Course Notes for 2.33.1: Dynamic Undecidability

Olivier Bournez

Version of November 16, 2018.

Chapter 1

Preliminaries

These are Course Notes for MPRI Course 2.33.1.
Theories of Computation.

Any comment (even about orthography) welcome: send an email to bournez@lix.polytechnique.fr

Chapter 2

Some basic notions

Let's consider that we are working in $\mathbb{R}^n$ (in general, we could consider any vector space with a norm). Let us consider $f : E \to \mathbb{R}^n$, where $E \subset \mathbb{R}^n$ is open.

2.1 Ordinary Differential Equations

An Ordinary Differential Equation (ODE) is given by $y' = f(y)$ and its solution is a differentiable function $y : I \subset \mathbb{R} \to E$ that satisfies the equation.

For any $x \in E$, the fundamental existence-uniqueness theorem (see e.g. [14]) for differential equations states that if f is Lipschitz on E, i.e. if there exists K such that $||f(y_1) - f(y_2)|| < k||y_1 - y_2||$ for all $y_1, y_2 \in E$, then the solution of

$$y' = f(y), \qquad y(t_0) = x \tag{2.1}$$

exists and is unique on a certain maximal interval of existence $I \subset \mathbb{R}$. In the terminology of dynamical systems, $y(t)$ is referred to as the *trajectory*, $\mathbb{R}^n$ as the *phase space*, and the function $\phi(t, x)$, which gives the position $y(t)$ of the solution at time t with initial condition x, as the *flow*. The graph of y in $\mathbb{R}^n$ is called the *orbit*.

In particular, if f is continuously differentiable on E then the existence-uniqueness condition is fulfilled [14]. Most of the mathematical theory has been developed in this case, but can be extended to weaker conditions. In particular, if f is assumed to be only continuous, then uniqueness is lost, but existence is guaranteed: see for example [11]. If f is allowed to be discontinuous, then the definition of solution needs to be refined. This is explored by Filippov in [12]. Some hybrid system models use distinct and ad hoc notions of solutions. For example, a solution of a piecewise constant differential equation in [5] is a continuous function whose right derivative satisfies the equation.

2.2 Dynamical Systems

In general, a dynamical system can be defined as the action of a subgroup $\mathcal{T}$ of $\mathbb{R}$ on a space X, i.e. by a function (a flow) $\phi : \mathcal{T} \times X \to X$ satisfying the following two equations

$$\phi(0, x) = x \tag{2.2}$$

$$\phi(t, \phi(s, x)) = \phi(t + s, x). \tag{2.3}$$

It is well known that subgroups $\mathcal{T}$ of $\mathbb{R}$ are either dense in $\mathbb{R}$ or isomorphic to the integers. In the first case, the time is called continuous, in the latter case, discrete.

2.2.1 Continuous Time Dynamical Systems

Since flows obtained by initial value problems (IVP) of the form (2.1) satisfy equations (2.2) and (2.3), they correspond to specific continuous time and space dynamical systems. Although not all continuous time and space dynamical systems can be put in a form of a differential equation, IVPs of the form (2.1) are sufficiently general to cover a very wide class of such systems. In particular, if ϕ is continuously differentiable, then $y' = f(y)$, with $f(y) = \frac{d}{dt}\phi(t, y)\big|_{t=0}$, describes the dynamical system.

2.2.2 Discrete Time Dynamical Systems

For discrete time systems, we can assume without loss of generality that $\mathcal{T}$ is the integers. The analog of of Initial Value Problem (2.1) for discrete time systems is a recurrence equation of type

$$y_{t+1} = f(y_t), \qquad y_0 = x. \tag{2.4}$$

Chapter 3

Static vs Dynamic Undecidability

The previous results are *static undecidability* results. They don't really say things about the hardness of

- simulating dynamical systems;
- verifying dynamical systems.

3.1 A provocative point of view

As observed in [2] and in [23], it is relatively simple but not very informative to get undecidability results with continuous time dynamical systems, if f encodes a undecidable problem.

To illustrate this, we consider the following example taken from [23]. In this paper, Ruohonen discusses the event detection problem: given a differential equation $y' = f(t, y)$, with initial value $y(0)$, decide if a given condition $g_j(t, y(t), y'(t)) = 0, j = 1, \cdots, k$ happens at some time t in a given interval I.

Given the Turing machine $\mathcal{M}$, the sequence $f_0, f_1, \cdots$ of rationals defined by

$$f_n = \begin{cases} 2^{-m} & \text{if} \quad \mathcal{M} \text{ stops in } m \text{ steps on input } n \\ 0 & \text{if} \quad \mathcal{M} \text{ does not stop on input } n \end{cases}$$

is not a computable sequence of rationals, but is a computable sequence of reals.

Now, the detection of the event $y(t) = 0$ for the ordinary differential equation $y' = 0$, given n, and the initial value $y(0) = f_n$, is undecidable over any interval containing 0, because $f_n = 0$ is undecidable.

A further modification can be obtained as follows [23].

Consider the smooth function

$$g(x) = f_{\lfloor x+1/2 \rfloor} e^{-\tan^2 \pi x},$$

which is computable on $[0, \infty)$. The detection of the event $y_1(t) = 0$ for the ODE

$$\begin{cases} y_1' &= g(y_2) - 1 \\ y_2' &= 0 \end{cases}$$

given an initial value $y_1(0) = 1$, $y_2(0) = n$, where n is a nonnegative integer is then undecidable on $[0, 1]$.

As put forth in [2] undecidability results given by recursive analysis are somehow built similarly.

3.2 Dynamic undecidability

To be able to discuss in more detail computability of differential equations, we will focus on dynamical systems that encode the transitions of a Turing machine instead of the result of the whole computation simulation[1]. Typically, we start with some (simple) computable injective function which encodes any configuration of a Turing machine M as a point in $\mathbb{R}^n$. Let x be the encoding of the initial configuration of $\mathcal{M}$. Then, we look for a function $f : E \subset \mathbb{R}^{n+1} \to \mathbb{R}^n$ such that

- Discrete Time Case: the solution of $y^{t+1} = f(y, t)$ with $y(0) = x$, is such that x_t is the encoding of the configuration of $\mathcal{M}$ after t steps.
- Continuous time Case: the solution of $y'(t) = f(y, t)$, with $y(0) = x$, at time $T \in N$ is the encoding of the configuration of $\mathcal{M}$ after T steps.

We will see, in the remainder of this section, that f can be restricted to have low dimension, to be smooth or even analytic, or to be defined on a compact domain.

Instead of stating that the property above is a Turing machine simulation, we can address it as a reachability result. Given the IVP defined by f and x, and any region $A \subset \mathbb{R}^n$, we are interested in deciding if there is a $t \geq 0$ such $y(t) \in A$, i.e., if the flow starting in x crosses A. It is clear that if f simulates a Turing machine in the previous sense, then reachability for that system is undecidable (just consider A as encoding the halting configurations of $\mathcal{M}$). So, reachability is another way to address the computability of ODEs and a negative result is often a byproduct of the simulation of Turing machines. Similarly, undecidability of event detection follows from Turing simulation results.

[1] This is called dynamic undecidability in [22].

Chapter 4

Some Dynamic Undecidability Results: Using a Discrete Time

4.1 Some models

4.1.1 The PAM Model

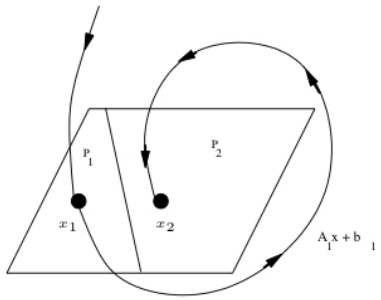
Here is a simple toy model.

Definition 1 (PAM [4]) *A Piecewise Affine Map (PAM) is a discrete time dynamical system $\mathcal{H}$, defined by $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$ on $X \subset \mathbb{R}^d$, where $f : X \to \mathbb{R}^d$ is piecewise affine: that is to say,*

$$f(\mathbf{x}) = f_i(\mathbf{x}) \text{ for } \mathbf{x} \in P_i,\ i = 1, \ldots, n$$

where f_i is some affine function with rational coefficients, and the P_i constitutes a partition of X into finitely many rational convex polyhedra.

Recall that a convex polyhedra is the convex hull of a finite number of points. A rational convex polyhedra is the convex hull of a finite number of points with rational coordinates.



4.1.2 The PCD Model

We are going to discuss the Piecewise Constant Derivative (PCD) model that has been introduced by Eugene Asarin, Oded Maler and Amir Pnueli in [5], as a simple model for hybrid systems. It has later on been discussed in several papers such as [3, 4, 7].

A hybrid system is a system that combines continuous evolutions with discrete transitions. Such models appear as soon as one tries to model some systems where a discrete system, such as a computer, evolves in a continuous environment: See e.g. [1].

From a theoretical computer science point of view, one interest of the hybrid systems models, is that they generalize both discrete time transition systems and continuous time dynamical systems.

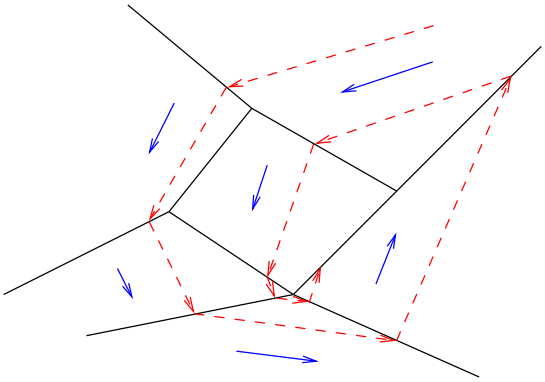
Definition 2 (PCD System [4]) *A (rational) piecewise-constant derivative (PCD) system is a continuous time dynamical system $\mathcal{H}$, defined by differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$ on $X \subset \mathbb{R}^d$, where $f : X \to \mathbb{R}^d$, can be represented by the formula*

$$f(\mathbf{x}) = \mathbf{c}_i \text{ for } \mathbf{x} \in P_i,\ i = 1, \ldots, n$$

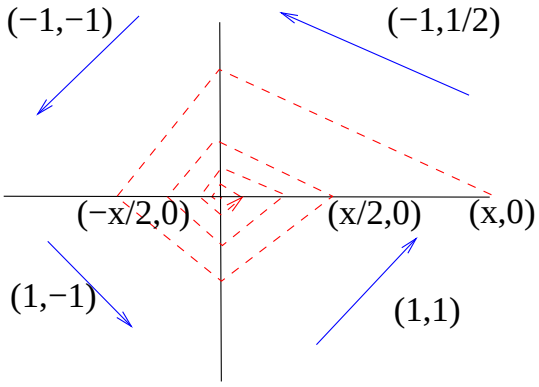
where $c_i \in \mathbb{Q}^d$, and the P_i constitutes a finitely many partition of X into rational convex polyhedra.

A trajectory of $\mathcal{H}$ starting from some $\mathbf{x}_0 \in X$, is a solution of the differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$ with initial condition $\mathbf{x}(0) = \mathbf{x}_0$: that is a continuous function $\phi : \mathbb{R}^+ \to X$ such that $\phi(0) = \mathbf{x}_0$, and for every t, $f(\phi(t))$ is equal to the *right derivative* of $\phi(t)$.

In other words, a PCD system consists of partitioning the space into convex polyhedral sets ("regions"), and assigning a constant derivative $\mathbf{c}$ ("slope") to all the points sharing the same region. The trajectories of such systems are broken lines, with the breakpoints occurring on the boundaries of the regions [5]: see the following figure.

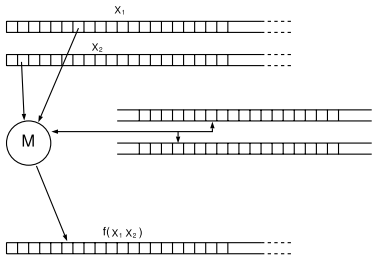


Here is an example of a trajectory of a PCD system:



4.2 The most fundamental model: Turing Machines

Turing machines can also be considered as particular discrete time dynamical systems.



A Turing machine is indeed a discrete time dynamical system $(\Gamma, \vdash)$, where

- $\Gamma = Q \times \Sigma^* \times \mathbb{Z}$ corresponds to configurations (a configuration (q, w, z) is given by some internal state $q \in Q$ of the machine, some position $z \in \mathbb{Z}$ of the head of the machine, and the content $w \in \Sigma^*$, that can be seen as a word over the alphabet Σ of the machine, of the tape).

- $\vdash$ is the "next configuration relation": it relates a configuration to its direct successor (when the machine is deterministic, or to its direct successors when the machine is non-deterministic).

4.3 Some Facts

4.3.1 A Key Decision Problem

The reachability problem is the following decision problem.

- Given
  1. a system $\mathcal{H} = (X, f)$,
  2. some $A \subset X$,
  3. some $B \subset X$,
- determine whether there is a trajectory starting from A ($x(0) \in A$) that reaches B ($x(t) \in B$ for some t).

4.3.2 Some Undecidability Results

- For all 4 models,
  - Reachability is undecidable;
  - Reachability is recursively enumerable;
  - Reachability is Σ_1^0-complete: any r.e. set can be reduced to reachability of a system S.

4.4 Proof method

The idea is to simulate 2-counters (Minsky) machines, or Turing machines.

4.4.1 The involved notion of simulation

Consider some machine M: M can be a Turing machine, a pushdown automaton, or a counter machine. M corresponds to a particular discrete time dynamical system $(\Gamma, \vdash)$.

By a discrete time dynamical system

A PAM simulates M if there is a piecewise affine function $f : I \to I$, where $I \subset \mathbb{R}^d$, and a f-stable $D \subset I$ and a bijective function $\phi : \Gamma \to D$ such that

$$\vdash = \phi^{-1} \circ f \circ \phi.$$

Intuitively, this means that in order to apply T, one can encode the configuration with ϕ, apply f, and then decode the result with ϕ^{-1}.

By a continuous time dynamical system

To a continuous time dynamical system (X, f), one can associate its *stroboscopic map*: this is the discrete time dynamical system (X, g), where $g(x)$ is the solution at time 1 of $x' = f(x)$ with $x(0)) = x$. That is to say, what is obtained by considering the system at discrete time.

We can say that a continuous time dynamical system simulates M if its stroboscopic map simulates M.

4.4.2 Counter machines

We recall that a k-counter machine has k counters: a counter C takes values in $\mathbb{N}$ with operations $C++$ (incrementation), $C--$ (decrementation), test $C > 0$.

A program is then made of these very basic instructions.

For example,

q_1: $D++$; goto q_2

q_2: $C--$; goto q_3

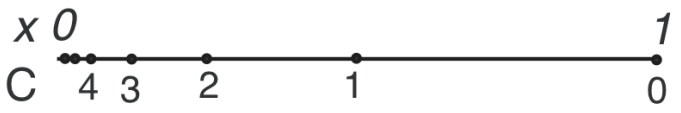
q_3: if $C > 0$ then goto q_2 else goto q_1

is a program.

Recall that reachability is undecidable (and Σ_1^0-complete) for 2-counters (Minsky) machines.

The idea to simulate a counter is to represent the fact that $C = n$ in the counter machine by the fact that some variable x is such that $x = 2^{-n}$.

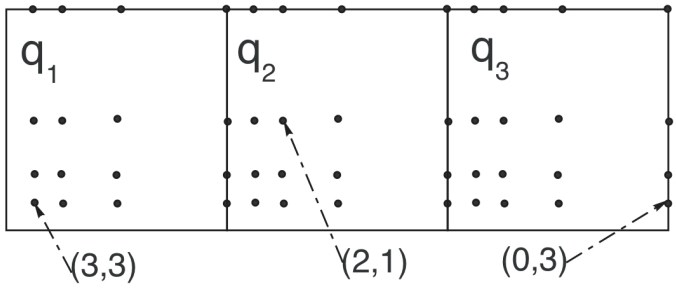
Basically, this can be represented graphically by:



One then uses the following correspondance:

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

To represent a Minsky machine, one then encodes the two counters, and its states into two reals using the following idea:



Minsky Machine	PAM
State Space $\{q_1, q_2, \cdots, q_k\} \times \mathbb{N} \times \mathbb{N}$	State Space $[1, k+1] \times [0,1]$
State (q_i, m, n)	State $x = i + 2^{-m}$, $y = 2^{-n}$

If one prefers, the function ϕ of subsection 4.4.1 is the function that maps $(q, n_1, n_2) \in Q \times \mathbb{N} \times \mathbb{N}$ to $(q + 2^{-n_1}, 2^{-n_2})$.

On the previous example, one just need to consider the following PAM (the value of the piecewise affine function can be defined in any arbitrary way outside the above sets/definition).

Minsky Machine	PAM
State Space $\{q_1, \cdots, q_k\} \times \mathbb{N} \times \mathbb{N}$	State Space $[1, k+1] \times [0,1]$
State (q_i, m, 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 $x = 4$

4.4.3 From Minsky to Turing Machines

This is even possible to do a real time simulation of a 2-stacks (that is to say a Turing) machines.

A stack $S = s_1 s_2 \cdots$ over alphabet $\{0, 1, 2, \cdots, k-1\}$, where s_i is the top of the stack, can be encoded in several ways:

1. Idea 1:

   by
   $$r(S) = \sum_{i=1}^{\infty} \frac{s_i}{k^i}$$

2. Improved idea $1'$:
   $$r(S) = \sum_{i=1}^{\infty} \frac{2s_i + 1}{(2k)^i}$$

For both encoding, stack operations have then arithmetic counterparts:
For the coding corresponding to Idea 1:

$$S' = PUSH(v, S) \text{ if } r(S') = (r(s) + v)/k$$

$$(S', v) = POP(S) \text{ if } r(S') = kr(S) - v$$

As a Turing machine can be considered as a 2-stacks machine: We can encode easily a Turing machine using a PAM $(\mathbb{R}^2, f)$.

The interest of "Improved Idea $1'$" is that it allows to state that we can even assume the piecewise affine map f to be

1. a *continuous* function.

2. and one can even consider domain $[0, 1]^2$, instead of $\mathbb{R}^2$.

Theorem 1 (Theorem 3.1 of [16]) *An arbitrary Turing machine can be simulated in linear time by a* continuous *piecewise linear function* $f : [0, 1]^2 \to [0, 1]^2$.

Proof: Any Turing machine can be considered as a particular 2-stacks automaton, i.e. as a discrete time dynamical system $M = (Q \times \Sigma^* \times \Sigma^*, \vdash)$: (q, γ_1, γ_2) corresponds to internal state q, and to stacks γ_1 and γ_2 seen as words over the alphabet Σ. In order to simplify the description, we suppose wlog in what follows that $Q = \{1, 3\}^{p_1} \times \{1, 3\}^{p_2}$ (you can assume $p_2 = 0$ but this form is more symmetric) and that $\Sigma = \{1, 3\}$.

Each configuration (q, γ_1, γ_2) of M is encoded in the radix-4 expansion of a point (x_1, x_2) of $[0, 1]^2$ as follows: if $q = (q_{1,1}, q_{1,2}, \ldots, q_{1,p_1}, q_{2,1}, q_{2,2}, \ldots, q_{2,p_2}) \in Q = \{1, 3\}^{p_1} \times \{1, 3\}^{p_2}$ and $\gamma_i = s_{i,1}, s_{i,2}, \ldots, s_{i,n}, \ldots$, then

$$x_i = \sum_{j=1}^{p_i} \frac{q_{i,j}}{4^j} + \sum_{j=1}^{\infty} \frac{s_{i,j}}{4^{p_i + j}}$$

We will denote $\overline{abc}$ the real number with radix-4 expansion abc.

Let $I_{1,l_1} \times I_{2,l_2}$ be all the sets defined by:

- $I_{i,l_i} = [l_i, l_i + 1/4^{p_i+p}]$ and $l_i = \overline{0.q_{i,1}q_{i,2}, \ldots, q_{i,p_i}, s_{i,1}}$
- or $I_{i,l_i} = \{l_i\}$ and $l_i = \overline{0.q_{i,1}q_{i,2}, \ldots, q_{i,p_i}}$

for any $s_{i,1}$ and $q_{i,j}$ elements of $\{1, 3\}$.

The stack is nonempty in the first case, and empty in the second one. In what follows, we will not make any more this distinction, and we will suppose, in the case of an empty stack, that $s_{i,1}, s_{i,2}, \ldots, s_{i,p} = 0$.

Assume that $(x_1, x_2) \in I_{1,l_1} \times I_{2,l_2}$ encodes the configuration $(q, a_1\gamma_1, a_2\gamma_2)$ of M at time t, where $a_1, a_2 \in \Sigma$, $\gamma_1, \gamma_2 \in \Sigma^*$ and

$$q = (q_1, q_2) = (q_{1,1}, \ldots, q_{1,p_1}, q_{2,1}, \ldots, q_{2,p_2}) \in Q.$$

Call $\Delta x_i = x_i - l_i$, for $i \in \{1, 2\}$.

On $I_{1,l_1} \times I_{2,l_2}$, we define f such that $f(x_1, x_2) = (x'_1, x'_2)$ with

$$x'_i = \overline{0.q'_{i,1}, \ldots, q'_{i,p_i}} + \Delta x'_i$$

where

$$q' = (q'_1, q'_2) = (q'_{1,1}, q'_{1,2} \ldots, q'_{1,p_1}, q'_{2,1}, q'_{2,2}, \ldots, q'_{2,p_2})$$

is the next internal state of M (fully determined by the current state q and the top-of-stack letters a_1 and a_2), and $\Delta x'_i$ defined by:

- $\Delta x'_i = 4\Delta x_i$ if stack i is popped,
- $\Delta x'_i = \frac{s_{i,1}}{4^{p_i+1}} + \Delta x_i$ if stack i is unchanged
- $\Delta x'_i = \frac{a_i}{4^{p_i+1}} + \frac{s_{i,1}}{4^{p_i+2}} + \frac{\Delta x_i}{4}$ if a_i is pushed on stack i

It can be checked that, in any case, f is built such that $f(x_1, x_2)$ encodes configuration M at time $t + 1$ whenever (x_1, x_2) encodes configuration M at time t.

f is piecewise linear, as it is defined as linear on each of the products $I_{1,l_1} \times I_{2,l_2}$.

In order to complete the proof, we have to define f outside

$$C = \bigcup_{l_1,l_2} I_{1,l_1} \times I_{2,l_2},$$

to the whole of $[0, 1]^2$: this extension cannot interfere with the simulation of M since only points of C are used in a computation. There are continuous piecewise linear extensions of f since the distance between two distinct products is greater than 0. As a matter of fact, the suppremum distance is bounded below by $min(1/4^{p_1+1}, 1/4^{p_2+1}$. □

Remark 1 *Observe that the proofs shows that one need very simple piecewise affine functions to be able to simulate a Turing machine: basically, we only need to be able to do additions, multiplications by* 4 *and divisions by* 4*, on appropriate pieces.*

Remark 2 *Observe that instead of extending the function outside C piecewise linearly, we can even extend it in a $\mathcal{C}^\infty$ way. Hence, an arbitrary Turing machine can be simulated in linear time by a mathcalC^∞ function $f : [0,1]^2 \to [0,1]^2$.*

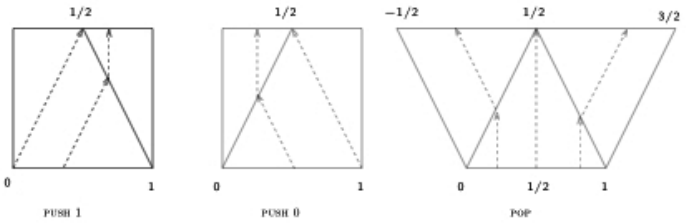
It is believe that however this is not possible using an analytic function over a compact domain.

4.4.4 Using PCDs

Suppose we want to go further and simulate a Turing Machine with PCDs.

The first step is to see that one can do basic operations with PCDS.

For the encoding corresponding to "Idea 1", assuming that the alphabet is $\Sigma = \{0, 1\}$, one just need to consider the following basic PCDs.

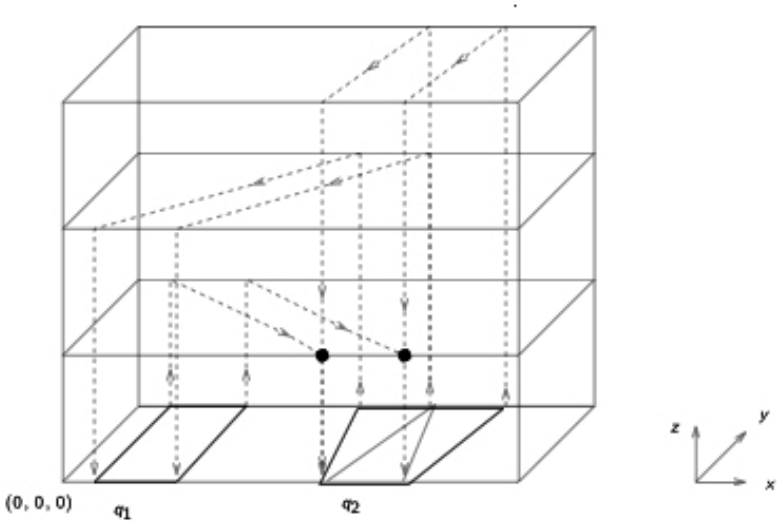


Of course, this is easy to generalize the idea to build basic blocks for the encoding to "Idea 1′": basically, one just need to do multiplications and divisions by 4 instead of 2, which can be done using the same principles.

With these blocks, this is easy to get a PCD that simulates any Push-Down automaton with a PCD.

For example, for the Push-Down automaton

q_1: $S := PUSH(1, S)$; goto q_2
q_2: $(v, S) := POP(S)$; if $v = 1$ then goto q_2 else q_1,
we just need to build a PCD like this:



We get:

Theorem 2 (Asarin-Maler-Pnueli 94) *Every Turing machine can be simulated by a* 4*-dimensional PCD system.*

Playing a little bit with the construction trying to reduce the dimension, one can be easily convinced that dimension 3 is enough:

Theorem 3 (Asarin-Maler-Pnueli 94) *Every Turing machine can be simulated by a* 3*-dimensional PCD system.*

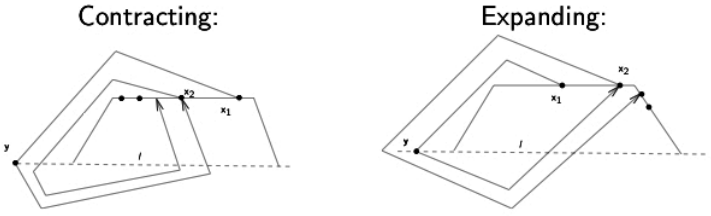
However, this is not possible in dimension 2:

Theorem 4 (Asarin-Maler-Pnueli 94) *But not by a* 2*-dimensional PCD system.*

Indeed:

Theorem 5 (Asarin-Maler-Pnueli 94) *Reachability for planar* 2*-dimensional PCD systems is decidable.*

The main ingredient of the proof: Jordan's theorem: All repetitive behaviors are either contracting or expanding spirals:



4.5 Extensions

Turing machines can be embedded into analog space discrete time systems with low dimensional systems with other simple dynamics: [18], [22], [10], [23] consider general dynamical systems, [16] piecewise affine maps, [24] sigmoidal neural nets, [24], closed form analytic maps, which can be extended to be robust [13], and [17] one dimensional very restricted piecewise defined maps.

Chapter 5

Some Dynamic Undecidability Results: Using a Continuous Time with Smooth Dynamics

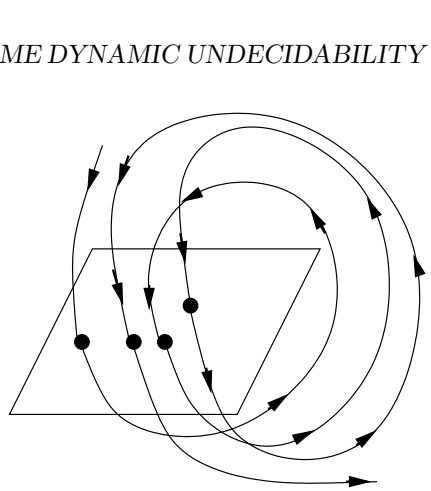
The embedding of Turing machines in continuous dynamical systems is often realized in two steps. Turing machines are first embedded into analog space discrete time systems, and then the obtained systems are in turn embedded into analog space and time systems.

We saw the first step in previous chapter.

For the second step, the most common technique is to build a continuous time and space system whose discretization corresponds to the embedded analog space discrete time system.

We saw an example in previous chapter with PCDs. However, in PCD the dynamic is non-smooth: it is non-continuous. We discuss here what can be said for smooth dynamics (at least continuous).

In a general setting, there are several classical ways to discretize a continuous time and space system: One way is to use a virtual stroboscope: the flow $x_t = \phi(t, x)$, when t is restricted to integers, defines the trajectories of a discrete time dynamical system. Another possibility is through a Poincaré section: the sequence x_t of the intersections of trajectories with, for example, a hypersurface can provide the flow of a discrete time dynamical system. See [14].



The opposite operation, called *suspension*, is usually achieved by extending and smoothing equations, and usually requires higher dimensional systems. This explains why Turing machines are simulated by three-dimensional smooth continuous time systems in [18], [**?**], [10] or by three-dimensional piecewise constant differential equations in [5], while they are known to be simulated in discrete time by only two-dimensional piecewise affine maps in [16]. It is known that two-dimensional piecewise constant differential equations cannot[1] simulate arbitrary Turing machines [5], while the question whether one-dimensional piecewise affine maps can simulate arbitrary Turing machines is open. Other simulations of Turing machines by continuous time dynamical systems include the robust simulation with polynomial ODEs in [13], [**?**]. This result is an improved version of the simulation of Turing machines with real recursive functions in [**?**], where it is shown that smooth but non-analytic classes of real recursive functions are closed under iteration. Notice that while the solution of a polynomial ODE is computable on its maximal interval of existence, the simulation result shows that the reachability problem is undecidable for polynomial ODEs.

5.1 Discussion

The key technique in embedding the time evolution of a Turing machine in a flow is to use "continuous clocks" as in [10].[2]

The idea is to start from the function $f : \mathbb{R} \to \mathbb{R}$, preserving the integers, and build the ordinary differential equation over $\mathbb{R}^3$

$$\begin{aligned} y_1' &= c(f(r(y_2)) - y_1)^3\theta(\sin(2\pi y_3)) \\ y_2' &= c(r(y_1) - y_2)^3\theta(-\sin(2\pi y_3)) \\ y_3' &= 1. \end{aligned}$$

Here $r(x)$ is a rounding-like function that has value n whenever $x \in [n-1/4, n+1/4]$ for some integer n, and $\theta(x)$ is 0 for $x \leq 0$, $\exp(-1/x)$ for $x > 0$, and c is some suitable constant.

[1] See also already mentioned generalizations of this result in [**?**] and [**?**].

[2] Branicky attributes the idea of a two phase computation to [**?**] and [**?**]. A similar trick is actually present in [22]. We will actually not follow [10] but its presentation in [**?**].

The variable $y_3 = t$ is the time variable. Suppose $y_1(0) = y_2(0) = x \in \mathbb{N}$. For $t \in [0, 1/2]$, $y_2' = 0$, and hence y_2 is kept fixed to x. Now, if $f(x) = x$, then y_1 will be kept to x. If $f(x) \neq x$, then $y_1(t)$ will approach $f(x)$ on this time interval, and from the computations in [**?**], if a large enough number is chosen for c we can be sure that $|y_1(1/2) - f(x)| \leq 1/4$. Consequently, we will have $r(y_1(1/2)) = f(x)$. Now, for $t \in [1/2, 1]$, roles are inverted: $y_1' = 0$, and hence y_1 is kept fixed to the value $f(x)$. On that interval, y_2 approaches $f(x)$, and $r(y_2(1)) = f(x)$. The equation has a similar behavior for all subsequent intervals of the form $[n, n + 1/2]$ and $[n + 1/2, n + 1]$. Hence, at all integer time t, $f^{[t]}(x) = r(y_1(t))$.[3] [**?**] proposes a similar construction that returns $f^{[\lfloor t \rfloor]}(x)$ for all $t \in \mathbb{R}$.

In other words, the construction above transforms a function over $\mathbb{R}$ into a higher dimensional ordinary differential equation that simulates its iterations. To do so, $\theta(\sin(2\pi y_3))$ is used as a kind of clock. Therefore, the construction is essentially "hybrid" since it combines smooth dynamics with non-differentiable, or at least non-analytic clocks to simulate the discrete dynamics of a Turing machine. Even if the flow is smooth (i.e. in C^∞) with respect to time, the orbit does not admit a tangent at every point since y_1 and y_2 are alternatively constant. Arguably, one can overcome this limitation by restricting Turing machine simulations to analytic flows and maps. While it was shown that analytic maps over unbounded domains are able to simulate the transition function of any Turing machine in [**?**], only recently it was shown that Turing machines can be simulated with analytic flows over unbounded domains in [13]. It would be desirable to extend the result to compact domains. However, it is conjectured in [**?**] that this is not possible, i.e. that no analytic map on a compact finite-dimensional space can simulate a Turing machine through a reasonable input and output encoding.

5.2 Some Dynamic Undecidability Results

We review some dynamic undecidability results obtained in literature (non-exhaustive list).

- [Moore90]: simulation of Turing machine with a $\mathcal{C}^\infty$ discrete-time dynamic over $\mathbb{R}^2$.
- [Ruohonen93]: simulation of a n-counter machine.
- [Asarin-Maler-Pnueli95]: simulation of a Turing machine with a PCD-system over $\mathbb{R}^3$.
- [Branicky95]: simulation of a Turing machine with hybrid systems.
- [Siegelmann95]: simulation of an extended automata with a mirror system.

[3] $f^{[t]}(x)$ denotes de tth iteration of f on x.

- [Graça-Campagnolo-Buescu2005]: simulation of a Turing machine with a GPAC.
- ...

Chapter 6

Space and Time Contraction for Continuous Time Systems

6.1 Considering Dynamical Systems as Language Recognizers

Dynamical systems can be considered as recognizers of languages: let Σ denote alphabet $\{0,1\}$. Σ^* denotes words over this alphabet.

Two (very classical) encodings of words into real numbers will play some important role in what follows:

- ν_X is the function that maps Σ^* to $[0,1]$ as follows: word $w = w_1 \dots w_n \in \{0,1\}^*$ is mapped to $\nu_X(w) = \sum_{i=1}^{n} \frac{(2w_i+1)}{4^i}$.
- $\nu_{\mathbb{N}}$ is the function that maps Σ^* to $\mathbb{N}$ as follows: word $w = w_1 \dots w_n \in \{0,1\}^*$ is mapped to $\nu_{\mathbb{N}}(w) = \sum_{i=1}^{n} (2w_i + 1)4^i$.

We can now define.

Definition 3 (Dynamical Systems as Language Recognizers) *Let $\mathcal{H}$ be a continuous time or discrete time dynamical system over space X. We will consider two cases: the case $X = [-1,1]^d$ (compact case), or $X = \mathbb{R}^d$ (unrestricted case). Consider $\nu = \nu_X$ for the compact case, $\nu = \nu_{\mathbb{N}}$ for the unrestricted case. Let V_{accept} be the set of $\mathbf{x} \in X$ with $\|\mathbf{x}\| \leq 1/4$. Let $V_{compute}$ be the set of $\mathbf{x} \in X$ with $\|\mathbf{x}\| \geq 1/2$. (or take V_{accept} and $V_{compute}$ to any two disjoint subsets corresponding to a polyhedron with rational coefficients, at a strictly positive distance one from the other).*

We will say that $\mathcal{H}$ computes (or semi-recognize) some language $L \subset \Sigma^*$, over alphabet $\Sigma = \{0,1\}$, if the following holds: for all $w \in \Sigma^*$, $w \in L$ iff the trajectory of $\mathcal{H}$ starting from $(\nu(w), 0, \cdots, 0, 1)$ reaches V_{accept}.

For robustness reasons, we assume that, for any $w \notin L$, the corresponding trajectory stay forever in $V_{compute}$.

Given some notion of time associated to trajectories, we will say that L is recognized in time T, if furthermore when the trajectory reaches V_{accept}, trajectory has a time bounded above by T. It is said accepted in time $f : \mathbb{N} \to \mathbb{N}$ if furthermore $T \leq f(|w|)$, for all w, where $|w|$ stands for the length of w.

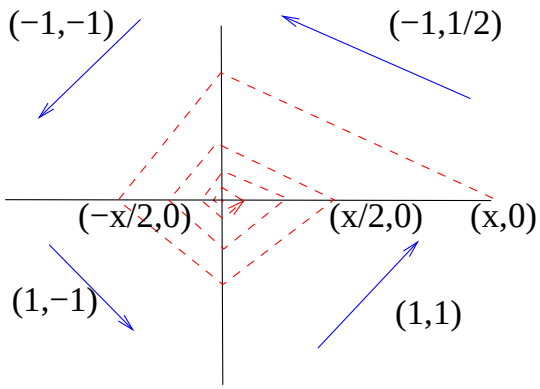
In this chapter, we consider continuous-time dynamical systems, and talk about computability issues. In next chapter, we will focus on complexity.

6.2 Time Contraction for PCD systems

6.2.1 Zeno's Paradox

We come back to PCD systems.

Consider the following PCD:



It takes a time $5x * 1/2$ to go from $(x, 0)$ to $(x/2, 0)$ that is to say to make a turn of the spiral.

Second turn is made in time $5x * 1/4$. Third in times $5x * 1/8$. And so on.

Considering that

$$5/2(x + x/2 + x/4 + \dots) = 5x$$

is finite, one can consider that in time $5x$ the trajectory has reached $(0, 0,$ the limit of the spiral. This happens in finite time, but requires a transfinite number of crossing of regions.

This is called *Zeno's paradox*: to a continuous finite time can correspond a transfinite number of discrete steps.

6.2.2 Using Zeno's Paradox

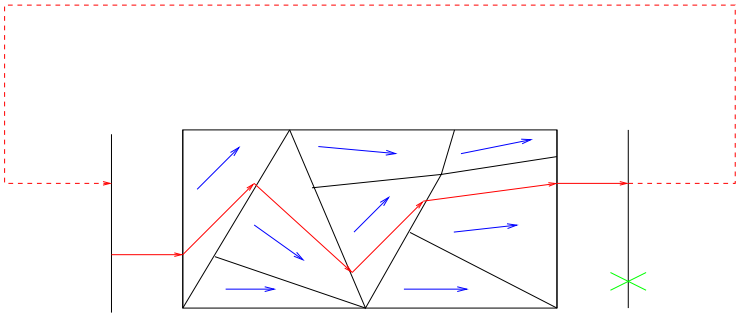
Actually, using the fact that one can simulate a Turing machine M using a PCD system in dimension 3, and this idea, it is possible to build a PCD system

of dimension 4 that decides whether M terminates or not, i.e. that solves the halting problem of M.

The idea of the construction is the following.

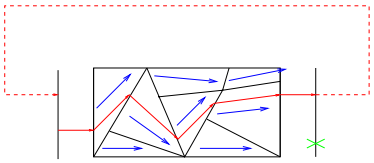
Recall that one can build a PCD system in dimension 3 that simulates a Turing machine M.

Let it correspond to the following (very abstract) picture:



Suppose that you divide all dimensions by 2, but keep speeds unchanged.

You get the following PCD:

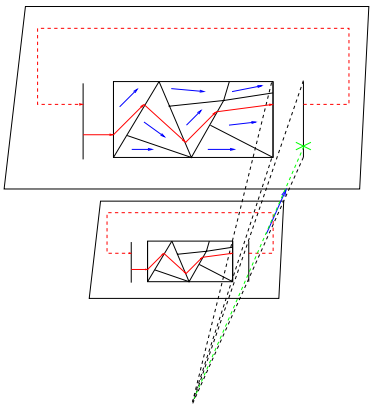


The point is that if it takes a time T for the first one to make a turn, it will make a time $T/2$ for the second to do exactly the same turn.

Of course, instead of dividing everything by 2, you can divide by 4, by 8 and so on.

The trick is that if one put all of this in a 4-dimension space, say x, y, z, u, by putting the first at $u = 1$, the second at $u = 1/2$, the third at $u = 1/4$ and so one, one gets a "pyramid" that corresponds indeed to a PCD.

Graphically:



In the first one, we may assume that there is a particular point that corresponds to the accepting state of Turing machine M.

In the pyramid, it will correspond to an open segment. Assume that we fix that the speed on this segment goes upward.

Assume that we add some regions that maps u to $u/2$ (and all variables divided by 2 also), and that set the system that these regions are crossed at each turn: we will get a PCD that simulates one step of Turing machine M in time 1, another step in time $1/2$, another step in time $1/4$ and so one. As $1 + 1/2 + 1/4 + 1/8 + \cdots = 2$, in time 2 either we will reach the accepting segment. Otherwise, we will reach the summit of the pyramid.

In other words, if M accepts we will reach the accepting point at $u = 1$. If M rejects, we will reach the point $(0, 0, 0, 0)$. That is to say, we decide whether M accepts or not in finite time !!

Details can be found in [4].

6.2.3 What can be computed?

Hierarchies of undecidable problems

We recall the following definition:

Definition 4 (Arithmetical hierarchy [21, 19]) *The classes $\Sigma_k, \Pi_k, \Delta_k$, for $k \in \mathbb{N}$, are defined inductively by:*

- *Σ_0 is the class of the languages that are recursive;*
- *For $k \geq 1$, Σ_k is the class of the languages that are recursively enumerable in a set in Σ_{k-1} (that is semi-recognized by a Turing machine with an oracle in Σ_{k-1});*
- *For $k \in \mathbb{N}$, Π_k is defined as the class of languages whose complement are in Σ_k, and Δ_k is defined as $\Delta_k = \Pi_k \cap \Sigma_k$.*

Several characterizations of the sets of the arithmetical hierarchy are known: see [19, 21]. In particular assume a first order formula F, over some recursive predicates, characterizing the elements of a set $S \subset \mathbb{N}$, is given. Then S is in the arithmetical hierarchy and the Tarski-Kuratowski algorithm on formula F returns a level of the arithmetical hierarchy containing S: see [19, 21] for the full details.

The hyper-arithmetical hierarchy is an extension of the arithmetical hierarchy to constructive ordinal numbers. It consists of the classes of languages $\Sigma_1, \Sigma_2, \ldots, \Sigma_k, \ldots, \Sigma_\omega, \Sigma_{\omega+1}, \Sigma_{\omega+2}, \ldots, \Sigma_{\omega 2}, \Sigma_{\omega 2+1}, \ldots, \Sigma_{\omega^2}, \ldots$ indexed by the constructive ordinal numbers. It is a strict hierarchy and it satisfies the strict inclusions $\Sigma_\alpha \subset \Sigma_\beta$ whenever $\alpha < \beta$. It can be related to the analytical hierarchy by $\Delta_1^1 = \cup_\beta \Sigma_\beta$: see [21].

The idea of the construction of this hierarchy is the following:

- Σ_1 is defined as the class of the recursively enumerable sets: that is to say Σ_1 is the class of the languages that are semi-recognized by a Turing machine.
- When k is a constructive ordinal and when the class Σ_k is defined, Σ_{k+1} is defined as the class of the languages that are recursively enumerable in a set in Σ_k: that is to say Σ_{k+1} is the class of the languages that are semi-recognized by some oracle Turing machine whose oracle is a language in Σ_k.
- When k is a constructive limit ordinal, $k = lim\ k_i$, and when the classes $(\Sigma_{k_i})_{i\in\mathbb{N}}$ are defined, Σ_k is defined as the class of the languages that are recursively enumerable in some fixed diagonalization of classes $(\Sigma_{k_i})_i$.

Summary:

If you prefer:

- Σ_1 = Recursively enumerable sets.
- ...
- Σ_{k+1} = Sets recursively enumerable in a set in Σ_k.
- ...
- Σ_ω = Sets recursively enumerable in a diagonalisation of $\Sigma_{\gamma<\omega}$
- $\Sigma_{\omega+1}$ = Sets recursively enumerable in a set in Σ_ω
- ...
- $\Sigma_{\alpha=\lim\gamma}$ = Sets recursively enumerable in a diagonalisation of $\Sigma_{\gamma<\alpha}$.
- $\Sigma_{\alpha+1}$ = Sets recursively enumerable in a set of Σ_α

- ...

It is then possible to relate the computational power of PCD systems in finite continuous time to the hyperarithmetical hierarchy ([4, 7]).

Dimension	Languages semi-recognized
2	$< \Sigma_1$
3	Σ_1
4	Σ_2
5	Σ_ω
6	$\Sigma_{\omega+1}$
7	Σ_{ω^2}
8	Σ_{ω^2+1}
...	...
2p+1	$\Sigma_{\omega^{p-1}}$
2p+2	$\Sigma_{\omega^{p-1}+1}$

In particular, any set definable by some arithmetical formula is decided in dimension 5 !!!

Chapter 7

Dynamical Systems with Non-Necessarily Rational Coefficients

Recall that we saw in Section 6.1 how a dynamical system can be considered as a language recognizer.

We said that given some notion of time associated to trajectories, a language L is recognized in time T, if furthermore when the trajectory reaches V_{accept}, trajectory has a time bounded above by T. It is said accepted in time $f : \mathbb{N} \to \mathbb{N}$ if furthermore $T \leq f(|w|)$, for all w, where $|w|$ stands for the length of w.

For discrete time dynamical systems, the most natural notion of time is the number of discrete steps of the trajectory: in particular, a language $L \subset \Sigma^*$ is said to be recognized in *polynomial time* by dynamical system $\mathcal{H}$ if for some k, or all $w \in \Sigma^*$, $w \in L$ iff the trajectory of $\mathcal{H}$ starting from $(\nu(w), 0, \cdots, 0, 1)$ reaches V_{accept} in less than n^k steps, where $n = |w|$: that is to say, $f^t(\nu(w), 0, \cdots, 0, 1) \in V_{accept}$ for some $t \leq n^k$.

We want to understand what can be recognized by our previous models of dynamical systems in polynomial time.

Recall that in definition 6, we assumed all coefficients defining the affine functions and all rational coefficients defining the polyhedra to be rational.

Suppose we relax these constraints, calling previous PAM as *rational PAM* in opposition to more general PAM where coefficients can be arbitrary reals.

Definition 5 (PAM [4]) *A* Rational *Piecewise Affine Map (PAM) is a discrete time dynamical system $\mathcal{H}$, defined by $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$ on $X \subset \mathbb{R}^d$, where $f : X \to \mathbb{R}^d$ is piecewise affine: that is to say,*

$$f(\mathbf{x}) = f_i(\mathbf{x}) \text{ for } \mathbf{x} \in P_i,\ i = 1, \ldots, n$$

where f_i is some affine function with rational coefficients, and the P_i constitutes a finitely many partition of X into rational convex polyhedra.

Definition 6 (PAM [4]) *A* General *Piecewise Affine Map (PAM) is a discrete time dynamical system $\mathcal{H}$, defined by $\mathbf{x}_{t+1} = f(\mathbf{x}_t)$ on $X \subset \mathbb{R}^d$, where $f : X \to \mathbb{R}^d$ is piecewise affine: that is to say,*

$$f(\mathbf{x}) = f_i(\mathbf{x}) \text{ for } \mathbf{x} \in P_i,\ i = 1, \ldots, n$$

where f_i is some affine function with (arbitrary) reals coefficients, and the P_i constitutes a finitely many partition of X into (arbitrary) convex polyhedra.

It is rather easy to see that the languages recognized by Rational PAM correspond to P, the class of languages recognized by Turing machines in polynomial time (the P of the P = NP question).

The purpose of this chapter is to determine what correspond to General PAM.

7.1 "Analog/Advice" Automata

It may help to consider the following (rather artificial) model (following [9]).

Definition 7 *An analog automaton (or advice automaton) (with k stacks) is exactly like a pushdown automata with k stacks, except that it has the possibility in addition of making appear in time* 1 *an infinite word $W \in \Sigma^\omega$ (that we can call* advice*) over some stack: in time* 1*, the content of the stack is replaced by W.*

As the program of an analog automaton is finite, there is only a finite number of possible advices that a given machine can make appear.

If you prefer considering that pushdown automata are actually Turing machines, if you prefer talking only about Turing machines, this corresponds to consider Turing machines that can replace the content of the tape at the right (or left) of the head by some infinite words W in time 1.

More formal definitions can be found in [9], but basically they are only formalizing the previous ideas.

The question is then to understand what can be computed by analog automata.

7.1.1 In exponential time

Theorem 6 *Every language $L \subset \{0,1\}^*$ can be recognized by a (deterministic) analog two stack automaton in exponential time.*

Proof: Let $L \subset \{0,1\}^+$ be a language. Let the word γ, be the concatenation, with delimiters, by increasing word length order, of all the words of L. Let M be an analog automaton that, on input $w \in \{0,1\}^+$ on its first stack, makes advice γ appear on its second stack. Then M seeks in γ if w is present. If it is, M accepts. M stops processing as soon as it encounters a word of length greater than the length of w. L is recognized by M in exponential time. □

7.1.2 In polynomial time

In order to talk about polynomial time, we need to talk about class P/*poly*.

In computational complexity theory, an advice string is an extra input to a Turing machine which is allowed to depend on the length n of the input, but not on input itself. A decision problem is in the complexity class $P/f(n)$ if there is a polynomial time Turing machine M with the following property: for any n, there is an advice string A of length $f(n)$ such that, for any input x of length n, the machine M correctly decides the problem on the input x, given x and A.

P/*poly* is obtained by considering there case where functions f correspond to polynomials.

More formally, this correspond to the following definition.

Definition 8 (P/*poly***)** P/*poly is the class of languages B such that there is some language A recognized in polynomial time (i.e. $A \in$ P), some function $f : \mathbb{N} \to \Sigma^*$ and some polynomial p such that for all n, $|f(n)| \leq p(n)$, and*

$$B = \{x| < x, f(|x|) > \in A\}.$$

Theorem 7 *The languages $L \subset \{0,1\}^*$ accepted by analog (deterministic) two stack automata in polynomial time are exactly the languages belonging to the complexity class $P/poly$.*

Proof: Let k be the number of different advices that the analog automaton M can possibly use. In polynomial time $p(n)$, M can at most read the $p(n)$ first letters of the k advices. So it is possible to simulate M with a Turing machine M', which gets as advice of polynomial size $kp(n)$ the $p(n)$ first letters of each of the k advices of M, and then simulates M. Hence the computational power of analog automata in polynomial time is bounded by $P/poly$.

Let L be a language in $P/poly$. By definition, L is recognized by a Turing machine M' with an advice function $f : \mathbb{N} \to \{0,1\}^*$. We can construct a word γ of infinite length as the concatenation, with delimiters, of $f(1), f(2), etc....$ In order to recognize L, an analog automaton M, on input $w \in \{0,1\}^+$, first makes advice γ appear. Then M seeks in γ the value of $f(|w|)$. This operation can be done in polynomial time, since there exists a polynomial p, such that, for all $i \in \mathbb{N}$, the size of $f(i)$ is bounded by $p(i)$: so M has at most to read $p(1) + p(2) + \cdots + p(|w|)$ characters, that is at most a polynomial number of characters. Finally, M simulates Turing machine M' on $(w, f(|w|))$. Hence L is recognized by M in polynomial time. □

7.2 More on class P/*poly*

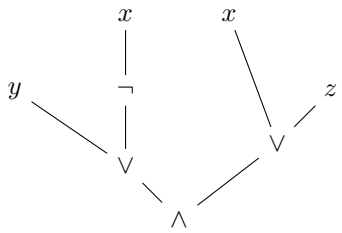
7.2.1 Boolean circuits

One can see a boolean circuit as a mean to describe a boolean function as a sequence of OR ($\vee$), AND ($\wedge$) and NOT ($\neg$) on bits given as input.

Definition 9 (Boolean circuit) *Let n be an integer. A* boolean circuit *with n inputs and one output is a DAG (directly oriented graph) with*

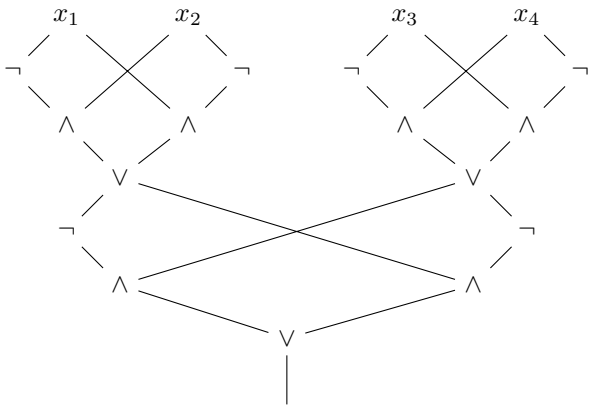
- n inputs*, that is to say n vertices without ingoing arc.*
- *exactly one output, that is to say a vertex without outgoing arc*
- *each input is labeled either by constant* 0*, or by* 1 *or by some symbol of variable $x_1, x_2, \cdots, x_n$.*
- *any other vertex is a called a* gate *and is labeled either by $\vee$, $\wedge$ or $\neg$. The* fanin *is the ingoing degree of a gate.*
- *gates labeled by $\vee$ or $\wedge$ have fanin* 2*.*
- *gates labeled by $\neg$ have fanin* 1*.*

Example 1 *Here is an example of circuit corresponding to function $S(x, y, z) = \wedge(\vee(\neg(x), y), \vee(x, z))$.*



Observe that several inputs can be labeled by a same symbol.

Example 2 *Here is a less trivial example.*



Given a boolean circuit C with n inputs, and $\mathbf{x} \in \{0,1\}^n$, *the output of C* on $\mathbf{x} = (x_1, \cdots, x_n)$, written $C(\mathbf{x})$ is defined inductively as expected.

Example 3 *The circuit of Example 1 computes the function from* $\{0,1\}^3$ *to* $\{0,1\}$ *defined by* $S(x,y,z) = \wedge(\vee(\neg(x),y),\vee(x,z))$,

Example 4 *The circuit of Example 2 computes the function* $PARITY$ *from* $\{0,1\}^4$ *to* $\{0,1\}$ *that values* 1 *iff an odd number of its arguments values* 1.

The *size* of a circuit C, is the number of vertices of the circuit. Its *depth* is the length of the longest path from an input to the output.

Example 5 *The circuit of example 2 is of size* 19 *and of depth* 6.

A given circuit recognizes only words over $\{0,1\}$ of fixed lengths. If one want consider recognition of languages, one needs to talk about family of circuits.

Recall (see e.g. [6, 20]) that a family of boolean circuits $\mathcal{C} = (C_i)_{i\in\mathbb{N}}$, with C_i with i inputs and 1 output, recognizes a language $L \subset \Sigma^*$, iff for all $w \in \Sigma^*$, $w \in L$ if and only if $C_{|w|}$ accepts w.

We assume fixed a reasonable way to encode circuits: evaluation of a circuit C on some input x can be done in polynomial time. Denote by CVP the decision problem that consists in evaluation C on input x.

7.2.2 Polynomial time

Polynomial time can be characterized by circuits:

Theorem 8 (P versus P/*poly* (see e.g. [20])) *A language* $L \subset \Sigma^*$ *is recognized in polynomial time by a Turing machine, iff*

1. *L is recognized by a family of circuits of polynomial size: there exists some polynomial p, with $size(C_n) = p(n)$ for all n.*

2. *the function that maps 1^n to the encoding of circuit C_n is computable in polynomial time.*

Proof: The idea of the direct sense of the proof is to see that for a given length, as the Turing machine M works in less than $p(n)$ steps for some polynomial p, using less than $q(n)$ cells of the tape, by unfolding the program of M (as in the proof of Cook's Theorem (NP-completenes of SAT)), one can build a circuit C_n that decides if a word w of length n is accepted by M.

Then to observe that the circuit C_n, being obtained as a simple unfolding of the program of M, the function that maps 1^n to the encoding of circuit C_n is indeed computable in polynomial time.

Conversely, given a word w, one can compute its length $n = |w|$, then C_n and simulates C_n on w. With the two hypotheses, all of this can be done in polynomial time, and hence the language L is in P. □

7.2.3 Non uniform polynomial time

Class P/*poly* corresponds to non-uniform polynomial time, since it consists in relaxing second condition in next characterization of polynomial time.

Theorem 9 *A language $L \subset \Sigma^*$ is in* P/*poly iff L is recognized by a family of circuits of polynomial size: there exists some polynomial p, with $size(C_n) = p(n)$ for all n.*

Proof: Let L be a set with polynomial size circuits. Let C_n be the encoding of the polynomial size circuit that recognizes $L \cap \Sigma^n$, where Σ^n is the set of words of length n. We have $|C_n| \leq p(n)$ for some polynomial p. Define the advice function f as $f(n) = C_n$. For any length n, and for any w of length n, we have $w \in L$ iff C_n outputs 1 on input w iff $< w, f(|w|) > \in CVP$. As CVP is in P, L is in P/*poly*.

Conversely, let L be in P/*poly*. By definition there exists a set A in P and an advice function f such that $x \in L$ iff $w \in L$ iff $< w, f(|w|) > \in A$. Since A is in P, there exists a polynomial time Turing machine M such that $A = L(M)$. By the same idea as in the proof of Theorem 8, there is a circuit that outputs 1 iff its input is in B.

The idea is then to consider as family of circuits the family of circuits that corresponds to each length, plugging the advice $f(n)$ in the circuit corresponding to length n. □

7.2.4 Other characterizations

A set $S \subset \Sigma^*$ is said to be *sparse* if there exists a polynomial $p(n)$ sucht that for every n, S has less than $p(n)$ words of length $\leq n$.

P(S) denotes the languages recognized in polynomial time with oracle S.

Theorem 10 P/*poly* $= \bigcup_{S\ sparse}$ P(S).

Proof: We prove that a set L is in P/*poly* iff there is a sparse set S such that $L \in P(S)$.

Assume that $L \in$ P/*poly* via the advice function f and the set $A \in$ P. Define S as follows:

$$S = \{< 0^n, w > | w \text{ is a prefix of } f(n) >\}.$$

S is sparse: each word in S of length m is of the form $< 0^n, w >$ for $n \leq m$. There are $m + 1$ possible values of n. Each of them contributes at most $m + 1$ different prefixes of $f(n)$, one for each length up to m. The total number of words of length m is at most $O(m^2)$.

Now L is in $P(S)$, as using S as oracle, one can easily build the advice z (extending bit by bit z by 0 or 1's, querying for each bit whether it should be 0 or 1), and then use it to compute L.

Conversely, assume that $L \in P(S)$, where S is sparse. Let p be the polynomial bounding the running time of the machine that decides L. Define the advice function such that, for each n, it gives the encoding of the set of words

in S up to size $p(n)$. This is a polynomially long encoding. Using this advice, this is easy to simulate queries to S in polynomial time. □

A set $S \subset \Sigma^*$ is said to be *tally* if $S \subset \{a\}^*$ for some symbol a.

It is possible to prove the following (left as an exercice):

Theorem 11 $\mathrm{P}/poly = \bigcup_{S\ tally} \mathrm{P}(S)$.

7.2.5 Known Facts

- $\mathrm{P}/poly$ contains some non-computable languages: consider some non-computable $A \subset \mathbb{N}$, and consider the language A made of words of length n with $n \in A$. A is non-computable, as A is. By definition, it is indeed in $P/poly$, as the function that maps n to 0 or 1 according to whether $n \in A$ or not is a valid advice function.
- $\mathrm{P}/poly$ contains P and BPP (Adleman's theorem).
- If NP $\subset \mathrm{P}/poly$ then the polynomial hierarchy collapses to Σ_2^P (Karp-Lipton's theorem).
- etc...

7.3 General PAM

7.3.1 General PAM can simulate Analog Automata

Theorem 12 (Theorem 3.1 of [16]) *An arbitrary "Analog" 2-stacks automaton can be simulated in linear time by a* continuous *piecewise linear function* $f : [0,1]^2 \to [0,1]^2$.

Proof: Basically, the proof is exactly the same as in Theorem 1, except that one just need to explain how to simulate the apparition of an advice by a PAM.

But this is only adding the line

- $\Delta x'_i = \gamma$, if γ encodes advice W, if the program says to make appear advice W on stack i.

to the itemize defining $\Delta x'_i$ in that proof. □

7.3.2 Consequences

Proposition 1 (Unrestricted power in exponential time) *Any language $L \in \Sigma^*$ is recognized in exponential time by a PAM.*

Proposition 2 (At least $\mathrm{P}/poly$ in polynomial time) *Any language $L \in \mathrm{P}/poly$ is recognized in polynomial time by a PAM.*

This is not hard to see that this also holds for PCD systems, taking as time measure the number of regions crossed.

As it is known that P/*poly* contains some non-computable sets, it is possible to say that PAMs (and PCD systems)with non-rational coefficients are stronger than classical Turing Machines [9].

7.4 Upper bounding the power of PAMs

It turns out that they cannot compute more.

Theorem 13 *A language $L \subset \Sigma^*$ is recognized in polynomial time by a PAM iff it belongs to* P/*poly*.

7.4.1 The case of continuous PAMS

We do the proof first for the case of continuous piecewise affine maps.

Definition 10 *A PAM $\mathcal{H} = (X, f)$ is said to be* continuous *if function f is continuous.*

We say that the digits $x_1, x_2, \ldots, x_n$, with $x_i \in \{0, 1\}$ are the first n digits of $x \in [0, 1]$ if the real number y whose base 2 expansion is $y = 0.x_1x_2 \ldots, x_n$ is such that $|x - y| \leq 2^{-n}$ (there is of course no uniqueness).

We note $y = Trunc_n(x)$ in that case. Similarly, we define $Trunc_n(x)$ for $x \in [0, 1]^d$ by taking the first n digits componentwise.

Recall that f is said to be Lipschitz if there is some constant C such that for all x, y in its domain, $||f(x) - f(y)|| \leq C||.x - y||$.

Lemma 1 *Assume that $f : [0, 1]^d \to [0, 1]^d$ is piecewise affine and continuous. Then f is Lipschitz and $Trunc_n(f(x))$ can be computed in polynomial time on an input $x \in D_n^d$, where D_n is the set of real numbers of the form $0.x_1 \ldots x_n$, $x_i \in \{0, 1\}$ in radix 2.*

Proof: Such a continuous piecewise affine map is clearly Lipchitz: on each polyhedron P on which f is affine, f is Lipchitz: each component $f_j(x)$ is of the form $f_j(x) = \sum_{i=1}^{d} w_{i,j}x_i + \theta_j$. The Lipschitz constant of f on P is $L_P = \max_j \sum_{i=1}^{d} |w_{i,j}|$. The Lipschitz constant of f is then $\max_{P \ polyhedron} L_P$.

In order to show that f is computable in polynomial time in the above sense, we first show that f is a Lipschitz function of its parameters. We will now write $f(W, x)$ in order to take them into account.

- Let us first define a convention allowing to associate a unique vector of parameters $W \in \mathbb{R}^k$ to f: we can assume the polyhedra to be a triangulation, i.e. that f is a affine on a family of polytopes (the convex hull of $d+1$ points in general position). f is then uniquely defined by the list W of the vertices of the polytopes, together with the value of f at these vertices,

enumerated in some fixed order. Some vectors $W \subset \mathbb{R}^k$ are not associated to a triangulation of $[0,1]^d$ (this occurs when two polytopes intersect); others are associated to a triangulation that does not cover $[0,1]^d$. Call such vectors *invalid*.

- The set of valid vectors is open: for any valid vector W_0, there is a parallelepiped $B(W_0, r)$ of valid vectors. We will now work on such a parallelepiped, and consider given $x \in [0,1]^d$ the function

$$\begin{array}{rcl} g & : & B(W_0,r) \to \mathbb{R}^d \\ & & W \mapsto f(W,x) \end{array}$$

This function is continuous and piecewise-linear, since the equation determining the polytope in which x is located are linear. Hence the analysis of $x \mapsto f(W,x)$ made at the beginning of the proof can be applied to g: g is Lipschitz and its Lipschitz constant is smaller than $\sum_{i=1}^{d} x_i \leq d$.

We are now ready to show how to compute $f(W_0, x)$ in polynomial time. Let $x \in D_d^n$. An approximation of $f(W_0, x)$ can be obtained simply by computing $f(Trunc_q(W_0), x)$ for q large enough. Since $g := W \mapsto f(W,x)$ is Lipschitz, $q = n + C$ is sufficient to obtain n digits of $f(W_0, x)$, for some constant C. The computation takes polynomial time, and the advice is $Trunc_q(W_0)$. □

Lemma 2 *Let $(f_t)_{t\in\mathbb{N}}$ be a family of Lipschitz functions on $[0,1]^d$ having their Lipschitz constant uniformly bounded above by $C > 1$ and $\epsilon > 0$. Let $(x(t))_{t\in\mathbb{N}}$ and $(\overline{x}(t))_{t\in\mathbb{N}}$ be two sequences defined by $x(t+1) = f_t(x(t))$ and*

$$\overline{x}(t+1) = f_t(\overline{x}(t)) + \epsilon(t)$$

with for all $t \geq 0$, $||\epsilon(t)|| \leq \epsilon$ and $||\overline{x}(0) - x(0)|| \leq \epsilon$.

Then

$$\forall t \geq 0, ||\overline{x}(t) - x(t)|| \leq \frac{\epsilon}{C-1}(C^{t+1} - 1).$$

Proof: By induction. The result is true for $t = 0$. Assume that it holds at time t. Then

$$\begin{array}{rcl} ||\overline{x}(t+1) - x(t+1)|| & \leq & C||\overline{x}(t) - x(t)|| + \epsilon \\ & \leq & \frac{C\epsilon}{C-1}(C^{t+1} - 1) + \epsilon = \frac{\epsilon}{C-1}(C^{t+2} - 1). \end{array}$$

□

Theorem 14 *Assume that $L \subset \{0,1\}^*$ is computed by some* continuous *PAM $\mathcal{H} = ([0,1]^d, f)$ in polynomial time. Then $L \in \mathrm{P}/poly$.*

Proof: In order to simplify the proof, consider that the PAM accepts when $x_1 \in [a_0, b_0]$ and rejects when $x_1 \in [a_1, b_1]$ and that all computations are either accepting or rejecting: as the time is upper bounded (by some polynomial) this latter hypothesis is easy to fulfill.

To decide L, we just need to be able to compute $x()$ with the accuracy $\frac{a_1-b_0}{3}$ for $0 \leq t \leq t(u)$, where $t(u)$ is the computation time on input u. Let $n = |u|$ be the size of u. Starting from $\overline{x}(0) = Trunc_q(\nu(u))$, define $\overline{x}(t+1) = Trunc_q(f(\overline{x}(t)))$. Let $C > 1$ be an upper bound of the Lipschitz constant of f. According to Lemma 2, it is sufficient to have

$$\epsilon = 2^{-q} \leq \frac{(a_1 - b_0)(C-1)}{3(C^{T(n)} - 1)}.$$

Hence we can take $q = O(n^k)$ where k is a constant. $\overline{x}(t+1)$ can be computed from $\overline{x}(t)$ in polynomial time with a polynomial size advice from Lemma 1. □

7.4.2 The case of possibly discontinuous PAMS

Theorem 15 *Assume that $L \subset \{0,1\}^*$ is computed by some PAM $\mathcal{H} = ([0,1]^d, f)$ in polynomial time. Then $L \in \mathrm{P}/poly$.*

This follows from the fact that such a PAM can be simulated by a linear machine (that is is in in P_{lin} with the notes to come) , and that the boolean part of languages recognized by linear machines is upper-bounded by $\mathrm{P}/poly$.

Definitions and explanations to come.

Chapter 8

Bibliography

This document has been written using mainly [8, 9], and [16] and some personal documents or slides for the first 6 chapters.

Chapter 7 is mainly based on [15] and [9].

Several illustrations or pictures have been borrowed or stolen from talks or from papers of Eugen Asarin and Oded Maler.

Bibliography

[1] Panos J. Antsaklis, editor. *Proceedings of the IEEE*, volume 88, July 2000.

[2] Eugene Asarin. Chaos and undecidabilty (draft), 1995. Avalaible in `http://www.liafa.jussieu.fr/~asarin/`.

[3] Eugene Asarin and Ahmed Bouajjani. Perturbed Turing machines and hybrid systems. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science (LICS-01)*, pages 269–278, Los Alamitos, CA, June 16–19 2001. IEEE Computer Society Press.

[4] Eugene Asarin and Oded Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences*, 57(3):389–398, December 1998.

[5] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, February 1995.

[6] José Luis Balcázar, Josep Diáz, and Joaquim Gabarró. *Structural Complexity I*, volume 11 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[7] Olivier Bournez. *Complexité Algorithmique des Systèmes Dynamiques Continus et Hybrides*. Phd thesis, Ecole Normale Supérieure de Lyon, 18 Janvier 1999.

[8] Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.

[9] Olivier Bournez and Michel Cosnard. On the computational power of dynamical systems and hybrid systems. *Theoretical Computer Science*, 168(2):417–459, November 1996.

[10] M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 6 February 1995.

[11] E. A. Coddington and N. Levinson. *Theory of Ordinary Differentiel Equations*. McGraw-Hill, April 1972.

[12] A. Filippov. *Differential equations with discontinuous right-hand sides.* Kluwer Academic Publishers, 1988.

[13] Daniel S. Graça, Manuel L. Campagnolo, and Jorge Buescu. Robust simulations of Turing machines with analytic maps and flows. In B. Cooper, B. Loewe, and L. Torenvliet, editors, *Proceedings of CiE'05, New Computational Paradigms*, volume 3526 of *Lecture Notes in Computer Science*, pages 169–179. Springer-Verlag, 2005.

[14] Morris W. Hirsch, Stephen Smale, and Robert Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos.* Elsevier Academic Press, 2003.

[15] Pascal Koiran. On the relations between dynamical systems and boolean circuits. Technical Report 01, Ecole Normale Supérieure de Lyon, January 1993.

[16] Pascal Koiran, Michel Cosnard, and Max Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132(1-2):113–128, September 1994.

[17] Oleksiy Kurganskyy and Igor Potapov. Computation in one-dimensional piecewise maps and planar pseudo-billiard systems. In Cristian Calude, Michael J. Dinneen, Gheorghe Paun, Mario J. Pérez-Jiménez, and Grzegorz Rozenberg, editors, *Unconventional Computation, 4th International Conference, UC 2005, Sevilla, Spain, October 3-7, 2005, Proceedings*, volume 3699 of *Lecture Notes in Computer Science*, pages 169–175. Springer, 2005.

[18] Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354–2357, May 1990.

[19] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the foundations of mathematics*. North-Holland, April 1992.

[20] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[21] Hartley Rogers Jr. *Theory of Recursive Functions and Effective Computability.* MIT Press, April 1987.

[22] Keijo Ruohonen. Undecidability of event detection for ODEs. *Journal of Information Processing and Cybernetics*, 29:101–113, 1993.

[23] Keijo Ruohonen. Undecidable event detection problems for ODEs of dimension one and two. *Theoretical Informatics and Applications*, 31(1):67–79, 1997.

[24] Hava T. Siegelmann and Eduardo D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, February 1995.