

A characterization of polynomial time computable functions from the integers to the reals using discrete ordinary differential equations

Manon Blanc, Olivier Bournez



Discrete differential equations

Computable analysis: Computability and Complexity

LDL and FP

LDL and FP

Computing real numbers in polynomial time

Computing sequences in polynomial time

Computing sequences in polynomial time

Conclusion

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.
⇒ 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

A function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time if and only if all its components belong to $\overline{\text{LDL}}^\bullet$.

with $\overline{\text{LDL}}^\bullet$ a finite set of functions and operators.

Discrete differential equations

We characterize 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 \mathcal{PR} , if and only if it belongs to the smallest set of functions that contains

- ▶ constant function $\mathbf{0}$,
- ▶ the projection functions π_i^p ,
- ▶ the functions successor \mathbf{s} ,
- ▶ and that is closed under **composition** and **primitive recursion**.

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

Example: Primitive recursive functions

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

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

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

We can reformulate f through differential discrete equation:

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

Another example: Elementary functions and Grzegorzczuk's hierarchy

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

Known results:

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

Linear ODEs give exactly the elementary functions.

Bounded recursion

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

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

$$f(0, \mathbf{y}) = g(\mathbf{y})$$

$$f(x + 1, \mathbf{y}) = h(f(x, \mathbf{y}), x, \mathbf{y})$$

under the condition that:

$$f(x, \mathbf{y}) \leq i(x, \mathbf{y}).$$

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 $\mathbf{s}_0, \mathbf{s}_1 : \mathbb{N} \rightarrow \mathbb{N}$

$$\mathbf{s}_0(x) = 2 \cdot x \text{ and } \mathbf{s}_1(x) = 2 \cdot x + 1.$$

Definition

Function f defined by **bounded recursion on notations**, i.e. **BRN**, from functions g, h_0, h_1 et k when:

$$\left\{ \begin{array}{l} f(0, \mathbf{y}) = g(\mathbf{y}) \\ f(\mathbf{s}_0(x), \mathbf{y}) = h_0(x, \mathbf{y}, f(x, \mathbf{y})) \text{ for } x \neq 0 \\ f(\mathbf{s}_1(x), \mathbf{y}) = h_1(x, \mathbf{y}, f(x, \mathbf{y})) \\ f(x, \mathbf{y}) \leq k(x, \mathbf{y}) \end{array} \right.$$

Cobham's approach

F_P smallest subset of primitive recursive functions

- ▶ Containing basis functions :

$$F_P = [0, \pi_i^k, s_0, s_1, \sharp; \textit{Composition}, \textit{BRN}]$$

with \sharp , a "smash function" 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

$$\left\{ \begin{array}{l} f(0, \mathbf{y}) = g(\mathbf{y}) \\ f(\mathbf{s}_0(x), \mathbf{y}) = h_0(x, \mathbf{y}, f(x, \mathbf{y})) \text{ for } x \neq 0 \\ f(\mathbf{s}_1(x), \mathbf{y}) = h_1(x, \mathbf{y}, f(x, \mathbf{y})) \\ f(x, \mathbf{y}) \leq k(x, \mathbf{y}) \end{array} \right.$$

Why it works

$$\left\{ \begin{array}{l} f(0, \mathbf{y}) = g(\mathbf{y}) \\ f(\mathbf{s}_0(x), \mathbf{y}) = h_0(x, \mathbf{y}, f(x, \mathbf{y})) \text{ for } x \neq 0 \\ f(\mathbf{s}_1(x), \mathbf{y}) = h_1(x, \mathbf{y}, f(x, \mathbf{y})) \\ f(x, \mathbf{y}) \leq k(x, \mathbf{y}) \end{array} \right.$$

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

Why it works

$$\begin{cases} f(0, \mathbf{y}) = g(\mathbf{y}) \\ f(\mathbf{s}_0(x), \mathbf{y}) = h_0(x, \mathbf{y}, f(x, \mathbf{y})) \text{ for } x \neq 0 \\ f(\mathbf{s}_1(x), \mathbf{y}) = h_1(x, \mathbf{y}, f(x, \mathbf{y})) \\ f(x, \mathbf{y}) \leq k(x, \mathbf{y}) \end{cases}$$

- ▶ $|\mathbf{s}_1(x)| = |\mathbf{s}_0(x)| = |x| + 1$
- ▶ Then the number of induction steps is in $O(|x|)$.

Why it works

- ▶ Definition of useful functions (addition, concatenation, conditionals, etc) "easy"
- ▶ $\#(x, y) = 2^{|x| \times |y|}$, Hence $|\#(x, y)| = |x| + |y| + 1$.
- ▶ 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 $\mathbb{LDL} = FPTIME$

where $\mathbb{LDL} =$

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

Definition (*f* Linear length ODE)

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

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

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

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

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

LDL = FPTIME: Why it works

Proof of (\subseteq): (main ideas)

- ▶ The derivation along $\ell(x)$ (or any \mathcal{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

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

with $\overline{\text{cond}}(x)$ 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.

$$\text{LDL} \subseteq \text{FP}$$

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 : $FP \subseteq LDL^\bullet$

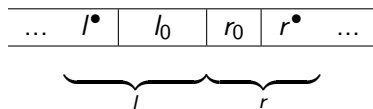
For any polynomial time computable function $f : \mathbb{N}^d \mapsto \mathbb{R}^d$, we can construct some function $\tilde{f} \in LDL^\bullet$ that **simulates the computation of f** :

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

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

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 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 LDL^\bullet

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

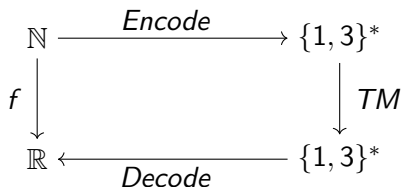
Computing sequences in polynomial time

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 $\mathbb{L}\mathbb{D}\mathbb{L}^\bullet$
- ▶ We use the same trick as for computing reals.
- ▶ We **decode** the result: computable in $\mathbb{L}\mathbb{D}\mathbb{L}^\bullet$.



Conclusion

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

Theorem

A function $\mathbf{f} : \mathbb{N}^d \rightarrow \mathbb{R}^{d'}$ is computable in polynomial time if and only if all its components belong to $\overline{\text{LDL}}^\bullet$.

- ▶ We still need to characterise functions over the reals in polynomial time (ongoing work)

