Achilles and the Tortoise climbing up the hyper-arithmetical hierarchy

Olivier Bournez*
obournez@lip.ens-lyon.fr

Abstract

In this paper, we characterize the computational power of dynamical systems with piecewise constant derivatives (PCD) considered as computational machines working on a continuous real space with a continuous real time: we prove that piecewise constant derivative systems recognize precisely the languages of the $\omega^{k^{th}}$ (respectively: $\omega^k + 1^{th}$) level of the hyper-arithmetical hierarchy in dimension $d = 2k + 3$ (respectively: $d = 2k + 4$), $k \geq 0$.

Hence we prove that the reachability problem for PCD systems of dimension $d = 2k + 3$ (resp. $d = 2k + 4$), $k \geq 1$, is hyper-arithmetical and is Σ_{ω^k}-complete (resp. Σ_{ω^k+1}-complete).

1 Introduction

There has been recently an increasing interest in the fields of control theory and computer science about hybrid systems. A hybrid system is a system that combines discrete and continuous dynamics. Several models have been proposed in literature. In particular, in [2, 3, 4], the authors introduce Piecewise Constant Derivative systems (PCD systems), a sub-class of the so-called linear hybrid automata of [1]: such systems consist in partitioning the Euclidean space into a finite number of convex polyhedra such that the derivative within any region is constant.

Hybrid systems can be considered either as computational machines working on a continuous space with a discrete time or as machines working on a continuous space with a continuous time: see [2, 3, 4, 9, 10].

Several theoretical computational models of machines working on a continuous space with a discrete time are known: in particular, in [5], Blum, Shub and Smale introduce the real Turing machine (see [13] for an up-to-date survey). When PCD systems are considered as machines working on a continuous space with a discrete time their computational power is known: it is proved in [2, 3, 9] that PCD systems of dimension $d \geq 3$ are equivalent to Turing machines or to a restriction of real Turing machines.

The study of machines working on a continuous space with a continuous time is only beginning. In [14], Moore proposes a recursion theory for computations on the reals in continuous time. When PCD systems are considered as machines working on continuous space with a continuous time no precise characterization of their computational power was known: recently, Asarin and Maler [3] showed using Zeno's paradox, that every set of the arithmetical hierarchy can be recognized by a PCD system of finite dimension. We gave in [7, 8] a characterization of the computational power of a restricted class of PCD systems: the purely rational PCD systems. But no characterization was given for the general class of PCD systems.

In this paper, we provide such a characterization: we prove that the languages recognized by PCD systems in dimension $d = 2k + 3$ (respectively: $d = 2k + 4$), $k \geq 0$, in finite continuous time are precisely the languages of the $\omega^{k^{th}}$ (resp. $\omega^k + 1^{th}$) level of the hyper-arithmetical hierarchy. In other words, the reachability problem for PCD systems of dimension $d = 2k + 3$ (resp. $d = 2k + 4$) is not decidable and is Σ_{ω^k}-complete (resp. Σ_{ω^k+1}-complete). In particular, that means that the reachability problem for PCD systems of dimension greater than 5 is hyper-arithmetical but is not analytic.

*Support by Esprit Project 8556. NeuroColt is acknowledged.

In section 2 we introduce PCD systems and the hyper–arithmetical hierarchy. In section 3, we introduce Real Continuous Time (RCT) machines: we prove that RCT machines can recognize some hyper–arithmetical sets. In section 4, we show that RCT machines can be simulated by PCD systems and we deduce that PCD systems can also recognize some hyper–arithmetical sets. In section 5, we prove that the bounds given in section 4 are optimal: the languages recognized by PCD systems in dimension $d = 2k + 3$ (respectively: $d = 2k + 4$), $k \geq 0$ in finite continuous time are precisely the languages of the $\omega^{k^{th}}$ (resp. $\omega^k + 1^{th}$) level of the hyper-arithmetical hierarchy.

2 Definitions

2.1 PCD systems

A convex polyhedron of $\mathbb{R}^d$ is a finite intersection of open or closed half spaces of $\mathbb{R}^d$. A polyhedron of $\mathbb{R}^d$ is a finite union of convex polyhedral of $\mathbb{R}^d$. In particular, a polyhedron may be unbounded or flat.

Definition 2.1 (PCD system) *A* piecewise constant derivative (PCD) system *[3, 4] is a dynamical system $\mathcal{H} = (X, f)$ where $X = \mathbb{R}^d$ for some d is the* state space *and f is a function from X to X such that the range of f is a finite set of vectors $C \subset X$, and for every $c \in C$ $f^{-1}(c)$ is a polyhedron, and $\frac{d^+x}{dt} = f(x)$ is the differential equation governing the evolution of x: a trajectory of $\mathcal{H}$ starting at some $x_0 \in X$ is $\Phi : \mathbb{R}^+ \to X$ such that Φ is a maximal solution of the equation with initial condition $x = x_0$, i.e $\Phi(0) = x_0$, and for every t, $f(\Phi(t))$ is defined and is equal to the right derivative of $\Phi(t)$.*

In other words a PCD system consists in partitioning the space into convex polyhedral sets, called *regions*, and assigning a constant derivative c, called *slope* to all the points sharing the same region. The trajectories of such systems are broken lines with the breakpoints occurring on the boundaries of the regions [3]: see figure 1.

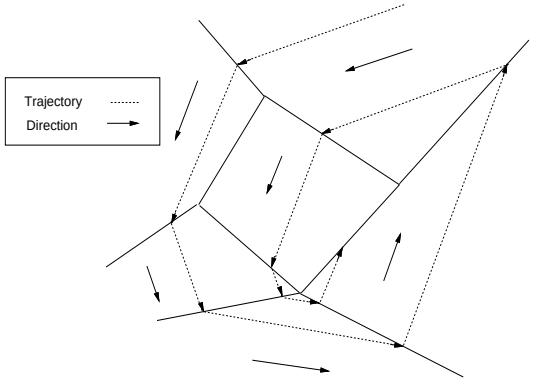

Figure 1: A PCD system in dimension 2.

A description of a PCD system is simply a list of the regions (expressed as intersections of linear inequalities) and their corresponding slope vectors. From now on, we assume that all the constants in the systems' definitions are rational (we consider only rational PCD systems with the terminology of [7, 8]).

Given a description of a PCD system $\mathcal{H}$, the reachability problem for $\mathcal{H}$ is the following: given $x, x' \in X \cap \mathbb{Q}^d$, are there a trajectory Φ and $t \geq 0$ such that $\Phi(0) = x$ and $\Phi(t) = x'$.

We can say some words on the existence of trajectories in a PCD system: let $x_0 \in X$. We say that x_0 is *trajectory well-defined* if there exists a $\epsilon > 0$ such that $f(x) = f(x_0)$ for all $x \in [x_0, x_0 + \epsilon * f(x_0)]$. It is clear that, for any $x_0 \in X$, there exists a trajectory starting from x_0 iff x_0 is trajectory well-defined. Moreover, it is clear that, given a PCD system $\mathcal{H}$, one can effectively compute the set $NoEvolution(\mathcal{H})$ of the points of X that are not trajectory well-defined.

2.2 Computing with PCD systems

Let Σ be a finite alphabet with at least two letters. Without loss of generality, assume that $\Sigma = \{1, 2, \ldots, n_\Sigma\}$.

We write Σ^* (respectively: Σ^ω) for the the set of the finite (respectively: finite and infinite) words over alphabet Σ. We write ϵ for the empty word. If $w \in \Sigma^*$, we write $length(w)$ for the length of word w. We fix a recursive encoding of the integers over the words of Σ^*: for all integer $n \in \mathbb{N}$, we denote by $\overline{n}$ the word of Σ^* encoding integer n.

We describe now how to encode a word of Σ^ω into a real of $[0, 1]$. Denote by b_Σ the first power of 2 that is greater than $2n_\Sigma + 2$: $b_\Sigma = 2^{b'_\Sigma}$ for some $b'_\Sigma \in \mathbb{N}$.

Definition 2.2 (Encoding by $\mathcal{J}$) *Let $\Sigma = \{1, 2, \ldots, n_\Sigma\}$ be the fixed finite alphabet.*

We denote by $\mathcal{J}$ the mapping from $\Sigma^\omega \to J \subset [0, 1]$ that maps any word $w = a_1 a_2 \ldots a_i \ldots$, with $a_1, a_2, \ldots \in \Sigma$, to real number

$$\mathcal{J}(w) = \sum_{j=1}^{\infty} \frac{(2a_j)}{(b_\Sigma)^j}$$

We denote by Λ the image of Σ^* by $\mathcal{J}$: $\Lambda = \mathcal{J}(\Sigma^*)$.

PCD systems can be considered as machines recognizing some languages $L \subset \Sigma^*$ as follows:

Definition 2.3 (Computation [3])

- *Let $\mathcal{H} = (X, f)$ be a PCD system of dimension d. Let x^1, x^0 be two distinct points of $\mathbb{R}^d$. A computation of system $\hat{H} = (\mathbb{R}^d, f, \mathcal{J}, x^1, x^0)$ on entry $n \in \Sigma^*$ is a trajectory of $\mathcal{H} = (X, f)$ starting at $(\mathcal{J}(n), 0, \ldots, 0)$. The computation is accepting if the trajectory eventually reaches x^1, and refusing if it reaches x^0. It is assumed that the derivatives at x^1 and x^0 are zero.*
- *Language $L \subset \Sigma^*$ is semi-recognized by $\hat{H}$ if, for every $n \in \Sigma^*$, there is a computation on entry n and the computation is accepting iff $n \in L$. L is said to be (fully-)recognized by $\hat{H}$ when, in addition, the computation is refusing iff $n \notin L$.*

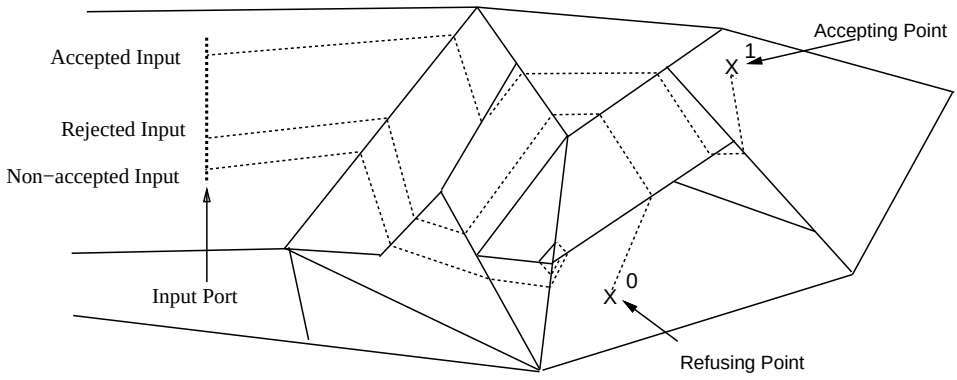

Figure 2: Some examples of computations by a PCD system.

In other words every input is encoded into a rational point of the space, and its membership is indicated by whether the trajectory starting at this point is settles in an accepting (refusing) point after a finite amount of time.

2.3 Hyper-arithmetical hierarchy

2.3.1 Presentation

We recall the definition of the hyper-arithmetical hierarchy. The hyper-arithmetical hierarchy is an extension of the arithmetical hierarchy to constructive ordinal numbers. It consists of the classes of languages $\Sigma_1, \Sigma_2, \ldots, \Sigma_k, \ldots, \Sigma_\omega, \Sigma_{\omega+1}, \Sigma_{\omega+2}, \ldots, \Sigma_{\omega 2}, \Sigma_{\omega 2+1}, \ldots, \Sigma_{\omega^2}, \ldots$ indexed by the constructive

ordinal numbers. It is a strict hierarchy and it satisfies the strict inclusions $\Sigma_\alpha \subset \Sigma_\beta$ whenever $\alpha < \beta$. It can be related to the analytical hierarchy by $\Delta_1^1 = \cup_\beta \Sigma_\beta$: see [16].

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

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

2.3.2 Formal definition

We give here the formal definitions. We use the classical notations of [15, 16]: in particular W_n (respectively: W_n^X) denotes the language recognized by the n^{th} Turing machine (resp. by the n^{th} Turing machine with oracle X); ϕ_n (respectively ϕ_n^X) denotes the function computed by the n^{th} Turing machine (resp. by the n^{th} Turing machine with oracle X); $<,>$ denotes a bijective recursive coding of pairs of words. When $X \subset \Sigma^*$ is a language, the *jump of* X, denoted by X' is defined by $X = \{\overline{u} | u \in \mathbb{N} \wedge \overline{u} \in W_u^X\}$.

The constructive ordinals are defined as follows:

Definition 2.4 (Constructive ordinals[16]) *We define by transfinite induction simultaneously $O \subset \mathbb{N}$, mapping $|\ |$ from 0 to a segment of the ordinal numbers and partial ordering $<_O$ on O.*

The ordinals in the range of $|\ |$ are called the constructive ordinals. *An ordinal α is said to have notation x iff $x \in O$ and $|x| = \alpha$.*

The transfinite induction is as follows:

- *Ordinal 0 receives notation 1: $1 \in O, |1| = 0$.*
- *Let γ be an ordinal. Assume that all the ordinals $< \gamma$ have received a notation, and assume that $<_o$ has been defined on these notations.*
  - *If $\gamma = \alpha + 1$ is a successor, γ receives notation 2^x, for all notation x of α: for all $x \in O$, if $|x| = \alpha$, then $2^x \in O, |2^x| = \gamma$ and $z <_0 2^x$ for all $z \in O$ with either $z = x$ or $z <_0 x$.*
  - *If γ is a limit, γ receives notation 3.5^y for all y such that $\{\phi_y(n)\}_{n=1}^{n=\infty}$ is an increasing sequence of notations of ordinals of limit γ: for all $y \in \mathbb{N}$, if $\{\phi_y(n)\}_{n=1}^{n=\infty}$ is a sequence of integers in O, if $\{|\phi_y(n)|\}_{n=1}^{n=\infty}$ is an increasing sequence of ordinals with limit γ such that $\forall i \forall j\ i < j \Rightarrow \phi_y(i) <_0 \phi_y(j)$, then $3.5^y \in O, |3.5^y| = \gamma$ and $z <_0 3.5^y$ for all z for which there exists n such that $z <_0 \phi_y(n)$.*
- *No other integer $y \in \mathbb{N}$ is in O.*

Function H^X that maps constructive ordinals to languages is defined as follows: $H^X(0)$ is defined as X. For all constructive ordinal k, $H^X(k+1)$ is defined as the jump of $H^X(k)$. For all constructive limit ordinal $k = lim\ k_i$, $H^X(k)$ is defined as a fixed diagonalization of the sets $(H^X(k_i))_i$:

Definition 2.5 *Let $X \subset \Sigma^*$. We define H^X as a mapping from O to the subsets of Σ^* by:*

- $H^X(1) = X$.
- $H^X(2^x) = (H^X(x))' = \{\overline{u} | u \in \mathbb{N} \wedge \overline{u} \in W_u^{H^X(x)}\}$.
- $H^X(3.5^y) = \{< u, \overline{v} > | v \in O\ \wedge\ v <_0 3.5^y\ \wedge\ u \in H^X(v)\}$.

The hyper-arithmetical hierarchy is defined by (see [16] for a proof that this definition defines unambiguously the classes Σ_α for all constructive ordinals α):

Definition 2.6 (Hyper-arithmetical hierarchy) *Let $X \subset \Sigma^*$.*

- *For all constructive ordinal $1 \leq \alpha < \omega$, and for any y such that $\alpha = |2^y|$: Σ_α^X is the class of the sets that are recursively enumerable in $H^X(y)$*

- *For all constructive ordinal $\alpha \geq \omega$ and for any y such that $|y| = \alpha$: Σ_α^X is the class of the sets that are recursively enumerable in $H^X(y)$*

We denote Σ_α for $\Sigma_\alpha^\emptyset$ for all constructive ordinal α.

3 RCT machines and the hyper–arithmetical hierarchy

We want to prove that PCD systems can recognize some sets of the hyper-arithmetical hierarchy. Our proofs are constructive. However, instead of manipulating directly PCD systems and their lists of polyhedra and slopes, we prefer dealing with a programming language. This programming language is a language for machines working with a continuous time.

In this section, we define this language and we determine how powerful it is. In the next section, we will prove that any program of this language can be translated effectively into a PCD system.

3.1 RCT machines

3.1.1 First example

We present here an informal description of Real Continuous Time (RCT) machines. A formal definition will be given in next subsection. Keep in mind that this language is in some sense ad-hoc to represent what can be computed by PCD systems.

We deal with machines that have a finite number of real registers $x_1, x_2, \ldots, x_d$ whose values can be any real of $[0, 1]$. These machines evolve according to a finite program made of assignments and of tests between the real registers. Any instruction I (assignment or test), has some associated real function $c_I : [0, 1]^d \to \mathbb{R}^+$ called *the cost of the instruction*. The execution of instruction I takes a time equal to its costs: I is executed in time $c_I(x_1, \ldots, x_d)$, where $x_1, \ldots, x_d$ are the values of the real registers of the machine when the instruction is executed.

Take for example the instruction $x_1 := 2x_2$ $[x_3]$: this instruction does $x_1 := 2x_2$ with cost x_3: if this instruction is executed at date $t \in \mathbb{R}$, then the value of the first real register at date $t + x_3$ will be equal to two times the value of the second real register at date t, where x_3 is the value of the third real register at date t.

Let us consider a first example of program:

Algorithm 1 program "Hello world".

$x_1 := 1$ [1]	
$x_2 := 1$ [1]	
$x_3 := 0$ [1]	/*set $(x_1, x_2, x_3) = (1, 1, 0)$ at date 3.*/
while (true) **do**	/*transform $(x_1, x_2, x_3) = (1/2^n, 1/2^n, 1 + 1/2 + \ldots 1/2^{n-1})$ at date $3 + 3(1 + 1/2 + \ldots 1/2^{n-1})$ for some n, to $(x_1, x_2, x_3) = (1/2^{n+1}, 1/2^{n+1}, 1 + 1/2 + \ldots 1/2^n)$ at date $3 + 3(1 + 1/2 + \ldots 1/2^n)$*/
$x_3 := x_3 + x_1$ $[x_2]$	
$x_1 := x_1/2$ $[x_2]$	
$x_2 := x_2/2$ $[x_2]$	
end while	
*limit**:	/*here, we have $(x_1, x_2, x_3) = (0, 0, 1)$ at date 9. */
$x_1 := x_3$ [1]	/*now set, $(x_1, x_2, x_3) = (1, 0, 1)$ at date 10. */

Try to simulate the evolution of this program. At time 3, $(x_1, x_2, x_3) = (1, 1, 0)$ and the program is starting to execute the while loop. At time $3 + 3$ the program is starting to execute the loop for the second time. At time $3 + 3 + 3/2$ the program is executing the loop for the third time. At time

$3+3+3/2+3/2^2+\ldots+3/2^{n-1}$, for all $n \in \mathbb{N}$, the program is executing the loop for the n^{th} time. And at time $9 = 3+\sum_{j=0}^{\infty} 3/2^j$?

This is the role played by the label $limit^*$: this labels indicates what to do when time becomes "Zeno": that is to say when an unbounded number of operations are executed in a finite time. Hence, at time 9, the program executes the instruction labeled by $limit^*$ and copies x_3 into x_1. Check that variables x_2 and x_1 tend to 0 and that variable x_3 tends to 1 during the execution of the infinite while loop. As a consequence, we consider that at time 9 the value of the first two real registers of the machine is 0 and that the value of the third real register is 1. Therefore the previous program is a program that always halts and that stops with $x_1 = 1, x_2 = 0, x_3 = 1$ at time 10.

In other words, a RCT program is a finite program made of assignments and of tests with a real cost, with possibly a special label denoted by $limit^*$ that denotes the instruction to do when time becomes "Zeno".

3.1.2 Formal definitions

We define now formally our programming language: see previous section for an example.

Definition 3.1 (Instruction, Test) • *An* assignment *in dimension d is a couple (f, c) where f, called the* operation*, is a partial mapping from $[0,1]^d$ to $[0,1]^d$ and c, called the* cost function*, is a partial mapping from $[0,1]^d$ to $\mathbb{R}^+$.*

- *A* test *in dimension d is a couple (R, c), where R is a partial relation over $[0,1]^d$, and c, called the* cost function*, is a partial mapping from $[0,1]^d$ to $\mathbb{R}^+$.*
- *An* instruction of dimension d *is either an assignment or a test of dimension d.*

For the simplicity of notations, we denote by "$x_i := g(x_1, \ldots, x_d)[c]$", the assignment (g', h') where, for all $x_1, \ldots, x_d \in [0,1]$, g' and h' are defined on $(x_1, \ldots, x_d)$ iff $g(x_1, \ldots, x_d) \in [0,1]$, and when g' and h' are defined on $(x_1, \ldots, x_d)$, then $g'(x_1, \ldots, x_d) = (x_1, \ldots, x_{i-1}, g(x_1, \ldots, x_d), x_{i+1}, \ldots, x_d)$ and $h'(x_1, \ldots, x_d) = c$. We denote by "$x_i := g(x_1, \ldots, x_d)$" the assignment $x_i := g(x_1, \ldots, x_d)$ [1]. We denote by "$R?$ $[c]$", where R is a relation, the test (R, c). We denote by "$R?$" the test $R?$ [1].

We will define below the set of the assignments and the set of the tests denoted by $Assgnmt_d$ and by $Test_d$ respectively that are admissible in dimension d.

A RCT machine of dimension d is a machine with d real registers that evolves according to its program. Its program is finite and is made of the assignments of $Assgnmt_d$ and of the tests of $Test_d$. The execution of any instruction takes a time equal to the cost of the instruction. Whenever the time becomes "Zeno" and the variables converge, the machine enters a special limit state $limit^*$, and the execution goes on from this state. Formally:

Definition 3.2 (RCT machine) • *A* Real Continuous Time machine *(RCT machine) M, or a* RCT program *of dimension d, is given by $P = (Q, q_0, q_f^+, q_f^-, limit^*, \delta)$ where:*

- *Q is the set of the* internal states of M*: Q is a finite set. $q_0, q_f^+, q_f^-, limit^* \in Q$ are the* initial state*, the* accepting, the refusing state *and the* limit state *respectively.*
- *δ defines the instructions of the program: δ is a mapping from Q to $Q \times Q \times (Assgnmt_d \cup Test_d)$.*

- *An* instantaneous description (ID) *of M is given by $(q, x_1, \ldots, x_d, t) \in Q \times [0,1]^d \times \mathbb{R}^+$. q is the* internal state*, t is the* time *and $x_1, \ldots, x_d$ are the* values of the real registers of M at time t.
- *Let $ID_1 = (q, x_1, \ldots, x_d, t)$ and $ID_2 = (q', x'_1, \ldots, x'_d, t')$ be two IDs of M. We write $ID_1 \vdash ID_2$ iff*
  - *either the instruction corresponding to ID_1 is an assignment and ID_2 is the result of the assignment: $\delta(q) = (q', q'', Assgnmt)$, with $Assgnmt = (f, c) \in Assgnmt_d$, and:*
    - *$(x_1, \ldots, x_d)$ is in the domain of function f and of function c.*
    - *$(x'_1, \ldots, x'_d) = f(x_1, \ldots, x_d)$.*
  - *or the instruction corresponding to ID_1 is a test and ID_2 and ID_2 is the result of the test: $\delta(q) = (q'', q''', Test)$, with $Test = (R, c) \in Test_d$, and:*
    - *$(x_1, \ldots, x_d)$ is in the domain of relation R and of function c.*
    - *$(x'_1, \ldots, x'_d) = (x_1, \ldots, x_d)$*
    - *$(q' = q''$ and $R(x_1, \ldots, x_d))$ or $(q' = q'''$ and $\neg R(x_1, \ldots, x_d))$.*

- *A computation of M starting from $(x_1, \ldots, x_d)$ is a sequence $(ID_i = (q^i, x_1^i, \ldots, x_d^i, t^i))_{i \leq I}$ of IDs of M, where I is an ordinal, such that:*
  - *its starts in the initial state: $ID_0 = (q_0, x_1, \ldots, x_d, 0)$*
  - *its evolves according to the instructions of the program: for all $j \leq I$, if j is a successor ordinal then $ID_{j-1} \vdash ID_j$*
  - *whenever the time becomes Zeno, its goes to label limit*: for all $j \leq I$, if j is a limit ordinal then*
    - *$t^j = \sup\{t^{j'} | j' < j\}$*
    - *for all $1 \leq i \leq d$, $x_i^j = lim_{j' \to j, j' < j} x_i^{j'}$*
    - *$q^j = limit^*$*
- *The computation is accepting (respectively: refusing) if there exists $j_0 \leq I$ with $q^{j_0} = q_f^+$ (respectively: $q^{j_0} = q_f^-$) with $\forall j < j_0, q^j \notin \{q_f^+, q_f^-\}$. In that case, $t^{j_0} \in \mathbb{R}$ is called the* time taken by the computation. *Whenever the computation is accepting, we say that M* maps $(x_1, \ldots, x_d)$ to $(x_1^{j_0}, \ldots, x_d^{j_0})$ in time t^{j_0}.

RCT machines can naturally be considered as machines recognizing some languages $L \subset \Sigma^*$ as follows:

Definition 3.3 *Let Σ be the fixed finite alphabet.*

Language $L \subset \Sigma^*$ is semi-recognized by RCT machine M *if, for all $n \in \Sigma^*$, there is an accepting computation of M starting from $(\mathcal{J}(n), 0, \ldots, 0)$ iff $n \in L$. L is* fully-recognized *if in addition, for all $n \in \Sigma^*$, there is a refusing computation starting from $(\mathcal{J}(n), 0, \ldots, 0)$ iff $n \notin L$.*

The *assignment corresponding to the execution of program M* is the assignment (f', c') of dimension d where functions f' and c' are defined on $(x_1, \ldots, x_d) \in [0,1]^d$ iff there is an accepting computation starting from $(x_1, \ldots, x_d)$; when functions f' and c' are defined on $(x_1, \ldots, x_d)$ then $f'(x_1, \ldots, x_d) = (x'_1, \ldots, x'_d)$, $c'(x_1, \ldots, x_d) = t'$ iff M maps $(x_1, \ldots, x_d)$ to $(x'_1, \ldots, x'_d)$ in time t'.

An instruction of dimension $d-1$ will be considered as an instruction of dimension d straightforwardly: for example, if (f, c) is an assignment in dimension $d-1$, we still denote by (f, c) the assignment (f', c') of dimension d defined, for all $x_1, \ldots, x_d \in [0,1]$ by $f'(x_1, \ldots, x_{d-1}, x_d) = (f(x_1, \ldots, x_{d-1}), x_d)$, $c'(x_1, \ldots, x_{d-1}, x_d) = c(x_1, \ldots, x_{d-1})$.

We define now a special transformation on instructions: when I is an instruction of dimension d, we denote by I/x_{d+1} the instruction of dimension $d+1$ that one gets by making the change of variable x_i becomes x_i/x_{d+1} for all variable x_i. This operations can seems unnatural, but keep in mind that we want a programming language that models what can be computed by PCD systems: we will see in next section that when one can realize some instruction I by a PCD system of dimension d', one can realize the instruction I/x_{d+1} by some PCD system of dimension $d'+1$. This motivates the following definition:

Definition 3.4 (Transformation $/x_{d+1}$ on instructions)
- *Let (f, c) be an assignment in dimension d. $(f/x_{d+1}, c/x_{d+1})$ is the assignment of dimension $d+1$ defined by:*
  - *f/x_{d+1} and $c/_{x_{d+1}}$ are defined on $(x_1, \ldots, x_{d+1})$ iff all the following conditions hold:*
    - *$x_{d+1} > 0$*
    - *$(x_1/x_{d+1}, \ldots, x_d/x_{d+1}) \in [0,1]^d$*
    - *function f and c are defined on value $(x_1/x_{d+1}, \ldots, x_d/x_{d+1})$*
  - *when f/x_{d+1} and $c/_{x_{d+1}}$ are defined on $(x_1, \ldots, x_d)$,*

$$\begin{aligned} f/x_{d+1}(x_1, \ldots, x_{d+1}) &= x_{d+1} f(x_1/x_{d+1}, \ldots, x_d/x_{d+1}) \\ c/x_{d+1}(x_1, \ldots, x_{d+1}) &= x_{d+1} c(x_1/x_{d+1}, \ldots, x_d/x_{d+1}) \end{aligned}$$

- *Let (R, c) be a test in dimension d. $(R/x_{d+1}, c/x_{d+1})$ is the test of dimension $d+1$ defined by:*
  - *R/x_{d+1} and c/x_{d+1} are defined on $(x_1, \ldots, x_{d+1})$ iff all the following conditions hold:*
    - *$x_{d+1} > 0$,*
    - *$(x_1/x_{d+1}, \ldots, x_d/x_{d+1}) \in [0,1]^d$*
    - *R and c are defined on value $(x_1/x_{d+1}, \ldots, x_d/x_{d+1})$.*

– *when* R/x_{d+1} *and* c/x_{d+1} *are defined on* $(x_1, \ldots, x_{d+1})$,

$$\begin{array}{rcl} R/x_{d+1}(x_1, \ldots, x_{d+1}) & = & R(x_1/x_{d+1}, \ldots, x_d/x_{d+1}) \\ c/x_{d+1}(x_1, \ldots, x_{d+1}) & = & x_{d+1}c(x_1/x_{d+1}, \ldots, x_d/x_{d+1}) \end{array}$$

Let us take an example: let us consider the instruction I defined by $x_1 := x_1 + \lambda$ [1] where $\lambda \in \mathbb{Q}$. When $x_3 > 0$ and $x_1/x_3 \in [0, 1]$, I/x_3 is equivalent to instruction $x_1 := x_1 + \lambda x_3[x_3]$.

We are ready to define the admissible instructions in dim d: this is done inductively (keep in mind that we want the instructions to correspond to what can be implemented by PCD systems):

Definition 3.5 (Admissible operations in dim d) *We define inductively the set of the assignments denoted by* $Assgnmt_d$ *(respectively: the set of the tests denoted by* $Test_d$*) that are admissible in dimension* d*:*

For all d*, for all* $i, j, k \in \{1, 2, \ldots, d\}$*, for all* $\lambda \in \mathbb{Q}$*, for all* $\lambda^+ \in \mathbb{Q}^+$*, for all* $\# \in \{>, \geq, <, \leq, =, \neq\}$*:*

- *"Linear machines instructions"*
  - "$x_i := x_i + x_k$ [1]" $\in Assgnmt_d$
  - "$x_i := x_j$ [1]" $\in Assgnmt_d$.
  - "$x_i := \lambda^+ x_i$ [1]" $\in Assgnmt_d$.
  - "$x_i := \lambda$ [1]" $\in Assgnmt_d$.
  - "$x_i := x_i + \lambda$ [1]" $\in Assgnmt_d$.
  - "$x_i \# \lambda^+?$ [1]" $\in Test_d$.
- *"Special instructions"*
  - "$x_d := x_d/2$ [x_d]" $\in Assgnmt_d$, *if* $d > 2$.
  - "$x_d := 2x_d$ [x_d]" $\in Assgnmt_d$, *if* $d > 2$.
  - "$x_d := x_d + \lambda x_k$ [x_k]" $\in Assgnmt_d$, *if* $2 < k < d$
- *"Subprograms"*
  - *If* P *is a program of dimension* d*, then* $(f, c + 1) \in Assgnmt_d$*, where* (f, c) *is the assignment corresponding to the execution of* P.
- "$Assgnmt_{d-1} \subset Assgnmt_d$", "$Test_{d-1} \subset Test_d$"
  - *If* $(f, c) \in Assgnmt_{d-1}$ *then* $(f, c) \in Assgnmt_d$.
  - *If* $(R, c) \in Test_{d-1}$ *then* $(R, c) \in Test_d$.
- *"Zeno instructions"*
  - *If* $(f, c) \in Assgnmt_{d-1}$ *and* $d > 2$ *then* $(f/x_d, c/x_d) \in Assgnmt_d$.
  - *If* $(R, c) \in Test_{d-1}$ *and* $d > 2$ *then* $(R/x_d, c/x_d) \in Test_d$.

3.2 RCT machines simulate two-stack pushdown automata and Turing machines

We recall first what a two-stack pushdown automaton is (see [11]): assume alphabet $\Sigma = \{1, 2, \ldots, n_\Sigma\}$ is fixed. A stack is a word of Σ^ω. The functions $PUSH : \Sigma \times \Sigma^\omega \to \Sigma^\omega$ and $POP : \Sigma^\omega \to \Sigma \times \Sigma^\omega$ are defined by $PUSH(v, w) = vw$, $POP(vw) = (v, w)$. Two-stack pushdown automata are defined by:

Definition 3.6 (2PDA)

- *A deterministic two-stack pushdown automaton (2PDA)* M *is given by* (Q, δ, q_0) *where*
  - Q *is the set of the* internal states *of* M*:* Q *is a finite set.* $q_0 \in Q$ *is the* initial state.
  - δ *defines the instructions of the program:* δ *maps each internal state* $q \in Q$ *to an instruction of one of the two forms:*

$$\begin{array}{lll} q_i : & w_\alpha := PUSH(v, w_\alpha) & \\ & Goto\ q_j & \end{array} \qquad \begin{array}{ll} q_i : & (v, w_\alpha) = POP(w_\alpha) \\ & If\ v = 1\ Goto\ q_{i_1} \\ & If\ v = 2\ Goto\ q_{i_2} \\ & \ldots \\ & If\ v = n_\Sigma\ Goto\ q_{n_\Sigma} \end{array}$$

Acronym	RCT machine instructions
$Push^i(j), i \in \{1,2\}, j \in \Sigma$	$x_i := x_i/b_\Sigma + (2 * j)/b_\Sigma$ [1]
$Pop^i(j), i \in \{1,2\}, j \in \Sigma$	$x_1 := b_\Sigma * (x_1 - (2 * j)/b_\Sigma)$ [1]
$Top^i(\epsilon)?, i \in \{1,2\}$	$x_i = 0?$ [1]
$Top^i(j)?, i \in \{1,2\}, j \in \Sigma$	$((x_i \geq (2 * j)/b_\Sigma)?$ [1] and $(x_i \leq (2 * j + 1)/b_\Sigma)?$ [1])

Figure 3: Correspondence between 2PDA instructions and RCT instructions.

where $\alpha \in \{1,2\}$, $v \in \Sigma$.

- *An* instantaneous description (ID) *of* M *is given by* $(q, w_1, w_2) \in Q \times \Sigma^\omega \times \Sigma^\omega$. q *is the* internal state, w_1 *and* w_2 *are the stacks of* M.
- *We write* $ID_1 \vdash ID_2$, *if, when the 2PDA is in the state given by* $ID_1 = (q, w_1, w_2)$, *if the 2PDA executes the instruction* $\delta(q)$ *(whose semantic was described above) then the 2PDA is in the state given by* ID_2.
- *A* computation of M starting with stacks w_1, w_2 *is a sequence of IDs* $(ID_i)_{i \in \mathbb{N}}$ *such that* $ID_0 = (q_0, w_1, w_2)$ *and such that for all* $i \in \mathbb{N}$ $ID_i \vdash ID_{i+1}$.

It is well known that linear machines can simulate 2PDA (see [12]). In our context:

Lemma 3.1 ([12]) *Any two-stack pushdown automaton* M *can be simulated by a RCT machine* M' *of dimension* 2 *whose program is made only of linear machine instructions.*

Proof: Given any 2PDA M, M is simulated by a RCT program M' that uses the following convention: whenever the stacks of M are $w_1 \in \Sigma^\omega, w_2 \in \Sigma^\omega$, the real registers of M' are $x_1 = \mathcal{J}(w_1), x_2 = \mathcal{J}(w_2)$. The program of M' is made such that each step of M is simulated by some instructions of M' that keeps this fact true.

More concretely, the program of M' is obtained by taking the program P of M, and replacing the instructions of P by some RCT instructions using the following correspondence:

Instruction of type	RCT instructions
q_i : $w_\alpha := PUSH(v, w_\alpha)$ Goto q_j	q_i: $Push^v(\alpha)$ Goto q_j
q_i : $(v, w_\alpha) = POP(w_\alpha)$ If $v = 1$ Goto q_{i_1} If $v = 2$ Goto q_{i_2} ... If $v = n_\Sigma$ Goto q_{n_Σ}	If $Top^1(\alpha)$ then $Pop^1(\alpha)$; Goto q_{i_1} If $Top^2(\alpha)$ then $Pop^2(\alpha)$; Goto q_{i_2} ... If $Top^{n_\Sigma}(\alpha)$ then $Pop^{n_\Sigma}(\alpha)$; Goto q_{n_Σ}

Where RCT instructions $Top^i(j)?$ $Pop^i(j)$, $Push^i(j)$, $i \in \Sigma, j \in \{1,2\}$, are defined in figure 3 (and have the purpose of testing if the top of stack j is letter i, popping letter i from stack j and pushing letter i onto stack j respectively).

□

By the well-known equivalence between two-stack pushdown automata and Turing machines (see [11]) we get:

Theorem 3.1 *Let* S *be a discrete language.*

- *Assume that* S *is recursively enumerable. Then* S *is semi-recognized by a RCT machine of dimension* 2
- *Assume that* S *is recursive. Then* S *is fully-recognized by a RCT machine of dimension* 2.

We use the following convention:

Convention 3.1 *We write*

$$\left[\begin{array}{l}(w_1,w_2)\\ \mapsto (w_1',w_2')\\ \textbf{where}\quad \textit{"conditions"}\end{array}\right]$$

for a RCT program M' that, for all w_1, w_2 verifying "conditions", maps real registers $x_1 = \mathcal{J}(w_1)$, $x_2 = \mathcal{J}(w_2)$ to $x_1 = \mathcal{J}(w_1'), x_2 = \mathcal{J}(w_2')$: to obtain M', consider any 2PDA M such that, for all $w_1 \in \Sigma^\omega, w_2 \in \Sigma^\omega$ verifying "conditions", if M is started with stacks (w_1, w_2) then M halts with stacks (w_1', w_2') (recall that 2PDAs are equivalent to Turing machines). Now apply lemma 3.1 on M to get RCT machine M'.

As an example, we write $\left[\begin{array}{l}(w,w')\\ \mapsto (ww,\epsilon)\\ \textbf{where}\quad w,w' \in \Sigma^*, w = w'\end{array}\right]$ for a RCT program that, for all $w, w' \in \Sigma^*$ such that $w = w'$, maps $(\mathcal{J}(w), \mathcal{J}(w'))$ to $(\mathcal{J}(ww), 0)$.

3.3 RCT machines and the arithmetical hierarchy

We prove in this subsection that every arithmetical set is recognized by some RCT program: we are adapting the arguments of [3] to RCT machines.

3.3.1 From semi-recognition to recognition

In definition 3.4, we defined the transformation $/x_{d+1}$ on instructions: the transformation $/x_{d+1}$ on programs is obtained by transforming one after the other the instructions of the programs. In other words, the transformation $/x_{d+1}$ on programs is equivalent to making in the programs the change of variable x_i becomes x_i/x_{d+1} for all variable x_i.

Definition 3.7 (Transformation $/x_{d+1}$ on RCT programs) *Let P be a RCT program of dimension d: $P = (Q, q_0, q_f^+, q_f^-, limit^*, \delta)$.*

We denote by P/x_{d+1} the RCT program of dimension $d+1$ defined by

$$P/x_{d+1} = (Q, q_0, q_f^+, q_f^-, limit^*, \delta')$$

where for all q, q', q'' and $Instr \in Test_d \cup Assgnmt_d$,

$$\delta(q) = (q', q'', Instr) \Leftrightarrow \delta'(q) = (q', q'', Instr/x_{d+1})$$

The interest of this transformation is the following: assume that we have a program P of dimension d that does some work on its real registers $x_1, \ldots, x_d$: for example started with real registers $(x_1, \ldots, x_d)$ P halts with real registers $f(x_1, \ldots, x_d)$ for some function $f : \mathbb{R}^d \to \mathbb{R}^d$.

Consider $P' = P/x_{d+1}$. Assume $x_{d+1} \in (0, 1]$. For all $i \leq d$, define x_i' as x_i/x_{d+1}. The first fundamental observation is to see that, since P' is obtained by making the change of variable x_i becomes x_i/x_{d+1}, the work performed by P' on $x_1', \ldots, x_d'$ is equivalent to the work performed by P on $x_1, \ldots, x_d$: in the example, started with real registers $(x_1', \ldots, x_d', x_{d+1})$ P' halts with real registers $(f(x_1', \ldots, x_d'), x_{d+1})$.

The second observation is to see that there is a big difference: P' does it faster than P: since whenever P does some instruction I at cost c, P' does equivalent instruction I/x_{d+1} at cost cx_{d+1}, P' goes $1/x_{d+1}$ times faster than P. When $x_{d+1} = 1$, that makes no difference. But when $x_{d+1} = 1/2$ for example, P' goes two times faster than P. When $x_{d+1} = 1/2^n$, P' goes 2^n times faster ...

Hence, when one has a program P that does something, one can build a program P' that does something equivalent but faster !

In fact we can improve this argument to go further: one can transform any program that semi-recognizes a set to a higher dimensional program that fully recognizes it.

Before proving so, we start by defining the following program, for $d \geq 2$:

Algorithm 2 Program $Div2^{d+1}$

$x_1 := x_1/2\ [x_{d+1}]$
$x_2 := x_2/2\ [x_{d+1}]$

...
$x_{d-1} := x_{d-1}/2\ [x_{d+1}]$
$x_d := x_d/2\ [x_{d+1}]$
$x_{d+1} := x_{d+1}/2\ [x_{d+1}]$

And the following program:

Algorithm 3 Program $Mul2^{d+1}$

$x_1 := 2x_1\ [x_{d+1}]$
$x_2 := 2x_2\ [x_{d+1}]$
...
$x_{d-1} := 2x_{d-1}\ [x_{d+1}]$
$x_d := 2x_d\ [x_{d+1}]$
$x_{d+1} := 2x_{d+1}\ [x_{d+1}]$

A program P will be said to be *clocked*, if some instructions of P are marked such that they can be used as the tops of a clock: there exists some $\Delta > 0$, such that in every execution (computation) of program P, in every time interval of length greater than Δ, at least one of the marked instructions is executed, and only a finite number of them is executed. In that case, Δ is called the *period of* P.

In particular, in a clocked program, the fact that a finite number of marked instructions is executed iff the program halts is guaranteed.

When P is a clocked program , and Q is a sub-program that always halts, write $P*Q$ for the program obtained by inserting a copy of program Q at each marked instruction of program P: the execution of program $P*Q$ corresponds to executing P, but making in addition an execution of subprogram Q for every marked instruction.

We are ready to give the idea on how one one can transform a program that semi-recognizes a set to a higher dimensional program that fully recognizes it: assume P semi-recognizes a set S: for all $w \in \Sigma^*$, on input $x_1 = \mathcal{J}(w)$, P performs some computations and eventually halts iff $w \in S$. Assume that P is clocked.

Consider P' defined by:

Algorithm 4 P'

$x_{d+1} := 1\ [1]$
$(P/x_{d+1}) * (Div2^{d+1})$
$limit^*$: Refuse

The execution of any copy of the Div2 subprograms keeps $x'_1, \ldots, x'_d$ unchanged. As a consequence, for all $w \in \Sigma^*$, on input $x_1 = \mathcal{J}(w)$, P' simulates P: when $w \in S$, P' simulates P and eventually halts at some finite time like P. When $w \notin S$, P' simulates all the instructions of the infinite non-accepting computation $\mathcal{C}$ of P on $\mathcal{J}(w)$ with the difference that P' does it faster, and that a Div2 subprogram is executed for each marked instruction of $\mathcal{C}$. Check that every execution of a Div2 subprogram multiply the speed of the simulation by 2. Since $w \notin S$, $\mathcal{C}$ contains an unbounded number of marked instructions: that implies that P' must do an unbounded number of executions of subprogram Div2. That means that P' is necessarily Zeno, and that P' eventually executes label $limit^*$ that tells it to refuse.

In other words, from a program P that semi-recognizes S, one can construct a program P' of higher dimension that fully-recognizes S. Formally we have the following theorem:

Theorem 3.2 *Assume that S is semi-recognized by some clocked program P in dimension d. Then there exists a program of dimension $d+1$ that fully recognizes S: this program is the program P' of algorithm 4.*

The proof is immediate from the following lemma:

Lemma 3.2 *Let P be a clocked RCT program of dimension d with period Δ. For all $\lambda \in (0,1]$, for all $x_1, x_2, \ldots, x_d \in [0,1]$, $(P/x_{d+1}) * Div2^{d+1}$ started with real registers $(\lambda x_1, \lambda x_2, \ldots, \lambda x_d, \lambda)$ simulates the evolution of P on $(x_1, \ldots, x_k)$ but the whole simulation of P by P/x_{d+1} is made in a finite bounded time upper bounded by $2\lambda(\Delta + d + 1)$.*

Moreover, whenever P accepts, $(P/x_{d+1}) * Div2^{d+1}$ accepts. Whenever P does not accept, $(P/x_{d+1}) * Div2^{d+1}$ converges to its limit state with all its real registers set to 0.

Proof: Let $\lambda \in (0,1]$ and $x_1, \ldots, x_d \in [0,1]$ be fixed.

Denote by $(q^j, x_1^j, \ldots, x_d^j, t^j)_{j \in J}$ the computation of P starting from $(x_1, \ldots, x_d)$. Let $Q_0 \subset Q$ gives the marked instructions of P.

Denote by $(q^{'j'}, x_1^{'j'}, \ldots, x_d^{'j'}, t^{'j'})_{j' \in J'}$ the computation of $(P/x_{d+1}) * Div2^{d+1}$ starting from $(\lambda x_1, \ldots, \lambda x_d, \lambda)$.

Denote by $j_1 < j_2 < \ldots \in J$ the sequence of the indexes corresponding to the execution of the marked instructions of P: for all $j \in J$, either $q^j \in Q_0$ and $j = j_k$ for some k or $q^j \notin Q_0$.

For $j \in J$, let n_j denote the number of marked instructions of P executed between time 0 and time t^j: n_j is the cardinality of set $\{k | t^{j_k} < t^j\}$.

It is easy to prove by transfinite induction on $j \in J$ that, for all $j \in J'$, one has $j \in J$, $q^{'j+n_j(d+1)} = q^j$, $x_i^{'j+n_j(d+1)} = \lambda x_i^j / 2^{n_j}$ for all $1 \leq i \leq d$, $x_{d+1}^{'j+n_j(d+1)} = \lambda / 2^{n_j}$, $t^{'j+n_j(d+1)} = \lambda \sum_{k=1}^{n_j} (t^{j_k} - t^{j_{k-1}} + d + 1)/2^{k-1} + \lambda(t^j - t^{j_{n_j}})/2^{n_j}$ with $t^0 = 0$, and that for all k and $l \leq d+1$, $q^{'t^{j_k}+(k-1)(d+1)+l}$ corresponds to an instruction of program $Div2^{d+1}$.

This means that $(P/x_{d+1}) * Div2^{d+1}$ simulates P: if P accepts then $(P/x_{d+1}) * Div2^{d+1}$ accepts: $q^{j_0} = q_f^+$ for some $j_0 \in J$ implies $q^{'j_0+n_{j_0}(d+1)} = q_f^+$. For all $k \in \mathbb{N}$, we have $t^{j_k} - t^{j_{k-1}} \leq \Delta$. As a consequence, for all $j \in J$, $t^j \leq 2\lambda(\Delta + d + 1)$. Hence, $(P/x_{d+1}) * Div2^{d+1}$ accepts at some finite time bounded above by $2\lambda(\Delta + d + 1)$.

If P does not accept its input, since P is assumed to be clocked, a non finite number of $Div2^{d+1}$ are executed. As a consequence, for all $1 \leq i \leq d+1$, the sequence $(x_i^j)_{j \in J}$ converge to 0. One has have $sup_{j \in J} t^j \leq 2\lambda(\Delta + d + 1)$. That means that $(P/x_{d+1}) * Div2^{d+1}$ reaches the ID $(limit^*, 0, \ldots, 0, t^*)$ at finite time $t^* = sup_{j \in J} t^{'j}$, with $t^* \leq 2\lambda(\Delta + d + 1)$.

□

Theorem 3.2 can be improved slightly: one can assume in addition that the program that fully-recognizes S returns its input when it accepts. this is the following technical lemma whose proof is detailed in the technical report [6]:

Lemma 3.3 *Let S be a discrete language. Assume that S is semi-recognized by a clocked program P in dimension d. Then*

S is fully-recognized in dimension $d+1$ by some clocked program $\tilde{P}$ with the following property: for all $w \in \Sigma^*$

- *if $w \notin S$, $\tilde{P}$ refuses input $\mathcal{J}(w)$ and stops with all its real registers set to 0.*
- *if $w \in S$, $\tilde{P}$ accepts input $\mathcal{J}(w)$ and stops with its first real register set to $\mathcal{J}(w)$ and all its other real registers set to 0.*

3.3.2 Climbing up the arithmetical hierarchy

The natural consequence of the transformation from semi-recognition to full-recognition proved in the previous section is the possibility of climbing up the arithmetical hierarchy: going from semi-recognition to full recognition allows to climb one level of the arithmetical hierarchy, and by recurrent application of this principle, allows to recognize all sets of the arithmetical hierarchy:

Theorem 3.3 *Let S be a discrete language. Assume $S \in \Sigma_k$, $k \geq 1$. Then S is semi-recognized by a clocked RCT program of dimension $1 + k$.*

The proof is immediate from theorem 3.1 and from the following lemma with $B = \emptyset$:

Lemma 3.4 *Assume that B is a discrete language such that all the languages of Σ_1^B are semi-recognized by some clocked RCT program in dimension $d' \geq 2$.*

Let S be a discrete language with $S \in \Sigma_k^B$, $k \in \mathbb{N}, k \geq 1$. Then S is semi-recognized by a clocked RCT program in dimension $d' + k - 1$.

Proof: We prove the assertion by induction over k. Case $k = 1$ is true by hypothesis.

Assume $k \geq 2$ and the hypothesis at rank $k-1$. Let $S \in \Sigma_k^B$. There exists $S' \in \Sigma_{k-1}^B$ such that $x \in S \Leftrightarrow \exists n \in \mathbb{N} \ < \overline{n}, x > \notin S'$: see [16]. By induction hypothesis S' is semi-recognized in dimension $k + d' - 2$ by a clocked RCT program P_{k-1}. Let $\tilde{P_{k-1}}$ be the marked program that one gets by applying lemma 3.3 on program P_{k-1}.

S is semi-recognized by the following RCT program P_k:

Algorithm 5 Program P_k

$$\left[\begin{array}{l}(w, w') \\ \mapsto (< \overline{0}, w >, w') \\ \textbf{where} \ \ w, w' \in \Sigma^*\end{array}\right]$$

while ($\tilde{P_{k-1}}$ accepts)

$$\left[\begin{array}{l}(< \overline{n}, w >, w') \\ \mapsto (< \overline{n+1}, w >, w') \\ \textbf{where} \ \ w, w' \in \Sigma^*, n \in \mathbb{N}\end{array}\right]$$

end while

Accept

□

3.4 RCT machines and the hyper-arithmetical hierarchy

Now we prove that RCT machines can recognize some hyper-arithmetical sets: the idea is to build more and more powerful machines that write digit by digit one of their real register.

In next subsection, we start by showing a technical lemma. In the next subsection, we show how to write some digit of a real register. In the following subsection, we prove that, whenever one can enumerate a set, one can build a machine that outputs a real encoding the set. Finally, in subsection 3.4.4, we use this principle to build machines that recognize some hyper-arithmetical sets in higher and higher levels of the hyper-arithmetical hierarchy.

3.4.1 Realizing any 2PDA program in time kx_{d+1}

Let M be a two-stack pushdown automaton. Assume that M always halts. By lemma 3.1, one can build a RCT program P that simulates M. By the discussion of the previous section, P' defined by $(P/x_{d+1}) * Div2$ simulates P (and M).

We want a program P'' equivalent to P' but where we have the guarantee that when P'' halts the value of x_{d+1} is equal to its original value. This is the following technical lemma whose proof can be found in the technical report [6] and consists in adding to P' some instructions that undo the $Div2$ subprograms before accepting:

Lemma 3.5 *Let $d \geq 2$. Let M be an 2PDA. Assume that, for all $w_1 \in \Sigma^\omega, w_2 \in \Sigma^*$, M maps (w_1, w_2) to $(f_1(w_1, w_2), f_2(w_1, w_2)) \in \Sigma^\omega \times \Sigma^*$: that is to say, when M is started with stacks w_1, w_2, M eventually halts with stacks $f_1(w_1, w_2), f_2(w_1, w_2)$.*

There exists some $k_M \in \mathbb{R}^+$ and a RCT machine M' of dimension $d+1$ that, for all $w_1 \in \Sigma^\omega, w_2 \in \Sigma^*$, for all $y_3, \ldots, y_d \in [0, 1]$, for all $n \in \mathbb{N}$, maps

$$(\mathcal{J}(w_1)/2^n, \mathcal{J}(w_2)/2^n, y_3/2^n, \ldots, y_d/2^n, 1/2^n)$$

to

$$(\mathcal{J}(f_1(w_1, w_2))/2^n, \mathcal{J}(f_2(w_1, w_2))/2^n, y_3/2^n, \ldots, y_d/2^n, 1/2^n)$$

in a time bounded above by $k_M/2^n$.

We use the following convention:

Convention 3.2 *We denote by*

$$\left(\begin{array}{l}(w_1,w_2)\\ \mapsto (w_1',w_2')\\ \textbf{where}\quad \textit{"conditions"}\end{array}\right)|x_{d+1}$$

a RCT program of dimension $d+1$ given by lemma 3.5 that for all $w_1, w_1' \in \Sigma^\omega$, $w_2, w_2' \in \Sigma^*$ verifying "conditions", and for all $y_3, \ldots, y_d \in [0,1]$, for all $n \in \mathbb{N}$, maps $(\mathcal{J}(w_1)/2^n, \mathcal{J}(w_2)/2^n, y_3/2^n, \ldots, y_d/2^n, 1/2^n)$ to $(\mathcal{J}(w_1')/2^n, \mathcal{J}(w_2')/2^n, y_3/2^n, \ldots, y_d/2^n, 1/2^n)$ in a time bounded by $k/2^n$ for some k.

3.4.2 Setting the m^{th} digit of a real in time $k/2^m$ for some k

Now, we show that one can write some particular digit of a real register: we show that one can build a RCT machine of dimension $d+2$ that, on input $m \in \mathbb{N}$, add $1/2^m$ to real register x_{d+2} in a time proportional to $maximum(1/2^m, x_{d+1})$:

Lemma 3.6 *Let $\#, \$ \in \Sigma$ be two distinct letters of Σ used as delimiters.*

For all $d \geq 2$, there exists some $k \in \mathbb{R}^+$ and a RCT machine $WriteDigit_{d+2}$ of dimension $d+2$ that, for all $y_3, \ldots, y_d, y_{d+2} \in [0,1], m, n \in \mathbb{N}, w \in \Sigma^\omega, w' \in \Sigma^*$, maps

$$(\mathcal{J}(\#^n\overline{m}\$w)/2^n, \mathcal{J}(w')/2^n, y_3/2^n \ldots, y_d/2^n, 1/2^n, y_{d+2})$$

to

$$(\mathcal{J}(\#^n\overline{m}\$w)/2^n, \mathcal{J}(w')/2^n, y_3/2^n, \ldots, y_d/2^n, 1/2^n, y_{d+2} + 1/2^m)$$

in a time upper bounded by $k1/2^{minimum(m,n)}$.

Proof: The general idea is to do some $Mul2^{d+1}/Div2^{d+1}$ instructions in order to get $x_{d+1} = 1/2^m$, then to do a $x_{d+2} := x_{d+2} + x_{d+1}$ $[x_{d+1}]$ instruction, and then to do some $Div2^{d+1}/Mul2^{d+1}$ instructions to come back to $x_{d+1} = 1/2^n$.

Assume without loss of generality that one can find two distinct letters $\uparrow$ and $\downarrow$ in Σ different from letter \$ and from letter $\#$.

$WriteDigit_{d+2}$ is the following program, where RCT instructions $Top^i(j)?$ $Pop^i(j)$, $Push^i(j)$ are defined in figure 3:

Algorithm 6 $WriteDigit_{d+2}$

$$\left(\begin{array}{l}(\#^n\overline{m}\$w, w')\\ \mapsto (move_1\$move_2\$w'\$\#^n\overline{m}\$w, \epsilon)\\ \qquad w \in \Sigma^\omega, w' \in \Sigma^*, m, n \in \mathbb{N}\\ \qquad move_1, move_2 \in \Sigma^*\\ \textbf{where}\quad (move_1, move_2) = \left\{\begin{array}{ll}(\downarrow^{m-n}, \uparrow^{m-n}) & \text{if } m > n\\ (\uparrow^{n-m}, \downarrow^{n-m}) & \text{if } m < n\\ (\epsilon, \epsilon) & \text{if } m = n\end{array}\right.\end{array}\right)|x_{d+1}$$

$$/*\ Map \left\{\begin{array}{l}x_1 = \mathcal{J}(\#^n\overline{m}\$w)/2^n\\ x_2 = \mathcal{J}(w')/2^n\\ x_{d+1} = 1/2^n\end{array}\right. \quad to$$

$$\left\{\begin{array}{l}x_1 = \mathcal{J}(move_1\$move_2\$w'\$\#^n\overline{m}\$w)\\ \quad /2^n\\ x_{d+1} = 1/2^n\end{array}\right.$$

in a time bounded by $k_1 1/2^n$ for some k_1: see lemma 3.2./*

GoUpOrDown /*Call some $Mul2^{d+1}/Div2^{d+1}$ instructions to get $x_{d+1} = 1/2^m$*/

$x_{d+2} := x_{d+2} + x_{d+1}\ [x_{d+1}]$ /* *Add* $1/2^m$ *to* x_{d+2}*/

GoUpOrDown /**Call some* $Div2^{d+1}/Mul2^{d+1}$ *instructions to get* $x_{d+1} = 1/2^n$*/

$$\left(\begin{array}{l}(w'\$\#^n\overline{m}\$w, \epsilon)\\ \mapsto (\#^n\overline{m}\$w, w')\\ \textbf{where}\quad w \in \Sigma^\omega, w' \in \Sigma^*, m, n \in \mathbb{N}\end{array}\right)|x_{d+1}$$

/**Set* $x_1 = \mathcal{J}(\#^n\overline{m}\$w)/2^n$, $x_2 = \mathcal{J}(w')/2^n$ *in time bounded by* $k_2 1/2^n$ *for some* k_2*. */

where program *GoUpOrDown* is the following RCT program:

Algorithm 7 program *GoUpOrDown*

```
if (Top^1(↑))/x_{d+1}
    then
    while ((Top^1(↑))/x_{d+1}) do
        (Pop^1(↑))/x_{d+1}
        Mul2^{d+1}
    end while
end if
if (Top^1(↓))/x_{d+1}
    then
        while ((Top^1(↓))/x_{d+1}) do
            (Pop^1(↓))/x_{d+1}
            Div2^{d+1}
        end while
    end if
    (Pop^1($))/x_{d+1}
```

The execution of the calls to program *GoUpOrDown* are done in a time upper bounded by $k_3 1/2^{minimum(m,n)}$ for some k_3. As a consequence, there exists some $k \in \mathbb{R}^+$ such that the time of execution of program $WriteDigit_{d+2}$ is bounded above by $k1/2^{minimum(m,n)}$.

□

3.4.3 Outputting reals encoding languages

We will encode languages into real numbers of $[0, 1]$ as follows:

Definition 3.8 (Encoding by $\mathcal{L}$) *Let $\Sigma = \{0, 1, \ldots, n_\Sigma\}$ be the fixed alphabet. Assume an enumeration of the words of Σ^* is fixed. Let α and β be two letters of Σ with $\alpha \neq \beta$.*

- *Let $L \subset \Sigma^*$ be a language. We denote by w_L the infinite word $a_0a_1a_2\ldots a_i\ldots$ such that, for all $i \in \mathbb{N}$, $a_i = \alpha$ (respectively: $a_i = \beta$) iff the i^{th} word of Σ^* is in L (respectively: is not in L)*
- *Denote by $\mathcal{P}(\Sigma^*)$ the class of the languages: $\mathcal{P}(\Sigma^*) = \{L | L \subset \Sigma^*\}$.*
  We denote by $\mathcal{L} : \mathcal{P}(\Sigma^*) \to [0, 1]$ the mapping that maps L to $\mathcal{L}(L) = \mathcal{J}(w_L)$ for all L.

A RCT machine M_f is said to enumerate language $L(w)$ on inputs $L \subset \Sigma^*, w \in \Sigma^*$, if M_f computes some function $f : \mathbb{N} \times \Sigma^* \times \Sigma^\omega \to \Sigma^*$ in the following sense: for all $n \in \mathbb{N}, w \in \Sigma^*, y_2 \in \Lambda, L \subset \Sigma^*$, M_f maps

$$(\mathcal{J}(\#^n\$w\$w_L)/2^n, y_2/2^n, ., \ldots, ., 1/2^n)$$

to

$$(\mathcal{J}(f(n, w, w_L)\$w_L)/2^n, y_2/2^n, ., \ldots, ., 1/2^n)$$

and such that the image of f is $L(w)$ in the following sense:

$$L(w) = \{w' | w' \in \Sigma^* \ \wedge \ \exists n \in \mathbb{N} \ f(n, w, w_L) = w'\}$$

We prove that if one has a machine M_f of dimension $d+1$ that enumerates a language $L(w)$ on inputs L, w, then one can build a RCT machine of dimension $d+2$ that, on inputs w and w_L, outputs w and $w_{L(w)}$: in other words, if one has a machine of dimension $d+1$ that enumerates some language $L(w)$ on inputs L, w, then one can build a machine of dimension $d+2$ that maps w and an encoding of L to w and an encoding of $L(w)$ (the idea is to use the previous subsection to build a machine that writes a real number digit by digit in finite time):

Lemma 3.7 *Let $d \geq 2$. Let $\$, \#$ be two letters of Σ used as delimiters.*

Assume that some machine M_f enumerates $L(w)$ on inputs L, w in time $1/2^n$: there exists a function $f : \mathbb{N} \times \Sigma^* \times \Sigma^\omega \to \Sigma^*$, a constant $k_f \in \mathbb{R}^+$ and a RCT machine M_f of dimension $d+1$ that, for all $n \in \mathbb{N}, w \in \Sigma^*, L \subset \Sigma^*, y_2 \in \Lambda$, maps

$$(\mathcal{J}(\#^n \$ w \$ w_L)/2^n, y_2/2^n, ., \ldots, ., 1/2^n)$$

to

$$(\mathcal{J}(f(n, w, w_L) \$ w_L)/2^n, y_2/2^n, ., \ldots, ., 1/2^n)$$

in a time bounded above by $k_f 1/2^n$ where $L(w) = \{w' | w' \in \Sigma^* \ \wedge \ \exists n \in \mathbb{N} \ f(n, w, w_L) = w'\}$.

Then there exists a RCT machine M'_f of dimension $d+2$ that for all discrete language $L \subset \Sigma^*$, for all word $w \in \Sigma^*$ and real $y_2 \in \Lambda$, maps $(\mathcal{J}(w \$ w_L), y_2, \ldots.)$ to $(\mathcal{J}(w \$ w_{L(w)}), y_2, 0, \ldots, 0)$ in a bounded time.

Proof: The general idea is to write a program that, on inputs $x_1 = \mathcal{J}(w \$ w_L)$, $x_2 = \mathcal{I}(w')$, using lemma 3.6, writes digit by digit onto its real register x_{d+2} the real value of $\mathcal{J}(w \$ w' \$ w_{L(w)})$.

Denote by $number : \Sigma^* \to \mathbb{N}$ the function that maps any word $w \in \Sigma^*$ onto its number in the fixed enumeration of the words of Σ^*. For $k \in \mathbb{N}, w \in \Sigma^*$, denote $lttr(w, k)$ for k^{th} letter of word w. We assume fixed a recursive enumeration of the finite subsets of Σ^* similar to the one of [16]: for any integer $n \in \mathbb{N}$, D_n denotes the n^{th} finite subset of Σ^*.

M'_f is given by the following algorithm, where RCT instructions $Top^i(j)$, $Pop^i(j)$,$Push^i(j)$ are defined in figure 3 and integer b'_Σ is defined page 3:

Algorithm 8 Program M'_f

$$\left[\begin{array}{l} (w \$ w_L, w') \\ \mapsto (\$ w_L, w \$ w' \$ \overline{u_0}) \\ \textbf{where} \quad w_L \in \Sigma^\omega, w, w' \in \Sigma^* \\ \qquad\quad u_0 \in \mathbb{N}, D_{u_0} = \emptyset \end{array}\right]$$

/*Initialize the computation: set $n = 0$*/

$x_{d+1} := 1$ /*Set initial speed to 1*/

$x_{d+2} := 0$ /*Set x_{d+2} to 0*/

while (**true**) **do** /*While (true)*/

$$\left(\begin{array}{l} (\#^n \$ w_L, w \$ w' \$ \overline{u_n}) \\ \mapsto (\#^n \overline{b'_\Sigma * n - 2a_n} \$ w_L, w \$ w' \$ \overline{u_n}) \\ \qquad\quad w_L \in \Sigma^\omega, w, w' \in \Sigma^*, \\ \textbf{where} \quad n, u_n \in \mathbb{N}, a_n \in \Sigma \\ \qquad\quad a_n = \left\{\begin{array}{ll} \alpha & \text{if } n > length(w \$ w' \$) \\ \text{lttr(w\$w'\$,n)} & \text{if } n \leq length(w \$ w' \$) \end{array}\right. \end{array}\right) | x_{d+1}$$

$WriteDigit_{d+2}$ /*Set the n^{th} digit of x_{d+2} to default value a_n*/

$$\left(\begin{array}{l} (\#^n \overline{m} \$ w_L, w \$ w' \$ \overline{u_n}) \\ \mapsto (\#^n \$ w \$ w_L, \#^n \$ w \$ w' \$ \overline{u_n}) \\ \textbf{where} \quad w_L \in \Sigma^\omega, w, w' \in \Sigma^*, n, u_n \in \mathbb{N} \end{array}\right) | x_{d+1}$$

M_f /*Get w'' the n^{th} word of the enumeration given by M_f*/

$$\left(\begin{array}{l} (w''\$w_L, \#^n\$w\$w'\$\overline{u_n}) \\ \mapsto (\#^n\overline{m}\$w_L, already in\$w\$w'\$\overline{u_{n+1}}) \\ \qquad w, w', w'' \in \Sigma^*, w_L \in \Sigma^\omega, \\ \qquad n, u_n, u_{n+1} \in \mathbb{N}, already in \in \Sigma \\ \qquad m = b'_\Sigma * (number(w) + length(w\$w'\$) + 1) \\ \qquad\ -2(\beta - \alpha) \\ \textbf{where} \quad \text{if } w'' \in D_{u_n} \text{ then} \\ \qquad already in = \#,\ u_{n+1} = u_n \\ \qquad \text{else} \\ \qquad already in = \$, D_{u_{n+1}} = D_{u_n} \cup \{w''\} \end{array}\right) | x_{d+1}$$

if $((Top^2(\$))/x_{d+1})$ **then** /*If word w'' has not been yet output*/

$(Pop^2(\$))/x_{d+1}$

$WriteDigit_{d+2}$ /*Then change the digit of real register x_{d+2} corresponding to w'' from value α to value β.*/

else

$(Pop^2(\#))/x_{d+1}$ /*Else do nothing*/

end if

$$\left(\begin{array}{l} (\#^n\overline{m}\$w_L, \$w\$w'\$\overline{u_{n+1}}) \\ \mapsto (\#^{n+1}\$w_L, w\$w'\$\overline{u_{n+1}}) \\ \textbf{where} \quad w_L \in \Sigma^\omega, w, w' \in \Sigma^*, u_{n+1}, n, m \in \mathbb{N} \end{array}\right) | x_{d+1}$$

/*Do $n := n + 1$*/

$Div2^{d+1}$

end while

$limit^*$:

$x_1 := x_{d+2}$ /*Copy the result into x_1*/

$x_{d+2} := 0$ /*Set x_{d+2} to 0*/

$$\left[\begin{array}{l} (w\$w'\$w_L, \epsilon) \\ \mapsto (w\$w_L, w') \\ \textbf{where} \quad w_L \in \Sigma^\omega, w, w' \in \Sigma^* \end{array}\right]$$

/*Put back the result in the good form*/

□

3.4.4 Climbing up the hyper–arithmetical hierarchy

In this subsection, we apply recurrently lemma 3.7 to reach higher and higher levels of the hyper–arithmetical hiearchy: we produce machines that output $\mathcal{L}(L)$ for some discrete languages $L \in \Sigma^*$ in higher and higher levels of the hyper-arithmetical hierarchy.

We define ω^0 as 1. We denote by $+_o$ the addition between constructive ordinal numbers: $+_O$ is a recursive function that verifies $z +_O z' \in O$ and $|z +_0 z'| = |z| + |z'|$ for all $z, z' \in O$: see [16].

We prove by induction that for all $k \geq 0$ one can find a constructive ordinal $z_k \in O$ with $|z_k| = \omega^k$ and a machine that, for all constructive ordinal z, enumerates language $H(z +_o z_k)$ on inputs $H(z), z$:

Lemma 3.8 *Let $k \geq 0$.*

There exists a constructive ordinal $z_k \in O$ with $|z_k| = \omega^k$, and a machine that for all constructive ordinal z enumerates language $H(z +_0 z_k)$ on inputs $H(z), z$: there exists $f_k : \mathbb{N} \times \Sigma^* \times \Sigma^\omega \to \Sigma^*$, there exists some fixed constant $C_k \in \mathbb{R}^+$ and a RCT machine M'_k of dimension $2k + 3$ that, for all

$n \in \mathbb{N}, w \in \Sigma^*, L \subset \Sigma^*, y_2 \in \Lambda$ *maps*

$$(\mathcal{J}(\#^n\$w\$w_L)/2^n, y_2/2^n, ., \ldots, ., 1/2^n)$$

to

$$(\mathcal{J}(f_k(n, w, w_L)\$w_L), y_2/2^n, ., \ldots, ., 1/2^n)$$

in a time bounded above by $C_k/2^n$, *where for all* $z \in O$, $H(z +_0 z_k) = \{w' | w' \in \Sigma^* \wedge \exists n \in \mathbb{N}\ f(n, \overline{z}, w_{H(z)}) = w'\}$,

Furthermore, f *is also computed by a RCT machine* M_k *of dimension* $2k+2$ *in the following sense: for all* $n \in \mathbb{N}, w \in \Sigma^*, L \subset \Sigma^*, y_2 \in \Lambda$, M_k *maps*

$$(\mathcal{J}(\#^n\$w\$w_L), y_2, ., \ldots, .)$$

to

$$(\mathcal{J}(f_k(n, w, w_L)\$w_L), y_2, ., \ldots, .)$$

Proof: It is known that there exists a recursive g such that for all $L \in \Sigma^*, m \in \mathbb{N}$, the range of function $\phi^L_{g(m)}$ is is W^L_m [16]. Let M_{univ} be a 2PDA such that on input $\#^n\$\overline{m}$, M_{univ} simulates $M^L_{g(m)}$ on input n, answering the queries of $M^L_{g(m)}$ on any word w' to its oracle L by comparing the digit of w_L corresponding to w' to letter α. M_{univ} is a 2PDA that for all $n \in \mathbb{N}, m \in \mathbb{N}, w' \in \Sigma^*, L \subset \Sigma^*$, maps $(\#^n\$\overline{m}\$w_L, w')$ to $(w^m_n\$w_L, w')$, where $w^m_n = \phi^L_{g(m)}(n)$. Denote by P_{univ} the RCT machine given by lemma 3.1 that simulates M_{univ}. Using lemma 3.5, for all $d \geq 2$ one can build a RCT machine P'^{d+1}_{univ} of dimension $d+1$ that, for all $n \in \mathbb{N}, m \in \mathbb{N}, y_2 \in \Lambda, L \subset \Sigma^*$, maps $(\mathcal{J}(\#^n\$\overline{m}\$w_L)/2^n, y_2/2^n, ., \ldots, ., 1/2^n)$ to $(\mathcal{J}(w^n_m\$w_L)\ /2^n, y_2/2^n, ., \ldots, ., 1/2^n)$ in time $k/2^n$ for some fixed $k \in \mathbb{R}^+$. Apply lemma 3.7 on this machine: one gets a RCT machine of dimension $d+2$ that, for all $L \subset \Sigma^*, m \in \mathbb{N}, y_2 \in \Lambda$, maps $(\mathcal{J}(\overline{m}\$w_L), y_2, ., \ldots, .)$ to $(\mathcal{J}(\overline{m}\$w_{W^L_m}), y_2, 0, \ldots, 0)$ in finite time. Denote this RCT machine by P''^{d+2}_{univ}.

Now, we are ready to prove the assertions of the lemma by induction over k:

Assume $k = 0$: it is known that there exists $m_0 \in \mathbb{N}$, such that for all $L \subset \Sigma^*$, $L' = H^L(1) = W^L_{m_0}$ [16]. Consider M as the 2PDA that on input $(\#^n\$w\$w_L, w')$ calls M_{univ} with input $(\#^n\$\overline{m_0}\$w_L, w')$. M_0 is the RCT machine of dimension 2 given by lemma 3.1 that simulates M, and M'_0 is the RCT machine of dimension 3 given by lemma 3.5 that simulates M.

Assume now $k \geq 1$: denote by $\Pi_1, \Pi_2, \Pi_3 : \mathbb{N} \to \mathbb{N}$ some recursive functions such that

$$n \mapsto (\Pi_1(n), \Pi_2(n), \Pi_3(n))$$

is a bijective recursive function from $\mathbb{N}$ to $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$. Denote by f and l the recursive functions of lemma 3.9.

Lemma 3.7 can be applied on machine M'_{k-1}: one gets a RCT machine M''_{k-1} of dimension $2k+2$ that for all $z' \in O$, for all $y_2 \in \Lambda$ maps $(\mathcal{J}(\overline{z'}\$w_{H(z')}), y_2, ., \ldots, .)$ to $\mathcal{J}(\overline{z'}\$w_{H(z' +_{z_{k-1}})}, y_2, 0, \ldots, 0)$ in a bounded time. Set $z_k = 3.5^{n_k}$ where $\phi_{n_k}(0) = 1$ and $\phi_{n_k}(n+1) = \phi_{n_k}(n) +_0 z_{k-1}$ for all $n \in \mathbb{N}$.

M_k is given by the following program, where RCT instructions $Top^i(j)$, $Pop^i(j)$,$Push^i(j)$ are defined in figure 3:

Algorithm 9 Program M_k

$$\left[\begin{array}{l} (\#^n\$\overline{z}\$w_L, w') \\ \mapsto (\overline{z}\$w_L, \#\$\#^n\$\overline{z}\$w') \\ \textbf{where}\ \ w_L \in \Sigma^\omega, w' \in \Sigma^*, z \in O, n \in \mathbb{N} \end{array}\right]$$

while $(Top^2(\#)?)$ **do** /**Call* $\Pi_1(n)$ *times program* M''_{k-1} */

M''_{k-1}

$$\left[\begin{array}{l} (\overline{z_p}\$w_L, \#\$\#^n\$\overline{z}\$w') \\ \mapsto (\overline{z_{p+1}}\$w_L, continue\$\#^n\$\overline{z}\$w') \\ \\ \qquad w_L \in \Sigma^\omega, w' \in \Sigma^*, z_p, z_{p+1} \in O, n \in \mathbb{N} \\ \textbf{where}\ \ z_{p+1} = z_p +_0 y_{k-1} \\ \qquad \text{if } z_{p+1} \leq_o \phi_{n_k}(\Pi_1(n)) \\ \qquad\ \text{then } continue = \# \\ \qquad\ \text{else } continue = \$ \end{array}\right]$$

end while /*Here, if the initial input was $\#^n\$\overline{z}\$w_{H(z)}$, $z \in O$, we have $x_1 = \mathcal{J}(\overline{z_p}\$w_{H(z_p)})$ where $z_p = \phi_{n_k}(\Pi_1(n))$ */

$$\left[\begin{array}{l} (\overline{z_p}\$w_L, \$\$\#^n\$\overline{z}\$w') \\ \mapsto (\#^{\Pi_2(n)}\$\overline{m}\$w_L, \#^n\$\overline{z}\$w'\$\overline{z''}\$\overline{z_p}) \\ \qquad w_L \in \Sigma^\omega, w' \in \Sigma^*, z \in O, m, n \in \mathbb{N} \\ \textbf{where} \quad z'' = \phi_{f(z_p)}(\Pi_3(n)) \text{ (we have } z' \leq_0 z_p) \\ \qquad m = l(z'', z_p) \text{ is the integer such that } W_m^{H(z_p)} = H(z'') \end{array}\right]$$

P_{univ} /* *Compute* $z'' = \phi_{f(z_p)}(\Pi_3(n))$. *We have* $z'' \leq_o z_p$. *Get* w'' *the* $\Pi_2(n)^{th}$ *word of* $H(z'')$*/

$$\left[\begin{array}{l} (w''\$w_L, \#^n\$\overline{z}\$w'\$\overline{z''}\$\overline{z_p}) \\ \mapsto (\overline{m}\$w_L, \#^n\$w''\$\overline{z''}\$w') \\ \qquad w_L \in \Sigma^\omega, w', w'' \in \Sigma^*, z, z'', z_p \in O, m, n \in \mathbb{N} \\ \textbf{where} \\ \qquad m = l(z, z_p) \text{ is the integer such that } W_m^{H(z_p)} = H(z) \end{array}\right]$$

P''^{2k+2}_{univ} /*Put back in x_1 the value of $w_{H(z)}$*/

$$\left[\begin{array}{l} (\overline{m}\$w_L, \#^n\$w''\$\overline{z''}\$w') \\ \mapsto (< w'', z'' > \$w_L, w') \\ \textbf{where} \quad w_L \in \Sigma^\omega, w', w'' \in \Sigma^*, z'' \in O, n \in \mathbb{N} \end{array}\right]$$

/*Output $< w'', z'' > \$w_{H(z)}$ */

M'_k is easy to obtained from the program of M_k: add the instruction $x_{2k+3} := 1[1]$ at the beginning of program M_k, replace

$$\left[\begin{array}{l} (\overline{m}\$w_L, \#^n\$w''\$\overline{z''}\$w') \\ \mapsto (< w'', z'' > \$w_L, w') \\ \textbf{where} \quad w_L \in \Sigma^\omega, w', w'' \in \Sigma^*, z'' \in O, n \in \mathbb{N} \end{array}\right]$$

by

$$\left(\begin{array}{l} (\overline{m}\$w_L, \#^n\$w''\$\overline{z''}\$w') \\ \mapsto (< w'', z'' > \$w_L, \uparrow^{\Pi_1(n)} \$w') \\ \textbf{where} \quad w_L \in \Sigma^\omega, w', w'' \in \Sigma^*, z'' \in O, n \in \mathbb{N} \end{array}\right) | x_{d+1}$$

replace in program M_k all the other instructions of type

$$\left[\begin{array}{l} (w_1, w_2) \\ \mapsto (w'_1, w'_2) \\ \textbf{where} \quad \text{conditions} \end{array}\right]$$

by

$$\left(\begin{array}{l} (w_1, w_2) \\ \mapsto (w'_1, w'_2) \\ \textbf{where} \quad \text{conditions} \end{array}\right) | x_{d+1}$$

replace P_{univ} by P'^{2k+3}_{univ}, P''^{2k+2}_{univ} by $P''^{2k+2}_{univ}/x_{2k+3}$, and replace the call to M''_{k-1} by the instructions $M''_{k-1}; Div 2^{2k+3}$, and add the program *GoUpOrDown* defined page 15 at the end of the program.

□

Where lemma 3.9 is the following and is proved in [6]:

Lemma 3.9
- *For all* $y \in O$, $\{\overline{x} | x \in O \ \wedge \ x <_0 y\}$ *is recursively enumerable uniformly in* y: *there exists a recursive* $f : \mathbb{N} \to \mathbb{N}$ *such that for all* $y \in O$, *the range of* $\phi_{f(y)}$ *is* $\{\overline{x} | x <_0 y\}$
- *Given* $x, y \in O$ *with* $x \leq_o y$ *or* $y \leq_0$ *as input, a Turing machine can effectively tell if* $x = y$, *if* $x <_0 y$ *or if* $y <_0 x$.

- *There exists a recursive l such that, for all z_1, z_2 with $z_1 \leq_0 z_2$, $H(z_1) = W^{H(z_2)}_{l(z_1,z_2)}$*

By lemma 3.8 we have a program that, for all $z \in O$, enumerates $H(z +_0 z_k)$ when it gets $H(z), z$ as input. By feeding to this program $H(1) = \emptyset$ and 1 as input one can enumerate $H(z_k) = H(1 +_0 z_k)$. Since Σ_{ω^k} is the class of the languages that are recursively enumerable in $H(z_k)$, all the sets of Σ_{w^k} can be recognized by some RCT program of dimension $2k + 2$: this is the following lemma.

Lemma 3.10 *Let $k \geq 1$. Any language of Σ_{ω^k} can be semi-recognized by a RCT machine of dimension $2k + 2$.*

Proof: Consider the machine M_k and the integer $z_k \in O$ of lemma 3.8. M_k is of dimension $2k + 2$, and $|z_k| = \omega^k$. Let L be a language of Σ_{ω^k}. L is recursively enumerable in $H(z_k)$ by some Turing machine $M^{H(z_k)}$ with oracle $H(z_k)$.

See that there exists a recursive g such that, for all $u \in \Sigma^*, v \in O$, $u \notin H(v) \Leftrightarrow g(u) \in H(2^v)$: see [16].

L is semi-recognized by the RCT machine of dimension $2k+2$ that simulates $M^{H(z_k)}$, simulating every query of $M^{H(z_k)}$ of type $< u, v > \in H(z_k)$? by a subprogram that runs M_k on input $x_1 = \mathcal{J}(\#^n\$\overline{1}\$w_\emptyset)$ for $n = 1, 2, \ldots$, until either $x_1 = \mathcal{J}(< u, v > \$w_\emptyset)$ or $x_1 = \mathcal{J}(< g(u), 2^v > \$w_\emptyset)$ is output.

□

We already now by theorem 3.3 some lower bounds for the computational power of PCD systems of dimensions 2 and 3. The previous lemma gives some lower bounds for PCD systems of even dimensions greater than 4. Using lemma 3.4 with $B = H(\omega^k)$, one can easily get some lower bounds for all dimensions that can be summarized by:

Theorem 3.4 *Let $k \geq 0$.*

- *Any language of Σ_{ω^k} can be semi-recognized by a RCT machine of dimension $2k + 2$.*
- *Any language of Σ_{ω^k+1} can be semi-recognized by a RCT machine of dimension $2k + 3$.*

4 PCD systems can simulate RCT Machines

In this section, we prove that PCD systems can simulate RCT machines: more detailed proofs can be found in [6].

4.1 Basic constructions

Let d be an integer. A *k-dimensional box of $\mathbb{R}^d$*, $k < d$, is given by (P, B) where P is a polyhedral subset of $\mathbb{R}^d$ of dimension k, and B is a affine basis $(O, e_1, e_2, \ldots, e_d)$ of $\mathbb{R}^d$, $O \in P$, such that $(O, e_1, \ldots, e_k)$ is an affine basis[1] of P.

The *point of coordinates $(x_1, \ldots, x_k)$ on (P, B)* denotes the point of P of coordinates $(x_1, \ldots, x_k, 0, \ldots, 0)$ in basis B.

Let $\mathcal{H}$ be a PCD system of dimension d and d' be an integer with $d' < d$. Let $I = (f, c)$ be an assignment[2] of dimension d'. $\mathcal{H}$ is said to *realize assignment I via input port In and output port Out* if there exist some d'-dimensional boxes In and Out of $\mathbb{R}^d$, such that, for all $x \in [0, 1]^{d'}$, the trajectory of $\mathcal{H}$ starting from the point of coordinates $x \in [0, 1]^{d'}$ on In at time 0 reaches Out in point of coordinates $f(x)$ on Out at time $c(x)$: see figure 4 and figure 5 for some examples of assignments realized by some PCD systems.

For all $d' \in \mathbb{N}$, denote by $Id_{d'}$ the identity function of $[0, 1]^{d'}$. Let $I = (R, c)$ be a test of dimension d'. $\mathcal{H}$ is said to *realize test I* if there exist three d'-dimensional boxes In, Out^+,Out^- of $\mathbb{R}^d$ such that for all x such that $R(x)$ is true, $\mathcal{H}$ realizes assignment $(Id_{d'}, c)$ via input port In and output port Out^+, and for all x such that $R(x)$ is false, $\mathcal{H}$ realizes assignment $(Id_{d'}, c)$ via input port In and output port Out^-: see figure 4 for the example of test $y > \lambda$? [1].

All the linear machine instructions can be realized by some PCD systems:

[1] That is to say B is an affine basis of V, where V is the minimal affine variety such that $P \subset V$.

[2] We do not assume here that I is necessarily an admissible assignment.

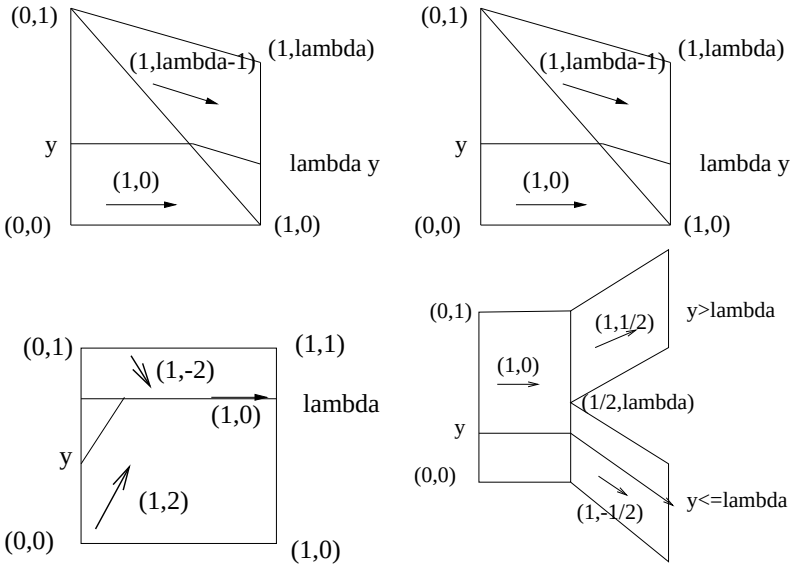

Figure 4: PCD systems realizing instructions $y := \lambda y$ [1], $y := y + \lambda$ [1],$y := \lambda$ [1] and $y > \lambda$? [1] respectively in dimension 2.

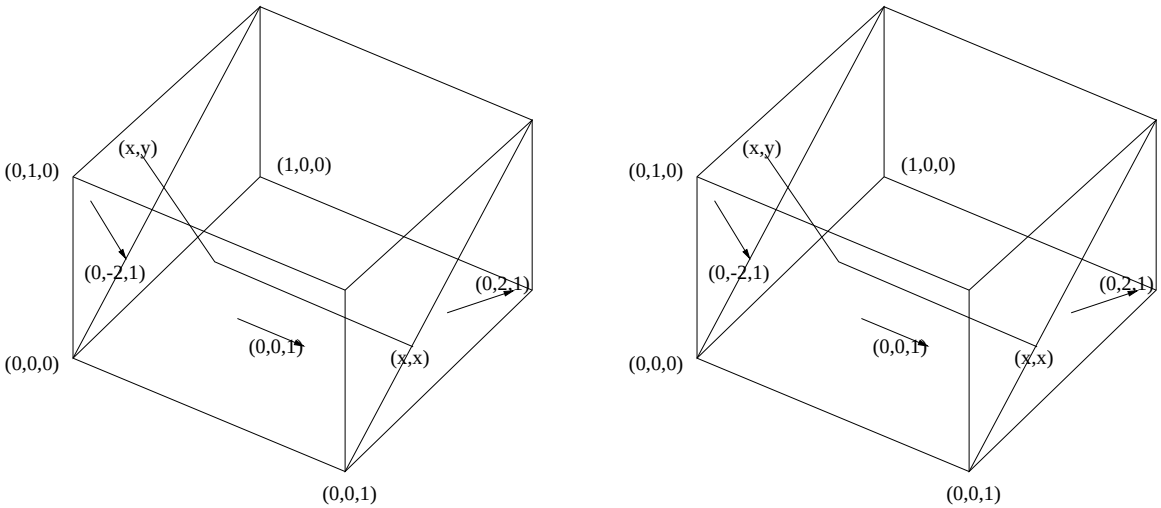

Figure 5: PCD systems realizing $y := x$ [1] and $y := y + x$ [1] in dimension 3.

Lemma 4.1 (Basic linear machine instructions) *Let $d \in \mathbb{N}$. Let $d' \geq d+1$.*

Let $I = (f, c) \in Assgnmt_d$ be an admissible assignment (respectively: Let $I = (R, c) \in Test_d$ be an admissible test) of dimension d. Assume that I is one of the "linear machine instructions" of definition 3.5.

For all $\mu \in \mathbb{R}^+$, one can build a PCD system of dimension d' that realizes assignment $I = (f, \mu c)$ (resp. that realizes test $I = (R, \mu c)$).

The proof is easy: for $\mu = 1$, generalize the PCD systems of figure 4 and figure 5 to higher dimensions; for $\mu \neq 1$, multiply in addition all the slopes by $1/\mu$.

One can artificially slow down a trajectory: by taking some big enough $k, k' \in \mathbb{R}$ and by constructing a PCD system like the one of figure 6 one gets:

Lemma 4.2 (Delay) *Let $d \in \mathbb{N}$. Let $d' \geq d+1$ be an integer.*

For any affine function $c : [0,1]^d \to \mathbb{R}^+$, one can build in dimension d' a delay of time c plus some constant*: one can build a PCD system of dimension d' that realizes assignment $(Id_d, c + \lambda)$ for some $\lambda \in \mathbb{R}^+$.*

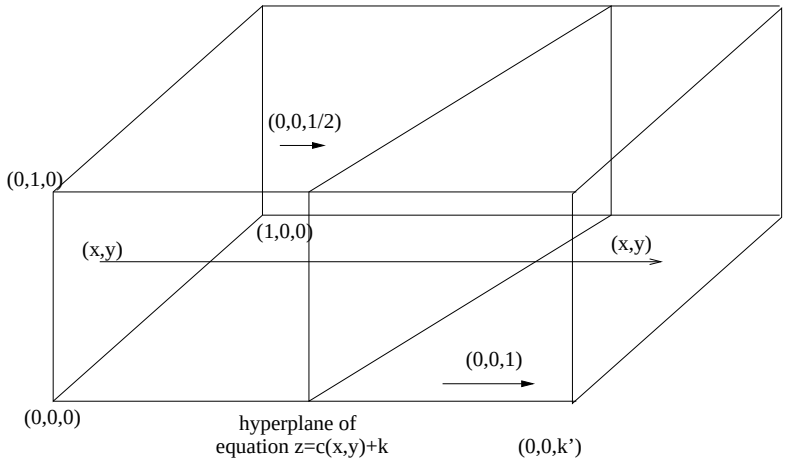

Figure 6: A PCD system realizing delay $c(x, y)$ plus some constant in dimension 3, where $c : \mathbb{R}^2 \to \mathbb{R}$ is an affine function.

One can build some "paths" using the regions of PCD systems:

Lemma 4.3 (Paths) *Let $d \in \mathbb{N}$. Let $d' \geq d+1$ be an integer. Let In and Out be two d-dimensional boxes of $\mathbb{R}^d$.*

For all $\mu \in \mathbb{R}^+$, one can build in dimension d' a path *of time μ between In and Out: one can build a PCD system of dimension d' that realizes the assignment (Id_d, μ) via input port In and via output port Out.*

Proof: Using "angles" and "straight parts" it is easy build some regions that bring any point of coordinates x on In to point of coordinates x on Out as in figure 7. The time taken by a trajectory to go from point of coordinates x on In to point of coordinates x on Out through these regions is some affine function $t : \mathbb{R}^d \to \mathbb{R}^+$ of x. Using lemma 4.2, insert in one of the regions some regions that realize a delay of time $-t(x)$ plus some constant. In the obtained PCD system, the time required by a trajectory to go from In to Out is now a constant k independent of $x \in In$. Multiply all the slopes by μ/k to set this constant to time μ: see figure 7.

□

4.2 PCD systems can simulate RCT machines

We need to prove that all the non-linear RCT machines instructions can be implemented by PCD systems. In particular, we must prove that one can realize the "Zeno instructions".

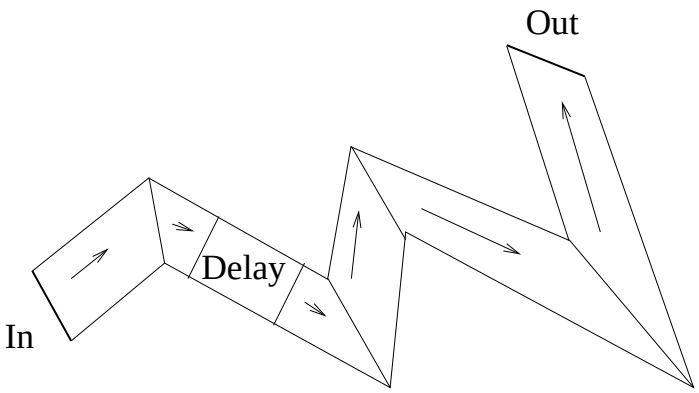

Figure 7: A path between 1-dimensional port In and 1-dimensional port Out.

Definition 4.1 (Homogenization,Translation) *Let R' be a region of a PCD system of dimension d': that is to say R' is a polyhedral subset of $\mathbb{R}^{d'}$ with some associated slope $s' = (s'_1, \ldots, s'_{d'}) \in \mathbb{R}^{d'}$.*

We define the translation and the homogenization of region R':

- *The* translation of region R' *is the region R whose equation in the canonical basis of $\mathbb{R}^{d'+1}$ is:*

$$R = \{(x_1, \ldots, x_{d'+1}) | 0 \leq x_{d'+1} \leq 1 \ \wedge \ (x_1, \ldots, x_{d'}) \in R'\}$$

  and whose slope is $s = (s'_1, \ldots, s'_{d'}, 0)$.

- *If $I' = (P', B')$ is a d-dimensional box of $\mathbb{R}^{d'}$, the* translation of I *is the $d+1$-dimensional box defined by $I = (P, B)$, P is the translation of P', $B = (O, e_1, \ldots, e_{d'+1})$ where $B' = (0, e_1, \ldots, e_{d'})$ and $e_{d'+1}$ is the vector of coordinates $(0, \ldots, 0, 1)$ in the canonical basis of $\mathbb{R}^{d'+1}$.*

- *The* homogenization of region R' *is the region R whose equation in the canonical basis of $\mathbb{R}^{d'+1}$ is:*

$$R = \{(x_1, \ldots, x_{d'+1}) | 0 < x_{d'+1} \leq 1 \ \wedge \ (x_1/x_{d'+1}, \ldots, x_{d'}/x_{d'+1}) \in R'\}$$

  and whose slope is $s = (s'_1, \ldots, s'_{d'}, 0)$.

- *If $I' = (P', B')$ is a d-dimensional box of $\mathbb{R}^{d'}$, the* homogenization of I *is the $d+1$-dimensional box defined by $I = (P, B)$, P is the homogenization of P', $B = (0, e_1, \ldots, e_{d'+1})$ where $B' = (0', e_1, \ldots, e_{d'})$, point O and vector $e_{d'+1}$ have coordinates $(0, \ldots, 0)$ and $(o_1, o_2, \ldots, o_{d'}, 1)$ respectively in the canonical basis of $\mathbb{R}^{d'+1}$ where $(o_1, o_2, \ldots, o_{d'})$ are the coordinates of O' in the canonical basis of $\mathbb{R}^{d'}$.*

We show now that translations correspond to embedding instructions into higher dimensions, and homogenizations correspond to transforming instructions I into instructions I/x_{d+1}:

Lemma 4.4 *Let $\mathcal{H}'$ be a PCD system of dimension d' realizing assignment (f, c) (respectively: test (R, c)) of dimension d via input port In and via output port Out (resp. via output ports Out^+, Out^-).*

- *Let $\mathcal{H}$ be the PCD system of dimension $d'+1$ whose regions are the translations of the regions of $\mathcal{H}'$.*

  $\mathcal{H}$ realizes assignment (f, c) (respectively: test (R, c)) considered as an instruction of dimension $d+1$ via input port the translation of In and via output port the translation of Out (resp. via output ports the translations of Out^+ and of Out^-).

- *Let $\mathcal{H}$ be the PCD system of dimension $d'+1$ whose regions are the homogenizations of the regions of $\mathcal{H}'$.*

  $\mathcal{H}$ realizes assignment $(f/x_{d+1}, c/x_{d+1})$ (respectively: test $(R/x_{d+1}, c/x_{d+1})$) via input port the homogenization of In and via output port the homogenization of Out (resp. via output ports the homogenizations of Out^+ and of Out^-).

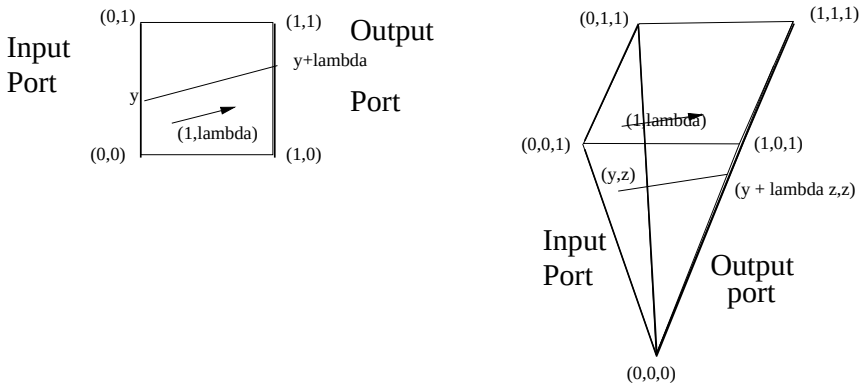

Figure 8: A PCD system realizing instruction I defined by $y := y + \lambda$ [1] in dimension 2 (on left) and its homogenization realizing instruction I/z in dimension 3 (I/z is $y := y+\lambda$ $[z]$ whenever $z \in (0,1]$, $y/z \in [0,1]$).

Proof: Immediate from the definitions: see figure 8 for an example. □

Here is the main theorem of the section: one can simulate a RCT machine[3] by a PCD system:

Theorem 4.1 *Let M be a RCT machine[4] of dimension d.*
One can build a PCD system $\mathcal{H}$ of dimension $d+1$ that simulates M.

Proof: Assume M is of dimension d. Denote $M = (Q, q_0, q_f^+, q_f^-, limit^*, \delta)$. For all $q \in Q$, denote $\delta(q) = (q^+, q^-, Instr_q)$.

We prove the theorem by induction over the dimension d and by structural induction over the program of M: we prove that for all RCT machine M of dimension d one can build a PCD system $\mathcal{H}$ of dimension $d+1$: to each state $q \in \mathbb{Q}$ one can associate a d-dimensional box I_q and some regions of $\mathcal{H}$ such that $\mathcal{H}$ realizes the assignment (respectively: the test) $Instr_q$ via input port I_q and via output port I_{q^+} (resp. via output ports I_{q^+} and I_{q^-}) using these regions. Moreover, $\mathcal{H}$ has a box I_* corresponding to the limit state.

Denote $Q' \subset Q$ for the subset of the states of M such that $q \in Q'$ iff $Instr_q$ is either a special instruction, or a Zeno instruction, or obtained from an instruction of dimension $d-1$ which is not a linear machine instruction nor a subprogram of dimension $d-1$.

See that Q' is empty if $d \leq 2$: if $d \leq 2$ skip the five following paragraphs.

Consider $M' = (Q, q'_0, q_f^+, q_f^-, limit^*, \delta')$ as a program of dimension $d-1$ where, for all $q \in Q$, $\delta'(q) = (q', q'', Instr')$ iff $q^+, q^- \in Q', q' = q^+, q'' = q^-$, and $Instr' = (f, c)$ if $Instr_q = (f/x_d, c/x_d)$, $Instr' = (R, c)$ if $Instr_q = (R/x_d, c/x_d)$, $Instr' = (Id_{d-1}, 1)$ if $Instr_q$ is a special instruction of type $x_d := x_d/2$ $[x_d]$ or of type $x_d := 2x_d[x_d]$, $Instr' = (x_1 := x_1$ $[x_k])$ if $Instr_q$ is a special instruction of type $x_d := x_d + \lambda x_k, 2 < k < d$, $Instr' = (f, c)$ if $Instr_q$ is obtained from the instruction (f, c) of dimension $d-1$, and $Instr' = (R, c)$ if $Instr_q$ is obtained from the test (R, c) of dimension $d-1$.

By induction hypothesis one can build a PCD system $\mathcal{H}'$ of dimension d that simulates M'. To each state $q' \in Q'$ of M' corresponds a $d-1$-dimensional port $I'_{q'}$. Moreover, some $d-1$ dimensional box I'_* corresponds to the limit state.

Consider $\mathcal{H}$ as the PCD system built as follows: for all $q \in Q'$, for all region $R' \subset \mathbb{R}^d$ of PCD system $\mathcal{H}'$ associated to q:

- if $Instr_q$ corresponds to a special instruction of type $x_d := x_d/2$ $[x_d]$ or $x_d := 2x_d[x_d]$, or to a Zeno instruction, then add to $\mathcal{H}$ the homogenization of region R' and take I_q as the homogenization of I'_q.

[3] For the clarity of the exposure and for the concision of this paper, we omit intentionally to state that the RCT machine of dimension d must actually have some special properties in order to be simulated in dimension $d+1$ (these special properties allow to avoid some connection problems between d-dimensional paths in dimension $d+1$ that we intentionally do not mention in the proof). See technical report [6] for a complete proof.

[4] see footnote 3.

- if $Instr_q$ corresponds to an instruction obtained from an instruction of dimension $d-1$ or a special instruction of type $x_d := x_d + \lambda x_k, 2 < k < d$ then add to $\mathcal{H}$ the translation of region R' and take I_q as the translation of I'_q.

For each state $q \in Q'$ such that $Instr_q$ is a "special instruction" modify $\mathcal{H}$ as follows: if $Instr_q$ is of type $x_d := x_d/2$ $[x_d]$ or of type $x_d := 2x_d$ $[x_d]$ we can assume without loss of generality that one region already constructed R of slope s of $\mathcal{H}$ corresponding to state q is the homogenization of a region R' of $\mathcal{H}'$ of slope s' and that R' is of type $R' = A' + [0,1]^d$ for some point $A' \in \mathbb{R}^d$, where s' is of type $s' = (v, 0, \ldots, 0)$, for some $v \in \mathbb{R}^+$. In that case, replace the slope s of R by $s = (v, 0, \ldots, 0, -v/2)$ if $Instr_q$ is of type $x_d := x_d/2$ $[x_d]$ and by $s = (v, 0, \ldots, 0, v)$ if $Instr_q$ is of type $x_d := 2x_d$ $[x_d]$. If $Instr_q$ is of type $x_d := x_d + \lambda x_k, 2 < k < d$, we can assume without loss of generality that one region already constructed R of slope s of $\mathcal{H}$ corresponding to state q is the translation of the translation of the translation of the ... translation of the homogenization of some region R'' of $\mathbb{R}^k$ with slope s'' and that R'' is of type $R'' = A'' + [0,1]^k$ for some point $A'' \in \mathbb{R}^k$, where s'' is of type $s'' = (v, 0, \ldots, 0)$, for some $v \in \mathbb{R}^+$. In that case, replace the slope s of R by $s = (v, 0, \ldots, 0, v)$.

All the ports I_q constructed up to know are either the homogenization of I'_q or the translation of I'_q. By lemma 4.4, for all $q \in Q'$, it is true that $\mathcal{H}$ realizes the assignment (respectively: the test) $Instr_q$ via d-dimensional input port I_q and via d-dimensional output port I_q^+ (resp. via output ports I_q^+, I_q^-).

Now, for all $q \in Q, q \notin Q'$ does the following: choose any arbitrary d-dimensional port I_q of $\mathbb{R}^{d+1}$ not containing the point of coordinates $(0, \ldots, 0)$. See that $Instr_q$ corresponds to an instruction $Instr_q$ that is either equivalent to a linear machine instruction or either a subprogram of dimension d or $d-1$.

- If $Instr_q$ corresponds to a subprogram of dimension d (respectively: $d-1$), by induction hypothesis, one can build some regions of a PCD system $\mathcal{H}_{instr_q}$ of dimension $d+1$ (resp. d) that realizes $Instr_q$. Add to $\mathcal{H}$ the regions of $\mathcal{H}_{instr_q}$ (resp. the translation of the regions of $\mathcal{H}_{instr_q}$) and a path of time $1/2$ between the d-dimensional port I_q of $\mathcal{H}$ and the input port of $\mathcal{H}_{instr_q}$ (resp. and the translation of the input port of $\mathcal{H}_{instr_q}$) and a path of time $1/2$ between the output port of $\mathcal{H}_{instr_q}$ (resp. between the translation of the output port of $\mathcal{H}_{instr_q}$) and the d-dimensional port I_{q^+} of $\mathcal{H}$.
- If $Instr_q$ corresponds to a linear machine assignment (f, c) (respectively: to a test (R, c)), by lemma 4.1, build a PCD system $\mathcal{H}_{instr_q}$ of dimension $d+1$ that realizes $(f, c/3)$ (resp. $(R, c/3)$). Add to $\mathcal{H}$ the regions of $\mathcal{H}_{instr_q}$ and a path of time $1/3$ between the $d+1$-dimensional port I_q of $\mathcal{H}$ and the input port of $\mathcal{H}_{instr_q}$ and a path (respectively: and two paths) of time $1/3$ between the output port of $\mathcal{H}_{instr}$ and the d-dimensional port I_{q^+} of $\mathcal{H}$ (resp. and the d-dimensional ports I_{q^+} and I_{q^-}).

Define I_* as $\{(x_1, \ldots, x_{d+1}) | 0 \leq x_{d+1} \leq 1 \ \wedge \ (x_1, \ldots, x_d) \in I'_*\}$: I_* is the translation of I'_*. Add a path from port I_* to the port I_{limit^*}.

One gets a PCD system $\mathcal{H}$ that simulates M: $\mathcal{H}$ realizes the assignment corresponding to the execution of M via input port I_{q_0} and via output port $I_{q_f^+}$. This proves the assertion for dimension d from the assertion in dimension $d-1$.

□

As a consequence, we get immediately from theorem 4.1 and from theorem 3.4:

Theorem 4.2 *Let $k' \geq 0$.*

- *Any language of $\Sigma_{\omega^{k'}}$ can be semi-recognized by a PCD system of dimension $2k'+3$ in finite continuous time.*
- *Any language of $\Sigma_{\omega^{k'}+1}$ can be semi-recognized by a PCD system of dimension $2k'+4$ in finite continuous time.*

5 Upper bounds on the computational power of PCD systems

In this section, we give some upper bounds on the computational power of PCD systems.

To get these upper bounds we need to prove that if a PCD system recognizes a language S then S is in the hyper–arithmetical hierarchy and we need to characterize in which level of the hierarchy S is: assume language S is semi-recognized by PCD system $\hat{H} = (\mathbb{R}^d, f, \mathcal{J}, x^1, x^0)$. S is the set of the words

w such that the trajectory Φ_w starting at $(\mathcal{J}(w), 0, \ldots, 0)$ reaches x^1. Our idea used to get the upper bounds is to prove that one can build some machines, with appropriate oracles in the hyper–arithmetical hierarchy, that on input $w \in \Sigma^*$, simulate trajectory Φ_w and tell if Φ_w reaches or not x^1. The existence of these machines will prove that S is in the hyper–arithmetical hierarchy and will give a level of the hyper-arithmetical hierarchy containing S.

In subsection 5.1, we define a way to represent real points and parameterized sequences of real points. Then we introduce the notion of *sampling of a PCD system*: the idea is that a sampling gives for any trajectory a sequence of couples date position reached by the trajectory. We define what a Zeno sampling is and we show that a Zeno sampling is necessarily converging. We also show how one can build effectively a sampling of a PCD system.

In subsection 5.2, we have some geometrical considerations: first we associate to any point of the space of any PCD system $\mathcal{H}$ of dimension d an integer d' that we call its *local dimension*. We prove a first property and we show that the limit of a sequence of points of some fixed local dimension is necessarily of higher local dimension. Then we show that the effective sampling constructed at the end of subsection 5.1 can be improved in order to get a sampling up to local dimension 3.

In subsection 5.3 we show how one can build some samplings of higher local dimension using oracles in the hyper–arithmetical hierarchy: we start by giving the general idea of the method. We recall some general properties of the hyper–arithmetical hierarchy. Then we define the hyper-jump operation on samplings and discuss its properties. Next, we define the cycle-free operation on samplings and its properties. Finally, using these transformations recurrently, we get upper-bounds on the computational power of PCD systems and we reach a full characterization of the computational power of PCD systems.

5.1 Sampling trajectories

In this section we start by giving a way to represent real points of the space and parameterized sequences of real points of the space as languages over alphabet Σ. We define the sampling of a trajectory and we show that a Zeno sampling is necessarily converging. Finally, we show how to build effectively a sampling of a PCD system.

5.1.1 Representing reals by languages

Even if a trajectory starts from some rational point, the trajectory Φ can reach some intermediate points with real coordinates: we need to find a way to represent such real points of the space: we represent every point x of the space $\mathbb{R}^d$, $d \in \mathbb{N}$ by a language: a *rational polyhedron* is any polyhedron whose coefficients are rational numbers: that is to say such that it can be expressed as an union of intersections of linear inequalities with rational coefficients. Denote by $\mathcal{P} \subset \Sigma^*$ the set of the rational polyhedral of $\mathbb{R}^d$. We assume that a representation of the elements of $\mathcal{P}$ as words of Σ^* is fixed.

Definition 5.1 (Encoding reals by languages) *Let $d \in \mathbb{N}$. Let $x \in \mathbb{R}^d$. $\lceil x \rceil$ is the language defined as the set of the words $w \in \Sigma^*$ that encodes a rational polyhedron P of $\mathbb{R}^d$ such that $x \in P$. $\lceil x \rceil$ is called the* language associated to x.

In a similar way, we need to encode parameterized sequences of real points: for us a parameterized sequence of real points is a function f from $\mathbb{N} \times \Sigma^* \times \mathbb{R}^d$ to $\mathbb{R}^d$ for some integer d: when parameters x and w are fixed, it corresponds to the sequence of points $(f(n, w, x))_{n \in \mathbb{N}}$ of $\mathbb{R}^d$.

Definition 5.2 (Encoding parameterized sequences)

- *A* parameterized sequence *is a function h from $\mathbb{N} \times \Sigma^* \times \mathbb{R}^d$ to $\mathbb{R}^d$ for some integers d.*
- *For all $x \in \mathbb{R}^d$, the* relation associated to h at real parameter x *is the language $R_h(x) \subset \Sigma^*$ defined by $R_h = \{< n, w, P > | n \in \mathbb{N}, w \in \Sigma^*, P \in \mathcal{P}$ and $P \in \lceil h(n, w, x) \rceil\}$.*
- *For all $k \in \mathbb{N}, x \in \mathbb{R}^d$, the* relation associated to h at real parameter x up to rank k *is the language $R_h^{<k}(x) \subset \Sigma^*$ defined by $R_h = \{< n, w, P > | n \in \mathbb{N}, n < k, w \in \Sigma^*, P \in \mathcal{P}$ and $P \in \lceil h(n, w, x) \rceil\}$.*

5.1.2 Sampling PCD systems

We define now the samplings of a PCD system: the idea is that a sampling gives for any trajectory of the PCD system a sequence of couples date position reached by the trajectory. More precisely, given x and t, and a polyhedron Q, a sampling g gives a sequence $(g(k, Q, t, x))_{k \in \mathbb{N}}$ of couples $(t_k, x_k)_{k \in \mathbb{N}}$ such

that the trajectory Φ of $\mathcal{H}$ starting at x at time t is at time t_k in point x_k, with in addition the following properties: the sequence $(t_k)_{k\in\mathbb{N}}$ is increasing and if t_{sup} denotes $sup_k(t_k)$, then either Φ reaches Q before time t_{sup} and we have $t_{sup} = t_k$ for some k, or Φ does not reach Q before time t_{sup} and sequence $(t_k)_{k\in\mathbb{N}}$ is strictly increasing. When $(t_k)_{k\in\mathbb{N}}$ is strictly increasing and t_{sup} is finite the sampling is said to be Zeno.

Here are the formal definitions:

Definition 5.3 (Sampling of a PCD system) *Assume a PCD system $\mathcal{H} = (X, f)$ of dimension d is fixed.*

- *A* sampling of $\mathcal{H}$ *is a mapping g from $\mathbb{N}\times\mathcal{P}\times\mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ with the following properties: assume $Q \in \mathcal{P}$, $t \in \mathbb{R}$, $x \in \mathbb{R}^d$ are fixed. Denote by Φ the trajectory of $\mathcal{H}$ starting from x at time t.*
  - *For all $k \in \mathbb{N}$, $g(k, Q, t, x) = (t_k, x_k)$, for some $t_k \in \mathbb{R}, x_k \in \mathbb{R}^d$ with $\Phi(t_k) = x_k$.*
  - *$t_0 = t, x_0 = x$.*
  - *$t_{k+1} \geq t_k$ for all $k \in \mathbb{N}$.*
  - *Only one of two following cases hold:*
    - *there is some $k_0 \in \mathbb{N}$ with $x_{k_0} \in Q \cup NoEvolution(\mathcal{H})$, and for all $k \geq k_0$, $x_k = x_{k_0}, t_k = t_{k_0}$.*
    - *$t_k < t_{k+1}$ for all $k \in \mathbb{N}$ and Φ does not reach $Q \cup NoEvolution(\mathcal{H})$ at some time $t < sup_{k\in\mathbb{N}} t_k$.*
- *If $t_k < t_{k+1}$ for all $k \in \mathbb{N}$ and if $sup_{k\in\mathbb{N}}\ t_k$ is finite, then g is said to be* Zeno at parameters Q, t and x.

See that a sampling is a parameterized real sequence. We see now that a sampling that is Zeno is necessarily converging to some point of the space:

Lemma 5.1 (A Zeno sampling converges to some point) *Assume a sampling g is Zeno at some parameters Q, t, x. Take the notations of definition 5.3. Denote $t_{sup} = sup_k\ t_k$. Then necessarily*

- *Trajectory Φ reaches some point x_{sup} at time t_{sup}: $\Phi(t_{sup}) = x_{sup}$ for some x_{sup}.*
- *This point is the limit of the sampling: $t_{sup} = lim_{k\in\mathbb{N}}\ t_k$, $x_{sup} = lim_{k\in\mathbb{N}}\ x_k$.*

Proof: By a well-known result of analysis, since Φ is a continuous function and has a bounded right derivative, Φ can always be extended to a continuous function defined at time t_{sup}. Hence, one can always assume that Φ is defined at time t_{sup}.

Now, since Φ is a continuous function, and since $t_{sup} = lim_k\ t_k$, we must have $x_{sup} = \Phi(t_{sup}) = lim_k\ \Phi(t_k) = lim_k\ x_k$.

□

5.1.3 Sampling effectively a PCD system

For all PCD system $\mathcal{H}$, one can effectively compute a sampling of $\mathcal{H}$:

Proposition 5.1 *Let $\mathcal{H}$ be a PCD system.*

There exists a Turing machine that computes a sampling of $\mathcal{H}$: there exists a Turing machine M with oracle and a sampling $g : \mathbb{N}\times\mathcal{P}\times\mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ of $\mathcal{H}$ such that, for all $t \in \mathbb{R}$, for all $x \in \mathbb{R}^d$, when M has oracle $\lceil(t,x)\rceil$ it computes the relation $R_g(t,x)$ associated to g at real parameter (t,x).

Proof: Assume a PCD system $\mathcal{H}$ is fixed. Assume we want to produce effectively a sampling g of $\mathcal{H}$: the most natural algorithm is the following: on inputs t, x, Q and k, simulate k discrete steps (that is to say k region crossing) of the trajectory Φ starting at x at time t. If Q is reached by the trajectory at some discrete step $k' \leq k$, return the date and position of the intersection. Otherwise return (t_k, x_k) where t_k and x_k are the date and the position of Φ at the end of this k discrete steps simulation.

This natural algorithm computes a sampling g of $\mathcal{H}$.

This algorithm is not directly a Turing machine algorithm since it requires to manipulate some real points of the space. But by using symbolically the representation of real points by their associated languages defined in 5.1 it can be transformed into a Turing machine algorithm that computes the relation associated to g.

□

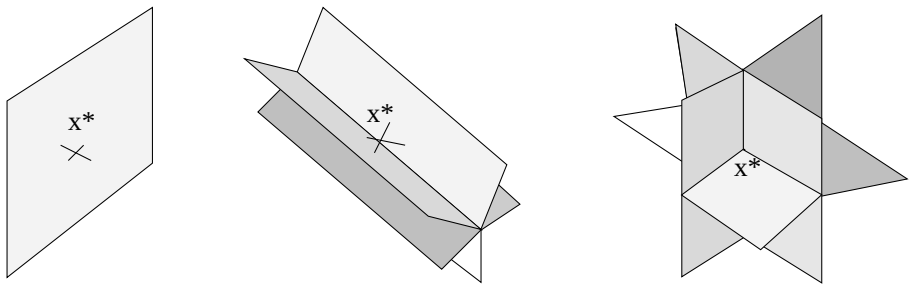

Figure 9: From left to right: x^* is of local dimension $1, 2, 3$ in a PCD system of dimension 3.

5.2 Geometrical considerations

We now have some geometrical considerations: first we associate to any point of the space of a PCD system $\mathcal{H}$ of dimension d an integer d' $(1 \leq d' \leq d)$ that we call its *local dimension*. Then we show a technical property and we show that the limit of a sequence of points of some fixed local dimension is necessarily of higher local dimension. Then we see that one can build effectively a sampling of any PCD system up to local dimension 3.

5.2.1 Local dimension

We define the local dimension of any point as follows:

Definition 5.4 (Local dimension) *Let $\mathcal{H} = (X, f)$ be a PCD system in dimension d. Let x^* be a point of X. Let Δ be a polyhedral subset $\Delta \subset X$ of maximal dimension $d - d'$ $(1 \leq d' \leq d)$ such that there exists an open convex polyhedron $V \subset X$, with $x^* \in \Delta \cap V$, $\Delta \subset V$, and such that, for any region F of $\mathcal{H}$, $F \cap V \neq \emptyset$ implies $\Delta \subset \overline{F}$ ($\overline{F}$ is the topological closure of F).*

x^* is said to be of local dimension *d': see figure 9.*

Note that given a rational PCD system $\mathcal{H} = (X, f)$ and an integer d' one can effectively compute $LocDim(\mathcal{H}, d')$ defined as the set of the points $x \in X$ that have a local dimension equal to d'.

5.2.2 First property

The idea behind the definition of the local dimension is given by the next proposition: if a point x^* is of local dimension d' in a PCD of dimension d, to study the trajectories in a neighborhood of x^*, one can restrict the attention to a PCD system of dimension d'.

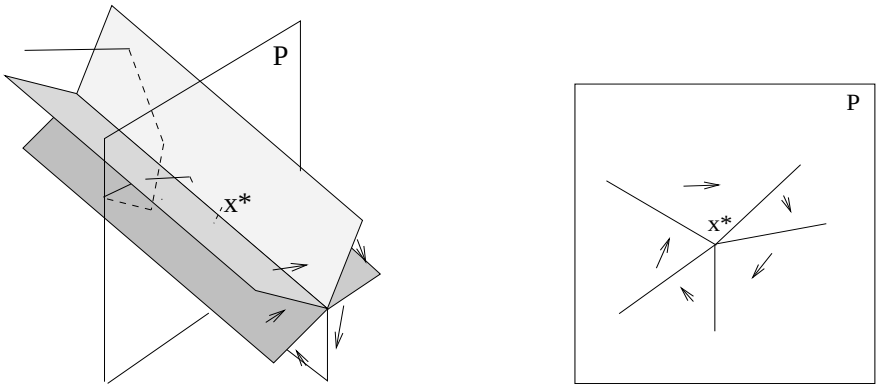

Figure 10: Proposition 5.2: if x^* is of local dimension 2 in a PCD $\mathcal{H}$ of dimension 3, the projections on P of the trajectories of $\mathcal{H}$ in a neighborhood V of x^* are the trajectories of a PCD system $\mathcal{H}_{x^*}$ of dimension 2.

Proposition 5.2 *Let $\mathcal{H} = (X, f)$ be a PCD system in dimension d. Let x^* be a point of local dimension d' with $d' < d$. Call P the affine variety of dimension d' which is the orthogonal of Δ in x^*. It is possible to construct a PCD system $\mathcal{H}' = (X' = \mathbb{R}^{d'}, f')$ in dimension d' such that the trajectories of $\mathcal{H}'$ are the orthogonal projections on P of the trajectories of $\mathcal{H}$ in V.*

Proof: Choose an affine basis of $\mathbb{R}^d$ of the form $(x^*, e_1, e_2, \ldots, e_{d'}, \ldots, e_d)$ with $(x^*, e_1, e_2, \ldots, e_{d'})$ taken as a basis of P and $(x^*, e_{d'+1}, \ldots, e_d)$ taken as a basis of Δ. Call $p : \mathbb{R}^d \to \mathbb{R}^{d'}$ the projection that sends $(x_1, x_2, \ldots, x_d)$ to $(x_1, \ldots, x_{d'})$. By hypothesis, in V the regions are organized as a 'pencil of regions': therefore speed in point $(x_1, x_2, \ldots, x_{d'}, \ldots, x_d) \in V$ does not depend on the coordinates $x_{d'+1}, x_{d'+2}, \ldots, x_d$. The reader can check that $\mathcal{H}' = (X' = \mathbb{R}^{d'}, f')$ where $f'(x_1, x_2, \ldots, x_{d'}) = p(f(x_1, x_2, \ldots, x_{d'}, 0, \ldots, 0))$ is a solution. See figure 10.

□

For any point x^*, the corresponding open convex polyhedron V is denoted by V_{x^*}. See that when the local dimension of x^* is the dimension of the space $(d' = d)$ one can always assume V_{x^*} small enough such that x^* is the only point of local dimension d' in $\overline{V_{x^*}}$. $\mathcal{H}'$, Δ are respectively denoted by $\mathcal{H}_{x^*}$ and Δ_{x^*}. If $d' < d$ we denote by p_{x^*} and q_{x^*} the functions that map all point $x \in X$ onto its orthogonal projection on P and onto its orthogonal projection on Δ respectively. If $d' = d$, we define p_{x^*} and q_{x^*} as respectively the identity function and the null function.

5.2.3 Local dimension of a limit of a sequence of points

We show that the limit of a sequence of real points of some fixed local dimension is necessarily of higher local dimension:

Lemma 5.2 *Let $\mathcal{H} = (X, f)$ be a PCD system of dimension d. Let d' be an integer. Let Φ be a trajectory of $\mathcal{H}$.*

Assume that $(t_i)_{i\in\mathbb{N}} \in \mathbb{R}^+$ is a bounded increasing sequence . Assume that for all $i \in \mathbb{N}$, $\Phi(t_i)$ is of local dimension d'. Denote $t_{sup} = sup_{i\in\mathbb{N}}\ t_i$ and $x_{sup} = \Phi(t_{sup})$.

Then necessarily $x_{sup} = \Phi(t_{sup})$ is of local dimension $> d'$.

Proof: Denote by d'' the local dimension of x_{sup}. By continuity of Φ, there exists $i_0 \in \mathbb{N}$ such that for all $i \geq i_0$, $\Phi(t_i) \in V_{x_{sup}}$. For all $i \geq i_0$, point $\Phi(t_i)$ is of local dimension d' and is in $V_{x_{sup}}$. By considering the dimension of affine subspace $\Delta_{\Phi(t_i)}$, for any $i \geq i_0$, one gets $d'' \geq d'$.

Assume $d'' = d'$: it is easy to see that x_{sup} is necessarily the only point of local dimension d' in $p_{x_{sup}}(V_{x_{sup}})$. As a consequence, for all $i \geq i_0$, $\Phi(t_i) \in \Delta_{x_{sup}}$. Denote by t_{first} the first point of local dimension $d' = d''$ reached by Φ after time t_{i_0}: $t_{first} = \inf\{t | t \in \mathbb{R} \ \wedge\ t > t_{i_0} \ \wedge\ \Phi(t) \in LocDim(\mathcal{H}, d')\}$. By lemma 5.2, $\Phi' = p_{x_{sup}}(\Phi)$ must be a trajectory of $\mathcal{H}_{x_{sup}}$. Φ does not reach any point of local dimension d' at any time t with $t_{i_0} < t < t_{first}$. One has $\Phi'(t_{i_0}) = \Phi'(t_{first})$. As a consequence, for all $n \in \mathbb{N}$, $\Phi'(t_{i_0} + n(t - t_{i_0})) = \Phi'(t_{i_0})$ and all the points of local dimension d' reached by Φ at some time $t > t_{i_0}$ must necessarily be reached at some time t of type $t = t_{i_0} + n(t - t_{i_0})$ for some $n \in \mathbb{N}$. In particular, sequence $(t_i)_{i\in\mathbb{N}}$ must be a subsequence of sequence $(t_{i_0} + i(t - t_{i_0}))_{i\in\mathbb{N}}$. We reach a contradiction, since $(t_i)_{i\in\mathbb{N}}$ is assumed to be a bounded sequence. Hence, it is not possible that $d'' = d'$ and necessarily $d'' > d'$.

□

The following corollary is an easy consequence:

Corollary 5.1 *Let $\mathcal{H} = (X, f)$ be a PCD system of dimension d. Let d' be an integer. Let Φ be a trajectory of $\mathcal{H}$.*

Assume that $(t_i)_{i\in\mathbb{N}} \in \mathbb{R}^+$ is a bounded increasing sequence . Assume that for all $i \in \mathbb{N}$, $\Phi(t_i)$ is of local dimension $\geq d'$. Denote $t_{sup} = sup_{i\in\mathbb{N}}\ t_i$ and $x_{sup} = \Phi(t_{sup})$.

Then

- *necessarily $x_{sup} = \Phi(t_{sup})$ is of local dimension $> d'$.*
- *if d'' denotes the local dimension of x_{sup}, all but a finite number of the $x_i = \Phi(t_i), i \in \mathbb{N}$ are of local dimension $\leq (d'' - 1)$.*

5.2.4 Sampling effectively a PCD system up to local dimension 3

We start by the following definition:

Definition 5.5 (Sampling up to some local dimension) *A sampling g is said to be a* sampling up to local dimension d'*, where d' is an integer, if for all $Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$, whenever g is Zeno at parameters Q, t, x, its limit (that is to say the point reached at time $t_{sup} = sup_{k\in\mathbb{N}}\ t_k$ by the trajectory starting at x at time t that exists by lemma 5.1) is necessarily of local dimension $> d'$.*

Of course we can say that the effective samplings obtained in subsection 5.1.3 are samplings up to local dimension 1: but this says nothing!

Actually, one can slightly modify the algorithm described in subsection 5.1.3 to get more powerful samplings: using a lemma proved in [8] that shows that a trajectory converging toward some point of local dimension ≤ 3 has necessarily a cyclic signature and the fact that one can effectively decide if the signature of a trajectory is cyclic, we prove in [6] that the method of subsection 5.1.3 can be extended to provide samplings up to local dimension 3: this is the next proposition.

Proposition 5.3 ([6]) *Let $\mathcal{H}$ be a PCD system.*

There exists a Turing machine that computes a sampling of $\mathcal{H}$ up to local dimension 3: there exists a Turing machine M with oracle and a sampling $g : \mathbb{N} \times \mathcal{P} \times \mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ of $\mathcal{H}$ up to local dimension 3 such that, for all $t \in \mathbb{R}$, for all $x \in \mathbb{R}^d$, when M has oracle $\lceil (t, x) \rceil$ it computes the relation $R_g(t, x)$ associated to g at real parameter (t, x).

5.3 Building samplings up to higher local dimensions

In this subsection, we show that by using oracles in the hyper–arithmetical hierarchy one can get some samplings up to higher local dimensions. We start by giving the general idea of the method. Then we recall some results about the hyper–arithmetical hierarchy: we recall the well-known equivalence between logical definitions of sets and their levels in the arithmetical hierarchy. We prove that if a set is hyper-arithmetical in another hyper-arithmetical set, it is hyper-arithmetical of level the sum of the two levels: one can compose oracles by adding their levels. Then we define the *hyper-jump operation* on samplings. In the following two subsections, we show that the hyper-jump operation can be extended to the more powerful *cycle-free operation*. Finally, we use this transformation to get some upper bounds on the computational power of PCD systems.

5.3.1 General idea

Assume a PCD system $\mathcal{H}$ is fixed.

Using proposition 5.3, one can build a sampling g of $\mathcal{H}$ up to local dimension 3 that is computable by a Turing machine: as a consequence, given as input a starting date t and a starting point x, a Turing machine M can simulate the trajectory Φ of $\mathcal{H}$ starting at x at time t and give a list of couples date position (t_k, x_k) reached by Φ.

However, this simulation simulates Φ only up to time $t_{sup} = sup_{k\in\mathbb{N}}\ t_k$. If $t_{sup} = +\infty$, that is to say, if g is non-Zeno this is not restrictive: Turing machine M simulates completely trajectory Φ for all positive time.

But if $t_{sup} < +\infty$, that is to say if g is Zeno, Turing machine M simulates Φ only on time interval $[t, t_{sup})$. However, we know by lemma 5.1 that Φ reaches some point x_{sup} at time t_{sup}. But the simulation given by Turing machine M does not give this point nor the evolution of Φ after time t_{sup}.

We want to build a machine that computes a more powerful sampling: the idea is to use an oracle in the hyper-arithmetical hierarchy that, for all Q, t, x, tells whether sampling g is Zeno at parameters Q, t, x and when it is the case, that tells the value of the limit. We build a machine M' with this oracle that computes a sampling g' that extends sampling g as follows: M' simulates any trajectory Φ starting from some point x at time t by the following method: M' tests if the simulation given by g of the trajectory starting at x at time t is Zeno: that is to say, tests if g is Zeno at parameters Q, t, x. If it is so, M' computes (t', x') the limit of g and starts again the process from position x' at date t': M' tests if the simulation given by g of the trajectory starting at x' at time t' is Zeno: that is to say, tests if g is Zeno at parameters Q, t', x'. If it is so , it computes (t'', x'') the limit of g and starts again the process at position x'' and time t'': it tests if the simulation given by g of the trajectory starting at x'' at time

t'' is Zeno and if it is so computes the limit and starts again the process ... and so on ... If at some moment of this process, g is not Zeno at some tested parameters $Q, t^{(k)}, x^{(k)}$ then M' uses g to simulate the trajectory after time $t \geq t^{(k)}$.

The point is to see that one gets effectively a machine that computes a sampling g' that extends g and that by lemma 5.2 g' is a sampling up to a higher local dimension than g.

Hence, from a sampling g computed by some machine M, one can build a machine M' computing a sampling g' up to some higher local dimension. We call g' the *hyper-jump* of sampling g.

By applying this hyper-jump operation recurrently one can get samplings up to higher and higher local dimensions: take the hyper-jump of the hyper-jump of g and so on. This is the idea behind what we do in the next subsections to get more and more powerful samplings.

In fact, we will not use directly the hyper-jump transformation but a slight improvement of it, that we call the *cycle-free* transformation. Using this transformation instead of using the hyper-jump operation will give us directly optimal upper bounds.

5.3.2 Languages first order definable

We recall the well-known Tarski-Kuratowski equivalence between sets defined by a first order logical formula and arithmetical sets: we will not distinguish the relations on Σ^* from the languages over Σ^* : a relation R of arity k over Σ^* is considered as the language $\{< n_1, n_2, \ldots, n_k > | R(n_1, \ldots, n_k)\} \subset \Sigma^*$.

Recall what a first order definition of a set is:

Definition 5.6 (First order definition [16]) *Let $Fa_1 \ldots a_n$ a first–order logic expression with free variables $a_1 \ldots a_n$: that is to say $Fa_1 \ldots a_n$ is built up from quantifiers $\exists, \forall$, $=$, sentential connectives $\wedge, \vee, \Rightarrow, \neg$ and relation symbols R_1, R_2, ..., R_k*

Let the relation symbols $R_1, R_2, \ldots, R_k$ be interpreted as certain fixed relations $T_1, \ldots, T_k \subset \Sigma^*$.

Then the relation $R = \{< x_1, \ldots, x_n > | Fa_1 \ldots a_n$ is true over domain Σ^* when $a_1, \ldots, a_n$ are interpreted as $x_1, \ldots, x_n \in \Sigma^*$ respectively and $R_1, R_2, \ldots, R_k$ are interpreted as $T_1, \ldots, T_k \subset \Sigma^*$ respectively $\} \subset \Sigma^*$ is said to be definable by first order formula F from relations $T_1, \ldots, T_k$.

Hence in a first–order logic expression, the quantifications over functions are not allowed. All the quantifications are on variables and here, the variables are interpreted as words of Σ^*. As an example, if $T \subset \Sigma^*$ is some binary relation, then $\{n \in \Sigma^* | \exists t \in \Sigma^* \; T(n,t)$ *is true*$\}$ is first order definable by formula $\exists t \; R(n,t)$ from relation T.

Assume that $X \subset \Sigma^*$ is a recursively enumerable set (respectively: is a recursive set, is a Y-recursively enumerable set, is a Y-recursive set $(Y \subset \Sigma^*)$): $X = W_n$ (resp. $X = W_n^Y$) for some $n \in \mathbb{N}$. n is called a *recursively enumerable index* (resp: recursive index, Y-recursively enumerable index, Y-recursive index) of X.

The well-known Tarski-Kuratowski equivalence states that a set that is first order definable is necessarily in the arithmetical hierarchy and that the first order formula gives uniformly a level of the hierarchy containing the set:

Proposition 5.4 (Tarski-Kuratowski algorithm [16])

- *Let F be a first order formula. F can always be transformed into a first order formula in prenex form logically equivalent to F beginning with a quantifier $\exists$.*
- *Assume F is a first order formula in prenex form beginning with a quantifier $\exists$. Let $n \in N$ be the number of quantifier alternations[5] in formula F.*
  - *Let $R \subset \Sigma^*$ be a language defined by formula F from some recursive relations $T_1, \ldots, T_k$ (respectively: defined by formula F from some A-recursive relations $T_1, \ldots, T_k$).*
    Then R is in the arithmetical hierarchy (resp. in the A-arithmetical hierarchy): $R \in \Sigma_{n+1}$ (resp. $R \in \Sigma_{n+1}^A$).
  - *The dependence of R on relations $T_1, \ldots, T_k$ is uniform: assume first order formula F is fixed. There exists a recursive g_F, such that, for all $n_1, \ldots, n_k \in \mathbb{N}$ (resp. for all $n_1, \ldots, n_k \in \mathbb{N}$, for all $A \subset \Sigma^*$), if $n_1, \ldots, n_k$ are recursive (respectively: A-recursive) indexes of relations $T_1, \ldots, T_k$ respectively, then $g_F(n_1, \ldots, n_k)$ is an $H(y)$-recursively enumerable index (resp. is an $H^A(y)$-recursively enumerable index) of language R defined by formula F from relations $T_1, \ldots, T_k$.*

[5] The number of alternations is the number of pairs of adjacent but unlike quantifiers in the prefix of the prenex formula: see [16].

5.3.3 Compositions of oracles

We prove that if a set is hyper-arithmetical in another hyper-arithmetical set, then it is hyper-arithmetical of level the sum of the two levels: in other words we can compose oracles by adding their levels: see [16] or [6] for a proof.

Lemma 5.3 (Composition) *Let $X \subset \Sigma^*$.*

- *There exists a recursive g such that, for all $x, y \in O$, $H^{H^X(x)}(y) \leq_m H^X(x +_0 y)$ via $\Phi_{g(x,y)}$.*
- *This holds effectively on indexes: there exists a recursive h (resp. a recursive h') such that, for all $m, n \in \mathbb{N}, x, y \in O$, if m is some $H^X(x)$-recursively enumerable index of some set $S \subset \Sigma^*$, and if n is some $H^S(y)$-recursively enumerable index of some set $S' \subset \Sigma^*$, then $h(x, y, m, n)$ is an $H^X(x +_0 y)$-recursively enumerable index (resp. $H^X(x +_0 y +_0 2)$-recursive index) of S'.*

5.3.4 HyperJump operation

We define now formally what the hyper-jump transformation on samplings is:

Definition 5.7 (HyperJump operation) *Assume we have a sampling g of $\mathcal{H}$.*

We define $HyperJump[g] : \mathbb{N} \times \mathcal{P} \times \mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ as follows: assume parameters $Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$ are fixed.

- *Set $HyperJump[g](0, Q, t, x) = (t, x)$*
- *Let $k \geq 1$. Denote $HyperJump[g](k-1, Q, t, x) = (t_{k-1}, x_{k-1}), t_{k-1} \in \mathbb{R}, x_{k-1} \in \mathbb{R}^d$.*
  - *If g is Zeno at parameters Q, t_{k-1}, x_{k-1} or if there exists some $k_0 \in \mathbb{N}$ such that $x'_{k_0} \in Q \cup NoEvolution(\mathcal{H})$ where $g(k_0, Q, t_{k-1}, x_{k-1}) = (t'_{k_0}, x'_{k_0})$, then set $HyperJump[g](k, Q, t, x)$ as the limit of the sequence $(g(k', Q, t_{k-1}, x_{k-1}))_{k' \in \mathbb{N}}$*
  - *Otherwise, set $HyperJump[g](k, Q, t, x) = g(k, Q, t_{k-1}, x_{k-1})$*

Its properties are summarized in the following lemma:

Lemma 5.4 *Assume we have a sampling g of $\mathcal{H}$ up to local dimension d' for some integer d'.*

Then:

- *$HyperJump[g]$ is a sampling of $\mathcal{H}$ up to local dimension $(d'+1)$.*
- *Assume $HyperJump[g]$ is Zeno at some parameters $Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$. Denote for all $k \in \mathbb{N}$, $HyperJump[g]\ (k, Q, t, x) = (t_k, x_k)$, $t_k \in \mathbb{R}, x_k \in \mathbb{R}^d$. For all $k \in \mathbb{N}$, x_k is of local dimension $\geq (d'+1)$.*
- *For all $t \in \mathbb{R}$, for all $x \in \mathbb{R}^d$, denote by $R_g(t, x)$ the relation associated to real sequence g at real parameter (t, x).*

  There exists a fixed first order formula F such that for all $k \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$, $\lceil HyperJump[g](k+1, Q, t, x) \rceil$ is definable by formula F from some recursive relations and from relation $R_g(HyperJump[g](k, Q, t, x))$.

Proof: See that there exists a fixed first order formula F such that, for all $t \in \mathbb{R}, x \in \mathbb{R}^d$, the language $\{Q | g$ is Zeno for $Q, t, x\}$ is first order definable by formula F from relation $R_g(t, x)$: this formula F is $\exists t_{sup} \in \mathbb{Q}\, \forall k \in \mathbb{N}\, t_k \leq t_{sup} \wedge \forall k\, x_k \notin Q \cup NoEvolution(\mathcal{H})$. In a similar way, there exists a fixed first order formula G such that for all real sequence $(g'(k', Q, x))_{k' \in \mathbb{N}}$ converging to some $g'^*(Q, x) \in \mathbb{R}^d$, $\lceil g'^*(Q, x) \rceil$ is definable by formula G from relation $R_{g'}(t, x)$. As a consequence, definition 5.7 can be translated directly into a fixed first order formula F that, for all k, Q, t, x, defines $\lceil HyperJump[g](k+1, Q, t, x) \rceil$ from some recursive relations and from relation $R_g(HyperJump[g](k, Q, t, x))$. The last assertion of the lemma is then immediate using lemma 5.4.

We prove now that $HyperJump[g]$ is a sampling of $\mathcal{H}$ up to local dimension $(d'+1)$. Assume parameters $Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$ are fixed. Denote $HyperJump[g](k, Q, t, x) = (t_k, x_k)$, $t_k \in \mathbb{R}, x_k \in \mathbb{R}^d$, for all $k \in \mathbb{N}$. Let Φ be the trajectory of $\mathcal{H}$ starting from x at time t. From the fact that g is a sampling it is easy to show by induction over k that for all $k \in \mathbb{N}\ \Phi(t_k) = x_k$. Now, if there is some k_0 with $x_{k_0} \in Q \cup NoEvolution(\mathcal{H})$, since g is a sampling, it is clear than $x_k = x_{k_0}, t_k = t_{k_0}$ for all $k \geq k_0$. If for all $k, x_k \notin Q \cup NoEvolution(\mathcal{H})$, it is easy to see that $t_{k+1} > t_k$ for all $k \in \mathbb{N}$. Hence $HyperJump[g]$ is a sampling.

Assume that $HyperJump[g]$ is Zeno at some parameters Q, t, x. For all $k \in \mathbb{N}$, g must be Zeno at parameters Q, t_{k-1}, x_{k-1}: hence, (t_k, x_k) is the limit of $(g(k', Q, t_{k-1}, x_{k-1}))_{k' \in \mathbb{N}}$. Since g is a sampling up to local dimension (d'), the local dimension of x_k must be $> (d')$ for all $k \in \mathbb{N}$. This proves the second assertion.

Denote $t_{sup} = sup_{k \in \mathbb{N}}\ t_k$ and $x_{sup} = \Phi(t_{sup})$. By corollary 5.1, the local dimension of x_{sup} must be $> (d' + 1)$. This proves the first assertion.

□

5.3.5 Trajectories that make some cycles

Lemma 5.4 shows that the hyper-jump transformation allows to increase the local dimension of a sampling by 1: when g is a sampling up to local dimension d', $HyperJump(g)$ is a sampling up to local dimension $d' + 1$. By applying recurrently the transformation hyper-jump on the effective sampling of proposition 5.3, we could get directly some upper bounds on the computational power of PCD systems.

However, one can get some better upper bounds by introducing a new transformation on samplings: the cycle-free operation: this transformation extends the hyper-jump transformation and consists in detecting whenever a trajectory is making a cycle converging toward a computable limit.

The purpose of this section and of the following is to introduce the cycle-free transformation.

We start by characterizing geometrically the case where a trajectory is making an infinite cycle converging to some point. In order to do so, we define the relation *Cycle*: we will show that when this relation is true, the trajectory makes an infinite cycle converging to some point.

Definition 5.8 (Relation *Cycle*) *Let d be an integer. Let $\mathcal{H}$ be a PCD system of dimension d. Let z_1, z_2, x^* be three points of $\mathbb{R}^d$. Let Q be a polyhedron.*

We say that $Cycle(z_1, z_2, \mathcal{H}, Q, x^*)$ is true *iff all the following conditions hold simultaneously (see figure 11):*

- *$Q \subset V_{x^*}$, Q is a open convex polyhedron and $z_1, z_2 \in Q$.*
- *$z_1 \neq z_2$, $z_1, z_2 \notin \Delta_{x^*}$ and the line (z_1, z_2) defined by z_1 and z_2 intersects Δ_{x^*} in some point z^*.*
- *$z^* \in \overline{Q}$, where $\overline{Q}$ is the topological closure of polyhedron Q.*
- *$d(p_{x^*}(z_2), p_{x^*}(x^*)) < d(p_{x^*}(z_1), p_{x^*}(x^*))$ (d is the distance of the maximum).*

A positive instance of relation *Cycle* implies that the trajectory is cycling and converging to some point: see figure 11.

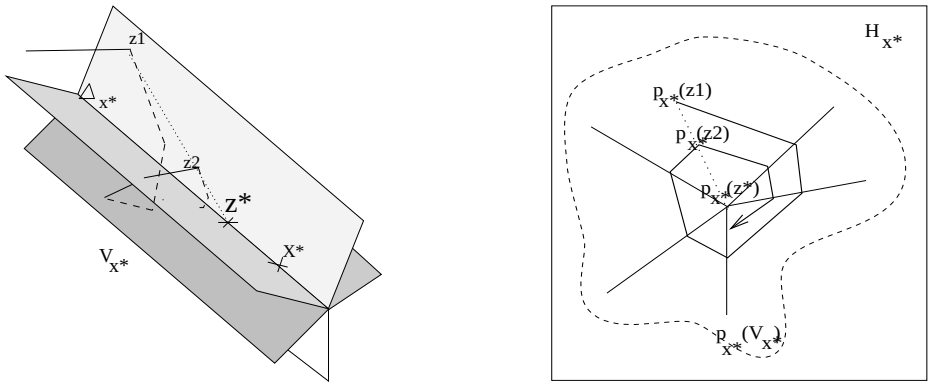

Figure 11: If predicate $Cycle(z_1, z_2, \mathcal{H}, Q, x^*)$ is true for some polyhedron Q and some point $x^* \in \mathbb{R}^d$, if the trajectory reaches z_1 and z_2 and does not leave Q between z_1 and z_2, then the trajectory is ultimately cycling and converging to z^*.

Lemma 5.5 *Let $\mathcal{H}$ be a PCD system of dimension d. Let Φ be a trajectory of $\mathcal{H}$. Let $z_1, z_2 \in \mathbb{R}^d$ be two points reached by Φ at time $t_1, t_2 \in \mathbb{R}^+$ respectively with $t_1 < t_2$. Let $x^* \in \mathbb{R}^d$. Let Q be a polyhedron.*

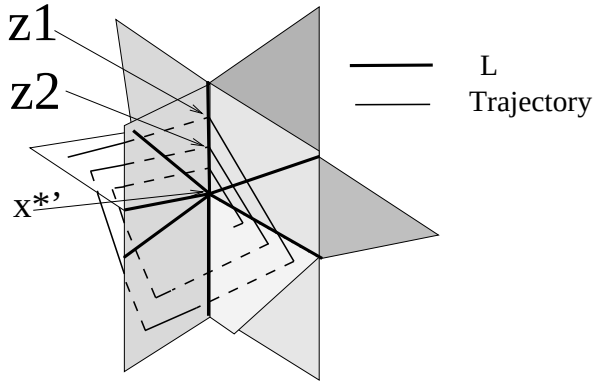

Figure 12: Proof of lemma 5.6: here $d = d' = 3$. $\mathcal{I}$ is the set of the one dimensional regions that intersect $p_{x_{sup}}(V_{x_{sup}})$. $\mathcal{I}$ is made of a finite number of segments. Every time the trajectory reaches a point of local dimension 2 , it reaches $\mathcal{I}$. If the trajectory reaches two times $\mathcal{I}$ in a same segment at point z_1 and at point z_2 then predicate $Cycle(z_1, z_2, \mathcal{H}, V_{x_{sup}}, x^{*'})$ is true for all point $x^{*'} \in \Delta_{x_{sup}}$ (here $d = d' = 3$ implies that $\Delta_{x_{sup}}$ is a singleton made of only one point).

Assume $Cycle(z_1, z_2, \mathcal{H}, Q, x^*)$ is true and that the trajectory stays in Q between time t_1 and time t_2: $\forall t \in [t_1, t_2], \Phi(t) \in Q$.

Then trajectory Φ is cycling and reaches the point z^* of definition 5.8 at time $t^* = t_1 + \sum_{j=0}^{\infty} \lambda^j (t_2 - t_1) = t_1 + (t_2 - t_1)1/(1-\lambda)$, where $\lambda \in (0,1)$ is such that $d(p_{x^*}(z_2), p_{x^*}(x^*)) = \lambda d(p_{x^*}(z_1), p_{x^*}(x^*))$.

Moreover the trajectory stays in Q between time t_1 and time t^*: for all $t \in [t_1, t^*)$, $\Phi(t) \in Q$.

We denote $Cycle_*((t_1, z_1), (t_2, z_2), \mathcal{H}, Q, x^*)$ for $(t^*, z^*) \in \mathbb{R}^{d+1}$, with the $t^* \in \mathbb{R}$ and the $z^* \in \mathbb{R}^d$ defined in the lemma.

Proof: Denote $\mathcal{H}_{x^*} = (X', f')$. By lemma 5.2, $\Phi' = p_{x^*}(\Phi)$ must be a trajectory of $\mathcal{H}_{x^*}$. Fix the origin in x^*. $Cycle(z_1, z_2, \mathcal{H}, Q, x^*)$ implies that there exists some real $0 < \lambda < 1$ with $p_{x^*}(z_2) = \lambda p_{x^*}(z_1)$: see figure 11.

By definition of V_{x^*} all the regions of $\mathcal{H}_{x^*}$ intersecting $p_{x^*}(V_{x^*})$ contain $p_{x^*}(x^*)$ in their topological closure. Hence we have $f'(x) = f'(\mu x)$, for all $x \in p_{x^*}(V_{x^*}), \mu \in (0, 1]$. If $\Phi'(t)$ is solution to differential equation $\dot{x}_d = f'(x)$, then $\Psi'(t) = \lambda \Phi'(t/\lambda)$ is also solution. As a consequence, for all $n \geq 2 \in \mathbb{N}$, trajectory Φ' must reach the point $\lambda^{n-1} p_{x^*}(z_1)$ at time $t_1 + (t_2 - t_1) \sum_{j=0}^{n-2} \lambda^j$: see figure 11.

From the definition of $\mathcal{H}_{x^*}$ this implies that, for all $n \geq 2 \in \mathbb{N}$, Φ reaches the point z_n defined by $p_{x^*}(z_n) = \lambda^{n-1} p_{x^*}(z_1)$ and $q_{x^*}(z_n) = q_{x^*}(z_1) + (q_{x^*}(z_2) - q_{x^*}(z_1)) \sum_{j=0}^{n-2} \lambda^j$ at time $t_1 + (t_2 - t_1) \sum_{j=0}^{n-2} \lambda^j$. Hence, trajectory Φ must reach z^* at time t^*: see figure 11. By convexity of Q, Φ must stay in Q between time t_1 and time t^*.

□

One can extend lemma 5.2 by showing that whenever the limit of a sequence of points of some fixed local dimension d' is of local dimension $d' + 1$, then necessarily relation $Cycle$ must be true for some points of the sequence:

Lemma 5.6 *Assume the hypotheses of lemma 5.2 hold. Take the notations of lemma 5.2.*

If x_{sup} is of local dimension $(d' + 1)$, then there must exists $i_1 < i_2 \in \mathbb{N}$, $x^{*'} \in \mathbb{Q}^d$, a rational polyhedron Q such that trajectory Φ stays in Q between time t_{i_1} and time t_{i_2} and such that predicate $Cycle(\Phi(t_{i_1}), \Phi(t_{i_2}), \mathcal{H}, Q, x^{*'})$ is true: that is to say, the hypotheses of lemma 5.5 hold with $z_1 = \Phi(t_{i_1}), z_2 = \Phi(t_{i_2}), Q, x^{*'}$.

Moreover, if $t^* \in \mathbb{R}, x^* \in \mathbb{R}^d$ denote $(t^*, x^*) = Cycle_*((t_{i_1}, \Phi(t_{i_1})), (t_{i_2}, \Phi(t_{i_2})), \mathcal{H}, Q, x_{sup})$ then the local dimension of x^* is $\geq (d' + 1)$.

Proof: We use the notations of the proof of lemma 5.2: assume $d'' = (d' + 1)$. The image $\mathcal{I}$ of $LocDim(\mathcal{H}, d')$ by $p_{x_{sup}}$ is a finite set of one-dimensional segments: see figure 12. Since $(\Phi'(t_i))_{i \geq i_0}$ is an infinite sequence, there must exists some $i_1 < i_2 \in \mathbb{N}$, $z_1 = \Phi(t_{i_1}), z_2 = \Phi(t_{i_2})$ such that $p_{x_{sup}}(z_1)$ and $p_{x_{sup}}(z_2)$ belong to a same segment of $\mathcal{I}$, and such that $d(p_{x_{sup}}(x_{sup}), p_{x_{sup}}(z_2)) <$

$d(p_{x_{sup}}(x_{sup}), p_{x_{sup}}(z_1))$: see figure 12 or figure 11. Take $Q = V_{x_{sup}}$. Check that predicate $Cycle(z_1, z_2, \mathcal{H}, Q, x^{*'})$ is true for any point $x^{*'} \in \mathbb{Q}^d \cap \Delta_{x_{sup}}$:

Denote $(t^*, x^*) = Cycle_*((t_{i_1}, z_1), (t_{i_2}, z_2)), \mathcal{H}, Q, x_{sup})$. By lemma 5.5, Φ must be converging to x^* at time t^*. By lemma 5.2, x^* must be of local dimension $\geq (d'+1)$.

□

5.3.6 CycleFree operation

From now, we assume that a PCD system $\mathcal{H}$ of dimension d is fixed.

We are ready to define the cycle-free operation on samplings: the idea is to extend the hyper-jump operation by detecting if trajectories are making some infinite cycles converging to some computable limit point: in that case, one can jump directly to the limit point:

Definition 5.9 (Cycle Free operation) *Assume we have a sampling g of $\mathcal{H}$.*

We define $CycleFree[g] : \mathbb{N} \times \mathcal{P} \times \mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ as follows: assume $Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$ are fixed.

- *Set $CycleFree[g](0, Q, t, x) = (t, x)$*
- *Let $k \geq 1$. Denote $CycleFree[g]\ (k-1, Q, t, x) = (t_{k-1}, x_{k-1})$, $t_{k-1} \in \mathbb{R}, x_{k-1} \in \mathbb{R}^d$.*
  - *Either there exists $k_2 \in \mathbb{N}$, $k_2 < k$, some $x^* \in \mathbb{Q}^d$, a rational polyhedron F not intersecting Q, such that $Cycle(x_{k-1}, z_2, \mathcal{H}, F, x^*)$ is true, $x_{k-1} \notin Q, z_2 \notin Q$, $z_2 \notin F^c$, where F^c is the complement of polyhedron F on $\mathbb{R}^d$ and $HyperJump[g](k_2, Q \cup F^c, t_{k-1}, x_{k-1}) = (t_2, z_2)$: set $CycleFree[g](k, Q, t, x) = Cycle_*((t_{k-1}, x_{k-1}), (t_2, z_2), \mathcal{H}, F, x^*)$*
  - *or this is false: Set $CycleFree[g](k, Q, t, x) = HyperJump[g](k, Q, t_{k-1}, x_{k-1})$*

The properties of the cycle-free transformation on samplings are summarized in the following lemma:

Lemma 5.7 *Assume g is a sampling of $\mathcal{H}$ up to local dimension d' for some integer $d' \in \mathbb{N}$.*

Then:

- *$CycleFree[g]$ is a sampling of $\mathcal{H}$ up to local dimension $(d'+2)$.*
- *For all $k \in \mathbb{N}$, $t \in \mathbb{R}, x \in \mathbb{R}^d$, denote by $R^{<k}_{HyperJump[g]}(t, x)$ the relation associated to real sequence $HyperJump[g]$ at real parameter (t, x) up to rank k.*

  There exists a fixed first order formula F such that for all $k \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$, $\lceil CycleFree[g](k+1, Q, t, x) \rceil$ is definable by formula F from some recursive relations and from relation $R^{<k}_{HyperJump[g]}(CycleFree[g](k, Q, t, x))$.

Proof: It is easy to see that there exists a fixed first order formula G such that, for all $z_1, z_2 \in \mathbb{R}^d$, $\{< Q, x^* > | Q \in \mathcal{P}, x^* \in \mathbb{Q}^d, Cycle(z_1, z_2, \mathcal{H}, Q, x^*)$ is true $\}$ is definable by formula G from relations $\lceil z_1 \rceil, \lceil z_2 \rceil$ and from some recursive relations. Now, see that there also exists a fixed first order formula H such that $\lceil Cycle_*((t_1, z_1), (t_2, z_2), \mathcal{H}, Q, x^*) \rceil$ is defined by formula H from relations $\lceil (t_1, z_1) \rceil$, $\lceil (t_2, z_2) \rceil$ and from some recursive relations. As a consequence, definition 5.6 can be translated directly into a fixed first order formula F such that, for all $k \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$, $\lceil CycleFree[g](k+1, Q, t, x) \rceil$ is definable by formula F from relation $R^{<k}_{HyperJump[g]}(CycleFree[g](k, Q, t, x))$ and from some recursive relations. The second assertion is immediate by using lemma 5.4.

We prove now that $CycleFree[g]$ is a sampling of $\mathcal{H}$ up to local dimension $(d'+2)$. Assume $Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$ are fixed. Denote $CycleFree[g](k, Q, t, x) = (t_k, x_k)$, $t_k \in \mathbb{R}, x_k \in \mathbb{R}^d$, for all $k \in \mathbb{N}$. Let Φ be the trajectory of $\mathcal{H}$ starting from x at time t.

Using lemma 5.4 and lemma 5.5, it is easy to show by induction over k that for all k, $\Phi(t_k) = x_k$. If there is some k_0 with $x_{k_0} \in Q \cup NoEvolution(\mathcal{H})$, since $HyperJump[g]$ is a sampling, it is clear than $x_k = x_{k_0}, t_k = t_{k_0}$ for all $k \geq k_0$. If for all $k \in \mathbb{N}$, $x_k \notin Q \cup NoEvolution(\mathcal{H})$, it is easy to see that $t_{k+1} > t_k$ for all $k \in \mathbb{N}$. Hence $CycleFree[g]$ is a sampling.

Assume that $CycleFree[g]$ is Zeno at parameters Q, t, x. Denote $t_{sup} = sup_{k \in \mathbb{N}} t_k$ and $x_{sup} = \Phi(t_{sup})$. If x_{sup} is of local dimension $> (d'+2)$ the lemma is proved. Assume now that the the local dimension of x_{sup} is $\leq (d'+2)$.

For all $k \in \mathbb{N}$, $HyperJump[g]$ must be Zeno at parameters Q, t_{k-1}, x_{k-1}. As a consequence, by lemma 5.4 and by lemma 5.6, all the $x_k, k \in \mathbb{N}$ must be of local dimension $\geq (d'+1)$. By corollary 5.1, only a finite number of the $x_k, k \in \mathbb{N}$ must be of local dimension $\geq (d'+2)$, and only a a finite number of the

$x_k, k \in \mathbb{N}$ must be of local dimension $(d'+1)$. Hence, there must exists some $k_0 \in \mathbb{N}$ such that for all $k \geq k_0$ x_k is of local dimension $(d'+1)$.

Apply lemma 5.6 on the subsequence $(x_k)_{k \geq k_0}$: There musts exists $k_0 \leq i_1 < i_2 \in \mathbb{N}, x_{sup} \in \mathbb{Q}^d$ an a rational polyhedron F such that $Cycle(x_{i_1}, x_{i_2}, \mathcal{H}, Q, x_{sup})$ is true and such that the trajectory does not leave F between time t_{i_1}, t_{i_2}. Take i_1 and i_2 as the least integers such that the previous property hold and such that $i_2 - i_1 < i_1$. By definition 5.6 we have $(t_{i_2+1}, x_{i_2+1}) = Cycle_*((t_{i_1}, x_{i_1}), (t_{i_2}, x_{i_2}), \mathcal{H}, F, x_{sup})$. This is impossible since by lemma 5.6 this would imply that the local dimension of x_{i_2+1} is $\geq (d'+2)$. □

Hence, by using the cycle-free operation instead of the hyper-jump operation we get more powerful samplings : when g is a sampling up to local dimension d', $CycleFree(g)$ is a sampling up to local dimension $d'+2$ (in comparison $HyperJump(g)$ is a sampling only up to local dimension $d'+1$).

5.3.7 Outputting a maximal sampling

We are ready to prove that one can build maximal samplings of $\mathcal{H}$ that are computable by some Turing machines with some hyper-arithmetical oracles: the idea is to apply recurrently the cycle-free operation on the effective sampling given by proposition 5.3 in order to get samplings up to higher and higher local dimensions:

Lemma 5.8 *For all $k \geq 0$,*

- *one can construct a sampling $g_k : \mathbb{N} \times \mathcal{P} \times \mathbb{R}^{d+1} \to \mathbb{R}^{d+1}$ up to local dimension $(3+2k)$.*
- *The sampling is computable by some Turing machine with an hyper–arithmetical oracle: for all $t \in \mathbb{R}, x \in \mathbb{R}^d$, denote by $R_{g_k}(t,x)$ the relation associated to g_k at real parameter (t,x).*
  There exists some $n_k \in \mathbb{N}$, $z_k \in O, |z_k| = \omega^k$ if $k \geq 1$, $|z_k| = 0$ if $k = 0$, such that, for all $t \in \mathbb{R}, x \in \mathbb{R}^d$, $W_{n_k}^{H^{\lceil (t,x) \rceil}(z_k)} = R_{g_k}(t,x)$

Proof: We prove the assertions by induction over $k \in N$.

The case $k = 0$ is corollary 5.3.

Assume $k \geq 1$. Consider $g_k = CycleFree[g_{k-1}]$. By lemma 5.7 and by induction hypothesis g_k is a sampling up to local dimension $(3+2k)$. By induction hypothesis, n'_{k-1} is a $H^{\lceil (t,x) \rceil}(z_{k-1})$-recursively enumerable index of $R_{g_{k-1}}(t,x)$ for all t, x.

Let $n \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$ be fixed. Assume we have $H(y)^{\lceil (t,x) \rceil}$-recursively enumerable index m of $HyperJump[g_{k-1}](n, Q, t, x)$, where $m \in \mathbb{N}, y \in O$. By lemma 5.3, there exists a recursive r that maps m to $r(m)$ where $r(m)$ is an an $H(y +_0 z_{k-1} +_0 1)^{\lceil (t,x) \rceil}$-recursive index of $R_{g_{k-1}}(HyperJump[g_{k-1}](n, Q, t, x))$. By lemma 5.4, there exists a fixed first order formula F such that for all $n \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$, $\lceil HyperJump[g_{k-1}](n+1, Q, t, x) \rceil$ is definable by formula F from relation $R_{g_{k-1}}(HyperJump[g_{k-1}]\ (n, Q, t, x))$ and from some recursive relations. By lemma 5.4, there exists $y_F \in O, |y_F| < \omega$ and a recursive g that maps $r(m)$ to $g(r(m))$, where $g(r(m))$ is an $H^{R_{g_{k-1}}(HyperJump[g_{k-1}](n,Q,t,x))}(y_F)$-recursively enumerable index of $\lceil HyperJump[g_{k-1}](n+1, Q, t, x) \rceil$. By lemma 5.3, there exists a recursive r' that maps $g(r(m))$ to $r'(g(r(m)))$, where $r'(g(r(m)))$ is an $H(y +_0 z_{k-1} +_0 1 +_o y_F)^{\lceil (t,x) \rceil}$-recursively enumerable index of $HyperJump[g_{k-1}](n+1, Q, t, x)$.

Denote by $h : \mathbb{N} \to O$ the recursive mapping such that $r(0) = 1, r(n+1) = r(n) +_0 z_{k-1} +_0 1 +_o y_F$ for all $n \in \mathbb{N}$.

As a consequence, for all $n \in \mathbb{N}$, $R^{<n}_{HyperJump[g_{k-1}]}$ is semi-recognized by the machine with oracle $H^{\lceil (t,x) \rceil}(h(n-1))$ that on input $< n, Q, P >$, compute for $i = 1, \ldots, n-1$ an $H^{\lceil (t,x) \rceil}(h(i-1))$-recursively enumerable index m_i of $HyperJump[g_{k-1}]\ (i, Q, t, x)$ from the $H^{\lceil (t,x) \rceil}(h(i-2))$-recursively enumerable index m_{i-1} of $HyperJump[g_{k-1}](i-1, Q, t, x)$ by the formula $m_i = r'(g(r(m_{i-1})))$ and then simulate the machine with oracle of number m_{n-1}. This machine has a fixed number independent of t, x.

Let $n \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$ be fixed. Assume we have $H(y)^{\lceil (t,x) \rceil}$-recursively enumerable index m of $CycleFree[g_{k-1}](n, Q, t, x)$, where $m \in \mathbb{N}, y \in O$. By lemma 5.3, there exists a recursive r that maps m to $r(m)$ where $r(m)$ is an $H(y +_0 h(n-1) +_0 1)^{\lceil (t,x) \rceil}$-recursive index of $R^{<n}_{HyperJump[g_{k-1}]}$ $(CycleFree[g_{k-1}](n, Q, t, x))$. By lemma 5.7, there exists a fixed first order formula G such that for all $k \in \mathbb{N}, Q \in \mathcal{P}, t \in \mathbb{R}, x \in \mathbb{R}^d$, $\lceil CycleFree[g_{k-1}](n+1, Q, t, x) \rceil$ is definable by G from relation $R^{<n}_{HyperJump[g_{k-1}]}$ $(CycleFree[g_{k-1}](n, Q, t, x))$ and from some recursive relations. As before, by lemma 5.4, and by lemma 5.3, there exists some recursive g and r' that maps m to $r'(g(r(m)))$ an $H(y +_0$

$h(n-1)+_0 1+_o y_G)^{\lceil (t,x)\rceil}$-recursively enumerable index of $CycleFree[g_{k-1}](n+1,Q,t,x)$, for some fixed $y_G \in O, |y_G| < \omega$.

Denote by $l : \mathbb{N} \to O$ the recursive mapping such that $l(0) = 1, l(n+1) = r(n) +_0 h(n-1) +_0 1 +_o y_G$ for all $n \in \mathbb{N}$. Take $z_k = 3.5^p$ where $p \in \mathbb{N}$ is the number of recursive function l.

$R_{CycleFree[g_{k-1}]}$ is semi-recognized by the machine with oracle $H^{\lceil (t,x)\rceil}(z_k)$ that on input $< n, Q, P >$, compute for $i = 0, 1, \ldots, n$ an $H^{\lceil (t,x)\rceil}(l(i))$-recursively enumerable index m_i of $CycleFree[g_{k-1}](i, Q, t, x)$ from the $H^{\lceil (t,x)\rceil}(l(i-1))$-recursively enumerable index m_{i-1} of $CycleFree[g_{k-1}](i-1, Q, t, x)$ ($m_i = r'(g(r(m_{i-1})))$) and then transform $H^{\lceil (t,x)\rceil}(l(n))$-recursively enumerable index m_n of $CycleFree[g_{k-1}](n, Q, t, x)$ into a $H^{\lceil (t,x)\rceil}(z_k)$ index m of $CycleFree[g_{k-1}](n, Q, t, x)$ using lemma 5.9, and then simulate the machine with oracle of number m. This machine has a fixed number n_k. independent of t, x.

One has $|z_k| = \omega^k$.

□

where lemma 5.9 is the following technical lemma proved in [6, 16]:

Lemma 5.9 *There exists a recursive function g such that, for all $y, z \in O$, for all $m \in \mathbb{N}$, if m is an $H(y)$-recursively enumerable index of some set $S \subset \Sigma^*$ and if $y \leq_o z$, then $g(y, m, z)$ is an $H(z)$-recursively enumerable index of S.*

5.3.8 Conclusion

Using the maximal samplings given by the previous subsection, we can prove the main result of this paper: theorem 3.4 is optimal.

Proposition 5.5 *Let $k \geq 1$.*

- *If a language L is semi-recognized by a PCD system of dimension $2k+3$ in finite continuous time then $L \in \Sigma_{\omega^k}$.*
- *If a language L is semi-recognized by a PCD system of dimension $2k+4$ in finite continuous time then $L \in \Sigma_{\omega^k+1}$.*

Proof: It is clear that for all $x \in \mathbb{Q}^d$, $\lceil x \rceil$ is recursive with a recursive index computable from x.

Let $k \geq 1$. By lemma 5.8, one can build a sampling g_k up to local dimension $(3+2k)$ and there exists some fixed n_k, and some fixed $z_k, z_k \in O, |z_k| = \omega^k$ such that, for all $t \in \mathbb{R}, x \in \mathbb{R}^d$, $W_{n_k}^{H^{\lceil (t,x)\rceil}(z_k)} = R_{g_k}(t, x)$.

Let $\mathcal{H} = (\mathbb{R}^d, f, \mathcal{J}, x^1, x^0)$ be a PCD system of dimension d recognizing language L.

Assume $d = 2k+3$: all the points of $\mathcal{H}$ have a local dimension $\leq (2k+3)$. As a consequence g_k can not be Zeno at any parameters Q, t, x. L is semi-recognized by the machine with oracle $H(z_k)$ that on input $n \in \Sigma^*$, compute the $H(z_k)$-recursively enumerable index m of $S = R_{g_k}(0, \mathcal{J}(n), 0, \ldots, 0)$, and then by simulating $M_m^{H(z_k)}$, tests for $i = 0, 1, \ldots, \infty$ if $< i, x^1, x^1 > \in S$. It there is such an i, the machine accepts. If no i is found, the machine continues for ever.

Assume $d = 2k+4$. It is easy to see that any point of local dimension equal to the dimension of the space is necessarily a rational point. Denote $Reach = \{< x, y, i > | x, y \in \mathbb{Q}^d \; i \in \mathbb{N} \; < i, y, y > \in R_{g_k}(0, x)\}$. $Reach$ is $H(z_k)$-recursively enumerable by the machine that on input $< x, y, i >$ computes the $H(z_k)$-recursively enumerable index m of $R_{g_k}(0, x)$, and then simulates $M_m^{H(z_k)}$ on input $< i, y, y >$. As a consequence, there exists a first order formula H with a quantifier $\exists$ and 0 alternation such that $Reach$ is definable from formula H from some $H(z_k)$-recursive relations: see [16].

Define the following relation $OneStep = \{< x, x^1 > | x \in \mathbb{Q}^d, x^1$ is a point of local dimension d and the trajectory starting from x reaches $x^1\} \subset \Sigma^*$. We claim that $OneStep$ is definable by some first order formula F from relation $Reach$: write F as the formula that says that x^1 is in $LocDim(\mathcal{H}, d'+1)$ and that either there exists some $i \in \mathbb{N}$ such that $< x, x^1, i > \in Reach$, or g_k is Zeno at parameters $x^1, 0, x$ and there exist some i_0 and some open polyhedron x' with $< x, x', i > \in Reach$, $x' \subset V_{x^1}$ and for all $i \geq i_0$, not $< x, V_{x^1}^c, i > \in Reach$, where $V_{x^1}^c$ is the complement of polyhedron V_{x^1} in $\mathbb{R}^d$.

If $< x, x^1 > \in OneStep$, it is clear that the formula must be true. Assume now that the formula is true: if there exists some $i \in \mathbb{N}$ such that $< x, x^1, i > \in Reach$, we are done: $< x, x^1 > \in OneStep$. Assume now that the second clause of the disjunction is true: we know that the trajectory starting from x is Zeno. Hence $(g_k(i, x^1, 0, x) = (t_i, x_i))_{i \in \mathbb{N}}$ is a converging sequence converging to some point x^* at time $t^* = sup_{i \in \mathbb{N}} t_i$. Since g_k is a sampling up to local dimension $(3+2k)$, x^* must be of local dimension

d. Since Φ is Lipschitz, since V_{x^1} is an open polyhedron, we know that for some big enough i_1, for all $t_{i_1} \leq t < t^*$, $\Phi(t) \in V_{x^1}$. Hence, we must have $x^* \in \overline{V_{x^1}}$, where $\overline{V_{x^1}}$ is the topological closure of V_{x^1}. But, x^1 is the only point of local dimension d in V_{x^1}. Hence $x^* = x^1$.

See that formula F starts by a quantifier $\exists$ and has 1 alternation.

Now, see that L is definable by some first order formula G from relation *Reach*, from relation *OneStep* and from some recursive relations: write that $n \in L$ iff there exists $m \in \mathbb{N}$, and an integer encoding m rational points $x_1, x_2, \ldots, x_m$, such that for all $1 \leq i < m < x_i, x_{i+1} >\in$ *OneStep*, and $x_0 = (\mathcal{J}(\backslash), 0, \ldots, 0)$, and there exists some $i \in \mathbb{N}$, with $< x_m, x^1, i >\in$ *Reach*.

Substitute every occurrence of relation *OneStep* in formula G by formula F and every occurrence of formula Reach by formula H. One gets a resulting formula defining L from some $H(z_k)$ recursive relations starting with a quantifier $\exists$ and with 1 alternation. By lemma 5.4, $L \in \Sigma_2^{H(z_k)} = \Sigma_{\omega^k+1}$.

□

We get immediately from theorem 4.2 and from proposition 5.5 for $k \geq 1$ and [6, 7, 8] for $k = 0$:

Theorem 5.1 *Let $k' \geq 0$.*

- *The languages that are semi-recognized by a PCD system of dimension $2k' + 3$ in finite continuous time are precisely the languages of $\Sigma_{\omega^{k'}}$.*
- *The languages that are semi-recognized by a PCD system of dimension $2k' + 4$ in finite continuous time are precisely the languages of $\Sigma_{\omega^{k'}+1}$*

In other words, we obtain a full characterization of the computational power of PCD systems.

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] 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.

[3] Eugene Asarin and Oded Maler. Achilles and the Tortoise Climbing Up the Arithmetical Hierarchy. In *Proceedings of FSTTCS*, pages 471–483, 1995. Lecture Notes in Computer Science, 1026.

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

[5] 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.

[6] Olivier Bournez. Achilles and the tortoise climbing up the hyperarithmetical hierachy. Technical report, LIP ENS-Lyon, 1997.

[7] Olivier Bournez. Some bounds on the computational power of piecewise constant derivative systems. In *Proceeding of ICALP'97*, pages 143–153, 1997. Lecture Notes in Computer Science, 1256.

[8] Olivier Bournez. Some bounds on the computational power of purely rational piecewise constant derivative systems. Technical report, LIP ENS-Lyon, 1997.

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

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

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

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

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

[14] Cristopher Moore. Recursion theory on the reals and continuous-time computation. *Theoretical Computer Science*, 162:23–44, 1996.

[15] P. Odifreddi. *Classical Recursion Theory*, volume 125 of *Studies in Logic and the foundations of mathematics*. Elsevier, 1992.

[16] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.