A Characterisation of Computable Functions over $\mathbb{R}$ Using Ordinary Differential Equations

Manon BLANC

École Polytechnique, LIX

What has been done?

- Ordinary differential equations: well-understood, used in many fields in applied science (describing dynamical systems for example).
- We also can see them as a computation model: Shannon ('42), Moore ('96), Costa, Graça, Hainry, Pouly, Bournez...
- We focus on their discrete counterparts: discrete ODEs.
  $\Rightarrow$ widely studied in numerical optimisation, combinatorial analysis
- Bournez & Durand ('19) established a connection with complexity theory : characterisation of polynomial time using discrete ODEs.

We prove:

Theorem (MCU'22)

A function $f : \mathbb{N}^d \to \mathbb{R}^{d'}$ *is computable in polynomial time if and only if there exists* $\tilde{f} : \mathbb{N}^{d+1} \to \mathbb{R}^{d'} \in \mathbb{LDL}^\bullet$ *such that for all* $x \in \mathbb{N}^d$, $n \in \mathbb{N}$, $\|\tilde{f}(x, 2^n) - f(x)\| \leq 2^{-n}$ *with* $\mathbb{LDL}^\bullet = [0, 1, \pi_i^k, \ell(x), +, -, \times, \bar{cond}(x), \frac{x}{2}; \textit{composition}, \textit{linear length ODE}]$

and:

Theorem

A continuous function $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ *is computable in polynomial time if and only if there exists* $\tilde{f} : \mathbb{R}^d \times \mathbb{N}^2 \to \mathbb{R}^{d'} \in \mathbb{LDL}^\oslash$ *such that for all* $x \in \mathbb{R}^d$, $X \in \mathbb{N}$, $x \in \left[-2^X, 2^X\right]$, $n \in \mathbb{N}$, $\|\tilde{f}(x, 2^X, 2^n) - f(x)\| \leq 2^{-n}$.

with $\mathbb{LDL}^\bullet$ and $\mathbb{LDL}^\oslash$ finite sets of functions and operators.

Discrete differential equations

We characterise polynomial time computable real numbers and polynomial time computable sequences over the reals (functions from $\mathbb{N}$ to $\mathbb{R}$).

Definition

Let $f : \mathbb{N} \times \mathbb{R} \mapsto \mathbb{R}$. The discrete derivative of f is defined by $f'(x, y) = \Delta f(x, y) = f(x+1, y) - f(x, y)$.

Computable analysis: Computability and Complexity

Definition (e.g. Ker I. Ko)

Let $x \in \mathbb{R}$. x is computable if and only if there exists a computable function $\phi : \mathbb{N} \mapsto \mathbb{D}$ such that for all $n \in \mathbb{N}$, $|\phi(n) - x| \leq 2^{-n}$.

Example

e, π are computable. $\sum_{i \geq 1} 2^{-BB(i)}$, where BB is the Busy Beavers function is not.

Computable analysis: Computability and Complexity

Definition (e.g. Ker I. Ko)

We say that the *time* (or *space*) complexity of a computable real number x is the time (or space) complexity of computing one of its Cauchy function, where the input n to a Cauchy function is written in unitary notation 0^n.

Definition (e.g. Ker I.Ko)

A function $f : \mathbb{R} \mapsto \mathbb{R}$ is computable if there exists a oracle Turing machine M such that, for all $x \in \mathbb{R}$ and ϕ a Cauchy function associated to x (CF_x), the function ψ computed by M with oracle ϕ ($\psi(n) = M^{\phi}(n)$) is in $CF_{f(x)}$.

Example: Primitive recursive functions

A function over the integers is primitive recursive, denoted $\mathscr{PR}$, if and only if it belongs to the smallest set of functions that contains

- constant function 0,
- the projection functions π_i^p,
- the functions successor s,
- and that is closed under composition and primitive recursion.

We have $\mathscr{PR} = [O, \pi_i^p, s; \mathit{composition}, \mathit{primitive\ recursion}]$

Example: Primitive recursive functions

Let $p \in \mathbb{N}$, $g : \mathbb{N}^p \to \mathbb{N}$ and $h : \mathbb{N}^{p+2} \to \mathbb{N}$.

The function $f = \mathrm{REC}(g, h) : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by primitive recursion from g and h if:

$$\begin{cases} f(0, y) = g(y) \\ f(x+1, y) = h(f(x, y), x, y) \end{cases}$$

We can reformulate f through differential discrete equation:

$$\begin{aligned} \frac{\partial f}{\partial x}(x, y) &= \overline{h}(f(x, y), x, y) \\ &= h(f(x+1, y), x+1, y) - h(f(x, y), x, y) \end{aligned}$$

Another example: Elementary functions and Grzegorczyk's hierarchy

- Class $\mathscr{E}^0$: contains the constant function 0, the projection functions π_i^p, the successor function s, and is closed under composition and bounded recursion.
- Class $\mathscr{E}^n$ for $n \geq 1$: defined similarly except that functions max and E_n are added to the list of initial functions.

Known results:

- $\mathscr{E}^3$: class of elementary functions (alternative definition by bounded sum and product)
- $\mathscr{E}^2_* = \mathsf{Linspace}$, $\mathscr{E}^2 = \mathscr{F}_{\mathsf{Linspace}}$ (linear space and polynomial growth)
- $\mathscr{E}^n \subsetneq \mathscr{E}^{n+1}$ for $n \geq 3$
- $\mathscr{PR} = \bigcup_i \mathscr{E}^i$

Linear ODEs give exactly the elementary functions.

Bounded recursion

Let $g : \mathbb{N}^p \to \mathbb{N}$, $h : \mathbb{N}^{p+2} \to \mathbb{N}$ and $i : \mathbb{N}^{p+1} \to \mathbb{N}$.

The function $f = \mathbf{BR}(g, h, i) : \mathbb{N}^{p+1} \to \mathbb{N}$ is defined by bounded recursion from g, h and i if

$$\begin{aligned} f(0, y) &= g(y) \\ f(x+1, y) &= h(f(x, y), x, y) \end{aligned}$$

under the condition that:

$$f(x, y) \leq i(x, y).$$

Algebras of functions

Summary

- Characterise **complexity classes** , in polynomial time, by algebras of functions
- How?
  - Take some basis functions
  - Allow classical operations such as composition
  - Use a recursion mechanism

Algebras of functions

Summary

- Characterise **complexity classes** , in polynomial time, by algebras of functions
- How?
  - Take some basis functions
  - Allow classical operations such as composition
  - Use a recursion mechanism
- Full recursion is too much (primitive recursion). Need to restrict it.
- Applications/goals: programming languages with performance guarantees

Recursion on notation (Cobham ('62))

Consider $s_0, s_1 : \mathbb{N} \to \mathbb{N}$

$$s_0(x) = 2 \cdot x \text{ and } s_1(x) = 2 \cdot x + 1.$$

Definition

Function f defined by bounded recursion on notations, i.e. $\mathbf{BRN}$, from functions g, h_0, h_1 et k when:

$$\begin{cases} f(0, y) = g(y) \\ f(s_0(x), y) = h_0(x, y, f(x, y)) \text{ for } x \neq 0 \\ f(s_1(x), y) = h_1(x, y, f(x, y)) \\ f(x, y) \leq k(x, y) \end{cases}$$

Cobham's approach

F_P smallest subset of primitive recursive functions

- Containing basis functions :
  $F_P = [0, \pi_i^k, s_0, s_1, \sharp; Composition, BRN]$
  with $\sharp$, a "smash funtion" defined by $\sharp(x, y) = 2^{|x| \times |y|}$

Cobham (62) : F_P is equal to FP, the class of polynomial time computable functions

Why it works

$$\begin{cases} f(0,y) = g(y) \\ f(s_0(x),y) = h_0(x,y,f(x,y)) \text{ for } x \neq 0 \\ f(s_1(x),y) = h_1(x,y,f(x,y)) \\ f(x,y) \leq k(x,y) \end{cases}$$

Why it works

$$\begin{cases} f(0,y) = g(y) \\ f(s_0(x),y) = h_0(x,y,f(x,y)) \text{ for } x \neq 0 \\ f(s_1(x),y) = h_1(x,y,f(x,y)) \\ f(x,y) \leq k(x,y) \end{cases}$$

- f is defined from h_0, h_1 and k.
- If $|k(x,y)|$ is polynomial in $|x|+|y|$, then so is $|f(x,y)|$
- Hence, inner terms do not grow too fast!

Why it works

$$\begin{cases} f(0,y) = g(y) \\ f(s_0(x),y) = h_0(x,y,f(x,y)) \text{ for } x \neq 0 \\ f(s_1(x),y) = h_1(x,y,f(x,y)) \\ f(x,y) \leq k(x,y) \end{cases}$$

- $|s_1(x)| = |s_0(x)| = |x| + 1$
- Then the number of induction steps is in $O(|x|)$.

Why it works

- Definition of useful functions (addition, concatenation, conditionals, etc) "easy"
- $\sharp(x, y) = 2^{|x| \times |y|}$, Hence $|\sharp(x, y)| = |x| + |y| + 1$.
- Help to obtain "counters" of polynomial size.

LDL and FP

Definition

- FP is the class of functions computed by deterministic Turing machines in polynomial time
- FPSPACE is the class of functions computed by deterministic Turing machines in polynomial space
- We have $FP \subseteq \#P \subseteq FPSPACE$

LDL and FP

Definition

- FP is the class of functions computed by deterministic Turing machines in polynomial time
- FPSPACE is the class of functions computed by deterministic Turing machines in polynomial space
- We have $FP \subseteq \#P \subseteq FPSPACE$

Theorem (Bournez & Durand, '19)

For discrete functions, we have $\mathbb{LDL} = FP$
where $\mathbb{LDL} =$
$[0, 1, \pi_i^k, \ell(x), +, -, \times, cond(x); composition, linear\ length\ ODE]$.

Definition (f Linear length ODE)

$$f(0,\mathrm{y}) = \mathrm{g}(\mathrm{y}) \quad \textit{and} \quad \frac{\partial \mathrm{f}(x,\mathrm{y})}{\partial \ell} = \mathrm{u}(\mathrm{f}(x,\mathrm{y}),\mathrm{h}(x,\mathrm{y}),x,\mathrm{y}) \qquad (1)$$

where u is *essentially linear* in $\mathrm{f}(x,\mathrm{y})$.

$\Rightarrow$ introduced in (Bournez & Durand '19) (1) is similar to classical formula for classical continuous ODEs:

$$\frac{\delta f(x,\mathrm{y})}{\delta x} = \frac{\delta \ell(x)}{\delta x} \cdot \frac{\delta f(x,\mathrm{y})}{\delta \ell(x)},$$

and hence this is similar to a change of variable: $t = \ell(x)$

ODE for complexity classes

- Elementary functions are of high complexity, but linear systems are the simplest kind of system
- Expressive and believed to help better understand computation for both discrete and continuous settings

$\mathbb{LDL} = \mathit{FPTIME}$: Why it works

Proof of ($\subseteq$): (main ideas)

- The derivation along $\ell(x)$ (or any $\mathscr{L}$ with polylog "jumps") permits to control the number of steps
- Linearity of the system permits to control the size of the output

Proof of ($\supseteq$): By a direct expression of a polynomial computation of a register machine.

Computing reals in polynomial time

$$\mathbb{LDL}^\bullet = [0, 1, \pi_i^k, \ell(x), +, -, \times, \bar{\text{cond}}(x), \tfrac{x}{2}; \textit{composition}, \textit{linear length ODE}]$$

with $\bar{\text{cond}}(x)$ some piecewise linear continuous function that takes value 1 for $x > \frac{3}{4}$ and 0 for $x < \frac{1}{4}$.

Proposition

All functions of $\mathbb{LDL}^\bullet$ are computable in the sense of computable analysis in polynomial time.

Proof.

By structural induction.

The case of the closure under composition:

- Take $COMP(f, g)$
- By induction hypothesis, f and g are in FP, so there exists two Turing machines M_f and M_g that respectively compute f and g
- Deterministic polynomial times Turing machines are closed under composition, so $COMP(f, g)$ is computable in deterministic polynomial time.

□

Computing reals in polynomial time : $FP \subseteq \mathbb{LDL}^\bullet$

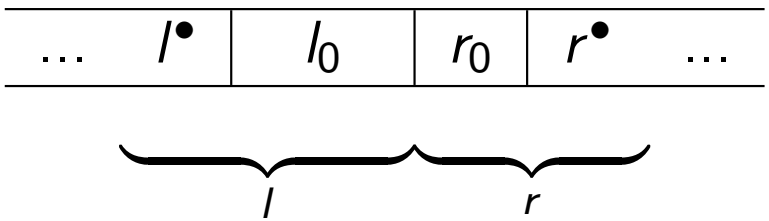
For any polynomial time computable function $f : \mathbb{N}^d \mapsto \mathbb{R}^d$, we can construct some function $\tilde{f} \in \mathbb{LDL}^\bullet$ that simulates the computation of f :

This basically requires to be able to simulate the computation of a Turing machine using the functions from $\mathbb{LDL}^\bullet$.

Computing reals in polynomial time : $FP \subseteq \mathbb{LDL}^\bullet$

Main ideas of the proof:

- We take a Turing machine M, with bi-infinite tapes computing f, we assume the reals are in base 4:



- We encode each transition of M with functions of $\mathbb{LDL}^\bullet$.
- We characterise the complete execution of M on some input as a composition of the transitions: we know M is polynomial, so we can bound the number of steps.

Computing sequences in polynomial time

We are now able to compute sequences in polynomial time $f : \mathbb{N} \mapsto \mathbb{R}$.
Main ideas of the proof:

- Encode the binary extension using only 1 and 3: computable in $\mathbb{LDL}^\bullet$

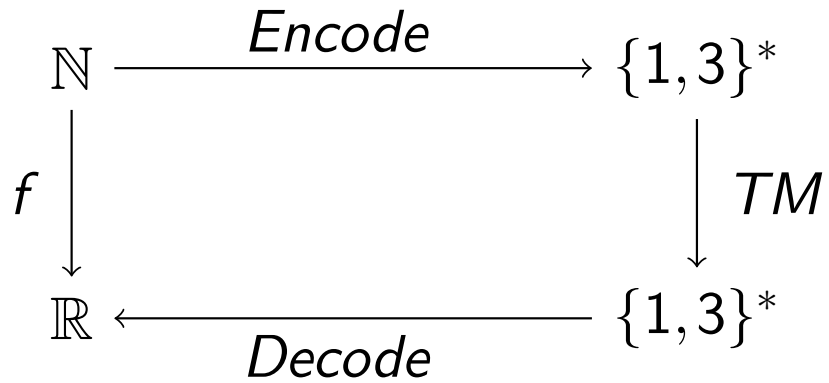
Why? $\rightarrow$ Functions in $\mathbb{LDL}^\bullet$ are all continuous. Here, we use discontinuous functions, but on disjoint intervals

Computing sequences in polynomial time

We are now able to compute sequences in polynomial time $f : \mathbb{N} \mapsto \mathbb{R}$.

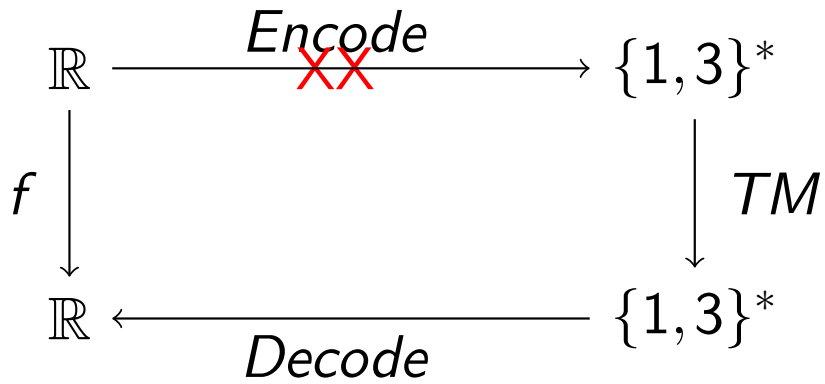
Main ideas of the proof:

- Encode the binary extension using only 1 and 3: computable in $\mathbb{LDL}^\bullet$
- We use the same trick as for computing reals.
- We decode the result: computable in $\mathbb{LDL}^\bullet$.



Function over the reals

We still need to characterise fuctions over the reals in polynomial time:



We introduce:

$\mathbb{LDL}^{\oslash} =$

$[0, 1, \pi_i^k, \ell(x), +, -, \bar{\text{cond}}(x), \frac{x}{2}; \textit{composition}, \textit{linear length ODE}]$,

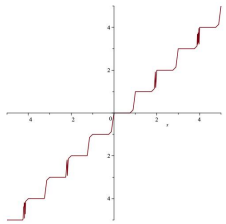
and we prove:

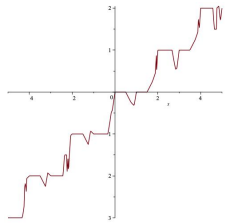
Theorem

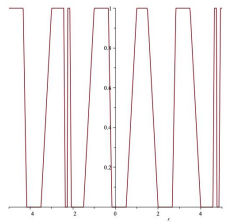
A continuous function $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ *is computable in polynomial time if and only if there exists* $\tilde{f} : \mathbb{R}^d \times \mathbb{N}^2 \to \mathbb{R}^{d'} \in \mathbb{LDL}^{\oslash}$ *such that for all* $x \in \mathbb{R}^d$, $X \in \mathbb{N}$, $x \in \left[-2^X, 2^X\right]$, $n \in \mathbb{N}$, $\|\tilde{f}(x, 2^X, 2^n) - f(x)\| \leq 2^{-n}$

Main ideas of the proof:

- We construct approximations of the fractional part, the integer part, the division by 2, the modulo 2 in $\mathbb{LDL}^{\oslash}$:


Integer Part


Division by 2


Modulo 2

- We construct two functions $f_1, f_2 \in \mathbb{LDL}^{\oslash}$ such that either f_1 or f_2 is equal to f on some intervals, and that overlap well.
- We use those functions to define another function λ, which is an adaptative barycenter such that :

$$f = \lambda f_1 + (1 - \lambda) f_2$$

- We use again the trick simulating the execution of the Turing machine with function in $\mathbb{LDL}^{\oslash}$.

Conclusion

We were able to characterise computable real numbers and sequences over the reals in polynomial time:

Theorem

A function $f : \mathbb{N}^d \to \mathbb{R}^{d'}$. *is computable in polynomial time if and only if all its components belong to* $\mathbb{L}\mathbb{D}\bar{\mathbb{L}}^\bullet$.

And functions over the reals:

Theorem

A continuous function $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ *is computable in polynomial time if and only if there exists* $\tilde{f} : \mathbb{R}^d \times \mathbb{N}^2 \to \mathbb{R}^{d'} \in \mathbb{L}\mathbb{D}\mathbb{L}^\oslash$ *such that for all* $x \in \mathbb{R}^d$, $X \in \mathbb{N}$, $x \in \left[-2^X, 2^X\right]$, $n \in \mathbb{N}$, $\|\tilde{f}(x, 2^X, 2^n) - f(x)\| \leq 2^{-n}$.