On the computational power of dynamical systems and hybrid systems*

Olivier Bournez and Michel Cosnard

Laboratoire de l'Informatique du Parallélisme
Ecole Normale Supérieure de Lyon
46, Allée d'Italie
F-69364 Lyon Cedex 07, France
e-mail: {obournez | cosnard }@lip.ens-lyon.fr

Abstract

We explore the simulation and computational capabilities of discrete and continuous dynamical systems. We introduce and compare several notions of simulation between discrete and continuous systems. We give a general framework that allows discrete and continuous dynamical systems to be considered as computational machines. We introduce a new discrete model of computation: the analog automaton model. We characterize the computational power of this model as $P/poly$ in polynomial time and as unbounded in exponential time. We prove that many very simple dynamical systems from literature are able to simulate analog automata. From this results we deduce that many dynamical systems have intrinsically super-Turing capabilities.

1 Introduction

The computational power of abstract machines which compute over the reals in unbounded precision in constant time is still an open problem. We refer the reader to [18] for an up-to date survey. Indeed, a basic model for their computations has been proposed by Blum Shub and Smale [8] and subsequently modified by Koiran [15]. When restricted to discrete inputs, such models were proved to compute in exponential time any boolean functions, and hence to have super-Turing capabilities. Recently, Siegelmann and Sontag studied the computational power of analog recurrent neural networks, with real weights. They proved that analog neural networks also have super-Turing capabilities [25].

Thus it is possible to get computational machines strictly more powerful than Turing machines, if the machines are able to compute with unbounded-precision reals. But, it may be argued that these machines (BSS machines, analog recurrent neural networks) are purely theoretical machines. The aim of this paper is to show that, actually, many dynamical systems or hybrid system models as defined in the literature also have super-Turing capabilities. Hence, we show that machines with the computational power of the analog recurrent neural networks may be physically plausible [23, 24]. Note that we will assume in this paper that world is continuous: space and time is supposed to be a continuous medium. We will not discuss here this hypothesis. See [24] for a similar assumption.

The models studied in this paper are dynamical systems or hybrid systems. We call *hybrid* systems that combine discrete and continuous dynamic. Several

*Support by Esprit Project 8556, NeuroColt is acknowledged.

formal definitions have been proposed in literature: see for example [1, 9, 21]. Some undecidability results are known [1, 2, 28, 11], but only a small number of papers have been devoted to the study of hybrid systems as computational models: the work of Asarin, Maler and Pnueli [3, 4, 5] about Piecewise Constant Derivative systems and the work of Branicky [9] about simulation capabilities of Ordinary Differential Equations can however be mentioned. This paper can also be considered as a generalization of the undecidability results known about hybrid and dynamical systems. In particular, we extend the results from [3, 5, 9, 19].

In first section we introduce the notions of off-line and on-line computation by a discrete system . The computational model of analog automaton is defined. We characterize precisely its computational power as the computational power of analog recurrent neural networks [25]. Then, several notions of simulation are introduced and compared. These notions are derived and adapted from [3, 5, 9, 13]. First section is ended by a study of the computational power of iterations of piecewise linear functions: we extend the results of [13, 14, 16] and prove that the computational power of one to one piecewise linear functions is exactly the computational power of analog automata.

Section two is devoted to continuous dynamical systems. A general framework is first given in order to consider continuous systems as computational machines. The notions of computation, of discretization of a continuous system, and the notions of simulation of a discrete system by a continuous system are defined. These notions are briefly compared to the notions in literature, and some of their properties are stated. We prove then, using arguments similar to [3], that there exist some Turing machines or some analog automata that cannot be simulated by any continuous system in dimension 2.

In section three, we prove that every analog automaton can be simulated by a continuous dynamical system in dimension 3: we prove that many continuous dynamical systems (mirror systems, piecewise constant derivative systems, ordinary differential equations, and hybrid systems) do have at least the computational power of analog automata. For piecewise constant derivative systems, linear hybrid systems, and partially for Lipschitz ordinary differential equations, we also prove that they cannot have much more computational power than analog automata.

2 Discrete machines

2.1 Transition systems without input and discrete computations

Our aim is to characterize the computational power of dynamical systems. Dynamical systems do not have a straightforward notion of input: we need to define the notion of transition system without input.

Definition 2.1 (Transition system without input [3]) *A transition system without input (also called "discrete dynamical system") is a pair $A = (Q, \delta)$ where Q is a set called* space*, and δ is a subset of $Q \times Q$. If δ is a function from Q to Q, A is said to be* deterministic*.*

A transition system without input is *reversible* if its transition function is one to one. We will call *iterations of function f in dimension d* a transition system without input defined by $A = (X \subset \mathbb{R}^d, f)$. A *piecewise linear function* in dimension d, is a function defined on $X \subset \mathbb{R}^d$, where X can be partitioned in a finite number of convex closed polyhedra X_i of non empty interiors, such that f is affine on every X_i.

We now add some inputs to transition systems. We will distinguish the notions of off-line computations (the input is encoded in the initial configuration) and on-line computations (the input is given bit after bit, during the evolution of the system). The definitions in this section and in the following section are derived from [13, 14, 16].

Definition 2.2 (Off-line system) *An* off-line system *is a 5-tuple*

$$S = (Q, \delta, \phi, A, R)$$

where

- (Q, δ) *is a transition system without input.*
- $\phi : \{0,1\}^+ \to Q$ *is an encoding function.*
- $A, R \subset Q$ *are subsets of* Q*, such that* $A \cap R = \emptyset$*, called the* accepting *and* rejecting sets*.*

On an input $u \in \{0,1\}^+$*, a computation of* S *is a sequence* $(x(k))_{k \in \mathbb{N}}$ *such that* $x(0) = \phi(u)$ *and* $(x(k), x(k+1)) \in \delta$ *for all* $k \in \mathbb{N}$*.*

Call V *the subset of the* $u \in \{0,1\}^+$ *such that there exists a computation* x*, and* $k \in \mathbb{N}$*, such that* $x(k) \in A \cup R$*.*

The computation time is defined on V *as*

$$\begin{array}{rrcl} t: & V & \to & \mathbb{N} \\ & u & \mapsto & \min\{k | x \text{ is a computation on } u \\ & & & \qquad \text{and } x(k) \in A \cup R\} \end{array}$$

The function computed by S *is the partial function* $F : \{0,1\}^+ \to \{0,1\}$*, defined on* V *by, if* x *is a computation on* u *such that* $x(t(u)) \in A \cup R$*,*

- $F(u) = 1$ *if* $x(t(u)) \in A$*.*
- $F(u) = 0$ *if* $x(t(u)) \in R$*.*

The time complexity *of the computation is the function* T *such that*

$$T(n) = \max_{|u|=n} t(u)$$

where $|u|$ *stands for the length of* u*.*

Thus off-line computing consists in encoding the input into the initial configuration, and then evolving according to a transition system without input. We can now define the notion of on-line computation:

Definition 2.3 (On-line system) *An* on-line system *is a 5-tuple*

$$S = (Q, \delta, \delta_0, \delta_1, q_0, A, R)$$

where:

- (Q, δ),(Q, δ_0) *and* (Q, δ_1) *are transition systems without input.*
- $A, R \subset Q$ *are subsets of* Q*, such that* $A \cap R = \emptyset$*, called respectively the* accepting *and* rejecting sets*.*
- $q_0 \in Q$ *is called the initial state.*

On an input $u = u_0 u_1 \dots u_{|u|-1} \in \{0,1\}^+$, a computation of S is a sequence $(x(k))_{k\in\mathbb{N}}$ such that $x(0) = q_0$, $(x(k), x(k+1)) \in \delta_{u_k}$ for $0 \le k < |u|$ and $(x(k), x(k+1)) \in \delta$ for all $k \ge |u|$.

The computation time and the function computed by S are defined exactly as in definition 2.2

So on-line computing consists in starting from a fixed given state, the initial state, then evolving first according to the bits of the input, and then according to a transition system without input.

We will say that a function $F : \{0,1\}^+ \to \{0,1\}$ is *off-line computable* by a class $\mathcal{C}$ of transition systems, if F is computed by an off-line system $S = (Q, \delta, \phi, A, R)$ where $(Q, \delta) \in \mathcal{C}$. We will say that a function $F : \{0,1\}^+ \to \{0,1\}$ is *on-line computable* by a class $\mathcal{C}$ of transition systems, if F is computed by an on-line system $S = (Q, \delta, \delta_0, \delta_1, q_0, A, R)$ where $(Q, \delta), (Q, \delta_0), (Q, \delta_1) \in \mathcal{C}$.

2.2 Analog automata

We propose a new model of computation: an analog two stack automaton is similar to a usual two stack automaton with the only difference that it is able to change the whole content of one of its stack in constant time 1.

Definition 2.4 (Analog automaton) *A deterministic analog (two stack) automaton is a system*

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- *Q is a finite set of states.*
- *Σ is an alphabet.*
- *$q_0 \in Q$ is the initial state.*
- *$F \subset Q$ is the set of final states.*
- *δ is a mapping from $Q \times (\Sigma \cup \{\epsilon\})^2$ to $Q \times \{Nop, Pop, \{Push\} \times \Sigma, \{Advice\} \times \Sigma^{\#}\}^2$ where $\Sigma^{\#} = \Sigma^* \cup \Sigma^\omega$ is the set of words with finite or infinite length.*

An *instantaneous description* (ID) of an analog automaton is a 3-tuple (q, γ_1, γ_2) where $q \in Q$ is called *the state* of the automaton, and $\gamma_1, \gamma_2 \in \Sigma^{\#}$ are called *the contents of the stacks*. We define the following relation $\vdash$ between IDs:

for $q \in Q$, $a_1, a_2 \in \Sigma, \gamma_1, \gamma_2, \gamma_1', \gamma_2' \in \Sigma^{\#}$,

with convention that if $a_i = \epsilon$ then $\gamma_i = \epsilon$,

$$(q, a_1\gamma_1, a_2\gamma_2) \vdash (q', \gamma_1', \gamma_2')$$

if, for $i \in \{1,2\}$, $q' = \delta_1(q, a_1, a_2)$, and

$$\gamma_i' = \begin{cases} \gamma_i & if\ \delta_{i+1}(q, a_1, a_2) & = Pop \\ a_i\gamma_i & & = Nop \\ ca_i\gamma_i & & = (Push, c) \\ w & & = (Advice, w) \end{cases}$$

We define the relation $\vdash^*$ as the transitive closure of $\vdash$. We say, when

$$\delta_{i+1}(q, a_1, a_2) = (Advice, w)$$

that M *uses* w or *makes advice* w *appear on stack* i. The language $L(M)$ accepted by M is defined by

$$L(M) = \{w \in \{0,1\}^+ | (q_0, w, \epsilon) \vdash^* (p, \gamma_1, \gamma_2) \wedge p \in F\}$$

The notion of *non-deterministic two stack automaton* is defined in a similar way. We shall call *discrete two stack automaton* the usual notion of two stack automaton: that is, a discrete two stack automaton is an analog automaton which never uses any advice. Any analog automaton (or discrete two stack automaton) M will also be considered as a transition system without input as $M = (Q \times \Sigma^\# \times \Sigma^\#, \vdash)$. Because a discrete two stack automaton is an analog two stack automaton, and since discrete two stack automata can simulate Turing machines [10], analog automata are able to simulate Turing machines. The exact computational power of analog automata is given by the following theorem (for the definition of the complexity classes $P/poly$ and $NP/poly$, see [6]):

Theorem 2.1

- *Every language $L \subset \{0,1\}^+$ can be recognized by a deterministic analog two stack automaton in exponential time.*
- *The languages $L \subset \{0,1\}^+$ accepted by deterministic (respectively: non–deterministic) analog two stack automata in polynomial time are exactly the languages belonging to the complexity class $P/poly$ (resp: $NP/poly$).*

Proof: We shall only detail the deterministic case:

- Let $L \subset \{0,1\}^+$ be a language. Let the word γ, of possibly infinite length, 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.
- 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 $f : \mathbb{N} \rightarrow \{0,1\}^+$ (see [6]). 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 $p(1) + p(2) + \ldots + p(|w|)$ 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.

□

Therefore, we have shown that the computational power of analog automata is exactly the computational power of recurrent analog neural networks: see [25]. It is well known [10] that there exist some languages $L \subset \{0,1\}^+$ which cannot be recognized by Turing machines. Since, from theorem 2.1, L can be recognized by an analog automaton, we conclude that the analog two stack automata do have super-Turing capabilities.

2.3 Simulation notions between discrete systems

In this section, we define several notions of simulation between discrete systems. We shall compare these notions later. The notion of simulation used in [13, 14, 16] is the following:

Definition 2.5 (K-simulation) *Let $A_1 = (Q_1, \delta_1)$ and $A_2 = (Q_2, \delta_2)$ be two transition systems without input. Let $D \subset Q_2$ be stable by δ_2 (that is $\delta_2(D) \subset D$) and Φ an onto function from D to Q_1. A_2* K-simulates *A_1 via Φ if*

$$\forall q_1, q_2 \in D, (q_1, q_2) \in \delta_2 \Leftrightarrow (\Phi(q_1), \Phi(q_2)) \in \delta_1$$

That is, A_2 *K-simulates* A_1 if there exists a sub-system of system A_2 which is identical to A_1, modulo Φ. We define the notion of trajectory of a transition system *cutting* a subset:

Definition 2.6 *Let $A = (Q, \delta)$ be a transition system without input.*

- *There is a trajectory $\mathcal{T}$ from x to x' of* real length *$i \in \mathbb{N}$ and* virtual length 1 cutting *$Y \subset Q$, if there exists a i-tuple $(x = x_0, x_1, x_2, \ldots, x_i = x')$ such that*
$$\begin{cases} i) & \forall 0 \leq j < i, (x_j, x_{j+1}) \in \delta \\ ii) & \forall 0 < j < i, x_j \notin Y \\ ii) & x, x' \in Y \end{cases}$$

- *There is a* trajectory *$\mathcal{T}$ from x to x' of* real length *$i \in \mathbb{N}$ and* virtual length *$j \in \mathbb{N}$ cutting Y if there exists a j-tuple $(x = x_0, x_1, x_2, \ldots, x_j = x')$ such that, for all $k \in \{1, 2, \ldots, j\}$, there exists a trajectory cutting Y of real length i_k and of virtual length 1 from x_{k-1} to x_k where $i = i_1 + i_2 + \ldots + i_j$.*

- *We will denote $length_{real}(\mathcal{T}) = i$ and $length_{virt}(\mathcal{T}) = j$.*

That allows us to define the notion of Q-simulation (inspired from [5]): we extend the notion of K-simulation by the possibility that a transition of system A_1 can be realized by several transitions of system A_2.

Definition 2.7 (Q_0/Q simulation)

- *Let $A_1 = (Q_1, \delta_1)$ and $A_2 = (Q_2, \delta_2)$ be two transition systems without input. Let $Q_0 \subset Q_1$.*

  A_2 Q_0-simulates *A_1 via Φ if there exists $Y \subset Q_2$ such that Φ is an onto function from Y to Q_0, where, for all $x, x' \in Y$, there exists a trajectory $\mathcal{T}'$ from x to x' in A_2 cutting Y if and only if there exists a trajectory $\mathcal{T}$ from $\Phi(x)$ to $\Phi(x')$ in A_1 cutting Q_0.*

- *When $Q_0 = Q_1$, we say that A_2* Q-simulates *A_1 via Φ.*

  If when $length_{virt}(\mathcal{T}) = i \in \mathbb{N}$ then $length_{real}(\mathcal{T}') = \Delta i$, for some constant Δ, we say that the simulation is in real time *Δ, or* in linear time.

  If when $length_{virt}(\mathcal{T}) = i \in \mathbb{N}$ then $length_{real}(\mathcal{T}') = O(p(i))$, for a given polynomial p, we say that the simulation is in polynomial time.

Hence, K-simulation is identical to Q-simulation in real time 1. In [3], the authors use a different notion: the notion of abstraction. Let us start by defining the abstraction of a trajectory σ, via a function φ, as the sequence of the images of the trajectory by φ. Formally:

Definition 2.8 ([3]) *Let $A = (Q, \delta)$ be a transition system without input.*

- *Let $q \in Q$. We denote $L(A,q)$ the set of the trajectories of A starting from q: that is the sequences $(q_0, q_1, \ldots, q_k, \ldots)$, with $q = q_0$, such that $(q_k, q_{k+1}) \in \delta$, for all $k \in \mathbb{N}$.*

- *Let $\sigma \in L(A,q)$. We denote $\sigma = (q_0, q_1, q_2, \ldots)$. Let φ be a function from Q to a set Q', onto, possibly partial. In a point $x \in Q$, where φ is not defined, we will write $\varphi(x) = \perp$. We say that φ is a* state abstraction *function from Q to Q'. We denote $\varphi(\sigma)$ the sequence $(q'_0, q'_1, q'_2, \ldots)$, where $q'_i = \varphi(q_{j_i})$, with for all $i \geq 1, j_i = \min\{j | j > j_{i-1} \wedge \varphi(q_j) \neq \perp\}$ and $j_0 = 0$.*

From these definitions we get the notion of abstraction between transition systems:

Definition 2.9 (Abstraction [3]) *Let $A_1 = (Q_1, \delta_1)$ and $A_2 = (Q_2, \delta_2)$ be two transition systems without input. Let φ be a state abstraction function from Q_2 to Q_1.*

We say that A_1 is an (φ-) abstraction *of A_2 via φ, or A_2* φ-realizes *A_1, denoted by $A_1 \leq_\varphi A_2$, if:*

$$\forall x \in Q_1, \forall y \in \varphi^{-1}(x), \quad \sigma \in L(A_2, y) \Rightarrow \varphi(\sigma) \in L(A_1, x) \tag{1}$$

$$\forall x \in Q_1, \forall \sigma_1 \in L(A_1, x), \exists y \in Q_2, \exists \sigma_2 \in L(A_2, y) \quad \sigma_1 = \varphi(\sigma_2) \tag{2}$$

That means that, A_1 is a φ-abstraction of A_2, if the set of the trajectories of A_1 is exactly the set of the abstractions of the trajectories of A_2, for the state abstraction function φ. We define the notion of simulation between classes of systems, for a given notion of simulation, by:

Definition 2.10 *Let $\mathcal{C}$ and $\mathcal{C}'$ be two classes of transition systems without input. We say that $\mathcal{C}'$ simulates $\mathcal{C}$, if for all system $\mathcal{S} \in \mathcal{C}$, there exists a system $\mathcal{S}' \in \mathcal{C}'$ such that $\mathcal{S}'$ simulates $\mathcal{S}$.*

2.4 Properties

We study now the links between the different notions of simulation:

Theorem 2.2

- *All the previous notions of simulation are reflexive and transitive*

- *Let $A_1 = (Q_1, \delta_1)$ and $A_2 = (Q_2, \delta_2)$ be two transition systems without input. The following implications are true:*

  A_2 K-simulates A_1 via $\varphi \Rightarrow A_2$ Q-simulates A_1 via $\varphi \Rightarrow A_1 \leq_\varphi A_2$

- *Assume that A_2 K-simulates (respectively: Q-simulates) A_1, and A_2 is deterministic, then A_1 is deterministic.*

- *We have the following relations between the computational models:*

  - *The non-deterministic analog automata K-simulate the deterministic analog automata and the non–deterministic discrete two stack automata.*
  - *The deterministic analog automata K-simulate the discrete deterministic two stack automata.*
  - *The non-deterministic discrete two stack automata K-simulate the deterministic discrete two stack automata and the non-deterministic finite state automata.*

– *The deterministic discrete two stack automata φ-realize the non–deter--ministic finite state automata.*

– *The non-deterministic finite state automata K-simulate the deterministic finite state automata.*

Proof: All the results are straightforward from the definitions. The only intricate point is that the discrete deterministic two stack automata φ-realize the non-deterministic finite state automata. This fact was already mentioned in [3]: let $A = (Q, \delta)$ be a non-deterministic finite state automaton. Let $d = \max_{q \in Q} |\{v/(q, v) \in \delta\}|$ be the maximum of the outgoing degrees of the vertices of the graph $G = (Q, \delta)$. In every state $q \in Q$, we call $e_{q,1}, e_{q,2}, \ldots, e_{q,n_q}$ the outgoing edges starting from q in G. Note that, by definition of d, necessarily, $n_q \leq d$. We construct $A' = (Q' = Q \times \Sigma^* \times \Sigma^*, \delta')$ as a deterministic discrete two stack automaton, with stack alphabet Σ defined by $\Sigma = \{1, 2, \ldots, d\}$. We define the transition function δ' of A' such that, in a state q, when A' reads symbol $s \in \Sigma$ on the top of its first stack, A' pops s, and makes a transition to state q', where $e_{q,s} = (q, q')$. It can be checked that A' φ-realizes A, via the function φ defined on every $q' = (q, \gamma_1, \gamma_2) \in Q'$ as $\varphi(q') = q$.

□

We can go further and precise the relations between the notions of simulation by:

Theorem 2.3 • *The notion of Q-simulation is strictly more powerful than the notion of K-simulation.*

• *The notion of abstraction is strictly more powerful than the other notions.*

Proof: It is easy to construct a transition system A_2 that simulates every step of a transition system A_1 by two steps. A_2 Q-simulates A_1 but A_2 does not K-simulate A_1. So the first point is straightforward.

The deterministic discrete two stack automata φ-realize the non-deterministic finite state automata from previous theorem, but the deterministic discrete two stack automata cannot Q-simulate or K-simulate the non-deterministic finite state automata from theorem 2.2.

□

The proof of the previous theorem shows that the notion of abstraction is very interesting, because this notion, unlike the other notions, allows non-deterministic systems to be simulated by deterministic systems. We will need the following result:

Theorem 2.4 *Every deterministic (respectively: non-deterministic) analog two stack automaton M can be Q-simulated in polynomial time by a deterministic (resp: non-deterministic) reversible analog two stack automaton M'.*

Proof:

We only give a sketch of the proof here. Let Σ be the stack alphabet of M. We will write every word $\alpha \in \Sigma^{\#}$ as an infinite sequence $a_1 a_2 \ldots a_n \ldots \in \Sigma^\omega$ with $a_k = \epsilon$, for all $k > |\alpha|$. Let $\alpha_0, \alpha_1, \ldots$ and α_{p-1} be p words of Σ^ω. For $i \in [0, p-1]$, we can write $\alpha_i = a_{i,0} a_{i,1} a_{i,2} \ldots a_{i,n} \ldots \in \Sigma^\omega$. We define the *mix* operation as $mix(\alpha_0, \alpha_1 \ldots, \alpha_{p-1}) = b_1 b_2 \ldots b_n, \ldots \in \Sigma^\omega$, where for all $j > 0$, $b_j = a_{imodp, idivp}$, where *div* is the integer division, and *mod* is the remainder of the integer division.

Let $\beta_1, \beta_2, \ldots, \beta_q$ be the q different advices that analog automaton M can possibly use in a computation. Call $\beta = mix(\beta_1, \beta_2, \ldots, \beta_q)$. At any time, let $\gamma_1 \in \Sigma^{\#}$ (respectively: $\gamma_2 \in \Sigma^{\#}$) be the content of the first (resp: the second) stack of M. Call $\gamma = mix(\gamma_1, \gamma_2)$. $M = (Q_M, \delta_M)$ is Q-simulated by $M' = (Q_{M'}, \delta_{M'})$ via φ, where M' and φ are built as follows: at any time M' keeps the simulated values of the contents of the two stacks of M by keeping γ in its first stack. That, is at any time, the state $(q', \gamma_1', \gamma_2') \in Q_{M'}$ of M' is such that, there exists $w \in \Sigma^*$ with $\gamma_1' = w\gamma$. Before simulating any step of M, M' makes advice β appear on its empty second stack, and keeps this value in its second stack: that is, at any time, there exists $w' \in \Sigma^*$, such that $\gamma_2' = w'\beta$. M' is built by simulating M on $\gamma = mix(\gamma_1, \gamma_2)$. It can be checked that M' is able to simulate all the operations of M on γ. If M tries to read a character in one of its advice, M' can simulate the operation by reading the characters of β. The reader can check that it is possible, using this way, to get an analog automaton M' that Q-simulates M in polynomial time.

Now remark that the advice β appears only in the first step of any simulation of M by M', appears only on the second stack of M' and only on an empty stack. If we except the first step that makes advice β appear, M' is a discrete two stack automaton, that is a Turing machine. Since we know that a Turing machine can always be simulated, modulo a polynomial time overhead, by a reversible one (see for example: [7]), we claim that M', from second step, can be built reversible. It can be checked that the first step (the apparition of advice β on the empty second stack of M') is reversible, and that the second step (that is the beginning of the reversible process of "Turing machine" M' on γ and β) is only reachable by the first step. Thus M' is reversible at any step and Q-simulates M in polynomial time.

□

We will also need the following result:

Lemma 2.1 • *Let $F : \{0,1\}^+ \to \{0,1\}$ be computed in polynomial time by an off-line system $S = (X, f, \phi, A, F)$.*

Suppose that:

- *X is a compact subset of $\mathbb{R}^n$.*
- *$f \in LP_d/poly$: that is, f is Lipschitz, and $f : X \to X$ can be approximated in polynomial time by a Turing machine with advice: see [13, 14].*
- *ϕ is in $PE_d/poly$, that is $\phi : \{0,1\}^+ \to X$ can be approximated in polynomial time by a Turing machine with advice: see [13, 14].*
- *A, R are convex polyhedra of $\mathbb{R}^n$.*

Then $F \in P/poly$.

• *Let $F : \{0,1\}^+ \to \{0,1\}$ be computed in polynomial time by an on-line system $S = (X, f, f_0, f_1, q_0, A, R)$.*

Suppose that:

- *X is a compact subset of $\mathbb{R}^n$.*
- *$f, f_0, f_1 \in LP_d/poly$.*
- *A, R are convex polyhedra of $\mathbb{R}^n$.*

Then $F \in P/poly$.

Proof: This is an easy generalization of [13, 14].

□

2.5 Computational power of piecewise linear functions

We study now the computational power of iterations of piecewise linear functions. The results are extensions of [13, 14, 16]. We prove in this section that it is possible to use one to one functions. First we need the following definition:

Definition 2.11 (Disconnected piecewise linear function) *A function f is called* disconnected piecewise linear *with real coefficients (respectively: rational coefficients) if, for some $n \in \mathbb{N}$,*

1. *there exist n closed intervals $I_i = [a_i, b_i]$, with $a_i, b_i \in \mathbb{R}$ (resp: $a_i, b_i \in \mathbb{Q}$).*
2. *f can be written*

$$f : C = \bigcup_{i,j \in \{1,2,\ldots,n\}} C_{i,j} \subset [0.1]^2 \to [0,1]^2$$

where, for all $i,j \in \{1,2,\ldots,n\}$, $C_{i,j}$ is defined as $C_{i,j} = I_i \times I_j$.

3. *all the I_i are at a strictly positive distance: there exists ϵ such that, for all $i \neq j$, $x \in I_i, y \in I_j \Rightarrow d(x,y) \geq \epsilon$.*
4. *on each $C_{i,j}$, f is affine of type $f(x_1,x_2) = (\alpha_{i,j,1} + \beta_{i,j,1}x_1, \alpha_{i,j,2} + \beta_{i,j,2}x_2)$, where $\alpha_{i,j,1}, \alpha_{i,j,2}, \beta_{i,j,1}$ and $\beta_{i,j,2}$ are real (resp: rational) positive constants.*

Our main results come as:

Theorem 2.5
- *Every deterministic (respectively: reversible) discrete two stack automaton M can be K-simulated by iterations of a disconnected (resp: one to one) piecewise linear function $f : C \subset [0,1]^2 \to [0,1]^2$ with rational coefficients.*
- *Every deterministic (respectively: reversible) analog two stack automaton M can be K-simulated by iterations of a disconnected (resp: one to one) piecewise linear function $f : C \subset [0,1]^2 \to [0,1]^2$ with real coefficients.*

Proof:

We will only detail here the case of an analog automaton M being simulated by iterations of a disconnected piecewise linear function with real coefficients. To get the case of a discrete two stack automaton M being simulated by iterations of a disconnected piecewise linear function with rational coefficients, just consider M as an analog automaton which does not make any advice appear: the proof gives then a function with rational coefficients, instead of real coefficients.

We can suppose w.l.o.g. that the state set of M is $Q = \{1,3\}^{p_1} \times \{1,3\}^{p_2}$, and that the letters of Σ, the stack alphabet of M, are encoded onto the alphabet $\{1,3\}$. Let $p = \lceil log_2|\Sigma| \rceil$ be the number of bits needed to encode each letter of Σ.

Each ID (q, γ_1, γ_2) of M is encoded in the radix-4 expansion of a point (x_1, x_2) of $[0,1]^2$ where, 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 $\gamma_i \in \Sigma^{\#}$ can be written on alphabet $\{1,3\}$ as $\gamma_i = s_{i,1}, s_{i,2}, \ldots, s_{i,n}, \ldots$,

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

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 $l_i = \overline{0.q_{i,1}q_{i,2}, \ldots, q_{i,p_i}, s_{i,1}, s_{i,2}, \ldots, s_{i,p}}$
- 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 $s_{i,j}$ 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 $s_{i,1}, s_{i,2}, \ldots, s_{i,p} = 0$.

Let

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

Function f will be defined as piecewise linear on C, and the $(I_{j,l_j})_{j \in \{1,2\}, l_j}$ will play the role of the $(I_i)_{i \in [1 \ldots n]}$ of definition 2.11.

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

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

Write q, a_1 and a_2 as $q = q_{1,1}, \ldots, q_{1,p_1}, q_{2,1}, \ldots, q_{2,p_2}$, $a_1 = s_{1,1}, \ldots, s_{1,p}$ and $a_2 = s_{2,1}, \ldots, s_{2,p}$.

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

$$q'_{1,1}, q'_{1,2} \ldots, q'_{1,p_1} q'_{2,1}, q'_{2,2}, \ldots, q'_{2,p_2} = \delta_1(q, a_1, a_2)$$

and $\Delta x'_i$ defined by:

- $\Delta x'_i = 4^p \Delta x_i$ if $\delta_{i+1}(q, a_1, a_2) = Pop$
- $\Delta x'_i = \frac{s_{i,1}}{4^{p_i+1}} + \ldots + \frac{s_{i,p}}{4^{p_i+p}} + \Delta x_i$ if $\delta_{i+1}(q, a_1, a_2) = Nop$
- $\Delta x'_i = \frac{c_{i,1}}{4^{p_i+1}} + \ldots + \frac{c_{i,p}}{4^{p_i+p}} + \frac{s_{i,1}}{4^{p_i+p+1}} + \ldots + \frac{s_{i,p}}{4^{p_i+2*p}} + \frac{\Delta x_i}{4^p}$ if $\delta_{i+1}(q, a_1, a_2) = (Push, c = c_{i,1}, \ldots, c_{i,p})$
- $\Delta x'_i = \frac{b_1}{4^{p_i+1}} + \frac{b_2}{4^{p_i+2}} \ldots + \frac{b_n}{4^{p_i+n}} + \ldots$ if $\delta_{i+1}(q, a_1, a_2) = (Advice, \gamma = b_1 b_2 \ldots b_n \ldots)$

It can be checked that, in any case, f is built such that $f(x_1, x_2)$ encodes the ID of M at time $t+1$: that is encodes ID $(q', \gamma'_1, \gamma'_2)$ where $(q, a_1\gamma_1, a_2\gamma_2) \vdash (q', \gamma'_1, \gamma'_2)$. So M is K-simulated by the iterations of function f. Function f is a disconnected piecewise linear function with real coefficients, and the result for non (necessarily) reversible analog automata follows.

Suppose now that analog automaton M is reversible: we prove that, in this case, function f is one to one on C. Assume that there exist $x = (x_1, x_2) \in C$ and $y = (y_1, y_2) \in C$ such that $f(x) = f(y)$. We want to prove that $x = y$.

We need to define a Mod operator: let $r \in \mathbb{N}$. Let $z \in [0, 1/4^r[$. Assume that z has a finite radix-4 expansion. We can write this unique finite expansion as $z = 4^{-r}\overline{0.\alpha_1\alpha_2\alpha_3 \ldots \alpha_k}$, where $\alpha_k \neq 0$ and $k \in \mathbb{N}$. Suppose that z does not have any finite expansion: in this case, we write the unique infinite expansion of z as $z = 4^{-r}\overline{0.\alpha_1\alpha_2\alpha_3 \ldots}$ and we take $k = \infty$.

In any case, we define the Mod operator on z as $Mod_r(z) = 4^{-r}\overline{0.\alpha'_1\alpha'_2\alpha'_3 \ldots}$, where, for all $1 \leq j \leq k$,

$$\alpha'_j = \begin{cases} 1 & if \quad \alpha_j = 0 \quad or \quad \alpha_j = 1 \\ 3 & if \quad \alpha_j = 2 \quad or \quad \alpha_j = 3 \end{cases}$$

From now, we will denote by an x exponent the definitions relative to x, and by an y exponent the definitions relative to y. We will only deal with x in the definitions. Definitions relative to y are to be understood in a similar way.

There exists l_1^x, l_2^x, where $l_i^x = \overline{0.q_{i,1}^x, q_{i,2}^x, \ldots, q_{i,p_i}^x, s_{i,1}^x, \ldots, s_{i,p}^x}$, $i \in \{1,2\}$, such that $x \in I^x = I_{1,l_1^x} \times I_{2,l_2^x}$.

Let $\Delta x_i = x_i - l_i^x$. We have $0 \leq \Delta x_i < 1/4^{p_i+p}$.

Let $q^x = q_{1,1}^x, q_{1,2}^x, \ldots, q_{1,p_1}^x, q_{2,1}^x, q_{2,2}^x, \ldots, q_{2,p_2}^x$.

Let $s_i^x = s_{i,1}^x, s_{i,2}^x, \ldots, s_{i,p}^x$, for $i \in \{1,2\}$.

Let $f(x) = (x_1', x_2')$.

So, if x is corresponding to a valid encoding of an ID $(q, a_1\gamma_1, a_2\gamma_2)$ of the analog automaton M, with $q \in Q, a_1, a_2 \in \Sigma$, then q^x, s_1^x, s_2^x are respectively such that $q^x = q$,$s_1^x = a_1$ and $s_2^x = a_2$.

Let $q'^x = q'^x_{1,1}, q'^x_{1,2}, \ldots, q'^x_{1,p_1}, q'^x_{2,1}, q'^x_{2,2}, \ldots, q'^x_{2,p_2} = \delta_1(q^x, s_1^x, s_2^x)$ and $l_i'^x = \overline{0.q'^x_{i,1}, q'^x_{i,2}, \ldots, q'^x_{i,p_i}}$.

By the definition of f, we have $x_i' = l_i'^x + \Delta x_i'$ where $0 \leq \Delta x_i' < 1/4^{p_i}$ and $y_i' = l_i'^y + \Delta y_i'$ where $0 \leq \Delta y_i' < 1/4^{p_i}$. From $f(x) = f(y)$ we get

$$l_i'^x = l_i'^y \tag{3}$$

$$\Delta x_i' = \Delta y_i' \tag{4}$$

Define $\overline{x} = (\overline{x_1}, \overline{x_2})$ as, for $i \in \{1,2\}$, $\overline{x_i} = l_i^x + Mod_{p_i+p}(\Delta x_i)$,

Since we do not change any digit of the radix 4 expansion before the $p+p_i+1$th digit, we have $\overline{x} \in I^x$. Let $f(\overline{x}) = \overline{x}' = (\overline{x_1}', \overline{x_2}')$. We know that f is linear on I^x. By studying the different possibilities, it can be checked that in any case

$$\overline{x_i}' = l_i'^x + Mod_{p_i}(\Delta x_i') \tag{5}$$

We define in a similar way $\overline{y} = (\overline{y_1}, \overline{y_2})$, where, for $i \in \{1,2\}$, $\overline{y_i} = l_i^y + Mod_{p_i+p}(\Delta y_i)$. Let $f(\overline{y}) = \overline{y}' = (\overline{y_1}', \overline{y_2}')$. We get similarly

$$\overline{y_i}' = l_i'^y + Mod_{p_i}(\Delta y_i') \tag{6}$$

From (3),(4),(5) and (6) we get, for $i \in \{1,2\}$, $\overline{x_i}' = \overline{y_i}'$.

So we have $f(\overline{x}) = f(\overline{y})$. Now, it can be seen that $\overline{x}$ and $\overline{y}$ are encoding valid IDs. Call $ID_{\overline{x}}$, and $ID_{\overline{y}}$ the IDs encoded by respectively $\overline{x}$ and $\overline{y}$. Since f K-simulates M, we get that $f(\overline{x})$ encodes ID', where ID' is given by $ID_{\overline{x}} \vdash ID'$. Similarly, we get that $f(\overline{y})$ encodes also ID', with $ID_{\overline{y}} \vdash ID'$. From the fact that M is reversible, we get $ID_{\overline{x}} = ID_{\overline{y}}$. Thus, we get also necessarily $I^x = I^y$. Now, f is defined as a one to one linear function on every $I = I_{1,l_1} \times I_{2,l_2}$. Thus we obtain $x = y$, and that f is one to one.

$\square$

Note that,

- a disconnected piecewise linear function $f : C \to [0,1]^2$ with rational (respectively: real) coefficients can be completed for example by triangulation, to a piecewise linear continuous function $f : [0,1]^2 \to [0,1]^2$ with rational (respectively: real) coefficients.

- a disconnected one to one piecewise linear function $f : C \to [0,1]^2$ with rational (respectively: real) coefficients can be completed to a one to one piecewise linear function $f : [0,1]^2 \to [0,1]^2$ with rational (resp: real) coefficients.

So in theorem 2.5, all the results can be stated with continuous or one to one piecewise linear functions on all $[0,1]^2$ instead of disconnected piecewise linear functions defined only on $C \subset [0,1]^2$.

We now give some technical considerations about the one to one disconnected piecewise linear functions f given by theorem 2.5, in the case of a reversible analog (or discrete) two stack automaton M. We use the notations of definition 2.11.

From theorem 2.5 we know that f is one to one on C. For $i,j \in [1\dots n]$, call $C'_{i,j} = f(C_{i,j})$. Since f is one to one, we have necessarily:

$$(i,j) \neq (i',j') \Rightarrow C'_{i,j} \cap C'_{i',j'} = \emptyset \tag{7}$$

Let $i,j \in [1\dots n]$. We have $C_{i,j} = I_i \times I_j$, with $I_i = [a_i, b_i]$ and $I_j = [a_j, b_j]$. Call the boundaries as c_1, c_2, d_1, d_2 with $c_1 = a_i, c_2 = a_j, d_1 = b_i, d_2 = b_j$, such that $C_{i,j} = [c_1, d_1] \times [c_2, d_2]$.

On $C_{i,j}$, f can be written $f(x_1, x_2) = (\alpha_{i,j,1} + \beta_{i,j,1} x_1, \alpha_{i,j,2} + \beta_{i,j,2} x_2)$. Let $l \in \{1,2\}$. We know that the constants $\alpha_{i,j,l}, \beta_{i,j,l}$ are positive. Since f is one to one on C, we get two possible cases:

- either $\beta_{i,j,l}$ is strictly positive.
- either $\beta_{i,j,l} = 0$ and $c_l = d_l$

The interest of these remarks will appear later in this paper.

With Theorem 2.5, we are able to generalize all the results of [13, 14, 16] to one to one piecewise linear functions. Thus, we get:

Theorem 2.6 *Every function $F : \{0,1\}^+ \to \{0,1\}$ can be off-line computed by iterations of a function $f : I^1 = [0,1] \to I^1$, one to one, piecewise linear, with real positive coefficients in dimension 1 in exponential time.*

Moreover, the encoding function is computable by Turing Machine (that is in PE_1: see [13, 14]), is one to one, and independent from F. The accepting and rejecting sets are also independent of F, and defined as intervals with rational boundaries.

Proof: Nothing to do, but say that the functions used in [13, 14] are one to one functions.

□

We also get:

Theorem 2.7 *Every function $F : \{0,1\}^+ \to \{0,1\}$ in P (respectively: in $P/poly$)*

- *can be off-line computed, in polynomial time, by iterations of an one to one piecewise linear function, with rational (resp: real) positive coefficients, in dimension 2.*

  The encoding function is computable by a Turing machine (that is in PE_2: [13, 14].).

- *can be on-line computed, in polynomial time, by iterations of an one to one piecewise linear function, with rational (resp: real) positive coefficients in dimension 2.*

  The "encoding functions" are one to one, piecewise linear in dimension 2, with rational positive coefficients.

Proof:

- Let M be a reversible discrete (resp: analog) two stack automaton that recognizes F. From theorem 2.5, we know that M is K-simulated by the iterations of a piecewise one to one linear function f, via a function Φ. Function F is computed by the off-line system $S = ([0,1]^2, f, \Phi, A, R)$ where A, R are the subsets of $[0,1]^2$ that encode respectively the accepting and rejecting configurations of M. Moreover, it can be checked that Φ is in PE_2: see [13, 14].

- Let M_0 (respectively: M_1) be the reversible discrete two stack automaton that, on every step, pushes systematically 0 (resp: 1) on its first stack, and leaves its second stack unchanged. Let M be a reversible discrete (resp: analog) two stack automaton that recognizes F^r, where $F^r(a_1, a_2, \dots a_n) = 1$ if and only if $F(a_n, \dots, a_2, a_1) = 1$. From theorem 2.5, we know that there exist f_0, f_1 and f that K-simulate M_0, M_1 and M, via the functions Φ_0, Φ_1 and Φ respectively. It can be checked that, if the state sets of M_0 and M_1 are chosen to be the same as the state set of M, then functions Φ_0, Φ_1 and Φ are identical. We claim then, that function F is computed by the on-line system $S = ([0,1]^2, f, f_0, f_1, q, A, R)$, where A, R are the subsets of $[0,1]^2$ of points that encode the accepting and rejecting configurations of M, and $q_0 \in [0,1]^2$ is the encoding of the initial state of M.

□

Actually, we can give an upper bound to the computational power of iterations of piecewise linear functions, using results from [12]:

Theorem 2.8

- *Let $F : \{0,1\}^+ \to \{0,1\}$ be a function off-line computed by iterations of a piecewise linear function f in dimension d: that is by an off-line system $S = (X, f, \phi, A, R)$, where $X \subset \mathbb{R}^d$. Assume that:*
  - *ϕ is computable by a linear machine: there exists a linear machine (restriction of the BSS machine [8] which is only allowed to compute linear operations: i.e. which is not allowed to compute multiplications between its variables [12]) that is able, given $w \in \{0,1\}^+$, to return the real number $\phi(w)$.*
  - *$A, R \subset X$ are convex polyhedra.*
  - *F is computed in polynomial time.*

  Then F is in $P/poly$.
- *Let $F : \{0,1\}^+ \to \{0,1\}$ be a function on-line computed by iterations of piecewise linear functions f_0, f_1, f in dimension d: that is, by an on-line system $S = (X, f_0, f_1, f, q_0, A, R)$, where $X \subset \mathbb{R}^d$. Assume that:*
  - *$A, R \subset X$ are convex polyhedra.*
  - *F is computed in polynomial time.*

  Then F is in $P/poly$.

Proof: The hypotheses of the theorem are chosen so that, in any case, it is possible to construct a linear machine [12] M that simulates the evolution of S. From the fact that the computational power of linear machines with discrete inputs is bounded by $P/poly$ (see: [12]), we get that $F \in P/poly$.

□

As a conclusion, from the two previous theorems, and from the fact that the iterations of piecewise linear functions with rational coefficients can be simulated by some Turing machines, we get that the computational power of iterations of one to one piecewise linear functions with rational (respectively: real) coefficients from $\mathbb{R}^p$ to $\mathbb{R}^p$, for $p \geq 2$, is exactly

- P (resp: $P/poly$) in polynomial time.
- EXP (resp: unbounded) in exponential time.

3 Continuous dynamical systems

3.1 Continuous systems

The continuous dynamical systems that we shall study can be formalized by:

Definition 3.1 (Continuous system) • *A* continuous system *is a pair $H = (X, F)$ where,*

- *X is a set, called* space.
- *F is a set of functions $f : \mathbb{R} \to X$.*
- *$\forall t \in \mathbb{R}, \forall f \in F$, $(f + t) \in F$, where $(f + t) : \mathbb{R} \to X$ is defined for all $t' \in \mathbb{R}$, by $(f + t)(t') = f(t + t')$.*

- *A* trajectory *of H starting from $x \in X$ is a function $f \in F$ such that $f(0) = x$.*
- *There is a trajectory* of time-length t *between x and x' if there exists a function $f \in F$ such that $f(0) = x$ and $f(t) = x'$.*
- *If, for all $x \in X$, there is exactly one trajectory starting from x, the continuous system is said to be* deterministic.

The continuous systems $H = (X, F)$ that we shall study in this paper are all such that there exists an integer p, such that $X \subset \mathbb{R}^p$. We will call integer p the *dimension* of H. Note that continuous deterministic systems can be defined in an equivalent way using a flow:

Proposition 3.1 *A continuous system $H = (X, F)$ is deterministic if and only if*

$$\exists \varphi : X \times \mathbb{R}^+ \to X$$

such that:

$$\left\{ \begin{array}{ll} i) & \varphi(x)(0) = x \\ ii) & \forall t, t' \in \mathbb{R}^+, \forall x \in X, \varphi(x)(t + t') = \varphi(\varphi(x)(t))(t') \\ iii) & F = \{\varphi(x)(.) | x \in X\} \end{array} \right.$$

Hence, definition 3.1 is more general that the flow formalization of continuous systems, since non-deterministic continuous systems can also be defined. We propose some definitions in order to compare the models for continuous systems:

Definition 3.2 (Differential system) *A continuous system $H = (X, F)$ is* differential *if F is defined as the set of the solutions of a given ordinary differential equation.*

Definition 3.3 (System with continuous trajectories) *A continuous system $H = (X, F)$, if X is a topological space, is a* system with continuous trajectories *if, for all $f \in F$, $f : \mathbb{R} \to X$ is a continuous function.*

3.2 Discretizations

In order to compare continuous systems to discrete systems, we will need to discretize them. In that purpose, we define the notion of state abstraction:

Definition 3.4 • *Let $H = (X, F)$ be a continuous system. Let φ be an onto partial function from X to a set Q. Function φ is called a* state abstraction *for H to Q. In a point x, where φ is not defined, we will denote $\varphi(x) = \perp$.*

- *Let H be a continuous system and φ a state abstraction. Let $f \in F$ be a trajectory such that $f(0) = x$. We call* φ-signature *[3], or* abstraction *of f, the sequence $(q_1, q_2, \ldots, q_n, \ldots)$ of the values of $\varphi(f(t))$, when t describes $\mathbb{R}^+$. Formally, there exist two sequences $(l_i)_{i\in\mathbb{N}}$, $(u_i)_{i\in\mathbb{N}}$ with, for all $i \in \mathbb{N}^*$,*
  - $l_i = \inf\{t > u_{i-1} | \varphi(f(t)) \neq \perp\}$ *($u_0 = 0$)*
  - $u_i = \inf\{t > l_i | \varphi(f(t)) = \perp\}$
  - $q_i = \varphi(f(t))$ *for some and every* $t \in (l_i, u_i)$.

- *Let H be a continuous system and φ a state abstraction. There is a* trajectory from x to x' cutting $\varphi^{-1}(Q)$, *if there exist $f \in F, 0 \leq t_1 < t_2 \leq t_3 \in \mathbb{R}$ such that $f(0) = x, f(t_3) = x'$, with $\varphi(x) \neq \perp, \varphi(x') \neq \perp$, and $\forall t \in (0, t_1), \varphi(f(t)) = \varphi(x)$, $\forall t \in (t_1, t_2), \varphi(f(t)) = \perp$, $\forall t \in (t_2, t_3), \varphi(f(t)) = \varphi(x')$, $\varphi(f(t_1)) \in \{\perp, \varphi(x)\}$ and $\varphi(f(t_2)) \in \{\perp, \varphi(x')\}$.*

We define the following notions of discretizations:

- **by section** the system is discretized by observing, through a state abstraction every t time-units, for a given $t \in \mathbb{R}$, the state of the system.

- **by interval** the system is discretized by observing only the sequence of the states of the system through a state abstraction, independently of the time of the system. It is required that the abstractions of all trajectories starting from points with same abstraction must be identical.

- **by abstraction** the system is discretized by observing only the sequence of the states of the system through a state abstraction, independently of the time of the system. It is not required that the abstractions of all trajectories starting from points with same abstraction must be identical.

The definitions are derived from [3, 5, 9]. Formally:

Definition 3.5 *Let $H = (X, F)$ be a continuous system.*

- *A transition system without input $A = (Q, \delta)$ is a* discretization by section, *or* S-discretization *of H via φ, state abstraction for H to Q, if there exists $t_0 \in \mathbb{R}$, such that, for all $x, x' \in \varphi^{-1}(Q)$, there is a trajectory of time-length t_0 from x to x' if and only if $(\varphi(x), \varphi(x')) \in \delta$.*

- *A transition system without input $A = (Q, \delta)$ is a* discretization by interval, *or* I-discretization *of H via φ, state abstraction for H to Q, if for all $x, x' \in \varphi^{-1}(Q)$, there is a trajectory of H from x to x' cutting $\varphi^{-1}(Q)$ if and only if $(\varphi(x), \varphi(x')) \in \delta$.*

- *A transition system without input $A = (Q, \delta)$ is a* discretization by interval and by section *or* SI-discretization *of H via φ, state abstraction for H to Q, if A is simultaneously a S–discretization of H via φ and an I-discretization of H via φ.*

- *A transition system without input $A = (Q, \delta)$ is a* discretization by abstraction, *or* A-discretization *of H via φ, state abstraction for H to Q, if the set of the trajectories of A is exactly the set of the φ-signatures of the trajectories of H.*

We get the notions of simulation by:

Definition 3.6 *Let $H = (X, F)$ be a continuous system.*
Let $A = (Q, \delta)$ be a transition system without input.

- H I-simulates A *if* A *is an I-discretization of* H.
- H S-simulates A *if* A *is a S-discretization of* H.
- H SI-simulates A *if* A *is a SI-discretization of* H.
- A *is an* abstraction *of* H, *or* H φ-realizes A, *denoted by* $A \leq_\varphi H$ *if* A *is an* A*-abstraction of* H.

The links between these definitions and the definitions in literature can be stated as follows. Our definition of I-simulation for deterministic systems is similar to the definition of [9], if we add that φ must be continuous, $\varphi^{-1}(Q)$ must be an open set, and there must exist $\epsilon > 0$ such that, in definition 3.4, $t_1 \geq \epsilon$ and $t_3 - t_2 \geq \epsilon$. Definition 3.4 is also changed so that necessarily $\varphi(f(t_1)) = \varphi(f(t_2)) = \perp$. Our definition of I-simulation for deterministic systems is similar to the definition of Q-simulation in [5] if we add that φ must be an one to one function, and if conditions $t_1 = 0$, $t_2 = t_3$ are added to definition 3.4. Our definition of S-simulation for deterministic systems is similar to the notion of S-simulation in [9] if we add that φ must be continuous. Our definition of A-simulation for deterministic systems is similar to the notion of abstraction in [3], if we add that φ must be such that, for all $q \in Q$, $\varphi^{-1}(q)$ is a convex relatively open set, and φ is not necessarily required to be surjective.

In all the incoming results of this paper, it is possible to add the previous hypotheses (φ continuous, one to one,$t_1 = 0$,$t_3 = t_2$,etc...) without any loss of generality. As a consequence, all our results can also be stated using the definitions of the notions of simulation in [3, 5, 9].

3.3 Notions of computation

We define the notion of input for continuous systems, by considering their discretizations:

Definition 3.7 (Off-line computation) *Let $\mathcal{S}$ be a class of continuous systems.*

A decision function $F : \{0,1\}^+ \rightarrow \{0,1\}$ is off-line S-computable *(respectively:* I-computable, SI-computable, A-computable*) by $\mathcal{S}$ in time T, if there exist $H = (X, F) \in \mathcal{S}$, a state abstraction $\varphi : X \rightarrow Q$ for H to Q, an off-line system $S = (Q, \delta, \phi, A, R)$ that computes F in time T, such that $A = (Q, \delta)$ is a S-discretization (resp: I-discretization, SI-discretization, A-discretization) of $H = (X, F)$ via φ.*

Thus function $F : \{0,1\}^+ \rightarrow \{0,1\}$ is considered as off-line recognized by continuous system $H = (X, F)$ in time T, if there exists a function $\varphi : X \rightarrow Q$ such that a discretization of H via φ computes off-line F in time T. Let H and φ be fixed. Define $A_H = \varphi^{-1}(A)$, $R_H = \varphi^{-1}(R)$. $A_H \subset X$ and $R_H \subset X$ are called *the accepting* and *rejecting sets* of H. We say that $x_w \in X$ *encodes* $w \in \{0,1\}^+$ if $\varphi(x_w) = \phi(w)$. For $q \in Q$, denote $V_q = \varphi^{-1}(q)$. We call *encoding function* a function $\psi : \{0,1\}^+ \rightarrow X$ that maps each $w \in \{0,1\}^+$ to $w' = \psi(w)$ such that w' encodes w.

The definition means that the words $w \in \{0,1\}^+$ accepted by H (that is such that $F(w) = 1$) are the words such that, for some x_w that encodes w, there exists a trajectory $f \in F$ starting from x_w ($f(0) = x_w$) that intersects the accepting set A_H (that is there exists $t \in \mathbb{R}^+$ such that $f(t) \in A_H$) . The words $w \in \{0,1\}^+ \rightarrow \{0,1\}$ that are rejected by H, are the words such that, for some $x_w \in X$ that encodes w, there exists a trajectory starting from x_w that intersects the rejecting set R_H.

Thus, H is considered as a computational machine by using its discretization: a computation of H (that is what corresponds to a computation of S) is a trajectory of H. The acception or rejection is given by the fact that the trajectory crosses or not the accepting or rejecting sets. The computation time is given by the computation time of the discretization. For example suppose that A is a I-discretization or A-discretization of H: time T of a computation of H is given by the number of sets $V_q = \varphi^{-1}(q)$ crossed by the trajectory. That is, for a trajectory $f \in F$ from $x \in X$ ($f(0) = x$) to $x' \in X$ ($f(t) = x'$, for some $t \in \mathbb{R}^+$), T is given by n where $q_1 q_2 \dots q_n$ is the φ-signature of trajectory f from x to x'. If now for example A is a S-discretization of H, time T of a computation of H is given by $T = t/t_0$ where t_0 is the constant t_0 of definition 3.5.

Note that there might be no correspondence between the time of a computation and the time of the continuous system: in other words, T can be different from t. In the case of a S-discretization (or SI-discretization) computation time T and continuous system time t are equivalent, but T and t are usually different in all the other cases.

Similarly, we define the notion of on-line computation:

Definition 3.8 (On-line computation) *Let $\mathcal{S}$ be a class of continuous systems.*

A decision function $F : \{0,1\}^+ \to \{0,1\}$ is on-line S-computable *(respectively:* I-computable, SI-computable*) by $\mathcal{S}$ in time T, if there exist $H = (X, F) \in \mathcal{S}$, $H_0 = (X, F_0) \in \mathcal{S}$, $H_1 = (X, F_1)$, a state abstraction $\varphi : X \to Q$, an on-line system $S = (Q, \delta, \delta_0, \delta_1, q_0, A, R)$ that computes F in time T, such that $A = (Q, \delta)$, $A_0 = (Q, \delta_0)$, $A_1 = (Q, \delta_1)$) are S-discretizations (resp: I-discretizations, SI-discretizations) of respectively $H = (X, F)$, $H_0 = (X, F_0)$ and $H_1 = (X, F_1)$ via same function φ.*

Let $H = (X, F), H_0 = (X, F_0)$ and $H_1 = (X, F_1)$ be fixed. Thus, a computation is given by a trajectory f of a continuous system $H' = (X, F')$ where F' is either F_0, F_1 or F depending of time: every computation trajectory f starts from a point x_0 that encodes q_0 (that is, $\varphi(x_0) = q_0$). Suppose $u = u_0 u_1 \dots u_{|u|-1} \in \{0,1\}^+$ is the input. The evolution of trajectory f is first given by a function of F_{u_0} during one computation time unit (that is until time $\sqcup_0$, where $\sqcup_0$ is the first positive real with $\varphi(f(\sqcup_0)) \neq \perp$ for the case of I-computability, or during time $\sqcup_0 = t_0$ for the case of S-computability): $\exists f_0 \in F_{u_0}, f_0(0) = x_0, \forall t \in [0, \sqcup_0], f(t) = f(t_0)$. Then the evolution of trajectory f starts from $f(\sqcup_0)$ and evolves during one computation time unit to $f(\sqcup_1)$ according to a function of F_{u_1}: $\exists f_1 \in F_{u_1}, f_1(\sqcup_0) = f(\sqcup_0), \forall t \in [\sqcup_0, \sqcup_1], f(t) = f_1$, then according to a function of $F_{u_2}, \dots, F_{u_{|u|-1}}$, and finally according to a function of F for all the next computation time units. The acception or rejection is given by the fact that trajectory f crosses or not the *accepting or rejecting sets A_H, R_H*, where $A_H = \varphi^{-1}(A)$ and $R_H = \varphi^{-1}(R)$.

3.4 Properties

We can classify the notions of simulation by the following theorem:

Theorem 3.1 *The following relations between the notions of simulation are true:*

- *The notions of S-simulation and I-simulation are not comparable.*
- *The notions of S-simulation and abstraction are not comparable.*
- *The notion of abstraction is strictly more powerful than the notion of I-simulation:*

$$H \text{ I-simulates } A \text{ via } \varphi \Rightarrow A \leq_\varphi H.$$
$$A \leq_\varphi H \not\Rightarrow H \text{ I-simulates } A \text{ via } \varphi.$$

The following transitivity results are true:

- *Suppose that a class $\mathcal{C}$ of continuous systems I-simulates a class $\mathcal{C}'$ of transition systems without input. Suppose that class C' Q-simulates a class $\mathcal{C}''$ of transition systems without input. Then class $\mathcal{C}$ I-simulates class $\mathcal{C}''$.*
- *Suppose that a class $\mathcal{C}$ of continuous systems φ-realizes a class $\mathcal{C}'$ of transition systems without input. Suppose that class $\mathcal{C}'$ φ-realizes a class $\mathcal{C}''$ of transition systems without input. Then class $\mathcal{C}$ φ-realizes class $\mathcal{C}''$.*

Proof: First two points are straightforward. Third point is proved using arguments similar to theorem 2.2: a deterministic continuous system H that φ realizes a non-deterministic system A is built. Non-deterministic system A cannot be S-simulated or I-simulated by deterministic system H via φ.

Let H be a continuous system that I-simulates (respectively: φ-realizes) a transition system without input A via φ. Suppose that A Q-simulates (resp: ψ realizes) a transition system without input B via ψ. Then, it can be checked that H I-simulates B via $\psi \circ \varphi$. The first (resp: second) transitivity result follows.

□

As before, the notion of abstraction for continuous systems is very powerful since with this notion non-deterministic machines can be simulated by deterministic continuous systems.

The previous notions of simulations give us the tools to study the computational power of continuous systems. Several such systems will be studied in section 4. In order to simplify these studies, we relate them to the simulations of analog two stack automata. We need the following definition:

Definition 3.9 *Suppose that a class $\mathcal{S}$ of continuous systems simulates (whatever the notion of simulation used) a class $\mathcal{C}$ of transition systems without input: for all $C \in \mathcal{C}$, $C = (Q_M, \delta_M)$ there exists a system $S_C = (X_C, F_C) \in \mathcal{S}$ such that S_C simulates C via a function φ_C.*

Suppose that,

$$\left\{ \begin{array}{l} C = (Q_C, \delta_C) \\ C' = (Q_C, \delta'_C) \end{array} \right. \Rightarrow \varphi_C = \varphi'_C \wedge X_C = X'_C$$

Then we say that $\mathcal{S}$ simulates $\mathcal{C}$ via transition independent functions.

We can then state:

Theorem 3.2
- *Let $\mathcal{C}$ be a class of continuous systems that I-simulates (respectively: SI-simulates) the reversible deterministic analog two stack automata.*

  Then:

  - *Every function $F : \{0,1\}^+ \to \{0,1\}$ in P/poly is off-line I-computable (resp: SI-computable) in polynomial time by $\mathcal{C}$.*
  - *Every function $F : \{0,1\}^+ \to \{0,1\}$ is off-line I-computable (resp: SI-computable) in exponential time by $\mathcal{C}$.*

- *Let $\mathcal{C}$ be a class of continuous systems that I-simulates (respectively: SI-simulates) the reversible deterministic analog two stack automata via transition independent functions.*

  - *Every function $F : \{0,1\}^+ \to \{0,1\}$ in P/poly is on-line I-computable (resp: SI-computable) in polynomial time by $\mathcal{C}$.*
  - *Every function $F : \{0,1\}^+ \to \{0,1\}$ is on-line I-computable (resp: SI-computable) in exponential time by $\mathcal{C}$.*

Proof:

- Let $M = (Q, \delta)$ be a reversible analog two stack automaton that recognizes F. There exists a system $H \in C$ such that M is the I-discretization (resp: SI-discretization) of H via φ. Automaton M can be considered as an off-line system.

- Let $M = (Q, \delta)$ be a reversible analog two stack automaton, with stack alphabet Σ. that recognizes F^r, where $F^r(a_1 a_2 \dots a_n) = F(a_n \dots a_2 a_1)$ for all words $a_1 a_2 \dots a_n \in \Sigma^*$. Let $M_0 = (Q, \delta_0)$ (respectively: $M_1 = (Q, \delta_1)$) be a stack automaton such that δ_0 (resp: δ_1) on every step systematically pushes 0 (resp: 1) on the first stack and leaves the second stack unchanged. By definition, since $\mathcal{C}$ simulates the analog automata via transition independent functions, we get that there exist continuous systems $H = (X, F), H_0 = (X, F_0), H_1 = (X, F_1)$ such that M, M_0, M_1 are their respective I-discretizations (resp: SI-discretizations) via a same function φ. F is computed by on-line system $S = (Q, \delta, \delta_0, \delta_1, a_0, A, R)$ where q_0 is the initial state of M, A, R are the accepting and rejecting sets of M.

$\square$

In section 4, we will prove that many classes of continuous systems (the class of mirror systems, piecewise constant derivative systems, differential systems and linear hybrid systems) I-simulate or SI-simulate reversible analog two stack automata via transition independent functions. With previous theorem, we will be able to conclude for each of them that they can off-line and on-line compute every function of $P/poly$.

3.5 Necessity of dimension 3

We prove in this subsection that dimension 2 is not sufficient to simulate Turing machines. Our result is based on arguments from [3]. We will show in the next sections that, in dimension 3, continuous systems have super-Turing capabilities. We need the following definition:

Definition 3.10 (Abstraction relative to ψ) *Let $A = (Q, \delta)$ be a transition system without input. Let ψ be a function from Q to a set Q'. The* abstraction of A relative to ψ *is the transition system $A' = (Q', \delta')$ such that $(q, q') \in \delta'$ if and only if there exist $q_1, q_2, \dots, q_n \in Q$, such that, for all $i \in \{1, 2, \dots, n-1\}$, $(q_i, q_{i+1}) \in \delta$, and there exists n_0, $1 \leq n_0 < n$, such that, for all $1 \leq i \leq n_0$, $\psi(q_i) = q$, and for all $n_0 < i \leq n$, $\psi(q_i) = q'$.*

Note that, the abstraction A' of A relative to ψ is defined such that A' is an abstraction of A via ψ. We define now the notion of *regular state abstraction*:

Definition 3.11 (Regular state abstraction) *Let $\varphi : X \to Q$ be a state abstraction (i.e: a function), with $X \subset \mathbb{R}^d$. Let $\psi : Q \to Q'$ be a state abstraction. φ is* regular relatively to ψ *if there exist $|Q'|$ convex-subsets $V_1, V_2, \dots, V_{|Q'|} \subset \mathbb{R}^d$, such that $V_q \cap V_{q'} = \emptyset$ for all $q \neq q' \in Q$, and such that $\varphi^{-1}(\psi^{-1}(q')) \subset V_{q'}$ for all $q' \in Q'$.*

Using arguments similar to [3], we state:

Theorem 3.3 *Let $H = (X, F)$ be a deterministic system with continuous trajectories in dimension 2 (i.e: $X \subset \mathbb{R}^2$). Let $A = (Q, \delta)$ be a transition system without input. Assume that H I-simulates (respectively: SI-simulates, φ-realizes) A via φ. Let ψ be a function from Q to Q'. Let $A' = (Q', \delta')$ be the abstraction of A relative to ψ. Assume that φ is regular relatively to ψ. Then graph $G' = (Q', \delta')$ is necessarily a planar graph.*

Proof: From the transitivity relations in theorems 2.2 and 3.1, we get that $A' = (Q', \delta')$ is realized by H via $\varphi' = \psi \circ \varphi$. It can be checked that φ' is such that, for all $q' \in Q'$, ${\varphi'}^{-1}(q')$ is included into a convex set V'_q. The proof of the necessity of dimension 3 in [3] can be easily generalized to this case, and we get that A' cannot be realized by H if G' is not a planar graph. The result follows.

□

In what follows, we will deal only with the simulation of discrete or analog two stack automata $M = (Q, \Sigma, \delta, q_0, F)$. M can always be considered as a transition system without input $M = (Q' = Q \times \Sigma^{\#} \times \Sigma^{\#}, \vdash)$. We define a particular state abstraction $\psi_M : Q' \to Q$ defined by, for all $\gamma_1, \gamma_2, \in \Sigma^{\#}$, $q \in Q$, $\psi_M(q, \gamma_1, \gamma_2) = q$.

We can now define the notion of *state regular simulation*:

Definition 3.12 (State regular simulation) *Let $H = (X, F)$ be a continuous system. We say that H* state regularly *simulates (whatever the notion of simulation used) a discrete or analog two stack automaton M if H simulates M via a function φ which is regular relatively to ψ_M.*

All the simulations that we will use in this paper will be state regular simulations. We get the following corollary from theorem 3.3

Corollary 3.1 *Analog or discrete two stack automata can not be state regularly I-simulated (respectively: SI-simulated, φ-realized) by deterministic systems with continuous trajectories in dimension 2.*

Proof: It is easy to construct an analog or discrete two stack automaton M such that its abstraction relative to ψ_M is not a planar graph. Henceforth, theorem 3.3, proves that M can not be simulated by a deterministic system with continuous trajectories in dimension 2, via a function φ which is regular relatively to ψ_M.

□

Note that the condition of state regular simulations avoids the unfolding on the plane of the transition graph of the machine to be simulated. As a conclusion, dimension 2 is not sufficient to get universality, unless non deterministic systems, non continuous trajectories or non regular state simulations are used. Hence, from now, we are mainly going to focus on continuous systems in dimension 3. We will show that in dimension 3, deterministic systems with continuous trajectories do have super-Turing capabilities.

4 Computational power of continuous systems

4.1 Mirror systems

In [19, 20], Moore studies the unpredictability and the undecidability of dynamical systems. He proposed a transformation called Generalized Shift Map, that has the computational power of Turing machines. He claims that it is possible, using planar and parabolic mirrors, to conceive physical systems that realize the generalized shift map transformations. The Generalized Shift Map was extended to an "Analog Shift Map" by Siegelmann in [23, 24]. We generalize here the results of Moore and prove that mirror systems are also able to realize analog automata. This generalization is similar to the one done in [23, 24].

Definition 4.1 (Mirror system)
- *A* mirror system *(or* billiard*) is a physical system made of a finite number of mirrors. A trajectory of the system is given by the evolution of a particle in the system: the particle reflects on the mirrors according to the physical reflection laws. Between two reflections, the trajectory of the particle is a straight line.*

- *A* planar parabolic mirror system *S is a mirror system such that all the mirrors of S are either planar or parabolic.*

We claim:

Theorem 4.1 *Planar parabolic mirror systems I-simulate deterministic analog two stack automata.*

Thus, it is possible to conceive a physical system that has the computational power of analog two stack automata. The computation is done by a particle that reflects on the mirrors. The sequence of the states of the system is given by the sequence of the intersections of the particle trajectory with a fixed section of plane (see proof and figure 1).

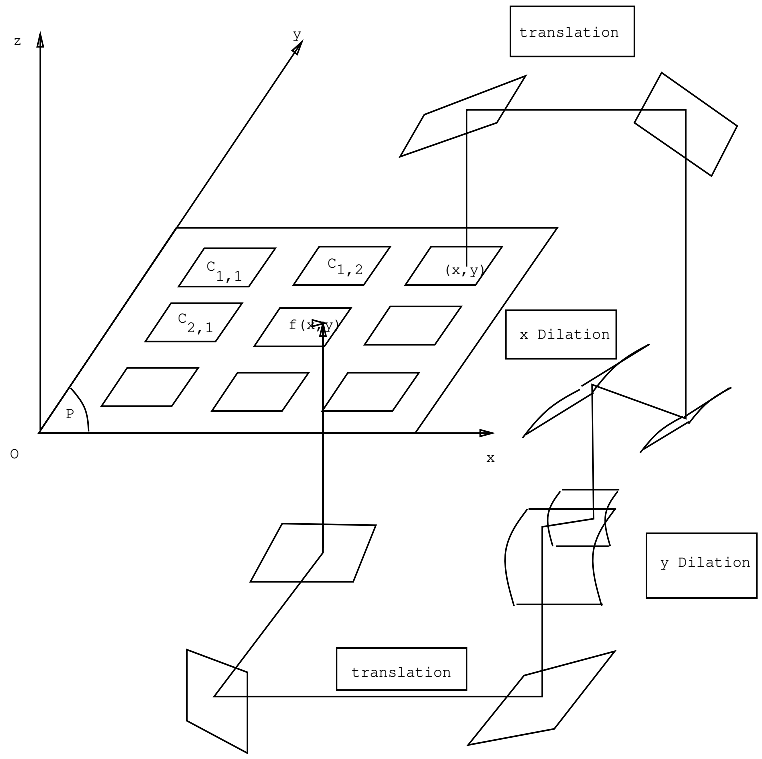

Figure 1: Mirror system simulating an analog automaton (Partially represented: only one path $\mathcal{C}_{i,j}$ has been represented.)

Proof: We prove that every deterministic reversible analog two stack automaton M can be I-simulated by a planar parabolic mirror system S. The result follows from theorems 2.4 and 3.1 since every deterministic analog two stack automaton can be Q-simulated by a reversible one.

From theorem 2.5, we know that M is K-simulated by the iterations of a disconnected one to one piecewise linear function f. We use the notations of definition 2.11 and the notations of the technical considerations in section 2.5 page 13. Let P be the plane section $P = \{(x, y, 0) | (x, y) \in [0, 1]^2\}$ in the space (O, x, y, z). We build S such that, if a particle p crosses P perpendicularly in a point $(x, y, 0)$ in $z > 0$ direction, then particle p necessarily crosses again P perpendicularly in $z > 0$ direction, in $(x', y', 0)$, where $(x', y') = f(x, y)$.

In [19], using homothetic parabolic mirrors, Moore gives a way to realize every dilation of coefficient k with $k > 0$: see figure 2. Using planar mirrors, for each

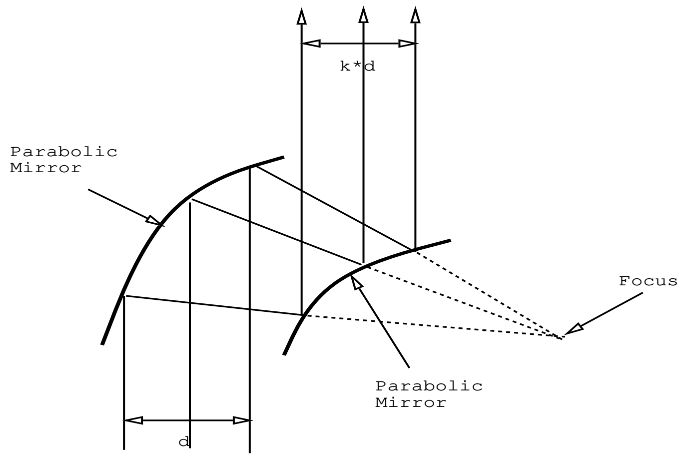

Figure 2: Homothetic parabolic mirrors realizing a dilation.

$C_{i,j} = [c_1, d_1] \times [c_2, d_2]$ we build a "path" $\mathcal{P}_{i,j}$ that brings a particle p crossing P in $(x, y, 0)$, with $(x, y) \in C_{i,j}$, through parabolic mirror systems that realize dilations on x and y direction by the coefficients $\beta_{i,j,1}$ and $\beta_{i,j,2}$ corresponding to function $f(x_1, x_2) = (\alpha_{i,j,1} + \beta_{i,j,1}x_1, \alpha_{i,j,2} + \beta_{i,j,2}x_2)$ on $C_{i,j}$. Then, using other planar mirrors, path $\mathcal{P}_{i,j}$ brings particle p to cross again P in $(x', y', 0)$, where $(x', y') = f(x, y) \in C'_{i,j}$: see figure 1. Remark that, from the considerations page 13, for all $l \in \{1, 2\}$,

- either the dilations are by strictly positive coefficients ($\beta_{i,j,l} > 0$)
- either $\beta_{i,j,l} = 0$ implies $c_l = d_l$, that is that no dilation at all is needed. Only a translation by $\alpha_{i,j,l}$ is required.

Hence, the whole construction can be done using only dilations by strictly positive coefficients.

From equation (7) page 13, we know that none of the path $\mathcal{P}_{i,j}$, for $i, j \in [1, n]$, have to intersect each another. So, all the path $\mathcal{P}_{i,j}$ can be built independently, and we get that M is I-simulated by system S, made of the union of the paths $\mathcal{P}_{i,j}$ of planar and parabolic mirrors.

□

It is interesting to outline that, with theorem 4.1, the unpredictability and undecidability of mirror systems is actually greater than claimed by Moore [19]. For example, Moore proved that any non-trivial property is undecidable for mirror systems. But, we can go further and state that there exist physical systems S such that no Turing machine is able to give the state of system S, at time n, for an arbitrary $n \in \mathbb{N}$ unless you feed the Turing machine with more and more information during the simulation. Note that it would be possible to construct Turing machines that give the state of these mirror systems at time n, if we do not suppose n arbitrary in $\mathbb{N}$, but bounded by an integer $n_0 \in \mathbb{N}$.

Of course, the mirror systems that are strictly more powerful than Turing machines are some for which the function G of the corresponding Generalized Shift Map (see terminology in [19, 20]) has an infinite Domain of Effect (DoE). The reader can refer to [23, 24] for a discussion along this line. Recall that we assume a continuous space and time medium.

We can now also consider mirror systems as computational models, using theorem 3.2.

Corollary 4.1 • *Every function $F : \{0,1\}^+ \to \{0,1\}$ in P/poly is off-line and on-line I-computable in polynomial time by planar parabolic mirror systems.*

- *Every function $F : \{0,1\}^+ \to \{0,1\}$ is off-line and on-line I-computable in exponential time by planar parabolic mirror systems.*

Proof: Just check for the second point that the simulation of analog automata by planar parabolic mirror systems given by theorem 4.1 is actually done via transition independent functions.

□

We also get, from theorems 2.2 and 3.1, that:

Corollary 4.2 *Planar parabolic mirror systems:*

- *I-simulate deterministic discrete two stack automata.*
- *I-simulate deterministic pushdown automata*
- *I-simulate deterministic finite state automata*
- *φ-realizes non-deterministic finite state automata.*

4.2 Piecewise constant derivative systems

The notion of simulation used in previous section was the notion of I-simulation. We go further and present here systems that simulate analog automata using the SI-simulation notion. Actually we pursue the work of [3, 5, 17] about Piecewise Constant Derivative systems. Note that similar systems have also been studied in [27, 26].

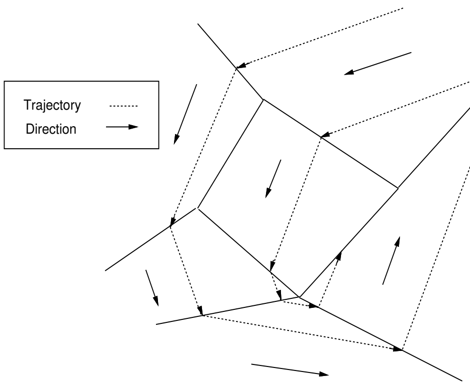

Figure 3: A PCD system in dimension 2.

Definition 4.2 (PCD System[5, 17]) *A Piecewise Constant Derivative system (PCD) is a pair $H = (X, g)$ where X is the* state-space, *g is a (possibly partial) function from X to a finite set of vectors $C \subset X$, and for every $c \in C$, $g^{-1}(c)$ is a finite union of convex polyhedral sets. The trajectories of the PCD system are given by the solutions of the differential equation $\dot{x} = g(x)$.*

In other words, a PCD system consists of partionning the space into convex polyhedral sets, called *regions*, and assigning a constant derivative, called *slope*, to all the points sharing the same region. The trajectories of such systems are broken lines, with the breakpoints occuring on the boundaries of the regions [5]. The reachability problem for PCD system was proved to be decidable for PCD systems in dimension 2 [17], and undecidable for PCD systems in dimension 3 [3, 5]. We go further and prove that, in dimension 3, PCD systems are also able to simulate analog automata:

Theorem 4.2 • *PCD systems in dimension 3 SI-simulate deterministic reversible analog two stack automata.*

- *PCD systems in dimension 3 I-simulate deterministic analog two stack automata.*

Proof: The proof is quite similar to the proof of theorem 4.1: we prove that every deterministic reversible analog two stack automaton M can be SI-simulated by a PCD system S in dimension 3. Since every deterministic analog two stack automaton can be Q-simulated by a reversible one, the results follow from theorems 2.4 and 3.1.

From theorem 2.5, we know that M is K-simulated by the iterations of a disconnected one to one piecewise linear function f. We will use the notations of definition 2.11 and the notations of the technical considerations in section 2.5 page 13. Let P be the plane section $P = \{(x, y, 0) | (x, y) \in [0, 1]^2\}$ in the space (O, x, y, z). We build S such that, if a trajectory t crosses P perpendicularly in a point $(x, y, 0)$ in $z > 0$ direction, then trajectory t necessarily crosses again P perpendicularly in $z > 0$ direction, in $(x', y', 0)$, where $(x', y') = f(x, y)$, one unit time later. So we will get SI-simulation of M by S.

We claim that, with a PCD system, it is possible to compute every multiplication of one of the coordinates by k, for $k \geq 0$: on region $Z_1 = \{(x, y, z) | 0 \leq x \leq 1 \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1 - x\}$ the slope is defined as $(0, 0, 1)$. On region $Z_2 = \{(x, y, z) | 0 \leq x \leq k \wedge 0 \leq y \leq 1 \wedge 1 - x \leq z \leq 1\}$ the slope is defined as $(k, 0, 1)$. Every trajectory entering in $(x, y, 0)$ at time 0 in Z_1 will leave Z_2 in $(kx, y, 1)$ at time 0: see figure 4. We call such a part of a PCD system a *dilation unit*.

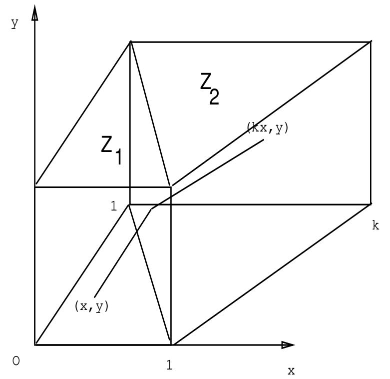

Figure 4: Dilation realized by a PCD system: dilation unit.

We claim now that, with a PCD system, it is possible to realize a "right angle": on region $Z_1 = \{(x, y, z) | 0 \leq x < 1 - z \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$ the slope is defined as $(0, 0, 1)$. On region $Z_2 = \{(x, y, z) | 1 - z \leq x < 1 + z \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$ the slope is $(1, 0, 0)$. On region $Z_3 = \{(x, y, z) | 1 + z \leq x < 2 \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$ the slope is chosen as $(1/3, 0, 0)$. Every trajectory entering in $(x, y, 0)$ at time 0 in Z_1 will leave Z_3 at time 3 in $(2, y, 1 - x)$: see figure 5. We call such a part of a PCD system a *right angle unit*.

It is also possible to build *linear units*, of length l, and time-length t, for any $l, t \in \mathbb{R}^{+*}$: on region $Z_1 = \{(x, y, z) | 0 \leq x \leq 1 \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq l\}$ the slope is

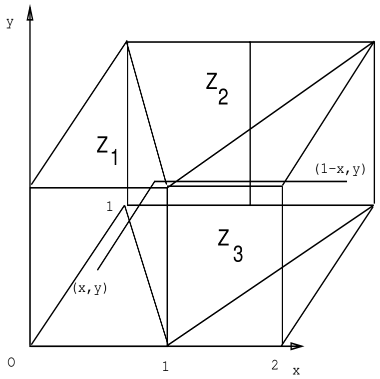

Figure 5: Right angle unit.

chosen as $(0, 0, t/l)$. Every trajectory entering in $(x, y, 0)$ at time 0 in Z_1 will leave Z_1 in (x, y, l) at time t: see figure 6.

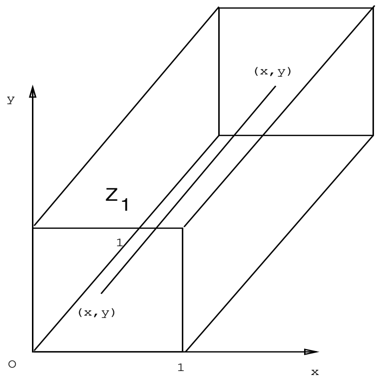

Figure 6: Linear unit.

Using linear units and right angle units, for each $C_{i,j}$, we build a "path" $\mathcal{P}_{i,j}$ that brings any trajectory t crossing P in $(x, y, 0)$, with $(x, y) \in C_{i,j}$ through dilations units, that realize the x and y dilations by the coefficients $\beta_{i,j,1}$ and $\beta_{i,j,2}$ corresponding to function $f(x_1, x_2) = (\alpha_{i,j,1} + \beta_{i,j,1}x_1, \alpha_{i,j,2} + \beta_{i,j,2}x_2)$ on $C_{i,j}$. Then using linear and right angle units, path $\mathcal{P}_{i,j}$ brings back trajectory t to cross again P in $(x', y', 0)$, where $(x', y') = f(x, y) \in C'_{i,j}$: see figure 7.

Note that actually, as in theorem 4.1, from technical considerations of page 13 only dilations by strictly positive coefficients are needed: see proof of theorem 4.1. Similarly, none of the paths $\mathcal{P}_{i,j}$ have to intersect, and the paths can be built independently: see proof of theorem 4.1. The global PCD system is made of the union of the paths $\mathcal{P}_{i,j}$, for $i, j \in [1, n]$.

The right angle, linear and dilation units are made such that the time $t_{i,j}$ taken by a trajectory t to follow entirely path $\mathcal{P}_{i,j}$, from $(x, y, 0), (x, y) \in C_{i,j}$ to $(x', y', 0), (x', y') = f(x, y) \in C'_{i,j}$, is independent of trajectory t (i.e: independent of (x, y)). We call *time-length* of $\mathcal{P}_{i,j}$ the value of $t_{i,j}$ Let i_0, j_0 be such that $t_{i_0,j_0} = max\{t_{i,j} | i, j \in [1, n]\}$. $\mathcal{P}_{i_0,j_0}$ is the slowest path. It is always possible to adjust the time-lengths of the linear units of all the other paths, such that the time-lengths of all paths $\mathcal{P}_{i,j}$, for $i, j \in [1, n]$, are set to the same value t_{i_0,j_0}. Note that, by multiplying all slopes by the constant $1/t_{i_0,j_0}$, is is possible to set the time-lengths of all the paths to exactly one time unit.

Hence, we get that M is SI-simulated by S.

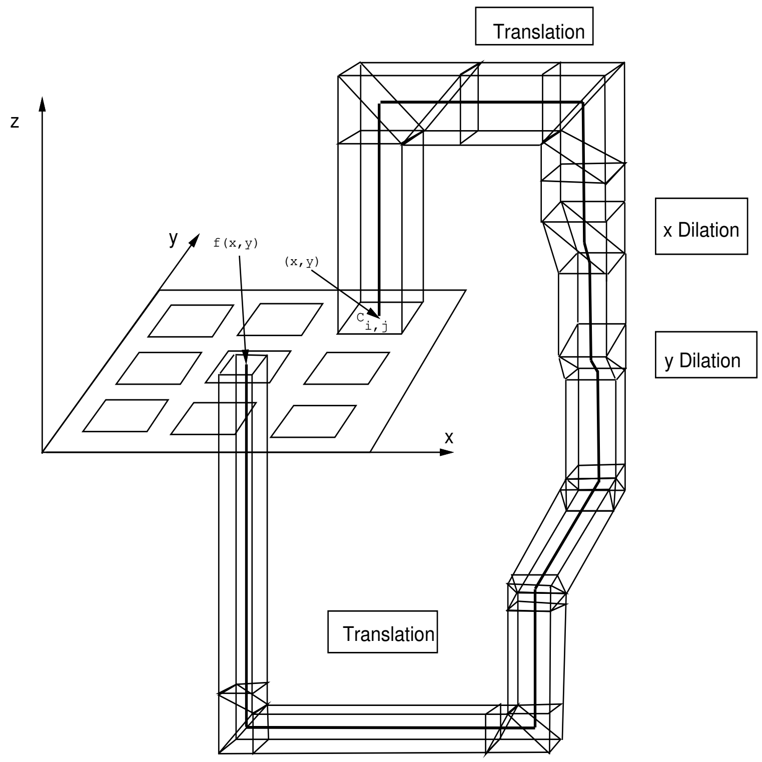

Figure 7: A PCD system in dimension 3 simulating an analog two stack automaton. Only one path $\mathcal{P}_{i,j}$ have been represented.

$\square$

Since analog two stack automata can simulate Turing-machines, the undecidability results of [3, 5] can be seen as consequences of theorem 4.2. We can determine the computational power of PCD systems by the following results:

Corollary 4.3

- *Every function $F : \{0,1\}^+ \to \{0,1\}$ in P/poly is off-line and on-line SI-computable in polynomial time by a PCD system in dimension* 3.
- *Every function $F : \{0,1\}^+ \to \{0,1\}$ is off-line and on-line SI-computable in exponential time by a PCD system in dimension* 3.

Proof: Immediate from theorem 3.2: it can be checked that the SI-simulation of reversible analog two stack automata by PCD systems in dimension 3 given by theorem 4.2 is done via transition independent functions.

$\square$

Note that, very recently, Asarin and Maler [4] proved some super-Turing capabilities of PCD systems even with purely rational coefficients, using some Zeno properties of these systems. However, the notion of time of a computation used in [4] is different from ours: they define the computation time as the intrinsic time of the dynamical system [4]. Our notion of computation time is here equivalent to the number of regions crossed by the trajectory.

Actually, with our notion of computation time, we can prove that we cannot get more power from PCD systems:

Theorem 4.3 • *Let $F : \{0,1\}^+ \to \{0,1\}$ be a function off-line I–computable (respectively: S-computable, SI-computable) by a PCD system $H = (X, F)$, where $X \subset \mathbb{R}^p$.*

- *such that an encoding function ψ is computable by a linear machine: that is, there exists a linear machine[12] that is able, given $w \in \{0,1\}^+$, to return the real number $\psi(w)$.*
- *the accepting and rejecting sets are convex polyhedra of $\mathbb{R}^p$.*
- *Each trajectory of H crosses at most a polynomial number, in the size of the input, of regions.*

Then $F \in P/poly$.

• *Let $F : \{0,1\}^+ \to \{0,1\}$ be a function on-line I-computable (respectively: S-computable, SI-computable) by PCD systems.*

- *such that the accepting and rejecting sets are convex polyhedra of $\mathbb{R}^p$.*
- *Each trajectory crosses at most a polynomial number, in the size of the input, of regions.*

Then $F \in P/poly$.

Proof: The hypotheses are chosen so that, it is always possible to simulate the computation of the PCD systems by linear machines in polynomial time. The result follows from a result in [12]: every language recognized in polynomial time by a linear machine with discrete inputs is in *P/poly*.

□

As a conclusion, we have characterized the computational power of PCD systems as exactly the computational power of analog automata: that is *P/poly* in polynomial time, and unbounded in exponential time.

4.3 Differential systems

We are now going to focus on the computational power of differential systems: we consider the class of continuous systems $H = (X, F)$, where $X \subset \mathbb{R}^n$, and F is given by the set of solutions of an ordinary differential equation (ODE) $\dot{x} = g(x)$ over $\mathbb{R}^n$.

First remark is that PCD systems are differential continuous systems: the trajectories of a PCD systems are given by the solutions of $\dot{x} = g(x)$, where g is defined as a piecewise constant function. But function g is usually supposed to be Lipschitz, or at least continuous. One main reason is that the existence of solutions to a given ODE is easily proved only in these two cases. Cauchy theorem states that, with a given initial condition, there is existence and unicity of the solution for Lipschitz ODEs, and only existence but not unicity for continuous ODEs. The question that we want to answer is to know if the previous results of super-Turing capabilities of dynamical systems can be generalized to Lipschitz ODE systems, or by default, to continuous ODE systems.

Note that some results are already known: see [9]. Branicky proved that Turing machines, stack automata and finite state automata can be SI-simulated by continuous ODEs in $\mathbb{R}^3$, and that finite state automata can be I-simulated by Lipschitz continuous ODEs in $\mathbb{R}^3$. We state:

Theorem 4.4 • *Ordinary differential equations defined by $\dot{x} = g(x)$, with g Lipschitz continuous piecewise linear on $[0,1]^3$, SI-simulate deterministic reversible analog two stack automata.*

- *Ordinary differential equations defined by $\dot{x} = g(x)$, with g Lipschitz continuous piecewise linear on $[0,1]^3$, I-simulate deterministic analog two stack automata.*

Proof: The proof is based on the proof of theorem 4.2. We use exactly the same arguments, except that the right angle units, linear units and dilation units are different. The new units U are chosen such that the modulus of the speed of any trajectory entering an unit U is equal to 1, and such that the modulus of the speed of any trajectory leaving U is also equal to 1. Moreover, the speed $g(x)$ in any unit U is built as a continuous function. To do so, interpolation regions are inserted in the right angle, linear and dilation units of theorem 4.2 to get the new ones.

Thus, the new linear unit, of length l, and time-length t, for $l, t \in \mathbb{R}^+$ is defined as: let $\alpha = 1/3$, and β such that $(2\beta \ln(\beta) + \beta - 1)/(3\beta(\beta - 1)) = t$. On $Z_1 = \{(x,y,z) | 0 \leq x \leq \alpha l \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$, function g is defined as $g(P) = (1 - x/(\alpha l))(1,0,0) + x/(\alpha l))(\beta, 0, 0)$ on $P = (x,y,z)$. On $Z_2 = \{(x,y,z) | \alpha l \leq x \leq (1-\alpha)l \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$, function g is defined as $g(P) = (\beta, 0, 0)$. On $Z_3 = \{(x,y,z) | (1-\alpha)l \leq x \leq l \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$ g is defined as $g(P) = (l-x)/(\alpha l)(\beta,0,0) + (x - l(1-\alpha))/(\alpha_l)(1,0,0)$. Any trajectory entering in Z_1 at time 0 with speed $(1,0,0)$ in $(0,y,z)$ leaves Z_3 at time t with speed $(1,0,0)$ in (l,y,z): see figure 8.

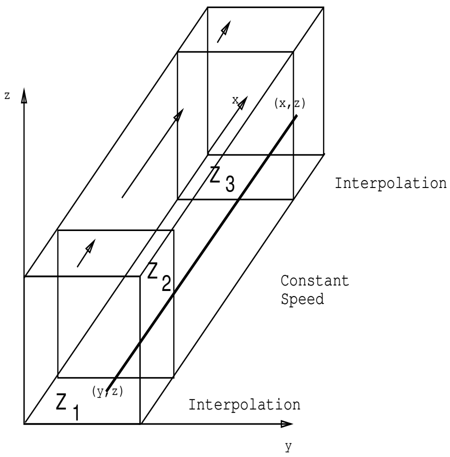

Figure 8: Linear unit.

The new right angle unit is build in the following way: on $Z_1 = \{(x,y,z) | 0 \leq x \leq 3/2 \wedge 0 \leq y \leq 1 \wedge 0 \leq z \leq 1\}$ function g is defined as $g(P) = z(0,0,1) + (1-z)(1,0,1)$: that is, Z_1 is an interpolation region that interpolates speed from $(0,0,1)$ to $(1,0,1)$. On $Z_2 = \{(x,y,z) | 1/2 \leq x \leq 3/2 \wedge 0 \leq y \leq 1 \wedge 1 \leq z \leq 2\}$ we define $g(P) = (1,0,1)$. $Z_3 = \{(x,y,z) | 3/2 \leq x \leq 5/2 \wedge 0 \leq y \leq 1 \wedge 1 \leq z \leq 5/2\}$ is chosen to be an interpolation between $(1,0,1)$ and $(1,0,0)$: $g(P) = (5/2 - x)(1,0,1) + (x - 3/2)(1,0,0)$. On $Z_4 = \{(x,y,z) | 5/2 \leq x \leq z+1 \wedge 0 \leq y \leq 1 \wedge 3/2 \leq z \leq 5/2\}$, $g(P) = (1,0,0)$. $Z_5 = \{(x,y,z) | z+1 \leq x \leq z + 3/2 \wedge 0 \leq y \leq 1 \wedge 3/2 \leq z \leq 5/2\}$ is an interpolation region between speed $(1,0,0)$ and $(1/3,0,0)$: $g(P) = (z + 3/2 - x)(1,0,0) + (x - z - 1)(1/3,0,0)$. On $Z_6 = \{(x,y,z) | z + 3/2 \leq x \leq 4 \wedge 0 \leq y \leq 1 \wedge 3/2 \leq z \leq 5/2\}$ we define $g(P) = (1/3,0,0)$. $Z_7 = \{(x,y,z) | 4 \leq x \leq 5 \wedge 0 \leq y \leq 1 \wedge 3/2 \leq z \leq 5/2\}$ is an interpolation region between $(1/3,0,0)$ and $(1,0,0)$: $g(P) = (5-x)(1/3,0,0) + (x-4)(1,0,0)$. Any trajectory entering Z_1 at time 0 in $(x,y,0)$ with $x, y \in [0,1]$ with speed $(0,0,1)$ leaves Z_7 in $(5, y, 2-x)$ a constant time later with speed $(1,0,0)$: see figure 9.

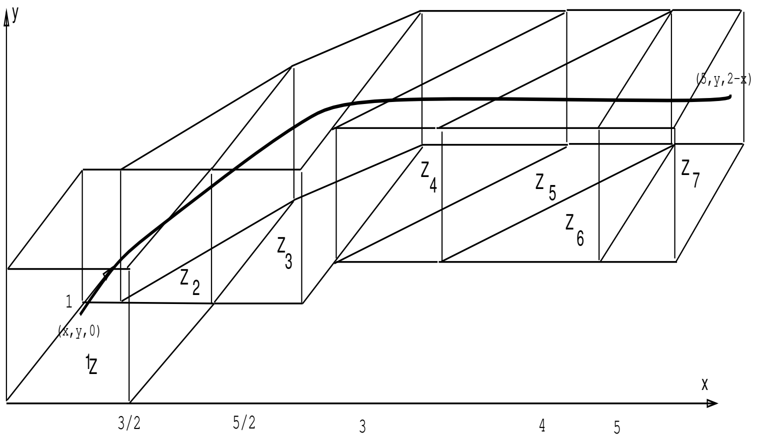

Figure 9: Right angle unit

The dilation unit is built in a similar way: we consider the dilation unit from theorem 4.2, for $k > 0$ and its two regions. We insert two interpolation regions Z_2 and Z_4 in between that do respectively interpolation from speed $(0, 0, 1)$ to $(k, 0, 1)$ and from speed $(k, 0, 1)$ to speed $(0, 0, 1)$: see figure 10. Any trajectory entering in $(x, y, 0)$ with speed $(0, 0, 1)$ at time 0 will leave Z_4 in $(kx + \alpha_x, y, 1 + \alpha_z)$ at time γ, where α_x, α_z and γ are some constants.

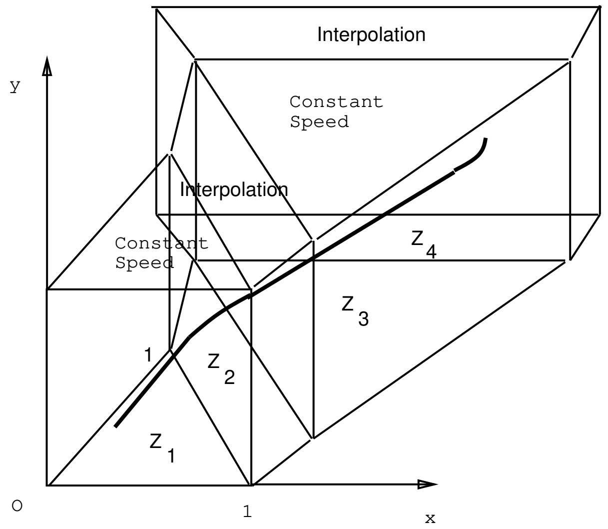

Figure 10: Dilation unit

As in the proof of theorem 4.2, the paths $\mathcal{P}_{i,j}$ are built using right angle, linear and dilation units. The time-lengths of the linear parts are chosen such that the time-lengths of all the paths $\mathcal{P}_{i,j}$ are identical, using a process similar to proof of theorem 4.2. All the dimensions can be dilated by some constants such that the whole construction enters in $[0, 1]^3$. We get then, a partially defined function g that corresponds to the union of all the paths $\mathcal{P}_{i,j}$, for $i, j \in [1 \dots n]$. Partial continuous piecewise linear function g can be completed, for example by triangulation, to a continuous piecewise linear function defined on all $[0, 1]^3$. Since a continuous function on a compact subset is Lipschitz, the result follows.

□

Remark that we extend the results from [9]: theorem 4.4 implies that (respectively: reversible) Turing machines can be I-simulated (resp: SI-simulated) by bounded Lipschitz ordinary differential equations. Furthermore, we have proved that bounded continuous piecewise linear functions can be used. We can also go further and state:

Theorem 4.5
- *Ordinary differential equations defined by $\dot{x} = g(x)$, with g Lipschitz smooth $\mathcal{C}^\infty$ on $[0,1]^3$, SI-simulate deterministic reversible analog two stack automata.*
- *Ordinary differential equations defined by $\dot{x} = g(x)$, with g Lipschitz smooth $\mathcal{C}^\infty$ on $[0,1]^3$, I-simulate deterministic analog two stack automata.*

Proof: In the proof of theorem 4.4, we used linear interpolations. But we could also use $\mathcal{C}^\infty$ interpolations, using the usual mathematical methods.

□

Then, we get:

Corollary 4.4
- *Every function $F : \{0,1\}^+ \to \{0,1\}$ in P/poly is off-line and on-line SI-computable in polynomial time by ordinary differential equations continuous Lipschitz piecewise linear on $[0,1]^3$.*
- *Every function $F : \{0,1\}^+ \to \{0,1\}$ is off-line and on-line SI-computable in exponential time by ordinary differential equations continuous Lipschitz piecewise linear on $[0,1]^3$.*

Proof: Immediate from theorem 3.2, since the simulations of reversible analog two stack automata by Lipschitz ordinary differential equations, given by theorem 4.4, are done via transition independent functions.

□

Hence, we get that Lipschitz ODEs have at least the computational power of analog automata. We turn now to the problem of finding an upper bound to the computational power of ordinary differential equations: the following result shows the difficulty of this problem: every deterministic discrete transition system is SI-computable by a system defined by a continuous ordinary differential equation in dimension 3.

Theorem 4.6 *(Consequence of [9]) Let $A = (Q, \delta)$ be a deterministic transition system without input, where $A \subset \mathbb{Z}^n$. Then, there exists a continuous system $H = (\mathbb{R}^3, F)$, where F is given by the set of the solutions of a continuous ordinary differential equation in dimension 3, that SI-simulates A.*

Proof: A state $q = (q_1, q_2, \ldots, q_n) \in \mathbb{Z}^n$ of A can be encoded by integer $p = \prod_{i=1}^n p_i^{q_i}$, where p_i is the i^{th} prime number. Hence, transition system A can be K-simulated by a transition system $A' = (\mathbb{Z}, \delta')$. The result follows from theorem 5.7 in [9] applied to system A'.

□

Note that, in the previous proof, unbounded spaces are used. However we get that the computational power of continuous ordinary differential equations is unbounded in dimension 3.

Corollary 4.5 *Let $F : \{0,1\}^+ \to \{0,1\}$*

- *F is off-line SI-computable in constant time by continuous ordinary differential equations in dimension 3.*

- *F is on-line SI-computable in linear time by continuous ordinary differential equations in dimension 3.*

Proof: Let $A = (\mathbb{Z}, \delta)$ be the transition system without input defined, for all $q \in \mathbb{Z}$, by $\delta(q) = -1 - F(q)$. Let $H = (\mathbb{R}^3, F)$ that SI-simulates A, given by theorem 4.6.

- Function F is off-line computed by the system $S = (\mathbb{Z}, \delta, \phi, Acc, Rej)$, where $Acc = \{-2\}$, $Rej = \{-1\}$ and $\phi : \{0,1\}^+ \to \mathbb{Z}$ is the function that maps w to the integer that has w as radix-2 expansion. By definition, we get that F is off-line SI-computable in constant time by continuous ODEs.
- Function F is on-line computed by the system $S = (\mathbb{Z}, \delta, \delta_0, \delta_1, 0, Acc, Rej)$, where $\delta_0(q) = 2q$ and $\delta_1(q) = 2 * q + 1$, with $Acc = \{-2\}$, $Rej = \{-1\}$. $(\mathbb{Z}, \delta)$, $(\mathbb{Z}, \delta_0)$ and $(\mathbb{Z}, \delta_1)$ can be SI-simulated by continuous ODEs on $\mathbb{R}^3$, from theorem 4.6, via a same abstraction function φ, since it can be checked that the simulations given by theorem 4.6 are simulations via transition independent functions. By definition, we get that F is on-line SI-computable in linear time by continuous ODEs.

$\square$

As a consequence, it seems that continuous differential equations on unbounded spaces do not give "reasonable" computational models. Hence, from now, we focus on Lipschitz ordinary differential equations on bounded sets: at this time, the only case where we can answer is:

Theorem 4.7
- *Let $F : \{0,1\}^+ \to \{0,1\}$ be off-line S-computable in polynomial time by a differential system $H = (X, F)$, where F is the set of the solutions to a Lipschitz ordinary equation $\dot{x} = g(x)$ on compact subset $X \subset \mathbb{R}^n$.*
  - *Suppose that an encoding function ψ is in $PE_d/poly$: cf [13, 14].*
  - *Suppose that the accepting and rejecting sets of H are convex polyhedra of $\mathbb{R}^n$.*
  - *Suppose that the solutions of $\dot{x} = g(x)$ are in $P_d/poly$[13, 14].*

  Then F is in $P/poly$.
- *Let $F : \{0,1\}^+ \to \{0,1\}$ be on-line S-computable in polynomial time by Lipschitz ordinary differential equations on a compact subset $X \subset \mathbb{R}^n$.*
  - *Suppose that the accepting and rejecting sets are convex polyhedra of $\mathbb{R}^n$.*
  - *Suppose that the solutions of the ODEs are in $P_d/poly$[13, 14].*

  Then F is in $P/poly$.

Proof:

- Let $H = (X, F')$ be a differential continuous system that off-line S-computes decision function F, such that F' is the set of the solutions of an ordinary Lipschitz differential equation $\dot{x} = g(x)$. Let t_0 be the real of definition 3.5 for the definition of S-discretization. Let $x \in X$. For $x \in X$, denote f_x the unique solution of $\dot{x} = g(x)$ such that $f_x(0) = x$. Since F is off-line S-computable by H, we get that F is computed by off-line system $S = (X, f, \phi, A, R)$ where $f : X \to X$ is defined, for all $x \in X$ as $f(x) = f_x(t_0)$, and ϕ,A,R are respectively an encoding function, the accepting and rejecting sets of continuous system

H. It is known that for Lipschitz ODE the solutions depend in a Lipschitz way of initial conditions. Precisely, the following assertion is true: for all $t \in \mathbb{R}^+$:

$$|f_x(t) - f_y(t)| \leq |x - y| \exp^{kt}$$

We get that F is recognized by off-line system $S = (X, f, \phi, A, R)$ where $f : X \to X$ is ($\exp^{kt_0}$-)Lipschitz. The result follows from lemma 2.1.

- Similarly, it can be proved that if F is on-line S-computed by Lipschitz ordinary differential equations, F is computed by an on-line system

$$S = (X, f, f_0, f_1, q_0, A, R)$$

where f, f_0, f_1 are Lipschitz functions. The result is immediate from lemma 2.1.

□

Note that requiring solutions of the ODE to be in $P_d/poly$ seems a very strong condition.

4.4 Hybrid systems

Alur and al. propose in [1] the following definition:

Definition 4.3 (Hybrid System [1]) *A* hybrid system *is made of 6-components:*

$$H = (Loc, Var, Lab, Edg, Act, Inv)$$

where:

- Loc *is a finite set of vertices called* locations.
- Var *is a finite set of real-valued variables. A* valuation *is a function* $v : Var \to \mathbb{R}$. *The set of valuations will be written* V. *A* state *is a pair (l,v) with* $l \in Loc$ *and* $v \in V$. *The set of states will be written* Σ.
- Lab *is a finite set of* synchronization labels *that contains the* stutter label τ.
- Edg *is a finite part of* $Loc \times Lab \times \mathcal{P}(V^2) \times Loc$. *Let* $e = (l, a, \mu, l') \in Edg$*:* l *is called the* source location*,* l' *is called the* target location *and* μ *is called the* transition relation*. The following condition is required:* $\forall l \in Loc, (l, \tau, Id, l) \in Edg$*, where* $Id = \{(v, v) | v \in V\}$.

  The transition e *is* enabled *in a state* (l, v) *if for some valuation* $v' \in V$, $(v, v') \in \mu$. *The state* (l', v')*, then, is a* transition successor *of the state* (l, v).
- Act *is a function which maps each* $l \in Loc$ *to a subset* $Act(l)$ *of the functions from* $\mathbb{R}^+$ *to* V. *The following condition is required:* $\forall l \in Loc, \forall f \in Act(l), \forall t \in \mathbb{R}^+, (f + t) \in Act(l)$ *where* $(f + t)(t') = f(t + t'), \forall t, t' \in \mathbb{R}^+$.
- Inv *is a function which maps each* $l \in Loc$ *to a subset* $Inv(l) \subset V$.

At any time instant, the state of a hybrid system is given by a control location and values for all variables. The states change in two ways: by discrete and instantaneous transitions that change both the control location and the values of variables, and by time delays that change only the values of the variables according to the activities of the current location [1].

A *run* [1] of the hybrid system H is a finite or infinite sequence $\rho : \sigma_0 \longmapsto^{t_0}_{f_0} \sigma_1 \longmapsto^{t_1}_{f_1} \sigma_2 \longmapsto^{t_2}_{f_2} \ldots$ of states $\sigma_i = (l_i, v_i) \in \Sigma$, nonnegative reals $t_i \in \mathbb{R}^+$, and activities $f_i \in Act(l_i)$ such that for all $i \geq 0$,

1. $f_i(0) = v_i$

2. for all $0 \leq t \leq t_i$, $f_i(t) \in Inv(l_i)$

3. the state σ_{i+1} is a transition successor of the state $(l_i, f_i(t_i))$

We will call *dimension* of hybrid system H, and denote $dim(H)$, the cardinality of Var. We propose also the following definitions:

Definition 4.4
- *A hybrid system H is* time-deterministic *[1] if for every $l \in Loc$ and every $v \in V$, there is at most one function $f \in Act(l)$ with $f(0) = v$.*

- *A hybrid system H is* (full-)deterministic *if H is simultaneously time--deterministic and such that for every $l \in Loc$, every $v \in V$, every $f \in Act(l)$ and every $t, t' \in \mathbb{R}^+$, we have*

$$\left\{ \begin{array}{l} (l, v) \longmapsto_f^t (l', v') \\ (l, v) \longmapsto_f^{t'} (l'', v'') \end{array} \right. \Rightarrow (l', v') = (l'', v'') \wedge t = t'$$

We need also the formalism about linear hybrid systems in [1]: we just suppress the fact that in a linear term all the coefficients are integers. Actually, if we suppose, that the coefficients can only be integers or rationals, that means, for example, that a PCD system [3, 4, 5, 17] cannot be considered as a linear hybrid system. Assuming real coefficients seems more realistic.

Definition 4.5 *A linear term over the set Var of variables is a linear combination with real coefficients.*

A linear formula over Var is a boolean combination of inequalities between linear terms over Var.

Definition 4.6 (Linear hybrid systems [1]) *A hybrid system H is* linear *if H is time-deterministic, and its activities, invariants, and transition relations can be defined by linear expressions over the set Var of variables:*

1. *For all $l \in Loc$, the activities $Act(l)$ are defined by a set of differential equations of the form $\dot{x} = k_x$ where k_x is a real constant. The* rate *k_x of the variable x at location l, is denoted by $Act(l, x) = k_x$.*

2. *For all locations $l \in Loc$, the invariant $Inv(l)$ is defined by a linear formula ψ over Var*
$$v \in Inv(l) \Leftrightarrow v(\psi)$$

3. *For all transitions $e \in Edg$, the transition relation μ is defined by a guarded set of non-deterministic assignments.*

$$\psi \Rightarrow \{x := [\alpha_x, \beta_x] | x \in Var\}$$

where the guard ψ is a linear formula and for each variable $x \in Var$, both interval boundaries α_x and β_x are linear terms:

$$((v, v') \in \mu) \Leftrightarrow v(\psi) \wedge (\ \forall x \in Var,\ v(\alpha_x) \leq v'(x) \leq v(\beta_x))$$

If $\alpha_x = \beta_x$, the updated value α_x of variable x after transition e, is denoted by $\mu(e, x) = \alpha_x$.

We will need also the following definition [1]:

Definition 4.7 ([1]) • *If $Act(l,x) = 0$ for each location $l \in Loc$, and $\mu(e,x) \in \{0,1\}$ for each transition $e \in Edg$, x is a* proposition.

- *If there is a nonzero integer $k \in \mathbb{Z}$ such that $Act(l,x)=k$ for each location l and $\mu(e,x) \in \{0,x\}$ for each transition e, then x is a* skewed clock. *A* multirate timed system *is a linear hybrid system all of whose variables are propositions and skewed clocks. An* n-rate timed system *is a multirate timed system whose skewed clocks proceed at n different rates.*

See [1, 2] for the definitions of the following special cases of linear hybrid systems: discrete systems, finite-state systems, timed automata, multi-rate timed systems, n-rate time systems, integrator systems. Examples of linear hybrid systems can also be found in [28, 11, 22, 21]. The reader should also refer to [27, 26] for some study of these systems from the control point of view.

We focus now on the computational power of linear hybrid systems. Thus, we study continuous systems that are not necessarily systems with continuous trajectories. Theorem 3.3 cannot be applied any more, and we obtain that now, dimension 2 is sufficient to get universality and super-Turing capabilities: we construct some linear hybrid systems with the computational power of analog automata in dimension 2.

Theorem 4.8 • *Linear hybrid systems in dimension 2 SI-simulate non--deterministic analog two stack automata.*

- *Full-deterministic linear hybrid systems in dimension 2 SI-simulate deterministic analog two stack automata.*

Proof: Let M be a deterministic analog two stack automaton. From theorem 2.5, we now that M is K-simulated by the iterations of a disconnected piecewise linear function f. We use the notations of definition 2.11. It is easy to construct a linear hybrid system H with two variables x_1, x_2 such that the sequence of the values of the two variables x_1, x_2 after each discrete transition corresponds to the sequence of the values of the iterations of function f: the location $l \in Loc = [1 \dots n] \times [1 \dots n]$ of H corresponds at any time to the pair (i,j) such that $(x_1, x_2) \in C_{i,j}$. Since f is linear on every $C_{i,j}$, it is sufficient to build the discrete transitions of H on location $l = (i,j) \in Loc$, such that their correspond to function f on $C_{i,j}$.

It is an easy exercise to generalize the whole construction to non deterministic two stack automata using non-deterministic transitions.

□

Furthermore, we give an extension of the results in [1] about the undecidability of the reachability problem for 2-rate timed systems: we prove that it is also possible to get super-Turing capabilities with 2-rate timed systems.

Theorem 4.9 *2-rate timed systems SI-simulate non–deterministic analog two stack automata.*

Proof: We use accurate clocks of rate 1, and skewed clocks of rate 4. Using methods similar to proof of theorem 3.2 in [1], we are able to realize the piecewise linear functions f, given by theorem 2.5. Theorem 3.2 in [1] gives a mean to realize multiplication and division by 4. To realize addition of α to the real number representing the content of a stack, just reset the corresponding clock when it reaches $1-\alpha$, instead of reseting the clock when it reaches 1: see [1].

□

Using theorem 3.2 (generalized to non-deterministic systems) and from the fact that the simulations given by 4.8 are done via transition independent functions, we get:

Theorem 4.10 • *Every function $F : \{0,1\}^+ \rightarrow \{0,1\}$ in $NP/poly$ (respectively: $P/poly$) is off-line and on-line SI-computable in polynomial time by (resp: deterministic) linear hybrid systems in dimension 2.*

- *Every function $F : \{0,1\}^+ \rightarrow \{0,1\}$ is off-line and on-line SI-computable in exponential time by deterministic linear hybrid systems in dimension 2.*

The most interesting fact is that, for linear hybrid systems, we are able to give an upper bound to their computational power:

Theorem 4.11 • *Let $F : \{0,1\}^+ \rightarrow \{0,1\}$ be a function off-line S-computable (resp: I-computable,SI-computable) in polynomial time by a linear (respectively: deterministic) hybrid system*

  - *such that an encoding function ϕ is computable by a linear machine: there exists a linear machine[12] M, such that, given $w \in \{0,1\}^+$, M is able to give the value of $\phi(w)$ in polynomial time.*
  - *such that the accepting (respectively: rejecting set) is given by a particular location: that is defined by $A_H = \{(l,v) | v \in V\}$ (resp: $R_H = \{(l',v) | v \in V\}$) for some $l, l' \in Loc$.*

  Then $F \in NP/poly$ (resp: $F \in P/poly$).

- *Let $F : \{0,1\}^+ \rightarrow \{0,1\}$ be a decision function on-line S-computable (resp: I-computable, SI-computable) in polynomial time by linear (respectively: deterministic) hybrid systems*

  - *such that the accepting (rejecting set) is given by a particular location*

  Then $F \in NP/poly$ (resp: $F \in P/poly$).

Proof: The hypotheses are chosen such that linear machines [12] are able to simulate the computations of the hybrid systems. The result follows from a result in [12] that proves that every language recognized in polynomial time by a deterministic (resp: non-deterministic) linear machine with discrete inputs is in $P/poly$ (resp: $NP/poly$).

$\square$

Hence, we characterize the computational power of deterministic (respectively: non-deterministic) linear hybrid systems as exactly the computational power of analog automata: $P/poly$ (resp: $NP/poly$) in polynomial time, and unbounded in exponential time.

5 Discussion

This paper shows that many dynamical systems and hybrid systems are strictly more powerful than Turing machines. This super-Turing power comes from the dynamical systems capabilities to be "analog" machines: a continuous system computation may make an arbitrary infinite precision real number "appear", which can be used later as an advice. This was the main property used in this paper to prove the super-Turing capabilities of continuous systems.

These results have direct consequences for the decidability issues: since analog two stack automata simulate Turing machines, we get, for example, that the reachability problem is undecidable in dimension 3 for mirror systems, PCD systems, differential systems and in dimension 2 for linear hybrid systems.

But this paper also shows that there is "more" than undecidability: continuous systems are able to simulate some machines that cannot be simulated by Turing machines: hence there exist some continuous systems H, such that no Turing machine M exists, such that, given $n \in \mathbb{N}$, M is able to give the state of H, at time n. Thus, there exist systems that cannot be numerically simulated by the usual discrete models of computation (except if we add the restriction that n is not an arbitrary integer , but is an integer smaller than a given $n_0 \in \mathbb{N}$) . These systems can only be simulated by computational machines that are allowed to compute over the real numbers in unbounded-precision in constant time. For example, by the Blum Shub and Smale machine [8].

Thus, this paper outlines the limitations of the belief that all physical systems and all computational models can be simulated by Turing machines. Actually, only the discrete models can be simulated. That must be kept in mind whenever an explicit or implicit reference to Church thesis is made. Actually, one very interesting question would be to find the equivalent of the Church thesis for the continuous models: in [25], Siegelmann and Sontag proved that analog recurrent networks are very robust: allowing high order networks, polynomial activations, arbitrary Lipschitz transition functions do not give much power that the initial model of analog neural recurrent networks. They proposed the *SiSo* thesis[23, 24, 25]: every reasonable continuous computational model does not have more power than recurrent analog neural networks. Stated in terms of analog automata: the computational power of analog automata is an upper bound to the computational power of any reasonable computational model. This paper shows that many continuous systems are at least as powerful as analog automata. But the full question is still open.

One aim of this paper was also to show that the machines computing over the reals in unbounded precision are physically plausible. We have proved in this paper that it is theoretically possible to construct with a finite number of planar and parabolic mirrors a machine that is more powerful than all the Turing machines. So analog recurrent networks [25] and all the machines that compute in unbounded precision [8] may have some reality [23, 24]. However, recall that we assume a continuous physical time and space.

We would like to outline that hybrid systems are "natural" analog computational models. We proved in this paper that they have at least the power of analog two stack automata. It can be checked that hybrid systems considered as computational models can do operations that the usual analog computational models (the BSS machine [8] and its restrictions for example) cannot do: for example, a polynomial hybrid system is able to compute semi-algebraic functions in constant time in unbounded-precision: take a polynomial activation and a polynomial condition of transition. If we put away the condition that the variables must be in finite number, the BSS machine [8] can be seen itself as a particular hybrid system. Henceforth hybrid systems can be considered as very general computational models which may have even more power than all other analog machines, in particular than BSS machines [8].

6 Acknowledgments

The authors would like to thank Pascal Koiran for all his helpful comments and discussions about this paper.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin,

A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. *Lecture Notes in Computer Science*, 736:340–354, 1993.

[3] Eugene Asarin and Oded Maler. On some Relations between Dynamical Systems and Transition Systems. In *Proceedings of ICALP*, pages 59–72, 1994. Lecture Notes in Computer Science, 820.

[4] Eugene Asarin and Oded Maler. Achilles and the Tortoise Climbing Up the Arithmetical Hierarchy. In *Proceedings of FST/TCS'95*, pages 471–483, 1995. Lecture Notes in Computer Science, 1026.

[5] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:33–65, 1995.

[6] J. L. Balcázar, J. Diáz, and J. Gabarró. *Structural Complexity I.* EATCS Monographs on Theoretical Computer Science, 1988.

[7] C. H. Bennett. Logical Reversibility of computation. *IBM Journal Research Development*, 6:525–532, 1973.

[8] Lenore Blum, Mike Shub, and Steve Smale. On a Theory of Computation and Complexity over the Real Numbers: NP-completeness, Recursive Functions and Universal Machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, July 1989.

[9] Michael S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138:67–100, 1995.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory Languages and Computation.* Addison-Wesley, 1979.

[11] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration Graphs: A Class of Decidable Hybrid Systems. *Lecture Notes in Computer Science*, 736:179–208, 1993.

[12] P. Koiran. Computing over the reals with addition and order. *Theoretical Computer Science*, 133:35–47, 1994.

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

[14] Pascal Koiran. *Puissance de Calcul des réseaux de neurones artificiels.* PhD thesis, École Normale Supérieure de Lyon, June 1993.

[15] Pascal Koiran. A Weak Version of the Blum Shub Smale Model. Technical Report 005, NeuroCOLT Technical Report Series, August 1994. A preliminary version can be found in *Proceedings of 34th IEEE Symposium on Foundations of Computer Science*, pages 486-495, 1993.

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

[17] Oded Maler and Amir Pnueli. Reachability Analysis of Planar Multi-linear Systems. *Lecture Notes in Computer Science*, 697:194–209, 1995.

[18] K. Meer and C. Michaux. A Survey on real Structural Complexity Theory. To be published in *Bulletin of the Belgian Mathematical Society - Simon Stevin.*

[19] Cristopher Moore. Unpredictability and Undecidability in Dynamical Systems. *Physical Review Letters*, 64(20):2354–2357, May 1990.

[20] Cristopher Moore. Generalized shifts: unpredictability and undecidability in dynamical systems. *Nonlinearity*, 4:199–230, 1991.

[21] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An Approach to the Description and Analysis of Hybrid Systems. *Lecture Notes in Computer Science*, 736:149–178, 1993.

[22] Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. From ATP to Timed Graphs and Hybrid Systems. *Acta Inform*, 30:181–202, 1993.

[23] Hava T. Siegelmann. The Simple Dynamics of Super Turing Theories. Technical Report NN-1, Technion, 1994.

[24] Hava T. Siegelmann. Computation Beyond the Turing Limit. *Science*, 268(52110):545–548, April 1995.

[25] Hava T. Siegelmann and Eduardo D. Sontag. Analog computation via neural networks. *Theoretical Computer Science*, 131:331–360, 1994.

[26] Eduardo Sontag. From Linear to Nonlinear: Some Complexity Comparisons. In *IEEE Conf. Decision and Control*, pages 2916–2920, New Orleans, December 1995.

[27] Eduardo D. Sontag. Nonlinear Regulation: The Piecewise Linear Approach. *IEEE Transactions on Automatic Control*, AC-26(2), April 1981.

[28] Kārlis Čerāns. Decidability of Bissimulation Equivalences for Parallel Timer Processes. *Lecture Notes in Computer Science*, 663:269–300, 1992.