SS 2017
Software Verification
TCTL Model Checking

Prof. Dr. Holger Schlingloff\(^1,2\)
Dr. Esteban Pavese\(^1\)

(1) Institut für Informatik der Humboldt Universität
(2) Fraunhofer Institut für offene Kommunikationssysteme FOKUS
Timed CTL Syntax

- TCTL syntax is very similar to CTL

\[ \phi ::= \text{true} | \text{ap} | \text{cc} | \phi \land \phi | \neg \phi | \exists \phi | \forall \psi \]

\[ \psi ::= \phi U^I \phi \]

- \( \phi \) formulae are *state* formulae
- \( \psi \) formulae are *path* formulae
- Note 1: Interval I must be natural-bounded
- Note 2: no X operator. Why?
Timed CTL Semantics

\( (s, v) \models true \)

\( (s, v) \models a \) \iff \( a \in L(s) \)

\( (s, v) \models cc \) \iff \( v \models cc \)

\( (s, v) \models \neg \phi \) \iff \( \neg((s, v) \models \phi) \)

\( (s, v) \models \phi_1 \land \phi_2 \) \iff \( (s, v) \models \phi_1 \land (s, v) \models \phi_2 \)

\( (s, v) \models \exists \phi \) \iff \( \pi \models \phi \) for some (time-divergent) path \( \pi \) from \( s \)

\( (s, v) \models \forall \phi \) \iff \( \pi \models \phi \) for every (time-divergent) path \( \pi \) from \( s \)

This should not be surprising. It is exactly the same as for CTL
Timed CTL Semantics

• Until semantics are a bit different

• TA paths can be written as

\[ s_0 \xrightarrow{d_0} s_1 \xrightarrow{d_1} s_2 \xrightarrow{d_2} s_3 \xrightarrow{d_3} s_4 \ldots \]

• 0-delay denotes a location change
Timed CTL Semantics

\[ \pi \models \phi \mathcal{U}^I \psi \iff \exists \ i \geq 0 \cdot s_i + d \models \phi \ (d \in [0, d_i]) \text{ and } \]

\[ 1) \quad \sum_{j=0}^{i-1} d_k + d \in I \]

or, there is a state in the trace such that a delay on that state

- satisfies \( \psi \)
- the total elapsed time is in \( I \)
Timed CTL Semantics

\[ \pi \models \phi U^I \psi \iff \exists i \geq 0 \cdot s_i + d \models \phi \ (d \in [0, d_i]) \text{ and } \]

\[ \forall k \leq i \ s_k + d' \models (\phi \lor \psi) \] for every \( d' \in [0, d_k] \) and

\[ \sum_{j=0}^{k-1} d_j + d' \leq \sum_{j=0}^{i-1} d_j + d \]

before that point, \((\phi \lor \psi)\) must hold at every time
Timed CTL Semantics for TA

- For a formula $\phi$ and timed automata TA, the satisfaction state set is given by
  $$\text{Sat}(\phi) = \{(s, v) \in (S \times V(C)) | (s, v) \models \phi\}$$
- ... and a timed automaton is said to satisfy the formula if all initial states satisfy it
  $$TA \models \phi \text{ if and only if } (s_0, v_0) \in \text{Sat}(\phi)$$
- ... it does look a lot like CTL model checking
- in fact...

$$TA \models_{TCTL} \forall \phi U [0, \infty) \psi \sim TA \models_{CTL} \forall \phi U \psi$$
Timed CTL semantics for TA

- The transition system for a FA is the same FA
- ...with variables, we need to transform it
  - ... but we still get a FA
- In the case of timed automata, the transition system does NOT correspond to a finite automaton
- The transition system of a TA is uncountably infinite
Making the transition system finite

Uncountably infinite states to check!
Making the transition system finite

IDEA – Partition the *uncountably infinite* state space into *finite* portions

(SPOILER) It can be done! And the partition is a *bisimulation*
Making the transition system finite

- The outline of the idea is
  - Obtain the finite partition - the *region transition system*
  - Transform the property into an “equivalent” CTL formula
  - CTL-check the new formula on the *region* transition system
  - ...and if all goes well, the procedure is equivalent to checking TCTL on the original model
Making the transition system finite

- First step
  - eliminate timing constraints in the formula to be checked
  - Introduce a new clock to the system

\[
(s, v) \models \forall \phi \mathcal{U}^I \psi \quad \iff \quad (s, v\{z := 0\}) \models \forall((\phi \vee \psi) \mathcal{U}(z \in I \land \psi))
\]

- (same with existential)
- we end up with CTL properties!
Making the transition system finite

• Second step
  ▪ construct a timing bisimulation such that when
    \[(s, v) \cong (s', v')\]
  ▪ then:
    
    \[s = s' \land v \cong v'\]

\[v \cong v' \implies \forall cc \in CC(TA) \cup CC(\phi) \cdot v \models cc \iff v' \models cc\]

\[\Pi(s, v) \cong \Pi(s', v')\]

and such that the bisimulation is finite
Making the transition system finite

- Third step
  - recall clock conditions have *natural* bounds. Then,

\[
v'(x) < c \iff \lfloor v'(x) \rfloor < c
\]

\[
v'(x) \leq c \iff \lfloor v'(x) \rfloor < c \lor (\lfloor v'(x) \rfloor = c) \land \text{frac}(v'(x)) = 0
\]

Therefore the bisimulation must satisfy that (1)

\[
\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor \land \text{frac}(v(x)) = 0 \iff \text{frac}(v'(x)) = 0
\]
Making the transition system finite

These conditions make the bisimulation at most *enumerable*. Finiteness is obtained by bounding via the *largest constant* in the TA or property.
Making the transition system finite

• Fourth step
  - the conditions are still too coarse

The “closest” clock will enable the transition first
Making the transition system finite

• Fourth step
  - given two valuations satisfying (1), we need to consider the ordering between clocks in the same zone
  - Valuations in the same region agree on integral part
  - Now they must agree in ordering

\[
\frac{v(x)}{v(y)} \leq \frac{v'(x)}{v'(y)}
\]
Making the transition system finite

• Fourth step

\[ 3 < x < 4, 1 < y < 2, x - y < 2 \]

\[ -3 < x < 4, 1 < y < 2, x - y = 2 \]

\[ 3 < x < 4, 1 < y < 2, x - y > 2 \]
How many regions?

- Now we have a bisimulation generating a finite number of equivalence
- but how many exactly?

Let $C$ = number of clocks, $c_x$ maximum constant comparing clock $x$, then the number of clock regions is bounded by

$$|C|! \times \prod_{x \in C} c_x$$

$$|C|! \times 2^{|C|-1} \times \prod_{x \in C} 2(c_x + 1)$$
Building the transition system

- Each state of the real time transition system consists of a discrete location $l$ and a clock region $r$
- We will note $[v]$ as the region containing $v$, that is, all other $v'$ such that $v \equiv v'$
- Since action transitions take no time, they do not change the clock region
  - except for resets!
- Delay transitions do not change the location
  - ...but may traverse many clock regions
- Once a clock reaches its maximum bound, further delays do *nothing*
Building the transition system

- Clock resets yield valid regions
- Given a zone \( r \), we can define \( \text{reset}(D, r) \) as the reset of clocks in \( D \)
  \[
  \text{reset}(D, r) = \{ v[c := 0] | v \in r, c \in D \}
  \]
- Since the region distribution is an equivalence
  \[
  v \equiv v' \implies v[c := 0] \equiv v'[c := 0]
  \]
- and therefore
  \[
  r \equiv r' \implies \text{reset}(D, r) \equiv \text{reset}(D, r')
  \]
Building the transition system

- We can now define the action transitions on the region transition system

\[
<l, r> \xrightarrow{a,D} <l', \text{reset}(D,r)> \in R_{RTS}
\]

\[
\iff
\]

\[
l \xrightarrow{a,D} l' \in R_{TA}
\]
Building the transition system

- Delay transitions need to be decomposed so that every delay traverses every region in between

\[ \langle l, r \rangle \xrightarrow{d} \langle l, r' \rangle \]

whenever:

1. \( r = r_\infty \) (the unbounded region) and \( r = r' \); or

2. \( r \neq r_\infty, r \neq r' \) and \( \forall v \in r \)

\[ \exists d \in \mathbb{R}_{>0} (v + d \in r' \land \forall d' \in [0, d] v + d \in (r \cup r')) \]
Example!

\[ x \geq 2 : \alpha \]

\[ \text{reset}(x) \]

Figure 9.23: Region transition system for a simple timed automaton with \( \Phi = \text{true} \).

Figure 9.24: Region transition system for a simple timed automaton with \( \Phi \text{ with } c \).
TCTL checking algorithm

\[ R := RTS(TA \oplus z, \Phi); \]  

(* with state space \( S_{rts} \) and labeling \( L_{rts} \) *)

for all \( i \leq |\Phi| \) do

\[ \text{for all } \Psi \in \text{Sub}(\Phi) \text{ with } |\Psi| = i \text{ do} \]

switch(\( \Psi \)):

true : \( \text{Sat}_R(\Psi) := S_{rts}; \)

a : \( \text{Sat}_R(\Psi) := \{ s \in S_{rts} \mid a \in L_{rts}(s) \}; \)

\( \Psi_1 \land \Psi_2 \) : \( \text{Sat}_R(\Psi) := \{ s \in S_{rts} \mid \{a_{\Psi_1}, a_{\Psi_2}\} \subseteq L_{rts}(s) \}; \)

\( \neg \Psi' \) : \( \text{Sat}_R(\Psi) := \{ s \in S_{rts} \mid a_{\Psi'} \notin L_{rts}(s) \}; \)

\( \exists(\Psi_1 U J \Psi_2) \) : \( \text{Sat}_R(\Psi) := \text{Sat}_{CTL}\left( \exists((a_{\Psi_1} \lor a_{\Psi_2}) U ((z \in J) \land a_{\Psi_2})) \right); \)

\( \forall(\Psi_1 U J \Psi_2) \) : \( \text{Sat}_R(\Psi) := \text{Sat}_{CTL}\left( \forall((a_{\Psi_1} \lor a_{\Psi_2}) U ((z \in J) \land a_{\Psi_2})) \right); \)

end switch

(* add \( a_{\Psi} \) to the labeling of all state regions where \( \Psi \) holds *)

forall \( s \in S_{rts} \) with \( s\{z := 0\} \in \text{Sat}_R(\Psi) \) do \( L_{rts}(s) := L_{rts}(s) \cup \{a_{\Psi}\} \) od;

od

dod
if \( I_{rts} \subseteq \text{Sat}_R(\Phi) \) then return “yes” else return “no” fi
Some more remarks

- We are forgetting location invariants!
  - Not to worry. We simply intersect the region corresponding to the invariant
  - Intersection of regions yields a region
  - Unions will make things harder later on

- We are forgetting nested Until properties!
  - easy! just add a fresh clock for each nested U!
  - but clock size grows linearly!
  - we can do it with just one clock though
Some more remarks

- TCTL model checking is in PSPACE-complete
- Model checking of CTL or LTL over TA is in PSPACE-complete

- Satisfiability of TCTL is undecidable
- Is TLTL a thing?
  - Model checking for Timed LTL is undecidable
Implementing the monster
Implementing the monster

- The state explosion problem in verification requires efficient data structures and algorithms
  - We’ve already seen BDDs for symbolic representation of discrete states
  - Also on-the-fly construction of the RTS
- What about clock regions?
  - We can still use BDDs for discrete states
  - New structures appear for handling clock regions symbolically
Difference Bound Matrix (DBM)

• First, we need a reference point
  ▪ Let’s add a constant clock 0 (which naturally is set to zero all the time)

• Now all clock constraints can be written as
  \[ x - y \leq c, \leq \in \{<, \leq\} \]

• We have at most \((|C| + 1)^2\) constraints (avoiding redundant ones)

• Any region can be encoded like this
Difference Bound Matrix

• Rows and columns represent clocks (including the fixed clock 0)
• A region encoded in DBM $Z$ satisfies
  $$Z(i, j) = (\prec, c) \iff x_i - x_j < c$$
• Special note for unbounded region, it can be that $Z(i,j) = (\prec, \text{infinity})$ in this case
Example

\[(x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\]

Is this the only possible valid representation?
Example

\[(x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\]

\[\begin{pmatrix}
  x_0 \\
  x_1 \\
  x_2
\end{pmatrix} = \begin{pmatrix}
  +\infty \\
  +\infty \\
  5
\end{pmatrix} \begin{pmatrix}
  +\infty \\
  -3 \\
  4
\end{pmatrix} + \begin{pmatrix}
  +\infty \\
  +\infty \\
  +\infty
\end{pmatrix}

Is this only possible valid representation?
Clearly not!
Is there a *best representation?*
Canonical DBM

- Fortunately, each DBM has a canonical form that is, a standard upon we can define all representations.

- A DBM is in canonical form if strengthening a condition results in reducing the represented area.
Canonical DBM

- Every zone has a canonical form, and this canonical form is unique
- Obtaining the canonical DBM is equivalent to finding the least difference between clocks
- ... or equivalently, finding the shortest (time) distance between them
- so it must hold that a DBM is canonical iff

\[ Z(i, j) \leq Z(i, k) + Z(k, j) \quad \forall x_i, x_j, x_k \]
Canonical DBM

- Hmm... that *really* rings a bell...
Canonical DBM

- Hmm... that *really* rings a bell...
- The problem of making a DBM canonical is equivalent to finding *all—shortest paths*
- Use a standard algorithm like Floyd-Warshall, complexity in $O(|C + 1|^3)$
- Improve efficiency by maintaining canonicity throughout all operations during region transition graph construction
Reduced canonical DBM

- Further memory consumption improvement can be made
- We already know this to be true
  \[ Z(i, j) \leq Z(i, k) + Z(k, j) \forall x_i, x_j, x_k \]
- but what if for some \( i, j, k \)
  \[ Z(i, j) = Z(i, k) + Z(k, j) \]
- !! Now \( Z(i,j) \) is redundant and can be dropped
DBM operations

- Non-emptiness check (satisfiability)
  - check for negative cycles
  - or check for
    \[ x_i - x_j \leq c \text{ and } x_j - x_i \leq' c' \text{ and } (c, \leq) < (c', \leq') \]

- Zone inclusion test
  - check matrices element by element
    \[ Z \subseteq Z' \iff Z(i, j) \leq Z'(i, j) \forall x_i, x_j \]

- Constraint satisfaction
  - add the constraint + non-emptiness check
DBM operations

• Future successors
  ▪ move all single clock upper bounds to infinity
  ▪ preserves canonicity!

• Past successors
  ▪ important for on-the-fly backwards model checking
    ▪ Set single clock lower bounds to zero
    ▪ does not preserve canonicity
DBM operations

- **Intersection**
  - pick the lower bound for each element pair
  - re-canonization can be improved if few elements are modified

- **Reset**
  - resetting clock $i$ yields $Z(i,j)=Z(0,j)$ and $Z(j,i)=Z(j,0)$
DBM Operations

- Union
  - required when coming back to a discrete location with a different clock region
  - Joining convex regions *does not* in general yield a convex region
  - DBMs can only represent convex areas, so...
  - use a list of DBMs
  - many other (failed) alternatives