

# On-the-Fly Symbolic Model Checking for Real-Time Systems

Paper: Ahmed Bouajjani, Stavros Tripakis, Sergio Yovine  
Slides: Martin Milata

October 19, 2010

# Motivation – checking complex properties

We know how to determine reachability of state satisfying some proposition in timed automaton.

- Reachability by itself cannot express some interesting properties.
- These can often be stated as a formula  $\varphi$  of some logic. In this case, we need to be able to decide whether  $\mathcal{A} \models \varphi$ .

Example of such logic - TCTL (timed computation tree logic):

$$\varphi ::= \mathbf{false} \mid P \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \forall[\varphi_1 \mathbf{U}_{\sim c} \varphi_2] \mid \exists[\varphi_1 \mathbf{U}_{\sim c} \varphi_2]$$

for  $\sim \in \{<, \leq, =, \geq, >\}$ ,  $c \in \mathbb{N}$  and proposition  $P$ .

Reachability of  $P$  can be expressed as  $\exists[\neg\mathbf{false} \mathbf{U}_{\geq 0} P]$ .

# Motivation – dealing with state explosion

Model checking algorithms usually have to work with number of states exponential in the size of the checked system. Two approaches have proven to be helpful:

**On-the-Fly algorithms** – computing and storing in memory only the part of the state space that is needed at the moment.

**Symbolic algorithms** – working with some representation of sets of states instead of individual states (or individual regions in case of TAs).

Goal of this paper: On-the-Fly symbolic algorithm for model checking of  $\text{TECTL}_{\exists}^*$  logic in timed automata.

# Timed systems

Let  $C_{\mathcal{X}}$  be the set of clock constraints over  $\mathcal{X}$ . **Timed system** over set of labels  $\Sigma$  is 5-tuple

$$\mathcal{T} = (\mathcal{X}, \mathcal{Q}, \mathcal{E}, \Pi, \Gamma),$$

where:

- $\mathcal{X}$  is finite set of clocks,
- $\mathcal{Q}$  is finite set of control locations,
- $\mathcal{E}$  is set of edges of the form  $(q, g, \mathcal{X}, q')$ :
  - $q, q' \in \mathcal{Q}$  are source/destination,
  - $g$  is a clock constraint,
  - $\mathcal{X} \subseteq \mathcal{X}$  clocks to be reset,
- $\Pi : \mathcal{Q} \rightarrow \Sigma$  is labelling function,
- $\Gamma : \mathcal{Q} \rightarrow C_{\mathcal{X}}$  is function that assigns time-progress condition (invariant) to locations.

**Parallel composition** of two timed systems  $\mathcal{T}_1, \mathcal{T}_2$ , denoted  $\mathcal{T}_1 \parallel \mathcal{T}_2$  is timed system obtained by letting run  $\mathcal{T}_1, \mathcal{T}_2$  at the same time, taking transitions either individually or simultaneously.

**Region graph** of timed system  $\mathcal{T}$ , denoted  $RG(\mathcal{T})$ , is the set of nodes  $r = (q, g)$ , where  $q \in \mathcal{Q}$  and  $g$  the set of clock valuations forming a region, together with two types of edges:

- discrete:  $(q, g) \xrightarrow{e} (q', e(g))$
- timed:  $(q, g) \xrightarrow{\varepsilon} (q, g')$

**Region set** is a set  $R = \{(q, g_1), \dots, (q, g_n)\}$  of regions with the same control location,  $\text{loc}(R) = q$ . The part of region graph reachable from initial region set  $R$  is denoted  $RG(\mathcal{T}, R)$ .

**Path** in  $RG(\mathcal{T}, R)$  is infinite sequence of regions

$\sigma = r_0 \xrightarrow{l_0} r_1 \xrightarrow{l_1} \dots$  with  $r_0 \in R$ . The  $i$ -th region is denoted  $\sigma_i$ .

Path  $\sigma$  is called **non-zero** if it contains infinite number of  $\xrightarrow{\varepsilon}$  transitions and for each clock  $x \in \mathcal{X}$ :

- value of  $x$  grows beyond  $c_x$  in  $\sigma$ ,
- or  $x$  is reset infinitely often in  $\sigma$ .

We first define some operations on region sets:

$$\text{time-succ}(R) = \{r' \mid \exists r \in R. r (\overset{\varepsilon}{\rightsquigarrow})^* r'\}$$

$$\text{e-succ}(e, R) = \{r' \mid \exists r \in R. r \overset{e}{\rightsquigarrow} r'\}$$

$$\text{post}(e, R) = \text{time-succ}(\text{e-succ}(e, R))$$

**Simulation graph** of  $\mathcal{T}$  starting from  $R_0$ , denoted  $SG(\mathcal{T}, R_0)$ , is the smallest graph  $(\mathcal{R}, \rightarrow)$  such that:

- $\text{time-succ}(R_0) \in \mathcal{R}$
- if  $R \in \mathcal{R}$  and  $R' = \text{post}(e, R)$ , then  $R' \in \mathcal{R}$  and  $R \xrightarrow{e} R'$ .

**Timed automaton** is the tuple  $\mathcal{A} = (\mathcal{T}, \mathcal{F}, a)$ , where  $\mathcal{T}$  is timed system,  $\mathcal{F} \subseteq \mathcal{Q}$  is the set of **repeating locations** and  $a \in \{\overset{\infty}{\forall}, \overset{\infty}{\exists}\}$  denotes **acceptance condition**.

Path  $\sigma$  is **accepted** by  $\mathcal{A}$  for initial region  $r_0$  iff  $\sigma_0 = r_0$  and

- if  $a = \overset{\infty}{\exists}$ , then some location in  $\mathcal{F}$  is repeated infinitely often,
- if  $a = \overset{\infty}{\forall}$ , then  $\sigma$  is composed only of locations from  $\mathcal{F}$ , except for a finite prefix.

For a set of regions  $R$ , the set of *non-zero* paths accepted by  $\mathcal{A}$  is denoted  $L(\mathcal{A}, R)$ .

# Checking emptiness

We will need to solve following problem: for given TA  $\mathcal{A}$  and  $R$ , find the subset  $\hat{R} \subseteq R$  such that accepting path in  $\mathcal{A}$  starts from every  $r \in \hat{R}$ . This is called **emptiness problem**, because  $\hat{R}$  is empty iff  $L(\mathcal{A}, R)$  is empty.

This can be done by finding elementary accepting cycles in the simulation graph – by DFS, storing only the stack of already visited nodes.

**function**  $Acc_{\mathcal{A}}(R)$

$R_a := \emptyset$

**while**  $\exists$  elementary accepting cycle  $C$  in  $SG(\mathcal{A}, R)$  **do**

$R' := \text{root}(\text{pre-stable}(C))$

$R_a := R_a \cup R'$

$R := R \setminus R'$

**end while**

**return**  $R_a$

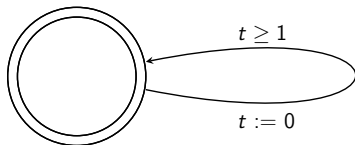
**end function**

# Checking for non-zero paths

We know how to check for existence of accepting paths, but we also require them to be non-zero.

$\forall^\infty$ -acceptance: Cycle  $C$  in  $SG(\mathcal{A}, R)$  is non-zero iff it is not **trivially zero** (i.e. does not allow time to pass at all) and if every clock is either reset infinitely often or remains unbounded in  $C$ .

$\exists^\infty$ -acceptance: Problem – non-zero  $\exists^\infty$ -accepting cycle does not have to be elementary. We have to force the accepted path to be non-zero by doing **fair product** with following automaton:



# The logic TECTL<sub>∃</sub><sup>\*</sup> – syntax

The model checking is performed against TECTL<sub>∃</sub><sup>\*</sup> formulae:

$$\varphi ::= \mathbf{true} \mid P \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists\mathcal{A}(\varphi_1, \dots, \varphi_n)$$

where  $P \in \mathcal{P}$  for finite set of atomic propositions  $\mathcal{P}$  and  $\mathcal{A}$  is TA over  $\{\varphi_1, \dots, \varphi_n\}$  (i.e.  $\Pi : \mathcal{Q} \rightarrow \{\varphi_1, \dots, \varphi_n\}$ ). Automaton  $\mathcal{A}$  has to be *time-progressive*, i.e. all of its invariants must be *true*.

The logic TECTL<sub>∃</sub><sup>\*</sup> is strictly more expressive than TCTL – can express properties such as:

- There is a region such that all paths starting from it are zero.
- There is an execution of TA along which  $\varphi$  holds infinitely often, and the intervals where  $\varphi$  does not hold last between  $l$  and  $u$  time units.

Formula  $\varphi$  is interpreted over regions  $r = (q', g')$  of TS  $\mathcal{T}'$  with labelling function  $\Pi' : Q' \rightarrow 2^{\mathcal{P}}$  as follows:

$r \models_{\mathcal{T}'} \mathbf{true}$

$r \models_{\mathcal{T}'} P$  iff  $P \in \Pi'(q')$

$r \models_{\mathcal{T}'} \neg\varphi$  iff  $r \not\models_{\mathcal{T}'} \varphi$

$r \models_{\mathcal{T}'} \varphi_1 \vee \varphi_2$  iff  $r \models_{\mathcal{T}'} \varphi_1$  or  $r \models_{\mathcal{T}'} \varphi_2$

$r \models_{\mathcal{T}'} \exists \mathcal{A}(\varphi_1, \dots, \varphi_n)$

iff  $\exists \sigma \in L(\bar{\mathcal{A}}, \bar{r}). \forall i. \sigma_i / \mathcal{T}' \models_{\mathcal{T}'} \Pi(\text{loc}(\sigma_i / \mathcal{T}'))$

where  $\bar{\mathcal{A}}$  is the TA  $(\mathcal{T}' \parallel \mathcal{T}, Q' \times \mathcal{F}, a)$  and  $\bar{r}$  the initial region  $((q', q), \bar{g})$ ,  $\bar{g} = g' \wedge \bigwedge_{x \in \mathcal{X}} x = 0$ .

The function  $\text{ev}(\mathcal{T}', R, \varphi)$  computes subset  $R_\varphi \subseteq R$  such that for each  $r \in R_\varphi$ ,  $r \models_{\mathcal{T}'} \varphi$ :

$$\text{ev}(\mathcal{T}', R, \mathbf{true}) \equiv R$$

$$\text{ev}(\mathcal{T}', R, P) \equiv \text{if } P \in \Pi'(\text{loc}(R)) \text{ then } R \text{ else } 0$$

$$\text{ev}(\mathcal{T}', R, \neg\varphi) \equiv R \setminus \text{ev}(\mathcal{T}', R, \varphi)$$

$$\text{ev}(\mathcal{T}', R, \varphi_1 \vee \varphi_2) \equiv \text{ev}(\mathcal{T}', R, \varphi_1) \cup \text{ev}(\mathcal{T}', R, \varphi_2)$$

$$\text{ev}(\mathcal{T}', R, \exists \mathcal{A}(\varphi_1, \dots, \varphi_n)) \equiv \text{mc-Acc}_{\overline{\mathcal{A}}, a}(R)$$

where  $\text{mc-Acc}_{\overline{\mathcal{A}}, a}(R)$  represents the value of  $\text{Acc}_{\overline{\mathcal{A}}, a}(R)$  on a simulation graph of  $\overline{\mathcal{A}}$  computed using function **mc-time-succ** instead of **time-succ**.

```

function mc-time-succ( $R$ )
   $R'$  := time-succ( $R$ )
   $\psi$  :=  $\Pi(\text{loc}(R'/\mathcal{T})$ 
   $R'_{\psi}$  :=  $R' \cap \text{ev}(\mathcal{T}', R'/\mathcal{T}', \psi)$ 
  return ( $(R' \cap R'_{\psi}) \triangleleft R'_{\psi}$ )
end function
  
```

Here,  $R \triangleleft R'$  means *set of regions in  $R'$  reachable from  $R$  by letting time pass while staying in  $R$  or  $R'$* :

$$R \triangleleft R' = \{r' \in R' \mid \exists r \in R. r = r_0 \xrightarrow{\varepsilon} \dots \xrightarrow{\varepsilon} r_n = r' \wedge \forall i \leq n. r_i \in R \cup R'\}$$

# Putting it all together

Input: timed system  $\mathcal{T}$  and TECTL $_{\exists}^*$  formula  $\varphi$ .

- Compute initial region set  $R_0$ .
- Compute  $R = \text{ev}(\mathcal{T}, R_0, \varphi)$ , which might involve:
  - Detection of non-zero accepting cycles in (modified) simulation graph of parallel composition of  $\mathcal{T}$  and automata appearing in  $\varphi$ .
  - Recursively calling  $\text{ev}$  during time-successor computation.
- System satisfies the property iff  $R_0 = R$ .

This is done:

**On-the-Fly** – while running DFS on the simulation graph, we only need to store the stack of already-explored symbolic states.

**Symbolically** – the algorithm works in terms of region sets, which can possibly be represented with zones.

# Putting it all together

Input: timed system  $\mathcal{T}$  and TECTL $_{\exists}^*$  formula  $\varphi$ .

- Compute initial region set  $R_0$ .
- Compute  $R = \text{ev}(\mathcal{T}, R_0, \varphi)$ , which might involve:
  - Detection of non-zero accepting cycles in (modified) simulation graph of parallel composition of  $\mathcal{T}$  and automata appearing in  $\varphi$ .
  - Recursively calling  $\text{ev}$  during time-successor computation.
- System satisfies the property iff  $R_0 = R$ .

This is done:

**On-the-Fly** – while running DFS on the simulation graph, we only need to store the stack of already-explored symbolic states.

**Symbolically** – the algorithm works in terms of region sets, which can possibly be represented with zones.

**Thank you.**