in which a change is possible in the local state of an actor (called an *event*). As such changes happen only as a result of taking or executing a message, the earliest and the latest times the messages can be taken determine the order of events in the system. To specify the ordering of the events, a message bag is defined for every system state called "system state message bag", denoted by $B(gs)$, which consists of all messages in message bags of all actors in that system state (which may contain duplicate messages).

The ordering of the events regarding to a state $gs \in S$ is denoted by $EE_i(gs)$ which is the i^{th} smallest value in the set of all lower and uppoer bounds of the time intervals of all messages in $B(gs)$.

4.5 Transitions Definition

As explained before, transitions in ITTS are classified into two major types: *message processing* and *time progress*. The former includes taking a message from message bag or processing a message by an actor. As with Timed Rebeca, we assume executing the statements in a message server is instantaneous.

Message Processing For a system state (s, α) we call an actor x *idle*, if it has no remaining statement to execute from its previously taken message. If such an actor has a message in its mailbox whose *after* interval starts from α_ℓ, a message processing transition can take place:

$$\frac{gs = (s, \alpha) \in S \wedge s(x) = (v, mb, \epsilon) \wedge msg = (y, x, m, \beta) \in mb \wedge \beta_\ell = \alpha_\ell}{gs \xrightarrow{m} (s[x \mapsto (v \cup \{(\mathit{self}, x), (\mathit{sender}, y)\}, mb \ominus msg, \mathit{body}(x, msg))], \alpha)} \quad (1)$$

The execution of each statement in the body of a message handler is considered an internal action in ITTS. The statements such as assignments, conditionals and loops only alter the local state of the executing actor. The only statement that affects the state of other actors is *send* which may put a message in another

actor's message bag. The semantics of send and assignment statements are stated here and the others are left out to save space.

An assignment statement of the form $var = expr$ overrides the value of var in the executing actor's state variables to the value of the expression $expr$, denoted by $eval_x(expr)$. To keep the description simple, it is assumed that the message servers do not have local variables, so the left side of the assignment is always a state variable of the actor executing the message server.

$$\frac{gs = (s, \alpha) \in S \wedge s(x) = (v, mb, \langle var = expr | \sigma \rangle)}{gs \rightarrow (s[x \mapsto (v[var \mapsto eval_x(expr)], mb, \sigma)], \alpha)} \tag{2}$$

The send statement $y.m()after\ \gamma$ denotes sending a message m with the after interval γ to the receiver y. The semantics of the send statement is defined as

$$\frac{gs = (s, \alpha) \in S \wedge s(x) = (v, mb, \langle y.m()after\ \gamma | \sigma \rangle) \wedge s(y) = (v', mb', \sigma')}{gs \rightarrow (s[x \mapsto (v, mb, \sigma)][y \mapsto (v', mb' \oplus (x, y, m, \beta), \sigma')], \alpha_{r \leftarrow t})} \tag{3}$$

where β is the interval whose left and right endpoints are $\alpha_\ell + \gamma_\ell$ and $\alpha_r + \gamma_r$ respectively, and is right-closed if both α and γ are right-closed and is right-open otherwise. Furthermore, $t = min(\alpha_r, \beta_\ell)$ which means that after the message is sent, β_ℓ may be the second earliest event, replacing α_r in that case.

Time Progress In ITTS, two types of time progress transitions are defined. The first type corresponds to the case that the only possible transition is time progress, while for the second type, a nondeterministic choice between time progress and another transition is enabled.

Type 1 time progress transition: If the lower bound of the time interval for a system state gs is smaller than the earliest event of that system state ($\alpha_\ell < EE_1(gs)$), the only possible transition is time progress. After executing Type 1 time progress transition, the time interval for the successor state is $[EE_1(gs), EE_2(gs))$.

$$\frac{gs = (s, \alpha) \wedge \alpha_\ell < EE_1(gs)}{gs \overset{TP}{\rightarrow} (s, [EE_1(gs), EE_2(gs))} \tag{4}$$

Type 2 time progress transition: In ITTS, in situations which in a system state, time progress transition and at least one other transition are enabled, second type of time progress transition can occur. Consider a global state $gs = (s, \alpha)$. Unlike Type 1 transitions we have $\alpha_\ell = EE_1(gs)$, so there exists a message msg_1 in the system with interval β such that $\alpha_\ell = \beta_\ell$. Now, if no other message exists which can be taken before β_r, the only possible transition from gs is to take msg_1 and there is no time progress transition. Hence, there must be a message msg_2 in the system with interval γ such that $EE_2(gs) = \gamma_\ell$ and $\beta_\ell \leq \gamma_\ell < \beta_r$. In such a state gs, two transitions are possible: taking msg_1, and waiting till γ_ℓ. Note that this nondeterminism enables the interleaving of processing msg_1 and msg_2.

To model this type of time progress, we shift the lower bound of the time interval of msg_1 from β_ℓ to γ_ℓ and update the time interval of the global state

accordingly. Note that there may be multiple messages that start from β_ℓ. Hence, we define the function *ds* (short for *delay starts*), such that $ds(mb, t)$ changes the lower bound of the messages in mb which start earlier than t to t. Formally,

$$ds(\epsilon, t) = \epsilon$$
$$ds(\langle(x, y, m, \alpha|T\rangle, t) = \langle(x, y, m, \alpha_{\ell \leftarrow max(\alpha_\ell, t)})|ds(T, t)\rangle$$

We lift the definition of *ds* to the function s which returns the local state of the actors:

$$ds(s, t) = \{x \mapsto (v, ds(mb, t), \sigma)|x \mapsto (v, mb, \sigma) \in s\}$$

Now, we can define Type 2 time progress transitions as below.

$$\frac{gs = (s, \alpha) \wedge \alpha_\ell = EE_1(gs) \wedge (x, y, m, \gamma) \in B(gs) \wedge \gamma_\ell = EE_2(gs)}{gs \xrightarrow{TP} (ds(s, EE_2(gs)), [EE_2(gs), EE_3(gs))} \quad (5)$$

Figure 2 shows part of the ITTS of the Ping-Pong example of Listing 2 which is generated based on the proposed semantics of this section.

5 Making State Space Finite

Based on the semantics of Timed Rebeca, there is no explicit time reset operator in the language; so, the progress of time results in an infinite number of states in ITTS of models. However, reactive systems which generally show periodic or recurrent behaviors are modeled using Timed Rebeca, i.e. performing recurrent behaviors over infinite time. This fact enables us to propose the notion for equivalence relation between two states with time intervals, aiming to make ITTSs finite, called *shift equivalence relation in ITTS*. The idea of defining shift equivalence relation in states of ITTSs is inspired from [12]. Intuitively, in the shift equivalence relation two states are equivalent if and only if they are the same except for the parts related to the time and the timed parts can be mapped by shifting them with an specific value.

5.1 Shift Equivalence Relation in ITTS

In the first step, the shift equivalence of two time intervals with distance $c \in R$ is defined as:

$$\alpha \approx_c \alpha' \stackrel{def}{=} \alpha'_\ell = \alpha_\ell + c \wedge \alpha'_r = \alpha_r + c \quad (6)$$

Then, the shift equivalence of two messages with distance $c \in R$ is defined as:

$$\forall sID, rID \in ActorID, m \in MName, \beta, \beta' \in TInterval \cdot$$
$$(sID, rID, m, \beta) \approx_c (sID, rID, m, \beta') \Leftrightarrow \beta \approx_c \beta' \quad (7)$$

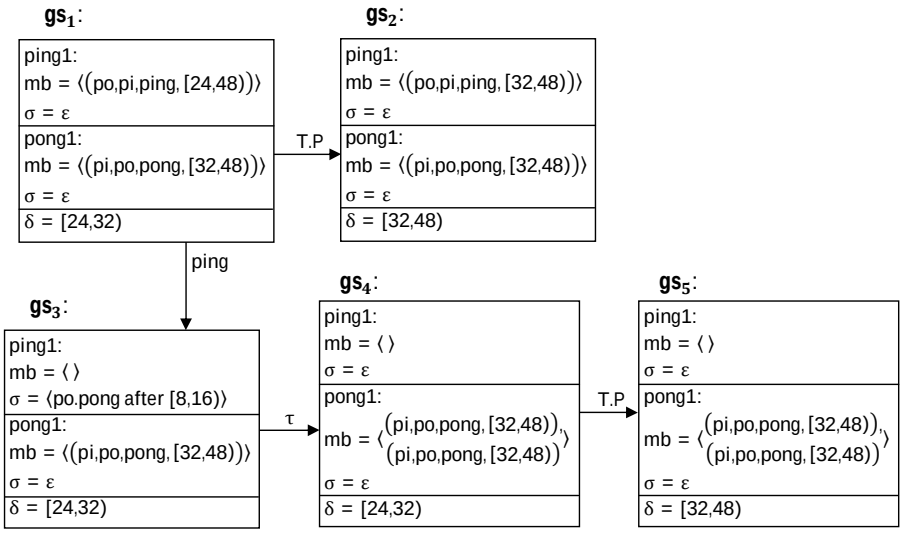

Fig. 2. A part of the interval time transition system of the Ping-Pong example. The transition from gs_1 to gs_3 is a take message transition. The transition from gs_3 to gs_4 is an internal transition. The transition from gs_4 to gs_5 is a Type 1 time progress. The transition from gs_1 to gs_2 is a Type 2 time progress.

Using Equation 7, the shift equivalence of two message bags $B = \langle m_1, ..., m_n \rangle$ and $B'\langle m'_1, ..., m'_n \rangle$ with distance $c \in R$ is defined as:

$$B \approx_c B' \Leftrightarrow \exists 1 \leq i \leq n, 1 \leq i' \leq n \cdot m_i \approx_c m'_i \wedge B \ominus m_i \approx_c B' \ominus m'_{i'} \quad (8)$$

Now, the shift equivalence of two local states of an actor with ID x with distance $c \in R$ is defined as:

$$\big(s(x) = (v_x, mb_x, \sigma_x)\big) \equiv_c \big(s'(x) = (v'_x, mb'_x, \sigma'_x)\big) \Leftrightarrow \\ v_x = v'_x \wedge \sigma_x = \sigma'_x \wedge mb_x \approx_c mb'_x \quad (9)$$

Consequently, the shift equivalence of two system states gs and gs' ($gs, gs' \in S$) with distance $c \in R$ is defined as:

$$\big(gs = (s, \alpha)\big) \equiv_c \big(gs' = (s', \alpha')\big) \Leftrightarrow \alpha \approx_c \alpha' \wedge \forall x \in \mathit{ActorID}.s(x) \equiv_c s'(x) \quad (10)$$

5.2 Shift Equivalence Relation in ITTS is a Bisimulation Relation

The shift equivalence relation aims to make ITTSs of models finite. To this end, we have to show that there is a timed bisimulation relation between finite and infinite ITTSs of a given model to prove that they preserve the same set of timed branching-time properties (i.e., $\equiv_c$ is a bi-simulation relation). To this end, the following theorem should be proven:

Theorem 1. $\forall(gs_1, gs'_1) \in \equiv_c$ and $\forall a \in Act$:

$$\forall gs_2 \in S.gs_1 \xrightarrow{a} gs_2 \Rightarrow \exists gs'_2 \in S.gs'_1 \xrightarrow{a} gs'_2 \wedge (gs_2, gs'_2) \in \equiv_c \tag{11}$$

$$\forall gs'_2 \in S.gs'_1 \xrightarrow{a} gs'_2 \Rightarrow \exists gs_2 \in S.gs_1 \xrightarrow{a} gs_2 \wedge (gs_2, gs'_2) \in \equiv_c \tag{12}$$

To prove the first condition of bisimulation relation (i.e., Equation (11)) it has to be proven that it holds for for all transition types in ITTS.

Take Message Transitions: Assume that $gs_1 = (s_1, \alpha)$, $s_1(x) = (v_x, mb_x, \epsilon)$, $msg = (y, x, m, \beta) \in mb_x$ and $\beta_\ell = \alpha_\ell$. Thus take message transition is enabled in gs_1 and $gs_1 \xrightarrow{m} gs_2$. Assume that $gs'_1 = (s'_1, \alpha')$, $s'_1(x) = (v'_x, mb'_x, \epsilon)$ and $gs_1 \equiv_c gs'_1$. Therefore $\alpha'_\ell = \alpha_\ell + c$. From $s_1(x) \equiv_c s'_1(x)$, it can be concluded that a message msg' exists in mb'_x such that $msg \approx_c msg'$. Thus msg' is of the form (y, x, m, β') such that $\beta'_\ell = \alpha'_\ell$ and $\beta'_r = \beta_r + c$. Therefore take message transition is enabled in gs'_1. Hence, gs'_2 exists in ITTS such that $gs'_1 \xrightarrow{m} gs'_2$. From equation (1), It can be concluded that:

$$gs_2 = (s_2, \alpha) = (s_1[x \mapsto (v_x \cup A, mb_x \ominus msg, body(x, msg)], \alpha)$$
$$gs'_2 = (s'_2, \alpha') = (s'_1[x \mapsto (v'_x \cup A, mb'_x \ominus msg', body(x, msg')], \alpha')$$
$$A = \{(self, x), (sender, y)\}$$

To prove that $gs_2(x) \equiv_c gs'_2(x)$, the following points should be considered:

1. $s_1(x) \equiv_c s'_1(x) \Rightarrow v_x = v'_x$
   $\Rightarrow v_x \cup \{(self, x), (sender, y)\} = v'_x \cup \{(self, x), (sender, y)\}$
2. $mb_x \approx_c mb'_x \wedge msg \approx_c msg' \Rightarrow mb_x \ominus msg \approx_c mb'_x \ominus msg'$
3. $msg \approx_c msg' \Rightarrow body(x, msg) = body(x, msg')$
4. $gs_1 \equiv_c gs'_1 \Rightarrow \alpha \approx_c \alpha'$

Based on the above equations, $gs_2 \equiv_c gs'_2$ can be concluded, which is required for proving Equation 11.

Internal Transitions: Assuming that $gs_1 = (s_1, \alpha)$, $gs'_1 = (s'_1, \alpha')$, $gs_1 \equiv_c gs'_1$, two cases are possible:

- **Assignment Statement:** Assume that $s_1(x) = (v_x, mb_{1x}, \langle var = expr|\sigma_x\rangle)$. Therefore internal action is enabled in gs_1 and $gs_1 \xrightarrow{\tau} gs_2$. From $gs_1 \equiv_c gs'_1$, it can be concluded that $s_1(x) \equiv_c s'_1(x)$, so $s'_1(x) = (v_x, mb'_{1x}, \langle var = expr|\sigma_x\rangle)$. Thus internal action is enabled in gs'_1 and $gs'_1 \xrightarrow{\tau} gs'_2$. The produced next states gs_2 and gs'_2 are:

$$gs_2 = (s_1[x \mapsto (v_x[var \mapsto eval_x(expr)], mb_{1x}, \sigma_x)], \alpha)$$
$$gs'_2 = (s'_1[x \mapsto (v_x[var \mapsto eval_x(expr)], mb'_{1x}, \sigma_x)], \alpha')$$

To prove that $gs_2 \equiv_c gs'_2$, the following points should be considered:

1. $gs_1 \equiv_c gs'_1 \Rightarrow \alpha \approx_c \alpha'$
2. $s_1(x) \equiv_c s'_1(x) \Rightarrow mb_{1x} \approx_c mb'_{1x}$

On the basis of the above results (1,2), $gs_2 \equiv_c gs'_2$ can be concluded.

– **Send Statement:** Assume that $s_{1x} = (v_x, mb_{1x}, \langle y.m()after\ \gamma|\sigma_x\rangle)$ such that γ_ℓ and γ_r are relative time values and $s_{1y} = (v_y, mb_{1y}, \sigma_y)$. After executing send statement in gs_1, $msg = (x, y, m, \beta)$ will be appended to mb_{1y} such that $\beta_\ell = \alpha_\ell + \gamma_\ell$ and $\beta_r = \alpha_r + \gamma_r$. From $gs_1 \equiv_c gs'_1$, it can be concluded that $s'_{1x} = (v_x, mb'_{1x}, \langle y.m()after\ \gamma|\sigma_x\rangle)$ and $s'_{1y} = (v_y, mb'_{1y}, \sigma_y)$ exist in ITTs. After executing send statement in gs'_1, $msg' = (x, y, m, \beta')$ will be appended to mb'_{1y} such that $\beta'_\ell = \alpha'_\ell + \gamma_\ell$ and $\beta'_r = \alpha'_r + \gamma_r$:

$$mb_{2y} = mb_{1y} \oplus msg = mb_{1y} \oplus (x, y, m, \beta)$$
$$mb'_{2y} = mb'_{1y} \oplus msg' = mb'_{1y} \oplus (x, y, m, \beta')$$

Time interval of gs_2 is $\alpha_{r\leftarrow t}$ such that $t = min(\alpha_r, \beta_\ell)$ and Time interval of gs'_2 is $\alpha'_{r\leftarrow t'}$ such that $t' = min(\alpha'_r, \beta'_\ell)$:

$$gs_2 = (s_2, \alpha_{r\leftarrow t}) = (s_1[x \mapsto (v_x, mb_{1x}, \sigma_x)][y \mapsto (v_y, mb_{2y}, \sigma_y)], \alpha_{r\leftarrow t})$$
$$gs'_2 = (s'_2, \alpha'_{r\leftarrow t'}) = (s'_1[x \mapsto (v_x, mb'_{1x}, \sigma_x)][y \mapsto (v_y, mb'_{2y}, \sigma_y)], \alpha'_{r\leftarrow t'})$$

To prove the equivalency of gs_2 and gs'_2, the following points should be considered:

1. $s_2(x) \equiv_c s'_2(x)$
2. $msg \approx_c msg' \wedge mb_{1y} \approx_c mb'_{1y} \Rightarrow mb_{2y} \approx_c mb'_{2y} \Rightarrow s_2(y) \equiv_c s'_2(y)$
3. $\alpha_{r\leftarrow t} \approx_c \alpha'_{r\leftarrow t'}$

On the basis of the above results (1-3), it can be concluded that $gs_2 \equiv_c gs'_2$.

So, in both cases of internal transitions, $gs_2 \equiv_c gs'_2$ as required for proving Equation 11.

Time Progress Transition: In ITTS, two types of time progress transitions were defined (Equations 4 and 5). So, it has to be proven that the first condition holds for these two types. In the following, we assume that $gs_1 = (s_1, \alpha)$, $gs'_1 = (s'_1, \alpha')$, $gs_1 \equiv_c gs'_1$.

– **Type 1 Time Progress Transition:** Assume that Type 1 time progress transition is enabled in gs_1. On the basis of (4), $\alpha_\ell < EE_1(gs_1)$ and $gs_1 \overset{TP}{\rightarrow} gs_2$. From $gs_1 \equiv_c gs'_1$, it can be concluded that $\alpha'_\ell < EE_1(gs'_1)$. Therefore Type 1 time progress transition is enabled in gs'_1 and $gs'_1 \overset{TP}{\rightarrow} gs'_2$. According to equation (4), gs_2 and gs'_2 are of the following forms:

$$gs_2 = (s_1, [EE_1(gs_1), EE_2(gs_1)))$$
$$gs'_2 = (s'_1, [EE_1(gs'_1), EE_2(gs'_1)))$$

To prove that $gs_2 \equiv_c gs'_2$, the following points should be considered:

1. $gs_1 \equiv_c gs'_1 \Rightarrow s_1 \equiv_c s'_1$

2. $gs_1 \equiv_c gs'_1 \Rightarrow EE_1(gs'_1) = EE_1(gs_1) + c \wedge EE_2(gs'_1) = EE_2(gs_1) + c \Rightarrow [EE_1(gs_1), EE_2(gs_1)) \approx_c [EE_1(gs'_1), EE_2(gs'_1))$

On the basis of the above results (1,2), $gs_2 \equiv_c gs'_2$ can be concluded.

– **Type 2** Time Progress Transition: Assume that Type 2 time progress transition is enabled in gs_1. On the basis of equation (5), $\alpha_\ell = EE_1(gs_1)$ and message (x, y, m, γ) exists in $B(gs_1)$ such that $\gamma_\ell = EE_2(gs_1)$. Therefore $gs_1 \xrightarrow{TP} gs_2$. On the basis of $gs_1 \equiv_c gs'_1$, it can be concluded that $\alpha'_\ell = EE_1(gs'_1)$ and message (x, y, m, γ') exists in $B(gs'_1)$ such that $\gamma'_\ell = EE_2(gs'_1)$. Therefore Type 2 time progress transition is enabled in gs'_1 and $gs'_1 \xrightarrow{TP} gs'_2$. According to equation (5), gs_2 and gs'_2 are of the following forms:

$$gs_2 = (s_2, [EE_2(gs_1), EE_3(gs_1))) = (ds(s_1, EE_2(gs_1)), [EE_2(gs_1), EE_3(gs_1)))$$
$$gs'_2 = (s'_2, [EE_2(gs'_1), EE_3(gs'_1))) = (ds(s'_1, EE_2(gs'_1)), [EE_2(gs'_1), EE_3(gs'_1)))$$

To prove that $gs_2 \equiv_c gs'_2$, the following points should be considered:

1. $gs_1 \equiv_c gs'_1 \Rightarrow ds(s_1, EE_2(gs_1)) \equiv_c ds(s'_1, EE_2(gs'_1)) \Rightarrow s_2 \equiv_c s'_2$
2. $gs_1 \equiv_c gs'_1 \Rightarrow [EE_2(gs_1), EE_3(gs_1)) \approx_c [EE_2(gs'_1), EE_3(gs'_1))$

On the basis of the above results (1,2), it can be concluded that $gs_2 \equiv_c gs'_2$.

Therefore Equation 11 holds for both types of time progress transitions.

The proof of the second condition of the bisimulation relation (i.e. Equation 12) is almost the same as the first condition and omitted from this paper because of lack of space.

6 Conclusion

The correctness of behavior in real-time systems, depends on the calculated values and the time of producing theses values [14, 22]. In many real life applications, nondeterministic time behavior is present; In such circumstances, time intervals can be used to define the time behavior of a real-time system.

In this paper, a time interval extension to Timed Rebeca was presented. Timed Rebeca with intervals can be used for modeling real-time systems with nondeterministic time behavior. Using this method, the models of such real-time systems can be described with a high-level language and they can be efficiently analyzed. The semantics of Timed Rebeca with intervals models was defined as Interval Time Transition System (ITTS). In ITTS, every time feature is defined as an interval of non-negative real numbers . A time interval is associated with every system state. The semantics of ITTS was explained and messages, system states, and transitions for different action types were defined. Using the presented semantics, the state space of timed actor systems with time intervals could be generated. In order to determine equivalent system states, shift equivalence relation in ITTS was defined. Using bi-simulation method, it was proved that shift equivalence relation in ITTS could help detect equivalent system states. In many cases with finite time intervals, state space explosion could be prevented using shift equivalence relation in ITTS.

Other equivalence relations can be proposed in the future for detection of equivalent states in cases with infinite time intervals. Another line of research is implementation of the proposed semantics and testing its efficiency on actor models.

References

1. Aceto, L., Cimini, M., Ingólfsdóttir, A., Reynisson, A.H., Sigurdarson, S.H., Sirjani, M.: Modelling and simulation of asynchronous real-time systems using timed rebeca. In: Mousavi, M.R., Ravara, A. (eds.) Proceedings 10th International Workshop on the Foundations of Coordination Languages and Software Architectures, FOCLASA 2011, Aachen, Germany, 10th September, 2011. EPTCS, vol. 58, pp. 1–19 (2011), https://doi.org/10.4204/EPTCS.58.1
2. Agha, G.A.: ACTORS - a model of concurrent computation in distributed systems. MIT Press series in artificial intelligence, MIT Press (1990)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science 126(2), 183–235 (1994)
4. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., Yi, W.: UPPAAL - a Tool Suite for Automatic Verification of Real-Time Systems. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) Hybrid Systems. Lecture Notes in Computer Science, vol. 1066, pp. 232–243. Springer (1995)
5. Bjørk, J., Johnsen, E.B., Owe, O., Schlatte, R.: Lightweight time modeling in timed creol. In: Ölveczky, P.C. (ed.) Proceedings First International Workshop on Rewriting Techniques for Real-Time Systems, RTRTS 2010, Longyearbyen, Norway, April 6-9, 2010. EPTCS, vol. 36, pp. 67–81 (2010), https://doi.org/10.4204/EPTCS.36.4
6. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (1986), https://doi.org/10.1145/5397.5399
7. Gomes, L., Fernandes, J.M., Gomes, L., Fernandes, J.M.: Behavioral modeling for embedded systems and technologies: applications for design and implementation. Information Science Reference (2010)
8. Green, M.: "How long does it take to stop?"Methodological analysis of driver perception-brake times. Transportation human factors 2(3), 195–216 (2000)
9. Henzinger, T.A., Manna, Z., Pnueli, A.: Timed transition systems. In: de Bakker, J.W., Huizing, C., de Roever, W.P., Rozenberg, G. (eds.) Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings. Lecture Notes in Computer Science, vol. 600, pp. 226–251. Springer (1991), https://doi.org/10.1007/BFb0031995
10. Hewitt, C.: Description and theoretical analysis (using schemata) of PLANNER: A language for proving theorems and manipulating models in a robot. MIT Artificial Intelligence Technical Report 258, Department of Computer Science, MIT (Apr 1972)
11. Khamespanah, E., Khosravi, R., Sirjani, M.: An efficient TCTL model checking algorithm and a reduction technique for verification of timed actor models. Sci. Comput. Program. 153, 1–29 (2018), https://doi.org/10.1016/j.scico.2017.11.004
12. Khamespanah, E., Sirjani, M., Sabahi-Kaviani, Z., Khosravi, R., Izadi, M.: Timed rebeca schedulability and deadlock freedom analysis using bounded floating time transition system. Sci. Comput. Program. 98, 184–204 (2015), https://doi.org/10.1016/j.scico.2014.07.005

13. Khamespanah, E., Sirjani, M., Viswanathan, M., Khosravi, R.: Floating time transition system: More efficient analysis of timed actors. In: Braga, C., Ölveczky, P.C. (eds.) Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9539, pp. 237–255. Springer (2015), https://doi.org/10.1007/978-3-319-28934-2_13
14. Kopetz, H.: Real-Time Systems - Design Principles for Distributed Embedded Applications. Real-Time Systems Series, Springer (2011), https://doi.org/10.1007/978-1-4419-8237-7
15. Liu, J.W.: Real-Time Systems. Integre Technical Publishing Co., Inc. (2000)
16. Manolache, S., Eles, P., Peng, Z.: Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In: Proceedings 13th EUROMICRO conference on Real-time Systems. pp. 19–26. IEEE (2001)
17. Reynisson, A.H., Sirjani, M., Aceto, L., Cimini, M., Jafari, A., Ingólfsdóttir, A., Sigurdarson, S.H.: Modelling and simulation of asynchronous real-time systems using timed rebeca. Sci. Comput. Program. 89, 41–68 (2014), https://doi.org/10.1016/j.scico.2014.01.008
18. Sirjani, M., de Boer, F.S., Movaghar-Rahimabadi, A.: Modular verification of a component-based actor language. J. UCS 11(10), 1695–1717 (2005), https://doi.org/10.3217/jucs-011-10-1695
19. Sirjani, M., Jaghoori, M.M.: Ten years of analyzing actors: Rebeca experience. In: Formal Modeling: Actors, Open Systems, Biological Systems. Lecture Notes in Computer Science, vol. 7000, pp. 20–56. Springer (2011)
20. Sirjani, M., Khamespanah, E.: On time actors. In: Theory and Practice of Formal Methods. Lecture Notes in Computer Science, vol. 9660, pp. 373–392. Springer (2016)
21. Sirjani, M., Movaghar, A., Shali, A., De Boer, F.S.: Modeling and verification of reactive systems using rebeca. Fundamenta Informaticae 63(4), 385–410 (2004)
22. Stankovic, J.A., Spuri, M., Ramamritham, K., Buttazzo, G.C.: Deadline Scheduling for Real-Time Systems: EDF and Related Algorithms. The Springer International Series in Engineering and Computer Science, Springer US (2012), https://books.google.com/books?id=1TrSBwAAQBAJ