ARITHMETIC COMPLEXITY

YIANNIS N. MOSCHOVAKIS

Contents

Many of the results in these Notes are due to Lou van den Dries [2] and jointly to van den Dries and the author, [3], [4]. The most interesting of these results are credited in the text.

Introduction. Perhaps the simplest way to introduce the subject of *Arithmetic Complexity* is by giving a fairly precise formulation of its main open problem.

The Euclidean algorithm on the natural numbers 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. It is an algorithm *from* (*relative to*) the remainder function rem, meaning that in its execution, ε has access to an "oracle" which provides on demand (in one "time unit") the value $\operatorname{rem}(x,y)$ for any x and y. It is not hard to prove that

$$c_\varepsilon(a,b) \leq 2\log b \quad (a \geq b \geq 2),$$

where $c_\varepsilon(a,b)$ is the number of divisions (calls to the rem-oracle) required for the computation of $\gcd(a,b)$ by the Euclidean, and logarithms are to the base 2. Much more is actually known about $c_\varepsilon(a,b)$, but this upper bound suggests the proper formulation of the Euclidean's (worst case) optimality among its *peers*—algorithms from rem:

Main Conjecture. *For every algorithm α from* rem *which computes the function* $\gcd(x,y)$, *there is a (positive, rational) constant r and a sequence $\{x_n, y_n\}_n$ of pairs with $x_n \geq y_n \geq 1$ and $\lim_n y_n = \infty$, such that for all n,*

$$c_\alpha(x_n,y_n) \geq r \log y_n,$$

where $c_\alpha(x,y)$ is the number of calls to the rem*-oracle executed by α in the computation of* $\gcd(x,y)$.

Now, there are Turing machines which compute $\gcd(x,y)$ making no calls at all to any rem-oracle, simply because $\gcd(x,y)$ is a computable function; thus, to make the conjecture precise, we must employ a notion of (relative) *algorithm from given functions* $\mathbf{\Phi}$ which does not take any functions outside $\mathbf{\Phi}$ for granted. (Turing machines have free access to the successor and predecessor operations, which are built into their definitions.) The simplest, appropriate notion is that of *recursive programs*, and one of the results we will prove is the following partial answer to the Main Conjecture:

Theorem A ([3]). *If a recursive program α decides the coprimeness relation $x \perp y$ from $=, >, +, \dot{-}$,* iq *and* rem, *then for infinitely many coprime pairs $a > b > 1$,*

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

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

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

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

Here $c_\alpha(a, b)$ is the number of calls to the givens,

$$x \dot{-} y = \begin{cases} x - y & \text{if } x \geq y, \\ 0, & \text{otherwise,} \end{cases}$$

$\mathrm{iq}(x, y)$ is the (integer) quotient of x by y, and two numbers are coprime if they are both positive and they have no common divisors,

$$x \perp y \iff x, t \geq 1 \ \& \ \gcd(x, y) = 1.$$

The result is in some ways stronger than the Main Conjecture (for recursive programs), since it decides the relation of coprimeness (which is more complex than the gcd function) and it allows more givens; 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. On the other hand, Theorem A *locates difficult inputs* for all recursive programs which decide coprimeness in terms of natural, number-theoretic conditions and reveals a potentially interesting connection between *computational* and *Diophantine* complexity.

In addition to the greatest common divisor and coprimeness, we will also obtain lower bounds for algorithms (of various kinds and from various sets of given functions) which compute many interesting classical functions on $\mathbb{N}$ or decide interesting relations on numbers, e.g.,

$$a \text{ divides } b, \ \ a \text{ is prime}, \ \ a \text{ is square-free}, \ \ a \text{ is a perfect square.}$$

Our results will be about algorithms from various *givens* (operations which are assumed as primitives), and relative to various measures of complexity, and they will apply to a wide variety of *computation models.* The methods of proof involve simple ideas from algebra, logic and recursion theory, as well as some basic, elementary facts from number theory whose proofs will be included in these notes.

§1. Preliminaries and notation. We will use (mostly) standard notation:

$$\mathbb{N} = \{0, 1, \dots\}, \ \mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}, \ \mathbb{Q} = \{\frac{m}{n} \mid m \in \mathbb{Z}, n \in \mathbb{N}, n > 0\}$$

are the sets of *natural numbers* (or just *numbers*), the *integers*, the *rational numbers*, and (for the few cases where we will need it), $\mathbb{R}$ is the set of real numbers. With each relation $R \subseteq A^n$ on some set A we associate its *characteristic function*

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

in some cases it is useful to "identify" R with χ_R, i.e., speak of R as if it were a function—meaning χ_R. An algorithm *decides* R if it computes χ_R.

1.1. Partial functions and functionals. 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. We write

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

and on occasion (in definitions) the ungrammatical "$f(\vec{x}) = \uparrow$" which is synonymous with "$f(\vec{x})\uparrow$". Notice that if f is total and $f \sqsubseteq g$, then $f = g$.

Partial functions compose *strictly*:

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

so that in particular,

$$f(g_1(\vec{x}), \ldots, g_n(\vec{x}))\downarrow \implies g_1(\vec{x})\downarrow, \ldots, g_n(\vec{x})\downarrow .$$

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

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

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

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

and it is *continuous* if

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

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

From Recursion Theory, we will need the following, simple result which justifies definitions by mutual recursion:

1.1.1. Lemma (The Fixed Point Lemma). *For every monotone and continuous functional*

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

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

$$\begin{gathered}
\overline{f}(\vec{x}) = F(\vec{x}, \overline{f}) \quad (\vec{x} \in A^n);\\
\text{if } (\forall \vec{x})[F(\vec{x}, g)\downarrow \implies F(\vec{x}, g) = g(\vec{x})], \text{ then } \overline{f} \sqsubseteq g.
\end{gathered}$$

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 (canonical) solution tuple* $\overline{f}_1, \dots, \overline{f}_K$.

Proof. For the one-equation case, define by recursion on $\mathbb{N}$ the *iterates*

$$
\begin{aligned}
\overline{f}^0(\vec{x}) &= \uparrow \quad \text{(i.e., } \overline{f}^0 \text{ is the totally undefined partial function)},\\
\overline{f}^{n+1}(\vec{x}) &= F(\vec{x}, \overline{f}^n);
\end{aligned}
$$

prove by induction, using monotonicity, that $\overline{f}^n \sqsubseteq \overline{f}^{n+1}$, so that

$$\overline{f}^0 \sqsubseteq \overline{f}^1 \sqsubseteq \overline{f}^2 \sqsubseteq \cdots;$$

and set $\overline{f} = \bigcup\{\overline{f}^n \mid n \in \mathbb{N}\}$, i.e.,

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

If $\overline{f}(\vec{x}) = w$, then, for some n,

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

by the definition, and hence $F(\vec{x}, \overline{f}) = w$, by monotonicity, since $\overline{f}^n \sqsubseteq \overline{f}$. On the other hand, if $F(\vec{x}, \overline{f}) = w$, then there is a finite $g \sqsubseteq \overline{f}$ such that $F(\vec{x}, g) = w$, by continuity, and then there is some n such that $g \sqsubseteq \overline{f}^n$; thus, by monotonicity, $F(\vec{x}, \overline{f}^n) = \overline{f}^{n+1}(\vec{x}) = w$, and so $\overline{f}(\vec{x}) = w$, which completes the proof that, for all $\vec{x}$,

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

To verify the minimality of $\overline{f}$, suppose that

$$(\forall \vec{x})[F(\vec{x}, g)\downarrow \implies F(\vec{x}, g) = g(\vec{x})],$$

show (by an easy induction on n, using monotonicity) that $\overline{f}^n \sqsubseteq g$, and infer the required $\overline{f} \sqsubseteq g$.

The argument for recursive systems of equations is similar. $\dashv$

1.2. Notations from arithmetic. The same, familiar symbols $+, -, \cdot, =, \leq$ etc., will be used for addition, subtraction, multiplication ... in any of the sets $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$. We will use *true, binary* logarithms, defined for all positive real numbers:

$$\log(x) = \text{the unique real number } y \text{ such that } 2^y = x \quad (x \in \mathbb{R}, x > 0).$$

The *floor* or *integer part* of $x \in \mathbb{R}$ is

$$\lfloor x \rfloor = \text{the largest integer } \leq x.$$

By the DIVISION THEOREM, if $x, y \in \mathbb{N}$ and $y > 0$, then there exist unique numbers q and r such that

$$x = yq + r \qquad (0 \le r < y); \tag{3}$$

if $x < y$, then $q = 0$ and $r = x$, while if $x \ge y$, then $q \ge 1$. We refer to (3) as the *correct division equation* (**c.d.e.**) for x, y, and we set

$$\mathrm{iq}(x, y) = q, \quad \mathrm{rem}(x, y) = r \quad (x \ge y \ge 1)$$

with the unique q and the r for which it holds. For the *divisibility relation*, we write

$$y \mid x \iff \mathrm{rem}(x, y) = 0 \quad (y > 0).$$

Two positive numbers are *coprime* if they have no common divisors,

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

Finally, the *greatest common divisor* of two natural numbers is what its name means:

$$\gcd(x, y) =_{\mathrm{df}} \text{ the largest } d \text{ such that } d \mid x \text{ and } d \mid y. \tag{4}$$

Thus,

$$x \perp y \iff x, y \ge 1 \ \&\ \gcd(x, y) = 1.$$

1.3. The Landau symbols. We define here—for the record—the most commonly used ways of comparing the growth rate of unary functions on $\mathbb{N}$:

$$\begin{aligned} f(n) = o(g(n)) &\iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \\ f(n) = O(g(n)) &\iff (\exists K, C)(\forall n \ge K)[f(n) \le Cg(n)] \\ f(n) = \Theta(g(n)) &\iff f(n) = O(g(n)) \ \&\ g(n) = O(f(n)) \\ f(n) = \Omega(g(n)) &\iff (\exists K, r)(\forall n \ge K)[f(n) \ge rg(n)) \end{aligned}$$

where the constants $K, C \in \mathbb{N}$ while r is a positive fraction.

1.4. The Euclidean algorithm. This classical process, first described in Book VII of the *Elements*, is (perhaps) the most ancient and still one of the most important algorithms in mathematics. We consider the "iterated division" rather than the original "iterated subtraction" version of the algorithm, which is implicit in Euclid and offers itself more easily to complexity analysis.

1.4.1. LEMMA. *The function* $\gcd(x, y)$ *satisfies the following recursion equation, for* $x \ge y \ge 1$*:*

$$\gcd(x, y) = \text{if } (y \mid x) \text{ then } y \text{ else } \gcd(y, \mathrm{rem}(x, y)). \tag{5}$$

PROOF. The pairs $\{x, y\}$ and $\{y, \mathrm{rem}(x, y)\}$ have exactly the same common divisors. ⊣

Equation (5) is an example of a *recursive program*, and it provides a procedure for computing $\gcd(x, y)$:

if $y \mid x$ *give output* y*, else set* $x := y, y := \mathrm{rem}(x, y)$ *and repeat.*

For example:

$$\begin{aligned}\gcd(231,165) &= \gcd(165,66) && \textbf{c.d.e. } 231 = 165\cdot 1 + 66\\ &= \gcd(66,33) && \textbf{c.d.e. } 165 = 66\cdot 2 + 33\\ &= 33 && \text{because } 33 \mid 66.\end{aligned}$$

The computation required three divisions in this case—the last one verifying that $33 \mid 66$. In general, we set

$$\begin{aligned}c_\varepsilon(x,y) = {} & \text{the number of divisions required to compute}\\ & \gcd(x,y) \text{ using (5)} \quad (x \geq y \geq 1),\end{aligned}$$

so that, directly from (5), for $x \geq y \geq 1$,

(6) $$c_\varepsilon(x,y) = \text{ if } (y \mid x) \text{ then } 1 \text{ else } 1 + c_\varepsilon(y, \mathrm{rem}(x,y)).$$

1.4.2. PROPOSITION. *For all $x \geq y \geq 2$, $c_\varepsilon(x,y) \leq 2\log y$.*

PROOF is by (complete) induction on y, and we must consider three cases:

Case 1, $y \mid x$; now $c_\varepsilon(x,y) = 1 \leq 2\log y$, since $y \geq 2$ and so $\log y \geq 1$.

Case 2, $x = yq_1 + r_1$ with $0 < r_1 < y$, but $r_1 \mid y$; now $c_\varepsilon(x,y) = 2$, and $2 \leq 2\log y$, as above.

Case 3, $x = yq_1 + r_1$ and $y = r_1q_2 + r_2$ with $0 < r_2 < r_1 < y$. Notice that the last, triple inequality implies that $y \geq 3$. If $r_2 = 1$, then only one more division is needed, so $c_\varepsilon(x,y) = 3$, and (easily) $3 < 2\log 3 \leq 2\log y$. Suppose then that $r_2 \geq 2$, and consider the next division,

$$r_1 = r_2q_3 + r_3 \quad (q_3 \geq 1, 0 \leq r_3 < r_2).$$

Using the facts that $q_2 \geq 1$ and $r_2 < r_1$,

$$y = r_1q_2 + r_2 \geq r_1 + r_2 > 2r_2,$$

which by the Induction Hypothesis for $r_2 \geq 2$ gives

$$\begin{aligned}c_\varepsilon(x,y) = 2 + c_\varepsilon(r_1,r_2) &\leq 2 + 2\log r_2\\ &\leq 2 + 2\log(\frac{y}{2}) = 2\Big(1 + \log(\frac{y}{2})\Big) = 2\log y,\end{aligned}$$

as required. ⊣

The best estimates for the function $c_\varepsilon(x,y)$ are easiest to express in terms of the classical *Fibonacci sequence*, which is defined by the recursion

$$F_0 = 0,\ F_1 = 1,\ F_{k+2} = F_k + F_{k+1},$$

so that $F_2 = 0 + 1 = 1$, $F_3 = 1 + 1 = 2$, $F_4 = 3$, $F_5 = 5$, etc. We leave these for the problems.

We end this section with the consideration of one more interesting algorithm, whose complexity analysis is substantially easier than that of the Euclidean.

1.5. The binary (Stein) algorithm. This modern algorithm computes $\gcd(x, y)$ in $O(\log x + \log y)$ steps, from "linear" operations, which are much simpler than division.

1.5.1. Proposition ([7], Vol. 2, Sect. 4.5.2). *The gcd satisfies the following recursive equation, using which it can be computed (for $x, y \geq 1$) using $O(\log x + \log y)$ steps:*

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

where $\mathrm{Parity}(x) = \mathrm{rem}(x, 2)$.

Proof. That the gcd satisfies these equations and is determined by them is trivial. To check the number of steps required, 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 will prove the following result, which establishes the optimality of the Stein algorithm (up to a multiplicative constant) from its natural givens:

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

$$c_\alpha(a, a^2 - 1) \geq \frac{1}{10}\log(a^2 - 1). \tag{7}$$

Problems for Section 1

x1.1. Show that if $\rho = \frac{1}{2}(1 + \sqrt{5})$ is the positive root of the quadratic equation $x + 1 = x^2$, then for all $k \geq 2$, $F_k \geq \rho^{k-2}$.

x1.2. Show that if $\rho = \frac{1+\sqrt{5}}{2}$ and $\lambda = \frac{1-\sqrt{5}}{2}$ are the two roots of the quadratic equation $x^2 = x + 1$, then for all k,

$$F_k = \frac{\rho^k - \lambda^k}{\sqrt{5}} \geq \frac{\rho^k}{\sqrt{5}} - 1.$$

Hint: Use induction on k for the equation, and infer the inquality from the fact that

$$\left|\frac{\lambda^k}{\sqrt{5}}\right| < 1.$$

x1.3. Show that successive Fibonacci numbers F_k, F_{k+1} with $k \geq 2$ are relatively prime, and $c_\varepsilon(F_{k+1}, F_k) = k - 1$.

x1.4. LAMÉ'S LEMMA. Show that if $y \leq F_k$ with $k \geq 2$, then, for every $x \geq y$, $c_\varepsilon(x, y) \leq k-1$. HINT: Use induction on $k \geq 2$, checking separately (by hand) the two basis cases $k = 2, 3$ and imitating the argument in the proof of Proposition 1.4.2.

Lamé's Lemma predicts the following upper bounds for $c_\varepsilon(x, y)$ for small values of y (and any $x \geq y$):

Values of y	$c_\varepsilon(x, y)$
1	1
2	2
3	3
4 - 5	4
6 - 8	5
9 - 13	6

These are a bit better than the simple $2 \log y$ bound. The next two problems clarify the situation (but require some arithmetic).

x1.5. Show that if $x \geq y \geq 2$, then

$$c_\varepsilon(x, y) \leq \frac{\log(\sqrt{5}y)}{\log \rho},$$

where ρ is the positive root of $x + 1 = x^2$. HINT: Use Lamé's Lemma and the inequality in Problem x1.2.

x1.6. Show that for all real numbers $y \geq 16$, $\dfrac{\log(\sqrt{5}y)}{\log \rho} < 2 \log y$. HINT: Check the inequality by hand for $y = 16$, and then check that the function

$$f(y) = 2 \log y - \frac{\log(\sqrt{5}y)}{\log \rho}$$

is increasing for $y > 0$.

x1.7 (Bezout's Lemma). Show that for all natural numbers $x, y \geq 1$, there exist integers $\alpha, \beta \in \mathbb{Z}$ such that

$$\gcd(x, y) = \alpha x + \beta y.$$

In fact, we can set $\alpha = \alpha(x, y), \beta = \beta(x, y)$ where the functions

$$\alpha, \beta : \mathbb{N} \times \mathbb{N} \to \mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$$

satisfy the following system of recursion equations, for $x \geq y \geq 1$:

$$\begin{aligned} \alpha(x, y) = &\text{ if } (y \mid x) \text{ then } 0 \text{ else } \beta(y, \operatorname{rem}(x, y)), \\ \beta(x, y) = &\text{ if } (y \mid x) \text{ then } 1 \\ &\text{ else } \alpha(y, \operatorname{rem}(x, y)) - \operatorname{iq}(x, y)\beta(y, \operatorname{rem}(x, y)). \end{aligned}$$

Use this recursion to express $\gcd(231, 165)$ as an integer, linear combination of 231 and 165.

x1.8. Show that two numbers $x, y \geq 1$ are coprime if and only if there exist integers $\alpha, \beta \in \mathbb{Z}$ such that $1 = \alpha x + \beta y$.

x1.9. For positive numbers, show: if $x \perp a$ and $x \mid ab$, then $x \mid b$.

x1.10. Show that for all $x \geq y \geq 1$, there are infinitely many choices of rational integers α and β such that

$$\gcd(x, y) = \alpha x + \beta y,$$

but only one choice such that $0 \leq \alpha < \frac{y}{d}$, if $d = \gcd(x, y)$.

x1.11. Show that the following strengthening of the Main Conjecture fails: *For every algorithm α from the remainder operation which computes the function* $\gcd(x, y)$*, there is a (positive, rational) constant r such that for all sufficiently large pairs $x \geq y \geq 1$,*

$$c_\alpha(x, y) \geq r \log y.$$

("For all sufficiently large pairs" means "for all $x \geq y \geq K$", where K is specified in advance.)

§2. (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

$$\mathbf{A} = (A, 0, 1, \mathbf{\Phi}) = (A, 0, 1, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi}), \tag{8}$$

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

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

is a partial function of some arity n associated by the *signature* Φ with the symbol ϕ. When these conditions hold, we will refer to $\mathbf{A}$ as a Φ-algebra.

Typical examples of algebras are the basic structures of arithmetic

$$\mathbf{N}_u = (\mathbb{N}, 0, 1, S, \mathrm{Pd}), \quad \mathbf{N}_b = (\mathbb{N}, 0, 1, \mathrm{parity}, \mathrm{iq}_2, \mathrm{em}_2, \mathrm{om}_2) \tag{9}$$

which represent the natural numbers "given" in unary and binary notation. Here $S(x) = x+1, \mathrm{Pd}(x) = x \dot{-} 1$ are the operations of successor and predecessor, and

$$\mathrm{parity}(x) = \mathrm{rem}(x, 2),\ \mathrm{iq}_2(x) = \mathrm{iq}(x, 2),\ \mathrm{em}_2(x) = 2x,\ \mathrm{om}_2(x) = 2x + 1 \tag{10}$$

are the less familiar operations which are the natural primitives on "binary numbers". (We read $\mathrm{em}_2(x)$ and $\mathrm{om}_2(x)$ as *even* and *odd* multiplication by 2.) These are *total algebras*, as is the standard (in model theory) structure on the natural numbers $\mathbf{N} = (\mathbb{N}, 0, 1, =, +, \times)$ on $\mathbb{N}$. Genuinely partial algebras typically arise as *restrictions* of total algebras, often to finite sets: if $\{0, 1\} \subseteq B \subseteq A$, then

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

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

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

An *expansion* of an algebra $\mathbf{A}$ is obtained by adding partial functions on A to $\mathbf{A}$,

$$(\mathbf{A}, \mathbf{\Psi}) = (A, 0, 1, \mathbf{\Phi} \cup \mathbf{\Psi}).$$

2.1. 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

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

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

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

2.2. Terms with parameters. It is also convenient to introduce for each set A, the $\Phi[A]$-*terms* in which we allow members of A as parameters:

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

The members of A which occur in a $\Phi[A]$-term E are its *parameters*, and a $\Phi[A]$-term is *pure* if it has no parameters (i.e., if it is a Φ-term).

If we use the familiar convention of naming terms as $E(\mathsf{x}_1, \dots, \mathsf{x}_n)$[1] to indicate that all the variables which occur in E are in the list $\mathsf{x}_1, \dots, \mathsf{x}_n$, and $M_1, \dots, M_n$ are $\Phi[A]$-terms, then $E(M_1, \dots, M_n)$ is the result of replacing each x_i by M_i; in particular, if $x_1, \dots, x_n \in A$, then $E(x_1, \dots, x_n)$ is the closed $\Phi[A]$-term resulting by replacing each x_i by x_i.

A term is *closed* if no variables occur in it.

2.3. Semantics. For a given partial algebra $\mathbf{A}$ with universe A, we define the partial function den : $\{\text{closed } \Phi[A]\text{-terms}\} \rightharpoonup A$ by the following recursive clauses:

$$\mathrm{den}(\mathsf{0}) = 0, \quad \mathrm{den}(\mathsf{1}) = 1, \quad \mathrm{den}(x) = x$$
$$\mathrm{den}(\phi(E_1, \dots, E_n)) = \phi^{\mathbf{A}}(\mathrm{den}(E_1), \dots, \mathrm{den}(E_n))$$
$$\mathrm{den}(\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2) = \begin{cases} \mathrm{den}(E_1), & \text{if } \mathrm{den}(E_0) = 0, \\ \mathrm{den}(E_2), & \text{if } \mathrm{den}(E_0)\downarrow \;\; \& \; \mathrm{den}(E_0) \neq 0. \end{cases}$$

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

$$\mathbf{A} \models E = w \iff \mathrm{den}(\mathbf{A}, E) = w.$$

If $E \equiv E(\vec{\mathsf{x}})$ is a term which may not be closed and σ is any *assignment* of values in A to the variables, then

$$\mathrm{den}(\mathbf{A}, E, \sigma) = \mathrm{den}(\mathbf{A}, E(\sigma(\mathsf{v}_1), \dots, \sigma(\mathsf{v}_n)),$$

and in the model theoretic notation, for any E,

$$\mathbf{A}, \sigma \models E = w \iff \mathrm{den}(\mathbf{A}, E, \sigma) = w.$$

[1]Formally, $E(\mathsf{x}_1, \dots, \mathsf{x}_n) \equiv (E, (\mathsf{x}_1, \dots, \mathsf{x}_n))$ is a pair of a term E and a list of variables which includes all the free variables of E, but it is traditional in logic to abuse terminology and call these somewhat more complex objects "terms".

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,

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

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

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

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

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

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

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

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

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

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

$$\mathbf{B} \models E(\vec{x}) = w \implies \mathbf{A} \models E(\pi(\vec{x})) = \pi(w), \tag{11}$$

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

2.5. 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 an algebra $\mathbf{A}$ in the obvious way:

$$\begin{aligned} G_0(X) &= \{0, 1\} \cup X, \\ G_{m+1}(X) &= G_m(X) \cup \{\phi^{\mathbf{A}}(\vec{x}) \mid \vec{x} \in G_m(X), \phi \in \Phi, \phi^{\mathbf{A}}(\vec{x})\downarrow\}. \end{aligned} \tag{12}$$

Clearly

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

and by induction on k, for all m

$$G_k(G_m(X)) = G_{m+k}(X) \tag{13}$$

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

Problems for Section 2

x2.1. Give an example of two terms $E_1(\mathsf{x})$ and $E_2(\mathsf{x})$ with one free variable, such that for every $x \in A$, $\mathrm{den}(E_1(x)) = \mathrm{den}(E_2(x))$, but if M is closed and $\mathrm{den}(M)\uparrow$, then $\mathrm{den}(E_1(M)) \neq \mathrm{den}(E_2(M))$.

x2.2. Show that for every term $E(\mathsf{x})$ and closed term M,

$$\mathrm{den}(M) = w \Longrightarrow \mathrm{den}(E(M)) = \mathrm{den}(E(w)).$$

x2.3. Show that if $\mathrm{den}(E_1(x)) = \mathrm{den}(E_2(x))$ for every $x \in A$, and M is closed and $\mathrm{den}(M)\downarrow$, then

$$\mathrm{den}(E_1(M)) = \mathrm{den}(E_2(M)).$$

These results extend trivially to simultaneous substitutions.

x2.4. Suppose $E(\vec{\mathsf{v}})$ is a term with all its free variables in the list $\vec{\mathsf{v}}$ which defines the partial function

$$f(\vec{x}) = \mathrm{den}(E(\vec{x})) \tag{14}$$

in a partial algebra $\mathbf{A}$. Associate with each term $M(\vec{\mathsf{y}})$ (possibly with parameters) in the signature $\Phi \cup \{f\}$ extended by f, a term $M^*(\vec{\mathsf{y}})$ in the original signature Φ, with the same free variables, and such that if M is closed, then

$$\mathrm{den}(M) = \mathrm{den}(M^*).$$

x2.5. Show that if $\pi : \mathbf{A} \rightarrowtail \mathbf{B}$ is an imbedding of one partial algebra into another (with the same signature), then for every Φ-term $M(\vec{\mathsf{x}})$ and all $\vec{x} \in A$,

$$\text{if } \mathrm{den}_{\mathbf{A}}(M(\vec{x}))\downarrow, \text{ then } \pi(\mathrm{den}_{\mathbf{A}}(M(\vec{x})) = \mathrm{den}_{\mathbf{B}}(M(\pi(\vec{x})))$$

where, naturally,

$$\pi(x_1, \dots, x_n) = (\pi(x_1), \dots, \pi(x_n)).$$

The *explicit* (or *algebraic*) Φ-terms are those Φ-terms in which no conditional occurs, i.e., recursively

$$\text{(Explicit } \Phi\text{-terms)} \qquad E :\equiv \mathsf{0} \mid \mathsf{1} \mid \mathsf{v}_i \mid \phi(E_1, \dots, E_n),$$

and the *depth* of each explicit term is defined by the corresponding recursion

$$\mathrm{depth}(0) = \mathrm{depth}(1) = \mathrm{depth}(\mathsf{v}_i) = 0$$
$$\mathrm{depth}(\phi(E_1, \dots, E_n)) = \max\{\mathrm{depth}(E_1), \dots, \mathrm{depth}(E_n)\} + 1.$$

x2.6. For each Φ-algebra $\mathbf{A}$, all $m \geq 0$ and all $X \subseteq A$,

$$G_m(X) = \{\mathrm{den}(E(x_1, \dots, x_n)) \mid x_1, \dots, x_n \in X \ \&\ \mathrm{den}(E(x_1, \dots, x_n))\downarrow \\ \&\ \mathrm{depth}(E(\mathsf{x}_1, \dots, \mathsf{x}_n)) \leq m\}.$$

x2.7. Show that for each finite signature Φ, each Φ-algebra $\mathbf{A}$ and each finite $X \subseteq A$, there is a number a such that for all m,

$$|G_m(X)| \leq C^{2^{am}} \qquad (C = |X| + 2).$$

Give an example where $|G_m(X)|$ cannot be bounded by a single exponential in m.

§3. Recursive (McCarthy) programs. A (formal) *recursive program* on the signature Φ is a system of recursive equations

$$(15)\qquad \alpha : \quad \begin{cases} \mathsf{f}_\alpha(\vec{\mathsf{x}}) = \mathsf{f}_0(\vec{\mathsf{x}}_0) &= E_0(\vec{\mathsf{x}}_0, \mathsf{f}_1, \dots, \mathsf{f}_K), \\ \mathsf{f}_1(\vec{\mathsf{x}}_1) &= E_1(\vec{\mathsf{x}}_1, \mathsf{f}_1, \dots, \mathsf{f}_K), \\ &\vdots \\ \mathsf{f}_K(\vec{\mathsf{x}}_K) &= E_K(\vec{\mathsf{x}}_K, \mathsf{f}_1, \dots, \mathsf{f}_K), \end{cases}$$

where $\mathsf{f}_0 \equiv \mathsf{f}_\alpha, \mathsf{f}_1, \dots, \mathsf{f}_K$ are distinct function variables, the *recursion variables* of α; each $E_i(\vec{\mathsf{x}}_i, \mathsf{f}_1, \dots, \mathsf{f}_K)$ is a term in the (extended) *program signature*

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

in which only the indicated individual and function variables may occur; and the arities of the function variables f_i are such that these equations make sense. The term E_0 is the *head* of α, and it may be its only *part*, since we allow $K = 0$. If $K > 0$, then the remaining parts $E_1, \dots, E_K$ comprise the *body* of α. The *arity* of α is the number of variables in the head symbol f_0; the *total arity* of α is the largest arity of all function constants in Φ and function variables of α. If $\mathbf{A}$ is a Φ-algebra, then Φ-programs are also called $\mathbf{A}$-programs.

Sometimes it is convenient to think of programs as (extended) *program terms* and rewrite (15) in the form

$$(16)\qquad \alpha \equiv E_0 \ \mathsf{where}\ \{\mathsf{f}_1 = \lambda(\vec{\mathsf{x}}_1)E_1, \dots, \mathsf{f}_K = \lambda(\vec{\mathsf{x}}_K)E_K\},$$

in an expanded language with a symbol for the (abstraction) λ-operator and mutual recursion. This makes it clear that the function variables $\mathsf{f}_1, \dots, \mathsf{f}_K$ and the occurrences of the individual variables $\vec{\mathsf{x}}_i$ in E_i $(i > 0)$ are *bound* in the program α, and that the putative "head variable" f_α does not actually occur in α. An individual variable z is free in α if it occurs in E_0.[2]

If $K = 0$ in this definition, then the program is just a Φ-term (with no recursion variables). On the other hand, algorithms are often expressed by a single recursive equation, e.g.,

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

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

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

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

An example of a recursive program with two equations in the body is the formal version of the system which computes the Bezout representation of $\gcd(x, y)$ in Problem x1.7, suitably formalized and augmented with a head

$$\mathsf{f}_0(x, y) = \alpha(x, y) \text{ or } \mathsf{f}_0(x, y) = \beta(x, y),$$

depending on which of the two functions α or β we want the program to compute. (The relevant algebra has universe $\mathbb{Z}$.)

[2]The notation (16) does not specify a function variable $\mathsf{f}_\alpha \equiv \mathsf{f}_0$, but it is convenient to assume that each recursive program α has such an associated *head symbol*, with arity the arity of α.

3.1. Semantics of recursive programs. Fix a recursive program α on the signature Φ, and a Φ-algebra $\mathbf{A}$. For any sig(α)-term $E(\vec{\mathsf{x}}, \mathsf{f}_0, \dots, \mathsf{f}_K)$, let

$$F_E(\vec{x}, f_0, \dots, f_K) = \mathrm{den}(\mathbf{A}, E(\vec{x}, f_0, \dots, f_K)), \tag{17}$$

where the replacement of the symbols $\mathsf{f}_0, \dots, \mathsf{f}_K$ by the partial functions $f_0, \dots, f_K$ signifies that we are computing the denotation in the sig(α)-algebra

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

3.1.1. Lemma. *For each term $E(\vec{\mathsf{x}}, \mathsf{f}_0, \dots, \mathsf{f}_K)$, the functional F_E defined by* (17) *is monotone and continuous.*

Proof is very easy, by induction on the term E. For example, if $E \equiv \mathsf{x}$, then $F_E(x, f_0, \dots, f_K) = x$ is independent of its partial function arguments, and so trivially monotone and continuous. If $E \equiv \phi(E_1, \dots, E_n)$ with $\phi \in \Phi$, then

$$F_E(\vec{x}, f_0, \dots, f_K) = \phi^{\mathbf{A}}(F_{E_1}(\vec{x}, f_0, \dots, f_K), \dots, F_{E_n}(\vec{x}, f_0, \dots, f_K)),$$

and the monotonicity and continuity of the F_{E_i} imply the same properties for F_E; the argument for the conditional is similar; and finally, if $E \equiv \mathsf{f}_i(E_1, \dots, E_n)$, then to compute

$$F_E(\vec{x}, f_0, \dots, f_K) = f_i(F_{E_1}(\vec{x}, f_0, \dots, f_K), \dots, F_{E_n}(\vec{x}, f_0, \dots, f_K))$$

we need just one value of f_i in addition to those needed for the computation of the parts $F_{E_i}(\vec{x}, f_0, \dots, f_K)$. ⊣

It now follows by the Fixed Point Lemma 1.1.1 that the system of recursive equations

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

defined by α has a canonical, least solution tuple

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

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

$$\overline{\alpha} = \overline{\alpha}^{\mathbf{A}} = \overline{f}_\alpha.$$

Notice that if $\alpha \equiv E_0(\vec{\mathsf{x}})$ is a Φ-term, then this definition yields

$$\overline{\alpha}(\vec{x}) = \mathrm{den}^{\mathbf{A}}(E_0(\vec{x})),$$

in agreement with the usual semantics of explicit terms on the algebra $\mathbf{A}$.

3.2. A-recursive functions. A partial function $f : A^n \rightharpoonup A$ is **A**-*recursive* (or recursive in the givens $\Phi = \{\phi^{\mathbf{A}} \mid \phi \in \Phi\}$ of **A**) if it is computed by some recursive program in **A**. We let

$$\mathbf{rec}(\mathbf{A}) = \text{the set of } \mathbf{A}\text{-recursive partial functions.}$$

The classical example is the (total) algebra $\mathbf{N}_u = (\mathbb{N}, 0, 1, S, \mathrm{Pd})$ of unary arithmetic whose recursive partial functions are exactly the Turing computable partial functions. This elegant characterization of Turing computability is due to [10], and so recursive programs are also called *McCarthy programs.* (The $\mathbf{N}_b$-recursive and $(\mathbb{N}, 0, 1, +, \cdot)$-recursive partial functions are also the Turing computable partial functions, cf. Problem x3.7.)

3.3. Simple fixed points. One interesting aspect of the definition is the use of *systems* rather than single recursive equations. A partial function $f : A^n \rightharpoonup A$ is a *simple fixed point* of **A** if it is the least solution of an equation

$$f(\vec{x}) = \mathrm{den}(\mathbf{A}, E(\vec{x}, f)) \tag{18}$$

for some (Φ, f)-term E. One might think that every **A**-recursive function f is a fixed point, but this is not true:

3.3.1. Proposition. *Let* $\mathbf{A} = (\mathbb{N}, 0, 1, S, \mathrm{Pd}, \phi_1, \ldots, \phi_k)$*, where* $\phi_1, \ldots, \phi_k$ *is any sequence of total, Turing-computable functions on* $\mathbb{N}$ (of any arities)*. There exists a total function* $f : \mathbb{N} \to \mathbb{N}$ *which is Turing computable* (and hence **A**-recursive, by McCarthy's characterization)*, but not a simple fixed point of* **A**.

A proof of this is outlined in problems x3.8 - x3.10.

Gregory McColm has also shown that *multiplication is not a simple fixed point of* $(\mathbb{N}, 0, 1, S, \mathrm{Pd})$, along with several other results in this classical case. The general problem of characterizing the *simple fixed points* of a partial algebra **A** is largely open, and it is one of many open problems in *First Order Abstract Recursion*, the study of the class of recursive partial functions on an arbitrary partial algebra **A**. This is an interesting subject, but not centrally related to our concerns here, and so we will confine ourselves to a few results in the problems.

Problems for Section 3

To solve a recursive equation (or a system of recursive equations) means to identify its canonical solution in explicit terms. For example, the solution of

$$f(x, y) = \text{if } (y = 0) \text{ then } x \text{ else } S(f(x, \mathrm{Pd}(y)))$$

in $\mathbb{N}$ (i.e., where the variables range over $\mathbb{N}$) is $\overline{f}(x, y) = x + y$, since addition satisfies the equation and it is total, so it must be exactly the (unique) least solution. Thus addition is a simple fixed point of $\mathbf{N}_u$. In the problems which follow, individual variables vary over $\mathbb{N}$, and function variables vary over partial functions on $\mathbb{N}$ (of various arities).

x3.1. Solve the recursive equation

$$f(x, y) = \text{if } (y = 0) \text{ then } 0 \text{ else } f(x, \mathrm{Pd}(y)) + x$$

in the expansion $(\mathbf{N}_u, +)$ of $\mathbf{N}_u$ by addition.

x3.2. Solve the recursive equation

$$\begin{aligned} f(x,y) = {} & \text{if } (y=0) \text{ then } 0 \\ & \text{else if } (y=1) \text{ then } x \\ & \text{else } 2\cdot f(x,\mathrm{iq}(y,2)) + f(x,\mathrm{parity}(y)) \end{aligned}$$

in $(\mathbf{N}_b, +)$.

x3.3. Consider the following recursive equation:

$$f(x,y,r) = \begin{cases} r, & \text{if } x=y=0, \\ 2f(\mathrm{iq}_2(x),\mathrm{iq}_2(y),0), & \text{ow., if } \mathrm{parity}(x)+\mathrm{parity}(y)+r=0, \\ 2f(\mathrm{iq}_2(x),\mathrm{iq}_2(y),0)+1, & \text{ow., if } \mathrm{parity}(x)+\mathrm{parity}(y)+r=1, \\ 2f(\mathrm{iq}_2(x),\mathrm{iq}_2(y),1), & \text{ow., if } \mathrm{parity}(x)+\mathrm{parity}(y)+r=2, \\ 2f(\mathrm{iq}_2(x),\mathrm{iq}_2(y),1)+1, & \text{ow.} \end{cases}$$

Argue that the expression on the right can be defined by a term $E(x,y,r,f)$ (in the signature) of the algebra $\mathbf{N}_b$, and find the (partial) function $\overline{\alpha}(x,y)$ computed in $\mathbf{N}_b$ by the program

$$\alpha(x,y) = f(x,y,0) \text{ where } \{f(x,y,r) = E(x,y,r,f)\}.$$

x3.4. Solve the recursive equation

$$f(x,y) = \text{if } (\phi(x,y)=0) \text{ then } y \text{ else } f(x,y+1)$$

in the partial algebra $(\mathbb{N}, 0, 1, \phi, S)$.

x3.5. Solve the recursive equation

$$f(x,y) = \text{if } (x=0) \text{ then } 1 \text{ else } f(\mathrm{Pd}(x), f(x,y))$$

in $\mathbf{N}_u$.

x3.6. Show that if f is $\mathbf{A}$-recursive and g is $(\mathbf{A}, f)$-recursive, then g is $\mathbf{A}$-recursive.

x3.7. Show that $\mathbf{rec}(\mathbf{N}_u) = \mathbf{rec}(\mathbf{N}_b) = \mathbf{rec}(\mathbb{N}, 0, 1, +, \cdot)$.

The remaining problems in this section lead to a proof of Proposition 3.3.1, and they are not needed for the sequel.

x3.8. Suppose $F(x,f)$ is a monotone, continuous functional whose fixed point $\overline{f}$ is a total function, and let

$$\mathrm{stage}(x) = \mathrm{stage}_F(x) = \text{the least } k \text{ such that } \overline{f}^k(x)\downarrow -1 \tag{19}$$

in the notation of Lemma 1.1.1. Show that for infinitely many x,

$$\mathrm{stage}(x) \le x.$$

x3.9. Suppose $\psi : \mathbb{N} \to \mathbb{N}$ is strictly increasing, i.e.,

$$x < y \Longrightarrow \psi(x) < \psi(y),$$

and let by recursion,

$$\begin{aligned} \psi^{(0)}(x) &= x, \\ \psi^{(n+1)}(x) &= \psi(\psi^{(n)}(x)). \end{aligned}$$

A unary partial function f is *n-bounded* (relative to ψ, for $n > 0$) if

$$f(x)\downarrow \implies f(x) \leq \psi^{(n)}(x);$$

and a functional $F(x, f)$ (with f a variable over unary partial functions) is *ℓ-bounded* (relative to ψ), if for all f and $n \geq 1$,

$$\text{if } f \text{ is } n\text{-bounded, then for all } x,\ F(x, f) \leq \psi^{(\ell n)}(x).$$

Suppose $\mathbf{A} = (\mathbb{N}, 0, 1, \{\phi^{\mathbf{A}}\}_{\phi\in\Phi})$ is a structure such that Φ is finite, and every given $\phi^{\mathbf{A}}$ is total and bounded by some fixed ψ as above, in the sense that

$$\phi(\vec{x}) \leq \psi(\max \vec{x}).$$

Prove that for every term $E(\mathsf{x}, f)$ in the signature $\Phi \cup \{f\}$, there is an ℓ such that the functional

$$F(x, f) = \mathrm{den}(E(x, f))$$

is ℓ-bounded. HINT: You will need to verify that $\psi(x) \geq x$, because ψ is increasing, and hence, for all ℓ, ℓ'

$$\ell \leq \ell' \Longrightarrow \psi^{(\ell)}(x) \leq \psi^{(\ell')}(x).$$

(This is also needed in the next problem.)

x3.10. Prove Proposition 3.3.1.

§4. Complexity measures. Suppose Π is a class of algorithms which compute partial functions $f : A^n \rightharpoonup A$ on some set A, most likely (in our case) those expressed by Φ-recursive programs on a Φ-algebra $\mathbf{A}$. In the most general terms, a *complexity measure* for Π associates with each n-ary $\alpha \in \Pi$ an n-ary partial function

$$\mathfrak{k}_\alpha : A^n \rightharpoonup \mathbb{N} \tag{20}$$

which intuitively assigns to each $\vec{x}$ such that $\overline{\alpha}(\vec{x})\downarrow$ a *cost* of some sort of the computation of $\overline{\alpha}(\vec{x})$ by α. We will introduce five such basic complexity measures in this section.

4.1. The tree-depth complexity $D(M)$. Fix a Φ-algebra $\mathbf{A}$ and a Φ-program α, 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)

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

This is the value of M when the recursive variables are interpreted by the canonical fixed points of the program, so that, for example,

$$\overline{\mathsf{f}_0(x)} = \bar{\mathsf{f}}_0(x) = \overline{\alpha}(x) \quad (x \in A).$$

If $\overline{M}\downarrow$, we can associate with M a finite *computation tree*, whose nodes are labelled with term equations and which represents an abstract, parallel computation of $\overline{M}$, so that, for example, the *root* is labelled with

$$M = \overline{M}.$$

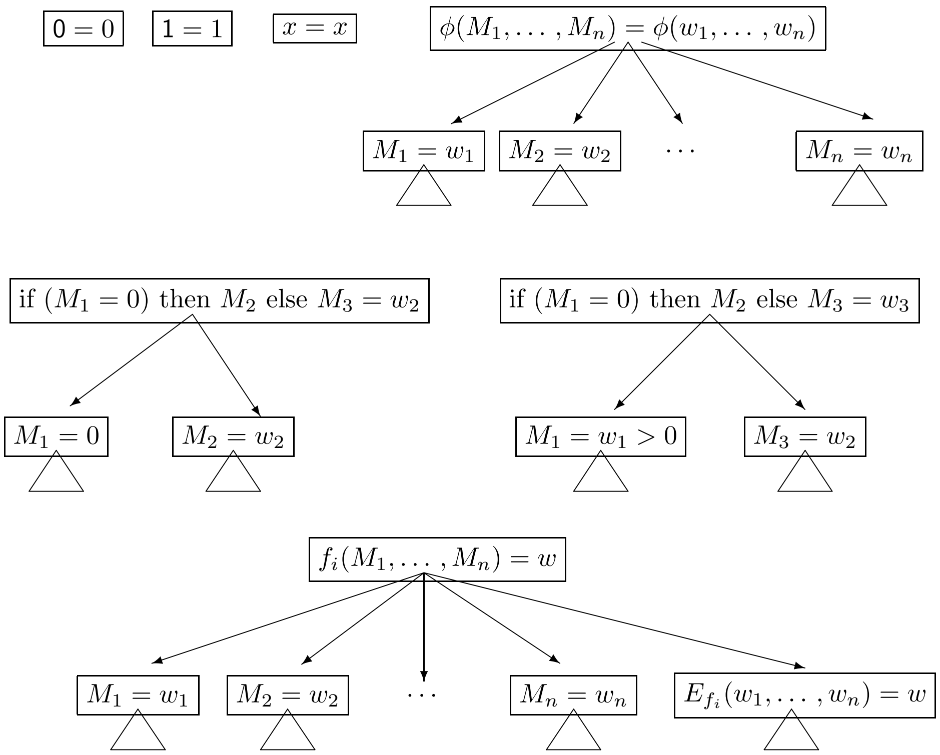

FIGURE 1. Computation trees.

These trees are illustrated in Figure 1. We leave for the problems the precise definitions, since they are not needed for the proofs—although they illuminate them, and they help explain the meaning of the results. Our main interest is in the function

$$D(M) = \text{the depth of the computation tree for } M \quad (\text{if } \overline{M}\downarrow),$$

which can be defined directly as follows.

4.1.1. LEMMA. *Fix a Φ-algebra $\mathbf{A}$ and a Φ-program α. There is exactly one function D defined on every closed* $\mathrm{sig}(\alpha)[A]$*-term M such that $\overline{M}\downarrow$, which satisfies the following conditions:*

(D1) $D(0) = D(1) = D(x) = 0$.

(D2) $D(\phi(M_1, \dots, M_n)) = \max\{D(M_1), \dots, D(M_n)\} + 1$.

(D3) *If f is a recursion variable of α, then*

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

where $d_{\mathsf{f}}(\vec{w}) = D(E_{\mathsf{f}}(\vec{w}, \mathsf{f}_1, \dots, \mathsf{f}_K))$.

(D4) *If* $M \equiv (\text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2)$*, then*

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \text{if } \overline{M}_0 = 0, \\ \max\{D(M_0), D(M_2)\} + 1, & \text{if } \overline{M}_0\downarrow \ \& \ \overline{M}_0 \neq 0. \end{cases}$$

Proof. If $\overline{M} = \text{den}(M(\vec{x}, \overline{f}_1, \dots, \overline{f}_K))\downarrow$, then there is some k such that

$$\overline{M}^k = \text{den}(M(\vec{x}, \overline{f}^k_1, \dots, \overline{f}^k_K))\downarrow,$$

by Lemma 3.1.1; we will define $D(M)$ by recursion on

$$\text{stage}(M) = \text{the least } k \text{ such that } \overline{M}^k\downarrow,$$

and recursion on the length of terms within this. We consider cases on the form of M.

(D1) If M is $\mathsf{0}, \mathsf{1}$ or a parameter x, then $D(M) = 0$.

(D2) If $M \equiv \phi(M_1, \dots, M_n)$ for some $\phi \in \Phi$ and $\overline{M}\downarrow$, then

$$\text{stage}(M) = \max\{\text{stage}(M_1), \dots, \text{stage}(M_n)\},$$

and these subterms are all smaller than M, so we may assume that $D(M_i)$ is defined for $i = 1, \dots, n$; we set

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

(D3) If $M \equiv \mathsf{f}(M_1, \dots, M_n)$ with a recursive symbol f an $\overline{M}\downarrow$ and $k = \text{stage}(M)$, then

$$\overline{M}^k = \overline{\mathsf{f}}^k(\overline{M}^k_1, \dots, \overline{M}^k_n),$$

and so $\text{stage}(M_i) \leq k$ and we can assume that $D(M_i)$ is defined for $i = 1, \dots, n$, since these terms are smaller than M. Moreover, if $\overline{M}_1 = w_1, \dots, \overline{M}_n = w_n$, then

$$\overline{\mathsf{f}}^k(w_1, \dots, w_n) = \text{den}(E_{\mathsf{f}}(w_1, \dots, w_n, \overline{f}^{k-1}_1, \dots, \overline{f}^{k-1}_K))\downarrow,$$

by the definition of the iterates in the proof of Lemma 1.1.1, and so

$$\text{stage}(E_f(w_1, \dots, w_n, f_1, \dots, f_K)) < k;$$

thus we may assume that $D(E_f(w_1, \dots, w_n, f_1, \dots, f_K))$ is defined, and define $D(M)$ so that (D3) in the Lemma holds.

The case (D4) for conditional terms is handled similarly, and the uniqueness of D is proved by a simple induction on $\text{stage}(M)$, following the definition. ⊣

It is convenient to have $D(M)$ defined on all closed $\text{sig}(\alpha)[A]$-terms by setting

$$D(M) = \infty \quad (\overline{M}\uparrow),$$

where $n < \infty$ for every $n \in \mathbb{N}$. The *tree-depth complexity* of a program α is that of its head symbol,

$$d_\alpha(\vec{x}) = D(\mathsf{f}_\alpha(\vec{x})) = D(E_0(\vec{x}, \mathsf{f}_1, \dots, \mathsf{f}_K)).$$

We can think of $D(M)$ as measuring the "depth" of the computation pictured by the associated tree, but these computations are very abstract indeed: for example, to attain

$$D(f(M)) = \max\{D(M), D(E_f(\overline{M}))\} + 1,$$

we are supposed to compute in parallel $\overline{M}$ and $\overline{f}(\overline{M})$, which means that we should start on $\overline{f}(\overline{M})$ before we have completed the computation of $\overline{M}$; this cannot be implemented, except in terms of an "infinitely parallel" computation, which computes $\overline{f}(x)$ for every x, simultaneously, and use only the value $\overline{f}(\overline{M})$ when we finally learn the value $\overline{M}$. In any case, our methods do not yield any interesting lower bounds for the complexity $D(M)$; but it is a very useful technical aid in the definition of other, more natural complexity measures, as follows.

4.2. The number-of-calls complexity $C^s(M)$. For a fixed Φ-algebra $\mathbf{A}$ and a Φ-program α, the *call complexity* $C^s(M)$ of each convergent $\mathrm{sig}(\alpha)[A]$-term M is defined by the following recursion on $D(M)$:

(C^s1) $C^s(\mathsf{0}) = C^s(\mathsf{1}) = C^s(x) = 0 \quad (x \in A)$.

(C^s2) $C^s(\phi(M_1, \dots , M_n)) = C^s(M_1) + \cdots + C^s(M_n) + 1$.

(C^s3) If f is a recursive variable of α, then

$$C^s(\mathsf{f}(M_1, \dots , M_n)) = C^s(M_1) + \cdots + C^s(M_n) + c^s_{\mathsf{f}}(\overline{M}_1, \dots , \overline{M}_n),$$

where $c^s_{\mathsf{f}}(\vec{x}) = C^s(E_{\mathsf{f}}(\vec{x}, \mathsf{f}_1, \dots , \mathsf{f}_K))$.

(C^s4) If $M \equiv (\text{if } (M_0 = \mathsf{0}) \text{ then } M_1 \text{ else } M_2)$, then

$$C^s(M) = \begin{cases} C^s(M_0) + C^s(M_1), & \text{if } \overline{M}_0 = 0,\\ C^s(M_0) + C^s(M_2), & \text{if } \overline{M}_0\downarrow \ \& \ \overline{M}_0 \neq 0. \end{cases}$$

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

$$(C^s) \qquad c^s_\alpha(\vec{x}) = c^s_{\mathsf{f}_0}(\vec{x}) = C^s(E_0(\vec{x}, \mathsf{f}_1, \dots , \mathsf{f}_K)).$$

When we need to make explicit the algebra or the program for which the calls complexity is computed, we will write

$$C^s(\mathbf{A}, \alpha, M) \text{ or } c^s_\alpha(\mathbf{A}, \vec{x}).$$

This is a very natural complexity measure: $C^s(M)$ counts the number of calls to the primitives which are required for the computation of $\overline{M}$ using "the algorithm expressed" by the program α, disregarding the "logical steps", tests for the conditional and recursive calls. It does not distinguish between parallel and sequential implementations of α, although it is probably more directly relevant to the second—so we will sometimes refer to it as the *sequential calls complexity*. Notice that α may (stupidly) call many times for the same value of one of the givens, and all these calls will be counted separately by $C^s(M)$; we will put off until Subsection 12.4 introducing a more sophisticated complexity measure which does not count duplicate calls.

4.3. The depth-of-calls complexity measure $C^p(M)$. For a fixed Φ-algebra $\mathbf{A}$ and a Φ-program α, the *depth-of-calls* complexity $C^p(M)$ of each convergent $\mathrm{sig}(\alpha)[A]$-term M is defined by the following recursion on $D(M)$:

(C^p1) $C^p(\mathsf{0}) = C^p(\mathsf{1}) = C^p(x) = 0 \quad (x \in A)$.

(C^p2) $C^p(\phi(M_1, \dots , M_n)) = \max\{C^p(M_1), \dots , C^p(M_n)\} + 1$.

(C^p3) If f is a recursive variable of α, then

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

where $c^p_{\mathsf{f}}(\vec{x}) = C^p(E_{\mathsf{f}}(\vec{x},\mathsf{f}_1,\dots,\mathsf{f}_K))$.

(C^p4) If $M \equiv (\text{if } (M_0 = \mathsf{0}) \text{ then } M_1 \text{ else } M_2)$, then

$$C^p(M) = \begin{cases} \max\{C^p(M_0), C^p(M_1)\}, & \text{if } \overline{M}_0 = 0, \\ \max\{C^p(M_0), C^p(M_2)\}, & \text{if } \overline{M}_0\!\downarrow\ \&\ \overline{M}_0 \neq 0. \end{cases}$$

We have again included the definition of the complexity $c^p_{\mathsf{f}}(\vec{x})$ of each of the mutual least fixed points of α, and the calls-depth complexity of α is that of its head symbol,

$$c^p_\alpha(\vec{x}) = c^p_{\mathsf{f}_0}(\vec{x}) = C^p(E_0(\vec{x},\mathsf{f}_1,\dots,\mathsf{f}_K)), \tag{C}$$

The number $C^p(M)$ counts *the maximal depth of nested calls to the givens* in the computation of $\overline{M}$ by the algorithm expressed by α, and it is more directly relevant to parallel implementations—which is why we will sometimes call it the *parallel calls complexity*. Like the tree depth complexity $D(M)$, this $C^p(M)$ cannot be attained by actual implementations, but (as we will see) it is majorized by the natural complexity functions of all reasonable implementations of recursive programs, and so lower bound results about it have wide applicability. Most of the lower bound results we will derive are for the parallel call complexity $C^p(M)$.

Next we turn to complexity measures which count the logical steps (tests for the conditional and recursive calls) as well as calls to the givens.

4.4. The sequential logical complexity $L^s(M)$. For a fixed Φ-algebra $\mathbf{A}$ and a Φ-program α, the *sequential logical complexity* $L^s(M)$ of each convergent sig(α)[A]-term M is defined by the following recursion on $D(M)$:

(L^s1) $L^s(\mathsf{0}) = L^s(\mathsf{1}) = L^s(x) = 0 \quad (x \in A)$.

(L^s2) $L^s(\phi(M_1,\dots,M_n)) = L^s(M_1) + L^s(M_2) + \cdots + L^s(M_n) + 1$.

(L^s3) If f is a recursive variable of α, then

$$L^s(\mathsf{f}(M_1,\dots,M_n)) = L^s(M_1) + \cdots + L^s(M_n) + l^s_{\mathsf{f}}(\overline{M}_1,\dots,\overline{M}_n) + 1$$

where $l^s_{\mathsf{f}}(\vec{x}) = L^s(E_{\mathsf{f}}(\vec{x},\mathsf{f}_1,\dots,\mathsf{f}_K))$.

(L^s4) If $M \equiv (\text{if } (M_0 = \mathsf{0}) \text{ then } M_1 \text{ else } M_2)$, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = 0, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0\!\downarrow\ \&\ \overline{M}_0 \neq 0. \end{cases}$$

The sequential logical complexity

$$l^s_\alpha(\vec{x}) = l^s_{\mathsf{f}_0}(\vec{x}) = L^s(E_0(\vec{x}))$$

counts the number of steps that a sequential (call-by-value) implementation of the program would execute, and is, in some sense, the most natural (realistic) time complexity measure for recursive programs.

4.5. The parallel logical complexity $L^p(M)$. For a fixed Φ-algebra $\mathbf{A}$ and a Φ-program α, the *parallel logical complexity* $L^p(M)$ of each convergent $\mathrm{sig}(\alpha)[A]$-term M is defined by the following recursion on $D(M)$:

(L^p1) $L^p(\mathsf{0}) = L^p(\mathsf{1}) = L^p(x) = 0 \quad (x \in A)$.
(L^p2) $L^p(\phi(M_1, \dots, M_n)) = \max\{L^p(M_1), \dots, L^p(M_n)\} + 1$.
(L^p3) If f is a recursive variable of α, then

$$L^p(\mathsf{f}(M_1, \dots, M_n)) = \max\{L^p(M_1), \dots, L^p(M_n)\} + l^p_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n) + 1,$$

where $l^p_{\mathsf{f}}(\vec{x}) = L^p(E_{\mathsf{f}}(\vec{x}, \mathsf{f}_1, \dots, \mathsf{f}_K))$.
(L^p4) If $M \equiv (\text{if } (M_0 = \mathsf{0}) \text{ then } M_1 \text{ else } M_2)$, then

$$L^p(M) = \begin{cases} \max\{L^p(M_0), L^p(M_1)\} + 1, & \text{if } \overline{M}_0 = 0, \\ \max\{L^p(M_0), L^p(M_2)\} + 1, & \text{if } \overline{M}_0\!\downarrow \ \& \ \overline{M}_0 \neq 0. \end{cases}$$

The parallel logical complexity

$$l^p_\alpha(\vec{x}) = l^p_{\mathsf{f}_0}(\vec{x}) = L^p(E_0(\vec{x}))$$

measures time in an abstract, call-by-value, parallel implementation of the program.

4.5.1. Proposition. *For each Φ-algebra $\mathbf{A}$, each Φ-program α and each closed, convergent $\Phi[A]$-term M:*

$$\begin{array}{ccccc} & & C^s(M) & & \\ & \rotatebox{45}{$\leq$} & & \rotatebox{-45}{$\geq$} & \\ C^p(M) & & & & L^s(M), \\ & \rotatebox{-45}{$\geq$} & & \rotatebox{45}{$\leq$} & \\ & & L^p(M) & & \end{array}$$

and, in particular, for all $\vec{x}$ such that $\overline{\alpha}(\vec{x})\!\downarrow$,

$$\begin{array}{ccccc} & & c^s_\alpha(\vec{x}) & & \\ & \rotatebox{45}{$\leq$} & & \rotatebox{-45}{$\geq$} & \\ c^p_\alpha(\vec{x}) & & & & l^s_\alpha(\vec{x}). \\ & \rotatebox{-45}{$\geq$} & & \rotatebox{45}{$\leq$} & \\ & & l^p_\alpha(\vec{x}) & & \end{array}$$

Proof. All four claimed inequalities are very easy to check, by induction on $D(M)$. $\dashv$

The difference between $l^s_\alpha(\vec{x})$ and $l^p_\alpha(\vec{x})$ measures (in some vague sense) how "parallel" the algorithm expressed by α is and it is no more than exponential by the next result—although the base of the exponential interestingly depends on the program α and not just on the givens of the structure.

Recall that the *total arity* of a Φ-program α is the largest arity of all the function symbols in $\mathrm{sig}(\alpha)$, i.e.,

$$\ell(\alpha) = \max\{\max\{\mathrm{arity}(\phi) \mid \phi \in \Phi\}, \mathrm{arity}(\mathsf{f}_0), \dots, \mathrm{arity}(\mathsf{f}_K)\}. \tag{21}$$

4.5.2. PROPOSITION. *For each Φ-algebra* $\mathbf{A}$, *each Φ-program α of total arity $\ell \geq 2$,*[3] *and each closed, convergent $\Phi[A]$-term M:*

$$L^s(M) \leq (\ell+1)^{L^p(M)}, \tag{22}$$

and hence, for all $\vec{x}$ such that $\overline{\alpha}(\vec{x})\downarrow$,

$$l^s_\alpha(\vec{x}) \leq (\ell+1)^{l^p_\alpha(\vec{x})}.$$

PROOF is by induction on $D(M)$, and the result is trivial when M is 0, 1 or a variable, so that $L^p(M) = L^s(M) = 0$.

If $M \equiv \phi(M_1, \dots, M_n)$, the induction hypothesis gives us the result for $M_1, \dots, M_n$, and we compute:

$$\begin{aligned}
L^s(M) &= L^s(M_1) + \cdots + L^s(M_n) + 1\\
&\leq (\ell+1)^{L^p(M_1)} + \cdots + (\ell+1)^{L^p(M_n)} + 1\\
&\leq \ell(\ell+1)^A + 1 \qquad\qquad (A = \max\{L^p(M_1), \dots, L^p(M_n)\})\\
&\leq \ell(\ell+1)^A + (\ell+1)^A\\
&= (\ell+1)^{A+1} = (\ell+1)^{L^p(M)}.
\end{aligned}$$

The argument is the same when M is a conditional, except that we need to use the hypothesis $\ell \geq 2$ because $2(\ell+1)^A$ shows up where $\ell(\ell+1)^A$ occurs in this computation.

Finally, when $M \equiv f_i(M_1, \dots, M_n)$ with one of the recursion variables of the program, then

$$L^s(M) = L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \dots, \overline{M}_n)) + 1.$$

If $L^p(M_1) = \cdots = L^p(M_n) = 0$, then each M_i is 0, 1 or a variable, so their sequential logical complexities are also $= 0$, and then, using the induction hypothesis:

$$\begin{aligned}
L^s(M) &= L^s(E_i(\overline{M}_1, \dots, \overline{M}_n)) + 1\\
&\leq (\ell+1)^{L^p(E_i(\overline{M}_1, \dots, \overline{M}_n))} + 1 \leq (\ell+1)^{L^p(E_i(\overline{M}_1, \dots, \overline{M}_n))+1} = (\ell+1)^{L^p(M)}.
\end{aligned}$$

In the opposite case, setting

$$A = \max\{L^p(M_1), \dots, L^p(M_n)\} \geq 1, \quad B = L^p(E_i(\overline{M}_1, \dots, \overline{M}_n)),$$

we can compute as above:

$$\begin{aligned}
L^s(M) &= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_i(\overline{M}_1, \dots, \overline{M}_n)) + 1\\
&\leq (\ell+1)^{L^p(M_1)} + \cdots + (\ell+1)^{L^p(M_n)} + (\ell+1)^B + 1\\
&\leq \ell(\ell+1)^A + (\ell+1)^{B+1},
\end{aligned}$$

and it sufficed to prove that for all $A \geq 1$ and all $B \in \mathbb{N}$,

$$\ell(\ell+1)^A + (\ell+1)^{B+1} \leq (\ell+1)^{A+B+1};$$

but this inequality is equivalent to

$$\frac{\ell}{(\ell+1)^{B+1}} + \frac{1}{(\ell+1)^A} \leq 1,$$

[3] If $\ell(\alpha) < 2$, then the result clearly holds with ℓ replaced by $\ell + 2$ throughout.

which is obvious for $A \geq 1$. $\dashv$

I do not know of a corresponding inequality

$$C^s_\alpha(M) \leq k_\alpha^{C^p_\alpha(M)}$$

for the complexities that count only calls to the givens, disregarding the logical operations, cf. Problem x4.6.

4.6. Syntactic complexities of explicit terms. Since Φ-terms can be viewed as recursive programs (with no "body"), it is natural to compare their complexities with the usual, syntactic measures of their sizes. We formulate the most natural of these here for easy reference.

The **length** of a $\Phi[A]$-term E is defined recursively by the clauses

$$\text{length}(0) = \text{length}(1) = \text{length}(v_i) = \text{length}(x) = 0,$$
$$\text{length}(\phi(E_1, \dots, E_n) = \text{length}(E_1) + \cdots + \text{length}(E_n) + 1,$$
$$\text{length}(\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2) = \text{length}(E_1) + \text{length}(E_1) + \text{length}(E_2) + 1,$$

and its **Φ-length** by the same clauses when we count only Φ-symbols:

$$\text{length}_\Phi(0) = \text{length}_\Phi(1) = \text{length}_\Phi(v_i) = \text{length}(x) = 0,$$
$$\text{length}_\Phi(\phi(E_1, \dots, E_n) = \text{length}_\Phi(E_1) + \cdots + \text{length}_\Phi(E_n) + 1,$$
$$\text{length}_\Phi(\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2) = \text{length}_\Phi(E_1) + \text{length}_\Phi(E_1) + \text{length}_\Phi(E_2) + 1.$$

Similarly, the **depth** of a $\Phi[A]$-term E is defined recursively by the clauses

$$\text{depth}(0) = \text{depth}(1) = \text{depth}(v_i) = \text{depth}(x) = 0,$$
$$\text{depth}(\phi(E_1, \dots, E_n) = \max\{\text{depth}(E_1), \dots, \text{depth}(E_n)\} + 1,$$
$$\text{depth}(\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2) = \max\{\text{depth}(E_1), \text{depth}(E_1), \text{depth}(E_2)\} + 1,$$

and its **Φ-depth** (which only counts occurrences of symbols in the signature Φ) is defined by the corresponding clauses

$$\text{depth}_\Phi(0) = \text{depth}_\Phi(1) = \text{depth}_\Phi(v_i) = \text{depth}_\Phi(x) = 0,$$
$$\text{depth}_\Phi(\phi(E_1, \dots, E_n) = \max\{\text{depth}_\Phi(E_1), \dots, \text{depth}_\Phi(E_n)\} + 1,$$
$$\text{depth}_\Phi(\text{if } (E_0 = \mathsf{0}) \text{ then } E_1 \text{ else } E_2) = \max\{\text{depth}_\Phi(E_1), \text{depth}_\Phi(E_1), \text{depth}_\Phi(E_2)\}.$$

4.7. Recursive vs. explicit definability. It is easy to verify that if $E(\vec{\mathsf{x}})$ is a Φ-term such that $\text{den}(E(\vec{x}))\downarrow$, then

$$\text{(23)} \quad C^p(E(\vec{x})) \leq \text{depth}_\Phi(E(\vec{x})), \qquad C^s(E(\vec{x})) \leq \text{length}_\Phi(E(\vec{x})),$$
$$L^p(E(\vec{x})) \leq \text{depth}(E(\vec{x})), \qquad L^s(E(\vec{x})) \leq \text{length}(E(\vec{x})),$$

see Problem x4.7. This implies, in particular, that if we derive a non-constant lower bound for $l^p_\alpha(\vec{x})$ for any program α which computes a certain $f : A^n \rightharpoonup A$, then no Φ-term defines f on A. We establish here the converse of this proposition, which shows that explicit definability is equivalent to computability with bounded logical complexity.[4]

[4]Because of Proposition 4.5.2, we can speak of "bounded logical complexity" without specifying which of the two logical complexities l^p or l^s we have in mind.

4.7.1. THEOREM. *Given $f : A^n \rightharpoonup A$, $S \subseteq \{\vec{x} \mid f(\vec{x})\downarrow\}$ and partial functions $\mathbf{\Phi}$ on A, the following are equivalent:*

(a) *There is a Φ-term $E(\vec{x})$ which defines f on S, i.e.,*

$$\vec{x} \in S \Longrightarrow f(\vec{x}) = \mathrm{den}(E(\vec{x})).$$

(b) *There is a Φ-program α and a number k such that for every $\vec{x} \in S$,*

$$\overline{\alpha}(\vec{x}) = f(\vec{x}) \text{ and } l^p_\alpha(\vec{x}) \leq k.$$

In particular, a function $f : A^n \to A$ is definable by a Φ-term if and only if it is computed by a Φ-program α with bounded logical complexity.

PROOF. (a) $\Longrightarrow$ (b) follows immediately from (23), with α the term $E(\vec{x})$ (as a program with no body) and $k = \mathrm{depth}(E(\vec{x}))$. For the converse implication we need some preliminary work.

It is convenient for this proof to add a nullary function constant $\emptyset$ to the language which provides a name for the nullary, undefined partial function; the semantics of these extended $(\Phi, \emptyset)$-terms and the corresponding $(\Phi, \emptyset)[A]$-terms (where we allow parameters from A) are obvious, and we will need only one (*monotonicity*) property of them, which is established by a trivial induction on M:

Lemma 1. If $M(\emptyset)$ is a closed $(\Phi, \emptyset)[A]$-term such that $\mathrm{den}(M(\emptyset))\downarrow$, *$N$ is any closed $(\Phi, \emptyset)[A]$-term, and $M(N)$ is the $\Phi[A]$-term constructed by replacing every occurrence of $\emptyset$ in $M(\emptyset)$ by N, then* $\mathrm{den}(M(\emptyset)) = \mathrm{den}(M(N))$. In particular,

$$\text{if } \mathrm{den}(M(\emptyset))\downarrow, \text{ then } \mathrm{den}(M(\emptyset)) = \mathrm{den}(M(0)).$$

Next we associate with each Φ-program α, each $\mathrm{sig}(\alpha)$-term

$$M \equiv M(\vec{x}, \vec{f})$$

and each k, a $(\Phi, \emptyset)$-term $[M]^{(k)}(\vec{x})$ with the same individual variables (the *term approximation* of M to depth k) by the following recursion on k, and within this, recursion on $\mathrm{depth}(M)$:

1. If M is a variable, or 0, or 1, then $[M]^{(k)} \equiv M$.
2. $[M]^{(0)}(\vec{x}) \equiv M(\vec{x}, \vec{\emptyset})$, where the substitution $f_i \mapsto \emptyset$ means that every subterm of $M(\vec{x}, \vec{f})$ of the form $f_i(M_1, \dots, M_n)$ is replaced by $\emptyset$.
3. If $M \equiv \phi(M_1, \dots, M_n)$, then $[M]^{(k+1)} \equiv \phi([M_1]^{(k)}, \dots, [M_n]^{(k)})$.
4. If $M \equiv \text{if } (M_0 = 0) \text{ then } M_1 \text{ else } M_2$, then

$$[M]^{(k+1)} \equiv \text{if } ([M_0]^{(k+1)} = 0) \text{ then } [M_1]^{(k+1)} \text{ else } [M_2]^{(k+1)}.$$

5. If $M \equiv f_i(M_1, \dots, M_n)$, then

$$[M]^{(k+1)} = \text{if } ([M_1]^{(k)}\downarrow \ \& \ \cdots \ \& \ [M_n]^{(k)}\downarrow) \text{ then } [E_i]^{(k)}([M_1]^{(k)}, \cdots, [M_n]^{(k)}) \text{ else } \emptyset.$$

The complex conditional in the last clause abbreviates a nested sequence of conditionals which insures that the term on the right will not converge unless all the terms $[M_i]^{(k)}$ converge.[5] The definition of $[M]^{(k)}$ depends on the program α (so that we should properly write $[M]^{(k)}_\alpha$), but not on any specific Φ-algebra $\mathbf{A}$.

Lemma 2. For each Φ-program α, each $\mathrm{sig}(\alpha)$*-term $M(\vec{x})$, each k and any $\vec{x} \in A^n$:*

(i) *If* $\mathrm{den}([M]^{(k)}(\vec{x}))\downarrow$, *then* $\mathrm{den}([M]^{(k)}(\vec{x})) = \mathrm{den}([M]^{(k+1)}(\vec{x}))$.

(ii) *If $L^p(M(\vec{x})) \le k$, then* $\mathrm{den}([M]^{(k)}(\vec{x})) = \overline{M(\vec{x})}$.

Proof. (i) is verified by induction on k, and within this by induction on $\mathrm{depth}(M)$.

Basis, $k = 0$. The result is obvious when M is 0, 1 or a variable, and there is nothing to prove if $M \equiv f_i(M_1, \dots, M_n)$, since in that case

$$\mathrm{den}([f_i(M_1, \dots, M_n)]^{(0)}(\vec{x})) = \mathrm{den}(\emptyset)\uparrow .$$

For one of the other two cases where the hypothesis may hold, (i) follows immediately from the definitions:

$$\begin{aligned}[\phi(M_1, \dots, M_n)]^{(1)} &\equiv \phi([M_1]^{(0)}, \dots, [M_n]^{(0)})\\ &\equiv \phi(M_1(\vec{\mathsf{x}}, \vec{\emptyset}), \dots, M_n(\vec{\mathsf{x}}, \vec{\emptyset})) \equiv [\phi(M_1, \dots, M_n)]^{(0)};\end{aligned}$$

so $[M]^{(0)}$ and $[M]^{(1)}$ certainly have the same denotations. Finally, when M is a conditional we use again the induction hypothesis on the depth of M: assuming that $\mathrm{den}([M_0]^{(0)} = 0$ and $[M_1]^{(0)}\downarrow$, we compute:

$$\mathrm{den}([M]^{(1)}) = \mathrm{den}([M_1]^{(1)}) = \mathrm{den}([M_1]^{(0)}) = \mathrm{den}([M]^{(0)}).$$

Induction Step. We use again induction on the depth of M and the argument is exactly like that in the basis, except for the new case of $M \equiv f_i(M_1, \dots, M_n)$ which may now arise. In this case, the hypothesis gives us that $\mathrm{den}([M_j]^{(k)}(\vec{x}))\downarrow$ for all j, and so by the definition and the induction hypothesis

$$\begin{aligned}\mathrm{den}([M]^{(k+1)}(\vec{x})) &= \mathrm{den}([E_i]^{(k)}([M_1]^{(k)}(\vec{x}), \dots, [M_n]^{(k)}(\vec{x})))\\ &= \mathrm{den}([E_i]^{(k+1)}([M_1]^{(k+1)}(\vec{x}), \dots, [M_n]^{(k+1)}(\vec{x}))) = \mathrm{den}([M]^{(k+2)}(\vec{x})).\end{aligned}$$

(ii) is proved by induction on $L^p(M)$, and it is obvious when M is 0, 1 or a variable and quite routine in the induction step, with the help of the monotonicity properties established in (i), as follows.

If $M \equiv \phi(M_1, \dots, M_n)$, then

$$L^p(M) = \max\{L^p(M_1), \dots, L^p(M_n)\} + 1 = k + 1,$$

and by the induction hypothesis

$$\mathrm{den}([M_i]^{(k)}) = \overline{M}_i,$$

[5] For any sequence of terms $N_1, \dots, N_n$, the term $N_1\downarrow \ \& \ \cdots \ \& \ N_n\downarrow$ is defined by recursion on n so that it converges (and has value 0) exactly when all N_i converge:

$$N_1\downarrow \equiv \text{if } (N_1 = 0) \text{ then } 0 \text{ else } 0,$$

$$N_1\downarrow \ \& \ \cdots \ \& \ N_{n+1}\downarrow \equiv \text{if } (N_{n+1} = 0) \text{ then } (N_1\downarrow \ \& \ \cdots \ \& \ N_n\downarrow) \text{ else } (N_1\downarrow \ \& \ \cdots \ \& \ N_n\downarrow).$$

so that

$$\mathrm{den}([M]^{(k+1)}) = \phi(\mathrm{den}([M_1]^{(k)}), \ldots, \mathrm{den}([M_n]^{(k)})) = \phi(\overline{M}_1, \ldots, \overline{M}_n) = \overline{M}.$$

The argument is similar for conditionals, and if $M \equiv f_i(M_1, \ldots, M_n)$, then

$$k + 1 = L^p(M) = \max\{L^p(M_1), \ldots, L^p(M_n)\} + L^p(E_i(\overline{M}_1, \ldots, \overline{M}_n)) + 1,$$

so that the induction hypothesis applies to the terms $M_1, \ldots, M_n, E_i(\overline{M}_1, \ldots, \overline{M}_n)$ and yields

$$\mathrm{den}([M_i]^{(k)}) = \overline{M}_i, \quad \mathrm{den}([E]^{(k)}(\overline{M}_1, \ldots, \overline{M}_n)) = \overline{E_i(\overline{M}_1, \ldots, \overline{M}_n)} = \overline{M};$$

the required $\mathrm{den}([M_i]^{(k+1)}) = \overline{M}$ now follows by (i).

$\dashv$ (Lemma 2)

To prove that (b) $\Longrightarrow$ (a) in the Theorem, take

$$E(\vec{\mathsf{x}}) \equiv [E_0(\vec{\mathsf{x}})]^{(k)}\{\emptyset :\equiv 0\},$$

where $E_0(\vec{\mathsf{x}})$ is the head of the given program α, $k = \max\{l^p_\alpha(\vec{x}) \mid \vec{x} \in S\}$, and $[E_0(\vec{\mathsf{x}})]^{(k)}\{\emptyset :\equiv 0\}$ is the Φ-term constructed by replacing $\emptyset$ by 0 in $[E_0(\vec{\mathsf{x}})]^{(k)}$. $\dashv$

4.8. Simulating Turing machines with $\mathbf{N}_b$-programs. To connect the complexity functions of this section with classical, Turing-machine time complexity, we show here (in outline) one typical result:[6]

4.8.1. Proposition. *If a function $f : \mathbf{N} \to \mathbf{N}$ is computable by a Turing machine M in time $T(\log n)$ for $n > 0$, then there is a natural number k and a $\mathbf{N}_b$-program α which computes f with $l^s_\alpha(n) \le kT(\log n)$ for $n > 0$.*

In particular, if f is computable in polynomial time, then it is computable by a $\mathbf{N}_b$-program α with sequential logical complexity $l^s_\alpha(n) \le k \log^d n$, for some natural numbers k, d and all $n > 0$.

We are assuming here that the input n is entered on a tape of M in binary notation (which is why we express the complexity as a function of $\log n$), but other than that, the result holds in full generality: the machine M may or may not have separate input and output tapes, it may have one or many, semi-infinite or infinite work tapes, etc. An analogous result holds also for functions of several variables.

Outline of proof. Consider the simplest case, where M has a two-way infinite tape and only one symbol in its alphabet, 1. We use 0 to denote the blank symbol, so that the "full configuration" of a machine at a stage in a computation is a triple (q, τ, i), where q is a state, $\tau : \mathbb{Z} \to \{0, 1\}$, $\tau(j) = 0$ for all but finitely many j's, and $i \in \mathbb{Z}$ is the location of the scanned cell. If we write τ as a pair of sequences emanating from the scanned cell y_0

$$\cdots x_3 x_2 x_1 x_0 \underset{\uparrow}{y_0} y_1 y_2 y_3 \cdots$$

one "growing" to the left and the other to the right, we can then code (τ, i) by the pair of numbers

$$(x, y) = (\textstyle\sum_j x_j 2^j, \sum_j y_j 2^j).$$

[6] This is not needed for the sequel, and its proof assumes some familiarity with Turing machines.

Notice that $y_0 = \mathrm{parity}(y)$ and $x_0 = \mathrm{parity}(x)$, so that the scanned symbol and the symbol immediately to its left are computed from x and y by $\mathbf{N}_b$-operations. The input configuration for the number z is coded by the triple $(q_0, 0, z)$, with q_0 the starting state, and all machine operations correspond to simple $\mathbf{N}_b$-functions on these codes. For example:

$$\text{move to the right} : x \mapsto 2x + \mathrm{parity}(y), \quad y \mapsto \mathrm{iq}_2(y),$$
$$\text{move to the left} : x \mapsto \mathrm{iq}_2(x), \quad y \mapsto 2y + \mathrm{parity}(x),$$

where (in the notation of (10)),

$$2x + \mathrm{parity}(y) = \text{if } (\mathrm{parity}(y) = 0) \text{ then } \mathrm{em}_2(x) \text{ else } \mathrm{om}_2(x).$$

It is not difficult to write a $\mathbf{N}_b$-program using these functions which simulates M.

If M has $K > 1$ symbols, we appeal to some of the standard reductions of K-symbol Turing machines to 1-symbol machines. The codings are somewhat messier when M has semi-infinite tapes. ⊣

Problems for Section 4

For any two tuples

$$\vec{x} = (x_1, \ldots, x_n) \in A^n, \quad \vec{y} = (y_1, \ldots, y_n) \in A^n,$$

we set

$$\vec{x} \backsim \vec{y} \iff x_i = y_i \text{ for all } i \text{ such that } x_i \in \{0,1\} \text{ or } y_i \in \{0,1\}. \tag{24}$$

x4.1. Show that for a fixed recursive program on a partial algebra $\mathbf{A}$,

$$\text{if } C^p(M(\vec{x})) = 0 \text{ and } \vec{x} \backsim \vec{y}, \text{ then } \overline{M(\vec{x})} \backsim \overline{M(\vec{y})}.$$

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

$$c^p_\alpha(\vec{x}) \geq 1. \tag{25}$$

Hint: Use induction on $D(M)$.

This is a funny-looking result, but (25) often prohibits 0-complexity; for example, if α decides primality, then for every prime p, $c^p_\alpha(p) \geq 1$—simply because $\overline{\alpha}(p) = 1$ and $\overline{\alpha}(2p) = 0$, while $p \backsim 2p$.

x4.2. Compute (up to a multiplicative constant) $c^p_\alpha(x, y)$ for the program defined (informally) in Problem x3.1.

x4.3. Compute (up to a multiplicative constant) $c^p_\alpha(x, y)$ for the program defined (informally) in Problem x3.2.

x4.4. Compute (up to a multiplicative constant) $c^p_\alpha(x, y)$ for the program in Problem x3.3.

x4.5. Consider the program α with the single equation

$$f(x) = \text{if } (x = 0) \text{ then } 0 \text{ else } f(\mathrm{Pd}(x)) + f(\mathrm{Pd}(x))$$

in the algebra $(\mathbb{N}, 0, 1, \mathrm{Pd}, +)$. Determine the function $\overline{\alpha}$ computed by this program, as well as the complexity functions $c^p_\alpha(x)$ and $c^s_\alpha(x)$. Verify that for some constant $k > 1$,

$$c^s_\alpha(x) \geq k^{c^p_\alpha(x)} \quad (x > 0).$$

Whether the converse inequality holds is still an open problem:

x4.6 (**Open**). True or false: for each Φ-program α and each Φ-algebra $\mathbf{A}$, there is a constant k such that for all tuples $\vec{x}$ with sufficiently large $\min(\vec{x})$,

$$c^s_\alpha(\vec{x}) \leq k^{c^p_\alpha(\vec{x})}.$$

x4.7. Prove the inequalities in (23). HINT: Show first that if $E(\mathsf{x})$ is a Φ-term in which the indicated variable x may occur, F is a Φ-term, and $E(F)$ is the result of replacing each occurrence of x in $E(\mathsf{x})$ by F, then

$$\mathrm{depth}(E(F)) \leq \mathrm{depth}(E(\mathsf{x})) + \mathrm{depth}(F)$$
$$\text{and } \mathrm{depth}_\Phi(E(F)) \leq \mathrm{depth}_\Phi(E(\mathsf{x})) + \mathrm{depth}_\Phi(F).$$

TREES. Fix an algebra $\mathbf{A}$ and a recursive program α in the same signature. A *label* is an equation of the form $M = w$, where M is a closed, convergent $\mathrm{sig}(\alpha)[A]$-term and $w \in A$, and a *node* is any finite, non-empty sequence $(L_1, L_2, \dots, L_n)$ of labels. The *depth* of the node $(L_1, L_2, \dots, L_n)$ is $n - 1$. A (finite, rooted) *tree* (of labels) is any finite set T of nodes, which contains all initial segments of its nodes and only one node of depth 0, its *root*. The *leaves* of T are the nodes which have no proper extension in T, and the *depth* of T is the maximum of the depths of its nodes.

OPERATIONS ON TREES. If M is a closed $\mathrm{sig}(\alpha)[A]$ term, $w \in A$ and $T_1, \dots, T_k$ are trees, then $F(M = w, T_1, \dots, T_k)$ is the tree with root $M = w$ and $T_1, \dots, T_k$ immediately below it. Formally:

$$(26) \quad F(M = w, T_1, \dots, T_k) = \{(M = w, L_1, \dots, L_n) \mid \text{ for some } i, (L_1, \dots, L_n) \in T_i\}.$$

If T and S are trees, then $T \star S$ is constructed by putting a copy of S below every leaf of T, formally,

$$(27) \quad T \star S = \{(L^T_1, \dots, L^T_n, L^S_1, \dots, L^S_m) \mid (L^T_1, \dots, L^T_n) \text{ is a leaf of } T, \text{ and } (L^S_1, \dots, L^S_m) \in S\}.$$

x4.8. Define precisely the computation tree $T(M)$ for a convergent, closed $\mathrm{sig}(\alpha)[A]$-term M using the operation in (26), as it is illustrated in Figure 1, and prove that

$$D(M) = \text{the depth of } T(M).$$

§5. The basic test. We will prove in this section a very simple fact, which, however, implies some interesting complexity lower bounds.

In the notation of Section 4, if $\mathbf{A}$ is an algebra, α is a program, and M is a closed, convergent $\mathrm{sig}(\alpha)[A]$-term, then $\overline{M}$ is its value in $\mathbf{A}$, and $D(M), C^p(M)$ are its tree-depth and parallel logical complexities. We write

$$\mathbf{A}, \alpha \models M = w \iff \overline{M} = \mathrm{den}((\mathbf{A}, \overline{f}_0, \dots, \overline{f}_K), M) = w, \tag{28}$$

$$\mathbf{A}, \alpha \models^m M = w \iff \overline{M} = \mathrm{den}((\mathbf{A}, \overline{f}_0, \dots, \overline{f}_K), M) = w \;\&\; C^p(\mathbf{A}, M) \leq m, \tag{29}$$

$$\mathbf{A}, \alpha \models^m \overline{\alpha}(\vec{x}) = w \iff \mathbf{A}, \alpha \models \overline{\alpha}(\vec{x}) = w \;\&\; c^p_\alpha(\vec{x}) \leq m. \tag{30}$$

This model theoretic notation gives us a reasonably simple way to indicate the algebra and the program relative to which the denotations $\overline{M}$, $\overline{\alpha}(\vec{x})$ and the complexities $C^p(M)$, $c^p_\alpha(\vec{x})$ are computed.

Recall the sets $G_m(X)$ defined in Subsection 2.5.

5.0.2. Lemma (The First Downward Lemma). *If X contains all the parameters of a closed,* $\mathrm{sig}(\alpha)[A]$*-term M and $\mathbf{A}, \alpha \models^m M = w$, then $w \in G_m(X)$.*

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

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

$$\overline{M}_i \in G_{m-1}(X);$$

hence $\overline{M} = \phi(\overline{M}_1, \dots, \overline{M}_n) \in G_m(\vec{x})$.

(3) Suppose $M \equiv \mathsf{f}(M_1, \dots, M_n)$, with a recursive variable f. By the induction hypothesis then, there are k, l such that $k + l \leq m$, $\overline{M}_i \in G_k(X)$ for $i = 1, \dots, n$, and

$$\mathrm{den}(E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)) \in G_l(\overline{M}_1, \dots, \overline{M}_n);$$

therefore

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

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

5.0.3. Corollary (The basic test). *If $\overline{\alpha}(\vec{x}) \notin G_m(\vec{x})$, then $c^p_\alpha(\vec{x}) > m$.*

The intended applications are, of course, in the cases where α computes some given function $f : \mathbb{N}^n \to \mathbb{N}$ whose complexity we want to study.

5.0.4. Example. If $\overline{\alpha}(x, y) = x + y$ in the algebra $\mathbf{N}_u = (\mathbb{N}, 0, 1, S, \mathrm{Pd})$ of unary arithmetic and $x, y \geq 2$,

$$c^p_\alpha(x, y) \geq \min(x, y).$$

Proof. It is enough to show that if $x, y \geq 2$ and $m < \min(x, y)$, then

$$x + y \notin G_m(\mathbf{N}_u, x, y);$$

this follows from

$$G_m(x, y) = \{0, 1, \dots, m+1\} \cup \{x + z \in \mathbb{N} \mid z \in \mathbb{Z}, |z| \leq m\} \cup \{y + z \in \mathbb{N} \mid z \in \mathbb{Z}, |z| \leq m\},$$

which is easily verified by induction on m. ⊣

5.1. The set of primitives $\mathbf{Lin}_0$; multiplication by halving. The result simply expresses the fact that to compute $x + y$ we must read at least one of the variables, and this requires at least $\min(x, y)$ steps if we only have S and Pd at our disposal. More interesting lower bound results can be proved using Corollary 5.0.3 when we enlarge the givens of the structure. Set

$$\mathbf{Lin}_0 = \{=, <, \text{parity}, \text{em}_2, \text{iq}_2, +, \dot{-}\} \tag{31}$$

where the relations stand for their characteristic functions and (as above),

$$\text{em}_2(x) = 2x, \quad \text{iq}_2(x) = \text{iq}(x, 2).$$

This is a natural, finite subset of the set **Lin** of *piecewise linear* functions on $\mathbb{N}$ which we will study later on. We also set

$$\mathbf{N}_0 = (\mathbb{N}, 0, 1, \mathbf{Lin}_0) = (\mathbb{N}, 0, 1, =, <, \text{parity}, \text{em}_2, \text{iq}_2, +, \dot{-}) \tag{32}$$

To illustrate the use of $\mathbf{Lin}_0$ givens, consider the following.

5.1.1. Lemma. *There is a recursive program α which computes the product $x \cdot y$ from $\mathbf{Lin}_0$ with complexity*

$$c^p_\alpha(x, y) \le 3\log(\min(x, y)) \quad (x, y \ge 2).$$

Proof. The idea (from Problem x3.2) is to reduce multiplication by x to multiplication by 2 and $\text{iq}_2(x)$ (and addition), using the identity

$$(2x_1 + r) \cdot y = 2(x_1 \cdot y) + r \cdot y,$$

by which the multiplication function satisfies the following recursive equation:

$$\begin{aligned} f(x, y) = {} & \text{if } (x = 0) \text{ then } 0 \\ & \text{else if } (x = 1) \text{ then } y \\ & \text{else if } (\text{parity}(x) = 0) \text{ then } 2(f(\text{iq}_2(x), y)) \\ & \text{else } 2(f(\text{iq}_2(x), y)) + y. \end{aligned}$$

Now, obviously,

$$c^p_f(0, y) = 0, \quad c^p_f(1, y) = 1,$$

and with a careful reading of the equation, for $x \ge 2$,

$$c^p_f(x, y) \le c^p_f(\text{iq}_2(x), y) + 2.$$

To get an explicit form for an upper bound to $c^p_f(x, y)$, we prove by (complete) induction the inequality

$$c^p_f(x, y) \le 3\log(x) \quad (x \ge 2),$$

the basis being trivial, since $c^p_f(2, y) = c^p_f(1, y) + 2 = 3 = 3\log 2$, directly from the definition. In the inductive step,

$$c^p_f(x, y) \;\le\; c^p_f(\text{iq}_2(x), y) + 2 \;\le\; 3\log(\frac{x}{2}) + 3 \;=\; 3(\log(\frac{x}{2}) + 1) \;=\; 3\log x.$$

Finally, to complete the proof, we add a head equation which insures that the first argument of f is the minimum of x and y:

$$g(x, y) = \text{if } (y < x) \text{ then } f(y, x) \text{ else } f(x, y);$$

now the program α comprising these two recursive equations satisfies the required complexity bound. $\dashv$

The key to proving a matching lower bound for multiplication from $\mathbf{Lin}_0$ is a simple representation result for the sets $G_m(X)$ in the algebra $\mathbf{N}_0$. For each tuple of natural numbers $\vec{a} = (a_1, \dots, a_n)$, let

$$(33)\quad B_m(\vec{a}) = \Big\{ \frac{x_0 + x_1a_1 + \cdots + x_na_n}{2^m} \in \mathbb{N} \\ \mid x_0, \dots, x_n \in \mathbb{Z} \text{ and } |x_i| \le 2^{2m}, i \le n \Big\}.$$

The members of $B_m(\vec{a})$ are *natural numbers*. In full detail:

$$\begin{aligned} x \in B_m(\vec{a}) \iff\ & x \in \mathbb{N} \text{ and there exist } x_0, \dots, x_n \in \mathbb{Z} \\ & \text{such that } x = \frac{x_0 + x_1a_1 + \cdots + x_na_n}{2^m}, \\ & \text{and for } i = 0, \dots, n, |x_i| \le 2^{2m}, \end{aligned}$$

which, in particular, implies that $2^m \mid x_0 + x_1a_1 + \cdots + x_na_n \ge 0$.

5.1.2. Lemma ($\mathbf{Lin}_0$-inclusion). *For all $\vec{a} \in \mathbb{N}^n$ and all m.*

(1) $0, 1, a_1, \dots, a_n \in B_m(\vec{a}) \subseteq B_{m+1}(\vec{a})$.

(2) *For every k-ary function ϕ in $\mathbf{Lin}_0$,*

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

(3) $G_m(\vec{a}) = G_m(\mathbf{N}_0, \vec{a}) \subseteq B_m(\vec{a})$.

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 $1, a, b$. The second inclusion holds because

$$\frac{x_0 + x_1a + x_2b}{2^m} = \frac{2x_0 + 2x_1a + 2x_2b}{2^{m+1}}$$

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

(2) This is obvious for the relations in $\mathbf{Lin}_0$, whose characteristic functions take values in $B_0(a, b)$. For addition, let $x, y \in B_m(a, b)$, so

$$\begin{aligned} x + y = \frac{x_0 + x_1a + x_2b}{2^m} + \frac{y_0 + y_1a + y_2b}{2^m} \\ = \frac{2(x_0 + y_0) + 2(x_1 + y_1)a + 2(x_2 + y_2)b}{2^{m+1}} \end{aligned}$$

and the coefficients in the numerator satisfy

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

The same works for subtraction. For multiplication by 2, notice that

$$2\Big(\frac{x_0 + x_1a + x_2b}{2^m}\Big) = \frac{4x_0 + 4x_1a + 4x_2b}{2^{m+1}},$$

and $|4x_0| \le 4 \cdot 2^{2m} = 2^{2(m+1)}$. Finally, for integer division by 2, if $2 \mid x$, then

$$\mathrm{iq}_2(x) = \frac{1}{2}\Big(\frac{x_0 + x_1a + x_2b}{2^m}\Big) = \frac{x_0 + x_1a + x_2b}{2^{m+1}} \in B_{m+1}(a, b),$$

and if $2 \nmid x$, then

$$\mathrm{iq}_2(x) = \frac{1}{2}(x-1) = \frac{(x_0 - 2^m) + x_1 a + x_2 b}{2^{m+1}} \in B_{m+1}(a,b),$$

which implies that this number is in B_{m+1} as above.

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

5.1.3. Proposition (Lower bound for multiplication from $\mathbf{Lin}_0$). *If α is a recursive program, $a \geq 2$, and $\overline{\alpha}(a) = a^2$ in $\mathbf{N}_0$, then*

$$c^p_\alpha(a) \geq \log\Big(\frac{a^2}{a+1}\Big).$$

As a consequence, for every program β which computes multiplication from $\mathbf{Lin}_0$ and every $a \geq 3$,

$$c_\beta(a,a) \geq \log\Big(\frac{a^2}{a+1}\Big) > \frac{1}{2}\log a,$$

and, in particular, for infinitely many pairs (a,b),

$$c_\beta(a,b) > \frac{1}{2}\log(\max(a,b)).$$

Proof. To infer the main claim in the proposition from Corollary 5.0.3, it is enough to show that for $a \geq 2$,

$$a^2 \in G_m(a) \Longrightarrow m \geq \log\Big(\frac{a^2}{a+1}\Big)$$

with $G_m(a)$ computed from $\mathbf{Lin}_0$, so assume that $a^2 \in G_m(a)$. By the Lemma, there exist $x_0, x_1 \in \mathbb{Z}$, $|x_0|, |x_1| \leq 2^{2m}$, such that

$$a^2 = \frac{x_0 + x_1 a}{2^m},$$

from which we get $2^m a^2 = |x_0 + x_1 a| \leq 2^{2m} + 2^{2m} a$; thus $a^2 \leq 2^m + 2^m a$, which yields

$$2^m \geq \frac{a^2}{a+1}.$$

The required inequality follows by taking logarithms.

The second claim (with the same bound) follows from the first by adding a new head symbol $f(x) = f_0(x,x)$ to any program which computes multiplication. Finally, to get the simpler-looking, logarithmic bound, it is enough to show that

$$\log\Big(\frac{a^2}{a+1}\Big) > \frac{1}{2}\log a = \log(\sqrt{a}) \qquad (a \geq 3),$$

or, equivalently, that for every $a \geq 3$,

$$\frac{a^2}{a+1} > \sqrt{a}, \text{i.e., } a^{\frac{3}{2}} > a+1.$$

To see this, let

$$g(x) = x^{\frac{3}{2}} - (x+1),$$

check that $g(3) = 3^{\frac{3}{2}} - 3 = \sqrt{27} - 3 > 0$, and check also that this function has positive derivative for $x \geq 3$:

$$g(x) = \frac{3}{2}\sqrt{x} - 1 \geq \frac{1}{2}.$$

This implies that g is increasing for $x \geq 3$, and so always positive. $\dashv$

5.2. Optimal programs. Suppose $f : A^n \to A$ with $0, 1 \in A$ and $\mathbf{\Phi}$ is a set of functions on A. A program α is *optimal* (up to a multiplicative constant) *for f from $\mathbf{\Phi}$*, if it computes f in $(A, 0, 1, \mathbf{\Phi})$, and there is a real number $r > 0$ such that for every program β which computes f from $\mathbf{\Phi}$,

$$\text{for infinitely many } \vec{x}, c^p_\beta(\vec{x}) \geq rc^p_\alpha(\vec{x}). \tag{34}$$

In this terminology, the (less detailed) "in particular" part of Proposition 5.1.3 says that the program in Lemma 5.1.1 is optimal for multiplication from $\mathbf{Lin}_0$. Notice, however, that the main part of the Lemma says something stronger—and potentially more interesting—by identifying the pairs (a, a) on which the lower bound is realized. In many cases, these more specific formulations of optimality have substantial, independent interest.

Problems for Section 5

x5.1. Specify a system of two recursive equations

$$\begin{aligned} q(x, y) &= E_q(x, y, q, r) \\ r(x, y) &= E_r(x, y, q, r), \end{aligned}$$

in the signature $\mathbf{Lin}_0 \cup \{q, r\}$, such that in $(\mathbb{N}, 0, 1, \mathbf{Lin}_0)$,

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

and the corresponding complexities are $O(\log(x))$, i.e., for some B and all sufficiently large x,

$$c_q(x, y) \leq B \log x, \quad c_r(x, y) \leq B \log x.$$

(With the appropriate head equations, this system defines two programs from $\mathbf{Lin}_0$, one for $\mathrm{iq}(x, y)$ and the other for $\mathrm{rem}(x, y)$.)

x5.2. Show that the program for the integer quotient function in Problem x5.1 is optimal from $\mathbf{Lin}_0$, i.e., there is a rational number $r > 0$, such that if α computes $\mathrm{iq}(x, y)$ from $\mathbf{Lin}_0$, then for infinitely many pairs (x, y), $c^p_\alpha(x, y) \geq r \log(x)$.

x5.3. Show that the program for $\mathrm{rem}(x, y)$ in Problem x5.1 is optimal from $\mathbf{Lin}_0$. (The proof of this directly from Corollary 5.0.3 is due to Tim Hu.)

x5.4. Define an optimal from $\mathbf{Lin}_0$ program which computes the exponential function $f(x, y) = x^y$ (with $0^0 = (x + 1)^0 = 1$).

§6. The imbedding test. We cannot establish lower bounds for the complexity of relations by direct applications of the basic test 5.0.3, since (as functions) these take on only the values 0 and 1. In this section we will establish two very simple lemmas which cover this ground.

6.0.1. LEMMA (The Imbedding Lemma). *If $\pi : \mathbf{B} \rightarrowtail \mathbf{A}$ is an imbedding and α is a recursive program, then for every term without parameters*

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

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

$$\mathbf{B}, \alpha \models^m M(\vec{x}) = w \Longrightarrow \mathbf{A}, \alpha \models^m M(\pi(\vec{x})) = \pi(w),$$
$$\mathbf{B}, \alpha \models^m M(\vec{x})\downarrow \Longrightarrow \mathbf{A}, \alpha \models^m M(\pi(\vec{x}))\downarrow .$$

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

PROOF. The second claim follows from the first, which is proved by induction on $D(M(\vec{x}))$ computed in $\mathbf{B}$, 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

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

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

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

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 $\mathbf{A}$ in place of $\mathbf{B}$ and $\pi(x_i), \pi(w_j), \pi(w)$ in place of x_i, w_j, w; and so the definition of the calls-depth complexity applied to $\mathbf{A}$ yields the required

$$\mathrm{den}(\mathbf{A}, M(\pi(\vec{x}))) = \pi(w), \quad C^p(\mathbf{A}, M(\pi(\vec{x}))) \leq m. \qquad \dashv$$

Next we strengthen the First Downward Lemma 5.0.2 to an "absoluteness" result—and it is hard to prove the strengthened version without reproving the original, so we will indulge in some repetition.

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

$$w \in G_m(\mathbf{A}, X) \text{ and } \mathbf{A} \restriction G_m(\mathbf{A}, X), \alpha \models^m M = w. \tag{35}$$

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 $i = 1, \dots, n$,

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

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

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

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 $i = 1, \dots, n$,

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

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

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

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

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

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. ⊣

6.0.3. Lemma (The Imbedding Test). *If there is an imbedding*

$$\pi : \mathbf{A}{\restriction}G_m(\vec{x}) \rightarrowtail \mathbf{A} \text{ such that } \overline{\alpha}(\pi\vec{x}) \neq \pi(\overline{\alpha}(\vec{x})),$$

then $c^p_\alpha(\vec{x}) > m$.

Proof. Suppose, towards a contradiction, that $c^p_\alpha(\vec{x}) \leq m$ but an imbedding with the claimed properties exists, and let $\overline{\alpha}(\vec{x}) = w$. Thus

$$\mathbf{A}, \alpha \models^m \overline{\alpha}(\vec{x}) = w,$$

and so by Lemma 6.0.2, $w \in G_m(\vec{x})$ and

$$\mathbf{A}{\restriction}G_m(\vec{x}), \alpha \models^m \overline{\alpha}(\vec{x}) = w.$$

Now Lemma 6.0.1 implies that

$$\mathbf{A}, \alpha \models^m \overline{\alpha}(\pi\vec{x}) = \pi w,$$

so that, in particular, $\overline{\alpha}(\pi\vec{x}) = \pi w$, contradicting the hypothesis. ⊣

6.1. Primality from $\mathbf{Lin}_0$. We illustrate the use of the imbedding test by establishing a logarithmic lower bound for recursive programs which decide primality from $\mathbf{Lin}_0$. What needs to be done, of course, is to define an imbedding which does not preserve primality.

6.1.1. Lemma ($\mathbf{Lin}_0$-Uniqueness). *If* $x_i, y_i \in \mathbb{Z}$, $|x_i|, |y_i| < \frac{a}{2}$ *and* $\lambda \geq 1$, *then*

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

PROOF. It is enough to prove the result for $\lambda = 1$, since $a \leq \lambda a$, and so the general result follows from the special case with the same hypothesis.

The second equivalence easily implies the first one, and follows 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

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

Proof. This is obvious if $y = 0$; and if $y \neq 0$, then $x < a \leq |ya|$, so that the sign of $x + ya$ is the same as the sign of ya, i.e., the sign of y. $\dashv$

6.1.2. LEMMA ($\mathbf{Lin}_0$-Imbedding). *Suppose $2^{2m+2} \leq a$, and let $\lambda > 1$ be any number such that $2^{m+1} \mid \lambda - 1$. There exists an algebra imbedding*

$$\pi : \mathbf{N}_0 \restriction G_m(a) \rightarrowtail \mathbf{N}_0,$$

such that $\pi a = \lambda a$.

PROOF. By Lemma 6.1.1 and part (3) of Lemma 5.1.2, the equation

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

defines a map $\pi : G_m(a) \to \mathbb{Q}$, since

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

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

$$x_0 + \lambda x_1 a = x_0 + x_1 a + (\lambda - 1) x_1 a, \tag{36}$$

so that if $2^m \mid (x_0 + x_1 a)$, then also $2^m \mid (x_0 + \lambda x_1 a)$ since $2^m \mid (\lambda - 1)$ by the hypothesis. It is injective and order-preserving, by Lemma 6.1.1 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 $|x_i|, |y_i|, |z_i| \leq 2^{2m}$. Now

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

and $|x_0 + y_0|, |x_1 + y_1| \leq 2 \cdot 2^{2m} = 2^{2m+1} < \dfrac{a}{2}$ by the hypothesis, and so by the Uniqueness Lemma 6.1.1,

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

which gives $\pi X + \pi Y = \pi Z$.

The same argument works for subtraction, and multiplication by 2 is just addition.

For parity, notice that $2 \mid X$ exactly when $2^{m+1} \mid x_0 + x_1 a$, which is equivalent to $2^{m+1} \mid x_0 + x_1 \lambda a$ by the assumption $2^{m+1} \mid (\lambda - 1)$ and the key equivalence (36).

Finally, for division by 2, suppose X as above is odd, so that

$$\mathrm{iq}_2(X) = \frac{1}{2}\Big(\frac{x_0 + x_1 a}{2^m} - 1\Big) = \frac{x_0 - 2^m + x_1 a}{2^{m+1}} = Z = \frac{z_0 + z_1 a}{2^m},$$

and $Z \in G_m$. These representations are now unique, since

$$|x_0 - 2^m| \le 2^{2m} + 2^m < 2^{2m+1} \le \frac{a}{2},$$

and so $x_0 - 2^m = 2z_0$ and $x_1 = 2z_1$. Since $\mathrm{parity}(X) = \mathrm{parity}(\lambda X)$, by the same argument we have the required

$$\mathrm{iq}_2(\pi X) = \frac{x_0 - 2^m + x_1\lambda a}{2^{m+1}} = \frac{2z_0 + 2z_1\lambda a}{2^{m+1}} = \frac{z_0 + z_1\lambda a}{2^m} = \pi Z, \qquad \dashv$$

6.1.3. Theorem ([3]). *If a recursive program α decides primality from* $\mathbf{Lin}_0$*, then for all primes p,*

$$c^p_\alpha(p) > \frac{1}{4}\log p.$$

Proof. Suppose α decides primality, p is prime, $m = c^p_\alpha(p)$ and

$$2^{2m+2} \le p. \tag{37}$$

The preceding Lemma 6.1.2 guarantees an imbedding

$$\pi : \mathbf{N}_0 \restriction G_m(a) \rightarrowtail \mathbf{N}_0$$

with $\lambda = 1 + 2^{m+1}$, such that $\pi p = \lambda p$; and this contradicts the imbedding test Lemma 6.0.3, since λp is not a prime. Thus (37) fails, and so (taking logarithms and using the fact that $m \ge 1$ by (25)),

$$4c^p_\alpha(p) = 4m \ge 2m + 2 > \log p. \qquad \dashv$$

6.1.4. Question: The polynomial algorithm for primality in [1] together with Proposition 4.8.1 yield a $\log_2^d(p)$ upper bound for the logical, linear complexity of $\mathbf{Lin}_0$ programs which decide primality, and Theorem 6.1.3 is the best lower bound that I know. Is it trivial—i.e., can it be obtained using "information" considerations, that we need to read at least one quarter of the binary expansion of p to determine using $\mathbf{Lin}_0$-operations whether it is a prime? Whatever the answer, the problem of establishing a $\log^2(p)$ lower bound for primality from $\mathbf{Lin}_0$ (or even the primitives of $\mathbf{N}_b$) is certainly an important problem, and (to my knowledge) open.

6.2. Good examples. The method in the preceding subsection can be easily adapted to derive lower bound results for many unary relations on $\mathbb{N}$. Some of these are covered by the next, fairly general notion.

A unary relation $R(x)$ is a *good example* if for some polynomial

$$\lambda(\mu) = \lambda(\mu) = 1 + l_1\mu + l_2\mu^2 + \cdots + l_s\mu^s \tag{38}$$

with coefficients in $\mathbb{N}$, constant term 1 and not identically $= 1$, and for all $\mu \ge 1$,

$$R(x) \Longrightarrow \neg R(\lambda(\mu)x). \tag{39}$$

For example, primality is good, taking $\lambda(\mu) = 1 + \mu$, and being a power of 2 is good with $\lambda(\mu) = 1 + 2\mu$.

6.3. The optimality of Stein's algorithm from $\mathbf{Lin}_0$. This can be shown by simple extensions of the methods we used to get lower bounds for unary relations from $\mathbf{Lin}_0$, which also yield several additional lower bound results for various binary relations in $\mathbf{N}_0$.

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

6.3.1. LEMMA. *Suppose $a > 2$ and set $b = a^2 - 1$.*
(1) $a \perp b$, and if $|x_i|, |y_i| < \dfrac{a}{4}$ for $i = 0, 1, 2$ and $\lambda \geq 1$, then

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

$$\begin{aligned} x_0 + x_1\lambda a + x_2\lambda b > y_0 + y_1\lambda a + y_2\lambda b \\ \iff [x_0 > y_0 \ \&\ x_1 = y_1 \ \&\ x_2 = y_2] \\ \vee [x_1 > y_1 \ \&\ x_2 = y_2] \vee [x_2 > y_2]. \end{aligned}$$

(2) If $2^{2m+3} < a$ and $\lambda = 1 + 2^{m+1}$, then there is an imbedding

$$\pi : \mathbf{N}_0 \restriction G_m(a, b) \to \mathbf{N}_0$$

such that $\pi a = \lambda a, \pi b = \lambda b$.

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

The second equivalence implies clearly the first, and it follows from the next proposition applied to $(x_0 - y_0)$, $(x_1 - y_1)$, $(x_2 - y_2)$.

If $|x|, |y|, |z| < \dfrac{a}{2}$ and $\lambda \geq 1$, then $x + y\lambda a + z\lambda b > 0$ if and only if either $x > 0$ and $y = z = 0$; or $y > 0$ and $z = 0$; or $z > 0$.

Proof. It is enough to prove the result for $\lambda = 1$, since the general claim follows by applying the special case to λa and λb.

Compute: $x + ya + zb = x + ya + z(a^2 - 1) = (x - z) + ya + za^2$. If $z = 0$, then (2) follows from Lemma 6.1.1. If $z \neq 0$ and $|x|, |y|, |z| < \frac{a}{2}$, then

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

(using the hypothesis $a > 2$), and so the sign of $(x - z) + ya + za^2$ is the same as the sign of z.

(2) Assume $2^{2m+3} < a$, set $\lambda = 1 + 2^{m+1}$, and notice that $2^{2m} < \frac{1}{4}a$, so as in Lemma 6.1.2, we can define the required imbedding by

$$\pi\Big(\frac{x_0 + x_1 a + x_2 b}{2^m}\Big) = \frac{x_0 + x_1\lambda a + x_2\lambda b}{2^m} \qquad \Big(|x_0|, |x_1|, |x_2| \leq 2^{2m}, \frac{x_0 + x_1 a + x_2 b}{2^m} \in G_m(a, b)\Big),$$

using now Lemma 6.3.1 instead of Lemma 6.1.1. ⊣

6.3.2. THEOREM ([3]). *If α is a recursive program which decides coprimeness from $\mathbf{Lin}_0$, then for all $a > 2$,*

$$c^p_\alpha(a, a^2 - 1) > \frac{1}{10}\log(a^2 - 1).$$

PROOF. Suppose α decides coprimeness from $\mathbf{Lin}_0$ and let

$$m = c^p_\alpha(a, a^2 - 1)$$

for some $a > 2$. Since λa and $\lambda(a^2-1)$ are not coprime, part (2) of the preceding Lemma 6.3.1 and the imbedding test 6.0.3 imply that

$$2^{2m+3} \geq a;$$

taking the logarithms of both sides and using the fact that $m \geq 1$ (by (25)), we get

$$5m \geq \log a;$$

which with $\log(a^2 - 1) < 2\log a$ gives the required

$$5m > \frac{1}{2}\log(a^2 - 1). \qquad \dashv$$

Theorem B now follows, because from an algorithm which computes $\gcd(x, y)$ we can get one to decide coprimeness by adding just one more step—checking whether $\gcd(x, y) = 1$.

Problems for Section 6

x6.1. Design an optimal recursive program from $\mathbf{Lin}_0$ for the relation

$$P(x) \iff (\exists y)[x = 2^y].$$

x6.2 ([3]). Prove that if $R(x)$ is a good example, then for every program α from $\mathbf{Lin}_0$ which decides $R(x)$ and all $a \geq 2$,

$$R(a) \Longrightarrow c^p_\alpha(a) \geq \frac{1}{4}\log a$$

x6.3. Prove that if $m > 0$, then $(1 + m^2)n^2$ is not a perfect square. HINT: Show first that $1 + m^2$ is not a perfect square, and then reduce the result to the case where $m \perp n$.

x6.4. Prove that the following two relations are good examples:

$$R_1(a) \iff a \text{ is a perfect square}$$
$$R_2(a) \iff a \text{ is square-free.}$$

x6.5. Prove that if $\lambda(\mu)$ is as in (38), then there is a constant C such that

$$\log \lambda(\mu) \leq C \log \mu \quad (\mu \geq 2). \tag{40}$$

The next problem gives a logarithmic lower bound for the complexity of $\mathbf{Lin}_0$ programs which decide good relations at many points where the relations does not hold.

x6.6. Suppose $R(x)$ is a good example with associated polynomial $\lambda(\mu)$. Prove that there is a rational constant $r > 0$, such that for every program α which decides $R(x)$ from $\mathbf{Lin}_0$, and for all $a \geq 2$ and $m \geq 1$,

$$R(a) \Longrightarrow c^p_\alpha(\lambda(2^{m+1})a) \geq r \log(\lambda(2^{m+1})a).$$

Problem x5.1 defines a recursive program from $\mathbf{Lin}_0$ which computes $\mathrm{rem}(x, y)$ with $O(\log)$ complexity. The next problem claims that it is optimal—and a little more.

x6.7. Prove that there is a rational $r > 0$, such that for every recursive program which decides the divisibility relation $x \mid y$ from $\mathbf{Lin}_0$,

$$\text{for infinitely many pairs } (x, y), x \mid y \text{ and } c^p_\alpha(x, y) \geq r \log y.$$

Infer that the program for the remainder in Problem x5.1 is optimal in $\mathbf{Lin}_0$. HINT: One way is to show that when a, b, λ and μ satisfy suitable conditions, then the mapping

$$\frac{x_0 + x_1 a + x_2 b}{2^m} \mapsto \frac{x_0 + x_1 \lambda a + x_2 \mu b}{2^m}$$

is an imbedding on $\mathbf{N}_0 \restriction G_m(a, b)$. (There may be a simpler way.)

§7. Unary relations from $\mathbf{Lin}_0[\div]$. We now add to the givens the integer quotient and remainder operations:

$$\begin{aligned} \mathbf{Lin}_0[\div] &= \{=, <, \mathrm{parity}, \mathrm{em}_2, \mathrm{iq}_2, +, \dot{-}, \mathrm{iq}, \mathrm{rem}\} \\ (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) &= (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) \\ &= (\mathbb{N}, 0, 1, =, <, \mathrm{parity}, \mathrm{em}_2, \mathrm{iq}_2, +, \dot{-}, \mathrm{iq}, \mathrm{rem}). \end{aligned}$$

The derivation of lower bounds for programs which decide unary relations from $\mathbf{Lin}_0[\div]$ is similar to that from $\mathbf{Lin}_0$ in Section 6, except that the computations are a bit more complex—and the bounds correspondingly lower—because the givens are no longer "piecewise linear". In particular, the representation of members of $G_m(a)$ as fractions with the same denominator is no longer useful.

For each tuple $\vec{a} = (a_1, \ldots, a_n)$ of numbers and for each $h \geq 1$, we let

$$(41) \quad C(\vec{a}; h) = \Big\{ \frac{x_0 + x_1 a_1 + \cdots + x_n a_n}{x_{n+1}} \in \mathbb{N} \mid x_0, \ldots, x_{n+1} \in \mathbb{Z}, \\ x_{n+1} > 0 \text{ and } |x_0|, \ldots, |x_{n+1}| \leq h \Big\}.$$

The numbers in $C(\vec{a}; h)$ are said to have *height* (no more than) h with respect to $\vec{a}$, and, trivially,

$$h \leq h' \Longrightarrow C(\vec{a}; h) \subseteq C(\vec{a}; h').$$

We need to estimate how much the height is increased when we perform various operations on numbers. The results are very simple for the functions in $\mathbf{Lin}_0$:

7.0.3. LEMMA. *For all $\vec{a}, h \geq 2$ and every k-ary operation in $\mathbf{Lin}_0$,*

$$x_1, \ldots, x_k \in C(\vec{a}; h) \Longrightarrow f(x_1, \ldots, x_k) \in C(\vec{a}; h^3).$$

PROOF is very simple, by direct computation. For example (taking $n = 2$ for simplicity)

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

and for the typical coefficient,

$$|y_3 x_0 + x_3 y_0| \le h^2 + h^2 = 2h^2 \le h^3. \qquad \dashv$$

There is no simple, general result of this type for division with remainder, and in this section we will consider the simplest case $n = 1$, when $C(a; h)$ comprises the numbers of height h with respect to a single a. We will need the appropriate version of Lemma 6.1.1.

7.0.4. Lemma (**Lin**$_0[\div]$-Uniqueness). *If $|x_i|, |y_i| \le h$, $\lambda \ge 1$ and $2h^2 < a$, then:*

$$\frac{x_0 + x_1\lambda a}{x_2} = \frac{y_0 + y_1\lambda a}{y_2} \iff y_2 x_0 = x_2 y_0 \ \&\ y_2 x_1 = x_2 y_1,$$

$$\frac{x_0 + x_1\lambda a}{x_2} > \frac{y_0 + y_1\lambda a}{y_2} \iff [y_2 x_1 > x_2 y_1] \vee [y_2 x_1 = x_2 y_1 \ \&\ y_2 x_0 > x_2 y_0].$$

In particular, if $x \in C(a; h)$ and $2h^2 < a$, then there are unique x_0, x_1, x_2 such that in reduced form,

$$x = \frac{x_0 + x_1 a}{x_2} \quad (|x_0|, |x_1|, |x_2| \le h). \tag{42}$$

Proof is immediate from Lemma 6.1.1, since

$$\frac{x_0 + x_1 a}{x_2} > \frac{y_0 + y_1 a}{y_2} \iff y_2 x_0 + y_2 x_1 a > x_2 y_0 + x_2 y_1 a. \qquad \dashv$$

We will sometimes save a few words by referring to (42) as a *canonical representation* of x in $C(a; h)$, when $2h^2 < a$. (This is unique only if the fraction is assumed to be in lowest terms.)

7.0.5. Lemma. *If $x, y \in C(a; h)$, $x \ge y > 0$, $h \ge 2$ and $h^9 < a$, then $\mathrm{iq}(x, y), \mathrm{rem}(x, y) \in C(a; h^6)$.*

Proof. The hypothesis implies that $2h^2 < a$, and so we have canonical representations

$$x = \frac{x_0 + x_1 a}{x_2}, \quad y = \frac{y_0 + y_1 a}{y_2}$$

of x and y in $C(a; h)$. Suppose

$$x = yq + r \qquad (0 \le r < y)$$

is the canonical division equation for x, y. We consider two cases.

Case 1, $y_1 = 0$. Now $y = \dfrac{y_0}{y_2} \le h$, and so

$$r = \mathrm{rem}(x, y) < h.$$

Solving for q (and keeping in mind that $r \in \mathbb{N}$), we have

$$q = \mathrm{iq}(x, y) = \frac{y_2}{y_0} \cdot \Big[\frac{x_0 + x_1 a}{x_2} - r\Big] = \frac{(y_2 x_0 - y_2 x_2 r) + y_2 x_1 a}{x_2 y_0},$$

and the (potentially) highest coefficient in this expression is

$$|y_2 x_0 - y_2 x_2 r| \le h^2 + h^2 h \le 2h^3 \le h^4 < h^6.$$

Case 2, $y_1 \neq 0$. We must now have $y_1 > 0$, otherwise $y < 0$ by Lemma 7.0.4. Moreover,

$$y \cdot (2x_1y_2) = \frac{2y_0x_1y_2 + 2y_1x_1y_2a}{y_2} > \frac{x_0 + x_1a}{x_2} \geq yq$$

by the same Lemma, because $y_2x_1 < 2x_2y_1x_1y_2$, and the Lemma applies since (for the potentially largest coefficient),

$$2|2y_1x_1y_2|^2 \leq 2^3h^6 \leq h^9 < a.$$

It follows that

$$q = \mathrm{iq}(x, y) \leq 2x_1y_2 \leq h^3,$$

and then solving the canonical division equation for r, we get

$$r = \mathrm{rem}(x, y) = \frac{(y_2x_0 - x_2y_1q) + (y_2x_1 - x_2y_1q)a}{x_2y_2}.$$

The (potentially) highest coefficient in this expression is

$$|y_2x_0 - x_2y_1q| \leq h^2 + h^2h^3 \leq h^6. \qquad \dashv$$

7.0.6. Lemma. *For every m, if*

$$2^{6^{m+2}} < a \text{ and } G_m(a) = G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a),$$

then $G_m(a) \subseteq C(a; 2^{6^m})$.

Proof is by induction on m, the basis being trivial since $2^{6^0} = 2$ and $G_0(a) = \{0, 1, a\} \subseteq C(a; 2)$. For the induction step, set $h = 2^{6^m}$ to save typing exponents, and notice that $h \geq 2$, and

$$h^9 = \left(2^{6^m}\right)^9 = 2^{9 \cdot 6^m} < 2^{6^{m+2}} < a.$$

Thus by Lemmas 7.0.3 and 7.0.5, the value of any operation in $\mathbf{Lin}_0[\div]$ with arguments in $C(a; h)$ is in $C(a; h^6)$, and

$$h^6 = \left(2^{6^m}\right)^6 = 2^{6^{m+1}}. \qquad \dashv$$

7.0.7. Lemma ($\mathbf{Lin}_0[\div]$-Imbedding). *If $G_m(a) = G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a)$, and*

$$2^{6^{m+3}} < a \text{ and } a! \mid (\lambda - 1),$$

then there is an imbedding

$$\pi : (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) \restriction G_m(a) \rightarrowtail (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div])$$

such that $\pi(a) = \lambda a$.

Proof. Set

$$h = 2^{6^{m+1}},$$

so that from the hypothesis (easily), $2h^2 < a$, and then by Lemma 7.0.4, each $x \in C(a; h)$ can be expressed uniquely in the form

$$x = \frac{x_0 + x_1a}{x_2}$$

with all the coefficients $\le h$. We first set

$$\rho(x) = \rho\Big(\frac{x_0 + x_1 a}{x_2}\Big) = \frac{x_0 + x_1\lambda a}{x_2}. \quad (x \in C(a;h)).$$

The values of $\rho(x)$ are all natural numbers, since

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

so that for any $x_2 \le h \le a$,

$$x_2 \mid x_0 + x_1\lambda a \iff x_2 \mid x_0 + x_1 a,$$

by the hypothesis that $a! \mid (\lambda - 1)$. By another appeal to Lemma 7.0.4, we verify that ρ is an injection, and it is order-preserving.

The required imbedding is the restriction

$$\pi = \rho\restriction G_m(a),$$

and the verification that it respects all the operations in $\mathbf{Lin}_0$ follows along familiar lines. For addition, for example, set

$$h_1 = 2^{6^m},$$

and consider canonical representations

$$x = \frac{x_0 + x_1 a}{x_2}, \quad y = \frac{y_0 + y_1 a}{y_2}$$

of two numbers in $C(a;h_1)$. Adding the fractions, we get

$$x + y = \frac{x_0 + x_1 a}{x_2} + \frac{y_0 + y_1 a}{y_2} = \frac{(y_2 x_0 + x_2 y_0) + (y_1 x_1 + x_2 y_1)a}{x_2 y_2},$$

and we notice that the expression on the right is a canonical representation of $x + y$ in $C(a;h)$, since, with a typical coefficient,

$$|y_2 x_0 + x_2 y_0| \le h_1^5 < h^6 = h.$$

This means that

$$\pi(x+y) = \frac{(y_2 x_0 + x_2 y_0) + (y_1 x_1 + x_2 y_1)\lambda a}{x_2 y_2} = \pi(x) + \pi(y),$$

as required.

The argument that π respects the integer quotient and remainder operations is a bit more subtle, primarily because these are defined together: we need to show that

$$\text{if } \operatorname{iq}(x,y) \in G_m(a), \text{ then } \rho(\operatorname{iq}(x,y)) = \operatorname{iq}(\rho(x), \rho(y)),$$

even if $\operatorname{rem}(x,y) \notin G_m(a)$, but we cannot define one without the other. This is why we introduced ρ, which is defined on the larger set $C(a;h) \supseteq G_{m+1}(a)$, and we proceed as follows.

Assume again canonical representations of x and y in $C(a;h_1)$, and also that $x \ge y \ge 1$, we consider the correct division equation

$$x = yq + r \quad (0 \le r < y)$$

as in the proof of Lemma 7.0.5. Recall the two cases in that proof.

Case 1, $y_2 = 0$. Now $r \le h_1$, and

$$q = \mathrm{iq}(x, y) = \frac{(y_2 x_0 - y_2 x_2 r) + y_2 x_1 a}{x_2 y_0}$$

with all the coefficients $\le h_1^5 < h_1^6 = h$, so that this is the canonical representation of q in $C(a; h)$. It follows that

$$\rho(r) = r, \quad \rho(q) = \frac{(y_2 x_0 - y_2 x_2 r) + y_2 x_1 \lambda a}{x_2 y_0},$$

so that, by direct computation,

$$\rho(x) = \rho(y)\rho(q) + \rho(r). \tag{43}$$

Moreover, ρ is order-preserving, so that

$$0 \le \rho(r) < \rho(y),$$

and (43) is the correct division equation for $\rho(x), \rho(y)$. Thus

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

whether or not $\mathrm{iq}(x, y) \in G_m(a)$ or $\mathrm{rem}(x, y) \in G_m(a)$; but if it happens $\mathrm{iq}(x, y) \in G_m(a)$, then

$$\pi(\mathrm{iq}(x, y)) = \rho(\mathrm{iq}(x, y)) = \mathrm{iq}(\rho(x), \rho(y)) = \mathrm{iq}(\pi(x), \pi(y)),$$

independently of whether $\mathrm{rem}(x, y) \in G_m(a)$ or not. The same argument works for $\pi(\mathrm{rem}(x, y))$ and completes the proof in this case.

Case 2 is handled in the same way, and we skip it. ⊣

Recall the definition of *good examples* in Section 6.

7.0.8. THEOREM. *If a program α decides a good example $R(x)$ from $\mathbf{Lin}_0[\div]$, then for all $a \ge 2$*

$$R(a) \Longrightarrow c^p_\alpha(a) > \frac{1}{12} \log\log a.$$

PROOF. Suppose $R(a)$, let $m = c^p_\alpha(a)$, and assume that

$$2^{6^{m+3}} < a.$$

If $\lambda(\mu)$ is the polynomial which witnesses the goodness of R and

$$\lambda = \lambda(a!),$$

then Lemma 7.0.7 guarantees an imbedding

$$\pi : (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) \restriction G_m(a) \rightarrowtail (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]),$$

with $\pi a = \lambda a$; and since $\neg R(\lambda a)$, the imbedding test 6.0.3 yields a contradiction, so that

$$2^{6^{m+3}} \ge a. \tag{44}$$

Taking logarithms twice, we get from this

$$m + 3 \ge \frac{\log\log a}{\log 6};$$

and since $m \geq 1$ by (25), $4m \geq m + 3$, so that we get the required

$$m \geq \frac{\log\log a}{4\log 6} > \frac{\log\log a}{12}. \qquad \dashv$$

§8. Three results from number theory. We establish in this Section three elementary results from Diophantine Approximation, which give us just what we need to establish an analog of the $\mathbf{Lin}_0[\div]$-Uniqueness Lemma 7.0.4 for canonical forms involving two numbers, when these satisfy certain conditions.

8.0.9. THEOREM (Pell's equation). *The pairs (x_n, y_n) defined by the recursion*

$$(x_1, y_1) = (3, 2), \qquad (x_{n+1}, y_{n+1}) = (3x_n + 4y_n, 2x_n + 3y_n) \tag{45}$$

satisfy Pell's equation[7]

$$x_n^2 - 2y_n^2 = 1, \tag{46}$$

and they satisfy the inequalities

$$2^n \leq 2 \cdot 5^{n-1} \leq y_n < x_n \leq 7^{n+1} \tag{47}$$

$$0 < \frac{x_n}{y_n} - \sqrt{2} < \frac{1}{2y_n^2}. \tag{48}$$

PROOF. Equation (46) is true for $n = 1$, and inductively:

$$x_{n+1}^2 - 2y_{n+1}^2 = (3x_n - 4y_n)^2 - 2(2x_n - 3y_n)^2 = x_n^2 - 2y_n^2 = 1.$$

For (47), we first check that $2^n \leq 2 \cdot 5^{n-1}$ by a trivial induction on $n \geq 1$, and then, inductively again,

$$y_{n+1} = 2x_n + 3y_n \geq 5y_n \geq 5 \cdot 2 \cdot 5^{n-1} = 2 \cdot 5^n.$$

The last part of the triple inequality is proved similarly:

$$x_{n+1} = 3x_n + 4y_n \leq 7x_n \leq 7 \cdot 7^n = 7^{n+1}.$$

The crucial, last inequality (48) holds for any pair of positive numbers which satisfies Pell's equation. To see this, suppose $x^2 - 2y^2 = 1$, and notice first that since

$$\frac{x^2}{y^2} = 2 + \frac{1}{y^2} > 2,$$

we have $\dfrac{x}{y} > \sqrt{2}$, and hence

$$\frac{x}{y} + \sqrt{2} > 2\sqrt{2} > 2;$$

now

$$(\frac{x}{y} - \sqrt{2})(\frac{x}{y} + \sqrt{2}) = \frac{1}{y^2}$$

yields the required

$$0 < \frac{x}{y} - \sqrt{2} = \frac{1}{(\frac{x}{y} + \sqrt{2})y^2} < \frac{1}{2y^2}. \qquad \dashv$$

[7] In fact these are all the positive solutions of Pell's equations, see [6].

8.1. Good approximations of irrationals. A fraction $\frac{a}{b}$ (or, for easier typing, the pair of numbers (a,b)) is a *good approximation of* an irrational number ξ, if $a \perp b$ and

$$\left|\frac{a}{b} - \xi\right| < \frac{1}{b^2}. \tag{49}$$

Theorem 8.0.9 asserts in part that there are infinitely many good approximations; this is true of all irrational numbers, and it is worth understanding in the context of what we are doing, although we will never need it in its full generality.

8.1.1. THEOREM ([6][Theorem 188). *For each irrational number $\xi > 0$, there are infinitely many pairs (x,y) of relatively prime natural numbers such that*

$$\left|\xi - \frac{x}{y}\right| < \frac{1}{y^2}.$$

Of the many proofs of this result, we outline one which (according to [6]) is due to Dirichlet.

PROOF. For any real number ξ, let

$$\lfloor\xi\rfloor = \text{the largest natural number } \le \xi.$$

This $\lfloor\xi\rfloor$ is the *house*, and $\xi - \lfloor\xi\rfloor$ is its *fractional part*, for which, clearly

$$0 \le \xi - \lfloor\xi\rfloor < 1.$$

If we divide the half-open (real) unit interval into n disjoint, equal parts,

$$[0,1) = [0,\frac{1}{n}) \cup [\frac{1}{n},\frac{2}{n}) \cup \cdots \cup [\frac{n-1}{n},1),$$

then for every ξ, the fractional part $\xi - \lfloor\xi\rfloor$ will belong to exactly one of these subintervals. Now fix a number

$$n \ge 1,$$

and apply this observation to each of the $n+1$ numbers

$$0, \xi, 2\xi, \dots, n\xi;$$

at least two of their fractional parts will be in the same subinterval of $[0,1)$, so that, no matter what the $n \ge 1$, we get

$$0 \le j < k \le n$$

such that

$$\left|j\xi - \lfloor j\xi\rfloor - (k\xi - \lfloor k\xi\rfloor)\right| < \frac{1}{n};$$

and setting $y = k - j$, $x = \lfloor k\xi\rfloor - \lfloor j\xi\rfloor$, we get

$$\left|x - y\xi\right| < \frac{1}{n}.$$

We may assume that x and y are relatively prime in this inequality, since if we divide both by $\gcd(x,y)$ the inequality persists. Moreover, since $0 < y < n$, we can divide the inequality by y to get

$$\left|\frac{x}{y} - \xi\right| < \frac{1}{ny} < \frac{1}{y^2}.$$

Notice that if $n = 1$, then this construction gives $y = 1$, $x = \lfloor \xi \rfloor$, and the rather trivial good approximation

$$\Big|\frac{\lfloor \xi \rfloor}{1} - \xi\Big| < \frac{1}{1^2}.$$

However, we have not yet used the fact that ξ is irrational, which implies that

$$0 < \Big|\frac{x}{y} - \xi\Big|,$$

so that there is a number

$$m > \frac{1}{|\frac{x}{y} - \xi|}.$$

We now repeat the construction with m instead of n, to get x_1, y_1 such that

$$\Big|\frac{x_1}{y_1} - \xi\Big| < \frac{1}{y_1^2} \quad \text{and} \quad \Big|\frac{x_1}{y_1} - \xi\Big| < \frac{1}{my_1} \le \frac{1}{y_1} < \Big|\frac{x}{y} - \xi\Big|,$$

so that $\dfrac{x_1}{y_1}$ is a better, good approximation of ξ; and repeating the construction indefinitely, we get infinitely many, distinct good approximations. ⊣

Nest comes the most important result we need, which says, in effect, that algebraic irrational numbers cannot have "too good" approximations.

8.1.2. THEOREM (Liouville's Theorem). *Suppose ξ is an irrational root of an irreducible (over $\mathbb{Q}$) polynomial $f(x)$ with integer coefficients and of degree $n \ge 2$, and let*

$$c = \sup\{|f'(x)| \mid |x - \xi| \le 1\};$$

it follows that for all pairs (x, y) of relatively prime integers,

$$\Big|\xi - \frac{x}{y}\Big| > \frac{1}{cy^n}.$$

In particular, for all relatively prime (x, y),

$$\Big|\sqrt{2} - \frac{x}{y}\Big| > \frac{1}{5y^2}.$$

PROOF. Using the fact that $f(\xi) = 0$ and the Mean Value Theorem, we compute, for any $\frac{x}{y}$ within 1 of ξ,

$$|f(\frac{x}{y})| = |f(\xi) - f(\frac{x}{y})| \le c|\xi - \frac{x}{y}|.$$

Moreover, $f(\frac{x}{y}) \ne 0$, since $f(x)$ does not have any rational roots, and $y^n f(\frac{x}{y})$ is an integer, since all the coefficients of $f(x)$ are integers and the degree of $f(x)$ is n; thus

$$1 \le |y^n f(\frac{x}{y})| \le y^n c|\xi - \frac{x}{y}|,$$

from which we get the desired inequality (noticing that it must be strict, since ξ is not rational).

For the special case $\xi = \sqrt{2}$, we have $f(x) = x^2 - 2$, so that $f'(x) = 2x$ and

$$c = \sup\{2x \mid |\sqrt{2} - x| \le 1\} = 2(\sqrt{2} + 1) < 5.$$ ⊣

Liouville's Theorem implies immediately that good approximations of a non-rational algebraic number cannot be too-well approximated by fractions with a much smaller denominator. For concretenss, we establish this only for $\sqrt{2}$, since this is all we will need.

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

$$2 \le b < a < 2b, \tag{50}$$

and for all y, z,

$$0 < |z| < \frac{b}{\sqrt{10}} \implies \Big|\frac{a}{b} - \frac{y}{z}\Big| > \frac{1}{10z^2}. \tag{51}$$

8.2.1. LEMMA. (1) *Every good approximation of $\sqrt{2}$ other than 1 is a difficult pair; in particular, every solution (a, b) of Pell's equation is a difficult pair.*

(2) *If (a, b) is a difficult pair, then for all y, z,*

$$0 < |z| < \frac{b}{\sqrt{10}} \implies |za + yb| > \frac{b}{10|z|}. \tag{52}$$

PROOF. (1) Let (a, b) be a good approximation of $\sqrt{2}$ with $b \ge 2$. Then (50) follows from

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

To prove (51), suppose $0 < |z| < \dfrac{b}{\sqrt{10}}$, and use Liouville's Theorem 8.1.2:

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

The result holds for all solutions of Pell's equation because the proof of (48) was based only on the hypothesis $x^2 = 1 + 2y^2$.

(2) is very useful and easy: assuming the hypothesis of (52),

$$|za + yb| = |z|b\Big|\frac{a}{b} + \frac{y}{z}\Big| > |z|b\frac{1}{10z^2} = \frac{b}{10|z|}. \qquad \dashv$$

Problems for Section 8

x8.1. Prove that the pairs of numbers (x_n, y_n) defined in the proof of Theorem 8.0.9 are all the positive solutions of the Pell equation $a^2 = 1 + 2b^2$.

In the next three problems we outline a proof that *pairs (F_{k+1}, F_k) of successive Fibonacci numbers are difficult*, for $k \ge 2$. We recall from the problems of Section 1 that

$$\rho = \frac{1 + \sqrt{5}}{2}, \quad \lambda = \frac{1 - \sqrt{5}}{2}$$

are the two solutions of the quadratic equation $x + 1 = x^2$, so that

$$1 < \rho < 2, \quad \lambda < 0, \quad |\lambda| < 1,$$

and that the Fibonacci numbers are explicitly defined in terms of these,

$$F_k = \frac{\rho^k - \lambda^k}{\sqrt{5}}.$$

x8.2. Show that for $k \geq 1$,

$$\text{if } k \text{ is even, then } \rho < \frac{F_{k+1}}{F_k}, \text{ and if } k \text{ is odd, then } \frac{F_{k+1}}{F_k} < \rho.$$

Hint: Use the equation

$$\frac{F_{k+1}}{F_k} = \rho R(k) \text{ where } R(k) = \frac{1 - \frac{\lambda^{k+1}}{\rho^{k+1}}}{1 - \frac{\lambda^k}{\rho^k}},$$

and compute the sign and size of $R(k)$ for odd and even k.

x8.3. Show that for all $n \geq 1$,

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n. \tag{53}$$

Infer that

$$\left|\frac{F_{n+1}}{F_n} - \rho\right| < \frac{1}{F_n^2} \quad (n \geq 1).$$

x8.4. Prove that for each $n \geq 2$, the pair (F_{n+1}, F_n) is a difficult pair.

Hint: The golden mean ρ is a root of the polynomial $f(x) = x^2 - x - 1$. Use Liouville's Theorem 8.1.2 to show that for all coprime x, y,

$$\left|\frac{x}{y} - \rho\right| > \frac{1}{5y^2},$$

and then imitate the proof of(1) of Lemma 8.2.1 with ρ in place of $\sqrt{2}$.

The definition of *difficult pair* is tailor made to the good approximations of $\sqrt{2}$, and it is only a lucky coincidence that it also applies to pairs of successive Fibonacci numbers. It is, however, quite easy to fix the constants hard-wired in it so that it applies to the good approximations of any quadratic irrational, and then use it to extend the results in the next section to this more general case.

x8.5 (Liouville). Prove that the following number is *transcedental* (i.e., not algebraic):

$$\xi = \sum_{i=1}^{\infty} \frac{1}{10^{i!}} = \frac{1}{10} + \frac{1}{10^{2!}} + \frac{1}{10^{3!}} + \frac{1}{10^{4!}} + \cdots \tag{54}$$

Hint: Write each of the partial sums as a proper fraction,

$$\xi_n = \sum_{i=1}^{i=n} \frac{1}{10^{i!}} = \frac{p_n}{q_n},$$

and prove that for all n,

$$\left|\xi - \frac{p_n}{q_n}\right| < \frac{2}{(q_n)^n};$$

then invoke Liouville's Theorem.

§9. The complexity of coprimeness from $\mathbf{Lin}_0[\div]$. We can now combine the methods from Sections 6.3 and 7, to derive a double-log lower bound for the complexity of recursive programs which decide coprimeness from $\mathbf{Lin}_0[\div]$. The key is the following Uniqueness Lemma for linear combinations of a difficult pair.

9.0.2. LEMMA. *Suppose (a, b) is a difficult pair, $1 \leq \lambda \in \mathbb{N}$, and*

$$|x_3 y_i|, |y_3 x_i| < \frac{\sqrt{b}}{2\sqrt{10}}$$

for $i = 0, 1, 2, 3$ with $x_3, y_3 > 0$. Then

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

$$\frac{x_0 + x_1\lambda a + x_2\lambda b}{x_3} > \frac{y_0 + y_1\lambda a + y_2\lambda b}{y_3} \iff [y_3(x_1a + x_2b) > x_3(y_1a + y_2b)] \text{ or } \Big([y_3(x_1a + x_2b) = x_3(y_1a + y_2b)] \ \&\ y_3x_0 > x_3y_0\Big).$$

PROOF. The claimed equivalences follow from the following two facts, applied to $(y_3x_0 - x_3y_0) + (y_3x_1 - x_3y_1)\lambda a + (y_3x_2 - x_3y_2)\lambda b$.

(1) *If $x + z\lambda a + y\lambda b = 0$ and $|x|, |y|, |z| < \dfrac{\sqrt{b}}{\sqrt{10}}$, then $x = y = z = 0$.*

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

$$b \leq \lambda|y|b = |x| < \frac{\sqrt{b}}{\sqrt{10}},$$

which is absurd. So we may assume that $z \neq 0$. Now the assumed bound on z and (52) implies

$$|z\lambda a + y\lambda b| \geq |za + yb| > \frac{b}{10|z|} \geq |x|$$

the last because

$$|xz| \leq \frac{\sqrt{b}}{\sqrt{10}}\frac{\sqrt{b}}{\sqrt{10}} = \frac{b}{10};$$

and this contradicts the hypothesis $|z\lambda a + y\lambda b| = |-x|$.

(2) *If $|x|, |y|, |z| < \dfrac{\sqrt{b}}{\sqrt{10}}$, then*

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

Proof. If $z = 0$, then the equivalence follows from Lemma 6.1.1; and if $z \neq 0$, then $|z\lambda a + y\lambda b| > |x|$ as above, and so adding x to $z\lambda a + y\lambda b$ cannot change its sign. ⊣

9.0.3. LEMMA. *Suppose* (a, b) *is a difficult pair,* $h \geq 2$, $X, Y \in C(a, b; h)$, *and* $h^{28} \leq b$. *Then* $\mathrm{iq}(X, Y), \mathrm{rem}(X, Y) \in C(a, b; h^{12})$.

PROOF. Let us notice immediately that (by a simple computation, using $2 \leq h$) the assumption $h^{28} \leq b$ implies that

$$h^2 < \frac{\sqrt{b}}{2\sqrt{10}}; \tag{55}$$

this will allow us to appeal to Lemma 9.0.2 in several parts of the argument, and the more outrageous-looking $h^{28} \leq b$ will be needed for one more, specific application of the same Lemma. In effect, we just need to assume that h is sufficiently smaller than b to justify these appeals to Lemma 9.0.2, and the 28th power is what makes this particular argument work.

It is enough to prove the result when $X \geq Y > 0$, since it is trivial when $X < Y$. Suppose

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

where all $|x_i|, |y_i| \leq h$, and $x_3, y_3 > 0$, and consider the Correct Division Equation

$$\frac{x_0 + x_1 a + x_2 b}{x_3} = \frac{y_0 + y_1 a + y_2 b}{y_3} Q + R \quad (0 \leq R < Y). \tag{56}$$

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

Case 1, $y_1 a + y_2 b = 0$. Now (56) takes the form

$$\frac{x_0 + x_1 a + x_2 b}{x_3} = \frac{y_0}{y_3} Q + R \quad (0 \leq R < \frac{y_0}{y_3}),$$

so that $R < h$. Solving (56) for Q, we get in this case

$$Q = \frac{y_3}{y_0} \frac{(x_0 - x_3 R) + x_1 a + x_2 b}{x_3} \in C(a, b; h^4). \tag{57}$$

Case 2, $y_1 a + y_2 b \neq 0$. Then $y_1 a + y_2 b > 0$, by Lemma 9.0.2, since $Y > 0$, using (55). We are going to show that in this case

$$h^9 Y > X \tag{58}$$

so that $Q \leq h^9$. Assuming this, we can solve the division equation (56) for R, to get

$$R = \frac{(x_0 y_3 - y_0 x_3 Q) + (x_1 y_3 - y_1 x_3 Q) a + (x_2 y_3 - y_2 x_3 Q) b}{x_3 y_3}; \tag{59}$$

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

We show (58) by separately comparing the "infinite parts" (those involving a and b) of X and Y with b. Compute first:

$$y_3(x_1 a + x_2 b) \leq |y_3 x_1| a + |y_3 x_2| b \leq h^2 2b + h^2 b = 3h^2 b \leq h^4 b, \tag{60}$$

using $a < 2b$. On the other hand, if $y_2 = 0$, then $y_1 > 0$ and so

$$x_3(y_1 a + y_2 b) = x_3 y_1 a > b;$$

and if $y_2 \neq 0$, then by (52),

$$(y_1 a + y_2 b) > \frac{b}{10|y_1|}, \quad \text{so that} \quad 10|y_1|(y_1 a + y_2 b) > b,$$

and hence (since $10 < 2^4$), in either case,

$$h^5(y_1 a + y_2 b) > b. \tag{61}$$

Now (60) and (61) imply that

$$h^9 x_3(y_1 a + y_2 b) > h^4 h^5 (y_1 a + y_2 b) > h^4 b \geq y_3(x_1 a + x_2 b),$$

and we can finish the proof of (58) with an appeal to Lemma 9.0.2, provided that the coefficients of $h^9 Y$ and X in canonical form satisfy the hypotheses of this Lemma, which (for the worst case) are

$$|x_3 h^9 y_i| \leq \frac{\sqrt{b}}{2\sqrt{10}},$$

which is implied by

$$h^{11} \leq \frac{\sqrt{b}}{2\sqrt{10}};$$

if we square this and simplify (using that $40 < 2^6$), we see that it is implied by the assumed $h^{28} \leq b$. ⊣

9.0.4. Lemma (Inclusion). *Suppose (a, b) is a difficult pair, and for any m, let $G_m(a, b) = G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a, b)$; it follows that*

$$\text{if } 2^{2^{4m+5}} \leq a, \text{ then } G_m(a, b) \subseteq C(a, b; 2^{2^{4m}}).$$

Proof is by induction on m, the case $m = 0$ being trivial. To apply Lemmas 7.0.3 and 9.0.3 at the induction step, we need to verify (under the hypothesis on a and m) the following two inequalities.

(1) $\left(2^{2^{4m}}\right)^{12} \leq 2^{2^{4(m+1)}}$. 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) $\left(2^{2^{4m}}\right)^{28} \leq b$. So compute:

$$\left(2^{2^{4m}}\right)^{28} = 2^{28 \cdot 2^{4m}} < 2^{2^5 \cdot 2^{4m}} = 2^{2^{4m+5}} \leq a.$$ ⊣

9.0.5. Lemma (Imbedding). *Suppose (a, b) is a difficult pair, $2^{2^{4m+6}} \leq a$, and set $\lambda = 1 + a!$. Then there is an algebra imbedding*

$$\pi : (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) \restriction C(a, b; 2^{2^{4m}}) \rightarrowtail (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div])$$

such that $\pi(a) = \lambda a, \pi(b) = \lambda b$

PROOF. To simplify notation, let

$$h = 2^{2^{4m}}.$$

As in the proof of Lemma 7.0.7, we will actually need to define the imbedding on a somewhat larger subalgebra, $C(a, b; h^{12})$ to be precise, so let's first verify that the assumed bound on h is good enough to insure unique canonical forms in $C(a, b; h^{12})$: by Lemma 9.0.2, we need to check that

$$\left(h^{12}\right)^2 < \frac{\sqrt{b}}{2\sqrt{10}},$$

which is equivalent to

$$4 \cdot 10h^{48} < b; \tag{62}$$

and this is true, because

$$4 \cdot 10h^{49} < 2^2 \cdot 2^4 h^{49} \leq h^{55} = \left(2^{2^{4m}}\right)^{55} < 2^{2^6 \cdot 2^{4m}} = 2^{2^{4m+6}} < a,$$

by the hypothesis, and it yields (62) when we divide both of its sides by h.

Using Lemma 9.0.2 now, we define

$$\rho : C(a, b; h^{12}) \to \mathbb{N},$$

in the expected way,

$$\rho\Big(\frac{x_0 + x_1 a + x_2 b}{x_3}\Big) = \frac{x_0 + x_1 \lambda a + x_2 \lambda b}{x_3},$$

and we verify as in the proof of Lemma 7.0.7 that this is well-defined, order-preserving injection, with values in $\mathbb{N}$ (since $h < a$, and so $x_3 \mid \lambda - 1$), and it respects all the operations in $\mathbf{Lin}_0$. We let

$$\pi = \rho \restriction G_m(a, b),$$

and all that it remains is to show that π respects $\mathrm{iq}(x, y)$ and $\mathrm{rem}(x, y)$ when they are defined in $G_m(a, b)$: the key fact is that by Lemma 9.0.3 and the bound on h^{12},

$$X, Y \in G_m(a, b) \Longrightarrow \mathrm{iq}(X, Y), \mathrm{rem}(X, Y) \in C(a, b; h^{12}).$$

Thus it is enough to show that if

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

is the correct division equation for X, Y, then

$$\rho X = \rho Y \cdot \rho Q + \rho R \quad (0 \leq \rho R < \rho Y) \tag{63}$$

is the correct division equation for $\rho X, \rho Y$. We distinguish two cases, following the proof of Lemma 9.0.3.

Case 1, $y_1 a + y_2 b = 0$. Then $0 \leq R < Y = \dfrac{y_0}{y_3} \leq h$, so $\rho R = R$ and $\rho Y = Y$. Now $\rho R < \rho Y$ since ρ is order-preserving. The explicit formula (57) for Q yields

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

and a direct computation with these expressions for ρR, ρY and ρQ yields (63).

Case 2, $y_1 a + y_2 b > 0$. Now $Q \leq h^9$, which implies $\rho Q = Q$. The explicit formula (59) for R yields

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

with these expressions for ρR and ρQ we get again (63) by direct computation.⊣

9.0.6. THEOREM ([3]). *If a recursive program α decides the coprimeness relation $x \perp y$ from $\mathbf{Lin}_0[\div]$, then for all difficult pairs (a, b),*

$$c^p_\alpha(a,b) > \frac{1}{10}\log\log a. \tag{64}$$

PROOF. Suppose α decides coprimeness and $c^p_\alpha(a,b) = m$ for some difficult pair (a, b). If

$$2^{2^{4m+6}} \leq a,$$

then Lemma 9.0.5 provides an imbedding π for which $\pi a = \lambda a$ and $\pi b = \lambda b$, with some λ, so that by the imbedding test Lemma 6.0.3, we must have

$$2^{2^{4m+6}} > a;$$

in other words

$$4m + 6 \geq \log\log a;$$

and since $m \geq 1$ by (25),

$$10m \geq 4m + 6 \geq \log\log a,$$

as required. ⊣

9.0.7. COROLLARY. *If a recursive program α computes* $\gcd(x, y)$ *from* $\mathbf{Lin}_0[\div]$, *then for every difficult pair,*

$$c^p_\alpha(a,b) \geq \frac{1}{10}\log\log a.$$

PROOF. Suppose α satisfies the hypothesis, with head symbol g, and add to the program the single equation

$$f(x,y) = \text{if } (g(x,y) = 1) \text{ then } 1 \text{ else } 0.$$

This new program β decides the coprimeness relation, and clearly

$$c^p_f(x,y) = c_g(x,y) + 1,$$

so that the Theorem gives immediately the Corollary. ⊣

§10. Polynomials and their heights. To derive lower bounds from $\mathbf{Lin}_0[\div, \cdot]$ (which includes multiplication among the primitives) in the next section, we need some basic facts about the ring of (unary) polynomials $K[T]$ over a field K. We review these here, and we also derive some equally simple facts about the ring $\mathbb{Z}[T]$ of polynomials over the integers which are not always covered in algebra classes.

To fix terminology, a *polynomial* in the "indeterminate" T over a ring K is a "formula"

$$X = x_0 + x_1 T + x_1 T^2 + \cdots + x_n T^n,$$

where $x_i \in K$ and $x_n \neq 0$. It is sometimes useful to think of X as an infinite sum of monomials x_iT^i, in which only finitely many of the x_i's are not 0; however we do this, the *degree* of X is the largest power of T which appears in X (with a non-zero coefficient), and it is 0 when $X = x_0$ is just an element of K. (In particular, we assign here the degree 0 to the *zero polynomial*, in which all coefficients are 0—it is, sort of, the "empty formula".) Two polynomials are equal if they are literally identical as formulas., i.e., if the coefficients of like powers are equal.

The *sum*, *difference* and *product* of two polynomials are defined by the performing the obvious operations on the coefficients and collecting terms:

$$
\begin{aligned}
X + Y &= \sum_i (x_i + y_i)T^i, &\qquad& \deg(X+Y) \leq \max(\deg(X), \deg(Y)) \\
-X &= \sum_i (-x_i)T^i, && \deg(-X) = \deg(X) \\
XY &= \sum_k \Big(\sum_{i=0}^{i=k} x_i y_{k-i}\Big)T^k && \deg(XY) = \deg(X)\deg(Y).
\end{aligned}
$$

The last formula illustrates the occasional usefulness of thinking of a polynomial as an infinite sum with just finitely many non-zero terms.

With these operations, the set $K[T]$ of polynomials over a ring K is a (commutative) ring over K. For the more interesting division operation, we need to assume that K is a field.

10.0.8. Theorem (The Division Theorem for polynomials). *If K is a field, and $X, Y \in K[T]$ such that $\deg(X) \geq \deg(Y)$ and $Y \neq 0$, then there exist unique polynomials $Q, R \in K[T]$ such that*

$$
X = YQ + R \quad \text{and } R = 0 \text{ or } \deg(R) < \deg(Y). \tag{65}
$$

Proof is by induction on the difference $d = n - m$ of the degrees of the given polynomials, $n = \deg(X)$, $m = \deg(Y)$.

At the basis, if $m = n$, then

$$
X = Y\frac{x_n}{y_n} + R
$$

with R defined by this equation, so that either it is 0 or its degree is less than n, since X and $\dfrac{x_n}{y_n}Y$ have the same highest term x_nT^n.

In the induction step, with $d = n - m > 0$, first we divide X by YT^d, the two having the same degree:

$$
X = YT^dQ_1 + R_1 \quad (R_1 = 0 \text{ or } \deg(R_1) < n).
$$

If $R_1 = 0$ or $\deg(R_1) < m$, we are done; otherwise $\deg(R_1) \geq \deg(Y)$ and we can apply the induction hypothesis to get

$$
R_1 = YQ_2 + R_2 \quad (R_2 = 0 \text{ or } \deg(R_2) < \deg(Y)).
$$

We now have

$$
X = Y(T^dQ_1 + Q_2) + R_2 \quad (R_2 = 0 \text{ or } \deg(R_2) < \deg(Y)),
$$

which is what we needed.

We skip the proof of uniqueness, which basically follows from the construction. ⊣

This basic fact does not hold for polynomials in $\mathbb{Z}[T]$: for example, if $X = 3$ and $Y = 2$, then there are no Q, R which satisfy (65), simply because 2 does no divide 3 in $\mathbb{Z}$. To get at the results we need, it is most convenient to work with the larger ring $\mathbb{Q}[T]$, but study a particular "presentation" of it, in which the concept if *height* is made explicit.

The *height* of an integer polynomial is the maximum of the absolute values of its coefficients,

$$\text{height}(x_0 + x_1 T + \cdots + x_n T^n) = \max\{|x_i| \mid i = 0, \dots, n\} \quad (x_i \in \mathbb{Z}).$$

To extend the definition to $\mathbb{Q}[T]$, we let for each $n, h \in \mathbb{N}$,

$$(66) \quad Q_n(T; h) = \{\frac{x_0 + x_1 T + x_2 T^2 + \cdots + x_n T^n}{x^*} \mid \\ x_0, \dots, x_n, x^* \in \mathbb{Z}, x^* > 0 \text{ and } |x_0|, \dots, |x_n|, |x^*| \leq h\}.$$

This is the set of polynomials in the indeterminate T over $\mathbb{Q}$, with degree n and *height* no more than h. When the degree is not relevant, we skip the subscript,

$$Q(T; h) = \bigcup_n Q_n(T; h);$$

and in computing heights, it is sometimes convenient to use the abbreviation

$$X : h \iff X \in Q(T; h).$$

The *canonical form* (66) gives a unique height(X) if the coefficients x_i have no common factor with x^*, but this is not too important: most of the time we only care for an upper bound for height(X) which can be computed without necessarily bringing X to canonical form. Notice however, that (as a polynomial over $\mathbb{Q}$),

$$X = \frac{3 + 2T}{6} = \frac{1}{2} + \frac{1}{3}T,$$

but the height of X is neither 3 nor $\frac{1}{3}$; it is 6.

It is very easy to make "height estimates" for sums and products of polynomials:

10.0.9. Lemma. *If X, Y are in $\mathbb{Q}[T]$ with respective degrees n and m and $X : H, Y : h$, then*

$$X + Y : 2Hh, \quad XY : (n + m)Hh.$$

Proof. For addition,

$$X + Y = \frac{(y^* x_0 + x^* y_0) + (y^* x_1 + x^* y_1)T + \cdots}{x^* y^*},$$

and every term in the numerator clearly has absolute value $\leq 2Hh$. For multiplication,

$$XY = \frac{\sum_{k=0}^{n+m} \left(\sum_{i=0}^{i=k} x_i y_{k-i} \right) T^k}{x^* y^*}.$$

For $k < n+m$, the typical coefficient in the numerator can be estimated by

$$|\sum_{i=0}^{i=k} x_i y_{k-i}| \leq \sum_{i=0}^{i=k} Hh \leq (k+1)Hh \leq (n+m)Hh;$$

and if $k = n+m$, then

$$|\sum_{i=0}^{i=n+m} x_i y_{k-i}| = |x_n y_m| \leq Hh < (n+m)Hh$$

since $x_i = 0$ when $i > n$ and $y_j = 0$ when $j > m$. ⊣

The next result is a version of the Division Theorem 10.0.8 for $\mathbb{Q}[T]$ which supplies additional information about the heights.

10.0.10. Lemma (Lemma 2.3 of [9]). *Suppose X and Y are polynomials with integer coefficients,* $\deg(X) = n \geq m = \deg(Y)$, $X : H$, $Y : h$, *and* (65) *holds. Then*

$$Q = \frac{Q_1}{y_m^{d+1}}, \quad R = \frac{R_1}{y_m^{d+1}},$$

where $d = n - m$ and Q_1, R_1 are in $\mathbb{Z}[T]$ with height $\leq H(2h)^{d+1}$. It follows that

$$Q, R : H(2h)^{d+1}.$$

Proof is by induction on d. Basis, $\deg(X) = \deg(Y) = n$. In this case

$$X = Y\frac{x_n}{y_n} + R$$

with R defined by this equation, so that either it is 0 or it is of degree $< n$. Now $Q_1 = \dfrac{x_n}{y_n}$ has height $\leq H$, and

$$R_1 = y_n X - x_n Y$$

so that the typical coefficient of R_1 is of the form $y_n x_i - x_n y_i$, and the absolute value of this is bounded by $2Hh = H(2h)^{0+1}$.

Induction Step, $d = \deg(X) - \deg(Y) = n - m > 0$. Consider the polynomial

$$Z = y_m X - x_n Y T^d \tag{67}$$

whose degree is $< n = m + d$ since the coefficient of T^n in it is $y_m x_n - x_n y_m$. If $Z = 0$ or $\deg(Z) < m$, then

$$X = Y\frac{x_n T^d}{y_m} + \frac{Z}{y_m} = Y\frac{x_n y_m^d T^d}{y_m^{d+1}} + \frac{y_m^d Z}{y_m^{d+1}}$$

is the **c.d.e.** for X, Y, and it follows easily that

$$Q_1 = x_n y_m^d T^d : Hh^d < H(2h)^{d+1},$$
$$R_1 = y_m^d Z = y_m^{d+1} X - x_n y_m^d Y T^d : H(2h)^{d+1}.$$

This proves the Lemma for this case, and so we may assume that $\deg(Z) \geq m$ and apply the Induction Hypothesis on the pair Z and Y since, evidently,

$$\deg(Z) - \deg(Y) < n - m = d.$$

Now

$$\text{height}(Z) \leq hH + Hh = H(2h),$$

and so the Induction Hypothesis yields

(68) $$Z = Y\frac{Q_2}{y_m^d} + \frac{R_2}{y_m^d},$$

with

$$Q_2, R_2 : H(2h)(2h)^d = H(2h)^{d+1}.$$

Solving (67) for X, we get

$$\begin{aligned} X &= \frac{1}{y_m}Z + \frac{x_n}{y_m}T^dY \\ &= \frac{1}{y_m}\Big(Y\frac{Q_2}{y_m^d} + \frac{R_2}{y_m^d}\Big) + \frac{x_n}{y_m}T^dY \\ &= \frac{Q_2 + x_n y_m^d T^d}{y_m^{d+1}}Y + \frac{R_2}{y_m^{d+1}} \end{aligned}$$

which is the **c.d.e.** for X, Y. We have already computed that $R_2 : H(2h)^{d+1}$. To verify that $Q_2 + x_n y_m^d T^d : H(2h)^{d+1}$, it is sufficient to notice that $\deg(Q_2) < d$, since the opposite assumption implies with (68) that $\deg(Z) \geq m+d = n$, which contradicts the definition of Z; thus the coefficients of $Q_2 + x_n y_m^d T^d$ are the coefficients of Q_2 and $x_n y_m^d$, and they all have height $\leq H(2h)^{d+1}$, as required.⊣

10.0.11. THEOREM. *If $X, Y : h$, $\deg(X) \geq \deg(Y)$ and (65) holds, then*

$$Q, R : (2h)^{2n+5}.$$

PROOF. Theorem 10.0.10 applied to y^*X and x^*Y (with height $\leq h^2$) yields

$$y^*X = x^*Y\frac{Q}{(x^*y_m)^{d+1}} + \frac{R}{(x^*y_m)^{d+1}}$$

with $Q, R : h^2(2h^2)^{d+1} < (2h)^{2d+4}$; and if we divide by y^*, we get that

$$X = Y\frac{x^*Q}{y^*(x^*y_m)^{d+1}} + \frac{R}{y^*(x^*y_m)^{d+1}},$$

which is the **c.d.e.** for X, Y, and such that (with $d \leq n$)

$$\frac{x^*Q}{y^*(x^*y_m)^{d+1}}, \frac{R}{y^*(x^*y_m)^{d+1}} : (2h)^{2n+5}.$$ ⊣

§11. Unary relations from $\mathbf{Lin}_0[\div, \cdot]$. Let

(69) $$\mathbf{Lin}_0[\div, \cdot] = \mathbf{Lin}_0 \cup \{\text{iq}, \text{rem}, \cdot\} = \{=, <, \text{parity}, \text{em}_2, \text{iq}_2, +, \dot{-}, \text{iq}, \text{rem}, \cdot\}$$

be the set of functions in $\mathbf{Lin}_0[\div]$ augmented with multiplication. In this section we will establish suitable versions of Lemma 7.0.7 and Theorem 7.0.8 for the algebra

$$(\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div, \cdot]) = (N, 0, 1, =, <, \text{parity}, \text{em}_2, \text{iq}_2, +, \dot{-}, \text{iq}, \text{rem}, \cdot)$$

with a $\sqrt{\log\log}$ bound.

Set, for any $a, n, h \in \mathbb{N}$,

$$(70) \quad Q_n(a;h) = \{\frac{x_0 + x_1 a + x_2 a^2 + \cdots + x_n a^n}{x^*} \in \mathbb{N} \\ \mid x_0, \ldots, x_n, x^* \in \mathbb{Z}, x^* > 0 \text{ and } |x_0|, \ldots, |x_n|, |x^*| \leq h\}.$$

These are the values for $T := a$ of polynomials in $Q_n(T;h)$, but only those which are natural numbers; and they are the sort of numbers which occur (with various values of h) in the generated subalgebras $G_m(a)$ of $(\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div, \cdot])$. To help deal with them, we will be using the standard notations

$$(71) \quad x = f(a) = \frac{x_0 + x_1 a + x_2 a^2 + \cdots + x_n a^n}{x^*}, \\ y = g(a) = \frac{y_0 + y_1 a + y_2 a^2 + \cdots + y_m a^m}{y^*},$$

where it is assumed that $x_n, y_m \neq 0$ (unless, of course, $x = 0$, in which case, by convention, $n = 0$ and $x_0 = 0$). It is also convenient to set $x_i = 0$ for $i > n$, and similarly for y_j, and to use the same abbreviations we set up for $Q_n(T;h)$, especially

$$Q(a;h) = \bigcup_n Q_n(a;h), \quad x : h \iff x \in Q(a;h).$$

11.0.12. Lemma. *With x and y as in* (71)*, if $h \geq 2$ and $x, y \in Q_n(a;h)$, then $em_2(x), \mathrm{iq}_2(x), x + y, x \dot{-} y \in Q_n(a;h^3)$, and $xy \in Q_{2n}(a;nh^3)$.*

Proof. These are all immediate, using Lemma 10.0.9. ⊣

The analogous estimates for $\mathrm{iq}(x,y)$ and $\mathrm{rem}(x,y)$ are substantially more complex, and we need to establish first the uniqueness of the representations (71) when h is small relative to a.

11.0.13. Lemma. *(1) With all $x_i \in \mathbb{Z}$ and $a > 2$, if $|x_i| < a$ for $i \leq n$, then*

$$x_0 + x_1 a + \cdots + x_n a^n = 0 \iff x_0 = x_1 = \cdots = x_n = 0;$$

and if, in addition, $x_n \neq 0$, then

$$x_0 + x_1 a + \cdots + x_n a^n > 0 \iff x_n > 0.$$

(2) With x and y as in (71) *and assuming that $2h^2 < a$:*

$$x = y \iff m = n \;\&\; (\forall i \leq n)[y^* x_i = x^* y_i],$$
$$x > y \iff (\exists k \leq n)[(\forall i > k)[x_i = y_i] \;\&\; x_k > y_k].$$

In particular,

$$x > 0 \iff x_n > 0.$$

Proof. (1) If $x_n \neq 0$, then

$$|x_0 + x_1 a + \cdots x_{n-1} a^{n-1}| \leq (a-1)(1 + a + a^2 + \cdots + a^{n-1}) \\ = (a-1)\frac{a^n - 1}{a - 1} = a^n - 1 < a^n \leq |x_n| a^n,$$

and so adding $x_0 + x_1 a + \cdots x_{n-1} a^{n-1}$ to $x_n a^n$ cannot change its sign or yield 0.

(2) follows immediately from (1). ⊣

11.0.14. Lemma. *Let $c \geq 61$, $d \geq 33$, and assume that $h \geq 2$ and $h^{c(n+1)} < a$. If $x, y \in Q_n(a; h)$ and $x \geq y > 0$, then* $\mathrm{iq}(x,y), \mathrm{rem}(x,y) \in Q_n(a; h^{d(n+1)})$.

Proof. How large c and d must be will be determined by the proof, as we put successively stronger conditions on h to justify the compuations. To begin with, assume

$$\text{(H1)} \qquad c \geq 3, \text{ so that } 2h^2 \leq h^3 \leq h^{c(n+1)} < a,$$

and Lemma 11.0.13 guarantees that the canonical representations of x an y in $Q_n(a; h)$ are unique. By the same Lemma,

$$n = \deg(f(T)) \geq \deg(g(T)) = m,$$

and so we can put down the Correct Division Equation for these polynomials in $\mathbb{Q}[T]$,

$$\text{(72)} \qquad f(T) = g(T)Q(T) + R(T) \ (R(T) = 0 \text{ or } \deg(R(T)) < \deg(g(T)).$$

We record for later use the heights from Lemma 10.0.11,

$$\text{(73)} \qquad Q(T), R(T) : (2h)^{2n+5} \leq h^{4n+10}.$$

From (72) we get

$$\text{(74)} \qquad f(a) = g(a)Q(a) + R(a) \quad (R(a) = 0 \text{ or } R(a) < g(a)),$$

where the condition on $R(a)$ is trivial if $R(a) \leq 0$, and follows by Lemma 11.0.13 from the corresponding condition about degrees in (72) by Lemma 11.0.13, provided that the height of $R(a)$ is sufficiently small, specifically

$$2\Big(h^{4n+10}\Big)^2 < a;$$

so here we assume

$$\text{(H2)} \qquad c \geq 21, \text{ so that } 2\Big(h^{4n+10}\Big)^2 \leq hh^{8n+20} = h^{8n+21} < h^{21(n+1)} < a.$$

However, (74) need not be the Correct Division Equation for the numbers $f(a), g(a)$, because $Q(a)$ might not be integral or $R(a)$ might be negative. For an example where both of these occur, suppose

$$f(a) = a^2 - 1, \quad g(a) = 2a \text{ with } a \text{ odd},$$

in which case (72) and (74) take the form

$$T^2 - 1 = 2T(\frac{T}{2}) - 1, \quad a^2 - 1 = 2a(\frac{a}{2}) - 1.$$

To correct for this problem, we consider four cases.

Case 1, $Q(a) \in \mathbb{N}$ and $R(a) \geq 0$. In this case (74) is the **c.d.e.** for $f(a)$ and $g(a)$, and from (73),

$$Q(a), R(a) : h^{4n+10};$$

thus we assume at this point

$$\text{(H3)} \qquad d \geq 10, \text{ so that } h^{4n+10} \leq h^{d(n+1)}.$$

Case 2, $Q(a) \in \mathbb{N}$ but $R(a) < 0$. From (74) we get

$$f(a) = g(a)[Q(a) - 1] + \underbrace{g(a) + R(a)},$$

and we contend that this is the **c.d.e.** for $f(a), g(a)$ in $\mathbb{N}$, if c is large enough. We must show that

$$(1) \quad 0 \leq g(a) + R(a) \text{ and } (2) \quad g(a) + R(a) < g(a),$$

and (2) is immediate, because $R(a) < 0$. For (1), notice that the leading coefficient of $g(T) + R(T)$ is the same as the leading coefficient of $g(T)$ (because $\deg(R(T) < \deg(g(T))$, and that it is positive, since $g(a) > 0$. To infer from this that $g(a) + R(a) > 0$ using Lemma 11.0.13, we must know that

$$2\text{height}(g(a) + R(a))^2 < a,$$

and from Lemma 11.0.12,

$$2\text{height}(g(a) + R(a))^2 \leq h\Big((h^{4n+10})^3\Big)^2 = h^{24n+61},$$

and so for this case we assume that

$$\text{(H4)} \qquad c \geq 61, \text{ so that } h^{24n+61} \leq h^{c(n+1)} < a.$$

Using Lemma 11.0.12, easily (and overestimating again grossly)

$$Q(a) - 1, g(a) + R(a) : (h^{4n+10})^3 = h^{12n+30} \leq h^{30(n+1)};$$

and so we also assume

$$\text{(H5)} \qquad d \geq 30, \text{ so that } h^{30(n+1)} \leq h^{d(n+1)}.$$

Case 3, $Q(a) = \dfrac{Q_1(a)}{z} \notin \mathbb{N}$, and $Q_1(a) \geq z > 1$. We note that

$$Q_1(a) : h^{4n+10}, \quad z \leq h^{4n+10},$$

and both $Q_1(a)$ and z are positive, and so we can put down the **c.d.e.** for them in $\mathbb{N}$:

$$Q_1(a) = zQ_2 + R_2 \quad (0 < R_2 < z),$$

where we know that $R_2 > 0$ by the case hypothesis. From this it follows that

$$R_2 < z \leq h^{4n+10}, \text{ and } Q_2 = \frac{Q_1(a) - R_2}{z} : \Big(h^{4n+10}\Big)^3 = h^{12n+30}$$

by Lemma 11.0.12 again. Replacing these values in (74), we get

$$\text{(75)} \qquad f(a) = g(a)Q_2 + \underbrace{g(a)\frac{R_2}{z} + R(a)}$$

and the number above the underbrace is in $\mathbb{Z}$, as the difference between two numbers in $\mathbb{N}$. This number is the value for $T := a$ of the polynomial

$$g(T)\frac{R_2}{z} + R(T)$$

whose leading coefficient is that of $g(T)$—since $\deg(R(T)) < \deg(g(T))$—and hence positive. We would like to infer from this that

$$g(a)\frac{R_2}{z} + R(a) > 0,$$

using Lemma 11.0.13, and so we make sure that its height is suitably small. From the two summands, the second has the larger height, h^{4n+10}, and so by Lemma 11.0.12, the height of the sum is bounded by

$$\big(h^{4n+10})^3 = h^{12n+30};$$

and to apply Lemma 11.0.13, we must insure that

$$2(h^{12n+30})^2 = 2h^{24n+60} < a,$$

and so for this step we assume that

$$\text{(H6)} \qquad c \geq 61, \text{ so that } 2h^{24n+60} \leq h^{24n+61} \leq h^{c(n+1)}.$$

This number is also less than $g(a)$, again because its leading coefficient is that of $g(a)$ multiplied by $\dfrac{R_2}{z} < 1$. It follows that this is the **c.d.e.** for $f(a), g(a)$, so it is enough to compute the height of the quotient (above the underbrace) since we have already computed that

$$Q_2 : h^{12n+30} \leq h^{30(n+1)};$$

using the known heights on $g(a)$, R_2, z and $R(a)$, it is easy to check that

$$g(a)\frac{R_2}{z} + R(a) : ((h)^{4n+11})^3 = h^{12n+33},$$

so at this point we assume that

$$\text{(H7)} \qquad d \geq 33, \text{ so that } h^{12n+30}, h^{12n+33} \leq h^{d(n+1)}.$$

Case 4, $Q(a) = \dfrac{Q_1(a)}{z} \notin \mathbb{N}$, and $Q_1(a) < z$. Since $Q_1(a) > 0$, this case hypothesis implies that $\deg(Q_1(T)) = 0$, so that $\deg(f(T)) = \deg(g(T))$. By now familiar arguments, this means that

$$f(a) \leq \frac{y^* x_n}{x^* y_n} g(a)$$

and so the quotient of these two numbers is some number

$$Q \leq \frac{y^* x_n}{x^* y_n} \leq h^2.$$

Thus (74) takes the form

$$f(a) = g(a)Q + R \quad \text{with } 0 \leq Q \leq h^2,$$

from which it follows immediately that

$$R = f(a) - g(a)Q : (h^5)^3 = h^{15},$$

and the hypotheses we have already made on d insure $h^{15} \leq h^{d(n+1)}$, so that we need not make any more.

In fact then, the Lemma holds with

$$c \geq 61, \quad d \geq 33. \qquad \dashv$$

11.0.15. Lemma ($\mathbf{Lin}_0[\div, \cdot]$-imbedding). *Let $e = 61$ and for any m,*

$$G_m(a) = G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div, \cdot]), a).$$

(1) If $2^{2^{e(m+1)^2}} < a$, then $G_m(a) \subseteq Q_{2^m}(a; 2^{2^{em^2}})$.
(2) If $2^{2^{e(m+2)^2}} < a$ and $a! \mid (\lambda - 1)$, then there is an imbedding

$$\pi : (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div, \cdot]) \restriction G_m(a) \rightarrowtail (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div, \cdot])$$

such that $\pi(a) = \lambda a$.

Proof of (1) is by induction on m, the basis being trivial. To apply Lemmas 11.0.12 and 11.0.14 at the induction step, we need the inequalities

$$2^m \Big(2^{2^{em^2}}\Big)^3 \leq 2^{2^{e(m+1)^2}} \quad \Big(2^{2^{em^2}}\Big)^{61(2^m+1)} < a, \quad \Big(2^{2^{em^2}}\Big)^{33(2^m+1)} \leq 2^{2^{e(m+1)^2}},$$

and these are easily verified by direct computation.

(2) is proved very much like Lemma 9.0.5, the only subtlety being that we need to start with an injection

$$\rho : G_{m+1}(a) \rightarrowtail \mathbb{N}$$

on the larger set, which (by (1)) contains $\mathrm{iq}(x, y)$ and $\mathrm{rem}(x, y)$ for all $x, y \in G_m(a)$. The stronger hypothesis on m and a imply that the numbers in $G_{m+1}(a)$ have unique canonical forms in

$$Q_{2^{m+1}}(a; 2^{2^{e(m+1)^2}});$$

and so we can set (with $n = 2^m$)

$$\rho\Big(\frac{x_0 + x_1 a + \cdots x_n a^n}{x^*}\Big) = \frac{x_0 + x_1 \lambda a + \cdots x_n \lambda a^n}{x^*},$$

take $\pi = \rho \restriction G_m(a)$ and finish the proof as in Lemma 9.0.5. $\dashv$

11.0.16. Theorem ([9, 4]). *If a program α decides a good example $R(x)$ from $\mathbf{Lin}_0[\div, \cdot]$, then for all $a \geq 2$*

$$R(a) \Longrightarrow c^p_\alpha(a) > \frac{1}{24}\sqrt{\log\log a}.$$

Proof. By the imbedding test 6.0.3 and the preceding Lemma, we know that the complexity $m = c^p_\alpha(a)$ of any program which decides $R(x)$ satisfies

$$2^{2^{e(m+2)^2}} > a,$$

with $e = 61$. Taking logarithms twice, then the square root and finally using the fact that $3m \geq m + 2$ by (25), we get the required

$$3m \geq m + 2 \geq \sqrt{\frac{\log\log a}{e}} \geq \frac{1}{8}\sqrt{\log\log a},$$

the last step justified because $\sqrt{e} = \sqrt{61} < 8$. $\dashv$

§12. Non-uniform complexity. All the results we have established so far are for recursive programs which compute a certain function or decide a certain relation for *all* inputs, or at least a great number of them. In the proof of the main Theorem 9.0.6, for example, we assumed that α decides coprimeness for some a, b and also for $\lambda a, \lambda b$, where (tracing the proofs) $\lambda = 1 + a!$, which is immensely bigger than a; could we get a "faster" program α for coprimeness if we demanded of it only that it decides coprimeness in some bounded range, say when $a, b < 2^n$ for some (large) n? In this section we will give a brief introduction to this sort of *non-uniform complexity problems*. It will turn out that the lower bounds we have established for $\mathbf{Lin}_0$ persist in this more general context unscathed (except for the constants), but for $\mathbf{Lin}_0[\div]$ and $\mathbf{Lin}_0[\div, \cdot]$, we can only establish weaker results—and even for those, the proofs must be refined and require some new recursion-theoretic notions and techniques.

12.1. Bit complexity and $\mathfrak{k}$-$g(n)$-hardness. A natural number x is *n-bit* if the binary expansion of x requires no more than n 0's or 1's (bits), i.e., if $x < 2^n$. Thus 0 and 1 are 1-bit, and the 2-bit numbers are $0, 1, 2, 3$ with binary expansions $0, 1, 10, 11$. For a k-tuple $\vec{x}$, we write

$$\vec{x} = (x_1, \dots, x_k) < 2^n \iff x_1 < 2^n \ \& \ \cdots \ \& \ x_k < 2^n.$$

Suppose $f : \mathbb{N}^k \to \mathbb{N}$ is a k-ary on function on the natural numbers; Π is a class of algorithms, perhaps those expressed by recursive programs from a set of givens $\mathbf{\Phi}$; and $\alpha \mapsto \mathfrak{k}_\alpha$ is a complexity measure on Π as in (20). The *bit complexity of f relative to* Π and $\mathfrak{k}$ is the function

$$\mathfrak{k}^b(f, \Pi, n) = \inf \Big\{ \sup\{\mathfrak{k}_\alpha(\vec{x}) \mid \vec{x} < 2^n\} \mid \alpha \in \Pi \ \& \ (\forall \vec{x} < 2^n)[\overline{\alpha}(\vec{x}) = f(\vec{x})] \Big\} \tag{76}$$

Thus $c^{p,b}(f, \mathbf{Lin}_0, n)$ and $c^{s,b}(f, \mathbf{Lin}_0, n)$ are the worst-case parallel and sequential calls-complexity measures for computing $f(\vec{x})$ for n-bit numbers by a $\mathbf{Lin}_0$ program *as functions of* n: it is important to register this, since $n = \log(2^n)$, and so values of bit-complexity seem to be one log lower than the corresponding lower bound results expressed as functions of $\vec{x}$.

A function $f(\vec{x})$ is *$\mathfrak{k}$-$g(n)$-hard* for a class of algorithms Π and a complexity measure $\alpha \mapsto \mathfrak{k}_\alpha$, if there is a rational number $r > 0$, such that for all sufficiently large n,

$$\mathfrak{k}^b(f, \Pi, n) \geq r g(n). \tag{77}$$

This implies that if some α in Π computes $f(\vec{x})$ for all $\vec{x}$, then

$$\sup\{\mathfrak{k}_\alpha(\vec{x}) \mid x_1, \dots, x_k < 2^n\} = \Omega(g(n))$$

in the notation of Subsection 1.3, but is is much stronger than that, as it gives a single lower bound (with the same constant) for the complexity of any algorithm from Π which computes $f(\vec{x})$ for all n-bit inputs.

12.2. The lookup algorithm. It is important to notice at this point that it is not possible to derive better than linear lower bounds for the bit complexity of relations from givens that include the primitives of binary arithmetic (9), because of the next simple result which uses *binary search*:

12.2.1. Proposition (The lookup algorithm). *For each k, every k-ary relation $R(\vec{x})$ on $\mathbb{N}$ can be decided for inputs $x_1, \dots, x_k < 2^n$ by a* $(\text{parity}, \text{iq}_2)$*-term $E_n(\vec{x})$ of* $(\text{parity}, \text{iq}_2)$*-depth $\leq n$—and hence by a* $(\text{parity}, \text{iq}_2)$*-recursive program α with $c^p_\alpha(\vec{x}) \leq n$ whenever $x_1, \dots, x_k < 2^n$.*

With the notation introduced in Subsection 12.1, for any k-ary relation R on $\mathbb{N}$,

$$c^{p,b}(R, (\text{parity}, \text{iq}_2), n) \leq n.$$

Proof. We prove that *for every $i < 2^n$, the relation*

$$\text{Eq}_i(x) \iff x = i$$

is defined by a $(\text{parity}, \text{iq}_2)$*-term* $\text{Eq}_i(x)$ *of* $(\text{parity}, \text{iq}_2)$*-depth $\leq n$.* To simplify the nested conditional constructions in this argument, we notice that for any relation $R(\vec{x})$ and partial functions $g(\vec{x}), h(\vec{x})$,

$$\text{if } R(\vec{x}) \text{ then } g(\vec{x}) \text{ else } h(\vec{x}) = \text{if } (\chi_R(\vec{x}) = 0) \text{ then } h(\vec{x}) \text{ else } g(\vec{x});$$

this allows us to branch on $R(\vec{x})$ when we have a term which defines the (total) characteristic function of R.

Proof of the italicized claim is by induction on n, as follows, where we simplify notation by setting for any term M,

$$\text{depth}(M) = \text{depth}_{\text{parity},\text{iq}_2}(M).$$

Basis, $n = 0$, so the only $i < 2^0 = 1$ is 0. We set

$$\text{Eq}_0(x) = \text{if } (x = 0) \text{ then } 1 \text{ else } 0.$$

Clearly, $\text{depth}(\text{Eq}_0(x)) = 0$.

Induction Step. It is enough to infer the result for $i = 2j$ and $i = 2j + 1$, using the induction hypothesis which gives a term $\text{Eq}_j(x)$, and so we set:

$$\begin{aligned} \text{Eq}_{2j}(x) &= \text{if } (\text{parity}(x) = 0) \text{ then } \text{Eq}_j(\text{iq}_2(x)) \text{ else } 0, \\ \text{Eq}_{2j+1} &= \text{if } (\text{parity}(x) = 1) \text{ then } \text{Eq}_j(\text{iq}_2(x)) \text{ else } 0. \end{aligned}$$

If $n = 1$, then only $i = 1 = 2j + 1$ with $j = 0$ comes up, and the term $\text{Eq}_1(x)$ has depth 1. For larger n, the induction hypothesis gives the result, because

$$\text{depth}(\text{Eq}_j(\text{iq}_2(x)) = \text{depth}(\text{Eq}_j(x)) + 1$$

(by Problem x4.7).

From this the Proposition follows for unary R by setting

$$\begin{aligned} E_n(x) = \ &\text{if } E_0(x) \text{ else } \chi_R(0) \\ &\text{else if } E_1(x) \text{ else } \chi_R(1) \\ &\dots \\ &\text{else if } E_{2^n-1}(x) \text{ then } \chi_R(2^n - 1) \\ &\text{else } 0. \end{aligned}$$

The general case is proved by induction on k, setting

$$E_n(y,\vec{x}) = \begin{cases} E^k_0(\vec{x}), & \text{if } y=0,\\ E^k_1(\vec{x}), & \text{if } y=1,\\ \vdots, & \\ E^k_{2^n-1}(\vec{x}), & \text{otherwise,}\end{cases}$$

where $E^k_i(\vec{x})$ is given by the induction hypothesis for the "section relation" $R_i(\vec{x}) \iff R(i,\vec{x})$.

The second claim follows by (23). ⊣

12.3. Non-uniform lower bounds from $\mathbf{Lin}_0$. We will show here that the uniform lower bounds from $\mathbf{Lin}_0$ obtained in Section 6 hold also for non-uniform programs—except for some changes in the constants. This means that for these problems, *the lookup algorithm is optimal for bit complexity from* $\mathbf{Lin}_0$, and in some cases (e.g., for coprimeness) that a (much more interesting) uniform algorithm is also optimal for bit complexity from $\mathbf{Lin}_0$.

12.3.1. THEOREM ([4]). *If $n \geq 8$ and the recursive program α decides primality from $\mathbf{Lin}_0$ correctly for all n-bit numbers, then there is a prime number $p < 2^n$ such that*

$$c^p_\alpha(p) \geq \frac{n}{8}.$$

In the notation for bit complexity introduced by (76)

$$c^{p,b}(\text{Primality}, \mathbf{Lin}_0, n) \geq \frac{n}{8} \qquad (n \geq 8),$$

and so primaliry is c^p-n-hard for programs from $\mathbf{Lin}_0$.

PROOF. We fix $n \geq 8$ and α from $\mathbf{Lin}_0$ which decides primality for all $a < 2^n$. By Lemma 6.1.2, if $2^{2m+2} < p < 2^n$ and $\lambda = 1{+}2^{m+1}$, then there is an imbedding

$$\pi : (\mathbb{N}, \mathbf{Lin}_0)\restriction G_m(p) \to (\mathbb{N}, \mathbf{Lin}_0)$$

such that $\pi(p) = \lambda p$, and so, if $m = c^p_\alpha(p)$, then by the Imbedding Lemma 6.0.1,

$$\text{either } 2^{2m+2} > p, \text{ or } p(1+2^{m+1}) \geq 2^n.$$

To exploit this alternative, we need to apply it to primes not much smaller than 2^n, but small enough so that $p(1+2^{m+1}) < 2^n$, and to find them we will appeal to Bertrand's Postulate, Theorem 418 of [6],[8] which guarantees primes between l and $2l$ when $l \geq 3$.

Let then $k = \lfloor \frac{n}{2} \rfloor$ so that $k \geq 4$,

$$2k \leq n \leq 2k+1 \text{ and } k+1 < 2k \leq n.$$

Choose a prime p such that

$$2^k < p < 2^{k+1}$$

and let $m = c^p_\alpha(p)$. By Problem x4.1, $m \geq 1$. We consider the two cases in the displayed alternative above.

[8]We could avoid appealing to this classical result, but at some cost in the length and complexity of the proof, as we would still need some estimate of the density of the prime numbers.

Case 1: $2^{2m+2} \geq p > 2^k$; then $2m+2 > k$, hence $4m+4 > 2k$, hence $4m+4 \geq 2k+1 \geq n$, which yields the desired $8m \geq n$.

Case 2: $p(1+2^{m+1}) \geq 2^n$; then $2^{k+1}2^{m+2} > 2^n$, hence $k+m+3 > n$, hence $2k+2m+6 > 2n$, and finally $2m+6 > n$ (since $n \geq 2k$), which gives again the required $8m > n$, ⊣

Similar mild elucidations of the proofs we have given extend all the lower bound results about $\mathbf{Lin}_0$ in Section 6 to bit complexity, and they are simple enough to leave for the problems.

This method, however, does not work for the results about $\mathbf{Lin}_0[\div]$: in Lemma 7.0.7, which is the analog for $\mathbf{Lin}_0[\div]$ of Lemma 6.1.2 for $\mathbf{Lin}_0$, we needed to assume that

$$2^{6^{m+3}} \leq p \text{ and set } \lambda = 1 + p!;$$

so if we try to follow the same idea, we would obtain a contradiction from the assumption that $c^p_\alpha(p) = m$ for some m such that

$$2^{6^{m+3}+1}(1 + 2^{6^{m+1}}!) \leq 2^n,$$

and the negation of this inequality leads to a lower bound for m which is of the order of $\log\log n$—one log lower than what we would like. Thus we need to apply the Imbedding Test 6.0.3 with a smaller λ than $1+p!$, and to do this we will derive a version of the Absoluteness Lemma 6.0.2 for subalgebras which are (in general) much smaller than $\mathbf{A}\restriction G_m(X)$.

12.4. The computation space. Fix a recursive program α in the algebra $\mathbf{A}$. For each closed term (with parameters) M in $\mathrm{sig}(\alpha)$ such that $\overline{M}\downarrow$, we define a set $S[M]$ by the following recursion on $D(M)$, taking cases on the form of M.

(S1) $S[x] = \{x\}$, $S[0] = \{0\}$, $S[1] = \{1\}$.
(S2) $S[\phi(M_1, \dots, M_n)] = S[M_1] \cup \cdots \cup S[M_n] \cup \{\phi(\overline{M}_1, \dots, \overline{M}_n)\}$.
(S3) $S[f(M_1, \dots, M_n)] = S[M_1] \cup \cdots \cup S[M_n] \cup S[E_f(\overline{M}_1, \dots, \overline{M}_n)]$.
(S4) Suppose $M \equiv$ if $(M_1 = 0)$ then M_2 else M_3. If $\overline{M}_1 = 0$ and $\overline{M}_2\downarrow$, then $S[M] = S[M_1] \cup S[M_2]$; and if $\overline{M}_1\downarrow$, $\overline{M}_1 \neq 0$ and $\overline{M}_3\downarrow$, then $S[M] = S[M_1] \cup S[M_3]$.

If α computes an n-ary partial function $\overline{a} : A^n \rightharpoonup A$ with head function symbol f_α, then

$$S_\alpha(\vec{x}) = S[f_\alpha(\vec{x})] \quad (\overline{\alpha}(\vec{x})\downarrow) \tag{78}$$

is the *computation space* of α for the input $\vec{x}$.

The next lemma makes it clear that the computation space $S[M]$ of a convergent term M is exactly the part of A that is required for "the computation" of $\overline{M}$.

12.4.1. Lemma (The Second Absoluteness Lemma). *Suppose α is a recursive program on $\mathbf{A}$ and M is a closed* $\mathrm{sig}(\alpha)[A]$*-term whose parameters are in $X \subseteq A$.*

(79) *If* $\mathbf{A}, \alpha \models^m M = w$,

then $w \in S[M] \subseteq G_m(X)$ *and* $\mathbf{A}\restriction S[M], \alpha \models^m M = w$.

Proof is 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 $i = 1, \dots, n$,

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

Hence $\overline{M} = \phi(\overline{M}_1, \dots, \overline{M}_n) \in S[M]$, and from the definition, $S[M] \subseteq G_m(X)$. This implies immediately that

$$\mathbf{A}\restriction S[M], \alpha \models^m M = \overline{M}.$$

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 $k + l \le m$, and for $i = 1, \dots, n$,

$$\overline{M}_i \in S[M_i] \subseteq G_k(X), \; \mathbf{A}\restriction S[M_i], \alpha \models^k M_i = \overline{M}_i,$$

and also

$$\overline{M} \in S[E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)] \subseteq G_l(\overline{M}_1, \dots, \overline{M}_n),$$
$$\mathbf{A}\restriction S[E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)], \alpha \models^l E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n) = \overline{M}.$$

From these and the definitions we get immediately that

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

Finally, that

$$\mathbf{A}\restriction S[M], \alpha \models^{(k+1)} M = w$$

follows immediately from the interpretation of the relation $\alpha \models^m M = w$ in the algebra $\mathbf{A}\restriction S[M]$.

The case for the conditional is simpler and we skip it. $\dashv$

The size of the computation space is (essentially) bounded by the sequential calls complexity:

12.4.2. Proposition. *If α is a recursive program on $\mathbf{A}$, M is a closed, convergent* $\mathrm{sig}(\alpha)[A]$*-term whose parameters are in the finite set $X \subseteq A$, then*

$$|S[M]| \le C^s(M) + |X|. \tag{80}$$

In particular, if α has arity n and total arity $\ell \ge 2$, then

$$|S_\alpha(\vec{x})| \le c^s_\alpha(\vec{x}) + n \le l^s_\alpha(\vec{x}) + n \le (\ell+1)^{l^p_\alpha(\vec{x})} + n \quad (\overline{\alpha}(\vec{x})\downarrow). \tag{81}$$

Proof. The basic inequality (80) follows immediately from

$$|S[M] \setminus X| \le C^s(M) \quad (\overline{M}\downarrow), \tag{82}$$

which we establish by induction of $D(M)$.

If $M \equiv x$ for some $x \in A$ (including 0 and 1), then $S[M] \setminus X = \emptyset$ and $C^s(M) = 0$, so that (82) holds.

If $M \equiv \phi(M_1, \dots, M_n)$, then

$$\begin{aligned} S[M] \setminus X &= \Big(S[M_1] \cup \dots \cup S[M_n] \cup \{\phi(\overline{M}_1, \dots, \overline{M}_n)\}\Big) \setminus X \\ &\subseteq (S[M_1] \setminus X) \cup \dots \cup (S[M_n] \setminus X) \cup \{\phi(\overline{M}_1, \dots, \overline{M}_n)\}, \end{aligned}$$

and so, using the induction hypothesis,

$$|S[M] \setminus X| \leq |C^s(M_1)| + \cdots + |C^s(M_1)| + 1 = C^s(M).$$

Similarly, if $M \equiv \mathsf{f}(M_1, \dots, M_n)$ with a recursion variable f, then

$$\begin{aligned} S[M] \setminus X &= \Big(S[M_1] \cup \cdots \cup S[M_n] \cup S[E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)]\Big) \setminus X \\ &\subseteq (S[M_1] \setminus X) \cup \cdots \cup (S[M_n] \setminus X) \cup (S[E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)] \setminus \{\overline{M}_1, \dots, \overline{M}_n\}). \end{aligned}$$

To see this, suppose that $w \notin X$ but

$$w \in \Big(S[M_1] \cup \cdots \cup S[M_n] \cup S[E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)]\Big)$$

and consider cases on whether $w \in S[M_i]$ for some i or not: if $w \in S[M_i]$ for some i, then $w \in S[M_i] \setminus X$, and so w belongs to the set on the right-hand-side of the claimed inclusion; and if $w \notin S[M_i]$ for every i, then $w \in S[E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)]$ and $w \neq \overline{M}_i$ for any i (since $\overline{M}_i \in S[M_i]$), and hence w gets in the last difference of the union on the right. From this inclusion and the induction hypothesis we get the desired

$$|S[M] \setminus X| \leq |C^s(M_1)| + \cdots + |C^s(M_1)| \\ + |C^s(E_{\mathsf{f}}(\overline{M}_1, \dots, \overline{M}_n)| = C^s(M).$$

The last case of the conditional is shown similarly, and (81) then follows by Propositions 4.5.1 and 4.5.2. ⊣

Next comes a computation space version of Lemma 7.0.7:

12.4.3. Lemma (**Lin**$_0[\div]$-Imbedding for the computation space). *Suppose α is a unary program from* **Lin**$_0[\div]$*, and for some $a \in \mathbb{N}$ such that $\overline{\alpha}(a)\downarrow$, let*

$$m = c^p_\alpha(a), \quad \nu = |S_\alpha(a)|$$

be the parallel calls-complexity and the size of the computation space of α at a. If

$$2^{6^{m+3}} < a,$$

then there is a number $\lambda < 2^{\nu 6^{m+2}}$ and an imbedding

$$\pi : (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) \restriction S_\alpha(a) \rightarrowtail (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div])$$

such that $\pi(a) = \lambda a$.

Proof. As in the proof of Lemma 7.0.7, the assumed inequality on m implies that each $x \in G_m(a)$ can be expressed uniquely as a proper fraction of the form

$$x = \frac{x_0 + x_1 a}{x_2} \quad (|x_i| \leq 2^{6^{m+1}}). \tag{83}$$

Now $S_\alpha(a) \subseteq G_m(a)$ by Lemma 12.4.1, and so with each $x \in S_\alpha(a)$ there is associated a unique *denominator*

$$\text{denom}(x) = \text{the unique } x_2 \leq 2^{6^{m+1}} \text{ such that (83) holds.}$$

We claim that the conclusion of the Lemma holds with

$$\lambda = 1 + \textstyle\prod_{x \in S_\alpha(a)} \text{denom}(x) \leq 1 + \Big(2^{6^{m+1}}\Big)^\nu < 2^{\nu 6^{m+2}}, \tag{84}$$

and to prove this we follow very closely the proof of Lemma 7.0.7: we set

$$\rho(x) = \rho\Big(\frac{x_0 + x_1 a}{x_2}\Big) = \frac{x_0 + x_1\lambda a}{x_2};$$

check that this is a well defined injection on $S_\alpha(a)$ which takes values in $\mathbb{N}$, because

$$x \in S_\alpha(a) \Longrightarrow \mathrm{denom}(x) \mid (\lambda - 1);$$

and finally verify that it is an imbedding from $(\mathbb{N}, \mathbf{Lin}_0[\div]) \restriction S_\alpha(a)$ to $(\mathbb{N}, \mathbf{Lin}_0[\div])$ exactly as in the proof of Lemma 7.0.7. ⊣

12.4.4. Theorem. *If $n \geq 4$ and α is a recursive program from $\mathbf{Lin}_0[\div]$ which decides the primality relation for all $x < 2^n$, then there exists a prime $p < 2^n$ such that*

$$2\nu 6^{m+3} \geq n \qquad (m = c^p_\alpha(p), \nu = |S_\alpha(p)|). \tag{85}$$

Proof. As in the proof of Lemma 12.3.1, we let $k = \lfloor \frac{n}{2} \rfloor$ so

$$2k \leq n \leq 2k+1 \text{ and } k+1 < 2k \leq n.$$

The hypothesis on n yields $2^k \geq 4$, so Bertrand's Postulate insures that there exists some prime p such that $2^k < p < 2^{k+1}$. This is the prime we want: we set

$$m = c^p_\alpha(p), \quad \nu = |S_\alpha(p)|.$$

By Lemmas 12.4.3 and 6.0.1,

$$\text{either } 2^{6^{m+3}} \geq p, \text{ or } \lambda p \geq 2^n \text{ with some } \lambda < 2^{\nu 6^{m+2}},$$

and it is enough to derive (85) in each of these cases.

Case 1: $2^{6^{m+3}} \geq p > 2^k$; thus $6^{m+3} > k$, hence $2 \cdot 6^{m+3} > 2k$, which implies that $2 \cdot 6^{m+3} \geq 2k + 1 \geq n$, from which (85) follows because $\nu \geq 1$.

Case 2: $2^{\nu 6^{m+2}} p \geq 2^n$. This gives that $2^{\nu 6^{m+2}} 2^{k+1} > 2^n$, so that $\nu 6^{m+2} + k + 1 > n$, hence $2\nu 6^{m+2} + 2k + 2 > 2n$, and since $n \geq 2k$, we have $2\nu 6^{m+2} + 2 > n$; but (easily) $2\nu 6^{m+3} > 2\nu 6^{m+2} + 2$, which gives the required (85). ⊣

The awkward form of this lower bound—mixing two complexity functions—is due to our ignorance of the true relation between $c^p_\alpha(a)$, $c^s_\alpha(a)$ and $|S_\alpha(a)|$. We collect in a corollary its simplest-to-state consequences:

12.4.5. Corollary. *Suppose $n \geq 4$ and α is a recursive program from $\mathbf{Lin}_0[\div]$ which decides the primality relation for all $a < 2^n$. Then there exists a prime $p < 2^n$ such that*

$$c^s(p) > \frac{\log n}{5 \log 6} > \frac{\log n}{15}, \tag{86}$$

and if α has total arity $\ell \geq 2$, then

$$l^p_\alpha(p) > \frac{\log n}{7 \log(\ell + 4)}. \tag{87}$$

Thus primality is c^s-$\log n$-hard for programs from $\mathbf{Lin}_0[\div]$, and l^p-$\log n$-hard for programs from $\mathbf{Lin}_0[\div]$ of total arity $\ell(\alpha) \leq \ell$ for any fixed ℓ.

Proof. Let p be the prime guaranteed by the Theorem, and to prove (86) first, let $m = c^s_\alpha(p)$, so that by Proposition 12.4.2, we have $2(m+1)6^{m+3} \geq n$; now $2(m+1) < 6^m$, trivially, since $m \geq 1$, which yields $6^{2m+3} > n$, hence $6^{5m} > n$, from which (86) follows by taking logarithms.

For the second inequality, let $k = l^p_\alpha(p)$, so that $\nu \leq (\ell+1)^k + 1$ by (81). Thus the Theorem gives

$$2((\ell+1)^k + 1)6^{k+3} \geq n,$$

from which we derive (87) by the following, trivial computation (in which we use the known $\ell \geq 2$, $k \geq 1$):

$$\begin{aligned} 2((\ell+1)^k + 1)6^{k+3} &\leq 2(\ell+1)^{k+2}6^{k+3} \\ &< (\ell+1)^{k+1}6^{k+4} \leq (\ell+4)^{2k+5} \leq (\ell+4)^{7k}. \end{aligned} \qquad \dashv$$

Problems for Section 12

x12.1. Give an example of a term M with n parameters in it, such that $\overline{M}\downarrow$, $C^p(M) = 0$, but $|S[M]| = n$.

Recall the definition of good examples in Subsection 6.2.

x12.2 ([4]). Suppose $P(x)$ is a good example such that for some k and all sufficiantly large m, there exists some x such that

$$P(x) \ \&\ 2^m \leq x < 2^{km}.$$

Prove that $P(x)$ is c^p-n-hard for recursive programs from $\mathbf{Lin}_0$, and verify that all the good examples in Problem x6.4 satisfy the hypothesis.

x12.3 ([4]). Prove that coprimeness and the function $\gcd(x, y)$ are c^p-n-hard for recursive programs from $\mathbf{Lin}_0$.

x12.4. Suppose $P(x)$ is a good example with the property that for some k and each m, there is some x such that $P(x)$ and

$$2^{6^m} < x < 2^{6^{km}}.$$

Prove that $P(x)$ is c^s-$\log n$-hard for programs from $\mathbf{Lin}_0[\div]$ and l^p-log-hard for programs from $\mathbf{Lin}_0[\div]$ of total arity $\ell(\alpha) \leq \ell$.

Verify also that the good examples in Problem x6.4 satisfy the hypothesis.

x12.5 ([4]). Prove that coprimeness is c^s-$\log n$-hard for recursive programs from $\mathbf{Lin}_0[\div]$, and for each fixed ℓ, it is l^p-log-hard for programs from $\mathbf{Lin}_0[\div]$ of total arity $\ell(\alpha) \leq \ell$.

x12.6. Formulate and prove a hardness result for good examples for recursive programs from $\mathbf{Lin}_0[\div, \cdot]$.

§13. Logical extensions and random access machines. In this section we show that the lower bounds we have obtained also hold for *random access machines.* The method of proof involves a technique which extends the lower bounds for recursive programs that we have established to quite general computational models, and we will start by developing the method.

13.1. Logical extensions of algebras. Suppose $B \supseteq A$ is a set which contains the universe A of an algebra $\mathbf{A}$. A partial function

$$\psi : B^n \rightharpoonup B$$

is *logical* (on B, over A), if for every permutation

$$\rho : A \rightarrowtail\!\!\!\!\!\twoheadrightarrow A$$

such that $\rho(0) = 0$, $\rho(1) = 1$, there is an extension

$$\rho^B : B \rightarrowtail\!\!\!\!\!\twoheadrightarrow B$$

of ρ to B such that

$$\rho^B \psi(x_1, \dots, x_n) = \psi(\rho^B x_1, \dots, \rho^B x_n), \quad (x_1, \dots, x_n \in B). \tag{88}$$

More generally, an algebra

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

is a *logical extension*[9] of

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

if $A \subseteq B$, and every permutation $\rho : A^n \rightarrowtail\!\!\!\!\!\twoheadrightarrow A$ such that $\rho(0) = 0$, $\rho(1) = 1$ has an extension $\rho^B : B \rightarrowtail\!\!\!\!\!\twoheadrightarrow B$ such that (88) holds for every *fresh given* $\psi^{\mathbf{B}}$ of $\mathbf{B}$. 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 a logical extension $\mathbf{B}$ as enriching $\mathbf{A}$ with various data structures (arrays, trees, etc.) which are used to express some algorithm from $\mathbf{\Phi}$, but without introducing new "non-logical" givens or changing in any essential way how the partial functions in $\mathbf{\Phi}$ are used by the algorithm. Notice, for example, that a simple constant function

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

is not logical on any B over A; because that would demand of every bijection $\rho : A \rightarrowtail\!\!\!\!\!\twoheadrightarrow A$ that

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

which evidently fails if ρ moves w_0. In a similar way, if we just add a new name for an old given,

$$\psi(x) = \phi(x),$$

the resulting algebra is not automatically a logical extension, because ϕ need not satisfy (88) for every ρ. For a classification of all the logical functions on A over A, cf. Problem x13.1.

13.1.1. Lemma. *Suppose* $\mathbf{B}$ *is a logical extension of an infinite algebra* $\mathbf{A}$*, and* $X \subseteq A$.

(1) If $\rho : A \rightarrowtail\!\!\!\!\!\twoheadrightarrow A$ *fixes every* $y \in G_m(\mathbf{A}, X)$ *and* $\rho^B : B \rightarrowtail\!\!\!\!\!\twoheadrightarrow B$ *extends* ρ *so that* (88) *holds, then* ρ^B *fixes every* $y \in G_m(\mathbf{B}, X)$.

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

[9]This notion is due to Itay Neeman, and it simplified considerably the original proof of Theorem 13.3.3.

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

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

At the induction step, to prove (1) first, suppose that ρ fixes every $y \in G_{m+1}(\mathbf{A}, X)$ and consider a typical member y of $G_{m+1}(\mathbf{B}, X)$. If $y = \phi(\vec{x})$ with $x_1, \dots, x_n \in G_m(\mathbf{B}, X)$ and an old given ϕ, then $x_1, \dots, x_n \in G_m(\mathbf{A}, X)$ by (2) of the induction hypothesis, and so $y \in G_{m+1}(\mathbf{A}, X)$ and $\rho(y) = y$ by the hypothesis on ρ. If

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

and a fresh given ψ, then ρ^B fixes $x_1, \dots, x_n$ by (1) of the induction hypothesis, and so by (88),

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

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

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

and define $\rho : A \rightarrowtail\!\!\!\!\twoheadrightarrow A$ so that it fixes every member of $G_{m+1}(\mathbf{A}, X)$ but $\rho y \neq y$. This is possible because $G_{m+1}(\mathbf{A}, X)$ is finite, while A is infinite, and it immediately contradicts (1) for $m + 1$, since $\rho^B(y) = \rho y$. ⊣

13.1.2. LEMMA. *If β decides coprimeness in a logical extension $\mathbf{B}$ of*

$$(\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]) = (\mathbb{N}, 0, 1, \mathbf{Lin}, \mathrm{iq}, \mathrm{rem})$$

and (a, b) is a difficult pair, then

$$c^p_\beta(a, b) \geq \frac{1}{10} \log\log a.$$

PROOF. Let $m = c_\beta(a, b)$. By the First Absoluteness Lemma 6.0.2,

$$\mathbf{B} \restriction G_m(\mathbf{B}, a, b) \models^m \overline{\beta}(a, b) = 1.$$

Assume towards a contradiction that $2^{2^{4m+6}} \leq a$. Lemma 9.0.5 supplies an imbedding

$$\pi : \mathbf{A} \restriction G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a, b) \rightarrowtail (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div])$$

such that for a suitable $\lambda > 1$,

$$\pi a = \lambda a, \quad \pi b = \lambda b.$$

The domain $G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a, b)$ of π is equinumerous with its image $\pi[G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a, b)]$, and hence so are their complements: so we can extend π to a permutation of A using any bijection of these complements. Let π^B be an extension of this π to B such that (88) holds. But now π^B is an imbedding

$$\pi^B : \mathbf{B} \restriction G_m(\mathbf{B}, a, b) \rightarrowtail \mathbf{B},$$

because the relevant two conditions

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

follow from the hypothesis on π and Lemma 13.1.1 (which guarantees that the only elements to which an old given ϕ is applied are, indeed, in $G_m((\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div]), a, b))$ and (88). The Imbedding Test Lemma 6.0.3 now gives us a contradiction, since λa and λb are not coprime; thus $2^{2^{4m+6}} > a$, and this implies the claimed lower bound on m as in the proof of Theorem 9.0.6. ⊣

The same argument can be used to show that all the lower bound results we have established for recursive programs in the algebras $\mathbf{N}_0, (\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div])$ and $(\mathbb{N}, 0, 1, \mathbf{Lin}_0[\div, \cdot])$ apply to recursive programs on logical extensions of these structures. The plan is to establish the same lower bounds for RAMs, by simulating them by recursive programs in such logical extensions.

13.2. Random access machines. By [5], a random access machine (RAM) M which computes an k-ary partial function on $\mathbb{N}$ from given (partial) functions in $\mathbf{\Phi}$ 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* $\mathrm{acc}_1, \dots, \mathrm{acc}_K$ ($K \geq k+1$); each accumulator stores a natural number. The computation starts with the input $\vec{x}$ stored in the first k of these, and 0 in the other accumulators; at the end of a convergent computation the output is the content of acc_{k+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 following 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* acc_i *else goto* acc_j
0. *End*

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

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

A *state* of M is a tuple

$$s = (a, \mathrm{acc}_1, \dots, \mathrm{acc}_K, R),$$

where $a \leq N$ is a label (for an instruction), $\mathrm{acc}_1, \dots, \mathrm{acc}_K \in \mathbb{N}$, and $R : \mathbb{N} \to \mathbb{N}$. The *initial state* (for a tuple $\vec{x} \in \mathbb{N}^k$) is

$$s_0(\vec{x}) = (1, x_1, \dots, x_k, \vec{0}, R_0) \quad (R_0[i] = 0 \text{ for all } i),$$

and the states $s_1(\vec{x}), s_2(\vec{x}), \dots$ in the *computation* of M from $\vec{x}$ are defined by successively applying to each

$$s_i(\vec{x}) = (a_i(\vec{x}), \text{—})$$

the command labelled $a_i(\vec{x})$ in the obvious way. The computation *terminates* in stage i if $a_i(\vec{x}) = 0$, and in that case its *length* (time complexity) is declared to be i,

$$c_M^{\text{RAM}}(\vec{x}) = \text{the least } i \text{ such that } a_i(\vec{x}) = 0. \tag{89}$$

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

13.3. Adding functions to an algebra. 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 $a, b, c, t \ldots$ over A and $X, Y, Z, \ldots$ over $(A \to A)$, define the following operations:

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

Consider an algebra $\mathbf{A} = (A, 0, 1, \{\phi^{\mathbf{A}}\}_{\phi \in \mathbf{\Phi}})$ and its extension

$$\mathbf{B} = (B, 0, 1, \{\phi^{\mathbf{A}}\}_{\phi \in \mathbf{\Phi}}, \varepsilon, \text{ap}, \text{update}, \text{indupdate}),$$

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

13.3.1. LEMMA. *The algebra* $\mathbf{B}$ *is a logical extension of* $\mathbf{A}$.

PROOF. Given $\rho : A \rightarrowtail\!\!\!\!\twoheadrightarrow A$ which fixes 0 and 1, we extend it to $(A \to A)$ (using the same name) by the obvious

$$\rho X = (\lambda t)\rho(X(\rho^{-1}(t))) \text{ i.e., } (\rho X)(t) = \rho(X(\rho^{-1}(t))).$$

This is clearly a bijection of B onto B, and the crucial (88) is easy to verify for the new givens. For example,

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

and for application,

$$\text{ap}(\rho X, \rho t) = (\rho X)(\rho t) = \rho(X(\rho^{-1}(\rho t))) = \rho(X(t)) = \rho\text{ap}(X, t).$$

We will omit the details. ⊣

13.3.2. LEMMA. *Let M be any Random Access Machine which computes a partial function $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ from the givens in $\mathbf{\Phi}$ with time complexity $c_M^{\text{RAM}}(\vec{x})$, where $\mathbf{A} = (\mathbb{N}, 0, 1, \mathbf{\Phi})$ is an algebra. Then there is a recursive program α in a logical extension $\mathbf{B}$ of $\mathbf{A}$ which also computes f, such that*

$$c_\alpha^p(\vec{x}) \leq c_M^{\text{RAM}}(\vec{x}) \qquad (f(\vec{x})\downarrow).$$

PROOF. Let us first assume for simplicity that the functions

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

are among the givens in $\mathbf{\Phi}$—it will be easy to remove this assumption at the end. The required program α has the two equations

$$f(\vec{x}) = g(1, \vec{x}, \vec{0}, \varepsilon)$$
$$g(a, \vec{u}, X) = \begin{cases} E_0, & \text{if } a = 0, \\ E_1, & \text{if } a = 1, \\ & \vdots \\ E_N, & \text{if } a = N, \\ g(a, \vec{u}, X) & \text{otherwise,} \end{cases}$$

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

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

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

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

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

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

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

a. *if* $\mathrm{acc}_j = 0$ *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* $\mathrm{acc}_j > N$ *then goto* acc_i else *goto* acc_j:
$$E_a \equiv \text{if } (u_j > N) \text{ then } g(u_i, \vec{u}, X) \text{ else } g(u_j, \vec{u}, X)$$

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

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

To avoid the assumption that the constant functions ψ_a and the tests χ_a are included in $\mathbf{\Phi}$, 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. ⊣

With Lemma 13.1.2 (and its versions for $\mathbf{Lin}_0$ and $\mathbf{Lin}_0[\div, \cdot]$), this gives lower bounds for RAM computations of all the functions we have considered. We state here the two most interesting of these results.

13.3.3. Theorem ([3]). *(1) If a Random Access Machine M decides the coprimeness relation $x \perp y$ from $\mathbf{Lin}_0$ with time complexity $c^{\mathrm{RAM}}_M(x, y)$, then for for all $a > 2$,*

$$c^{\mathrm{RAM}}_M(a, a^2 - 1) > \frac{1}{10} \log(a^2 - 1).$$

(2) If a Random Access Machine M decides the coprimeness relation $x \perp y$ from $\mathbf{Lin}_0[\div]$ with time complexity $c^{\mathrm{RAM}}_M(x,y)$, then for for all difficult pairs (a,b),

$$c^{\mathrm{RAM}}_M(a,b) > \frac{1}{10}\log\log a.$$

Problems for Section 13

x13.1. Prove that if $\psi : A \to A$ is a unary logical function on A over A (A infinite), and for some $x \notin \{0,1\}$, $\psi(x) \notin \{0,1\}$, then for all $x \notin \{0,1\}$, $\psi(x) = x$. Use this to prove that there are exactly twelve unary logical functions $\psi : A \to A$, and describe them.

Formulate and prove a similar result about binary functions $\psi : A^2 \to A$ which are logical over A, and show that for each n, there are only finitely many logical functions $f^n \to A$.

x13.2. Prove that primality and the relations of being square-freedom or a perfect square defined in Problem x6.4 are $c^{\mathrm{RAM}}(\log n)$-hard for RAMs from $\mathbf{Lin}_0[\div]$ and $c^{\mathrm{RAM}}(\sqrt{\log n})$-hard for RAMS from $\mathbf{Lin}_0[\div, \cdot]$. (The last result is due to [8].)

x13.3. Prove that coprimeness is $c^{\mathrm{RAM}}(n)$-hard for RAMS from $\mathbf{Lin}_0$ and $c^{\mathrm{RAM}}(\log n)$-hard for RAMS from $\mathbf{Lin}_0[\div]$.

REFERENCES

[1] M. Agrawal, N. Kayal, and N. Saxena, *Primes is in P*, ***Annals of Mathematics, Second Series***, vol. 160 (2004), pp. 781–793.

[2] 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.

[3] Lou van den Dries and Yiannis N. Moschovakis, *Is the Euclidean algorithm optimal among its peers?*, ***The Bulletin of Symbolic Logic***, (2004), pp. 390–418.

[4] ———, *Arithmetic complexity*, submitted.

[5] 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.

[6] G. H. Hardy and E. M. Wright, ***An introduction to the theory of numbers***, Clarendon Press, Oxford, fifth edition (2000), originally published in 1938.

[7] D. E. Knuth, ***The Art of Computer Programming. Fundamental Algorithms***, second ed., Addison-Wesley, 1973.

[8] 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.

[9] ———, *Lower bounds for computations with the floor operation*, ***SIAM Journal on Computing***, vol. 20 (1991), pp. 315–327.

[10] 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.

E-mail: ynm@math.ucla.edu