Hybrid and Timed Systems

modeling, theory, verification

Eugene Asarin

Université de Paris Cité [redacted] and CNRS

Prepared for Shanghai Computer Science Summer School 2010

Revised for MPRI course 2-8-2
2022-23
(barred slides are not in my programme this year)

Introductory equations

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

Cyber-Physical!

Introductory equations

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 + [redacted] computer scientists

Cyber-Physical

Timed Systems

and a nice automata/ languages theory

- **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 (see Laurent Fribourg's lectures)
2. Timed Automata
3. ~~Back to Hybrid: Decidable Subclasses~~

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

Part I

Hybrid Automata

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 $\dot{x} = \dot{x}(t) = x'(t) = dx/dt$.

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

The first (cyber-physical) example

Notation

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

[redacted]

- 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:

$$\dot{x} = H - x$$

X

- When [redacted] $> M$ it switches OFF
- When [redacted] $< m$ it switches ON

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

A strange creature...

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

A bad syntax

Some mathematicians prefer to write

$$\dot{x} = f(x, q)$$

where

$$\begin{aligned} f(x, \mathrm{Off}) &= -x \\ f(x, \mathrm{On}) &= H - x \end{aligned}$$

with some switching rules on q.

A bad syntax

Some mathematicians prefer to write

$$\dot{x} = f(x, q)$$

where

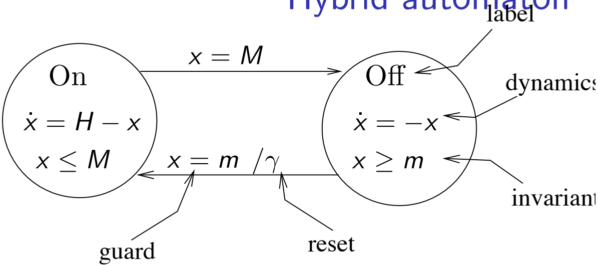
$$\begin{aligned} f(x, \mathrm{Off}) &= -x \\ f(x, \mathrm{On}) &= H - x \end{aligned}$$

with some switching rules on q.

But we are computer scientists

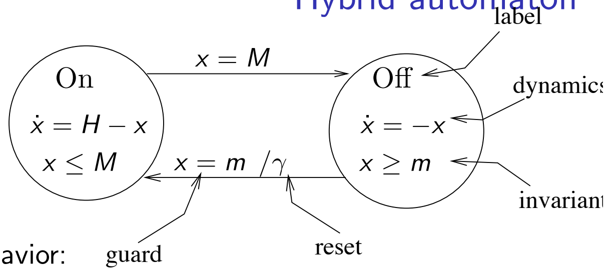
and draw an *automaton*

Hybrid automaton

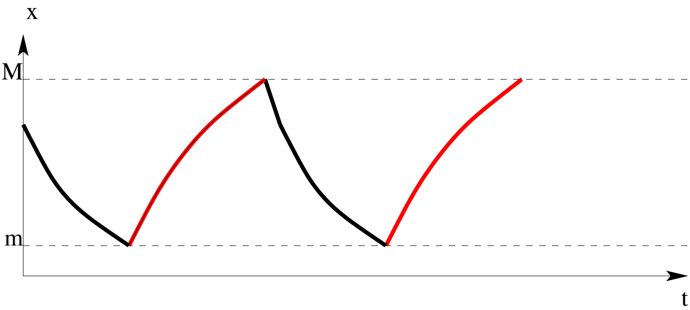


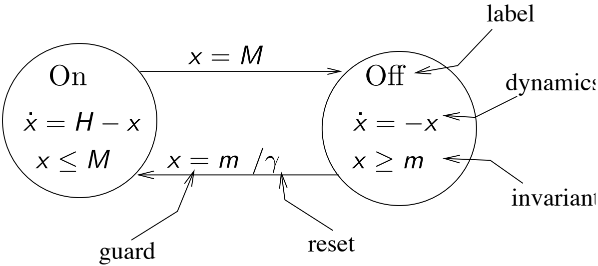
A formal definition: It is a tuple . . .

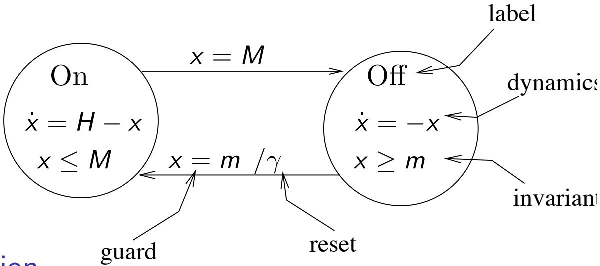
Hybrid automaton



Its behavior:



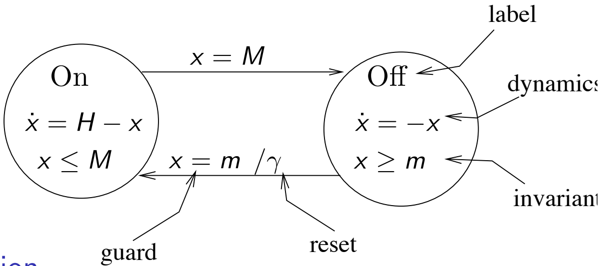

label
dynamics
invariant
guard
reset
On
$\dot{x} = H - x$
$x \leq M$
Off
$\dot{x} = -x$
$x \geq m$
$x = M$
$x = m \ /\gamma$



Definition

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

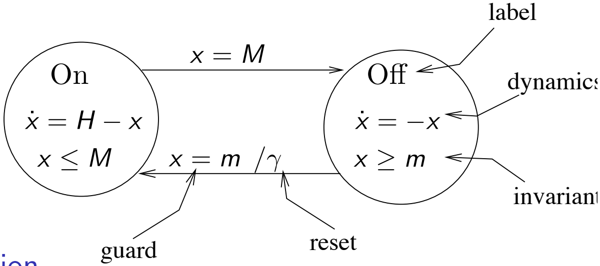
- Q finite set of locations
- $X = \mathbb{R}^n$, continuous state space
- Dyn, dynamics on X for every $q \in Q$
- I, invariant, staying condition in X
- Δ, finite set of transitions $\delta = (p, q, a, g, r)$



Definition

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

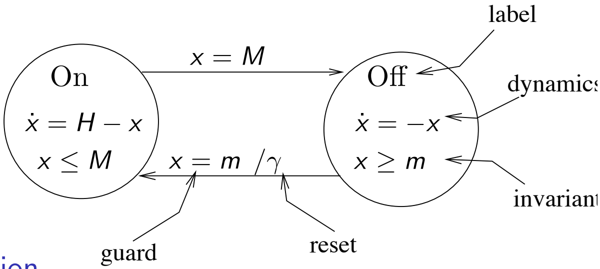
- Q finite set of locations
- $X = \mathbb{R}^n$, continuous state space , a point in X = valuation of continuous variables $\mathbf{x} = x_1, \dots, x_n$
- Dyn, dynamics on X for every $q \in Q$
- I, invariant, staying condition in X
- Δ, finite set of transitions $\delta = (p, q, a, g, r)$



Definition

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

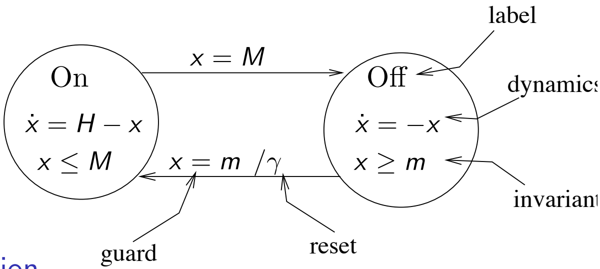
- Q finite set of locations
- $X = \mathbb{R}^n$, continuous state space
- *Dyn*, dynamics on X for every $q \in Q$, $Dyn(q) = f_q$, whenever in location q the continuous state obeys $\dot{\mathbf{x}} = f_q(\mathbf{x})$.
- I, invariant, staying condition in X
- Δ, finite set of transitions $\delta =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



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, whenever in location q the continuous state obeys $\mathbf{x} \in I(q)$.
- Δ, finite set of transitions $\delta = (p, q, a, g, r)$

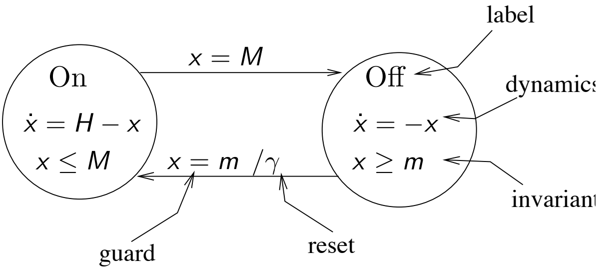


Definition

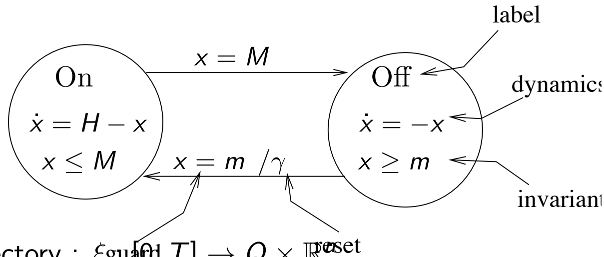
A hybrid automaton is $H = (Q, X, \Sigma, Dyn, I, \Delta)$ with

- 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
- Δ, 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; $g(\mathbf{x})$ required to take δ
  - r a reset (or jump); $\mathbf{x} := r(\mathbf{x})$ when taking δ

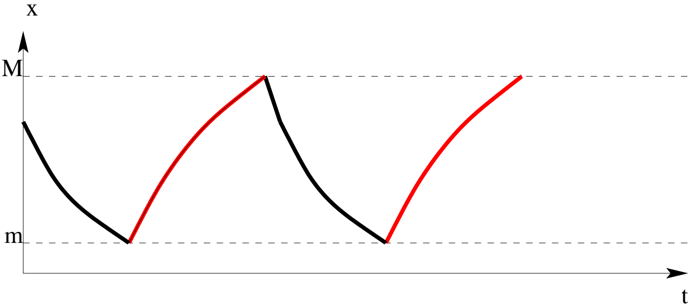
Trajectory-based semantics



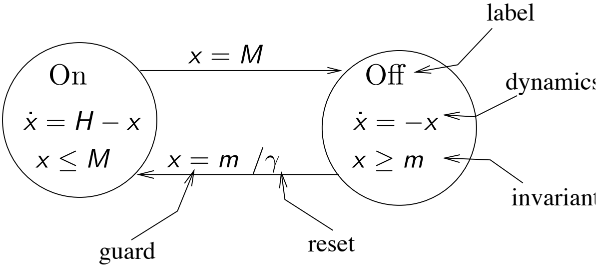
Trajectory-based semantics



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



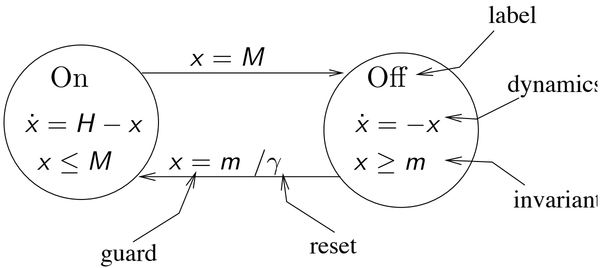
Transition system semantics



Transition system (S, T) of a HA

- **States:** $S = Q \times \mathbb{R}^n$
- **Transitions:** $T = T_{\text{flow}} \cup T_{\text{jump}}$

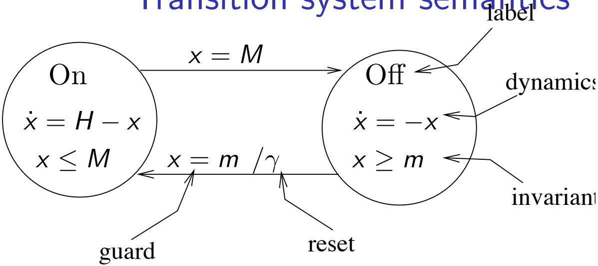
Transition system semantics



Transition system (S, T) of a HA

- **States:** $S = Q \times \mathbb{R}^n$
- **Transitions:** $T = T_{\text{flow}} \cup T_{\text{jump}}$

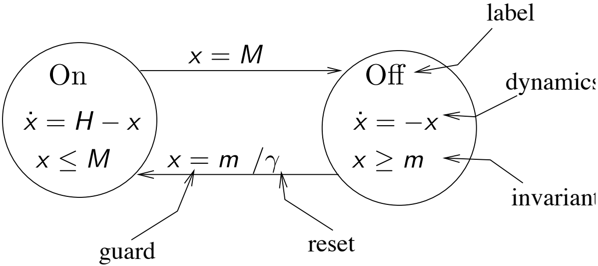
Transition system semantics



Transition system (S, T) of a HA

- **States:** $S = Q \times \mathbb{R}^n$
- **Transitions:** $T = T_{\text{flow}} \cup T_{\text{jump}}$
    - $(q, \mathbf{x}_1) \overset{\text{flow}}{\to} (q, \mathbf{x}_2) \Leftrightarrow$ we can go from $\mathbf{x}_1$ to $\mathbf{x}_2$ in ODE $\dot{\mathbf{x}} = f_q(\mathbf{x})$
    - $(q_1, \mathbf{x}_1) \overset{\text{jump}}{\to} (q_2, \mathbf{x}_2) \Leftrightarrow$ if we can jump.

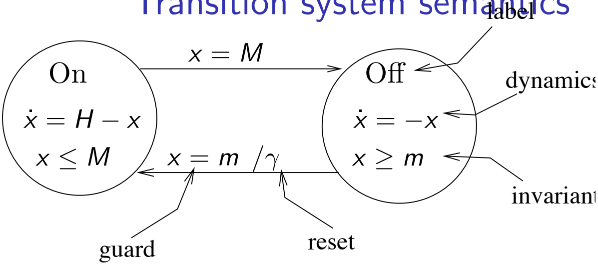
Transition system semantics



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 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.

$$(\mathrm{On}, 0) \xrightarrow{\text{flow}} (\mathrm{On}, M) \xrightarrow{\text{jump}} (\mathrm{Off}, M) \xrightarrow{\text{flow}} (\mathrm{Off}, m) \xrightarrow{\text{jump}} (\mathrm{On}, m) \cdots$$

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

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/invarians/jumps

We will consider TIMED AUTOMATA

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

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

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 $d - b \cdot$(concentration de 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.

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

Verification and reachability problems

- Is automatic verification possible for HA?

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

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

$$\neg \mathrm{Reach}(\mathit{Init}, \mathit{Bad})$$

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

$$\neg\mathrm{Reach}(\mathit{Init}, \mathit{Bad})$$

- It is a natural and challenging mathematical problem.
- Many works on decidability
- Some works on approximated techniques

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

The reachability problem for a class C

Problem

Given

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

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 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.

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

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:

$$\begin{array}{lll} q_1: & D++; & \text{goto } q_2 \\ q_2: & C--; & \text{goto } q_3 \\ q_3: & \text{if } C>0 & \text{then goto } q_2 \text{ else } q_1 \end{array}$$

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:

$$\begin{array}{llll} q_1: & D++; & \text{goto } q_2 \\ q_2: & C--; & \text{goto } q_3 \\ q_3: & \text{if } C>0 & \text{then goto } q_2 \text{ else } q_1 \end{array}$$

Theorem (Minsky)

Reachability is undecidable for Minsky machines.

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

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:

$$\begin{array}{llll} q_1: & D++; & \text{goto } q_2 \\ q_2: & C--; & \text{goto } q_3 \\ q_3: & \text{if } C > 0 & \text{then goto } q_2 \text{ else } q_1 \end{array}$$

Theorem (Minsky)

(All variants: (p,0,0)->(q,0,0); (p,0,0)->(q,*,*); or (p,n,0)-> (q,*,*) even for a fixed machine, etc

Reachability is undecidable for Minsky machines.

Fact

Any algorithm can be programmed on a Minsky machine. But they are slooooooow.

A typical undecidability theorem

Theorem (Koiran, Cosnard, Garzon)

Reach *is undecidable for 2d PAM.*

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

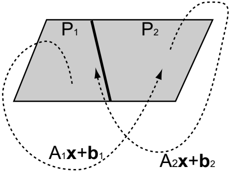
A typical undecidability theorem

Theorem (Koiran, Cosnard, Garzon)

Reach *is undecidable for 2d PAM.*

Reminder

A 2 dimensional PAM:



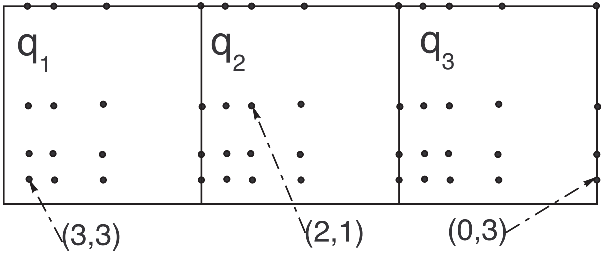
$\mathbf{x} := A_i\mathbf{x} + \mathbf{b}_i$ for $\mathbf{x} \in P_i$

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

Simulating a counter by a PAM

Counter	PAM
State space $\mathbb{N}$	State space $[0;1]$
State $C = n$	$x = 2^{-n}$
$C++$	$x := x/2$
$C--$	$x := 2x$
$C > 0?$	$x < 0.75?$

Encoding a state of a Minsky Machine



Minsky Machine	**PAM**
State space $\{q_1, \dots, q_k\} \times \mathbb{N} \times \mathbb{N}$	State space $[1; k+1] \times [0; 1]$
State $(q_i, C = m, D = n)$	$x = i + 2^{-m}, y = 2^{-n}$

Simulating a Minsky Machine

Minsky Machine	**PAM**
State space $\{q_1, \dots, q_k\} \times \mathbb{N} \times \mathbb{N}$	State space $[1; k+1] \times [0; 1]$
State $(q_i, C = m, D = n)$	$x = i + 2^{-m}, y = 2^{-n}$
$q_1 : D{+}{+};$ goto q_2	$\begin{cases} x := x+1 \\ y := y/2 \end{cases}$ if $1 < x \leq 2$
$q_2 : C{-}{-};$ goto q_3	$\begin{cases} x := 2(x-2)+3 \\ y := y \end{cases}$ if $2 < x \leq 3$
q_3 : if $C > 0$ then goto q_2 else q_1	$\begin{cases} x := x-1 \\ y := y \end{cases}$ if $3 < x < 4$ $\begin{cases} x := x-2 \\ y := y \end{cases}$ if $x = 4$

MM: (q_i,0,0)…->(q_j,*,*)

ssi PAM: (i+1,1) …->le carré j<x<=j+1

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

. . . finally we have proved:

Theorem (Koiran et al.)

Reach *is undecidable for 2d PAMs.*

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.

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

verification

Abstract algorithm - important

A generic verification algorithm A

Forward breadth-first search

F=Init
repeat
    F=F $\cup$ SuccFlow(F) $\cup$ SuccJump(F)
until (F$\cap$ 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=Init

repeat

    F=F $\cup$ SuccFlow(F) $\cup$ SuccJump(F)

until ($F \cap$ 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=Init
repeat
    $F = F \cup \mathrm{SuccFlow}(F) \cup \mathrm{SuccJump}(F)$
until $(F \cap \mathrm{Bad} \neq \emptyset)$ | 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

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.*

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".*

TA: an interesting subclass of HA

Decidability

Automata and language theory

Verification of TA in practice

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

Outline

3 TA: an interesting subclass of HA

4 Decidability

5 Automata and language theory

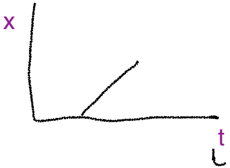
6 Verification of TA in practice

Definition of TA

Definition

Timed automata are a subclass of hybrid automata:

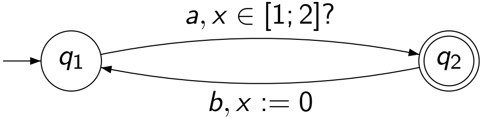
- **Variables** $x_1, \dots, x_n$, called clocks.
- **Dynamics** $\dot{x}_i = 1$, for all clocks, in all locations.
- **Guards and invariants** Conjunctions of $x_i < c$ (or $\leq, =, . \geq$))with $c \in \mathbb{N}$
- **Resets** $x_i := 0$ for some clocks.



x= temps écoulé après le dernier reset

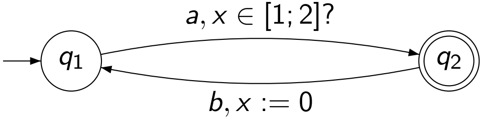
An example of a timed automaton

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



An example of a timed automaton

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

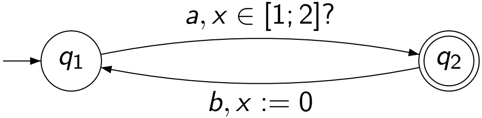


- Its run

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

An example of a timed automaton

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



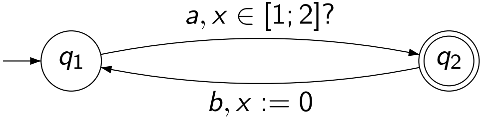
- Its run

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

- 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 $\dot{x} = 1$):



- Its run

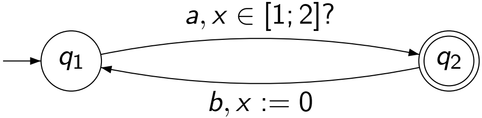
$(q_1, 0) \xrightarrow{1.83} (q_1, 1.83) \xrightarrow{a} (q_2, 1.83) \xrightarrow{4.1} (q_2, 5.93) \xrightarrow{b} (q_1, 0) \xrightarrow{1} (q_1, 1)$

- 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 $\dot{x} = 1$):



- 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:

$$\{t_1\, a\, s_1\, b\, t_2\, a\, s_2\, b \ldots t_n\, a \mid \forall i . t_i \in [1;2]\}$$

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.

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

Unimed language of a timed automaton is regular (and can be computed).

Theorem (Alur, Dill)

Reachability is decidable for timed automata.

Classical formulation

Empty language problem is decidable for TA

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.~~

  use it to recognize the untimed language

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~~.

use it to recognize the untimed language

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.~~

  use it to recognize the untimed language

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.

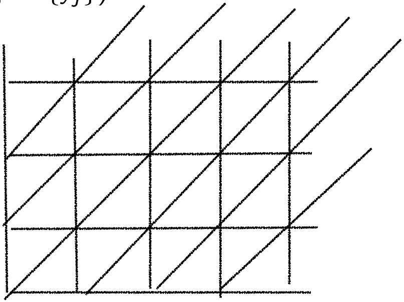
Region equivalence

Definition

Two states of a TA are region equivalent: $(q, \mathbf{x}) \approx (p, \mathbf{y})$ 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 $\forall i, j \, (\{x_i\} < \{x_j\} \Leftrightarrow \{y_i\} < \{y_j\})$

Look at the picture!



<=> x and y satisfy the same constraints of forms x_3<5 and x_1-x_2<2

Region equivalence

Definition

Two states of a TA are region equivalent: $(q, \mathbf{x}) \approx (p, \mathbf{y})$ 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 $\forall i, j \, (\{x_i\} < \{x_j\} \Leftrightarrow \{y_i\} < \{y_j\})$

Look at the picture!

An issue

- Infinitely many equivalence classes.

Region equivalence

Definition

Two states of a TA are region equivalent: $(q, \mathbf{x}) \approx (p, \mathbf{y})$ 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 $\forall \text{small} i, j \,(\{x_i\} < \{x_j\} \Leftrightarrow \{y_i\} < \{y_j\})$
- Or they are both big : $\forall i \,((x_i > M) \Leftrightarrow (y_i > M))$

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: $(q, \mathbf{x}) \approx (p, \mathbf{y})$ 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 $\forall \text{small} i, j \, (\{x_i\} < \{x_j\} \Leftrightarrow \{y_i\} < \{y_j\})$
- Or they are both big : $\forall i \, ((x_i > M) \Leftrightarrow (y_i > M))$

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 (time-abstract bisimulation)

Suppose $(q, \mathbf{x}) \approx (p, \mathbf{y})$. *Then*

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

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

Reading a timed word

Iterating the previous lemma we get

Lemma

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

The untiming is the same

Reading a timed word

Iterating the previous lemma we get

Lemma

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

The untiming is the same

Corollary

The same set of regions is reachable from elements of one region. **(using the same untiming)**

Untiming ~~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 $t > 0$

(τ should be ε)

Untiming ~~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.~~

RA recognizes the untiming of the initial language

initial states of RA: regions of (i,0) for initial i of TA

final states of RA: regions of (f,x) for final f of TA, and any x

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. $\square$

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

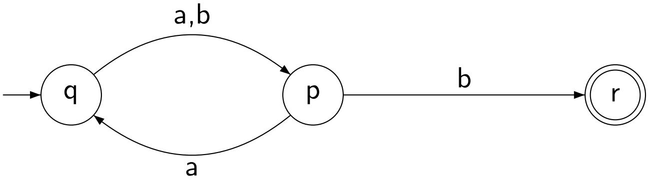
Definition

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

Theorem (Kleene)

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

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?

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}} \mid a \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle_I \mid E \wedge E \mid [a \mapsto z]E$$

Timed regular expressions

A natural question

How to define regular expressions for timed languages?

$$E ::= 0 \mid \varepsilon \mid \underline{\mathbf{t}} \mid a \mid E + E \mid E \cdot E \mid E^* \mid \langle E \rangle_I \mid E \wedge E \mid [a \mapsto z]E$$

Semantics:

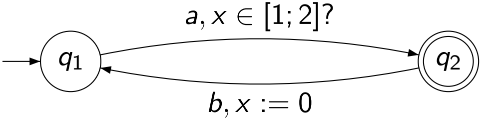
$$\|\underline{\mathbf{t}}\| = \mathbb{R}_{\geq 0} \quad \|a\| = \{a\} \qquad \|0\| = \emptyset \quad \|\varepsilon\| = \{\varepsilon\}$$

$$\|E_1 \cdot E_2\| = \|E_1\| \cdot \|E_2\| \qquad \|E_1 + E_2\| = \|E_1\| \cup \|E_2\|$$

$$\|\langle E \rangle\|_I = \{\sigma \in \|E\| \mid \ell(\sigma) \in I\} \qquad \|E^*\| = \|E\|^*$$

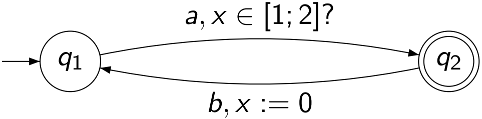
$$\|E_1 \wedge E_2\| = \|E_1\| \cap \|E_2\| \qquad \|[a \mapsto z]E\| = [a \mapsto z]\|E\|$$

A good example and a theorem



$$\{L = \{t_1\, a\, s_1\, b\, t_2\, a\, s_2\, b \dots t_n\, a \mid \forall i. t_i \in [1;2]\}$$

A good example and a theorem



$$\{L = \{t_1\, a\, s_1\, b\, t_2\, a\, s_2\, b \dots t_n\, a \mid \forall i. t_i \in [1;2]\}$$

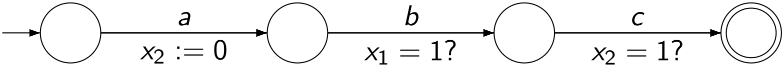
An expression for L : $\left(\langle \underline{\mathbf{t}} a\rangle_{[1;2]} \underline{\mathbf{t}} b\right)^*$

Theorem (A., Caspi, Maler)

Timed Automata and Timed regular expressions (with $\wedge$ and $[a \mapsto z]$) define the same class of timed languages

A nasty example

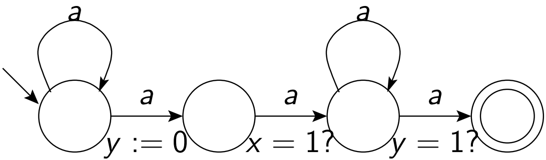
Intersection needed [ACM]



$$\{t_1 a t_2 b t_3 c \mid t_1 + t_2 = 1, t_2 + t_3 = 1\} = \underline{\mathbf{t}} a \langle \underline{\mathbf{t}} b \underline{\mathbf{t}} c \rangle_1 \wedge \langle \underline{\mathbf{t}} a \underline{\mathbf{t}} b \rangle_1 \underline{\mathbf{t}} c$$

Another nasty example

Renaming needed [Herrmann]



$$[b \mapsto a]\Big((\underline{\mathbf{t}}a)^*\langle \underline{\mathbf{t}}b(\underline{\mathbf{t}}a)^*\rangle_1 \wedge \langle (\underline{\mathbf{t}}a)^*\underline{\mathbf{t}}b\rangle_1(\underline{\mathbf{t}}a)^*\Big).$$

Outline

3 TA: an interesting subclass of HA

4 Decidability

5 Automata and language theory

6 Verification of TA in practice

Model-checking etc.

Reminder: decidability for TA

PSPACE-complete

- 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 ¬Reach(Init,Bad). Empty language
- Apply reachability algorithm.(empty language)

For all kind of properties

(even with ω-behaviors)

- Represent S by a TA A_S. language = possible behaviors
- Represent $\neg P$ by a TA $A_{\neg P}$. language=bad behaviors
- 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 ω-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 $\cup$ SuccFlow(F) $\cup$ SuccJump(F)
    Widen(F)
until (F$\cap$ 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 $<$) wirh $d_{IJ} \in \mathbb{N}$.
- *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 ($O(n) - O(n^3)$):

- $Z_1 = Z_2$?; $Z = \emptyset$?; $Z_1 \cap Z_2$.
- $SuccFlow(Z)$ *and* $Succ_\delta(Z)$ *- both are zones.*

Zones and verification of TA

Fact

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

- $Z_1 = Z_2$?; $Z = \emptyset$?; $Z_1 \cap Z_2$.
- $SuccFlow(Z)$ *and* $Succ_\delta(Z)$ *- both are zones.*

See Cormen, graph algorithms.

Zones and verification of TA

Fact

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

- $Z_1 = Z_2$?; $Z = \emptyset$?; $Z_1 \cap Z_2$.
- $SuccFlow(Z)$ *and* $Succ_\delta(Z)$ *- both are zones.*

Corollary

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

Termination

Algorithm B

F=Init
repeat
F=F $\cup$ SuccFlow(F) $\cup$ SuccJump(F)
Widen(F)
until (F$\cap$ Final $\neq \emptyset$)| fixpoint
say "reachable" | "unreachable"

Termination

Algorithm B

F=Init
repeat
F=F $\cup$ SuccFlow(F) $\cup$ SuccJump(F)
Widen(F)
until (F$\cap$ Final $\neq \emptyset$)| fixpoint
say "reachable" | "unreachable"

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

When C_ij <-M replace c_ij:=-M

Termination

Algorithm B

F=Init
repeat
F=F $\cup$ SuccFlow(F) $\cup$ SuccJump(F)
Widen(F)
until (F$\cap$ Final $\neq \emptyset$)| fixpoint
say "reachable" | "unreachable"

To ensure termination we must widen

In each DBM, when $c_{ij} > M$ replace $c_{ij} := \infty$.

When C_ij <-M replace c_ij:=-M

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

Outline

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* $p \to q$*, 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 $p \to q$*, 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 $x_1, \dots x_n$.
- 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.

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.

Theorem (Henzinger et al.)

Reachability is decidable for Initialized 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.

Theorem (Henzinger et al.)

Reachability is decidable for Initialized RHA.

Probably the "largest" known decidable class of HA!

Outline

7 Decision by reduction to TA

8 Decision using finite bisimulations

9 Decision using planar topology

o-minimal automata

They have a complex, sometimes nonlinear dynamic, but they also forget the variable, when its equation changes.

Part IV

Conclusions and perspectives

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.

Quantitative verification
Information theory
Runtime verification/monitoring
Pattern-matching
Machine learning