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 φ 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):

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

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

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

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 $\mathsf{TECTL}^*_\exists$ 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 Σ 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, X, q'):
  - $q, q' \in \mathcal{Q}$ are source/destination,
  - g is a clock constraint,
  - $X \subseteq \mathcal{X}$ clocks to be reset,
- $\Pi : \mathcal{Q} \to \Sigma$ is labelling function,
- $\Gamma : \mathcal{Q} \to C_{\mathcal{X}}$ is function that assigns time-progress condition (invariant) to locations.

Parallel composition of two timed systems $\mathcal{T}_1, \mathcal{T}_2$, denoted $\mathcal{T}_1 \| \mathcal{T}_2$ 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

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: $(q, g) \overset{e}{\rightsquigarrow} (q', e(g))$
- timed: $(q, g) \overset{\varepsilon}{\rightsquigarrow} (q, g')$

Region set is a set $R = \{(q, g_1), \ldots, (q, g_n)\}$ 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 $\sigma = r_0 \overset{l_0}{\rightsquigarrow} r_1 \overset{l_1}{\rightsquigarrow} \cdots$ with $r_0 \in R$. The i-th region is denoted σ_i.

Path σ is called non-zeno if it contains infinite number of $\overset{\varepsilon}{\rightsquigarrow}$ transitions and for each clock $x \in \mathcal{X}$:

- value of x grows beyond c_x in σ,
- or x is reset infinitely often in σ.

Simulation graph

We first define some operations on region sets:

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

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 $R' \in R$ and $R \xrightarrow{e} R'$.

Timed automata

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 σ 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 σ is composed only of locations from $\mathcal{F}$, except for a finite prefix.

For a set of regions R, the set of *non-zeno* 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 $\widehat{R} \subseteq R$ such that accepting path in $\mathcal{A}$ starts from every $r \in \widehat{R}$. This is called emptiness problem, because $\widehat{R}$ 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.

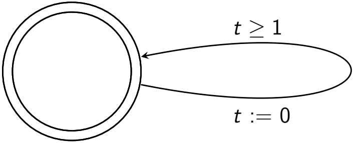
function $Acc_{\mathcal{A}}(R)$
    $R_a := \emptyset$
    while $\exists$ elementary accepting cycle C in $SG(\mathcal{A}, R)$ **do**
        $R' :=$ root(pre-stable(C))
        $R_a := R_a \cup R'$
        $R := R \setminus R'$
    end while
    return R_a
end function

Checking for non-zeno paths

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

$\overset{\infty}{\forall}$-acceptance: Cycle C in $SG(\mathcal{A}, R)$ is non-zeno 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.

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



The logic $\mathrm{TECTL}^*_\exists$ – syntax

The model checking is performed against $\mathrm{TECTL}^*_\exists$ formulae:

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

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

The logic $\mathrm{TECTL}^*_\exists$ is strictly more expressive than TCTL – can express properties such as:

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

The logic $\mathsf{TECTL}^*_\exists$ – semantics

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

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

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

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

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

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

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

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

Model checking in $\text{TECTL}^*_\exists$

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

$$\begin{aligned}
&\mathtt{ev}(\mathcal{T}', R, \mathbf{true}) \equiv R \\
&\mathtt{ev}(\mathcal{T}', R, P) \equiv \text{if } P \in \Pi'(\text{loc}(R)) \text{ then } R \text{ else } 0 \\
&\mathtt{ev}(\mathcal{T}', R, \neg\varphi) \equiv R \setminus \mathtt{ev}(\mathcal{T}', R, \varphi) \\
&\mathtt{ev}(\mathcal{T}', R, \varphi_1 \vee \varphi_2) \equiv \mathtt{ev}(\mathcal{T}', R, \varphi_1) \cup \mathtt{ev}(\mathcal{T}', R, \varphi_2) \\
&\mathtt{ev}(\mathcal{T}', R, \exists\mathcal{A}(\varphi_1, \ldots, \varphi_n)) \equiv \text{mc-Acc}_{\overline{A},a}(R)
\end{aligned}$$

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)
  $\psi := \Pi(\text{loc}(R'/_{\mathcal{T}})$
  $R'_\psi := R' \cap \mathtt{ev}(\mathcal{T}', R'/_{\mathcal{T}'}, \psi)$
  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'*:

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

Putting it all together

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

- Compute initial region set R_0.
- Compute $R = \mathtt{ev}(\mathcal{T}, R_0, \varphi)$, which might involve:
  - Detection of non-zeno accepting cycles in (modified) simulation graph of parallel composition of $\mathcal{T}$ and automata appearing in φ.
  - Recursively calling $\mathtt{ev}$ 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 $\mathrm{TECTL}^*_\exists$ formula φ.

- Compute initial region set R_0.
- Compute $R = \mathtt{ev}(\mathcal{T}, R_0, \varphi)$, which might involve:
  - Detection of non-zeno accepting cycles in (modified) simulation graph of parallel composition of $\mathcal{T}$ and automata appearing in φ.
  - Recursively calling $\mathtt{ev}$ 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.