THE BULLETIN OF SYMBOLIC LOGIC
Volume 10, Number 3, Sept. 2004

IS THE EUCLIDEAN ALGORITHM OPTIMAL AMONG ITS PEERS?

LOU VAN DEN DRIES AND YIANNIS N. MOSCHOVAKIS

The Euclidean algorithm on the natural numbers $\mathbb{N} = \{0, 1, \dots\}$ can be specified succinctly by the *recursive program*

$$\varepsilon : \quad \gcd(a,b) = \begin{cases} b, & \text{if } \operatorname{rem}(a,b) = 0, \\ \gcd(b, \operatorname{rem}(a,b)), & \text{otherwise} \end{cases} \qquad (a \geq b \geq 1),$$

where $\operatorname{rem}(a,b)$ is the remainder in the division of a by b, the unique natural number r such that for some natural number q,

$$a = bq + r \qquad (0 \leq r < b). \tag{1}$$

It is an algorithm *from* (*relative to*) the remainder function rem, meaning that in computing its *time complexity function* $c_\varepsilon(a,b)$, we assume that the values $\operatorname{rem}(x,y)$ are provided on demand by some "oracle" in one "time unit". It is easy to prove that

$$c_\varepsilon(a,b) \leq 3 \log_2 a \quad (a > b > 1).$$

Much more is known about $c_\varepsilon(a,b)$, but this simple-to-prove upper bound suggests the proper formulation of the Euclidean's (worst case) optimality among its *peers*—algorithms from rem:

CONJECTURE. *If an algorithm α computes* $\gcd(x,y)$ *from* rem *with time complexity $c_\alpha(x,y)$, then there is a rational number $r > 0$ such that for infinitely many pairs $a > b > 1$, $c_\alpha(a,b) > r \log_2 a$.*

Our main aim here is to prove the following relevant result:

THEOREM A. *If a recursive program α decides the coprimeness relation $x \perp y$ from* $=, <, +, \dot{-}$, iq *and* rem, *then for infinitely many coprime pairs $a > b > 1$,*

$$c_\alpha(a,b) > \frac{1}{10} \log_2 \log_2 a. \tag{2}$$

Received December 22, 2003; revised April 22, 2004.

Van den Dries acknowledges support from NSF grant DMS 01-00979. Moschovakis acknowledges support from the Graduate Program in Algorithms and Computation (ΜΠΛΑ) and University of Athens Grant 70/4/5633.

In fact, (2) *holds for all coprime* $a > b > 1$ *such that*

$$\left|\frac{a}{b} - \sqrt{2}\right| < \frac{1}{b^2} \tag{3}$$

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

We use standard notations: throughout a, b, m, n are natural numbers, $a \dot{-} b = a - b$ if $a \geq b$ and 0 otherwise, $a \mid b \iff a$ divides b,

$$a \perp b \iff \gcd(a, b) = 1 \quad (a, b \geq 1), \tag{4}$$

and $\mathrm{iq}(a, b)$ is the *integer quotient* of a by b, the unique q in (1) if $b > 0$; we make iq and rem totally defined on $\mathbb{N}^2$ by setting $\mathrm{iq}(a, 0) = \mathrm{rem}(a, 0) = a$. An algorithm decides a relation $R(\vec{x})$ if it computes its characteristic function

$$\chi_R(\vec{x}) = \begin{cases} 1, & \text{if } R(\vec{x}), \\ 0, & \text{otherwise.} \end{cases}$$

Our result is (in one direction) stronger than the Conjecture (for recursive programs), since it allows more given functions and it takes just one step to decide $a \perp b$ from $\gcd(a, b)$ by (4), but it gives a lower bound one log below what we would like to prove, and in this business one log is infinitely far away.

We also prove a few additional lower bounds, including the following, which establishes the optimality of the *Stein algorithm* for the gcd among its peers. (The Stein algorithm is specified in Section 4.)

THEOREM B. *If a recursive program* α *decides coprimeness from* $=, <, +, \dot{-}, 2 \cdot x, \mathrm{iq}(x, 2)$, *and* $\mathrm{rem}(x, 2)$, *then for all* $a > 2$,

$$c_\alpha(a, a^2 - 1) > \frac{1}{10} \log_2(a^2 - 1). \tag{5}$$

There are few lower bound results for relative number-theoretic algorithms in the literature, and for coprimeness we can point only to [6]. The authors of [6], however, allow multiplication among the givens, and so they can only establish (for the RAM and decision tree models) a lower bound that is "one log lower" than Theorem A. It may be that the lower bound claim in Theorem A can be extracted from their proof, but this is not immediate. In any case, we believe that the main value of our results lies in the number-theoretic specification of "difficult inputs", on which *all recursive programs* that compute a certain function *from fixed givens* exhibit *the same poor performance*—up to the constants involved, the 1/10 in Theorems A and B; this suggests a connection between computational and Diophantine complexity that may be worth investigating. Notice also that Theorem A may be viewed as a general version (for arbitrary recursive programs from rem) of Lamé's Theorem, which identifies the pairs of successive Fibonacci numbers as inputs where the Euclidean algorithm exhibits its worst performance.

The work for this communication started with [9], on the related but more special topic of *relative primitive recursive algorithms*. Van den Dries [1] solved a basic problem left open in [9] by a method that includes the key idea of considering inputs which are "good approximations" of irrationals as in (3). We have attempted to keep this communication as elementary as possible, and we have included in the first two sections enough background material so that it can serve as a broadly accessible entry into *Arithmetic Complexity*. The forthcoming [2] contains many stronger results about uniform and non-uniform algorithms from various given functions on $\mathbb{N}$ (including multiplication), and it provides a more comprehensive introduction to this fascinating subject.

Organization of the paper. After specifying our notational (and other) conventions in Section 1, we introduce *recursive programs* and establish their basic properties in Section 2. These programs are well-known—they go back to [7], after all—but they have not been used much in complexity studies; the lemmas in Sections 2 and 6 suggest that recursive programs are a most useful tool for establishing *worst-case lower bounds*[1] that apply to all (call-by-value) computational models. The main result of Section 3 is Theorem 6, a logarithmic lower bound for recursive programs deciding primality from the "givens" of Theorem B. In Section 4 we prove Theorem B. The arguments in these two sections, while very simple, introduce most of the ideas and some of the preliminary facts used in the more difficult proof of Theorem A in Section 5. In Section 6 we extend our results to Random Access Machines by a method which suggests that these lower bounds hold for all algorithms from the relevant givens, and we formulate a *Refined Church-Turing Thesis* which expresses this proposal in a concrete, mathematical form.[2]

§1. Preliminaries and notation.

For any two sets A, W, an n-ary partial function $f : A^n \rightharpoonup W$ is a function $f : D_f \to W$ defined on some subset of A^n. For such f we use notation as follows:

$$\begin{aligned}
f(\vec{x})\downarrow &\iff \vec{x} \in D_f \quad (f(\vec{x}) \text{ converges}),\\
f(\vec{x})\uparrow &\iff \vec{x} \notin D_f \quad (f(\vec{x}) \text{ diverges}),\\
f(\vec{x}) = g(\vec{x}) &\iff [f(\vec{x})\downarrow \,\&\, g(\vec{x})\downarrow \,\&\, f(\vec{x}) = g(\vec{x})] \text{ or } [f(\vec{x})\uparrow \,\&\, g(\vec{x})\uparrow],\\
f \sqsubseteq g &\iff (\forall \vec{x})[f(\vec{x})\downarrow \implies f(\vec{x}) = g(\vec{x})],
\end{aligned}$$

and on occasion (in definitions) we use the ungrammatical "$f(\vec{x}) = \uparrow$" as synonymous with "$f(\vec{x})\uparrow$". Partial functions compose *strictly*:

[1]It is quite possible that recursive programs are also useful in the study of *average case complexity*, but we do not now have any relevant results.

[2]Suitable versions of these results also hold for non-uniform algorithms, such as *arithmetic circuits*, but we leave those versions for [2].

$$\begin{aligned} &f(g_1(\vec{x}),\dots,g_n(\vec{x})) = w \\ &\qquad \iff (\exists w_1,\dots,w_n)[g_1(\vec{x}) = w_1 \,\&\cdots\&\, g_n(\vec{x}) = w_n \\ &\qquad\qquad\qquad\qquad\qquad\qquad \& \, f(w_1,\dots,w_n) = w]. \end{aligned}$$

Let $P(A^n, W)$ be the set of all $f : A^n \rightharpoonup W$, and consider *functionals*, partial functions

$$F : A^m \times P(A_1^{n_1}, W_1) \times \cdots \times P(A_k^{n_k}, W_k) \rightharpoonup W$$

which take tuples in some A and partial functions (on various sets) as arguments and give a value in some set W (in case of convergence). Such a functional F is *monotone*, if

$$\begin{aligned} &F(\vec{x}, f_1,\dots,f_k)\downarrow \,\&\, f_1 \sqsubseteq g_1 \,\&\cdots\&\, f_k \sqsubseteq g_k \\ &\qquad\qquad\qquad \Longrightarrow F(\vec{x}, f_1,\dots,f_k) = F(\vec{x}, g_1,\dots,g_k), \end{aligned}$$

and it is *continuous* if

$$\begin{aligned} &F(\vec{x}, f_1,\dots,f_k)\downarrow \\ &\Longrightarrow (\exists \text{ finite } f_1^0 \sqsubseteq f_1,\dots,f_k^0 \sqsubseteq f_k)[F(\vec{x}, f_1^0,\dots,f_k^0) = F(\vec{x}, f_1,\dots,f_k)], \end{aligned}$$

where a partial function is *finite* if it has finite domain of convergence.

Mutual recursion. The properties of recursive programs can all be deduced from the following, well-known result:

THE FIXED POINT LEMMA. *For every monotone and continuous functional*

$$F : A^n \times P(A^n, W) \rightharpoonup W,$$

the recursive equation

$$f(\vec{x}) = F(\vec{x}, f)$$

has a $\sqsubseteq$-least solution $\overline{f} : A^n \rightharpoonup W$, characterized by the conditions

$$(\forall \vec{x})[\overline{f}(\vec{x}) = F(\vec{x}, \overline{f})];$$
$$\text{if } (\forall \vec{x})[g(\vec{x}) = F(\vec{x}, g)], \text{ then } \overline{f} \sqsubseteq g.$$

Similarly, every system of mutual monotone and continuous recursive equations

$$\begin{aligned} f_1(\vec{x}_1) &= F_1(\vec{x}_1, f_1,\dots,f_K) \\ &\vdots \\ f_K(\vec{x}_K) &= F_K(\vec{x}_K, f_1,\dots,f_K) \end{aligned}$$

(with domains and ranges matching so that the equations make sense) *has a $\sqsubseteq$-least solution tuple $\overline{f}_1,\dots,\overline{f}_K$.*

The one-equation case is proved by defining recursively

$$\overline{f}^0(\vec{x}) = \uparrow,$$
$$\overline{f}^{n+1}(\vec{x}) = F(\vec{x}, \overline{f}^n),$$

and setting $\overline{f} = \sup\{\overline{f}^n \mid n = 0, 1, \dots\}$, i.e.,

$$\overline{f}(\vec{x}) = w \iff (\exists n)[\overline{f}^n(\vec{x}) = w].$$

The general case is similar.

This is all we need from recursion theory.

Partial algebras. To apply the basic model-theoretic notions to the theory of computation, we must introduce two small wrinkles, as follows.

First, we need to allow partial functions and relations in structures, since computations often diverge: a (pointed) *partial algebra* is a structure of the form

$$\boldsymbol{A} = (A, 0, 1, \{\phi^{\boldsymbol{A}}\}_{\phi \in \Phi}), \tag{6}$$

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 of a partial algebra 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 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

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

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

$$f \upharpoonright B(x_1, \dots, x_n) = w \iff x_1, \dots, x_n, w \in B \,\&\, f(x_1, \dots, x_n) = w.$$

Conditionals. The second wrinkle on the classical notions is that we must allow *conditionals* among the terms, which are now defined from the symbols in the signature Φ and variables $\mathsf{v}_0, \mathsf{v}_1, \dots$ by the recursion

$$E :\equiv \mathsf{0} \mid \mathsf{1} \mid \mathsf{v}_i \mid \phi(E_1, \dots, E_n) \mid (\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2). \tag{Φ}$$

Written out in full, this says that the Φ-terms comprise the smallest set of strings of symbols which contains $\mathsf{0}$, $\mathsf{1}$ and all the variables, and is closed under the formation rules

$$E_1, \dots, E_n \mapsto \phi(E_1, \dots, E_n),$$
$$E_0, E_1, E_2 \mapsto (\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2).$$

For the semantics, we introduce for each set A the (closed) $\Phi[A]$-*terms* by replacing the variables by arbitrary members of A in the definition of terms, i.e., with $x \in A$,

$$(\Phi[A]) \qquad E :\equiv 0 \mid 1 \mid x \mid \phi(E_1, \dots, E_n) \mid (\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2).$$

The members of A which occur in a $\Phi[A]$-term E are its *parameters*.

If we use the familiar convention of naming Φ-terms by $E(\mathsf{x}_1, \dots, \mathsf{x}_n)$ to indicate the variables which (may) occur in them and $x_1, \dots, x_n \in A$, then $E(x_1, \dots, x_n)$ is the $\Phi[A]$-term resulting by replacing each x_i by x_i.

Semantics. The partial function $E \mapsto \text{den}(E) : \{\Phi[A]\text{-terms}\} \rightharpoonup A$ is defined by the following recursive clauses:

$$\text{den}(0) = 0, \quad \text{den}(1) = 1, \quad \text{den}(x) = x$$

$$\text{den}(\phi(E_1, \dots, E_n)) = \phi^{\boldsymbol{A}}(\text{den}(E_1), \dots, \text{den}(E_n))$$

$$\text{den}(\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2) = \begin{cases} \text{den}(E_1), & \text{if den}(E_0) = 0, \\ \text{den}(E_2), & \text{if den}(E_0)\downarrow \ \& \, \text{den}(E_0) \neq 0. \end{cases}$$

We call $\text{den}(E)$ the *denotation* of the $\Phi[A]$-term E (if $\text{den}(E)\downarrow$). When we need to exhibit the algebra in which the denotation is computed, we write $\text{den}(\boldsymbol{A}, E)$, or we use model-theoretic notation,

$$\boldsymbol{A} \models E = w \iff \text{den}(\boldsymbol{A}, E) = w.$$

Partiality introduces some complications which deserve notice. For example, if we view subtraction as a partial function on $\mathbb{N}$, then for all $x, y, z \in \mathbb{N}$,

$$(\mathbb{N}, 0, 1, +, -) \models (x + y) - y = x;$$

but if $x < y$, then

$$(\mathbb{N}, 0, 1, +, -) \not\models (x - y) + y = x$$

because $(x - y)\uparrow$ – and then, by the strictness of composition, $(x - y) + y\uparrow$ also. On the other hand,

$$\text{den}(E_0) = 0 \implies \text{den}(\text{if } (E_0 = 0) \text{ then } E_1 \text{ else } E_2) = \text{den}(E_1),$$

whether $\text{den}(E_2)$ converges or not.

Partial subalgebras and imbeddings. A (partial) *subalgebra* of $\boldsymbol{A}$ is any partial algebra $\boldsymbol{B} = (B, 0, 1, \{\phi^{\boldsymbol{B}}\}_{\phi \in \Phi})$, such that $B \subseteq A$ and for every $\phi \in \Phi$,

$$\phi^{\boldsymbol{B}}(\vec{x}) = w \implies \phi^{\boldsymbol{A}}(\vec{x}) = w.$$

We indicate this by $\boldsymbol{B} \subseteq_p \boldsymbol{A}$. If $\{0, 1\} \subseteq B \subseteq A$, then clearly

$$\boldsymbol{A} \restriction B \subseteq_p \boldsymbol{A};$$

but also $\boldsymbol{B}_0 \subseteq_p \boldsymbol{A}$, if $\boldsymbol{B}_0$ is the partial algebra on B in which all symbols of Φ are interpreted by totally undefined partial functions.

In the same way, a (partial algebra) *imbedding* $\imath : \boldsymbol{B} \rightarrowtail \boldsymbol{A}$ is an injective map $\imath : B \rightarrowtail A$ such that $\imath(0) = 0$, $\imath(1) = 1$, and for all $\phi \in \Phi$, $\vec{x}, w$ in B,

$$\phi^{\boldsymbol{B}}(\vec{x}) = w \implies \phi^{\boldsymbol{A}}(\imath(\vec{x})) = \imath(w),$$

where, of course, $\imath(x_1, \dots, x_n) = (\imath(x_1), \dots, \imath(x_n))$. This implies (by an easy induction) that for every Φ-term $E(\vec{x})$ and $x_1, \dots, x_n, w \in B$,

$$\boldsymbol{B} \models E(\vec{x}) = w \implies \boldsymbol{A} \models E(\imath(\vec{x})) = \imath(w), \tag{7}$$

with $\imath(\vec{x}) = (\imath(x_1), \dots, \imath(x_n))$.

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

$$\begin{aligned} G_0(X) &= \{0, 1\} \cup X, \\ G_{m+1}(X) &= G_m(X) \cup \{\phi^{\boldsymbol{A}}(\vec{x}) \mid \vec{x} \in G_m(X), \phi \in \Phi, \phi^{\boldsymbol{A}}(\vec{x})\downarrow\}. \end{aligned} \tag{8}$$

Clearly

$$\boldsymbol{A} \restriction G_0(X) \subseteq_p \boldsymbol{A} \restriction G_1(X) \subseteq_p \cdots,$$
$$X \subseteq Y \implies \boldsymbol{A} \restriction G_m(X) \subseteq_p \boldsymbol{A} \restriction G_m(Y),$$

and by induction on k, for all m

$$G_k(G_m(X)) = G_{m+k}(X). \tag{9}$$

When we need to indicate the algebra in which the generation is taking place, we write $G_m(\boldsymbol{A}, X)$ instead of $G_m(X)$.

§2. Recursive (McCarthy) programs.

A (formal) *recursive program* on the signature Φ is a system of recursive term equations

$$\alpha : \begin{cases} \mathsf{f}_\alpha(\vec{\mathsf{x}}) = \mathsf{f}_0(\vec{\mathsf{x}}_0) = E_0(\vec{\mathsf{x}}_0, \mathsf{f}_1, \dots, \mathsf{f}_K), \\ \qquad\qquad \mathsf{f}_1(\vec{\mathsf{x}}_1) = E_1(\vec{\mathsf{x}}_1, \mathsf{f}_1, \dots, \mathsf{f}_K), \\ \qquad\qquad\qquad \vdots \\ \qquad\qquad \mathsf{f}_K(\vec{\mathsf{x}}_K) = E_K(\vec{\mathsf{x}}_K, \mathsf{f}_1, \dots, \mathsf{f}_K), \end{cases} \tag{10}$$

where each $E_i(\vec{\mathsf{x}}_i, \mathsf{f}_1, \dots, \mathsf{f}_K)$ is a term in the (extended) *program signature*

$$\mathrm{sig}(\alpha) = \Phi \cup \{\mathsf{f}_0, \mathsf{f}_1, \dots, \mathsf{f}_K\}.$$

As indicated, the individual variables in E_i are all from the list $\vec{\mathsf{x}}_i$ of (distinct) variables. The (distinct) new function symbols $\mathsf{f}_0, \dots, \mathsf{f}_K$ are the *recursion variables* of α, the terms E_i are its *parts*, and E_0 is its *head*. Notice that the head recursion variable f_0 does not occur in any of the parts of α. For $\mathsf{f} = \mathsf{f}_i$ we shall also write E_f for the part E_i.

We allow $K = 0$ in this definition, in which case the program is just an explicit definition assigning a name to the partial function defined by a Φ-term. On the other hand, algorithms are often expressed by a single recursive equation, e.g.,

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

for the Euclidean, and in this case we need to add a trivial head equation

$$\mathsf{f}_0(x, y) = \mathrm{gcd}(x, y)$$

to accord with the "official" definition; we will assume this is done when needed.

Semantics of recursive programs. Fix a partial algebra $\boldsymbol{A}$ and a recursive program α on the same signature Φ. For any term $E(\vec{\mathsf{x}}, \mathsf{f}_0, \dots, \mathsf{f}_K)$, tuple $\vec{x}$ from A of the same length as $\vec{\mathsf{x}}$, and partial functions $f_0, \dots, f_K$ on A of the right arities, set

$$\mathrm{den}(E(\vec{x}, f_0, \dots, f_K)) := \mathrm{den}(E(\vec{x}, \mathsf{f}_0, \dots, \mathsf{f}_K))$$

where the righthand side is computed in the sig(α)-partial algebra

$$(\boldsymbol{A}, f_0, \dots, f_K) = (A, 0, 1, \{\phi^{\boldsymbol{A}}\}_{\phi \in \Phi}, f_0, \dots, f_K).$$

It is easy to check by induction on E that such functionals

$$(\vec{x}, f_0, \dots, f_K) \mapsto \mathrm{den}(E(\vec{x}, f_0, \dots, f_K))$$

are monotone and continuous, and so the Fixed Point Lemma implies that the system of recursive equations

$$\begin{aligned} f_\alpha(\vec{x}_0) = f_0(\vec{x}_0) &= \mathrm{den}(E_0(\vec{x}_0, f_1, \dots, f_K)), \\ f_1(\vec{x}_1) &= \mathrm{den}(E_1(\vec{x}_1, f_1, \dots, f_K)), \\ &\vdots \\ f_K(\vec{x}_K) &= \mathrm{den}(E_K(\vec{x}_K, f_1, \dots, f_K)) \end{aligned}$$

defined by α has a least solution tuple

$$\overline{f}_0, \dots, \overline{f}_K.$$

The partial function *computed by* α in $\boldsymbol{A}$ is the one associated with the head part,

$$\overline{\alpha} = \overline{\alpha}^{\boldsymbol{A}} = \overline{f}_\alpha,$$

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

The classical example is the (total) algebra $(\mathbb{N}, 0, 1, S, \mathrm{Pd})$ of *unary arithmetic* with the successor and the predecessor functions $S(x) = x + 1$, $\mathrm{Pd}(0) = 0$, $\mathrm{Pd}(x + 1) = x$, whose recursive partial functions are exactly the Turing computable partial functions. This elegant characterization of Turing computability is due to [7], and so recursive programs are also called *McCarthy programs*.

The tree-depth and parallel complexity functions. Fix again a partial algebra A and a recursive program α on the same signature Φ, and for each (closed) $\mathrm{sig}(\alpha)[A]$-term

$$M \equiv M(\vec{x}, \mathsf{f}_0, \dots, \mathsf{f}_K),$$

let (to simplify notation)

$$\overline{M} = \mathrm{den}(M(\vec{x}, \overline{f}_0, \dots, \overline{f}_K)).$$

We will define two partial functions

$$D, C : \mathrm{sig}(\alpha)[A]\text{-terms} \rightharpoonup \mathbb{N}$$

such that

$$D(M)\downarrow \iff C(M)\downarrow \iff \overline{M}\downarrow . \tag{11}$$

When $\overline{M}\downarrow$, then $D(M)$ is the depth of the *computation tree* which is naturally associated with the computation of $\overline{M}$ using the equations of α, and $C(M)$ represents intuitively the maximal "depth" of nested calls to the givens in this computation. These partial functions satisfy the conditions (1) – (4) below, which taken together comprise recursive definitions of them; in other words, D and C are the least partial functions on $\mathrm{sig}(\alpha)[A]$-terms which satisfy (1) – (4). The definitions are justified by appealing to the Fixed Point Lemma.

(1) $D(0) = D(1) = D(x) = 0 \quad (x \in A)$,
$C(0) = C(1) = C(x) = 0 \quad (x \in A)$.

(2) If $\phi \in \Phi$ and $\overline{M}_1\downarrow, \dots, \overline{M}_n\downarrow, \phi(\overline{M}_1, \dots, \overline{M}_n)\downarrow$, then

$$D(\phi(M_1, \dots, M_n)) = \max\{D(M_1), \dots, D(M_n)\} + 1,$$
$$C(\phi(M_1, \dots, M_n)) = \max\{C(M_1), \dots, C(M_n)\} + 1.$$

(3) If f is a recursion variable of α, then

$$D(\mathsf{f}(M_1, \dots, M_n)) = \max\{D(M_1), \dots, D(M_n), d_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n\} + 1,$$
$$C(\mathsf{f}(M_1, \dots, M_n)) = \max\{C(M_1), \dots, C(M_n)\} + c_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n),$$

where

$$d_{\mathsf{f}}(\vec{x}) = D(E_{\mathsf{f}}(\vec{x}, \mathsf{f}_1, \dots, \mathsf{f}_K)),$$
$$c_{\mathsf{f}}(\vec{x}) = C(E_{\mathsf{f}}(\vec{x}, \mathsf{f}_1, \dots, \mathsf{f}_K)).$$

(4) For conditional terms:

$$D(\text{if } (N_0 = 0) \text{ then } N_1 \text{ else } N_2)$$
$$= \begin{cases} \max\{D(N_0), D(N_1)\} + 1, & \text{if } \overline{N}_0 = 0, \\ \max\{D(N_0), D(N_2)\} + 1, & \text{if } \overline{N}_0\downarrow \ \& \overline{N}_0 \neq 0. \end{cases}$$

$$C(\text{if } (N_0 = 0) \text{ then } N_1 \text{ else } N_2) = \begin{cases} \max\{C(N_0), C(N_1)\}, & \text{if } \overline{N}_0 = 0, \\ \max\{C(N_0), C(N_2)\}, & \text{if } \overline{N}_0\downarrow \ \& \ \overline{N}_0 \neq 0. \end{cases}$$

We have incidentally defined the complexity $c_{\mathsf{f}}(\vec{x})$ of each of the mutual least fixed points of α, and the strict complexity of α is that of its head symbol,

$$\text{(SC2)} \qquad c_\alpha(\vec{x}) = c_{\mathsf{f}_0}(\vec{x}) = C(E_0(\vec{x}, \mathsf{f}_1, \dots, \mathsf{f}_K)).$$

The basic notion here is the complexity function $c_\alpha(\vec{x})$, but the tree-depth complexity $D(M)$ is a useful tool for proofs. For example, for tuples $\vec{x} = (x_1, \dots, x_n) \in A^n$ and $\vec{y} = (y_1, \dots, y_n) \in A^n$, set

$$\vec{x} \backsim \vec{y} \iff x_i = y_i \text{ for all } i \text{ such that } x_i \in \{0, 1\} \text{ or } y_i \in \{0, 1\}.$$

By an easy induction on $D(M(\vec{x}))$, if $C(M(\vec{x})) = 0$, then

$$\vec{x} \backsim \vec{y} \implies \overline{M(\vec{y})}\downarrow \ \& \ \overline{M(\vec{x})} \backsim \overline{M(\vec{y})}.$$

It follows that if $\overline{\alpha}(\vec{x}) = 1$ but $\overline{\alpha}(\vec{y}) = 0$ for some $\vec{y} \backsim \vec{x}$, then

$$(12) \qquad c_\alpha(\vec{x}) \geq 1,$$

an inequality which simplifies the form of some lower bounds.

When we need to indicate the partial algebra in which the program is interpreted, we write $C(\boldsymbol{A}, M)$, $c_\alpha(\boldsymbol{A}, \vec{x})$ and use the following notations:

$$(13) \quad \boldsymbol{A}, \alpha \models M^{(m)} = w \iff \text{den}(\boldsymbol{A}, M) = w \ \& \ C(\boldsymbol{A}, M) \leq m,$$

$$\begin{aligned} (14) \quad \boldsymbol{A} \models \alpha^{(m)}(\vec{x}) = w &\iff \overline{\alpha}^{\boldsymbol{A}}(\vec{x}) = w \ \& \ c_\alpha(\boldsymbol{A}, \vec{x}) \leq m, \\ &\iff \boldsymbol{A}, \alpha \models E_0(\vec{x}, \mathsf{f}_1, \dots, \mathsf{f}_K)^{(m)} = w. \end{aligned}$$

We also set

$$\begin{aligned} \boldsymbol{A}, \alpha \models M^{(m)}\downarrow &\iff \text{for some } w, \boldsymbol{A}, \alpha \models M^{(m)} = w, \\ \boldsymbol{A} \models \alpha^{(m)}(\vec{x})\downarrow &\iff \text{for some } w, \boldsymbol{A} \models \alpha^{(m)}(\vec{x}) = w. \end{aligned}$$

Lemma 1 (The Imbedding Lemma). *If $\imath : \boldsymbol{B} \rightarrowtail \boldsymbol{A}$ is an imbedding and α is a recursive program, then for every term*

$$M(\vec{x}) \equiv M(\vec{x}, \mathsf{f}_0, \dots, \mathsf{f}_K)$$

in the signature of α, every m, and all $\vec{x}$ in B and $w \in B$,

$$\begin{aligned} \boldsymbol{B}, \alpha \models M(\vec{x})^{(m)} = w &\implies \boldsymbol{A}, \alpha \models M(\imath(\vec{x}))^{(m)} = \imath(w), \\ \boldsymbol{B}, \alpha \models M(\vec{x})^{(m)}\downarrow &\implies \boldsymbol{A}, \alpha \models M(\imath(\vec{x}))^{(m)}\downarrow . \end{aligned}$$

In particular, these implications hold for the identity imbedding, when $\boldsymbol{B} \subseteq_p \boldsymbol{A}$.

Proof. The second claim follows from the first, which is proved by induction on $D(M(\vec{x}))$ computed in $\boldsymbol{B}$, taking cases on the form of $M(\vec{x})$. For example, in the case of a recursive call

$$M(\vec{x}) \equiv \mathsf{f}(M_1(\vec{x}), \dots, M_n(\vec{x})),$$

suppose

$$\mathrm{den}(\boldsymbol{B}, M(\vec{x})) = w, \quad C(\boldsymbol{B}, M(\vec{x})) = m.$$

By the definition of complexity, there are k, l such that $k + l \leq m$ and suitable w_i such that

$$\begin{aligned} \mathrm{den}(\boldsymbol{B}, M_i(\vec{x})) = w_i, \quad C(\boldsymbol{B}, M_i(\vec{x})) \leq k \quad (i = 1, \dots, n), \\ \mathrm{den}(\boldsymbol{B}, E_{\mathsf{f}}(w_1, \dots, w_n)) = w, \quad C(\boldsymbol{B}, E_{\mathsf{f}}(w_1, \dots, w_n)) \leq l. \end{aligned}$$

Now all $D(M_i(\vec{x}))$ as well as $D(E_{\mathsf{f}}(w_1, \dots, w_n))$ are less than $D(M(\vec{x}))$, so the induction hypothesis supplies the same equations and inequalities with $\boldsymbol{A}$ in place of $\boldsymbol{B}$ and $\imath(x_i), \imath(w_j), \imath(w)$ in place of x_i, w_j, w. The definition of complexity applied to $\boldsymbol{A}$ yields now the required

$$\mathrm{den}(\boldsymbol{A}, M(\imath(\vec{x}))) = \imath(w), \quad C(\boldsymbol{A}, M(\imath(\vec{x}))) \leq m. \qquad \dashv$$

Next we show that the relation $\boldsymbol{A}, \alpha \models M^{(m)} = w$ is *absolute* for the generated partial subalgebras $\boldsymbol{A} \restriction G_m(\boldsymbol{A}, X)$.

Lemma 2 (The Absoluteness Lemma). *Let α be a recursive program on a partial algebra $\boldsymbol{A}$, and M a* $\mathrm{sig}(\alpha)[A]$*-term whose parameters are in $X \subseteq A$, such that $\boldsymbol{A}, \alpha \models M^{(m)} = w$. Then*

$$w \in G_m(\boldsymbol{A}, X) \text{ and } \boldsymbol{A} \restriction G_m(\boldsymbol{A}, X), \alpha \models M^{(m)} = w. \tag{15}$$

Proof is again by induction on $D(M)$, the result being trivial when $D(M) = 0$, so that M is 0, 1 or x.

If $M \equiv \phi(M_1, \dots, M_n)$ with a given ϕ, then the induction hypothesis guarantees that for every $i = 1, \dots, n$,

$$\overline{M}_i \in G_{m-1}(X), \text{ and } \boldsymbol{A} \restriction G_{m-1}(X), \alpha \models M_i^{(m-1)} = \overline{M}_i.$$

Hence $\overline{M} = \phi(\overline{M}_1, \dots, \overline{M}_n) \in G_m(X)$, and thus

$$\boldsymbol{A} \restriction G_m(X), \alpha \models M^{(m)} = \overline{M}.$$

Suppose next that $M \equiv \mathsf{f}(M_1, \dots, M_n)$, with a recursive variable f. Then by the induction hypothesis, there are k, l such that $k + l \leq m$, and for $i = 1, \dots, n$,

$$\overline{M}_i \in G_k(X), \ \boldsymbol{A} \restriction G_k(X), \alpha \models M_i^{(k)} = \overline{M}_i,$$

and also $E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n) \in G_l(\overline{M}_1, \dots, \overline{M}_n)$, and

$$\boldsymbol{A} \restriction G_l(\overline{M}_1, \dots, \overline{M}_n), \alpha \models E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)^{(l)} = \overline{M}.$$

Therefore

$$\overline{M} \in G_l(\overline{M}_1, \dots, \overline{M}_n) \subseteq G_l(G_k(X)) = G_{k+l}(X) \subseteq G_m(X).$$

It follows that

$$\begin{aligned} &\mathbf{A} \restriction G_m(X), \alpha \models M_i^{(k)} = \overline{M}_i, \quad i = 1, \dots, n, \\ &\qquad\qquad \mathbf{A} \restriction G_m(X), \alpha \models E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)^{(l)} = \overline{M}, \end{aligned}$$

and so $\mathbf{A} \restriction G_m(X), \alpha \models M^{(m)} = \overline{M}$.

The case for the conditional is simpler and we skip it. ⊣

A stronger absoluteness result for "computation space" is proved in [2], and used there to get lower bounds for *non-uniform algorithms*.

§3. Primality. We establish here a logarithmic lower bound for recursive programs which decide primality from

$$\text{(16)} \qquad \boldsymbol{Lin} = \{=, <, +, \dot{-}, 2\cdot, \tfrac{1}{2}\cdot, \text{Parity}\} \qquad \text{(operations on } \mathbb{N})$$

where $2 \cdot (x) = 2x$, $\frac{1}{2} \cdot (x) = \text{iq}(x, 2)$ and $\text{Parity}(x) = \text{rem}(x, 2)$. It is a simple result, but it sets the pattern for all that follows.

LEMMA 3 (Uniqueness). *If $x_i, y_i \in \mathbb{Z}$ and $|x_i|, |y_i| < \frac{a}{2}$, then*

$$\begin{aligned} x_0 + x_1 a = y_0 + y_1 a &\iff [x_0 = y_0 \,\&\, x_1 = y_1], \\ x_0 + x_1 a > y_0 + y_1 a &\iff [x_1 > y_1 \vee (x_1 = y_1 \,\&\, x_0 > y_0)]. \end{aligned}$$

PROOF. The second equivalence easily implies the first one, and follows itself from the next equivalence applied to $(x_0 - y_0) + (x_1 - y_1)a$.

Let $x, y \in \mathbb{Z}$ and $|x|, |y| < a$; then

$$x + ya > 0 \iff [y > 0] \vee [y = 0 \,\&\, x > 0].$$

To see this, suppose first that $x + ya > 0$. If $y < 0$, then $a \leq (-y)a < x$ which contradicts the hypothesis, so $y \geq 0$; and if $y = 0$, then $x = x + ya > 0$. For the converse, if $y > 0$, then $y \geq ya > |x|$, so $x + ya > 0$; and the second disjunct $y = 0 \,\&\, x > 0$ yields $x + ya = x > 0$. ⊣

For each tuple of natural numbers $\vec{a} = (a_1, \dots, a_n)$, let

$$\text{(17)} \quad B_m(\vec{a}) = \Big\{ \frac{x_0 + x_1 a_1 + \cdots + x_n a_n}{2^m} \in \mathbb{N} \\ \qquad\qquad \mid x_0, \dots, x_n \in \mathbb{Z} \text{ and } |x_i| \leq 2^{2m}, i \leq n \Big\},$$

where $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ is the set of integers. The members of $B_m(\vec{a})$ are *natural numbers*. In full detail:

$$\begin{aligned} x \in B_m(\vec{a}) \iff & x \in \mathbb{N} \text{ and there exist } x_0, \dots, x_n \in \mathbb{Z} \\ & \text{such that } x = \frac{x_0 + x_1 a_1 + \cdots + x_n a_n}{2^m}, \\ & \text{and for } i = 0, \dots, n, |x_i| \leq 2^{2m}, \end{aligned}$$

which, in particular, implies that $2^m \mid x_0 + x_1 a_1 + \cdots + x_n a_n \geq 0$. We let

$$(18) \qquad \boldsymbol{N_{Lin}} = (\mathbb{N}, 0, 1, \boldsymbol{Lin}) = (\mathbb{N}, 0, 1, =, <, +, \dot{-}, 2\cdot, \frac{1}{2}\cdot, \text{Parity}).$$

The generated sets $G_m(X)$ below are with respect to $\boldsymbol{N_{Lin}}$.

Lemma 4 (Inclusion). *For all $\vec{a} \in \mathbb{N}^n$ and all m:*

(1) $0, 1 \in B_m(\vec{a}) \subseteq B_{m+1}(\vec{a})$.

(2) *For every k-ary function f in **Lin**,*

$$x_1, \dots, x_k \in B_m(\vec{a}) \Longrightarrow f(x_1, \dots, x_k) \in B_{m+1}(\vec{a}).$$

(3) $G_m(\vec{a}) \subseteq B_m(\vec{a})$.

Proof. (1) Taking $\vec{a} = (a, b)$ for simplicity, observe first that

$$0 = \frac{0 + 0 \cdot a + 0 \cdot b}{2^0} \in B_0(a, b),$$

and similarly for 1. The second inclusion holds because

$$\frac{x_0 + x_1 a + x_2 b}{2^m} = \frac{2x_0 + 2x_1 a + 2x_2 b}{2^{m+1}}$$

and $|2x_i| \leq 2 \cdot 2^{2m} < 2^{2(m+1)}$.

(2) For addition, let $x, y \in B_m(a, b)$, so

$$\begin{aligned} x + y = \frac{x_0 + x_1 a + x_2 b}{2^m} &+ \frac{y_0 + y_1 a + y_2 b}{2^m} \\ &= \frac{2(x_0 + y_0) + 2(x_1 + y_1)a + 2(x_2 + y_2)b}{2^{m+1}} \end{aligned}$$

and the coefficients in the numerator satisfy

$$|2(x_i + y_i)| \leq 2(2^{2m} + 2^{2m}) = 2^{2(m+1)}.$$

The same works for subtraction. The claim about division by 2 is trivial, and for multiplication by 2, notice that

$$2\Big(\frac{x_0 + x_1 a + x_2 b}{2^m}\Big) = \frac{4x_0 + 4x_1 a + 4x_2 b}{2^{m+1}},$$

and $|4x_0| \leq 4 \cdot 2^{2m} = 2^{2(m+1)}$.

(3) follows immediately from (2), by induction on m. ⊣

LEMMA 5 (Imbedding). *Suppose* $2^{2m+2} \le a$, *and set* $\lambda = 1 + 2^m$. *Then we have a partial algebra imbedding* $\iota : \boldsymbol{N}_{\boldsymbol{Lin}} \upharpoonright G_m(a) \rightarrowtail \boldsymbol{N}_{\boldsymbol{Lin}}$ *given by*

$$\iota\Big(\frac{x_0 + x_1 a}{2^m}\Big) = \frac{x_0 + x_1 \lambda a}{2^m} \quad \Big(|x_0|, |x_1| \le 2^{2m}, \frac{x_0 + x_1 a}{2^m} \in G_m(a)\Big).$$

PROOF. Lemma 3 and part (3) of Lemma 4 imply that the equation above defines a map $\iota : G_m(a) \to \mathbb{Q}$, since

$$2^{2m} = \frac{1}{2}2^{2m+1} < \frac{a}{2}$$

by the hypothesis. This map takes values in $\mathbb{N}$, because

$$x_0 + \lambda x_1 a = x_0 + x_1 a + (\lambda - 1)x_1 a = x_0 + x_1 a + 2^m x_1 a,$$

so that if $2^m \mid (x_0 + x_1 a)$, then also $2^m \mid (x_0 + \lambda x_1 a)$. It is injective and order-preserving, by Lemma 3 again, applied to both a and λa.

To check that it preserves addition when the sum is in $G_m(a)$, suppose that $X, Y, X + Y \in G_m(a)$, and write

$$X = \frac{x_0 + x_1 a}{2^m}, \quad Y = \frac{y_0 + y_1 a}{2^m}, \quad X + Y = Z = \frac{z_0 + z_1 a}{2^m}$$

with all $|x_i|, |y_i|, |z_i| \le 2^{2m}$. Now

$$Z = \frac{(x_0 + y_0) + (x_1 + y_1)a}{2^m},$$

and $|x_0 + y_0|, |x_1 + y_1| \le 2 \cdot 2^{2m} = 2^{2m+1} < \frac{a}{2}$ by the hypothesis, and so by the Uniqueness Lemma 3,

$$x_0 + y_0 = z_0, \quad x_1 + y_1 = z_1,$$

which gives $\iota X + \iota Y = \iota Z$. The same argument works for subtraction, multiplication and division by 2, and parity. $\dashv$

THEOREM 6. *If a recursive program* α *decides primality from* ***Lin***, *then for all primes* p,

$$c_\alpha(p) > \frac{1}{4}\log_2 p.$$

PROOF. Assume the hypothesis, let $v = c_\alpha(p)$ where p is prime, and suppose towards a contradiction, that

$$\boldsymbol{N}_{\boldsymbol{Lin}} \models \alpha^{(v)}(p) = 1 \quad \text{with } 2^{2v+2} \le p. \tag{19}$$

(1) $\boldsymbol{N}_{\boldsymbol{Lin}} \upharpoonright G_v(p) \models \alpha^{(v)}(p) = 1$, by (15).
(2) $\boldsymbol{N}_{\boldsymbol{Lin}} \models \alpha^{(v)}(\lambda p) = 1$ with $\lambda = 1 + 2^v$, by Lemma 5.
(3) α decides primality, so λp is prime—which is absurd.

Thus (19) is false, and so $2^{2v+2} > p$, which gives $2v + 2 > \log_2 p$; since $v \ge 1$ by (12), we have the required

$$4v \ge 2v + 2 > \log_2 p. \qquad \dashv$$

The same argument (with judicious choices of λ) can be used to establish the same lower bounds for the complexity of recursive programs from ***Lin*** which decide the relations

$$x \text{ is a power of } 2, \quad x \text{ is a perfect square}, \quad \text{and } x \text{ is square-free}.$$

We leave these (and more general results for extensions of ***Lin*** and non-uniform algorithms) for [2].

§4. The optimality of the Stein algorithm. This modern algorithm computes $\gcd(x, y)$ from the givens in ***Lin***, defined in (16).

PROPOSITION 7 ([5], Vol. 2, Sect. 4.5.2). *The program σ with the single recursive equation*

$$\gcd(x,y) = \begin{cases} x & \text{if } x = y \\ 2\gcd(x/2, y/2) & \text{otherwise, if Parity}(x) = \text{Parity}(y) = 0, \\ \gcd(x/2, y) & \text{otherwise, if Parity}(x) = 0, \text{Parity}(y) = 1, \\ \gcd(x, y/2) & \text{otherwise, if Parity}(x) = 1, \text{Parity}(y) = 0, \\ \gcd(x \dot{-} y, y) & \text{otherwise, if } x > y, \\ \gcd(x, y \dot{-} x) & \text{otherwise.} \end{cases}$$

computes $\gcd(x, y)$ *for* $x, y > 0$ *with*

$$c_\sigma(x, y) \le C \cdot \log_2 \max(x, y)$$

for some positive constant C.

PROOF. That the gcd satisfies these equations and is determined by them is trivial. To check the complexity bound, notice that (at worst) every other application of one of the clauses involves halving one of the arguments—the worst case being subtraction, which, however must then be immediately followed by a division, since the difference of two odd numbers is even. ⊣

We prove here Theorem B, which establishes the optimality (up to a multiplicative constant) of Stein's algorithm among recursive programs from ***Lin***.

For the remainder of this section, x, y, z, x_i, y_i, z_i range over $\mathbb{Z}$.

LEMMA 8. *Let $a > 2$ and $b = a^2 - 1$. Then $a \perp b$, and if $|x_i|, |y_i| < \dfrac{a}{4}$ for $i = 0, 1, 2$, then*

$$x_0 + x_1 a + x_2 b = y_0 + y_1 a + y_2 b \iff x_0 = y_0 \,\&\, x_1 = y_1 \,\&\, x_2 = y_2,$$

$$x_0 + x_1 a + x_2 b > y_0 + y_1 a + y_2 b$$
$$\iff [x_0 > y_0 \,\&\, x_1 = y_1 \,\&\, x_2 = y_2] \vee [x_1 > y_1 \,\&\, x_2 = y_2] \vee [x_2 > y_2].$$

PROOF. The identity $1 = a \cdot a - b$ exhibits that $a \perp b$.

The second equivalence implies clearly the first one, and follows itself from the following Claim, applied to $(x_0 - y_0)$, $(x_1 - y_1)$, $(x_2 - y_2)$.

Claim. If $|x|, |y|, |z| < \frac{a}{2}$, *then* $x + ya + zb > 0$ *if and only if either* $x > 0$ *and* $y = z = 0$; *or* $y > 0$ *and* $z = 0$; *or* $z > 0$.

If $z = 0$, then the Claim follows from Lemma 3. If $z \neq 0$, compute $x + ya + zb = x + ya + z(a^2 - 1) = (x - z) + ya + za^2$; now

$$|(x - z) + ya| \leq a + \frac{a^2}{2} < a^2 \leq |z|a^2$$

(using the hypothesis $a > 2$), and so the sign of $(x - z) + ya + za^2$ is the same as the sign of z, as required to complete the proof of the Claim. $\dashv$

Proof of Theorem B. It suffices to derive a contradiction from the hypothesis that α decides coprimeness from ***Lin***, but for some $a > 2$ and $v \geq 1$, with $b = a^2 - 1$,

$$\text{(20)} \qquad \boldsymbol{N}_{\boldsymbol{Lin}} \models \alpha^{(v)}(a, b) = 1 \quad \text{with } 2^{2v+3} < a;$$

because for each such α and every $a > 2$ we must then have

$$2c_\alpha(a, b) + 3 \geq \log_2 a,$$

and since $c_\alpha(a, b) \geq 1$ by (12) and $\log_2(a^2 - 1) < 2 \log_2 a$,

$$5c_\alpha(a, b) \geq 2c_\alpha(a, b) + 3 \geq \log_2 a > \frac{1}{2} \log_2(a^2 - 1)$$

which is the desired conclusion.

So assume (20), set $\lambda = 1 + 2^v$, and notice that $2^{2v} < \frac{1}{4}a$, so that as in Lemma 5 we have an imbedding

$$\imath : \boldsymbol{N}_{\boldsymbol{Lin}} \restriction G_v(a, b) \rightarrowtail \boldsymbol{N}_{\boldsymbol{Lin}},$$

defined by

$$\imath\Big(\frac{x_0 + x_1 a + x_2 b}{2^v}\Big) = \frac{x_0 + x_1 \lambda a + x_2 \lambda b}{2^v}$$
$$\Big(|x_0|, |x_1|, |x_2| \leq 2^{2v}, \frac{x_0 + x_1 a + x_2 b}{2^v} \in G_v(a, b)\Big),$$

using now Lemma 8 instead of Lemma 3. This gives

$$\boldsymbol{N}_{\boldsymbol{Lin}} \models \alpha^{(v)}(\lambda a, \lambda b) = 1$$

by Lemma 1, so λa and λb are coprime—which they are not. $\dashv$

§5. The main result.

To prove Theorem A using the strategy in the proof of Theorem 6, we need to establish analogs of the key *Uniqueness*, *Inclusion* and *Imbedding* Lemmas 3, 4, 5 when division with remainder are among the givens, and these require just a bit of number theory. In this section x, y, z and x_i, y_i, z_i range over $\mathbb{Z}$.

A fraction $\dfrac{a}{b}$ (or, for easier typing, the pair of numbers (a, b)) is a *good approximation of* $\sqrt{2}$ if $b \geq 2$, $a \perp b$ and

$$\left|\frac{a}{b} - \sqrt{2}\right| < \frac{1}{b^2}. \tag{21}$$

There are infinitely many good approximations of $\sqrt{2}$, [4, Theorem 185]. On the other hand, by Liouville's Theorem, none of these is too good, i.e., *for all* y, z *with* $z \neq 0$,[3]

$$\left|\frac{y}{z} - \sqrt{2}\right| > \frac{1}{5z^2}. \tag{22}$$

A pair of numbers (a, b) is *difficult* if $a \perp b$,

$$2 \leq b < a < 2b, \tag{23}$$

and for all y, z,

$$0 < |z| < \frac{a}{2\sqrt{10}} \implies \left|\frac{a}{b} - \frac{y}{z}\right| > \frac{1}{10z^2}. \tag{24}$$

LEMMA 9. (1) *Each good approximation of* $\sqrt{2}$ *is difficult.*
(2) *If* (a, b) *is a difficult pair, then for all* y, z,

$$0 < |y| < \frac{a}{2\sqrt{10}} \implies |ya + zb| > \frac{a}{20|y|}. \tag{25}$$

PROOF. (1) Let (a, b) be a good approximation of $\sqrt{2}$. Then (23) follows from

$$1 < \sqrt{2} - \frac{1}{4} \leq \sqrt{2} - \frac{1}{b^2} < \frac{a}{b} < \sqrt{2} + \frac{1}{b^2} \leq \sqrt{2} + \frac{1}{4} < 2.$$

For property (24) of difficult pairs, let $0 < |z| < \dfrac{a}{2\sqrt{10}}$, and use (22):

$$\begin{aligned}\left|\frac{a}{b} - \frac{y}{z}\right| &\geq \left|\frac{y}{z} - \sqrt{2}\right| - \left|\frac{a}{b} - \sqrt{2}\right| \\ &> \frac{1}{5z^2} - \frac{1}{b^2} > \frac{1}{5z^2} - \frac{4}{a^2} > \frac{1}{5z^2} - \frac{1}{10z^2} = \frac{1}{10z^2}.\end{aligned}$$

[3]The proof of Theorem 191 in [4] shows that if $\xi \in \mathbb{R}$ and $f(\xi) = 0$ for some irreducible polynomial $f(T) \in \mathbb{Z}[T]$ of degree $n > 1$, then for all y, z with $z \neq 0$,

$$\left|\frac{y}{z} - \xi\right| > \frac{1}{C|z|^n},$$

with $C = \sup\{|f'(t)| \mid \xi - 1 < t < \xi + 1\}$. For $f(T) = T^2 - 2$ and $\xi = \sqrt{2}$, we can set $C = 5$ because $\sup\{2t \mid \sqrt{2} - 1 < t < \sqrt{2} + 1\} < 5$.

(2) is very useful and easy: Let (a, b) be a difficult pair and suppose that $0 < |y| < \frac{a}{2\sqrt{10}}$. Then

$$|ya + zb| = |y|b\left|\frac{a}{b} + \frac{z}{y}\right| > \frac{|y|b}{10y^2} = \frac{b}{10|y|} > \frac{a}{20|y|}. \qquad \dashv$$

Lemma 10 (Uniqueness). *Suppose (a, b) is a difficult pair, $1 \le \lambda \in \mathbb{N}$, and $|x_i|^2, |y_i|^2 < \frac{\sqrt{a}}{2\sqrt{20}}$ for $i = 0, 1, 2, 3$ with $x_3, y_3 > 0$. Then*

$$\frac{x_0 + x_1\lambda a + x_2\lambda b}{x_3} = \frac{y_0 + y_1\lambda a + x_2\lambda b}{y_3}$$
$$\iff [y_3x_0 = x_3y_0 \,\&\, y_3x_1 = x_3y_1 \,\&\, y_3x_2 = x_3y_2],$$
$$\frac{x_0 + x_1\lambda a + x_2\lambda b}{x_3} > \frac{y_0 + y_1\lambda a + y_2\lambda b}{y_3}$$
$$\iff [y_3x_1 = x_3y_1 \,\&\, y_3x_2 = x_3y_2 \,\&\, y_3x_0 > x_3y_0]$$
$$\text{or } [y_3(x_1\lambda a + x_2\lambda b) > x_3(y_1\lambda a + y_2\lambda b)].$$

Proof. The two equivalences follow from the next two Claims applied to $(y_3x_0 - x_3y_0) + (y_3x_1 - x_3y_1)\lambda a + (y_3x_2 - x_3y_2)\lambda b$.

Claim 1. If $x + y\lambda a + z\lambda b = 0$ and $|x|, |y|, |z| < \frac{\sqrt{a}}{\sqrt{20}}$, then

$$x = y = z = 0.$$

Proof of Claim 1. Assume the hypothesis. The case $y = z = 0$ is trivial, and if $y = 0$ and $z \neq 0$, then

$$b \le \lambda|z|b = |x| < \frac{\sqrt{a}}{\sqrt{20}},$$

which contradicts $a < 2b$. So we may assume that $y \neq 0$. From $a > 2$ we obtain $0 < |y| < \frac{\sqrt{a}}{\sqrt{20}} < \frac{a}{2\sqrt{10}}$, so by (25):

$$|y\lambda a + z\lambda b| \ge |ya + zb| > \frac{a}{20|y|} > |x|.$$

This contradicts the assumption that $y\lambda a + z\lambda b = -x$, and completes the proof of Claim 1.

Claim 2. Suppose that $|x|, |y|, |z| < \frac{\sqrt{a}}{\sqrt{20}}$. Then

$$x + y\lambda a + z\lambda b > 0 \iff [x > 0 \,\&\, y = z = 0] \vee [y\lambda a + z\lambda b > 0].$$

Proof of Claim 2. If $y = 0$, then the equivalence follows from Lemma 3; and if $y \neq 0$, then $|y\lambda a + z\lambda b| > |x|$ as above, and so adding x to $y\lambda a + z\lambda b$ cannot change its sign. This completes the proof of Claim 2, and of the Lemma. $\dashv$

Next we need a version of the Inclusion Lemma 4 that allows division with remainder among the givens, and for this we must use forms with varying denominators. Below we let $h \in \mathbb{N}$ and set

$$C(a,b\,;h) = \Big\{ \frac{x_0 + x_1 a + x_2 b}{x_3} \in \mathbb{N} \mid \ |x_0|, |x_1|, |x_2| \le h,\ 0 < x_3 \le h \Big\}.$$

When (a,b) is difficult and $h^2 < \dfrac{\sqrt{a}}{2\sqrt{20}}$, the fractions in this set have the form that makes Lemma 10 applicable. Their closure under the operations in ***Lin*** is easy:

Lemma 11. *For $h \ge 2$ and any k-ary function $f \in$ **Lin**,*

$$z_1, \dots, z_k \in C(a,b\,;h) \implies f(z_1,\dots,z_k) \in C(a,b\,;h^3).$$

Proof. For the case of addition, for example,

$$\frac{x_0 + x_1 a + x_2 b}{x_3} + \frac{y_0 + y_1 a + y_2 b}{y_3} \\ = \frac{(y_3x_0 + x_3y_0) + (y_3x_1 + x_3y_1)a + (y_3x_2 + x_3y_2)b}{x_3y_3},$$

and (using $h \ge 2$), $|(y_3x_i + x_3y_i)| \le h^2 + h^2 \le h^3$.

The other cases are equally simple. ⊣

Lemma 12. *Suppose (a,b) is a difficult pair, $h \ge 2$, $h^{44} \le a$, and $X, Y \in C(a,b\,;h)$. Then $\mathrm{iq}(X,Y), \mathrm{rem}(X,Y) \in C(a,b\,;h^{12})$.*

Proof. We can assume $Y > 0$. Write

$$X = \frac{x_0 + x_1 a + x_2 b}{x_3}, \quad Y = \frac{y_0 + y_1 a + y_2 b}{y_3}$$

where all $|x_i|, |y_i| \le h$, and $x_3, y_3 > 0$. Put $Q = \mathrm{iq}(X,Y)$, $R = \mathrm{rem}(X,Y)$, so

$$X = YQ + R \quad (0 \le R < Y). \tag{26}$$

We must show that $Q, R \in C(a,b\,;h^{12})$.

Case 1, $y_1 a + y_2 b = 0$. Now $R < \dfrac{y_0}{y_3} \le h$, and solving (26) for Q we get

$$Q = \frac{y_3}{y_0} \frac{(x_0 - x_3R) + x_1 a + x_2 b}{x_3} \in C(a,b\,;h^4). \tag{27}$$

Case 2, $y_1 a + y_2 b \ne 0$. Then $y_1 a + y_2 b > 0$, by Lemma 10, since $Y > 0$. We are going to show that in this case

$$KY > X \text{ with some natural number } K \le h^9,$$

so that $Q \leq h^9$. This yields the Lemma as follows: Solving the division equation (26) for R gives

$$R = \frac{(x_0y_3 - y_0x_3Q) + (x_1y_3 - y_1x_3Q)a + (x_2y_3 - y_2x_3Q)b}{x_3y_3}, \tag{28}$$

from which, easily, $R \in C(a, b\,; h^{12})$.

Suppose first that $y_1 = 0$; we claim that then $KY > X$ with $K = 4h^2$. To see why, note that $2x_3(y_1a + y_2b) = 2x_3y_2b > a$, but

$$|y_3(x_1a + x_2b)| \leq h^2a + h^2a = 2h^2a,$$

and so

$$4h^2x_3(y_1a + y_2b) > y_3(x_1a + x_2b).$$

Now Lemma 10 implies that

$$4h^2\frac{y_0 + y_1a + y_2b}{y_3} > \frac{x_0 + x_1a + x_2b}{x_3} \qquad (\text{that is, } 4h^2Y > X),$$

provided the squares of the relevant coefficients are $< \dfrac{\sqrt{a}}{2\sqrt{20}}$; but they clearly are, using $h^{44} \leq a$.

Next, suppose that $y_1 \neq 0$; we claim that then $KY > X$ with $K = 40h^2|y_1| \leq h^9$. By (25), we have $y_1a + y_2b > \dfrac{a}{20|y_1|}$, so

$$20x_3|y_1|(y_1a + y_2b) > a;$$

but (as in the case $y_1 = 0$)

$$|y_3(x_1a + x_2b)| \leq 2h^2a$$

and so

$$40h^2x_3|y_1|(y_1a + y_2b) > y_3(x_1a + x_2b).$$

Now Lemma 10 implies that

$$40h^2|y_1|\frac{y_0 + y_1a + y_2b}{y_3} > \frac{x_0 + x_1a + x_2b}{x_3},$$

provided the squares of the relevant coefficients are $< \dfrac{\sqrt{a}}{2\sqrt{20}}$; but they are, because (multiplying by $2\sqrt{20}$ and squaring),

$$\left(2\sqrt{20}(40h^2|y_1y_i|)^2\right)^2 \leq 2^{16} \cdot 5^5 \cdot h^{16} < 2^{28} \cdot h^{16} \leq h^{44} \leq a.$$

Thus $40h^2|y_1|Y > X$, as promised. $\dashv$

Now let

$$\boldsymbol{N}_{\boldsymbol{Lin}[\div]} = (\mathbb{N}, 0, 1, \boldsymbol{Lin}, \text{iq}, \text{rem})$$
$$= (\mathbb{N}, 0, 1, =, <, +, \dot{-}, 2\cdot, \tfrac{1}{2}\cdot, \text{Parity}, \text{iq}, \text{rem}),$$

and for any a, b, set

$$C_m(a,b) = C(a, b\,; 2^{2^{4m}})$$

Lemma 13 (Inclusion). *For every difficult pair (a,b) and every m,*

$$\text{if } 2^{2^{4m+6}} \le a, \text{ then } G_m(\boldsymbol{N}_{\boldsymbol{Lin}[\div]}, a, b) \subseteq C_m(a,b).$$

Proof is by induction on m, the case $m = 0$ being trivial. To apply Lemmas 11 and 12 at the induction step, we need to verify (under the hypothesis on a and m) the following two inequalities:

(1) $\left(2^{2^{4m}}\right)^{12} \le 2^{2^{4(m+1)}}$. This holds because

$$\left(2^{2^{4m}}\right)^{12} = 2^{12\cdot 2^{4m}} < 2^{2^4\cdot 2^{4m}} = 2^{2^{4(m+1)}}.$$

(2) $\left(2^{2^{4m}}\right)^{44} \le a$. In the same way:

$$\left(2^{2^{4m}}\right)^{44} = 2^{44\cdot 2^{4m}} < 2^{2^6\cdot 2^{4m}} = 2^{2^{4m+6}} \le a. \qquad \dashv$$

Lemma 14 (Imbedding). *Suppose (a,b) is a difficult pair, $2^{2^{4m+6}} \le a$, and set $\lambda = 1 + a!$. Then we have a partial algebra imbedding*

$$\imath : \boldsymbol{N}_{\boldsymbol{Lin}[\div]} \restriction C_m(a,b) \rightarrowtail \boldsymbol{N}_{\boldsymbol{Lin}[\div]}$$

defined by

$$\imath\Big(\frac{x_0 + x_1 a + x_2 b}{x_3}\Big) = \frac{x_0 + x_1\lambda a + x_2\lambda b}{x_3}$$
$$(\textit{all } |x_i| \le 2^{2^{4m}}, x_3 > 0, \frac{x_0 + x_1 a + x_2 b}{x_3} \in C_m(a,b)).$$

Proof. To simplify notation we set $h = 2^{2^{4m}}$, so $C_m(a,b) = C(a,b;h)$. Then $h^{64} = h^{2^6} = 2^{2^{4m}\cdot 2^6} = 2^{2^{4m+6}}$, so $h^{64} \le a$. It follows that

$$h^{24} < \frac{\sqrt{a}}{2\sqrt{20}}, \tag{29}$$

because multiplying by $2\sqrt{20}$ and squaring yields

$$80 \cdot h^{48} < 2^7 \cdot h^{48} \le h^{55} < h^{64} \le a.$$

It will be useful to define the map $\imath$ of the Lemma on a larger domain: By inequality 29 and Lemma 10 (with h^{12} instead of h and $\lambda = 1$) we can define $\imath : C(a,b;h^{12}) \to \mathbb{Q}$ by the equation of the Lemma, subject to

$\dfrac{x_0 + x_1 a + x_2 b}{x_3} \in C(a, b; h^{12})$, all $|x_i| \leq h^{12}, x_3 > 0$. Lemma 10 (with the present value of λ) also shows that $\imath$ is injective and order-preserving. Moreover, $\imath X \in \mathbb{N}$ for $X \in C(a, b; h^{12})$, because

$$x_0 + x_1 \lambda a + x_2 \lambda b = x_0 + x_1 a + x_2 b + (\lambda - 1)(x_1 a + x_2 b),$$

and $x_3 \mid \lambda - 1$ whenever $0 < x_3 \leq h^{12}$.

In the rest of the proof, let $X, Y \in C_m(a, b)$, and write

$$X = \frac{x_0 + x_1 a + x_2 b}{x_3}, \quad Y = \frac{y_0 + y_1 a + y_2 b}{y_3} \quad (|x_i|, |y_i| \leq h, x_3, y_3 > 0).$$

To show that $\imath$ preserves addition on $C_m(a, b)$, note that

$$X + Y = \frac{(y_3 x_0 + x_3 y_0) + (y_3 x_1 + x_3 y_1)a + (y_3 x_2 + x_3 y_2)b}{x_3 y_3},$$

and $X + Y \in C(a, b; h^{12})$, because $|y_3 x_i + x_3 y_i| \leq 2h^2 \leq h^{12}$. A direct computation with these expressions shows that $\imath(X + Y) = \imath X + \imath Y$. The same argument works for all the functions in ***Lin***.

For division with remainder, let $Y > 0$, put $Q = \mathrm{iq}(X, Y)$, $R = \mathrm{rem}(X, Y)$, so

$$X = YQ + R \quad (0 \leq R < Y).$$

Because $h^{44} \leq a$, Lemma 12 gives $Q, R \in C(a, b; h^{12})$. We claim that

$$\imath Q = \mathrm{iq}(\imath X, \imath Y), \quad \imath R = \mathrm{rem}(\imath X, \imath Y). \tag{30}$$

(This is more than sufficient for our purpose.) We distinguish two cases, following the proof of Lemma 12.

Case 1, $y_1 a + y_2 b = 0$. Then $0 \leq R < Y = \dfrac{y_0}{y_3} \leq h$, so $\imath R = R$ and $\imath Y = Y$. The explicit formula (27) for Q yields

$$\imath Q = \frac{y_3}{y_0} \frac{(x_0 - x_3 R) + x_1 \lambda a + x_2 \lambda b}{x_3},$$

and a direct computation with these expressions for $\imath R$, $\imath Y$ and $\imath Q$ gives $\imath X = \imath Y \cdot \imath Q + \imath R$, which yields (30) in view of $0 \leq \imath R < \imath Y$.

Case 2, $y_1 a + y_2 b > 0$. Now $Q \leq h^9$, which implies $\imath Q = Q$. The explicit formula (28) for R yields

$$\imath R = \frac{(x_0 y_3 - y_0 x_3 Q) + (x_1 y_3 - y_1 x_3 Q)\lambda a + (x_2 y_3 - y_2 x_3 Q)\lambda b}{x_3 y_3};$$

with these expressions for $\imath R$ and $\imath Q$ we get $\imath X = \imath Y \cdot \imath Q + \imath R$, which yields (30) in view of $0 \leq \imath R < \imath Y$. ⊣

PROOF OF THEOREM A. Let α be a program as in the hypothesis of the theorem. We prove the (possibly) stronger result that (2) holds for all difficult pairs (a, b). So let (a, b) be a difficult pair, put $v = c_\alpha(a, b)$, and suppose, towards a contradiction, that $v \leq \dfrac{1}{10}\log_2\log_2 a$. Since $v \geq 1$ by (12), this gives $4v + 6 \leq 10v \leq \log_2\log_2 a$, so

$$\mathbf{N}_{\boldsymbol{Lin}[\div]} \models \alpha^{(v)}(a, b) = 1 \text{ and } 2^{2^{4v+6}} \leq a. \tag{31}$$

We now proceed as in the proof of Theorem 6. From (31):

(1) $\mathbf{N}_{\boldsymbol{Lin}[\div]} \upharpoonright G_v(\mathbf{N}_{\boldsymbol{Lin}}, a, b) \models \alpha^{(v)}(a, b) = 1$, by (15).

(2) $\mathbf{N}_{\boldsymbol{Lin}[\div]} \models \alpha^{(v)}(\lambda a, \lambda b) = 1$, by Lemma 14, with $\lambda = 1 + a!$.

(3) α decides coprimeness, so $\lambda a \perp \lambda b$—which is absurd. ⊣

§6. Lower bounds for RAMs and other algorithms.

We show in this section that the lower bounds of Theorems A and B hold for recursive programs on extensions of the structures $\mathbf{N}_{\boldsymbol{Lin}}, \mathbf{N}_{\boldsymbol{Lin}[\div]}$ by arbitrary "logical" data structures. This covers Random Access Machines, which can be faithfully represented by such extended recursive programs; but it also suggests that the results hold for *all algorithms* from the relevant givens, and we discuss this possibility briefly in the last paragraph.

Inessential Extensions. Let $\mathbf{A} = (A, 0, 1, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi})$ be a partial algebra. An *inessential extension*[4] of $\mathbf{A}$ is a partial algebra

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

with the following properties:

(IE1) $A \subseteq B$, and $\mathbf{A}$ and $\mathbf{B}$ 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" $\psi \in \Psi^{\mathbf{B}}$ of arity n and all $x_1, \dots, x_n \in B$,

$$\pi^B \psi(x_1, \dots, x_n) = \psi(\pi^B x_1, \dots, \pi^B x_n). \tag{32}$$

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

We may think of an inessential extension $\mathbf{B}$ as enriching $\mathbf{A}$ with various "logical" data structures (arrays, trees, etc.) which are used to express some algorithm from Φ, but do not include any non-trivial, new functions on A or affect in any essential way how the partial functions in Φ are used by the algorithm. Notice, for example, that a simple constant function

$$\psi(x) = w_0 \quad (w_0 \in A, w_0 \neq 0, 1),$$

[4]This notion is due to Itay Neeman, and improves considerably our original version of this section.

cannot be a new given of an inessential extension of $\boldsymbol{A}$ if A is infinite; because that would demand of every permutation π of A fixing 0 and 1 that

$$\pi w_0 = \pi\psi(x) = \psi(\pi x) = w_0,$$

which evidently fails if π moves w_0. In a similar way, if we just add a new name for an old given ϕ, the resulting algebra is not automatically an inessential extension, because ϕ need not satisfy (32) for every π.

LEMMA 15. *Suppose $\boldsymbol{B}$ is an inessential extension of an infinite partial algebra $\boldsymbol{A}$, and $X \subseteq A$.*

(1) *If π and π^B are as in* (IE2) *and π fixes every $y \in G_m(\boldsymbol{A}, X)$, then π^B fixes every $y \in G_m(\boldsymbol{B}, X)$.*

(2) *$G_m(\boldsymbol{B}, X) \cap A = G_m(\boldsymbol{A}, X)$.*

PROOF. We show (1) and (2) together by induction on m, the basis being trivial since

$$G_0(\boldsymbol{B}, X) = X \cup \{0, 1\} = G_0(\boldsymbol{A}, X).$$

At the induction step, suppose that π and π^B are as in (IE2), and that π fixes every element of $G_{m+1}(\boldsymbol{A}, X)$. Let y be an element of $G_{m+1}(\boldsymbol{B}, X)$. If $y = \phi(x_1, \dots, x_n)$ with $x_1, \dots, x_n \in G_m(\boldsymbol{B}, X)$ and an old given ϕ, then $x_1, \dots, x_n \in G_m(\boldsymbol{A}, X)$ by (2) of the induction hypothesis, and so $y \in G_{m+1}(\boldsymbol{A}, X)$ and $\pi(y) = y$ by the hypothesis on π. If

$$y = \psi(x_1, \dots, x_n) \text{ with } x_1, \dots, x_n \in G_m(\boldsymbol{B}, X)$$

and a new given ψ, then π^B fixes $x_1, \dots, x_n$ by (1) of the induction hypothesis, and so by (32),

$$\pi^B y = \pi^B \psi(x_1, \dots, x_n) = \psi(\pi^B x_1, \dots, \pi^B x_n) = \psi(x_1, \dots, x_n) = y,$$

as required. To prove (2), assume towards a contradiction that $y \in A$ is such that

$$y \in G_{m+1}(\boldsymbol{B}, X) \setminus G_{m+1}(\boldsymbol{A}, X).$$

Take a permutation π of A that fixes every member of $G_{m+1}(\boldsymbol{A}, X)$ but moves y. This is possible because $G_{m+1}(\boldsymbol{A}, X)$ is finite and A is infinite, and it contradicts (1), since $\pi^B(y) = \pi y$. ⊣

THEOREM 16. *Suppose the recursive program β decides coprimeness in an inessential extension $\boldsymbol{B}$ of*

$$\boldsymbol{N}_{\boldsymbol{Lin}[\div]} = (\mathbb{N}, 0, 1, \boldsymbol{Lin}, \text{iq}, \text{rem}).$$

Then for every difficult pair (a, b),

$$c_\beta(a, b) > \frac{1}{10} \log_2 \log_2 a.$$

Proof. Let $v = c_\beta(a,b)$, so $v \geq 1$. By the Absoluteness Lemma 2,

$$\boldsymbol{B} \upharpoonright G_v(\boldsymbol{B}, a, b) \models \beta^{(v)}(a,b) = 1.$$

Assume towards a contradiction that $v \leq \dfrac{1}{10}\log_2\log_2 a$. As in the proof of Theorem A it follows that $2^{2^{4v+6}} \leq a$. Lemma 14 supplies an imbedding

$$\iota : \boldsymbol{A} \upharpoonright G_v(\boldsymbol{A}, a, b) \rightarrowtail \boldsymbol{A} \quad (\boldsymbol{A} = \boldsymbol{N}_{\boldsymbol{Lin}[\div]})$$

such that for a suitable $\lambda > 1$, $\iota a = \lambda a$ and $\iota b = \lambda b$. Since ι is injective, the complements of $G_m(\boldsymbol{A}, a, b)$ and $\iota[G_m(\boldsymbol{A}, a, b)]$ are equinumerous, so we can extend ι to a permutation π of A, and take π^B to be an extension of π to $\boldsymbol{B}$ as in (IE2). Then $\iota^B := \pi^B \upharpoonright G_v(\boldsymbol{B}, a, b)$ is an imbedding

$$\iota^B : \boldsymbol{B} \upharpoonright G_v(\boldsymbol{B}, a, b) \rightarrowtail \boldsymbol{B},$$

because, for the old givens,

$$\iota\phi(x_1, \dots, x_n) = \phi(\iota x_1, \dots, \iota x_n)$$

is guaranteed by the hypothesis on ι and Lemma 15, which insures that the only elements to which an old given ϕ is applied are, indeed, in $G_v(\boldsymbol{A}, a, b)$; and for the new givens,

$$\iota^B\psi(x_1, \dots, x_n) = \psi(\iota^B x_1, \dots, \iota^B x_n)$$

is guaranteed by (32).

The Imbedding Lemma 1 now yields $\boldsymbol{B} \models \beta^{(v)}(\lambda a, \lambda b) = 1$, so $\lambda a \perp \lambda b$, which is absurd. ⊣

The same argument can be used to show that all the lower bound results we have established for recursive programs in $\boldsymbol{N}_{\boldsymbol{Lin}}$ and $\boldsymbol{N}_{\boldsymbol{Lin}[\div]}$ apply to recursive programs on inessential extensions of these structures, and we proceed to show that these "include" RAMs.

Adding functions to an algebra. Given a partial algebra

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

let $(A \to A)$ be the set of all functions from the set A to itself, let ε be the constant function in $(A \to A)$ with value 0, and, using variables $a, b, c, t \dots$ over A and $X, Y, Z, \dots$ over $(A \to A)$, define the operations ap, update, and indupdate as follows:

(1) $\mathrm{ap}(X, a) = X(a)$ (application).
(2) $\mathrm{update}(X, a, b) = Z$, where $Z(a) = b$, and $Z(t) = X(t)$ for $t \neq a$.
(3) $\mathrm{indupdate}(X, Y, a, b) = Z$, where $Z(Y(a)) = b$, and $Z(t) = X(t)$ for $t \neq Y(a)$ (indirect update).

Consider now the extended partial algebra

$$\boldsymbol{A}' = (A', 0, 1, \{\phi^{\boldsymbol{A}}\}_{\phi\in\Phi}, \varepsilon, \mathrm{ap}, \mathrm{update}, \mathrm{indupdate}),$$

where $A' = A \cup (A \to A)$, with $A \cap (A \to A) = \emptyset$.

LEMMA 17. *The partial algebra $\boldsymbol{A}'$ is an inessential extension of $\boldsymbol{A}$.*[5]

PROOF. Given a permutation π of A fixing 0 and 1, we extend it to a permutation of A' (using the same name) by the obvious

$$(\pi X)(t) = \pi(X(\pi^{-1}(t))).$$

The crucial (32) is easy to verify for the new givens. For example,

$$\pi\varepsilon(x) = \pi 0 = 0 = \varepsilon(\pi x),$$

and for application,

$$\mathrm{ap}(\pi X, \pi t) = (\pi X)(\pi t) = \pi(X(\pi^{-1}(\pi t))) = \pi(X(t)) = \pi \mathrm{ap}(X, t).$$

We will omit the details. ⊣

LEMMA 18. *Let M be a Random Access Machine which computes a partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ from the partial functions in a set Φ with time complexity $T(\vec{x})$. Then there is a recursive program α in the inessential extension $\boldsymbol{A}'$ of $\boldsymbol{A} = (\mathbb{N}, 0, 1, \Phi)$ which also computes f, so that*

$$c_\alpha(\vec{x}) \leq T(\vec{x}) \qquad (f(\vec{x})\downarrow).$$

PROOF. By [3] the machine M has the following components:

1. *Instructions* labelled $0, \dots, N$, with $N \geq 1$. We assume that 1 is the *initial* instruction, where the machine begins the computation, and 0 is the *terminal* instruction, which stops the computation.
2. *Accumulators* $\mathrm{acc}_1, \dots, \mathrm{acc}_K$ $(K \geq n+1)$; each accumulator stores a natural number. The computation starts with the input $\vec{x}$ stored in the first n of these, and 0 in the other accumulators; at the end of a convergent computation the output is the content of acc_{n+1}.
3. Infinitely many *registers* $R[0], R[1], \dots$; each register stores a natural number. Initially, $R[j] = 0$ for every j.

Each instruction has one of the forms

a. *command*; *goto* b
a. *if* $\mathrm{acc}_j = 0$ *then goto* b *else goto* c
a. *if* $\mathrm{acc}_j > N$ *then goto* 0 *else goto* acc_j
0. *End*

where $1 \leq a, b, c \leq N$ and the commands are of the following types:

1. $\mathrm{acc}_i := a$ $(0 \leq a \leq N)$.
2. $\mathrm{acc}_i := \mathrm{acc}_j$.
3. $\mathrm{acc}_i := \phi(\mathrm{acc}_{j_1}, \dots, \mathrm{acc}_{j_m})$. where $\phi \in \Phi$ is m-ary.
4. $R[\mathrm{acc}_i] := \mathrm{acc}_j$.
5. $\mathrm{acc}_i := R[\mathrm{acc}_j]$.
6. $R[R[\mathrm{acc}_i]] := \mathrm{acc}_j$.

[5]The functions $\varepsilon, \mathrm{ap}, \mathrm{update}, \mathrm{indupdate}$ are *logical* in the sense of Tarski [10], and the Lemma holds for any extension of $\boldsymbol{A}$ by logical, higher-type objects over $\boldsymbol{A}$.

There are many variations on these instructions, which, however, do not affect the result or require any essential changes in the proof.

Let us first assume for simplicity that the functions

$$C_a(x) = a,\ \chi_a(x) = \text{if } (x = a) \text{ then } 0 \text{ else } 1 \quad (0 \le a \le N)$$

are among the givens in Φ—it will be easy to remove this assumption at the end. The required program α has the two equations

$$f(\vec{x}) = g(1, \vec{x}, \vec{0}, X), \qquad g(a, \vec{u}, X) = \begin{cases} E_0, & \text{if } a = 0, \\ & \vdots \\ E_N, & \text{if } a = N, \\ g(a, \vec{u}, X) & \text{otherwise,} \end{cases}$$

where the terms E_a are defined for the various types of instructions as follows (with simple indices, to avoid too many dots):

a. $\text{acc}_1 := 0$, *goto* b: $E_a \equiv g(b, 0, u_2, \dots, X)$

a. $\text{acc}_1 := \text{acc}_2$, *goto* b: $E_a \equiv g(b, u_2, u_2, \dots, X)$

a. $\text{acc}_1 := \phi(\text{acc}_2, \dots, \text{acc}_m)$, *goto* b: $E_a \equiv g(b, \phi(u_2, \dots, u_m), u_2, \dots, X)$

a. $\text{acc}_1 := R[\text{acc}_2]$, *goto* b: $E_a \equiv g(b, X(u_2), u_2, \dots, X)$

a. $R[\text{acc}_1] := \text{acc}_2$; *goto* b: $E_a \equiv g(b, \vec{u}, \text{update}(X, u_1, u_2))$

a. $R[R[\text{acc}_1]] := \text{acc}_2$, *goto* b: $E_a \equiv g(b, \vec{u}, \text{indupdate}(X, X, u_1, u_2))$

a. *if* $\text{acc}_j = 0$ *goto b else goto c*:

$$E_a \equiv \text{if } (u_j = 0) \text{ then } g(b, \vec{u}, X) \text{ else } g(c, \vec{u}, X)$$

a. *if* $\text{acc}_j > N$ *then goto* 0 else *goto* acc_j:

$$E_a \equiv \text{if } (u_j > N) \text{ then } g(0, \vec{u}, X) \text{ else } g(u_j, \vec{u}, X)$$

a. End: $E_a \equiv u_{n+1}$.

The precise definition of g is by a nested conditional which combines all the cases. The proof of the following key fact is by induction on k and uses critically the definition of $C(E)$ for E a conditional term: *if M is started with instruction $a > 0$, $\vec{u}$ loaded in the accumulators, $R[j] = X(j)$ for all j, and M terminates in k steps with w in* acc_{n+1}, *then $\overline{g}(a, \vec{u}, X) = w$ and $c_g(a, \vec{u}, X) \le k$.*

To avoid the assumption that the constant functions C_a and the tests χ_a are included in Φ, we replace $0, 1, \dots, N$ by tuples $\vec{a}_0, \vec{a}_1, \dots, \vec{a}_N$ of 0's and 1's, for which we can define the corresponding functions using the constants 0, 1 and conditionals. ⊣

Together with Lemma 16 (and its version for ***Lin***), this gives:

THEOREM 19. (A) *If a Random Access Machine decides the coprimeness relation $x \perp y$ from $=$, $<$, $+$, $\dot{-}$,* iq *and* rem *with time complexity $T(x, y)$, then for every difficult pair (a, b),*

$$T(a, b) > \frac{1}{10} \log_2 \log_2 a.$$

(B) *If a Random Access Machine decides the coprimeness relation* $x \perp y$ *from* $=, <, +, \dot{-}, 2 \cdot x, \mathrm{iq}(x, 2)$, *and* $\mathrm{rem}(x, 2)$ *with time complexity* $T(x, y)$, *then for all* $a > 2$,

$$T(a, a^2 - 1) > \frac{1}{10} \log_2(a^2 - 1).$$

The Refined Church-Turing Thesis. In [8] (and earlier articles cited there), Moschovakis has proposed that all algorithms are *relative to givens*, and that they can be *faithfully represented* by recursive programs on suitable structures which codify these givens. Now this does not mean that an algorithm which is (intuitively) relative to certain functions Φ on $\mathbb{N}$ can be represented faithfully by a recursive program in $(\mathbb{N}, 0, 1, \Phi)$; this fails for RAMS, for example, whose simulation by recursive programs requires $T(n) \log_2 T(n)$ time, because strings of numbers must be coded and manipulated in $\mathbb{N}$ [3]. The results of this section suggest—and provide some evidence for—the following, concrete interpretation of the proposal in [8] for the case of *first-order-algorithms*:

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

This proposal identifies the intuitive notion of (worst case, time) *optimality* for relative first-order algorithms with a precise mathematical notion, much like the classical Church-Turing Thesis does the same for the intuitive notion of (relative) *computability*. It is not possible to discuss it intelligently in this communication, but, if it is true, then the lower bounds we have established from ***Lin*** and ***Lin***$[\div]$ hold for all algorithms.

REFERENCES

[1] Lou van den Dries, *Generating the greatest common divisor, and limitations of primitive recursive algorithms*, ***Foundations of computational mathematics***, vol. 3 (2003), pp. 297–324.

[2] Lou van den Dries and Yiannis N. Moschovakis, *Arithmetic complexity*, (200?), in preparation.

[3] P. van Emde Boas, *Machine models and simulations*, ***Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity*** (Jan van Leeuwen, editor), Elsevier and MIT Press, 1994, pp. 1–66.

[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***, second ed., Addison-Wesley, 1973.

[6] Yishay Mansoor, Baruch Schieber, and Prasoon Tiwari, *A lower bound for integer greatest common divisor computations*, ***Journal of the Association for Computing Machinery***, vol. 38 (1991), pp. 453–471.

[7] J. McCarthy, *A basis for a mathematical theory of computation*, ***Computer programming and formal systems*** (P. Braffort and D Herschberg, editors), North-Holland, 1963, pp. 33–70.

[8] Yiannis N. Moschovakis, *What is an algorithm?*, ***Mathematics unlimited – 2001 and beyond*** (B. Engquist and W. Schmid, editors), Springer, 2001, pp. 919–936.

[9] ———, *On primitive recursive algorithms and the greatest common divisor function*, ***Theoretical Computer Science***, vol. 301 (2003), pp. 1–30.

[10] Alfred Tarski, *What are logical notions?*, ***History and Philosophy of Logic***, vol. 7 (1986), pp. 143–154, edited by John Corcoran.

UNIVERSITY OF ILLINOIS
DEPARTMENT OF MATHEMATICS
1409 W. GREEN STREET
URBANA, IL 61801, USA
E-mail: vddries@math.uiuc.edu

DEPARTMENT OF MATHEMATICS
UNIVERSITY OF CALIFORNIA
LOS ANGELES, CA 90095-1555, USA

and DEPARTMENT OF MATHEMATICS
UNIVERSITY OF ATHENS
ATHENS, GREECE

E-mail: ynm@math.ucla.edu