Some foundational questions (and some answers) about algorithms

Yiannis N. Moschovakis
UCLA and University of Athens

LSFA Workshop, Fortaleza, 26 September, 2018

There is no standard definition of **algorithms**

. . . which makes it difficult to formulate and prove results about all of them; but we can try!

- Using as a basic example the classical Euclidean algorithm which computes the greatest common divisor of two numbers, I will ask some natural questions about algorithms
- More than half the lecture will be dedicated to introducing intuitively and then formulating precise versions of these questions
- At the end I will discuss answers to three of these questions which are somewhat surprizing

- I will simplify a little, but (quoting John Steel) all lies are white

► This material is from the monograph

Abstract recursion and intrinsic complexity (ARIC)

forthcoming in the *Lecture Notes in Logic* series published by the Association for Symbolic Logic and Cambridge University Press

The Euclidean algorithm ε (with division) for $\gcd(x, y)$

- The Division Theorem for $\mathbb{N} = \{0, 1, \ldots\}$: For $x, y \in \mathbb{N}$ with $y > 0$, there are unique numbers $q = \mathrm{iq}(x, y), r = \mathrm{rem}(x, y)$ such that

$$x = yq + r, \quad 0 \leq r < y$$

- $\gcd(x, y) = \max\{t \mid \mathrm{rem}(x, t) = \mathrm{rem}(y, t) = 0\} \quad (x, y \geq 1)$
- Specification of ε by a $\boxed{\texttt{while program}}$: given $x, y \in \mathbb{N}$:

$$(\varepsilon) \qquad \texttt{while } y \neq 0 \ \Big\{ x := y; \ \ y := \mathrm{rem}(x, y) \Big\} \ \texttt{return } x$$

▶ **Fact**. *If $y = 0$, then ε returns x, and if $y \neq 0$, then ε returns* $\gcd(x, y)$

- Equivalent specification of ε by a $\boxed{\text{recursive program}}$:

▶ **Fact**. *The recursive equation (in the function variable p)*

$$(\varepsilon) \qquad p(x, y) = \text{if } (y = 0) \text{ then } x \text{ else } p(y, \mathrm{rem}(x, y))$$

has a unique (total) solution $\overline{p}(x, y)$, and

$$\overline{p}(x, y) = \text{if } (y = 0) \text{ then } x \text{ else } \overline{p}(x, y) = \gcd(x, y)$$

The complexity of the Euclidean

- $\boxed{c_\varepsilon(x,y) = \text{the number of calls to rem that } \varepsilon \text{ makes on the input } x, y}$

  (We do not count calls to $\mathrm{eq}_0(y) \iff y = 0$—we could)

▶ **Fact**: *If* $x \geq y \geq 2$, *then* $\boxed{c_\varepsilon(x,y) \leq 2\log y \leq 2\log x}$ $\quad(\log = \log_2)$

- **Basic question**: *Is the Euclidean optimal* (in some natural sense), *on some infinite set of inputs?*

- **Main Conjecture:** *For every* $\boxed{\textit{algorithm } \alpha \textit{ from } \mathrm{rem} \textit{ and } \mathrm{eq}_0}$ *which computes* $\gcd(x,y)$ *when* $x, y \geq 1$, *there is a number* $\delta > 0$, *such that for infinitely many pairs* (x,y) *with* $x > y \geq 1$,

$$\boxed{c_\alpha(x,y) = \text{the number of calls } \alpha \text{ makes to rem} \geq \delta \log x}$$

- *The Main Conjecture is not about Turing machines with oracles, which can compute* $\gcd(x,y)$ *with no oracle calls at all*

▶ **Fact**. *For the Fibonacci numbers* $F_0 = 0,\ F_1 = 1,\ F_{k+2} = F_k + F_{k+1}$,

$\boxed{c_\varepsilon(F_{k+1}, F_k) \geq (1/2)\varphi \log F_{k+1}}$ $\quad$ (where $\varphi = (1/2)(1+\sqrt{5}),\ k \geq 2$)

Partial functions and (partial) structures

- A partial function $f : X \rightharpoonup W$ is a (total) function $f : D_f \to W$ on some set $D_f \subseteq X$, its domain of convergence
- $$f(x)\downarrow \iff x \in D_f, \quad f(x)\uparrow \iff x \notin D_f,$$
$$f(x) = g(x) \iff [f(x)\uparrow \ \& \ g(x)\uparrow] \vee (\exists w \in W)[f(x) = g(x) = w],$$
$$f \sqsubseteq g \iff (\forall x)[x \in D_f \implies f(x) = g(x)],$$
$$f(g(x), h(x)) = w$$
$$\iff (\exists u, v)[g(x) = u \ \& \ h(x) = v \ \& \ f(u, v) = w]$$
- Unified notation for n-ary partial functions and relations on a set A:
$$\boxed{f : A^n \rightharpoonup A_s} \quad (s \in \{\mathtt{ind}, \mathtt{boole}\}, A_{\mathtt{ind}} = A, A_{\mathtt{boole}} = \{\mathrm{tt}, \mathrm{ff}\})$$
- A vocabulary is a finite set $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ of function symbols, each with an assigned sort s and arity n_i
- A (partial) Φ-structure is a tuple $\boxed{\mathbf{A} = (A, \mathbf{\Phi}) = (A, \varphi_1^{\mathbf{A}}, \ldots, \varphi_m^{\mathbf{A}})}$, where for each i, $\boxed{\varphi_i^{\mathbf{A}} : A^{n_i} : A_s}$
- (Structures with many sorts (data types) are disjoint unions of these)

Two kinds of algorithms on a Φ-structure $\mathbf{A} = (A, \mathbf{\Phi})$

- With variables v_i of sort `ind` over A (and obvious restrictions):

(Terms) $\boxed{E :\equiv \mathrm{v}_i \mid \varphi_i(E_1, \ldots, E_n) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2}$

► (1) The iterative algorithms of $\mathbf{A}$ (of sort `ind` or `boole`) are specified by `while` programs, using partial functions on A defined by terms

— these include all algorithms on $\mathbf{A}$ specified by the familiar computation models (Turing machines, straight line and finite register programs, decision trees, random access machines, etc.)

- Adding pf variables $\mathrm{p}_i^{n,\mathtt{ind}}, \mathrm{p}_i^{n,\mathtt{boole}}$ on A of every arity $n \in \mathbb{N}$:

(Terms) $\boxed{E :\equiv \mathrm{v}_i \mid \mathrm{p}_i^{n,s}(E_1, \ldots, E_n) \mid \varphi_i(E_1, \ldots, E_n) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2}$

► (2) The recursive algorithms of $\mathbf{A}$ are specified by recursive programs

$$E :\equiv E_0 \text{ where } \left\{\mathrm{p}_1(\vec{\mathrm{u}}_1) = E_1, \ldots, \mathrm{p}_K(\vec{\mathrm{u}}_K) = E_K\right\}$$

- Compute: plug the least fixed points $\overline{p}_1, \ldots, \overline{p}_K$ of the body into E_0

Iteration vs. recursion on a ("nice", infinite) structure **A**

- If $f : A^n \rightharpoonup A_s$ with $s = \texttt{ind}$ or $s = \texttt{boole}$:

f is iterative on **A** $\iff$ f is computed in **A** by a `while` program,
f is recursive in **A** $\iff$ f is computed in **A** by a recursive program

- **Fact**. Reduction of iteration to recursion: *Every iterative partial function of* **A** *is recursive in* **A** (effectively)
- **Fact**. Partial reduction of recursion to iteration: *Every recursive partial function of* **A** *is iterative in an* **expansion of an extension of A**

(defined by an implementation of the recursive program which computes f)

- **Fact** (Patterson-Hewitt 1970, Stolboushkin-Taitslin 1983, Tiuryn 1989): *There are* (nice, total) *structures* **A** *in which some total relation* $f : A^n \to \{tt, ff\}$ *is recursive but not iterative* (Tiuryn's is a forest)

- *The distinction between interaction and recursion is not trivial and foundationally significant* (Recent work by Neil Jones, Siddharth Bhaskar, ...)

Counting calls to primitives for recursive programs

Fix a Φ-structure $\mathbf{A}$ and a recursive program

$$E :\equiv E_0 \text{ where } \Big\{ \mathrm{p}_1(\vec{\mathrm{u}}_1) = E_1, \ldots, \mathrm{p}_K(\vec{\mathrm{u}}_K) = E_K \Big\}$$

of $\mathbf{A}$ which computes $f_E : A^n \rightharpoonup A_s \quad (s \in \{\mathtt{ind}, \mathtt{boole}\})$

- For each $\Phi_0 \subseteq \Phi = \{\varphi_1, \ldots, \varphi_m\}$, there is a function

$\boxed{c_E(\Phi_0) : \{\vec{x} \in A^n \mid f_E(\vec{x})\downarrow\} \to \mathbb{N}}$ such that (intuitively)

$(*) \quad c_E(\Phi_0)(\vec{x}) =$ the number of calls to $\varphi_i^{\mathbf{A}}$ (with $\varphi_i \in \Phi_0$) that
$\underline{\text{E makes to compute } f(\vec{x})} \quad (f(\vec{x})\downarrow)$

- $\boxed{c_E(\Phi_0)(\vec{x}) \text{ is determined by the least-fixed-point definition of } f_E(\vec{x})}$

- **Fact**. *If E is* (the recursive program expressing) *a* `while` *program in* $\mathbf{A}$, *then* (**??**) *is a theorem*

- **Fact**. *If E^* is a* (standard) `while` *program implementing E* (in an expansion of an extension of $\mathbf{A}$), *then* $c_E(\Phi_0)(\vec{x}) = c_{E^*}(\Phi_0)(\vec{x})$ $(\vec{x} \in A^n)$

Counting all calls for recursive programs

Fix a recursive program E of $\mathbf{A}$ which computes $f_E : A^n \rightharpoonup A_s$

$c_E(\vec{x}) = c_E(\Phi)(\vec{x}) =$ the number of calls to all the primitives that E makes to compute $f(\vec{x})$ $\quad (f(\vec{x})\downarrow)$

- Logical calls: $\underline{\mathrm{p}_i(E_1, \dots, E_n)}$ $\underline{\text{if } E_0 \text{ then } E_1 \text{ else } E_2}$: (roughly, add 1)
- There is a function $\boxed{l_E : \{\vec{x} \in A^n \mid f_E(\vec{x})\downarrow\} \to \mathbb{N}}$ such that

  $(**)$ $\quad l_E(\vec{x}) =$ the number of all calls (to the primitives or logical) that E makes to compute $f(\vec{x})$ $\quad (f(\vec{x})\downarrow)$

- $\boxed{l_E(\vec{x}) \text{ is defined directly from the least-fixed-point definition of } f_E(\vec{x})}$
- It counts the (logical) time required by E to compute $f_E(\vec{x})$

► **Fact**. *If E is a* `while` *program, then* (with the usual definition of time for `while` programs)

$$l_E(\vec{x}) = \Theta(\mathrm{Time}_E(\vec{x})) \quad (f_E(\vec{x})\downarrow)$$

Tserunyan's First Theorem

Fix a recursive program E of $\mathbf{A}$ which computes $f_E : A^n \rightharpoonup A_s$

$$c_E(\vec{x}) = \text{the number of calls to the primitives that } E \text{ makes to compute } f_E(\vec{x}) \quad (f_E(\vec{x})\downarrow)$$

$$l_E(\vec{x}) = \text{the number of all calls that } E \text{ makes to compute } f_E(\vec{x}) \quad (f_E(\vec{x})\downarrow)$$

so clearly $\boxed{c_E(\vec{x}) \leq l_E(\vec{x}) \quad (f_E(\vec{x})\downarrow)}$

- **Theorem** (Anush Tserunyan, in her 2013 Ph.D. Thesis). *There is a constant* $K = K_{E,\mathbf{A}} \in \mathbb{N}$ *such that*

$$l_E(\vec{x}) \leq K(c_E(\vec{x}) + 1) \quad (f_E(\vec{x})\downarrow)$$

- It provides an explanation of why all the proofs of lower bounds for queries on structures (that I know) count needed calls to the primitives and derive a lower bound for time
- $\boxed{\text{The complexity functions } c_E(\vec{x}), l_E(\vec{x}) \text{ are defined on recursive programs}}$, not on implementations

Non-deterministic recursion

- A nondeterministic (nd) recursive program of a structure $\mathbf{A}$ is just like a (deterministic) program

$$E :\equiv E_0 \text{ where } \Big\{ \mathrm{p}_1(\vec{\mathrm{u}}_1) = E_1, \ldots, \mathrm{p}_K(\vec{\mathrm{u}}_K) = E_K \Big\}$$

  except that we allow $\mathrm{p}_i \equiv \mathrm{p}_j$ for some i, j

- **Pratt's** `nuclid` **program** of the Euclidean structure $(\mathbb{N}, \mathrm{rem}, \mathrm{eq}_0)$:

$$E_P \equiv \mathrm{nuclid}(a, b, a, b) \text{ where } \Big\{$$

$$\begin{aligned} \mathrm{nuclid}(a, b, m, n) = \ & \text{if } (n \neq 0) \text{ then } \mathrm{nuclid}(a, b, n, \mathrm{rem}(\mathrm{choose}(a, b, m), n \\ & \text{else if } (\mathrm{rem}(a, m) \neq 0) \text{ then } \mathrm{nuclid}(a, b, m, \mathrm{rem}(a, m \\ & \text{else if } (\mathrm{rem}(b, m) \neq 0) \text{ then } \mathrm{nuclid}(a, b, m, \mathrm{rem}(b, m \\ & \text{else } m, \end{aligned}$$

$$\mathrm{choose}(a, b, m) = m, \quad \mathrm{choose}(a, b, m) = a, \quad \mathrm{choose}(a, b, m) = b$$

- Fixed point semantics and the complexity functions $c_E(\vec{x}), l_E(\vec{x})$ can be extended to nd programs (with some work)

► `nuclid` *computes* $\mathrm{gcd}(x, y)$

A lower bound for coprimeness

- Def. Difficult pairs. A pair of numbers (x, y) is difficult if $2 \leq y < x < 2y$, $x \perp\!\!\!\perp y$ and (some technical condition)
  - Every pair (F_{k+1}, F_k) of successive Fibonaccis with $k \geq 3$ is difficult; every solution (x, y) of Pell's equation $x^2 = 1 + 2y^2$ is difficult; ...
  - **Theorem** (Lou van den Dries, ynm, 2004) *If E is a nd recursive program on* $(\mathbb{N}, 0, 1, =, <, +, \dot{-}, \mathrm{iq}, \mathrm{rem})$ *which computes* $\gcd(x, y)$ *:*

    for every difficult pair (x, y), $\boxed{c_E(\mathrm{rem})(x, y) \geq \frac{1}{10} \log \log x}$

  - A precise version of the **Main Conjecture**: *For every* (deterministic) *recursive program E of* $(\mathbb{N}, \mathrm{eq}_0, \mathrm{rem})$ *which computes* $\gcd(x, y)$ *when $x, y \geq 1$, there is a number $\delta > 0$, such that*

    for infinitely many pairs (x, y) with $x > y \geq 1$, $\boxed{c_E(\mathrm{rem})(x, y) \geq \delta \log x}$

- The theorem gives a nondeterministic complexity inequality which is one log below the claim of the Main Conjecture—too weak!

The calls complexity of Pratt's nuclid

- **Corollary**(vdd,ynm). *For every nd recursive program E on* $(\mathbb{N}, 0, 1, =, <, +, \dot{-}, \mathrm{iq}, \mathrm{rem})$ *which computes* $\gcd(x, y)$,

$$\boxed{c_E(\mathrm{rem})(F_{k+1}, F_k) \geq \frac{1}{10} \log\log F_{k+1}} \quad (k \geq 2)$$

- **Pratt's Theorem** (2008) *If E_P is Pratt's nd recursive* nuclid *program of the Euclidean structure* $(\mathbb{N}, \mathrm{rem}, \mathrm{eq}_0)$ *which computes* $\gcd(x, y)$, *then*

$$\boxed{c_{E_P}(\mathrm{rem})(F_{k+1}, F_k) \leq r \log\log F_{k+1}} \quad (\text{some } r, \text{ all } k \geq 3)$$

- So: vdd and ynm (2004, 2009) have the best version of their Theorem — but the main conjecture could be true with another infinite set of pairs; **it is still open**
- It may be that **the Main conjecture is true but only for deterministic programs**

Comments

- The most interesting foundational aspects of this work are that

  (1) the partial function computed by a recursive program and

  (2) its complexity measures

  are defined directly from the program (by fixed point recursion) rather than through its implementation.

  (1) simplifies greatly proofs of correctness, which come down to showing that the function which we want our algorithm to compute (together with some auxiliary functions) satisfy a system and is often trivial; and

  (2) insures that lower bounds for algorithms proved for these complexities hold for all (correct) implementations