Recursion and Complexity

Yiannis N. Moschovakis

Department of Mathematics
University of California, Los Angeles, CA 90095-1555, USA
and Department of Mathematics
Graduate Program in Logic, Algorithms and Computation ($M\Pi\Lambda A$)
University of Athens, Athens, Greece
ynm@math.ucla.edu

My purpose in this lecture is to explain how the representation of algorithms by recursive programs can be used in complexity theory, especially in the derivation of lower bounds for worst-case time complexity, which apply to *all*—or, at least, a very large class of—algorithms. It may be argued that recursive programs are not a new computational paradigm, since their manifestation as Herbrand-Gödel-Kleene systems was present at the very beginning of the modern theory of computability, in 1934. But they have been dissed as tools for complexity analysis, and part of my mission here is to rehabilitate them.

I will draw my examples primarily from van den Dries' [1] and the joint work in [3, 2], incidentally providing some publicity for the fine results in those papers. Some of these results are stated in Section 3; before that, I will set the stage in Sections 1 and 2, and in the last Section 4 of this abstract I will outline very briefly some conclusions about recursion and complexity which I believe that they support.

1 Partial Algebras

A (pointed) *partial algebra* is a structure of the form

$$\boldsymbol{A} = (A, 0, 1, \boldsymbol{\Phi}) = (A, 0, 1, \{\phi^{\boldsymbol{A}}\}_{\phi \in \Phi}), \tag{1}$$

where 0, 1 are distinct points in the universe A, and for every $\phi \in \Phi$,

$$\phi^{\boldsymbol{A}} : A^n \rightharpoonup A$$

is a partial function of some arity n associated by the *signature* Φ with the symbol ϕ. Typical example is the structure of arithmetic

$$\boldsymbol{N} = (\mathbb{N}, 0, 1, =, +, \cdot),$$

which happens to be *total*, i.e., the symbols '=', '+' and '$\cdot$' are interpreted by total functions, the characteristic function of the identity in the first case, and addition

[⋆] This is an outline of a projected lecture, in which I will refer extensively to and draw conclusions from results already published in [2]; and to make it as self-contained as possible, it has been necessary to quote extensively from [2], including the verbatim repetition of some of the basic definitions.

and multiplication for the other two. Genuinely partial algebras typically arise as *restrictions* of total algebras, often to finite sets: if $\{0,1\} \subseteq B \subseteq A$, then

$$\boldsymbol{A} \restriction B = (B, 0, 1, \{\phi^{\boldsymbol{A}} \restriction B\}_{\phi \in \Phi}),$$

where, for any $f : A^n \rightharpoonup A$,

$$f \restriction B(x_1, \ldots, x_n) = w \iff x_1, \ldots, x_n, w \in B \;\&\; f(x_1, \ldots, x_n) = w.$$

An *imbedding* $\iota : \boldsymbol{B} \rightarrowtail \boldsymbol{A}$ from one partial algebra into another (of the same signature) is any injective (total) map

$$\iota : B \rightarrowtail A,$$

such that $\iota(0) = 0$, $\iota(1) = 1$, and for all $\phi \in \Phi$, $\boldsymbol{x} = (x_1, \ldots, x_n), w$ in B,

$$\text{if } \phi^{\boldsymbol{B}}(\boldsymbol{x}) = w, \text{ then } \phi^{\boldsymbol{A}}(\iota(\boldsymbol{x})) = \iota(w),$$

where, of course, $\iota(x_1, \ldots, x_n) = (\iota(x_1), \ldots, \iota(x_n))$. If the identity $\iota(x) = x$ is an imbedding of $\boldsymbol{B}$ into $\boldsymbol{A}$, we call $\boldsymbol{B}$ a *partial subalgebra* of $\boldsymbol{A}$ and write $\boldsymbol{B} \subseteq_p \boldsymbol{A}$. Notice that the definitions here are in the spirit of graph theory, not model theory, i.e., we do not insist that partial subalgebras be closed under the given operations; in particular, for any $B \subseteq A$,

$$(\{0,1\}, --) \subseteq_p \boldsymbol{A} \restriction B \subseteq_p \boldsymbol{A},$$

where $(\{0,1,\}, --))$ is the trivial algebra with universe $\{0,1\}$ and all symbols in Φ interpreted by completely undefined partial functions.

For any $X \subseteq A$ and any number m, we define the set $G_m(X)$ *generated in* $\boldsymbol{A}$ *from* X *in* m *steps* in the obvious way:

$$\begin{aligned} G_0(X) &= \{0,1\} \cup X, \\ G_{m+1}(X) &= G_m(X) \cup \{\phi^{\boldsymbol{A}}(\boldsymbol{x}) \mid \boldsymbol{x} \in G_m(X), \phi \in \Phi, \phi^{\boldsymbol{A}}(\boldsymbol{x})\downarrow\}. \end{aligned} \tag{2}$$

Notice that if X is finite, then each $G_m(X)$ is a finite set. The set generated by X is the union

$$G(X) = \bigcup\nolimits_{m \in \mathbb{N}} G_m(X),$$

and it determines the partial subalgebra $\boldsymbol{A} \restriction G(X)$ of $\boldsymbol{A}$ generated by X.

The Complexity of Values

Suppose now that $\boldsymbol{A}$ is a partial algebra as in (1) and

$$\chi : A^n \to A$$

is an n-ary function on A which we want to compute *from the givens* $\{\phi^{\boldsymbol{A}}\}_{\phi \in \Phi}$ of $\boldsymbol{A}$—*and nothing else*. It is natural to suppose that this cannot be done unless each value $\chi(\boldsymbol{x})$ can be generated from the arguments $\boldsymbol{x}$ by successively applying the givens, i.e.,

$$\chi(\boldsymbol{x}) \in G(\boldsymbol{x}) \quad (\boldsymbol{x} \in A^n);$$

and that if this holds and we set

$$g_\chi(\boldsymbol{x}) = \text{the least } m \text{ such that } \chi(\boldsymbol{x}) \in G_m(\boldsymbol{x}),$$

then any algorithm which computes $\chi(\boldsymbol{x})$ will need at least $g_\chi(\boldsymbol{x})$ steps. This can be argued very generally, and it can be used to derive hard lower bounds for all algorithms which compute $\chi(\boldsymbol{x})$ from specific givens. Van den Dries used it in [1] to derive a triple logarithmic lower bound for the function

$$\gcd(x, y) = \text{the greatest common divisor of } x \text{ and } y \quad (x, y \in \mathbb{N}, x \geq y \geq 1)$$

from addition, subtraction and division with remainder, i.e., the two functions

$$\mathrm{iq}(x, y) = q, \quad \mathrm{rem}(x, y) = r,$$

where q and r are the unique natural numbers such that

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

His proof introduced some ingenious ideas from number theory (which we will mention further down), and the result was at least a first step in an effort to establish that the classical Euclidean algorithm is optimal; the Euclidean, of course, has single logarithmic complexity, and it uses only the remainder function $\mathrm{rem}(x, y)$.

The big advantage of the complexity function $g_\psi(\boldsymbol{x})$ is that lower bound results about it apply to all algorithms, whatever algorithms are. On the other hand, it cannot be used to establish lower bounds for decision problems, where the function that we want to compute takes on only the values 0 or 1: for that we need to make some assumptions about algorithms, which we do next.

2 Recursive Programs

The *terms* of the language $L(\Phi)$ of programs in the signature Φ are defined by the recursion

$$E :\equiv 0 \mid 1 \mid \mathsf{v}_i \mid \phi(E_1, \ldots, E_n) \mid \mathsf{p}_i^n(E_1, \ldots, E_n) \mid \text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2,$$

where v_i is one of a list of *individual variables*; p_i^n is one of a list of n-ary (partial) *function variables*; and ϕ is any n-ary symbol in Φ. These terms are interpreted as usually in any Φ-partial algebra $\boldsymbol{A}$ and relative to any assignment π which assigns some $\pi(\mathsf{v}_i) \in A$ to each individual variable v_i, and some n-ary partial function $\pi(\mathsf{p}_i^n) : A^n \rightharpoonup A$ to each function variable p_i^n:

$$[\![E]\!](\pi) = [\![E]\!](\boldsymbol{A}, \pi) = \text{the value (if defined) of } E \text{ in } \boldsymbol{A} \text{ for the assignment } \pi;$$

and if the variables which occur in E are in the list $(\mathsf{x}_1, \dots, \mathsf{x}_n, \mathsf{p}_1, \dots, \mathsf{p}_m)$, then E defines a (partial) *functional*

$$F_E(\boldsymbol{x}, \boldsymbol{p}) = [\![E]\!](\{(\mathsf{x}_1, \dots, \mathsf{x}_n, \mathsf{p}_1, \dots, \mathsf{p}_m) := \boldsymbol{x}, \boldsymbol{p}\}) \tag{3}$$

which is monotone and continuous.

A *recursive* (or McCarthy) *program* of $L(\Phi)$ is any system of *recursive term equations*

$$\alpha : \qquad \begin{cases} p_\alpha(\boldsymbol{x}) = E_0 \\ p_1(\boldsymbol{x}_1) = E_1 \\ \quad\vdots \\ p_K(\boldsymbol{x}_K) = E_K \end{cases} \tag{4}$$

such that $p_\alpha, p_1, \dots, p_K$ are distinct function variables; $p_1, \dots, p_K$ are the only function variables which occur in $E_0, \dots, E_K$; and for each i, the free, individual variables in each E_i are in the list $\boldsymbol{x}_i$. The term E_0 is the *head* of α, the remaining terms are its *body*, and we may describe the mutual recursive definition expressed by α by the simple notation

$$\alpha \equiv E_0 \text{ where } \{p_1 = E_1, \dots, p_K = E_K\}.$$

The function variables $p_1, \dots, p_K$ are bound in this expression.

To interpret a program α on a Φ-structure $\boldsymbol{A}$, we observe that its parts define the system of mutually recursive equations

$$\begin{aligned} p_\alpha(\boldsymbol{x}) &= F_{E_0}(\boldsymbol{x}, p_1, \dots, p_K), \\ p_1(\boldsymbol{x}_1) &= F_{E_1}(\boldsymbol{x}_1, p_1, \dots, p_K), \\ &\vdots \\ p_K(\boldsymbol{x}) &= F_{E_K}(\boldsymbol{x}_K, p_1, \dots, p_K), \end{aligned}$$

using (3), which by the usual methods has a set of least, mutual solutions

$$\overline{p}_\alpha, \overline{p}_1, \dots, \overline{p}_K;$$

we then let

$$[\![\alpha]\!] = [\![\alpha]\!](\boldsymbol{A}) = \overline{p}_\alpha : A^n \rightharpoonup A,$$

so that the partial function "computed" by α on $\boldsymbol{A}$ is the component of the tuple of least solutions of the mutual recursion determined by α which corresponds to the head term.

A partial function $f : A^n \rightharpoonup A$ is $\boldsymbol{A}$*-recursive* if it is computed by some recursive program.

Except for the notation, these are the programs introduced by John McCarthy in [6]. McCarthy proved that if $\boldsymbol{N}_0 = (\mathbb{N}, 0, 1, S, P)$ is the simplest structure on the natural numbers with just the successor and the predecessor operations as given, then the $\boldsymbol{N}_0$-recursive partial functions are exactly the Turing-computable ones. To justify the connection with computability in general, one must of course explain how recursive programs compute partial functions, in effect to construct an *implementation* of $L(\Phi)$ on an arbitrary partial Φ-algebra $\boldsymbol{A}$; but it is well-known how to do this (in many ways), and we will not be concerned with it.

We note two basic lemmas, whose proofs are very easy:

Lemma 1 (Imbedding). *If $\iota : \boldsymbol{B} \to \boldsymbol{A}$ is an imbedding from one Φ-algebra into another, then for every Φ-program α and all $\boldsymbol{x}, w \in B$,*

$$\textit{if } [\![\alpha]\!](\boldsymbol{B}, \boldsymbol{x}) = w, \quad \textit{then } [\![\alpha]\!](\boldsymbol{A}, \iota(\boldsymbol{x})) = \iota(w).$$

In particular, if $\boldsymbol{B} \subseteq_p \boldsymbol{A}$, then for every Φ-program α and all $\boldsymbol{x}, w \in B$,

$$\textit{if } [\![\alpha]\!](\boldsymbol{B}, \boldsymbol{x}) = w, \quad \textit{then } [\![\alpha]\!](\boldsymbol{A}, \boldsymbol{x}) = w.$$

Lemma 2 (Finiteness). *For every Φ-algebra $\boldsymbol{A}$, every recursive Φ-program α and all $\boldsymbol{x}, w \in A$, if $[\![\alpha]\!](\boldsymbol{A}, \boldsymbol{x}) = w$, then there exists some m such that*

$$w \in G_m(\boldsymbol{x}) \quad \textit{and} \quad [\![\alpha]\!](\boldsymbol{A} \restriction G_m(\boldsymbol{x}), \boldsymbol{x}) = w.$$

The Finiteness Lemma expresses the simple proposition that computations are finite, and so they live in some finite subset $G_m(\boldsymbol{x})$ of A generated by the input; but it leads directly to the next, fundamental notion of complexity.

The Basic (Structural) Complexity

For each recursive program α, each partial algebra $\boldsymbol{A}$, and each $\boldsymbol{x}$ such that $[\![\alpha]\!](\boldsymbol{x})\downarrow$, we set

$$C_\alpha(\boldsymbol{x}) = \text{the least } m \text{ such that } [\![\alpha]\!](\boldsymbol{A} \restriction G_m(\boldsymbol{x}), \boldsymbol{x}) = [\![\alpha]\!](\boldsymbol{x}).$$

Roughly speaking, the basic complexity $C_\alpha(\boldsymbol{x})$ measures the minimum number of nested calls to the givens which is required for the computation of $[\![\alpha]\!](\boldsymbol{x})$. It cannot be realistically attained by any actual implementation of α, but it is a plausible lower bound for the time complexity of any implementation.

The Finiteness Lemma now yields immediately the key tool for deriving lower bound results about the basic complexity of recursive programs:

Lemma 3 (The Imbedding Test). *Let $\boldsymbol{A}$ be a partial algebra as in* (1)*, suppose that $\chi : A^n \to A$, and assume that for some $\boldsymbol{x} \in A^n$ and some m, there is an imbedding*

$$\iota : \boldsymbol{A} \restriction G_m(\boldsymbol{x}) \to \boldsymbol{A}$$

such that

$$\chi(\iota(\boldsymbol{x})) \neq \iota(\chi(\boldsymbol{x}));$$

it follows that for every recursive program α which computes χ in $\boldsymbol{A}$,

$$C_\alpha(\boldsymbol{x}) > m.$$

3 Two Lower Bound Results About Coprimeness

We quote here from [2] two lower bound results about the relation of *coprimeness*

$$x \perp y \iff x, y \geq 1 \;\&\; (\forall d > 1)[d \nmid x \vee d \nmid y],$$

which are obtained using the Imbedding Test, Lemma 3.

For the first, let

$$\mathbf{Lin}_0 = \{=, <, \mathrm{parity}, 2\cdot, \frac{1}{2}\cdot, +, \dot{-}\} \tag{5}$$

where the relations stand for their characteristic functions and

$$2 \cdot (x) = 2x, \quad \frac{1}{2} \cdot (x) = \mathrm{iq}(x, 2).$$

Knuth [5] describes the *binary algorithm* algorithm of Stein which computes the gcd (and hence decides coprimeness) from $\mathbf{Lin}_0$ in logarithmic time.[1] The Stein algorithm is optimal among recursive algorithms (up to a multiplicative constant), because of the following:

Theorem 1 ([2]). *If a recursive program α decides the coprimeness relation in the algebra $\boldsymbol{A}_0 = (\mathbb{N}, \mathbf{Lin}_0)$, then for all $a > 2$,*

$$C_\alpha(a, a^2 - 1) \geq \frac{1}{10} \log(a^2 - 1). \tag{6}$$

The proof goes by showing that if $m < \frac{1}{10}\log(a^2 - 1)$, then there is an imbedding

$$\iota : \boldsymbol{A}_0 \restriction G_m(a, a^2 - 1) \to \boldsymbol{A}$$

such that for some λ,

$$\iota(a) = \lambda a, \quad \iota(a^2 - 1) = \lambda(a^2 - 1),$$

so that ι carries the coprime pair $(a, a^2 - 1)$ to a pair of numbers which are not coprime, and hence $C_\alpha(a, a^2 - 1) > m$ by the Imbedding Test. It is not possible to describe in this abstract how this imbedding is defined, but it is quite simple. Not so for the next result—the Main Theorem of [2]—where the same idea is used, but the relevant imbedding is much harder to define and depends on Liouville's Theorem on "good approximations" of algebraic irrationals, cf. [4]:

Theorem 2 ([2]). *If a recursive program α decides the coprimeness relation in the algebra $\boldsymbol{A}_1 = (\mathbb{N}, \mathbf{Lin}_0, \mathrm{iq}, \mathrm{rem})$, then for infinitely many coprime pairs $a > b > 1$,*

$$C_\alpha(a, b) \geq \frac{1}{10} \log \log a. \tag{7}$$

[1] This is also specified in [2].

In fact, (7) *holds for all coprime* $a > b > 1$ *such that*

$$\left|\frac{a}{b} - \sqrt{2}\right| < \frac{1}{b^2} \tag{8}$$

(and there are infinitely many such a, b by a classical result).

The target here is the Euclidean algorithm which decides coprimeness in logarithmic time, and so the theorem is one log short of what is needed to establish the (plausible) optimality of the Euclidean. But it is as good as any known lower bound for coprimeness from its givens, and perhaps unique in its uniformity—it yields the same lower bound, on the same inputs for all recursive programs.

4 Inessential (Logical) Extensions of Partial Algebras

The next natural question is whether Theorems 1 and 2 also hold for other "computational paradigms", for example random access machines. Of course they do, and by more-or-less the same proofs, adapted to the idiosyncracies of each model which do not affect the basic, arithmetical facts that enter into the arguments. Rather than do this one computation model at a time, however, we look for a general result which might suggest that these lower bounds hold *for all algorithms.*

Let $\boldsymbol{A} = (A, 0, 1, \{\phi^{\boldsymbol{A}}\}_{\phi\in\Phi})$ be a partial algebra. An *inessential extension* of $\boldsymbol{A}$ is any partial algebra

$$\boldsymbol{B} = (B, 0, 1, \{\phi^{\boldsymbol{A}}\}_{\phi\in\Phi}, \{\psi^{\boldsymbol{B}}\}_{\psi\in\Psi})$$

with the following properties:

(IE1) $A \subseteq B$, and $\boldsymbol{A}$ and $\boldsymbol{B}$ have the same 0 and 1;

(IE2) every permutation π of A fixing 0 and 1 can be extended to a permutation π^B of B such that for every "new given" $\psi = \psi^{\boldsymbol{B}}$ of arity n and all $x_1, \ldots, x_n \in B$,

$$\pi^B\psi(x_1, \ldots, x_n) = \psi(\pi^B x_1, \ldots, \pi^B x_n). \tag{9}$$

Here we view each "old given" $\phi^{\boldsymbol{A}}$ as a partial function on B, undefined when one of its arguments is not in A.

We might also call these extensions *logical*, since the property we demand of the new givens in $\boldsymbol{B}$ is reminiscent (or better: a relativization to the given algebra $\boldsymbol{A}$) of Tarski's *logical functions.* In any case, Lemma 3 extends directly (and easily) to them:

Lemma 4 (The Extended Imbedding Test, [2]). *Let* $\boldsymbol{A}$ *be a partial algebra as in* (1)*, suppose that* $\chi : A^n \to A$*, and assume that for some* $\boldsymbol{x} \in A^n$ *and some* m*, there is an imbedding*

$$\iota : \boldsymbol{A}\restriction G_m(\boldsymbol{x}) \to \boldsymbol{A}$$

such that

$$\chi(\iota(\boldsymbol{x})) \neq \iota(\chi(\boldsymbol{x}));$$

it follows that for every recursive program α which computes χ in some inessential extension $\boldsymbol{B}$ of $\boldsymbol{A}$,

$$C_\alpha(\boldsymbol{x}) > m.$$

And, of course, random access machines relative to any set $\boldsymbol{\Phi}$ of functions on $\mathbb{N}$ can be faithfully represented by recursive programs on inessential extensions of $(\mathbb{N}, 0, 1, \boldsymbol{\Phi})$, as are all computational models *relative to* $\boldsymbol{\Phi}$; and so Theorems 1 and 2 also hold for them.

In fact, the familiar computational paradigms for computation on $\mathbb{N}$ accept some fixed, number-theoretic functions $\boldsymbol{\Phi}$ as givens (the successor and predecessor, equality, addition, etc.), and they also assume some "computational constructs" which are characteristic of them and independent of $\boldsymbol{\Phi}$, e.g., branching, recursion, reading from and writing to registers or stacks, higher-type logical operations like λ-abstraction and β-conversion, etc. In [7, 8] it is argued that all algorithms from given operations can be faithfully represented by suitable recursive programs on the structure determined by the givens: now the results we have discussed here suggest the following, more concrete interpretation of that proposal for algorithms from first-order givens on some set A:

Refined Church-Turing Thesis for Algorithms, ([2]). *Every algorithm α from a set $\boldsymbol{\Phi}$ of partial functions and relations on a set A can be represented faithfully by a recursive program β on some inessential extension $\boldsymbol{B}$ of the partial algebra $\boldsymbol{A} = (A, 0, 1, \boldsymbol{\Phi})$.*

If we assume this Thesis, then the Extended Imbedding Test makes it possible to establish lower bounds for all algorithms from a set of first-order givens $\boldsymbol{\Phi}$ on A, whenever we can produce the appropriate imbeddings.

References

1. Lou van den Dries. Generating the greatest common divisor, and limitations of primitive recursive algorithms. *Foundations of computational mathematics*, 3:297–324, 2003.
2. Lou van den Dries and Yiannis N. Moschovakis. Is the Euclidean algorithm optimal among its peers? *The Bulletin of Symbolic Logic*, 10:390–418, 2004.
3. Lou van den Dries and Yiannis N. Moschovakis. Arithmetic complexity. 200? in preparation.
4. G. H. Hardy and E. M. Wright. *An introduction to the theory of numbers.* Clarendon Press, Oxford, fifth edition (2000). originally published in 1938.
5. D. E. Knuth. *The Art of Computer Programming. Fundamental Algorithms.* Addison-Wesley, second edition, 1973.
6. J. McCarthy. A basis for a mathematical theory of computation. In P. Braffort and D Herschberg, editors, *Computer programming and formal systems*, pages 33–70. North-Holland, 1963.
7. Yiannis N. Moschovakis. On founding the theory of algorithms. In H. G. Dales and G. Oliveri, editors, *Truth in mathematics*, pages 71–104. Clarendon Press, Oxford, 1998.
8. Yiannis N. Moschovakis. What is an algorithm? In B. Engquist and W. Schmid, editors, *Mathematics unlimited – 2001 and beyond*, pages 919–936. Springer, 2001.