Turing machines can be efficiently simulated by the General Purpose Analog Computer

Olivier Bournez[1], Daniel S. Graça[2,3], and Amaury Pouly[1,2]

[1] Ecole Polytechnique, LIX, 91128 Palaiseau Cedex, France
Olivier.Bournez@lix.polytechnique.fr
[2] CEDMES/FCT, Universidade do Algarve, C. Gambelas, 8005-139 Faro, Portugal
dgraca@ualg.pt
[3] SQIG /Instituto de Telecomunicações, Lisbon, Portugal

Abstract. The Church-Turing thesis states that any sufficiently powerful computational model which captures the notion of algorithm is computationally equivalent to the Turing machine. This equivalence usually holds both at a computability level and at a computational complexity level modulo polynomial reductions. However, the situation is less clear in what concerns models of computation using real numbers, and no analog of the Church-Turing thesis exists for this case. Recently it was shown that some models of computation with real numbers were equivalent from a computability perspective. In particular it was shown that Shannon's General Purpose Analog Computer (GPAC) is equivalent to Computable Analysis. However, little is known about what happens at a computational complexity level. In this paper we shed some light on the connections between this two models, from a computational complexity level, by showing that, modulo polynomial reductions, computations of Turing machines can be simulated by GPACs, without the need of using more (space) resources than those used in the original Turing computation, as long as we are talking about bounded computations. In other words, computations done by the GPAC are as space-efficient as computations done in the context of Computable Analysis.

1 Introduction

The Church-Turing thesis is a cornerstone statement in theoretical computer science, stating that any (discrete time, digital) sufficiently powerful computational model which captures the notion of algorithm is computationally equivalent to the Turing machine (see e.g. [19], [23]). It also relates various aspects of models in a very surprising and strong way.

The Church-Turing thesis, although not formally a theorem, follows from many equivalence results for discrete models and is considered to be valid by the scientific community [19]. When considering non-discrete time or non-digital models, the situation is far from being so clear. In particular, when considering models working over real numbers, several models are clearly not equivalent [9].

However, a question of interest is whether physically *realistic* models of computation over the real numbers are equivalent, or can be related. Some of the results of non-equivalence involve models, like the BSS model [5], [4], which are claimed not to be physically realistic [9] (although they certainly are interesting from an algebraic perspective), or models that depend critically of computations which use exact numbers to obtain super-Turing power, e.g. [1], [3].

Realistic models of computation over the reals clearly include the *General Purpose Analog Computer (GPAC)* [21], an analog continuous-time model of computation and *Computable Analysis* (see e.g. [24]). The GPAC is a mathematical model introduced by Claude Shannon of an earlier analog computer, the Differential Analyzer. The first general-purpose Differential Analyzer is generally attributed to Vannevar Bush [10]. Differential Analyzers have been used intensively up to the 1950's as computational machines to solve various problems from ballistic to aircraft design, before the era of the digital computer [18].

Computable analysis, based on Turing machines, can be considered as today's most used model for talking about computability and complexity over reals. In this approach, real numbers are encoded as sequences of discrete quantities and a discrete model is used to compute over these sequences. More details can be found in the books [20], [17], [24]. As this model is based on classical (digital and discrete time) models like Turing machines, which are considered to be realistic models of today's computers, one can consider that Computable Analysis is a realistic model (or, more correctly, a theory) of computation.

Understanding whether there could exist something similar to a Church-Turing thesis models of computation involving real numbers, or whether analog models of computation could be more powerful than today's classical models of computation motivated us to try to relate GPAC computable functions to functions computable in the sense of computable analysis.

The paper [6] was a first step towards the objective of obtaining a version of the Church-Turing thesis for physically feasible models over the real numbers. This paper proves that, from a computability perspective, Computable Analysis and the GPAC are equivalent: GPAC computable functions are computable and, conversely, functions computable by Turing machines or in the computable analysis sense can be computed by GPACs. However this is about *computability*, and not *computational complexity*. This proves that one cannot solve more problems using the GPAC than those we can solve using discrete-based approaches such as Computable Analysis. But this leaves open the question whether one could solve some problems *faster* using analog models of computations (see e.g. what happens for quantum models of computations...). In other words, the question of whether the above models are equivalent at a computational complexity level remained open. Part of the difficulty stems from finding an appropriate notion of complexity (see e.g. [22], [2]) for analog models of computations.

In the present paper we study both the GPAC and Computable Analysis at a complexity level. In particular, we introduce measures for space complexity and show that, using these measures, both models are equivalent, even at a computational complexity level, as long as we consider time-bounded simulations. Since we already have shown in our previous paper [7] that Turing machines can simulate efficiently GPACs, this paper is a big step towards showing the converse direction: GPACs can simulate Turing machines in an efficient manner.

More concretely we show that computations of Turing machines can be simulated in polynomial space by GPACs as long as we use bounded (but arbitrary) time. We firmly believe that this construction can be used as a building brick to show the more general result that the computations of Turing machines can be simulated in polynomial space by GPACs, removing the hypothesis of arbitrary but fixed time. This latter construction would probably be much more involved, and we intend to focus on it in the near future since this result would show that computations done by the GPAC and in the context of Computable Analysis are equivalent modulo polynomial space reductions.

We believe that these results open the way for some sort of more general Church-Turing thesis, which applies not only to discrete-based models of computation but also to physically realistic models of computation, and which holds both at a computability and computational complexity (modulo polynomial reductions) level.

Incidently, these kind of results can also be the first step towards a well-founded complexity theory for analog models of computations and for continuous dynamical systems.

Notice that it has been observed in several papers that, since continuous time systems might undergo arbitrary space and time contractions, Turing machines, as well as even accelerating Turing machines[4] [14], [13], [12] or even oracle Turing machines, can actually be simulated in an arbitrary short time by ordinary differential equations in an arbitrary short time or space. This is sometimes also called *Zeno's phenomenon*: an infinite number of discrete transitions may happen in a finite time: see e.g. [8]. Such constructions or facts have been deep obstacles to various attempts to build a well founded complexity theory for analog models of computations: see [8] for discussions. One way to interpret our results is then the following: all these time and space phenomena, or Zeno's phenomena do not hold (or, at least, they do not hold in a problematic manner) for ordinary differential equations corresponding to GPACs, that is to say for *realistic* models, for carefully chosen measures of complexity. Moreover, these measures of complexity relate naturally to standard computational complexity measures involving discrete models of computation

2 Preliminaries

2.1 Notation

Throughout the paper we will use the following notation:

$$\|(x_1,\ldots,x_n)\| = \max_{1\leqslant i\leqslant n} |x_i| \qquad \|(x_1,\ldots,x_n)\|_2 = \sqrt{|x_1|^2+\cdots+|x_n|^2}$$

[4] Similar possibilities of simulating accelerating Turing machines through quantum mechanics are discussed in [11].

$$\pi_i(x_1,\dots,x_k)=x_i \qquad \mathrm{int}(x)=\lfloor x\rfloor \qquad \mathrm{frac}(x)=x-\lfloor x\rfloor$$

$$\mathrm{int}_n(x)=\min(n,\mathrm{int}(x)) \qquad \mathrm{frac}_n(x)=x-\mathrm{int}_n(x)$$

$$f^{[n]}=\begin{cases}\mathrm{id} & \text{if } n=0\\ f^{[n-1]} & \text{otherwise}\end{cases}$$

$$\mathrm{sgn}(x)=\begin{cases}-1 & \text{if } x<0\\ 0 & \text{if } x=0\\ 1 & \text{if } x>0\end{cases} \qquad \mathbb{R}^*=\mathbb{R}\setminus\{0\}$$

2.2 Computational complexity measures for the GPAC

It is known [16] that a function is generable by a GPAC iff it is a component of the solution a polynomial initial-value problem. In other words, a function $f: I \to \mathbb{R}$ is GPAC-generable iff it belongs to following class.

Definition 1. *Let $I \subseteq \mathbb{R}$ be an open interval and $f : I \to \mathbb{R}$. We say that $f \in \mathrm{GPAC}(I)$ if there exists $d \in \mathbb{N}$, a vector of polynomials p, $t_0 \in I$ and $y_0 \in \mathbb{R}^d$ such that for all $t \in I$ one has $f(t) = y_1(t)$, where $y : I \to \mathbb{R}$ is the unique solution over I of*

$$\begin{cases}\dot{y} &= p(y)\\ y(t_0) &= y_0\end{cases} \tag{1}$$

Next we introduce a subclass of GPAC generable functions which allow us to talk about space complexity. The idea is that a function f generated by a GPAC belongs to the class $\mathrm{GSPACE}\,(I,g)$ if f can be generated by a GPAC in I and does not grow faster that g. Since the value of f in physical implementations of the GPAC correspond to some physical quantity (e.g. electric tension), limiting the growth of f corresponds to effectively limiting the size of resources (i.e. magnitude of signals) needed to compute f by a GPAC.

Definition 2. *Let $I \subseteq \mathbb{R}$ be an open interval and $f, g : I \to \mathbb{R}$ be functions. The function f belongs to the class $\mathrm{GSPACE}\,(I,g)$ if there exist $d \in \mathbb{N}$, a vector of polynomials p, $t_0 \in I$ and $y_0 \in \mathbb{R}^d$ such that for all $t \in I$ one has $f(t) = y_1(t)$ and $\|y(t)\| \leqslant g(t)$, where $y : I \to \mathbb{R}$ is the unique solution over I of (1). More generally, a function $f : I \to \mathbb{R}^d$ belongs to $f \in \mathrm{GSPACE}\,(I,g)$ if all its components are also in the same class.*

We can generalize the complexity class GSPACE to multidimensional open sets I defined over $\mathbb{R}^d$. The idea is to reduce it to the one-dimensional case defined above through the introduction of a subset $J \subseteq \mathbb{R}$ and of a map $g : J \to I$.

Definition 3. *Let $I \subseteq \mathbb{R}^d$ be an open set and $f, s_f : I \to \mathbb{R}$ be functions. Then $f \in \mathrm{GSPACE}\,(I, s_f)$ if for any open interval $J \subseteq \mathbb{R}$ and any function $(g : J \to \mathbb{R}^d \in \mathrm{GSPACE}\,(J, s_g)$ such that $g(J) \subseteq I$, one has $f \circ g \in$ $\mathrm{GSPACE}\,(J, \max(s_g, s_f \circ s_g))$.*

The following closure results can be proved (proofs are omitted for reasons of space).

Lemma 1. *Let $I, J \subseteq \mathbb{R}^d$ be open sets, and $(f : I \to \mathbb{R}^n)$ and $(g : J \to \mathbb{R}^m)$ be functions which belong to* $\mathrm{GSPACE}\,(I, s_f)$ *and* $\mathrm{GSPACE}\,(J, s_g)$*, respectively. Then:*

- $f + g, f - g \in \mathrm{GSPACE}\,(I \cap J, s_f + s_g)$ *if* $n = m$.
- $fg \in \mathrm{GSPACE}\,(I \cap J, \max(s_f, s_g, s_f s_g))$ *if* $n = m$.
- $f \circ g \in \mathrm{GSPACE}\,(J, \max(s_g, s_f \circ s_g))$ *if* $m = d$ *and* $g(J) \subseteq I$.

2.3 Main result

Our main result states that any Turing machine can be simulated by a GPAC using a space bounded by a polynomial, where T and S are respectively the time and the space used by the Turing machine.

If one prefers, (formal statement in Theorem 3):

Theorem 1. *Let $\mathcal{M}$ be a Turing Machine. Then there is a GPAC-generable function f_M and a polynomial p with the following properties:*

1. *Let S, T be arbitrary positive integers. Then $f_{\mathcal{M}}(S, T, [e], n)$ gives the configuration of $\mathcal{M}$ on input e at step n, as long as $n \leq T$ and $\mathcal{M}$ uses space bounded by S.*
2. *$f_{\mathcal{M}}(S, T, [e], t)$ is bounded by $p(T + S)$ as long as $0 \leq t \leq n$.*

The first condition of the theorem states that the GPAC simulates TMs on bounded space and time, while the second condition states that amount of resources used by the GPAC computation is polynomial on the amount of resources used by original Turing computation.

3 The construction

3.1 Helper functions

Our simulation will be performed on a real domain and may be subject to (small) errors. Thus, to simulate a Turing machine over a large number of steps, we need tools which allow us to keep errors under control. In this section we present functions which are specially designed to fulfill this objective. We call these functions *helper functions*. Notice that since functions generated by GPACs are analytic, all helper functions are required to be analytic. As a building block for creating more complex functions, it will be useful to obtain analytic approximations of the functions $\mathrm{int}(x)$ and $\mathrm{frac}(x)$. Notice that we are only concerned about non-negative numbers so there is no need to discuss the definition of these functions on negative numbers.

Definition 4. *For any $x, y, \lambda \in \mathbb{R}$ define $\xi(x, y, \lambda) = \tanh(xy\lambda)$.*

Lemma 2. *For any $x \in \mathbb{R}$ and $\lambda > 0, y \geqslant 1$,*

$$|\operatorname{sgn}(x) - \xi(x, y, \lambda)| < 1$$

Furthermore if $|x| \geqslant \lambda^{-1}$ then

$$|\operatorname{sgn}(x) - \xi(x, y, \lambda)| < e^{-y}$$

and $\xi \in \mathrm{GSPACE}\left(\mathbb{R}^3, 1\right)$.

Definition 5. *For any $x, y, \lambda \in \mathbb{R}$, define*

$$\sigma_1(x, y, \lambda) = \frac{1 + \xi(x-1, y, \lambda)}{2}$$

Corollary 1. *For any $x \in \mathbb{R}$ and $y > 0, \lambda > 2$,*

$$|\operatorname{int}_1(x) - \sigma_1(x, y, \lambda)| \leqslant 1/2$$

Furthermore if $|1 - x| \geqslant \lambda^{-1}$ then

$$|\operatorname{int}_1(x) - \sigma_1(x, y, \lambda)| < e^{-y}$$

and $\sigma_1 \in \mathrm{GSPACE}\left(\mathbb{R}^3, 1\right)$.

Definition 6. *For any $p \in \mathbb{N}$, $x, y, \lambda \in \mathbb{R}$, define*

$$\sigma_p(x, y, \lambda) = \sum_{i=0}^{k-1} \sigma_1(x - i, y + \ln p, \lambda)$$

Lemma 3. *For any $p \in \mathbb{N}$, $x \in \mathbb{R}$ and $y > 0, \lambda > 2$,*

$$|\operatorname{int}_p(x) - \sigma_p(x, y, \lambda)| \leqslant 1/2 + e^{-y}$$

Furthermore if $x < 1 - \lambda^{-1}$ or $x > p + \lambda^{-1}$ or $d(x, \mathbb{N}) > \lambda^{-1}$ then

$$|\operatorname{int}_p(x) - \sigma_p(x, y, \lambda)| < e^{-y}$$

and $\sigma_p \in \mathrm{GSPACE}\left(\mathbb{R}^3, p\right)$.

Finally, we build a square wave like function which we be useful later on.

Definition 7. *For any $t \in \mathbb{R}$, and $\lambda > 0$, define $\theta(t, \lambda) = e^{-\lambda(1-\sin(2\pi t))^2}$*

Lemma 4. *For any $\lambda > 0$, $\theta(\cdot, \lambda)$ is a positive and 1-periodic function bounded by 1, furthermore*

$$\forall t \in [1/2, 1], |\theta(t, \lambda)| \leqslant \frac{e^{-\lambda}}{2}$$

$$\int_0^{\frac{1}{2}} \theta(t, \lambda) dt \geqslant \frac{(e\lambda)^{-\frac{1}{4}}}{\pi}$$

and $\theta \in \mathrm{GSPACE}\left(\mathbb{R} \times \mathbb{R}_+^*, (t, \lambda) \mapsto \max(1, \lambda)\right)$.

3.2 Polynomial interpolation

In order to implement the transition function of the Turing Machine, we will use polynomial interpolation techniques (Lagrange interpolation). But since our simulation may have to deal with some amount of error in inputs, we have to investigate how this error propagates through the interpolating polynomial.

Definition 8 (Lagrange polynomial). *Let $d \in \mathbb{N}$ and $f : G \to \mathbb{R}$ where G is a finite subset of $\mathbb{R}^d$, we define*

$$L_f(x) = \sum_{\bar{x} \in G} f(\bar{x}) \prod_{i=1}^{d} \prod_{\substack{y \in G \\ y \neq \bar{x}}} \frac{x_i - y_i}{\bar{x}_i - y_i}$$

Lemma 5. *Let $n \in \mathbb{N}$, $x, y \in \mathbb{R}^n$, $K > 0$ be such that $\|x\|, \|y\| \leqslant K$, then*

$$\left| \prod_{i=1}^{n} x_i - \prod_{i=1}^{n} y_i \right| \leqslant K^{n-1} \sum_{i=1}^{n} |x_i - y_i|$$

3.3 Turing Machines — assumptions

Let $\mathcal{M} = (Q, \Sigma, b, \delta, q_0, F)$ be a Turing Machine which will be fixed for the whole simulation. Without loss of generality we assume that:

- When the machine reaches a final state, it stays in this state
- $Q = \{0, \ldots, m-1\}$ are the states of the machines; $q_0 \in Q$ is the initial state; $F \subseteq Q$ are the accepting states
- $\Sigma = \{0, \ldots, k-2\}$ is the alphabet and $b = 0$ is the blank symbol.
- $\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$ is the transition function, and we identify $\{L, R\}$ with $\{0, 1\}$ ($L = 0$ and $R = 1$). The components of δ are denoted by $\delta_1, \delta_2, \delta_3$. That is $\delta(q, \sigma) = (\delta_1(q, \sigma), \delta_2(q, \sigma), \delta_3(q, \sigma))$ where δ_1 is the new state, δ_2 the new symbol and δ_3 the head move direction.

Notice that the alphabet of the Turing machine has $k-1$ symbols. This will be important for lemma 6. Consider a configuration $c = (x, \sigma, y, q)$ of the machine as described in figure **??**. We can encode it as a triple of integers as done in [15] (e.g. if $x_0, x_1, \ldots$ are the digits of x in base k, encode x as the number $x_0 + x_1 k + x_2 k^2 + \cdots + x_n k^n$), but this encoding is not suitable for our needs. We define the *rational encoding* $[c]$ of c as follows.

Definition 9. *Let $c = (x, s, y, q)$ be a configuration of $\mathcal{M}$, we define the* rational encoding *$[c]$ of c as $[c] = (0.x, s, 0.y, q)$ where:*

$$0.x = x_0 k^{-1} + x_1 k^{-2} + \cdots + x_n k^{-n-1} \in \mathbb{Q} \qquad \text{if} \qquad x = x_0 + x_1 k + \cdots + x_n k^n \in \mathbb{N}$$

The following lemma explains the consequences on the rational encoding of configurations of the assumptions we made for $\mathcal{M}$.

Lemma 6. *Let c be a reachable configuration of $\mathcal{M}$ and $[c] = (0.x, \sigma, 0.y, q)$, then $0.x \in [0, \frac{k-1}{k}]$ and similarly for $0.y$.*

3.4 Simulation of Turing machines — step 1: Capturing the transition function

The first step towards a simulation of a Turing Machine $\mathcal{M}$ using a GPAC is to simulate the transition function of $\mathcal{M}$ with a GPAC-computable function $\text{step}_{\mathcal{M}}$. The next step is to iterate the function $\text{step}_{\mathcal{M}}$ with a GPAC. Instead of considering configurations c of the machine, we will consider its rational configurations $[c]$ and use the helper functions defined previously. Theoretically, because $[c]$ is rational, we just need that the simulation works over rationals. But, in practice, because errors are allowed on inputs, the function $\text{step}_{\mathcal{M}}$ has to simulate the transition function of $\mathcal{M}$ in a manner which tolerates small errors on the input. We recall that δ is the transition function of the $\mathcal{M}$ and we write δ_i the i^{th} component of δ.

Definition 10. *We define:*

$$\text{step}_{\mathcal{M}} : \begin{cases} \mathbb{R}^4 & \longrightarrow \qquad \mathbb{R}^4 \\ \begin{pmatrix} x \\ s \\ y \\ q \end{pmatrix} & \longmapsto \begin{pmatrix} \text{choose}\left[\text{frac}(kx), \frac{x+L_{\delta_2}(q,s)}{k}\right] \\ \text{choose}\left[\text{int}(kx), \text{int}(ky)\right] \\ \text{choose}\left[\frac{y+L_{\delta_2}(q,s)}{k}, \text{frac}(ky)\right] \\ L_{\delta_1}(q,s) \end{pmatrix} \end{cases}$$

where $\text{choose}[a,b] = (1-L_{\delta_3}(q,s))a + L_{\delta_3}(q,s)b$ *and* L_{δ_i} *is given by definition 8.*

The function $\text{step}_{\mathcal{M}}$ simulates the transition function of the Turing Machine $\mathcal{M}$, as shown in the following result.

Lemma 7. *Let* $c_0, c_1, \ldots$ *be the sequence of configurations of* $\mathcal{M}$ *starting from* c_0. *Then*

$$\forall n \in \mathbb{N}, [c_n] = \text{step}_{\mathcal{M}}{}^{[n]}([c_0])$$

Now we want to extend the function $\text{step}_{\mathcal{M}}$ to work not only on rationals encodings of configurations but also on reals close to configurations, in a way which tolerates small errors on the input. That is we want to build a robust approximation of $\text{step}_{\mathcal{M}}$. We already have some results on L thanks to lemma **??**. We also have some results on $\text{int}(\cdot)$ and $\text{frac}(\cdot)$. However, we need to pay attention to the case of nearly empty tapes. This can be done by a shifting x by a small amount $(1/(2k))$ before computing the interger/fractional part. Then lemma 6 and lemma 2 ensure that the result is correct.

Definition 11. *Define:*

$$\overline{\text{step}}_{\mathcal{M}}(\tau,\lambda) : \begin{cases} \mathbb{R}^4 & \longrightarrow \qquad \mathbb{R}^4 \\ \begin{pmatrix} x \\ s \\ y \\ q \end{pmatrix} & \longmapsto \begin{pmatrix} \text{choose}\left[\overline{\text{frac}}(kx), \frac{x+L_{\delta_2}(q,s)}{k}, q, s\right] \\ \text{choose}\left[\overline{\text{int}}(kx), \overline{\text{int}}(ky), q, s\right] \\ \text{choose}\left[\frac{y+L_{\delta_2}(q,s)}{k}, \overline{\text{frac}}(ky), q, s\right] \\ L_{\delta_1}(q,s) \end{pmatrix} \end{cases}$$

where

$$\operatorname{choose}[a,b,q,s] = (1 - L_{\delta_3}(q,s))a + L_{\delta_3}(q,s)b$$

$$\overline{\operatorname{int}}(x) = \sigma_k\left(x + \frac{1}{2k}, \tau, \lambda\right)$$

$$\overline{\operatorname{frac}}(x) = x - \overline{\operatorname{int}}(x)$$

We now show that $\overline{\operatorname{step}}_{\mathcal{M}}$ is a robust version of $\operatorname{step}_{\mathcal{M}}$. We first begin with a lemma about function *choose*.

Lemma 8. *There exists $A_3 > 0$ and $B_3 > 0$ such that $\forall q, \bar{q}, s, \bar{s}, a, b, \bar{a}, \bar{b} \in \mathbb{R}$, if*

$$\left\|(\bar{a}, \bar{b})\right\| \leqslant M \qquad \text{and} \qquad q \in Q, s \in \Sigma \qquad \text{and} \qquad \|(q,s) - (\bar{q}, \bar{s})\| \leqslant 1$$

then

$$\left|\operatorname{choose}[a,b,q,s] - \operatorname{choose}[\bar{a}, \bar{b}, \bar{q}, \bar{s}]\right| \leqslant \left\|(a,b) - (\bar{a}, \bar{b})\right\| + 2MA_3\|(q,s) - (\bar{q}, \bar{s})\|$$

Furthermore, choose *is computable in polynomial space by a GPAC.*

Lemma 9. *There exists $a, b, c, d, e > 0$ such that for any $\tau, \lambda > 0$, any valid rational configuration $c = (x, s, y, q) \in \mathbb{R}^4$ and any $\bar{c} = (\bar{x}, \bar{s}, \bar{y}, \bar{q}) \in \mathbb{R}^4$, if*

$$\|(x,y) - (\bar{x}, \bar{y})\| \leqslant \frac{1}{2k^2} - \frac{1}{k\lambda} \qquad \text{and} \qquad \|(q,s) - (\bar{q}, \bar{s})\| \leqslant 1$$

then, for $p \in \{1, 3\}$

$$\begin{aligned}
&|\operatorname{step}_{\mathcal{M}}(c)_p - \overline{\operatorname{step}}_{\mathcal{M}}(\tau,\lambda)(\bar{c})_p| \leqslant k\|(x,y) - (\bar{x},\bar{y})\| + a\|(q,s) - (\bar{q},\bar{s})\| + b\\
&|\operatorname{step}_{\mathcal{M}}(c)_2 - \overline{\operatorname{step}}_{\mathcal{M}}(\tau,\lambda)(\bar{c})_2| \leqslant c\|(q,s) - (\bar{q},\bar{s})\| + d\\
&|\operatorname{step}_{\mathcal{M}}(c)_4 - \overline{\operatorname{step}}_{\mathcal{M}}(\tau,\lambda)(\bar{c})_4| \leqslant e\|(q,s) - (\bar{q},\bar{s})\|
\end{aligned}$$

Furthermore, $\overline{\operatorname{step}}_{\mathcal{M}}$ is computable in polynomial space by a GPAC.

We summarize the previous lemma into the following simpler form.

Corollary 2. *For any $\tau, \lambda > 0$, any valid rational configuration $c = (x, s, y, q) \in \mathbb{R}^4$ and any $\bar{c} = (\bar{x}, \bar{s}, \bar{y}, \bar{q}) \in \mathbb{R}^4$, if*

$$\|(x,y) - (\bar{x}, \bar{y})\| \leqslant \frac{1}{2k^2} - \frac{1}{k\lambda} \qquad \text{and} \qquad \|(q,s) - (\bar{q}, \bar{s})\| \leqslant 1$$

then

$$\left\|\operatorname{step}_{\mathcal{M}}(c) - \overline{\operatorname{step}}_{\mathcal{M}}(\tau,\lambda)(\bar{c})\right\| \leqslant O(1)(e^{-\tau} + \|c - \bar{c}\|)$$

Furthermore,

$$\overline{\operatorname{step}}_{\mathcal{M}} \in \operatorname{GSPACE}\left((\mathbb{R}_+^*)^2 \times [-1,1] \times [-m,m] \times [-1,1] \times [-k,k], O(1)\right)$$

3.5 Simulation of Turing machines — step 2: Iterating functions with differential equations

We will use a special kind of differential equations to perform the iteration of a map with differential equations. In essence, it relies on the following core differential equation

$$\dot{x}(t) = A\phi(t)(g - x(t)) \tag{Reach}$$

We will see that with proper assumptions, the solution converges very quickly to the *goal* g. However, (Reach) is a simplistic idealization of the system so we need to consider a perturbed equation where the goal is not a constant anymore and the derivative is subject to small errors

$$\dot{x}(t) = A\phi(t)(\bar{g}(t) - x(t)) + E(t) \tag{ReachPerturbed}$$

We will again see that, with proper assumptions, the solution converges quickly to the *goal* within a small error. Finally we will see how to build a differential equation which iterates a map within a small error.

We first focus on (Reach) and then (ReachPerturbed) to show that they behave as expected. In this section we assume ϕ is a positive C^1 function.

Lemma 10. *Let x be a solution of* (Reach)*, let $T, \lambda > 0$ and assume $A \geqslant \frac{\lambda}{\int_0^T \phi(u)du}$ then $|x(T) - g| \leqslant |g - x(0)|e^{-\lambda}$.*

Lemma 11. *Let $T, \lambda > 0$ and let x be the solution of* (ReachPerturbed) *with initial condition $x(0) = x_0$. Assume $|\bar{g}(t) - g| \leqslant \eta$, $A \geqslant \frac{\lambda}{\int_0^T \phi(u)du}$ and $E(t) = 0$ for $t \in [0, T]$. Then*

$$|x(T) - g| \leqslant \eta(1 + e^{-\lambda}) + |x_0 - g|e^{-\lambda}$$

We can now define a system that simulates the iteration of a function using a system based on (ReachPerturbed). It work as described in [15]. There are two variables for simulating each component f_i, $i = 1, \ldots, n$, of the function f to be iterated. There will be periods in which the function is iterated one time. In half of the period, half (n) of the variables will stay (nearly) constant and close to values $\alpha_1, \ldots, \alpha_n$, while the other remaining n variables update their value to $f_i(\alpha_1, \ldots, \alpha_n)$, for $i = 1, \ldots, n$. In the other half period, the second subset of variables is then kept constant, and now it is the first subset of variables which is updated to $f_i(\alpha_1, \ldots, \alpha_n)$, for $i = 1, \ldots, n$.

Definition 12. *Let $d \in \mathbb{N}$, $F : \mathbb{R}^d \to \mathbb{R}^d$, $\lambda \geqslant 1, \mu \geqslant 0$ and $u_0 \in \mathbb{R}^d$, we define*

$$\begin{cases} z(0) = u_0 \\ u(0) = u_0 \end{cases} \qquad \begin{cases} \dot{z}_i(t) = A\theta(t, B)(F_i(u(t)) - z_i(t)) \\ \dot{u}_i(t) = A\theta(t - 1/2, B)(z_i(t) - u_i(t)) \end{cases} \tag{Iterate}$$

where $A = 10(\lambda + \mu)^2$ and $B = 4(\lambda + \mu)$.

Theorem 2. *Let $d \in \mathbb{N}$, $F : \mathbb{R}^d \to \mathbb{R}^d$, $\lambda \geqslant 1$, $\mu \geqslant 0$, $u_0, c_0 \in \mathbb{R}^d$. Assume z, u are solutions to* (Iterate) *and let $\varDelta F$ and $M \geqslant 1$ be such that*

$$\forall k \in \mathbb{N}, \forall \varepsilon > 0, \forall x \in]-\varepsilon, \varepsilon[^d, \left\| F^{[k+1]}(c_0) - F\left(F^{[k]}(c_0) + x\right) \right\| \leqslant \varDelta F(\varepsilon)$$

$$\forall t \geqslant 0, \|u(t)\|, \|z(t)\|, \|F(u(t))\| \leqslant M = e^{\mu}$$

and consider

$$\begin{cases} \varepsilon_0 = \|u_0 - c_0\| \\ \varepsilon_{k+1} = (1 + 3e^{-\lambda})\varDelta F(\varepsilon_k + 2e^{-\lambda}) + 5e^{-\lambda} \end{cases}$$

Then

$$\forall k \in \mathbb{N}, \left\| u(k) - F^{[k]}(c_0) \right\| \leqslant \varepsilon_k$$

Furthermore, if $F \in \mathrm{GSPACE}\left([-M, M]^d, s_F\right)$ for $s_F : [-M, M] \to \mathbb{R}$ then $((\lambda, \mu, t, u_0) \mapsto u(t))$ is computable in polynomial space by a GPAC.

3.6 Simulation of Turing machines — step 3: Putting all pieces together

In this section, we will use results of both section 3.3 and section 3.5 to simulate Turing Machines with differential equations. Indeed, in section 3.3 we showed that we could simulate a Turing Machine by iterating a robust real map, and in section 3.5 we showed how to efficiently iterate a robust map with differential equations. Now we just have to put these results together.

Theorem 3. *Let $\mathcal{M}$ be a Turing Machine as in section 3.3, then there are functions $s_f : I \to \mathbb{R}^4$ and $f_{\mathcal{M}} \in \mathrm{GSPACE}\left(\mathbb{R}^4, s_f\right)$ such that for any sequence $c_0, c_1, \ldots,$ of configurations of $\mathcal{M}$ starting with input e:*

$$\forall S, T \in \mathbb{R}_+^*, \forall n \leqslant T, \|[c_n] - f_{\mathcal{M}}(S, T, n, e)\| \leqslant e^{-S}$$

and

$$\forall S, T \in \mathbb{R}_+^*, \forall n \leqslant T, s_f(S, T, n, e) = O(poly(S, T))$$

Acknowledgments. D.S. Graça was partially supported by *Fundação para a Ciência e a Tecnologia* and EU FEDER POCTI/POCI via SQIG - Instituto de Telecomunicações through the FCT project PEst-OE/EEI/LA0008/2011.

Olivier Bournez and Amaury Pouly were partially supported by ANR project SHAMAN, by DGA, and by DIM LSC DISCOVER project.

References

1. E. Asarin and O. Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *J. Comput. System Sci.*, 57(3):389–398, 1998.
2. A. Ben-Hur, H. T. Siegelmann, and S. Fishman. A theory of complexity for continuous time systems. *J. Complexity*, 18(1):51–86, 2002.

3. V. D. Blondel, O. Bournez, P. Koiran, and J. N. Tsitsiklis. The stability of saturated linear dynamical systems is undecidable. *J. Comput. System Sci.*, 62:442–462, 2001.
4. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer, 1998.
5. L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Amer. Math. Soc.*, 21(1):1–46, 1989.
6. O. Bournez, M. L. Campagnolo, D. S. Graça, and E. Hainry. Polynomial differential equations compute all real computable functions on computable compact intervals. *J. Complexity*, 23(3):317–335, 2007.
7. O. Bournez, D. S. Graça, and A. Pouly. On the complexity of solving initial value problems. Submitted to the conference ISSAC 2012: International Symposium on Symbolic and Algebraic Computation.
8. Olivier Bournez and Manuel L. Campagnolo. *New Computational Paradigms. Changing Conceptions of What is Computable*, chapter A Survey on Continuous Time Computations, pages 383–423. Springer-Verlag, New York, 2008.
9. V. Brattka. The emperor's new recursiveness: the epigraph of the exponential function in two models of computability. In M. Ito and T. Imaoka, editors, *Words, Languages & Combinatorics III*, Kyoto, Japan, 2000. ICWLC 2000.
10. V. Bush. The differential analyzer. A new machine for solving differential equations. *J. Franklin Inst.*, 212:447–488, 1931.
11. C. S. Calude and B. Pavlov. Coins, quantum measurements, and Turing's barrier. *Quantum Information Processing*, 1(1-2):107–127, April 2002.
12. B. Jack Copeland. Accelerating Turing machines. *Minds and Machines*, 12:281–301, 2002.
13. J. Copeland. Even Turing machines can compute uncomputable functions. In J. Casti, C. Calude, and M. Dinneen, editors, *Unconventional Models of Computation (UMC'98)*, pages 150–164. Springer, 1998.
14. E. B. Davies. Building infinite machines. *The British Journal for the Philosophy of Science*, 52:671–682, 2001.
15. D. S. Graça, M. L. Campagnolo, and J. Buescu. Computability with polynomial differential equations. *Adv. Appl. Math.*, 40(3):330–349, 2008.
16. D. S. Graça and J. F. Costa. Analog computers and recursive functions over the reals. *J. Complexity*, 19(5):644–664, 2003.
17. K.-I Ko. *Computational Complexity of Real Functions.* Birkhäuser, 1991.
18. J. M. Nyce. Guest editor's introduction. *IEEE Ann. Hist. Comput.*, 18:3–4, 1996.
19. P. Odifreddi. *Classical Recursion Theory*, volume 1. Elsevier, 1989.
20. M. B. Pour-El and J. I. Richards. *Computability in Analysis and Physics.* Springer, 1989.
21. C. E. Shannon. Mathematical theory of the differential analyzer. *J. Math. Phys. MIT*, 20:337–354, 1941.
22. H. T. Siegelmann, A. Ben-Hur, and S. Fishman. Computational complexity for continuous time dynamics. *Phys. Rev. Lett.*, 83(7):1463–1466, 1999.
23. M. Sipser. *Introduction to the Theory of Computation.* Course Technology, 2nd edition, 2005.
24. K. Weihrauch. *Computable Analysis: an Introduction.* Springer, 2000.