Hybrid and Timed Systems
modeling, theory, verification

Eugene Asarin

LIAFA - University Paris Diderot and CNRS

Shanghai Computer Science Summer School 2010
Hybrid Systems

- Hybrid Systems = Discrete + Continuous
- Hybrid Automata = A model of Hybrid systems
- Original motivation = Physical plant + Digital controller
- New applications = biology, economy, numerics, circuits
- Hybrid community = Control scientists + Applied mathematicians + Some computer scientists
### Hybrid Systems

- **Hybrid Systems** = Discrete + Continuous
- **Hybrid Automata** = A model of Hybrid systems
- **Original motivation** = Physical plant + Digital controller
- **New applications** = biology, economy, numerics, circuits
- **Hybrid community** = Control scientists + Applied mathematicians + Some computer scientists

### Timed Systems

- **Timed Systems** = Discrete behavior + Continuous Time
- **Timed Automata** = A subclass of Hybrid automata
- **The starting point** = A beautiful result by Alur & Dill.
- **Applications** = Real-time digital system, etc...
- **Timed community** = Computer scientists
Global Outline

1. Hybrid Automata
2. Timed Automata
3. Back to Hybrid: Decidable Subclasses
Hybrid and Timed Systems

Eugene Asarin

Outline

1 Hybrid automata: the model
   An example
   Definition of HA
   Classes of HA
   A couple of exercises

2 Verification of HA
   The reachability problem
   The curse of undecidability
   How to verify HA: theory and practice
Outline

1 Hybrid automata: the model
   - An example
   - Definition of HA
   - Classes of HA
   - A couple of exercises

2 Verification of HA
   - The reachability problem
   - The curse of undecidability
   - How to verify HA: theory and practice
The first (cyber-physical) example

Notation
For $x = x(t)$ we write $\dot{x} = \dot{x}(t) = x'(t) = \frac{dx}{dt}$.
The first (cyber-physical) example

Notation
For $x = x(t)$ we write $\dot{x} = \dot{x}(t) = x'(t) = dx/dt$.

A thermostat

- When the heater is OFF, the room cools down:
  \[ \dot{x} = -x \]
- When it is ON, the room heats:
  \[ \dot{x} = H - x \]
The first (cyber-physical) example

Notation
For \( x = x(t) \) we write \( \dot{x} = \dot{x}(t) = x'(t) = dx/dt \).

A thermostat

• When the heater is OFF, the room cools down:

  \[ \dot{x} = -x \]

• When it is ON, the room heats:

  \[ \dot{x} = H - x \]

• When \( t > M \) it switches OFF
• When \( t < m \) it switches ON
The first (cyber-physical) example

Notation
For $x = x(t)$ we write $\dot{x} = \dot{x}(t) = x'(t) = dx/dt$.

A thermostat

- When the heater is OFF, the room cools down:
  $$\dot{x} = -x$$

- When it is ON, the room heats:
  $$\dot{x} = H - x$$

- When $t > M$ it switches OFF
- When $t < m$ it switches ON

A strange creature...
A bad syntax

Some mathematicians prefer to write

\[ \dot{x} = f(x, q) \]

where

\begin{align*}
    f(x, \text{Off}) &= -x \\
    f(x, \text{On}) &= H - x
\end{align*}

with some switching rules on \( q \).
Some mathematicians prefer to write

\[ \dot{x} = f(x, q) \]

where

\[ f(x, \text{Off}) = -x \]
\[ f(x, \text{On}) = H - x \]

with some switching rules on \( q \).

But we are computer scientists and draw an automaton
Hybrid and Timed Systems

Eugene Asarin

Hybrid automata: the model
An example
Definition of HA
Classes of HA
A couple of exercises

Verification of HA
The reachability problem
The curse of undecidability
How to verify HA: theory and practice

Hybrid automaton

\[
\begin{align*}
\text{On:} & \quad \dot{x} = H - x \\
& \quad x \leq M \\
\text{Off:} & \quad \dot{x} = -x \\
& \quad x \geq m
\end{align*}
\]

\[ x = m / \gamma \]

Guard
Reset
Invariant
Dynamics
Label
A formal definition: It is a tuple . . .
Hybrid automata: the model

An example

Definition of HA

Classes of HA

A couple of exercises

Verification of HA

The reachability problem

The curse of undecidability

How to verify HA: theory and practice
Hybrid automata: the model

Definition of HA

Classes of HA

A couple of exercises

Verification of HA

The reachability problem
The curse of undecidability
How to verify HA: theory and practice

Hybrid and Timed Systems

Eugene Asarin

On

\[ \dot{x} = H - x \]
\[ x \leq M \]

Off

\[ \dot{x} = -x \]
\[ x \geq m \]

\[ x = m / \gamma \] guard

\[ x = M \] reset

label

dynamics

invariant
**Definition**

A hybrid automaton is $H = (Q, X, \Sigma, Dyn, I, \Delta)$ with

- $Q$ finite set of locations
- $X = \mathbb{R}^n$, continuous state space
- $Dyn$, dynamics on $X$ for every $q \in Q$
- $I$, invariant, staying condition in $X$
- $\Delta$, finite set of transitions $\delta = (p, q, a, g, r)$
**Definition**

A hybrid automaton is $H = (Q, X, \Sigma, Dyn, I, \Delta)$ with

- $Q$ finite set of locations
- $X = \mathbb{R}^n$, continuous state space, a point in $X = \{x_1, \ldots, x_n\}$
- $Dyn$, dynamics on $X$ for every $q \in Q$
- $I$, invariant, staying condition in $X$
- $\Delta$, finite set of transitions $\delta = (p, q, a, g, r)$
Definition

A hybrid automaton is $H = (Q, X, \Sigma, Dyn, I, \Delta)$ with

- $Q$, finite set of locations
- $X = \mathbb{R}^n$, continuous state space
- $Dyn$, dynamics on $X$ for every $q \in Q$, $Dyn(q) = f_q$, whenever in location $q$ the continuous state obeys $\dot{x} = f_q(x)$.
- $I$, invariant, staying condition in $X$
- $\Delta$, finite set of transitions $\delta = (p, q, a, g, r)$
Definition

A hybrid automaton is \( H = (Q, X, \Sigma, \text{Dyn}, I, \Delta) \) with

- \( Q \) finite set of locations
- \( X = \mathbb{R}^n \), continuous state space
- \( \text{Dyn} \), dynamics on \( X \) for every \( q \in Q \)
- \( I \), invariant, staying condition in \( X \), whenever in location \( q \) the continuous state obeys \( x \in I(q) \).
- \( \Delta \), finite set of transitions \( \delta = (p, q, a, g, r) \)
Definition

A hybrid automaton is $H = (Q, X, \Sigma, Dyn, I, \Delta)$ with

- $Q$ finite set of locations
- $X = \mathbb{R}^n$, continuous state space
- $Dyn$, dynamics on $X$ for every $q \in Q$
- $I$, invariant, staying condition in $X$
- $\Delta$, finite set of transitions $\delta = (p, q, a, g, r)$
  - $p, q \in Q$, from $p$ to $q$
  - $a \in \Sigma$ a label
  - $g$ a guard; $g(x)$ required to take $\delta$
  - $r$ a reset (or jump); $x := r(x)$ when taking $\delta$
Hybrid and Timed Systems

Eugene Asarin

Hybrid automata: the model
An example
Definition of HA
Classes of HA
A couple of exercises

Verification of HA
The reachability problem
The curse of undecidability
How to verify HA: theory and practice

Trajectory-based semantics

On
\[ \dot{x} = H - x \]
\[ x \leq M \]

Off
\[ \dot{x} = -x \]
\[ x \geq m \]

Guard
\[ x = M \]

Reset
\[ \dot{x} = m / \gamma \]

Invariant

Label

Dynamics
A trajectory-based semantics

An example

Definition of HA

Classes of HA

A couple of exercises

Verification of HA

The reachability problem

The curse of undecidability

How to verify HA: theory and practice

A trajectory: $\xi: [0, T] \rightarrow Q \times \mathbb{R}$

Trajectory: $\xi: [0, T] \rightarrow Q \times \mathbb{R}$

$\dot{x} = H - x$

$x \leq M$

$\dot{x} = -x$

$x \geq m$

$x = M$

$x = m / \gamma$

On

Off
Transition system semantics

Transition system \((S, T)\) of a HA

- **States:** \(S = Q \times \mathbb{R}^n\)
- **Transitions:** \(T = T_{\text{flow}} \cup T_{\text{jump}}\)
Transition system semantics

Transition system \((S, T)\) of a HA

- **States:** \(S = Q \times \mathbb{R}^n\)
- **Transitions:** \(T = T_{\text{flow}} \cup T_{\text{jump}}\)
  - \((q, x_1) \xrightarrow{\text{flow}} (q, x_2) \iff\) we can go from \(x_1\) to \(x_2\) in ODE \(\dot{x} = f_q(x)\)
  - \((q_1, x_1) \xrightarrow{\text{jump}} (q_2, x_2) \iff\) if we can jump.
Transition system semantics

Transition system \((S, T)\) of a HA

- **States:** \(S = Q \times \mathbb{R}^n\)
- **Transitions:** \(T = T_{\text{flow}} \cup T_{\text{jump}}\)
- **Runs:** sequences of states and transitions.
Transition system (S, T) of a HA

- **States:** \( S = Q \times \mathbb{R}^n \)
- **Transitions:** \( T = T_{\text{flow}} \cup T_{\text{jump}} \)
- **Runs:** sequences of states and transitions.

\[(\text{On}, 0) \xrightarrow{\text{flow}} (\text{On}, M) \xrightarrow{\text{jump}} (\text{Off}, M) \xrightarrow{\text{flow}} (\text{Off}, m) \xrightarrow{\text{jump}} (\text{On}, m) \cdot \]
The two semantics are almost equivalent

- **Trajectories**: the system is filmed continuously
  
  \[ \begin{align*}
  \text{flow} & : (\text{On}, 0) \rightarrow (\text{On}, M) \\
  \text{jump} & : (\text{On}, M) \rightarrow (\text{Off}, M) \\
  \text{flow} & : (\text{Off}, M) \rightarrow (\text{Off}, m) \\
  \text{jump} & : (\text{Off}, m) \rightarrow (\text{On}, m)
  \end{align*} \]

- **Runs (in transition system)**: photos are taken when something happens

\[ (\text{On}, 0) \xrightarrow{\text{flow}} (\text{On}, M) \xrightarrow{\text{jump}} (\text{Off}, M) \xrightarrow{\text{flow}} (\text{Off}, m) \xrightarrow{\text{jump}} (\text{On}, m) \]
Semantic issues

- Some mathematical complications (notion of solution, existence and unicity not so evident).
- Zeno trajectories (infinitely many transitions in a finite period of time).
  - can be forbidden
  - one can consider trajectories up to the first anomaly (Sastry et al., everything OK)
  - one can consider the complete Zeno trajectories (very funny: Asarin-Maler 95)
Variants

- Discrete-time \((x_{n+1} = f_q(x_n))\) or continuous-time \(\dot{x} = f(x)\)
- Deterministic (e.g. \(\dot{x} = f(x)\)) or non-deterministic (e.g. \(\dot{x} \in F(x)\))
- Eager or lazy.
- With control and/or disturbance (e.g. \(\dot{x} = f(x, u, d)\))
- Various restrictions on dynamics, guards and resets: “Piecewise trivial dynamics”. LHA, RectA, PCD, PAM, SPDI . . . They are still highly non-trivial.
Classes of Hybrid Automata

Why classes?
Because HA are too reach; it is impossible to establish, decide, analyze properties of all HA.
Classes of Hybrid Automata

Why classes?
Because HA are too reach; it is impossible to establish, decide, analyze properties of all HA.

How to define a class of HA
- dimension, discrete or continuous time, eager or lazy
- what kind of dynamics
- what kind of guards/invariants/jumps
Special classes of Hybrid Automata 1

The famous one

*Linear Hybrid Automata*

\[ \dot{x} = c_1 \]

\[ \dot{x} = c_2 \]

\[ x \in P_1/x := A_1 x + b_1 \]

\[ x \in P_2/x := A_2 x + b_2 \]

Class specification

Dynamics \( \dot{x} = c \); polyhedral guards and invariants; linear resets.
Special classes of Hybrid Automata 1’

Zoology of variables

- Memory cell: $\dot{x} = 0$ (but $x$ can be reset)
- Clock: $\dot{x} = 1$.
- Stopwatch: in some locations $\dot{x} = 1$, in others $\dot{x} = 0$.
- Skewed clock: $\dot{x} = c$ (the same for every location).
Special classes of Hybrid Automata 1’

Zoology of variables

- Memory cell: \( \dot{x} = 0 \) (but \( x \) can be reset)
- Clock: \( \dot{x} = 1 \).
- Stopwatch: in some locations \( \dot{x} = 1 \), in others \( \dot{x} = 0 \).
- Skewed clock: \( \dot{x} = c \) (the same for every location).

One can define subclasses of LHA like that:

*LHA with 3 stopwatches and 1 skewed clock, with resets only to 0 and guards only \( x < c \).*
Special classes of Hybrid Automata 2

My favorite class

\[ PCD = \text{Piecewise Constant Derivatives} \]

\[ \dot{x} = c_i \text{ for } x \in P_i \]
PCD is a linear hybrid automaton (LHA)
PCD is a linear hybrid automaton (LHA)
PCD is a linear hybrid automaton (LHA)
PCD is a linear hybrid automaton (LHA)
PCD is a linear hybrid automaton (LHA)
**Special classes of Hybrid Automata 3**

**The most illustrative**

*Piecewise Affine Maps*

\[ x := A_i x + b_i \text{ for } x \in P_i \]
Special classes of Hybrid Automata 3

The most illustrative

*Piecewise Affine Maps*

\[ x := A_i x + b_i \text{ for } x \in P_i \]

Remark: PAM are discrete time.
How to model?

Different systems

- a control system
- a scheduler with preemption
- a genetic network
How to model?

Different systems

- a control system
- a scheduler with preemption
- a genetic network

The same class of models
A network of interacting Hybrid automata
Hybrid tools

Try to play with some tools, see at:
Maybe start with UPPAAL (timed) and PHAVER (hybrid).
 Modeling exercise 1

Genetic network

We consider expression of two genes A and B, i.e. production of two proteins P and Q

- The proteins are degraded with rate $k$.
- P catalyzes expression of B:
  - Production of Q is proportional to the concentration of P with a coefficient $a$.
  - Concentration of P crosses a threshold $s \Rightarrow$ production of Q constant $= as$.
- Q inhibits expression of A:
  - Production of P equals $d - b \cdot \text{(concentration of Q)}$.
  - Concentration of Q crosses a threshold $r \Rightarrow$ production of P blocks.
Modeling exercise 2

Scheduling
Schedule two jobs on one CPU and one printer with a total execution time up to 16 minutes.

- Job 1: Compute (10 min); Print (5 min)
- Job 2: Download (3 min); Compute (1 min); Print (2 min)

Try it:
1. without preemption;
2. with preemptible computing.
What to do with a hybrid model

- Simulate
  - With Matlab/Simulink
  - With dedicated tools
- Analyze with techniques from control science:
  - Stability analysis
  - Optimal control
  - etc..
- Analyze with your favorite techniques. The most important invention is the model.
What to do with a hybrid model

- Simulate
  - With Matlab/Simulink
  - With dedicated tools
- Analyze with techniques from control science:
  - Stability analysis
  - Optimal control
  - etc..
- Analyze with your favorite techniques. The most important invention is the model.
  - We will try **verification**
Outline

1 Hybrid automata: the model
   - An example
   - Definition of HA
   - Classes of HA
   - A couple of exercises

2 Verification of HA
   - The reachability problem
   - The curse of undecidability
   - How to verify HA: theory and practice
• Is automatic verification possible for HA?
Verification and reachability problems

- Is automatic verification possible for HA?
- Safety: are we sure that HA never enters a bad state?
- It can be seen as reachability: verify that
  \[ \neg \text{Reach}(\text{Init}, \text{Bad}) \]
Verification and reachability problems

- Is automatic verification possible for HA?
- *Safety*: are we sure that HA never enters a bad state?
- It can be seen as reachability: verify that
  \[ \neg \text{Reach}(\text{Init}, \text{Bad}) \]
- It is a natural and challenging mathematical problem.
- Many works on decidability
- Some works on approximated techniques
The reachability problem for a class $C$

**Problem**

**Given**

- a hybrid automaton $\mathcal{H} \in C$
- two sets $A, B \subseteq Q \times \mathbb{R}^n$

find out whether there exists a trajectory of $\mathcal{H}$ starting in $A$ and arriving to $B$.

All parameters rational.
Exact methods: The curse of undecidability

Bad news

- Koiran et al.: Reach is undecidable for 2d PAM.
- AM95: Reach is undecidable for 3d PCD.
- HPKV95 Many results of the type: “3clocks + 2 stopwatches = undecidable”
Exact methods: The curse of undecidability

Bad news

- Koiran et al.: Reach is undecidable for 2d PAM.
- AM95: Reach is undecidable for 3d PCD.
- HPKV95: Many results of the type: “3clocks + 2 stopwatches = undecidable”

They are really bad

- Reachability is undecidable for very simple HA.
- Thus, other verification problems are also undecidable.
Hybrid and Timed Systems

Eugene Asarin

Hybrid automata: the model
An example
Definition of HA
Classes of HA
A couple of exercises

Verification of HA
The reachability problem
The curse of undecidability
How to verify HA: theory and practice

Undecidability Proofs — Preliminaries

Proof method:
simulation of Minsky Machine, Turing Machine etc.
Undecidability Proofs — Preliminaries

Proof method: simulation of Minsky Machine, Turing Machine etc.

Details: proof schema

- Reachability undecidable for Minsky Machines (well-known).
- A class of HA can simulate MM (to prove).
- Reach for MM $\preceq$ Reach for HA.
- Conclude that Reach for HA is undecidable.
Minsky Machines

Definition

- A counter: values in \( \mathbb{N} \); operations: \( C++ \), \( C-- \); test \( C > 0 \)?
- A Minsky machine has 2 counters
- Its program has finitely many lines like that:
  - \( q_1 : \quad D++; \quad \text{goto} \ q_2 \)
  - \( q_2 : \quad C--; \quad \text{goto} \ q_3 \)
  - \( q_3 : \quad \text{if} \ C > 0 \quad \text{then goto} \ q_2 \ \text{else} \ q_1 \)
Minsky Machines

Definition

- A counter: values in $\mathbb{N}$; operations: $C + +$, $C − −$; test $C > 0$?
- A Minsky machine has 2 counters
- Its program has finitely many lines like that:
  \[
  q_1 : \quad D + + ; \quad \text{goto } q_2 \\
  q_2 : \quad C − − ; \quad \text{goto } q_3 \\
  q_3 : \quad \text{if } C > 0 \quad \text{then goto } q_2 \quad \text{else } q_1
  \]

Theorem (Minsky)

Reachability is undecidable for Minsky machines.
Minsky Machines

Definition

- A counter: values in $\mathbb{N}$; operations: $C++$, $C--$; test $C > 0$?
- A Minsky machine has 2 counters
- Its program has finitely many lines like that:
  
  $q_1 : \quad D++ ; \text{ goto } q_2$
  
  $q_2 : \quad C-- ; \text{ goto } q_3$
  
  $q_3 : \text{ if } C > 0 \text{ then goto } q_2 \text{ else } q_1$

Theorem (Minsky)

*Reachability is undecidable for Minsky machines.*

Fact

*Any algorithm can be programmed on a Minsky machine. But they are slooooonooow.*
A typical undecidability theorem

Theorem (Koiran, Cosnard, Garzon)
Reach is undecidable for 2d PAM.
A typical undecidability theorem

**Theorem (Koiran, Cosnard, Garzon)**

Reach is undecidable for 2d PAM.

**Reminder**

A 2 dimensional PAM:

\[ \mathbf{x} := A_i \mathbf{x} + \mathbf{b}_i \text{ for } \mathbf{x} \in P_i \]
Simulating a counter by a PAM

<table>
<thead>
<tr>
<th>Counter</th>
<th>PAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>State space $\mathbb{N}$</td>
<td>State space $[0; 1]$</td>
</tr>
<tr>
<td>State $C = n$</td>
<td>$x = 2^{-n}$</td>
</tr>
<tr>
<td>$C++$</td>
<td>$x := x/2$</td>
</tr>
<tr>
<td>$C--$</td>
<td>$x := 2x$</td>
</tr>
<tr>
<td>$C &gt; 0?$</td>
<td>$x &lt; 0.75?$</td>
</tr>
</tbody>
</table>
### Encoding a state of a Minsky Machine

<table>
<thead>
<tr>
<th>Minsky Machine</th>
<th>PAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>State space ${q_1, \ldots, q_k} \times \mathbb{N} \times \mathbb{N}$</td>
<td>State space $[1; k+1] \times [0; 1]$</td>
</tr>
<tr>
<td>State $(q_i, C = m, D = n)$</td>
<td>$x = i + 2^{-m}, y = 2^{-n}$</td>
</tr>
</tbody>
</table>
### Simulating a Minsky Machine

<table>
<thead>
<tr>
<th>Minsky Machine</th>
<th>PAM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State space</strong> ${q_1, \ldots, q_k} \times \mathbb{N} \times \mathbb{N}$</td>
<td>State space $[1; k + 1] \times [0; 1]$</td>
</tr>
<tr>
<td><strong>State</strong> $(q_i, C = m, D = n)$</td>
<td>$x = i + 2^{-m}, y = 2^{-n}$</td>
</tr>
</tbody>
</table>

- **$q_1$**: $D++$; goto $q_2$
  \[
  \begin{align*}
  x &:= x + 1 \\
  y &:= y/2
  \end{align*}
  \quad \text{if } 1 < x \leq 2
  
- **$q_2$**: $C--$; goto $q_3$
  \[
  \begin{align*}
  x &:= 2(x - 2) + 3 \\
  y &:= y
  \end{align*}
  \quad \text{if } 2 < x \leq 3
  
- **$q_3$**: if $C > 0$ then goto $q_2$ else $q_1$
  \[
  \begin{align*}
  x &:= x - 1 \\
  y &:= y
  \end{align*}
  \quad \text{if } 3 < x < 4
  
  \begin{align*}
  x &:= x - 2 \\
  y &:= y
  \end{align*}
  \quad \text{if } x = 4
...finally we have proved:

**Theorem (Koiran et al.)**

Reach *is undecidable for 2d PAMs.*
... finally we have proved:

**Theorem (Koiran et al.)**
Reach *is undecidable for 2d PAMs.*

It follows immediately:

**Theorem (Henzinger et al.)**
Reach *is undecidable for LHA.*
Theorem (Koiran et al.)
Reach is undecidable for 2d PAMs.
It follows immediately:

Theorem (Henzinger et al.)
Reach is undecidable for LHA.
Jumps are not necessary for undecidability:

Theorem (A., Maler, Pnueli)
Reach is undecidable for 3d PCD.
Unrelated: a difficult problem

- 1d piecewise affine maps (PAMs): $f : \mathbb{R} \rightarrow \mathbb{R}$
  
  $f(x) = a_i x + b_i$ for $x \in l_i$

**Diagram**

- $a_1 x + b_1$
- $\mathbb{R}$
- $l_1$, $l_4$, $l_3$, $l_2$, $l_5$
Unrelated: a difficult problem

- 1d piecewise affine maps (PAMs): \( f : \mathbb{R} \rightarrow \mathbb{R} \)
  \[ f(x) = a_i x + b_i \text{ for } x \in I_i \]
Unrelated: a difficult problem

- 1d piecewise affine maps (PAMs): \( f : \mathbb{R} \rightarrow \mathbb{R} \)
  \( f(x) = a_i x + b_i \) for \( x \in l_i \)

\[ a_1 x + b_1 \]
\[ a_4 x + b_4 \]
\[ a_5 x + b_5 \]
Unrelated: a difficult problem

- 1d piecewise affine maps (PAMs): \( f : \mathbb{R} \rightarrow \mathbb{R} \)
  \[ f(x) = a_i x + b_i \text{ for } x \in l_i \]
Unrelated: a difficult problem

- 1d piecewise affine maps (PAMs): $f : \mathbb{R} \rightarrow \mathbb{R}$
  $f(x) = a_i x + b_i$ for $x \in l_i$

Old Open Problem

Is reachability decidable for 1d PAM?
Conclusions of Day 1

We have learned today

- What is a Hybrid Automaton.
- How to read yet another definition of HA and its semantics.
- How to model things using HA.
- Famous classes of HA.
- Safety verification as reachability problem.
- How to prove undecidability by simulation of Minsky Machines.
- Even the simplest classes of HA have undecidable reachability.
In practice approximate/bounded methods should be used for safety verification.

Several tools, many methods.

General principles are easy, implementation difficult.
Abstract algorithm - important

A generic verification algorithm A

Forward breadth-first search

F=Init

repeat

\[ F := F \cup \text{SuccFlow}(F) \cup \text{SuccJump}(F) \]

until \( (F \cap \text{Bad} \neq \emptyset) \) | fixpoint | tired

say "reachable" | "unreachable" | "timeout"

Most verification methods and tools are variants of it.
Abstract algorithm - important

A generic verification semi-algorithm A

Forward breadth-first search

\[ F := \text{Init} \]
\[ \text{repeat} \]
\[ F := F \cup \text{SuccFlow}(F) \cup \text{SuccJump}(F) \]
\[ \text{until} \ (F \cap \text{Bad} \neq \emptyset) \mid \text{fixpoint} \mid \text{tired} \]
\[ \text{say } "\text{reachable}" \mid "\text{unreachable}" \mid "\text{timeout}" \]

Most verification methods and tools are variants of it.
Abstract algorithm - important

A generic verification semi-algorithm A
Forward breadth-first search

F = Init
repeat
    F = F ∪ SuccFlow(F) ∪ SuccJump(F)
until (F ∩ Bad ≠ ∅) | fixpoint | tired
say "reachable" | "unreachable" | "timeout"

There are variants:

• forward/backward
• breadth first/depth first/best first/etc.

Most verification methods and tools are variants of it.
How to implement it

Needed data structure for representation of subsets of $\mathbb{R}^n$, and algorithms for efficient computing of

- unions, intersections;
- inclusion tests;
- SuccFlow;
- SuccJump.
How to implement it

Needed data structure for representation of subsets of $\mathbb{R}^n$, and algorithms for efficient computing of

- unions, intersections;
- inclusion tests;
- SuccFlow;
- SuccJump.

It could be exact or over-approximate.
Some trivial results

Theorem

\textit{If for a class of HA the Algorithm A can be implemented (exactly), then}

\begin{itemize}
  \item Reach \textit{is semi-decidable};
  \item \textit{bounded} Reach \textit{in n steps is decidable};
  \item a verification tool can be built.
\end{itemize}
Some trivial results

**Theorem**

*If for a class of HA the Algorithm A can be implemented (exactly), then*

- Reach is semi-decidable;
- *bounded* Reach in *n* steps is decidable;
- a verification tool can be built.

**Fact**

*Suppose for a class of HA the Algorithm A can be implemented approximately. Then we can build a verification tool saying:*

- “Unreachable”.
- “Maybe reachable”.
- “Timeout”.
One important implementation for linear hybrid automata

We want to represent subsets of $Q \times \mathbb{R}^n$.

Objects and data structure

- A polyhedron $P$ defined by $Ax \leq \vec{b}$. Data structure $(A, \vec{b})$.
- Several polyhedra $P_1, \ldots, P_n$. Data structure $(A_1, \vec{b}_1), \ldots, (A_n, \vec{b}_n)$,
- The same with the discrete state: $(q_1, P_1), \ldots, (q_n, P_n)$.

Fact

The data structure $(q_1, A_1, \vec{b}_1), \ldots, (q_n, A_n, \vec{b}_n)$ representing union of polyhedra in $Q \times \mathbb{R}^n$ allows implementing semi-algorithm $A$ for LHA.

We will prove it.
Towards the proof

(for simplicity, I forget about $q$)

**Definition**
Linear constraint $C$ on $\mathbb{R}^n$ is $\sum a_ix_i \leq b$ (or $<$).

**Polyhedra and logic**
Our data structure can be written in logic

- A polyhedron: $\bigwedge_i C_i$.
- A union of polyhedra - a DNF: $\bigvee_i \bigwedge_j C_{ij}$

Example: what is $(x > 0 \land y > 0 \land x + y < 3) \lor x > 7$?
Definition

A first order logic of linear constraints (FOLC).

\[ F ::= \sum a_i x_i \leq c | \neg F | F \lor F | \exists x F \]

(we can also express \(<, =, \wedge, \forall\))
A richer formalism

**Definition**

A first order logic of linear constraints (FOLC).

\[ F ::= \sum a_i x_i \leq c | \neg F | F \lor F | \exists x F \]

(we can also express <, =, \&\&, \forall)

It expresses polyhedra, unions of polyhedra and other things

\[ Q(u, v) = \exists x \forall y (x < 3 \lor y > 2 \lor 3u + x + v < 17) \]
A richer formalism

Definition
A first order logic of linear constraints (FOLC).

\[ F ::= \sum a_i x_i \leq c | \neg F | F \lor F | \exists x F \]

(we can also express <, =, ∧, ∀)

It expresses polyhedra, unions of polyhedra and other things

\[ Q(u, v) = \exists x \forall y (x < 3 \lor y > 2 \lor 3u + x + v < 17) \]

Theorem

\textit{FOLC admits quantifier elimination}
A richer formalism

Definition
A first order logic of linear constraints (FOLC).

\[ F ::= \sum a_i x_i \leq c | \neg F | F \lor F | \exists x F \]

(we can also express \(<, =, \land, \forall\))

It expresses polyhedra, unions of polyhedra and nothing else

\[ Q(u, v) = \exists x \forall y (x < 3 \lor y > 2 \lor 3u + x + v < 17) \]

Theorem
FOLC admits quantifier elimination

That is, every formula ⇔ a DNF of linear constraints (a union of polyhedra).
How to remove quantifiers?

**Theorem**

\textit{FOLC admits quantifier elimination}
How to remove quantifiers?

Theorem

**FOLC admits quantifier elimination**

Let us remove one quantifier.

Lemma (Fourier-Motzkine)

\[ \exists y \bigwedge_j C_j \text{ can be written as } \bigwedge_j C'_j. \]
How to remove quantifiers?

Theorem

*FOLC admits quantifier elimination*

Let us remove one quantifier.

**Lemma (Fourier-Motzkine)**

\[ \exists y \land_j C_j \text{ can be written as } \land_j C'_j. \]

The same thing in geometric language

Projection of a polyhedron in \( \mathbb{R}^{n+1} \) onto \( \mathbb{R}^n \) is a polyhedron.
How to remove quantifiers?

**Theorem**

*FOLC admits quantifier elimination*

Let us remove one quantifier.

**Lemma (Fourier-Motzkine)**

\[ \exists y \land_j C_j \text{ can be written as } \land_j C'_j. \]

**The same thing in geometric language**

Projection of a polyhedron in \( \mathbb{R}^{n+1} \) onto \( \mathbb{R}^n \) is a polyhedron.

To prove theorem apply FM several times.
Understanding Fourier-Motzkine

Project a 3D polyhedron to the plane \((x, y)\).

\[
F = \exists z \left\{ \begin{array}{c}
2x + y + z \leq 10 \\
3x + 2y \leq 11 \\
-x - y + z \leq 4 \\
6x - y - z \leq 9 \\
\end{array} \right. 
\]
Understanding Fourier-Motzkine

Project a 3D polyhedron to the plane \((x, y)\).

\[
F = \exists z \begin{cases} 
2x + y + z & \leq 10 \\
3x + 2y & \leq 11 \\
-x - y + z & \leq 4 \\
6x - y - z & \leq 9 
\end{cases}
\]

Let us isolate \(z\):

\[
\begin{cases} 
z & \leq 10 - 2x - 2y \\
3x + 2y & \leq 11 \\
z & \leq 4 + x + y \\
6x - y - 9 & \leq z 
\end{cases}
\]
Understanding Fourier-Motzkine

Project a 3D polyhedron to the plane \((x, y)\).

\[
F = \exists z \left\{ \begin{array}{c}
2x + y + z \leq 10 \\
3x + 2y \leq 11 \\
-x - y + z \leq 4 \\
6x - y - z \leq 9
\end{array} \right\}
\]

Let us isolate \(z\):

\[
\left\{ \begin{array}{c}
z \leq 10 - 2x - 2y \\
3x + 2y \leq 11 \\
z \leq 4 + x + y \\
6x - y - 9 \leq z
\end{array} \right\}
\]

3 groups: no \(z\); lower bounds on \(z\); upper bounds on \(z\).

\[
\left\{ \begin{array}{c}
3x + 2y \leq 11 \\
6x - y - 9 \leq z
\end{array} \right\} \left\{ \begin{array}{c}
z \leq 10 - 2x - 2y \\
z \leq 4 + x + y
\end{array} \right\}
\]
Hybrid and Timed Systems

Eugene Asarin

Hybrid automata: the model

An example
Definition of HA
Classes of HA
A couple of exercises

Verification of HA

The reachability problem
The curse of undecidability
How to verify HA: theory and practice

Understanding Fourier-Motzkine

Project a 3D polyhedron to the plane \((x, y)\).

\[
F = \exists z \begin{cases}
2x + y + z & \leq 10 \\
3x + 2y & \leq 11 \\
-x - y + z & \leq 4 \\
6x - y - z & \leq 9 \\
\end{cases}
\]

Let us isolate \(z\):

\[
\begin{cases}
z & \leq 10 - 2x - 2y \\
3x + 2y & \leq 11 \\
z & \leq 4 + x + y \\
6x - y - 9 & \leq z \\
\end{cases}
\]

3 groups: no \(z\); lower bounds \(\leq\) upper bounds on \(z\).

\[
\begin{cases}
3x + 2y & \leq 11 \\
6x - y - 9 & \leq z \\
\end{cases}
\]

Eliminate \(z\): no \(z\) remain; lower bounds \(\leq\) upper bounds

\[
F \iff \begin{cases}
3x + 2y & \leq 11 \\
6x - y - 9 & \leq 10 - 2x - 2y \\
6x - y - 9 & \leq 4 + x + y \\
\end{cases}
\]
Terminating the proof - recall what we are doing

We know that

**Theorem**

Every FOLC formula $\iff$ a DNF of linear constraints (a union of polyhedra).

We want prove that

**Fact**

$\text{DNF of linear constraints (union of polyhedra) is a good data structure for verification of LHA}$
Terminating the proof - boolean operation and tests

Let $D_1$ and $D_2$ be two DNFs (union of polyhedra).

**Union** $D_1 \cup D_2$ is already a DNF.
Terminating the proof - boolean operation and tests

Let $D_1$ and $D_2$ be two DNFs (union of polyhedra).

**Union** $D_1 \cup D_2$ is already a DNF.

**Intersection** $D_1 \cap D_2$ can be transformed to DNF using boolean logic.

**Emptiness test** To know if $D_1 = \emptyset$, eliminate quantifiers from $\exists x D_1(x)$

**Inclusion test** to know if $D_1 \subset D_2$, eliminate quantifiers from $\forall x (D_1(x) \Rightarrow D_2(x))$. 
Terminating the proof - successors

Let $D$ be a DNF (union of polyhedra)

**Flow successor** For a state with dynamics $\dot{x} = c$ and invariant $I$, flow successor of $D$ is:

$$D'(x) = \exists y \exists t (D(y) \land t > 0 \land x = y + ct \land I(x) \land I(y))$$

Eliminate quantifiers, and obtain the DNF for the successor.

**Jump successor** For a transition with guard $G$ and the reset $R$, jump successor of $D$ is:

$$D''(x) = \exists y (D(y) \land G(y) \land x = R(y))$$

Eliminate quantifiers, and obtain the DNF for the successor.

Look at the pictures!
Concluding

We are done

We have implemented (exactly) all the operations for LHA, using unions of polyhedra (DNFs of constraints) as data structure.
Concluding

We are done

We have implemented (exactly) all the operations for LHA, using unions of polyhedra (DNFs of constraints) as data structure.

Remarks

- Fourier-Motzkine is very costly.
- Our method is very costly.
- More efficient geometric method exist for some operations.
- They are still costly.
Theorems for linear hybrid automata

Theorem

For Linear Hybrid Automata

- Reach is semi-decidable (recursively enumerable)
- Reach\(_n\) is decidable
Known implementations

- Polyhedra (HyTech - exact. PHAVER - exact and approximate)
- “Gridded polyhedra” (d/dt)
- Zonotopes (Le Guernic)
- Ellipsoids (Kurzhanski, Bochkarev)
- Level sets of functions (Tomlin)
- Support functions (Le Guernic)
Does it work?

Up to 10-20 dimensions. Sometimes.
Using advanced verification techniques

- Searching for better data-structures and basic operations
- Abstraction and refinement
- Combining model-checking and theorem proving
- Acceleration
- Bounded model-checking
Part II

Timed Automata
Outline

3. **TA: an interesting subclass of HA**

4. **Decidability**

5. **Automata and language theory**

6. **Verification of TA in practice**
3. **TA: an interesting subclass of HA**

4. **Decidability**

5. **Automata and language theory**

6. **Verification of TA in practice**
Definition of TA

**Definition**
Timed automata are a subclass of hybrid automata:

- **Variables** $x_1, \ldots, x_n$, called clocks.
- **Dynamics** $\dot{x}_i = 1$, for all clocks, in all locations.
- ** Guards and invariants** Conjunctions of $x_i < c$ (or $\leq, =, \geq$) with $c \in \mathbb{N}$
- ** Resets** $x_i := 0$ for some clocks.
An example of a timed automaton

- Timed automaton (we forget to write $\dot{x} = 1$):

\[ q_1 \xrightarrow{a, x \in [1; 2]} q_2, \quad b, x := 0 \]
An example of a timed automaton

- Timed automaton (we forget to write $\dot{x} = 1$):
  
  \[
  q_1 \xrightarrow{a, x \in [1; 2]?} q_2 \xleftarrow{b, x := 0}
  \]

- Its run
  
  \[
  (q_1, 0) \xrightarrow{1.83} (q_1, 1.83) \xrightarrow{a} (q_2, 1.83) \xrightarrow{4.1} (q_2, 5.93) \xrightarrow{b} (q_1, 0) \xrightarrow{1} (q_1, 1)
  \]
An example of a timed automaton

• Timed automaton (we forget to write $\dot{x} = 1$):

$$a, x \in [1; 2] ?$$

• Its run

$$(q_1, 0) \xrightarrow{1.83} (q_1, 1.83) \xrightarrow{a} (q_2, 1.83) \xrightarrow{4.1} (q_2, 5.93) \xrightarrow{b} (q_1, 0) \xrightarrow{1} (q_1, 1)$$

• Its trace $1.83 a 4.1 b 1 a$ a timed word
An example of a timed automaton

- Timed automaton (we forget to write $\dot{x} = 1$):
  
  $a, x \in [1; 2]$?

  $b, x := 0$

- Its run
  
  $(q_1, 0) \xrightarrow{1.83} (q_1, 1.83) \xrightarrow{a} (q_2, 1.83) \xrightarrow{4.1} (q_2, 5.93) \xrightarrow{b} (q_1, 0) \xrightarrow{1} (q_1, 1)$

- Its trace 1.83 $a$ 4.1 $b$ 1 $a$ a timed word

- Its timed language: set of all the traces starting in $q_1$, ending in $q_2$:

  $\{ t_1 a s_1 b t_2 a s_2 b \ldots t_n a \mid \forall i. t_i \in [1; 2] \}$
An example of a timed automaton

- Timed automaton (we forget to write $\dot{x} = 1$):
  \[
  a, x \in [1; 2]?
  \]
  \[
  b, x := 0
  \]
  \[
  q_1 \quad q_2
  \]

- Its run
  \[
  (q_1, 0) \xrightarrow{1.83} (q_1, 1.83) \xrightarrow{a} (q_2, 1.83) \xrightarrow{4.1} (q_2, 5.93) \xrightarrow{b} (q_1, 0) \xrightarrow{1} (q_1, 1)
  \]

- Its trace $1.83 a 4.1 b 1 a a \text{ timed word}$

- Its timed language: set of all the traces starting in $q_1$, ending in $q_2$:
  \[
  \{ t_1 a s_1 b t_2 a s_2 b \ldots t_n a \mid \forall i. t_i \in [1; 2] \}
  \]

Observation

Clock value of $x$: time since the last reset of $x$. 
Some simple exercises

Draw timed automata for specifications:

- Request $a$ arrives every 5 minutes.
Some simple exercises

Draw timed automata for specifications:

- Request a arrives every 5 minutes.
- Request a arrives every 5 to 7 minutes.
Some simple exercises

Draw timed automata for specifications:

- Request $a$ arrives every 5 minutes.
- Request $a$ arrives every 5 to 7 minutes.
- $a$ arrives every 5 to 7 minutes; and $b$ arrives every 3 to 10 minutes.
Some simple exercises

Draw timed automata for specifications:

- Request $a$ arrives every 5 minutes.
- Request $a$ arrives every 5 to 7 minutes.
- $a$ arrives every 5 to 7 minutes; and $b$ arrives every 3 to 10 minutes.
- Request $a$ is serviced within 2 minutes by $c$ or rejected within 1 minute by $r$. 
Some simple exercises

Draw timed automata for specifications:

- Request $a$ arrives every 5 minutes.
- Request $a$ arrives every 5 to 7 minutes.
- $a$ arrives every 5 to 7 minutes; and $b$ arrives every 3 to 10 minutes.
- Request $a$ is serviced within 2 minutes by $c$ or rejected within 1 minute by $r$.
- The same, but $a$ arrives every 5 to 7 minutes.
Meditation on TA

Compared to HA

Very restricted: only time progress remains from all physics.
Meditation on TA

Compared to HA
Very restricted: only time progress remains from all physics.

Compared to finite automata
Time and events together. Interesting . . . . .
Meditation on TA

Compared to HA
Very restricted: only time progress remains from all physics.

Compared to finite automata
Time and events together. Interesting . . . .

As modeling formalism
For timed protocols, scheduling, timed aspects of embedded/real-time software (non-functional). See scheduling exercise.
Meditation on TA

Compared to HA
Very restricted: only time progress remains from all physics.

Compared to finite automata
Time and events together. Interesting . . . .

As modeling formalism
For timed protocols, scheduling, timed aspects of embedded/real-time software (non-functional). See scheduling exercise.

As specification formalism
For timed non-functional specifications. See exercises just above.
Meditation on TA

Compared to HA
Very restricted: only time progress remains from all physics.

Compared to finite automata
Time and events together. Interesting . . . . .

As modeling formalism
For timed protocols, scheduling, timed aspects of embedded/real-time software (non-functional). See scheduling exercise.

As specification formalism
For timed non-functional specifications. See exercises just above.
Hybrid and Timed Systems

Eugene Asarin

Outline

3. TA: an interesting subclass of HA

4. Decidability

5. Automata and language theory

6. Verification of TA in practice
Main theorem

Theorem (Alur, Dill)

*Reachability is decidable for timed automata.*
Main theorem

Theorem (Alur, Dill)

*Reachability is decidable for timed automata.*

Classical formulation

Empty language problem is decidable for TA
Main theorem

Theorem (Alur, Dill)

Reachability is decidable for timed automata.

Classical formulation

Empty language problem is decidable for TA

Both are the same

Non-empty language $\Leftrightarrow$ Reach(Init,Fin)
Proof idea

- Split the state space $Q \times \mathbb{R}^n$ into regions s.t.
  - all the states in one region have the same behavior;
  - there are finitely many regions;
Proof idea

- Split the state space $Q \times \mathbb{R}^n$ into regions s.t.
  - all the states in one region have the same behavior;
  - there are finitely many regions;
- Build a region automaton (its states are regions)
Proof idea

- Split the state space $Q \times \mathbb{R}^n$ into regions s.t.
  - all the states in one region have the same behavior;
  - there are finitely many regions;
- Build a finite region automaton (its states are regions)
Proof idea

• Split the state space $Q \times \mathbb{R}^n$ into regions s.t.
  • all the states in one region have the same behavior;
  • there are finitely many regions;
• Build a finite region automaton (its states are regions)
• Test reachability in this region automaton.
Proof idea

- Split the state space $Q \times \mathbb{R}^n$ into regions s.t.
  - all the states in one region have the same behavior;
  - there are finitely many regions;
- Build a finite region automaton (its states are regions)
- Test reachability in this region automaton.

Two difficulties

- What does it mean: the same behavior?
- How to invent it?
Proof idea

- Split the state space $Q \times \mathbb{R}^n$ into regions s.t.
  - all the states in one region have the same behavior;
  - there are finitely many regions;
- Build a finite region automaton (its states are regions)
- Test reachability in this region automaton.

Two difficulties

- What does it mean: the same behavior? Bisimulation.
- How to invent it? A&D invented it using ideas of Berthomieu (Time Petri nets). In fact it is rather natural.
Region equivalence

Definition

Two states of a TA are region equivalent: \((q, x) \approx (p, y)\) if

- Same location: \(p = q\)
- Same integer parts of clocks: \(\forall i (\lfloor x_i \rfloor = \lfloor y_i \rfloor)\)
- Same order of fractional parts of clocks
  \(\forall i, j (\{x_i\} < \{x_j\} \iff \{y_i\} < \{y_j\})\)

Look at the picture!
Region equivalence

Definition
Two states of a TA are region equivalent: \((q, x) \simeq (p, y)\) if

- Same location: \(p = q\)
- Same integer parts of clocks: \(\forall i (\lfloor x_i \rfloor = \lfloor y_i \rfloor)\)
- Same order of fractional parts of clocks
  \(\forall i, j (\{x_i\} < \{x_j\} \iff \{y_i\} < \{y_j\})\)

Look at the picture!

An issue

- Infinitely many equivalence classes.
Region equivalence

Definition
Two states of a TA are region equivalent: \((q, x) \approx (p, y)\) if

- Same location: \(p = q\)
- Same integer parts of small clocks: \(\forall\text{small}i (\lfloor x_i \rfloor = \lfloor y_i \rfloor)\)
- Same order of fractional parts small of clocks
  \(\forall\text{small}i, j (\{x_i\} < \{x_j\} \iff \{y_i\} < \{y_j\})\)
- Or they are both big: \(\forall i ((x_i > M) \iff (y_i > M))\)

Look at the picture!

An issue, and a solution

- finitely many equivalence classes.
- Solution: when a variable is BIG, we don’t care about it.
Region equivalence

Definition

Two states of a TA are region equivalent: $(q, x) \approx (p, y)$ if

- Same location: $p = q$
- Same integer parts of small clocks: $\forall \text{small } i (\lfloor x_i \rfloor = \lfloor y_i \rfloor)$
- Same order of fractional parts small of clocks
  $\forall \text{small } i, j (\{x_i\} < \{x_j\} \iff \{y_i\} < \{y_j\})$
- Or they are both big: $\forall i ((x_i > M) \iff (y_i > M))$

Look at the picture!

An issue

- finitely many equivalence classes.
- Solution: when a variable is BIG, we don’t care about it.

Definition

Equivalence classes of $\approx$ are called regions.
Region equivalence is a bisimulation

very informal
Equivalent states can make the same transitions, and arrive to equivalent states.
Region equivalence is a bisimulation

very informal
Equivalent states can make the same transitions, and arrive to equivalent states.
Let us formalize it:

Lemma
Suppose $(q, x) \approx (p, y)$. Then

- **Jump** If $(q, x) \xrightarrow{a} (q', x')$ then $(p, y) \xrightarrow{a} (p', y')$ with $(q', x') \approx (p', y')$.

- **Time** If $(q, x) \xrightarrow{t} (q', x')$ then $(p, y) \xrightarrow{\hat{t}} (p', y')$ with $(q', x') \approx (p', y')$ (the time can be different!).
Iterating the previous lemma we get

**Lemma**

Suppose \((q, x) \approx (p, y)\), and \(q \xrightarrow{w} (q', x')\) (with some timed word \(w\)), then \((p, y) \xrightarrow{\hat{w}} (p', y')\) with \((q', x') \approx (p', y')\) (the timing in \(\hat{w}\) can be different from \(w\)).
Iterating the previous lemma we get

**Lemma**

Suppose \((q, x) \approx (p, y)\), and \(q \xrightarrow{w} (q', x')\) (with some timed word \(w\)), then \((p, y) \xrightarrow{\hat{w}} (p', y')\) with \((q', x') \approx (p', y')\) (the timing in \(\hat{w}\) can be different from \(w\)).

**Corollary**

The same set of regions is reachable from elements of one region.
Decision algorithm

- Build a region automaton RA
  - States are regions.
  - There is a transition $r_1 \xrightarrow{a} r_2$ if some (all) element of $r_1$ can go to some element of $r_2$ on $a$.
  - There is a transition $r_1 \xrightarrow{τ} r_2$ if some (all) element of $r_1$ can go to some element of $r_2$ on some $t > 0$
Decision algorithm

- Build a region automaton RA
  - States are regions.
  - There is a transition $r_1 \xrightarrow{a} r_2$ if some (all) element of $r_1$ can go to some element of $r_2$ on $a$.
  - There is a transition $r_1 \xrightarrow{\tau} r_2$ if some (all) element of $r_1$ can go to some element of $r_2$ on some $t > 0$.
- Check whether some final region in RA is reachable from initial region.
Outline

3 TA: an interesting subclass of HA

4 Decidability

5 Automata and language theory

6 Verification of TA in practice
Closure property

Definition
Timed regular language is a language accepted by a TA
Closure property

**Definition**
Timed regular language is a language accepted by a TA

**Theorem**
*Timed regular languages are closed under $\cap, \cup$, projection, but not complementation.*
Closure property

Definition
Timed regular language is a language accepted by a TA

Theorem
Timed regular languages are closed under $\cap$, $\cup$, projection, but not complementation.

Fact
Determinization impossible for timed automata.
Decidability properties

**Definition**

*Timed regular language* (TRL) is a language accepted by a TA.
Decidability properties

Definition
Timed regular language (TRL) is a language accepted by a TA

Theorem
Decidable for TRL (represented by TA): $L = \emptyset$, $w \in L$, $L \cap M = \emptyset$. 
Decidability properties

**Definition**

Timed regular language (TRL) is a language accepted by a TA.

**Theorem**

Decidable for TRL (represented by TA): $L = \emptyset$, $w \in L$, $L \cap M = \emptyset$.

**Proof.**

Immediate from Alur&Dill’s theorem.
Decidability properties

Definition

Timed regular language (TRL) is a language accepted by a TA

Theorem

Decidable for TRL (represented by TA): $L = \emptyset$, $w \in L$, $L \cap M = \emptyset$.

Theorem

Undecidable for TRL (represented by TA): $L$ universal (contains all the timed words), $L \subset M$, $L = M$. 
Decidability properties

Definition

Timed regular language (TRL) is a language accepted by a TA.

Theorem

Decidable for TRL (represented by TA): \( L = \emptyset, w \in L, L \cap M = \emptyset \).

Theorem

Undecidable for TRL (represented by TA): L universal (contains all the timed words), \( L \subset M, L = M \).

Proof.

Encoding of runs of Minsky Machine as a timed languages. \( \square \)
Reminder: regular expressions

Definition
Regular expressions: $E ::= 0 \mid \varepsilon \mid a \mid E + E \mid E \cdot E \mid E^*$

Theorem (Kleene)

*Finite automata and regular expression define the same class of languages.*
Reminder: regular expressions

Definition
Regular expressions: \( E ::= 0 | \varepsilon | a | E + E | E \cdot E | E^* \)

Theorem (Kleene)

Finite automata and regular expression define the same class of languages.

Example

\[ (((a + b)a)^*(a + b)b \]
Timed regular expressions

A natural question
How to define regular expressions for timed languages?
Timed regular expressions

A natural question
How to define regular expressions for timed languages?

\[ E ::= 0 \mid \varepsilon \mid t \mid a \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle \mid E \land E \mid [a \mapsto z]E \]
Timed regular expressions

A natural question
How to define regular expressions for timed languages?

\[ E ::= 0 \mid \varepsilon \mid t \mid a \mid E + E \mid E \cdot E \mid E^{*} \mid \langle E \rangle_{I} \mid E \land E \mid [a \mapsto z]E \]

Semantics:

\[ \|t\| = \mathbb{R}_{\geq 0} \quad \|a\| = \{a\} \]
\[ \|E_{1} \cdot E_{2}\| = \|E_{1}\| \cdot \|E_{2}\| \]
\[ \|\langle E \rangle\|_{I} = \{\sigma \in \|E\| \mid \ell(\sigma) \in I\} \]
\[ \|E_{1} \land E_{2}\| = \|E_{1}\| \cap \|E_{2}\| \]
\[ \|E^{*}\| = \|E\|^{*} \]
\[ \|[a \mapsto z]E\| = [a \mapsto z]\|E\| \]
A good example and a theorem

\[ L = \{ t_1 \ a \ s_1 \ b \ t_2 \ a \ s_2 \ b \ldots \ t_n \ a \mid \forall i. t_i \in [1; 2] \} \]
A good example and a theorem

\[
\begin{align*}
q_1 & \rightarrow a, x \in [1; 2]?
\end{align*}
\]

\[
\begin{align*}
q_2 & \rightarrow b, x := 0
\end{align*}
\]

\[
L = \{ t_1 a s_1 b t_2 a s_2 b \ldots t_n a | \forall i. t_i \in [1; 2] \}
\]

An expression for \( L \):

\[
(\langle ta \rangle_{[1;2]}tb)^*\]

**Theorem (A., Caspi, Maler)**

*Timed Automata and Timed regular expressions (with \( \land \) and \( [a \mapsto z] \)) define the same class of timed languages*
A nasty example

Intersection needed [ACM]

\[
\{ t_1 a t_2 b t_3 c \mid t_1 + t_2 = 1, t_2 + t_3 = 1 \} = t a \langle t b t c \rangle_1 \land \langle t a t b \rangle_1 t c
\]
Another nasty example

Renaming needed [Herrmann]

\[
[b \mapsto a](\langle ta \rangle^* \langle tb(ta)^* \rangle_1 \land \langle ta \rangle^* \langle tb \rangle_1(ta)^*).
\]
Outline

3 TA: an interesting subclass of HA

4 Decidability

5 Automata and language theory

6 Verification of TA in practice
Reminder: decidability for TA

- **We can decide**: Reach, $L \neq \emptyset$, $L \cap M = \emptyset$, $w \in L$
- **Undecidable**: $L =$ all the words; $L \subset M$, $L = M$
Model-checking etc.

Reminder: decidability for TA

- We can decide: Reach, $L \neq \emptyset$, $L \cap M = \emptyset$, $w \in L$
- Undecidable: $L =$ all the words; $L \subset M$, $L = M$

Verification problem

Given a system $S$ and a property $P$, verify that $S$ satisfies $P$. 
Verification approaches

For simple safety properties:

- Represent $S$ by a TA $A_S$.
- Represent $P$ as $\neg \text{Reach}(\text{Init}, \text{Bad})$.
- Apply reachability algorithm.

For all kind of properties
(even with $\omega$-behaviors)

- Represent $S$ by a TA $A_S$.
- Represent $\neg P$ by a TA $A_{\neg P}$.
- Check that $L(A_S) \cap L(A_{\neg P}) = \emptyset$
Verification approaches

For simple safety properties:

- Represent $S$ by a TA $A_S$.
- Represent $P$ as $\neg$Reach(Init, Bad).
- Apply reachability algorithm.

For all kind of properties
(even with $\omega$-behaviors)

- Represent $S$ by a TA $A_S$.
- Represent $\neg P$ by a TA $A_{\neg P}$.
- Check that $L(A_S) \cap L(A_{\neg P}) = \emptyset$

Or express $P$ in a temporal logic and use some model-checking.
A simple verification example

Exercise
How to verify this?

System  A bus passes every 7 to 9 minutes. A taxi passes every 6 to 8 minutes. At noon a bus and a taxi passed.

Property  Between 12:05 and 12:30, within 5 minutes after every bus, a taxi passes.
Reachability in practice: no regions

Fact

Real verification tools, e.g. UPPAAL, do not use the region automaton. They apply a variant of the algorithm we know.
Reachability in practice: no regions

Fact

*Real verification tools, e.g. UPPAAL, do not use the region automaton. They apply a variant of the algorithm we know.*

Algorithm B

F=Init
repeat
\[ F = F \cup \text{SuccFlow}(F) \cup \text{SuccJump}(F) \]
Widen(F)
until \((F \cap \text{Final} \neq \emptyset)\) | fixpoint
say "reachable" | "unreachable"
Zones and DBMs

What is needed to implement Algorithm B
Data structure and basic algorithms for subsets of $Q \times \mathbb{R}^n$
Zones and DBMs

What is needed to implement Algorithm B
Data structure and basic algorithms for subsets of $Q \times \mathbb{R}^n$

Definition
Let $x_0 = 0$; let $x_1, \ldots, x_n$ - clocks.

- Zone: polyhedron defined by a conjunction of constraints $x_i - x_j \leq d_{ij}$ (or $<$) with $d_{ij} \in \mathbb{N}$.
- Difference bound matrix (DBM) for a zone: $D = (d_{ij})$.

Fact
A zone is a union of regions.
Zones and verification of TA

Fact

*Using DBMs, the following tests and operations on zones are easy ($O(n) - O(n^3)$):*

- $Z_1 = Z_2$?; $Z = \emptyset$?; $Z_1 \cap Z_2$.
- $\text{SuccFlow}(Z)$ and $\text{Succ}_{\delta}(Z)$ - both are zones.
Zones and verification of TA

Fact

Using DBMs, the following tests and operations on zones are easy ($O(n) - O(n^3)$):

- $Z_1 = Z_2$; $Z = \emptyset$; $Z_1 \cap Z_2$.
- $\text{SuccFlow}(Z)$ and $\text{Succ}_\delta(Z)$ - both are zones.

See Cormen, graph algorithms.
Zones and verification of TA

Fact

Using DBMs, the following tests and operations on zones are easy (\(O(n) - O(n^3)\)):

- \(Z_1 = Z_2\); \(Z = \emptyset\); \(Z_1 \cap Z_2\).
- \(\text{SuccFlow}(Z)\) and \(\text{Succ}_\delta(Z)\) - both are zones.

Corollary

Unions of zones, represented \((q_1, D_1), \ldots (q_n, D_n)\), are suitable to implement Algorithm B
Algorithm B

\[ F = \text{Init} \]
\[ \text{repeat} \]
\[ F = F \cup \text{SuccFlow}(F) \cup \text{SuccJump}(F) \]
\[ \text{Widen}(F) \]
\[ \text{until } (F \cap \text{Final} \neq \emptyset) \]
\[ \text{fixpoint} \]
\[ \text{say } "\text{reachable}" \mid "\text{unreachable}" \]
Termination

Algorithm B

\begin{verbatim}
F=Init
repeat
  F=F ∪ SuccFlow(F) ∪ SuccJump(F)
  Widen(F)
until (F ∩ Final ≠ ∅) | fixpoint
say "reachable" | "unreachable"
\end{verbatim}

To ensure termination we must widen
In each DBM, when \( c_{ij} > M \) replace \( c_{ij} := \infty \).
Algorithm B

\[ F = \text{Init} \]
\[ \begin{array}{l}
\text{repeat} \\
\quad F = F \cup \text{SuccFlow}(F) \cup \text{SuccJump}(F) \\
\quad \text{Widen}(F)
\end{array} \]
\[ \text{until} \quad (F \cap \text{Final} \neq \emptyset) \mid \text{fixpoint} \]
\[ \text{say} \quad \text{"reachable" } \mid \text{"unreachable"} \]

To ensure termination we must widen
In each DBM, when \( c_{ij} > M \) replace \( c_{ij} := \infty \).

Theorem

Algorithm B is correct and terminates (and used in practice)
Part III

Back to Hybrid automata: decidability
Outline

7 Decision by reduction to TA

8 Decision using finite bisimulations

9 Decision using planar topology
7 Decision by reduction to TA

8 Decision using finite bisimulations

9 Decision using planar topology
Reduction to TA: simple cases

Fact

Reachability is decidable for the following subclasses of HA, it is reduced to TA reachability.

• Like TA, rational constants.
Reduction to TA: simple cases

Fact

Reachability is decidable for the following subclasses of HA, it is reduced to TA reachability.

- Like TA, rational constants.

Reduction: Multiply all the guards by the common denominator $K$, you obtain a timed automaton with the same reachability (location to location).
Reduction to TA: simple cases

Fact
Reachability is decidable for the following subclasses of HA, it is reduced to TA reachability.

- Like TA, rational constants.
- Like TA, but the rate of each clock = arbitrary rational: \( \dot{x}_i = r_i \) (the same everywhere).
Reduction to TA: simple cases

Fact

Reachability is decidable for the following subclasses of HA, it is reduced to TA reachability.

- Like TA, rational constants.
- Like TA, but the rate of each clock = arbitrary rational: \( \dot{x}_i = r_i \) (the same everywhere).

Reduction: Change of variables \( \bar{x}_i = x_i / r_i \) (and corresponding change guards) transform the system into a TA with the same reachability.
Reduction to TA: simple cases

Fact

Reachability is decidable for the following subclasses of $HA$, it is reduced to $TA$ reachability.

- **Like $TA$, rational constants.**
- **Like $TA$, but the rate of each clock $=$ arbitrary rational**: $\dot{x}_i = r_i$ (the same everywhere).
- **Initialized skewed-clock automata** Like $TA$, but in a state $q$ we have that $\dot{x}_i = r_{iq}$ (may depend on the state). 

**Restriction:** when we change rate, we forget the value.

Formally, for any transition $p \rightarrow q$, either $r_{ip} = r_{iq}$ or $x_i$ is reset.
Reduction to TA: simple cases

Fact
Reachability is decidable for the following subclasses of HA, it is reduced to TA reachability.

- Like TA, rational constants.
- Like TA, but the rate of each clock = arbitrary rational: $\dot{x}_i = r_i$ (the same everywhere).
- **Initialized skewed-clock automata** Like TA, but in a state $q$ we have that $\dot{x}_i = r_{iq}$ (may depend on the state).

Restriction: when we change rate, we forget the value.

Formally, for any transition $p \rightarrow q$, either $r_{ip} = r_{iq}$ or $x_i$ is reset.

Reduction: Change of variables $\bar{x}_i = x_i / r_{iq}$ at state $q$. It works because of the restriction.
Rectangular Hybrid Automata

Let us generalize

We want to extend the previous example to the largest possible decidable class.
Rectangular Hybrid Automata

Let us generalize

We want to extend the previous example to the largest possible decidable class.

Definition

The class of Rectangular Hybrid automata is defined as follows:

- Variables \( x_1, \ldots, x_n \).
- Dynamics at each state \( q \) : inclusion \( \dot{x}_i \in [a_{iq}, b_{iq}] \) (for each \( i \))
- Invariant at each state \( q \), and guard of each transition : \( x_i \in [a., b.] \)
- Reset on each transition : either \( x_i \) is unchanged, or it is set to an arbitrary point of some interval : \( x_i \in [a., b.] \).
Rectangular Hybrid Automata

Let us generalize

We want to extend the previous example to the largest possible decidable class.

Definition

The class of Rectangular Hybrid automata is defined as follows:

- Variables $x_1, \ldots x_n$.
- Dynamics at each state $q$: inclusion $\dot{x}_i \in [a_{iq}, b_{iq}]$ (for each $i$)
- Invariant at each state $q$, and guard of each transition: $x_i \in [a., b.]$
- Reset on each transition: either $x_i$ is unchanged, or it is set to an arbitrary point of some interval: $x_i :\in [a., b.]$.

Fact

Reachability is undecidable for RHA.
Initialized Rectangular Hybrid Automata

To obtain reachability one needs a restriction:

**Definition** *(When we change rate, we forget the value)*

Initialized RHA should reset $x_i$ on each transition that changes its rate.
To obtain reachability one needs a restriction:

**Definition (When we change rate, we forget the value)**

Initialized RHA should reset $x_i$ on each transition that changes its rate.

**Theorem (Henzinger et al.)**

*Reachability is decidable for Initialized RHA.*
To obtain reachability one needs a restriction:

**Definition** *(When we change rate, we forget the value)*

Initialized RHA should reset $x_i$ on each transition that changes its rate.

**Theorem (Henzinger et al.)**

*Reachability is decidable for Initialized RHA.*

Probably the “largest” known decidable class of HA!
Outline

1. Decision by reduction to TA
2. Decision using finite bisimulations
3. Decision using planar topology
They have a complex, sometimes nonlinear dynamic, but they also forget the variable, when its equation changes.
Outline

1. Decision by reduction to TA
2. Decision using finite bisimulations
3. Decision using planar topology
Topological decision method: 2D only

Reach is decidable for

- MP94: 2d PCD + Key idea
- CV96: 2d multi-polynomial systems.
- ASY01: 2d “non-deterministic PCD” (wait a minute)
SPDI

Simple Polygonal Differential Inclusion = the non-deterministic version of PCD =

- A partition of the plane into polygonal regions
- A constant differential inclusion for each region

\[ \dot{x} \in \angle_{a}^{b} \text{ if } x \in R_{i} \]
Simple Polygonal Differential Inclusion =

\[ \begin{align*}
R_1 & \quad R_2 \\
R_3 & \quad R_4 \\
R_5 & \quad R_6 \\
R_7 & \quad R_8 \\
e_1 & \\
e_2 & \\
e_3 & \\
e_4 & \\
e_5 & \\
e_6 & \\
e_7 & \\
e_8 &
\end{align*} \]
Difficulties

Too many trajectories (even locally)

Diagram showing various trajectories between different states. States are labeled as $e_1$ to $e_8$. Arrows indicate transitions between these states.
Difficulties

Too many signatures

Decision by reduction to TA
Decision using finite bisimulations
Decision using planar topology
Difficulties

Self-crossing trajectories

Diagram showing self-crossing trajectories with labeled edges and vertices.
Plan of solution

- Simplify trajectories
- Enumerate types of signatures
- Test reachability for each type using accelerations
Simplification 1: Straightening

\[ R_i \]

\[ \mathbf{a} \]

\[ \mathbf{b} \]

\[ \mathbf{x} \]

\[ \mathbf{x}' \]
Simplification 2: Removing self-crossings
Simplification 2: Removing self-crossings

Bottom line: \( \text{Reach}(x, y) \iff \exists \text{ a simple piecewise straight trajectory from } x \text{ to } y \)
Key topological remark

Fact (Jordan, Poincaré-Bendixson, applied by Maler-Pnueli)

Simple curves on the plane are very simple.
Signatures of simplified trajectories

- **Lemma (Representation Theorem:)**

  Any edge signature can be represented as

  \[ \sigma = r_1(s_1)^{k_1} r_2(s_2)^{k_2} \cdots r_n(s_n)^{k_n} r_{n+1} \]

- **Properties**
  - \( r_i \) is a seq. of pairwise different edges;
  - \( s_i \) is a simple cycle;
  - \( r_i \) and \( r_j \) are disjoint
  - \( s_i \) and \( s_j \) are different

Proof based on Jordan’s theorem (MP94)
Classification of signatures

Any edge signature belongs to a type

\[ r_1(s_1)^* r_2(s_2)^* \ldots r_n(s_n)^* r_{n+1} \]

There are finitely many types!
How to explore one type?

**Recipe:** compute successors and accelerate cycles.
Successors (by $\sigma$)

One step ($\sigma = e_1 e_2$)

Decision by reduction to TA
Decision using finite bisimulations
Decision using planar topology
Successors (by $\sigma$)

Several steps ($\sigma = e_1 e_2 e_3$)
Successors (by $\sigma$)

Several steps ($\sigma = e_1 e_2 e_3 e_4 e_5$)
Successors (by $\sigma$)

One cycle ($\sigma = s = e_1 e_2 \cdots e_8 e_1$)
5 One cycle iterated: \( \approx \text{solution of fixpoint equation (acceleration)} \) \( \text{Succ}_\sigma(I) = I \)
The calculus of TAMF

- **Fact:** All successors are TAMF
- Affine function (AF):
  \[ f(x) = ax + b \text{ with } a > 0 \]
- Affine multi-valued function (AMF):
  \[ \tilde{F}(x) = [f_1(x), f_2(x)] \]
- Truncated affine multi-valued function (TAMF):
  \[ F(x) = \tilde{F}(x) \cap J \text{ if } x \in S \]

**Lemma**

AF, AMF and TAMF are closed under composition.

**Lemma**

Fixpoint equations \( F(I) = I \) can be explicitly solved (without iterating)
Reachability Algorithm

for each type of signature $\tau$ do
   test whether $x \xrightarrow{\tau} y$

To test $x \xrightarrow{\tau} y$ for $\tau = r_1(s_1)^*r_2(s_2)^* \ldots r_n(s_n)^*r_{n+1}$ compute $\text{Succ}_r$ and accelerate $(\text{Succ}_s)^*$
Main result for SPDI

Theorem (A., Schneider, Yovine)

*Reachability is decidable for SPDI.*
Hybrid and Timed Systems

Eugene Asarin

Decision by reduction to TA

Decision using finite bisimulations

Decision using planar topology

SPeeDI the tool
Part IV

Conclusions and perspectives
Outline
Hybrid: conclusions for a pragmatical user

- A useful and proper model of cyber-physical systems: HA. Modeling languages available.
- Simulation possible with old and new tools
- No hope for exact analysis
- In simple cases approximated analysis (and synthesis) with guarantee is possible using verification paradigm. Tools available
- (Not discussed) Some control-theoretical techniques available (stability, optimal control etc).
Hybrid: perspectives for a researcher

- Obtain new decidability results (nobody cares for undecidability).
- Find better data structures and algorithms for approximated verification.
- Apply modern model-checking techniques
- Create hybrid theory of formal languages
- etc.
Timed: Conclusions for a pragmatical user

- A useful and proper model of computer systems immersed in physical time: TA.
- Modeling and specification languages available.
- Efficient simulation, verification and synthesis tools available.
Timed: perspectives for a researcher

- Develop a theory of timed languages. Algebra, logic, topology etc. (see my text http://hal.archives-ouvertes.fr/hal-00157685)
- Improve verification techniques.
- Study rich and decidable specification formalisms (logical, algebraic, etc.) for timed languages.
- etc.