On defining "algorithm" — why and how?

Yiannis N. Moschovakis
UCLA and University of Athens (MPLA)

14th CLMPS of the DLMPS of the IUHPS, Nancy, 20 June, 2011

Outline

(I) Features of algorithms:
dependence on primitives, uniformity, effectiveness

(II) Recursion and computation

(III) Exploiting uniformity:
The axiomatic derivation of absolute lower bounds

Basic reference *On founding the theory of algorithms*, 1998

Together with other, newer articles and references to others' articles in their bibliographies, on my homepage
www.math.ucla.edu$\sim$ynm

The Church-Turing Thesis (CT) for $\mathbb{N} = \{0, 1, \ldots\}$

- (CT): If $f : \mathbb{N}^n \to \mathbb{N}$ is computable (by some algorithm), then f is also computed by a Turing machine
- Does not define (in fact avoids defining) algorithms
- Main application: undecidability results
  To show that a relation cannot be decided (by any algorithm), we prove that its characteristic function cannot be computed by a Turing machine
- Key methods of proof: diagonalization and reduction (of some known undecidable problem to the problem we want to prove undecidable)
- The precise definition of algorithms is almost certainly irrelevant to the development of undecidability theory on $\mathbb{N}$.
  (It may be useful for extending the theory to other domains)

The Euclidean algorithm

For $x, y \in \mathbb{N},\ x \geq y \geq 1$,

$$(\varepsilon) \quad \boxed{\mathrm{gcd}(x, y) = \text{if } (\mathrm{rem}(x, y) = 0) \text{ then } y \text{ else } \mathrm{gcd}(y, \mathrm{rem}(x, y))}$$

where $\mathrm{rem}(x, y)$ is the remainder of the division of x by y,

$$x = \mathrm{iq}(x, y)y + \mathrm{rem}(x, y) \quad (0 \leq \mathrm{iq}(x, y),\ 0 \leq \mathrm{rem}(x, y) < y)$$

$$\begin{aligned} \mathrm{calls}_\varepsilon(x, y) = &\text{ the number of divisions (calls to rem)} \\ &\quad \text{required to compute } \mathrm{gcd}(x, y) \text{ by the Euclidean algorithm} \\ \leq &\ 2\log(y) \qquad (x \geq y \geq 2) \end{aligned}$$

- Is the Euclidean optimal for computing $\mathrm{gcd}(x, y)$ from rem?
- Is the Euclidean optimal for deciding coprimeness from rem?

$$x \perp\!\!\!\perp y \iff \mathrm{gcd}(x, y) = 1$$

We need a notion of **algorithm from rem** to answer these questions

Dependence of algorithms on primitives

- Algorithms are not absolute but from (relative to) specified primitives

- For simplicity, I will consider here only finitary algorithms: they compute a partial function $f : A^n \rightharpoonup A$ on a given set A, from given partial functions (of any arity) on A

- For convenience, we also assume (harmlessly): A contains two distinct elements $0 =$falsity and $1 =$truth; so each relation $R \subseteq A^N$ is identified with its characteristic function $\chi_R : A^n \to \{0, 1\} \subseteq A$

- $\boxed{\text{Algorithms of a partial algebra: } \mathbf{A} = (A, 0, 1, \phi_1^{\mathbf{A}}, \ldots, \phi_k^{\mathbf{A}})}$

A weak, relative lower bound for coprimeness

Theorem (van den Dries, ynm, 2004, 2009)

If a recursive algorithm α decides the coprimeness relation $x \perp\!\!\!\perp y$ on $\mathbb{N}$ from the primitives $\leq, +, -, \mathrm{iq}, \mathrm{rem}$, then for infinitely many x, y with $x > y$,

$$\mathrm{calls}_\alpha(x,y) \geq \mathrm{depth}_\alpha(x,y) > \frac{1}{10}\log\log x \qquad (*)$$

where $\mathrm{depth}_\alpha(x,y)$ is the least number of applications of the primitives which must be executed in sequence in the computation

- $\mathrm{depth}_\alpha(x,y)$ is a natural parallel time complexity measure
- The result is one log short of establishing the optimality of the Euclidean (and one log $= \infty$ in this context)
- (*) holds for all sufficiently large x, y such that
  - $x^2 = 1 + 2y^2$ (solutions of Pell's equation),
  - or $x = F_{n+1}, y = F_n$ (successive Fibonacci numbers)
- Claim: This applies to all algorithms from the specified primitives

Uniformity of algorithms

- To establish that

$$\mathrm{depth}_\alpha(x,y) > \frac{1}{10}\log\log x$$

for the specified x, y, we appeal to the fact that

$\boxed{\textit{the same algorithm } \alpha \textit{ decides whether } \lambda x \perp\!\!\!\perp \lambda y, \textit{ where } \lambda = 1 + x!}$

- An $\boxed{\text{algorithm of } \mathbf{A} = (A, 0, 1, \phi_1^{\mathbf{A}}, \ldots, \phi_k^{\mathbf{A}})}$ must compute

$f(\vec{x})$ for every $\vec{x} \in A^n$ such that $f(\vec{x})$ is defined (converges)

General features of finitary algorithms

Suppose α is a finitary algorithm which computes some $f : A^n \rightharpoonup A$:

- Dependence on primitives: α computes f from specified (partial) functions $\psi_1, \ldots, \psi_k$ on A
- Uniformity: α computes $f(\vec{x})$ for every $\vec{x}$ in the domain of convergence of f
- Effectiveness: α is effective
- Effectiveness is the most difficult feature of algorithms to make precise in a general (abstract, logical, not ad hoc) way
- Surprizingly: Sometimes effectiveness does not come into the derivation of interesting lower bounds for algorithms

Iterative algorithms (mechanical procedures)

vs. recursive algorithms

The mergesort (recursive) algorithm

L is a set , $L^* = \{w = (w_0, \dots, w_{n-1}) \mid w_i \in L\}$, $|w| = n \geq 0$

$x * w = (x, w_0, \dots, w_{n-1})$, $\mathrm{head}(w) = w_0$, $\mathrm{tail}(w) = (w_1, \dots, w_{n-1})$,

$\mathrm{half}_1(w) = (w_0, \dots, w_{\mathrm{iq}(n,2) \dot{-} 1})$, $\mathrm{half}_2(w) = (w_{\mathrm{iq}(n,2)}, \dots, w_{n-1})$

$\leq$ a (total) ordering of L, w is sorted $\iff w_0 \leq w_1 \leq \cdots \leq w_{n-1}$,

$\mathrm{sort}(w) =$ the unique, sorted w' such that

for some $\pi : \{0, \dots, n-1\} \rightarrowtail\!\!\!\!\twoheadrightarrow \{0, \dots, n-1\}$, $w'_i = w_{\pi(i)}$

$$\mathrm{sort}(w) = \mathrm{merge}(\mathrm{sort}(\mathrm{half}_1(w)), \mathrm{sort}(\mathrm{half}_2(w)))$$

$$
\begin{aligned}
\mathrm{merge}(u, v) = {} & \text{if } (|u| = 0) \text{ then } v \\
& \text{else if } (|v| = 0) \text{ then } u \\
& \text{else if } (\boxed{\mathrm{head}(u) \leq \mathrm{head}(v)}) \text{ then } \mathrm{head}(u) * \mathrm{merge}(\mathrm{tail}(u), v) \\
& \text{else } \mathrm{head}(v) * \mathrm{merge}(u, \mathrm{tail}(v))
\end{aligned}
$$

The mergesort computes $\mathrm{sort}(w)$ *using* $\leq |w|(\log |w|)$*comparisons*

The (first order) Formal Language of Recursion FLR(Φ)

$$\Phi = \{\phi_1, \ldots, \phi_k\} \quad \text{(partial function constants)}$$

Variables : $v_0, v_1, \ldots \quad p_0^m, p_1^m, \ldots$ (for m-ary partial functions)

$$\begin{aligned}\text{Terms (programs)} : t :\equiv\ & 0 \mid 1 \mid v_i \mid \phi_i(t_1, \ldots, t_{k_i}) \mid p_i^m(t_1, \ldots, t_m) \\ & \mid \text{if } (t_0 = 0) \text{ then } t_1 \text{ else } t_2 \\ & \mid t_0 \text{ where } \{p_1(\vec{u}_1) = t_1, \ldots, p_n(\vec{u}_n) = t_n\}\end{aligned}$$

(In the recursion construct $p_1, \vec{u}_1, p_2, \vec{u}_2, \ldots, p_n, \vec{u}_n$ are bound in t)

- FLR(Φ) is interpreted on (partial) Φ-algebras
$$\mathbf{A} = (A, 0, 1, \Phi^{\mathbf{A}}) = (A, 0, 1, \phi^{\mathbf{A}}, \ldots \phi_k^{\mathbf{A}})$$
- Denotational semantics: the recursion construct is interpreted by the taking of least fixed points

$$\boxed{\text{REC}(\mathbf{A}) = \text{the recursive partial functionals of } \mathbf{A}}$$

the partial functionals on A defined by terms of FLR(Φ)

(For $\mathbf{N} = (\mathbb{N}, 0, 1, S, \text{Pd})$, essentially McCarthy 1963)

The intensional semantics of FLR(Φ)

- A reduction calculus $s \Rightarrow t$ is defined on the terms of FLR(Φ). It models partial program compilation (not computation)
- It is shown (easily) that every term t reduces (compiles) to exactly one (up to congruence) irreducible, recursive term

$$t \Rightarrow \mathrm{cf}(t) \equiv t_0 \text{ where } \{p_1(\vec{u}_1) = t_1, \dots, p_n(\vec{u}_n) = t_n\}$$

$\boxed{\mathrm{cf}(t) \text{ is the canonical form of } t}$

- If $\vec{x}$ contains all the free variables of $t(\vec{x})$ and $\mathbf{A}$ is a Φ-algebra, we set

$$f_i^{\mathbf{A}}(\vec{x}, \vec{u}_i, p_1, \dots, p_n) = \mathrm{den}(t_i(\vec{x}))(\mathbf{A}, \vec{u}_i, p_1, \dots, p_n) \quad (i = 0, \dots, n)$$

$$\boxed{\mathrm{int}^{\mathbf{A}}(t(\vec{x})) = (f_0^{\mathbf{A}}, \dots, f_n^{\mathbf{A}})} = \text{the referential intension of } t(\vec{x}) \text{ in } \mathbf{A}$$

- Claim: $\mathrm{int}^{\mathbf{A}}(t(\vec{x}))$ models faithfully the recursive algorithm expressed by $t(\vec{x})$ in $\mathbf{A}$

$\boxed{\text{Key assumption: mutual recursion is a primitive algorithmic construct}}$

Summary

- Every algorithm α of an algebra $\mathbf{A} = (A, 0, 1, \Phi^{\mathbf{A}})$ which computes $\overline{\alpha} : A^n \rightharpoonup A$ is faithfully represented by the referential intension $\mathrm{int}^{\mathbf{A}}(t(\vec{x}))$ of some FLR(Φ) program $t(\vec{x})$
- $\mathrm{int}^{\mathbf{A}}(t(\vec{x}))$ is a recursor, a tuple of functionals on A satisfying certain conditions. (It is a semantic object)
- Two algorithms are equal if their representing recursors are naturally isomorphic
(the same tuples of functionals except for "rearrangement")
- Suppose $\mathbf{A} = (0, 1, \phi^{\mathbf{A}}, \psi^{\mathbf{A}})$ and for all $x \in A$, $\phi^{\mathbf{A}}(x) = \psi^{\mathbf{A}}(x)$; then
$$\mathrm{int}^{\mathbf{A}}(\phi(x)) = \mathrm{int}^{\mathbf{A}}(\psi(x))$$

Recursive vs. iterative algorithms

- Iterative algorithms can be viewed as special cases of recursive algorithms (tail recursions)
- Many familiar complexity measures can be defined directly for FLR(Φ) programs so that implementation independent upper bounds for them can be easily established (e.g., the mergesort)
- Sample (classical) lower bound result: *every recursive algorithm which computes* $\mathrm{sort}(w)$ *for every* $w \in L^*$ *from the primitives of the mergesort executes at least* $\log(|w|!)$ *comparisons*

★ The theory extends naturally to infinitary algorithms:

- Algorithms which interact with their environment, drop bombs, ...
- Recursion in higher types (Kleene). Adds algorithmic content:
  - Gentzen's infinitary cut elimination algorithm for arithmetic
  - Modeling Frege's sense of a term t in Montague semantics by the referential intension of t (which computes its denotation)

Model theory of arbitrary structures $\Rightarrow$ finite model theory

vs. direct and independent development of finite model theory

The Recursive Computability Thesis

- RCT: *If a partial function $f : A^n \rightharpoonup A$ is recursively computable from $\psi_1, \ldots, \psi_k$, then* $f \in \mathrm{REC}(A, 0, 1, \psi_1, \ldots, \psi_k)$
  - RCT basically accepts calling (composition) and branching as fundamental algorithmic constructs, and claims that the primary (algorithmic) interpretation of self-referential definitions is by the taking of least fixed points (grounded recursion)
  - The definition of $\mathrm{REC}(\mathbf{A})$ does not involve any objects outside $\mathbf{A}$
  - $\mathbf{A}$-recursion is a logical notion, preserved by isomorphisms

Theorem (RCT). $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is $\boxed{\text{recursively computable}}$

if and only if f is $\boxed{\text{Turing computable}}$ (No primitives mentioned)

Proof. If f is recursive on the natural numbers, then
$f \in \mathrm{REC}(\mathbb{N}, 0, 1, S, =)$ because (up to isomorphism)

$$\underline{\text{the natural numbers are the algebra } (\mathbb{N}, 0, 1, S, =)}$$

and hence f is Turing computable, by classical results $\dashv$

$$\boxed{\text{Computability on } \mathbb{N} = \text{recursive computability} + \text{what the numbers are}}$$

Uniform processes (skipping definitions of underlined terms)

An n-ary uniform process α of an algebra $\mathbf{A} = (A, 0, 1, \Phi^{\mathbf{A}})$ is a mapping

$$\mathbf{A} \supseteq_p \mathbf{U} \mapsto \overline{\alpha}^{\mathbf{U}} : U^n \rightharpoonup U$$

on <u>subalgebras</u> of **A** to partial functions on their universes such that:

(1) Embedding property: If $\pi : \mathbf{U} \rightarrowtail \mathbf{V}$ is an <u>embedding</u> of one subalgebra of **A** into another, then

$$\overline{\alpha}^{\mathbf{U}}(\vec{x}) = w \implies \overline{\alpha}^{\mathbf{V}}(\pi(\vec{x})) = \pi(w)$$

(2) Finiteness property: If $\overline{\alpha}^{\mathbf{A}}(\vec{x}) = w$, then there exists a finite $\mathbf{U} \subseteq_p \mathbf{A}$, <u>generated by</u> $\{0, 1, \vec{x}\}$ such that $\overline{\alpha}^{\mathbf{U}}(\vec{x}) = w$

A uniform process α of **A** computes the partial function $\overline{\alpha}^{\mathbf{A}} : A^n \rightharpoonup A$

- Every recursive algorithm of **A** induces a uniform process which computes the same function and with the same <u>complexity measures</u>. Plausibly: every "algorithm" of **A**, too
- Every $f : \mathbb{N} \to \mathbb{N}$ is computed by a uniform process of $(\mathbb{N}, 0, 1, S)$

Some lower bounds depend only on uniformity

Theorem (van den Dries, ynm, 2004, 2009)

If a uniform process α *of* $(\mathbb{N}, 0, 1, \leq, +, -, \mathsf{iq}, \mathsf{rem})$, *decides the coprimeness relation* $x \perp\!\!\!\perp y$ *on* $\mathbb{N}$, *then for infinitely many* x, y *with* $x > y$ *(e.g., if* $x^2 = 1 + 2y^2$ *or* $x = F_{n+1}, y = F_n$, x, y *large)*

$$\mathsf{calls}_\alpha(x, y) \geq \mathsf{depth}_\alpha(x, y) > \frac{1}{10} \log \log x$$

So, in particular, this holds for all recursive programs and all other, familiar models of relative computation (deterministic or non-deterministic) which decide coprimeness from the primitives of $(\mathbb{N}, 0, 1, \leq, +, -, \mathsf{iq}, \mathsf{rem})$

Poly evaluation and 0-testing in $\mathbf{C} = (\mathbb{C}, 0, 1, +, -, \cdot, \div, =)$

For $a_0, \ldots, a_n, x \in \mathbb{C}$ and $n \geq 1$:

$$\begin{aligned} \mathrm{Value}(a_0, a_0, \ldots, a_n, x) = V(\vec{a}, x) \quad &= \quad a_0 + a_1 x + \cdots + a_n x^n, \\ \mathrm{Nullity}(a_0, a_0, \ldots, a_n, x) \iff N(\vec{a}, x) \; &\iff \; a_0 + a_1 x + \cdots + a_n x^n = 0 \end{aligned}$$

- Horner's Rule: $V(\vec{a}, x)$ can be computed by a straight line program using n $(\cdot)$ and n $(+)$
- (Pan 1966, Winograd 1967): Every straight line program which computes $V(\vec{a}, x)$ for all $\vec{a}, x \in \mathbb{C}$ executes at least n $(\cdot/\div)$ and n $(+/-)$ when $\vec{a}, x$ are algebraically independent
- Every uniform process of $\mathbf{C}$ which decides $N(\vec{a}, x)$ for all $\vec{a}, x \in \mathbb{C}$ executes at least n $(\cdot/\div)$ when $\vec{a}, x$ are algebraically independent

(Bürgisser, Lickteig 1991 for straight line programs with conds)