Software Verification

Lecture 9: Verification of Real-time Systems

Carlo A. Furia
Program Verification: the very idea

P: a program

max (a, b: INTEGER): INTEGER is
  do
    if a > b then
      Result := a
    else
      Result := b
  end
end

S: a specification

require
  true
ensure
  Result >= a
  Result >= b

Does \( P \vDash S \) hold?

The Program Verification problem:

- **Given**: a program \( P \) and a specification \( S \)
- **Determine**: if every execution of \( P \), for any value of input parameters, satisfies \( S \)
Real-time Verification

P: a program

max (a, b: INTEGER): INTEGER is
  do
    if a > b then
      Result := a
    else
      Result := b
  end
end

S: a specification

ensure
  Result >= a
  Result >= b

ensure -- real-time
  "max terminates no sooner than 3 ms and no later than 10 ms after invocation"

Does \( P \models S \) hold?

The Real-time Verification problem:

- **Given**: program \( P \), embedded in system \( E \), and real-time specification \( S \)
- **Determine**: if every execution of \( P \) (within \( E \)) satisfies \( S \)
Real-time Programs and Systems

Def. Real-time specification: specification that includes exact timing information.

Def. Real-time computation: computation whose specification is real-time. In other words: computation whose correctness depends not only on the value of the result but also on when the result is available.

• The timing of a piece of software is usually dependent on the environment where the computation takes place

• Hence, in real-time verification the focus shifts from programs to (software-intensive) systems
  – In a system, even the physical environment is often relevant

• The purely computational aspects can often be analyzed in isolation

• Real-time verification can then focus on real-time aspects of the system
  – e.g., synchronization, deadlines, delays, ... while abstracting away most of the rest
Decidability vs. Expressiveness Trade-Off

The Real-time Verification problem:

- **Given:** program $P$, embedded in system $E$, and real-time specification $S$
- **Determine:** if every execution of $P$ (within $E$) satisfies $S$

\[
\begin{align*}
& P: \text{a system} & & S: \text{a real-time specification} \\
& F(P): \text{some formal model of } P & & N(S): \text{some formal notation for } S \\
\end{align*}
\]

Does $F(P) \models N(S)$ hold?

- The **classes** for $F(P)$ and $N(S)$ should guarantee:
  - enough **expressiveness** to include a **quantitative notion of time**
  - **decidability** of the verification problem
Real-time Model-Checking

The Real-time Model Checking problem:

- **Given**: a timed automaton \( A \) and a metric temporal-logic formula \( F \)
- **Determine**: if every run of \( A \) satisfies \( F \) or not
  - if not, also provide a counterexample: a run of \( A \) where \( F \) does not hold

\[
A \vdash F
\]

- The model-checking paradigm is naturally extended to real-time systems
- Different choices are possible for the family of automata and of formulae
  - The linear vs. branching time dichotomy is usually not significant for real-time
    - linear time is almost invariably preferred
  - A different attribute of time that becomes relevant in quantitative models is
    - discrete vs. dense time
Discrete vs. dense (continuous) time

- **Discrete time**
  - sequence of isolated “steps”
  - every instant has a unique successor
  - e.g.: the naturals \( \mathbb{N} = \{0, 1, 2, \ldots\} \)

  + simple and intuitive
  + verification usually decidable (and acceptably complex)
  + robust and elegant theoretical framework

  - cannot express true asynchrony
  - unsuitable to model physical variables

- **Dense time**
  - arbitrarily small distances
  - the successor of an instant is not defined
  - e.g.: the reals \( \mathbb{R} \)

  + can model true asynchrony
  + accurate modeling of physical variables

  - tricky to understand
  - verification easily undecidable (or highly complex)
  - lacks a unifying framework

- merely **dense vs. continuous** is usually not as relevant

  - e.g.: \( \mathbb{Q} \) vs. \( \mathbb{R} \)
Dense Real-time Model-Checking

Timed Automata and Metric Temporal Logic
Dense Real-time Model-Checking

Dense real-time model checking extends standard “untimed” model checking:

- The *Timed Automaton (TA)* extends the *Finite-State Automaton (FSA)*
- *Metric Temporal Logic (MTL)* extends *Linear Temporal Logic (LTL)*

The Dense Real-time Model Checking problem:

- **Given**: a dense TA $A$ and an MTL formula $F$
- **Determine**: if every run of $A$ satisfies $F$ or not
  - if not, also provide a counterexample: a run of $A$ where $F$ does not hold

\[ A \nvdash F \]

$A$: a TA  
$F$: an MTL formula
Timed Automata: Syntax

Def. Nondeterministic Timed Automaton (TA):
a tuple \([\Sigma, S, C, I, E, F]\):

- \(\Sigma\): finite nonempty (input) alphabet
- \(S\): finite nonempty set of locations (i.e., discrete states)
- \(C\): finite set of clocks
- \(I, F\): set of initial/final states
- \(E\): finite set of edges \([s, \sigma, c, \rho, s']\)
  - \(s \in S\): source location
  - \(s' \in S\): target location
  - \(\sigma \in \Sigma\): input character (also “label”)
  - \(c\): clock constraint in the form:
    \[c ::= x \approx k \mid x - y \approx k \mid \neg c \mid c_1 \land c_2\]
    - \(x, y \in C\) are clocks
    - \(k \in \mathbb{Z}\) is an integer constant
    - \(\approx\) is a comparison operator among \(<, \leq, >, \geq, =\)
  - \(\rho \subseteq C\): set of clock that are reset (to 0)
\textbf{Timed Automata: Semantics}

\textbf{Def.} A timed word \( w = w(1) \ldots w(n) \in (\Sigma \times \mathbb{R})^* \) is a sequence of pairs \([\sigma(i), t(i)]\) such that:

- the sequence of timestamps \( t(1), t(2), \ldots, t(n) \) is \textbf{increasing}

- \([\sigma(i), t(i)]\) represents the \( i \)-th character \( \sigma(i) \) read at time \( t(i) \)

\textbf{Def.} An accepting run of a TA \( A=\langle \Sigma, S, C, I, E, F \rangle \) over input timed word \( w = [\sigma(1), t(1)] \ldots [\sigma(n), t(n)] \in (\Sigma \times \mathbb{R})^* \) is a sequence \( r = [s(0), v(0,1), \ldots, v(0,|C|)] \ldots [s(n), v(n,1), \ldots, v(n,|C|)] \in (S \times \mathbb{R}^{|C|})^* \) of (extended) states such that:

\begin{itemize}
  \item it \textbf{starts} from an initial state and \textbf{ends} in an accepting state: \( s(0) \in I \) and \( s(n) \in F \)
  \item initially all clocks are reset to 0: \( v(0,k) = 0 \) for all \( 1 \leq k \leq |C| \)
  \item for every \textbf{transition} \( 0 \leq i < n \):
    \[ [s(i) \ v(i,1) \ldots v(i,|C|)] \rightarrow [s(i+1) \ v(i+1,1) \ldots v(i+1,|C|)] \]
    some \textbf{edge} \([s(i), \sigma(i+1), c, \rho, s(i+1)]\) in \( E \) is followed:
    \begin{itemize}
      \item the clock values \( v(i,1) + (t(i+1) - t(i)) \ldots v(i,|C|) + (t(i+1) - t(i)) \) satisfy the constraint \( c \)
      \item \( v(i+1,k) = \text{if } k\text{-th clock is in } \rho \text{ then } 0 \text{ else } v(i,k) + t(i+1) - t(i) \)
    \end{itemize}
\end{itemize}
Timed Automata: Semantics

- **Accepting run:**

  \[
  r = \{ \text{off, } (x=0, y=0) \} \\
  \{ \text{on, } (x=0, y=3.2) \} \\
  \{ \text{cooking, } (x=8.5, y=0) \} \\
  \{ \text{on, } (x=81.7, y=73.2) \} \\
  \{ \text{off, } (x=84.91, y=76.41) \}
  \]

- **Over input timed word:**

  \[
  w = \{ \text{turn\_on, } 3.2 \} \\
  \{ \text{start, } 11.7 \} \\
  \{ \text{stop, } 84.9 \} \\
  \{ \text{turn\_off, } 88.11 \}
  \]
Timed Automata: Semantics

Def. Any TA \( A = [\Sigma, S, C, I, E, F] \) defines a set of input timed words \( \langle A \rangle \):
\[
\langle A \rangle \triangleq \{ w \in (\Sigma \times \mathbb{R})^* \mid \text{there is an accepting run of } A \text{ over } w \}
\]
\( \langle A \rangle \) is called the language of \( A \)

With regular expressions and arithmetic:
\[
\langle A \rangle = ( [\text{turn\_on, } t_1] \\
( [\text{start, } t_2] [\text{stop, } t_3] )^* \\
[\text{turn\_off, } t_4] )^*
\]
with \( t_3 - t_2 \leq 300 \) and \( t_4 - t_1 > 1 \)
**Metric (Linear) Temporal Logic: Syntax**

**Def.** Propositional Metric Temporal Logic (MTL) formulae are defined by the grammar:

\[ F ::= p \mid \neg F \mid F \land G \mid F \ U <a,b> G \]

with \( p \in P \) any atomic proposition and \( <a,b> \) is an interval of the time domain (w.l.o.g. with integer endpoints).

**Temporal (modal) operators:**

- **next:** \( X F \triangleq \text{True} U[1,1] F \)
- **bounded until:** \( F U <a,b> G \)
- **bounded release:** \( F R <a,b> G \triangleq \neg (\neg F U <a,b> \neg G) \)
- **bounded eventually:** \( \Diamond <a,b> F \triangleq \text{True} U <a,b> F \)
- **bounded always:** \( \Box <a,b> F \triangleq \neg \Diamond <a,b> \neg F \)
- **intervals can be unbounded; e.g., [3, \infty)\)
- **intervals with pseudo-arithmetic expressions, e.g.:**
  - \( \geq 3 \) for [3, \infty)
  - \( = 1 \) for [1,1]
  - \( [0, \infty) \) is simply omitted

\( \square ( \text{start } \Rightarrow \Diamond (3,10] \text{ stop} ) \)
Def. A timed word $w = [\sigma(1), t(1)] [\sigma(2), t(2)] \ldots [\sigma(n), t(n)] \in (2^P \times \mathbb{R})^*$ satisfies an LTL formula $F$ at position $1 \leq i \leq n$, denoted $w, i \models F$, under the following conditions:

- $w, i \models p$ iff $p \in \sigma(i)$
- $w, i \models \neg F$ iff $w, i \models F$ does not hold
- $w, i \models F \land G$ iff both $w, i \models F$ and $w, i \models G$ hold
- $w, i \models F \cup_{a,b} G$ iff for some $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is:
  - $w, j \models G$ and for all $i \leq k < j$ it is $w, k \models F$

  *i.e.*, $F$ holds until $G$ will hold within $\langle a, b \rangle$

For derived operators:

- $w, i \models \diamond_{a,b} F$ iff for some $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models F$
  - *i.e.*, $F$ holds eventually within $\langle a, b \rangle$
- $w, i \models \Box_{a,b} F$ iff for all $i \leq j \leq n$ such that $t(j) - t(i) \in \langle a, b \rangle$ it is: $w, j \models F$
  - *i.e.*, $F$ holds always within $\langle a, b \rangle$
Metric Temporal Logic: Semantics

Def. Satisfaction:

\[ w \models F \iff w, 1 \models F \]

i.e., timed word \( w \) satisfies formula \( F \) initially

Def. Any MTL formula \( F \) defines a set of timed words \( \langle F \rangle \):

\[ \langle F \rangle \triangleq \{ w \in (2^P \times \mathbb{R})^* \mid w \models F \} \]

\( \langle F \rangle \) is called the language of \( F \)
Dense Real-time Model-Checking

What's Decidable?
TAs not Closed under Complement

A: a dense TA
F: a dense-time MTL formula

\[ A \not\equiv F \]

Fundamental problem:

- Dense timed automata are not closed under complement
  - The complement of the language of this TA isn't accepted by any TA:
    - **language** of this TA:
      "there exist two \( p \)'s separated by one t.u."
    - **complement** language:
      "no two \( p \)'s are separated by one t.u."
    - **intuition**: need a clock for each \( p \) within the past time unit, but there can be an unbounded number of such \( p \)'s because time is dense
TAs not Closed under Complement

Fundamental problem:

- Dense TAs are not closed under complement
- MTL is clearly closed under complement
  - Language of the TA: \( \Diamond ( p \land \Diamond =1 p ) \)
  - Complement language of the TA:
    \( \neg \Diamond ( p \land \Diamond =1 p ) = \Box ( p \Rightarrow \neg \Diamond =1 p ) \)
- Hence, automata-theoretic dense real-time model-checking is unfeasible
Dense MTL Model Checking is Undecidable

An even more fundamental problem:

- The **dense-time model-checking problem** for MTL and TAs is **undecidable** (for infinite words)
  - no approach is going to work, not just the automata-theoretic one

- MTL and TAs are “too expressive” over dense time
What's Decidable about Timed Automata

Let's revisit the three algorithmic components of automata-theoretic model checking:

- **MTL2TA**: given MTL formula $F$ build TA $a(F)$ such that $\langle F \rangle = \langle a(F) \rangle$
  - *undecidable* problem (for infinite words)
- **TA-Intersection**: given TAs $A$, $B$ build TA $C$ such that $\langle A \rangle \cap \langle B \rangle = \langle C \rangle$
  - *decidable*
- **TA-Emptiness**: given TA $A$ check whether $\langle A \rangle = \emptyset$ is the case
  - *decidable!
Dense Real-time Model-Checking

Intersection of Timed Automata
TA-Intersection: running TAs in parallel

Given TAs $A$, $B$ it is always possible to build automatically a TA $C$ that accepts precisely the words that both $A$ and $B$ accept.

TA $C$ represents all possible parallel runs of $A$ and $B$ where a timed word is accepted if and only if both $A$ and $B$ would accept it. The construction is called "product automaton".
**TA-Intersection: running TAs in parallel**

**Def.** Given TAs $A=[\Sigma, S^A, C^A, I^A, E^A, F^A]$ and $B=[\Sigma, S^B, C^B, I^B, E^B, F^B]$

let $C \triangleq A \times B \triangleq [\Sigma, S^C, C^C, I^C, E^C, F^C]$ be defined as:

- $S^C \triangleq S^A \times S^B$
- $C^C \triangleq C^A \cup C^B$ (assuming w.l.o.g. that they are disjoint sets)
- $I^C \triangleq \{ (s, t) \mid s \in I^A \text{ and } t \in I^B \}$
- $[(s, t), \sigma, c^A \land c^B, \rho^A \cup \rho^B, (s', t')] \in E^C$ iff
  
  $[s, \sigma, c^A, \rho^A, s'] \in E^A$ and $[t, \sigma, c^B, \rho^B, t'] \in E^B$

- $F^C \triangleq \{ (s, t) \mid s \in F^A \text{ and } t \in F^B \}$

**Theorem.**

$$\langle A \times B \rangle = \langle A \rangle \cap \langle B \rangle$$
TA-Intersection: Example

\[ x := 0 \quad \text{turn\_off} \quad \text{turn\_on}$ \quad x > 2 \]

\[ z := 0 \quad \text{start} \]

\[ z > 3 \quad \text{turn\_off} \]

\[ y := 0 \quad \text{stop} \quad y \leq 3 \]

\[ \text{cooking} \]

\[ \text{on} \]

\[ \text{cooking} \]

\[ \text{on} \]

\[ \text{off} \]

\[ \text{stop} \quad y := 0 \quad y \leq 3 \quad \text{start} \]
Dense Real-time Model-Checking

Checking the Emptiness of Timed Automata
TA-Emptiness

Given a TA $A$ it is always possible to check automatically if it accepts any timed word.

Outline of the algorithm:

- Assume that clock constraints involve **integer constants** only
  - this is without loss of generality as it amounts to scaling
- Define an **equivalence relation** over extended states
  - an extended state is a tuple $[s, v(1), ..., v(|C|)]$
    - with a location $s$ and a value $v(i)$ for every clock in $C$.
- All extended states in the same equivalence class are **equivalent w.r.t.** satisfaction of clock constraints
- The equivalence relation is such that there is a **finite number of equivalence classes** for any given TA
- Given a TA $A$, build an FSA $\text{reg}(A)$ - the “region automaton”:
  - the **states** of $\text{reg}(A)$ represent the equivalence classes of the extended states of any run of of $A$
  - the **edges** of $\text{reg}(A)$ represent evolution of any extended state within the equivalence class over any run of $A$
- Checking the **emptiness** of $\text{reg}(A)$ is equivalent to checking the emptiness of $A$
Integer vs. Rational vs. Irrational

- The domain for time is \( \mathbb{IR}_{\geq 0} \)

- What about the domain for time constraints?
  - Constants in clock constraints of TAs (e.g.: \( x < k \))

1. Same as the domain for time: \( \mathbb{IR}_{\geq 0} \)
   - e.g.: \( x < \pi \)
   - Emptiness becomes undecidable!

2. Discrete time domain: integers \( \mathbb{IN} \)
   - e.g.: \( x < 5 \)
   - Emptiness fully decidable (see algorithm next)

3. Dense but not continuous: rationals \( \mathbb{IQ}_{\geq 0} \)
   - e.g.: \( x < 1/3 \)
   - Emptiness is reducible to the integer case
Integer vs. Rational

- **Dense** but not continuous: rationals \( \mathbb{Q}_{\geq 0} \)
  
  - Let \( A \) be a TA with rational constants
    
    - let \( m \) be the least common multiple of denominators of all constants appearing in the clock constraints of \( A \)
    
    - let \( A^*m \) be the TA obtained from \( A \) by multiplying every constants in the clock constraints by \( m \)
      
      - \( A^*m \) has only integers constants in its clock constraints
    
    - \( A \) accepts any timed word
      
      \[ [\sigma(1), t(1)] [\sigma(2), t(2)] \ldots [\sigma(n), t(n)] \]
    
    iff \( A^*m \) accepts the “scaled” timed word
      
      \[ [\sigma(1), m*t(1)] [\sigma(2), m*t(2)] \ldots [\sigma(n), m*t(n)] \]
  
  - Hence checking the emptiness of \( A^*m \) is equivalent to checking the emptiness of \( A \)
Let us fix a TA $A = [\Sigma, S, C, I, E, F]$ with $C = [x(1), ..., x(n)]$

- For any clock $x(i)$ in $C$ let $M(i)$ be the largest constant involving clock $x(i)$ in any clock constraint in $E$
- Let $[v(1), ..., v(n)] \in \mathbb{R}_{\geq 0}^n$ denote a “clock evaluation” representing any assignment of values to clocks
- **Equivalence** of two clock evaluations: $[v(1), ..., v(n)] \sim [v'(1), ..., v'(n)]$ iff all of the following hold:
  1. For all $1 \leq i \leq n$: $\text{int}(v(i)) = \text{int}(v'(i))$ or $v(i), v'(i) > M(i)$
  2. For all $1 \leq i, j \leq n$ such that $v(i) \leq M(i)$ and $v(j) \leq M(j)$:
     $\text{frac}(v(i)) \leq \text{frac}(v(j))$ iff $\text{frac}(v'(i)) \leq \text{frac}(v'(j))$
  3. For all $1 \leq i \leq n$ such that $v(i) \leq M(i)$:
     $\text{frac}(v(i)) = 0$ iff $\text{frac}(v'(i)) = 0$
- Note: $\text{int}(x)$ is the integer part of $x$; $\text{frac}(x)$ is the fractional part of $x$
**Clock Regions**

**Def.** A clock region is an equivalence class of clock evaluations induced by the equivalence relation $\sim$.

- For a clock evaluation $\mathbf{v} = [v(1), \ldots, v(n)] \in \mathbb{R}_{\geq 0}^n$, $[[\mathbf{v}]]$ denotes the clock region $\mathbf{v}$ belongs to.
- As a consequence of the definition of $\sim$, any clock region can be uniquely characterized by a finite set of constraints on clocks.
  - $\mathbf{v} = [0.4; 0.9; 0.7; 0]$ for 4 clocks $w, x, y, z$
  - $[[\mathbf{v}]]$ is $z = 0 < w < y < x < 1$
- **Fact:** clock regions are always in finite number.
Clock Regions (cont'd)

More systematically:

- given a set of clocks $C = [x(1), \ldots, x(n)]$
- with $M(i)$ the largest constant appearing in constraints on clock $x(i)$

A clock region is uniquely characterized by

- For each clock $x(i)$ a constraint in the form:
  - $x(i) = c$ with $c = 0, 1, \ldots, M(i)$; or
  - $c - 1 < x(i) < c$ with $c = 1, \ldots, M(i)$; or
  - $x(i) > M(i)$

- For each pair of clocks $x(i), x(j)$ a constraint in the form

  - $\text{frac}(x(i)) < \text{frac}(x(j))$
  - $\text{frac}(x(i)) = \text{frac}(x(j))$
  - $\text{frac}(x(i)) > \text{frac}(x(j))$

(These are unnecessary if $x(i) = c$, $x(j) = c$, $x(i) > M(i)$, or $x(j) > M(j)$)
Clock Regions: Example

- Clocks $C = [x, y]$
- $M(x) = 2$; $M(y) = 3$
- All 60 possible clock regions:
  - 12 corner points
  - 30 open line segments
  - 18 open regions
Time-successors of Regions

- **Fact:** A clock evaluation \( \mathbf{v} \) satisfies a clock constraint \( \mathbf{c} \) iff any other clock evaluation in \( [[\mathbf{v}]] \) satisfies \( \mathbf{c} \)
  
  - Hence, we can say that a "clock region satisfies a clock constraint"

---

**Def.** The *time successor* \( \text{time-succ}(R) \) of a clock region \( R \) is the set of all clock regions (including \( R \) itself) that can be reached from \( R \) by letting time pass (i.e., without resetting any clock).

---

Given a clock region \( R \) it is always possible to compute all other clock regions that can be reached from \( R \) by letting time pass (i.e., without resetting any clock)

- **Graphically:**
  
  - the time-successors of a region \( R \) are the regions that can be reached by moving along a line parallel to the diagonal in the upward direction, starting from any point in \( R \)

( For a precise definition see e.g.: Alur & Dill, 1994 )
Time-successors of Regions: Example

- **Graphically:**
  - the time-successors of a region $R$ are the regions that can be reached by moving along a line parallel to the diagonal in the upward direction, starting from any point in $R$.

- **Example:**
  - successors of region $2 < y < 3; 1 < x < y-1$
    (other than the region itself):
      - $y > 3; 1 < x < 2$
      - $y > 3; x = 2$
      - $y = 3; 1 < x < 2$
      - $y > 3; x > 2$
  - successors of region $y = 1; x = 2$
    (other than the region itself):
      - $2 < y < 3; x > 2$
      - ...

![Graph](image_url)
Region Automaton Construction

For a timed automaton $A$ it is always possible to build an FSA $\text{reg}(A)$ (the "region automaton" of $A$) such that:

$$\langle A \rangle = \emptyset \quad \text{iff} \quad \langle \text{reg}(A) \rangle = \emptyset$$

**Def.** Given a TA $A = [\Sigma, S, C, I, E, F]$ its region automaton $\text{reg}(A) \triangleq [\Sigma, rS, rI, rE, rF]$ is defined as:

- $rS \triangleq \{ (s, r) \mid s \in S \text{ and } r \text{ is a clock region} \}$
- $rI \triangleq \{ (s, [[0, 0, \ldots, 0]]) \mid s \in I \}$
  - the clock region where all clocks are reset to 0
- $rE(\sigma, [s, r]) \triangleq \{ (s', r') \mid [s, \sigma, c, \rho, s'] \in E$
  
  and there exists a region $r'' \in \text{time-succ}(r)$ such that $r''$ satisfies $c$, and $r'$ is obtained from $r''$ by resetting all clocks in $\rho$ to 0

- $rF \triangleq \{ (s, r) \mid s \in F \}$
Region Automaton: Example

$x := 0$ turn_off
turn_on $x > 1$

$y := 0$ stop
start $y \leq 1$

cooking

off $x = y = 0$
off $x > 1; y > 1$
off $0 < y < 1 < x$
off $y = 1 < x$

on $0 = x < y < 1$

on $x = 0; y = 1$

on $x = 0; y > 1$

cooking $0 = y < x < 1$

cooking $y = 0; x = 1$

cooking $y = 0; x > 1$

on $x > 1; y > 1$

on $0 < y < x < 1$

on $0 < y < 1 = x$

on $y = 1; x > 1$

$x$

1

$y$

0

1
Dense Real-time Model-Checking

Tools
Tools for the Analysis of TAs

- **Uppaal** (Larsen, Pettersson, Yi et al., ~1995)
- **Kronos** (Tripakis, Yovine et al., ~1995)
- **HyTech** (Henzinger et al., ~1994)
- **PHAVer** (Frehse, ~2005)

**Remark:** emptiness checking is also called "reachability analysis"

- the language of a TA $A$ is empty iff the accepting states of $A$ cannot be reached in any computation