Recursion schemes, discrete differential equations and characterization of polynomial time computation

Olivier Bournez[1] Arnaud Durand[2]

[1]Institut Polytechnique de Paris

[2]Université de Paris

GT Calculabilités
Florence, 26 Nov 2019

Our article

- The basis for a **presentation of Complexity Theory** based on **discrete Ordinary Differential Equations**

#jekiffelesmachinesdeTuring

#jekiffeleséquationsdifférentielles

- Important demonstrations:
  - The particular role played by **linear (affine) ordinary differential equations** in complexity theory, and algorithm design,
  - The concept of **derivation along some particular function** (i.e. change of variable) to guarantee a low complexity.

Menu

Discrete ordinary differential equations

Programming with discrete ODEs

Algebra of functions for computability and complexity : the early days

On the expressive power of discrete ODE

Conclusion

Discrete derivative

Definition

Let $f : \mathbb{N} \to \mathbb{Z}$, the discrete derivative (a.k.a finite difference) is defined as:

$$\Delta \mathbf{f}(x) = \mathbf{f}(x+1) - \mathbf{f}(x).$$

When $f : \mathbb{N}^p \to Z^q$, denote:

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{f}(x+1, \mathbf{y}) - \mathbf{f}(x, \mathbf{y}).$$

Sometimes use $\mathbf{f}'(x)$ instead of $\Delta(\mathbf{f}(x))$

Discrete integral

Definition (Discrete Integral)

we write $\int_a^b \mathbf{f}(x)\delta x$ as a synonym for

$$\int_a^b \mathbf{f}(x)\delta x = \sum_{x=a}^{x=b-1} \mathbf{f}(x)$$

with the conventions: $\int_a^a \mathbf{f}(x)\delta x = 0$ and $\int_a^b \mathbf{f}(x)\delta x = -\int_b^a \mathbf{f}(x)\delta x$ when $a > b$.

It follows easily by the telescope formula that:

Theorem (Fundamental Theorem of Finite Calculus)

Let $\mathbf{F}(x)$ *be some function. Then,*

$$\int_a^b \mathbf{F}'(x)\delta x = \mathbf{F}(b) - \mathbf{F}(a).$$

- **Several results from classical derivatives generalize** to this settings:
  - $(a \cdot f(x) + b \cdot g(x))' = a \cdot f'(x) + b \cdot g'(x)$
  - $(f(x) \cdot g(x))' = f'(x) \cdot g(x+1) + f(x) \cdot g'(x) = f(x+1)g'(x) + f'(x)g(x)$
  - integration by parts . . .
  - etc.

- Many names/rediscovery of (sometimes rather surprising) **generalizations of classical (continuous) statements:**
  - also called: umbral calculus, discrete calculus, discrete fractional calculus, calculus of finite differences, difference equations, finite operator calculus, etc...

- **Several results from classical derivatives generalize** to this settings:
  - $(a \cdot f(x) + b \cdot g(x))' = a \cdot f'(x) + b \cdot g'(x)$
  - $(f(x) \cdot g(x))' = f'(x) \cdot g(x+1) + f(x) \cdot g'(x) = f(x+1)g'(x) + f'(x)g(x)$
  - integration by parts ...
  - etc.

- Many names/rediscovery of (sometimes rather surprising) **generalizations of classical (continuous) statements:**
  - also called: umbral calculus, discrete calculus, discrete fractional calculus, calculus of finite differences, difference equations, finite operator calculus, etc...

- We do not want to talk about combinatoris/computer algebra/math but **program** with ODEs!

Discrete Ordinary Differential Equation (ODE)

Discrete ODE: System of equations of the form, where h is some function:

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial x} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}), \tag{1}$$

When some initial value $\mathbf{f}(0, \mathbf{y}) = \mathbf{g}(\mathbf{y})$ is added, this is called an *Initial Value Problem (IVP)* or a *Cauchy Problem*.

An IVP can always be put in integral form

$$\mathbf{f}(x, \mathbf{y}) = \mathbf{f}(0, \mathbf{y}) + \int_0^x \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \delta x.$$

- Hence, a discrete ODE always have a solution $f : \mathbb{N}^p \to \mathbb{Z}^q$
- Not always true if one wants $f : \mathbb{Z}^p \to \mathbb{Z}^q$

Useful functions

- Falling power: $x^{\underline{m}} = x \cdot (x-1) \cdot (x-2) \cdots (x-(m-1))$.
  Derivation rule: $(x^{\underline{m}})' = m \cdot x^{\underline{m-1}}$
- Falling exponential:

$$\overline{2}^{\mathbf{U}(x)} = (1+\mathbf{U}'(x-1)) \cdots (1+\mathbf{U}'(1)) \cdot (1+\mathbf{U}'(0))$$
$$= \prod_{t=0}^{t=x-1} (1+\mathbf{U}'(t)).$$

  with the convention that $\prod_0^0 = \mathbf{id}$, where **id** is the identity (e.g. 1 for the scalar case)
  Derivation rule:

$$\left(\overline{2}^{\mathbf{U}(x)}\right)' = \mathbf{U}'(x) \cdot \overline{2}^{\mathbf{U}(x)}$$

Linear system of discrete ODE

Linear ODE: system of the form

$$\begin{cases} \mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(x, \mathbf{y}) \\ \mathbf{f}(0, \mathbf{y}) = \mathbf{G}(\mathbf{y}) \text{ (initial conditions)} \end{cases}$$

For matrices **A** and vectors **B** and **G**.

- Well known and simple kind of system
- Easy to solve in the continous setting

Linear system of discrete ODE

Easy to see that solution is of the form:

$$\mathbf{f}(x,\mathbf{y}) = \left(\overline{2}^{\int_0^x \mathbf{A}(t,\mathbf{y})\delta t}\right) \cdot \mathbf{G}(\mathbf{y}) + \int_0^x \left(\overline{2}^{\int_{u+1}^x \mathbf{A}(t,\mathbf{y})\delta t}\right) \cdot \mathbf{B}(u,\mathbf{y})\delta u.$$

Or, alternatively:

$$\mathbf{f}(x,\mathbf{y}) = \sum_{u=-1}^{x-1} \left(\prod_{t=u+1}^{x-1} (1+\mathbf{A}(t,\mathbf{y})) \right) \cdot \mathbf{B}(u,\mathbf{y})$$

with the conventions that $\prod_x^{x-1} \kappa(x) = 1$ and $\mathbf{B}(-1,\mathbf{y}) = \mathbf{G}(\mathbf{y})$

Computational content is clear: the solution can be computed

Some examples

Always suppose from now that we can use and compose the following functions:

- arithmetic operations: $+$, $-$, $\times$;
- $\ell(x)$ returns the length of $|x|$ written in binary;
- $\mathrm{sg}(x) : \mathbb{Z} \to \mathbb{Z}$ (respectively: $\mathrm{sg}_{\mathbb{N}}(x) : \mathbb{N} \to \mathbb{Z}$) that takes value 1 for $x > 0$ and 0 in the other case;

From these it comes:

- $\bar{\mathrm{sg}}(x)$ that stands for $\bar{\mathrm{sg}}(x) = (1 - \mathrm{sg}(x)) \times (1 - \mathrm{sg}(-x))$: it tests if $x = 0$ for $x \in \mathbb{Z}$;

- Conditional statements
  $\mathrm{if}(x < x', y, z)$ for $\mathrm{if}(\mathrm{sg}(x' - x + 1), y, z)$
  $\mathrm{if}(x = x', y, z)$ for $\mathrm{if}(1 - \bar{\mathrm{sg}}(x - x'), y, z)$.

Menu

Programming with discrete ODEs: Example 1: $\min f : x \mapsto min\{f(y) : 0 \leq y \leq x\}$

- Given by $F(x,x)$ where F is solution of the discrete ODE

$$\begin{aligned} F(0,x) &= f(0); \\ \frac{\partial F(t,x)}{\partial t} &= H(F(t,x), f(x), t, x), \end{aligned}$$

where $H(F,f,t,x) = 0$ if $F < f$, $f - F$ if $F \geq f$.

- In integral form:

$$F(x,y) = F(0) + \int_0^x H(F(t,y),t,y)\delta t.$$

Programming with discrete ODEs: Example 1: $\min f : x \mapsto min\{f(y) : 0 \leq y \leq x\}$

- Given by $F(x,x)$ where F is solution of the discrete ODE

$$\begin{array}{rcl} F(0,x) & = & f(0); \\ \frac{\partial F(t,x)}{\partial t} & = & H(F(t,x), f(x), t, x), \end{array}$$

  where $H(F,f,t,x) = 0$ if $F < f$, $f - F$ if $F \geq f$.

- In integral form:

$$F(x,y) = F(0) + \int_0^x H(F(t,y),t,y)\delta t.$$

- Basically: $F(t+1,x) = \text{if}(F(t,x) < f(x), F(t,x), f(x))$.

Programming with discrete ODEs: Example 1: $\min f : x \mapsto min\{f(y) : 0 \leq y \leq x\}$

- Given by $F(x,x)$ where F is solution of the discrete ODE

$$\begin{aligned} F(0,x) &= f(0); \\ \frac{\partial F(t,x)}{\partial t} &= H(F(t,x), f(x), t, x), \end{aligned}$$

where $H(F,f,t,x) = 0$ if $F < f$, $f - F$ if $F \geq f$.

- In integral form:

$$F(x,y) = F(0) + \int_0^x H(F(t,y),t,y)\delta t.$$

- Basically: $F(t+1,x) = \text{if}(F(t,x) < f(x), F(t,x), f(x))$.

- **Morality: An integral is an algorithm! and conversely!**

Programming with discrete ODEs: Example 1: $\min f : x \mapsto min\{f(y) : 0 \leq y \leq x\}$

- Given by $F(x, x)$ where F is solution of the discrete ODE

$$\begin{aligned} F(0, x) &= f(0); \\ \frac{\partial F(t,x)}{\partial t} &= H(F(t,x), f(x), t, x), \end{aligned}$$

where $H(F, f, t, x) = 0$ if $F < f$, $f - F$ if $F \geq f$.

- In integral form:

$$F(x, y) = F(0) + \int_0^x H(F(t, y), t, y)\delta t.$$

- Basically: $F(t+1, x) = \text{if}(F(t, x) < f(x), F(t, x), f(x))$.
- **Morality: An integral is an algorithm! and conversely!**
- **Time complexity?** x, not polynomial in **the size** $\ell(x)$ of x.

Programming with discrete ODEs: Example 2: $\lfloor\sqrt{x}\rfloor = \max\{y \leq x : y \cdot y \leq x\}$

- **General method:**
  - Let f, h be some functions with h being non decreasing.
  - **Compute some_h with $\mathrm{some}_h(x) = y$ s.t. $|f(x) - h(y)|$ is minimal.**
  - When $h(x) = x^2$ and $f(x) = x$, it holds that: $\lfloor\sqrt{x}\rfloor = \mathrm{if}(\mathrm{some}_h(x)^2 \leq x, \mathrm{some}_h(x), \mathrm{some}_h(x) - 1)$.
- The function some_h can be computed (in non-polynomial time) as a solution of an ODE as previously.
- More efficient (polynomial time) way: **perform a change of variable** so that the search becomes logarithmic in x!

Compute $\text{some}_h(x) = y$ s.t. $|f(x) - h(y)|$ is minimal h being non decreasing.

- Write:

$$\text{some}_h(x) = G(\ell(x), x)$$

for some function $G(t, x)$ solution of

$$\begin{aligned} G(0, x) &= x; \\ \frac{\partial G(t,x)}{\partial t} &= E(G(t, x), t, x) \end{aligned}$$

where

$$E(G, t, x) = \begin{cases} 2^{\ell(x)-t-1} & \text{whenever } h(G) > f(x), \\ 0 & \text{whenever } h(G) = f(x) \\ -2^{\ell(x)-t-1} & \text{whenever } h(G) < f(x). \end{cases}$$

- Other examples:
  - In the article: Computing Suffix using ODEs, . . . ,
  - . . .

- **Which functions can be programmed with discrete ODEs?**

Menu

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:

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

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.

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

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

under the condition that:

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

Elementary functions and Grzegorczyk's hierarchy

- Bounded recursion makes sense when initial functions are restricted
- Consider the family of functions E_n defined by induction as follows. When f is a function, $f^{[d]}$ denotes its d-th iterate.

$$
\begin{aligned}
\mathbf{E}_0(x) &= s(x) = x + 1, \\
\mathbf{E}_1(x, y) &= x + y, \\
\mathbf{E}_2(x, y) &= (x + 1) \cdot (y + 1), \\
\mathbf{E}_3(x) &= 2^x, \\
\mathbf{E}_{n+1}(x) &= \mathbf{E}_n^{[x]}(1) \text{ for } n \geq 3.
\end{aligned}
$$

Elementary functions and Grzegorczyk'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.
- $\mathcal{E}_*^n$: associated relational class

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 $n \geq 3$
- $\mathcal{PR} = \bigcup_i \mathcal{E}^i$

Algebras of functions

Summary

- Charaterize complexity classes 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)

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

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

Definition

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

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

Cobham's approach

$\mathscr{F}_P$ smallest subset of primitive recursive functions

- Containing basis functions :
  Function**0**, projections p_i^k, successor functions $\mathbf{s}_0(x) = 2 \cdot x$ and $\mathbf{s}_1(x) = 2 \cdot x + 1$, "smash" function $x \sharp y = 2^{|x| \times |y|}$
- Closed by composition
- Closed by bounded recursion on notations

Cobham (62) : $\mathscr{F}_P$ is equal to **FP**, the class of polynomial time computable functions

Why it works

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

Why it works

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

Why it works

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

- f is defined from h_0, h_1 and k.
- If $|k(x,\mathbf{y})|$ is polynomial in $|x| + |y|$, 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 $O(|x|)$.

Going further: syntactic restriction, ramified recursion

- Cobham's work was the starting point of numerous attempts to capture complexity classes by recursion algebras
- Generalize to $\mathbf{L}, \mathrm{NC}^i, \mathrm{AC}^i$ classes
- Alternative approaches that do not require to bound the function *a priori*.
  - Predicative recursion (Bellantoni, Cook)
  - Ramified recurrence (Leivant, Leivant-Marion)
  - ....

Menu

Discrete ODE for primitive recursive functions

Definition ((Scalar) Discrete ODE schemata)

Let $g : \mathbb{N}^p \to \mathbb{N}$ and $h : \mathbb{Z} \times \mathbb{N}^{p+1} \to \mathbb{Z}$.
Function f is defined by discrete ODE solving from g and h, denoted by $f = \mathrm{ODE}(g, h)$, if $f : \mathbb{N}^{p+1} \to \mathbb{Z}$ if f solution of:

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

When h is linear : LI schemata.

Discrete ODE for primitive recursive functions

What about the smallest classes of functions

- that contains $\mathbf{0}$, the projections π_i^p, the successor $\mathbf{s}$, addition $+$, subtraction $-$
- that is closed under composition and discrete ODE schemata (respectively: scalar discrete ODE schemata) LI.

Result: Its restriction to functions with values in $\mathbb{N}$ is equal to the set of primitive recursive functions.

Discrete ODE for elementary functions

What about the smallest classes of functions

- that contains **0**, the projections π_i^p, the successor **s**, addition $+$, subtraction $-$
- that is closed under composition and discrete **linear** ODE schemata (respectively: scalar discrete **linear** ODE schemata) $\mathbb{LI}$.

Result: Its restriction to functions with values in $\mathbb{N}$ is equal to $\mathcal{E}$, the set of elementary functions.

Remark: recall the definition of bounded sum and bounded product.

Bounded sum and product

Let $g : \mathbb{N}^{p+1} \to \mathbb{N}$,

- Let $f = \mathrm{BSUM}_<(g) : \mathbb{N}^{p+1} \to \mathbb{N}$ be defined as $f : (x, \mathbf{y}) \mapsto \sum_{z<x} g(z, \mathbf{y})$ for $x \neq 0$, and 0 for $x = 0$.
  Function f is the unique solution of initial value problem :

$$\begin{cases} \frac{\partial f(x,\mathbf{y})}{\partial x} = g(x, \mathbf{y}) \\ f(0, \mathbf{y}) = 0 \end{cases}$$

- Let $f = \mathrm{BPROD}_<(g)$ be defined as $f : (x, \mathbf{y}) \mapsto \prod_{z<x} g(z, \mathbf{y})$ for $x \neq 0$, and 1 for $x = 0$.
  Function f is the unique solution of initial value problem

$$\begin{cases} \frac{\partial f(x,\mathbf{y})}{\partial x} = f(x, \mathbf{y}) \cdot (g(x, \mathbf{y}) - 1) \\ f(0, \mathbf{y}) = 1 \end{cases}$$

ODE for complexity classes ?

- Elementary functions are of high complexity
- But linear systems are the simplest kid of system
- What can we do (i.e. what can we restrict more) to capture smaller complexity classes and in particular the class of polynomial time computable functions **FP**?

Derivation along a function

Definition ($\mathcal{L}$-ODE)

Let $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$. We write

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}(x, \mathbf{y})} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}), \tag{2}$$

as a formal synonym for
$\mathbf{f}(x+1, \mathbf{y}) = \mathbf{f}(x, \mathbf{y}) + (\mathcal{L}(x+1, \mathbf{y}) - \mathcal{L}(x, \mathbf{y})) \cdot \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y})$.

Inspired by the classical formula:

$$\frac{\delta f(x, \mathbf{y})}{\delta x} = \frac{\delta \mathcal{L}(x, \mathbf{y})}{\delta x} \cdot \frac{\delta f(x, \mathbf{y})}{\delta \mathcal{L}(x, \mathbf{y})}.$$

$\mathcal{L}$-ODE

The equality

$$\frac{\delta f(x,\mathbf{y})}{\delta x} = (\mathcal{L}(x+1,\mathbf{y}) - \mathcal{L}(x,\mathbf{y})) \cdot \mathbf{h}(\mathbf{f}(x,\mathbf{y}), x, \mathbf{y})$$

implies that the value of the derivative i.e. the variation of the function has to be considered only when

$$\mathcal{L}(x+1,\mathbf{y}) - \mathcal{L}(x,\mathbf{y}) \neq 0$$

Consequence: only as many values to consider to compute $f(x,\mathbf{y})$ as the number of times $\mathcal{L}(t,\mathbf{y})$ changes between $t=0$ and $t=x$...

Application: if $\mathcal{L}(x,\mathbf{y}) = \ell(x)$ then only a logarithmic in x number of values

Key observation:
Relating to a change of variable.

- **Key observation.** Assume that (2) holds.
  Then $\mathbf{f}(x, \mathbf{y})$ is given by

$$\mathbf{f}(x, \mathbf{y}) = \mathbf{F}(\mathcal{L}(x, \mathbf{y}), \mathbf{y})$$

  where $\mathbf{F}$ is the solution of initial value problem

$$\begin{array}{rcl} \mathbf{F}(0, \mathbf{y}) & = & \mathbf{f}(\mathcal{L}(0, \mathbf{x}), \mathbf{y}); \\ \frac{\partial \mathbf{F}(t, \mathbf{y})}{\partial t} & = & \Delta \mathcal{L}(t, \mathbf{y}) \cdot \mathbf{h}(\mathbf{F}(t, \mathbf{y}), t, \mathbf{y}). \end{array}$$

  where $k \in \mathbb{N}$, $f : \mathbb{N}^{p+1} \to \mathbb{Z}^d$, $\mathcal{L} : \mathbb{N}^{p+1} \to \mathbb{Z}$ are some functions.

Fundamental observation: Linear ODEs

- **Fundamental observation:** Consider the ODE

$$\mathbf{f}'(x, \mathbf{y}) = \mathbf{A}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}) \cdot \mathbf{f}(x, \mathbf{y}) + \mathbf{B}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}). \tag{3}$$

Assume:

1. The initial condition $\mathbf{G}(\mathbf{y}) =^{def} \mathbf{f}(0, \mathbf{y})$, as well as Matrix $\mathbf{A}$ and vector $\mathbf{B}$ are polynomial time computable.
2. $\ell(\|\mathbf{A}(f, x, \mathbf{y})\|) \leq \ell(\|\mathbf{f}\|) + p_{\mathbf{A}}(x, \ell(\mathbf{y}))$ for some polynomial p_A
3. $\ell(\|\mathbf{B}(f, x, \mathbf{y})\|) \leq \ell(\|\mathbf{f}\|) + p_{\mathbf{B}}(x, \ell(\mathbf{y}))$ for some polynomial p_B

Then[1] its solution $\mathbf{f}(x, \mathbf{y})$ is **polynomial time computable** in x and the length of $\mathbf{y}$.

[1] $\| \cdots \|$ stands for the sup norm.

Towards capturing **FP**

- It is easily seen that the solution of

$$\frac{\partial f(x)}{\partial \ell(x)} = f(x) \cdot (f(x) - 1) \tag{4}$$

  is a fast growing function (output is exponential in size)
- Idea: combine linearity and derivation along some particular function $\mathcal{L}$ i.e. systems :

$$\frac{\partial \mathbf{f}(x, \mathbf{y})}{\partial \mathcal{L}} = \mathbf{h}(\mathbf{f}(x, \mathbf{y}), x, \mathbf{y}), \tag{5}$$

  where
  - h is "linear"
  - $\mathcal{L}$ has a polylogarithmic number of jumps. For example, $\mathcal{L}(x) = \ell(x)$.

A extended notion of linearity

$P(x_1, ..., x_h)$: expression built-on

- $+, -, \times$ and sg()
- over variables $V = \{x_1, ..., x_h\}$ and integer constants.

The degree $\deg(x, P)$ of a term $x \in V$ in P is defined inductively:

- $\deg(x, x) = 1$ and for $x' \in X \cup \mathbb{Z}$ such that $x' \neq x$, $\deg(x, x') = 0$
- $\deg(x, P + Q) = \max\{\deg(x, P), \deg(x, Q)\}$
- $\deg(x, P \times Q) = \deg(x, P) + \deg(x, Q)$
- $\deg(x, \mathrm{sg}(P)) = 0$

A extended notion of linearity

- An expression P is essentially constant in x if $\deg(x, P) = 0$.
- It is essentially linear in x if it is of the form $A \cdot x + B$ where A, B are essentially constant in x.

Example

- The expression $P(x, y, z) = x \cdot \mathrm{sg}((x^2 - z) \cdot y) + y^3$ is linear in x, essentially constant in z and not linear in y.
- The expression $P(x, 2^{\ell(y)}, z) = \mathrm{sg}(x^2 - z) \cdot z^2 + 2^{\ell(y)}$ is essentially constant in x, essentially linear in $2^{\ell(y)}$ (but not essentially constant) and not essentially linear in z.

Linear $\mathcal{L}$-ODE

Definition

Function $\mathbf{f}$ is linear $\mathcal{L}$-ODE definable from $\mathbf{u}$, $\mathbf{g}$ and $\mathbf{h}$ if it corresponds to the solution of the $\mathcal{L}$-IVP

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

where $\mathbf{u}$ is *essentially linear* in $\mathbf{f}(x, \mathbf{y})$.
When $\mathcal{L}(x, \mathbf{y}) = \ell(x)$, such a system is called linear length-ODE.

Definition ($\mathbb{DL}$)

Let $\mathbb{DL}$ be the smallest subset of functions,

- that contains $\mathbf{0}$, $\mathbf{1}$, projections π_i^p, the length $\ell(x)$, functions $x+y$, $x-y$, $x \times y$, the sign function $\mathrm{sg}(x)$
- closed under composition (when defined) and linear length-ODE scheme.

A characterization of **FP**

Theorem: $\mathbb{DL} = \mathbf{FP}$

Proof of ($\subseteq$): Roughly speaking

- 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.

Some Other Results in the Article

- **Normal form theorem:**:

details

- **A characterization of FNP:**

details

Skip

Some Other Results in the Article

- **Normal form theorem:**:

  - A function $\mathbf{f} : \mathbb{N}^p \to \mathbb{Z}$ is in **FP** iff

    $$f(\mathbf{y}) = \mathbf{g}(\ell(\mathbf{y})^c, \mathbf{y})$$

    for some integer c and some $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ solution of a **normal linear length-ODE**.

details

- **A characterization of FNP:**

details

Skip

Some Other Results in the Article

- **Normal form theorem:**:
  - A function $\mathbf{f} : \mathbb{N}^p \to \mathbb{Z}$ is in **FP** iff

$$f(\mathbf{y}) = \mathbf{g}(\ell(\mathbf{y})^c, \mathbf{y})$$

    for some integer c and some $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{Z}^k$ solution of a **normal linear length-ODE**.

details

- **A characterization of FNP:**
  - Take above $\mathbf{g} : \mathbb{N}^{p+1} \to \mathbb{N}^k$, solution of a **normal linear length-ODE with parameter**

$$\frac{\partial \mathbf{g}(x, \mathbf{y})}{\partial \ell(x)} = \mathbf{u}(\mathbf{g}(x, \mathbf{y}), w(x, \mathbf{y}), x, \mathbf{y})$$

    for some bounded $w : \mathbb{N}^{p+1} \to \mathbb{N}$.

details

Skip

Menu

Conclusion

- The basis for a **presentation of Complexity Theory** based on **discrete Ordinary Differential Equations**
- Important demonstrations:
  - The particular role played by **linear (affine) ordinary differential equations** in complexity theory, and algorithm design,
  - the concept of **derivation along some particular function** (i.e. change of variable) to guarantee a low complexity.

Further works

- Characterizations of **Other complexity classes**?
  - $\mathbf{P}_{[0,1]}$ of functions computable in polynomial time over the reals in the sense of computable analysis.
  - **FPSPACE**.
  - other classes?
- **Revisiting classical algorithmic under this viewpoint:**
  - Ex: The Master Theorem can be basically read as a result on (the growth of) a particular class of discrete time length ODEs.
  - Several recursive algorithms can then be reexpressed as particular discrete ODEs of specific type.
- Relations to **analog computability, computability with continous ODEs.**

Stephen Bellantoni and Stephen Cook.
A new recursion-theoretic characterization of the poly-time functions.
Computational Complexity, 2:97–110, 1992.

Olivier Bournez, Daniel S. Graça, and Amaury Pouly.
Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. The General Purpose Analog Computer and Computable Analysis are two efficiently equivalent models of computations.
In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 109:1–109:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

Olivier Bournez and Amaury Pouly.
A universal ordinary differential equation.
In *International Colloquium on Automata Language Programming, ICALP'2017*, 2017.

Manuel L. Campagnolo.
Computational Complexity of Real Valued Recursive Functions and Analog Circuits.
PhD thesis, Universidade Técnica de Lisboa, 2001.

Manuel L. Campagnolo, Cristopher Moore, and José Félix Costa.
An analog characterization of the Grzegorczyk hierarchy.
Journal of Complexity, 18(4):977–1000, 2002.

Alan Cobham.
The intrinsic computational difficulty of functions.
In Y. Bar-Hillel, editor, *Proceedings of the International Conference on Logic, Methodology, and Philosophy of Science*, pages 24–30. North-Holland, Amsterdam, 1962.

Francois Fages, Guillaume Le Guludec, Olivier Bournez, and Amaury Pouly.
Strong turing completeness of continuous chemical reaction networks and compilation of mixed analog-digital programs.

In *Computational Methods in Systems Biology-CMSB 2017*, 2017.

Daniel Leivant.
Predicative recurrence and computational complexity I: Word recurrence and poly-time.
In Peter Clote and Jeffery Remmel, editors, *Feasible Mathematics II*, pages 320–343. Birkhäuser, 1994.

Daniel Leivant and Jean-Yves Marion.
Lambda calculus characterizations of Poly-Time.
Fundamenta Informatica, 19(1,2):167,184, September 1993.

Programming with discrete ODEs: Example 3: suffix(x, y)

- **suffix**(x, y) outputs the $\ell(y) = t$ least significant bits of the binary decomposition of x.

- suffix$(x, y) = F(\ell(x), x)$ where

$$\begin{array}{rcl} F(0, y) & = & x; \\ \frac{\partial F(T,y)}{\partial T} & = & \text{if}(\ell(F(t,x)) = 1, 0, -2^{\ell(F(t,x))-1}). \end{array}$$

back

Programming with discrete ODEs: Example 3: suffix(x, y)

- **suffix**(x, y) outputs the $\ell(y) = t$ least significant bits of the binary decomposition of x.

- suffix$(x, y) = F(\ell(x), x)$ where

$$\begin{array}{rcl} F(0, y) & = & x; \\ \frac{\partial F(T,y)}{\partial T} & = & \text{if}(\ell(F(t,x)) = 1, 0, -2^{\ell(F(t,x))-1}). \end{array}$$

- **Basically:** we are using a fix-point definition of the function: suffix$(x, y) = F(\ell(x), y)$ where $F(0, x) = x$; $F(t + 1, x) = \text{if}(\ell(F(t, x)) = 1, F(t, x), F(t, x) - 2^{\ell(F(t,x))-1})$.

back