SS 2017
Software Verification
Timed Automata

Prof. Dr. Holger Schlingloff $^{1,2}$
Dr. Esteban Pavese $^1$

$^1$ Institut für Informatik der Humboldt Universität
$^2$ Fraunhofer Institut für offene Kommunikationssysteme FOKUS
Recap

- What is a fixpoint?
- How is CTL model checking performed?
- What is a symbolic representation?
- What are BDDs?
- How/why are BDDs used in model checking (not only CTL)?
Back to roots

We use automata to model (abstracted) real world situations, and check properties over these models.
See the hidden problem?

This model is too abstract to capture some finer points of the real system...
The hidden problem

What is going on in this case?

Is there a way to “solve it”? Maybe extra restrictions on the transitions? Do they make sense? Maybe extra assumptions? Are we being too abstract?
Re-thinking about the train

Some assumptions are not captured by the model

- The train has a maximum speed
  - therefore there is a lower bound on the time needed from *approach* to *enter*

- The gate does not take too long to close the road
  - an upper bound on the time it takes exists

- The controller will not take too long (nor too short) a time to signal the gate to lower
Refining the model

TRAIN
- approach
- exit
- enter

GATE
- lower
- raise

CONTROLLER
- approach
- raise
- lower
- exit

Takes at least 2 minutes
Takes at most 1 minute
Will signal after 1 minute
Refining the model

Makes sense! Now, if only these annotations were formal ...
First thing to note!

- This is easy for *synchronous* systems
- Every process moves at the same time
  - There is a global clock that “ticks”
- So simply do
  - define time resolution
  - define how much time each action takes
  - redefine each action with as many steps as needed
- ... and use LTL / CTL as usual
Timed automata

- Timed automata are a \textit{proper} extension of the finite automata we know so far
- Timed automata = finite automata + finite clocks
- Clocks are \textit{real} valued variables
  - but we can only read them, or reset them to zero
  - all of them increase at the same rate
  - like having many, many stopwatches
Timed automata

- We can establish *conditions* on clock values

\[ x \in C, \quad a \in \mathbb{R} \quad cc ::= x < a \mid x \leq a \mid x > a \mid x \geq a \mid cc \land cc \]

- Clock conditions will be used as guards in
  - transitions (enabling transition)
  - states (triggering transition)
Second thing to note!

- If clocks were *rational*, this is easy
- Take all rational clock constraints
  - find the smallest constant – this gives the time resolution
  - find the lcm of all denominators. Multiply all constants (and also the time resolution) by this number
  - Everything is now an integer!
  - Define variables instead of clocks, use LTL / CTL as usual
Timed automata

Formally

\[ TA = \langle S, s_0, C, A, R, Inv, AP, L \rangle \]

- \( S \) is a finite set of states, \( s_0 \) is the initial state
- \( C \) is a finite set of clocks, all begin at zero
- \( A \) is a finite set of action labels
- \( R \subseteq S \times CC \times A \times 2^C \times S \) is the transition relation
- \( Inv : S \rightarrow CC \) is the invariant function
- \( AP \) is a set of atomic propositions, \( L : S \rightarrow 2^{AP} \) is the labelling function
Timed automata

\[ R \subseteq S \times CC \times A \times 2^C \times S \]

- **source state**
- **action label**
- **transition condition**
- **clocks to reset**
- **destination state**

**Inv : S → CC**

- **source state**
- **how long can we stay here?**
Transition conditions and invariants

- Transitions conditions control when a transition is \textit{enabled}
- Invariants control when a transition \textit{must} be taken

\begin{align*}
\text{x} &\geq 2 : \tau \\
\text{reset}(x)
\end{align*}

Figure 9.5: Some timed automata with a single clock and one of their evolutions.

Figure 9.5(b) gives an example of an execution of this timed automaton, by depicting the value of clock \(x\) vs. the elapsed time since the start of the automaton. Each time the clock is reset to 0, the automaton traverses the self-loop at location \(\ell\).

As \(\text{Inv}(\ell) = \text{true}\), time can progress without any restriction while residing in \(\ell\). In particular, an algebraic behavior of this automaton is to stay in location \(\ell\) ad infinitum. Formally, 

\[
\text{Loc} = \text{Loc}_0 = \{\ell\}, \quad \text{C} = \{x\}, \quad x \geq 2 : \tau, \quad \text{reset}(x) \quad \text{and} \quad \text{Inv}(\ell) = \text{true}.
\]
Transition conditions and invariants

- Transitions conditions control when a transition is **enabled**
- Invariants control when a transition **must** be taken

\[ x \geq 2 : \tau \]

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

Figure 9.5: Some timed automata with a single clock and one of their evolutions.

Figure 9.5(b) gives an example of an execution of this timed automaton, by depicting the value of clock \( x \) vs. the elapsed time since the start of the automaton. Each time the clock is reset to 0, the automaton traverses the self-loop at location \( \ell \).

As \( \text{Inv}(\ell) = \text{true} \), time can progress without any restriction while residing in \( \ell \). In particular, an algebraic behavior of this automaton is to stay in location \( \ell \) ad infinitum. Formally,

\[ \text{Loc} = \text{Loc}_0 = \{ \ell \}, \quad C = \{ x \}, \quad \ell \cup \{ x \} \to \ell, \quad \text{and} \quad \text{Inv}(\ell) = \text{true}. \]
Transition conditions and invariants

- Transitions conditions control when a transition is *enabled*
- Invariants control when a transition *must* be taken

\[ 2 \leq x \leq 3 : \tau \]
\[ \text{reset}(x) \]
Train/Gate/Controller redesigned

Train:
- Approach: \textit{reset(x)}
- \( x < 5 \)
- Enter: \( x > 2 \)
- Exit

Gate:
- Lower: \textit{reset(z)}
- \( z \leq 1 \)
- Raise: \textit{reset(z)}
- \( z \leq 1 \)
- Takes at most 1 minute
- Takes at least 2 minutes

Controller:
- Approach: \textit{reset(y)}
- \( y \leq 1 \)
- Raise: \( y = 1 \)
- Exit
- Lower: \( y = 1 \)
- \( y \leq 1 \)

Will signal after 1 minute
Parallel composition

- As with finite automata, it makes sense to model different concerns separately
- Parallel composition still synchronizes on labels, but now must take clocks into account
- Non-shared rule is easy:

\[
\begin{align*}
\ell_1 & \leftarrow g: \alpha, D \rightarrow 1 \ell'_1 \\
\langle \ell_1, \ell_2 \rangle & \leftarrow g: \alpha, D \rightarrow \langle \ell'_1, \ell_2 \rangle \\
\end{align*}
\]  
and \[
\begin{align*}
\ell_2 & \leftarrow g: \alpha, D \rightarrow 2 \ell'_2 \\
\langle \ell_1, \ell_2 \rangle & \leftarrow g: \alpha, D \rightarrow \langle \ell_1, \ell'_2 \rangle \\
\end{align*}
\]
Parallel composition

- Shared rule is slightly more involved

\[ \ell_1 \xleftarrow{g_1: \alpha, D_1} 1 \ell'_1 \land \ell_2 \xrightarrow{g_2: \alpha, D_2} 2 \ell'_2 \]

\[ \langle \ell_1, \ell_2 \rangle \xleftarrow{g_1 \land g_2: \alpha, D_1 \cup D_2} \langle \ell'_1, \ell'_2 \rangle \]

- Note that a conjunction of constraints is still a valid constraint

- The invariant of a composite state is also the conjunction of both invariants
  - Again, still a valid constraint. This will be important later on.
Semantics

- The semantics of a TA are given by
  - current discrete state (also called location)
  - clock values
- This is now an *infinite* state machine!
  - Worse, *uncountably* infinite!
- The trace semantics of a TA are successions of <location, clock valuation>
- The branching semantics is an *infinitely branching* tree
Transitions

- There exist then two types of transitions
- Location transitions, where location changes through an action label
  - location transitions take zero time
- Delay transitions, where clocks advance a time $d$, remaining in the same location
  - ...as long as the location invariant holds
- The elapsed time of a trace (or trace fragment) is the sum of all its delays
Semantics

- The trace semantics of a TA are successions of \(<\text{location}, \text{clock valuation}\>\)

- The branching semantics is an *infinitely branching* tree of \(<\text{location}, \text{clock valuation}\>\)
Consider the simple example of a light switch

Can you describe some of its traces?
Can you describe all \textit{reachable} states?
Deadlock was bad enough? Timelock and other issues

- The addition of time and clocks causes some traces to not be acceptable

\[ \langle \text{off}, 0 \rangle \rightarrow \langle \text{off}, 0.5 \rangle \rightarrow \langle \text{off}, 0.75 \rangle \rightarrow \langle \text{off}, 0.875 \rangle \rightarrow \langle \text{off}, 0.9375 \rangle \rightarrow \ldots \]

\[ s_i = \langle \text{off}, 1 - 2^{-i} \rangle \]

Any fragment of this trace (even infinite!) takes a finite amount of time...
Time convergence and divergence

- An infinite trace fragment is said to be *time divergent* if its elapsed time is *infinite*
- On the contrary, an infinite path fragment with finite elapsed time is said to be *time convergent*
- Time convergent traces are unrealistic, and therefore we do not consider them
Timelock

- A state from which no divergent traces originate is said to be in *timelock*

- In timelock, time either cannot carry on
- Or, it can do so only up to a bound
- If no traces can escape this bound, the state is unrealistic
Timelock

1 \leq x < 2

\textit{switch\_off} \\
\textit{switch\_on} \\
\textit{reset}(x)

\textit{on} \\
\underline{x \leq 2}

\textit{off} \\
\textit{on} \\
\underline{x < 3}

\textit{switch\_off} \\
\textit{switch\_on} \\
\textit{reset}(x)

H. Schlingloff, E. Pavese SS2017: Software-Verifikation
Zenoness

- An execution trace is said to be *zeno* if
  - It is time convergent, and
  - It contains an infinite number of location transitions, that is, infinite actions are taken in a finite time.

![Timed Automaton Diagram]
Zenoness

- A timed automaton is said to be *zeno* if it contains *at least one* zeno trace from its initial state
- Zeno automata are unrealistic – they require an infinitely fast processor!
- Zeno automata are a design flaw. We need only non-zeno automata
- Checking for non-zenoness is hard
(Approximate) non-zeno check

• Instead, it is easy to check for *sufficient* conditions for non-zenoness
• It suffices to check that in every control cycle (location cycle), time increases
• Formally, for every control cycle there exists a clock $x$ such that
  - $x$ is reset in the cycle
  - For every clock valuation $v$, a natural number $c$ and a position $j$ in the cycle exist such that if $v(x) < c$, then either
    - the location invariant at $j$ is not satisfied; or
    - the guard for the transition at $j$ is not satisfied
  - that is, at point $j$ the clock $x$ must be larger than $c$
(Approximate) non-zeno check

The condition is also *compositional*.

If two TAs A and B satisfy the condition, then their composition A||B will also satisfy it.

Exercise: check the condition on the Train, Gate and Controller models.
Timed CTL

- In order to reason about TAs we use the logic TCTL
- TCTL is a proper extension of CTL
- We change the U operator by enriching it with a time interval – $U^I$
- A path satisfies $\phi U^I \psi$ if $\phi$ is true along the path, until $\psi$ becomes true and, additionally, it takes a time $t \in I$ for the latter to be true