RECURSION AND COMPLEXITY

YIANNIS N. MOSCHOVAKIS

ynm@math.ucla.edu

Version 2.0, April 2015

CONTENTS

PREFACE

This is the present state of a developing set of lecture notes that I have used in several courses, seminars and workshops, mostly at UCLA and in the *Graduate Program in Logic and Algorithms* (MPLA) at the University of Athens. The general topic is the theory of *abstract* (first-order) *recursion* and its relevance for the *foundations of the theory of algorithms and computational complexity*, but the work on this broad project is very incomplete and so the choice of topics which are covered is somewhat eclectic.

Part I, almost half of the material, is an introduction to the theory of *recursive* (McCarthy) *programs* on abstract structures. Chapters 1 and 2 provide an elementary exposition of the basic facts about these objects and there is very little that is new in them, but I do not know of another easily accessible, self-contained and reasonably complete source for this material. Chapter 3 introduces the natural *complexity measures* for recursive programs and it has some new (at least to me) results, most notably Theorems 3B.8 and 3B.11; these are due to Anush Tserunyan and have substantial foundational significance.

The remaining and main part of these notes is about the derivation of lower bounds (especially) for problems in arithmetic and algebra, and perhaps the simplest way to introduce my take on this is to give a fairly precise formulation of a fundamental conjecture about a well-known problem.

The Euclidean algorithm on the natural numbers can be specified succinctly by the *recursive program*

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

where $\mathrm{rem}(a,b)$ is the remainder in the division of a by b. It is an algorithm *from* (*relative to*) the remainder function rem and the relation eq_0 of equality with 0, meaning that in its execution, ε has access to "oracles" which provides on demand the value $\mathrm{rem}(x,y)$ for any x and y and the truth value of $\mathrm{eq}_0(x)$. 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 $\mathrm{gcd}(a,b)$ by the Euclidean and logarithms are to the base 2. Much more is known about $c_\varepsilon(a,b)$, but this upper bound suggests one plausible formulation of the Euclidean's (worst-case) *weak optimality* among algorithms from rem and eq_0 which compute the gcd:

Main Conjecture. *For every algorithm α from* rem *and* eq_0 *which computes the function* $\mathrm{gcd}(x,y)$*, there is a (positive, rational) constant r such that for infinitely many pairs (a,b) with $a > b$,*

$$c_\alpha(a,b) \geq r \log a.$$

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

This is a classical fact about the Euclidean, taking for example the pairs (F_{n+3}, F_{n+2}) of successive Fibonacci numbers, cf. Problems x1D.9, x1D.10. The general case is open, probably not easy and certainly not precise as it stands, without specifying precisely what algorithms it is about.

Now, there are Turing machines which compute $\mathrm{gcd}(x,y)$ making no oracle calls at all, simply because $\mathrm{gcd}(x,y)$ is a computable function. So to make the conjecture precise and meaningful, we must employ a notion of (relative) *algorithm from given functions and relations* $\mathbf{\Phi}$ which does not take for granted any operations outside $\mathbf{\Phi}$. (Turing machines have free access to the basic operations on strings of symbols on which they operate built into their definitions.)

In fact, there is no generally accepted, rigorous definition of *what algorithms are.* This is not a problem when we study particular algorithms, which are typically specified precisely in some form or other without any need to investigate whether *all algorithms* can be similarly specified. In *Complexity Theory*—and especially when we want to establish *lower bounds* for some measure of computational complexity—the standard methodology is to ground proofs on rigorously defined *models of computation*, such as Turing machines, register or random access machines, decision trees, straight line computation, etc., and sometimes also on specific *representations of the input*, e.g., *unary* or *binary* notation for natural numbers, *adjacency matrices* for graphs, etc. There is a problem with this practice, when we try to compare lower bound results obtained for different models, typically attacked by establishing *simulations* of one model by another, cf. van Emde Boas [1990]; and this problem becomes acute when we want to establish *absolute* (or at least "very widely applicable") lower bounds which are small, polynomial or even linear (in the length of the input) as in the Main Conjecture, generally less complex than the known simulations.

So there are two, related and perhaps equally important aims of research in this area.

One is to derive *lower bounds for mathematical problems*; the other is to develop a foundational framework in which one may be able to prove (or at least argue convincingly) that these bounds are *absolute*, that they restrict *all algorithms*. The first of these inevitably requires mathematical tools from the area in which the problems arise; and the second inevitably involves logic. *Recursion* enters the picture because it provides the most straightforward method to model algorithms faithfully and to analyze them using methods from logic.

Recursive programs were introduced by McCarthy [1963] and they can be viewed as just another model of computation, but they have some distinct advantages over the better-known, machine-based models:

(1) Recursive programs naturally compute partial functions $f : A^n \rightharpoonup A$ and decide relations $R \subseteq A^n$ on arbitrary sets A, relative to arbitrary *given* functions and relations on A, their *primitives*.

(2) All familiar models of computation can be simulated faithfully and simply by recursive programs, once the primitives they use are identified.

(3) All time-, space- and resource use- complexity functions can be defined directly for recursive programs, so that it is easy to compare them.

In fact, most algorithms which compute partial functions $f : A^n \rightharpoonup A$ or decide relations $R \subseteq A^n$ from specified primitives on a set A can be expressed simply and directly by recursive programs.[1] Moreover, there are both foundational arguments and mathematical results which support the claim that *all elementary algorithms* (as in the footnote) *can be faithfully represented by recursive programs*, so that lower bounds established for them should be absolute, cf. Moschovakis [1984], [1989], [1998], [2001]. (1) – (3) and this view are the chief motivations for including Part I in these notes. Here, however, I will take a different approach to the derivation of lower bounds, which is stronger, more widely applicable and does not tie us to a specific foundational view of what algorithms are.

In Chapter 4, which is the heart of these notes, we formulate three simple axioms about algorithms in the style of *abstract model theory*. These are bundled into the notion of a *uniform process* on an arbitrary (first order) structure: all "concrete" algorithms specified by computation models are uniform processes, as are their usual *non-deterministic* versions. Uniform processes can "compute" functions which are not computable, they are not about that; but they carry a rich complexity theory which, when applied

[1] There are algorithms whose implementations print output (or drop bombs), ask "the user" if she prefers business or coach class and may never terminate—or stop after a transfinite (ordinal) "length of time". In these notes we will confine ourselves to pure, finitary algorithms which compute partial functions or decide relations from given partial functions, for which complexity theory is most fully developed. We will sometimes call them *elementary algorithms*, for short. The extension of most of what we say to algorithms with *side effects* or *interaction*, *infinitary algorithms* and algorithms *from higher-type primitives* requires combining the methods we will use with classical *domain theory* and *recursion in higher types*, and is not as different from what we will be doing as one might think. In any case, we will not go into it here.

to concrete algorithms yields non-trivial lower bound results, in some cases optimal, absolutely or up to a multiplicative constant.

For a sample result, suppose

$$\mathbf{A} = (A, R_1^{\mathbf{A}}, \dots, R_k^{\mathbf{A}}, \phi_1^{\mathbf{A}}, \dots, \phi_l^{\mathbf{A}}) = (A, \mathbf{\Phi})$$

is a first order structure on the vocabulary $\Phi = \{R_1, \dots, R_k, \phi_1, \dots, \phi_l\}$, suppose $P \subseteq A^n$ is an n-ary relation on A and let $\Phi_0 \subseteq \Phi$. From these data, we will define a function

$$\text{calls}_{\Phi_0}(\mathbf{A}, P) : A^n \to \mathbb{N} \cup \{\infty\} = \{0, 1, \dots, \infty\},$$

the *intrinsic* calls_{Φ_0}*-complexity function* of P, such that *if α is any algorithm from $\mathbf{\Phi}$ which decides P, then for all $\vec{x} \in A^n$*,

$$(*) \quad \text{calls}_{\Phi_0}(\mathbf{A}, P, \vec{x}) \leq \text{ the number of calls to primitives in } \mathbf{\Phi}_0 \\ \text{that } \alpha \text{ must execute to decide } P(\vec{x}) \text{ from } \mathbf{\Phi}.$$

This is a theorem if α is expressed by a concrete algorithm from $\mathbf{\Phi}$ so that, in particular, the complexity measure on the right is precisely defined; it will be made plausible for all algorithms, by a brief conceptual analysis of what it means (minimally) to *compute from primitives*; and it is not trivial, e.g., we will show that *if $x \perp\!\!\!\perp y$ is the coprimeness relation on $\mathbb{N}$, then for infinitely many pairs (a, b) with $a > b$*,

$$(**) \qquad \text{calls}_{\{\text{rem}\}}((\mathbb{N}, \text{rem}, \text{eq}_0), \perp\!\!\!\perp, a, b) > \frac{1}{10} \log \log a.$$

This follows from the (much stronger) Theorem 6C.5, an abstract version of one of the main results in van den Dries and Moschovakis [2004]. It gives a (very) partial result towards the Main Conjecture, one log below what we would like to prove.

The main tool for defining the intrinsic complexities and deriving lower bounds for them in Chapter 4 is the *homomorphism method*, an abstract and mildly extended version of the *embedding method* developed in van den Dries and Moschovakis [2004], [2009]. We will use it in Part III to get somewhat strengthened versions of the lower bound results about arithmetic in these two papers and then again in Part IV to get similar results in algebra. Very few of these are new: my aim here is not to establish new results, but to explain the homomorphism method, illustrate its wide applicability in two different areas and (mostly) identify some basic notions of the theory of computational complexity which (perhaps) have not been noticed.

The exposition is elementary, aimed at advanced undergraduate and graduate students with some knowledge of logic. I will be grateful for any comments, corrections and (especially) pointers to relevant results in the literature, as this draft in very thin and surely incomplete in assigning proper credits.

Yiannis N. Moschovakis
April 2015

PART I, RECURSIVE PROGRAMS

CHAPTER 1

INTRODUCTION

We summarize some basic facts from recursion theory, logic, arithmetic and algebra, primarily to set up notation. One should scan this generally well-known material, perhaps solve some of the problems, and then go on, coming back to this chapter as the need arises.

1A. Notation and preliminaries

We will use (mostly) standard notation: $\mathbb{N} = \{0, 1, \dots\}$ is the set of *natural numbers*, $\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$ is the set of *integers*, $\mathbb{Q}$ is the set of fractions and $\mathbb{R}, \mathbb{C}$ are the sets of *real* and *complex numbers* respectively. As usual, we use the same symbols $0, 1, +, -, \cdot, \div$ for the corresponding objects and functions in all these sets—and in all rings and fields, in fact.

We also set

$$S(x) = x + 1, \quad x \dot{-} y = \text{if } (x < y) \text{ then } 0 \text{ else } x - y, \quad \mathrm{Pd}(x) = x \dot{-} 1$$

for the *successor*, *arithmetic subtraction* and *predecessor* functions on $\mathbb{N}$, and

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

This is the "true", binary logarithm function. We will sometimes compose it with one of the functions

$$\begin{aligned} \lfloor x \rfloor &= \text{the largest integer } \leq x && (\text{the } \textit{floor} \text{ of } x), \\ \lceil x \rceil &= \text{the least integer } \geq x && (\text{the } \textit{ceiling} \text{ of } x) \end{aligned}$$

to get an integer value.

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

(1A-1) $$x = yq + r \qquad (0 \leq r < y);$$

if $x < y$, then $q = 0$ and $r = x$, while if $x \geq y$, then $q \geq 1$. We refer to (1A-1) as the *correct division equation* for x, y, and we set

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

with the unique q and the r for which it holds. We will also write

$$\mathrm{iq}_m(x) = \mathrm{iq}(x, m), \quad \mathrm{parity}(x) = \mathrm{rem}(x, 2) \quad (m \geq 2).$$

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 $\neq 1$,

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

The *greatest common divisor* of two natural numbers is what its name means:

(1A-2) $\mathrm{gcd}(x, y) =_{\mathrm{df}}$ the largest d such that $d \mid x$ and $d \mid y \quad (x, y \geq 1)$.

Thus,

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

The most commonly used notations for comparing the growth rate of unary functions on $\mathbb{N}$ are the *Landau symbols*:

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

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

Finally, the *characteristic function* of a relation $R \subseteq A^n$ on a set A is defined by

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

We will identify a relation R with χ_R and write synonymously

(1A-3) $$R(\vec{x}) \iff \chi_R(\vec{x}) = \mathrm{tt}, \quad \neg R(\vec{x}) \iff \chi_R(\vec{x}) = \mathrm{ff}.$$

An algorithm *decides* R if it computes χ_R.

Strings. For any set L, $L^* = L^{<\omega}$ is the set of *strings*[2] (words, finite sequences) from L, and we will use mostly standard notations for them:

$$
\begin{aligned}
&\mathrm{nil} = (\,) \quad \text{(the empty string)},\\
&|(u_0,\dots,u_{m-1})| = m, \quad (u_0,\dots,u_{m-1})_i = u_i \quad (i < m),\\
&\mathrm{app}((t),(u_0,\dots,u_{m-1})) = (t,u_0,\dots,u_{m-1})\\
&\qquad\qquad \text{(with } \mathrm{app}(v,u) = \mathrm{nil} \text{ is } |v| \neq 1)\\
&\mathrm{head}((u_0,\dots,u_{m-1})) = (u_0) \quad (= \mathrm{nil} \text{ if } u = \mathrm{nil}),\\
&\mathrm{tail}((u_0,\dots,u_{m-1})) = (u_1,\dots,u_{m-1}) \quad (= \mathrm{nil} \text{ if } u = \mathrm{nil}),\\
&(u_0,\dots,u_{m-1}) * (v_0,\dots,v_{n-1}) = (u_0,\dots,u_{m-1},v_0,\dots,v_{n-1}),\\
&u \sqsubseteq v \iff (\exists w)[u * w = v], \quad \text{(the initial segment relation)},\\
&u \sqsubsetneq v \iff u \sqsubseteq v \;\&\; u \neq v.
\end{aligned}
\tag{1A-4}
$$

The definitions of $\mathrm{app}(v,u)$ (*append*) and $\mathrm{head}(u)$ in effect identify a member t of L with the string $(t) \in L^*$, which simplifies in some ways dealing with strings.

Trees. For our purposes, a (finite, non-empty, rooted, $\mathbb{N}$-labelled) *tree* on a set X is any finite set $\mathcal{T} \subset (\mathbb{N} \times X)^{<\omega}$ of non-empty finite sequences (*nodes*) from $\mathbb{N} \times X$ which has a unique node of length 1, its *root*, and is closed under initial segments,

$$\emptyset \neq u \sqsubseteq v \in \mathcal{T} \Longrightarrow u \in \mathcal{T}.$$

The *children* of a node $u \in \mathcal{T}$ are all one-point extensions $u * (y) \in \mathcal{T}$, and its (out-) *degree* is the maximal number of children that any node has. A node is a *leaf* if it has no children.

A node v is *below* a node u if there is some w such that $v = u * w$, and in that case

$$\mathrm{distance}(u,v) = \begin{cases} \mathrm{length}(w) & \text{if } u * w = v,\\ \infty & \text{if } u \not\sqsubseteq v. \end{cases}$$

The *depth* of a node is its distance from the root and the depth of the tree is the largest of these numbers,

$$\mathrm{depth}(\mathcal{T}) = \max\{\mathrm{distance}(\mathrm{root},u) : u \in \mathcal{T}\}.$$

The *size* of a tree is the number of its nodes

$$\mathrm{size}(\mathcal{T}) = |\mathcal{T}|,$$

[2]Sometimes we denote strings by simply listing their elements

$$u_0u_1\cdots v_{m-1} = (u_0,u_1,\dots,u_{m-1}),$$

especially when we think of them as "words" from some alphabet L of "symbols"; and in such cases, we typically use "$\equiv$" to denote the equality relation on words, since "$=$" is often one of the symbols in the alphabet.

and it is easy to check that

$$\text{size}(\mathcal{T}) \leq \text{degree}(\mathcal{T})^{\text{depth}(\mathcal{T})} \tag{1A-5}$$

by induction on depth$(\mathcal{T})$.[3]

For each $u \in \mathcal{T}$, let

$$\mathcal{T}_u = \{v \in X^{<\omega} : u * v \in \mathcal{T}\}.$$

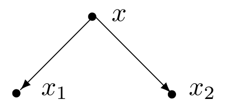
If u is not a leaf, then this is the *subtree* of $\mathcal{T}$ below u, with root u, and if u is a leaf then $\mathcal{T}_u = \emptyset$.

For each tree $\mathcal{T}$, let

$$\mathcal{T}' = \{u \in \mathcal{T} : u \text{ is not a leaf}\}.$$

If $\mathcal{T}$ has more than one element, then this is the *derived* (pruned) subtree of $\mathcal{T}$, and clearly depth$(\mathcal{T}')$ = depth$(\mathcal{T}) - 1$. Again, it is convenient to have the notation for a singleton $\mathcal{T}$, for which $\mathcal{T}' = \emptyset$.

In dealing with these trees, we will think of them as sets of sequences from X mostly disregarding the labels: their only purpose is to allow a node to have several "identical" children which are counted separately. For example, we will want to draw the tree



and assume it has this structure (with a root which has two children) even if it happens that $x_1 = x_2$; so formally

$$T = \{((0, x)), ((0, x), (0, x_1)), ((0, x), (1, x_2))\},$$

but we will indicate this by the simpler

$$T = \{(x), (x, x_1), (x, x_2)\}.$$

For example, if $z \in X$ and $\mathcal{T}, \mathcal{T}_1, \dots, \mathcal{T}_k$ are trees on X, then Top$(z, \mathcal{T}_1, \dots, \mathcal{T}_k)$ is the tree with root z and $\mathcal{T}_1, \dots, \mathcal{T}_k$ immediately below it. We will draw the result of this operation as if

$$\begin{aligned} \text{Top}(z, \mathcal{T}_1, \dots, \mathcal{T}_k) & \\ = \{(z, x_1, \dots, x_n) \mid & \text{ for some } i = 1, \dots, k, (x_1, \dots, x_n) \in \mathcal{T}_i\}; \end{aligned} \tag{1A-6}$$

the formal definition, with the labels, is the more formidable

$$\begin{aligned} \text{Top}(z, \mathcal{T}_1, \dots, \mathcal{T}_k) = \{((0, z), (\langle i, j_1 \rangle, x_1), \dots, (\langle i, j_n \rangle, x_n)) & \\ \mid i = 1, \dots, k, ((j_1, x_1), \dots, (j_n, x_n)) \in \mathcal{T}_i\}, & \end{aligned}$$

and $\langle i, j \rangle$ is some pairing function on $\mathbb{N}$, e.g., $\langle i, j \rangle = 2^{i+1}3^{j+1}$.

[3]Setting $0^0 = 1$ to cover the basis depth$(\mathcal{T})$ = degree$(\mathcal{T}) = 0$.

Problems for Section 1A

x1A.1. **Problem.** Suppose $\mathcal{T}$ is a tree on X, $C \subseteq X$, $H \in \mathbb{N}$, and for all $u, x_0, \dots, x_n$ with $n \geq H$,

$$u * (x_0, \dots, x_{n-1}) \in \mathcal{T} \Longrightarrow (\exists i)[x_i \in C];$$

then

$$\text{size}(\mathcal{T}) \leq \text{degree}(\mathcal{T})^H + |C|\text{degree}(\mathcal{T})^H.$$

1B. Partial functions and the Fixed Point Lemma

For any two sets A, W, an n-ary partial function $f : A^n \rightharpoonup W$ is a function $f : D_f \to W$ defined on some subset of A^n. For $\vec{x} \in A^n$, we set

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

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*:

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

so that in particular,

$$f(g_1(\vec{x}), \dots, g_n(\vec{x}))\downarrow \Longrightarrow g_1(\vec{x})\downarrow, \dots, g_n(\vec{x})\downarrow .$$

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 set A and partial functions (on various sets) as arguments and give a value in some set W (when they converge). Such a functional F is *monotone*, if

$$\begin{aligned} &F(\vec{x}, p_1, \dots, p_k)\downarrow \ \&\ p_1 \sqsubseteq q_1 \ \&\ \cdots \ \&\ p_k \sqsubseteq q_k\\ &\qquad\qquad\qquad \implies F(\vec{x}, p_1, \dots, p_k) = F(\vec{x}, q_1, \dots, q_k), \end{aligned}$$

and it is *continuous* if

$$\begin{aligned}&F(\vec{x},p_1,\dots,p_k)\downarrow\\ \Longrightarrow\ &(\exists \text{ finite } p_1^0 \sqsubseteq p_1,\dots,p_k^0 \sqsubseteq p_k)[F(\vec{x},p_1^0,\dots,p_k^0) = F(\vec{x},p_1,\dots,p_k)],\end{aligned}$$

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:

1B.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

$$p(\vec{x}) = F(\vec{x},p) \tag{1B-1}$$

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

$$\overline{p}(\vec{x}) = F(\vec{x},\overline{p}) \quad (\vec{x} \in A^n),$$
$$\text{if } (\forall \vec{x})[F(\vec{x},q)\downarrow \implies F(\vec{x},q) = q(\vec{x})],\ \text{then } \overline{p} \sqsubseteq q.$$

Similarly, every system of mutual monotone and continuous recursive equations

$$\begin{cases} p_1(\vec{x}_1) &= F_1(\vec{x}_1,p_1,\dots,p_K)\\ &\vdots\\ p_K(\vec{x}_K) &= F_K(\vec{x}_K,p_1,\dots,p_K)\end{cases} \tag{1B-2}$$

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

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

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

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

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

and set $\overline{p} = \bigcup\{\overline{p}^k : k \in \mathbb{N}\}$, i.e.,

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

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

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

by the definition, and hence $F(\vec{x},\overline{p}) = w$, by monotonicity, since $\overline{p}^k \sqsubseteq \overline{p}$. On the other hand, if $F(\vec{x},\overline{p}) = w$, then there is a finite $q \sqsubseteq \overline{p}$ such that $F(\vec{x},q) = w$, by continuity, and then there is some k such that $q \sqsubseteq \overline{p}^k$;

thus, by monotonicity, $F(\vec{x}, \overline{p}^k) = \overline{p}^{k+1}(\vec{x}) = w$, and so $\overline{p}(\vec{x}) = w$, which completes the proof that, for all $\vec{x}$,

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

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

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

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

The argument for recursive systems of equations is similar. ⊣

It is well known that the hypothesis of continuity is not needed for this basic lemma, cf. Theorem 7.36 in Moschovakis [2006]. This is a classical result of elementary set theory, whose proof, however, requires some work. We will not need it in these notes.

Problems for Section 1B

To solve a recursive equation (1B-1) or a system (1B-2) means to identify the least solution(s) in explicit terms. For example, the solution of

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

in $\mathbb{N}$ 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. In the problems which follow, individual variables vary over $\mathbb{N}$, and function variables vary over partial functions on $\mathbb{N}$ (of various arities).

x1B.1. **Problem.** Solve in $\mathbb{N}$ the recursive equation

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

x1B.2. **Problem.** Solve in $\mathbb{N}$ the recursive equation

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

x1B.3. **Problem.** Consider the following recursive equation in $\mathbb{N}$:

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

Prove that if $\overline{f}$ is its least solution, then $\overline{f}(x, y, 0) = x + y$.

x1B.4. **Problem.** Solve in $\mathbb{N}$ the recursive equation

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

where $\phi : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ is some fixed, given partial function.

x1B.5. **Problem.** Solve in $\mathbb{N}$ the recursive equation

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

x1B.6. **Problem.** Solve in L^* the recursive equation

$$f(u) = \text{if } (u = \mathrm{nil}) \text{ then nil else } f(\mathrm{tail}(u)) * \mathrm{head}(u).$$

1C. Equational logic with partial terms and conditionals

To apply the basic notions of equational logic to the theory of computation, we must introduce two small wrinkles: allow the interpretations of function symbols by partial functions, since computations often diverge, and add *branching* (conditionals) to the term-formation rules. We also need to embrace finite structures (even empty ones) and structures in which the identity relation $=$ is not one of the primitives.

(Many-sorted, partial) structures. A pair (S, Φ) is a *signature* if the *set of sorts* S is not empty, containing in particular the *boolean sort* `boole`, and the *vocabulary* Φ is a set of function symbols, each with an assigned *type* of the form

$$\mathrm{type}(\phi) \equiv (s_1, \dots, s_n, \mathrm{sort}(\phi))$$

with $s_1, \dots, s_n \in S \setminus \{\texttt{boole}\}$ and $\mathrm{sort}(\phi) \in S$. A (partial) (S, Φ)-*structure* is a pair

$$\mathbf{A} = (\{A_s\}_{s \in S}, \boldsymbol{\Phi}) = (\{A_s\}_{s \in S}, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi}), \tag{1C-1}$$

where each A_s is a set; $A_{\texttt{boole}}$ is the set of truth values $\{\mathrm{tt}, \mathrm{ff}\}$; and for each $\phi \in \Phi$,

$$\text{if } \mathrm{type}(\phi) = (s_1, \dots, s_n, s), \text{ then } \phi^{\mathbf{A}} : A_{s_1} \times \cdots \times A_{s_n} \rightharpoonup A_s.$$

The convergent objects $\phi^{\mathbf{A}}$ with $\mathrm{type}(\phi) = (s)$ are the *distinguished elements of sort s* of $\mathbf{A}$.

We will adopt the natural convention about the identity symbol: if $=_s$ occurs in the vocabulary Φ, it is then interpreted in every (S, Φ)-structure $\mathbf{A}$ by *some subfunction* $=_s^{\mathbf{A}} \sqsubseteq =_{A_s}$ of the (total) identity relation $=_{A_s} : A_s \to \{\mathrm{tt}, \mathrm{ff}\}$ on A_s—*not necessarily by* $=_{A_s}$. We will also use the notations

$$\mathrm{eq}(x,y) \iff x = y, \quad \mathrm{eq}_w(x) \iff x = w$$

if there is any danger of confusing the formal symbol "=" in the signature with the relation of identity between values of partial functions as this was defined in Section 1B.

Φ-structures. Most often there is just one sort $\mathtt{a}$ (other than $\mathtt{boole}$) and Φ is finite: we describe these structures as usual, by identifying the *universe* $A = A_{\mathtt{a}}$, listing $\mathbf{\Phi}$, and letting the notation suggest

$$\text{type}(\phi) \equiv (n_\phi, s) \equiv (\text{arity}(\phi), \text{sort}(\phi)) :\equiv (\underbrace{\mathtt{a}, \dots, \mathtt{a}}_{n_\phi}, s)$$

for every $\phi \in \Phi$. Typical are the basic structures of unary and binary arithmetic

(1C-2) $\quad \mathbf{N}_u = (\mathbb{N}, 0, S, \text{Pd}, \text{eq}_0), \quad \mathbf{N}_b = (\mathbb{N}, 0, \text{parity}, \text{iq}_2, \text{em}_2, \text{om}_2, \text{eq}_0),$

where

$$\text{em}_2(x) = 2x, \ \text{om}_2(x) = 2x + 1$$

are the operations of *even* and *odd multiplication by* 2. More generally, for any $k \geq 3$, the structure of *k-ary arithmetic* is

(1C-3) $$\mathbf{N}_k = (\mathbb{N}, 0, \text{m}_{k,0}, \dots, \text{m}_{k,k-1}, \text{iq}_k, \text{rem}_k, \text{eq}_0),$$

where $\text{m}_{k,i}(x) = kx + i$, $\text{iq}_k(x) = \text{iq}(x, k)$ and $\text{rem}_k(x) = \text{rem}(x, k)$. These are *total structures*, as is the standard structure of Peano arithmetic

(1C-4) $$\mathbf{N} = (\mathbb{N}, 0, 1, +, \cdot, =)$$

Another interesting structure is that of strings (or lists) from a set L,

(1C-5) $$\mathbf{L}^* = (L^*, \text{nil}, =_{\text{nil}}, \text{head}, \text{tail}, \text{app}).$$

An example of a genuinely partial structure is a *field* (with identity)

$$\mathbf{F} = (F, 0, 1, +, -, \cdot, \div, =),$$

where the quotient $x \div y$ is defined only when $y \neq 0$.

There are many interesting examples of many-sorted structures, e.g., a *vector space* V over a field F

$$\mathbf{V} = (V, F, 0_F, 1_F, +_F, -_F, \cdot_F, \div_F, 0_V, +_V, -_V, \cdot)$$

where the last primitive $\cdot : F \times V \to V$ is scalar-vector multiplication and the other symbols have their natural meanings. On the other hand, dealing directly with many sorts is tedious, and we will work with one-sorted Φ-structures. The more general versions follow by "identifying" a many-sorted $\mathbf{A}$ as in (1C-1) with the single-sorted

(1C-6) $$(\uplus_{s \in S'} A_s, \{A_s : s \in S'\}, \mathbf{\Phi}) \quad (S' = S \setminus \{\mathtt{boole}\}),$$

where $\uplus_{s \in S'} A_s = \bigcup\{(s, x) : s \in S' \ \& \ x \in A_s\}$ is the *disjoint union* of the basic universes of $\mathbf{A}$, $A_s(x) \Leftrightarrow x \in A_s$ for $s \neq \mathtt{boole}$, and the primitives in $\mathbf{\Phi}$ are as before, undefined on arguments not of the appropriate type. There are still two sorts in Φ-structures, $\mathtt{a}$ and $\mathtt{boole}$, and we will need to deal with both partial functions and relations on their universe. Typically we will write

$$f : A^n \rightharpoonup A_s \quad (s \in \{\mathtt{a}, \mathtt{boole}\})$$

to cover both partial functions and relations on the universe A, most often skipping the tiresome side notation explaining what this "s" stands for.

Restrictions. If $\mathbf{A} = (A, \mathbf{\Phi})$ is a Φ-structure and $U \subseteq A = A_{\mathtt{a}}$, we set

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

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

$$f \restriction U(x_1, \dots , x_n) = w \iff x_1, \dots , x_n \in U, w \in U_s \ \&\ f(x_1, \dots , x_n) = w.$$

Expansions and reducts. An *expansion* of a Φ-structure $\mathbf{A}$ is obtained by adding new primitives to $\mathbf{A}$,

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

Conversely, the *reduct* $\mathbf{A} \restriction \Phi_0$ of a structure $\mathbf{A} = (A, \mathbf{\Phi})$ to a subset $\Phi_0 \subseteq \Phi$ of its vocabulary is defined by removing all the operations in $\mathbf{\Phi} \setminus \mathbf{\Phi}_0$. For example, the reduct of the field of real numbers to $\{0, +, -\}$ is the additive group on $\mathbb{R}$,

$$(\mathbb{R}, 0, 1, +, -, \cdot, \div) \restriction \{0, +, -\} = (\mathbb{R}, 0, +, -).$$

Diagrams. The (equational) *diagram* of a Φ-structure $\mathbf{A}$ is the set

$$\mathrm{eqdiag}(\mathbf{A}) = \{(\phi, \vec{x}, w) : \phi \in \Phi, \vec{x} \in A^n, w \in A_{\mathrm{sort}(\phi)} \text{ and } \phi^{\mathbf{A}}(\vec{x}) = w\},$$

and its *visible universe* is the set of members of A which occur in eqdiag($\mathbf{A}$),

$$A_{\mathrm{vis}} = \{x \in A : \exists(\phi, x_0, \dots , x_{n-1}, x_n) \in \mathrm{eqdiag}(\mathbf{A})(\exists i)[x = x_i]\}.$$

It is sometimes convenient to specify a structure $\mathbf{A}$ by giving its equational diagram, which (by convention then) means that $A = A_{\mathrm{vis}}$. For example, if we set

(1C-7) $$\mathbf{U} = \{2 + 1 = 3,\ 2 + 3 = 5,\ 2 \leq 5,\ 5 \not\leq 1\}$$

with $\Phi = \{0, S, +, \leq\}$, then $U = U_{\mathrm{vis}} = \{1, 2, 3, 5\}$ and $\mathbf{U}$ is a finite structure in which (among other things) S is interpreted by the empty partial function. And we have used here the obvious conventions, to write in diagrams

$$\phi(\vec{x}) = w,\ R(\vec{x}),\ \neg R(\vec{x})$$

rather than the more pedantic

$$(\phi, \vec{x}, w),\ (R, \vec{x}, \mathrm{tt}),\ (R, \vec{x}, \mathrm{ff})$$

and to use "infix notation", i.e., write $x + y$ rather than $+(x, y)$.

Substructures or pieces. A (partial) *substructure* $\mathbf{U} \subseteq_p \mathbf{A}$ or *piece* of a Φ-structure $\mathbf{A}$ is a structure of the same vocabulary Φ, such that $U \subseteq A$ and for every $\phi \in \Phi$, $\phi^{\mathbf{U}} \sqsubseteq \phi^{\mathbf{A}}$, i.e.,

$$\Big(\vec{x} \in U^n \;\&\; w \in U_s \;\&\; \phi^{\mathbf{U}}(\vec{x}) = w\Big) \Longrightarrow \phi^{\mathbf{A}}(\vec{x}) = w.$$

A piece $\mathbf{U}$ is *strong* (or *induced*) if in addition

$$\Big(\vec{x} \in U^n \;\&\; w \in U_s \;\&\; \phi^{\mathbf{A}}(\vec{x}) = w\Big) \Longrightarrow \phi^{\mathbf{U}}(\vec{x}) = w,$$

in which case $\mathbf{U} = \mathbf{A} \restriction U$, the restriction of $\mathbf{A}$ to the universe of $\mathbf{U}$. Notice that

$$\mathbf{U} \subseteq_p \mathbf{A} \iff U \subseteq A \;\&\; \mathrm{eqdiag}(\mathbf{U}) \subseteq \mathrm{eqdiag}(\mathbf{A}),$$

and if $U = U_{\mathrm{vis}}$, then

$$\mathbf{U} \subseteq_p \mathbf{A} \iff \mathrm{eqdiag}(\mathbf{U}) \subseteq \mathrm{eqdiag}(\mathbf{A}).$$

Notice also that we allow $U = \emptyset$, and *we do not insist that a substructure* $\mathbf{U} \subseteq_p \mathbf{A}$ *be closed under the primitives of* $\mathbf{A}$—in particular, it need not contain all the distinguished elements of $\mathbf{A}$. This is contrary to the usual terminology in mathematics and logic, where, for example, a *subfield* of a field F must (by definition) contain $0, 1$ and be closed under $+ .-, \cdot$ and $\div$. To avoid confusion, we have introduced and will sometimes use the awkward term "piece" for these objects, but it should be remembered that a piece $\mathbf{U}$ of a Φ-structure $\mathbf{A}$ is a Φ-structure in its own right.

Homomorphisms and embeddings. A *homomorphism* $\pi : \mathbf{U} \to \mathbf{V}$ of one Φ-structure into another is any mapping $\pi : U \to V$ such that

$$\phi^{\mathbf{U}}(x_1, \ldots, x_n) = w \Longrightarrow \phi^{\mathbf{V}}(\pi(x_1), \ldots, \pi(x_n)) = \pi(w). \tag{1C-8}$$

In reading this we extend π to $\{\mathrm{tt}, \mathrm{ff}\}$ by $\pi(\mathrm{tt}) = \mathrm{tt}, \pi(\mathrm{ff}) = \mathrm{ff}$, so that for partial relations it insures

$$\begin{aligned} R^{\mathbf{U}}(x_1, \ldots, x_n) &\Longrightarrow R^{\mathbf{V}}(\pi(x_1), \ldots, \pi(x_n)), \\ \neg R^{\mathbf{U}}(x_1, \ldots, x_n) &\Longrightarrow \neg R^{\mathbf{V}}(\pi(x_1), \ldots, \pi(x_n)). \end{aligned}$$

A homomorphism is an *embedding* $\pi : \mathbf{U} \rightarrowtail \mathbf{V}$ if it is injective (one-to-one), and it is an *isomorphism* $\pi : \mathbf{U} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbf{V}$ if it is a surjective embedding and, in addition, the inverse map $\pi^{-1} : U \rightarrowtail\!\!\!\!\twoheadrightarrow V$ is also an embedding. Clearly

$$\mathbf{U} \subseteq_p \mathbf{V} \iff U \subseteq V \text{ and the identity } \mathrm{id}_U : U \rightarrowtail V \text{ is an embedding.}$$

If $\pi : \mathbf{U} \to \mathbf{A}$ is a homomorphism, then $\pi[\mathbf{U}]$ is the piece of $\mathbf{A}$ with universe $\pi[U]$ and

$$\begin{aligned}\text{eqdiag}(\pi[\mathbf{U}]) = \{(\phi, \pi(x_1), \dots, \pi(x_n), \pi(w)) \\ : (\phi, x_1, \dots, x_n), w) \in \text{eqdiag}(\mathbf{U})\}.\end{aligned}$$

This construction is especially useful when $\pi : \mathbf{U} \rightarrowtail \mathbf{A}$ is an embedding, in which case $\pi : \mathbf{U} \rightarrowtail\!\!\!\!\twoheadrightarrow \pi[\mathbf{U}]$ is an isomorphism.

Syntax. The *terms* (with parameters) of a Φ-structure $\mathbf{A}$ are defined by the structural recursion

$$\begin{aligned}(\mathbf{A}\text{-terms}) \quad E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid x \ (x \in A) \\ \mid \mathsf{v}_i \mid \phi(E_1, \dots, E_{n_\phi}) \mid \text{if } E_0 \text{ then } E_1 \text{ else } E_2,\end{aligned}$$

where $\mathsf{v}_0, \mathsf{v}_1, \dots$ is a fixed sequence of individual variables of sort $\mathtt{a}$.[4] The definition assigns to each term a sort $\mathtt{boole}$ or $\mathtt{a}$ and sets type restrictions on the formation rules in the obvious way; for the conditional it is required that $\text{sort}(E_0) \equiv \mathtt{boole}$ and $\text{sort}(E_1) \equiv \text{sort}(E_2)$, and then $\text{sort}(E) \equiv \text{sort}(E_1)$. The propositional connectives on terms of boolean sort can be defined using the conditional:

$$\begin{aligned}(\text{1C-9}) \quad \neg E :\equiv \text{if } E \text{ then ff else tt}, \\ E_1 \ \& \ E_2 :\equiv \text{if } E_1 \text{ then } E_2 \text{ else ff}, \\ E_1 \vee E_2 :\equiv \text{if } E_1 \text{ then tt else } E_2, \\ E_1 \to E_2 :\equiv \neg E_1 \vee E_2, \quad E_1 \leftrightarrow E_2 :\equiv (E_1 \to E_2) \ \& \ (E_2 \to E_1).\end{aligned}$$

The *parameters* of an $\mathbf{A}$-term E are the members of A which occur in it. A term E is **pure** (or a Φ-*term*) if it has no parameters, and **closed** if no variables occur in it.

The *subterms* of a term are defined as usually.

We will also need the terms without conditionals, which we will now call **algebraic A-terms**. They are defined by the simpler recursion

$$(\text{Algebraic } \mathbf{A}\text{-terms}) \qquad E :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathsf{v}_i \mid x \mid \phi(E_1, \dots, E_{n_\phi}).$$

Notice that these include terms of sort $\mathtt{boole}$, e.g., tt, ff and $\mathsf{R}(E_1, \dots, E_n)$ if $\mathsf{R} \in \Phi$ is of $\mathtt{boole}$ sort, so they are more general than the usual terms of logic which are all of sort $\mathtt{a}$.

The *depth* of an algebraic term is defined by the recursion

$$\begin{aligned}\text{depth}(\mathrm{tt}) = \text{depth}(\mathrm{ff}) = \text{depth}(\mathsf{v}_i) = \text{depth}(x) = 0, \\ \text{depth}(\phi(E_1, \dots, E_n)) = \max\{\text{depth}(E_1), \dots, \text{depth}(E_n)\} + 1.\end{aligned}$$

[4]We do not allow variables of boolean sort. This is a convenient choice for the kinds of algorithms we will want to analyze, and it does not affect in any serious way the breadth of applicability of the results we will prove.

Formal substitution. For any two **A**-terms E, M and any variable x,

$$E\{\mathsf{x} :\equiv M\} = \text{ the result of replacing every occurrence of } \mathsf{x} \text{ in } E \text{ by } M,$$

which is (easily) also a term. We will use extensively the familiar notation[5]

$$E(\mathsf{x}_1, \dots, \mathsf{x}_n) :\equiv (E, (\mathsf{x}_1, \dots, \mathsf{x}_n))$$

for a pair of a term and a sequence of distinct variables which includes all the variables that occur in E. The convention provides a useful notation for substitution: if $M_1, \dots, M_n$ are **A**-terms, then

$$E(M_1, \dots, M_n) :\equiv E\{\mathsf{x}_1 :\equiv M_1, \dots, \mathsf{x}_n :\equiv M_n\}.$$

In particular, if $x_1, \dots, x_n \in A$, then $E(x_1, \dots, x_n)$ is the closed **A**-term constructed by replacing each x_i by x_i.

Semantics. For a fixed Φ-structure **A**, we define

$$\text{den} : \{\text{closed } \mathbf{A}\text{-terms}\} \rightharpoonup A \cup \{\mathrm{tt}, \mathrm{ff}\}$$

by the obvious recursive clauses:

$$\text{den}(\mathrm{tt}) = \mathrm{tt}, \quad \text{den}(\mathrm{ff}) = \mathrm{ff}, \quad \text{den}(x) = x$$

$$\text{den}(\phi(M_1, \dots, M_{n_\phi})) = \phi^{\mathbf{A}}(\text{den}(M_1), \dots, \text{den}(M_{n_\phi}))$$

$$\text{den}(\text{if } M_0 \text{ then } M_1 \text{ else } M_2) = \begin{cases} \text{den}(M_1), & \text{if den}(M_0) = \mathrm{tt}, \\ \text{den}(M_2), & \text{if den}(M_0) = \mathrm{ff} \\ \uparrow, & \text{otherwise.} \end{cases}$$

We call den(M) the *denotation* of the closed **A**-term M (if den$(M)\downarrow$), and in that case, clearly

$$\text{sort}(M) = \mathtt{boole} \Longrightarrow \text{den}(M) \in \{\mathrm{tt}, \mathrm{ff}\}, \quad \text{sort}(M) = \mathtt{a} \Longrightarrow \text{den}(M) \in A.$$

When we need to exhibit the structure in which the denotation is computed, we write den$(\mathbf{A}, M)$, or we use model-theoretic notation,

$$\mathbf{A} \models E = M \iff \text{den}(\mathbf{A}, E) = \text{den}(\mathbf{A}, M).$$

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

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

[5]We will also adopt the familiar abuse of this notation and refer to these pairs $E(\vec{\mathsf{x}})$ as "terms", as there is no generally accepted name for them.

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

$$\mathrm{den}(M_0) = \mathrm{tt} \implies \mathrm{den}(\text{if } M_0 \text{ then } M_1 \text{ else } M_2) = \mathrm{den}(M_1),$$

whether $\mathrm{den}(M_2)$ converges or not.

Notice that as we defined them in (1C-9), the connectives of propositional logic give the correct truth value only when both their constituents converge.

Explicit definability. A partial function $f : A^n \rightharpoonup A_s$ is *explicitly defined* or just *explicit* in $\mathbf{A}$ if there is a pure Φ-term $E(\vec{\mathsf{x}})$ such that

(1C-10) $$f(\vec{x}) = \mathrm{den}(\mathbf{A}, E(\vec{x})) \quad (\vec{x} \in A^n).$$

We set

$$\mathbf{expl}(\mathbf{A}) = \{f : A^n \rightharpoonup A_s : f \text{ is explicit in } \mathbf{A}\}.$$

If (1C-10) holds with a term $E(\vec{\mathsf{x}})$ with parameters, we say that f is *explicit with parameters* in $\mathbf{A}$.

First order logic. If all the primitives of a Φ-structure $\mathbf{A}$ are total functions (or relations), we can think of $\mathbf{A}$ as a first-order structure and interpret first-order logic on it. We will not have many occasions to do this in general, but the *quantifier-free* formulas are occasionally needed and so we include here their definition:

(1C-11)
$$\theta :\equiv \mathrm{tt} \mid \mathrm{ff} \mid \mathsf{R}(E_1, \ldots E_m) \mid (\neg\theta_1) \mid (\theta_1 \;\&\; \theta_2) \mid (\theta_1 \vee \theta_2) \mid (\theta_1 \rightarrow \theta_2)$$

where $\mathsf{R} \in \Phi$ is of Boolean sort and $E_1, \ldots, E_n$ are pure, algebraic Φ-terms of sort a. These are interpreted naturally on any total Φ-structure and they comprise the *quantifier-free relations* of $\mathbf{A}$.

Generation. For a fixed Φ-structure $\mathbf{A}$ and any $X \subseteq A$, we set

$$\begin{aligned} G_0[X] &= X, \\ G_{m+1}[X] &= G_m[X] \cup \{\phi^{\mathbf{A}}(u_1, \ldots, u_{n_f}) : u_1, \ldots, u_{n_\phi} \in G_m[X]\}, \\ G_\infty[X] &= \textstyle\bigcup_m G_m[X]. \end{aligned}$$

By a simple induction on m,

(1C-12) $$G_{m+k}[X] = G_m[G_k[X]].$$

We write

$$G_m(\vec{x}) = G_m[\{x_1, \ldots, x_n\}], \quad G_\infty(\vec{x}) = \textstyle\bigcup_m G_m(\vec{x})$$

for the set generated in m steps by a tuple $\vec{x} = (x_1, \ldots, x_n) \in A^n$. If the structure in which these sets are computed is not obvious, we write $G_m[\mathbf{A}, X], G_m(\mathbf{A}, \vec{x})$, and for the corresponding induced pieces of $\mathbf{A}$,

$$\mathbf{G}_m(\mathbf{A}, \vec{x}) = \mathbf{A} \restriction G_m(\mathbf{A}, \vec{x}), \quad \mathbf{G}_\infty(\mathbf{A}, \vec{x}) = \mathbf{A} \restriction G_\infty(\mathbf{A}, \vec{x}).$$

A structure $\mathbf{A}$ is *generated by* $\vec{x}$ if $\mathbf{A} = \mathbf{G}_\infty(\mathbf{A}, \vec{x})$, so that if it also finite, then $\mathbf{A} = \mathbf{G}_m(\mathbf{A}, \vec{x})$ for some m. Most often we will be concerned with finite pieces of some fixed $\mathbf{A}$ which are generated by a tuple of their members, and we set

$$\text{depth}(\mathbf{U}, \vec{x}) = \min\{m : \vec{x} \in U^n, \mathbf{U} = \mathbf{G}_m(\mathbf{U}, \vec{x}) \subseteq_p \mathbf{A}\}.$$

The *size* of a finite $\mathbf{U} \subseteq_p \mathbf{A}$ is the number of all visible elements of its universe,

$$\text{(1C-13)} \quad \text{size}(\mathbf{U}) = |U_{\text{vis}}| = \Big|\{v \in U : v \text{ occurs in some } (\phi, \vec{u}, w) \in \text{eqdiag}(\mathbf{U})\}\Big| \leq |U|.$$

We also need the *depth of an element below a tuple*,

$$\text{(1C-14)} \quad \text{depth}(w; \mathbf{A}, \vec{x}) = \min\{m : w \in G_m(\mathbf{A}, \vec{x})\}, \quad (w \in G_\infty(\mathbf{A}, \vec{x})).$$

Clearly,

$$\text{depth}(x_i; \mathbf{A}, \vec{x}) = 0,$$
$$\text{depth}(\phi^{\mathbf{A}}(u_1, \dots, u_{n_\phi}); \mathbf{A}, \vec{x}) = \max\{\text{depth}(u_i; \mathbf{A}, \vec{x}) : i = 1, \dots, n_\phi\} + 1.$$

1C.1. **Proposition.** *If* $\mathbf{U}$ *is a* Φ*-structure,* $\vec{x} \in U^n$ *and* $\text{depth}(w; \mathbf{U}, \vec{x}) = m$ *for some* $w \in U$*, then*

$$\text{(1C-15)} \qquad m \leq \text{size}(\mathbf{G}_m(\mathbf{U}, \vec{x})) \leq |\text{eqdiag}(\mathbf{G}_m(\mathbf{U}, \vec{x}))|.$$

It follows that if $\mathbf{U}$ *is finite and generated by* $\vec{x}$*, then*

$$\text{(1C-16)} \qquad \text{depth}(\mathbf{U}, \vec{x}) \leq \text{size}(\mathbf{U}) \leq |\text{eqdiag}(\mathbf{U})|.$$

Proof of (1C-15) is by induction on m, the basis being trivial since all three numbers in it are 0.

For the induction step in the first inequality, we are given some w with

$$\text{depth}(w; \mathbf{U}, \vec{x}) = m + 1,$$

so that $w = \phi^{\mathbf{U}}(u_1, \dots, u_n)$ and for some i, $\text{depth}(u_i; \mathbf{U}, \vec{x}) = m$. By the induction hypothesis,

$$m \leq \text{size}(\mathbf{G}_m(\mathbf{U}, \vec{x})) \leq \text{size}(\mathbf{G}_{m+1}(\mathbf{U}, \vec{x})) - 1,$$

the latter because w occurs in the entry

$$(\phi, u_1, \dots, u_n, w) \in \text{eqdiag}(\mathbf{G}_{m+1}(\mathbf{U}, \vec{x}))$$

and is not a member of $G_m(\mathbf{U}, \vec{x})$. So $m + 1 \leq \text{size}(G_{m+1}(\mathbf{U}, \vec{x}))$.

For the induction step in the proof of the second inequality, notice that (skipping $\mathbf{U}$ and $\vec{x}$ which remain constant in the argument),

$$G_{m+1} = G_m \cup \{\phi^{\mathbf{U}}(u_1, \dots, u_k) : u_1, \dots, u_k \in G_m \;\&\; \phi^{\mathbf{U}}(u_1, \dots, u_k) \notin G_m\}.$$

The first of these two disjoint pieces has size $\leq |\mathrm{eqdiag}(\mathbf{G}_m)|$ by the induction hypothesis, and to each w is the second piece we can associate in a one-to-one way some entry $(\phi, \vec{u}, w)$ in the diagram of $\mathbf{G}_{m+1}$ which is not in the diagram of $\mathbf{G}_m$, because $w \notin G_m$; so

$$\begin{aligned}\mathrm{size}(G_{m+1}) &\leq |\mathrm{eqdiag}(\mathbf{G}_m)| + \Big(|\mathrm{eqdiag}(\mathbf{G}_{m+1})| - |\mathrm{eqdiag}(\mathbf{G}_m)|\Big) \\ &= |\mathrm{eqdiag}(\mathbf{G}_{m+1})|. \qquad \dashv\end{aligned}$$

Problems for Section 1C

x1C.1. **Problem.** Give an example of an embedding $\phi : \mathbf{U} \rightarrowtail \mathbf{V}$ of one Φ-structure to another which is bijective but not an isomorphism.

x1C.2. **Problem** (Parsing for terms)**.** Prove that for any Φ-structure $\mathbf{A}$, every $\mathbf{A}$-term E satisfies exactly one of the following conditions.

1. $E \equiv \mathrm{tt}$, or $E \equiv \mathrm{ff}$, or $E \equiv x$ for some $x \in A$, or $E \equiv v$ for a variable v.
2. $E \equiv \phi(E_1, \dots, E_n)$ for a uniquely determined $\phi \in \Phi$ and uniquely determined terms $E_1, \dots, E_n$.
3. $E \equiv \text{if } E_0 \text{ then } E_1 \text{ else } E_2$ for uniquely determined E_0, E_1, E_2.

x1C.3. **Problem.** Give an example of two terms $E_1(\mathsf{x})$ and $E_2(\mathsf{x})$ 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 $\mathrm{den}(E_1(M)) \neq \mathrm{den}(E_2(M))$.

x1C.4. **Problem.** Show that for every term $E(\mathsf{x})$ and closed term M,

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

x1C.5. **Problem.** Show that for any two terms $E_1(\mathsf{x}), E_2(\mathsf{x})$, if M is closed, $\mathrm{den}(M)\downarrow$ and $\mathrm{den}(E_1(x)) = \mathrm{den}(E_2(x))$ for every $x \in A$, then

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

These results extend trivially to simultaneous substitutions.

x1C.6. **Problem.** For each of the following (and using the definitions in (1C-9), determine whether it is true or false in every structure $\mathbf{A}$ for all closed terms of boolean sort (sentences).

(1) $\models \text{if } \phi \text{ then } \psi_1 \text{ else } \psi_2 = \text{if } \neg\phi \text{ then } \psi_2 \text{ else } \psi_1$.
(2) $\models \neg(\phi \ \&\ \psi) = (\neg\phi) \vee (\neg\psi)$.
(3) $\models \neg(\phi \ \&\ \psi) \leftrightarrow (\neg\phi) \vee (\neg\psi) = \mathrm{tt}$.
(4) $\models \phi \ \&\ \psi = \psi \ \&\ \phi$.

x1C.7*. **Problem** (Explicit expansions)**.** Suppose the vocabulary Φ has a relation symbol R of arity $k > 0$ and $\mathbf{A}$ is a Φ-structure such that $R^{\mathbf{A}} : A^k \to \{\mathrm{tt}, \mathrm{ff}\}$ is total. Suppose $f : A^n \rightharpoonup A$ is explicit in $\mathbf{A}$. Let f be a fresh function symbol and let $(\mathbf{A}, f)$ be the expansion of $\mathbf{A}$ in which f is interpreted by f. Define a mapping

$$M \mapsto M^*$$

which assigns to each term M in the vocabulary $\Phi \cup \{\mathsf{f}\}$ a Φ-term M^* with the same variables, so that

$$\mathrm{den}((\mathbf{A}, f), M(\vec{y})) = \mathrm{den}(\mathbf{A}, M^*(\vec{y})).$$

Show also by a counterexample that the hypothesis about R cannot be removed.

x1C.8. **Problem.** Show that if $\pi : \mathbf{A} \to \mathbf{B}$ is a homomorphism of one Φ-structure into another, then for every Φ-term $M(\vec{\mathsf{x}})$ and all $\vec{x} \in A^n$,

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

where, naturally,

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

x1C.9. **Problem.** Prove that for every m and $\vec{x} \in A^n$,

$$G_m(\mathbf{A}, \vec{x}) = \{\mathrm{den}(\mathbf{A}, E(\vec{x})) : E(\vec{\mathsf{x}}) \text{ is pure, algebraic, } \mathrm{sort}(E(\vec{\mathsf{x}})) = \mathsf{a} \text{ and } \mathrm{depth}(E(\vec{\mathsf{x}})) \le m\},$$

x1C.10. **Problem.** Show that for every vocabulary Φ, there is a number a such that for every Φ-structure $\mathbf{A}$, every $\vec{x} \in A^n$ and every m,

$$|G_m(\vec{x})| \le C^{2^{am}} \qquad (C = n + 2).$$

Give an example of a structure $\mathbf{A}$ where $|G_m(x)|$ cannot be bounded by a single exponential in m.

x1C.11. **Problem.** Prove that a partial function $f : A^n \rightharpoonup A_s$ is $\mathbf{A}$-explicit if and only if there are pure, algebraic terms $C_i(\vec{\mathsf{x}})$ of boolean sort and algebraic terms $V_i(\vec{\mathsf{x}})$ such that

$$f(\vec{x}) = \begin{cases} \mathrm{den}(V_0(\vec{x})) & \text{if } \mathrm{den}(C_0(\vec{x})) = \mathrm{tt}, \\ \mathrm{den}(V_1(\vec{x})) & \text{ow., if } \mathrm{den}(C_0(\vec{x}))\downarrow \ \& \ \mathrm{den}(C_1(\vec{x})) = \mathrm{tt}, \\ \ \vdots & \\ \mathrm{den}(V_k(\vec{x})) & \text{ow., if } \mathrm{den}(C_{k-1}(\vec{x}))\downarrow \ \& \ \mathrm{den}(C_k(\vec{x})) = \mathrm{tt}, \\ \mathrm{den}(V_{k+1}(\vec{x})) & \text{ow., if } \mathrm{den}(C_k(\vec{x}))\downarrow . \end{cases}$$

Infer that for a total structure $\mathbf{A}$, a relation $R \subseteq A^n$ is $\mathbf{A}$-explicit if and only if it is definable by a quantifier-free formula, as these are defined in Section 1C.

This representation of explicit functions and relations is especially interesting (and has been much studied) for the *Presburger structure*

$$\mathbf{N}_{\mathrm{Pres}} = (\mathbb{N}, 0, 1, +, \dot{-}, <, =, \{\mathrm{rem}_m, \mathrm{iq}_m\}_{m \geq 2}) \tag{1C-17}$$

where $\mathrm{rem}_m(x) = \mathrm{rem}(x, m), \mathrm{iq}_m(x) = \mathrm{iq}(x, m)$. This is because $\mathbf{expl}(\mathbf{N}_{\mathrm{Pres}})$ (the *Presburger functions*) is the set of all functions on $\mathbb{N}$ whose graphs are first-order definable in additive (Presburger) arithmetic

$$\mathbf{N}_{+} = (\mathbb{N}, 0, 1, +, =).$$

This follows from the classical *quantifier elimination result* for Presburger arithmetic, cf. Enderton [2001]. The Presburger functions are *piecewise linear* in the following, precise sense:

x1C.12*. **Problem.** Prove that if $f : \mathbb{N}^n \to \mathbb{N}$ is a Presburger function, then there is a partition of $\mathbb{N}^n$ into disjoint sets $D_1, \ldots, D_k \subseteq \mathbb{N}^n$ that are definable by quantifier free formulas of $\mathbf{N}_{\mathrm{Pres}}$, such that for each $i = 1, \ldots, k$, and suitable rational numbers $q_0, q_1, \ldots, q_n$,

$$f(\vec{x}) = q_0 + q_1 x_1 + \cdots + q_n x_n \qquad (\vec{x} \in D_i).$$

HINT: Check that the result holds for the primitives of $\mathbf{N}_{\mathrm{Pres}}$ and that the class of functions which satisfy it is closed under composition. Notice that $f(\vec{x}) \in \mathbb{N}$ in this expression, although some of the $q_i \in \mathbb{Q}$ may be proper, positive or negative fractions.

The next three problems will follow from results that we will prove later, but perhaps there are elementary proofs of them that can be given now:

x1C.13*. **Problem.** Prove that the successor function $S : \mathbb{N} \to \mathbb{N}$ is not explicit in binary arithmetic $\mathbf{N}_b$.

x1C.14*. **Problem.** Prove that the parity relation

$$\mathrm{parity}(x) \iff 2 \mid x$$

is not quantifier-free definable in unary arithmetic $\mathbf{N}_u$, and the successor relation

$$S(x, y) \iff x + 1 = y$$

is not quantifier-free definable in binary arithmetic $\mathbf{N}_b$.

x1C.15*. **Problem.** Prove that the divisibility relation

$$x \mid y \iff y \neq 0 \ \&\ \mathrm{rem}(x, y) = 0$$

is not quantifier-free definable in the Presburger structure $\mathbf{N}_{\mathrm{Pres}}$.

There are many obvious, similar questions relating the various primitives of Presburger arithmetic which also do not seem to be easy to answer now.

1D. Some basic algorithms

We review here briefly some classical examples of *algorithms from primitives*, primarily to illustrate how recursive equations can be interpreted as instructions for the computation of their least fixed points. This process will be made precise in the next chapter.

The merge-sort algorithm. Suppose L is a set with a fixed total ordering $\le$ on it. A string

$$v = v_0 v_1 \cdots v_{n-1} = (v_0, \dots , v_{n-1}) \in L^*$$

is *sorted* (in non-decreasing order), if $v_0 \le v_1 \le \cdots \le v_{n-1}$, and for each $u \in L^*$, $\mathrm{sort}(u)$ is the sorted "rearrangement" of u,

(1D-1) $$\mathrm{sort}(u) =_{\mathrm{df}} \text{the unique, sorted } v \in L^* \text{ such that for some}$$
$$\text{bijection } \pi : \{0,\dots,n-1\} \rightarrowtail\!\!\!\!\twoheadrightarrow \{0,\dots,n-1\},$$
$$v = (u_{\pi(0)}, u_{\pi(1)}, \dots , u_{\pi(n-1)}).$$

The efficient computation of $\mathrm{sort}(u)$ is important in many computing applications and many sorting algorithms have been studied. We consider here just one of these algorithms, which is easily expressed by a system of two, simple, recursive equations.

The merge-sort uses as a "subroutine" an algorithm for *merging* two strings, which is defined as follows.

1D.1. **Proposition.** *The equation*

(1D-2) $$\begin{aligned}\mathrm{merge}(w,v) = {} & \text{if } (|w| = 0) \text{ then } v\\ & \text{else if } (|v| = 0) \text{ then } w\\ & \text{else if } (w_0 \le v_0) \text{ then } (w_0) * \mathrm{merge}(\mathrm{tail}(w), v)\\ & \text{else } (v_0) * \mathrm{merge}(w, \mathrm{tail}(v))\end{aligned}$$

determines a value $\mathrm{merge}(w,v)$ *for all strings* $w, v \in L^*$*, and if* w *and* v *are both sorted, then*

(1D-3) $$\mathrm{merge}(w,v) = \mathrm{sort}(w * v).$$

Moreover, the value $\mathrm{merge}(w,v)$ *can be computed by successive applications of* (1D-2)*, using no more than* $|w| + |v| \dot{-} 1$ *comparisons.*

Proof. That (1D-2) determines a function and that (1D-3) holds are both trivial, by induction on $|w| + |v|$. For the comparison counting, notice first that (1D-2) computes $\mathrm{merge}(w,v)$ using no comparisons at all, if one of w or v is nil; if both $|w| > 0$ and $|v| > 0$, we make one initial comparison to decide whether $w_0 \le v_0$, and no more than $|w| + |v| - 2$ additional comparisons after that (by the induction hypothesis, in either case), for a total of $|w| + |v| - 1$. ⊣

We did not define precisely what it means *to compute* merge(w, v) *by successive applications of* (1D-2) (or *from* (1D-2), as we will sometimes say), but the procedure is obvious; for example, when $L = \mathbb{N}$ with the natural ordering:

$$\begin{aligned} \text{merge}((3,1),(2,4)) &= (2) * \text{merge}((3,1),(4)) \\ &= (2,3) * \text{merge}((1),(4)) \\ &= (2,3,1) * \text{merge}((\,),(4)) \\ &= (2,3,1,4). \end{aligned}$$

For each sequence u with $|u| = m > 1$ and $k = \lfloor \frac{m}{2} \rfloor$ the integer part of $\frac{1}{2}|u|$, let:

(1D-4) $$\text{half}_1(u) = (u_0, \dots, u_{k-1}), \quad \text{half}_2(u) = (u_k, \dots, u_{m-1}),$$

and for $|u| \leq 1$, set

(1D-5)
$$\text{half}_1(\text{nil}) = \text{nil},\ \text{half}_2(\text{nil}) = \text{nil},\ \text{half}_1((x)) = \text{nil},\ \text{half}_2((x)) = (x),$$

so that in any case

$$u = \text{half}_1(u) * \text{half}_2(u)$$

and each of the two halves of u has length within 1 of $\frac{1}{2}|u|$.

1D.2. **Proposition.** *The sort function satisfies the equation*

(1D-6) $$\begin{aligned} \text{sort}(u) = \text{if } |u| \leq 1 \text{ then } u \\ \text{else merge}(\text{sort}(\text{half}_1(u)), \text{sort}(\text{half}_2(u))) \end{aligned}$$

and it can be computed from (1D-2) *and* (1D-6) *using no more than* $|u| \log |u|$ *comparisons.*

Proof. The validity of (1D-6) is immediate, by induction on $|u|$. To prove the bound on comparisons, also by induction, note that it is trivial when $|u| \leq 1$, and suppose that $\lceil \log |u| \rceil = k + 1$, so that (easily) both halves of u have length $\leq 2^k$. Thus, by the induction hypothesis and Proposition 1D.1, we can compute sort(u) using no more than

$$k2^k + k2^k + 2^k + 2^k - 1 < (k+1)2^{k+1}$$

comparisons. $\dashv$

The merge-sort is optimal (in a very strong sense) for the number of comparisons required to sort a string, e.g., see Problem x1D.7*.

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.

1D.3. **Lemma.** *The function* $\mathrm{gcd}(x, y)$ *satisfies the following recursive equation, for* $x \geq y \geq 1$*:*

(1D-7) $\quad \mathrm{gcd}(x, y) = \text{if } (\mathrm{rem}(x, y) = 0) \text{ then } y \text{ else } \mathrm{gcd}(y, \mathrm{rem}(x, y)).$

Proof. If $y \nmid x$, then the pairs $\{x, y\}$ and $\{y, \mathrm{rem}(x, y)\}$ have exactly the same common divisors. ⊣

Equation (1D-7) is an example of a *recursive program* from the primitives $\mathrm{rem}, \mathrm{eq}_0$, and it provides a procedure for computing $\mathrm{gcd}(x, y)$:

if $\mathrm{rem}(x, y) = 0$ *give output* y*, else set* $x := y, y := \mathrm{rem}(x, y)$ *and repeat.*

For example:

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

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

$$\begin{aligned}c_{\{\mathrm{rem}\}}(\varepsilon, x, y) = {}&\text{the number of divisions required to compute}\\ &\mathrm{gcd}(x, y) \text{ using (1D-7)} \quad (x \geq y \geq 1),\end{aligned}$$

so that, directly from (1D-7), for $x \geq y \geq 1$,

(1D-8) $\quad c_{\{\mathrm{rem}\}}(\varepsilon, x, y) = \text{ if } (y \mid x) \text{ then } 1 \text{ else } 1 + c_{\{\mathrm{rem}\}}(\varepsilon, y, \mathrm{rem}(x, y)).$

1D.4. **Proposition.** *For all* $x \geq y \geq 2$*,* $c_{\{\mathrm{rem}\}}(x, y) \leq 2 \log y$*.*

Proof is by (complete) induction on y, and we must consider three cases (with $c(x, y) = c_{\{\mathrm{rem}\}}(x, y)$):

Case 1, $y \mid x$; now $c(x, y) = 1 \leq 2 \log y$, 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 $c(x, y) = 2$, and $2 \leq 2 \log y$, as in Case 1.

Case 3, $x = yq_1 + r_1$ and $y = r_1 q_2 + r_2$ 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 $c(x, y) = 3$, and (easily) $3 < 2 \log 3 \leq 2 \log y$. Suppose then that $r_2 \geq 2$, and consider the next division,

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

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

$$y = r_1 q_2 + r_2 \geq r_1 + r_2 > 2r_2,$$

which by the induction hypothesis for $r_2 \geq 2$ gives

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

as required. ⊣

The lower bounds for the complexity measure $c_{\{\text{rem}\}}(\varepsilon, x, y)$ are best expressed in terms of the classical *Fibonacci sequence*, defined by the recursion

(1D-9) $$F_0 = 0,\ F_1 = 1,\ F_{k+2} = F_k + F_{k+1},$$

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

Coprimeness by the Euclidean. In the formal terminology that we will introduce in the next Chapter, the Euclidean is a recursive algorithm of the structure $(\mathbb{N}, \text{rem}, \text{eq}_0)$. If we use it to check the coprimeness relation, we also need to test at the end whether $\gcd(x,y) = 1$, so that as a decision method for coprimeness, the Euclidean is a recursive algorithm of the structure

$$\mathbf{N}_\varepsilon = (\mathbb{N}, \text{rem}, \text{eq}_0, \text{eq}_1).$$

As we mentioned in the Preface, it is not known whether the Euclidean algorithm is optimal (in any natural sense) among algorithms from its primitives, either for computing the gcd or for deciding coprimeness. One of our main aims is to make these questions precise and establish the strongest, known partial results about them.

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

1D.5. **Proposition** (Stein [1967], Knuth [1973], Vol. 2, Sect. 4.5.2). *The gcd satisfies the following recursive equation for $x, y \geq 1$, using which it can be computed in $O(\log x + \log y)$ steps:*

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

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 halving, since the difference of two odd numbers is even. $\dashv$

The structure in which the Stein algorithm lives is clearly

$$\mathbf{N}_{\text{st}} = (\mathbb{N}, \text{parity}, \text{em}_2, \text{iq}_2, \dot{-}, =, <).$$

We will show that the Stein algorithm is weakly optimal for coprimeness from Presburger primitives.

Horner's rule. For any field F, Horner's rule computes the value

$$V_{F,n}(a_0, \dots, a_n, x) = \chi(x) = a_0 + a_1 x + \cdots + a_n x^n \qquad (n \geq 1)$$

of a polynomial $\chi(x)$ of degree n using no more than n multiplications and n additions in F as follows:

$$\begin{aligned}
\chi_0(x) &= a_n,\\
\chi_1(x) &= a_{n-1} + x\chi_0(x) = a_{n-1} + a_n x\\
&\vdots\\
\chi_j(x) &= a_{n-j} + x\chi_{j-1}(x) = a_{n-j} + a_{n-j+1}x + \cdots + a_n x^j\\
&\vdots\\
\chi(x) = \chi_n(x) &= a_0 + x\chi_{n-1}(x) = a_0 + a_1 x + \cdots + a_n x^n.
\end{aligned}$$

This is an example of a simple but important *finite algorithm* from the field primitives of F. It can also be used to decide the (plausibly simpler) *nullity relation* on F,

$$\text{(1D-10)} \qquad N_{F,n}(a_0, \dots, a_n, x) \iff a_0 + a_1 x + \cdots + a_n x^n = 0,$$

from the primitives of the expansion of F by the identity relation

$$\mathbf{F} = (F, 0, 1, +, -, \cdot, \div, =).$$

It is known that Horner's rule is optimal for many fields and inputs, both for the number of multiplications and the number of additions that are needed to compute $V_{F,n}(\vec{a}, x)$ or to decide $N_{F,n}(\vec{a}, x)$, in fact the relevant theorems (from the 1960s) were the first significant lower bounds for natural problems in algebra. We will establish some of them in Chapter 9.

Problems for Section 1D

x1D.1. **Problem.** Prove that if $x > v_0 > v_1 > \cdots > v_{n-1}$, then the computation of $\mathrm{merge}((x), v)$ by (1D-2) will require n comparisons.

In the next two problems we define and analyze a simple algorithm for sorting, which is much less efficient than the merge-sort.

x1D.2. **Problem.** Prove that the equation

$$\text{(1D-11)}\qquad \begin{aligned}\mathrm{insert}(x,u) = {}& \text{if } (|u| = 0) \text{ then } (x)\\ & \text{else } \text{ if } x \le u_0 \text{ then } (x) * u\\ & \text{else } (u_0) * \mathrm{insert}(x, \mathrm{tail}(u))\end{aligned}$$

determines a value $\mathrm{insert}(x,u) \in L^*$ for any $x \in L$ and $u \in L^*$, and if u is sorted, then

$$\text{(1D-12)}\qquad \mathrm{insert}(x,u) = \mathrm{sort}((x) * u).$$

Moreover, $\mathrm{insert}(x,u)$ can be computed from (1D-11) using no more than $|u|$ comparisons.

x1D.3. **Problem** (The insert-sort algorithm)**.** Prove that the sort function satisfies the equation

$$\text{(1D-13)}\qquad \begin{aligned}\mathrm{sort}(u) = {}& \text{if } |u| \le 1 \text{ then } u\\ & \text{else } \mathrm{insert}(u_0, \mathrm{sort}(\mathrm{tail}(u))),\end{aligned}$$

and can be computed from (1D-13) and (1D-11) using no more than $\frac{1}{2}|u|(|u|-1)$ comparisons. Illustrate the computation with some examples, and show also that if u is inversely ordered, then this computation of $\mathrm{sort}(u)$ requires exactly $\frac{1}{2}|u|(|u|-1)$ comparisons.

To see the difference between the merge-sort and the insert-sort, note that when $|u| = 64 = 2^6$, then the insert-sort may need as many as 2016 comparisons, while the merge-sort will need no more than 384. On the other hand, as the next two problems show, there is nothing wrong with the idea of sorting by repeated inserting—it is only that (1D-11) expresses a very inefficient algorithm for insertion.

x1D.4*. **Problem** (Binary insertion)**.** Prove that the equation

$$\begin{aligned}\mathrm{binsert}(x,u) = {}& \text{if } (|u| = 0) \text{ then } (x)\\ & \text{else } \text{ if } (x \le \mathrm{half}_2(u)_0)\\ & \qquad \text{then } \mathrm{binsert}(x, \mathrm{half}_1(u)) * \mathrm{half}_2(u)\\ & \text{else } \mathrm{half}_1(u) * (\mathrm{half}_2(u)_0) * \mathrm{binsert}(x, \mathrm{tail}(\mathrm{half}_2(u)))\end{aligned}$$

determines a value $\mathrm{binsert}(x, u) \in L^*$ for any $x \in L$ and $u \in L^*$, and if u is sorted, then

$$\mathrm{binsert}(x, u) = \mathrm{insert}(x, u) = \mathrm{sort}((x) * u).$$

Moreover, $\mathrm{binsert}(x, u)$ can be computed from (1D-14) using (for $|u| > 0$) no more than $b(|u|)$ comparisons, where

$$b(m) = \begin{cases} \log m + 1, & \text{if } m \text{ is a power of 2,} \\ \lfloor \log m \rfloor, & \text{otherwise.} \end{cases}$$

x1D.5*. **Problem** (Binary-insert-sort)**.** Prove that the sort function satisfies the equation

$$\text{(1D-14)} \qquad \mathrm{sort}(u) = \text{if } |u| \leq 1 \text{ then } u \\ \text{else } \mathrm{binsert}(u_0, \mathrm{sort}(\mathrm{tail}(u))),$$

and can be computed from (1D-14) and (1D-14) using no more than $s(|u|)$ comparisons, where for $m > 0$,

$$\text{(1D-15)} \quad s(m) = \lfloor \log((m-1)!) \rfloor + (m-1) \leq \log((m-1)!) + (m-1).$$

x1D.6*. **Problem.** For the function $s(m)$ defined in (1D-15), prove that

$$\lim_{m\to\infty} \tfrac{s(m)}{\log(m!)} = 1.$$

By Stirling's formula,

$$\lim_{m\to\infty} \tfrac{m \log m}{\log(m!)} = 1,$$

and so the merge-sort and the binary-insert-sort algorithms are asymptotically equally efficient for the required number of comparisons.

In fact, these algorithms are asymptotically worst-case-optimal for this complexity measure among all "deterministic sorting algorithms", in a very strong sense which we will not make precise until later. Here we state only one version of this result for "Turing machines with oracles" which uses and illustrates the idea of this basic argument, one of the standard methods for establishing lower bounds. (This problem naturally requires some knowledge of Turing machines.)

x1D.7*. **Problem** (Lower bound for sorting)**.** Suppose L is a finite set with $n \geq 2$ elements and $\leq$ is a fixed ordering of L. A Turing machine M is a *Turing sorter for* $(L, \leq)$ if

- it has a special *query tape* which it can read and on which it can write;
- an *output tape* on which it can write;
- *query states* $?, a_0, a_1$;

and M acts on an arbitrary $u = (u_0, \dots, u_{n-1}) \in L^n$ so that the following conditions hold:

(1) The computation starts with all tapes empty.
(2) *Consulting the input oracle*: If the computation reaches the state ?, then the query tape has two numbers k_i, k_j on it and the computation moves to state a_0 if $u_{k_i} \le u_{k_j}$ or to state a_1 if $u_{k_i} > u_{k_j}$.
(3) The computation terminates, and when it does, then the output tape supplies a sequence of numbers $k_0, \dots, k_{n-1}$ such that
$$\mathrm{sort}(u_0, \dots, u_{n-1}) = (u_{k_0}, \dots, u_{k_{n-1}}).$$

Other than this, M may have any number of tapes, of any kind (infinite in one or both directions), it may "read" and "write" natural numbers on the query and output tapes in unary, binary (or any unambiguous way) to give meaning to (2) and (3), and it may have additional "oracle tapes" which supply the values of arbitrary functions on $\mathbb{N}$.

Prove that every such Turing sorter will consult the input oracle for at least $\lceil \log(n!) \rceil$ times, for at least one string $u \in L^n$.

Hint: The computation of M for any $u \in L^n$ proceeds deterministically until the first question to the input oracle, after which it may split depending on whether the next state is a_0 or a_1—and then it may split again on the second question, etc. Consider the tree of all computations of M structured in this way and use (1A-5).

We now turn to some problems related to the Euclidean algorithm.

Recall the definition of the Fibonacci sequence $\{F_k\}_k$ in (1D-9).

x1D.8. **Problem.** Show that if $\varphi = \frac{1}{2}(1+\sqrt{5})$ is the positive root of the quadratic equation $x^2 = x + 1$, then for all $k \ge 2$,
$$\varphi^{k-2} \le F_k \le \varphi^k.$$

x1D.9. **Problem.** Show that if $\varphi = \frac{1+\sqrt{5}}{2}$ and $\hat{\varphi} = \frac{1-\sqrt{5}}{2}$ are the two roots of the quadratic equation $x^2 = x + 1$, then for all k,
$$F_k = \frac{\varphi^k - \hat{\varphi}^k}{\sqrt{5}} \ge \frac{\varphi^k}{\sqrt{5}} - 1.$$
Hint: Use induction on k for the equation, and infer the inequality from the fact that $\left|\dfrac{\hat{\varphi}^k}{\sqrt{5}}\right| < 1$.

x1D.10. **Problem.** Show that successive Fibonacci numbers F_k, F_{k+1} with $k \ge 2$ are relatively prime, and $c(\varepsilon, F_{k+1}, F_k) = k - 1$.

x1D.11. **Problem.** Lamé's Lemma. Show that if $y \le F_k$ with $k \ge 2$, then, for every $x \ge y$, $c(\varepsilon, x, y) \le k - 1$. Hint: Use induction on $k \ge 2$, checking separately (by hand) the two basis cases $k = 2, 3$ and imitating the argument in the proof of Proposition 1D.4.

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 (of the sort that we will often "leave for an exercise"):

x1D.12. **Problem.** Show that if $x \geq y \geq 2$, then

$$c(\varepsilon, x, y) \leq \frac{\log(\sqrt{5}y)}{\log \varphi},$$

where φ is the positive root of $x + 1 = x^2$.

x1D.13. **Problem.** Show that for all real numbers $y \geq 16$,

$$\frac{\log(\sqrt{5}y)}{\log \varphi} < 2 \log y.$$

Hint: Check the inequality by hand for $y = 16$, and then check that the function

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

on $\mathbb{R}$ is increasing for $y > 0$.

x1D.14. **Problem** (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 $x \geq y \geq 1$:

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

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

x1D.15. **Problem.** Show that two numbers $x, y \geq 1$ are coprime if and only if there exist integers $\alpha, \beta \in \mathbb{Z}$ such that $1 = \alpha x + \beta y$.

x1D.16. **Problem.** For positive numbers, show: if $x \perp\!\!\!\perp a$ and $x \mid ab$, then $x \mid b$.

x1D.17. **Problem.** 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 $d = \gcd(x, y)$.

x1D.18. **Problem.** Define a (finite) algorithm from the primitives of a field F of characteristic $\neq 2$, which decides the nullity relation (1D-10) using no more than $n - 1$ additions and/or subtractions, and count how many multiplications and/or divisions and equality tests it needs. Hint: Show first that you can test whether $ax + b = 0$ using no additions or subtractions, just multiplications and equality tests.

CHAPTER 2

RECURSIVE (McCARTHY) PROGRAMS

Recursive programs are deterministic versions of the classical *Herbrand-Gödel-Kleene systems of recursive equations*, and they can be used to develop very elegantly the classical theory of *recursive* (computable) *functions* on the natural numbers. Here we will study them on arbitrary, partial structures, and we will use them primarily to introduce some natural and robust notions of complexity for algorithms which compute functions from (relative to) specified primitives.

We will also discuss briefly (finitely) *nondeterministic* recursive programs in Section 2C.

2A. Syntax and semantics

Syntax. A (deterministic) *recursive program* on the vocabulary Φ is a syntactic expression

(2A-1) $$E \equiv E_0(\vec{\mathsf{x}}) \text{ where } \{\mathsf{p}_1(\vec{\mathsf{x}}_1) = E_1(\vec{\mathsf{x}}_1), \dots, \mathsf{p}_K(\vec{\mathsf{x}}_K) = E_K(\vec{\mathsf{x}}_K)\},$$

where $\mathsf{p}_1, \dots, \mathsf{p}_K$ are distinct function symbols; each $E_i(\vec{\mathsf{x}}_i)$ is a pure term in the *program vocabulary*

$$\mathrm{voc}(E) = \Phi \cup \{\mathsf{p}_1, \dots, \mathsf{p}_K\}$$

whose variables are, as usual, in the list $\vec{\mathsf{x}}_i$, with $\vec{\mathsf{x}}_0 \equiv \vec{\mathsf{x}}$; and the arities and sorts of the *recursive variables* $\mathsf{p}_1, \dots, \mathsf{p}_K$ of E are such that the equations in E make sense.

The term E_0 is the *head* of E, and the remaining parts $E_1, \dots, E_K$ comprise the *body* of E. The recursive variables $\mathsf{p}_1, \dots, \mathsf{p}_K$ and the individual variables in the lists $\vec{\mathsf{x}}_i$ for $i = 1, \dots, K$ are *bound* in E, so that its only free variables (if any) are those which occur in the head $E_0(\vec{\mathsf{x}})$. The idea is that $E(\vec{x})$ denotes the value of $E_0(\vec{x})$ when $\mathsf{p}_1, \dots, \mathsf{p}_K$ are interpreted by the canonical (least) solutions of the recursive equations in the body of E.

We allow $K = 0$ in this definition, so that every Φ-term is identified with a Φ-program with empty body,

$$E \equiv E \text{ where } \{\ \}.$$

In general, we think of recursive programs as generalized Φ-terms, we write, as usual, $E(\vec{\mathsf{x}}) = (E, \vec{\mathsf{x}})$ for any list of individual variables $\vec{\mathsf{x}} \equiv \mathsf{x}_1, \dots, \mathsf{x}_n$ which includes all the free variables of E, i.e., those which occur in the head E_0, and we set

$$\text{sort}(E) = \text{sort}(E_0), \quad \text{arity}(E) = n.$$

The *total arity* of E is the maximum of arity(E) and the arities of all the function symbols $\phi \in \Phi$ and the recursive variables of E.

Algorithms are often expressed by a single recursive equation, e.g.,

$$\text{gcd}(x, y) = \text{if } \text{eq}_0(\text{rem}(x, y)) \text{ then } y \text{ else } \text{gcd}(y, \text{rem}(x, y))$$

for the Euclidean, and in this case we need to add a trivial head term to accord with the "official" definition: so the formal recursive program which expresses the Euclidean is

$$\begin{aligned} E_\varepsilon(x, y) \equiv p(x, y) \text{ where}& \\ &\{p(x, y) = \text{if } \text{eq}_0(\text{rem}(x, y)) \text{ then } y \text{ else } p(y, \text{rem}(x, y))\}. \end{aligned}$$

We will assume that this addition of a head term is done when needed.

If $\mathbf{A}$ is a Φ-structure, then Φ-programs are also called $\mathbf{A}$-programs.

All the recursive equations and systems of equations in the problems of Sections 1A and 1D are really recursive programs, just not sufficiently formalized. Problem x1D.14, for example, determines two programs in the structure $(\mathbb{Z}, 0, 1, +, -, \cdot, \text{rem}, \text{iq}, \text{eq}_0)$, one for each of the needed coefficients in Bezout's Lemma. The first of these is

$$\begin{aligned} \alpha(x, y) \text{ where } \Big\{ &\alpha(x, y) = \text{if } \text{eq}_0(\text{rem}(x, y)) \text{ then } 0 \text{ else } \beta(y, \text{rem}(x, y)), \\ &\beta(x, y) = \text{if } \text{eq}_0(\text{rem}(x, y)) \text{ then } 1 \\ &\qquad \text{else } \alpha(y, \text{rem}(x, y)) - \text{iq}(x, y) \cdot \beta(y, \text{rem}(x, y)) \Big\}, \end{aligned}$$

and the second is obtained from this by changing the head to $\beta(x, y)$. Both programs have the binary recursive variables α and β, and the addition symbol is not used by either of them.

Semantics. Fix a recursive program E on the vocabulary Φ and a Φ-structure $\mathbf{A}$. For any pure voc(E)-term $N(\vec{\mathsf{x}}, \mathsf{p}_1, \dots, \mathsf{p}_K)$, let

(2A-2) $$F_N(\vec{x}, p_1, \dots, p_K) = \text{den}(\mathbf{A}, N(\vec{x}, p_1, \dots, p_K)),$$

where the replacement of the recursive variables $\mathsf{p}_1, \dots, \mathsf{p}_K$ by the partial functions $p_1, \dots, p_K$ signifies that we are computing the denotation in the voc(E)-structure

$$(\mathbf{A}, p_1, \dots, p_K) = (A, \{\phi^{\mathbf{A}}\}_{\phi \in \Phi}, p_1, \dots, p_K).$$

2A.1. **Lemma.** *For each* voc(E)*-term* $N(\vec{\mathsf{x}}, \mathsf{p}_1, \dots, \mathsf{p}_K)$*, the functional* F_N *defined by* (2A-2) *is monotone and continuous.*

Proof is very easy, by induction on the term N. For example, if $N \equiv \mathsf{x}$, then $F_N(x, p_1, \dots, p_K) = x$ is independent of its partial function arguments, and so trivially monotone and continuous. If $N \equiv \phi(N_1, \dots, N_n)$ with $\phi \in \Phi$, then

$$F_N(\vec{x}, p_1, \dots, p_K) = \phi^{\mathbf{A}}(F_{N_1}(\vec{x}, p_1, \dots, p_K), \dots, F_{N_n}(\vec{x}, p_1, \dots, p_K)),$$

and the monotonicity and continuity of the F_{N_i} imply the same properties for F_N. The argument for the conditional is similar. Finally, if $N \equiv \mathsf{p}_i(N_1, \dots, N_n)$, then to compute

$$F_N(\vec{x}, p_1, \dots, p_K) = p_i(F_{N_1}(\vec{x}, p_1, \dots, p_K), \dots, F_{N_n}(\vec{x}, p_1, \dots, p_K))$$

we need just one value of p_i in addition to those needed for the computation of the parts $F_{N_i}(\vec{x}, p_1, \dots, p_K)$. ⊣

Let p_E be a fresh recursion variable of arity and sort those of E. It follows by the Fixed Point Lemma 1B.1 that the system of recursive equations

$$\text{(2A-3)} \qquad \begin{cases} p_E(\vec{x}) &= \text{den}(\mathbf{A}, E_0(\vec{x}, p_1, \dots, p_K)), \\ p_1(\vec{x}_1) &= \text{den}(\mathbf{A}, E_1(\vec{x}_1, p_1, \dots, p_K)), \\ &\vdots \\ p_K(\vec{x}_K) &= \text{den}(\mathbf{A}, E_K(\vec{x}_K, p_1, \dots, p_K)) \end{cases}$$

defined by the body of E has a canonical, least solution tuple

$$\overline{p}_E, \overline{p}_1, \dots, \overline{p}_K,$$

and we set

$$\text{(2A-4)} \qquad \overline{p}_E^{\mathbf{A}}(\vec{x}) = \text{den}((\mathbf{A}, \overline{p}_1, \dots, \overline{p}_K), E_0(\vec{x})).$$

This is *the partial function on A computed in $\mathbf{A}$ by E*, and its value at $\vec{x}$ is *the denotation of the program E in $\mathbf{A}$ at $\vec{x}$*,

$$\text{(2A-5)} \qquad \text{den}(\mathbf{A}, E, \vec{x}) = \overline{p}_E^{\mathbf{A}}(\vec{x}).$$

We will also use the notation

$$\text{(2A-6)} \qquad \mathbf{A} \vdash E(\vec{x}) = w \iff \text{den}(\mathbf{A}, E, \vec{x}) = w,$$

where "$\vdash$" is read *proves*.

Notice that if $E \equiv E_0(\vec{\mathsf{x}})$ is a program with no body, i.e., a Φ-term, then

$$\text{den}(\mathbf{A}, E, \vec{x}) = \overline{p}_E^{\mathbf{A}}(\vec{x}) = \text{den}(\mathbf{A}, E_0(\vec{x})),$$

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

2A.1. A-recursive functions. Suppose $\mathbf{A} = (A, \mathbf{\Phi})$ is a structure. A partial function $f : A^n \rightharpoonup A_s$ is $\mathbf{A}$-*recursive* or *recursive in* (or *from*) the primitives $\mathbf{\Phi} = \{\phi^{\mathbf{A}} : \phi \in \Phi\}$ of $\mathbf{A}$ if f is computed by some recursive program in $\mathbf{A}$. We let

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

The classical example is the (total) structure $\mathbf{N}_u = (\mathbb{N}, 0, S, \mathrm{Pd}, \mathrm{eq}_0)$ of unary arithmetic whose recursive partial functions are exactly the Turing computable partial functions. This elegant characterization of Turing computability is due to McCarthy [1963], 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 x2A.3.)

As we do with the propositional connectives and quantifiers of logic, we often use the " where " construct informally, in definitions of the form

$$f(\vec{x}) = f_0(\vec{x}, \vec{p}) \text{ where } \{p_1(\vec{x}_1) = f_1(\vec{x}_1, \vec{p}), \dots, p_K(\vec{x}_1) = f_K(\vec{x}_K, \vec{p})\}$$

when the functionals $f_i(\vec{x}_i, \vec{p})$ are monotone and continuous (and the arities and sorts match in the obvious way); a partial function $f : A^n \rightharpoonup A_s$ defined this way is $\mathbf{A}$-recursive if the f_i's are explicitly defined in $\mathbf{A}$—or even in expansions $(\mathbf{A}, \mathbf{\Phi})$ by $\mathbf{A}$-recursive functions by Problem x2A.2. We can also give a single recursive equation which determines a function: for example, arithmetic subtraction is $\mathbf{N}_u$-recursive because it satisfies the equation

$$x \dot{-} y = \text{if } (y = 0) \text{ then } x \text{ else } \mathrm{Pd}(x \dot{-} \mathrm{Pd}(y)), \tag{2A-7}$$

see Problem x2A.1.

The study of $\mathbf{rec}(\mathbf{A})$ for various structures is generally known as (elementary or first-order) *abstract recursion theory* in logic, and by various other names in theoretical computer science, where many questions about it arise naturally. It is an interesting subject, but not centrally related to our concerns here, and so we will confine ourselves to the next two remarks and a few relevant results in the problems.

2A.2. Tail recursion. A partial function $g : A^k \rightharpoonup A_s$ is defined by *tail recursion* from test : $A^k \to \{\mathrm{tt}, \mathrm{ff}\}$, output : $A^k \to A_s$ and $\sigma : A^k \rightharpoonup A^k$ if it is the least partial function which satisfies the equation

$$g(\vec{u}) = \text{if } \mathrm{test}(\vec{u}) \text{ then } \mathrm{output}(\vec{u}) \text{ else } g(\sigma(\vec{u})). \tag{2A-8}$$

Most often the *transition function* σ is total, as in the characteristic example of the $\gcd(x, y)$ in (1D-7) where

$$\mathrm{test}(x, y) = \mathrm{eq}_0(\mathrm{rem}(x, y)),\ \mathrm{output}(x, y) = y,\ \sigma(x, y), = (y, \mathrm{rem}(x, y)),$$

but it is useful, in general, to allow σ to be partial.

A partial function $f : A^n \rightharpoonup A_s$ is *tail recursive* in $\mathbf{A}$ if

$$f(\vec{x}) = g(h_1(\vec{x}), \dots , h_m(\vec{x})),$$

where $h_1, \dots , h_m$ are explicit in $\mathbf{A}$ and g is defined by a tail recursion (2A-8) with explicit test, output and σ, i.e.,

$$\begin{aligned}&f(\vec{x}) = p(h_1(\vec{x}), \dots , h_m(\vec{x}))\\ &\qquad \mathsf{where}\ \{p(\vec{u}) = \text{if test}(\vec{u}) \text{ then output}(\vec{u}) \text{ else } p(\sigma_1(\vec{u}), \dots , \sigma_k(\vec{u}))\}\end{aligned}$$

with all the (partial) functions on the right explicit in $\mathbf{A}$. We set

$$\mathbf{tailrec}(\mathbf{A}) = \text{the } \mathbf{A}\text{-tail recursive partial functions.}$$

It can be shown that for every expansion $(\mathbf{N}_u, \mathbf{\Phi})$ of the unary numbers by total functions,

$$\mathbf{rec}(\mathbf{N}_u, \mathbf{\Phi}) = \mathbf{tailrec}(\mathbf{N}_u, \mathbf{\Phi}), \tag{2A-9}$$

This is a basic result about recursion in $\mathbf{N}_u$ which, in fact, holds for many other "rich" structures. At the same time, there are interesting examples of total structures in which tail recursion does not exhaust all recursive functions, cf. Stolboushkin and Taitslin [1983], Tiuryn [1989]. This is a rather difficult result which we will skip,[6] as we will also skip most of the theory of $\mathbf{tailrec}(\mathbf{A})$. However, the relation between $\mathbf{rec}(\mathbf{A})$ and $\mathbf{tailrec}(\mathbf{A})$ for arbitrary $\mathbf{A}$ is important and not very well understood.

2A.3. Simple fixed points. One interesting aspect of recursive programs is the use of *systems* rather than single recursive equations. A partial function $f : A^n \rightharpoonup A$ is a *simple fixed point* of $\mathbf{A}$ if it is the least solution of an equation

$$\mathsf{p}(\vec{x}) = \text{den}(\mathbf{A}, E(\vec{x}, \mathsf{p})) \tag{2A-10}$$

for some (pure) (Φ, p)-term E. Addition, for example, is a simple fixed point of $\mathbf{N}_u$ because it is the least (unique) solution of the recursive equation

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

One might think that every $\mathbf{A}$-recursive function $f : A^n \rightharpoonup A$ is a simple fixed point, and this is almost true, cf. Problems x2A.16, x2A.17*. But it is not exactly true, even in $\mathbf{N}_u$:

2A.2. **Proposition** (Moschovakis [1984]). *If $\mathbf{A} = (\mathbf{N}_u, \mathbf{\Phi})$ is an expansion of the unary numbers with any set $\mathbf{\Phi} = (\phi_1, \dots , \phi_k)$ of total, Turing computable functions, then there exists a total function $f : \mathbb{N} \to \mathbb{N}$ which is Turing computable* (and hence $\mathbf{A}$-recursive) *but is not a simple fixed point of $\mathbf{A}$.*

[6]Tiuryn's example is defined in Problem x2A.15.

A proof of this is outlined in problems x2A.18 - x2A.20.

McColm [1989] 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 in a natural way the *simple fixed points* of a structure $\mathbf{A}$ is largely open.

Problems for Section 2A

x2A.1. **Problem.** Prove that arithmetic subtraction $x \dot{-} y$ is $\mathbf{N}_u$-recursive, by verifying and (for once) formalizing (2A-7).

x2A.2. **Problem** (Transitivity)**.** Show that if f is $\mathbf{A}$-recursive and g is $(\mathbf{A}, f)$-recursive, then g is $\mathbf{A}$-recursive. It follows that if $(\mathbf{A}, \mathbf{\Psi})$ is an expansion of $\mathbf{A}$ by partial functions which are $\mathbf{A}$-recursive, then

$$\mathbf{rec}(\mathbf{A}, \mathbf{\Psi}) = \mathbf{rec}(\mathbf{A}).$$

This means that to show that a certain $f : A^n \rightharpoonup A_s$ is $\mathbf{A}$-recursive, it is enough to find a recursive program which computes it using any partial functions already known to be $\mathbf{A}$-recursive.

x2A.3. **Problem.** Show that $\mathbf{rec}(\mathbf{N}_u) = \mathbf{rec}(\mathbf{N}_b) = \mathbf{rec}(\mathbb{N}, 0, S, =)$.

x2A.4. **Problem** (Composition)**.** Prove that if $h, g_1, \dots, g_n$ are $\mathbf{A}$-recursive (of the appropriate arities) and

$$f(\vec{x}) = h(g_1(\vec{x}), \dots, g_n(\vec{x})),$$

then f is also $\mathbf{A}$-recursive.

In the next few problems we consider the special case of recursion in the structure $\mathbf{N}_u$ of unary numbers and its expansions by total functions. These structures are not our main concern, but they are important because they provided the context in which most (nearly all) results about recursion were first discovered. Moreover, some of the complexity results we will study further on have interesting applications to $(\mathbf{N}_u, \mathbf{\Psi})$-recursion which were missed in the classical theory.

x2A.5. **Problem** (Primitive recursion)**.** Suppose $\mathbf{A} = (\mathbf{N}_u, \Phi)$ is an expansion of $\mathbf{N}_u$, $g : \mathbb{N}^n \rightharpoonup \mathbb{N}$ and $h : \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$ are $\mathbf{A}$-recursive, and $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ satisfies the following two equations:

(2A-11)
$$\begin{aligned} f(0, \vec{x}) &= g(\vec{x}), \\ f(y+1, \vec{x}) &= h(f(y, \vec{x}), y, \vec{x}). \end{aligned}$$

Prove that f is $\mathbf{A}$-recursive. Verify also that $f(y, \vec{x})\downarrow \implies (\forall i < y)[f(i, \vec{x})\downarrow]$.

The class of *primitive recursive functions* on $\mathbb{N}$ is the smallest class of (total) functions on $\mathbb{N}$ which contains the successor S and the n-ary constant function $C_0^n(\vec{x}) = 0$ and projection functions $P_i^n(\vec{x}) = x_i$ for each n, and which is closed under composition and primitive recursion. A relation $R : \mathbb{N}^n \to \{\mathrm{tt}, \mathrm{ff}\}$ is primitive recursive if the corresponding function

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

is primitive recursive.

The last two problems imply that every primitive recursive function $f : \mathbb{N}^n \to \mathbb{N}_s$ is $\mathbf{N}_u$-recursive.

x2A.6. **Problem** (Minimalization)**.** Prove that if $g : \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ is recursive in some expansion $\mathbf{A} = (\mathbf{N}_u, \Psi)$ of $\mathbf{N}_u$, then so is the partial function

$$\begin{aligned} f(\vec{x}) &= \mu y[g(y, \vec{x}) = 0] \\ &= \text{ the least } y \text{ such that } (\forall i < y)(\exists w)[g(i, \vec{x}) = w + 1 \ \& \ g(y, \vec{x}) = 0]. \end{aligned}$$

x2A.7. **Problem** (McCarthy [1963])**.** Prove that $\mathbf{rec}(\mathbf{N}_u)$ comprises the class of Turing computable partial functions on $\mathbb{N}$.

Hint: This (obviously) requires some knowledge of Turing computability. In one direction, use the

Kleene Normal Form Theorem: *Every Turing computable $f : \mathbb{N}^n \rightharpoonup \mathbb{N}$ satisfies an equation of the form*

$$f(\vec{x}) = \varphi_e(\vec{x}) = U(\mu y T_n(e, \vec{x}, y)) \quad (\vec{x} \in \mathbb{N}^n) \tag{2A-12}$$

with some e and fixed primitive recursive $T_n : \mathbb{N}^{n+1} \to \{\mathrm{tt}, \mathrm{ff}\}$, $U : \mathbb{N} \to \mathbb{N}$.

For the other direction, appeal to the proof of the Fixed Point Lemma 1B.1: show that for every system of recursive equations as in (2A-3), there is a total recursive function $u(m, k)$ such that

$$\overline{p}_m^k = \varphi_{u(m,k)},$$

where φ_e is the recursive partial function with code (Gödel number) e, and then take (recursively) the union of these partial functions to get the required least fixed points.

x2A.8*. **Problem** (Rózsa Péter)**.** A function $f : \mathbb{N}^{n+1} \to \mathbb{N}$ is defined by *nested recursion* from g, h and $\tau_1, \ldots, \tau_n$ if it satisfies the following equations:

$$\begin{aligned} f(0, \vec{y}) &= g(\vec{y}), \\ f(x + 1, \vec{y}) &= h(f(x, \tau_1(x, y), \ldots, \tau_n(x, \vec{y})), x, \vec{y}). \end{aligned} \tag{2A-13}$$

(1) Prove that if f is defined by nested recursion from $(\mathbf{N}_u, \mathbf{\Phi})$-recursive functions, then it is $(\mathbf{N}_u, \mathbf{\Phi})$-recursive.

(2) Prove that if f is defined from primitive recursive functions by nested recursion, then it is primitive recursive.

Primitive recursive functions have been studied extensively. The *Grzegorczyk hierarchy* ramifies them by the number of primitive recursions required for their definition, and its extension into the (constructive) transfinite has very important applications to proof theory. At the bottom end, the *elementary functions* in the first three levels of the Grzegorczyk hierarchy also have important applications to "intermediate" (exponential and doubly exponential) complexity theory.

x2A.9. **Problem** (Double recursion). A function $f : \mathbb{N}^{2+n} \to \mathbb{N}$ is defined by *double recursion* from g, h_1, σ, h_2 if it satisfies the following equations for all $x, y, \vec{z}$:

$$\begin{aligned} f(0, y, \vec{z}) &= g(y, \vec{z}), \\ f(x + 1, 0, \vec{z}) &= h_1(f(x, \sigma(x, \vec{z}), \vec{z}), x, \vec{z}), \\ f(x + 1, y + 1, \vec{z}) &= h_2(f(x + 1, y, \vec{z}), x, y, \vec{z}). \end{aligned} \tag{2A-14}$$

Prove that if f is defined by double recursion from $(\mathbf{N}_u, \mathbf{\Psi})$-recursive functions, then it is $(\mathbf{N}_u, \mathbf{\Psi})$-recursive.

x2A.10*. **Problem** (The Ackermann function). Consider the system of equations

$$\begin{aligned} A(0, x) &= x + 1 \\ A(n + 1, 0) &= A(n, 1) \\ A(n + 1, x + 1) &= A(n, A(n + 1, x)), \end{aligned} \tag{2A-15}$$

on a function $A : \mathbb{N}^2 \to \mathbb{N}$.

(1) Verify that this defines A by double recursion.

(2) Prove that the Ackermann function is not primitive recursive.

Hint: For (2), prove first that every *Ackermann section*

$$A_n(x) = A(n, x)$$

is primitive recursive and then show that *for every primitive recursive function $f(\vec{x})$ there is some m such that*

$$f(\vec{x}) < A_m(\max \vec{x}) \quad (\vec{x} \in \mathbb{N}^n). \tag{2A-16}$$

This requires establishing some basic inequalities about these functions, including

$$\begin{gathered} A_n(x) \geq 1, \quad x < y \Longrightarrow A_n(x) < A_n(y), \\ n < m \Longrightarrow A_n(x) < A_m(x), \quad A_n(A_n(x)) < A_{n+2}(x) \end{gathered}$$

which are also needed for the punchline—that $A(n, x)$ is not primitive recursive.

The function $A(n, x)$ is sometimes called *the Ackermann-Péter function.* It is a modification of the original example of a doubly recursive function defined by Ackermann in the 1930s and it is due to Rózsa Péter, who introduced and studied the more general *n-fold recursive definitions* in her classic Péter [1951].[7]

These problems about recursion in total expansions of $\mathbf{N}_u$ suggest that an interesting theory of *recursive complexity* can be developed for $\mathbf{rec}(\mathbf{A})$, at least for these structures: a function $f : A^n \to A_s$ is "recursively less complex" than $g : A^m \to A_s$ if f can be defined by (syntactically) "simpler" recursions than the recursions needed to define g. One would think that hierarchies constructed along these ideas must be related to our more direct intuitions about *computational complexity* and, of course, they are. We will not pursue these ideas in any detail, but we will study in the sequel some more direct connections between recursive and computational complexity.

x2A.11. **Problem.** Prove that the following functions on L^* (from (1A-4), (1D-4) and (1D-5)) are recursive in the string structure $\mathbf{L}^*$ defined in (1C-5):

$$u * v, \quad \mathrm{half}_1(u), \quad \mathrm{half}_2(u),$$

and $u \mapsto (u_i)$, set to nil if $i \geq |u|$. Infer that for every ordering $\leq$ of L, the functions $\mathrm{merge}(u, v)$ and $\mathrm{sort}(u)$ are $(\mathbf{L}^*, \leq)$-recursive.

x2A.12. **Problem.** Prove that if $f : A^n \rightharpoonup A$ is $\mathbf{A}$-recursive, then

$$f(\vec{x}) = w \Longrightarrow w \in G_\infty(\mathbf{A}, \vec{x}).$$

Infer that $S \notin \mathbf{rec}(\mathbb{N}, 0, \mathrm{Pd}, \mathrm{eq}_0)$.

It is also true that $\mathrm{Pd} \notin \mathbf{rec}(\mathbb{N}, 0, S, \mathrm{eq}_0)$, but (perhaps) this is not so immediate at this point, see Problem x2B.7. However:

x2A.13. **Problem.** Prove that $\mathbf{rec}(\mathbf{N}_u) = \mathbf{rec}(\mathbb{N}, 0, =, S)$.

x2A.14. **Problem.** Let $\mathbf{A} = (A, 0, \cdot, \mathrm{eq}_0)$ be a structure with $\cdot$ binary, and define x^n for $x \in A$ and $n \geq 1$ as usual,

$$x^1 = x, \ x^{n+1} = x \cdot x^n.$$

Define $f : A^2 \rightharpoonup A$ by

$$f(x, y) = x^k \text{ where } k = \text{ the least } n \text{ such that } y^n = 0,$$

and prove that f is $\mathbf{A}$-recursive.

x2A.15. **Problem.** Let

$$T = \{(n, v_0, \dots, v_{m-1}) : n \in \mathbb{N} \ \& \ m \leq n \ \& \ (\forall i < m)[v_i \leq 1]\},$$

[7]References to the original papers in this work can be found in Kleene [1952].

and consider the structure $\mathbf{T} = (T, l, r, =)$ where

$$l(u) = \text{if length}(u) \le u_0 \text{ then } u * (0) \text{ else } u,$$
$$r(u) = \text{if length}(u) \le u_0 \text{ then } u * (1) \text{ else } u.$$

Prove that the relation

(2A-17) $\quad B(u, v) \iff (\exists w)[u * w = v] \quad (v \text{ is equal to or below } u)$

is recursive in $\mathbf{T}$.

Tiuryn [1989] proves that the relation $B(u, v)$ is not tail recursive in $\mathbf{T}$.

Next we consider the connection between $\mathbf{A}$-recursive partial functions and the fixed points of $\mathbf{A}$.

An element $a \in A$ in the universe of a structure $\mathbf{A}$ is *strongly explicit* if both the constant a and the equality-with-a relation eq_a are $\mathbf{A}$-explicit. For example, 0 and 1 are strongly explicit in $\mathbf{N}_u$ and $\mathbf{N}_b$.

x2A.16. **Problem.** Suppose $\mathbf{A}$ is a structure with two strongly explicit points a, b. Prove that a partial function $f : A^n \rightharpoonup A_s$ is $\mathbf{A}$-recursive if and only if there is a simple fixed point $g : A^{m+n} \rightharpoonup A_s$ of $\mathbf{A}$ such that

(2A-18) $\quad f(\vec{x}) = g(\vec{a}, \vec{x}) \quad (\vec{x} \in A^n, \vec{a} = \underbrace{a, \dots, a}_{m \text{ times}}).$

When this equation holds, we say that f is a *section of* g by explicit constants.

x2A.17*. **Problem.** For any Φ-structure $\mathbf{A}$, let

$$\mathbf{A}[a, b] = (A \cup \{a, b\}, \mathbf{\Phi}^{a,b}, a, b, \text{eq}_a, \text{eq}_b)$$

where a, b are distinct objects not in A and for each $\phi \in \Phi$, $\phi^{a,b}$ is the extension of $\phi^{\mathbf{A}}$ to $A \cup \{a, b\}$ set equal to a or tt (depending on the sort of ϕ) when any one of its arguments is a or b. Prove that for every $f : A^n \rightharpoonup A_s$,

(2A-19) $\quad f \in \mathbf{rec}(\mathbf{A}) \iff f \in \mathbf{rec}(\mathbf{A}[a, b]).$

These two problems together say that every $\mathbf{A}$-recursive partial function is a section of a fixed point, except that to realize this, we may have to add two strongly explicit points to $\mathbf{A}$. The remaining problems in this section lead to a proof of Proposition 2A.2.

x2A.18. **Problem.** Suppose $F(x, p)$ is a monotone, continuous functional whose fixed point $\overline{p} : \mathbb{N} \to \mathbb{N}$ is a total, unary function, and let

(2A-20) $\quad \text{stage}(x) = \text{stage}_F(x) = \text{the least } k \text{ such that } \overline{p}^k(x)\downarrow\ -1$

in the notation of Lemma 1B.1. Show that for infinitely many x,

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

x2A.19. **Problem.** Suppose $\psi : \mathbb{N} \to \mathbb{N}$ is strictly increasing, i.e.,

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

and let by recursion,

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

A unary partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is *n-bounded* (relative to ψ, for $n > 0$) if

$$f(x)\downarrow \Longrightarrow f(x) \le \psi^{(n)}(x);$$

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

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

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

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

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

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

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

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

(This is also needed in the next problem.)

x2A.20. **Problem.** Prove Proposition 2A.2.

x2A.21. **Problem** (Open)**.** Prove that if $\mathbf{A} = (\mathbf{N}_u, \mathbf{\Phi})$ is an expansion of the unary numbers with any set $\mathbf{\Phi} = (\phi_1, \dots, \phi_k)$ of total, Turing computable functions, then there exists a total relation $R : \mathbb{N} \to \{\mathrm{tt}, \mathrm{ff}\}$ which is recursive in $\mathbf{A}$ but not a simple fixed point of $\mathbf{A}$.

2B. Deterministic models of computation

All the standard deterministic models of computation for partial functions $f : X \rightharpoonup W$ on one set to another are captured by the following, well-known, general notion:

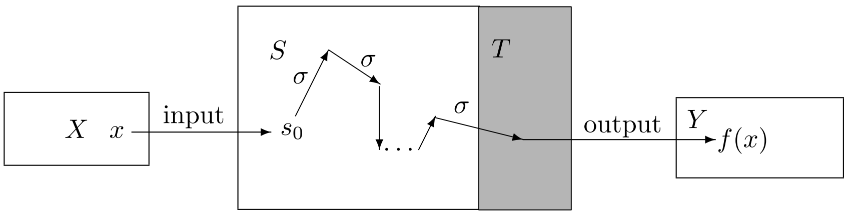


Figure 1. Iterator computing $f : X \rightharpoonup W$.

2B.1. Iterators. For any two sets X and W, a (partial) *iterator* (or *sequential machine*)

$$\mathfrak{i} : X \rightsquigarrow W$$

is a quintuple (input, S, σ, T, output), satisfying the following conditions:

(I1) S is an arbitrary (non-empty) set, the *set of states* of $\mathfrak{i}$;
(I2) input : $X \to S$ is the *input function* of $\mathfrak{i}$;
(I3) $\sigma : S \rightharpoonup S$ is the *transition function* of $\mathfrak{i}$;
(I4) $T \subseteq S$ is the set of *terminal states* of $\mathfrak{i}$, and $s \in T \Longrightarrow \sigma(s) = s$;
(I5) output : $T \to W$ is the *output function* of $\mathfrak{i}$.

The iterator $\mathfrak{i}$ is *total* if its transition function $\sigma : S \to S$ is total, which is the most usual and useful case.

A *partial computation* of $\mathfrak{i}$ is any finite sequence $(s_0, \dots, s_n)$ such that for all $i < n$, s_i is not terminal and $\sigma(s_i) = s_{i+1}$, and it is *convergent* if, in addition, $s_n \in T$. Note that (with $n = 0$), this includes every one-term sequence (s), and (s) is convergent if $s \in T$. We write

(2B-1) $\quad s \to_{\mathfrak{i}}^* s'$ there is a convergent computation with $s_0 = s, s_n = s'$,

and we say that $\mathfrak{i}$ *computes* a partial function $f : X \rightharpoonup W$ if

(2B-2) $\quad f(x) = w \iff (\exists s \in T)[\text{input}(x) \to_{\mathfrak{i}}^* s \ \&\ \text{output}(s) = w].$

It is clear that there is at most one convergent computation starting from any state s_0, and so exactly one partial function $\bar{\mathfrak{i}} : X \rightharpoonup W$ computed by $\mathfrak{i}$. *The computation* of $\mathfrak{i}$ on x is the finite sequence

(2B-3) $\quad \text{Comp}_{\mathfrak{i}}(x) = (\text{input}(x), s_1, \dots, s_n, \text{output}(s_n)) \quad (x \in X, \bar{\mathfrak{i}}(x)\downarrow),$

such that $(\text{input}(x), s_1, \dots, s_n)$ is a convergent computation, and its length

(2B-4) $$\text{Time}_{\mathfrak{i}}(x) = n + 2$$

is the natural *time complexity* of $\mathfrak{i}$.

There is little structure to this definition of course, and the important properties of specific computation models derive from the judicious choice of the set of states and the transition function, but also the input and output functions. The first two depend on what operations (on various data structures) are assumed as given (primitive) and regulate how the iterator calls them, while the input and output functions often involve *representing* the members of X and W in some specific way, taking for example numbers in unary or binary notation if $X = W = \mathbb{N}$. We will not study computation models in this version of the notes, beyond the few simple facts in the remainder of this section which relate them to recursion.

Fix an iterator $\mathfrak{i}$, let

$$A_{\mathfrak{i}} = X \uplus W \uplus S$$

be the *disjoint union* of the indicated sets. We identify, as usual, each set $Z \subseteq A_{\mathfrak{i}}$ with its characteristic function $Z : A_{\mathfrak{i}} \to \{\mathrm{tt}, \mathrm{ff}\}$. on the universe $A_{\mathfrak{i}}$, and we set

$$\mathbf{A}_{\mathfrak{i}} = (A_{\mathfrak{i}}, X, W, S, \mathrm{input}, \sigma, T, \mathrm{output}), \tag{2B-5}$$

where $\mathrm{input}, \sigma, \mathrm{output}$ are now total extensions of the functions of the iterator which just return their argument when it is not in the appropriate domain. This is the *structure of* $\mathfrak{i}$, it is a total structure if $\mathfrak{i}$ is total, and the *tail recursive program associated with* $\mathfrak{i}$ is

(2B-6) $E_{\mathfrak{i}}(\mathsf{x}) \equiv \mathsf{q}(\mathrm{input}(\mathsf{x}))$ where

$$\{\mathsf{q}(\mathsf{s}) = \text{if } T(\mathsf{s}) \text{ then output}(\mathsf{s}) \text{ else } \mathsf{q}(\sigma(\mathsf{s}))\}.$$

2B.1. **Proposition.** *For each iterator* $\mathfrak{i}$ *and the associated recursive program* $E \equiv E_{\mathfrak{i}}$ *on* $\mathbf{A}_{\mathfrak{i}}$*, and for all* $x \in X$*,*

$$\bar{\mathfrak{i}}(x) = \bar{p}_E^{\mathbf{A}_{\mathfrak{i}}}(x).$$

In particular, the partial function computed by an iterator $\mathfrak{i} : X \rightsquigarrow W$ *is tail recursive in the associated structure* $\mathbf{A}_{\mathfrak{i}}$*.*

Proof. Let $\overline{q}$ be the least fixed point of the equation in the body of $E_{\mathfrak{i}}$, and define $\widetilde{q} : S \rightharpoonup W$ by

$$\widetilde{q}(s) = w \iff (\exists s' \in T)[s \to_{\mathfrak{i}}^* s' \ \&\ \mathrm{output}(s') = w].$$

It is clear that $\widetilde{q}$ satisfies the recursive equation for q in $E_{\mathfrak{i}}$, and so

$$\overline{q} \sqsubseteq \widetilde{q}.$$

For the converse inclusion, we show by induction on n that

$$\text{if } [s \to s_1 \to \cdots \to s_n \in T], \text{ then } \mathrm{output}(s_n) = \overline{q}(s).$$

This is trivial at the base $n = 0$. At the induction step, the hypothesis

$$s \to s_1 \to \cdots \to s_{n+1} \in T$$

gives $\text{output}(s_{n+1}) = \overline{q}(s_1)$, by the induction hypothesis; but $s_1 = \sigma(s)$, and so

$$\overline{q}(s) = \overline{q}(\sigma s) = \overline{q}(s_1) = \text{output}(s_{n+1})$$

as required. This establishes that $\overline{q} = \widetilde{g}$, and so

$$\bar{\mathfrak{i}}(x) = \widetilde{q}(\text{input}(x)) = \overline{q}(\text{input}(x)) = \overline{p}_E(x)$$

as required. ⊣

This simple Proposition is foundationally significant, because it reduces *computability*, as it is captured by the notion of sequential machines to *recursiveness*, in fact tail recursiveness relative to the data structures and primitives of any specific model of computation. Next we prove a strong converse, which reduces recursiveness to computability: for every $\mathbf{A}$ and every $\mathbf{A}$-program E, the partial function $\overline{p}_E^{\mathbf{A}}$ computed in $\mathbf{A}$ by E can also be computed by an iterator $\mathfrak{i}(\mathbf{A}, E)$ constructed from $\mathbf{A}$ and E. The construction yields a total iterator when $\mathbf{A}$ is total.

2B.2. The recursive machine. Fix a Φ-structure $\mathbf{A}$ and a Φ-program E as in (2A-1).

An $(\mathbf{A}, E)$-*term* is a closed term

$$M \equiv N(x_1, \dots, x_m), \tag{2B-7}$$

where $N(\mathsf{x}_1, \dots, \mathsf{x}_m)$ is a subterm of one of the terms $E_i(\vec{\mathsf{x}}_i)$ of the recursive program E and $x_1, \dots, x_m \in A$. These are closed, voc(E)-terms with parameters from A, but not all such: the $(\mathbf{A}, E)$-terms are constructed by substituting parameters from A into the *finitely many* subterms of E.

The states of $\mathfrak{i} = \mathfrak{i}(\mathbf{A}, E)$ are all finite sequences s of the form

$$a_0 \ \cdots \ a_{m-1} : b_0 \ \cdots \ b_{n-1}$$

where the elements $a_0, \dots, a_{m_1}, b_0, \dots, b_{n-1}$ of s satisfy the following conditions:

- Each a_i is a function symbol in Φ, or one of $\mathsf{p}_1, \dots, \mathsf{p}_K$, or the special symbol ?, or an $(\mathbf{A}, E)$-term, and
- each b_j is a parameter from A or a truth value, i.e., $b_j \in A \cup \{\mathrm{tt}, \mathrm{ff}\}$.

The special *separator symbol* ':' has exactly one occurrence in each state, and the sequences $\vec{a}, \vec{b}$ are allowed to be empty, so that the following sequences are states (with $x \in A \cup \{\mathrm{tt}, \mathrm{ff}\}$):

$$x : \qquad : x \qquad :$$

The *terminal states* of $\mathfrak{i}$ are the sequences of the form

$$: w$$

(pass)	$\vec{a}\ \underline{x :}\ \vec{b} \ \to \vec{a}\ \underline{: x}\ \vec{b} \quad (x \in A)$
(e-call)	$\vec{a}\ \underline{\phi_i : \vec{x}}\ \vec{b} \ \to \vec{a}\ \underline{: \phi_i^{\mathbf{A}}(\vec{x})}\ \vec{b}$
(i-call)	$\vec{a}\ \underline{\mathsf{p}_i : \vec{x}}\ \vec{b} \ \to \vec{a}\ \underline{E_i(\vec{x}, \vec{\mathsf{p}}) :}\ \vec{b}$
(comp)	$\vec{a}\ \underline{h(F_1, \dots, F_n) :}\ \vec{b} \ \to \vec{a}\ \underline{h\ F_1\ \cdots\ F_n :}\ \vec{b}$
(br)	$\vec{a}\ \underline{\textit{if}\ F\ \textit{then}\ G\ \textit{else}\ H :}\ \vec{b} \ \to \vec{a}\ \underline{G\ H\ ?\ F :}\ \vec{b}$
(br0)	$\vec{a}\ \underline{G\ H\ ? : \mathrm{tt}}\ \vec{b} \ \to \vec{a}\ \underline{G :}\ \vec{b}$
(br1)	$\vec{a}\ \underline{G\ H\ ? : \mathrm{ff}}\ \vec{b} \ \to \vec{a}\ \underline{H :}\ \vec{b}$

- The underlined words are those which trigger a transition and are changed by it.
- In (pass), $x \in A \cup \{\mathrm{tt}, \mathrm{ff}\}$.
- In the *external call* (e-call), $\vec{x} = x_1, \dots, x_n$, $\phi_i \in \Phi$, and $\mathrm{arity}(\phi_i) = n$.
- In the *internal call* (i-call), p_i is an n-ary recursive variable of E defined by the equation $\mathsf{p}_i(\vec{\mathsf{x}}) = E_i(\vec{\mathsf{x}}, \vec{\mathsf{p}})$.
- In the *composition transition* (comp), h is a (constant or variable) function symbol in $\mathrm{voc}(E)$ with $\mathrm{arity}(h) = n$.

Table 1. Transition Table for the recursive machine $\mathfrak{i}(\mathbf{A}, E)$.

i.e., those with no elements on the left of ':' and just one constant on the right; and the *output* function of $\mathfrak{i}$ simply reads this constant w, i.e.,

$$\mathrm{output}(\ : w) = w.$$

The states, the terminal states and the output function of $\mathfrak{i}$ depend only on Φ, A and the recursive variables which occur in E. The input function of $\mathfrak{i}$ depends also on the head term $E_0(\vec{\mathsf{x}})$ of E,

$$\mathrm{input}(\vec{x}) \equiv E_0(\vec{x}) :$$

The transition function of $\mathfrak{i}$ is defined by the seven cases in the Transition Table 1, i.e.,

$$\sigma(s) = \begin{cases} s', & \text{if } s \to s' \text{ is a special case of some line in Table 1,} \\ s, & \text{otherwise,} \end{cases}$$

and it is a partial function, because for a given s (clearly) at most one transition $s \to s'$ is *activated* by s. Notice that only the external calls depend on the structure $\mathbf{A}$, and only the internal calls depend on the program E—and so, in particular, all programs with the same body share the same transition function.

An illustration of how these machines compute is given in Figure 2.

The next result is a trivial but very useful observation:

2B.2. **Lemma** (Transition locality). *If $s_0, s_1, \ldots, s_n$ is a partial computation of $\mathfrak{i}(\mathbf{A}, E)$ and $\vec{a}^*, \vec{b}^*$ are such that the sequence $\vec{a}^*\, s_0\, \vec{b}^*$ is a state, then the sequence*

$$\vec{a}^*\, s_0\, \vec{b}^*, \vec{a}^*\, s_1\, \vec{b}^*, \ldots, \vec{a}^*\, s_n\, \vec{b}^*$$

is also a partial computation of $\mathfrak{i}(\mathbf{A}, E)$.

2B.3. **Theorem** (Implementation correctness). *Suppose $\mathbf{A}$ is a Φ-structure, E is a Φ-program with recursive variables $\mathsf{p}_1, \ldots, \mathsf{p}_K$, $\overline{p}_1, \ldots, \overline{p}_K$ are the mutual fixed points of E in $\mathbf{A}$, and $M(\mathsf{p}_1, \ldots, \mathsf{p}_K)$ is a closed $(\mathbf{A}, E)$-term. Then for every $w \in A \cup \{\mathrm{tt}, \mathrm{ff}\}$,*

$$\text{(2B-8)}\quad \operatorname{den}(\mathbf{A}, M(\overline{p}_1, \ldots, \overline{p}_K)) = w \iff M(\mathsf{p}_1, \ldots, \mathsf{p}_K) :\ \to^*_{\mathfrak{i}(\mathbf{A},E)}\ : w.$$

In particular, with $M \equiv E_0(\vec{x}, \vec{\mathsf{p}})$

$$\operatorname{den}(\mathbf{A}, E, \vec{x}) = w \iff E_0(\vec{x}, \vec{\mathsf{p}}) \to^*_{\mathfrak{i}(\mathbf{A},E)}\ : w,$$

and so the recursive machine $\mathfrak{i}(\mathbf{A}, E)$ and the program E compute the same partial function in $\mathbf{A}$.

Outline of proof. First we define the partial functions computed by $\mathfrak{i}(\mathbf{A}, E)$ in the indicated way,

$$\widetilde{p}_i(\vec{x}_i) = w \iff \mathsf{p}_i(\vec{x}_i) :\ \to^*\ : w,$$

and show by an easy induction on the term F the version of (2B-8) for these,

$$\text{(2B-9)}\qquad \operatorname{den}(\mathbf{A}, F(\widetilde{p}_1, \ldots, \widetilde{p}_K)) = w \iff F :\ \to^*_{\mathfrak{i}(\mathbf{A},E)}\ : w.$$

When we apply this to the terms $E_i\{\vec{\mathsf{x}}_i :\equiv \vec{x}_i\}$ and use the form of the internal call transition rule, we get

$$\operatorname{den}(\mathbf{A}, E_i(\vec{x}_i, \widetilde{p}_1, \ldots, \widetilde{p}_K)) = w \iff \widetilde{p}_i(\vec{x}_i) = w,$$

which means that the partial functions $\widetilde{p}_1, \ldots, \widetilde{p}_K$ satisfy the system (2A-3), and so $\overline{p}_1 \le \widetilde{p}_1, \ldots, \overline{p}_K \le \widetilde{p}_K$.

Next we show that for any closed term F as above and any system $p_1, \ldots, p_K$ of solutions of (2A-3),

$$F :\ \to^*\ w \Longrightarrow \operatorname{den}(\mathbf{A}, F(\overline{p}_1, \ldots, \overline{p}_K)) = w.$$

$$\begin{array}{rcll}
f(2,3) & : & & \text{(comp)} \\
f\ 2\ 3 & : & & \text{(pass, pass)} \\
f & : & 2\ 3 & \text{(i-call)} \\
\textit{if}\ (2=0)\ \textit{then}\ 3\ \textit{else}\ S(f(\mathrm{Pd}(2),3)) & : & & \text{(br)} \\
3\ S(f(\mathrm{Pd}(2),3))\ ?\ \mathrm{eq}_0(2) & : & & \text{(pass)} \\
3\ S(f(\mathrm{Pd}(2),3))\ ? & : & \mathrm{ff} & \text{(br2)} \\
S(f(\mathrm{Pd}(2),3)) & : & & \text{(comp)} \\
S\ f(\mathrm{Pd}(2),3) & : & & \text{(comp)} \\
S\ f\ \mathrm{Pd}(2)\ 3 & : & & \text{(pass)} \\
S\ f\ \mathrm{Pd}(2) & : & 3 & \text{(comp)} \\
S\ f\ \mathrm{Pd}\ 2 & : & 3 & \text{(pass)} \\
S\ f\ \mathrm{Pd} & : & 2\ 3 & \text{(e-call)} \\
S\ f & : & 1\ 3 & \text{(i-call)} \\
S\ \textit{if}\ (1=0)\ \textit{then}\ 3\ \textit{else}\ S(f(\mathrm{Pd}(1),3)) & : & & \text{(br), (comp many times)} \\
S\ S\ f\ \mathrm{Pd}(1)\ 3 & : & & \text{(pass)} \\
S\ S\ f\ \mathrm{Pd}(1) & : & 3 & \text{(comp)} \\
S\ S\ f\ \mathrm{Pd}\ 1 & : & 3 & \text{(pass)} \\
S\ S\ f\ \mathrm{Pd} & : & 1\ 3 & \text{(e-call)} \\
S\ S\ f & : & 0\ 3 & \text{(i-call)} \\
S\ S\ \textit{if}\ (0=0)\ \textit{then}\ 3\ \textit{else}\ S(f(\mathrm{Pd}(0),3)) & : & & \text{(br), (comp many times), (pass)} \\
S\ S\ 3\ S\ f(\mathrm{Pd}(0),3)\ ?\ \mathrm{eq}_0(0) & : & & \\
S\ S\ 3\ S\ f(\mathrm{Pd}(0),3)\ ? & : & \mathrm{tt} & \text{(br0)} \\
S\ S\ 3 & : & & \text{(pass)} \\
S\ S & : & 3 & \text{(e-call)} \\
S & : & 4 & \text{(e-call)} \\
 & : & 5 &
\end{array}$$

Figure 2. The computation of $2 + 3$ by the program $f(i,x) = \text{if } \mathrm{eq}_0(i) \text{ then } x \text{ else } S(f(\mathrm{Pd}(i),x))$.

This is done by induction of the length of the computation which establishes the hypothesis, and setting $F \equiv E_0\{\vec{\mathrm{x}}_i :\equiv \vec{x}_i\}$, it implies that

$$\widetilde{p}_1 \le \overline{p}_1, \dots, \widetilde{p}_K \le \overline{p}_K.$$

It follows that $\widetilde{p}_1, \dots, \widetilde{p}_K$ are the least solutions of (2A-3), i.e., $\widetilde{p}_i = \overline{p}_i$, which together with (2B-9) completes the proof.

Both arguments appeal repeatedly to the simple but basic Lemma 2B.2.⊣

There are many natural complexity measures that can be associated with the recursive machine $\mathfrak{i}(\mathbf{A}, E)$, among them

$$\begin{aligned} \mathrm{Time}_E(\vec{x}) &= \text{the length of the computation starting with } E_0(\vec{x}) : \\ \mathrm{Time}^e_{E,\Phi_0}(\vec{x}) &= \text{the number of external calls} \\ &\qquad \text{to primitives } \phi \in \Phi_0 \text{ in this computation,} \end{aligned}$$

where $\Phi_0 \subseteq \Phi$ is a subset of the vocabulary. We will compare them in the next chapter with the natural, "structural" complexity measures which can be defined directly for each recursive program E, without reference to its implementations.

2B.3. Simulating Turing machines with $\mathbf{N}_b$-programs. To connect these complexity measures with classical, Turing-machine time complexity, we show here (in outline) one typical result:[8]

2B.4. **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 an $\mathbf{N}_b$-program E which computes f with $\mathrm{Time}_{\mathfrak{i}}(n) \le kT(\log n)$ for $n > 0$, where $\mathfrak{i} = \mathfrak{i}(\mathbf{N}_b, E)$ is the recursive machine associated with E.*

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

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

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

Notice that $y_0 = \text{parity}(y)$ and $x_0 = \text{parity}(x)$, 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:

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

where (in the notation of (1C)),

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

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

We leave for the problems the details and the proof of the general result.$\dashv$

Problems for Section 2B

x2B.1. **Problem.** Consider the following three recursive programs in $\mathbf{N}_u$:

$$E_1 \equiv p(x) \text{ where } \{p(\mathsf{x}) = S(p(x))\}$$
$$E_2 \equiv p(x) \text{ where } \{p(x) = p(q(x)), q(x) = x\},$$
$$E_3 \equiv p(x,y) \text{ where } \{p(x,y) = q(p(x,y),y), q(x,y) = x\}.$$

Determine the partial functions computed by these programs and discuss how their computations by the recursive machine differ.

x2B.2. **Problem.** Let $\mathbf{A}$ be a Φ structure where Φ contains the binary function constant ϕ and the unary function constant ψ which are interpreted by total functions in $\mathbf{A}$. Let

$$f(x) = \phi^{\mathbf{A}}(\psi^{\mathbf{A}}(x), \psi^{\mathbf{A}}(x)) \quad (x \in A).$$

(1) Check that the recursive machine for the obvious (explicit) program

$$E \equiv \phi(\psi(\mathsf{x}), \psi(\mathsf{x}))$$

which computes f in $\mathbf{A}$ will make two calls to ψ in its computations.

(2) Construct a better recursive program E which computes $f(x)$ in $\mathbf{A}$ using only one call to ψ.

x2B.3. **Problem.** Construct a program in $\mathbf{A}_\varepsilon$ which computes $\gcd(x,y)$ making exactly $c_{\{\text{rem}\}}(\varepsilon, x, y)$ calls to rem when $x \geq y \geq 1$, as this measure was defined in (1D-8).

x2B.4. **Problem** (Stack discipline)**.** (1) Show that for every program E in a *total* structure $\mathbf{A}$, and every closed $(\mathbf{A}, E)$-term M, there is no computation of $\mathfrak{i}(\mathbf{A}, E)$ of the form

$$M : \ \to s_1 \to \cdots \to s_m \tag{2B-10}$$

which is *stuck*, i.e., the state s_m is not terminal and there is no s' such that $s \to s'$.

(2) Show that if $\mathbf{A}$ is a partial structure, M is a closed $(\mathbf{A}, E)$-term and the finite computation (2B-10) is stuck, then its last state s_m is of the form

$$\vec{a}\ \phi_j : y_1, \dots , y_{n_j}\ \vec{b}$$

where ϕ_j is a primitive function of $\mathbf{A}$ of arity n_j and $\phi_j(y_1, \dots , y_{n_j})\uparrow$.

x2B.5. **Problem.** Give a detailed proof of Proposition 2B.4 for a Turing machine M which has two, two-way tapes, K symbols in its alphabet and computes a partial function $f : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ of two arguments. Hint: If there are two symbols a and b, represent the blank by 00, a by 01 and b by 10.

Symbolic computation. The *symbolic recursive machine* $\mathfrak{i} = \mathfrak{i}_s(\Phi, E)$ associated with a vocabulary Φ and a Φ-program E is defined as follows.

The states of $\mathfrak{i}$ are all finite sequences s of the form

$$a_0\ \dots\ a_{m-1} : b_0\ \dots\ b_{n-1}$$

where the elements $a_0, \dots , a_{m_1}, b_0, \dots , b_{n-1}$ of s satisfy the following conditions:

- Each a_i is a function symbol in Φ or one of $\mathsf{p}_1, \dots , \mathsf{p}_K$, or a pure voc$(E)$-term, or the special symbol ?, and
- each b_j is a pure, algebraic Φ-term.

The transitions of $\mathfrak{i}$ are those listed for a recursive machine in Table 1, except for the following three which are modified as follows:

$$\boxed{\begin{array}{lrll}
\text{(e-call)} & \vec{a}\ \underline{\phi_i : \vec{x}}\ \vec{b} & \to \vec{a}\ \underline{: \phi_i(\vec{x})}\ \vec{b} & \\
\text{(br0)} & \vec{a}\ \underline{G\ H\ ?: b_0}\ \vec{b} & \to \vec{a}\ \underline{G:}\ \vec{b} & (\text{if } b_0 = \mathrm{tt}) \\
\text{(br1)} & \vec{a}\ \underline{G\ H\ ?: b_0}\ \vec{b} & \to \vec{a}\ \underline{H:}\ \vec{b} & (\text{if } b_0 = \mathrm{ff})
\end{array}}$$

In the last two commands, b_0 is a pure, algebraic Φ-term (perhaps with variables in it), and the conditions $b_0 = \mathrm{tt}$ or $b_0 = \mathrm{ff}$ cannot be checked, unless b_0 is a term with no variables and no Φ-symbols. The computations of $\mathfrak{i}$ are defined relative to an *environment*, a set of boolean claims

$$\begin{aligned}\mathcal{E} = \{\mathrm{tt} = \mathrm{tt}, P_0 = \mathrm{tt}, P_1 = \mathrm{tt}, \dots , P_{m-1} = \mathrm{tt},& \\ \mathrm{ff} = \mathrm{ff}, N_0 = \mathrm{ff}, N_1 = \mathrm{ff}, \dots , N_{n-1} = \mathrm{ff}\},&\end{aligned}$$

where the P_i and N_j are pure, algebraic Φ-terms. We say that $\mathcal{E}$ *activates* (or *justifies*) the transition (br0) if $(b_0 = \mathrm{tt}) \in \mathcal{E}$, and $\mathcal{E}$ *activates* (br1) if $(b_0 = \mathrm{ff}) \in \mathcal{E}$. A computation relative to an environment $\mathcal{E}$ is a sequence of states $s_0, s_1, \dots, s_n$ where for each $i < n$ the Table and the environment justifies the transition $s_i \to s_{i+1}$.

Take, for example, the program which defines $2x$ in $\mathbf{N}_u$,

$$E \equiv p(u,u) \text{ where } \{p(u,v) = \text{if } (v=0) \text{ then } u \text{ else } S(p(u,\mathrm{Pd}(v)))\}$$

and consider the symbolic computation starting with the head $p(u,u)$:

$$\begin{aligned} p(u,u) : \to\ & \text{if } (\mathrm{eq}_0(u)) \text{ then } u \text{ else } S(p(u,\mathrm{Pd}(u))) : \\ & \to\ u\ S(p(u,\mathrm{Pd}(u)))\ ?\ \mathrm{eq}_0(u) : \to\ u\ S(p(u,\mathrm{Pd}(u)))\ ? : \mathrm{eq}_0(u) \end{aligned}$$

If the environment does not *decide* the term $\mathrm{eq}_0(u)$, then the computation cannot go any further, it *stalls*. If the environment has the condition $\mathrm{eq}_0(u) = \mathrm{ff}$, then (br1) is activated and we continue:

$$\begin{aligned} & u\ S(p(u,\mathrm{Pd}(u))\ ? : \mathrm{eq}_0(u)\ \to S(p(u,\mathrm{Pd}(u))) : \to\ S\ p(u,\mathrm{Pd}(u)) : \\ & \quad \to\ S\ p\ u, \mathrm{Pd}(u) : \to\ S\ p\ u, \mathrm{Pd}\ u : \\ & \qquad \to\ S\ p\ u, \mathrm{Pd} : u\ \to\ S\ p\ u : \mathrm{Pd}(u)\ \to\ S\ p : u\ \mathrm{Pd}(u) \\ & \qquad\quad \to\ S \text{ if } \mathrm{eq}_0(u) \text{ then } u \text{ else } S(p(u,\mathrm{Pd}(u))) : \mathrm{Pd}(u) \dots \end{aligned}$$

The next time that ? will show up, we will need to have one of the two conditions

$$\mathrm{eq}_0(\mathrm{Pd}(u)) = \mathrm{tt} \text{ or } \mathrm{eq}_0(\mathrm{Pd}(u)) = \mathrm{ff}$$

in the environment to continue, etc. The computation will go on forever unless the environment has a condition $\mathrm{eq}_0(\mathrm{Pd}^n(u)) = \mathrm{tt}$ for some n, which will then turn it around so that eventually it stops in the state

$$: S^n(u)$$

which gives the correct answer for $u = n$.

The next problem is very easy, once you define correctly the terminology which occurs in it—and it is part of the problem to do this.

x2B.6. **Problem.** Fix a Φ-structure and a Φ-program E, and suppose that

$$M(x_1,\dots,x_n) : \to s_1 \to \cdots \to : w$$

is a computation of the recursive machine of E which computes the value of the closed $(\mathbf{A}, E)$ term $M(x_1,\dots,x_n)$ with the indicated parameters. Prove that there is an environment $\mathcal{E}$ in the distinct variables $\mathsf{x}_1,\dots,\mathsf{x}_n$ which is *sound for* $x_1,\dots,x_n$ in $\mathbf{A}$, such that the given computation is obtained from the symbolic computation relative to $\mathcal{E}$ and starting with $M(v_1,\dots,v_n)$ by replacing each v_i in it by x_i.

x2B.7. **Problem.** Prove that $\mathrm{Pd}(x)$ is not $(\mathbb{N}, 0, S, \mathrm{eq}_0)$-recursive.

There are many other applications of symbolic computation, but we will not cover the topic. (And it is rather surprising that the simple and basic Problem x2B.7 seems to need it. Perhaps there is a simpler proof.)

2C. Finite non-determinism

Much of the material in Section 2B can be extended easily to *non-deterministic computation models*, in which the transition function $\sigma : S \rightharpoonup S$ is replaced by a relation $\sigma \subseteq S \times S$, usually assumed *total*, i.e., such that $(\forall s)(\exists s')\sigma(s, s')$. We do not have much use for these here, and the model theory of the structures associated with them is a bit messy. We will cover only the most useful, special case of finite non-determinism.

For any two sets X, W, a (finitely) *non-deterministic iterator* $\mathfrak{i} : X \rightsquigarrow W$ is a tuple

$$\mathfrak{i} = (\mathrm{input}, S, \sigma_1, \dots, \sigma_k, T, \mathrm{output})$$

which satisfies (I1) – (I5) in Section 2B.1 except that (I3) is replaced by the obvious

(I3′) for every $i = 1, \dots, k$, $\sigma_i : S \rightharpoonup S$.

So $\mathfrak{i}$ has k transition functions. *Computations* are defined as before, except that we allow transitions $s_{i+1} = \sigma_j(s_i)$ by any one of the transition functions, so that, for example, $s, \sigma_1(s), \sigma_3(\sigma_1(s)), \dots$ is a partial computation. This allows the possibility that the machine may produce more than one value on some input, and we must be careful in defining what if means for $\mathfrak{i}$ to compute some $f : X \rightharpoonup W$. The formal definition is the same as (2B-2) for deterministic iterators, i.e., f must satisfy the equaivalence

(2C-1) $$f(x) = w \iff (\exists s \in T)[\mathrm{input}(x) \to^*_{\mathfrak{i}} s \ \&\ \mathrm{output}(s) = w],$$

but it must be read more carefully now: $\mathfrak{i} : X \rightsquigarrow W$ *computes* f *if whenever* $f(x)\downarrow$*, then at least one convergent computation starting with* $\mathrm{input}(x)$ *produces the value* $f(x)$ *and no convergent computation from* $\mathrm{input}(x)$ *produces a different value.* Divergent computations are disregarded.

Non-deterministic recursive programs are defined exactly as before, except that we allow multiple definitions for each recursive variable. For example, in $(\mathbb{N}, 0, S, \phi)$, we might have

(2C-2) $$E^* \equiv \phi(\mathsf{p}(\vec{x}), \vec{x}) \text{ where } \{\mathsf{p}(\vec{x}) = 0, \mathsf{p}(\vec{x}) = S(\mathsf{p}(\vec{x}))\}.$$

It is possible to define least-fixed-point semantics for them, but the details are a bit complex and it is easier to use the associated recursive machines.

These are now non-deterministic: if both

$$\mathsf{p}(\vec{u}) = E^1(\vec{u}, \mathsf{p}_1, \dots, \mathsf{p}_n) \text{ and } \mathsf{p}(\vec{u}) = E^2(\vec{u}, \mathsf{p}_1, \dots, \mathsf{p}_n)$$

are in the body of E, then $\mathsf{i}(\mathbf{A}, E)$ allows both transitions

$$\mathsf{p} : \vec{x} \;\to\; E^1(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_n) : \text{ and } \mathsf{p} : \vec{x} \;\to\; E^2(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_n) :$$

And, again, we say that E computes $f : A^n \rightharpoonup A_s$ in a Φ-structure $\mathbf{A}$ if (2C-1) holds for the iterator $\mathsf{i}(\mathbf{A}, E)$ associated with the non-deterministic program E. The main difference from the deterministic case is that *not every non-deterministic Φ-program computes a partial function in every Φ-structure* $\mathbf{A}$. To express simply when this happens, put

(2C-3) $$\mathbf{A} \vdash E(\vec{x}) = w \iff E_0(\vec{x}) : \to_{\mathsf{i}(\mathbf{A},E)} : w.$$

This extends (2A-6) to non-deterministic programs, and clearly E computes a partial function in $\mathbf{A}$, if and only if for all $\vec{x} \in A^n$ and all $w, w' \in A_s$,

(2C-4) $$\Big(\mathbf{A} \vdash E(\vec{x}) = w \;\&\; \mathbf{A} \vdash E(\vec{x}) = w'\Big) \Longrightarrow w = w'.$$

If E computes f in $\mathbf{A}$, we also set for any $\Phi_0 \subseteq \Phi$,

(2C-5) $$\mathrm{Time}^e_{E,\Phi_0}(\vec{x}) = \min \Big(\text{number of external calls to } \phi \in \Phi_0 \\ \text{in any computation of } \mathsf{i}(\mathbf{A}, E) \text{ on the input } \vec{x}\Big).$$

This is the only complexity measure on non-deterministic programs that we will need (for now).

A partial function $f : A^n \rightharpoonup A_s$ is *non-deterministically recursive* in $\mathbf{A}$ if it is computed by a non-deterministic recursive program in $\mathbf{A}$ for $A_0 = A^n$, and we set

$$\mathbf{rec}_{\mathrm{nd}}(\mathbf{A}) = \text{the set of non-deterministic } \mathbf{A}\text{-recursive partial functions.}$$

The distinction between deterministic and non-deterministic algorithms underlies some of the most interesting and deep problems of computation theory, including the central P=NP? problem. Closer to the questions we have been considering, the results of Stolboushkin and Taitslin [1983] and Tiuryn [1989] that we mentioned above were established to distinguish *deterministic* from *non-deterministic tail recursion* (in the terminology we have been using) as well as (deterministic) tail recursion from full recursion. We will not go into the topic here, except for the following discussion of a non-deterministic algorithm for the gcd (and coprimeness) which is relevant to the Main Conjecture in the Preface.[9]

[9]This theorem and Problems x2C.6 – x2C.9 are in Pratt [2008] which has not been published. They are included here with Vaughan Pratt's permission.

2C.1. **Theorem** (Pratt's nuclid algorithm). *Consider the following non-deterministic recursive program E_P in the structure $\mathbf{N}_\varepsilon = (\mathbb{N}, \mathrm{rem}, \mathrm{eq}_0, \mathrm{eq}_1)$ of the Euclidean:*

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

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

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

If $a \geq b \geq 1$, then $f_{E_P}(a,b) = \gcd(a,b)$.

PROOF. Fix $a \geq b \geq 1$, and let

$$\begin{aligned}&(m,n) \to (m',n')\\ &\quad\iff \Big(n \neq 0 \ \&\ m' = n\\ &\qquad\qquad \&\ [n' = \mathrm{rem}(m,n) \vee n' = \mathrm{rem}(a,n) \vee n' = \mathrm{rem}(b,n)]\Big)\\ &\qquad \vee \Big(n = 0 \ \&\ \mathrm{rem}(a,m) \neq 0 \ \&\ m' = m \ \&\ n' = \mathrm{rem}(a,m)\Big)\\ &\qquad \vee \Big(n = 0 \ \&\ \mathrm{rem}(b,m) \neq 0 \ \&\ m' = m \ \&\ n' = \mathrm{rem}(b,m)\Big).\end{aligned}$$

This is the transition relation of the main loop of the program (with a, b omitted), and it obviously respects the property $m > 0$. The terminal states are

$$T(a,b,m,n) \iff n = 0 \ \&\ m \mid a \ \&\ m \mid b,$$

and the output on a terminal $(a,b,m,0)$ is m.

It is obvious that there is at least one computation which outputs $\gcd(a,b)$, because one of the choices at each step is the one that the Euclidean would make. To see that no convergent computation produces any other value, we observe that directly from the definition,

If x divides a, b, m and n and $(m,n) \to (m',n')$, then x divides m' and n'.

Since the input satisfies the hypothesis of this remark, every divisor of a and b divides the output m; and because of the conditions on the terminal state, every divisor of an output m divides both a and b, so that $m = \gcd(a,b)$.⊣

In fact, E_P does not have any divergent computations on its intended inputs, see Problem x2C.6. Pratt's algorithm allows at each stage replacing the Euclidean's $(m,n) \to (n, \mathrm{rem}(m,n))$ by $(m,n) \to (n, \mathrm{rem}(a,n))$ or $(m,n) \to (n, \mathrm{rem}(b,n))$ which does not lose any common divisors of a

and b, and then simply adds a check at the end which insures that the output is not some random divisor of (say) a which does not also divide b. The important thing about it is that in some cases this guessing can produce a much faster computation of $\gcd(a, b)$: see Problems x2C.7 – x2C.9 which outline a proof that for successive Fibonacci numbers it can compute $\gcd(F_{t+1}, F_t)$ using only

$$O(\log t) = O(\log\log(F_t))$$

divisions, thus beating the Euclidean on its worst case. A complete analysis of the inputs on which it does better than the Euclidean does not appear to be easy.

Problems for Section 2C

x2C.1. **Problem.** Prove that the following are equivalent for a Φ-structure $\mathbf{A}$ and a non-deterministic Φ-program E:

(a) E computes a partial function in $\mathbf{A}$.
(b) E computes a partial function in every substructure $\mathbf{U} \subseteq_p \mathbf{A}$.
(c) E computes a partial function in every finite substructure $\mathbf{U} \subseteq_p \mathbf{A}$.

x2C.2. **Problem.** Formulate for $\mathbf{rec}_{\rm nd}(\mathbf{A})$ and prove the properties in Problems x2A.2, x2A.4, x2A.5 and x2A.6.

x2C.3. **Problem.** Prove that if $\mathbf{\Phi}$ is a set of total functions on $\mathbb{N}$, then

$$\mathbf{rec}_{\rm nd}(\mathbf{N}_u, \mathbf{\Phi}) = \mathbf{rec}_{\rm nd}(\mathbf{N}_b, \mathbf{\Phi}) = \mathbf{rec}(\mathbf{N}_u, \Phi).$$

x2C.4*. **Problem.** Give an example of a partial function $\phi : \mathbb{N}^2 \rightharpoonup \mathbb{N}$ such that

$$\mathbf{rec}(\mathbf{N}_u, \phi) \subsetneq \mathbf{rec}_{\rm nd}(\mathbf{N}_u, \phi).$$

x2C.5. **Problem.** For the program E^* defined in (2C-2), prove that

$$\overline{p}_{E^*}(\vec{x}) = w \iff (\exists n)[\phi^{\mathbf{A}}(n, \vec{x}) = w]$$

provided that

$$(\forall n, m, u, v)[\phi^{\mathbf{A}}(n, \vec{x}) = u \ \&\ \phi^{\mathbf{A}}(m, \vec{x}) = v] \Longrightarrow u = v\};$$

if this condition does not hold, then E^* does not compute a partial function in $(\mathbb{N}, 0, S, \phi)$. Define also a related non-deterministic program E^{**} which computes in the same structure $(\mathbb{N}, 0, S, \phi)$ a partial function $\overline{p}_{E^{**}}$ such that

$$\overline{p}_{E^{**}}(\vec{x})\downarrow \iff (\exists n)[\phi^{\mathbf{A}}(n, \vec{x})\downarrow].$$

The remaining problems in this section are due to Vaughan Pratt.

x2C.6. **Problem.** Prove that the program E_P has no divergent (infinite) computations on inputs (a, b, a, b) with $a \geq b \geq 1$. Hint: Show convergence of the main loop under the hypothesis by induction on $\max(m, n)$ and within this by induction on n.

The complexity estimate for Pratt's algorithm depends on some classical identities that relate the Fibonacci numbers.

x2C.7. **Problem.** Prove that for all $t \geq 1$ and $m \geq t$,

$$F_m(F_{t+1} + F_{t-1}) = F_{m+t} + (-1)^t F_{m-t}. \tag{2C-6}$$

Hint: Show in sequence, by direct computation, that

$$\varphi\hat{\varphi} = -1; \quad \varphi + \frac{1}{\varphi} = \sqrt{5}; \quad \hat{\varphi} + \frac{1}{\hat{\varphi}} = -\sqrt{5}; \quad F_{t+1} + F_{t-1} = \varphi^t + \hat{\varphi}^t.$$

x2C.8. **Problem.** (1) Prove that for all odd $t \geq 2$ and $m \geq t$,

$$\mathrm{rem}(F_{m+t}, F_m) = F_{m-t} \qquad (t \text{ odd}). \tag{2C-7}$$

(2) Prove that for all even $t \geq 2$ and $m \geq t$,

$$\mathrm{rem}(F_{m+t}, F_m) = F_m - F_{m-t}, \tag{2C-8}$$

$$\mathrm{rem}(F_m, (\mathrm{rem}(F_{m+t}, F_m))) = F_{m-t}, \tag{2C-9}$$

Hint: For (2), check that for $t \geq 2$, $2F_{m-t} < F_m$.

x2C.9. **Problem.** Fix $t \geq 2$. Prove that for every $s \geq 1$ and every u such that $u \leq 2^s$ and $t - u \geq 2$, there is a computation of Pratt's algorithm which starts from $(F_{t+1}, F_t, F_{t+1}, F_t)$ and reaches a state $(F_{t+1}, F_t, ?, F_{t-u})$ doing no more than $2s$ divisions.

Infer that for all $t \geq 3$,

$$\mathrm{Time}^e_{E_P}(F_{t+1}, F_t) \leq 2\lceil \log(t-2) \rceil + 1 = O(\log(t-1)). \tag{2C-10}$$

(The measure $\mathrm{Time}^e_E(\vec{x})$ for non-deterministic recursive programs is defined in (2C-5).)

2D. The homomorphism and finiteness properties

In the notation introduced by (2C-3), Problem x2A.12 says that

$$\mathbf{A} \vdash E(\vec{x}) = w \in A \Longrightarrow w \in G_\infty(\mathbf{A}, \vec{x}). \tag{2D-1}$$

which also (easily) holds for non-deterministic programs, Problem x2D.2. This is a weak restriction on **A**-recursive functions which, in particular, has no consequences for **A**-recursive relations. We formulate here two related, simple but very basic properties of non-deterministic recursive programs that will prove very useful further on.

2D.1. **Theorem.** *For any Φ-structure* $\mathbf{A}$ *and any non-deterministic Φ-program E:*

(a) *The Homomorphism property: if $\pi : \mathbf{U} \to \mathbf{V}$ is any homomorphism of one substructure $\mathbf{U} \subseteq_p \mathbf{A}$ into another, then*

$$\mathbf{U} \vdash E(\vec{x}) = w \Longrightarrow \mathbf{V} \vdash E(\vec{\pi}(x)) = \pi(w) \quad (\vec{x} \in U^n).$$

(b) *The Finiteness property: for any $\vec{x} \in A^n$,*

$$\mathbf{A} \vdash E(\vec{x}) = w \Longrightarrow (\exists m)[\mathbf{G}_m(\mathbf{A}, \vec{x}) \vdash E(\vec{x}) = w].$$

Moreover, if E is a program with empty body (a pure Φ-term)*, then*

$$(\exists m)(\forall \vec{x}, w)[\mathbf{A} \vdash E(\vec{x}) = w \Longrightarrow \mathbf{G}_m(\mathbf{A}, \vec{x}) \vdash E(\vec{x}) = w].$$

PROOF is simple and we will leave it for a problem. ⊣

Problems for Section 2D

x2D.1. **Problem.** Prove Proposition 2D.1. HINT: For each computation

$$\mathrm{Comp}_{\mathfrak{i}_E}(\vec{x}) = (E_0(\vec{x}) :, \dots, : w)$$

of the recursive machine which proves that $\mathbf{U} \vdash E(\vec{x}) = w$, define a sequence of states

$$\pi(\mathrm{Comp}_{\mathfrak{i}_E}(\vec{x})) = (E_0(\pi(\vec{x})) :, \dots, : \pi(w))$$

by replacing every parameter $u \in A$ which occurs in $\mathrm{Comp}_(\vec{x})$ by $\pi(u)$ and verify that $\pi(\mathrm{Comp}_(\vec{x}))$ is a computation of the recursive machine which proves that $\mathbf{V} \vdash E(\pi(\vec{x})) = \pi(w)$.

x2D.2. **Problem.** Prove that Problem x2A.12 holds for any non-deterministic $\mathbf{A}$-recursive functions and infer that $S \notin \mathbf{rec}_{\mathrm{nd}}(\mathbb{N}, 0, \mathrm{Pd}, \mathrm{eq}_0)$.

x2D.3. **Problem.** True or false: $\mathrm{Pd} \in \mathbf{rec}_{\mathrm{nd}}(\mathbb{N}, 0, S, \mathrm{eq}_0)$?

x2D.4. **Problem.** Prove the claims in Problems x1C.13* – x1C.15* (and formulate the proofs so that they could have been given immediately after these problems were stated, without reference to recursive computation).

CHAPTER 3

COMPLEXITY THEORY FOR RECURSIVE PROGRAMS

Suppose Π is a class of programs which compute (in some precise sense) partial functions and relations on a set A. In the most general terms, a *complexity measure* for Π associates with each n-ary $E \in \Pi$ which computes $f : A^n \rightharpoonup A_s$ an n-ary partial function

(3-2) $$C_E : A^n \rightharpoonup \mathbb{N}$$

which intuitively assigns to each $\vec{x}$ such that $f(\vec{x})\downarrow$ a *cost* of some sort of the computation of $f(\vec{x})$ by E.

We introduce here several natural complexity measures on the (deterministic) recursive programs of a Φ-structure $\mathbf{A}$, directly from the programs, i.e., without reference to the recursive machine. These somewhat abstract, "implementation-independent" (*logical*) definitions help clarify some complexity questions, and they are also useful in the derivation of upper bounds for recursive programs and robust lower bounds for problems. Much of what we will say extends naturally to non-deterministic programs, but the formulas and arguments are substantially more complex and it is better to keep matters simpler by dealing first with the more important, deterministic case.

3A. The basic complexity measures

Suppose $\mathbf{A}$ is a structure, E is an n-ary $\mathbf{A}$-program and M is a closed $(\mathbf{A}, E)$-term as these were defined in (2B-7). We set

(3A-1) $$\overline{M} = \mathrm{den}(\mathbf{A}, E, M) = \mathrm{den}((\mathbf{A}, \overline{p}_1, \dots, \overline{p}_K), M)$$

where $\mathsf{p}_1, \dots, \mathsf{p}_K$ are the recursive variables of E and $\overline{p}_1, \dots, \overline{p}_K$ their mutual fixed points. This extends the notation in (2A-5) by which

$$\mathrm{den}(\mathbf{A}, E, \vec{x}) = \mathrm{den}(\mathbf{A}, E, E_0(\vec{x})).$$

Normally we will use the simpler $\overline{M}$ since $\mathbf{A}$ and E will be held constant in most of the arguments in this section. We also set

$$\text{(3A-2)} \qquad \mathrm{Conv}(\mathbf{A}, E) = \{M : M \text{ is an } (\mathbf{A}, E)\text{-term and } \overline{M}\downarrow\}.$$

3A.1. The tree-depth complexity $D^{\mathbf{A}}_E(M)$. With each convergent $(\mathbf{A}, E)$-term M, we can associate a *computation tree* $\mathcal{T}(M)$ which represents an abstract, parallel computation of $\overline{M}$ using E. The tree-depth complexity of M is the depth of $\mathcal{T}(M)$, but it is easier to define $D^{\mathbf{A}}_E(M)$ first and $\mathcal{T}(M)$ after that, by recursion on $D^{\mathbf{A}}_E(M)$.

3A.1. **Lemma.** *Fix a structure $\mathbf{A}$ and an $\mathbf{A}$-program E. There is exactly one function $D = D^{\mathbf{A}}_E$ which is defined for every $M \in \mathrm{Conv}(\mathbf{A}, E)$ and satisfies the following conditions:*

(D1) $D(\mathrm{tt}) = D(\mathrm{ff}) = D(x) = 0.$
(D2) $D(\phi(M_1, \dots, M_m)) = \max\{D(M_1), \dots, D(M_m)\} + 1.$
(D3) *If* $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, *then*

$$D(M) = \begin{cases} \max\{D(M_0), D(M_1)\} + 1, & \textit{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{D(M_0), D(M_2)\} + 1, & \textit{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(D4) *If* p *is a recursive variable of* E,[10] *then*

$$D(\mathsf{p}(M_1, \dots, M_m)) = \max\{D(M_1), \dots, D(M_m), d_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)\} + 1,$$

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

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

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

by Lemma 2A.1. We define $D(M)$ by recursion on

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

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

(D1) If M is tt, ff or a parameter x, set $D(M) = 0$.

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

$$\mathrm{stage}(M) = \max\{\mathrm{stage}(M_1), \dots, \mathrm{stage}(M_m)\},$$

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

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

[10] If $\mathsf{p} \equiv \mathsf{p}_i$ is a recursive variable of E, we sometimes write $E_{\mathsf{p}} \equiv E_i$ for the term which defines p in E. It is a useful convention which saves typing double subscripts.

(D3) If $M \equiv$ if M_0 then M_1 else M_2 and $\overline{M}\downarrow$, then either $\overline{M}_0 = \mathrm{tt}$, $\overline{M}_1\downarrow$ and $\mathrm{stage}(M) = \max\{\mathrm{stage}(M_0), \mathrm{stage}(M_1)\}$ or the corresponding conditions hold with M_0 and M_2. In either case, the terms M_0, M_i are proper subterms of M, and we can assume that D is defined for them and define $D(M)$ appropriately as in case (D2).

(D4) If $M \equiv \mathsf{p}(M_1, \dots, M_m)$ with a recursive variable p of E, $\overline{M}\downarrow$ and $k = \mathrm{stage}(M)$, then

$$\overline{M}^k = \overline{p}^k(\overline{M}_1^k, \dots, \overline{M}_m^k),$$

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

$$\overline{p}^k(w_1, \dots, w_m) = \mathrm{den}(E_{\mathsf{p}}(w_1, \dots, w_m, \overline{p}_1^{k-1}, \dots, \overline{p}_K^{k-1}))\downarrow,$$

by the definition of the iterates in the proof of Lemma 1B.1, and so

$$\mathrm{stage}(E_{\mathsf{p}}(w_1, \dots, w_m, \mathsf{p}_1, \dots, \mathsf{p}_K)) < k;$$

thus we may assume that $D(E_{\mathsf{p}}(w_1, \dots, w_m, \mathsf{p}_1, \dots, \mathsf{p}_K))$ is defined, and define $D(M)$ so that (D4) in the Lemma holds.

The uniqueness of D is proved by a simple induction on $\mathrm{stage}(M)$, following the definition. ⊣

The *tree-depth complexity function* of a program E is that of its head term,

$$d_E^{\mathbf{A}}(\vec{x}) = D(E_0(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_K)),$$

and it is defined exactly when $\overline{p}_E^{\mathbf{A}}(\vec{x})\downarrow$.

It should be clear that there is no reasonable way to implement recursive programs so that the number of steps required to compute $\overline{M}$ is $D(M)$. For example, to attain

$$D(\mathsf{p}(M)) = \max\{D(M), D(E_{\mathsf{p}}(\overline{M}))\} + 1,$$

we need to compute in parallel the value $\overline{M}$ of M and the value of $E_{\mathsf{p}}(\overline{M})$, but we cannot start on the second computation until we complete the first, so that we know $\overline{M}$. We can imagine a non-deterministic process which "guesses" the correct $\overline{M}$ and works with that; but if $\mathbf{A}$ is infinite, then this amounts to infinite non-determinism, which is not a useful idealization.

In any case, our methods do not yield any interesting lower bounds for tree-depth complexity, but it is a very useful tool for defining rigorously and analyzing many properties of recursive programs.

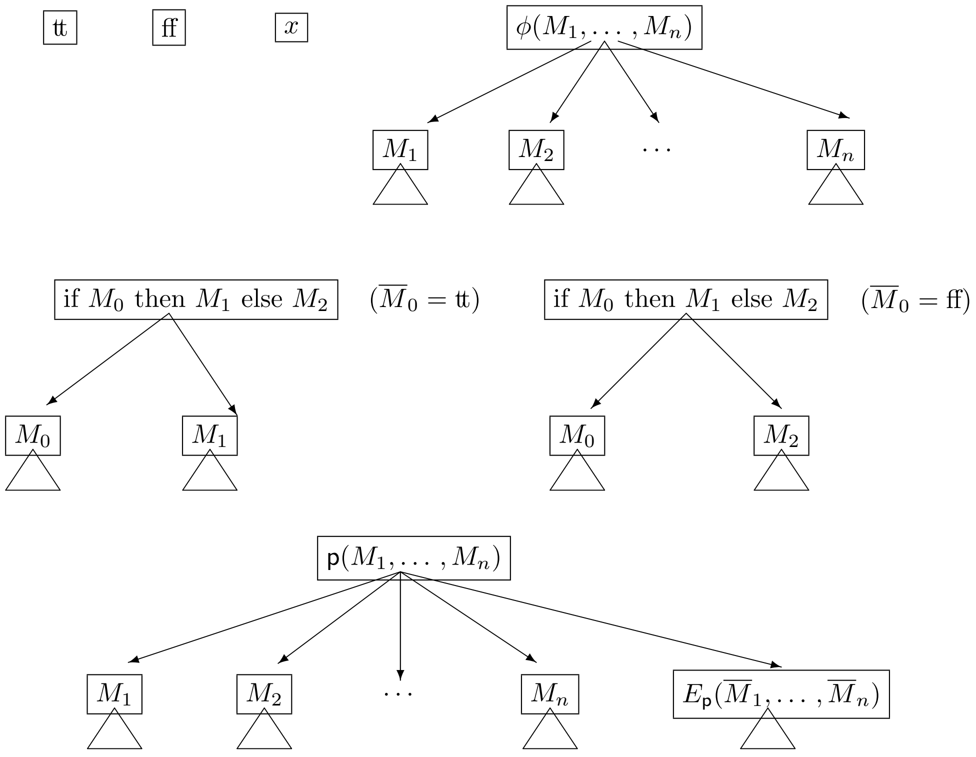

tt
ff
x
$\phi(M_1,\dots,M_n)$
M_1
M_2
$\cdots$
M_n
if M_0 then M_1 else M_2
$(\overline{M}_0 = \mathrm{tt})$
M_0
M_1
if M_0 then M_1 else M_2
$(\overline{M}_0 = \mathrm{ff})$
M_0
M_2
$\mathsf{p}(M_1,\dots,M_n)$
M_1
M_2
$\cdots$
M_n
$E_{\mathsf{p}}(\overline{M}_1,\dots,\overline{M}_n)$

Figure 3. Computation trees.

The computation tree. The *computation tree* $\mathcal{T}(M) = \mathcal{T}(\mathbf{A}, E, M)$ for $M \in \mathrm{Conv}(\mathbf{A}, E)$ is defined by recursion on $D(M)$ using the operation Top in (1A-6), see Figure 3. We take cases, corresponding to the definition of $D(M)$:

($\mathcal{T}1$) If M is tt, ff or some $x \in A$, set $\mathcal{T}(M) = \{(M)\}$.

($\mathcal{T}2$) If $M \equiv \mathcal{T}(\phi(M_1,\dots,M_m))$, set $\mathcal{T}(M) = \mathrm{Top}(M, \mathcal{T}(M_1),\dots,\mathcal{T}(M_m))$.

($\mathcal{T}3$) If $M \equiv$ if M_0 then M_1 else M_2, set

$$\mathcal{T}(M) = \begin{cases} \mathrm{Top}(M, \mathcal{T}(M_0), \mathcal{T}(M_1)) & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \mathrm{Top}(M, \mathcal{T}(M_0), \mathcal{T}(M_2)) & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

($\mathcal{T}4$) If $M \equiv \mathsf{p}(M_1,\dots,M_m)$, set

$$\mathcal{T}(M) = \mathrm{Top}(M, \mathcal{T}(M_1),\dots,\mathcal{T}(M_m), \mathcal{T}(E_{\mathsf{p}}(\overline{M}_1,\dots,\overline{M}_m))).$$

3A.2. **Proposition.** *For every $M \in \mathrm{Conv}(\mathbf{A}, E)$,*

$$D(M) = \mathrm{depth}(\mathcal{T}(M)).$$

Proof is immediate, by induction on $D(M)$. ⊣

3A.2. The sequential logical complexity $L^s(M)$. For a fixed Φ-structure $\mathbf{A}$ and a Φ-program E, the *sequential logical complexity* $L^s(M)$ of each $M \in \mathrm{Conv}(\mathbf{A}, E)$ is defined by the following recursion on $D(M)$:

(L^s1) $L^s(\mathrm{tt}) = L^s(\mathrm{ff}) = L^s(x) = 0 \quad (x \in A)$.

(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 $M \equiv$ if M_0 then M_1 else M_2, then

$$L^s(M) = \begin{cases} L^s(M_0) + L^s(M_1) + 1 & \text{if } \overline{M}_0 = \mathrm{tt}, \\ L^s(M_0) + L^s(M_2) + 1 & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(L^s4) If p is a recursive variable of E, then

$$L^s(\mathsf{p}(M_1, \dots, M_n)) = L^s(M_1) + \cdots + L^s(M_n) \\ + L^s(E_{\mathsf{p}}(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_K)) + 1.$$

The sequential logical complexity

$$(l^s) \qquad l^s_E(\vec{x}) = L^s(E_0(\vec{x})) \quad (\overline{p}^{\mathbf{A}}_E(\vec{x})\downarrow)$$

counts the minimal number of steps that a sequential (call-by-value) implementation of E would execute on the input $\vec{x}$. It aims to capture the most natural implementation-independent time complexity measure for recursive programs.

3A.3. The parallel logical complexity $L^p(M)$. For a fixed Φ-structure $\mathbf{A}$ and a Φ-program E, the *parallel logical complexity* $L^p(M)$ of each term $M \in \mathrm{Conv}(\mathbf{A}, E)$ is defined by the following recursion on $D(M)$:

(L^p1) $L^p(\mathrm{tt}) = L^p(\mathrm{ff}) = L^p(x) = 0 \quad (x \in A)$.

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

(L^p3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$L^p(M) = \begin{cases} \max\{L^p(M_0), L^p(M_1)\} + 1, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{L^p(M_0), L^p(M_2)\} + 1, & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(L^p4) If p is a recursive variable of E, then

$$L^p(\mathsf{p}(M_1, \dots, M_n)) = \max\{L^p(M_1), \dots, L^p(M_n)\} \\ + L^p(E_{\mathsf{p}}(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_K)) + 1.$$

We also set

$$(l^p) \qquad l^p_E(\vec{x}) = L^p(E_0(\vec{x})) \quad (\overline{p}^{\mathbf{A}}_E(\vec{x})\downarrow).$$

The measure $l^p_E(\vec{x})$ counts the (minimal) number of steps that must be executed in sequence by a fully parallel, call-by-value implementation of E on the input $\vec{x}$.

The difference between $l^s_E(\vec{x})$ and $l^p_E(\vec{x})$ measures (in some vague sense) how "parallel" the algorithm expressed by E is and it is no more than exponential by the next result. The base of the exponential is one more than the *total arity* of the program

$$\text{(3A-3)} \quad \ell(E) = \max\{\text{arity}(E), \max\{\text{arity}(\phi) : \phi \in \Phi\}, \max\{\text{arity}(\mathsf{p}_i) : i = 1, \dots, K\}\} \geq 1.$$

3A.3. **Theorem.** *For every Φ-structure $\mathbf{A}$, every Φ-program E of total arity $\ell \geq 1$, and every $M \in \text{Conv}(\mathbf{A}, E)$,*

(a) $L^s(M) \leq \text{size}(\mathcal{T}(M))$,
(b) $\text{depth}(\mathcal{T}(M)) \leq L^p(M)$,
(c) $L^s(M) \leq (\ell + 1)^{L^p(M)}$,

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

$$l^s_E(\vec{x}) \leq (\ell + 1)^{l^p_E(\vec{x})}.$$

Proof. (a) and (b) are verified by simple inductions on $D(M)$, and then (c) follows by (1A-5) since the degree of $\mathcal{T}(M)$ is obviously $\leq \ell + 1$.

For example, in the most complex case for (a) with $M \equiv \mathsf{p}(M_1, \dots, M_n)$:

$$\begin{aligned} L^s(M) &= L^s(M_1) + \cdots + L^s(M_n) + L^s(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n)) + 1 \\ &\leq \text{size}(\mathcal{T}(M_1)) + \cdots + \text{size}(\mathcal{T}(M_n)) \\ &\qquad + \text{size}(\mathcal{T}(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n))) + 1 \\ &= \text{size}(\mathcal{T}(M)). \end{aligned}$$

For the corresponding case for (b):

$$\begin{aligned} \text{depth}(\mathcal{T}(M)) &= \max\{\text{depth}(\mathcal{T}(M_1)), \dots, \text{depth}(\mathcal{T}(M_n)), \\ &\qquad \text{depth}(\mathcal{T}(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n)))\} + 1 \\ &\leq \max\{\text{depth}(\mathcal{T}(M_1)), \dots, \text{depth}(\mathcal{T}(M_n))\} \\ &\qquad + \text{depth}(\mathcal{T}(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n))) + 1 \\ &\leq \max\{L^p(M_1), \dots, L^p(M_n)\} \\ &\qquad + L^p(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n)) + 1 \\ &= L^p(M). \end{aligned}$$

(The inequalities on size and depth are obvious from the tree pictures in Figure 3.) $\dashv$

Next we define two complexity measures on recursive programs which disregard the logical steps and count only calls to the primitives.

3A.4. The number-of-calls complexity $C^s(M)$. Fix a Φ-structure $\mathbf{A}$, a subset $\Phi_0 \subseteq \Phi$ of the vocabulary and a Φ-program E. The *number of calls to* Φ_0

$$C^s_{\Phi_0}(M) = C^s_{\Phi_0}(\mathbf{A}, E, M)$$

of any $M \in \mathrm{Conv}(\mathbf{A}, E)$ is defined by the following recursion on $D(M)$:

(C^s1) $C^s_{\Phi_0}(\mathrm{tt}) = C^s_{\Phi_0}(\mathrm{ff}) = C^s(x) = 0 \quad (x \in A)$.

(C^s2) If $M \equiv \phi(M_1, \dots, M_m)$, then

$$C^s_{\Phi_0}(M) = \begin{cases} C^s_{\Phi_0}(M_1) + \cdots + C^s_{\Phi_0}(M_m) + 1, & \text{if } \phi \in \Phi_0, \\ C^s_{\Phi_0}(M_1) + \cdots + C^s_{\Phi_0}(M_m), & \text{otherwise.} \end{cases}$$

(C^s3) If $M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2$, then

$$C^s_{\Phi_0}(M) = \begin{cases} C^s_{\Phi_0}(M_0) + C^s_{\Phi_0}(M_1), & \text{if } \overline{M}_0 = \mathrm{tt}, \\ C^s_{\Phi_0}(M_0) + C^s_{\Phi_0}(M_2), & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(C^s4) If $M \equiv \mathsf{p}(M_1, \dots, M_m)$ with p a recursive variable of E, then

$$C^s_{\Phi_0}(M) = C^s_{\Phi_0}(M_1) + \cdots + C^s_{\Phi_0}(M_m) + C^s_{\Phi_0}(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)).$$

The number of Φ_0-calls complexity of the partial function $\overline{p}_E : A^n \rightharpoonup A_{S_s}$ to Φ_0 of E in $\mathbf{A}$ is that of its head term,

($c^s_{\Phi_0}$) $$c^s_{\Phi_0}(\vec{x}) = c^s_{\Phi_0}(\mathbf{A}, E, \vec{x}) = C^s_{\Phi_0}(\mathbf{A}, E, E_0(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_K)),$$

and it is defined exactly when $\overline{p}_E(\vec{x})\downarrow$. We also set

(C^s, c^s) $$C^s(M) = C^s_{\Phi}(M), \quad c^s(\vec{x}) = c^s_{\Phi}(\vec{x})$$

when we want to count the calls to all primitives.

This is a very natural complexity measure: $C^s_{\Phi_0}(M)$ counts *the number of calls to the primitives in* Φ_0 which are required for the computation of $\overline{M}$ using "the algorithm expressed" by the program E and disregarding the "logical steps" (branching and recursive calls) as well as calls to primitives not in Φ_0. It does not distinguish between parallel and sequential implementations of E, although it is more directly relevant to the second—so we will sometimes refer to it as the *sequential calls complexity.*

Notice that E may (stupidly) call many times for the same value of one of the primitives, and all these calls will be counted separately by $C^s_{\Phi_0}(M)$. Most of the lower bounds of algebraic problems that we will derive are about a somewhat smaller measure which counts only the number of distinct calls to the primitives in Φ_0.

3A.5. The depth-of-calls complexity $C^p(M)$. Fix again a Φ-structure $\mathbf{A}$, a subset $\Phi_0 \subseteq \Phi$ of the vocabulary and a Φ-program E. The *depth of calls to* Φ_0

$$C^p_{\Phi_0}(M) = C^p_{\Phi_0}(\mathbf{A}, E, M)$$

of any $M \in \mathrm{Conv}(\mathbf{A}, E)$ is defined by the following recursion on $D(M)$:

(C^p1) $C^p_{\Phi_0}(\mathrm{tt}) = C^p_{\Phi_0}(\mathrm{ff}) = C^p_{\Phi_0}(x) = 0 \quad (x \in A)$.

(C^p2) If $M \equiv \phi(M_1, \dots, M_m)$, then

$$C^p(M) = \begin{cases} \max\{C^p_{\Phi_0}(M_1), \dots, C^p_{\Phi_0}(M_m)\} + 1, & \text{if } \phi \in \Phi_0, \\ \max\{C^p_{\Phi_0}(M_1), \dots, C^p_{\Phi_0}(M_m)\}, & \text{otherwise.} \end{cases}$$

(C^p3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$C^p(M) = \begin{cases} \max\{C^p_{\Phi_0}(M_0), C^p_{\Phi_0}(M_1)\}, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{C^p_{\Phi_0}(M_0),_{\Phi_0} C^p(M_2)\}, & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(C^p4) If $M \equiv \mathsf{p}(M_1, \dots, M_m)$ of E with p a recursive variable of E, then

$$C^p_{\Phi_0}(M) = \max\{C^p_{\Phi_0}(M_1), \dots, C^p_{\Phi_0}(M_m)\} + C^p_{\Phi_0}(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$$

The depth of calls to Φ_0 of E in $\mathbf{A}$ is that of its head term,

($c^p_{\Phi_0}$) $$c^p_{\Phi_0}(\vec{x}) = c^p_{\Phi_0}(\mathbf{A}, E, \vec{x}) = C^p_{\Phi_0}(\mathbf{A}, E, E_0(\vec{x}, \mathsf{p}_1, \dots, \mathsf{p}_K)),$$

and we also skip the subscript Φ when $\Phi_0 = \Phi$, as with the sequential calls complexity,

(C^p, c^p) $$C^s(M) = C^s_\Phi(M), \quad c^p(\vec{x}) = c^p_\Phi(\vec{x}).$$

Intuitively, the number $C^p(M)$ counts *the maximal number of calls to the primitives that must be executed in sequence* in any computation of $\overline{M}$ by E. It is more directly relevant to parallel implementations—which is why we will sometimes call it the *parallel calls complexity.* It is not as easy to read it from $\mathcal{T}(M)$, however, which assumes not only parallelism but (potentially infinite) non-determinism. In the key recursive calls $\mathsf{p}(M_1, \dots, M_n)$, for example, we put the children on the same level,

$$M_1, \dots, M_n, E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n)$$

so that the depth of the tree is one more than the maximum of the depths of the trees for these terms; but $\overline{M}_1, \dots, \overline{M}_n$ must be computed *before* the computation of the rightmost child is started, which is why we set

$$\begin{aligned} &C^p(\mathsf{p}(M_1, \dots, M_n)) \\ &\qquad = \max\{C^p(M_1), \dots, C^p(M_n)\} + C^p(E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_n)). \end{aligned}$$

The same disconnect between the tree picture and the depth-of-calls in implementations comes up in the parallel logical complexity $L^p(M)$, of course.

The complexity measure $c^p(\vec{x})$ is majorized by all natural complexity measures of all reasonable implementations of recursive programs, and so lower bound results about it have wide applicability. Most of the lower bounds for problems in arithmetic we will derive are for a complexity measure somewhat smaller than $c^p(\vec{x})$.

Problems for Section 3A

The first problem is a more detailed version of Proposition 2B.1, which relates the time complexity of an iterator with the sequential calls-complexity of the associated tail recursion.

x3A.1. **Problem.** For each iterator $\mathfrak{i}$ and the associated recursive program $E \equiv E_{\mathfrak{i}}$ on $\mathbf{A} = \mathbf{A}_{\mathfrak{i}}$ and for all $x \in X$,

$$\bar{\mathfrak{i}}(x) = \overline{p}^{\mathbf{A}}_E(x),$$
$$\mathrm{Time}_{\mathfrak{i}}(x) = c^s_E(\mathbf{A}, x) \quad (\bar{\mathfrak{i}}(x)\downarrow).$$

This exact equality of the time complexity $\mathrm{Time}_{\mathfrak{i}}(x)$ with the sequential complexity $c^s_E(x)$ of the associated recursive program is due partly to some choices we made in defining $\mathrm{Time}_{\mathfrak{i}}(x)$—we could, for example, not "charge" for the calls to input(x) and output(s) and end up with a time complexity two units smaller. The precise definitions of time complexity for specific computation models frequently reflect various implementation concerns and we cannot expect a result as neat as this Lemma. It is always the case, however, that with each computation model $\mathfrak{c}$ there is a natural associated structure $\mathbf{A}_{\mathfrak{c}}$ whose primitives are the primitives of the model—not always explicitly identified; and an associated recursive program $E \equiv E_{\mathfrak{c}}$ on $\mathbf{A}_{\mathfrak{c}}$ so that

$$\mathrm{Time}_{\mathfrak{c}}(x) = \Theta(c^s_E(x)),$$

i.e., these two complexity measures are (essentially) linearly related. The same is true of all the other, natural complexity measures associated with computation models, which are similarly related to one or another of the measures we introduced in this section (and some more we will introduce later). We will not go into results of this type here.

x3A.2. **Problem.** Prove that the following are equivalent for any term $M \in \mathrm{Conv}(\mathbf{A}, E)$:

(i) $C^p(M) = 0$.

(ii) $C^s(M) = 0$.

(iii) The value $\overline{M}$ is independent of the primitives of $\mathbf{A}$, i.e., for any Φ-structure $\mathbf{A}' = (A, \mathbf{\Phi}')$ with the same universe
$$\mathrm{den}(\mathbf{A}, E, M) = \mathrm{den}(\mathbf{A}', E, M).$$
(iv) There are no Φ-nodes in the computation tree $\mathcal{T}(M)$.

We will sometimes appeal silently to this simple observation to simplify formulas, for example by dividing by $c^p_E(\vec{x})$ or using it in the form $c^p_E(\vec{x}) \geq 1$ when the value $\overline{p}^{\mathbf{A}}_E(\vec{x})$ obviously depends on the primitives of $\mathbf{A}$.

x3A.3. **Problem.** Compute (up to a multiplicative constant) $c^p_E(x, y)$ for the program defined (informally) in Problem x1B.1.

x3A.4. **Problem.** Compute (up to a multiplicative constant) $c^p_E(x, y)$ for the program defined (informally) in Problem x1B.2.

x3A.5. **Problem.** Compute (up to a multiplicative constant) $c^p_E(x, y)$ for the program in Problem x1B.3.

x3A.6. **Problem.** Fix a Φ-structure $\mathbf{A}$ and a Φ-program E of total arity $\ell = 1$. Prove that for every $M \in \mathrm{Conv}(\mathbf{A}, E)$,
$$L^s(M) \leq 2^{L^p(M)} - 1 < 2^{L^p(M)}.$$

x3A.7*. **Problem.** Give a direct proof, by induction on $D(M)$ of (c) in Theorem 3A.3, that
$$L^s(M) \leq (\ell + 1)^{L^p(M)}.$$
Hint: You may assume $\ell \geq 2$, since Problem x3A.6 gives the result when $\ell = 1$.

x3A.8. **Problem.** Consider the program E with the single equation
$$p(x) = \text{if } (x = 0) \text{ then } 0 \text{ else } p(\mathrm{Pd}(x)) + p(\mathrm{Pd}(x))$$
in the structure $(\mathbb{N}, 0, 1, \mathrm{Pd}, +, \mathrm{eq}_0)$. Determine the function $\overline{p}_E(x)$ computed by this program, and verify that that for all sufficiently large x,
$$l^s_E(x) \geq 2^{l^p_E(x)}, \quad c^s_E(x) \geq 2^{c^p_E(x)}$$

A Φ_0-*node* in a computation tree $\mathcal{T}(M)$ is a node of the form form $(M, L_1, \dots, L_k)$ in which $L_k \equiv \phi(N_1, \dots, N_n)$ for some $\phi \in \Phi_0$.

x3A.9. **Problem** (Open, vague)**.** Is there a conceptually simple and technically useful way to read $C^p_{\Phi_0}(\mathbf{A}, E, M)$ from the tree $\mathcal{T}(M)$ or the computation of the recursive machine for $\mathbf{A}$ which starts with M : , similar to the characterization of $C^s_{\Phi_0}(\mathbf{A}, E, M)$ in the next two problems?

x3A.10. **Problem.** Prove that $C^s_{\Phi_0}(\mathbf{A}, E, M)$ is the number of Φ_0-*nodes* in $\mathcal{T}(M)$.

x3A.11. **Problem.** Prove that $C^s_{\Phi_0}(\mathbf{A}, E, M)$ is the number of external calls $\vec{a}\ \phi : w_1\ \cdots\ w_n\ \vec{b}$ with $\phi \in \Phi_0$ in the computation of the recursive machine for $\mathbf{A}$ which starts with M : .

3B. Complexity inequalities

Next we derive the expected inequalities that relate these complexity measures.

3B.1. **Proposition.** *For each* Φ*-structure* $\mathbf{A}$*, each* Φ*-program* E *and each* $M \in \mathrm{Conv}(\mathbf{A}, E)$*:*

$$\begin{tikzcd}
 & C^s(M) & & (\ell+1)^{L^p(M)} \\
C^p(M) \arrow[ur, phantom, sloped, "\leq"] & & L^s(M) \arrow[ul, phantom, sloped, "\geq"] \arrow[ur, phantom, sloped, "\leq"] & \\
 & L^p(M) \arrow[ul, phantom, sloped, "\geq"] \arrow[ur, phantom, sloped, "\leq"] & &
\end{tikzcd}$$

and, in particular, for all $\vec{x}$ *such that* $\overline{p}_E(\vec{x})\downarrow$*,*

$$\begin{tikzcd}
 & c^s(\vec{x}) & & (\ell+1)^{l^p(\vec{x})} \\
c^p(\vec{x}) \arrow[ur, phantom, sloped, "\leq"] & & l^s(\vec{x}). \arrow[ul, phantom, sloped, "\geq"] \arrow[ur, phantom, sloped, "\leq"] & \\
 & l^p(\vec{x}) \arrow[ul, phantom, sloped, "\geq"] \arrow[ur, phantom, sloped, "\leq"] & &
\end{tikzcd}$$

Proof. The four inequalities on the left are very easy to check by induction on $D(M)$, and the last one on the right is Theorem 3A.3. ⊣

Together with Problem x3A.8, Theorem 3A.3 gives the expected relation between the sequential and parallel logical complexities: $l^s_E(\vec{x})$ is bounded by an exponential function of $l^p_E(x)$, and in some cases it attains this rate of growth.

What is less obvious is that for suitable constants K_s, K_p (which depend only on the program E),

(3B-1) (a) $l^s_E(\vec{x}) \leq K_s + K_s c^s_E(\vec{x})$, (b) $l^p_E(\vec{x}) \leq K_p + K_p c^p_E(\vec{x})$,

i.e., in both the sequential and the parallel measures, counting the logical steps in addition to the calls to the primitives produces at most a linear increase in complexity. From the point of view of deriving lower bounds, the significance of these inequalities becomes evident if we reverse them:

(3B-2) (a) $c^s_E(\vec{x}) \geq \dfrac{1}{K_s}(l^s_E(\vec{x}) - K_s)$, (b) $c^p_E(\vec{x}) \geq \dfrac{1}{K_p}(l^p_E(\vec{x}) - K_p)$.

Here (a) means that the high sequential computational complexity of a particular relation $R \subseteq A^n$ from specified primitives is not caused by the large number of logical operations that we must execute to decide $R(\vec{x})$—i.e., (literally) by the "computational complexity" of R—but is due to the large number of calls to the primitives that are necessary to decide $R(\vec{x})$, at least up to a linear factor. Ditto for the parallel computational complexity $l^p_E(\vec{x})$ and its "calls-counting" counterpart $c^p_E(\vec{x})$. This explains why lower bounds results are most often proved by counting calls to the primitives, and incidentally emphasizes the importance of identifying *all* the (non-logical) primitives that are used by an algorithm which decides a particular relation.

The proofs of these inequalities require some new ideas due to Anush Tserunyan.[11]

We fix a Φ-structure $\mathbf{A}$ and a recursive program E with recursive variables $\mathsf{p}_1, \dots, \mathsf{p}_K$ and total arity $\ell = \ell(E) \geq 1$. We can insure that

$$\text{the number of free and bound variables in } E \leq \ell$$

by making innocuous alphabetic changes to the bound variables of E. It follows that if

$$t = t(E) = \text{the number of distinct subterms of the terms in } E,$$
$$H = H(E) = t\ell^{\ell},$$

then H is an overall upper bound to the number of terms that can be constructed by a single assignment of parameters to the variables in all the subterms of E.

We start with some preliminary estimates which are needed for the proofs of both inequalities in (3B-1).

3B.2. **Lemma.** *If $M \in \mathrm{Conv}(\mathbf{A}, E)$ and $C^p(M) = 0$, then the value $\overline{M}$ occurs in M.*

Proof is by an easy induction on $D(M)$. $\dashv$

Notice that the Lemma applies even when $\overline{M}$ is tt or ff, which we do not formally count as "parameters".

3B.3. **Lemma.** *Suppose $M \in \mathrm{Conv}(\mathbf{A}, E)$.*

(a) *If $(M_1, \dots, M_k) \in \mathcal{T}(M)$, then $M_k \in \mathrm{Conv}(\mathbf{A}, E)$.*

(b) *If $(M_1, \dots, M_k) \in \mathcal{T}(M)$ and the parameters in every M_i occur in M, then $k \leq H$.*

[11]The results in the remainder of this section are due to Tserunyan and are part of her 2013 Ph.D. Thesis.

Proof. (a) is immediate by the construction of $\mathcal{T}(M)$.

(b) Suppose $x_1, \dots, x_m$ is a list of the parameters in $M_1 \equiv M$, so $m \le \ell$. Each M_i is an $(\mathbf{A}, E)$-term whose parameters are among $x_1, \dots, x_m$, and there are at most H distinct such terms; so if $k > H$, then $M_i \equiv M_j$ for some $1 \le i < j \le k$, and then $\overline{M}_1 \uparrow$. ⊣

It is clear from the construction of the computation tree that new parameters enter the tree only by Case ($\mathcal{T}$4), in the term $E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$, and then only if some $\overline{M}_i$ does not occur in the parent node. Isolating and counting these critical nodes is the main new tool we need.

Splitting. A term $M \in \mathrm{Conv}(\mathbf{A}, E)$ is *splitting*, if $M \equiv \mathsf{p}(M_1, \dots, M_n)$ with a recursive variable p of E and

$$\max\nolimits_{1 \le j \le n} C^p(M_j) > 0, \qquad C^p(E_i(\overline{M}_1, \dots, \overline{M}_n)) > 0.$$

By Problem x3A.2, these conditions are equivalent to their version with C^s rather than C^p.

3B.4. **Lemma.** *If $(M_1, \dots, M_k) \in \mathcal{T}(M) = \mathcal{T}(M_1)$ and no M_i is splitting, then $k \le 2H$.*

Proof. If the parameters in every M_i occur in $M_1 \equiv M$, then we apply (b) of Lemma 3B.3. Otherwise, let i be least such that M_{i+1} has parameters that do not occur in M_1. Now $i \le H$ by Lemma 3B.3 again, and by the definition of $\mathcal{T}(M)$,

$$M_i \equiv \mathsf{p}(N_1, \dots, N_n), \text{ and } M_{i+1} \equiv E_{\mathsf{p}}(\overline{N}_1, \dots, \overline{N}_n).$$

Moreover, $\max\{C^p(N_j) : 1 \le j \le n\} > 0$ since otherwise, each $\overline{N}_j$ occurs in N_j and hence $\overline{N}_j \in M_i$ by Lemma 3B.2, contradicting the choice of i. But M_i is not splitting, so $C^p(M_{i+1}) = C^p(E_{\mathsf{p}}(\overline{N}_1, \dots, \overline{N}_n)) = 0$. Hence for all $j \ge i+1$, $C^p(M_j) = 0$ and then by Lemma 3B.2 again, all parameters in M_j occur in M_{i+1}, and the length of $(M_{i+1}, \dots, M_k)$ is $\le H$; which means that $k = i + (k - i) \le H + H = 2H$. ⊣

Let

$$v(M) = \Big\{(M_1, \dots, M_n) \in T(M) : \forall i, M_i \text{ is not splitting}\Big\}.$$

This is the empty set if M is splitting and a subtree of $\mathcal{T}(M)$ if it is not. By Lemma 3B.4 and the fact that $\mathrm{degree}(\mathcal{T}(M)) \le (\ell + 1)$,

(3B-3) $$|v(M)| \le V, \text{ where } V = (\ell + 1)^{2H}.$$

3B.5. **Lemma.** *If $C^p(M) = 0$, then $L^s(M) \le |v(M)|$.*

Proof. If $C^p(M) = 0$, then there are no splitting terms below M, and so $v(M) = \mathcal{T}(M)$ and the inequality follows from (a) of Theorem 3A.3. ⊣

For the proof of (a) in (3B-1), we need the *sequential splitting complexity* of a closed, convergent $(\mathbf{A}, E)$-term:

(3B-4) $$F^s(M) = \text{the number of splitting nodes in } \mathcal{T}(M),$$

where $(M_0, \dots, M_k) \in \mathcal{T}(M_0)$ is splitting if M_k is splitting. This satisfies some obvious recursive conditions which we will introduce and use in the proof of the next lemma. It is clear, however, that

$$C^p(M) = 0 \Longrightarrow F^s(M) = 0;$$

because if $C^p(M) = 0$, then $C^p(N) = 0$ for every N in $\mathcal{T}(M)$ and so no such term can be splitting.

3B.6. **Lemma.** *For every $M \in \mathrm{Conv}(\mathbf{A}, E)$,*

$$F^s(M) \le C^s(M) \dot{-} 1.$$

PROOF is by induction on $D(M)$, as usual, and it is trivial in all cases except when M is splitting. If $M \equiv \mathsf{p}(M_1, \dots, M_m)$ is splitting, let $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$ and compute:

$$\begin{aligned} F^s(M) &= F^s(M_1) + \cdots + F^s(M_m) + F^s(M_{m+1}) + 1 \\ &\le (C^s(M_1) \dot{-} 1) + \cdots + (C^s(M_m) \dot{-} 1) + (C^s(M_{m+1}) \dot{-} 1) + 1 \\ &\le C^s(M_1) + \cdots + C^s(M_m) - 1 + C^s(M_{m+1}) - 1 + 1 \\ &= C^s(M) - 1. \end{aligned}$$

The only thing we used here is that if M is splitting, then $C^s(M_i) \ge C^p(M_i) > 0$ for at least one i, and similarly $C^s(M_{m+1}) > 0$. ⊣

3B.7. **Lemma.** *If $M \in \mathrm{Conv}(\mathbf{A}, E)$, then there is a* (possibly empty) *sequence of splitting terms $N_0, \dots, N_{k-1}$ in $\mathcal{T}(M)$ such that*

(3B-5) $$F^s(M) = \textstyle\sum_{i<k} F^s(N_i), \qquad L^s(M) \le \textstyle\sum_{i<k} L^s(N_i) + |v(M)|.$$

PROOF. If M is splitting, we take just one $N_0 \equiv M$, and if there are no splitting terms in $\mathcal{T}(M)$ we set $k = 0$ and understand $\sum_{i<k} L^s(N_i) = 0$, so that $\mathcal{T}(M) = v(M)$ by definition and (3B-5) follows from (a) of Theorem 3A.3. The lemma is proved in the general case by induction on $D(M)$, and the argument is trivial in most of the cases. We consider only the case of a non-splitting recursive call

$$M \equiv \mathsf{p}(M_1, \dots, M_m).$$

Set $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$. The induction hypothesis gives us (a possibly empty) sequence $N_0^i, \dots, N_{k_i}^i$ of splitting terms in each $\mathcal{T}(M_i)$ such that, to begin with,

$$\begin{aligned} L^s(M) &= L^s(M_1) + \cdots + L^s(M_m) + L^s(M_{m+1}) + 1 \\ &\le \textstyle\sum_{j<k_1} L^s(N^1_j) + |v(M_1)| + \cdots + \sum_{j<k_m} L^s(N^m_j) + |v(M_m)| \\ &\qquad\qquad + \textstyle\sum_{j<k_{m+1}} L^s(N^{m+1}_j) + |v(M_{m+1})| + 1 \\ &\le \textstyle\sum_{1\le i\le m+1, j<k_i} L^s(N^i_j) + |v(M_1)| + \cdots + |v(M_{m+1})| + 1. \end{aligned}$$

Now $v(M) = \bigcup_{i=1,\dots,m+1} v(M_i) \cup \{M\}$ because M is not splitting, and so

$$|v(M_1)| + \cdots + |v(M_{m+1})| + 1 = |v(M)|.$$

Moreover, $F^s(M_i) = \sum_{j<k_i} N^i_j$ by the induction hypothesis, and so

$$F^s(M) = F^s(M_1) + \cdots + F^s(M_{m+1}) = \textstyle\sum_{1\le i\le m+1, j<k_i} F^s(N^i_j)$$

as required, again because M is not splitting. ⊣

3B.8. **Theorem** (Tserunyan). *For every Φ-structure $\mathbf{A}$, every Φ-program E and every $M \in \mathrm{Conv}(\mathbf{A}, E)$, if V is the constant defined in* (3B-3)*, then:*

(a) *If M is splitting, then $L^s(M) \le ((\ell+1)V+1)F^s(M)$.*

(b) *If M is not splitting, then $L^s(M) \le ((\ell+1)V+1)F^s(M) + V$.*

It follows that $L^s(M) \le V + ((\ell+1)V+1)C^s(M)$, and so

$$l^s_E(\vec{x}) \le K_s + K_s c^s_E(\vec{x}) \quad (\overline{p}_E(\vec{x})\downarrow) \tag{3B-6}$$

with $K_s = (\ell+1)V+1$, a constant which depends only on the program E.

Proof. The main result (3B-6) follows from (a) and (b) by taking $M \equiv E_0(\vec{x})$ and appealing to Lemma 3B.6.

We prove (a) and (b) together by induction on $D(M)$, noting that (b) is a weaker inequality than (a) and so we can use it whether M is splitting or not when we invoke the induction hypothesis.

Case 1, $M \equiv \mathsf{p}(M_1,\dots,M_m)$ is splitting. Set $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1,\dots,\overline{M}_m)$ as above and compute:

$$\begin{aligned} L^s(M) &= L^s(M_1) + \cdots + L^s(M_{m+1}) + 1 \\ &\le ((\ell+1)V+1)[F^s(M_1) + \cdots + F^s(M_{m+1})] + (\ell+1)V + 1 \\ &= ((\ell+1)V+1)[F^s(M_1) + \cdots + F^s(M_{m+1}) + 1] \\ &= ((\ell+1)V+1)F^s(M), \end{aligned}$$

because $F^s(M) = F^s(M_1) + \cdots + F^s(M_{m+1}) + 1$ for splitting M.

Case 2, M is not splitting. Choose splitting terms $N_0,\dots,N_{k-1}$ by Lemma 3B.7 so that

$$F^s(M) = \textstyle\sum_{i<k} F^s(N_i), \quad L^s(M) \le \sum_{i<k} L^s(N_i) + |v(M)|$$

and compute using the result from Case 1:

$$\begin{aligned} L^s(M) &\le \textstyle\sum_{i<k} L^s(N_i) + |v(M)| \\ &\le ((\ell+1)V+1)\textstyle\sum_{i<k} F^s(N_i) + V = ((\ell+1)V+1)F^s(M) + V \end{aligned}$$

as required. $\dashv$

We now turn to the proof of (b) in (3B-1), and for this we need a count $F^p(M)$ of the splitting terms which parallels the way in which $C^p(M)$ counts "the depth" of calls to the primitives. This is easiest to define by induction on $D(M)$:

(F^p1) $F^p(\mathrm{tt}) = F^p(\mathrm{ff}) = F^p(x) = 0 \quad (x \in A)$.

(F^p2) If $M \equiv \phi(M_1, \dots, M_m)$, then

$$F^p(M) = \max\{F^p(M_1), \dots, F^p(M_m)\}.$$

(F^p3) If $M \equiv$ if M_0 then M_1 else M_2, then

$$F^p(M) = \begin{cases} \max\{F^p(M_0), F^p(M_1)\}, & \text{if } \overline{M}_0 = \mathrm{tt}, \\ \max\{F^p(M_0), F^p(M_2)\}, & \text{if } \overline{M}_0 = \mathrm{ff}. \end{cases}$$

(F^p4) If $M \equiv \mathsf{p}(M_1, \dots, M_m)$ of E, let $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$ and set

$$F^p(M) = \begin{cases} \max\{F^p(M_1), \dots, F^p(M_m)\} + F^p(M_{m+1}), & \text{if } M \text{ is not splitting,} \\ \max\{F^p(M_1), \dots, F^p(M_m)\} + F^p(M_{m+1}) + 1, & \text{if } M \text{ is splitting.} \end{cases}$$

Notice that if M is not splitting, then

$$F^p(M) = \max\{F^p(M_1), \dots, F^p(M_m), F^p(M_{m+1})\},$$

since one of $\max\{F^p(M_1), \dots, F^p(M_m)\}$ and $F^p(M_{m+1})$ is 0, so that the sum of these two numbers is the same as their maximum.

With this splitting complexity, the proof of (b) in (3B-1) is only a minor modification (a parallel version) of the proof of Theorem 3B.8.

3B.9. **Lemma.** *For every $M \in \mathrm{Conv}(\mathbf{A}, E)$,*

$$F^p(M) \le C^p(M) \dot{-} 1.$$

Proof is by induction on $D(M)$ and it is again trivial in all cases except when M is splitting. In this case, if $M \equiv \mathsf{p}(M_1, \dots, M_m)$ and we set $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$, then

$$\begin{aligned} F^p(M) &= \max\{F^p(M_1), \dots + F^p(M_m)\} + F^p(M_{m+1}) + 1 \\ &\le \max\{(C^p(M_1) \dot{-} 1), \dots + (C^p(M_m) \dot{-} 1)\} + C^p(M_{m+1}) \dot{-} 1 + 1 \\ &\le \max\{C^p(M_1), \dots, C^p(M_m)\} - 1 + C^p(M_{m+1}) - 1 + 1 \\ &= C^p(M) \dot{-} 1. \end{aligned}$$

As in the proof of Lemma 3B.6, the only thing we use here is that if M is splitting, then $C^p(M_i) > 0$ for at least one i, and similarly $C^p(M_{m+1}) > 0$.⊣

3B.10. **Lemma.** *If $M \in \mathrm{Conv}(\mathbf{A}, E)$, then there is a term N in $\mathcal{T}(M)$ which is either a leaf or splitting and such that*

$$L^p(M) \leq L^p(N) + |v(M)|. \tag{3B-7}$$

Proof is by induction on $D(M)$, as usual, and the result is trivial if M is a leaf or splitting, taking $N \equiv M$.

If $M \equiv \phi(M_1, \dots, M_m)$, then $L^p(M) = L^p(M_i) + 1$ for some i, and the induction hypothesis gives us a leaf or splitting term N in $\mathcal{T}(M_i)$ such that

$$L^p(M_i) \leq L^p(N) + |v(M_i)|.$$

It follows that

$$L^p(M) = L^p(M_i) + 1 \leq L^p(N) + |v(M_i)| + 1 \leq L^p(N) + |v(M)|$$

since M is not splitting and so $v(M) = \bigcup_{j=1,\dots M} v(M_j) \cup \{M\}$.

The argument is similar for conditional terms.

If M is a non-splitting recursive call

$$M \equiv \mathsf{p}(M_1, \dots, M_m),$$

set again $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$ and choose i such that $L^p(M_i) = \max\{L^p(M_1), \dots, L^p(M_m)\}$. If $C^p(M_i) = 0$, then then $L^p(M_i) \leq |v(M_i)|$ by Lemma 3B.5, and if we choose $N \in \mathcal{T}(M_{m+1})$ by the inductive hypothesis so that $L^p(M_{m+1}) \leq L^p(N) + |v(M_{m+1}|$, then

$$\begin{aligned} L^p(M) &= L^p(M_i) + L^p(M_{m+1}) + 1 \\ &\leq |v(M_i)| + L^p(N) + |v(M_{m+1})| + 1 = L^p(N) + |v(M)|. \end{aligned}$$

If $C^p(M_i) > 0$, then $C^p(M_{m+1}) = 0$, since M is not splitting, and we can repeat the argument with M_i and M_{m+1} interchanged. ⊣

3B.11. **Theorem** (Tserunyan). *For every Φ-structure $\mathbf{A}$, every Φ-program E and every $M \in \mathrm{Conv}(\mathbf{A}, E)$, if V is the constant defined in* (3B-3)*, then:*

(a) *If M is splitting, then $L^p(M) \leq (2V + 1)F^p(M)$.*

(b) *If M is not splitting, then $L^p(M) \leq (2V + 1)F^p(M) + V$.*

It follows that $L^p(M) \leq V + (2V + 1)C^p(M)$ and

$$l^p_E(\vec{x}) \leq K_p + K_p c^p_E(\vec{x}) \qquad (\overline{p}_E(\vec{x})\downarrow) \tag{3B-8}$$

with $K_p = 2V + 1$, which depends only on the program E.

Proof is a minor adaptation of the proof of Theorem 3B.8, with (3B-8) following from (a) and (b) by appealing to Lemma 3B.9. We prove (a) and (b) together by induction on $D(M)$.

Case 1, $M \equiv \mathsf{p}(M_1, \dots, M_m)$ is splitting. Set $M_{m+1} \equiv E_{\mathsf{p}}(\overline{M}_1, \dots, \overline{M}_m)$ as above and compute:

$$\begin{aligned}
L^p(M) &= \max_{1\le i\le m} L^p(M_i) + L^p(M_{m+1}) + 1\\
&\le (2V+1)\max_{1\le i\le m} F^p(M_i) + V + (2V+1)F^p(M_{m+1}) + V + 1\\
&= (2V+1)\Big(\max_{1\le i\le m} F^p(M_i) + F^p(M_{m+1})\Big) + 2V + 1\\
&= (2V+1)\Big(\max_{1\le i\le m} F^p(M_i) + F^p(M_{m+1}) + 1\Big) = (2V+1)F^p(M),
\end{aligned}$$

because $F^p(M) = \max_{1\le i\le m} F^p(M_i) + F^p(M_{m+1}) + 1$ for splitting M.

Case 2, M is not splitting. There is nothing to prove if M is a leaf. If it is not, choose a leaf or splitting term N in $\mathcal{T}(M)$ by Lemma 3B.10 such that $L^p(M) \le L^p(N) + |v(M)|$. If N is a leaf, then $L^p(N) = 0$ and so $L^p(M) \le |v(M)| \le V \le (2V+1)F(M) + V$ by (3B-3). If N is splitting, then the induction hypothesis applies to it since it is not M and hence $D(N) < D(M)$, and we have

$$L^p(M) \le L^p(N) + |v(M)| \le (2V+1)F^p(N) + V$$

by (3B-3) again, as required. ⊣

3B.12. **Corollary.** *For every Φ-program E, there is a constant K such that for every Φ-structure $\mathbf{A}$,*

$$\text{(3B-9)} \qquad c^s(\vec{x}) \le K^{c^p(\vec{x})} \qquad (\overline{p}_E^{\mathbf{A}}(\vec{x})\downarrow).$$

Proof. Notice that (3B-9) is true for any $K > 0$ if $c^p(\vec{x}) = 0$, because in that case $c^s(\vec{x}) = 0$ also by Problem x3A.2. So we assume that $\overline{p}(\vec{x})\downarrow$ and $c^p(\vec{x}) > 0$ and compute:

$$\begin{aligned}
c^s(\vec{x}) \le l^s(\vec{x}) &\le (\ell+1)^{l^p(\vec{x})} && \text{(Theorem 3A.3)}\\
&\le (\ell+1)^{K_p + K_p c^p(\vec{x})} && \text{(Theorem 3B.11)}\\
&= (\ell+1)^{2K_p c^p(\vec{x})} \le \Big((\ell+1)^{2K_p}\Big)^{c^p(\vec{x})}.
\end{aligned}$$

⊣

3B.13. **Corollary.** *For every Φ-program E of total arity ℓ, there is a constant K_p such that for every Φ-structure $\mathbf{A}$,*

$$\text{(3B-10)} \qquad l^s(\vec{x}) \le (\ell+1)^{K_p(c^p(\vec{x})+1)} \quad (\overline{p}(\vec{x})\downarrow).$$

Proof is immediate, from Theorems 3B.11 and 3A.3. ⊣

This last inequality bounds $l^s(\vec{x})$, the largest of the basic complexity functions associated with a recursive program by an exponential of the smallest, $c^p(\vec{x})$. One of its consequences is that we can talk of *programs of bounded complexity* without ambiguity, since

$$(\exists k)(\forall \vec{x})[\overline{p}^{\mathbf{A}}_E(\vec{x})\downarrow \Longrightarrow l^s_E(\vec{x}) \le k]$$
$$\iff (\exists k)(\forall \vec{x})[\overline{p}^{\mathbf{A}}_E(\vec{x})\downarrow \Longrightarrow c^p_E(\vec{x}) \le k].$$

3C. Recursive vs. explicit definability

The *length* of a pure Φ-term E is defined recursively by the clauses

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

and it is easy to check that if we think of $E(\vec{\mathsf{x}})$ as a program, then for any Φ-algebra $\mathbf{A}$,

$$\text{(3C-1)} \qquad L^s(E(\vec{x})) \le \text{length}(E(\vec{\mathsf{x}})) \quad (\vec{x} \in A, \text{den}(\mathbf{A}, E(\vec{x}))\downarrow),$$

see Problem x3C.1. Together with the results in the preceding section, this implies that if we derive a non-constant lower bound for $l^s_E(\vec{x})$ for every program E which computes $f : A^n \rightharpoonup A_s$, then no Φ-term defines f in $\mathbf{A}$. We establish here the converse of this proposition for structures which satisfy a mild "richness" condition.

3C.1. **Theorem.** *Suppose Φ has a relation symbol R of arity $k > 0$ and $\mathbf{A}$ is a Φ-structure such that $\mathsf{R}^{\mathbf{A}} : A^k \to \{\mathrm{tt}, \mathrm{ff}\}$ is total. Then: for any $f : A^n \rightharpoonup A_s$ and $S \subseteq \{\vec{x} : f(\vec{x})\downarrow\}$ the following are equivalent:*

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

$$\vec{x} \in S \Longrightarrow f(\vec{x}) = \text{den}(\mathbf{A}, M(\vec{x})).$$

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

$$f(\vec{x}) = \overline{p}^{\mathbf{A}}_E(\vec{x}) \text{ and } l^s_E(\vec{x}) \le k.$$

In particular, a function $f : A^n \to A$ is explicit in $\mathbf{A}$ if and only if it is computed in $\mathbf{A}$ by a Φ-program with bounded complexity.

Proof. (a) $\Longrightarrow$ (b) follows immediately from (3C-1). For the converse implication we need some preliminary work. We will assume for simplicity that the given relation symbol R is unary. (If it is not, simply replace $\mathbf{R}(\mathsf{x})$ by $\mathsf{R}(\mathsf{x}, \dots, \mathsf{x})$ in the construction.)

It is convenient for this proof to add to the language nullary function constants $\emptyset_{\mathtt{a}}, \emptyset_{\mathtt{boole}}$ which denote the nullary, totally undefined partial functions of sort `a` and `boole`. The semantics of these (extended) $(\Phi, \emptyset)$-*terms* in a Φ-structure $\mathbf{A}$ 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 is a closed $(\Phi, \emptyset)$-term such that $\mathrm{den}(M)\downarrow$ *and N is any closed $(\Phi, \emptyset)$-term of sort s, then* $\mathrm{den}(M\{\emptyset_s :\equiv N\}) = \mathrm{den}(M(N))$.

Next we associate with each Φ-program E, each pure $\mathrm{voc}(E)$-term

$$M \equiv M(\vec{\mathsf{x}}, \vec{\mathsf{p}})$$

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

1. If M is a variable or tt or ff, then $[M]^{(k)} \equiv M$.
2. $[M]^{(0)}(\vec{\mathsf{x}}) \equiv M(\vec{\mathsf{x}}, \vec{\emptyset})$, where the substitution $\mathsf{p}_i \mapsto \emptyset$ means that every subterm of $M(\vec{\mathsf{x}}, \vec{\mathsf{p}})$ of the form $\mathsf{p}_i(M_1, \dots, M_n)$ is replaced by $\emptyset_s$ with $s = \mathrm{sort}(\mathsf{p}_i)$.
3. If $M \equiv \phi(M_1, \dots, M_n)$, then $[M]^{(k+1)} \equiv \phi([M_1]^{(k+1)}, \dots, [M_n]^{(k+1)})$.
4. If $M \equiv$ if M_0 then M_1 else M_2, then
$$[M]^{(k+1)} \equiv \text{if } [M_0]^{(k+1)} \text{ then } [M_1]^{(k+1)} \text{ else } [M_2]^{(k+1)}.$$
5. If $M \equiv \mathsf{p}_i(M_1, \dots, M_n)$, then
$$[M]^{(k+1)} \equiv \text{if } ([M_1]^{(k)}\downarrow \ \& \ \cdots \ \& \ [M_n]^{(k)}\downarrow) \\ \text{then } [E_i]^{(k)}([M_1]^{(k)}, \cdots, [M_n]^{(k)}) \text{ else } \emptyset_s,$$
where $s = \mathrm{sort}(\mathsf{p}_i)$.

Here $N_1\downarrow \ \& \ \cdots \ \& \ N_n\downarrow$ is defined by the following recursion on n:

$$N_1\downarrow \ :\equiv \ \text{if } \mathsf{R}(N_1) \text{ then tt else tt},$$

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

The definition of $[M]^{(k)}$ depends on the program E (so that we should properly write $[M]^{(k)}_E$), but not on any specific Φ-structure $\mathbf{A}$.

Lemma 2. For each Φ-program E, each $\mathrm{voc}(E)$*-term $M(\vec{\mathsf{x}})$, 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 $L^s(M(\vec{x})) \leq k$, 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 length(M).

Basis, $k = 0$. The result is obvious when M is tt, ff or a variable, and there is nothing to prove if $M \equiv \mathsf{p}_i(M_1, \dots , M_n)$, since in that case

$$\mathrm{den}([\mathsf{p}_i(M_1, \dots , M_n)]^{(0)}(\vec{x})) = \mathrm{den}(\emptyset_s) \uparrow .$$

For the other two cases where the hypothesis may hold, (i) follows by an easy induction on the length.

Induction Step. We use again induction on the length of M and the argument is exactly like that in the basis, except for the new case of $M \equiv \mathsf{p}_i(M_1, \dots , M_n)$ 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 $L^s(M)$. It is obvious when M is tt, ff 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

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

and by the induction hypothesis and (i)

$$\mathrm{den}([M_i]^{(k+1)}) = \mathrm{den}([M_i]^{(k)}) = \overline{M}_i,$$

so that

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

The argument is similar for conditionals, and if $M \equiv \mathsf{p}_i(M_1, \dots , M_n)$, then

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

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

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

the required $\mathrm{den}([M_i]^{(k+1)}) = \overline{M}$ now follows by (i). ⊣ (Lemma 2)

To prove that (b) $\Longrightarrow$ (a) in the Theorem from Lemma 1, let $E_0(\vec{\mathsf{x}})$ be the head of the given program E, $k = \max\{l^s_E(\vec{x}) : \vec{x} \in S\}$. To construct the required $M(\vec{\mathsf{x}})$, start with $[E_0(\vec{\mathsf{x}})]^{(k)}$ and first replace $\emptyset_{\mathtt{boole}}$ in all its occurrences by tt. If $\vec{\mathsf{x}} \equiv \mathsf{x}_1, \dots , \mathsf{x}_n$ is a non-empty tuple, replace next each $\emptyset_{\mathsf{a}}$ by x_1, so that the resulting $M(\vec{\mathsf{x}})$ has the required property by

Lemma 2. Finally, if $n = 0$ so that E_0 is a closed term, then there must be some individual constant c in the vocabulary Φ—otherwise there are no closed voc(E)-terms (other than those which can be constructed from the truth values and which cause no problem); and in this case we can replace $\emptyset_{\mathsf{a}}$ by c and appeal again to Lemma 2. $\dashv$

Problems for Section 3C

x3C.1. **Problem.** Prove (3C-1).

PART II, INTRINSIC COMPLEXITIES

CHAPTER 4

THE HOMOMORPHISM METHOD

Most of the known lower bound results in the literature are established for specific computation models and the natural complexity measures associated with them, and so any claim that they are absolute—or even that they hold for a great variety of models—must be inferred from the proofs. The results about recursive programs in van den Dries and Moschovakis [2004], [2009] are somewhat more robust: they are proved directly from the abstract definitions of complexities in Chapter 3 using basically nothing more than the *homomorphism* and *finiteness properties* of Theorem 2D.1, without reference to the recursive machine or any other implementations of recursive programs. They imply immediately lower bounds for most computation models, because of the simulations discussed briefly in Section 2B and in the comment after Problem x3A.1.

Our main aim in this Chapter is to extract from the homomorphism and finiteness properties of recursive programs a general, "algebraic" method for deriving robust lower bounds for algorithms from specified primitives. The key notions of the chapter are those of a *uniform process* and *certification* in Sections 4C and 4E and the main result is the *Homomorphism Test*, Lemma 4F.2. We will start, however, with a brief discussion of "algorithms from primitives" which motivates our choice of notions.

4A. Axioms which capture the uniformity of algorithms

Our basic intuition is that an *n-ary algorithm* α of sort $s \in \{\mathtt{a}, \mathtt{boole}\}$ *of a structure* $\mathbf{A} = (A, \mathbf{\Phi})$—or *from* $\mathbf{\Phi}$—computes (in some way) an n-ary partial function

$$\overline{\alpha} = \overline{\alpha}^{\mathbf{A}} : A^n \rightharpoonup A_s \quad (\text{with } A_{\mathtt{boole}} = \{\mathrm{tt}, \mathrm{ff}\}, A_{\mathtt{a}} = A)$$

using the primitives in $\mathbf{\Phi}$ as *oracles.* We understand this to mean that in the course of a computation of $\overline{\alpha}(\vec{x})$, the algorithm may request from the oracle for any $\phi^{\mathbf{A}}$ any particular value $\phi^{\mathbf{A}}(u_1, \dots, u_{n_\phi})$, where each u_i either is given by the input or has already been computed; and that if the oracles cooperate and respond to all requests, then this computation of $\overline{\alpha}(\vec{x})$ is completed in a finite number of steps.

The three axioms we formulate in this section capture part of this minimal understanding of how algorithms from primitives operate in the style of *abstract model theory.*

The crucial first axiom expresses the possibility that the oracles may choose not to respond to a request for $\phi^{\mathbf{A}}(u_1, \dots, u_{n_\phi})$ unless

$$u_1, \dots, u_n \in U \;\&\; \phi^{\mathbf{U}}(u_1, \dots, u_n)\downarrow$$

for some fixed substructure $\mathbf{U} \subseteq_p \mathbf{A}$: the algorithm will still compute a partial function, which simply diverges on those inputs $\vec{x}$ for which no computation of $\overline{\alpha}(\vec{x})$ by α can be executed "inside" $\mathbf{U}$ (as far as calls to the oracles are involved).

I. Locality Axiom. *An n-ary algorithm α of sort $s \in \{\mathtt{a}, \mathtt{boole}\}$ of a structure $\mathbf{A}$ assigns to each substructure $\mathbf{U} \subseteq_p \mathbf{A}$ an n-ary partial function*

$$\overline{\alpha}^{\mathbf{U}} : U^n \rightharpoonup U_s.$$

We understand this axiom constructively, i.e., we claim that the *localization operation*

$$(\mathbf{U} \mapsto \overline{\alpha}^{\mathbf{U}}) \quad (\text{where } \mathbf{U} \subseteq_p \mathbf{A} \text{ and } \overline{\alpha}^{\mathbf{U}} : U^n \rightharpoonup U_s) \tag{4A-1}$$

is induced naturally by a specification of α. We set

$$\mathbf{U} \vdash \alpha(\vec{x}) = w \iff \vec{x} \in U^n \;\&\; \overline{\alpha}^{\mathbf{U}}(\vec{x}) = w \tag{4A-2}$$

$$\mathbf{U} \vdash \alpha(\vec{x})\downarrow \iff (\exists w)[\mathbf{U} \vdash \alpha(\vec{x}) = w], \tag{4A-3}$$

and we call $\overline{\alpha}^{\mathbf{U}}$ the partial function *computed* by α in $\mathbf{U}$. We read "$\vdash$" as *proves*.

In particular, α computes in $\mathbf{A}$ the partial function $\overline{\alpha} = \overline{\alpha}^{\mathbf{A}} : A^n \rightharpoonup A_s$.

For example, if E is a non-deterministic recursive program which computes a partial function in $\mathbf{A}$, then the localization of (the algorithm specified by) E is defined by

$$\mathbf{U} \vdash \alpha_E(\vec{x}) = w \iff \overline{p}_E^{\mathbf{U}}(\vec{x}) = w \quad (\mathbf{U} \subseteq_p \mathbf{A}). \tag{4A-4}$$

II. Homomorphism Axiom. *If α is an algorithm of $\mathbf{A}$ and $\pi : \mathbf{U} \to \mathbf{V}$ is a homomorphism of one substructure of $\mathbf{A}$ into another, then*

$$\mathbf{U} \vdash \alpha(\vec{x}) = w \Longrightarrow \mathbf{V} \vdash \alpha(\pi(\vec{x})) = \pi(w) \quad (\vec{x} \in U^n).$$

In particular, by applying this to the identity embedding $\mathrm{id}_U : \mathbf{U} \rightarrowtail \mathbf{A}$,

$$\mathbf{U} \subseteq_p \mathbf{A} \Longrightarrow \overline{\alpha}^{\mathbf{U}} \sqsubseteq \overline{\alpha}^{\mathbf{A}} = \overline{\alpha}.$$

The idea here is that the oracle for each $\phi^{\mathbf{A}}$ may consistently respond to each request for $\phi^{\mathbf{U}}(\vec{u})$ by delivering $\phi^{\mathbf{V}}(\pi(\vec{u}))$. This transforms any computation of $\overline{\alpha}^{\mathbf{U}}(\vec{x})$ into one of $\overline{\alpha}^{\mathbf{V}}(\pi(\vec{x}))$, which in the end delivers the value $\pi(w) = \pi(\overline{\alpha}^{\mathbf{U}}(\vec{x}))$.

This argument is convincing for the identity embedding $\mathrm{id}_U : \mathbf{U} \rightarrowtail \mathbf{V}$. It is not quite that simple in the general case, because α may utilize in its computations complex data structures and rich primitives, e.g., stacks, queues, trees, conditionals, the introduction of higher type objects by λ-abstraction and subsequent application of these objects to suitable arguments, etc. The claim is that any homomorphism $\pi : \mathbf{U} \to \mathbf{V}$ lifts naturally to these data structures, and so the image of a convergent computation of $\overline{\alpha}^{\mathbf{U}}(\vec{x})$ is a convergent computation of $\overline{\alpha}^{\mathbf{V}}(\pi(\vec{x}))$. Put another way: if some $\pi : \mathbf{U} \to \mathbf{V}$ does not lift naturally to a mapping of the relevant computations, then α *is using essentially some hidden primitives not included in* $\mathbf{A}$ and so *it is not an algorithm from* $\{\phi^{\mathbf{A}}\}_{\phi\in\Phi}$. It is clear, however, that the Homomorphism Axiom demands something more of algorithms (and how they use oracles) than the Locality Axiom. We will discuss this issue briefly in the introduction to Section 4B.

The Homomorphism Axiom is at the heart of this approach to the derivation of lower bounds.

III. Finiteness Axiom. *If* $\overline{\alpha}(\vec{x}) = w$*, then there is a finite* $\mathbf{U} \subseteq_p \mathbf{A}$ *generated by* $\vec{x}$ *such that* $\mathbf{U} \vdash \alpha(\vec{x}) = w$.

This combines two ingredients of the basic intuition: first that in the course of a computation, the algorithm may only request of the oracles values $\phi^{\mathbf{A}}(\vec{u})$ for arguments $\vec{u}$ that it has already constructed from the input, and second, that computations are finite. A suitable $\mathbf{U}$ is then determined by putting in $\mathrm{eqdiag}(\mathbf{U})$ all the *calls made by* α *in some computation of* $\overline{\alpha}(\vec{x})$.

This axiom implies, in particular, that partial functions computed by an $\mathbf{A}$-algorithm take values in the substructure generated by the input,

$$\overline{\alpha}^{\mathbf{A}}(\vec{x}) = w \Longrightarrow w \in G_\infty(\mathbf{A}, \vec{x}) \cup \{\mathrm{tt}, \mathrm{ff}\}.$$

This is a substantial restriction, e.g., it rules out notions of "algorithm" by which the function $(x \mapsto \sqrt{|x|})$ would be computable in the real field $(\mathbb{R}, 0, 1, =, +, -, \cdot, \div)$. It has no implications for decision problems, however, when $\overline{\alpha}$ is a relation and so for all $\vec{x}$, $\overline{\alpha}(\vec{x}) \in \{\mathrm{tt}, \mathrm{ff}\}$.

It is useful to set

(4A-5) $\mathbf{U} \vdash_c \alpha(\vec{x}) = w \iff \mathbf{U}$ is finite, generated by $\vec{x}$, and $\mathbf{U} \vdash \alpha(\vec{x}) = w$,

(4A-6) $\mathbf{U} \vdash_c \alpha(\vec{x})\downarrow \iff (\exists w)[\mathbf{U} \vdash_c \alpha(\vec{x}) = w]$.

In this notation, the Finiteness Axiom takes the form

(4A-7) $$\overline{\alpha}(\vec{x}) = w \Longrightarrow (\exists \mathbf{U} \subseteq_p \mathbf{A})[\mathbf{U} \vdash_c \alpha(\vec{x}) = w].$$

If we read "$\vdash_c$" as *computes*, then this form of the axiom suggests that the triples $(\mathbf{U}, \vec{x}, w)$ such that $\mathbf{U} \vdash_c \alpha(\vec{x}) = w$ play the role of *computations* in this abstract setting.

From Theorem 2D.1 we have immediately

4A.1. **Proposition.** *The algorithms expressed by non-deterministic recursive programs on a structure* **A** *satisfy axioms* **I** – **III**.

4B. Concrete algorithms and the Uniformity Thesis

We have been using the word *algorithm* informally, and we will not (and do not need to) "define algorithms" in these notes. Rigorous results in complexity theory are established for *concrete algorithms*, specified by *computation models*, e.g., *Turing machines*, *Random Access machines*, *recursive programs*, ... , and their *non-deterministic* versions.

Axioms **I** – **III** *are satisfied by all concrete algorithms which compute a partial function* $f : A^n \rightharpoonup A_s$ *from specified primitives.* This is plausible from the motivation for the axioms above, but complete proofs are rather tedious, as they must specify in detail and take into account the idiosyncracies of each model. Part of the difficulty comes from the fact that many computation models have some functions on the intended universe A "built-in", so to speak: e.g., Turing machines acting on $\mathbb{N}$ assume the successor and predecessor functions if we code numbers by strings in unary or the primitives of binary arithmetic if we code numbers in binary, and random access machines have the identity relation on $\mathbb{N}$ built in, in addition to whatever functions on $\mathbb{N}$ are explicitly identified in their definition. To prove rigorously that these models satisfy the axioms, we must identify *all* the *non-logical* primitives they assume—and then the result becomes basically trivial, either by direct verification or by appealing to the fact that the *iterators* defined in Section 2B satisfy the axioms, Problem x4C.2. In any case, we will not give in these notes any more detailed proofs of such facts which support the following

4B.1. **Uniformity Thesis.** *Every algorithm which computes a partial function* $f : A^n \rightharpoonup A$ *or decides a relation* $R \subseteq A^n$ *from the primitives of a* Φ*-structure* **A** *satisfies axioms* **I**, **II** *and* **III**.

This is a weak Church-Turing-type assumption about algorithms which, of course, cannot be established rigorously absent a precise definition of algorithms. It limits somewhat the notion of "algorithm", but not in any novel way which yields new undecidability results. We will show, however, that it can be used to derive *absolute lower bounds* for decidable relations, very much like the Church-Turing Thesis is used to establish *absolute undecidability*.

4C. Uniform processes

An n-ary *uniform process* of sort $s \in \{\mathtt{a}, \mathtt{boole}\}$ of a Φ-structure $\mathbf{A}$ is any operation

$$\alpha = (\mathbf{U} \mapsto \overline{\alpha}^{\mathbf{U}}) \quad (\mathbf{U} \subseteq_p \mathbf{A},\ \overline{\alpha}^{\mathbf{U}} : U^n \rightharpoonup U_s)$$

on the substructures of $\mathbf{A}$ which satisfies the Homomorphism and Finiteness Axioms. We say that α *computes* the partial function $\overline{\alpha} : A^n \rightharpoonup A_s$, and we set

$$\mathbf{unif}(\mathbf{A}) = \{f : A^n \rightharpoonup A_s : f \text{ is computed by a uniform process of } \mathbf{A}\}.$$

We will also use for uniform processes the notations introduced in (4A-2) – (4A-6) of 4A.[12]

We have argued (briefly) that every algorithm from specified primitives induces a uniform process and we have proved this for non-deterministic recursive algorithms in Proposition 4A.1,

(4C-1) $$\mathbf{rec}_{\mathrm{nd}}(\mathbf{A}) \subseteq \mathbf{unif}(\mathbf{A}).$$

The converse, however, is far from true: nothing in axioms **I** – **III** suggests that functions computed by uniform processes are "computable" from the primitives of $\mathbf{A}$ in any intuitive sense, and in general, they are not.

4C.1. **Proposition.** *If a Φ-structure $\mathbf{A}$ is generated by the empty tuple, then every $f : A^n \rightharpoonup A_s$ is computed by some uniform process of $\mathbf{A}$.*

In particular, every $f : \mathbb{N}^n \rightharpoonup \mathbb{N}_s$ is computed by some uniform process of $\mathbf{A} = (\mathbb{N}, 0, \Phi^{\mathbf{A}})$ if $\Phi^{\mathbf{A}}$ includes either the successor function S or the primitives of binary arithmetic $\mathrm{em}_2(x) = 2x$ and $\mathrm{om}_2(x) = 2x + 1$.

Proof. Let $G_m = G_m(\mathbf{A}, (\,))$ be the set generated in $\leq m$ steps by the empty tuple, so that $G_0 = \emptyset$, G_1 comprises the distinguished elements of $\mathbf{A}$, etc. Let

$$d(\vec{x}, w) = \min\{m : x_1, \dots, x_n, w \in G_m \cup \{\mathrm{tt}, \mathrm{ff}\}\},$$

and define $\overline{\alpha}^{\mathbf{U}}$ for each $\mathbf{U} \subseteq_p \mathbf{A}$ by

$$\overline{\alpha}^{\mathbf{U}}(\vec{x}) = w \iff f(\vec{x}) = w \ \&\ \mathbf{G}_{d(\vec{x},w)} \subseteq_p \mathbf{U}.$$

[12]In categorical terms, an n-ary uniform process α of $\mathbf{A}$ of sort $s \in \{\mathtt{boole}, \mathtt{a}\}$ is a continuous functor on the category $H_{\mathbf{A}}$ to $P_{\mathbf{A},s}$, where:

(1) The objects of $H_{\mathbf{A}}$ are all pairs $(\mathbf{U}, \vec{x})$ where $\mathbf{U} \subseteq_p \mathbf{A}$, $\vec{x} \in U^n$ and $\mathbf{U}$ is generated by $\vec{x}$, and a morphism $\phi : (\mathbf{U}, \vec{x}) \to (\mathbf{V}, \vec{y})$ is any homomorphism $\pi : \mathbf{U} \to \mathbf{V}$ which carries $\vec{x}$ to $\vec{y}$; and

(2) The objects of $P_{\mathbf{A},s}$ are all n-ary partial functions on A to A_s, and a morphism $\psi : p \to q$ is any partial function $\psi : A \rightharpoonup A$ such that

$$p(u_1, \dots, u_n) = w \Longrightarrow q(\psi(u_1), \dots, \psi(u_n)) = \psi(w).$$

The Finiteness Axiom is immediate taking $\mathbf{U} = \mathbf{G}_{d(\vec{x},w)}$, and the Homomorphism Axiom holds because if $\mathbf{G}_m \subseteq_p \mathbf{U}$, then every homomorphism $\pi : \mathbf{U} \to \mathbf{V}$ fixes every $u \in G_m$. ⊣

The axioms aim to capture the *uniformity* of algorithms—that they compute all their values following "the same procedure"— but surely do not capture their *effectiveness.*

Problems for Section 4C

x4C.1. **Problem.** Prove the categorical characterization of uniform processes in Footnote 12.

x4C.2. **Problem.** For each (possibly non-deterministic) iterator $\mathfrak{i}$ which computes a partial function $f : X \rightharpoonup W$, define a uniform process $\alpha_{\mathfrak{i}}$ which computes f in the associated structure $\mathbf{A}_{\mathfrak{i}}$.

x4C.3. **Problem.** Prove that if $f : A^n \rightharpoonup A_s$ is computable by a uniform process of $\mathbf{A}$, then so is every $g \in \mathbf{unif}(\mathbf{A}, f)$.

x4C.4. **Problem.** Prove that if $f : A^n \rightharpoonup A$ is computed by some uniform process of $\mathbf{A}$ and $\rho : \mathbf{A} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbf{A}$ is an automorphism of $\mathbf{A}$, then

$$f(\rho(\vec{x})) = \rho(f(\vec{x})) \quad (f(\vec{x})\downarrow).$$

x4C.5. **Problem.** Give an example of a finite structure $\mathbf{A}$ and a unary relation $P \subseteq A$ which is respected by all automorphisms of $\mathbf{A}$ but is not decided by any uniform process of $\mathbf{A}$.

x4C.6. **Problem.** Find all the total functions $f^n : A \to A_s$ which are computable by uniform processes in $\mathbf{A} = (A)$ (with no primitives), where A is infinite.

x4C.7*. **Problem.** Suppose $\mathbf{A} = (A, R_1, \dots, R_K)$ is a structure whose primitives are relations on A. What are the (total) relations $P \subseteq A^n$ which are decided by uniform processes of $\mathbf{A}$?

4D. Complexity measures on uniform processes

For any signature Φ, a **Φ-substructure norm** is an operation μ which assigns to every pair $(\mathbf{U}, \vec{x})$ of a finite Φ-structure $\mathbf{U}$ and a tuple $\vec{x} \in U^n$ that generates it a number $\mu(\mathbf{U}, \vec{x})$ and respects isomorphisms, i.e.,

(4D-1) $$\pi : \mathbf{U} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbf{V} \ \&\ x_1, \dots, x_n \in U \Longrightarrow \mu(\mathbf{U}, x_1, \dots, x_n) = \mu(\mathbf{V}, \pi(x_1), \dots, \pi(x_n)).$$

Typical examples are

$$\begin{aligned}\mathrm{depth}(\mathbf{U},\vec{x}) &= \min\{m : \mathbf{U} = \mathbf{G}_m(\mathbf{U},\vec{x})\},\\ \mathrm{size}(\mathbf{U},\vec{x}) &= |U_{\mathrm{vis}}|,\\ \mathrm{calls}_{\Phi_0}(\mathbf{U},\vec{x}) &= |\mathrm{eqdiag}(\mathbf{U}\restriction\Phi_0)| \quad (\Phi_0 \subseteq \Phi).\end{aligned}$$

If μ is a Φ-substructure norm, $\mathbf{A}$ is a Φ-structure and α is a uniform process in $\mathbf{A}$, then the μ-**complexity measure** of α is the partial function

$$C_\mu(\alpha,\vec{x}) = \min\{\mu(\mathbf{U},\vec{x}) : \mathbf{U} \vdash_c \alpha(\vec{x})\downarrow\}, \tag{4D-2}$$

defined on the domain of convergence of $\overline{\alpha}^{\mathbf{A}}$. By using the norms above, we get three natural complexity measures on uniform processes,[13]

$$\mathrm{depth}(\alpha,\vec{x}),\ \mathrm{size}(\alpha,\vec{x}),\ \mathrm{calls}_{\Phi_0}(\alpha,\vec{x}). \tag{4D-3}$$

The first and last of these three correspond to familiar complexity measures with roughly similar names for concrete algorithms but not exactly:[14]

- $\mathrm{calls}_{\Phi_0}(\alpha,\vec{x})$ intuitively counts the least number of distinct calls to primitives in Φ_0 required to compute $\overline{\alpha}(\vec{x})$ by the process α;
- the "parallel" measure $\mathrm{depth}(\alpha,\vec{x})$ counts the least number of distinct calls to the primitives of $\mathbf{A}$ which *must be executed in sequence* to compute $\overline{\alpha}(\vec{x})$; and
- the less familiar middle measure $\mathrm{size}(\alpha,\vec{x})$ counts *the least number of points in A that α must see to compute $\overline{\alpha}(\vec{x})$.*

These measures are typically lower than their versions with the same names for concrete algorithms, because they count *distinct* calls and points, while an algorithm may (stupidly or by design, e.g., to simplify the code) make the same call many times, cf. Problem x4D.1.

4D.1. **Lemma.** *For every uniform process α of a Φ-structure* $\mathbf{A}$ *and every $\vec{x}$ such that $\overline{\alpha}(\vec{x})\downarrow$,*

$$\mathrm{depth}(\overline{\alpha}(\vec{x});\mathbf{A},\vec{x}) \le \mathrm{depth}(\alpha,\vec{x}) \le \mathrm{size}(\alpha,\vec{x}) \le \mathrm{calls}(\alpha,\vec{x}), \tag{4D-4}$$

where, by convention, $\mathrm{depth}(\mathrm{tt},\mathbf{A}) = \mathrm{depth}(\mathrm{ff},\mathbf{A}) = 0$.

Proof. The first inequality is trivial if $w \in \{\mathrm{tt},\mathrm{ff}\}$ and immediate for $w \in A$, because if $\mathbf{U} \vdash_c \alpha(\vec{x}) = w$, then $w \in U$ and so

$$\mathrm{depth}(w;\mathbf{A},\vec{x}) \le \mathrm{depth}(w;\mathbf{U},\vec{x}) \le \mathrm{depth}(\mathbf{U},\vec{x}).$$

[13] The depth measure can also be relativized to arbitrary $\Phi_0 \subseteq \Phi$, but it is tedious and we have no interesting results about it.

[14] There are, of course, many other substructure norms which induce useful complexity measures, including those which come by combining the three basic ones: for example

$$\mu(\mathbf{U},\vec{x}) = \mathrm{size}(\mathbf{U},\vec{x})\cdot 6^{\mathrm{depth}(\mathbf{U},\vec{x})}$$

actually comes up naturally in a proof further on!

For the third claimed inequality, suppose $\overline{\alpha}(\vec{x}) = w$ and choose a substructure $\mathbf{U} \subseteq_p \mathbf{A}$ with least $|\mathrm{eqdiag}(\mathbf{U})|$ such that $\mathbf{U} \vdash_c \alpha(\vec{x}) = w$, so that $\mathrm{calls}(\alpha, \vec{x}) = |\mathrm{eqdiag}(\mathbf{U})|$. Now $\mathrm{size}(\mathbf{U}) \leq |\mathrm{eqdiag}(\mathbf{U})|$ by (1C-15) in Proposition 1C.1, and since $\mathbf{U}$ is among the substructures considered in the definition of $\mathrm{size}(\alpha, \vec{x})$, we have

$$\mathrm{size}(\alpha, \vec{x}) \leq |\mathrm{eqdiag}(\mathbf{U})| = \mathrm{calls}(\alpha, \vec{x}).$$

The second inequality is proved by a similar argument. ⊣

Problems for Section 4D

x4D.1. **Problem.** Let α_E be the uniform process induced by a deterministic program E in a Φ-structure $\mathbf{A}$ by (4A-4). Prove that for all $\vec{x} \in A^n$ such that $\overline{p}_E(\vec{x})\downarrow$,

$$\mathrm{depth}(\alpha_E, \vec{x}) \leq c^p(\mathbf{A}, E, \vec{x}),$$
$$\mathrm{calls}_{\Phi_0}(\alpha_E, \vec{x}) \leq c^s_{\Phi_0}(\mathbf{A}, E, \vec{x}) \quad (\Phi_0 \subseteq \Phi)$$

as these complexities were defined in Sections 3A.5 and 3A.4, and give an example where these inequalities are strict.

Hint: Show the following refinement of (b) of Proposition 2D.1 for deterministic programs: if $M \in \mathrm{Conv}(\mathbf{A}, E)$ and $X \subseteq A$ contains all the parameters which occur in M, then

$$\mathrm{den}(\mathbf{A}, E, M) = \mathrm{den}(\mathbf{G}_m[\mathbf{A}, X], E, M) \text{ with } m = C^p(\mathbf{A}, E, M).$$

x4D.2. **Problem.** Prove that if α_E is the uniform process induced by a non-deterministic recursive program in $\mathbf{A}$ by (4A-4), then

$$\mathrm{calls}(\alpha, \vec{x}) \leq \mathrm{Time}^e_E(\vec{x}) \qquad (\overline{\alpha}(\vec{x})\downarrow).$$

x4D.3. **Problem.** Prove that if the successor S is a primitive of a structure $\mathbf{A} = (\mathbb{N}, 0, \mathbf{\Phi})$, then every $f : \mathbb{N}^n \rightharpoonup \mathbb{N}_s$ is computed by some uniform process α of $\mathbf{A}$ with

$$\mathrm{calls}(\alpha, \vec{x}) \leq \max\{\vec{x}, f(\vec{x})\} \quad (f(\vec{x})\downarrow)$$

where $\max\{\vec{x}, w\} = \max\{\vec{x}\}$ if $w \in \{\mathrm{tt}, \mathrm{ff}\}$. Hint: Look up the proof of Lemma 4C.1.

x4D.4. **Problem.** Prove that if $0, 1$ and the binary primitives $\mathrm{em}_2(x) = 2x$, $\mathrm{om}_2(x) = 2x + 1$ are among the primitives of $\mathbf{A} = (\mathbb{N}, \mathbf{\Phi})$, then every $f : \mathbb{N}^n \rightharpoonup \mathbb{N}_s$ is computed by some uniform process α of $\mathbf{A}$ with

$$\mathrm{calls}(\alpha, \vec{x}) \leq 2\max\{\lfloor\log(x_1)\rfloor, \dots, \lfloor\log(x_n)\rfloor, \lfloor\log(f(\vec{x}))\rfloor\} \quad (f(\vec{x})\downarrow),$$

with the same convention about truth values as in the previous problem.

4E. Forcing and certification

Suppose $\mathbf{A}$ is a Φ-structure, $f : A^n \rightharpoonup A_s$ (with $s \in \{\mathtt{a}, \mathtt{boole}\}$), $\mathbf{U} \subseteq_p \mathbf{A}$, and $f(\vec{x})\downarrow$. A homomorphism $\pi : \mathbf{U} \to \mathbf{A}$ *respects f at $\vec{x}$* if

$$\vec{x} \in U^n \ \& \ f(\vec{x}) \in U_s \ \& \ \pi(f(\vec{x})) = f(\pi(\vec{x})). \tag{4E-1}$$

Next come *forcing* and *certification*, the two basic notions of this chapter:

$$\begin{aligned}\mathbf{U} \Vdash^{\mathbf{A}} f(\vec{x}) = w &\iff f(\vec{x}) = w\\ &\qquad \& \text{ every homomorphism } \pi : \mathbf{U} \to \mathbf{A} \text{ respects } f \text{ at } \vec{x},\\ \mathbf{U} \Vdash^{\mathbf{A}}_c f(\vec{x}) = w &\iff \mathbf{U} \text{ is finite, generated by } \vec{x} \ \& \ \mathbf{U} \Vdash^{\mathbf{A}} f(\vec{x}) = w,\\ \mathbf{U} \Vdash^{\mathbf{A}}_c f(\vec{x})\downarrow &\iff (\exists w)[\mathbf{U} \Vdash^{\mathbf{A}}_c f(\vec{x}) = w].\end{aligned}$$

If $\mathbf{U} \Vdash^{\mathbf{A}}_c f(\vec{x})\downarrow$, we call $\mathbf{U}$ a *certificate* for f at $\vec{x}$ in $\mathbf{A}$.[15]

Notice that

$$\Big(\mathbf{U}_1 \subseteq_p \mathbf{U}_2 \ \& \ \mathbf{U}_1 \Vdash^{\mathbf{A}} f(\vec{x}) = w\Big) \Longrightarrow \mathbf{U}_2 \Vdash^{\mathbf{A}} f(\vec{x}) = w,$$

so, in particular, if $\mathbf{U}_1$ forces $f(\vec{x}) = w$, then so does every $\mathbf{U}_2 \supseteq_p \mathbf{U}_1$, and similarly for certification for finite $\mathbf{U}_1, \mathbf{U}_2$ generated by the input.

We will sometimes write $\Vdash$ and $\Vdash_c$ for $\Vdash^{\mathbf{A}}$ and $\Vdash^{\mathbf{A}}_c$ when the relevant Φ-structure is clear from the context.

Example: the Euclidean algorithm. To illustrate the notions consider once more the Euclidean algorithm for coprimeness, specified by the recursive program

$$\begin{aligned}&\varepsilon \equiv \mathrm{eq}_1(\mathrm{gcd}(x, y)) \ \mathsf{where}\\ &\qquad\qquad \{\mathrm{gcd}(x, y) = \text{if } \mathrm{eq}_0(\mathrm{rem}(x, y)) \text{ then } y \text{ else } \mathrm{gcd}(y, \mathrm{rem}(x, y))\}\end{aligned}$$

of the structure $\mathbf{N}_\varepsilon = (\mathbb{N}, \mathrm{rem}, \mathrm{eq}_0, \mathrm{eq}_1)$. Given $x, y \geq 1$, the Euclidean computes $\mathrm{gcd}(x, y)$ by successive divisions and 0-tests (calls to the rem- and eq_0-oracles)

$$\begin{aligned}&\mathrm{rem}(x, y) = r_1, r_1 \neq 0, \mathrm{rem}(y, r_1) = r_2, r_2 \neq 0,\\ &\qquad\qquad\qquad \ldots, r_{n+1} \neq 0, \mathrm{rem}(r_n, r_{n+1}) = r_{n+2}, r_{n+2} = 0\end{aligned}$$

until the remainder 0 is obtained, at which time it is known that $\mathrm{gcd}(x, y) = r_{n+1}$; and to decide if $x \perp\!\!\!\perp y$, it must then do one last check to test whether

[15]To the best of my knowledge, certificates were first introduced in Pratt [1975] in his proof that primality is NP. The present notion is model theoretic and more abstract than Pratt's, but the idea is the same.

$r_{n+1} = 1$. Suppose $x \perp\!\!\!\perp y$ and collect all these calls into a substructure $\mathbf{U}_0$, writing $u \neq 0, u = 1$ for $\mathrm{eq}_0(u) = \mathrm{ff}, \mathrm{eq}_1(u) = \mathrm{tt}$ as above:

$$\begin{aligned}\mathrm{eqdiag}(\mathbf{U}_0) = \{&\mathrm{rem}(x,y) = r_1, r_1 \neq 0, \mathrm{rem}(y, r_1) = r_2, r_2 \neq 0,\\ &\dots, r_{n+1} \neq 0, \mathrm{rem}(r_n, r_{n+1}) = r_{n+2}, r_{n+2} = 0, r_{n+1} = 1\};\end{aligned}$$

it is now easy to check that

$$\mathbf{U}_0 \Vdash_c^{\mathbf{N}_\varepsilon} x \perp\!\!\!\perp y,$$

because if $\pi : \mathbf{U}_0 \to \mathbf{N}_\varepsilon$ is a homomorphism, then

$$\begin{aligned}\text{(4E-2)}\quad &\mathrm{rem}(\pi(x), \pi(y)) = \pi(r_1), \pi(r_1) \neq 0,\\ &\mathrm{rem}(\pi(y), \pi(r_1)) = \pi(r_2), \pi(r_2) \neq 0,\\ &\dots, \pi(r_{n+1}) \neq 0, \mathrm{rem}(\pi(r_n), \pi(r_{n+1})) = \pi(r_{n+2}), \pi(r_{n+2}) = 0,\\ &\pi(r_{n+1}) = 1,\end{aligned}$$

and this in turn guarantees that $\pi(x) \perp\!\!\!\perp \pi(y)$, so that π respects the coprimeness relation at x, y. This is how certificates for functions and relations can be constructed from computations, and it is the basic method of applying uniform process theory to the derivation of lower bounds for concrete algorithms.

On the other hand, $\mathbf{U}_0$ is not a *minimal* substructure of $\mathbf{N}_\varepsilon$ which certifies that $x \perp\!\!\!\perp y$. Let

$$\begin{aligned}\text{(4E-3)}\quad \mathbf{U}_1 = \{&\mathrm{rem}(x,y) = r_1, \mathrm{rem}(y, r_1) = r_2,\\ &\dots, \mathrm{rem}(r_n, r_{n+1}) = r_{n+2}, r_{n+1} = 1\},\end{aligned}$$

be the substructure of $\mathbf{U}_0$ with all the 0-tests deleted. We claim that $\mathbf{U}_1$ is also a certificate for $x \perp\!\!\!\perp y$, and to see this suppose that $\pi : \mathbf{U}_1 \to \mathbf{N}_\varepsilon$ is a homomorphism. To verify that π respects $x \perp\!\!\!\perp y$, check first that for $i = 1, \dots, n+1$, $\pi(r_i) \neq 0$; otherwise $\mathrm{rem}(\pi(r_{i-1}), \pi(r_i))$ would not be defined (with $r_0 = y$), since rem requires its second argument to be nonzero, and so π would not be totally defined in $\mathbf{U}_1$. So the homomorphism property for π guarantees that

$$\begin{aligned}&\mathrm{rem}(\pi(x), \pi(y)) = \pi(r_1), \pi(r_1) \neq 0, \mathrm{rem}(\pi(y), \pi(r_1)) = \pi(r_2), \pi(r_2) \neq 0,\\ &\dots, \pi(r_{n+1}) \neq 0, \mathrm{rem}(\pi(r_n), \pi(r_{n+1})) = \pi(r_{n+2}), \pi(r_{n+1}) = 1.\end{aligned}$$

The last two of these equations mean that for some q,

$$\pi(r_n) = q \cdot 1 + \pi(r_{n+2}), \quad 0 \leq \pi(r_{n+2}) < 1$$

so that we must have $\pi(r_{n+2}) = 0$; and then all the equations in (4E-2) hold and we have the required $\pi(x) \perp\!\!\!\perp \pi(y)$.

This is typical: although computations of concrete algorithms define certificates, they generally do not give minimal (or even least-in-size) certificates—which is why the lower bounds for uniform processes are typically not the best (largest) lower bounds that one might be able to prove for concrete algorithms using other methods.

Problems for Section 4E

x4E.1. **Problem.** Suppose E is a non-deterministic $\mathbf{A}$-recursive program which computes $f = \overline{p}_E : A^n \rightharpoonup A_s$ and

$$\mathbf{c} = (E_0(\vec{x}) :, \dots, : w)$$

is a computation of E, so that $w = f(\vec{x})$. Let $\mathbf{U_c}$ be the structure with

$$\text{eqdiag}(\mathbf{U_c}) = \{(\phi, \vec{u}, v) : \text{a transition } \vec{a}\ \phi : \vec{u}\ \vec{b} \to \vec{a} : v\ \vec{b} \text{ occurs in } \mathbf{c}\}.$$

Prove that $\mathbf{U_c} \Vdash_c f(\vec{x}) = w$.

x4E.2. **Problem.** Prove that for any coprime $x \geq 1 \geq 1$, the structure $\mathbf{U}_1$ defined in (4E-3) is a minimal certificate of $x \perp\!\!\!\perp y$ in $\mathbf{N}_\varepsilon$, i.e., no proper substructure of $\mathbf{U}_1$ certifies $x \perp\!\!\!\perp y$. Hint: For example, if we delete the last equation $r_{n+1} = 1$ from eqdiag($\mathbf{U}_1$), then the function $\pi(u) = 2u$ defines a homomorphism on the resulting substructure such that $\gcd(\pi(x), \pi(y)) = 2$.

4F. Intrinsic complexities of functions and relations

If μ is a Φ-substructure norm, $\mathbf{A}$ is a Φ-structure and $f : A^n \rightharpoonup A_s$, set

$$C_\mu(\mathbf{A}, f, \vec{x}) = \min\{\mu(\mathbf{U}, \vec{x}) : \mathbf{U} \Vdash_c f(\vec{x})\downarrow\} \qquad (f(\vec{x})\downarrow), \tag{4F-1}$$

where, as usual, $\min(\emptyset) = \infty$. This is the *intrinsic μ-complexity* (of f, in $\mathbf{A}$, at $\vec{x}$). It records the μ-smallest size of a piece of $\mathbf{A}$ that is needed to determine the value $f(\vec{x})$, and its significance derives from the following, trivial

4F.1. **Proposition.** *If a uniform process α computes $f : A^n \rightharpoonup A_s$ in a Φ-structure $\mathbf{A}$, then for any substructure norm μ on $\mathbf{A}$,*

$$C_\mu(\mathbf{A}, f, \vec{x}) \leq C_\mu(\alpha, \vec{x}) \qquad (f(\vec{x})\downarrow). \tag{4F-2}$$

Proof is immediate, because

$$\mathbf{U} \vdash_c \alpha(\vec{x}) = w \Longrightarrow \mathbf{U} \Vdash_c^{\mathbf{A}} f(\vec{x}) = w, \tag{4F-3}$$

directly from Axioms **II** and **III** for uniform processes and the definition of certification. ⊣

The key point here is that $C_\mu(\mathbf{A}, f, \vec{x})$ is defined directly from $\mathbf{A}$, μ and f, but it provides a lower bound for the μ-complexity of any uniform process which might compute f in $\mathbf{A}$—and a fortiori for the μ-complexity of any (deterministic or non-deterministic) algorithm which might compute f in $\mathbf{A}$ by the Uniformity Thesis 4B.1. The situation is most interesting, of course, when $C_\mu(\mathbf{A}, f, \vec{x})$ matches the μ-complexity of some known algorithm which computes f in $\mathbf{A}$, at least up to a multiplicative constant.

Moreover, the definitions yield a purely algebraic method for deriving these intrinsic lower bounds:

4F.2. **Lemma** (Homomorphism Test). *Suppose* $\mathbf{A}$ *is a* Φ*-structure,* μ *is a* Φ*-substructure norm,* $f : A^n \rightharpoonup A_s$, $f(\vec{x})\downarrow$, *and*

(4F-4) *for every finite* $\mathbf{U} \subseteq_p \mathbf{A}$ *which is generated by* $\vec{x}$,

$$\Big(f(\vec{x}) \in U_s \ \&\ \mu(\mathbf{U}, \vec{x}) < m\Big) \Longrightarrow (\exists \pi : \mathbf{U} \to \mathbf{A})[f(\pi(\vec{x})) \neq \pi(f(\vec{x}))];$$

then $C_\mu(f, \mathbf{A}, \vec{x}) \geq m$.

This implies (directly from the definitions) the following natural fact, which is also useful:

4F.3. **Theorem** (Isomorphism Test). *Suppose* $\pi : \mathbf{A} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbf{B}$ *is an isomorphism between two* Φ*-structures,* μ *is a* Φ*-substructure norm,* $f : A^n \to A_s$ (with $s \in \{\mathtt{a}, \mathtt{boole}\}$)*, and* $f^\pi : B^n \to B_s$ *is the image of* f *under* π*, i.e.,*

$$f^\pi(y_1, \dots, y_n) = \pi(f(\pi^{-1}(y_1), \dots, \pi^{-1}(y_n))) \quad (y_1, \dots, y_n \in B).$$

Then for all $x_1, \dots, x_n \in A$,

$$C_\mu(\mathbf{A}, f, x_1, \dots, x_n) = C_\mu(\mathbf{B}, f^\pi, \pi(x_1), \dots, \pi(x_n)).$$

Other than the definition of $C_\mu(\mathbf{A}, f, \vec{x})$, these are the main—in fact the only—tools we will use in the sequel to derive intrinsic lower bounds from specified primitives. The results will be (primarily) about the most important intrinsic complexity measures, when $\mu(\mathbf{U}, \vec{x})$ is $\mathrm{depth}(\mathbf{U}, \vec{x})$, $\mathrm{size}(\mathbf{U})$ or $\mathrm{calls}(\mathbf{U} \restriction \Phi_0)$, in symbols

$$\begin{aligned}
\mathrm{depth}_f(\mathbf{A}, \vec{x}) &= C_{\mathrm{depth}}(\mathbf{A}, f, \vec{x}) &&= \min\{\mathrm{depth}(\mathbf{U}, \vec{x}) : \mathbf{U} \Vdash_c^{\mathbf{A}} f(\vec{x})\downarrow\},\\
\mathrm{size}_f(\mathbf{A}, \vec{x}) &= C_{\mathrm{size}}(\mathbf{A}, f, \vec{x}) &&= \min\{\mathrm{size}(\mathbf{U}) : \mathbf{U} \Vdash_c^{\mathbf{A}} f(\vec{x})\downarrow\},\\
\mathrm{calls}_{f,\Phi_0}(\mathbf{A}, \vec{x}) &= C_{\mathrm{calls}_{\Phi_0}}(\mathbf{A}, f, \vec{x}) &&= \min\{\mathrm{calls}(\mathbf{U} \restriction \Phi_0) : \mathbf{U} \Vdash_c^{\mathbf{A}} f(\vec{x})\downarrow\}.
\end{aligned}$$

We will find both of these notations for the basic intrinsic complexities useful in different contexts.

As usually, $\mathrm{calls}_f(\mathbf{A}, \vec{x}) = \mathrm{calls}_{f,\Phi}(\mathbf{A}, \vec{x})$, so that by Lemma 4D.1,

$$\mathrm{depth}_f(\mathbf{A}, \vec{x}) \leq \mathrm{size}_f(\mathbf{A}, \vec{x}) \leq \mathrm{calls}_f(\mathbf{A}, \vec{x}).$$

Value-depth and intrinsic depth complexity. We note here that for any $f : A^n \rightharpoonup A$,

$$\text{(4F-5)} \qquad \text{depth}(f(\vec{x}); \mathbf{A}, \vec{x}) \leq \text{depth}_f(\mathbf{A}, \vec{x});$$

this holds simply because if $\mathbf{U} \Vdash_c f(\vec{x})\downarrow$, then $f(\vec{x}) \in U$, $\mathbf{U} = \mathbf{G}_\infty(\mathbf{U}, \vec{x})$, and so

$$\text{depth}(f(\vec{x}); \mathbf{A}, \vec{x}) \leq \text{depth}(f(\vec{x}); \mathbf{U}, \vec{x}) \leq \text{depth}_f(\mathbf{A}, \vec{x}).$$

The *value-depth complexity* $\text{depth}(f(\vec{x}); \mathbf{A}, \vec{x})$ provides a lower bound for any reasonable notion of algorithmic complexity measure which counts (among other things) the applications of primitives that must be executed in sequence, simply because an algorithm must (at least) construct from the input the value $f(\vec{x})$. This is well understood and used extensively to derive lower bounds in arithmetic and algebra which are clearly absolute.[16] We will consider some results of this type in Section 5A. The more sophisticated complexity measure $\text{depth}_f(\mathbf{A}, \vec{x})$ is especially useful when f takes simple values, e.g., when f is a relation: in this case $\text{depth}(f(\vec{x}); \mathbf{A}, \vec{x}) = 0$ and (4F-5) does not give us any information.

Explicit (term) reduction and equivalence. Intrinsic complexities are very fine measures of information. Sometimes, especially in algebra, we want to know the exact value $C_\mu(\mathbf{A}, f, \vec{x})$, which might be the degree of a polynomial or the dimension of a space. In other cases, especially in arithmetic, we only need to know $C_\mu(\mathbf{A}, f, \vec{x})$ up to a factor (a multiplicative constant), either because the exact value is too difficult to compute or because we care only for asymptotic estimates of computational complexity. The next, simple proposition gives a trivial way to relate the standard complexity measures of two structures when the primitives of one are explicitly definable in the other.

A structure $\mathbf{A} = (A, \mathbf{\Phi})$ is *explicitly reducible* to a structure $\mathbf{A}' = (A, \mathbf{\Psi})$ on the same universe if $\mathbf{\Phi} \subseteq \mathbf{expl}(\mathbf{A}')$, and *explicitly equivalent* to $\mathbf{A}'$ if in addition $\mathbf{\Psi} \subseteq \mathbf{expl}(\mathbf{A})$.

4F.4. **Proposition** (Explicit reduction). *If $\mathbf{A} = (A, \mathbf{\Phi})$ is explicitly reducible to $\mathbf{A}' = (A, \mathbf{\Psi})$, then there is constant $K \in \mathbb{N}$ such that for every $f : A^n \rightharpoonup A_s$ and each of the standard complexities $\mu = \text{depth}, \text{size}, \text{calls}$,*

$$C_\mu(\mathbf{A}', f, \vec{x}) \leq K\, C_\mu(\mathbf{A}, f, \vec{x}) \quad (f(\vec{x})\downarrow).$$

[16]The most interesting result of this type that I know is Theorem 4.1 in van den Dries and Moschovakis [2009], an $O\left(\sqrt{\log\log a}\right)$-lower bound on $\text{depth}(\text{gcd}(a+1, b), \mathbf{A}, a, b)$ with $\mathbf{A} = (\mathbb{N}, 0, 1, +, -, \cdot, \div)$ and (a, b) a Pell pair. This is due to van den Dries, and it is the largest lower bound known for the gcd from primitives that include multiplication. It is not known whether it holds for coprimeness, for which the best result is a log log log-lower bound for algebraic decision trees in Mansour, Schieber, and Tiwari [1991a].

It follows that if $\mathbf{A}$ *and* $\mathbf{A}'$ *are explicitly equivalent, then for suitable rational constants* $K, r > 0$, *every* $f : A^n \rightharpoonup A_s$ *and* $\mu = \text{depth}, \text{size}, \text{calls}$,

$$r\, C_\mu(\mathbf{A}, f, \vec{x}) \le C_\mu(\mathbf{A}', f, \vec{x}) \le K\, C_\mu(\mathbf{A}, f, \vec{x}) \quad (f(\vec{x})\downarrow).$$

Proof is fairly simple and we will leave it for Problem x4F.4*. ⊣

Problems for Section 4F

x4F.1. **Problem.** Prove that if $R : A^n \rightharpoonup \texttt{boole}$ is a partial relation and for some tuple $\vec{x} = (x_1, \dots, x_n) \in A^n$ of distinct elements

$$R(\vec{x})\downarrow \ \ \& \ \text{depth}_R(\mathbf{A}, \vec{x}) = 0,$$

then R is total and constant, i.e., either $(\forall \vec{y})R(\vec{y})$ or $(\forall \vec{y})\neg R(\vec{y})$.

Infer that if R is total and not identically true or identically false, then for every $\vec{x}$, $\text{depth}_R(\mathbf{A}, \vec{x}) > 0$.

So except for trivial relations, none of the most common depth, size and calls complexities are ever 0—which allows us to divide by them in computations, as we will often do.

x4F.2. **Problem.** Prove that for the coprimeness relation,

$$\text{depth}_{\perp\!\!\perp}(\mathbf{N}_\varepsilon, x, y) \le 2\log(\min(x, y)) + 1 \quad (x, y \ge 1).$$

x4F.3. **Problem.** Prove that for the coprimeness relation, some K and all $t \ge 3$,

$$\text{depth}_{\perp\!\!\perp}(\mathbf{N}_\varepsilon, F_{t+1}, F_t) \le K(\log t) = O(\log\log F_t),$$

where $F_0, F_1, \dots$ is the Fibonacci sequence. Hint: Use Pratt's algorithm.

x4F.4*. **Problem.** Prove Proposition 4F.4. Hint: Start with

$$K = \max\{C_\mu(\mathbf{A}', \phi^{\mathbf{A}}, \vec{x}) : \phi \in \Phi\}.$$

4G. The best uniform process

Is there a "best algorithm" which computes a given $f : A^n \rightharpoonup A_s$ from specified primitives on A? The question is vague, of course—and the answer is probably negative in the general case, no matter how you make it precise. The corresponding question about uniform processes has a positive (and very simple) answer.

For given $f : A^n \rightharpoonup A_s$, set

(4G-1) $$\overline{\boldsymbol{\beta}}^{\mathbf{U}}_{f,\mathbf{A}}(\vec{x}) = w \iff \mathbf{U} \Vdash^{\mathbf{A}} f(\vec{x}) = w \quad (\mathbf{U} \subseteq_p \mathbf{A}).$$

4G.1. **Theorem.** *The following are equivalent for any Φ-structure* $\mathbf{A}$ *and any partial function* $f : A^n \rightharpoonup A_s$, $s \in \{\mathtt{a}, \mathtt{boole}\}$.

(i) *Some uniform process* α *of* $\mathbf{A}$ *computes* f.

(ii) $(\forall \vec{x})\Big(f(\vec{x})\downarrow \implies (\exists \mathbf{U} \subseteq_p \mathbf{A})[\mathbf{U} \Vdash_c^{\mathbf{A}} f(\vec{x})\downarrow]\Big)$.

(iii) $\boldsymbol{\beta}_{f,\mathbf{A}}$ *is a uniform process of* $\mathbf{A}$ *which computes* f.

Moreover, if these conditions hold, then

$$C_\mu(\boldsymbol{\beta}_{f,\mathbf{A}}, \vec{x}) = C_\mu(\mathbf{A}, f, \vec{x}) \le C_\mu(\alpha, \vec{x}) \qquad (f(\vec{x})\downarrow),$$

for every Φ*-substructure norm and any uniform process* α *of* $\mathbf{A}$ *which computes* f.

Proof. (iii) $\implies$ (i) is immediate and (i) $\implies$ (ii) follows from (4F-3).

(ii) $\implies$ (iii). The operation $(\mathbf{U} \mapsto \overline{\boldsymbol{\beta}}^{\mathbf{U}}_{f,\mathbf{A}})$ satisfies the Finiteness Axiom **III** by (ii). To verify the Homomorphism Axiom **II**, suppose

$$\mathbf{U} \Vdash^{\mathbf{A}} f(\vec{x}) = w \ \&\ \pi : \mathbf{U} \to \mathbf{V}$$

so that $\pi(\vec{x}) \in V^n$, $\pi(w) \in V_s$ and (since $\pi : \mathbf{U} \to \mathbf{V}$ is a homomorphism), $f(\pi(\vec{x})) = \pi(w)$. Let $\rho : \mathbf{V} \to \mathbf{A}$ be a homomorphism. The composition $\rho \circ \pi : \mathbf{U} \rightarrowtail \mathbf{A}$ is also a homomorphism, and so by (ii) it respects f at $\vec{x}$, i.e.,

$$f(\rho(\pi(\vec{x})) = \rho(\pi(f(\vec{x})) = \rho(\pi(w)) = \rho(f(\pi(\vec{x}))).$$

So ρ respects f at $\pi(\vec{x})$, and since it is arbitrary, we have the required

$$\mathbf{V} \Vdash^{\mathbf{A}} f(\pi(\vec{x})) = \pi(w).$$

The second claim follows from the definition of $\boldsymbol{\beta}_{f,\mathbf{A}}$ and (4F-3). $\dashv$

Weak optimality. A uniform process α of $\mathbf{A}$ is *μ-weakly optimal for a total function* $f : A^n \to A_s$ if it computes f in $\mathbf{A}$ and

(4G-2) $$(\exists K)(\text{for infinitely many } \vec{x})[C_\mu(\alpha, \vec{x}) \le K C_\mu(\mathbf{A}, f, \vec{x})];$$

put another way, a uniform process α which computes f in $\mathbf{A}$ *is not μ-weakly optimal for* f if

$$(\forall r > 0)(\text{for all but finitely many } \vec{x})[C_\mu(\mathbf{A}, f, \vec{x}) < r C_\mu(\alpha, \vec{x})],$$

i.e., if the best uniform process which computes f in $\mathbf{A}$ is μ-better (up to a multiplicative constant) than α on a cofinite set. This is not necessarily the best—and certainly not the only—notion of optimality for algorithmic complexity, but it has the advantage that it holds for some specific, concrete algorithms and problems and it is often the strongest optimality result that we can prove.

4H. Deterministic uniform processes

An n-ary uniform process of a structure $\mathbf{A}$ is *deterministic* if it satisfies the following, stronger form of the Finiteness Axiom as expressed in (4A-7):

$$\text{(4H-1)} \quad \overline{\alpha}(\vec{x}) = w \Longrightarrow (\exists \mathbf{U} \subseteq_p \mathbf{A})\Big(\mathbf{U} \vdash_c \alpha(\vec{x}) = w] \\ \& \text{ (for all } \mathbf{V} \subseteq_p \mathbf{A})[\mathbf{V} \vdash_c \alpha(\vec{x}) = w \Longrightarrow \mathbf{U} \subseteq_p \mathbf{V}]\Big),$$

i.e., if whenever $\overline{\alpha}(\vec{x})\downarrow$, then there is a unique, $\subseteq_p$-least "abstract computation" of $\overline{\alpha}(\vec{x})$ by α. The notion is natural and interesting. We put it down here for completeness, but we have no real understanding of deterministic uniform processes and no methods for deriving lower bounds for them which are better (larger) that the lower bounds for all uniform processes which compute the same function.

Problems for Section 4H

x4H.1. **Problem.** Prove that the uniform process α_E induced by a deterministic recursive $\mathbf{A}$-program is deterministic.

x4H.2*. **Problem.** Give an example of a total, finite structure $\mathbf{A}$ and a unary relation R on A such that for some a, $\text{depth}(\mathbf{A}, R, a) = 1$, but for every deterministic uniform α which decides R in $\mathbf{A}$, $\text{depth}(\alpha, a) > 1$.

PART III, SOME LOWER BOUNDS IN ARITHMETIC

CHAPTER 5

LOWER BOUNDS FROM PRESBURGER PRIMITIVES

We establish here log-lower bounds for $\mathrm{depth}_f(\mathbf{Lin}_d, \vec{x})$ of various functions, where $\mathbf{Lin}_d = \{0, 1, \dots, d, +, \dot{-}, \mathrm{iq}_d, =, <\}$ and

$$\text{(5-2)} \qquad \mathbf{N}_d = (\mathbb{N}, \mathbf{Lin}_d) = (\mathbb{N}, 0, 1, \dots, d, +, \dot{-}, \mathrm{iq}_d, =, <) \quad (d \geq 2).$$

The structure $\mathbf{N}_d$ is clearly explicitly equivalent to its reduct without the constants $2, \dots, d$, but including them among the primitives simplifies some of the formulas below. Lower bounds for $\mathbf{N}_d$ have wide applicability: binary arithmetic $\mathbf{N}_b$ and the Stein structure $\mathbf{N}_{\mathrm{st}}$ are explicitly reducible to $\mathbf{N}_2$, and every structure on $\mathbb{N}$ with finitely many Presburger primitives is explicitly equivalent with some $\mathbf{N}_d$, cf. Problems x5A.1 and x5A.3*.

The results in this Chapter are interesting on their own, but they also illustrate the use of the Homomorphism Test 4F.2 in a very simple context, where the required arithmetic is trivial. They are mostly from van den Dries and Moschovakis [2004], [2009].

5A. Representing the numbers in $G_m(\mathbf{N}_d, \vec{a})$

To illustrate the use of the primitives of $\mathbf{Lin}_d$, consider the following.

5A.1. **Lemma.** *There is a recursive program E which computes the product $x \cdot y$ from $\mathbf{Lin}_2$ with parallel complexity*

$$c^p_E(x, y) \leq 3 \log(\min(x, y)) \quad (x, y \geq 2).$$

PROOF. The idea (from Problem x1B.2) is to reduce multiplication by x to multiplication by 2 and $\mathrm{iq}_2(x)$ (and addition), using the identity

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

which means that the multiplication function satisfies and is determined by the recursive equation

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

Now, obviously,

$$c_f^p(0,y)=1, \quad c_f^p(1,y)=\max\{1,1\}=1,$$

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

$$c_f^p(x,y) \leq c_f^p(\text{iq}_2(x),y)+2.$$

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

$$c_f^p(x,y) \leq 3\log(x) \quad (x \geq 2),$$

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

$$c_f^p(x,y) \leq c_f^p(\text{iq}_2(x),y)+2 \leq 3\log(\frac{x}{2})+3 = 3(\log(\frac{x}{2})+1) = 3\log x.$$

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:

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

the resulting program E has the claimed complexity bound. $\dashv$

The basic tool for the derivation of lower bounds in $\mathbf{N}_d$ is a canonical representation of numbers in $G_m(\mathbf{N}_d,\vec{a})$.

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

$$\text{(5A-1)} \quad B_m(\vec{a}) = B_m^d(\vec{a}) = \Big\{\frac{x_0+x_1a_1+\cdots+x_na_n}{d^m} \in \mathbb{N} \\ : x_0,\dots,x_n \in \mathbb{Z} \text{ and } |x_i| \leq d^{2m}, i \leq n\Big\}.$$

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

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

5A.2. **Lemma** (**Lin**$_d$-inclusion)**.** *For all $\vec{a} \in \mathbb{N}^n$ and all m:*

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

(2) *For every primitive* $\phi : \mathbb{N}^k \to \mathbb{N}$ *in* $\mathbf{Lin}_d$,

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

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

Proof. We take $n = 2$ to simplify the formulas, the general argument being only a notational variant.

(1) The first inclusion holds because $a_i = \dfrac{d^m a_i}{d^m}$ and the second because

$$\frac{x_0 + x_1 a_1 + x_2 a_2}{d^m} = \frac{dx_0 + dx_1 a_1 + dx_2 a_2}{d^{m+1}}$$

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

(2) Clearly $0, \dots, d \in B_m(\vec{a})$ for every $m \geq 1$, and so the constants stay in $B_m(\vec{a})$ once they get in.

For addition, let $x, y \in B_m(\vec{a})$, so

$$\begin{aligned} x + y &= \frac{x_0 + x_1 a_1 + x_2 a_2}{d^m} + \frac{y_0 + y_1 a_1 + y_2 a_2}{d^m} \\ &= \frac{d(x_0 + y_0) + d(x_1 + y_1)a_1 +_1 + d(x_2 + y_2)a_n}{d^{m+1}} \end{aligned}$$

and the coefficients in the numerator satisfy

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

The same works for arithmetic subtraction. Finally, for integer division by d, if $i = \mathrm{rem}_d(x) < d$, then

$$\mathrm{iq}_d(x) = \frac{1}{d}(x - i) = \frac{(x_0 - id^m) + x_1 a_1 + x_2 a_2}{d^{m+1}} \text{ for some } 1 \leq i < d$$

and this number is in $B_{m+1}(\vec{a})$ as above.

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

5A.3. **Proposition** (Multiplication from $\mathbf{Lin}_d$). *For every number* $a \geq 2$,

$$\mathrm{depth}(a^2; \mathbf{N}_d, a) \geq \frac{1}{\log d} \log\Big(\frac{a^2}{a+1}\Big).$$

Proof. It is enough to show that for $a \geq 2$,

$$a^2 \in G_m(\mathbf{N}_d, a) \Longrightarrow m \geq \frac{1}{\log d} \log\Big(\frac{a^2}{a+1}\Big),$$

so assume that $a^2 \in G_m(a)$. By Lemma 5A.2, there exist $x_0, x_1 \in \mathbb{Z}$ such that $|x_0|, |x_1| \leq d^{2m}$ and

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

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

$$d^m \ge \frac{a^2}{a+1} \qquad \dashv$$

Similar arguments can be used to establish value-depth lower bounds from $\mathbf{Lin}_d$ for all functions which grow faster than x.

Problems for Section 5A

x5A.1. **Problem.** Prove that the structures

$$\mathbf{N}_b = (\mathbb{N}, 0, \mathrm{parity}, \mathrm{iq}_2, \mathrm{em}_2, \mathrm{om}_2, \mathrm{eq}_0),$$
$$\mathbf{N}_{\mathrm{st}} = (\mathbb{N}, \mathrm{parity}, \mathrm{em}_2, \mathrm{iq}_2, \dot{-}, =, <)$$

of binary arithmetic and the Stein algorithm are explicitly reducible to $\mathbf{N}_2$.

x5A.2. **Problem.** Prove that $\mathrm{rem}_d(x)$ is explicit in $\mathbf{N}_d$.

x5A.3*. **Problem.** Prove that for all $m, n \ge 2$:
(i) $\mathrm{iq}_m \in \mathbf{expl}(\mathbf{N}_{mn})$;
(ii) $\mathrm{iq}_{mn} \in \mathbf{expl}(\mathbb{N}, 0, 1, \dots, d, +, \dot{-}, \mathrm{iq}_m, \mathrm{iq}_n, =, <)$.
Infer that if $\mathbf{\Phi}$ is any finite set of Presburger primitives, then the structure

$$(\mathbb{N}, 0, 1, +, \dot{-}, <, =, \mathbf{\Phi})$$

is explicitly equivalent with some $\mathbf{N}_d$. (For the definition of the Presburger primitives see (1C-17).)

x5A.4. **Problem.** Specify a system of two recursive equations

$$q(x,y) = E_q(x,y,q,r)$$
$$r(x,y) = E_r(x,y,q,r),$$

in the vocabulary $\mathbf{Lin}_2 \cup \{q, r\}$, such that in $\mathbf{N}_2$,

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

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

$$c_q^p(x,y) \le B \log x, \quad c_r^p(x,y) \le B \log x.$$

(With the appropriate head equations, this system defines two programs from $\mathbf{Lin}_2$, one for $\mathrm{iq}(x,y)$ and the other for $\mathrm{rem}(x,y)$.)

x5A.5. **Problem.** Show that the recursive program for the integer quotient function in Problem x5A.4 is weakly optimal from $\mathbf{Lin}_d$, for any $d \ge 2$.

x5A.6*. **Problem.** Prove that for every $d \geq 2$, there is an $r > 0$ and infinitely pairs of numbers (a, b) and every $d \geq 2$,

$$\operatorname{depth}(\operatorname{rem}(a,b); \mathbf{Lin}_d, a, b) > r \log(\max(a,b)).$$

Infer that the recursive program for $\operatorname{rem}(x, y)$ in Problem x5A.4 is weakly optimal for $\operatorname{rem}(x, y)$ in every $\mathbf{N}_d$. (Note: An easier proof of an $O(\log \max(a, b))$ lower bound for $\operatorname{depth}_{\mathrm{rem}}(\mathbf{N}_d, a, b)$ can be given using the Homomorphism Test, see Problem x5B.1. This proof that uses value-depth complexity requires a simple divisibility argument and is due to Tim Hu.)

x5A.7. **Problem.** Define a weakly optimal program from $\mathbf{Lin}_d$ which computes the exponential function $f(x, y) = x^y$ (with $0^0 = (x + 1)^0 = 1$).

5B. Primality from $\mathbf{Lin}_d$

To establish lower bounds from $\mathbf{Lin}_d$ for decision problems, we need to complement Lemma 5A.2 with a uniqueness result.

5B.1. **Lemma** ($\mathbf{Lin}_d$-Uniqueness). *If $x_i, y_i \in \mathbb{Z}$, $|x_i|, |y_i| < \dfrac{a}{2}$ 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 applied to λa.

The second equivalence easily implies the first one, and follows from the following fact applied to $(x_0 - y_0) + (x_1 - y_1)a$:

If $x, y \in \mathbb{Z}$ 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 $x + ya$ has the same sign as ya, which has the same sign as y. $\dashv$

5B.2. **Lemma** ($\mathbf{Lin}_d$-embedding). *Suppose $d^{2m+2} < a$, and let $\lambda > 1$ be any number such that $d^{m+1} \mid \lambda - 1$; then there exists an embedding*

$$\pi : \mathbf{G}_m(\mathbf{N}_d, a) \rightarrowtail \mathbf{N}_d,$$

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

Proof. By Lemma 5B.1 and part (3) of Lemma 5A.2, the equation

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

defines a map $\pi : \mathbf{G}_m(\mathbf{N}_d, a) \to \mathbb{Q}$, since

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

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

$$x_0 + \lambda x_1 a = x_0 + x_1 a + (\lambda - 1)x_1 a, \tag{5B-1}$$

so that if $d^m \mid (x_0 + x_1 a)$, then also $d^m \mid (x_0 + \lambda x_1 a)$ since $d^m \mid (\lambda - 1)$ by the hypothesis. It is injective and order-preserving, by Lemma 5B.1 again, applied to both a and λa.

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

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

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

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

and $|x_0 + y_0|, |x_1 + y_1| \le 2 \cdot d^{2m} \le d^{2m+1} < \frac{a}{2}$, and so by the Uniqueness Lemma 5B.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 arithmetic subtraction.

Finally, for division by d, suppose

$$X = \frac{x_0 + x_1 a}{d} = d\,\mathrm{iq}_d(X) + i \quad (i < d)$$

where $|x_0|, |x_1| \le d^{2m}$ as above, so that

$$\mathrm{iq}_d(X) = \frac{1}{d}\Big(\frac{x_0 + x_1 a}{d^m} - i\Big) = \frac{x_0 - id^m + x_1 a}{d^{m+1}} = Z = \frac{z_0 + z_1 a}{d^m}$$

for suitable z_0, z_1 with $|z_0|, |z_1| \le d^{2m}$, if $Z \in G_m$. These two representations of Z must now be identical since $|dz_i| \le dd^{2m} = d^{2m+1} < \frac{a}{2}$, and

$$\begin{aligned}|x_0 - id^m| \le d^{2m} + id^m < d^{2m} + d^{m+1}\\ = d^m(d^m + d) \le d^m d^{m+1} = d^{2m+1} < \frac{a}{2}.\end{aligned}$$

So $x_0 - id^m = dz_0$ and $x_1 = dz_1$. These two equations imply that

$$\frac{x_0 + x_1\lambda a}{d^m} = d\frac{z_0 + z_1 \lambda a}{d^m} + i$$

which means that

$$\mathrm{iq}_2(\pi X) = \frac{z_0 + z_1\lambda a}{d^m} = \pi(Z) = \pi(\mathrm{iq}_d(X))$$

as required. ⊣

5B.3. **Theorem.** *For every prime number p,*

$$\mathrm{depth}_{\mathrm{Prime}}(\mathbf{N}_d, p) \geq \frac{1}{4\log d}\log p.$$

PROOF. Let $m = \mathrm{depth}_{\mathrm{Prime}}(\mathbf{N}_d, p)$ and suppose that

$$d^{2m+2} < p. \tag{5B-2}$$

Lemma 5B.2 guarantees an embedding

$$\pi : \mathbf{G}_m(\mathbf{N}_d, p) \rightarrowtail \mathbf{N}_d$$

with $\lambda = 1 + d^{m+1}$ such that $\pi p = \lambda p$, and this π does not respect the primality relation at p, which is absurd. So (5B-2) fails, and so (taking logarithms and using the fact that $m \geq 1$ by Problem x3A.2),

$$4m \log d \geq (2m+2)\log d \geq \log p. \qquad \dashv$$

Problems for Section 5B

x5B.1. **Problem.** Suppose $e \geq 2$, set

$$|_e\,(x) \iff e \mid x \iff \mathrm{rem}_e(x) = 0$$

and assume that $e \perp\!\!\!\perp d$.

(a) Prove that for all a which are not divisible by e,

$$\mathrm{depth}_{|_e}(\mathbf{N}_d, a) \geq \frac{1}{4\log d}\log a.$$

(b) For some $r > 0$ and all a which are not divisible by e,

$$\mathrm{depth}(\mathbf{N}_d, \mathrm{iq}_e, a) > r\log a, \quad \mathrm{depth}(\mathbf{N}_d, \mathrm{rem}_e, a) > r\log a.$$

In particular, if e is coprime with d, then the relation $|_e\,(x)$ is not explicit in $\mathbf{N}_d$; the divisibility relation $x \mid y$ is not explicit in any $\mathbf{N}_d$; and the recursive programs for $\mathrm{iq}(x,y)$ and $\mathrm{rem}(x,y)$ in Problem x5A.4 are weakly optimal in $\mathbf{N}_2$ and in every $\mathbf{N}_d$ (such that $2 \mid d$, so they can be expressed).

HINT: Use the fact that if $x \perp\!\!\!\perp y$, then there are constants $A \in \mathbb{Z}$ and $B \in \mathbb{N}$ such that $1 = Ax - By$.

5C. Good examples: perfect square, square-free, etc.

The method in the preceding section 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

$$\lambda(\mu) = 1 + l_1\mu + l_2\mu^2 + \cdots + l_s\mu^s \tag{5C-1}$$

with coefficients in $\mathbb{N}$, constant term 1 and degree > 0 and for all $\mu \geq 1$,

$$R(x) \Longrightarrow \neg R(\lambda(\mu)x). \tag{5C-2}$$

For example, primality is good, taking $\lambda(\mu) = 1 + \mu$, and being a power of 2 is good with $\lambda(\mu) = 1 + 2\mu$. We leave for the problems several interesting results of this type.

Problems for Section 5C

x5C.1. **Problem.** Design a weakly optimal recursive program from $\mathbf{Lin}_d$ for the relation

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

x5C.2. **Problem** (van den Dries and Moschovakis [2004]). Prove that if $R(x)$ is a good example, then for all $a \geq 2$,

$$R(a) \Longrightarrow \text{depth}_R(\mathbf{N}_d, a) \geq \frac{1}{4\log d}\log a$$

x5C.3. **Problem.** 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 $m \perp\!\!\!\perp n$.

x5C.4. **Problem.** Prove that the following two relations are good examples:

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

x5C.5. **Problem.** Prove that if $\lambda(\mu)$ is as in (5C-1), then there is a constant C such that

$$\log \lambda(\mu) \leq C \log \mu \quad (\mu \geq 2). \tag{5C-3}$$

The next problem gives a logarithmic lower bound for $\text{depth}_R(\mathbf{N}_d, a)$ with good R at many points where $R(a)$ fails to hold.

x5C.6. **Problem.** Suppose $R(x)$ is a good example with associated polynomial $\lambda(\mu)$. Prove that there is a rational constant $r > 0$, such that for all $a \geq 2$ and $m \geq 1$,

$$R(a) \Longrightarrow \operatorname{depth}_R(\mathbf{N}_d, \lambda(d^{m+1})a) \geq r \log(\lambda(d^{m+1})a).$$

5D. Stein's algorithm is weakly optimal from $\mathbf{Lin}_d$

We extend here (mildly) the methods in the preceding section so they apply to binary functions, and we show the result in the heading.

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

5D.1. **Lemma.** *Suppose $a > 2$ and set $b = a^2 - 1$.*
(1) *$a \perp\!\!\!\perp b$, 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 $d^{2m+3} < a$ and $\lambda = 1 + d^{m+1}$, then there is an embedding*

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

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

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

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. If $z = 0$, then the result follows from Lemma 5B.1, so assume $z \neq 0$ and compute:

$$x + y\lambda a + z\lambda b = x + y\lambda a + z\lambda(a^2 - 1) = (x - \lambda z) + y\lambda a + z\lambda a^2.$$

Now

$$\Big|(x - \lambda z) + y\lambda a\Big| = \lambda\Big|(\frac{x}{\lambda} - z) + ya\Big| < \lambda(a + \frac{a^2}{2}) < \lambda a^2 \leq \lambda |z| a^2,$$

and so $x + y\lambda a + z\lambda b$ and $\lambda z a^2$ have the same sign, which is the sign of z.

(2) Assume $d^{2m+3} < a$, set $\lambda = 1 + d^{m+1}$, and notice that $d^{2m} < \frac{1}{4}a$, so as in Lemma 5B.2, we can define the required embedding by

$$\pi\Big(\frac{x_0 + x_1 a + x_2 b}{d^m}\Big) = \frac{x_0 + x_1\lambda a + x_2\lambda b}{d^m}$$

$$(|x_0|, |x_1|, |x_2| \leq d^{2m}, \frac{x_0 + x_1 a + x_2 b}{d^m} \in G_m(a,b)),$$

using now (1) instead of Lemma 5B.1. ⊣

5D.2. **Theorem** (van den Dries and Moschovakis [2004]). *For all $a > 2$,*

$$\text{depth}_{\perp\!\!\!\perp}(\mathbf{N}_d, a, a^2-1) > \frac{1}{10\log d}\log(a^2-1).$$

Proof. Let $m = \text{depth}_{\perp\!\!\!\perp}(\mathbf{N}_0, a, a^2-1)$ for some $a > 2$. Since λa and $\lambda(a^2-1)$ are not coprime, part (2) of the preceding Lemma 5D.1 and the Homomorphism Test 4F.2 imply that

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

taking the logarithms of both sides and using the fact that $m \geq 1$ (by Problem x3A.2), we get

$$5m\log d \geq (2m+3)\log d \geq \log a;$$

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

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

5D.3. **Corollary.** *Let E be the recursive program of $\mathbf{N}_2$ which decides $a \perp\!\!\!\perp b$ by adding to the Stein algorithm one step checking* $\gcd(a,b) = 1$. *For each $d \geq 2$, there is a constant $K > 0$ such that for all $a > 2$,*

$$l^s_E(a, a^2-1) \leq K\text{depth}_{\perp\!\!\!\perp}(\mathbf{N}_d, a, a^2-1).$$

In particular, the Stein algorithm is weakly optimal for coprimeness from Presburger primitives, for both the depth and calls complexity measures.

Proof. Choose K_1 such that

$$l^s_E(a,b) \leq K_1(\log a + \log b) \quad (a, b > 2),$$

and compute for $a > 2$:

$$l^s_E(a, a^2-1) < 2K_1\log(a^2-1) < 20\log dK_1\text{depth}_{\perp\!\!\!\perp}(a, a^2-1). \qquad ⊣$$

Problems for Section 5D

Problem x5A.4 defines a recursive program from $\mathbf{N}_2$ which computes rem(x, y) with $O(\log)$ complexity. The next problem claims that it is weakly optimal from Presburger primitives—and a little more.

x5D.1. **Problem.** Prove that for each $d \geq 2$, there is a rational $r > 0$, such that

$$\text{for infinitely many pairs } (x, y), x \mid y \text{ and } \text{depth}_{|}(\mathbf{N}_d, x, y) \geq r \log y.$$

Infer that the recursive program for the remainder in Problem x5A.4 is weakly optimal from $\mathbf{Lin}_d$.

Hint: Show that when a, b, λ and μ satisfy suitable conditions, then the mapping

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

is an embedding on $\mathbf{N}_d \upharpoonright G_m(a, b)$.

Busch [2007], [2009] has used "asymmetric" embeddings of this kind to derive lower bounds for several problems in number theory and algebra that are related to the Stein algorithm.

CHAPTER 6

LOWER BOUNDS FROM DIVISION WITH REMAINDER

We now add to the basic primitives of the Presburger structures *division with remainder*, i.e., the integer quotient and remainder operations. Set:

$$\mathbf{Lin}_0 = \{0, 1, =, <, +, \dot{-}\}, \quad \mathbf{N}_0 = (\mathbb{N}, \mathbf{Lin}_0),$$
$$\mathbf{Lin}_0[\div] = \mathbf{Lin}_0 \cup \{\mathrm{iq}, \mathrm{rem}\} = \{0, 1, =, <, +, \dot{-}, \mathrm{iq}, \mathrm{rem}\},$$
$$\mathbf{N}_0[\div] = (\mathbb{N}, \mathbf{Lin}_0[\div]) = (\mathbb{N}, 0, 1, =, <, +, \dot{-}, \mathrm{iq}, \mathrm{rem}).$$

Every expansion of a Presburger structure by division with remainder is obviously explicitly equivalent to $\mathbf{N}_0[\div]$, and so all the results of this chapter apply to these richer structures with only inessential changes in the constants.

The derivations of absolute lower bounds for unary relations from $\mathbf{Lin}_0[\div]$ is similar to those from $\mathbf{Lin}_d$ in Sections 5B and 5C and we will consider it first. For binary relations, however, like coprimeness, some new ideas are required as well as some elementary results from number theory.

6A. Unary relations from $\mathbf{Lin}_0[\div]$

We start with a representation of the numbers in $G_m(\mathbf{N}_0[\div], \vec{a})$ similar to that for $G_m(\mathbf{N}_d, \vec{a})$ in Lemma 5A.2, except that we cannot keep the denominators independent of $\vec{a}$.

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

(6A-1) $$C(\vec{a}; h) = \Big\{ \frac{x_0 + x_1 a_1 + \cdots + x_n a_n}{x_{n+1}} \in \mathbb{N} : x_0, \dots, x_{n+1} \in \mathbb{Z}, \\ x_{n+1} > 0 \text{ and } |x_0|, \dots, |x_{n+1}| \leq h \Big\}.$$

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

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

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

6A.1. **Lemma.** *For all $\vec{a} = (a_1, \dots, a_n), h \geq 2$ and every k-ary operation in $\mathbf{Lin}_0$,*

$$X_1, \dots, X_k \in C(\vec{a}; h) \Longrightarrow f(X_1, \dots, X_k) \in C(\vec{a}; h^3).$$

Proof is by direct computation. For example (taking $n = 2$ to keep the notation simple),

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

and for the typical coefficient,

$$|y_3 x_0 + x_3 y_0| \leq h^2 + h^2 = 2h^2 \leq h^3. \quad \dashv$$

There is no simple, general result of this type for division with remainder, and in this section we will consider only the simplest case $n = 1$, when $C(a; h)$ comprises the numbers of height h with respect to a single a. We start with the appropriate version of Lemma 5B.1.

6A.2. **Lemma** ($\mathbf{Lin}_0[\div]$-Uniqueness). *If $|x_i|, |y_i| \leq h$ for $i \leq 2$, $\lambda \geq 1$ 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 with no common factor other than 1 such that

$$x = \frac{x_0 + x_1 a}{x_2} \quad (|x_0|, |x_1|, |x_2| \leq h). \tag{6A-2}$$

Proof of the two equivalences is immediate from Lemma 5B.1, since

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

The uniqueness of relatively prime x_0, x_1, x_2 which satisfy (6A-2) and these equivalences requires a simple divisibility argument, Problem x6A.1. $\dashv$

We will sometimes save a few words by referring to (6A-2) as the *canonical representation* of x in $C(a; h)$, when $2h^2 < a$.

6A.3. **Lemma.** *If $x, y \in C(a;h)$, $x \geq y > 0$, $h \geq 2$ 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

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

and consider two cases.

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

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

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

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

and the (potentially) highest coefficient in this expression is

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

Case 2, $y_1 \neq 0$. We must now have $y_1 > 0$, otherwise $y < 0$ by Lemma 6A.2. Moreover,

$$y \cdot (2x_1 y_2) = \frac{2y_0 x_1 y_2 + 2y_1 x_1 y_2 a}{y_2} > \frac{x_0 + x_1 a}{x_2} \geq yq$$

by the same Lemma, because $|y_2 x_1| < 2|x_2 y_1 x_1 y_2|$, and the Lemma applies since (for the potentially largest coefficient),

$$2|2y_1 x_1 y_2|^2 \leq 2^3 h^6 \leq h^9 < a.$$

It follows that

$$q = \mathrm{iq}(x,y) \leq 2x_1 y_2 \leq h^3,$$

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

$$r = \mathrm{rem}(x,y) = \frac{(y_2 x_0 - x_2 y_1 q) + (y_2 x_1 - x_2 y_1 q)a}{x_2 y_2}.$$

The (potentially) highest coefficient in this expression is

$$|y_2 x_0 - x_2 y_1 q| \leq h^2 + h^2 h^3 \leq h^6. \qquad \dashv$$

6A.4. **Lemma.** *For every m, if*

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

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 $G_0(a) = \{a\} \subseteq C(a; 2)$. 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 6A.1 and 6A.3, the value of any operation in $\mathbf{Lin}_0[\div]$ with arguments in $C(a; h)$ is in $C(a; h^6)$, and

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

6A.5. **Lemma** ($\mathbf{Lin}_0[\div]$-embedding)**.** *If $G_m(a) = G_m(\mathbf{N}_0[\div], a)$, and*

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

then there is an embedding

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

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 6A.2, each $x \in C(a; h)$ can be expressed uniquely in the form

$$x = \frac{x_0 + x_1 a}{x_2}$$

with all the coefficients $\leq h$. 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 in $\mathbb{N}$, since

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

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

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

by the hypothesis that $a! \mid (\lambda - 1)$. By another appeal to Lemma 6A.2, we verify that ρ is an injection, and it is order-preserving.

The required embedding 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,

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

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

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

even if $\mathrm{rem}(x, y) \notin G_m(a)$, 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 6A.3. 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,

(6A-3) $$\rho(x) = \rho(y)\rho(q) + \rho(r).$$

Moreover, ρ is order-preserving, so that

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

and (6A-3) is the correct division equation for $\rho(x), \rho(y)$. Thus

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

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

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

independently of whether $\text{rem}(x, y) \in G_m(a)$ or not. The same argument works for $\pi(\text{rem}(x, y))$ 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 5C.

6A.6. **Theorem.** *If $R(x)$ is a good example, then for all $a \geq 2$*

$$R(a) \Longrightarrow \text{depth}_R(\mathbf{N}_0[\div], a) > \frac{1}{12} \log \log a.$$

Proof. Suppose $R(a)$, let $m = \text{depth}_R(\mathbf{N}_0[\div], a)$, 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 6A.5 guarantees an embedding

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

with $\pi a = \lambda a$; and since $\neg R(\lambda a)$, the Homomorphism Test 4F.2 yields a contradiction, so that

$$2^{6^{m+3}} \geq a. \tag{6A-4}$$

Taking logarithms twice, we get from this

$$m + 3 \geq \frac{\log \log a}{\log 6};$$

and since $m \geq 1$ by Problem x3A.2, $4m \geq m + 3$, so that we get the required

$$m \geq \frac{\log \log a}{4 \log 6} > \frac{\log \log a}{12}.$$ ⊣

Problems for Section 6A

x6A.1. **Problem.** Prove that if $x \in C(a; h)$ and $2h^2 < a$, then (6A-2) holds for uniquely determined, relatively prime x_0, x_1, x_2. Hint: By an (easy) extension of Bezout's Lemma, Problem x1D.14,

$$\gcd(x_0, x_1, x_2) = \alpha x_0 + \beta x_1 + \gamma x_2 \quad (\text{for some } \alpha, \beta, \gamma \in \mathbb{Z}).$$

Use this and the equivalences in Lemma 6A.2.

6B. 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 6A.2 for canonical forms involving two numbers, when these satisfy certain conditions. Those with some knowledge of number theory know these—in fact they probably know better proofs of them, which establish more; they should peruse the section quickly, just to get the terminology that we will be using—especially the definition of *difficult pairs*, which is not standard.

6B.1. **Theorem** (Pell pairs). *The pairs $(x_n, y_n) \in \mathbb{N}^2$ defined by the recursion*

$$(x_1, y_1) = (3, 2), \qquad (x_{n+1}, y_{n+1}) = (3x_n + 4y_n, 2x_n + 3y_n) \tag{6B-1}$$

satisfy Pell's equation

$$x_n^2 - 2y_n^2 = 1, \tag{6B-2}$$

and the inequalities

$$2^n \le 2 \cdot 5^{n-1} \le y_n < x_n \le 7^{n+1}, \tag{6B-3}$$

$$0 < \frac{x_n}{y_n} - \sqrt{2} < \frac{1}{2y_n^2}. \tag{6B-4}$$

Proof. Equation (6B-2) 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 (6B-3), we first check that $2^n \le 2 \cdot 5^{n-1}$ by a trivial induction on $n \ge 1$, and then, inductively again,

$$y_{n+1} = 2x_n + 3y_n \ge 5y_n \ge 5 \cdot 2 \cdot 5^{n-1} = 2 \cdot 5^n.$$

The last part of the triple inequality is proved similarly:

$$x_{n+1} = 3x_n + 4y_n \le 7x_n \le 7 \cdot 7^n = 7^{n+1}.$$

The crucial, last inequality (6B-4) 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 $\frac{x}{y} > \sqrt{2}$, 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

$$0 < \frac{x}{y} - \sqrt{2} = \frac{1}{(\frac{x}{y} + \sqrt{2})y^2} < \frac{1}{2y^2}. \qquad \dashv$$

In fact, the pairs (x_n, y_n) defined in (6B-1) comprise all positive solutions of Pell's equation, cf. Problem x6B.1.

Good approximations of irrationals. A pair of numbers (a, b) (or the proper fraction $\frac{a}{b}$) is a *good approximation of* an irrational number ξ, if $a \perp\!\!\!\perp b$ and

$$\left|\frac{a}{b} - \xi\right| < \frac{1}{b^2}. \tag{6B-5}$$

Theorem 6B.1 asserts in part that there are infinitely many good approximations of $\sqrt{2}$. This is true of all irrational numbers, and it is worth understanding it in the context of what we are doing, although we will never need it in its full generality.

6B.2. **Theorem** (Hardy and Wright [1938], 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 Hardy and Wright) is due to Dirichlet.

Proof. For any real number ξ, let

$$\lfloor \xi \rfloor = \text{the largest natural number } \leq \xi.$$

This $\lfloor \xi \rfloor$ is the *house*, and $\xi - \lfloor \xi \rfloor$ is its *fractional part*, for which, clearly

$$0 \leq \xi - \lfloor \xi \rfloor < 1.$$

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

$$n \geq 1,$$

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 $n \geq 1$, we get

$$0 \leq j < k \leq n$$

such that

$$\Big| j\xi - \lfloor j\xi \rfloor - (k\xi - \lfloor k\xi \rfloor)\Big| < \frac{1}{n};$$

and setting $y = k - j$, $x = \lfloor k\xi \rfloor - \lfloor j\xi \rfloor$, we get

$$\Big| x - y\xi \Big| < \frac{1}{n}.$$

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

$$\Big| \frac{x}{y} - \xi \Big| < \frac{1}{ny} < \frac{1}{y^2}.$$

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

$$\Big| \frac{x_1}{y_1} - \xi \Big| < \frac{1}{y_1^2} \text{ and } \Big| \frac{x_1}{y_1} - \xi \Big| < \frac{1}{my_1} \leq \frac{1}{m} < \Big| \frac{x}{y} - \xi \Big|,$$

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. $\dashv$

Next comes the most important result we need, which says, in effect, that algebraic irrational numbers cannot have "too good" approximations.

6B.3. **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 $n \geq 2$, and let*

$$c = \lceil \sup\{|f'(x)| \mid |x - \xi| \leq 1\} \rceil .$$

It follows that for all pairs (x, y) of relatively prime integers,

$$\Bigl|\xi - \frac{x}{y}\Bigr| > \frac{1}{cy^n}.$$

In particular, for all relatively prime (x, y),

$$\Bigl|\sqrt{2} - \frac{x}{y}\Bigr| > \frac{1}{5y^2}.$$

Proof. We may assume that $|\xi - \frac{x}{y}| \leq 1$, since the desired inequality is trivial in the opposite case. Using the fact that $f(\xi) = 0$ and the Mean Value Theorem, we compute, for any $\frac{x}{y}$ within 1 of ξ,

$$|f(\frac{x}{y})| = |f(\xi) - f(\frac{x}{y})| \leq c|\xi - \frac{x}{y}|.$$

Moreover, $f(\frac{x}{y}) \neq 0$, 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

$$1 \leq |y^n f(\frac{x}{y})| \leq y^n c|\xi - \frac{x}{y}|,$$

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

$$c = \lceil \sup\{2x \mid |\sqrt{2} - x| \leq 1\} \rceil = \lceil 2(\sqrt{2} + 1) \rceil = 5. \qquad \dashv$$

Liouville's Theorem implies that *good approximations of a non-rational algebraic number cannot be too-well approximated by fractions with a much smaller denominator.* We formulate precisely the special case of this general fact that we need.

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

$$2 \leq b < a < 2b, \tag{6B-6}$$

and for all y, z,

$$0 < |z| < \frac{b}{\sqrt{10}} \implies \Big|\frac{a}{b} - \frac{y}{z}\Big| > \frac{1}{10z^2}. \tag{6B-7}$$

6B.4. **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,

$$0 < |z| < \frac{b}{\sqrt{10}} \implies |za + yb| > \frac{b}{10|z|}. \tag{6B-8}$$

Proof. (1) Let (a, b) be a good approximation of $\sqrt{2}$ with $b \geq 2$. Then (6B-6) follows from

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

To prove (6B-7), suppose $0 < |z| < \dfrac{b}{\sqrt{10}}$, and use Liouville's Theorem 6B.3:

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

The result holds for all solutions of Pell's equation because the proof of (6B-4) was based only on the hypothesis $x^2 = 1 + 2y^2$.

(2) is very useful and easy: assuming the hypothesis of (6B-8),

$$|za + yb| = |z|b\Big|\frac{a}{b} + \frac{y}{z}\Big| > |z|b\frac{1}{10z^2} = \frac{b}{10|z|}. \qquad \dashv$$

We leave for the problems the similar proof that pairs (F_{k+1}, F_k) of successive Fibonacci numbers with $k \geq 3$ are also difficult.

Problems for Section 6B

x6B.1. **Problem.** Prove that the pairs of numbers (x_n, y_n) defined in the proof of Theorem 6B.1 comprise all the positive solutions of the Pell equation $a^2 = 2b^2 + 1$.

Recall from the problems of Section 1 that

$$\varphi = \frac{1+\sqrt{5}}{2}, \quad \hat{\varphi} = \frac{1-\sqrt{5}}{2}$$

are the two solutions of the quadratic equation $x + 1 = x^2$, so that

$$1 < \varphi < 2, \quad \hat{\varphi} < 0, \quad |\hat{\varphi}| < 1,$$

and that the Fibonacci numbers are explicitly defined in terms of these,

$$F_k = \frac{\varphi^k - \hat{\varphi}^k}{\sqrt{5}}.$$

x6B.2. **Problem.** Show that for $k \geq 1$,

$$\text{if } k \text{ is even, then } \varphi < \frac{F_{k+1}}{F_k}, \text{ and if } k \text{ is odd, then } \frac{F_{k+1}}{F_k} < \varphi.$$

Hint: Use the equation

$$\frac{F_{k+1}}{F_k} = \varphi R(k) \text{ where } R(k) = \frac{1 - \frac{\hat{\varphi}^{k+1}}{\varphi^{k+1}}}{1 - \frac{\hat{\varphi}^k}{\varphi^k}},$$

and compute the sign and size of $R(k)$ for odd and even k.

x6B.3. **Problem.** Show that for all $n \geq 1$,

$$F_{n+1}F_{n-1} - F_n^2 = (-1)^n. \tag{6B-9}$$

Infer that

$$\left|\frac{F_{n+1}}{F_n} - \varphi\right| < \frac{1}{F_n^2} \quad (n \geq 1).$$

x6B.4. **Problem.** Prove that for every $n \geq 3$, the pair (F_{n+1}, F_n) is a difficult pair.

Hint: The golden mean φ is a root of the polynomial $f(x) = x^2 - x - 1$. Use Liouville's Theorem 6B.3 to show that for all coprime x, y,

$$\left|\frac{x}{y} - \varphi\right| > \frac{1}{5y^2},$$

and then imitate the proof of (1) of Lemma 6B.4 with φ in place of $\sqrt{2}$.

The definition of *difficult pair* is tailor made for 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, cf. Problems x6B.5 and x6C.1*.

x6B.5. **Problem.** Suppose $\xi > 1$ is irrational, $C > 0$, $a \perp\!\!\!\perp b$, $2 \leq b$ and

$$(*) \qquad \frac{1}{Cb^2} < \left|\xi - \frac{a}{b}\right| < \frac{1}{b^2}.$$

Let $\lfloor \xi \rfloor = M \geq 1$, so that

$$1 \leq M < \xi < M + 1.$$

Show that:

(1) $a < (M+2)b$.

(2) For all $z, y \in \mathbb{Z}$,

$$0 < |z| < \frac{b}{\sqrt{2C}} \Longrightarrow \left|\frac{a}{b} - \frac{y}{z}\right| > \frac{1}{2Cz^2} \Longrightarrow |za - yb| > \frac{b}{2C|z|}.$$

Show also that for every quadratic irrational $\xi > 1$, $(*)$ holds for infinitely many coprime pairs a, b.

x6B.6. **Problem** (Liouville)**.** Prove that the following number is *transcendental* (i.e., not algebraic):

$$\text{(6B-10)} \qquad \xi = \sum_{i=1}^{\infty} \frac{1}{10^{i!}} = \frac{1}{10} + \frac{1}{10^{2!}} + \frac{1}{10^{3!}} + \frac{1}{10^{4!}} + \cdots$$

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.

6C. The complexity of coprimeness from $\mathbf{Lin}_0[\div]$

We can now combine the methods from Sections 5D and 6A, to derive a double-log lower bound for coprimeness from $\mathbf{Lin}_0[\div]$. The key is the following Uniqueness Lemma for linear combinations of a difficult pair.

6C.1. **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*

$$\begin{aligned}&\frac{x_0 + x_1\lambda a + x_2\lambda b}{x_3} = \frac{y_0 + y_1\lambda a + y_2\lambda b}{y_3}\\ &\qquad\qquad\iff [y_3x_0 = x_3y_0 \;\&\; y_3x_1 = x_3y_1 \;\&\; y_3x_2 = x_3y_2],\end{aligned}$$

$$\begin{aligned}&\frac{x_0 + x_1\lambda a + x_2\lambda b}{x_3} > \frac{y_0 + y_1\lambda a + y_2\lambda b}{y_3}\\ &\qquad\qquad\iff [y_3(x_1a + x_2b) > x_3(y_1a + y_2b)]\\ &\qquad\qquad\qquad \text{or } \Big([y_3(x_1a + x_2b) = x_3(y_1a + y_2b)]\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \& \; y_3x_0 > x_3y_0\Big).\end{aligned}$$

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* $|x|, |y|, |z| < \frac{\sqrt{b}}{\sqrt{10}}$, *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 (6B-8) 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* $|x|, |y|, |z| < \frac{\sqrt{b}}{\sqrt{10}}$, *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 5B.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. ⊣

6C.2. **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

$$h^2 < \frac{\sqrt{b}}{2\sqrt{10}}. \tag{6C-1}$$

This allows us to appeal to Lemma 6C.1 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 6C.1, 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

$$\frac{x_0 + x_1 a + x_2 b}{x_3} = \frac{y_0 + y_1 a + y_2 b}{y_3} Q + R \quad (0 \leq R < Y). \tag{6C-2}$$

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

Case 1, $y_1 a + y_2 b = 0$. Now (6C-2) 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 (6C-2) for Q, we get in this case

$$Q = \frac{y_3}{y_0} \frac{(x_0 - x_3 R) + x_1 a + x_2 b}{x_3} \in C(a, b; h^4). \tag{6C-3}$$

Case 2, $y_1 a + y_2 b \neq 0$. Then $y_1 a + y_2 b > 0$, by Lemma 6C.1, since $Y > 0$, using (6C-1). We are going to show that in this case

$$h^9 Y > X \tag{6C-4}$$

so that $Q \leq h^9$. Assuming this, we can solve the division equation (6C-2) for R, to get

$$R = \frac{(x_0 y_3 - y_0 x_3 Q) + (x_1 y_3 - y_1 x_3 Q) a + (x_2 y_3 - y_2 x_3 Q) b}{x_3 y_3}; \tag{6C-5}$$

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

We show (6C-4) by separately comparing the "infinite parts" (those involving a and b) of X and Y with b. Compute first:

$$y_3(x_1 a + x_2 b) \leq |y_3 x_1| a + |y_3 x_2| b \leq h^2 2b + h^2 b = 3h^2 b \leq h^4 b, \tag{6C-6}$$

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 (6B-8),

$$(y_1 a + y_2 b) > \frac{b}{10|y_1|}, \quad \text{so that} \quad 10|y_1|(y_1 a + y_2 b) > b,$$

and hence (since $10 < 2^4$), in either case,

$$h^5(y_1 a + y_2 b) > b. \tag{6C-7}$$

Now (6C-6) and (6C-7) 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 (6C-4) with an appeal to Lemma 6C.1, provided that the coefficients of $h^9 Y$ and X in canonical form satisfy the hypotheses of this Lemma. For the worst case, the required inequality is

$$|x_3 h^9 y_i| \leq \frac{\sqrt{b}}{2\sqrt{10}},$$

and it 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 follows from the assumed $h^{28} \leq b$. ⊣

6C.3. **Lemma** (Inclusion). *Suppose (a, b) is a difficult pair, and for any m, let $G_m(a, b) = G_m(\mathbf{N}_0[\div], a, b)$; 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 6A.1 and 6C.2 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:

$$\left(2^{2^{4m}}\right)^{28} = 2^{28 \cdot 2^{4m}} < 2^{2^5 \cdot 2^{4m}} = 2^{2^{4m+5}} \leq a. \qquad ⊣$$

6C.4. **Lemma.** *Suppose (a, b) is a difficult pair, $2^{2^{4m+6}} \leq a$, and set $\lambda = 1 + a!$. Then there is an embedding*

$$\pi : \mathbf{N}_0[\div] \restriction C(a, b; 2^{2^{4m}}) \rightarrowtail \mathbf{N}_0[\div]$$

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 6A.5, we will actually need to define the embedding on the larger substructure $\mathbf{N}_0[\div] \restriction C(a,b;h^{12})$, 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 6C.1, we need to check that

$$\left(h^{12}\right)^2 < \frac{\sqrt{b}}{2\sqrt{10}},$$

which is equivalent to

$$4 \cdot 10h^{48} < b; \tag{6C-8}$$

and this is true, because

$$4 \cdot 10h^{49} < 2^2 \cdot 2^4 h^{49} \le h^{55} = \left(2^{2^{4m}}\right)^{55} < 2^{2^6 \cdot 2^{4m}} = 2^{2^{4m+6}} < a,$$

by the hypothesis, and it yields (6C-8) when we divide both of its sides by h.

Using Lemma 6C.1 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 6A.5 that this is a 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 6C.2 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

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

is the correct division equation for X, Y, then

$$\rho X = \rho Y \cdot \rho Q + \rho R \quad (0 \le \rho R < \rho Y) \tag{6C-9}$$

is the correct division equation for $\rho X, \rho Y$. We distinguish two cases, following the proof of Lemma 6C.2.

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 (6C-3) 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 (6C-9).

Case 2, $y_1 a + y_2 b > 0$. Now $Q \leq h^9$, which implies $\rho Q = Q$. The explicit formula (6C-5) 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 (6C-9) by direct computation. $\dashv$

6C.5. **Theorem** (van den Dries and Moschovakis [2004]). *For all difficult pairs* (a, b),

(6C-10) $$\mathrm{depth}_{\perp\!\!\!\perp}(\mathbf{N}_0[\div], a, b) > \frac{1}{10}\log\log a.$$

Proof. Let $m = \mathrm{depth}_{\perp\!\!\!\perp}(\mathbf{N}_0[\div], a, b)$ for some difficult pair (a, b). If

$$2^{2^{4m+6}} \leq a,$$

then Lemma 6C.4 provides an embedding π which does not respect coprimeness at (a, b) since $\pi a = \lambda a$ and $\pi b = \lambda b$, with some λ. This contradicts the choice of m, and so

$$2^{2^{4m+6}} > a;$$

in other words

$$4m + 6 \geq \log\log a;$$

and since $m \geq 1$ by Problem x3A.2,

$$10m \geq 4m + 6 \geq \log\log a,$$

as required. $\dashv$

6C.6. **Corollary.** *For every difficult pair* (a, b),

$$\mathrm{depth}_{\mathrm{gcd}}(\mathbf{N}_0[\div], a, b) \geq \frac{1}{10}\log\log a.$$

Proof. For any $\mathbf{U}$, every embedding $\pi : \mathbf{U} \rightarrowtail \mathbf{N}_0[\div]$ which respects $\gcd(a, b)$ also respects $a \perp\!\!\!\perp b$, so

$$\mathrm{depth}_{\perp\!\!\!\perp}(\mathbf{N}_0[\div], a, b) \leq \mathrm{depth}_{\mathrm{gcd}}(\mathbf{N}_0[\div], a, b). \qquad \dashv$$

6C.7. **Corollary.** *Pratt's algorithm is optimal for coprimeness on the set of pairs* (F_{t+1}, F_t) *of successive Fibonacci numbers.*

PROOF is immediate from Problem x2C.8. ⊣

This corollary implies that Theorem 6C.5 is best possible (except, of course, for the specific constant 10), because the absolute lower bound it gives for all difficult pairs is matched by the Pratt algorithm on pairs of successive Fibonacci numbers. Note, however, that it does not rule out the possibility that the Main Conjecture in the Preface holds for all uniform processes of $\mathbf{N}_\varepsilon$, even if we formulate it for coprimeness rather than the gcd—because it might hold with another, more restrictive or different notion of "difficult pair"; in other words, we may have the wrong proof.

The most exciting possibility would be that the conjecture holds for all deterministic uniform processes but not for all uniform processes, which would exhibit the distinction between determinism and non-determinism in a novel context. This seems unlikely and, in any case, there is no obvious way how one would go about trying to prove it.

Problems for Section 6C

x6C.1*. **Problem** (van den Dries and Moschovakis [2004], [2009]). For every quadratic irrational $\xi > 1$, there is a rational number $r > 0$ such that for all but finitely many good approximations (a, b) of ξ,

$$\text{depth}(\mathbf{N}[\div], \perp\!\!\!\perp, a, b) \geq r \log\log a. \tag{6C-11}$$

HINT: Use Problem x6B.5 to adjust the argument for difficult pairs in this section.

The $O(\log\log)$ bound in this problem is best possible, because of Pratt's algorithm.

CHAPTER 7

LOWER BOUNDS FROM DIVISION AND MULTIPLICATION

The arithmetic becomes substantially more complex—and a little algebra needs to be brought in—when we add both division with remainder and multiplication to the primitives of $\mathbf{Lin}_0$. We will only derive here lower bounds for *unary functions and relations* from

$$\mathbf{Lin}_0[\div, \cdot] = \mathbf{Lin}_0 \cup \{\mathrm{iq}, \mathrm{rem}, \cdot\} = \{0, 1, =, <, +, \dot{-}, \mathrm{iq}, \mathrm{rem}, \cdot\},$$

leaving the general problem open.

7A. Polynomials and their heights

We review here briefly some basic results about the ring $K[T]$ of unary polynomials over a field K, 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 standard algebra classes.

To fix terminology, a *polynomial* in the *indeterminate* (variable) T over a ring K is a term

$$X = x_0 + x_1T + x_1T^2 + \cdots + x_nT^n,$$

where $x_i \in K$ and $x_n \neq 0$ together with the *zero polynomial* 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 a non-zero 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. We do not assign a degree to the zero polynomial.[17]

Two polynomials are equal if they are literally identical as terms, i.e., if the coefficients of like powers are equal.

[17]The usual convention is to set $\deg(0) = -\infty$, which saves some considerations of cases in stating results.

The *sum*, *difference* and *product* of two polynomials are defined by the performing the obvious operations on the coefficients and collecting terms:

$$\begin{aligned} X+Y &= \textstyle\sum_i (x_i+y_i)T^i, && \deg(X+Y) \le \max(\deg(X),\deg(Y)) \\ -X &= \textstyle\sum_i (-x_i)T^i, && \deg(-X) = \deg(X) \\ XY &= \textstyle\sum_k \Big(\sum_{i=0}^{i=k} x_i y_{k-i}\Big)T^k && \deg(XY) = \deg(X)+\deg(Y). \end{aligned}$$

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.

7A.1. **Theorem** (The Division Theorem for polynomials). *If K is a field, and $X, Y \in K[T]$ such that $\deg(X) \ge \deg(Y)$ and $Y \ne 0$, then there exist unique polynomials $Q, R \in K[T]$ such that*

$$X = YQ + R \quad \text{and } R = 0 \text{ or } \deg(R) < \deg(Y). \tag{7A-1}$$

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 $\deg(R_1) \ge \deg(Y)$ 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. $\dashv$

We call (7A-1) the *canonical division equation* (**c.d.e.**) for X, Y.

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 (7A-1), simply because 2 does not 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 a non-zero integer polynomial is the maximum of the absolute values of its coefficients,

$$\text{height}(x_0 + x_1T + \cdots + x_nT^n) = \max\{|x_i| \mid i = 0, \dots, n\} \quad (x_i \in \mathbb{Z}).$$

To extend the definition to $\mathbb{Q}[T]$, we let for each $n, h \in \mathbb{N}$,

$$\text{(7A-2)} \quad Q_n(T;h) = \Big\{\frac{x_0 + x_1T + x_2T^2 + \cdots + x_nT^n}{x^*} \mid \\ x_0, \dots, x_n, x^* \in \mathbb{Z}, x^* > 0 \text{ and } |x_0|, \dots, |x_n|, |x^*| \le h\Big\}.$$

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* (7A-2) 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 $\frac{1}{2}$; it is 6.

It is very easy to make "height estimates" for sums and products of polynomials:

7A.2. **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,

$$XY = \frac{\sum_{k=0}^{n+m}\Big(\sum_{i=0}^{i=k} x_i y_{k-i}\Big)T^k}{x^* y^*}.$$

For $k < n+m$, the typical coefficient in the numerator can be estimated by

$$\Big|\sum_{i=0}^{i=k} x_i y_{k-i}\Big| \leq \sum_{i=0}^{i=k} Hh \leq (k+1)Hh \leq (n+m)Hh;$$

and if $k = n+m$, then

$$\Big|\sum_{i=0}^{i=n+m} x_i y_{k-i}\Big| = |x_n y_m| \leq Hh < (n+m)Hh$$

since $x_i = 0$ when $i > n$ and $y_j = 0$ when $j > m$. ⊣

The next result is a version of the Division Theorem 7A.1 for $\mathbb{Q}[T]$ which supplies additional information about the heights.

7A.3. **Lemma** (Lemma 2.3 of Mansour, Schieber, and Tiwari [1991b]). *Suppose X and Y are polynomials with integer coefficients,*

$$\deg(X) = n \geq m = \deg(Y),\ X : H,\ Y : h,$$
$$\textit{and } X = YQ + R \quad \text{with } R = 0 \text{ or } \deg(R) < \deg(Y).$$

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

$$Z = y_m X - x_n Y T^d \tag{7A-3}$$

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. If $\deg(Z) \geq m$, then the Induction Hypothesis applies to 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

(7A-4) $$Z = Y\frac{Q_2}{y_m^d} + \frac{R_2}{y_m^d},$$

with

$$Q_2, R_2 : H(2h)(2h)^d = H(2h)^{d+1}.$$

Solving (7A-3) for X, we get

$$\begin{aligned} X &= \frac{1}{y_m}Z + \frac{x_n}{y_m}T^d Y \\ &= \frac{1}{y_m}\Big(Y\frac{Q_2}{y_m^d} + \frac{R_2}{y_m^d}\Big) + \frac{x_n}{y_m}T^d Y \\ &= \frac{Q_2 + x_n y_m^d T^d}{y_m^{d+1}}Y + \frac{R_2}{y_m^{d+1}} \end{aligned}$$

which is the **c.d.e.** for X, Y. We have already computed that $R_2 : H(2h)^{d+1}$. To verify that $Q_2 + x_n y_m^d T^d : H(2h)^{d+1}$, notice that $\deg(Q_2) < d$, since the opposite assumption implies with (7A-4) 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. $\dashv$

7A.4. **Theorem.** *If* $X, Y : h$, $\deg(X) \geq \deg(Y)$ *and* (7A-1) *holds, then*

$$Q, R : (2h)^{2n+5}.$$

Proof. Theorem 7A.3 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 $d \leq n$)

$$\frac{x^*Q}{y^*(x^*y_m)^{d+1}}, \frac{R}{y^*(x^*y_m)^{d+1}} : (2h)^{2n+5}. \qquad \dashv$$

7B. Unary relations from $\mathbf{Lin}_0[\div, \cdot]$

We establish here suitable versions of Lemma 6A.5 and Theorem 6A.6 for the structure

$$\mathbf{N}_0[\div, \cdot] = (\mathbf{N}_0, \mathbf{Lin}_0[\div, \cdot]) = (\mathbb{N}, 0, 1, =, <, +, \dot{-}, \mathrm{iq}, \mathrm{rem}, \cdot)$$

with a $\sqrt{\log\log}$ bound.

Set, for any $a, n, h \in \mathbb{N}$,

(7B-1) $$Q_n(a; h) = \Big\{\frac{x_0 + x_1a + x_2a^2 + \cdots + x_na^n}{x^*} \in \mathbb{N}$$
$$\mid x_0, \ldots, x_n, x^* \in \mathbb{Z}, x^* > 0 \text{ and } |x_0|, \ldots, |x_n|, |x^*| \leq h\Big\}.$$

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 $G_m(a) = G_m[(\mathbf{N}_0[\div, \cdot], a)$. To simplify dealing with them, we will be using the standard notations

(7B-2) $$x = f(a) = \frac{x_0 + x_1a + x_2a^2 + \cdots + x_na^n}{x^*},$$
$$y = g(a) = \frac{y_0 + y_1a + y_2a^2 + \cdots + y_ma^m}{y^*},$$

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

$$Q(a; h) = \bigcup_n Q_n(a; h), \quad x : h \iff x \in Q(a; h).$$

7B.1. **Lemma.** *With x and y as in* (7B-2)*, if $h \geq 2$ and $x, y \in Q_n(a; h)$, then $x + y, x \dot{-} y \in Q_n(a; h^3)$, and $xy \in Q_{2n}(a; nh^3)$.*

Proof. These are all immediate, using Lemma 7A.2. ⊣

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 (7B-2) when h is small relative to a.

7B.2. **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* (7B-2) *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

$$\begin{aligned}|x_0 + x_1 a + \cdots x_{n-1} a^{n-1}| &\leq (a-1)(1 + a + a^2 + \cdots + a^{n-1}) \\ &= (a-1)\frac{a^n - 1}{a-1} = a^n - 1 < a^n \leq |x_n| a^n,\end{aligned}$$

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). ⊣

7B.3. **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 computations. To begin with, assume

(H1) $\qquad c \geq 3, \text{ so that } 2h^2 \leq h^3 \leq h^{c(n+1)} < a,$

and Lemma 7B.2 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 $\mathbb{Q}[T]$,

(7B-3) $\quad f(T) = g(T)Q(T) + R(T)$
$$(R(T) = 0 \text{ or } \deg(R(T)) < \deg(g(T)).$$

We record for later use the heights from Lemma 7A.4,

(7B-4) $$Q(T), R(T) : (2h)^{2n+5} \le h^{4n+10}.$$

From (7B-3) we get

(7B-5) $$f(a) = g(a)Q(a) + R(a) \quad (R(a) = 0 \text{ or } R(a) < g(a)),$$

where the condition on $R(a)$ is trivial if $R(a) \le 0$, and follows from the corresponding condition about degrees in (7B-3) by Lemma 7B.2, provided that the height of $R(a)$ is sufficiently small, specifically

$$2\Big(h^{4n+10}\Big)^2 < a;$$

so here we assume

(H2) $$c \ge 21, \text{ so that } 2\Big(h^{4n+10}\Big)^2 \le hh^{8n+20} = h^{8n+21} < h^{21(n+1)} < a.$$

However, (7B-5) 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 (7B-3) and (7B-5) 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 $R(a) \ge 0$. In this case (7B-5) is the **c.d.e.** for $f(a)$ and $g(a)$, and from (7B-4),

$$Q(a), R(a) : h^{4n+10};$$

thus we assume at this point

(H3) $$d \ge 10, \text{ so that } h^{4n+10} \le h^{d(n+1)}.$$

Case 2, $Q(a) \in \mathbb{N}$ but $R(a) < 0$. From (7B-5) 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

$$(1) \quad 0 \le g(a) + R(a) \text{ and } (2) \quad g(a) + R(a) < g(a),$$

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 7B.2, we must know that

$$2\text{height}(g(a) + R(a))^2 < a,$$

and from Lemma 7B.1,

$$2\text{height}(g(a) + R(a))^2 \leq h\Big((h^{4n+10})^3\Big)^2 = h^{24n+61},$$

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 7B.1, 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 7B.1 again. Replacing these values in (7B-5), we get

$$\text{(7B-6)} \qquad f(a) = g(a)Q_2 + \underbrace{g(a)\frac{R_2}{z} + R(a)}$$

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 7B.2, 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 7B.1, the height of the sum is bounded by

$$\left(h^{4n+10}\right)^3 = h^{12n+30};$$

and to apply Lemma 7B.2, we must insure that

$$2(h^{12n+30})^2 = 2h^{24n+60} < a,$$

and so for this step we assume that

(H6) $\quad c \geq 61,$ 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 correct division equation 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

(H7) $\quad d \geq 33,$ 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 (7B-5) 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

$$c \geq 61, \quad d \geq 33. \qquad \dashv$$

7B.4. **Lemma** ($\mathbf{Lin}_0[\div, \cdot]$-embedding)**.** *Let $e = 61$ and for any m,*

$$G_m(a) = G_m(\mathbf{N}_0[\div, \cdot], a).$$

(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 embedding*

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

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

Proof of (1) is by induction on m, the basis being trivial. To apply Lemmas 7B.1 and 7B.3 at the induction step, we need the inequalities

$$2^m \Big(2^{2^{em^2}}\Big)^3 \leq 2^{2^{e(m+1)^2}}, \quad \Big(2^{2^{em^2}}\Big)^{61(2^m+1)} < a,$$
$$\Big(2^{2^{em^2}}\Big)^{33(2^m+1)} \leq 2^{2^{e(m+1)^2}},$$

and these are easily verified by direct computation.

(2) is proved very much like Lemma 6C.4, 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 6C.4. $\dashv$

7B.5. **Theorem.** (Mansour, Schieber, and Tiwari [1991b], van den Dries and Moschovakis [2009]). *If $R(x)$ is a good example, then for all $a \geq 2$*

$$R(a) \Longrightarrow \mathrm{depth}_R(\mathbf{N}_0[\div, \cdot], a) > \frac{1}{24}\sqrt{\log \log a}.$$

Proof. By the Homomorphism Test 4F.2 and the preceding Lemma, we know that if $m = \mathrm{depth}_R(\mathbf{N}_0[\div, \cdot], a)$, then

$$2^{2^{e(m+2)^2}} > a,$$

with $e = 61$. Taking logarithms twice, then the square root and finally using the fact that $3m \geq m + 2$ by Problem x3A.2, 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 $\sqrt{e} = \sqrt{61} < 8$. $\dashv$

CHAPTER 8

NON-UNIFORM COMPLEXITY IN $\mathbb{N}$

A computer has finite memory, and so it can only store and operate on a finite set of numbers. Because of this, complexity studies which aim to be closer to the applications are often restricted to the analysis of algorithms on structures with universe the set

$$[0, 2^N) = \{x \in \mathbb{N} : x < 2^N\}$$

of *N-bit numbers* for some fixed (large) N, typically restrictions to $[0, 2^N)$ of expansions of $\mathbf{N}_d$ or $\mathbf{N}_0$, e.g., $\mathbf{N}_0[\div], \mathbf{N}_0[\div, \cdot]$, etc. The aim now is to derive lower bounds for the *worst case behavior* of such algorithms as functions of N; and the field is sometimes called *non-uniform complexity theory*, since, in effect, we allow the use of a different algorithm for each N which solves a given problem in $\mathbf{A} \upharpoonright [0, 2^N)$.

For each structure $\mathbf{A} = (\mathbb{N}, \mathbf{\Phi})$ with universe $\mathbb{N}$, each relation $R \subseteq \mathbb{N}^n$ and each N, let

$$\text{(8-7)} \quad \text{depth}_R(\mathbf{A}, 2^N) = \max\Big\{\text{depth}_R(\mathbf{A} \upharpoonright [0, 2^N), \vec{x}) : x_1, \dots, x_n < 2^N\Big\}$$

and similarly for $\text{size}_R(\mathbf{A}, 2^N), \text{calls}_R(\mathbf{A}, 2^N)$. These are the *intrinsic* (worst case) *bit complexities* of R from the primitives of $\mathbf{A}$, at least those of them for which we can derive lower bounds. As it turns out, the results and the proofs are essentially the same for $\mathbf{N}_d$, except for the specific constants which are now functions of N (and somewhat smaller). For $\mathbf{N}_0[\div]$ and $\mathbf{N}_0[\div, \cdot]$, however, we need a finer analysis and we can only derive lower bounds for the larger complexity $\text{size}_R(\mathbf{A}, 2^N)$, primarily because there is "less room" in $[0, 2^N)$ for homomorphisms which exploit the uniformity assumption in the definition of intrinsic complexities. It is this new wrinkle in the proofs which is most interesting to us in this brief chapter.

8A. Non-uniform lower bounds from $\mathbf{Lin}_d$

We will show here that the intrinsic lower bounds from $\mathbf{Lin}_d$ of Chapter 5 hold also in the non-uniform case, with somewhat smaller constants.

This means (roughly) that for these problems, the lookup algorithm in Problem x8A.1 is weakly optimal for depth intrinsic bit complexity in $\mathbf{N}_d$.

8A.1. **Theorem** (van den Dries and Moschovakis [2009]). *If $N \geq 3$, then*

$$\text{depth}_{\text{Prime}}(\mathbf{N}_d, 2^N) > \frac{N}{5\log d}.$$

Proof. Suppose $p < 2^N$ is prime and let

$$m = \text{depth}_{\text{Prime}}(\mathbf{N}_d \restriction [0, 2^N), p), \quad \lambda = 1 + d^{m+1}.$$

If

$$(a) \quad d^{2m+2} < p \text{ and } (b) \quad \frac{x_0 + x_1\lambda p}{d^m} < 2^N \text{ for all } |x_0|, |x_1| \leq d^{2m},$$

then the proof of Lemma 5B.2 would produce an embedding

$$\pi : \mathbf{G}_m(\mathbf{N}_d \restriction [0, 2^n), p) \rightarrowtail \mathbf{N}_d \restriction [0, 2^N)$$

which does not respect the primality of p, yielding a contradiction. So for every prime $p < 2^N$, one of (a) or (b) must fail. To exploit this alternative, we need to apply it to primes not much smaller than 2^N, but small enough so that (b) holds, and to find them we appeal to Bertrand's Postulate, Theorem 418 of Hardy and Wright [1938]; this guarantees primes between l and $2l$ when $l \geq 3$. So choose p such that

$$2^{N-1} < p < 2^N,$$

which exists because $2^{N-1} > 3$ when $N \geq 3$.

Case 1, $d^{2m+2} \geq p$, and so $d^{2m+2} > 2^{N-1}$. Using as always the fact that $m \geq 1$, this gives easily $m > \frac{N}{5\log d}$.

Case 2, for some x_0, x_1 with $|x_0|, |x_1| \leq d^{2m}$, we have

$$\frac{x_0 + x_1(1 + d^{m+1})p}{d^m} \geq 2^N.$$

Compute (with $m \geq 1$):

$$\frac{x_0 + x_1(1 + d^{m+1})p}{d^m} < \frac{d^{2m} + d^{2m}(1 + d^{m+1})2^N}{d^m}$$
$$< d^m + d^m d^{m+2} 2^N \leq d^{2m+3}2^N \leq d^{5m}2^N$$

and so the case hypothesis implies that $m > \frac{N}{5\log d}$ again, as required. ⊣

Similar mild elucidations of the proofs we have given extend all the lower bound results about $\mathbf{N}_d$ in Chapter 5 to intrinsic bit complexity, and they are simple enough to leave for the problems.

Problems for Section 8A

x8A.1. **Problem** (The lookup algorithm). If $n \geq 1$, then every n-ary relation R on $\mathbb{N}$ can be decided for inputs $x_1, \dots, x_n < 2^N$ by an explicit $\mathbf{N}_b$-term $E^N(\vec{x})$ of depth $\leq N$. It follows that

$$\mathrm{depth}_R(\mathbf{N}_b, 2^N) \leq N.$$

Recall the definition of good examples in Subsection 5C.

x8A.2. **Problem** (van den Dries and Moschovakis [2009]). Suppose R is a good example such that for some k and all sufficiently large m, there exists some x such that

$$R(x) \ \&\ 2^m \leq x < 2^{km}.$$

Prove that for all $d \geq 2$ there is some $r > 0$ such that for all sufficiently large N,

$$\mathrm{depth}_R(\mathbf{N}_d, 2^N) > rN,$$

and verify that all the good examples in Problem x5C.4 satisfy the hypothesis.

x8A.3. **Problem** (van den Dries and Moschovakis [2009]). Prove that for some $r > 0$ and all sufficiently large N,

$$\mathrm{depth}_{\perp\!\!\!\perp}(\mathbf{N}_d, 2^N) > rN.$$

8B. Non-uniform lower bounds from $\mathbf{Lin}_0[\div]$

The methods of the preceding section do not extend immediately to $\mathbf{N}_0[\div]$, because the constant $\lambda = 1 + a!$ that we used in the proof of Lemma 6A.5 is too large: a direct adaptation of that proof to the non-uniform case leads to a $\log\log N$ lower bound for $\mathrm{depth}_{\mathrm{Prime}}(\mathbf{N}_0[\div], 2^N)$ which is way too low. In fact, it does not seem possible to get a decent lower bound for $\mathrm{depth}_{\mathrm{Prime}}(\mathbf{N}_0[\div], 2^N)$ with this method, but a small adjustment yields a lower bound for the size- and hence the calls- intrinsic bit complexities.

We start with the required modification of Lemma 6A.5.

8B.1. **Lemma.** *Suppose $\mathbf{U} \subseteq_p \mathbf{N}_0[\div]$ is generated by a, $m = \mathrm{depth}(\mathbf{U}, a)$, and $\nu = |U|$. If $2^{6^{m+3}} < a$, then there is a number $\lambda \leq 2^{\nu 6^{m+2}}$ and an embedding $\pi : \mathbf{U} \rightarrowtail \mathbf{N}_0[\div]$ such that $\pi(a) = \lambda a$.*

Proof. As in the proof of Lemma 6A.5, the assumed inequality on m implies that each $x \in U \subseteq G_m(\mathbf{N}_0[\div], a)$ can be expressed uniquely as a proper fraction of the form

$$x = \frac{x_0 + x_1 a}{x_2} \quad (|x_i| \le 2^{6^{m+1}}), \tag{8B-1}$$

and we can set

$$\text{denom}(x) = \text{the unique } x_2 \le 2^{6^{m+1}} \text{ such that (8B-1) holds} \quad (x \in U).$$

We claim that the conclusion of the Lemma holds with

$$\lambda = 1 + \textstyle\prod_{x \in U} \text{denom}(x) \le 1 + \left(2^{6^{m+1}}\right)^{\nu} < 2^{\nu 6^{m+2}}, \tag{8B-2}$$

and to prove this we follow very closely the proof of Lemma 6A.5: 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 U which takes values in $\mathbb{N}$, because

$$x \in U \Longrightarrow \text{denom}(x) \mid (\lambda - 1);$$

and finally verify that it is an embedding from $\mathbf{U}$ to $\mathbf{N}_0[\div]$ exactly as in the proof of Lemma 6A.5. ⊣

8B.2. **Theorem** (van den Dries and Moschovakis [2009]). *If $N \ge 8$, then*

$$\text{size}_{\text{Prime}}(\mathbf{N}_0[\div], 2^N) > \frac{1}{10} \log N. \tag{8B-3}$$

Proof. Let $k = \lfloor \frac{N}{2} \rfloor - 1$ so that

$$k + 1 \le \frac{N}{2} < k + 2 \text{ and so } k > \frac{N}{2} - 2.$$

The hypothesis on N yields $2^k > 4$, 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. Let $\mathbf{A} = \mathbf{N}_0[\div] \restriction [0, 2^N)$ (to simplify notation) and choose $\mathbf{U} \subseteq_p \mathbf{A}$ so that

$$\mathbf{U} \Vdash_c^{\mathbf{A}} \text{Prime}(p), \quad \nu = |U| = \text{size}_{\text{Prime}}(\mathbf{A}, p)$$

and set $m = \text{depth}(\mathbf{U}, p)$. (It could be that $m > \text{depth}_{\text{Prime}}(\mathbf{A}, p)$.) Let also

$$\lambda = 1 + \textstyle\prod_{x \in U} \text{denom}(x) < 2^{\nu 6^{m+2}}$$

as above. If

$$2^{6^{m+3}} < p \text{ and } \frac{x_0 + x_1 \lambda p}{x_2} < 2^N \text{ whenever } |x_i| \le 2^{6^{m+1}},$$

then the proof of Lemma 8B.1 produces an embedding $\pi : \mathbf{U} \rightarrowtail \mathbf{A}$ which does not respect the primality of p contrary to the choice of $\mathbf{U}$; so one of the two following cases must hold.

Case 1: $2^{6^{m+3}} \geq p > 2^k$. This gives $6^{m+3} > k > \frac{N}{2} - 2$, and this easily implies (with $m \geq 1$) that $\nu \geq m > \frac{\log N}{10}$ in this case, cf. Problem x8B.1.

Case 2: For some x_0, x_1, x_2 with $|x_i| \leq 2^{6^{m+1}}$,

$$\frac{x_0 + x_1 \lambda p}{x_2} \geq 2^N.$$

Compute:

$$\begin{aligned}\frac{x_0 + x_1 \lambda p}{x_2} &\leq 2^{6^{m+1}} + 2^{6^{m+1}} \cdot 2^{\nu 6^{m+2}} \cdot 2^{\frac{N}{2}} \leq 2 \cdot 2^{6^{m+1} + \nu 6^{m+2}} \cdot 2^{\frac{N}{2}} \\ &\leq 2^{2\nu 6^{m+2}+1} \cdot 2^{\frac{N}{2}} \leq 2^{3\nu 6^{m+2}} \cdot 2^{\frac{N}{2}}.\end{aligned}$$

So the case hypothesis gives $2^{3\nu 6^{m+2}} \cdot 2^{\frac{N}{2}} \geq 2^N$ which gives $3\nu 6^{m+2} \geq \frac{N}{2}$ and then

$$\nu 6^{m+3} \geq N.$$

This is the basic fact about the intrinsic bit complexity of primality, and it can be used to derive a lower bound for the measure induced by the substructure norm

$$\mu(\mathbf{U}, a) = |U| 6^{\text{depth}(\mathbf{U},a)+3}.$$

To derive a lower bound for $\nu = \text{size}_{\text{Prime}}(\mathbf{N}[\div], 2^N)$, we note as usual that $m \geq 1$, and so $m \leq \nu < 6^\nu$, $6^{2\nu+3} > N$ and so

$$\nu > \frac{1}{5 \log 6} \log N > \frac{1}{10} \log N. \qquad \dashv$$

It should be clear that the numerology in this proof was given in detail mostly for its amusement value, since from the first couple of lines in each of the cases one sees easily that $\nu > r \log N$ for some r. Moreover, one can certainly bring that $\frac{1}{10}$ up quite a bit, with cleverer numerology, a more judicious choice of p, or by weakening the result to show that (8B-3) holds only for very large N. The problems ask only for these more natural (if slightly weaker) results and leave it up to the solver whether they should indulge in manipulating numerical inequalities.

Problems for Section 8B

x8B.1. **Problem.** Prove that if $m \geq 1$, $N \geq 1$ and $6^{m+3} > \frac{N}{2} - 2$, then $m > \frac{\log N}{10}$. Hint: Check that $6^{6m} \geq 6^{5m+1} > 2 \cdot 6^{m+3} + 4 > N$.

x8B.2. **Problem.** Suppose R is a good example with the property that for some k and every $m \geq 1$, there is some x such that $R(x)$ and

$$2^{6^m} < x < 2^{6^{km}}.$$

Prove that for some $r > 0$ and sufficiently large N,

$$\mathrm{size}_R(\mathbf{N}[\div], 2^N) > r \log N.$$

Verify also that the good examples in Problem x5C.4 satisfy the hypothesis.

x8B.3. **Problem** (van den Dries and Moschovakis [2009])**.** Prove that for some $r > 0$ and all sufficiently large N,

$$\mathrm{size}_{\perp\!\!\!\perp}(\mathbf{N}[\div], 2^N) > r \log N.$$

Hint: Use the largest good approximation (a, b) of $\sqrt{2}$ with $a < 2^{\frac{N}{2}}$.

x8B.4. **Problem.** Derive a lower bound for $\mathrm{size}_R(\mathbf{Lin}_0[\div, \cdot], 2^N)$ when R is a good example or $\perp\!\!\!\perp$.

PART IV, SOME LOWER BOUNDS IN ALGEBRA

CHAPTER 9

POLYNOMIAL NULLITY (0-TESTING)

In this section we will prove four results on the intrinsic calls-complexities of evaluation and 0-testing of polynomials, all of them establishing the optimality or near-optimality of Horner's rule from various primitives for "generic" inputs. Polynomial evaluation is perhaps the simplest problem in algebraic complexity, and it has been much studied since its formulation as a complexity problem by Ostrowski [1954]; here we are concerned with 0-testing, a plausibly easier problem which has received considerably less attention.

For any field F, Horner's rule computes the value

$$V_F(a_0, \dots, a_n, b) = \textstyle\sum_{i \leq n} a_i b^i = a_0 + a_1 b + \cdots + a_n b^n \quad (n \geq 1)$$

of a polynomial of degree n using no more than n multiplications and n additions in F, by applying successively for $i = 1, \dots n$ the identity

$$a_0 + a_1 b + \cdots + a_i b^i = a_0 + (a_1 + a_2 b + \cdots + a_i b^{i-1})b.$$

For certain values of the coefficients, using division and subtraction might lead to a more efficient computation because of identities like

$$1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1},$$

and we also need the equality relation when we want to decide the *nullity* (or 0-*testing*) relation

$$N_F(a_0, \dots, a_n, b) \iff V_F(a_0, \dots, a_n, b) = 0 \quad (n \geq 1)$$

on F; and so it is natural to view Horner's rule as an algorithm from the primitives of the expansion

(9-4) $$\mathbf{F} = (F, 0, 1, +, -, \cdot, \div, =)$$

of the field structure of F by $=$.

9A. Preliminaries and notation

A *partial ring homomorphism*

$$\pi : F_1 \rightharpoonup F_2$$

on one field to another is a partial function whose domain of convergence Domain(π) is a subring R_1 (with 1) of F_1 and which respects as usual the ring operations, including $\sigma(1) = 1$. Notice that for every $\mathbf{U} \subseteq_p \mathbf{F}_1$,

(9A-1) *if* π *is total on* U *and* $0 \neq x \in U \Longrightarrow \pi(x) \neq 0$,
then $\pi \restriction U : \mathbf{U} \to \mathbf{F}_2$ *is a homomorphism*

i.e., $\pi \restriction U$ preserves not only the ring operations, but also all divisions in eqdiag($\mathbf{U}$). This is because if $(\div, u, v, w) \in$ eqdiag($\mathbf{U}$), then $\pi(u), \pi(v), \pi(w)$ are all defined and $v \neq 0$, so $\pi(v) \neq 0$ by the hypothesis of (9A-1); and since $vw = u$, we have $\pi(v)\pi(w) = \pi(u)$, which then gives $\pi(w) = \frac{\pi(u)}{\pi(v)}$.

For any field F and indeterminates $\vec{u} = u_1, \dots, u_k$, $F[\vec{u}]$ is the ring of all polynomials with coefficients in F and $F(\vec{u})$ is the field of all rational functions in $\vec{u}$ with coefficients in F. If $F(v, \vec{u}) \subseteq K$ for some field K and $\alpha \in K$, then the substitution $v \mapsto \alpha$ induces a ring homomorphism

$$\text{(9A-2)} \qquad \pi\Big(\frac{\chi_n(v, \vec{u})}{\chi_d(v, \vec{u})}\Big) = \frac{\chi_n(\alpha, \vec{u})}{\chi_d(\alpha, \vec{u})} \quad \Big(\frac{\chi_n(v, \vec{u})}{\chi_d(v, \vec{u})} \in F(v, \vec{u})\Big)$$

with

$$\text{Domain}(\pi) = \Big\{\frac{\chi_n(v, \vec{u})}{\chi_d(v, \vec{u})} : \chi_d(\alpha, \vec{u}) \neq 0\Big\} \subseteq F(v, \vec{u}).$$

Notice that $F[v, \vec{u}] \subset$ Domain(π) and $\pi(u_i) = u_i$. We will sometimes call π "the substitution" $(v \mapsto \alpha)$, and the only proper maps we will need are compositions of such substitutions.

9A.1. **The Substitution Lemma.** *Suppose F, K are fields, $v, \vec{u}$ are indeterminates, U is a finite subset of $F(v, \vec{u})$, $K \supseteq F(v, \vec{u})$ and $\{\alpha_t\}_{t \in I}$ is an infinite set of distinct elements of K. It follows that for all but finitely many $t \in I$, the homomorphism induced by the substitution*

$$v \mapsto \rho_t(v) = \alpha_t$$

is total and hence injective on U.

PROOF. It is convenient to prove first a

Sublemma. If $U' \subset F[v, \vec{u}]$ is any finite set of polynomials, then for all but finitely many $t \in I$,

$$\Big(\chi(v, \vec{u}) \in U' \text{ and } \chi(v, \vec{u}) \neq 0\Big) \Longrightarrow \chi(\rho_t(v), \vec{u}) \neq 0.$$

Proof of the Sublemma. Write each $\chi(v, \vec{u}) \in U'$ as a polynomial in v,

$$\chi(v, \vec{u}) = \chi_0(\vec{u}) + \chi_1(\vec{u})v + \cdots + \chi_l(\vec{u})v^l,$$

so that in some algebraically closed extension of K,

$$\chi(v, \vec{u}) = \chi_l(\vec{u})(v - a_1) \cdots (v - a_l).$$

Now, for each $t \in I$, easily,

$$\rho_t(\chi(v, \vec{u})) = \chi_l(\vec{u})(\alpha_t - a_1) \cdots (\alpha_t - a_l),$$

and this can only be 0 if $\alpha_t = a_i$ for some i. The conclusion then holds for any t such that α_t is not a root a_i of a non-0 polynomial in U'.⊣ (Sublemma)

We now fix a specific representation of the form

$$\text{(9A-3)} \qquad \chi(v, \vec{u}) = \frac{\chi_n(v, \vec{u})}{\chi_d(v, \vec{u})} \quad (\chi_d(v, \vec{u}) \neq 0)$$

for each $\chi(v, \vec{u}) \in U$, and we apply this Sublemma to the finite set U' comprising all polynomials in one of the forms

$$(i)\ \chi_d(v, \vec{u}), \qquad (ii)\ \chi_n(v, \vec{u})\chi'_d(v, \vec{u}) - \chi'_n(v, \vec{u})\chi_d(v, \vec{u})$$

with $\chi(v, \vec{u}), \chi'(v, \vec{u}) \in U$. ⊣

Note. If $\vec{a} = a_1, \ldots, a_k$ are elements of some field $K \supset F$ which are algebraically independent over F, then the extensions $F[\vec{a}], F(\vec{a}) \subseteq \overline{F}$ are naturally isomorphic with $F[\vec{u}]$ and $F(\vec{u})$ respectively by the *relabelling* isomorphism determined by $a_i \mapsto u_i$, for $i = 1, \ldots, k$. We will often establish some facts about one of $F(\vec{a})$ or $F(\vec{u})$ and then quote them for the other, sometimes without explicit mention. We will also use the same terminology for these isomorphic structures: if, for example, a, b are complex numbers which are algebraically independent (over $\mathbb{Q}$), then the members of $\mathbb{Q}(a, b)$ are "the rational functions of a, b".

9B. Horner's rule is $\{\cdot, \div\}$-optimal for nullity

The $\{\cdot, \div\}$-optimality of Horner's rule for polynomial evaluation was proved by Pan [1966]. Pan established his result for algorithms expressed by *computation sequences* and introduced the *method of substitution*, i.e., the use of partial homomorphisms induced by substitutions as above.

By Horner's rule, for all fields F and all $\vec{a} \in F^n$,

$$\text{calls}_{\{\cdot, \div\}}(\mathbf{F}, N_F, \vec{a}) \leq n.$$

9B.1. **Theorem** (Bürgisser and Lickteig [1992]). *If F is a field of characteristic 0, $n \geq 1$, and $a_0, \dots, a_n, b \in F$ are algebraically independent* (over $\mathbb{Q}$)*, then*

$$\text{(9B-1)} \qquad \text{calls}_{\{\cdot,\div\}}(\mathbf{F}, N_F, a_0, \dots, a_n, b) = n.$$

In particular, (9B-1) *holds for the reals $\mathbb{R}$ and the complexes $\mathbb{C}$ with algebraically independent $a_0, a_1, \dots, a_n, b$.*[18]

If $\vec{a}, b$ are algebraically independent, then

$$a_0 + a_1 b + \cdots + a_n b^n \neq 0;$$

so to prove the theorem by the Homomorphism Test 4F.2, we must construct for every algebraically independent tuple $\vec{a}, b \in F^{n+2}$ and every finite substructure $\mathbf{U} \subseteq_p \mathbf{F}$ generated by $\vec{a}, b$ such that $\text{calls}_{\{\cdot,\div\}}(\mathbf{U}, \vec{a}, b) < n$, a homomorphism $\sigma : \mathbf{U} \to \mathbf{F}$ satisfying

$$\text{(9B-2)} \qquad \sigma(a_0) + \sigma(a_1)\sigma(b) + \cdots + \sigma(a_n)\sigma(b)^n = 0.$$

Moreover, since eqdiag($\mathbf{U}$) may contain all true inequations $u \neq v$ for $u, v \in U$, we must make sure that π is an embedding. The construction is by induction on n, but we need a very strong "induction loading device" for it to go through. The appropriate lemma is an elaboration of the construction in Winograd [1967], [1970], which extends and generalizes Pan's results.

For a field F and indeterminates $z, \vec{x} = x_1, \dots, x_n, y$, we write as above

$$\mathbf{F}(z, \vec{x}, y) = (F(z, \vec{x}, y), 0, 1, +, -, \cdot, \div, =).$$

We will denote entries in the equational diagram eqdiag($\mathbf{F}(z, \vec{x}, y)$) using infix notation, $a + b = c$ for $(+, a, b, c)$, $a \cdot b = c$ for $(\cdot, a, b, c)$, etc. We write $\{\cdot, \div\}$ for *multiplications and divisions*, and we define the *trivial* $\{\cdot, \div\}$ by the following rules:

$$a \cdot b = c \text{ is trivial if } a \in F \text{ or } b \in F \text{ or } a, b \in F(y);$$
$$a \div b = c \text{ is trivial if } b \in F \text{ or } a, b \in F(y).$$

All additions, subtractions and inequations in $\mathbf{F}(z, \vec{x}, y)$ are also considered trivial.

9B.2. **Lemma.** *Suppose F is an infinite field, $n \geq 1$, z, $\vec{x} = x_1, \dots, x_n$, y are distinct indeterminates, $\mathbf{U} \subseteq_p \mathbf{F}(z, \vec{x}, y)$ is finite, and $\psi_1, \dots, \psi_n \in F(y)$ so that the following conditions hold:*

(1) $\mathbf{U}$ *is generated by $(F \cap U) \cup \{z, \vec{x}, y\}$.*
(2) *For any $f_1, \dots, f_n \in F$, if $f_1\psi_1 + \cdots + f_n\psi_n \in F$, then $f_1 = \cdots = f_n = 0$.*
(3) *There are no more than $n - 1$ non-trivial $\{\cdot, \div\}$ in* eqdiag($\mathbf{U}$).

[18]The proof in Bürgisser and Lickteig [1992] is for algebraic decision trees, i.e., (branching) computation sequences with equality tests.

Then there is a partial ring homomorphism

$$\pi : F(z, \vec{x}, y) \rightharpoonup F(\vec{x}, y)$$

which is total and injective on U, it is the identity on $F(y)$ and it satisfies

$$\pi(z) = \pi(x_1)\psi_1 + \cdots + \pi(x_n)\psi_n. \tag{9B-3}$$

It follows that $\pi \restriction U : \mathbf{U} \rightarrowtail \mathbf{F}(\vec{x}, y)$ is an embedding which satisfies (9B-3).

Proof of Theorem 9B.1 from Lemma 9B.2. The hypothesis implies that

$$a_0 + a_1 b + \cdots + a_n b^n \neq 0,$$

and so by the Homomorphism Test 4F.2 it is enough to show that for every finite $\mathbf{U} \subseteq_p \mathbf{F}$ which is generated by $a_0, \vec{a} = a_1, \dots, a_n, b$, if $|\text{eqdiag}(\mathbf{U} \restriction \{\cdot, \div\})| < n$, then there is an embedding $\pi : \mathbf{U} \rightarrowtail \mathbf{F}$ such that

$$\pi(a_0) + \pi(a_1)\pi(b) + \cdots + \pi(a_n)\pi(b)^n = 0.$$

We may assume that $0 - a_0 = -a_0, 0 - (-a_0) = a_0 \in \text{eqdiag}(\mathbf{U})$, by adding them if necessary, so that $\mathbf{U}$ is also generated by $-a_0, \vec{a}, b$. Moreover, $\mathbf{U} \subseteq_p \mathbb{Q}(-a_0, \vec{a}, b)$ and $\mathbb{Q}(-a_0, \vec{a}, b)$ is isomorphic with $\mathbb{Q}(z, \vec{x}, y)$ by the isomorphism ρ generated by the relabelling $-a_0 \mapsto z, a_i \mapsto x_i, b \mapsto y$. The required π is now constructed by applying Lemma 9B.2 to $\mathbf{U}' = \rho[\mathbf{U}]$ and $\psi_1 = y, \psi_2 = y^2, \dots, \psi_n = y^n$, carrying the embedding it gives back to an embedding $\pi : \mathbf{U} \rightarrowtail \mathbf{F}(\vec{a}, b)$ and noticing that $\pi(b) = b$. ⊣

Proof of Lemma 9B.2 is by induction on n, but it is useful to consider first a case which covers the basis and also arises in the induction step.

Preliminary case: there are no non-trivial $\{\cdot, \div\}$ in $\mathbf{U}$. It follows that every $X \in U$ is uniquely of the form

$$X = f_0 z + \textstyle\sum_{1 \le i \le n} f_i x_i + \phi(y) \tag{9B-4}$$

with $f_i \in F, \phi(y) \in F(y)$. If π is the partial ring homomorphism induced by the substitution

$$z \mapsto \textstyle\sum_{1 \le i \le n} x_i \psi_i,$$

then π is the identity on $F(\vec{x}, y)$ and it is total on U, because the only divisions in eqdiag($\mathbf{U}$) are with both arguments in $F(y)$ or the denominator in F. So it is enough to check that it is injective on the set of all elements of the form (9B-4) and that it satisfies (9B-3). To check injectivity, suppose that

$$\begin{aligned}\pi(X) = f_0\Big(\textstyle\sum_{1 \le i \le n} x_i \psi_i\Big) + \sum_{1 \le i \le n} f_i x_i + \phi(y) \\ = f_0'\Big(\textstyle\sum_{1 \le i \le n} x_i \psi_i\Big) + \sum_{1 \le i \le n} f_i' x_i + \phi'(y) = \pi(X')\end{aligned}$$

so that

$$\begin{aligned}(f_0 - f_0') \textstyle\sum_{1\le i\le n} x_i\psi_i + \sum_{1\le i\le n}(f_i - f_i')x_i + (\phi(y) - \phi'(y))\\ = \textstyle\sum_{1\le i\le n} \Big((f_0 - f_0')\psi_i + (f_i - f_i')\Big)x_i + (\phi(y) - \phi'(y)) = 0.\end{aligned}$$

This yields $\phi(y) = \phi'(y)$ and for each i, $(f_0 - f_0')\psi_i + (f_i - f_i') = 0$; and since no ψ_i is in F by (2) of the hypothesis, this implies that $f_0 = f_0'$, and finally that $f_i - f_i'$ for each i.

The identity (9B-3) is trivial because $\pi(z) = x_1\psi_1 + \cdots + x_n\psi_n$ and $\pi(x_i) = x_i$.

Basis, $n = 1$. This is covered by the preliminary case.

Induction Step, $n > 1$. If the preliminary case does not apply, then there is at least one non-trivial $\{\cdot, \div\}$ in eqdiag($\mathbf{U}$); so there is a least $m > 0$ such that some $\chi \in G_m(\mathbf{U}, z, \vec{x}, y)$ is a non-trivial product or quotient of elements of $\mathbf{G}_{m-1}(\mathbf{U}, z, \vec{x}, y)$ in which all $\{\cdot, \div\}$ are trivial; and so there is at least one non-trivial $\{\cdot, \div\}$ in eqdiag($\mathbf{U}$) of the form

(9B-5) $$(f_0'z + \textstyle\sum_{1\le i\le n} f_i'x_i + \phi'(y)) \circ (f_0z + \sum_{1\le i\le n} f_ix_i + \phi(y)) = \chi$$

where $\circ$ is $\cdot$ or $\div$. We consider cases of how this can arise.

Case 1: There is some $i \ge 1$ such that $f_i \ne 0$, and the first factor in (9B-5) *is not in F.* We assume without loss of generality that $f_1 \ne 0$, and then dividing the equation by f_1 we put the second factor in the form

(9B-6) $$f_0z + x_1 + \textstyle\sum_{2\le i\le n} f_ix_i + \phi(y).$$

By the Substitution Lemma 9A.1, there is some $\overline{f} \in F$ such that the substitution

$$\rho_1(x_1) = \overline{f} - f_0z - \textstyle\sum_{2\le i\le n} f_ix_i - \phi(y)$$

induces an isomorphism

$$\rho_1 \restriction U : \mathbf{U} \rightarrowtail\!\!\!\!\twoheadrightarrow \rho_1[\mathbf{U}] = \mathbf{U}_1 \subseteq_p \mathbf{F}(z, x_2, \dots, x_n, y).$$

Notice that ρ_1 does not introduce any new non-trivial multiplication or division (because it is the identity on $F(y)$), and it turns the chosen operation in $\mathbf{U}$ into a trivial one since

$$\rho_1(f_0z + x_1 + \textstyle\sum_{2\le i\le n} f_ix_i + \phi(y)) = \overline{f}.$$

So there are fewer than $n-1$ $\{\cdot, \div\}$ in eqdiag($\mathbf{U}_1$), and $\mathbf{U}_1$ is generated by $z, x_2, \dots, x_n, y$ and $\{\overline{f}\} \cup (F \cap U)$.

Applying Lemma 9A.1 again, fix some $\overline{g} \in F$ such that the substitution

$$\rho_2(z) = \frac{1}{1 + f_0\psi_1}\Big((\overline{f} - \phi(y))\psi_1 + \overline{g}z\Big)$$

induces an isomorphism

$$\rho_2 \restriction U_1 : \mathbf{U}_1 \rightarrowtail\!\!\!\!\rightarrow \rho_2[\mathbf{U}_1] = \mathbf{U}_2 \subseteq_p \mathbf{F}(z, x_2, \dots, x_n, y).$$

This too does not introduce any non-trivial multiplications or divisions, and $\mathbf{U}_2$ is generated by $z, x_2, \dots, x_n, y$ and $F \cap U_2$. The required partial ring homomorphism is the composition

$$\pi = \sigma \circ \rho_2 \circ \rho_1 : F(z, \vec{x}, y) \rightharpoonup F(\vec{x}, y)$$

of the three substitutions, where σ is guaranteed by the induction hypothesis so that $\sigma \restriction U_2 : \mathbf{U}_2 \rightarrowtail \mathbf{F}(x_2, \dots, x_n, y)$ and

$$\overline{g}\sigma(z) = \textstyle\sum_{2 \le i \le n} (\psi_i - f_i\psi_1)\sigma(x_i).$$

This exists because the functions

$$\frac{1}{\overline{g}}(\psi_i - f_i\psi_1) \quad (i = 2, \dots, n)$$

satisfy (2) in the theorem.

To see that π has the required property, notice first that

$$\pi(z) = \sigma(\rho_2(z))$$

because $\rho_1(z) = z$. Using the corresponding properties of ρ_2 and σ, we get:

$$\begin{aligned}
&\pi(x_1)\psi_1 + \textstyle\sum_{2\le i\le n} \pi(x_i)\psi_i \\
&\quad = \sigma(\rho_2(\overline{f} - \phi(y) - \textstyle\sum_{2\le i\le n} f_i x_i - f_0 z))\psi_1 + \textstyle\sum_{2\le i\le n} \sigma(x_i)\psi_i \\
&\quad = \sigma\Big(\overline{f} - \phi(y) - \textstyle\sum_{2\le i\le n} f_i x_i - f_0\rho_2(z)\Big)\psi_1 + \textstyle\sum_{2\le i\le n} \sigma(x_i)\psi_i \\
&\quad = (\overline{f} - \phi(y))\psi_1 - f_0\psi_1\sigma(\rho_2(z)) + \textstyle\sum_{2\le i\le n} (\psi_i - f_i\psi_1)\sigma(x_i) \\
&\qquad\qquad = (\overline{f} - \phi(y))\psi_1 - f_0\psi_1\sigma(\rho_2(z)) + \overline{g}\sigma(z).
\end{aligned}$$

So what we need to check is the equation

$$\begin{aligned}
\sigma(\rho_2(z)) &= (\overline{f} - \phi(y))\psi_1 - f_0\psi_1\sigma(\rho_2(z)) + \overline{g}\sigma(z) \\
\text{equivalently } (1 + f_0\psi_1)\sigma(\rho_2(z)) &= (\overline{f} - \phi(y))\psi_1 + \overline{g}\sigma(z) \\
\text{equivalently } (1 + f_0\psi_1)\rho_2(z) &= (\overline{f} - \phi(y))\psi_1 + \overline{g}z,
\end{aligned}$$

and the last is immediate from the definition of $\rho_2(z)$. (Note that we use repeatedly the fact that σ is injective on U_2 and the identity on $F(y)$.)

Case 2: $f_1 = \cdots = f_n = 0$, $f_0 \neq 0$, and the first factor in (9B-5) *is not in F*. We may assume without loss of generality that $f_0 = 1$, and so the second factor has the form

$$z + \phi(y).$$

By Lemma 9A.1, choose some $\overline{f} \in F$ such that the substitution

$$\rho_1(z) := \overline{f} - \phi(y)$$

induces an isomorphism

$$\rho_1 : \mathbf{U} \rightarrowtail\!\!\!\!\twoheadrightarrow \rho_1[\mathbf{U}] = \mathbf{U}_1 \subseteq_p \mathbf{F}(\vec{x}, y).$$

There is one fewer non-trivial operation in eqdiag$(\mathbf{U}_1)$, since ρ_1 does not introduce any new ones and $\rho_1(z + \phi(y)) = \overline{f}$. Next, choose $\overline{g} \in F$ by Lemma 9A.1 again so that the substitution

$$\rho_2(x_1) := \frac{1}{\psi_1}\Big(\overline{f} - \phi(y) - \overline{g}z\Big)$$

induces an isomorphism

$$\rho_2 : \mathbf{U}_1 \rightarrowtail\!\!\!\!\twoheadrightarrow \rho_2[\mathbf{U}_1] = \mathbf{U}_2 \subseteq_p \mathbf{F}(z, x_2, \dots, x_n, y).$$

There are fewer than $n - 1$ non-trivial $\{\cdot, \div\}$ in $\mathbf{U}_2$, and so the induction hypothesis gives us an embedding

$$\sigma : \mathbf{U}_2 \rightarrowtail \mathbf{F}(z, x_2, \dots, x_n, y)$$

such that

$$\overline{g}\sigma(z) = \textstyle\sum_{2 \le i \le n} \sigma(x_i)\psi_i.$$

The required partial ring homomorphism is the composition $\pi = \sigma \circ \rho_2 \circ \rho_1$, whose restriction to U is certainly total and injective. To check that it satisfies (9B-3), notice first that

$$\pi(z) = \sigma(\rho_2(\rho_1(z))) = \sigma(\rho_2(\overline{f} - \phi(y))) = \overline{f} - \phi(y).$$

On the other hand,

$$\begin{aligned}\pi(x_1)\psi_1 + \textstyle\sum_{2\le i\le n} \pi(x_i)\psi_i &= \sigma(\rho_2(x_1))\psi_1 + \textstyle\sum_{2\le i\le n} \sigma(x_i)\psi_i \\ &= \sigma\Big(\frac{1}{\psi_1}\Big(\overline{f} - \phi(y) - \overline{g}z\Big)\psi_1\Big) + \textstyle\sum_{2\le i\le n} \sigma(x_i)\psi_i \\ &\qquad = \overline{f} - \phi(y) - \overline{g}\sigma(z) + \overline{g}\sigma(z) = \pi(z).\end{aligned}$$

Cases 3 and 4: Cases 1 and 2 do not apply, some $f_i' \neq 0$, and the second factor in (9B-5) *is not in F*—which means that it is in $F(y) \setminus F$. These are handled exactly like Cases 1 and 2.

This completes the proof, because if none of these cases apply, then both factors of (9B-5) are in $F(y)$, and so the operation is trivial. $\dashv$

9C. Counting identity tests along with $\{\cdot, \div\}$

We outline here a proof of the following theorem, which is also implicit in Bürgisser and Lickteig [1992] for algebraic decision trees.

9C.1. **Theorem.** *If F is a field of characteristic 0, $n \geq 1$, and $a_0, \dots, a_n, b \in F$ are algebraically independent* (over the prime field $\mathbb{Q}$), *then*

$$\text{calls}_{\{\cdot,\div,=\}}(\mathbf{F}, N_F, a_0, \dots, a_n, b) = n + 1. \tag{9C-1}$$

In particular, (9C-1) *holds for the reals $\mathbb{R}$ and the complexes $\mathbb{C}$ with algebraically independent $a_0, a_1, \dots, a_n, b$.*

We define *trivial* $=$-tests exactly as for the multiplication and division operations: i.e., an entry $(=, a, b, w) \in \text{eqdiag}(\mathbf{U})$ with $\mathbf{U} \subseteq_p \mathbf{F}(z, \vec{x}, y)$ is trivial if $a \in F$, or $b \in F$, or $a, b \in F(y)$. Notice that we only need count inequation entries of the form $(=, a, b, \text{ff})$ since homomorphisms preserve equations.

The theorem will follow from the following lemma as in the preceding section.

9C.2. **Lemma.** *Suppose F is an infinite field, $n \geq 1$, z, $\vec{x} = x_1, \dots, x_n$, y are distinct indeterminates,*

$$\mathbf{U} \subseteq_p \mathbf{F}(z, \vec{x}, y) = (F(z, x_1, \dots, x_n, y), 0, 1, +, -, \cdot, \div, =)$$

is finite, and $\psi_1, \dots, \psi_n \in F(y)$ so that the following conditions hold:

(1) *$\mathbf{U}$ is generated by $(F \cap U) \cup \{z, \vec{x}, y\}$.*

(2) *For any $f_1, \dots, f_n \in F$, if $f_1\psi_1 + \cdots + f_n\psi_n \in F$, then $f_1 = \cdots = f_n = 0$.*

(3) *There are no more than n non-trivial $\{\cdot, \div, =\}$ entries in* eqdiag($\mathbf{U}$).

Then there is a homomorphism $\pi : \mathbf{U} \to \mathbf{F}(\vec{x}, y)$ which is the identity on $F(y) \cap U$ and satisfies

$$\pi(z) = \pi(x_1)\psi_1 + \cdots + \pi(x_n)\psi_n. \tag{9C-2}$$

Outline of the proof. If $\mathbf{U}$ has at least one non-trivial $=$ - test, then it has no more than $n - 1$ non-trivial $\{\cdot, \div\}$ and the lemma follows from Lemma 9B.2, since the π produced by it is injective on U (an embedding) and so it respects all inequations among members of U, including those in eqdiag($\mathbf{U}$). Similarly, if $\mathbf{U}$ has fewer than n non-trivial $\{\cdot, \div\}$, no matter how many $=$ - tests it has. This leaves only one case to consider:

$(*)$ *There are exactly n non-trivial $\{\cdot, \div\}$ and no non-trivial $=$ - tests in* eqdiag($\mathbf{U}$).

This would be the case, for example, it eqdiag($\mathbf{U}$) comprises all the calls made to compute $w = x_1y + \cdots + x_ny^n$ by the Horner Rule without entries in eqdiag($\mathbf{U}$) involving z or any $=$ - test; we can then take $\pi : \mathbf{U} \to \mathbf{U}$ to be the identity on $x_1, \dots, x_n, y$ and set $\pi(z) = w$, which is a homomorphism since the inequality $z \neq w$ is not in eqdiag($\mathbf{U}$).

For any $X \subseteq F(z, \vec{x}, y)$, let

$$C(X) = \text{the closure of } X \text{ under trivial operations},$$

and define the (non-trivial) *rank* of every $w \in U$ by the recursion

$$R_0 = C(\{z, \vec{x}, y\}),$$
$$R_{m+1} = C(\{uv, \frac{u}{v} : u, v \in R_m\})$$
$$\text{rank}(w) = \min\{m : w \in R_m\}.$$

Case 1, $U \subseteq R_0$, i.e., every element of U is constructed from the constants and the indeterminates by trivial operations. Define $\mathbf{U}_1 \subseteq_p \mathbf{U}$ by deleting from eqdiag($\mathbf{U}$) all non-trivial $\{\cdot, \div\}$, i.e., setting

$$\text{eqdiag}(\mathbf{U}_1) = \{(\circ, a, b, c) \in \text{eqdiag}(\mathbf{U}) : (\circ, a, b, c) \text{ is trivial}\}.$$

Let $\pi : F(z, \vec{x}, y) \to F(\vec{x}, y)$ be the partial ring homomorphism given by Lemma 9B.2 which is total and injective on $U_1 = U$, fixes $F(y)$ and satisfies (9C-2); its restriction $\pi \restriction U : \mathbf{U} \rightarrowtail \mathbf{F}(\vec{x}, y)$ is an embedding by (9A-1).

Case 2. $\max\{\text{rank}(w) : w \in U\} = m + 1$ *for some* m. Let

$$U_0 = \{x \in U : \text{rank}(u) \leq m\},$$

and define the structure $\mathbf{U}_0 \subseteq_p \mathbf{U}$ by

$$\begin{aligned}\text{eqdiag}(\mathbf{U}_0) = \{(\circ, a, b, c) \in \text{eqdiag}(\mathbf{U}) :\ & \text{either } a, b, c \in R_{m-1} \\ & \text{or } a, b, c \in R_m \text{ and } (\circ, a, b, c) \text{ is trivial}\}.\end{aligned}$$

The structure $\mathbf{U}_0$ has fewer than n non-trivial $\{\cdot, \div\}$ because $U_0 \subseteq R_m$, and so Lemma 9B.2 supplies us with an embedding

$$\pi : \mathbf{U}_0 \rightarrowtail \mathbf{F}(\vec{x}, y)$$

which fixes $F(y) \cap U_0$ and satisfies (9C-2). It is not defined on elements of U or rank $m + 1$, and the idea is to extend it to all of U by a sequence of steps as follows.

There must exist some $w \in U$ with rank$(w) = m + 1$ such that

$$w = uv \text{ or } w = \frac{u}{v} \text{ with } u, v \in U_0,$$

since these are the only operations in eqdiag($\mathbf{U}$) which increase rank. Choose one such w_1 and assume (for definiteness) that

$$w_1 = \frac{u_1}{v_1} \text{ with } u_1, v_1 \in U_0, v_1 \neq 0, \text{ so that } \pi(v_1) \neq 0.$$

Let

$$U_1 = \{x + f w_1 \in U : x \in U_0, f \in F\},$$

and, as above for $\mathbf{U}_0$, set

$$\begin{aligned}\text{eqdiag}(\mathbf{U}_1) = \{(\circ, a, b, c) \in \text{eqdiag}(\mathbf{U}) :\ & \text{either } (\circ, a, b, c) \in \text{eqdiag}(U_0) \\ & \text{or } a, b, c \in U_1 \text{ and } (\circ, a, b, c) \text{ is trivial}\}.\end{aligned}$$

Notice that *every* $y \in U_1$ *can be written uniquely in the form* $x + fw_1$, because if $x + fw_1 = x' + f'w_1$ with $f \neq f'$, then $(f' - f)w_1 = x - x'$ which implies that $\mathrm{rank}(w_1) \leq m$. So we can extend π to U_1 by setting

$$\pi(x + fw_1) = \pi(x) + f\frac{\pi(u_1)}{\pi(v_1)} \quad (x \in U_0).$$

Moreover, every entry $(\circ, a, b, c) \in \mathrm{eqdiag}(\mathbf{U}_1)$ in which one of a, b is $x + fw_1$ with $f \neq 0$ is trivial. This implies easily that $\pi : \mathbf{U}_1 \to \mathbf{F}(\vec{x}, y)$ is a homomorphism.

If $U_1 = U$, we are done. If not, then there is some $w_2 \in U \setminus U_1$ with rank $m+1$ introduced by a non-trivial multiplication or division with arguments in U_0 just like w_1. We let

$$U_2 = \{x + fw_2 \in U : x \in U_1, f \in F\},$$

$$\begin{aligned}\mathrm{eqdiag}(\mathbf{U}_2) = \{(\circ, a, b, c) \in \mathrm{eqdiag}(\mathbf{U}) :\ &\text{either } (\circ, a, b, c) \in \mathrm{eqdiag}(U_1)\\ &\text{or } a, b, c \in U_2 \text{ and } (\circ, a, b, c) \text{ is trivial}\}.\end{aligned}$$

The members of U_2 are (easily) uniquely represented in the form which puts them in U_2, and we can use the same extension procedure to get a homomorphism $\pi : \mathbf{U}_2 \to \mathbf{F}(\vec{x}, y)$.

To complete the proof, we repeat this process to get a sequence of structures

$$\mathbf{U}_0 \subseteq_p \mathbf{U}_1 \subseteq_p \cdots \subseteq_p U_k = \mathbf{U}$$

and successive extensions of π which finally produce the required homomorphism $\pi : \mathbf{U} \to \mathbf{F}(\vec{x}, y)$. ⊣

Note that Case 2 arises if $\mathbf{U}$ is constructed by using Horner's Rule to compute

$$z = x_1 y + \cdots + x_n y^n = yv = y(x_1 + x_2 y + \cdots x_n y^{n-1}).$$

Here z is an element of largest rank n introduced in U by the non-trivial multiplication of y with $v = x_1 + x_2 y + \cdots x_n y^{n-1}$. Now $U_1 = U \setminus \{z\}$ and

$$\mathrm{eqdiag}(\mathbf{U}_1) = \mathrm{eqdiag}(\mathbf{U}) \setminus \{yv = z\}.$$

The embedding defined on $\mathbf{U}_1$ is extended to a homomorphism on $\mathbf{U}$ which satisfies $\pi(z) = y\pi(v)$. So this is a case where π is not an embedding, because $z \neq yv$ but $\pi(z) = \pi(yv)$. The point is that although $\mathbf{U}$ can compute yv, it does not check that it is $\neq z$, and so π is a structure homomorphism.

9D. Horner's rule is $\{+,-\}$-optimal for nullity

Notice first that we can test whether $a_0 + a_1 w = 0$ by executing three multiplications, equality tests and no $\{+,-\}$ (additions/subtractions): first check if any of a_0, a_1, w is 0 and give the correct answer for these cases, and if none applies, set

$$f(a_0, a_1, w) = \text{if } \ a_0^2 \neq (a_1 w)^2 \text{ then ff else if } a_0 = a_1 w \text{ then ff else tt}.$$

The method works for any field with characteristic $\neq 2$ and combines with Horner's rule to decide whether $a_0 + a_1 x + \cdots + a_n x^n = 0$ using $(n-1)$ additions (and $(n+2)$ multiplications) along with equality tests: apply Horner's rule to compute $w = a_1 + \cdots + a_n x^{n-1}$ using $n-1$ multiplications and additions and then use the subroutine above with this w. This gives

$$\text{calls}_{\{+,-\}}(\mathbf{F}, N_F, a_0, \ldots, a_n, b) \leq n - 1 \quad (\text{char}(F) \neq 2, n \geq 1)$$

with $\mathbf{F} = (F, 0, 1, +, -, \cdot, \div, =)$, and so the lower bound for the number of $\{+,-\}$ required to test nullity with unlimited calls to $\cdot, \div, =$ is $\leq n-1$.[19] We will establish that $(n-1)$ is the correct lower bound, first in a special case which simplifies the computations and then get from it the following result by chasing isomorphisms.

9D.1. **Theorem.** *If* $\mathbf{F} = \mathbf{R}$ *or* $\mathbf{F} = \mathbf{C}$, $n \geq 2$ *and* $a_0, a_1, \ldots, a_n, b \in F$ *are algebraically independent* (over $\mathbb{Q}$), *then*

$$\text{calls}_{\{+,-\}}(\mathbf{F}, N_F, a_0, a_1, \ldots, a_n, b) = n - 1. \tag{9D-1}$$

The special case of this that we will prove first will follow by the Homomorphism Test 4F.2 from the following, stronger Lemma.

Suppose $y, z, x_1, \ldots, x_n$ are algebraically independent real numbers and let $\mathbb{Q}(y, z, x_1, \ldots, z_n)$ be the field of their rational functions.

$a \pm b = c$ in eqdiag$(\mathbb{Q}(y, z, \vec{x}))$ is *trivial* (with respect to y) if $a, b \in \mathbb{Q}(y)$.

9D.2. **Lemma.** *Suppose* $n \geq 2$, $z, x_1, \ldots, x_n, y$ *are algebraically independent, positive real numbers,* $h \in \mathbb{Q}^+$ *and* $\mathbf{U} \subseteq_p \mathbf{Q}(y, z, x_1, \ldots, x_n)$ *is finite, generated by*

$$(U \cap \mathbb{Q}) \cup \{y, z, \vec{x}\})$$

and having fewer than $(n-1)$ *non-trivial additions and subtractions.*

Then there is a partial ring homomorphism $\pi : \mathbb{Q}(y, z, x_1, \ldots, x_n) \rightharpoonup \mathbb{R}$ *which is the identity on* $\mathbb{Q}(y)$, *total and injective on* U *and such that*

$$h\pi(z) = \pi(x_1)y^1 + \cdots + \pi(x_n)y^n = 0. \tag{9D-2}$$

[19] I do not know if Horner rule's $2n$ is the correct lower bound for the combined number of operations with unlimited equality tests in the generic case. The current results give a possibly too low total lower bound of $2n - 1$.

With (9A-1), the Lemma delivers an embedding $\pi \restriction U : \mathbf{U} \rightarrowtail \mathbf{R}$ which satisfies (9D-2) and with the Homomorphism Test 4F.2 this easily gives Theorem 9D.1—we will give this argument further on.

Proof is by induction on $n \geq 2$ starting with the following

Sublemma 1 (Preliminary case). *There are no non-trivial* $\{+,-\}$ *in* $\mathbf{U}$.

Proof. It follows that every member of U is of the form

$$M = x_1^{b_1} \cdots x_n^{b_n} z^c p(y) \tag{9D-3}$$

where $b_1, \dots, b_n, c \in \mathbb{Z}$ and $p(y) \in \mathbb{Q}(y)$. Let

$$\pi : \mathbb{Q}(y, z, x_1, \dots, x_n) \rightharpoonup \mathbb{R}$$

be the partial homomorphism induced by the substitution

$$z \mapsto \frac{1}{h}\Big(x_1 y^1 + \cdots + x_n y^n\Big).$$

This is total on U and satisfies (9D-2), so it suffices to show that it it is injective on the set of all numbers of the form (9D-3) which includes U.

Suppose then that

$$x_1^{b_1} \cdots x_n^{b_n} \pi(z)^c p(y) = x_1^{b'_1} \cdots x_n^{b'_n} \pi(z)^{c'} p'(y)$$

where $p(y), p'(y) \in \mathbb{Q}(y)$. By clearing the denominators of the rational functions $p(y), p'(y)$ and the negative powers by cross-multiplying, we may assume that all exponents in this equation are in $\mathbb{N}$, $b_i b'_i = 0$ for $i = 1, \dots, n$, $cc' = 0$, and $p(y), p'(y)$ are polynomials in $\mathbb{Q}[y]$. The hypothesis now takes the form

$$x_1^{b_1} \cdots x_n^{b_n} \frac{1}{h^c}\Big(x_1 y^1 + \cdots + x_n y^n\Big)^c p(y) = x_1^{b'_1} \cdots x_n^{b'_n} \frac{1}{h^{c'}}\Big(x_1 y^1 + \cdots + x_n y^n\Big)^{c'} p'(y).$$

If we expand these two polynomials in powers of x_1, the leading terms must be equal, so

$$\frac{1}{h^c} x_2^{b_2} \cdots x_n^{b_n} y^c p(y) x_1^{b_1 + c} = \frac{1}{h^{c'}} x_2^{b'_2} \cdots x_n^{b'_n} y^{c'} p'(y) x_1^{b'_1 + c'},$$

hence $x_2^{b_2} \cdots x_n^{b_n} = x_2^{b'_2} \cdots x_n^{b_n}$, hence $b_i = b'_i$ for $i = 2, \dots, n$; and since $b_i b'_i = 0$, all these numbers are 0. If we repeat this argument[20] using x_n rather than x_1, we get that $b_1 = b'_1 = 0$ also, so that the original assumption takes the simpler form

$$\frac{1}{h^c}\Big(x_1 y^1 + \cdots + x_n y^n\Big)^c p(y) = \frac{1}{h^{c'}}\Big(x_1 y^1 + \cdots + x_n y^n\Big)^{c'} p'(y);$$

[20]This is a part of the proof where $n \geq 2$ is used.

and if we expand again in powers of x_1 and equate the leading terms we get

$$\frac{1}{h^c}y^c p(y)x_1^c = \frac{1}{h^{c'}}y^{c'}p'(y)x_1^{c'},$$

which yields $c = c'$ and finally $p(y) = p'(y)$ as required. ⊣ (Sublemma 1)

The basis of the induction $n = 2$ is covered by the preliminary case.

In the induction step with $n > 2$, if the preliminary case does not apply, then there must exist a "least complex" non-trivial addition or subtraction in **U** of the form

(9D-4) $$w = x_1^{b_1}\cdots x_n^{b_n}z^c p(y) \pm x_1^{b'_1}\cdots x_n^{b'_n}z^{c'}p'(y)$$

where $p(y), p'(y) \in \mathbb{K}(y)$ and the component parts

$$u = x_1^{b_1}\cdots x_n^{b_n}z^c p(y), \quad v = x_1^{b'_1}\cdots x_n^{b'_n}z^{c'}p'(y)$$

are also in U. We may, in fact, assume that this is an addition, by replacing $p'(y)$ by $-p'(y)$ if necessary.

Sublemma 2. We may assume that in (9D-4)*, $b'_i = 0$ for $i = 1, \dots, n$, $c' = 0$, and $p(y)$, $p'(y)$ are polynomials, i.e.,* (9D-4) *is of the form*

(9D-5) $$w = x_1^{b_1}x_2^{b_2}\cdots x_n^{b_n}z^c p(y) + p'(y)$$

with $p(y), p'(y) \in \mathbb{Q}[y]$.

Proof. Let

$$W = x_1^{-b'_1}x_2^{-b'_2}\cdots x_n^{-b'_n}z^{-c'}p_d(y)p'_d(y)$$

where $p_d(y), p'_d(y)$ are the denominators of $p(y), p'(y)$ and replace (9D-4) in eqdiag(**U**) by the operations

$$u_1 = Wu, \quad v_1 = Wv, \quad w_1 = u_1 + v_1, \quad w = \frac{w_1}{W}$$

along with all the multiplications, divisions and trivial additions and subtractions required to compute W. If $\mathbf{U}'$ is the resulting structure, then $U \subseteq U'$ and the fixed, non-trivial addition in **U** has been replaced by one of the form (9D-5). It is not quite true that $\mathbf{U} \subseteq_p \mathbf{U}'$, because the equation $w = u + v$ is in eqdiag(**U**) but not in eqdiag($\mathbf{U}'$). On the other hand, if $\pi : \mathbb{R} \rightharpoonup \mathbb{R}$ is a partial ring homomorphism which is total and injective on U', then its restriction $\pi \restriction U : \mathbf{U} \rightarrowtail \mathbf{R}$ is an embedding, because it preserves all the other entries in eqdiag(**U**) and $\pi(u+v) = \frac{\pi(u_1)+\pi(v_1)}{\pi(W)} = \frac{\pi(w_1)}{\pi(W)} = \pi(w)$. ⊣ (Sublemma 2)

Now, either some $b_i \neq 0$ or $b_i = 0$ for $i = 1, \dots, n$ and $c \neq 0$ in (9D-5), otherwise the chosen addition is trivial.

Case 1, *Some* $b_i \neq 0$ *in* (9D-5). We assume to simplify the notation that $b_1 \neq 0$ and in fact $b_1 > 0$, by applying the "reflection" of (9D-5) with the $x_j^{b_j}$ on the right, if only one $b_i \neq 0$ in (9D-5) and it is negative.

Step 1. Using the hypotheses on $x_1, \dots, x_n, z$, let for each $f \in \mathbb{Q}^+$

$$\rho_f : \mathbb{Q}(y, z, x_1, \dots, x_n) \rightharpoonup \mathbb{R}$$

be the partial homomorphism induced by the substitution

$$x_1 \mapsto \rho_f(x_1) = \Big(\frac{f^{b_1}}{x_2^{b_2} \cdots x_n^{b_n} z^c}\Big)^{\frac{1}{b_1}} = f\overline{x}_2^{-b_2} \cdots \overline{x}_n^{-b_n} \overline{z}^{-c}, \tag{9D-6}$$

where

$$\overline{x}_i = x_i^{\frac{1}{b_1}} \text{ for } i = 2, \dots, n \text{ and } \overline{z} = z^{\frac{1}{b_1}}.$$

The Substitution Lemma 9A.1 insures that for all but finitely $f \in \mathbb{Q}^+$, ρ_f is total and injective on U, we fix one such f and we let

$$\rho_1 = \rho_f.$$

The image structure $\rho_1[\mathbf{U}]$ is generated by $z, y, x_2, \dots, x_n, f\overline{x}_2^{-b_2} \cdots \overline{x}_n^{-b_n}\overline{z}^{-c}$. We define $\mathbf{U}_1$ by adding to its universe $\overline{x}_2, \dots, \overline{x}_n, \overline{z}$ and enough multiplications and inversions to compute $x_2, \dots, x_n, z$ from $\overline{x}_2, \dots, \overline{x}_n, \overline{z}$, so that $\mathbf{U}_1$ is generated by the algebraically independent set $y, \overline{z}, \overline{x}_2, \dots, \overline{x}_n$ (and some constants in $\mathbb{Q}$). The map

$$\rho_1 \restriction U : \mathbf{U} \rightarrowtail \mathbf{U}_1$$

is an embedding which takes trivial $\{+,-\}$ to trivial ones, because it is the identity on $\mathbb{Q}(y)$, and it transforms the non-trivial addition in (9D-5) into a trivial one since

$$\rho_1(x_1^{b_1} x_2^{b_2} \cdots x_n^{b_n} z^c p(y)) = f^{b_1} p(y).$$

So there are fewer than $n-2$ non-trivial $\{+,-\}$ in $\mathbf{U}_1$, but we will not use this: the significant feature of $\mathbf{U}_1$ is that it is generated by $y, \overline{z}, \overline{x}_2, \dots, \overline{x}_n$—there is no x_1 in it.

Step 2. For each $t \in \mathbb{Q}^+$, the map

$$\sigma_t : (y, \overline{z}, \overline{x}_2, \dots, \overline{x}_n) \mapsto (y, \overline{z}, t\overline{x}_2, \dots, t\overline{x}_n)$$

induces an embedding of $\mathbb{Q}(y, \overline{z}, \overline{x}_2, \dots, \overline{x}_n)$ into itself because $y, \overline{z}, t\overline{x}_2, \dots, t\overline{x}_n$ are algebraically independent. The idea is to follow it by some

$$\rho_t \text{ induced by a suitable substitution } \overline{z} \mapsto \alpha_t,$$

so that the composition $\pi = \rho_t \circ \sigma_t \circ \rho_1$ satisfies the conclusion of the Lemma. To see what conditions α_t must satisfy, we compute:

$$\begin{aligned}
&\pi(x_1)y + \pi(x_2)y^2 + \cdots + \pi(x_n)y^n \\
&\quad = \rho_t(\sigma_t(\rho_1(x_1)))y + t^{b_1}\overline{x}_2^{b_1}y^2 + \cdots + t^{b_1}\overline{x}_n^{b_1}y^n \\
&\quad\quad = \rho_t\sigma_t\Big(\frac{f}{\overline{x}_2^{b_2}\cdots\overline{x}_n^{b_n}\overline{z}^c}\Big)y + t^{b_1}\overline{x}_2^{b_1}y^2 + \cdots + t^{b_1}\overline{x}_n^{b_1}y^n \\
&\quad\quad = \frac{f}{t^{b_2}\overline{x}_2^{b_2}\cdots t^{b_n}\overline{x}_n^{b_n}}\rho_t(\overline{z})^{-c} + t^{b_1}(\overline{x}_2^{b_1}y^2 + \cdots + \overline{x}_n^{b_1}y^n) \\
&= \frac{f}{t^d(\overline{x}_2^{b_2}\cdots\overline{x}_n^{b_n})}\alpha_t^{-c} + t^{b_1}(\overline{x}_2^{b_1}y^2 + \cdots + \overline{x}_n^{b_1}y^n) \text{ where } d = b_2 + \cdots + b_n.
\end{aligned}$$

We need this to be equal to $h\pi(z) = h\rho_t(\sigma_t(z)) = h\rho_t(\overline{z}^{b_1}) = h\alpha_t^{b_1}$, i.e., we must choose α_t so that

$$\frac{f}{t^d(\overline{x}_2^{b_2}\cdots\overline{x}_n^{b_n})}\alpha_t^{-c} + t^{b_1}(\overline{x}_2^{b_1}y^2 + \cdots + \overline{x}_n^{b_1}y^n) = h\alpha_t^{b_1},$$

or, multiplying by α_t^c,

$$\frac{f}{t^d(\overline{x}_2^{b_2}\cdots\overline{x}_n^{b_n})} + t^{b_1}(\overline{x}_2^{b_1}y^2 + \cdots + \overline{x}_n^{b_1}y^n)\alpha_t^c = h\alpha_t^{c+b_1}.$$

In other words, we need α_t to satisfy the polynomial equation

$$hX^{c+b_1} - t^{b_1}(\overline{x}_2^{b_1}y^2 + \cdots + \overline{x}_n^{b_1}y^n)X^c - \frac{f}{t^d(\overline{x}_2^{b_2}\cdots\overline{x}_n^{b_n})} = 0.$$

For any positive t, the polynomial on the left has a negative value when $X = 0$ and it goes to ∞ as $X \to \infty$, so it has a root on the positive axis, and we fix α_t to be its least positive root. Moreover, for each $\alpha \in \mathbb{R}^+$, there are at most $d + b_1$ different t's such that $\alpha_t = \alpha$, because $n \geq 3 \geq 2$ and so t^{b_1} occurs with a positive coefficient in this equation, even if $c = d = 0$; so there are infinitely many distinct α_t's, and by the Substitution Lemma 9A.1 then, the partial homomorphism induced by the substitution $\overline{z} \mapsto \alpha_t$ is injective on U_1 for all but finitely many t's. We choose one such t to define ρ_t, and tracing back the computation, we verify that the composition $\pi = \rho_t\circ\sigma_t\circ\rho_1$ has the properties required by the Lemma.

Notice that we did not use the induction hypothesis in either the preliminary case or Case 1. We will need it in the remaining

Case 2, $b_1 = \cdots = b_n = 0$ *and* $c \neq 0$ *in* (9D-5), which now takes the form

(9D-7) $$w = z^c p(y) + p'(y).$$

Sublemma 3. *For all but finitely many $f \in \mathbb{Q}^+$, the partial ring homomorphism $\rho_f : \mathbb{Q}(y, z, x_1, \dots, x_n) \rightharpoonup \mathbb{R}$ induced by the substitution*

$$z \mapsto \rho_f(z) = f$$

is total and injective on U.

This follows from Lemma 9A.1. We fix one such f and we let

$$\mathbf{U}_1 = \rho_f[\mathbf{U}].$$

It follows that $\rho_f \restriction U : \mathbf{U} \rightarrowtail\!\!\!\!\rightarrow \mathbf{U}_1$ is an isomorphism and $\mathbf{U}_1$ has fewer than $n-2$ non-trivial $\{+,-\}$, since $\rho_f(z^c p(y)) = f^c p(y)$. We also note that $\mathbf{U}_f$ is generated by $\{y, x_1, \dots, x_n\}$ and some constants in $\mathbb{Q}$—there is no z in it.

By the induction hypothesis on $x_1, x_2, \dots, x_n$, treating x_1 as the z, for every $g \in \mathbb{Q}^+$ there is a partial homomorphism

$$\sigma_g : \mathbb{Q}(y, x_1, \dots, x_n) \rightharpoonup \mathbb{R}$$

which is total and injective on U_1 and such that

$$g\sigma_g(x_1) = \sigma_g(x_2)y + \cdots \sigma_g(x_n)y^{n-1}. \tag{9D-8}$$

The idea is to find some g, α_g such that if $\rho_2 : \mathbb{Q}(y, x_1, \dots, x_n) \rightharpoonup \mathbb{R}$ is the partial homomorphism generated by $x_1 \mapsto \alpha_g$, then the composition

$$\pi = \sigma_g \circ \rho_2 \circ \rho_f$$

does the trick. So assume we have g and α_g and compute:

$$\begin{aligned}
&\pi(x_1)y + \pi(x_2)y^2 + \cdots + \pi(x_n)y^n \\
&\qquad = \sigma_g(\rho_2(x_1))y + \sigma_g(x_2)y^2 + \cdots + \sigma_g(x_n)y^n \\
&\qquad = \sigma_g(\rho_2(x_1))y + y(\sigma_g(x_2)y + \cdots + \sigma_g(x_n)y^{n-1}) \\
&\qquad = \sigma_g(\rho_2(x_1)y - yg\sigma_g(x_1) = \sigma_g(\rho_2(x_1)y - ygx_1).
\end{aligned}$$

For ρ_2 to work, we must have

$$\sigma_g(\rho_2(x_1)y - ygx_1) = h\pi(z) = h\sigma_g(\rho_2(\rho_f(z))) = \sigma_g(hf);$$

and this is insured if $\rho_2(x_1)y - ygx_1 = hf$, i.e., if

$$\alpha_g = \frac{1}{y}(gyx_1 + hf).$$

There are infinitely many distinct α_g's, since $g \mapsto \alpha_g$ is injective, and so the homomorphism induced by $x_1 \mapsto \alpha_g$ is injective on U_1 for all but finitely g's, we choose one such g to define $\rho_2(x_1)$ and trace the computation backward to complete the proof. ⊣

Proof of Theorem 9D.1 from Lemma 9D.2. For any field F, let

$$N_F^*(z, x_1, \ldots, x_n, y) \iff z = x_1y + \cdots + x_ny^n \\ \iff -z + x_1y + \cdots + x_ny^n = 0 \iff N_F(-z, x_1, \ldots, x_n, y),$$

so that

$$\text{calls}_{\{+,-\}}(\mathbf{F}, N_F, -z, x_1, \ldots, x_n, y) = \text{calls}_{\{+,-\}}(\mathbf{F}, N_F^*, z, x_1, \ldots, x_n, y).$$

For $\mathbf{R}$ first, fix any algebraically independent $z, y, x_1, \ldots, x_n \in \mathbb{R}^+$, so that by Lemmas 4F.2 and 9D.2 (with $n \geq 2, h = 1$),

(9D-9) $$\text{calls}_{\{+,-\}}(\mathbf{R}, N_{\mathbb{R}}^*, z, x_1, \ldots, x_n, y) = n - 1.$$

For any algebraically independent $a_0, a_1, \ldots, a_n, b \in \mathbb{R}$, the map

$$\pi : \mathbb{Q}(a_0, a_1, \ldots, a_n, b) \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbb{Q}(-z, x_0, \ldots, x_n, y)$$

induced by the correspondence $a_0 \mapsto -z, a_i \mapsto x_i, b \mapsto y$ is an isomorphism, which can then be extended to an automorphism

$$\pi : \mathbf{R} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbf{R}.$$

It follows by Theorem 4F.3 and (9D-9) that

$$\text{calls}_{\{+,-\}}(\mathbf{R}, N_{\mathbb{R}}, a_0, \ldots, a_n, b) \\ = \text{calls}_{\{+,-\}}(\mathbf{R}, N_{\mathbb{R}}, -z, x_1, \ldots, x_n, y) \\ = \text{calls}_{\{+,-\}}(\mathbf{R}, N_{\mathbb{R}}^*, z, x_1, \ldots, x_n, y) = n - 1.$$

The argument is similar for $\mathbf{C}$, if we notice that the proof of Lemma 9D.2 also works for $\mathbf{C}$, since the embeddings into $\mathbb{R}$ constructed in it can be viewed as embeddings into $\mathbb{C}$. ⊣

9E. Counting identity tests along with $\{+,-\}$

If we also count calls to the equality relation, then Horner's rule clearly requires n additions and one equality test to decide the nullity relation, for a total of $n+1$. This may well be the correct lower bound for $\text{calls}_{\{+,-,=\}}(\mathbf{F}, N_{\mathbb{R}}, \vec{a}, b)$ on an infinite number of tuples $\vec{a}, b$ for $\mathbf{R}$ and $\mathbf{R}$, but I do not know how to prove this now. In any case, it fails for algebraically independent inputs:

9E.1. **Lemma.** *If $a_0, a_1, b \in \mathbb{C}$ are algebraically independent and $\mathbf{U} \subseteq_p \mathbf{C}$ with*

$$\text{eqdiag}(\mathbf{U}) = \{u = a_1b, w = a_0 + u, v = \frac{1}{w}\},$$

then $\mathbf{U} \Vdash_c^{\mathbf{C}} \frac{1}{a_0+a_1b}\downarrow$, and so

$$\text{calls}_{\{+,-,=\}}(\mathbf{C}, N_{\mathbb{R}}, a_1, a_2, b) \leq 1.$$

Proof. Every homomorphism $\pi : \mathbf{U} \to \mathbf{C}$ must be defined on v and satisfy

$$\pi(v) = \frac{1}{\pi(w)},$$

so that $\pi(w) \neq 0$, so $\pi(a_0) + \pi(a_1)\pi(b) = \pi(w) \neq 0$. $\dashv$

The trick here is to use division (which we are not counting) in place of the natural inequality test to check that $w \neq 0$, so one might think that allowing only multiplications would enforce at least two $+, -$ or $=$ - tests to certify $a_0 + a_1 b \neq 0$, but this does not work either: if the single inequality test $a_0^2 \neq (a_1 b)^2$ is in eqdiag($\mathbf{U}$), then, easily, $\mathbf{U} \Vdash a_0 + a_1 b \neq 0$. We outline a proof the best result for the generic case and leave open the possibility that Horner's rule is optimal on infinitely many non-generic inputs.

9E.2. **Theorem.**[21] *If $n \geq 1$, $\mathbf{F} = \mathbf{R}$ or $\mathbf{C}$ and $a_0, a_1, \dots, a_n, b \in F$ are algebraically independent, then*

$$\text{(9E-1)} \qquad \text{calls}_{\{+,-,=\}}(\mathbf{F}, N_F, a_0, a_1, \dots, a_n, b) = n.$$

A partial homomorphism $\pi : \mathbb{Q}(y, z, x_1, \dots, x_n) \rightharpoonup \mathbb{R}$ is *proper* on a set U if it is total on U and

$$[x \in U \ \&\ x \neq 0] \Longrightarrow \pi(x) \neq 0;$$

this insures, in particular, that if $u \div v = w \in$ eqdiag($\mathbf{U}$), then $\pi(v) \neq 0$.

Suppose $\mathbf{U} \subseteq_p \mathbb{Q}(y, z, x_1, \dots, x_n)$. An addition $u + v$, subtraction $u - v$ or inequality test $u \neq v$[22] in eqdiag($\mathbf{U}$) is *trivial* if $u, v \in \mathbb{Q}(y)$.

The theorem follows as before from the adaptation of Lemma 9D.2 (and the general version of Lemma 9E.1 for the upper bound).

9E.3. **Lemma.** *Suppose $n \geq 1$, $z, x_1, \dots, x_n, y$ are algebraically independent, positive real numbers, $h \in \mathbb{Q}^+$ and $\mathbf{U} \subseteq_p \mathbf{Q}(y, z, x_1, \dots, x_n)$ is finite, generated by*

$$(U \cap \mathbb{Q}) \cup \{y, z, \vec{x}\})$$

and having fewer than n non-trivial additions, subtractions and inequality tests.

Then there is a partial ring homomorphism $\pi : \mathbb{R} \rightharpoonup \mathbb{R}$ which is the identity on $\mathbb{Q}(y)$, proper on U and such that

$$\text{(9E-2)} \qquad h\pi(z) = \pi(x_1)y^1 + \cdots + \pi(x_n)y^n = 0.$$

[21] A differently formulated but equivalent result is proved for algebraic decision trees in Bürgisser, Lickteig, and Shub [1992].

[22] There is no need to include entries of the form $(=, u, v, \mathrm{tt})$ in eqdiag($\mathbf{U}$), because every homomorphism on $\mathbf{U}$ automatically respects them. So the number of significant $=$ - tests is the number of entries $(=, u, v, \mathrm{ff})$ in eqdiag($\mathbf{U}$).

Proof is by induction on $n \geq 1$. It is almost exactly (and a bit simpler) than the proof of Lemma 9D.2, and we will only describe the necessary changes, mostly in the Sublemma corresponding to 1 and in the mild modification of the statements of the other Sublemmas, to include inequations.

Sublemma 1 (Preliminary case). *There are no non-trivial* $\{+, -, \neq\}$ *in* $\mathbf{U}$.

Proof exactly as in Lemma 9D.2, except that we do not need to prove injectivity, for which $n \geq 2$ is used, only properness, which is practically trivial. ⊣ (Sublemma 1)

A counterexample to the injectivity of π in the preliminary case when $n = 1$ is given by the distinct polynomials z and $x_1 y$ with $g = 1$, for which

$$\pi(z) = x_1 y = \pi(x_1 y).$$

The rest of the proof of Lemma 9D.2 can be adapted to this case (with proper rather than injective homomorphisms) essentially word-for-word, as it does not depend on $n \geq 2$. ⊣

REFERENCES

P. Bürgisser, T. Lickteig, and M. Shub

[1992] *Test complexity of generic polynomials*, ***Journal of Complexity***, vol. 8, pp. 203–215. *171.*

P. Bürgisser and T. Lickteig

[1992] *Verification complexity of linear prime ideals*, ***Journal of pure and applied algebra***, vol. 81, pp. 247–267. *156, 160.*

Joseph Busch

[2007] *On the optimality of the binary algorithm for the Jacobi symbol*, ***Fundamenta Informaticae***, vol. 76, pp. 1–11. *111.*

[2009] *Lower bounds for decision problems in imaginary, norm-Euclidean quadratic integer rings*, ***Journal of Symbolic Computation***, vol. 44, pp. 683–689. *111.*

Lou van den Dries and Yiannis N. Moschovakis

[2004] *Is the Euclidean algorithm optimal among its peers?*, ***The Bulletin of Symbolic Logic***, vol. 10, pp. 390–418. *vi, 83, 101, 108, 110, 130, 131.*

[2009] *Arithmetic complexity*, ***ACM Trans. Comput. Logic***, vol. 10, no. 1, pp. 1–49. *vi, 83, 95, 101, 131, 143, 146, 147, 148, 150.*

P. van Emde Boas

[1990] *Machine models and simulations*, in van Leeuwen [1990], pp. 1–66. *iv.*

Herbert Enderton

[2001] ***A mathmatical introduction to logic***, Academic Press, Second edition. *20.*

G. H. HARDY AND E. M. WRIGHT
[1938] ***An introduction to the theory of numbers***, Clarendon Press, Oxford, fifth edition (2000). *120, 146.*

STEPHEN C. KLEENE
[1952] ***Introduction to metamathematics***, D. Van Nostrand Co, North Holland Co. *39.*

D. E. KNUTH
[1973] ***The Art of Computer Programming. Fundamental Algorithms***, second ed., Addison-Wesley. *24.*

J. VAN LEEUWEN
[1990] *Handbook of theoretical computer science*, vol. A, Algorithms and Complexity, Elsevier and the MIT Press. *173.*

YISHAY MANSOUR, BARUCH SCHIEBER, AND PRASOON TIWARI
[1991a] *A lower bound for integer greatest common divisor computations*, ***Journal of the Association for Computing Machinery***, vol. 38, pp. 453–471. *95.*
[1991b] *Lower bounds for computations with the floor operation*, ***SIAM Journal on Computing***, vol. 20, pp. 315–327. *136, 143.*

J. MCCARTHY
[1963] *A basis for a mathematical theory of computation*, ***Computer programming and formal systems*** (P. Braffort and D Herschberg, editors), North-Holland, pp. 33–70. *v, 34, 37.*

GREGORY L. MCCOLM
[1989] *Some restrictions on simple fixed points of the integers*, ***The Journal of Symbolic Logic***, vol. 54, pp. 1324–1345. *36.*

YIANNIS N. MOSCHOVAKIS
[1984] *Abstract recursion as a foundation of the theory of algorithms*, ***Computation and proof theory*** (M. M. et. al. Richter, editor), vol. 1104, Springer-Verlag, Berlin, Lecture Notes in Mathematics, pp. 289–364. *v, 35.*
[1989] *The formal language of recursion*, ***The Journal of Symbolic Logic***, vol. 54, pp. 1216–1252. *v.*
[1998] *On founding the theory of algorithms*, ***Truth in mathematics*** (H. G. Dales and G. Oliveri, editors), Clarendon Press, Oxford, posted on `www.math.ucla.edu/∼ynm/ papers/foundalg.pdf`, pp. 71–104. *v.*
[2001] *What is an algorithm?*, ***Mathematics unlimited – 2001 and beyond*** (B. Engquist and W. Schmid, editors), Springer, pp. 929–936. *v.*

[2006] ***Notes on set theory, second edition***, Undergraduate texts in mathematics, Springer. *9.*

A. M. Ostrowski

[1954] *On two problems in abstract algebra connected with Horner's rule*, ***Studies presented to R. von Mises***, Academic Press, New York, pp. 40–48. *153.*

V. Ya. Pan

[1966] *Methods for computing values of polynomials*, ***Russian Mathematical Surveys***, vol. 21, pp. 105–136. *155.*

Rózsa Péter

[1951] ***Rekursive funktionen***, Akadémia Kiadó, Budapest. *39.*

Vaughan Pratt

[1975] *Every prime has a succint certificate*, ***SIAM Journal of computing***, vol. 4, pp. 214–220. *91.*

[2008] *Euclidean gcd is exponentially suboptimal: why gcd is hard to analyse*, unpublished manuscript. *53.*

J. Stein

[1967] *Computational problems associated with Racah Algebra*, ***Journal of Computational Physics***, vol. 1, pp. 397–405. *24.*

A. P. Stolboushkin and M. A. Taitslin

[1983] *Deterministic dynamic logic is strictly weaker than dynamic logic*, ***Information and Control***, vol. 57, pp. 48–55. *35, 53.*

Jerzy Tiuryn

[1989] *A simplified proof of DDL $<$ DL*, ***Information and Computation***, vol. 82, pp. 1–12. *35, 40, 53.*

Shmuel Winograd

[1967] *On the number of multiplications required to compute certain functions*, ***Proceedings of the National Academy of Sciences, USA***, vol. 58, pp. 1840–1842. *156.*

[1970] *On the number of multiplications required to compute certain functions*, ***Communications on pure and applied mathematics***, vol. 23, pp. 165–179. *156.*