Title
Recursion versus Tail Recursion over Abstract Structures

Author
Bhaskar, Siddharth

Publication Date
2015

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Recursion versus Tail Recursion over Abstract Structures

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy in
Mathematics

by

Siddharth Kasi Bhaskar

2015

ABSTRACT OF THE DISSERTATION

Recursion versus Tail Recursion over Abstract Structures

by

Siddharth Kasi Bhaskar

Doctor of Philosophy in Mathematics

University of California, Los Angeles, 2015

Professor Yiannis N. Moschovakis, Chair

There are several ways to understand computability over first-order structures. We may admit functions given by arbitrary recursive definitions, or we may restrict ourselves to "iterative" functions computable by nothing more complicated than while loops.

In the classical case of recursion over the natural numbers, these two notions of computability coincide. However, this is not true in general. We ask whether there is a model-theoretic classification of structures over which iteration is as powerful as recursion. We give such a classification for non-locally finite structures, and examine the problem of iteration versus recursion for the locally finite structures of finite fields and finite abelian groups. Over these structures, we prove that the question of iteration versus recursion reduces to a hard open problem in computational complexity theory.

We also ask whether there are structures in which certain function may be more efficiently computable by recursion than iteration, according to some measure of complexity. We identify a family of such structures with arbitrarily large gaps in efficiency.

The dissertation of Siddharth Kasi Bhaskar is approved.

Itay Neeman

Matthias Aschenbrenner

Sheila A. Greibach

Yiannis N. Moschovakis, Committee Chair

University of California, Los Angeles

2015

Contents

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Yiannis Moschovakis, without whose guidance, support, and motivation this thesis would not exist.

I owe a debt of gratitude to all of my mathematics teachers. Among them I would like to name Richard Elman, who taught me algebra, and the late Ed Nelson, who introduced me to mathematical logic as an undergraduate.

I would also like to thank some of my fellow students and friends from graduate school. First of all, there are Sherwood Hachtman and Quinn Maurmann, rotating co-hosts of our semiweekly first year "logic dinner," which was both nutritious and educational. There is my roommate Will Feldman, with whom I spent several enjoyable days proving what turned out to be the first result in [2], and there is Andy Soffer, with whom I spent weeks studying algebra. Lastly, I would like to thank Rob Denomme, who taught me higher category theory and old-time fiddle bowing.

Outside of mathematics, there are dozens of people who helped me get to where I am today. In particular I would like to thank my parents, without whom neither I nor this thesis would exist. Then there are my two dear younger brothers, who instilled in me a tolerance for suffering that served me well during graduate school. Nor can I forget my *de facto* sibling, Ann Gong, who is as close a friend as friends can be despite the distance that separates us. Finally there is my lovely partner, Grace Haaland, who is more a part of this thesis than she knows.

CURRICULUM VITAE

2005-2009 Princeton University

Bachelor of Arts in Mathematics, June 2009

Chapter 1

Introduction and Preliminaries

The basic purpose of our research is to study the difference between *recursion* and *iteration* over abstract structures. By an abstract structure we mean a structure like in model theory, i.e. a tuple

$$\mathbf{A} = (A, f_1, \ldots, f_n)$$

where A is some set and $f_1, \ldots, f_n$ are functions (or relations) on A, which we call the *primitives*. The *signature* of $\mathbf{A}$ consists of one function symbol naming each f_i.

By a recursive function over $\mathbf{A}$, we mean a function specified by one or more syntactic recursive equations, which we call a *(McCarthy) recursive program*. The non-logical symbols in a recursive program come from the signature of $\mathbf{A}$. John McCarthy in [9] pioneered the idea of computing over abstract structures by making conditional tests, calls to primitives, and recursive calls.

For example, let $(\Sigma, <)$ be a linearly ordered set, and consider the structure of lists over Σ:

$$\mathbf{L} := (\Sigma^{<\omega}, <, \varepsilon, eq_\varepsilon, h, t, \hat{}, half_1, half_2)$$

where ε is the empty list, $<$ is the ordering on Σ, and for a list $u = \sigma_0 \dots \sigma_{n-1} \in \Sigma^{<\omega}$ and character $\sigma \in \Sigma$, we have

$$eq_\varepsilon(u) \iff n = 0$$

$$\sigma \hat{} u := \sigma\sigma_0 \dots \sigma_{n-1}$$

and if $n > 0$,

$$h(u) = \sigma_0$$

$$t(u) = \sigma_1 \dots \sigma_{n-1}$$

$$half_1(u) = \sigma_0 \dots \sigma_{\lceil n/2 \rceil - 1}$$

$$half_2(u) = \sigma_{\lceil n/2 \rceil} \dots \sigma_{n-1}$$

Then over this structure, we may program the familiar mergesort algorithm as follows:

$$\text{sort}(u) = \text{merge}(\text{sort}(\text{half}_1(u)), \text{sort}(\text{half}_2(u))) \tag{1.1}$$

where

$$\text{merge}(u, v) = \begin{cases} u & \text{if } eq_\varepsilon(v) \\ v & \text{if } eq_\varepsilon(u) \\ h(u) \hat{} \text{merge}(t(u), v) & \text{if } h(u) < h(v) \\ h(v) \hat{} \text{merge}(u, t(v)) & \text{if } h(u) < h(u) \end{cases} \tag{1.2}$$

Iterative programs A program over a given structure is *iterative* if, roughly speaking, it can be expressed using recursion no more complicated than "loops," such as for loops or while loops. This encompasses a broad class of familiar algorithms, but it does not apply to the program for mergesort above: the recursive equation 1.1 uses a more complicated "divide-and-conquer" construction.

On the other hand, the primitives $half_1$ and $half_2$ of $\mathbf{L}$ can be computed by iterative programs relative to the primitives eq_ε, h, t and $\hat{}$. For example,

$$half_2(u) = p(u, u)$$

where

$$p(u,v) = \begin{cases} p(t(u), t^2(v)) & \text{if } \neg eq_\varepsilon(v) \ \& \ \neg eq_\varepsilon(t(v)) \\ u & \text{otherwise} \end{cases}$$

The same algorithm can be expressed in pseudocode

$U := u,\ V := u$
`while`$(\neg eq_\varepsilon(V) \ \& \ \neg eq_\varepsilon(t(V)))$
  $U \leftarrow t(U)$
  $V \leftarrow t^2(V)$
`return` U

Iterative functions are exactly those computable by *tail recursive programs*, a restricted type of recursive program.

The classical case In the setting of recursion theory over abstract structures, classical recursion theory is the recursion theory of the structure of the natural numbers. The phrase "the structure of the natural numbers," could mean either

unary arithmetic

$$\mathbf{N}_u := (\mathbb{N}, 0, 1, S, Pd, =)$$

where S is the successor function and Pd is the predecessor function, or *binary arithmetic*

$$\mathbf{N}_b := (\mathbb{N}, 0, 1, \mathrm{mod}_2, \mathrm{iq}_2, em, om, =)$$

where mod_2 and iq_2 of n are the remainder and quotient respectively of n upon division by 2, and $em : n \mapsto 2n$, $om : n \mapsto 2n + 1$.

Many different formalisms have been proposed for defining recursive functions over the natural numbers, such as μ-recursion, Turing machines, and the Godel-Herbrand-Kleene calculus. These formalisms, as well as that of McCarthy recursive programs, all compute the same partial functions over the natural numbers.

Several of these formalisms, such as μ-recursion and Turing machines, express only iterative algorithms. Therefore over the natural numbers *recursion is no more powerful than tail recursion in terms of computability.*

Main Question However, this is not true in general. Jerzy Tiuryn, working in the field of dynamic logic, proved in [14] that for a certain structure $\mathbf{T}$ of binary trees, the reachability relation $R(x, y) \iff y \in \langle x \rangle_{\mathbf{T}}$ "y is in the subtree of $\mathbf{T}$ generated by x," is *not* computable by any tail recursion, while it is computable by a recursive program. Other examples of explicit separation results between recursion and tail recursion appear in [7, 8, 12, 16].

We ask over which structures is tail recursion as powerful as recursion. In other words:

Main Question 1: *For which structures* $\mathbf{A}$ *is every recursive partial function computable by a tail recursion?*

In [10], Moschovakis has developed a complexity theory for recursive programs that generalizes classical measures of complexity like time and space. It is natural, therefore, to ask

Main Question 2: *Are there structures* **A** *in which every recursive partial function is tail recursive, but some partial function is computed more efficiently by a recursive program than any tail recursion, for some natural measure of efficiency?*

Schematology Computer scientists working the field of schematology studied related questions in the 1960's through the 1980's. Instead of fixing a structure **A** and asking whether all recursive functions were tail recursive, they generally fixed a recursive program over a given signature and asked whether there was a tail recursive program equivalent to it for *all* structures in that signature (so-called *strong equivalence*). The problem of whether a given recursive program is strongly equivalent to some tail recursion is undecidable, but large classes of recursive programs were shown to be tail recursive. Walker and Strong have the most general result in [17] in this direction.

Organization of this paper In Chapter 1 we go over the basic definitions of structures, recursive programs, and tail recursive programs. In Chapter 2 we give a partial solution to Main Question 1 and introduce the structure of predecessor arithmetic. In Chapter 3 we investigate the question of recursion versus tail recursion over locally finite algebraic structures. In Chapter 4 we give a positive answer to Main Question 2.

1.1 Recursion over Abstract Structures

In this section we will precisely define our basic objects of study, namely *recursive programs* and *recursive functions over abstract structures*. We follow the exposition in [11, 10] throughout.

Notation For a set A, a *partial function f of arity k on A* is a function $f : D \to A$ for some subset $D \subseteq A^k$. For $\bar{a} \in A^k$ outside D, we say that "$f(\bar{a})$ diverges" and write $f(\bar{a}) \uparrow$.

If $g : A^k \rightharpoonup A$ is another partial function, then we write $f \sqsubseteq g$ to indicate that "g extends f," i.e. that

$$\forall \bar{a}, b \in A^k \ f(\bar{a}) = b \implies g(\bar{a}) = b$$

We write $f \simeq g$ to mean

$$f \sqsubseteq g \ \& \ g \sqsubseteq f$$

i.e., that f and g are identical partial functions.

We will reserve the word "function" to mean "total function;" then "partial function" means "partial or total function." However, we continue to use $\simeq$ to indicate identity of total functions.

We often discuss partial functions of a given arity and *co-arity*. A partial function f of arity k and co-arity ℓ on A is simply a partial function $f : A^k \rightharpoonup A^\ell$. We identify it with an ℓ-tuple $(f_1, \ldots, f_\ell)$ of partial functions of arity k on A.

1.1.1 Partial pointed structures

Our definition of structure differs in a few ways from the convention in model theory. As usual, a *signature* is a collection of function symbols with associated

sorts and arities. However our structures may be *partial*, i.e. the primitives may be partial functions, and they are not required to contain equality. They are also *pointed*, i.e., they are required to contain two distinct constants, 0 and 1.

At certain points we will abandon this last requirement. Assuming pointed structures simplifies the exposition considerably, but none of the results depend on it. Those few times it seems to be essential, we will indicate how to avoid it.

Definition 1.1.1. If Φ is a signature, then a Φ-*structure* $\mathbf{A}$ is a tuple

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

such that $\phi^{\mathbf{A}} : A^k \rightharpoonup A$ is a partial function where k is the arity of ϕ. In addition there are constants 0 and 1 such that $0^{\mathbf{A}}$ and $1^{\mathbf{A}}$ are distinct elements of A.

Following the usual convention, we write

$$\mathbf{A} \models \phi(\bar{x}) = w$$

for $\phi^{\mathbf{A}}(\bar{x}) = w$, where $w \in A$.

Unlike the case for functions, we will write *structure* to mean a partial or total structure. However, whenever we encounter an explicit partial structure, we will always recognize it as such. We just want to ensure that when we say, e.g., "let $\mathbf{A}$ be a Φ-structure," we allow $\mathbf{A}$ to be partial.

Predicates vs Functions In pointed partial structures, we identify *predicates* with functions into $\{0, 1\}$, with 0 expressing truth and 1 expressing falsity. This simplifies the development of the theory considerably and frees us from having to handle predicate symbols separately.

Now we make a series of basic definitions about different types of substruc-

tures and the maps between them.

Definition 1.1.2. For Φ-structures $\mathbf{A}$ and $\mathbf{B}$, $\mathbf{B}$ is a *substructure* of $\mathbf{A}$ in case $B \subseteq A$ and for any $\bar{x}, w \in B$,

$$\mathbf{B} \models \phi(\bar{x}) = w \implies \mathbf{A} \models \phi(\bar{x}) = w$$

In addition if

$$\mathbf{A} \models \phi(\bar{x}) = w \implies \mathbf{B} \models \phi(\bar{x}) = w$$

we say that $\mathbf{B}$ is an *induced substructure*.

Definition 1.1.3. For a Φ-structure $\mathbf{A}$ and a set $S \subseteq A$, the *substructure* $\mathbf{A} \restriction S$ *of* $\mathbf{A}$ *induced by* S is the unique structure $\mathbf{M}$ with domain S satisfying

$$\mathbf{M} \models \phi(\bar{x}) = w \iff \mathbf{A} \models \phi(\bar{x}) = w$$

for all $\bar{x}, w \in A$. In general $\mathbf{A} \restriction S$ may be partial even if $\mathbf{A}$ is total.

Definition 1.1.4. For a Φ-structure $\mathbf{A}$ and a set $S \subseteq A$, the *substructure* $\langle S \rangle_{\mathbf{A}}$ *of* $\mathbf{A}$ *generated by* S is the substructure of $\mathbf{A}$ induced by the set $S' \supseteq S$ of elements generated by S, i.e. the closure of S under the primitives of $\mathbf{A}$.

Definition 1.1.5. If $\mathbf{A}$ and $\mathbf{B}$ are two Φ structures, a Φ-*homomorphism* is a map $\pi : A \to B$ such that for all $\phi \in \Phi$ and $\bar{x} \in A$ and $w \in A_s$

$$\mathbf{A} \models \phi(\bar{x}) = w \implies \mathbf{B} \models \phi(\pi(\bar{x})) = \pi(w)$$

Notice that π fixes 0 and 1. In addition, if equality is among the primitives of $\mathbf{A}$ and $\mathbf{B}$, then π is necessarily injective, in which case we call π an *embedding*.

If in addition

$$\mathbf{A} \models \phi(\bar{x}) = w \iff \mathbf{B} \models \phi(\pi(\bar{x})) = \pi(w)$$

then we say π is a *strong homomorphism*.

If $\mathbf{A}$ is a Φ-structure and $f : A^k \rightharpoonup A_s$, then the *expansion of* $\mathbf{A}$ *by* f, $(\mathbf{A}, f)$, is the $\Phi \cup \{\phi_f\}$-structure where f interprets ϕ_f, a new function symbol of the same sort as f. The expansion of $\mathbf{A}$ by a set of functions is defined similarly.

A *reduct* of $\mathbf{A}$ is some $\mathbf{B}$ of which $\mathbf{A}$ is an expansion.

1.1.2 Syntax and Semantics of Recursive Programs

Recursive programs are the syntactic objects that define recursive functions. Recursive programs were first studied as objects in their own right in the 1960's and 70's in the field of Schematology under the heading of *recursive schemas*, see, e.g., [12, 5].

First we define terms, which are terms in first-order logic extended by function-valued variables and conditional statements.

Definition 1.1.6. A Φ-*term* M is generated by the grammar

$$M := x \mid \phi(M_1, \ldots, M_n) \mid p(M_1, \ldots, M_n) \mid \text{if } M_0 \text{ then } M_1 \text{ else } M_2$$

where x is a variable, p is a function-valued variable, and $\phi \in \Phi$.

Semantics of terms The semantics of a Φ-term M given a Φ-structure $\mathbf{A}$ is a functional that takes partial functions into partial functions.

A term without function variables is called *explicit* and denotes a single partial function.[1] A function over $\mathbf{A}$ denoted by an explicit term is itself called

[1] If the hypothesis M_0 of a conditional term M denotes 0, the denotation of M is the

explicit.

If M is a term with function variables $p_1, \ldots, p_k$, then M denotes a k-ary functional on partial functions by taking the tuple $(f_1, \ldots, f_k)$ to the explicit partial function denoted by M on the structure $(\mathbf{A}, f_1, \ldots, f_k)$ with p_i interpreting f_i.

Anatomy of a deterministic recursive program

Definition 1.1.7. A Φ-*recursive program* is a $(k+1)$-tuple of Φ-terms

$$E = (E_0, E_1, \ldots, E_k)$$

such that each function variable occurring in any term is p_i for some $1 \leq i \leq k$ and the arity of p_i is the number of variables in E_i.

We shall typically write such a program E as

$$E_0(\bar{x}_0, \bar{p}) \text{ where}$$

$$\{p_1(\bar{x}_1) = E_1(\bar{x}_1, \bar{p})$$

$$\vdots$$

$$p_k(\bar{x}_k) = E_k(\bar{x}_k, \bar{p})\}$$

where $\bar{p} = (p_1, \ldots, p_k)$ and $\bar{x}_i$ are the variables that occur in E_i. The term E_0 is called the *head* of E. The system of equations below the head is the *body* of E.

Definition 1.1.8. For a Φ-structure $\mathbf{A}$ and a Φ-recursive term $E = (E_0, \ldots, E_k)$,

denotation of M_1. If M_0 denotes anything else, the denotation of M is the denotation of M_2. If M_0 is undefined, so is M.

an $(\mathbf{A}, E)$-*term* is a term obtained by substituting all the variables in a Φ-subterm of some E_i with constants from A (the *parameters*).

There is a natural operational semantics for recursive programs which can be found in [10]. Briefly, one defines a sequential, two-stack machine implementation of recursive programs. Then $(\mathbf{A}, E)$-terms are the only terms that such a machine obtained from a recursive program E evaluates in the course of its computation over a structure $\mathbf{A}$.

Denotational semantics of recursive programs Recall that a Φ-term denotes a functional. A tuple $\bar{g} = (g_1, \ldots, g_k)$ of partial functions *solves* a system of equations $\{p_i(\bar{x}) \simeq E_i(\bar{x}, \bar{p})\}_{1 \leq i \leq k}$ in case g_i is the denotation of E_i applied to $\bar{g}$.

It is well known that there is always a *least* solution to a system of recursive equations, in the sense that any other solution extends the least solution as a tuple of partial functions (see [10]).

The denotation of a recursive program is the partial function f given by applying the head E_0 to the least solution $\bar{g}$ of the body of a recursive program. If $f(x) = w$, then we say

$$\mathbf{A} \vdash E(x) = w$$

and if $f(x) \uparrow$, we say

$$\mathbf{A} \vdash E(x) \uparrow$$

Definition 1.1.9. The set of all partial functions computed by Φ-recursive programs over $\mathbf{A}$ is $\mathrm{rec}(\mathbf{A})$.

Fundamental properties of recursive functions We now state three fundamental and standard facts about recursion. The first is that a function that is computable relative to a recursive function is itself recursive. The second is

that recursion is *local* in the sense that if f is a recursive partial function on $\mathbf{A}$ and $x \in A$, then if $f(x) = w$ for some w, w is contained in the substructure generated by x. The third states that recursive programs respect homomorphisms. All these results may be found in chapter 2 of [10].

Theorem 1.1.10. (Transitivity) *If $g \in rec(\mathbf{A})$ and $f \in rec(\mathbf{A}, g)$ then $f \in rec(\mathbf{A})$*

Proof. (Idea) If g is computed over $\mathbf{A}$ by E_0 where $\{p_i \simeq E_i\}_{1 \leq i \leq k}$ and f is computed over $\mathbf{A}$ by F_0 where $\{q_j \simeq F_j\}_{1 \leq j \leq \ell}$ then f is computed over $\mathbf{A}$ by E_0 where

$$\{p_i \simeq E_i\}_{1 \leq i \leq k} \cup \{q_j \simeq F_j\}_{1 \leq j \leq \ell}$$

□

Theorem 1.1.11. (Locality) *If $f \in rec(\mathbf{A})$ then for any $\bar{x}, w \in A$*

$$\mathbf{A} \vdash f(x) = w \implies \langle \bar{x} \rangle_{\mathbf{A}} \vdash f(x) = w$$

Theorem 1.1.12. (The homomorphism property) *For a homomorphism $\pi : \mathbf{A} \to \mathbf{B}$ of Φ-structures and any recursive Φ-program E,*

$$\mathbf{A} \models E(x_1, \ldots, x_n) = w \implies \mathbf{B} \models E(\pi(x_1), \ldots, \pi(x_n)) = \pi(w)$$

In addition if π is a strong homomorphism

$$\mathbf{A} \models E(x_1, \ldots, x_n) = w \iff \mathbf{B} \models E(\pi(x_1), \ldots, \pi(x_n)) = \pi(w)$$

1.2 Tail Recursion: Definition and Basic Properties

In this chapter we review the definition of and establish the crucial facts about tail recursion that we will use frequently throughout the rest of the paper. Recursive programs which are *tail recursive* express exactly the iterative algorithms over a given structure. The partial functions they compute form an extremely robust class, admitting several different equivalent definitions.

After giving our definition of a tail recursive program, we prove them computationally equivalent to *register machines*, which are more familiar objects used to characterize iterative algorithms. Then we show that the transitivity theorem (Theorem 1.1.10) specializes to tail recursion. Finally we discuss *linear recursion*, a syntactically more general type of recursion computationally equivalent to tail recursion.

Definition 1.2.1. A Φ-recursive program E is *tail recursive* in case there is only one recursive variable p, the body of E is of the form

$$p(\bar{x}) = \text{if } \tau(\bar{x}) \text{ then } o(\bar{x}) \text{ else } p(F(\bar{x}))$$

where τ and o are explicit Φ-terms of arity n, F is an explicit Φ-term of arity and co-arity $k = |\bar{x}|$, and the head E_0 has the form

$$f(x_1, \ldots, x_n) = p(G(x_1, \ldots, x_n))$$

for some $n \leq k$ and explicit Φ-term G of arity n and co-arity k.

A partial function $f : A^n \rightharpoonup A$ is *tail recursive* in case it is computable by some tail recursive program.

1.2.1 Tail Recursion and Register Machines

A *register program* is an assembly-language style GOTO-program with a finite control that can modify its registers and jump to different states based on atomic tests of its registers. We will use them to define *register machines*.

Register programs are equivalent computationally and very similar syntactically to the *program schemas* studied in schematology, see [5, 12].

Definition 1.2.2. For any signature Φ, a Φ-*register program* consists of a finite list of variables ("registers") $\bar{R} = (R_1, \ldots, R_k)$ and a finite list of lines $\langle \lambda_i : 1 \leq i \leq \ell \rangle$. Each line is of one of three forms:

1. Assignment:
$$\bar{R} \leftarrow T(\bar{R}) \tag{1.3}$$
where T is an explicit Φ-term of arity and co-arity k.

2. Branching:
$$\text{if } A_i(\bar{R}) \text{ then goto line } i_0 \text{ else goto line } i_1 \tag{1.4}$$
where A is an explicit Φ-term and $1 \leq i, j \leq \ell$.

3. Halt:
$$\text{halt and return } H_i(\bar{R})$$
where H is an explicit Φ-term.

Before defining register machines, we make a very general definition of machine, following [10].

Definition 1.2.3. A *sequential machine* $\mathfrak{i}$ is a tuple $(X, W, \text{in}, S, \tau, T, \text{out})$ where

1. X and W are sets, the *domain* and *range* respectively,

2. S is the set of *states*,

3. $\text{in} : X \to S$ is the *input function*,

4. $\tau : S \rightharpoonup S$ is the *transition function*,

5. $T \subseteq S$ is the set of *terminal states*. For $s \in T$, $\sigma(s) = s$,

6. $\text{out} : S \to W$ is the *output function*.

Semantics of a sequential machine A machine $\mathfrak{i}$ computes a partial function

$$f_{\mathfrak{i}} : X \rightharpoonup W$$

as follows. Given an element $x \in X$, consider the finite or infinite sequence of states

$$s_0 = \text{in}(x)$$

$$s_{i+1} = \tau(s_i) \text{ if } \tau(s_i) \downarrow$$

If for some i $s_i \in T$, then let $f(x) := \text{out}(s_i)$ for the least such i, otherwise we write $f(x) \uparrow$.

The register machine Given a Φ-structure $\mathbf{A}$, a Φ-register program P with k registers, and an explicit input function $g : A^n \to A^k$, we define the *register machine* $\mathfrak{i} = \mathfrak{i}_{(\mathbf{A},P,g)}$, which is a sequential machine.

The domain of $\mathfrak{i}$ is A^n, and the range is A. The set of states of $\mathfrak{i}$ is $\Lambda \times A^k$, where Λ is the set of lines in P and k is the number of registers in P. The subset of halt states consists simply of any states $(i, \bar{x})$ where i is a halting line of P.

The transition function τ maps $(i, x_1, \ldots, x_k) \mapsto (j, y_1, \ldots, y_k)$, where

1. If i is an assignment line of the form 1.3, $j = i + 1 \mod \ell$, and

$$\mathbf{A} \models \bar{y} = T_i(\bar{x})$$

2. If i is a branching line of the form 1.4, then $\bar{y} = \bar{x}$ and

$$j = \begin{cases} i_0 & \text{if } \mathbf{A} \models A_i(\bar{x}) = 0 \\ i_1 & \text{if } \mathbf{A} \models A_i(\bar{x}) = 1 \end{cases}$$

3. If i is a halt line, $i = j$ and $\bar{y} = \bar{x}$.

The input function in : $A^n \to \Lambda \times A^k$ is $\bar{a} \mapsto (1, g(\bar{a}))$. The first coordinate is 1 since computation starts at line 1 of P.

The output function takes a halt state $(i, \bar{x})$ into $h_i(\bar{x})$, where h is the explicit function over $\mathbf{A}$ denoted by H_i, which is the term in line i in the program P.

Now we prove the equivalence of tail recursion and register programs, which is a classical result in program schematology.

Theorem 1.2.4. *For any Φ-structure $\mathbf{A}$ and partial function $f : A^n \rightharpoonup A$, f is tail recursive iff it is computable by a register machine $\mathfrak{i}_{(\mathbf{A},P,g)}$ for some register program P and explicit function $g : A^n \to A^k$.*

Proof. Let

$$f(\bar{x}) = p(G(\bar{x})) \text{ where}$$

$$\{p(\bar{y}) = \text{if } \tau(\bar{y}) \text{ then } o(\bar{y}) \text{ else } p(F(\bar{y}))\}$$

be a tail recursion, where $|\bar{x}| = n$ and $|\bar{y}| = k$. Construct a register program P with k registers and lines

1. if $\tau(\bar{R})$ then goto line 2 else goto line 3
2. halt and output $o(\bar{R})$
3. $\bar{R} \leftarrow F(\bar{R})$

4. goto line 1[2]

Let g be the function computed by the term G. Then it's easy to see that the partial function computed by $\mathfrak{i}_{(\mathbf{A},P,g)}$ is exactly f.

Conversely, suppose that P is a register machine with k registers and ℓ lines, g is an explicit function $A^n \to A^k$, and $\mathfrak{i} = \mathfrak{i}_{(\mathbf{A},P,g)}$ computes the partial function f. We outline the definition of a tail recursive program computing f.

Let $m = \lceil \log_2 \ell \rceil$, and let $\alpha_i \in \{0,1\}^m$ be the m-digit binary expansion of i for $1 \leq i \leq \ell$. Let G be the explicit term computing g and consider the program

$$f'(\bar{x}) = p(1, G(\bar{x})) \text{ where}$$

$$p(\alpha_i, \bar{x}) = \text{if } i \text{ is a halt state then } O(\alpha_i, \bar{x}) \text{ else } p(F(\alpha_i, \bar{x})) \tag{1.5}$$

where

$$O(\alpha_i, \bar{x}) = H_i(\bar{x})$$

$$F(\alpha_i, \bar{x}) = \begin{cases} (\alpha_{i+1}, T_i(\bar{x})) & \text{if } i \text{ is an assignment} \\ (\alpha_{i_0}, \bar{x}) & \text{if } i \text{ is a branching statement and } A_i(\bar{x}) \\ (\alpha_{i_1}, \bar{x}) & \text{if } i \text{ is a branching statement and } \neg A_i(\bar{x}) \end{cases}$$

and G is a Φ-term computing g.

By definition, F computes the transition function τ and O computes the output function of $\mathfrak{i}$. Hence $p(\alpha_i, \bar{x})$ computes the output of the computation of $\mathfrak{i}$ starting in state $(i, \bar{x})$, and therefore $f' \simeq f$. □

1.2.1.1 Recursion and tail recursion on non-pointed structures

The proof of Theorem 1.2.4 relies in some essential way on the fact that $\mathbf{A}$ has constants for 0 and 1. We define tail recursion on a structure without these

[2]Such a line is easily constructed from a branching line

constants simply by adding them. That is, for non-pointed structures $\mathbf{A}$, we say that a partial function $f : A^k \rightharpoonup A$ is tail recursive iff it is computable in the structure $\mathbf{A} + \{0, 1\}$. Here $\mathbf{A} + S$ is defined to be the Φ-structure with domain $A \sqcup S$ such that the primitives of $\mathbf{A}$ fix S and there are primitives eq_s for each $s \in S$.

The $\mathbf{A}$-recursive functions do not change when we add the constants S. In other words,

Theorem. *For a finite set S, let $\mathbf{A} + S$ be the Φ-structure with domain $A \sqcup S$, the primitives of $\mathbf{A}$ undefined on S. Then, for any $\mathbf{A}$, if $f : A^n \rightharpoonup A$ is $(\mathbf{A}+S)$-recursive, it's already $\mathbf{A}$-recursive.*

(This routine result may be found in section 2A in [10]). Hence the question of recursion vs. tail recursion does not change when we pass from $\mathbf{A}$ to $\mathbf{A}+\{0, 1\}$.

1.2.1.2 Relative Tail Recursion

A partial function tail recursive relative to a tail recursive partial function (i.e. given such a function as a primitive) is itself tail recursive. This is the tail recursive analogue to Theorem 1.1.10, but justifies its own statement since the original construction does not preserve tail recursions. This result is lemma 3.32 of [15].

Theorem 1.2.5. *Suppose f is a partial function tail recursive over $\mathbf{A}$ and g is a partial function tail recursive over $(\mathbf{A}, f)$. Then g is tail recursive over $\mathbf{A}$.*

In fact, the program computing g over $\mathbf{A}$ is a function of the programs computing g over $(\mathbf{A}, f)$ and the function computing f over $\mathbf{A}$. More precisely, for any Φ-tail recursive program $E^\star$ and (Φ, f)-tail recursive program $E^\dagger$, there is a Φ-tail recursive program E such that for any Φ-structure $\mathbf{A}$, if f is the partial function computed by $E^\star$, then $E^\dagger$ and E compute identical partial functions.

1.2.2 Linear recursion

In this section we define *linear recursive programs*, which are a syntactically general class of programs which are nonetheless equivalent to tail recursion in terms of computability. The term "linear" comes from the fact that the size of the stack of the recursive machine grows at most linearly in the number of steps. The fact that linear recursions computable by tail recursions is stated in [12], but a more complete proof can be found in [5].

Linear recursions are not the most general family of programs known to be strongly equivalent to tail recursion, see [17]. However, linear recursions will be enough to carry out the programming in later chapters.

Definition 1.2.6. A recursive program $f(\bar{x}) = E_0(\bar{x})$ where $p(\bar{x}) = \{E_i(\bar{x}, \bar{p})\}_{1 \leq i \leq K}$ is *linear recursive* in case each E_i is of the form

$$\text{if } A_1 \text{ then } M_1 \text{ else if } A_2 \ \ldots \text{else } M_n \tag{1.6}$$

where each A_i is explicit and each M_i contains at most one occurrence of any recursive variable.

Suppose M is a term that contains at most one occurrence of any recursive variable p_j. Then M can be decomposed into $F(p_j(G(\bar{x})), \bar{x})$ where F and G are explicit terms (G may not be co-unary). In general, each equation in the body of a linear recursive program can be written as

$$p_i(\bar{x}) = \begin{cases} O_i(\bar{x}) & \text{if } \tau_i^0(\bar{x}) \\ F_i^1(p_1(G_i^1(\bar{x})), \bar{x}) & \text{if } \tau_i^1(\bar{x}) \\ \vdots & \vdots \\ F_i^k(p_k(G_i^k(\bar{x})), \bar{x}) & \text{if } \tau_i^k(\bar{x}) \end{cases} \tag{1.7}$$

where the O^i, F^i_j, and G^i_j are explicit.

Theorem 1.2.7. *A partial function is linear recursive iff it is tail recursive. In fact, for any tail recursive Φ-program E there's a linear recursive Φ-program E^L such that E and E^L compute the same partial function for any Φ-structure* **A**.

Proof. Let E be a linear recursive program with k recursive variables. For simplicity in notation, we assume that all the recursive variables $p_1, \ldots, p_k$ have the same arity. For $1 \leq i, j \leq k$, let O_i, F^j_i, G^j_i, and τ^j_i be defined as in (1.7).

First, we simplify the body of the equation. We use the standard trick of encoding a fixed finite set of numbers $1 \leq j \leq k$ by tuples of 0's and 1's, as in the proof of Theorem 1.2.4. For simplicity we just write a number, e.g. j, as a parameter with the understanding that it abbreviates such a tuple.

Let $\ell(i, \bar{x})$ be the unique j such that $\tau^j_i(\bar{x})$. Define the terms $O(i, \bar{x})$, $\tau(i, \bar{x})$ by

$$O(i, \bar{x}) = O_i(\bar{x}) \quad \tau(i, j, \bar{x}) = \tau^j_i(\bar{x})$$

Define $F(i, \bar{y}, \bar{x})$ and $G(i, \bar{x})$ by

$$F(i, \bar{y}, \bar{x}) = \begin{cases} \uparrow & \text{if } \tau^0_i(\bar{x}) \\ F^{\ell(i,\bar{x})}_i(\bar{y}, \bar{x}) & \text{otherwise} \end{cases} \qquad G(i, \bar{x}) = \begin{cases} \uparrow & \text{if } \tau^0_i(\bar{x}) \\ G^{\ell(i,\bar{x})}_i(\bar{x}) & \text{otherwise} \end{cases}$$

Then ℓ is an explicit function and O, τ, F, and G are explicit terms.

Then if we define $p(i, \bar{x}) := p_i(\bar{x})$, we see that

$$p(i, \bar{x}) = \begin{cases} O(i, \bar{x}) & \text{if } \tau(i, 0, \bar{x}) \\ F(i, p(\ell(i, \bar{x}), G(i, \bar{x})), \bar{x}) & \text{otherwise} \end{cases}$$

Clearly if we can compute p by a tail recursion we can compute each p_i by a

tail recursion. Then since the partial function computed by E is explicit in the p_i, we'd be done. Therefore, it suffices to be able to compute a linear recursive program of the form

$$p(\bar{x}) = \begin{cases} o(\bar{x}) & \text{if } \tau(\bar{x}) \\ F(p(G(\bar{x})), \bar{x}) & \text{otherwise} \end{cases}$$

by a tail recursion, where G has co-arity $|\bar{x}|$.

Let $n = n(\bar{x})$ be the least m such that $\tau(G^m(\bar{x}))$, if it exists. Then if we define $\bar{x}_m := G^m(\bar{x})$, we see that if there is such an n,

$$p(\bar{x}) = F(F(\ldots F(o(\bar{x}_n), \bar{x}_{n-1}) \ldots, \bar{x}_1), \bar{x}_0)$$

Define tail recursive functions

$$\gamma(\bar{x}) = \begin{cases} \bar{x} & \text{if } \tau(\bar{x}) \\ \gamma(G(\bar{x})) & \text{otherwise} \end{cases}$$

$$eq(\bar{x}, \bar{y}) = \begin{cases} \tau(\bar{y}) & \text{if } \tau(\bar{x}) \\ \tau(\bar{x}) & \text{if } \tau(\bar{y}) \\ eq(G(\bar{x}), G(\bar{y})) & \text{otherwise} \end{cases}$$

$$pd(\bar{x}, \bar{y}) = \begin{cases} \bar{x} & \text{if } eq(G(\bar{x}), \bar{y}) \\ pd(G(\bar{x}), \bar{y}) & \text{otherwise} \end{cases}$$

Then we have, for fixed $\bar{x}$ such that n exists,

$$\gamma(\bar{x}) = \bar{x}_n$$

$$eq(\bar{x}_m, \bar{x}_{m'}) \iff m = m'$$

$$pd(\bar{x}, \bar{x}_m) = \bar{x}_{m-1}$$

Therefore the tail recursive function P defined by

$$P(\bar{x}, \bar{y}, \bar{z}) = \begin{cases} \bar{y} & \text{if } eq(\bar{x}, \bar{z}) \\ P(\bar{x}, F(\bar{y}, \bar{z}), pd(\bar{x}, \bar{z})) & \text{otherwise} \end{cases}$$

satisfies

$$P(\bar{x}, \bar{y}, \bar{x}_m) = \begin{cases} \bar{y} & \text{if } m = 0 \\ P(\bar{x}, F(\bar{y}, \bar{x}_m), \bar{x}_{m-1}) & \text{otherwise} \end{cases}$$

It is straightforward to see that

$$p(\bar{x}) = P(\bar{x}, o(\gamma(\bar{x})), \gamma(\bar{x}))$$

and hence p is tail recursive.

Finally, note that this construction is completely independent of the particular Φ-structure $\mathbf{A}$. Hence we obtain a tail recursive program that computes the same partial function as E over any Φ-structure. $\square$

Chapter 2

Towards Main Question 1

In this part we outline a partial answer to Main Question 1. In section 2.2 we define family of problems which are complete for tail recursion over a certain class of "locally infinite" structures, which we shall define below.

This motivates us to focus on locally finite structures in the remainder of our paper. In section 2.3 we introduce Predecessor arithmetic, a paradigmatic locally finite structure that will be the basis for our work on locally finite algebraic structures in Part III.

However, before that, in section 2.1 we shall first introduce the machinery of *interpretations* and of *uniform recursion over families of structures*. We will use the language of this section throughout the rest of the paper, and our results in Part III will be specializations of the general results at the end of this section.

2.1 Interpretations and Uniform Families

First we use homomorphisms to define interpretations. In 2.1.1, we introduce the idea of uniform recursion over a family of structures. Then in 2.1.2, we introduce the main object of interest, a *uniform family of interpretations*, which

combines these ideas.

Recall by definition 1.1.5, if both $\mathbf{A}$ and $\mathbf{B}$ are Φ-structures, a strong homomorphism is map $\pi : A \to B$ such that

$$\mathbf{A} \models \phi(x_1, \ldots, x_n) = w \iff \mathbf{B} \models \phi(\pi(x_1), \ldots, \pi(x_n)) = \pi(w)$$

or in other words,

$$\pi \circ \phi^{\mathbf{A}} \simeq \phi^{\mathbf{B}} \circ \pi^n$$

where $\pi^n(a_1, \ldots, a_n) = (\pi(a_1), \ldots, \pi(a_n))$.

If π is a strong homomorphism, then the homomorphism property (theorem 1.1.12) states that for any recursive Φ-program E,

$$\mathbf{A} \models E(x_1, \ldots, x_n) = w \iff \mathbf{B} \models E(\pi(x_1), \ldots, \pi(x_n)) = \pi(w)$$

where n in the arity of E.

We use homomorphisms to define interpretations.

Definition 2.1.1. Suppose Φ and Ψ are signatures, and suppose $\mathbf{A}$ is a Φ-structure and $\mathbf{B}$ is a Ψ-structure. Then a *k-ary interpretation of $\mathbf{A}$ in $\mathbf{B}$* is a strong homomorphism of $\mathbf{A}$ into a Φ-structure $\mathbf{M}$ with domain B^k such that all the primitives of $\mathbf{M}$ are $\mathbf{B}$-tail recursive.

The terminology "k-ary" is perhaps bad; the map π associated with a k-ary interpretation of $\mathbf{A}$ in $\mathbf{B}$ is a map $\pi : A \to B^k$, which has co-arity, not arity, k. Still, calling it a "co-k-ary interpretation" seems excessively pedantic.

2.1.1 Families of Structures

We now define uniform recursion over a family of structures.

Definition 2.1.2. Let I be some index set. A *family of Φ-structures* is a collection $\{\mathbf{A}_i : i \in I\}$ of Φ-structures.

Definition 2.1.3. Let $\{\mathbf{A}_i : i \in I\}$ be a family of Φ-structures. Then $\{f_i : i \in I\}$ is a *family of partial functions* over $\{\mathbf{A}_i\}$ if for some k and all i, $f_i : A_i^k \rightharpoonup A_i$.

Definition 2.1.4. A family $\{f_i\}$ of partial functions over $\{\mathbf{A}_i\}$ is *uniformly $\{\mathbf{A}_i\}$-recursive* if there's a single recursive Φ-program E that computes f_i on $\mathbf{A}_i$ for each i.

Let $\mathrm{rec}(\{\mathbf{A}_i\})$ be the family of partial functions uniformly recursive (resp. uniformly tail recursive) over $\{\mathbf{A}_i\}$.

Define analogously *uniformly $\{\mathbf{A}_i\}$-tail recursive* and $\mathrm{tail}(\{\mathbf{A}_i\})$.

For the sake of readability, we will often write "f_i is uniformly recursive over $\mathbf{A}_i$."

The transitivity theorems 1.1.10 and 1.2.5 for relative recursion and tail recursion respectively extend to the uniform case.

Lemma 2.1.5. *Let $\{\mathbf{A}_i : i \in I\}$ be a family of Φ-structures and $\{f_i\}$, $\{g_i\}$ be families of functions over $\{\mathbf{A}_i\}$.*

If f_i is uniformly $\mathbf{A}_i$-recursive and g_i is uniformly $(\mathbf{A}_i, f_i)$-recursive then g_i is uniformly $\mathbf{A}_i$-recursive.

(Respectively for "tail recursive" instead of "recursive").

Proof. This is immediate from the observation that in either the recursive or tail recursive case, the program computing g_i over $\mathbf{A}_i$ for any i is a function of the program computing f_i over $\mathbf{A}_i$ and the program computing g_i over $(\mathbf{A}_i, f_i)$. Hence the program computing g_i is constant in i and we're done. □

2.1.2 Families of Interpretations

We now tackle the real object of interest: a uniform family of interpretations. This is simply a collection of interpretations between two families of structures $\mathbf{A}_i$ and $\mathbf{B}_i$ with the same index set such that the images of the primitives of $\mathbf{A}_i$ are uniformly tail recursive over $\mathbf{B}_i$.

Definition 2.1.6. Let $\{\mathbf{A}_i : i \in I\}$ be a family of Φ-structures and $\{\mathbf{B}_i : i \in I\}$ be a family of Ψ-structures. Then a k-ary *uniform family of interpretations from* $\mathbf{A}$ *to* $\mathbf{B}$ consists of a family of maps $\pi_i : A_i \to B_i^k$ such that for each $\phi \in \Phi$ there is a uniformly tail recursive family $e_i^\phi : B_i^{kn} \rightharpoonup B_i^k$, where n is the arity of ϕ, such that

$$\mathbf{A}_i \models \phi(x_1, \ldots, x_n) = w \iff \mathbf{B}_i \models e_i^\phi(\pi_i(x_1), \ldots, \pi_i(x_n)) = \pi_i(w)$$

for all i and $x_1, \ldots, x_n, w \in A_i$. More succinctly,

$$\pi_i \circ \phi^{\mathbf{A}_i} \simeq e_i^\phi \circ \pi_i^n$$

Before proving the fundamental result about uniform families of interpretations, we state a technical lemma without its tedious but straightforward proof.

Lemma 2.1.7. *If* $\mathbf{M}$ *is a* Φ*-structure with domain* X^k *and* $f : X^{kn} \rightharpoonup X^k$ *is* $\mathbf{M}$*-recursive, then* f *is* $\mathbf{M}'$*-recursive, where*

$$\mathbf{M}' = (X, \{\phi_i^{\mathbf{M}}\}_{i \in I, \phi \in \Phi})$$

where $\phi^{\mathbf{M}} = (\phi_1^{\mathbf{M}}, \ldots, \phi_k^{\mathbf{M}})$.

Moreover, the program computing each f_i is a function of the program computing $\mathbf{M}$, so this result extends to uniform recursion over families of structures.

This function from programs to programs preserves linear recursion, and so the result also holds for "tail recursive" and "uniformly tail recursive" in place of "recursive" and "uniformly recursive."

We now prove the fundamental result about uniform families of interpretations.

Theorem 2.1.8. *Let $\{\mathbf{A}_i : i \in I\}$ be a family of Φ-structures and $\{\mathbf{B}_i : i \in I\}$ be a family of Ψ-structures. Let $\{\pi_i, e_i^\phi : i \in I\ \phi \in \Phi\}$ be a uniform family of k-ary interpretations of $\mathbf{A}$ in $\mathbf{B}$.*

Then for each uniformly (tail) recursive family of partial functions $\{f_i : i \in I\}$ on $\{\mathbf{A}_i : i \in I\}$ there is a uniformly (tail) recursive family of partial functions $\{g_i : i \in I\}$ on $\{\mathbf{B}_i : i \in I\}$ such that for each i,

$$\pi_i \circ f_i \simeq g_i \circ \pi_i^n$$

where n is the arity of f_i.

Proof. Consider the family of Φ-structures $\{\mathbf{B}_i^\Phi : i \in I\}$, where

$$\mathbf{B}_i^\Phi = (B_i^k, \{e_i^\phi : \phi \in \Phi\})$$

Then π_i is a Φ-homomorphism from $\mathbf{A}_i$ to $\mathbf{B}_i^\Phi$.

Let E compute f_i on $\mathbf{A}_i$.Then by the homomorphism property 1.1.12, for each i and $x_1, \ldots, x_n, y \in A_i$,

$$\mathbf{A}_i \vdash E(x_1, \ldots, x_n) = y \iff \mathbf{B}_i^\Phi \vdash E(\pi_i(x_1), \ldots, \pi_i(x_n)) = \pi_i(y)$$

In other words,

$$\pi_i \circ f_i \simeq g_i \circ \pi_i^n$$

where g_i is the function computed by E on $\mathbf{B}_i^\Phi$.

It remains to show that g_i is uniformly recursive over $\mathbf{B}_i$. Since g_i is uniformly recursive over $\mathbf{B}_i^\Phi$, by Lemma 2.1.7 g_i is uniformly recursive over the family of structures

$$\mathbf{C}_i := (B_i, \{(e_i^\phi)_j : \phi \in \Phi,\ i \in I,\ 1 \leq j \leq k\})$$

where $(e_i^\phi)_j$ for $1 \leq j \leq k$ are the coordinates of e_i^ϕ. Hence g_i is uniformly recursive over the $\mathbf{C}_i$.

However, the primitives $(e_i^\phi)_j$ of $\mathbf{C}_i$ are each uniformly recursive over the structures $\mathbf{B}_i$. Hence by Lemma 2.1.5, the g_i are uniformly recursive over $\mathbf{B}_i$.

Lastly, this proof goes through replacing recursion by tail recursion throughout, since each e_i^ϕ is tail recursive. □

Taking $n = 0$ in the previous theorem gives us

Corollary. *For a uniform family of partial constants f_i over $\mathbf{A}_i$ there is a uniform family of partial constants g_i over $\mathbf{B}_i$ such that*

$$\pi_i(f_i) = g_i$$

2.1.3 Families of Bi-interpretations

In most of our applications of a uniform family of interpretations, we want interpretations in both directions. Hence we define:

Definition 2.1.9. A *uniform family of bi-interpretations* of $\{\mathbf{A}_i\}$ with $\{\mathbf{B}_i\}$ is a family of bijections $\pi_i : A_i \to B_i$ such that for each $\phi \in \Phi$ there is a uniformly tail recursive family e_i^ϕ of partial functions on $\mathbf{B}_i$ such that

$$\pi_i \circ \phi^{\mathbf{A}_i} \simeq e_i^\phi \circ \pi_i^n$$

and for each $\psi \in \Psi$ there is a uniformly tail recursive family e_i^ψ of partial functions on $\mathbf{A}_i$ such that

$$\psi^{\mathbf{B}_i} \circ \pi_i^m \simeq \pi_i \circ e_i^\psi$$

for each i, where n is the arity of ϕ and m is the arity of ψ.

Notice that a uniform family of bi-interpretations is always *unary*. It might be possible to develop a suitable notion of a uniform family of bi-interpretations with nontrivial arity, but we do not need it for our applications. (On the other hand, we *do* need non-unary uniform families of interpretations in section 2.3.)

A uniform family of bi-interpretations does actually consist of two uniform families of interpretations whose maps are inverse:

Lemma 2.1.10. *If $\{\pi_i, e_i^\phi, e_i^\psi\}$ is a uniform family of bi-interpretations of $\{\mathbf{A}_i\}$ with $\{\mathbf{B}_i\}$, then $\{\pi_i, e_i^\phi\}$ is a uniform family of interpretations from $\{\mathbf{A}_i\}$ to $\{\mathbf{B}_i\}$ and $\{\pi_i^{-1}, e_i^\psi\}$ is a uniform family of interpretations from $\{\mathbf{B}_i\}$ to $\{\mathbf{A}_i\}$.*

Proof. To show the first claim, for each $\phi \in \Phi$ and $i \in I$ we must prove

$$\pi_i \circ \phi^{\mathbf{A}_i} \simeq e_i^\phi \circ \pi_i^n$$

where n is the arity of ϕ. This is guaranteed by definition.

To show the second claim, we must prove for each $\psi \in \Psi$ and $i \in I$ that

$$\pi_i^{-1} \circ \psi^{\mathbf{B}_i} \simeq e_i^\psi \circ \pi_i^m$$

where m is the arity of ψ. This is obtained from

$$\psi^{\mathbf{B}_i} \circ \pi_i^m \simeq \pi_i \circ e_i^\psi$$

by applying π_i^{-1} to the left hand side and π_i^{-m} to the right. $\square$

Having a uniform family of bi-interpretations between two families of structures establishes a very strong connection between the recursion theories of these two families.

Theorem 2.1.11. *Suppose $\{\pi_i, e_i^\phi, e_i^\psi\}$ is a uniform family of bi-interpretations of $\{\mathbf{A}_i\}$ with $\{\mathbf{B}_i\}$. Then for every family of partial functions f_i on $\mathbf{A}_i$, f_i is uniformly recursive (tail recursive) if and only if $\pi_i \circ f_i \circ \pi_i^{-n}$ is uniformly recursive (tail recursive) over $\mathbf{B}_i$, where n is the arity of f_i.*

Proof. By Lemma 2.1.10, $\{\pi_i, e_i^\phi\}$ is a uniform family of interpretations from $\{\mathbf{A}_i\}$ to $\{\mathbf{B}_i\}$ and $\{\pi_i^{-1}, e_i^\psi\}$ is a uniform family of interpretations from $\{\mathbf{B}_i\}$ to $\{\mathbf{A}_i\}$.

Therefore, by Theorem 2.1.8, for each family of partial functions f_i uniformly recursive over the $\mathbf{A}_i$, there is a uniformly recursive family g_i of partial functions over $\mathbf{B}_i$ such that

$$\pi_i \circ f_i \simeq g_i \circ \pi_i^n$$

where n is the arity of the f_i. Since $g_i \simeq \pi_i \circ f_i \circ \pi_i^{-n}$, this proves the forward direction.

Now applying Theorem 2.1.8 to the g_i, we obtain a family h_i recursive over $\mathbf{A}_i$ such that

$$\pi_i^{-1} \circ g_i \simeq h_i \circ \pi_i^{-n}$$

so that

$$g_i \circ \pi_i^n \simeq \pi_i \circ h_i$$

But then, $f_i \simeq h_i$, so this proves the backwards direction.

The proof goes through identically for tail recursion. $\square$

Corollary 2.1.12. *There is a uniformly recursive not tail recursive family of partial functions on $\mathbf{A}_i$ iff there is a uniformly recursive not tail recursive family of functions on $\mathbf{B}_i$. Similarly for "constants" instead of "functions."*

2.1.4 Families of Interpretations on Substructures

Lastly, in most of our applications our families of structures $\{\mathbf{A}_i\}$ and $\{\mathbf{B}_i\}$ will be families of finitely generated substructures of $\mathbf{A}$ and $\mathbf{B}$ respectively. Suppose that we have a single Φ-structure $\mathbf{A}$ and Ψ-structure $\mathbf{B}$ and a map $\rho : A^k \to B^\ell$. For $\bar{a} \in A^k$, let $\mathbf{A}_{\bar{a}}$ be the $\Phi \cup \{\mathbf{a}_1, \ldots, \mathbf{a}_k\}$-structure with domain the set of elements generated by $\bar{a}$ and a constant for each element of $\bar{a}$. In other words,

$$\mathbf{A}_{\bar{a}} := (\langle \bar{a} \rangle_{\mathbf{A}}, \bar{a})$$

Similarly define

$$\mathbf{B}_{\bar{b}} := (\langle \bar{b} \rangle_{\mathbf{B}}, \bar{b})$$

for $\bar{b} \in B^\ell$.

Suppose that for some $\rho : A^k \to B^\ell$ we have a uniform family of bi-interpretations between the families of structures

$$\{\mathbf{A}_{\bar{a}}\}_{\bar{a} \in A^k} \quad \{\mathbf{B}_{\rho(\bar{a})}\}_{\bar{a} \in A^k}$$

Let $f : A^k \rightharpoonup A$ be a partial function. Then if $c_{\bar{a}} := f(\bar{a})$, f is $\mathbf{A}$-recursive iff $c_{\bar{a}}$ is a uniformly recursive family of partial constants over the structures $\mathbf{A}_{\bar{a}}$. By Theorem 2.1.11, $c_{\bar{a}}$ is a uniformly recursive family of partial constants over $\mathbf{A}_{\bar{a}}$ iff $d_{\bar{a}} := \pi_{\bar{a}}(c_{\bar{a}})$ is a uniformly recursive family of partial constants over the structures $\mathbf{B}_{\rho(\bar{a})}$. But this is true exactly when there's a $\mathbf{B}$-recursive partial

function $g : B^\ell \rightharpoonup B$ such that for every $\bar{a} \in A^k$,

$$g(\rho(\bar{a})) = d_{\bar{a}}$$

Moreover, exactly the same argument goes through with tail recursion replacing recursion throughout.

Hence we have shown

Theorem 2.1.13. *Suppose there is a uniform family of bi-interpretations between $\{\mathbf{A}_{\bar{a}}\}$ and $\{\mathbf{B}_{\rho(\bar{a})}\}$ for $\bar{a} \in A^k$. Then for any partial function $f : A^k \rightharpoonup A$, f is recursive (tail recursive) iff there is a recursive (tail recursive) partial function $g : B^\ell \rightharpoonup B$ satisfying*

$$\pi_{\bar{a}}(f(\bar{a})) = g(\rho(\bar{a}))$$

for all $\bar{a} \in A^k$.

Stated in terms of recursive subsets:

Corollary 2.1.14. *A subset $X \subseteq A^k$ is recursive iff there's a recursive subset $Y \subseteq B^\ell$ such that $\bar{a} \in X \iff \rho(\bar{a}) \in Y$, i.e.*

$$\rho[X] = Y \cap \rho[A^k]$$

2.2 Dividing lines

In this section, we introduce a property which is equivalent to Main Question 1 for a general class of structures and motivates the work in the remainder of this paper.

The paradigmatic example of a structure in which tail recursion is as powerful as recursion is arithmetic: if $\mathbf{N}$ is a (possibly partial) expansion of $(\mathbb{N}, 0, S, =)$

then $\mathrm{rec}(\mathbf{N}) = \mathrm{tail}(\mathbf{N})$. In fact, the proof shows that

Lemma 2.2.1. *If* $\{\mathbf{N}_i : i \in I\}$ *is a* family *of expansions of* $(\mathbb{N}, 0, S, =)$ *then any family of partial functions* f_i *uniformly recursive over* $\mathbf{N}_i$ *is uniformly tail recursive.*

It is natural to ask if there are structures which are not expansions of $(\mathbb{N}, 0, S, =)$ but are still similar enough for the proof of equivalence to go through. Towards that end, we define:

Definition 2.2.2. A sequence of partial functions $\{e_k : k \in \mathbb{N}\}$ over a structure $\mathbf{A}$ is said to *enumerate finitely generated substructures* in case

- $e_k : A^{k+1} \rightharpoonup A$
- For every tuple $\bar{a} = (a_0, \ldots, a_{k-1}) \in A^k$ the orbit of a_0 under $e_k(\cdot, \bar{a})$ is the domain of $\langle \bar{a} \rangle_{\mathbf{A}}$

A structure $\mathbf{A}$ is said to satisfy *tail recursive enumerability of finitely generated substructures* (TRE) in case there exists such a sequence $\{e_n\}$ of functions which are all tail recursive.

For a structure to satisfy TRE means, intuitively, that given an element, we can exhaustively search all other elements generated by that element with a tail recursive program. One consequence of this is that a total function is recursive iff its graph relation is. Indeed, suppose that $G^f(x, y)$ were recursive. Then $f(x)$ could be computed by $p(x, x)$ where

$$p(x, v) = \text{if } G^f(x, v) \text{ then } v \text{ else } G^f(x, e_1(v, x)) \tag{2.1}$$

What bearing does TRE have on Main Question 1? The following theorem of Urzyczyn (Theorem 2.2 in [16]) shows that for total structures with equality,

the failure of TRE implies that recursion is strictly more powerful than tail recursion.

Theorem 2.2.3. (Urzyczyn) *For any total structure* $\mathbf{A}$ *with equality as a primitive, there is a sequence of* $\mathbf{A}$*-recursive functions which enumerate finitely generated substructures.*

And hence we have:

Corollary. *For any total structure* $\mathbf{A}$ *with equality as a primitive,*

$$\mathbf{A} \models \neg\text{TRE} \implies \mathit{tail}(\mathbf{A}) \subsetneq \mathit{rec}(\mathbf{A})$$

Does the success of TRE, conversely, imply that tail recursion is as powerful as recursion? In the remainder of section 2.2, we show that the answer is yes under some "non-local finiteness" assumptions. (We say that a structure is *locally finite* if every finitely generated substructure is finite.) Therefore, for such structures, the question of recursion vs. tail recursion reduces to TRE. This motivates us to examine the question of tail recursion vs. recursion in locally finite structures in the remainder of this paper.

So in some sense, we have an answer to Main Question 1 for non-locally finite structures. This answer is subject to the criticism that TRE is not really a model-theoretic property–it asserts that a certain family of functions are tail recursive. What would be ideal would be a reduction of TRE to a property which makes no reference to the recursion theory of a structure.

Theorem 2.2.4. *Suppose every finitely generated substructure of* $\mathbf{A}$ *is infinite and* $\mathbf{A}$ *contains equality as a primitive. Then*

$$\mathbf{A} \models \text{TRE} \iff \mathit{tail}(\mathbf{A}) = \mathit{rec}(\mathbf{A})$$

More specifically, if e_k is an $\mathbf{A}$-tail recursive partial function, then any $\mathbf{A}$-recursive partial function of arity k is $\mathbf{A}$-tail recursive.

Proof. It remains to prove the $\Longrightarrow$ direction. Fix k. Let e_k be a tail recursive function that enumerates substructures in $\mathbf{A}$ generated by k-tuples. Then for each $\bar{a} = (a_0, \ldots, a_{k-1})$ in A^k, define a bijection $\pi_{\bar{a}} : \mathbb{N} \to \langle \bar{a} \rangle_{\mathbf{A}}$ by

$$\pi_{\bar{a}} : 0 \mapsto a_0$$

$$\pi_{\bar{a}} : n+1 \mapsto e_k(\pi_{\bar{a}}(n), \bar{a})$$

In other words,

$$\pi_{\bar{a}} \circ S \simeq (e_k)_{\bar{a}} \circ \pi_{\bar{a}}$$

where $(e_k)_{\bar{a}}$ is $e_k(\cdot, \bar{a})$.

For each $\bar{a} \in A^k$ and $\phi \in \Phi$, define $f^{\phi}_{\bar{a}} : \mathbb{N}^n \rightharpoonup \mathbb{N}$ by

$$\pi_{\bar{a}} \circ f^{\phi}_{\bar{a}} \simeq \phi^{\mathbf{A}} \circ \pi^n_{\bar{a}}$$

where n is the arity of ϕ.

Define the structure $\mathbf{N}_{\bar{a}}$ to be the expansion of $(\mathbb{N}, 0, S, =)$ by the partial functions $\{f^{\phi}_{\bar{a}} : \phi \in \Phi\}$ and the constants $\{\pi^{-1}_{\bar{a}}(a_i) : 0 \leq i < k\}$. Then there is an extremely simple uniform family of bi-interpretations of the family of structures $\{\mathbf{N}_{\bar{a}} : \bar{a} \in A^k\}$ with the family $\{\mathbf{A}_{\bar{a}} : \bar{a} \in A^k\}$, where $\mathbf{A}_{\bar{a}} := (\langle \bar{a} \rangle_{\mathbf{A}}, \bar{a})$.

The images of the primitives $f^{\phi}_{\bar{a}}$ of $\mathbf{N}_{\bar{a}}$ are of course the primitives ϕ of $\mathbf{A}_{\bar{a}}$. The image of 0 is a_0, the image of S is the $e_k(\cdot, \bar{a})$, and the image of equality is equality. The image of the constant $\pi^{-1}_{\bar{a}}(a_i)$ is of course a_i.

The pre-images of the primitives ϕ of $\mathbf{A}_{\bar{a}}$ are likewise the primitves $f^{\phi}_{\bar{a}}$ of $\mathbf{N}_{\bar{a}}$, and the pre-image of the constant a_i is obviously $\pi^{-1}_{\bar{a}}(a_i)$.

Hence by Corollary 2.1.12, there is a uniformly recursive non-tail recursive

family of partial constants on the $\mathbf{A}_{\bar{a}}$ iff there is such a family on the $\mathbf{N}_{\bar{a}}$. However, by Lemma 2.2.1, this is false.

Finally, there is a uniformly recursive non-tail recursive family of partial constants on the $\mathbf{A}_{\bar{a}}$ iff there is a recursive non-tail recursive k-ary function on $\mathbf{A}$. When this holds for each k, we conclude that $\mathrm{rec}(\mathbf{A}) = \mathrm{tail}(\mathbf{A})$. □

So to summarize, for a total structure $\mathbf{A}$ with equality in which every finitely generated substructure is infinite

$$\mathbf{A} \models \mathrm{TRE} \iff \mathrm{rec}(\mathbf{A}) = \mathrm{tail}(\mathbf{A}) \tag{2.2}$$

The assumptions of totality and testable equality are rather mild and cover most structures of interest. However, the assumption that any finitely generated substructure is infinite is strong, excluding many interesting examples of structures, especially *locally finite* ones in which no finitely generated substructure is infinite.

This motivates us to examine Main Question 1 in the context of locally finite structures that satisfy TRE for two reasons. One is that it seems like the simplest case excluded by the hypotheses of theorem 2.2.4 and corollary 2.2. Any non-locally finite structure has a *locally finite part*, the union of all its finite finitely generated substructures. We could say that TRE is the property which decides Main Question 1 for structures with a trivial locally finite part.

Secondly, it is a reasonable thing to allow arbitrary constants in our definition of recursive programs. (This is the usual case in model theory, where definability usually means definability with parameters). In this case, it turns out that equation (2.2) extends to *all* non-locally finite total structures with equality, which we now prove.

In this setting, we have that a partial function f is recursive over $\mathbf{A}$ iff it's

in $\mathrm{rec}(\mathbf{A}, \bar{c})$ for some tuple $\bar{c}$. Say that

$$f \in \mathrm{rec}^\star(\mathbf{A}) \iff \exists k, \bar{c} \in A^k \ f \in \mathrm{rec}(\mathbf{A}, \bar{c})$$

$$f \in \mathrm{tail}^\star(\mathbf{A}) \iff \exists k, \bar{c} \in A^k \ f \in \mathrm{tail}(\mathbf{A}, \bar{c})$$

Now say that $\mathbf{A} \models \mathrm{TRE}^\star$ in case there's a sequence of functions $\{e_n\}$ that enumerates finitely generated substructures of $\mathbf{A}$ such that $e_n \in \mathrm{tail}^\star(\mathbf{A})$ for each n.

Corollary. *Suppose* $\mathbf{A}$ *is total, not locally finite, and contains equality as a primitive. Then* $\mathbf{A} \models \mathrm{TRE}^\star \iff rec^\star(\mathbf{A}) = tail^\star(\mathbf{A})$

Proof. If $\mathbf{A}$ is not locally finite, then for some tuple $\bar{a}$, every finitely generated substructure of $(\mathbf{A}, \bar{a})$ is infinite.

Suppose that $\mathbf{A} \models \mathrm{TRE}^\star$ and that $f \in \mathrm{rec}^\star(\mathbf{A})$. Then for some $\bar{c}$, $f \in \mathrm{rec}(\mathbf{A}, \bar{c})$. Similarly, for some $\bar{d}$, $e_n \in \mathrm{tail}(\mathbf{A}, \bar{d})$ where n is the arity of f, by $\mathrm{TRE}^\star$. Therefore,

- $f \in \mathrm{rec}(\mathbf{A}, \bar{a}, \bar{c}, \bar{d})$
- $e_n \in \mathrm{tail}(\mathbf{A}, \bar{a}, \bar{c}, \bar{d})$
- The structure $(\mathbf{A}, \bar{a}, \bar{c}, \bar{d})$ has equality as a primitive and every finitely generated structure is infinite.

Hence by theorem 2.2.4, $f \in \mathrm{rec}(\mathbf{A}, \bar{a}, \bar{c}, \bar{d})$ and thus $f \in \mathrm{rec}^\star(\mathbf{A})$.

Conversely, suppose that $\mathrm{rec}^\star(\mathbf{A}) = \mathrm{tail}^\star(\mathbf{A})$. Then for each n, $e_n \in \mathrm{tail}^\star(\mathbf{A})$, and so $\mathbf{A} \models \mathrm{TRE}^\star$. □

We now examine Main Question 1 in the context of locally finite structures that satisfy TRE.

2.3 Predecessor Arithmetic

We may obtain a very natural locally finite structure by taking unary arithmetic $\mathbf{N}_u$ and removing the successor function to obtain so-called "Predecessor arithmetic"

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

It is easy to see that $\mathbf{N}_{Pd}$ satisfies TRE.

It turns out that the recursion theory of this rather innocuous-looking structure has some interesting properties. More importantly, $\mathbf{N}_{Pd}$ seems to be fundamental, recursion-theoretically, among locally finite structures. The results we collect in this section underlie our work in Chapter 3 on the recursion theory of locally finite algebraic structures.

The outline of this section as is follows.

1. In 2.3.1 we prove that a predicate is computable in $\mathbf{N}_{Pd}$ (resp. by a tail recursion) iff it's computable in $\mathbf{N}_u$ in polynomial parameters (resp. by a tail recursion), which we call the *Main Property of* $\mathbf{N}_{Pd}$.

2. In 2.3.2 we prove that a subset of numbers is computable in $\mathbf{N}_{Pd}$ iff the set of its binary expansions (a subset of $\{0,1\}^\star$) is computable in exponential time, and computable by a tail recursion iff it's computable in linear space. We also show that over $\mathbf{N}_{Pd}$, all semirecursive sets are recursive.

3. In 2.3.3, we conclude that the question of recursion vs. tail recursion in $\mathbf{N}_{Pd}$ is equivalent to the question of whether all subsets of binary strings decidable in exponential time are already decidable in linear space.

2.3.1 $\mathbf{N}_{Pd}$-computation vs $\mathbf{N}_u$-computation in polynomial parameters

Definition 2.3.1. For $x \in \mathbb{N}$, $\mathbf{N}_u \restriction x$ abbreviates $\mathbf{N}_u \restriction \{0, 1, \ldots, x-1\}$.

Definition 2.3.2. Let E be a $\mathbf{N}_u$-program of arity n. For a function $\alpha : \mathbb{N}^n \to \mathbb{N}$, *$E$ runs in parameters α* if for every $\bar{x} \in \mathbb{N}^n$,

$$\mathbf{N}_u \vdash E(\bar{x}) = y \implies \mathbf{N}_u \restriction \alpha(\bar{x}) \vdash E(\bar{x}) = y$$

If α is a polynomial, we say *E runs in polynomial parameters.*

In this section we show that any predicate decidable in polynomial parameters over $\mathbf{N}_u$ is already decidable in $\mathbf{N}_{Pd}$, as a corollary of the more general version of this theorem stated for functions. The idea is relatively simple: if we have a $\mathbf{N}_u$-recursive program which for simplicity is unary, and uses parameters no bigger than n^k on input n, then replace each variable in the program by a k-tuple of variables representing the digits in the $(n+1)$-ary expansion of the original variable. Since the largest parameter appearing is n^k, having k digits in base $(n+1)$ gives us ample room to carry out these computations. Moreover, the largest digit which appears in the computation is our input n, and it's not hard to show that simulating the arithmetic of the original program can be carried out in $\mathbf{N}_{Pd}$ in the base $(n+1)$-representation. The proof uses the machinery of uniform families of interpretations.

2.3.1.1 A Uniform Family of Interpretations

Fix $k, n \in \mathbb{N}$. Consider the family of $\{0, 1, S, Pd, \mathbf{a}_0, \ldots, \mathbf{a}_{n-1}\}$-structures

$$\{\mathbf{N}_u(\bar{a}) \restriction m^k : \bar{a} \in \mathbb{N}^n \ \& \ m = \max(\bar{a})\}$$

where a_i interprets $\mathbf{a}_i$. Let m implicitly be $\max(\bar{a})$. Define

$$\mathbf{A}_{\bar{a}} := \mathbf{N}_u(\bar{a}) \restriction m^k$$

and

$$\mathbf{B}_{\bar{a}} := \mathbf{N}_{Pd}(\bar{a}) \restriction m$$

These are $\{0, 1, S, Pd, \bar{\mathbf{a}}\}$- and $\{0, 1, Pd, \bar{\mathbf{a}}\}$-structures respectively. Moreover, $\mathbf{A}_{\bar{a}}$ is a partial structure, with S being undefined on m.

Every $n \in A_m$ has a unique m-ary expansion $\sum_{i<k} n_i m^i$ of k digits (with some leading digits possibly 0). Define $\pi_{\bar{a}} : A_{\bar{a}} \to B_{\bar{a}}^k$ by

$$\pi_{\bar{a}} : n \mapsto (n_0, n_1, \ldots, n_{k-1})$$

Now we define the functions $e_{\bar{a}}^{\phi}$ for $\phi \in \{0, 1, S, Pd, \bar{\mathbf{a}}\}$ satisfying $\pi_{\bar{a}} \circ \phi^{\mathbf{A}_i} \simeq e_{\bar{a}}^{\phi} \circ \pi_{\bar{a}}^n$. We first note that $<$, the constant m, and the "truncated successor" $n \mapsto \max\{n+1, m\}$ are all uniformly computable over $\mathbf{B}_{\bar{a}}$ by tail recursions, and so may freely be used in the computations below. In fact, all of the functions $e_{\bar{a}}^{\phi}$ will be uniformly *explicit* in these three functions.

The functions $e_{\bar{a}}^0$ and $e_{\bar{a}}^1$ These are simply the constants $(0, 0, \ldots, 0)$ and $(0, 0, \ldots, 1)$ respectively.

The function $e_{\bar{a}}^{\mathbf{a}_i}$ for $1 \leq i \leq \ell$ This is simply the constant $(0, 0, \ldots, \mathbf{a}_i)$.

The function $e_{\bar{a}}^{Pd}$ Observe that for $n \in A_{\bar{a}}$, if $\pi_{\bar{a}}(n) = (n_0, \ldots, n_{k-1})$ and $\pi(Pd(n)) = (n'_0, \ldots, n'_{k-1})$, then

$$n'_i = \begin{cases} n_i & \text{if } \exists j < i \; n_j > 0 \\ m & \text{otherwise, if } n_i = 0 \\ Pd(n_i) & \text{otherwise} \end{cases}$$

Hence $e_{\bar{a}}^{Pd}$ is uniformly computable over $\mathbf{B}_{\bar{a}}^k$.

The function $e_{\bar{a}}^{S}$ Observe that for $n \in A_{\bar{a}}$ not equal to $m^k - 1$, if $\pi_{\bar{a}}(n) = (n_0, \ldots, n_{k-1})$ and $\pi_{\bar{a}}(S(n)) = (n'_0, \ldots, n'_{k-1})$, then

$$n'_i = \begin{cases} n_i & \text{if } \exists j < i \; n_j < m \\ 0 & \text{otherwise, if } n_i = m \\ n_i + 1 & \text{if } n_i < m \end{cases}$$

Hence $e_{\bar{a}}^{S}$ is uniformly computable over $\mathbf{B}_{\bar{a}}^k$. Here the assumption that $n \leq m^k$ is important: if $\pi(n) = (m, \ldots, m)$, then $(n'_0, \ldots, n'_{k-1}) = (0, \ldots, 0)$. Of course in this case $S(n)$ is undefined.

2.3.1.2 Main Result

Now we can prove

Theorem 2.3.3. *Let $m = m(x_1, \ldots, x_n)$ be the maximum of $\{x_1, \ldots, x_n\}$. A function $f(x_1, \ldots, x_n)$ is computable over $\mathbf{N}_u$ in parameters m^k (by a tail recursion) if and only if there are (tail) recursive functions $f_0, \ldots, f_{k-1}$ over $\mathbf{N}_{Pd}$ such that*

$$f(\bar{x}) = \sum_{0 \leq i < k} m^i f_i(\bar{x})$$

Proof. Assume f is computable over $\mathbf{N}_u$ in parameters m^k. Then $\{f_{\bar{x}}\}$ is a uniformly recursive family of constants over finite structures $\{\mathbf{N}_u(\bar{x}) \restriction m^k\}$, where $f_{\bar{x}} = f(\bar{x})$, and hence there is a uniformly recursive family $\{g_{\bar{x}} = (g_{\bar{x}}^{(0)}, \ldots, g_{\bar{x}}^{(k-1)})\}$ of constants over $\{(\mathbf{N}_{Pd}(\bar{x}) \restriction m)^k\}$ such that

$$\pi_{\bar{x}}(f_{\bar{x}}) = g_{\bar{x}}$$

Let f_i be defined by $f_i(\bar{x}) = g_{\bar{x}}^{(i)}$. We see that f_i is recursive over $\mathbf{N}_{Pd}$ and that

$$\pi_{\bar{x}}(f(\bar{x})) = (f_0(\bar{x}), \ldots, f_{k-1}(\bar{x}))$$

Since $\pi_{\bar{x}}(n)$ is the base-m expansion of n, we see that

$$f(\bar{x}) = \sum_{0 \leq i < k} m^i f_i(\bar{x})$$

As for the backwards direction, if each f_i is computable in $\mathbf{N}_{Pd}$, then it's computable over $\mathbf{N}_u$ in parameters m, so f is clearly computable in parameters m^k. □

Since predicates can be identified with functions into $\{0, 1\}$ for which $f = f_0$, we conclude:

Corollary 2.3.4. (Main Property of $\mathbf{N}_{Pd}$) *A predicate is computable (by a tail recursion) over unary arithmetic in polynomial parameters if and only if it is computable (by a tail recursion) over* $\mathbf{N}_{Pd}$

Relative Computation in Polynomial Parameters

If f is computed in polynomial parameters over $\mathbf{N}_u$ and g is computed in polynomial parameters over $(\mathbf{N}_u, f)$, then g is computable over polynomial parameters in $\mathbf{N}_u$ (respectively, by a tail recursion). This ultimately follows from

the fact that polynomials are closed under composition.

This implies

Corollary 2.3.5. *The precomposition $P(f_1(\bar{x}), \ldots, f_n(\bar{x}))$ of a predicate computable over $\mathbf{N}_{Pd}$ by functions computable in polynomial parameters over unary arithmetic is a predicate computable over $\mathbf{N}_{Pd}$.*

Similarly for tail recursion instead of recursion.

Stated in terms of sets, if $X \subseteq \mathbb{N}^n$ is $\mathbf{N}_{Pd}$-recursive and $f : \mathbb{N}^m \to \mathbb{N}^n$ is computable over $\mathbf{N}_u$ in polynomial parameters, then $f^{-1}[X]$ is $\mathbf{N}_{Pd}$-recursive.

2.3.2 Turing Machines and $\mathbf{N}_{Pd}$

In this section we relate $\mathbf{N}_{Pd}$-recursion and tail recursion to complexity classes.

We assume that the reader is somewhat familiar with Turing machines. In this section we summarize the definition of Turing machines, highlighting the specific features we shall assume they all have.

A *Turing machine* consists of a single read-only input tape, k work tapes, a set Q of states containing accept and reject states, and a transition function δ. We work with Turing machines over the binary alphabet $\{0, 1\}$. We assume the input tape contains two additional symbols $\{\triangleright, \triangleleft\}$ that delimit the ends of the input.

The transition function then becomes a map

$$\delta : Q \times \{\triangleleft, 0, 1, \triangleright\} \times \{0,1\}^k \to Q \times \{0,1\}^k \times \{-, L, R\}^{k+1}$$

That is to say, the transition function takes as input the state and the character underneath each head. Then it writes 0 or 1 on each of the k work tapes, transitions to another state, and on each of the $k+1$ tapes, it moves left (L), right (R), or stays put ($-$).

On input $w \in \{0,1\}^\star$, the input tape is initialized to $\triangleright w \triangleleft$ with the head at the left and every other tape is initialized to all zeros.

We make the following assumptions about δ:

- The head of the input tape does not move left of $\triangleright$ or right of $\triangleleft$
- The Turing machine does not simultaneously move and write on a given tape; i.e. if it *changes* the character, then the head stays put
- The transition function fixes any tuple whose first coordinate is an accept or reject state.

These assumptions do not appreciably change the time or space complexity of a Turing machine; given any Turing machine, one could make it satisfy these properties with no change in space and a constant factor slowdown in time.

In this section, we seek to prove the following theorem, which characterizes $\mathbf{N}_{Pd}$-computable sets in terms of computational complexity. First we fix an encoding of binary strings by numbers.

Definition 2.3.6. We code binary strings as numbers as follows: put a 1 at the *end* of a binary string and interpret it as a binary number in reverse. E.g.,

$$\varepsilon \mapsto 1$$

$$0 \mapsto 01 = 2$$

$$100 \mapsto 1001 = 9$$

Call this map $\pi : \{0,1\}^\star \to \mathbb{N}$. It is a bijection onto $\mathbb{N} \setminus \{0\}$.

Remark. For $\bar{u} \in \{0,1\}^\star$, π satisfies

$$2^{|\bar{u}|} \leq \pi(\bar{u}) \leq 2^{|\bar{u}|+1}$$

where $|\bar{u}|$ is the length of $\bar{u}$.

Using this, we shall show that for any subset $X \subseteq \{0,1\}^\star$,

$$X \in \textsc{Exptime} \iff \pi[X] \in \mathrm{rec}(\mathbf{N}_{Pd}) \tag{2.3}$$

$$X \in \textsc{Linspace} \iff \pi[X] \in \mathrm{tail}(\mathbf{N}_{Pd}) \tag{2.4}$$

(In fact, we will show a stronger result for *partial* predicates $X : \{0,1\}^\star \rightharpoonup \{0,1\}$)

Each of the four implications requires a different simulation either of Turing machines by $\mathbf{N}_{Pd}$-recursive programs or vice versa. Briefly:

$X \in \textsc{Exptime} \implies \pi[X] \in \mathrm{rec}(\mathbf{N}_{Pd})$		"time-indexed" simulation
$X \in \textsc{Exptime} \impliedby \pi[X] \in \mathrm{rec}(\mathbf{N}_{Pd})$		dynamic programming
$X \in \textsc{Linspace} \implies \pi[X] \in \mathrm{tail}(\mathbf{N}_{Pd})$		standard
$X \in \textsc{Linspace} \impliedby \pi[X] \in \mathrm{tail}(\mathbf{N}_{Pd})$		standard

The "time-indexed" simulation due to Neil Jones in [6] is by far the least familiar and most clever simulation. We present it in detail.

2.3.2.1 $\mathbf{N}_{Pd}$-recursion and Exptime

For a partial function $X : \{0,1\}^\star \rightharpoonup \{0,1\}$, let $\pi[X] : \mathbb{N} \rightharpoonup \{0,1\}$ be defined by $\pi[X](n) \simeq X(\pi^{-1}(n))$. In this way, we generalize taking the image of a subset to partial predicates.

A Turing machine $\mathcal{M}$ is said to *compute* X in case

$$X(\bar{u}) = 1 \iff \mathcal{M} \text{ accepts } \bar{u}$$

$$X(\bar{u}) = 0 \iff \mathcal{M} \text{ rejects } \bar{u}$$

Finally, we say that a Turing machine $\mathcal{M}$ runs in *weakly exponential time* if there's some c such that on input of length n, $\mathcal{M}$ runs for at most 2^{cn} steps or diverges.

We will prove

Theorem 2.3.7. *For a partial predicate $X : \{0,1\}^\star \rightharpoonup \{0,1\}$, if there's a weakly exponential time Turing machine $\mathcal{M}$ that computes X, then $\pi[X] \in rec(\mathbf{N}_{Pd})$*

The main intermediate theorem will be

Theorem 2.3.8. *If $\mathcal{M}$ is any Turing machine, there is a $\mathbf{N}_{Pd}$-recursive function S such that $S(t,n)$ is the state of $\mathcal{M}$ on input $\pi^{-1}(n)$ after t steps.*[1]

Towards this we make several definitions. The final proofs can be found on page 51.

Fix a Turing machine $\mathcal{M}$ and identify its set Q of states with $\{0,1,\ldots,|Q|-1\}$. Consider the functions

- $J(t,n)$, the distance of the input head from the left $\triangleright$ on input $\pi^{-1}(n)$ at time t.
- $M^0(t,n)$, the character read by the input head on input $\pi^{-1}(n)$ at time t. (Identify $\triangleright = 2$, $\triangleleft = 3$)
- $S(t,n)$, the state on input $\pi^{-1}(n)$ at time t.

For each tape i, for $1 \leq i \leq k$,

- $\sigma^i_L(t,n)$ is the reverse of the string *strictly* between the leftmost 1 and the head on tape i, input $\pi^{-1}(n)$, and at time t. (If there is no such 1, it's the empty string)

[1] "At time t," formally abbreviates "after t applications of the transition function to the initial input."

- $\sigma_R^i(t,n)$ is the string between the head and the rightmost 1, *including* the head, *excluding* the 1 on tape i, input $\pi^{-1}(n)$, and at time t. (Same comment applies)

So if the content of tape i on input $\pi^{-1}(n)$ at time t is

$$1a_2a_1a_0b_0b_11$$

with the head on b_0, then

$$\sigma_L^i(t,n) = a_0a_1a_2 \quad \sigma_R^i(t,n) = b_0b_1$$

We further define

- $M_L^i(t,n)$ is the head of the string $\sigma_L^i(t,n)$. [2] Similarly for $M_R^i(t,n)$. Notice that $M_R^i(t,n)$ is the character that the head of tape i reads on input $\pi^{-1}(n)$ at time t.

Now we define several predicates. For $0 \leq i \leq k$, the following apply to the t-th application of the transition, $\delta^t(\iota_n) \mapsto \delta^{t+1}(\iota_n)$, where ι_n is the initial input to δ on input $\pi^{-1}(n)$.

- $A^i(t,n)$ holds iff $\mathcal{M}$ will not move nor change the bit on tape i
- $B^i(t,n)$ holds iff $\mathcal{M}$ will not move but will change the bit on tape i
- $C^i(t,n)$ holds iff $\mathcal{M}$ will move the head of tape i to the left
- $D^i(t,n)$ holds iff $\mathcal{M}$ will move the head of tape i to the right.

By the assumption that the Turing machines does not simultaneously move and write, A, B, C, D exhaust all the possibilities. Hence $A^i(t,n)$, $B^i(t,n)$, $C^i(t,n)$, and $D^i(t,n)$ are each explicit relative to $(S(t,n), M^0(t,n), M_R^1(t,n), \ldots, M_R^k(t,n))$.

[2] "head" means first character, not to be confused with head of a Turing machine on a particular tape. We take the head of an empty string to be 0.

Now we prove a curious lemma. It says that for all $t \geq 1$, the tail of the string appearing to the left (resp. right) of the head after t steps *was* the string appearing to the left (resp. right) of the head after t' steps for some $t' < t$.[3]

Lemma 2.3.9. *For each tape* $1 \leq i \leq k$,$\forall t \geq 1$ $\exists t' < t$ *such that*

$$tail(\sigma_L^i(t,n)) = \sigma_L^i(t',n)$$

and similarly for σ_R

Proof. We show this by induction on $t \geq 1$. The proof for R instead of L is nearly identical.

Case 1. Suppose $t = 1$. In this case, the tail of σ_L^i remains unchanged: if the head moved, the tape is still all zeros, and if the head wrote, σ_L^i was unaffected. Hence $\text{tail}(\sigma_L^i(1,n)) = \varepsilon$. Since $\sigma_L^i(0,n) = \varepsilon$, we may take $t' = 0$.

Case 2. Suppose $t > 1$, and the configuration after t steps immediately follows a transition in which the head stayed put, i.e. $A^i(t-1,n) \vee B^i(t-1,n)$. Then, clearly, the tail of σ_L^i remains unchanged, i.e.

$$\text{tail}(\sigma_L^i(t,n)) = \text{tail}(\sigma_L^i(t-1,n))$$

By induction, there is a $t'' < t-1$ such that $\text{tail}(\sigma_L^i(t-1,n)) = \sigma_L^i(t'',n)$. We may simply take $t' = t''$.

Case 3. Suppose $t > 1$, and the configuration after t steps immediately follows a move to the right of the head, i.e. $D^i(t-1,n)$. Then $\text{tail}(\sigma_L^i(t,n)) = \sigma_L^i(t-1,n)$, so we may take $t' = t-1$.

[3] We take the tail of an empty string to be empty.

Case 4. Suppose $t > 1$, and the configuration after t steps immediately follows a move to the left of the head, i.e. $C^i(t-1,n)$. Then $\text{tail}(\sigma_L^i(t,n)) = \text{tail}^2(\sigma_L^i(t-1,n))$. By induction, there's $t'' < t-1$ such that $\sigma_L^i(t'',n) = \text{tail}(\sigma_L^i(t-1,n))$ and $t''' < t''$ such that $\sigma_L^i(t''',n) = \text{tail}(\sigma_L^i(t'',n))$. Thus, we may take $t' = t'''$.

□

Notice that this number t' as a function of t may be computed by the following recursion:

$$I_L^i(0,n) = 0$$

and for $t \geq 1$,

$$I_L^i(t,n) = \begin{cases} I_L^i(t-1,n) & \text{if } A^i(t-1,n) \vee B^i(t-1,n) \\ I_L^i(I_L^i(t-1,n),n) & \text{if } C^i(t-1,n) \\ t-1 & \text{if } D^i(t-1,n) \end{cases} \tag{2.5}$$

Then I_L^i satisfies

$$\text{tail}(\sigma_L^i(t,n)) = \sigma_L^i(I_L^i(t,n),n)$$

It's the function I_L^i that we actually care about, as well as the analogous function I_R^i. The function I_R^i may be calculated by an identical recursion, except that we switch C^i and D^i.

$$I_R^i(t,n) = \begin{cases} I_R^i(t-1,n) & \text{if } A^i(t-1,n) \vee B^i(t-1,n) \\ t-1 & \text{if } C^i(t-1,n) \\ I_R^i(I_R^i(t-1,n),n) & \text{if } D^i(t-1,n) \end{cases} \tag{2.6}$$

It turns out that for $1 \leq i \leq k$, M_L^i and M_R^i also be obey a similar recursion. Observe that

$$\sigma_L^i(t,n) = \begin{cases} \sigma_L^i(t-1,n) & \text{if } A^i(t-1,n) \\ \sigma_L^i(t-1,n) & \text{if } B^i(t-1,n) \\ \text{tail}(\sigma_L^i(t-1,n)) & \text{if } C^i(t-1,n) \\ \text{head}(\sigma_R^i(t-1,n))\hat{}\,\sigma_L^i(t-1,n) & \text{if } D^i(t-1,n) \end{cases}$$

and

$$\sigma_R^i(t,n) = \begin{cases} \sigma_L^i(t-1,n) & \text{if } A^i(t-1,n) \\ \text{flip}(\sigma_R^i(t-1,n)) & \text{if } B^i(t-1,n) \\ \text{head}(\sigma_R^i(t-1,n))\hat{}\,\sigma_L^i(t-1,n) & \text{if } C^i(t-1,n) \\ \text{tail}(\sigma_R^i(t-1,n)) & \text{if } D^i(t-1,n) \end{cases}$$

where for $\tau \in \{0,1\}^\star$, $\text{flip}(\tau)$ is τ with the head bit switched. From this we deduce

$$M_L^i(t,n) = \begin{cases} M_L^i(t-1,n) & \text{if } A^i(t-1,n) \\ M_L^i(t-1,n) & \text{if } B^i(t-1,n) \\ M_L^i(I_L^i(t-1,n),n) & \text{if } C^i(t-1,n) \\ M_R^i(t-1,n) & \text{if } D^i(t-1,n) \end{cases} \tag{2.7}$$

and

$$M_R^i(t,n) = \begin{cases} M_R^i(t-1,n) & \text{if } A^i(t-1,n) \\ \neg M_R^i(t-1,n) & \text{if } B^i(t-1,n) \\ M_L^i(t-1,n) & \text{if } C^i(t-1,n) \\ M_R^i(I_R^i(t-1,n),n) & \text{if } D^i(t-1,n) \end{cases} \tag{2.8}$$

respectively.

Lastly, observe that the character $M^0(t,n)$ under the head of the input tape is a function of the input n and the distance $J(t,n)$ from the left $\triangleright$, since the input tape is read-only. Roughly speaking, M^0 is the J-th bit of n, and is certainly computable from J and n in $\mathbf{N}_{Pd}$.

The function J satisfies the recursion

$$J(t,n) = \begin{cases} J(t-1,n) & \text{if } A^0(t-1,n) \vee B^0(t-1,n) \\ J(t-1,n)-1 & \text{if } C^0(t-1,n) \\ J(t-1,n)+1 & \text{if } D^0(t-1,n) \end{cases} \tag{2.9}$$

The cleverness of this construction Given a Turing machine $\mathcal{M}$, we have defined a slew of associated functions–J, $M^i_{L/R}$, $I^i_{L/R}$, S–and observed that they obey a complicated system of recursive equations (2.5), (2.6), (2.7), (2.8), (2.9). What is the purpose?

The point is we never have to compute any $\sigma^i_{L/R}$, somewhat surprisingly. The only numbers which come up when computing $S(t,n)$, for example, are other "time values" less than t. The upshot of this is that the size of the parameters seen in simulating a $\mathcal{M}$ is purely a function of the time complexity of $\mathcal{M}$, not what appears on the tapes of $\mathcal{M}$, hence the term "time-indexed simulation."

This technique is due to Jones in [6], who applied it to prove a similar result for a structure of strings without a "cons" operation. We mimic his proof in the case of $\mathbf{N}_{Pd}$.

Proof of theorem 2.3.8. Let $\mathcal{M}$ be a Turing machine. We first show that each of the functions

$$J,\ M^i_{L/R},\ M^0,\ I^i_{L/R},\ S$$

are computable in $\mathbf{N}_{Pd}$.

Consider the system of equations given by equations (2.5), (2.6), (2.7), (2.8), (2.9), and the explicit expressions for

$$A^i(t,n)\ B^i(t,n)\ C^i(t,n)\ D^i(t,n)$$

in terms of

$$(S(t,n), M^0(t,n), \ldots, M^k_R(t,n))$$

It is easy to show, by induction on t, that the least fixed point solution of these equations converges everywhere. Hence the functions computed by these equations are literally the functions J, M, I, S, etc. defined below Theorem 2.3.8. □

Proof of theorem 2.3.7. If $\mathcal{M}$ is weakly exponential, there's some c such that on all convergent input $\bar{u}$, $\mathcal{M}$ runs for at most $2^{c|\bar{u}|}$ steps.

In other words, for $t \geq 2^{c|\bar{u}|}$, $S(t, \pi(\bar{u}))$ is either an accept or reject state or $\mathcal{M}$ runs forever on $\bar{u}$. Moreover, if $n = \pi(\bar{u})$, then $n \geq 2^{|\bar{u}|}$. So let the head of our program E be

$$f(n) = \begin{cases} 1 & \text{if } S(n^c, n) \text{ is an accept state} \\ 0 & \text{if } S(n^c, n) \text{ is a reject state} \\ \uparrow & \text{otherwise} \end{cases}$$

Then E computes exactly $\pi[X]$. Since E visibly runs in polynomial parameters, by Corollary 2.3.4 $\pi[X]$ is in fact computable by a $\mathbf{N}_{Pd}$-recursive program. □

Hence we've shown that given X is EXPTIME-computable, $\pi[X]$ is computable in $\mathbb{N}_{Pd}$.

To show the converse, we use *dynamic programming.*

Theorem 2.3.10. *For a partial predicate $X : \{0,1\}^\star \rightharpoonup \{0,1\}$, if $\pi[X] \in rec(\mathbf{N}_{Pd})$ then there is a weakly exponential time Turing machine $\mathcal{M}$ computing X.*

Proof. Suppose that E is a unary recursive $\mathbf{N}_{Pd}$-program. Then for fixed n, there are polynomially many $(\mathbf{N}_{Pd}, E)$-terms all of whose parameters are less than n (see definition 1.1.8 for the definition of a $(\mathbf{N}_{Pd}, E)$-term). Therefore, the number of distinct $(\mathbf{N}_{Pd}, E)$-terms that occur on input n is at most $p(n)$ for a polynomial p.

We shall write pseudocode for a Turing machine computing X in exponential time and, as is the usual practice, rely on the reader to convince himself or herself that the devil does not dwell in these particular details. The idea is, we write down an array indexed by all possible $(\mathbf{N}_{Pd}, E)$-terms reachable from $E(n)$. For a term M, the entry $A(M)$ will initially be undefined, and then eventually store the denotation of M. We pass through the array $p(n)$ times, and each time update more and more terms.

1. On input $\bar{u} = \pi^{-1}(n) \in \{0,1\}^\star$, initialize an array A with at $p(n)$ indices, each of which corresponds to a particular $(\mathbf{N}_{Pd}, E)$-term with parameters $\leq n$. This array takes values in $\mathbb{N} \cup \{\uparrow, \mathrm{tt}, \mathrm{ff}\}$, and all values are initially $\uparrow$.

2. We make $p(n)$ passes through the array. For $0 \leq i < p(n)$:

   (a) For $0 \leq j < p(n)$:

      i. consider the $(\mathbf{N}_{Pd}, E)$-term M_j. Let S be the set of all $(\mathbf{N}_{Pd}, E)$-terms immediately below M_j. If $A(M)$ is defined for each $M \in S$, then update $A(M_j)$ by the denotation of M_j.

3. Output $A(E_0(n))$ if it's defined, or run forever.

The correctness of this program, i.e. that it computes X, is verified by checking the following two statements:

- At any point in time in the program and for any $(\mathbf{N}_{Pd}, E)$-term M,

$$A(M) = w \implies \text{den}(\mathbb{N}, M) = w$$

- Either E fails to converge on $\bar{u}$, or at least one additional element in A gets updated with each subsequent pass.

Now we analyze the running time of the program. These next few statement are rife with obfuscation–we implicitly rely on the reader's belief that the Turing machine implementation of the pseudocode above is not much less efficient as the pseudocode itself.

- Each step 2.(a)i. takes time polynomial in the length of the array A.
- Step 2.(a)i. gets executed $p(n)^2$ times.
- Step 1. takes time polynomial in the length of the array A.

Since the length of the array is itself polynomial in A, the total time taken is polynomial in n for convergent input n. (For divergent input n, this Turing machine clearly diverges.) Since the input $\bar{u}$ is $\pi^{-1}(n)$, this is exponential in the length of $\bar{u}$.

Hence we have computed X with a weakly exponential time Turing machine.

□

2.3.2.2 $\mathbf{N}_{Pd}$-tail recursion vs. Linspace

Now we sketch the proof that a subset of binary strings is computable in linear space iff its image under the map π (definition (2.3.6)) is computable by an

$\mathbf{N}_{Pd}$-tail recursion. Both directions of this proof use a very familiar simulation of Turing machines by register programs and vice versa.

Theorem 2.3.11. *For a partial predicate $X : \{0,1\}^\star \rightharpoonup \{0,1\}$, if X is computable by a linear space Turing machine $\mathcal{M}$, then $\pi[X] \in tail(\mathbf{N}_{Pd})$*

Proof. Let $\mathcal{M}$ be a linear space Turing machine computing X. To show that X is computable over $\mathbf{N}_{Pd}$, it suffices to show that X is computable by a tail recursion over $\mathbf{N}_u$ in polynomial parameters. To show this, it suffices to show that X is computable by a tail recursion over $\mathbf{N}_b$ in polynomial parameters, since the primitives of $\mathbf{N}_b$ are themselves computable by tail recursions over $\mathbf{N}_u$ in polynomial parameters.

By the *configuration* of a Turing machine we mean its state and the contents of every tape. Identify the states Q of $\mathcal{M}$ with the set $\{0, 1, \ldots, |Q| - 1\}$.

Each configuration of $\mathcal{M}$ may be coded by a $(2k+2)$-tuple of natural numbers $\bar{n}$ as follows. If $\mathcal{M}$ is in state $q \in Q$, let $n_0 = q$. If the input tape contains $\triangleright\bar{u}\triangleleft$, let $n_1 = \pi(\bar{u})$. If work tape i is of the form

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

with the head on b_0 and zeros on the outside, let

$$n_{2i} = \pi(a_0 \ldots a_{n-1}) \quad n_{2i+1} = \pi(b_0 \ldots b_{m-1})$$

(As usual, if the left or rightmost 1 does not exist, then $\bar{a}$ and/or $\bar{b}$ is ε accordingly.)

Now observe that there is an *explicit* $\mathbf{N}_b$-term T of arity and co-arity $2k+2$ such that

$$\bar{n} \mapsto T(\bar{n})$$

is the image of the transition function of $\mathcal{M}$. Furthermore, there are explicit $\mathbf{N}_b$-predicate *halt* and *acc* such that

$$halt(\bar{n}) \iff \bar{n} \text{ is in a halt state}$$

$$halt(\bar{n}) \implies (acc(\bar{n}) \iff \bar{n} \text{ is in an accepting halt state})$$

There is also an explicit $\mathbf{N}_b$-term *in* of arity 1 and co-arity $2k+2$ such that $in(\pi(\bar{u}))$ is the code of the initial configuration of $\mathcal{M}$ on input $\bar{u}$.

Hence if E is the tail recursive program

$$p(in(n)) \text{ where}$$

$$p(\bar{n}) = \text{if } halt(\bar{n}) \text{ then } acc(\bar{n}) \text{ else } p(T(\bar{n}))$$

then E computes $\pi[X]$ over $\mathbf{N}_b$.

It remains to show that E runs in polynomial parameters. The key observation is that any parameter appearing in the computation of E on input $\pi(\bar{u})$ is $\pi(\bar{v})$ for some substring $\bar{v}$ of a string appearing on a tape of $\mathcal{M}$ on input $\bar{u}$. Since $\mathcal{M}$ runs in linear space, $|\bar{v}| \leq C|\bar{u}|$ for some constant C independent of $\bar{u}$. But this implies that $\pi(\bar{v}) \leq \pi(\bar{u})^C$, so we are done. □

Finally, we prove the remaining direction.

Theorem 2.3.12. *For a partial predicate* $X : \{0,1\}^\star \rightharpoonup \{0,1\}$, *if* $\pi[X] \in$ *tail*$(\mathbf{N}_{Pd})$ *then there is a linear space Turing machine computing* X.

Proof. We outline the proof. Suppose we have an $\mathbf{N}_{Pd}$−register program P computing X. We define a Turing machine $\mathcal{M}$ with one work tape for each register. Assume each work tape is one-sided; this makes no difference as far as complexity is concerned. We have to show we can simulate each of the register

machine's operations on the Turing machine using space efficiently.

$\mathcal{M}$ will have a group of instructions corresponding to each line of P. We may assume that every assignment line is of the form $X \leftarrow Y$, $X \leftarrow Pd(X)$, $X \leftarrow 0$, or $X \leftarrow 1$, where X and Y range over registers of P. Assume that before each of these instructions, the tape heads are at the ends of their respective tapes. Then the instructions corresponding to the assignment lines are as follows:

To simulate $X \leftarrow Y$, $\mathcal{M}$ copies the contents of tape Y to tape X and moves the heads back to the end of the tapes.

To simulate $X \leftarrow Pd(X)$, $\mathcal{M}$ subtracts 1 off the binary number in tape X and moves the heads back to the end of the tapes.

To simulate $X \leftarrow 0$ or $X \leftarrow 1$, $\mathcal{M}$ replaces the contents of X with $\pi^{-1}(0)$ or $\pi^{-1}(1)$ respectively and moves the heads back to the ends of the tapes.

To simulate a branching line requires testing $X = Y$, which is done by comparing the contents of tape X to tape Y.

It is easy to see that (1) if the tapes of $\mathcal{M}$ store the π-images of the contents of the registers of P, then the same holds after each of these steps. Moreover, (2) the heads do not need more space than the length of the longest string to execute any of these steps.

It is easy to complete the definition of $\mathcal{M}$ so that it computes X and, using (2), runs in linear space. □

2.3.2.3 Total functions suffice

We have shown that $\mathbf{N}_{Pd}$-recursion corresponds to computability by weakly exponential time Turing machines and $\mathbf{N}_{Pd}$-tail recursion corresponds to computability by linear space Turing machines. However, Exptime and Linspace are defined as families of subsets, i.e. *total* predicates, of $\{0,1\}^\star$. So we have not yet shown that if there is an $\mathbf{N}_{Pd}$-recursive non-tail recursive partial function $\mathbb{N} \rightharpoonup \{0,1\}$ then Exptime $\neq$ Linspace.

It turns out not to matter. If $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ is a $\mathbf{N}_{Pd}$-recursive (resp. tail recursive) partial function, then its domain of convergence is again recursive (resp. tail recursive), as we will show. In other words *over* $\mathbf{N}_{Pd}$*, all semirecursive sets are recursive.* So if $f^\star$ is defined by

$$f^\star(\bar{n}) = \begin{cases} f(\bar{n}) & \text{if } f(\bar{n}) \downarrow \\ 0 & \text{otherwise} \end{cases}$$

then f is recursive (resp. tail recursive) iff the total function $f^\star$ is recursive (resp. tail recursive). Therefore, if there is a recursive not tail recursive partial function, there is a recursive not tail recursive total function.

In light of the theorems in the previous sections, it suffices to prove

Lemma 2.3.13. *The domain of convergence of a weakly exponential time (resp. linear space) Turing machine is computable in exponential time (resp. linear space)*

Proof. (Sketch) Suppose $\mathcal{M}$ is a Turing machine that if it converges, converges in time 2^{cn}. Consider the Turing machine $\mathcal{M}^\varnothing$ ("$\mathcal{M}$-clock") which is $\mathcal{M}$ with one additional work tape that is initialized to 2^{cn} on input of length n. This is the "clock." The machine $\mathcal{M}^\varnothing$ alternates between performing a transition of $\mathcal{M}$ and decrementing the clock. If $\mathcal{M}$ converges before the clock hits 0, $\mathcal{M}^\varnothing$ accepts, otherwise it rejects.

Hence $\mathcal{M}^\varnothing$ computes the domain of convergence of $\mathcal{M}$. Crucially, (1) $\mathcal{M}^\varnothing$ still runs in exponential time, and (2) if $\mathcal{M}$ runs in linear space then so does $\mathcal{M}^\varnothing$. □

2.3.3 Recursion versus tail recursion in $\mathbf{N}_{Pd}$

In this section, we finally prove that

$$\mathrm{rec}(\mathbf{N}_{Pd}) = \mathrm{tail}(\mathbf{N}_{Pd}) \iff \textsc{Exptime} = \textsc{Linspace}$$

The forwards direction is immediate from theorems 2.3.10 and 2.3.11: if there were a subset of binary strings $X \in \textsc{Exptime} \setminus \textsc{Linspace}$, then $\pi[X]$ would be $\mathbf{N}_{Pd}$-recursive but not tail recursive. However, if we have an element in $\mathrm{rec}(\mathbf{N}_{Pd}) \setminus \mathrm{tail}(\mathbf{N}_{Pd})$, it may not be a unary partial predicate, in which case we cannot conclude $\pi^{-1}[\bar{X}]$ is a subset of strings in $\textsc{Exptime} \setminus \textsc{Linspace}$.

To complete the backwards direction of the proof, we will show:

1. If there is a recursive non tail recursive partial function in $\mathbf{N}_{Pd}$, there is a recursive non tail recursive partial predicate.

2. If there is a recursive non tail recursive partial predicate in $\mathbf{N}_{Pd}$, we may assume it's unary.

The first statement is verified by

Claim. A partial function $f : \mathbb{N}^k \rightharpoonup \mathbb{N}$ is (tail) recursive over $\mathbf{N}_{Pd}$ iff the partial graph relation $G^f : \mathbb{N}^{k+1} \rightharpoonup \{0,1\}$ is, where

$$f(x)\downarrow \implies G^f(x,y) = \begin{cases} 1 & \text{if } y = f(x) \\ 0 & \text{otherwise} \end{cases}$$

$$f(x)\uparrow \implies G^f(x,y)\uparrow$$

Proof. Clearly if f is recursive, so is G^f. On the other hand,

$$f(\bar{x}) = \mu y \leq \max\{\bar{x}\}\ G^f(\bar{x}, y)$$

is a tail recursion computing f in $(\mathbf{N}_{Pd}, G^f)$. $\square$

Now we prove the second statement.

Definition 2.3.14. Let $T_k(x_1, \ldots, x_k) = \langle x_1, \langle x_2, \langle \ldots, x_k \rangle \rangle \rangle$, where $\langle \cdot, \cdot \rangle$ is the Cantor pairing function. Then T_k is a bijection between $\mathbb{N}^k$ and $\mathbb{N}$. Let S_k be its inverse.

It is easy to see that both T_k and S_k are computable in $\mathbf{N}_u$ by tail recursions with polynomial parameters.

Theorem 2.3.15. *Let $X : \mathbb{N}^k \rightharpoonup \{0,1\}$. Then X is computable over $\mathbf{N}_{Pd}$ (by a tail recursion) if and only if $Y : \mathbb{N} \rightharpoonup \{0,1\}$ is computable over $\mathbf{N}_{Pd}$ (by a tail recursion) where Y is defined by*

$$Y \simeq X \circ S_k$$

Proof. By Corollary 2.3.5, if X is $\mathbf{N}_{Pd}$-recursive then so is Y, since since S_k is computable in polynomial parameters and $Y \simeq X \circ S_k$. Similarly, if Y is $\mathbf{N}_{Pd}$-recursive, so is X, since $X \simeq Y \circ T_k$. The same holds for tail recursion. $\square$

Since Y is unary, we have as an immediate consequence:

Corollary 2.3.16. *If there is a recursive non-tail-recursive predicate over $\mathbf{N}_{Pd}$, then there is recursive non-tail-recursive unary predicate.*

Chapter 3

Algebraic Structures

In this chapter we connect the recursion theory of $\mathbf{N}_{Pd}$ with the recursion theory of two locally finite algebraic structures. For the entirety of this paper, let p be a fixed prime. In sections 3.1 and 3.2 we show that

$$\mathrm{rec}(\bar{\mathbb{F}}_p) = \mathrm{tail}(\bar{\mathbb{F}}_p) \iff \mathrm{rec}(\mathbf{N}_{Pd}) = \mathrm{tail}(\mathbf{N}_{Pd}) \tag{3.1}$$

where

$$\bar{\mathbb{F}}_p := (\bar{\mathbb{F}}_p, 0, 1, +, \times, -, \div, =)$$

is the algebraic closure of $\mathbb{F}_p$, the finite field with p elements.

In section 3.3 we show that

$$\mathrm{rec}(\{\mathbf{A}_i\}) = \mathrm{tail}(\{\mathbf{A}_i\}) \iff \mathrm{rec}(\mathbf{N}_{Pd}) = \mathrm{tail}(\mathbf{N}_{Pd}) \tag{3.2}$$

where $\{\mathbf{A}_i\}$ is the family of all abelian groups in the signature $\{0, +, -, =\}$.

3.1 Finite Fields and their recursion theory

To link the recursion theory of $\bar{\mathbb{F}}_p$ with that of $\mathbf{N}_{Pd}$, we establish a uniform family of bi-interpretations between the families of finitely generated substructures of each structure. This takes quite a bit of machinery to carry through in full detail, but the idea is quite simple.

Imagine the collection $\{\mathbb{F}_p(x) : x \in \bar{\mathbb{F}}_p\}$ of subfields of $\bar{\mathbb{F}}_p$ generated by a single element. Each element in $\mathbb{F}_p(x)$ has a unique representation as $\sum_{i<d} a_i x^i$ for $a_i \in \mathbb{F}_p$ and d the degree of x over $\mathbb{F}_p$. Code such an element by the number $\sum_{i<d} a_i p^i$. Then the image of each field operation is computable in $\mathbf{N}_{Pd}$ by a tail recursion, given $d-1$ as an extra parameter. Similarly, the pre-image of the predecessor function is uniformly computable over $\mathbb{F}_p(x)$ by a tail recursion, a fact which uses one non-elementary fact about finite fields (theorem 3.1.8).

In this way, we construct a bi-interpretation. Its definition in the case of subfields with more than one generator takes substantially more room to define, but is not more complicated conceptually.

3.1.1 Review of relevant field theory

First we review finite fields and define $\bar{\mathbb{F}}_p$. For each (p, n) there is one field of order p^n up to isomorphism, denoted $\mathbb{F}_{p^n}$. It contains exactly one subfield of order p^m for $m|n$.

Definition 3.1.1. $\bar{\mathbb{F}}_p$ is the direct limit of the system $\{\mathbb{F}_{p^n}\}_{n\in\mathbb{N}}$, ordered by $\subseteq$. It is the algebraic closure of $\mathbb{F}_p$.

First we define minimal polynomials in general field extensions E/F.

Definition 3.1.2. Suppose E/F is a field extension, and suppose that $x \in E$ is algebraic over F. There is a unique monic irreducible polynomial $f \in F[t]$ called the *minimal polyomial* such that $f(x) = 0$.

Next we specialize to the case where E and F are finite fields, characterize $\mathrm{Aut}(E/F)$, and define the norm and trace. All of these definitions are standard.

Definition 3.1.3. If E/F is a field extension and E and F are finite fields, the *degree* of E/F is the dimension of E as an F-vector space. An *ordered basis* $(\alpha_0, \ldots, \alpha_{n-1})$ of E/F is simply an ordered basis of E as an F-vector space.

If $E = F(\alpha)$ for some $\alpha \in E$, then the degree of E/F is simply the degree of the minimal polynomial of α with respect to E/F. If that degree is n, the ordered basis $(1, \alpha, \ldots, \alpha^{n-1})$ of E/F is called a *power basis*.

Fact 3.1.4. *Let E/F be an extension of finite fields of degree d. Let $Aut(E/F)$ be the group of automorphisms of E fixing F. Then:*

1. *$Aut(E/F)$ is cyclic and generated by the map $\sigma : x \mapsto x^m$, where $m = |F|$*
2. *σ has order d in $Aut(E/F)$.*
3. *If $E = F(\alpha)$ for some $\alpha \in E$, then the orbit of α under σ is*

$$\{\alpha, \sigma(\alpha), \ldots, \sigma^{d-1}(\alpha)\}$$

In this case, if f is the minimal polynomial of α with respect to E/F,

$$f(t) = \prod_{0 \le i < d} (t - \sigma^i(\alpha))$$

Definition 3.1.5. Let E and F be finite fields of characteristic p. Let $G = \mathrm{Aut}(E/F)$. Then for $x \in E$, the *trace* and *norm* are defined by:

$$\mathrm{Tr}_{E/F}(x) = \sum_{g \in G} g(x)$$

$$\mathrm{No}_{E/F}(x) = \prod_{g \in G} g(x)$$

Fact 3.1.6. *The maps $Tr_{E/F}$ and $No_{E/F}$ map into F. In fact,*

$$Tr_{E/F} : E \to F$$

is F-linear.

Definition 3.1.7. Let E/F be an extension of finite fields of degree n and let $(\alpha_0, \ldots, \alpha_{n-1})$ and $(\gamma_0, \ldots, \gamma_{n-1})$ be two ordered bases of E/F. We say that they are *dual* to each other in case

$$\mathrm{Tr}_{E/F}(\alpha_i \beta_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

The next theorem, found in [1], gives a formula for the dual basis of a power basis.

Theorem 3.1.8. *Let F be a finite field of characteristic p, and $\alpha \in \bar{\mathbb{F}}_p$. Let $f \in F[t]$ be the minimal polynomial of α with respect to the extension $F(\alpha)/F$. Then $f(\alpha) = 0$, so we may write*

$$f(t) = (t - \alpha)\Big(\sum_{j<n} b_j t^j\Big) \tag{3.3}$$

for some $b_j \in F(\alpha)$ and $n \in \mathbb{N}$.

We know $(1, \alpha, \ldots, \alpha^{n-1})$ forms a basis of $F(\alpha)/F$. The dual basis $(\gamma_0, \ldots, \gamma_{n-1})$ is given by the formula

$$\gamma_i = \frac{b_i}{f'(\alpha)}$$

where $f'(t)$ is the formal derivative of $f(t)$.

Corollary 3.1.9. *For $\beta \in F(\alpha)$, write $\beta = \sum_{i<n} \beta_i \alpha^i$, where $\beta_i \in F$. Then*

for $0 \leq i < n$, β_i *satisfies*

$$\beta_i = Tr_{F(\alpha)/F}(\beta\gamma_i)$$

Proof. Omit the subscript $F(\alpha)/F$ from Tr for readability. Then:

$$\mathrm{Tr}(\beta\gamma_i) = \mathrm{Tr}(\sum_{j<n} \beta_j \alpha^i \gamma_i)$$

by F-linearity of the trace,

$$\mathrm{Tr}(\beta\gamma_i) = \sum_{j<n} \beta_j \mathrm{Tr}(\alpha^i \gamma_i)$$

Since $(1, \alpha, \ldots, \alpha^{n-1})$ and $(\gamma_0, \gamma_1, \ldots, \gamma_{n-1})$ are dual bases of $F(\alpha)/F$, the right hand side is simply β_i. □

3.1.2 Some functions uniformly recursive over $\{\mathbb{F}_p(\bar{x}, y)\}_{(\bar{x},y)\in\bar{\mathbb{F}}_p^{k+1}}$

Definition 3.1.10. For a finite field F and an element $y \in \bar{\mathbb{F}}_p$, let $\mathfrak{h} : F(y) \to F$ be defined by

$$\mathfrak{h} : \sum_{j<d} \beta_j y^j \mapsto \beta_0 \tag{3.4}$$

Then $\mathfrak{h}$ computes the constant term of an element in $F(y)$ according to the basis $(1, \ldots, y^{d-1})$. Using Corollary 3.1.9 we see that for $u \in F(y)$,

$$\mathfrak{h}(u) = \mathrm{Tr}_{F(y)/F}(u\gamma_0)$$

Let $\mathfrak{t} : F(y) \to F(y)$ be defined by

$$\mathfrak{t}(u) = \frac{u - \mathfrak{h}(u)}{y}$$

The functions $\mathfrak{h}$ and $\mathfrak{t}$ are supposed to suggest *head* and *tail*–if we think of an element of $F(y)$ as given by its list of coefficients in the power basis of y, then indeed $\mathfrak{h}$ and $\mathfrak{t}$ act like the head and tail functions on lists.

Let $\bar{x} = (x_1, \ldots, x_k) \in \bar{\mathbb{F}}_p^k$. Let $\{\mathbb{F}_p(\bar{x}, y) : (\bar{x}, y) \in \bar{\mathbb{F}}_p^{k+1}\}$ be the family of fields $\mathbb{F}_p(\bar{x}, y)$ with constants for $(\bar{x}, y)$.

Lemma 3.1.11. *The functions $\mathfrak{h}$ and $\mathfrak{t}$ are uniformly tail recursive over $\mathbb{F}_p(\bar{x}, y)$.*

To do this we will show that several intermediate functions are also uniformly tail recursive. Once we know that the $\mathrm{Tr}_{F(y)/F}$ and γ_0 are uniformly recursive, then clearly $\mathfrak{h}$ must be uniformly recursive. Finally, $\mathfrak{t}$ is explicit in terms of $\mathfrak{h}$.

First, some definitions. For the remainder of section 3.1:

- Fix $(\bar{x}, y) \in \bar{\mathbb{F}}_p^{k+1}$
- Let $F_0 = \mathbb{F}_p$ and for $1 \leq i \leq k$, let $F_i = \mathbb{F}_p(x_1, \ldots, x_i)$. Let $F_{k+1} = \mathbb{F}_p(\bar{x}, y)$. (Of course, not all F_i are necessarily distinct.)
- For $0 \leq i \leq k+1$, let $q_i = |F_i|$
- For $0 \leq i \leq k$, let $d_i = \deg(F_{i+1}/F_i)$, so that $q_i^{d_i} = q_{i+1}$.
- Let $G = \mathrm{Aut}(F_{k+1}/F_k)$
- For $0 \leq i \leq k$, let $\sigma_i : \mathbb{F}_p(\bar{x}, y) \to \mathbb{F}_p(\bar{x}, y)$ be defined by $\sigma_i : u \mapsto u^{q_i}$. The following statements are implied by Fact 3.1.4:
  - Since $F_{i+1} = F_i(x_{i+1})$, the orbit of x_{i+1} under σ_i has size d_i. (And the orbit of y under σ_k has size d_k)
  - The order of σ_i (as a group element) is d_i.
  - σ_k is a generator of G
- We omit the subscript $\mathbb{F}_p(\bar{x}, y)/\mathbb{F}_p(\bar{x})$ from Tr and No

First we uniformly compute the σ_i by tail recursive functions, then we uniformly compute Tr and No by tail recursions, and finally we uniformly compute γ_0 by a tail recursion. The programs uniformly computing these functions are programs in the language of $\mathbb{F}_p(\bar{x}, y)$, which is the language of fields with $k+1$ constants.

3.1.2.1 The σ_i

When $i = 0$, $q_0 = p$, so $\sigma_0 : u \mapsto u^p$. This is computed by the tail recursion

$$\sigma_0(u) = \alpha(1, u, u) \text{ where}$$

$$\alpha(a, b, c) = \text{if } a = 0 \text{ then } b \text{ else } \alpha(a+1, bc, c)$$

In words, we simultaneously add 1 to itself and multiply u by itself until the former is 0, at which point the latter is u^p.

When $i > 0$, we can compute σ_{i-1} by induction. Then σ_i is computed by

$$\sigma_i(u) = \alpha(\sigma_{i-1}(x_i), \sigma_{i-1}(u)) \text{ where}$$

$$\alpha(a, b) = \text{if } a = x_i \text{ then } b \text{ else } \alpha(\sigma_{i-1}(a), \sigma_{i-1}(b))$$

This algorithm is similar. It computes $\sigma_{i-1}^{d_{i-1}}$, which is σ_i due to the fact that $q_{i-1}\hat{}\, d_{i-1} = q_i$.

3.1.2.2 Norm and trace

Recall the definition 3.1.5 of trace and norm for an extension E/F of finite fields. In this case, $E = \mathbb{F}_p(\bar{x}, y)$ and $F = \mathbb{F}_p(\bar{x})$, and we know that G is generated by σ_k, which has order d_k. So in this case the definitions become

$$\mathrm{Tr}(x) = \sum_{0 \le i < d_k} \sigma_k^i(x)$$

$$\mathrm{No}(x) = \prod_{0 \le i < d_k} \sigma_k^i(x)$$

To compute the trace we start with $(x, 0)$ and repeatedly iterate $(a, b) \mapsto (\sigma_k(a), a + b)$ d_k times. To measure out d_k, we use the fact that the size of the orbit of y under σ_k has size d_k.

Computing the norm is similar, with ab replacing $a + b$.

3.1.2.3 The constant γ_0

We know that by theorem 3.1.8,

$$\gamma_0 = \frac{b_0}{f'(y)}$$

where f is the minimal polynomial for y. By fact 3.1.4, we know that

$$f(t) = \prod_{0 \le i < d_k} (t - \sigma_k^i(y))$$

By the equation (3.3), b_0 is the constant term of the polynomial $f(t)/(t-y)$. Hence

$$b_0 = \prod_{1 \le i < d_k} \sigma_k^i(y)$$

which can be computed by a tail recursion.

Similarly, it's easy to see that

$$f'(y) = \prod_{1 \le i < d_k} (y - \sigma_k^i(y))$$

can be computed by a tail recursion.

3.2 Recursion versus tail recursion for finite fields

In this section we use the functions from the previous section to construct an explicit family of bi-interpretations for each natural number k. Specifically, for each k we will find a number ℓ, a map $\rho_k : \bar{\mathbb{F}}_p^k \to \mathbb{N}^\ell$, and a family of bijections $\pi_{\bar{x}}$ of the family of structures

$$\{\mathbb{F}_p(\bar{x}) : \bar{x} \in \bar{\mathbb{F}}_p^k\} \tag{3.5}$$

with the family of structures

$$\{(\langle \rho(\bar{x}) \rangle_{\mathbf{N}_{Pd}}, \rho(\bar{x})) : \bar{x} \in \bar{\mathbb{F}}_p^k\} \tag{3.6}$$

such that the image of primitives in each family are uniformly computable over the other.

For brevity, denote by $\langle s_1, \ldots, s_n \rangle^\star$ the structure $\langle s_1, \ldots, s_n \rangle_{\mathbf{N}_{Pd}}$ expanded with constants for the s_i, so that the structures in equation (3.6) may be denoted $\langle \rho(\bar{x}) \rangle^\star$.

We define ρ_k and $\pi_{\bar{x}}$ for $|\bar{x}| = k$ by induction on k.

3.2.1 The functions ρ and π

Let $\ell = 3k + 1$. First we define a function $\rho_k : \bar{\mathbb{F}}_p^k \to \mathbb{N}^{3k+1}$ and a family of functions $\pi_{\bar{x}} : \mathbb{F}_p(\bar{x}) \to \mathbb{N}$ for $\bar{x} = (x_1, \ldots, x_k)$. Then we show $\pi_{\bar{x}}$ is a bijection into the desired set.

3.2.1.1 $k = 0$

When $k = 0$, there is one k-tuple, namely ε, the empty tuple. Then $\mathbb{F}_p(\varepsilon)$ is simply $\mathbb{F}_p$. Identify the domain of $\mathbb{F}_p$ with $\{0, 1, \ldots, p-1\}$ and let π_ε be the identity on $\mathbb{F}_p$.

When $k = 0$, $\rho_k = \rho_0$ is a nullary function into $\mathbb{N}$, i.e. a natural number. Define $\rho_0 = p - 1$.

3.2.1.2 $k > 0$

Denote a typical k-tuple of elements in $\bar{\mathbb{F}}_p$ by $(\bar{x}, y)$ for $\bar{x}$ a $(k-1)$-tuple. Then we will define ρ_k and $\pi_{(\bar{x},y)}$ given ρ_{k-1} and $\pi_{\bar{x}}$.

Definition 3.2.1. For fixed $(\bar{x}, y)$, define:

$$d := \deg \mathbb{F}_p(\bar{x}, y)/\mathbb{F}_p(\bar{x})$$
$$q := |\mathbb{F}_p(\bar{x})|$$
$$Q := |\mathbb{F}_p(\bar{x}, y)|$$

so that $d \geq 1$ and $Q = q^d$.

The definition of $\pi_{(\bar{x},y)}$ may look complicated, but actually has a simple, intuitive explanation. By induction, elements from $\mathbb{F}_p(\bar{x})$ will be coded (via $\pi_{\bar{x}}$) by numbers base p of length at most $D = \deg(\mathbb{F}_p(\bar{x})/\mathbb{F}_p)$. Given $\alpha \in \mathbb{F}_p(\bar{x}, y)$ consider the coefficients $\alpha_j \in \mathbb{F}_p(\bar{x})$ defined by $\alpha = \sum_{j<d} \alpha_j y^d$. By "concatenating" the d codes of the elements α_j together we obtain a number base p of length at most dD, which is the code for α.

Definition 3.2.2. Since $\{1, y, \ldots, y^{d-1}\}$ is a basis of $\mathbb{F}_p(\bar{x}, y)$ as a vector space over $\mathbb{F}_p(\bar{x})$, each element $\alpha \in \mathbb{F}_p(\bar{x}, y)$ has a unique representation $\sum_{j<d} \alpha_j y^j$ for $\alpha_j \in \mathbb{F}_p(\bar{x})$. Then define

$$\pi_{(\bar{x},y)}(\alpha) = \sum_{j<d} \pi_{\bar{x}}(\alpha_j) q^j \tag{3.7}$$

Notice that $\pi_{(\bar{x},y)}$ extends $\pi_{\bar{x}}$, i.e. $\pi_{\bar{x}} = \pi_{(\bar{x},y)} \restriction \mathbb{F}_p(\bar{x})$, and that for $u \in \mathbb{F}_p(\bar{x}, y)$, $u \in \mathbb{F}_p(\bar{x}) \iff \pi(u) < q$. Also notice that if $d > 1$, $\pi(y) = q$.

Now we define ρ_k.

Definition 3.2.3. For $k \geq 1$, define

$$\rho_k(\bar{x}, y) = (\rho_{k-1}(\bar{x}), d, r, Q-1) \tag{3.8}$$

where $r = \pi_{(\bar{x},y)}(-y^d)$

With these definitions, we can prove

Claim. $\pi_{(\bar{x},y)} : \mathbb{F}_p(\bar{x}, y) \to \{0, \ldots, Q-1\}$ is a bijection

Proof. Expressing an element of $\mathbb{F}_p(\bar{x}, y)$ in the power basis over $\mathbb{F}_p(\bar{x})$ gives rise to a bijection

$$\mathbb{F}_p(\bar{x}, y) \to \mathbb{F}_p(\bar{x})^d$$

The map

$$\pi_{\bar{x}} : \mathbb{F}_p(\bar{x}, y) \to \{0, \ldots, q-1\}$$

is a bijection by induction, and

$$(n_i)_{i<d} \to \sum_{i<d} n_i q^i$$

$$\{0, \ldots, q-1\}^d \to \{0, \ldots, Q-1\}$$

is a bijection. Their composition is $\pi_{(\bar{x},y)}$ □

Remark 3.2.4. It will also be helpful to make explicit $\pi^{-1}_{(\bar{x},y)}$. Each element $a \in \{0, \ldots, Q-1\}$ has a unique representation $\sum_{j<d} a_j q^j$ for $a_j \in \{0, \ldots, q-1\}$. Then

$$\pi^{-1}_{(\bar{x},y)}(a) = \sum_{j<d} \pi^{-1}_{\bar{x}}(a) y^j \tag{3.9}$$

Claim. The maximum element of the tuple $\rho_k(\bar{x}, y)$ is $Q-1$.

Proof. By induction, the maximum element of $\rho_{k-1}(\bar{x})$ is $q-1$. Then $d \leq Q-1$,

since $q^d = Q$ and $q > 1$. Finally, $r \leq Q - 1$, because r is in the range of $\pi_{(\bar{x},y)}$, which is a bijection onto $\{0, \ldots, Q-1\}$. □

3.2.2 The functions of the bi-interpretation

Having defined ρ and $\pi_{\bar{x}}$ for $\bar{x} \in \bar{\mathbb{F}}_p^k$, we now have to show that the image under $\pi_{\bar{x}}$ of any primitive in $\mathbb{F}_p(\bar{x})$ is computable uniformly by a tail recursion over $\langle \rho_k(\bar{x}) \rangle^\star$, and similarly that the pre-image of any primitive of $\langle \rho_k(\bar{x}) \rangle^\star$ is computable uniformly by a tail recursion over $\mathbb{F}_p(\bar{x})$.

The case $k = 0$

If $k = 0$, each family contains only one structure ($\mathbb{F}_p$ and $\langle p-1 \rangle^\star$ respectively). *Any* function on either of these structures is recursive (in fact, explicit) in that structure, so we're done. So we may assume $k > 0$.

As above, write $(\bar{x}, y)$ for a typical element of $\bar{\mathbb{F}}_p^k$, with $\bar{x} \in \bar{\mathbb{F}}_p^{k-1}$. We continue to use the variables d, r, q, and Q (see definition 3.2.1) which are functions of $(\bar{x}, y)$.

In 5.2.2, for brevity write ρ for ρ_k and π for $\pi_{(\bar{x},y)}$.

3.2.2.1 Computing the field primitives in arithmetic

Before computing the primitives of $\mathbb{F}_p(\bar{x}, y)$ uniformly over $\langle \rho_k(\bar{x}, y) \rangle^\star$, we introduce several auxiliary functions, all of which are uniformly linear recursive, and hence tail recursive, over $\langle \rho(\bar{x}, y) \rangle^\star$. The programs computing these functions take advantage of the fact that each element$n < Q$ has a unique representation $\sum_{j<d} n_j q^j$ for $u_j \in \{0, \ldots, q-1\}$. If $d = 1$, this representation is simply n. Often times these programs will split into two cases, depending on whether or not $d = 1$. Since d in contained in the tuple $\rho(\bar{x}, y)$, this is simply an explicit test.

Since the tuple $\rho(\bar{x}, y)$ contains both $q-1$ and $Q-1$, we can compute the constant q uniformly over $\langle \rho_k(\bar{x}, y)\rangle^\star$ in the case that $d > 1$. We will also frequently perform arithmetic mod Q, which is uniformly tail recursive over $\langle \rho(\bar{x}, y)\rangle^\star$.

Functions h, t that satisfy $\pi \circ \mathfrak{h} = h \circ \pi$ and $\pi \circ \mathfrak{t} = t \circ \pi$.

Define $h(n) = \pi(\mathfrak{h}(\pi^{-1}(n)))$. By equations(3.7), (3.4), and (3.9), it's easy to see that

$$h : \sum_{j<d} n_j q^j \mapsto n_0$$

Similarly define $t(n) = \pi(\mathfrak{t}(\pi^{-1}(n)))$. Then

$$t : \sum_{j<d} n_j q^j \mapsto \sum_{j<d-1} n_{j+1} q^j$$

In the case that $d = 1$, h is the identity and t is the constant 0. If $d > 1$, then $h(n)$ is simply the remainder and $t(n)$ is the quotient of n upon division by q, both of which are computable over $\mathbf{N}_{Pd}$ given q.

A function *app* that satisfies $n = app(h(n), t(n))$.

If $d = 1$, then $app(n, m) = n$. If $d > 1$, let *app* be defined by

$$app(n, m) = n + qm \mod Q$$

Addition

Define e^+ by

$$\pi(u + v) = e^+(\pi u, \pi v)$$

for $u, v \in \mathbb{F}_p(\bar{x}, y)$. Notice that

$$\sum_{i<d} u_i y^i + \sum_{i<d} v_i y^i = \sum_{i<d} (u_i + v_i) y^i$$

By equations (3.7) and (3.9) we see that

$$e^+(\sum_{i<d} n_i q^i, \sum_{i<d} m_i q^i) = \sum_{i<d} e^+_{k-1}(n_i, m_i) q^i$$

where e^+_{k-1} is the function that satisfies $\pi_{\bar{x}}(u+v) = e^+_{k-1}(\pi u, \pi v)$ for $u, v \in \mathbb{F}_p(\bar{x})$ and is linear recursive by induction.

We compute e^+ by:

- If $n, m \in \mathbb{F}_p(\bar{x})$, simply computing $e^+_{k-1}(n, m)$.
- Otherwise, computing $A = e^+_{k-1}(h(n), h(m))$ and recursively computing $B = e^+(t(n), t(n))$.
- Letting $e^+(n, m) = app(A, B)$.

We can make a similar construction to compute e^-.

Multiplication

Define $e^\times$ by

$$\pi(uv) = e^\times(\pi u, \pi v)$$

for each $u, v \in \mathbb{F}_p(\bar{x}, y)$.

If $d = 1$, then $e^\times$ is simply $e^\times_{k-1}$, which we already know how to calculate by induction.

Otherwise, notice that, for $u, v \in \mathbb{F}_p(\bar{x}, y)$,

$$u = \mathfrak{h}(u) + y \cdot \mathfrak{t}(u) \implies uv = \mathfrak{h}(u)v + y \cdot \mathfrak{t}(u)v$$

We can compute $e^\times$ given a "limited multiplication" function $\tilde{e}$ which satisfies

$$\pi(uv) = \tilde{e}(\pi u, \pi v) \tag{3.10}$$

for all $u \in \mathbb{F}_p(\bar{x})$, and a "shift function" S which satisfies

$$\pi(yu) = S(\pi u) \tag{3.11}$$

if $d > 1$.

It is straightforward to verify that

$$\pi(uv) = e^+(\tilde{e}(h(\pi u), \pi v), S(e^\times(t(\pi u), \pi v)))$$

Hence we can compute $e^\times$ by the linear recursion

$$e^\times(n,m) = \begin{cases} \tilde{e}(n,m) & \text{if } n < q \\ e^+(\tilde{e}(h(n),m), S(e^\times(t(n),m))) & \text{otherwise} \end{cases}$$

Limited multiplication Notice that if $u \in \mathbb{F}_p(\bar{x})$,

$$u \sum_{i<d} v_i y^i = \sum_{i<d} (uv_i) y^i$$

Now $u \in \mathbb{F}_p(\bar{x}) \iff \pi(u) < q$. Hence, for $n < q$, we want to define

$$\tilde{e}(n, \sum_{i<d} m_i q^i) = \sum_{i<d} e^\times_{k-1}(u, v_i) q^i$$

Thus $\tilde{e}$ may be computed by:

- If $m < q$, simply compute $e^\times_{k-1}(n,m)$, otherwise,
- Compute $A = e^\times_{k-1}(n, h(m))$ and $B = \tilde{e}(n, t(m))$ by recursion

- Let $\tilde{e}(n,m) = app(A,B)$

Shift Assuming $d > 1$, $q^{d-1} < Q$, and we can compute q^{d-1} uniformly by a tail recursion. Define

$$S(n) = e^+(\tilde{e}(n/q^{d-1}), e^-(0,r)), (q \cdot (n \mathrm{mod} q^{d-1}) \mathrm{mod} Q))$$

Claim 3.2.5. For $u \in \mathbb{F}_p(\bar{x}, y)$, if $d > 1$ $\pi(yu) = S(\pi(u))$

Proof. The proof is again purely formal, but long. Suppose $u \in \mathbb{F}_p(\bar{x}, y)$ and $n = \pi(u)$, so that

$$u = \sum_{i<d} u_i y^i \quad n = \sum_{i<d} n_i q^i$$

where $n_i = \pi_{\bar{x}}(u_i)$. Then

$$yu = u_{d-1} y^d + \sum_{1<i<d} u_{i-1} y^i$$

We prove that $\pi(yu) = S(\pi(u))$ by parsing the expression for S:

1. $\pi(yu) = e^+(\pi(u_{d-1}y^d), \pi(\sum_{1<i<d} u_{i-1}y^i))$
   (a) $\pi(u_{d-1}y^d) = \tilde{e}(\pi(u_{d-1}), \pi(y^d))$, since $u_{d-1} \in \mathbb{F}_p(\bar{x})$
      i. $\pi(u_{d-1}) = n_{d-1}$ is the integer quotient when n is divided by q^{d-1}
      ii. $\pi(y^d) = \pi(0 - (-y^d)) = e^-(\pi(0), \pi(-y^d)) = e^-(0,r)$
   (b) $\pi(\sum_{1<i<d} u_{i-1}y^i) = q \cdot (\sum_{0<i<d-1} n_i q^i)$. Since the product is less than Q,

$$\pi(\sum_{1<i<d} u_{i-1}y^i) = q \cdot (\sum_{0<i<d-1} n_i q^i) \mod Q$$

      i. $\sum_{0<i<d-1} n_i q^i = n \mod q^{d-1}$, the remainder when n is divided by q^{d-1}.

□

Division

For $u, v \in \mathbb{F}_p(\bar{x}, y)$, define $e^{\div}$ by

$$e^{\div}(\pi u, \pi v) = \begin{cases} \pi(u/v) & \text{if } v \neq 0 \\ \uparrow & \text{otherwise} \end{cases}$$

To calculate $e^{\div}(n, m)$ just search for the least $r < Q$ such that $e^{\times}(r, m) = n$. If $v \neq 0$, then $e^{\div}(\pi(u), \pi(v))$ will be $\pi(u/v)$, for $\pi(u/v)$ is the unique r such that $e^{\times}(r, \pi(v)) = \pi(u)$. Otherwise there will be no such r, and we can force the computation to diverge.

For $1 \leq i \leq k$, constants e^{x_i} so that $\pi(x_i) = e^{x_i}$

By induction we have constants $e^{x_i}_{k-1}$ such that

$$\pi_{\bar{x}}(x_i) = e^{x_i}_{k-1}$$

Simply let e^{x_i} be $e^{x_i}_{k-1}$. Then since π extends $\pi_{\bar{x}}$,

$$\pi(x_i) = e^{x_i}$$

A constant e^y such that $\pi(y) = e^y$

If $d = 1$, then $\pi(y) = \pi(y^d)$, but $\pi(y^d) = e^-(0, r)$. Otherwise if $d > 1$, then $\pi(y) = q$.

3.2.2.2 Computing arithmetical primitives in the field

For each of the primitives ψ in the structure $\langle \rho(\bar{x}, y) \rangle^\star$, we define a function e^ψ uniformly linear recursive on structures $\mathbb{F}_p(\bar{x}, y)$ such that $\pi \circ e^\psi \simeq \psi \circ \pi$. Recall

that the primitives of $\langle \rho(\bar{x}, y)\rangle^\star$ are Pd along with the constants in $\rho(\bar{x}, y)$

Predecessor e^{Pd}

For any $u \in \mathbb{F}_p(\bar{x}, y)$, we want

$$\pi(e^{Pd}(u)) = Pd(\pi(u))$$

Suppose $\pi(u) = n$, and

$$u = \sum_{i<d} u_i y^i \quad n = \sum_{i<d} n_i q^i$$

where $n_i = \pi_{\bar{x}}(u_i)$.

Notice that Pd satisfies

$$Pd(n) = \begin{cases} Pd(h(n)) + q \cdot t(n) & \text{if } h(n) \neq 0 \\ (q-1) + q \cdot Pd(t(n)) & \text{if } h(n) = 0 \ \& \ t(n) \neq 0 \\ 0 & \text{if } n = 0 \end{cases}$$

Therefore let e^{Pd} be defined by the linear recursion

$$e^{Pd}(u) = \begin{cases} e^{Pd}_{k-1}(\mathfrak{h}(u)) + y \cdot \mathfrak{t}(u) & \text{if } \mathfrak{h}(u) \neq 0 \\ e^{q-1} + y \cdot e^{Pd}(\mathfrak{t}(u)) & \text{if } \mathfrak{h}(u) = 0 \ \& \ \mathfrak{t}(u) \neq 0 \\ 0 & \text{if } u = 0 \end{cases}$$

The constant e^{q-1} satisfies $\pi(e^{q-1}) = q - 1$, and is defined immediately below. Its definition does not depend on e^{Pd}. By induction, the function e^{Pd}_{k-1} correctly computes the preimage of Pd on $\mathbb{F}_p(\bar{x})$. It is straightforward to show that $\pi(e^{Pd}(u)) = Pd(\pi(u))$.

Constants

Recall by 3.8 that $\rho(\bar{x}, y) = (\rho(\bar{x}), d, r, Q-1)$. For constants c in $\rho(\bar{x})$, we have by induction tail recursive family of constants e^c_{k-1} on $\mathbb{F}_p(\bar{x})$ such that

$$\pi_{\bar{x}}(e^c_{k-1}) = c$$

For such c, simply let $e^c = e^c_{k-1}$. Then since π extends $\pi_{\bar{x}}$,

$$\pi(e^c) = c$$

which is what we want. In particular, we have e^{q-1}.

Finally, we want a tail recursive family of constants e^d, e^r, and e^{Q-1} such that $\pi(e^d) = d$, $\pi(e^r) = r$, and $\pi(e^{Q-1}) = Q-1$.

- Since $Q = q^d$

  $$Q - 1 = \sum_{i<d} (q-1)q^i$$

  Hence let e^{Q-1} be defined by

  $$e^{Q-1} = \sum_{i<d} e^{q-1} y^i$$

  We can use Horner's method to calculate this sum. We can stop after d steps by applying a generator σ_k of the Galois group $\mathrm{Aut}(\mathbb{F}_p(\bar{x}, y)/\mathbb{F}_p(\bar{x}))$ to y and stopping when we recover y. (see Sec. 4.2.1 for how to compute σ_k)

- Using Pd, we can compute a "truncated successor" S on $\langle \rho(\bar{x}, y) \rangle^\star$, which computes $n \mapsto n+1$ for $n < Q-1$ and fixes $Q-1$. Similarly, we can

compute e^S such that for $u \neq Q-1$

$$\pi(e^S(u)) = \pi(u) + 1$$

and

$$\pi(e^S(e^{Q-1})) = e^{Q-1}$$

- Clearly, $\sum_{i<d} 1 = d$. Therefore, we can compute e^d by starting with 0 and applying e^S d times. We measure out d, again by applying σ_k to y until we recover y.
- We know that $\pi(-y^d) = r$. Hence let $e^r = -y^d$. We compute this by multiplying y by itself d times, measuring out d in the same way.

3.2.3 Denouement

In this section we prove that

$$\mathrm{rec}(\bar{\mathbb{F}}_p) = \mathrm{tail}(\bar{\mathbb{F}}_p) \iff \mathrm{rec}(\mathbf{N}_{Pd}) = \mathrm{tail}(\mathbf{N}_{Pd})$$

The $\Longleftarrow$ direction is immediate from theorem 2.1.13. If there is a recursive not tail recursive partial function f over $\bar{\mathbb{F}}_p$ of arity k, then there is a recursive partial function g over $\mathbf{N}_{Pd}$ of arity $3k+1$ such that $\pi_{\bar{x}}(f(\bar{x})) = g(\rho_k(\bar{x}))$, and g cannot be tail recursive.

It's the $\Longrightarrow$ direction which is not so obvious. If there is a recursive not tail recursive partial function over $\mathbf{N}_{Pd}$, then it could have arity not 1 mod 3, or it could agree with some tail recursive function on elements in the range of ρ_k. In either of these cases, theorem 2.1.13 does not allow us to pull back to a recursive not tail recursive f over $\bar{\mathbb{F}}_p$.

The prime number theorem for finite fields

The results in this sections are mostly routine applications of previous general theorems. However, in the proof of lemma 3.2.8 we are faced with the problem of showing that if we code elements of $\mathbb{F}_p[t]$ by numbers base p in the obvious way, the n-th largest number encoding a monic irreducible polynomial is bounded by a polynomial in n. This is ensured by the *prime number theorem over finite fields*, which counts the number of "primes" (i.e., codes of monic irreducible polynomials in $\mathbb{F}_p[t]$) less than a given number.

While the ordinary prime number theorem says the number of prime numbers less than N looks asymptotically like $N/\log N$, the prime number theorem for finite fields says that the number of codes of monic irreducible polynomials looks asymptotically like $N/\log_p N$. See [13].

Lemma 3.2.6. *For any n and k, suppose there are functions $g : \mathbb{N} \to \mathbb{N}^4$ and $h : \mathbb{N}^4 \to \mathbb{N}$ computable in polynomial parameters by a tail recursion in unary arithmetic such that $h(g(n)) = n$. Suppose that $g[\mathbb{N}] \subseteq \rho_1[\bar{\mathbb{F}}_p]$.*

Then if there's a recursive not tail recursive partial function over $\mathbf{N}_{Pd}$, there's a recursive non-tail-recursive subset of $\bar{\mathbb{F}}_p$.

Proof. By corollary 2.3.16, we know there's a recursive not tail recursive partial function over $\mathbf{N}_{Pd}$ iff there's such a total predicate.

Let $Y \subseteq \mathbb{N}$ be recursive not tail recursive over $\mathbf{N}_{Pd}$. Then $Z := Y \circ h$ is recursive and not tail recursive by Corollary 2.3.5. Define

$$X := \{x \in \bar{\mathbb{F}}_p : \rho_1(x) \in Z\}$$

so that$\rho_1[X] = Z \cap \rho_1[\bar{\mathbb{F}}_p]$.

By corollary 2.1.14, X is $\bar{\mathbb{F}}_p$-recursive. If X were $\bar{\mathbb{F}}_p$-tail recursive, then again

by Corollary 2.1.14 there would be a tail recursive Z' such that

$$Z \cap \rho_1[\bar{\mathbb{F}}_p] = Z' \cap \rho_1[\bar{\mathbb{F}}_p]$$

For any set $S \subseteq \mathbb{N}^4$, $g[\mathbb{N}] \subseteq \rho_1[\bar{\mathbb{F}}_p]$ implies

$$(S \cap \rho_1[\bar{\mathbb{F}}_p]) \circ g = S \circ g$$

as subsets of $\mathbb{N}$.

Hence

$$Y = Z \circ g = Z' \circ g$$

but then by Corollary 2.3.5, Y is tail recursive, contradiction. □

Finding g and h

The idea is for g to enumerate the range of ρ_1, and h to invert g. From the equation 3.8 it's easy to see that

$$\rho_1(x) = (p-1, d, f(p), p^d - 1)$$

where d is the degree of x over $\mathbb{F}_p$ and f is the minimal polynomial of x in $\mathbb{F}_p[t]$. (In writing $f(p)$ we slightly abuse notation: if we identify the domain of $\mathbb{F}_p$ with $\{0, 1, \ldots, p-1\}$, then we can evaluate f on integers.)

If f is a monic irreducible polynomial, we can recover d and $p^d - 1$ from $f(p)$. Namely, $d = \lfloor \log_p(f(p)) \rfloor$ base p, and p^d is the largest power of p less than or equal to $f(p)$. Hence the range of ρ_1 is

$$\{(p-1, \alpha(n), n, \beta(n)) : n \text{ encodes a monic irreducible polynomial}\} \qquad (3.12)$$

where $\alpha(n) = \lfloor \log_p(n) \rfloor$ and $\beta(n)$ is the largest power of p less than or equal to

n. Both α and β are computable over $\mathbf{N}_{Pd}$ by a tail recursion.

Definition 3.2.7. Let g be the function (g_1, g_2, g_3, g_4) where:

1. $g_1(n) = p - 1$
2. $g_2(n) = \alpha(g^\star(n))$
3. $g_3(n) = g^\star(n)$
4. $g_4(n) = \beta(g^\star(n))$

where $g^\star(n)$ is the code of n-th monic irreducible polynomial.

By equation (3.12), g enumerates the range of ρ_1. If $g^\star$ is computable in polynomial parameters by a tail recursion, so is g.

Lemma 3.2.8. *$g^\star$ is computable by a tail recursion that runs in polynomial parameters*

Proof. Let $P(n)$ denote whether n encodes a monic irreducible polynomial. This is computable in polynomial parameters. Since $g^\star(n)$ is the n-th least monic irreducible polynomial, it is computed by the tail recursion $g(n) = \gamma(n, 0)$ where

$$\gamma(n, b) = \begin{cases} b - 1 & \text{if } n = 0 \\ \gamma(n - 1, b + 1) & \text{else, if } P(b) \\ \gamma(n, b + 1) & \text{otherwise} \end{cases}$$

(The algorithm is simple: we just march up $\mathbb{N}$, ticking off numbers b such that $P(b)$, until we have found the n-th number)

It's easy to see that this program runs in polynomial parameters iff $g^\star(n)$ is bounded by a polynomial. To prove this, we use the prime number theorem over finite fields.

Following [13], let $\pi_p(N)$ be the number of codes of monic irreducible polynomials less than or equal to N. Then $\pi_p(N) \sim N/\log_p(N)$, where $\sim$ means that their quotient tends to 1 as $N \to \infty$. In particular, there exists N_0 such that for $N \geq N_0$, $\sqrt{N} \leq \pi_p(N)$, and $N \leq (\pi_p(n))^2$.

For all N, $N \leq (\pi_p(n))^2 + N_0$.

Since $g^\star(n)$ encodes the n-th monic irreducible polynomial, $\pi_p(g^\star(n)) = n$. Hence

$$g^\star(n) \leq n^2 + N_0$$

which finishes the proof. □

Lastly, we define h and show that it is computable by a tail recursion in polynomial parameters.

Definition 3.2.9. We define h' and h

$$h'(m) = \mu n \; g^\star(n) = m$$

Then (since g is clearly injective) $h'(g(n)) = n$, and since $g^\star$ is computable by a tail recursion in polynomial parameters, so is h'.

Finally, let $h(n_1, n_2, n_3, n_4) = h'(n_3)$, so that $h(g(n)) = h'(g^\star(n)) = n$. This completes the proof of 3.1 on page 61.

3.3 Recursion versus tail recursion for abelian groups

In this section we show that all families of functions uniformly recursive over finite abelian groups are uniformly tail recursive iff $\mathrm{rec}(\mathbf{N}_{Pd}) = \mathrm{tail}(\mathbf{N}_{Pd})$. Unlike in the case for finite fields, we deal with the uniform recursion theory of

finite abelian groups instead of the recursion theory of some infinite locally finite abelian group, because there is no canonical choice for such a group, like $\bar{\mathbb{F}}_p$ is for finite fields. (This is mostly a matter of taste; our results apply to any locally finite abelian group containing all finite abelian groups as subgroups.)

Let $\mathbf{M}$ be any locally finite abelian group that contains all finite abelian groups as subgroups. Let k be a fixed natural number. For $\bar{x} \in M^k$, define

$$\mathbf{A}_{\bar{x}} := (\langle \bar{x} \rangle_{\mathbf{M}}, \bar{x})$$

so that the family

$$\{\mathbf{A}_{\bar{x}} : \bar{x} \in M^k\}$$

is the family of all finite abelian groups with k generators along with constants for the generators. Notice that the definition of this family is completely independent of the specific choice for $\mathbf{M}$.

We shall construct a uniform family of bi-interpretations between $\{\mathbf{A}_{\bar{x}}\}$ and a certain family of substructures of $\mathbf{N}_{Pd}$.

Non-pointed structures Note that each of these structures has an element 0 but no 1. Therefore for the recursion theory, we add two separate constants to each structure, named *true* and *false*. For the purposes of this section, we assume that $\mathbf{N}_{Pd}$ has no constant denoting 1. This is not essential but makes treatment of the degenerate case of the uniform family of bi-interpretations a little more elegant.

Structured programming We augment register programs with if-else statements, while loops, and do-while loops to obtain *structured programs*. The fact that structured programs add no computational power to register programs is a fundamental fact of programming language theory, see, e.g. chapter IV in

[5]. We assume familiarity with structured programs and do not define them or their semantics.

3.3.1 Ordering abelian groups relative to their generators

We define an ordering on the elements of $A_{\bar{x}}$ and extend it to $A_{\bar{x}}^k$ lexicographically. Then we show that we can "iterate through" elements and k-tuples of $A_{\bar{x}}$ using structured programs.

Definition 3.3.1. (The ordering $\prec_{\bar{x}}$ on $A_{\bar{x}}$) For each $a \in A_{\bar{x}}$, there's a tuple $(n_1, \ldots, n_k) \in \mathbb{N}^k$ such that $n_1 x_1 + \cdots + n_k x_k = a$ and $0 \leq n_i < o_i$ where o_i is the order of x_i. Order such tuples lexicographically where n_1 is the least significant digit. Then define

$$a \prec_{\bar{x}} b$$

if the least tuple $\bar{n}$ such that $\sum n_i x_i = a$ is less than the least tuple $\bar{m}$ such that $\sum m_i x_i = b$.

Definition 3.3.2. (The ordering $\prec_{\bar{x}}^k$ on $A_{\bar{x}}^k$) We extend $\prec_{\bar{x}}$ lexicographically to k-tuples in $A_{\bar{x}}$, least significant digit first, to obtain $\prec_{\bar{x}}^k$.

Iterating through the elements of $\mathbf{A}_{\bar{x}}$ Suppose P is a structured program in the language of abelian groups with constants $(x_1, \ldots, x_k)$. Then let $\delta_{X,i}(P)$ denote the structured program

$X \leftarrow 0$

`do`

  P

  $X \leftarrow X + x_i$

`while`$(X \neq 0)$

Suppose the program P does not modify the register X, i.e., X does not appear on the left hand side of any assignment statement in P. Then (informally speaking), the program $\delta_{X,i}(P)$ executes P o_i times, where o_i is the order of x_i in $\mathbf{A}_{\bar{x}}$.

Now consider the operation on structured programs

$$\Delta_{X_1,\ldots,X_k,Y}(P) := \delta_{X_1,1} \circ \delta_{X_2,2} \circ \cdots \circ \delta_{X_k,k}(P^{(Y)})$$

where $P^{(Y)}$ is the line $Y \leftarrow X_1 + \cdots + X_k$ followed by P. Then Δ situates a given program P inside k nested do-while loops.

We state the following (easy) lemma without proof. It states that k nested loops behave as we would expect them to.

Lemma 3.3.3. *Suppose P does not modify the registers $X_1, \ldots, X_k, Y$. For $\bar{x} \in M^k$ let o_i be the order of x_i for $1 \leq i \leq k$ and let $[o_i] := \{n \in \mathbb{N} : n < o_i\}$. Define the bijection*

$$\beta_{\bar{x}} : \prod_{1 \leq i \leq k} [o_i] \to [\prod_{1 \leq i \leq k} o_i]$$

$$\beta_{\bar{x}} : (n_1, \ldots, n_k) \mapsto \sum_{1 \leq i \leq k} n_i \prod_{0 \leq j < i} o_j$$

Consider the execution of the program $\Delta_{X_1,\ldots,X_k,Y}(P)$ on $\mathbf{A}_{\bar{x}}$. The block of code P executes $\prod_{1 \leq i \leq k} o_i$ times, and on the $\beta_{\bar{x}}(n_1, \ldots, n_k)$-th time, the value in the register X_i is $n_i x_i$ for $1 \leq i \leq k$ and $0 \leq n_i < o_i$. Hence the value in the register Y is $n_1 x_1 + \cdots + n_k x_k$, which ranges over all elements in $A_{\bar{x}}$ during the execution of P.

With this lemma it's easy to show:

Theorem 3.3.4. *The following functions and relations are uniformly tail recursive over the indicated families of structures:*

- *$E(\bar{a}, b) \iff b \in \langle \bar{a} \rangle_{\mathbf{A}}$ over all finite abelian groups $\mathbf{A}$.*

- *$a \prec_{\bar{x}} b$ over all structures $\mathbf{A}_{\bar{x}}$.*
- *The $\prec_{\bar{x}}$-successor function $\sigma_{\bar{x}}$ which sends an element a to its $\prec_{\bar{x}}$-least upper bound and the $\prec_{\bar{x}}$-maximal element to 0, over all structures $\mathbf{A}_{\bar{x}}$.*
- *The $\prec_{\bar{x}}$-predecessor function (similarly defined, fixed 0).*

Proof. (Idea) For the first function, iterate over elements generated by $\bar{a}$, checking for each whether it's equal to b.

For the next two functions, notice that for $\bar{n}, \bar{m} \in \prod[o_i]$, $\bar{n} \prec \bar{m}$ lexicographically with the least significant bit first iff $\beta_{\bar{x}}(\bar{n}) < \beta_{\bar{x}}(\bar{m})$. Therefore, the register Y iterates through elements of $A_{\bar{x}}$ exactly in $\prec_{\bar{x}}$-order.

Lastly, the predecessor may be computed from the successor. □

Iterating through k-tuples of $\mathbf{A}_{\bar{x}}$ We can mimic the process above to iterate through tuples in $A^k_{\bar{x}}$.

Let $\gamma_X(P)$ denote the structured program

$X \leftarrow 0$
`do`
  P
  $X \leftarrow \sigma_{\bar{x}}(X)$
`while`$(X \neq 0)$

and let

$$\Gamma_{X_1,\ldots,X_k} := \gamma_{X_1} \circ \cdots \circ \gamma_{X_k}$$

Then we have:

Lemma 3.3.5. *Suppose P does not modify registers $X_1, \ldots, X_k$. The the value of $(X_1, \ldots, X_k)$ inside $\Gamma_{X_1,\ldots,X_k}(P)$ ranges over all tuples in $A^k_{\bar{x}}$ in the order*

given by $\prec^k_{\bar{x}}$.

3.3.2 Quasi-bases

Definition 3.3.6. Let $\mathbf{A}$ be a finite abelian group. We say that $\bar{x} \in A^k$ is a *quasi-basis* in case

- $\bar{x}$ generates $\mathbf{A}$, i.e. $A \subseteq \mathbb{Z}x_1 + \cdots + \mathbb{Z}x_k$
- for any $n_1, \ldots, n_k \in \mathbb{Z}$, if $n_1x_1 + \cdots + n_kx_k = 0$ then $n_ix_i = 0$ for all $1 \leq i \leq k$.

Equivalently, $\bar{x}$ is a quasi-basis in case the map

$$\langle x_1 \rangle \oplus \cdots \oplus \langle x_k \rangle \to \mathbf{A}$$

$$x_i \mapsto x_i$$

is an isomorphism.

Quasi-bases do not necessarily exclude 0. Indeed, for any quasi-basis $\bar{x}$ of $\mathbf{A}$, $(\bar{x}, 0)$ is another quasi-basis.

For the finite abelian group $\mathbf{A}_{\bar{x}}$, $\bar{x}$ may not be a quasi-basis; however there will always be a quasi-basis of exactly k elements computable from $\bar{x}$. This is what we now show.

Lemma 3.3.7. *It is uniformly tail recursive over* $\mathbf{A}_{\bar{x}}$ *to compute whether* $\bar{x}$ *forms a quasi-basis of* $\mathbf{A}_{\bar{x}}$

Proof. Clearly $\bar{x}$ generates $\mathbf{A}_{\bar{x}}$. We have to check that $\sum n_ix_i = 0 \implies \forall i \leq k\ n_ix_i = 0$. Then the program

$\Delta_{X_1,\ldots,X_k,Y}(P)$

```
halt and return true
```

where P is the program

```
if(Y = 0 & ∃i ≤ k X_i ≠ 0)
  halt and return false
```

computes whether or not $\bar{x}$ is a quasi-basis of $\mathbf{A}_{\bar{x}}$. □

Theorem 3.3.8. *There exists a quasi-basis $\bar{y}$ of $\mathbf{A}_{\bar{x}}$ of length k*

Proof. This follows directly from the statement that an abelian group with k generators may be expressed as the direct sum of at most k cyclic groups, which is a restatement of the structure theorem for finite abelian groups. □

Theorem 3.3.9. *The $\prec^k_{\bar{x}}$-least quasi-basis $\bar{q}_{\bar{x}}$ of length k is a uniformly tail recursive k-tuple of constants over $\mathbf{A}_{\bar{x}}$.*

Proof. To compute the least quasi-basis of length k, we iterate over all k-tuples from $A_{\bar{x}}$ in order $\prec^k_{\bar{x}}$ by lemma 3.3.5. For each such tuple $\bar{y}$ we check whether (1) $\bar{y}$ forms a quasi basis of $\mathbf{A}_{\bar{y}} \subseteq \mathbf{A}_{\bar{x}}$, by 3.3.7, and (2) whether each $x_i \in \langle \bar{y} \rangle$ by theorem 3.3.4. □

3.3.3 A uniform family of bi-interpretations

Let $\bar{z}_{\bar{x}} = (z_1, \ldots, z_k)$ be the $\prec^k_{\bar{x}}$-least quasi-basis of $\mathbf{A}_{\bar{x}}$. Suppose z_i has order d_i in $\mathbf{A}_{\bar{x}}$. Then

$$\mathbf{A}_{\bar{x}} \simeq \langle z_1 \rangle \oplus \cdots \oplus \langle z_k \rangle$$

and each $a \in A_{\bar{x}}$ has a unique representation $\sum n_i q_i$ for $0 \leq n_i < d_i$. Define $D_j := \prod_{i \leq j} d_i$ for $1 \leq j \leq k$ and $D_0 = 1$. Define the map $\pi_{\bar{x}}$ by

$$\pi_{\bar{x}} : A_{\bar{x}} \to [D_k]$$

$$\pi_{\bar{x}} : \sum_{1 \leq i \leq k} n_i z_i \mapsto \sum_{1 \leq i \leq k} n_i D_{i-1}$$

Define the structure

$$\mathbf{B}_{\bar{x}} := (\mathbf{N}_{Pd} \restriction D_k, D_1 - 1, \ldots, D_k - 1, \pi_{\bar{x}}(x_1), \ldots, \pi_{\bar{x}}(x_k))$$

for $\bar{x} \in M^k$. We give a uniform family of bi-intepretations of the family $\{\mathbf{A}_{\bar{x}}\}$ with the family $\{\mathbf{B}_{\bar{x}}\}$. The family of bijections is of course $\pi_{\bar{x}} : A_{\bar{x}} \to B_{\bar{x}}$.

We assume that $k > 0$ and there is some $x_i \neq 0$. Otherwise $A_{\bar{x}} = B_{\bar{x}} = \{0\}$.[1]

3.3.3.1 Images of the primitives of $\mathbf{A}_{\bar{x}}$

The primitives of $\mathbf{A}_{\bar{x}}$ are (1) the constant 0, (2) addition, (3) negation, (4) equality, and (4) the x_i for $1 \leq i \leq k$. The image of the constant 0 is of course 0, the image of equality is equality, and the image of x_i is $\pi_{\bar{x}}(x_i)$, all of which are trivially uniformly tail recursive over $\mathbf{B}_{\bar{x}}$.

The image of addition is

$$(\sum_{1 \leq i \leq k} n_i D_{i-1}, \sum_{1 \leq i \leq k} m_i D_{i-1}) \mapsto \sum_{1 \leq i \leq k} (n_i \oplus m_i) D_{i-1}$$

where $n_i \oplus m_i$ is addition mod d_i. The image of negation is

$$\sum_{1 \leq i \leq k} n_i D_{i-1} \mapsto \sum_{1 \leq i \leq k} (\ominus n_i) D_{i-1}$$

[1] This is why we assume $\mathbf{N}_{Pd}$ does not contain the constant 1.

where $\ominus n_i$ is negation mod d_i. These are both easily uniformly tail recursive over $\mathbf{B}_{\bar{x}}$ once we show we can compute each d_i uniformly. But $d_i = F(D_i - 1, D_{i-1} - 1)$, where

$$F(x, y) = \frac{x+1}{y+1}$$

which is $\mathbf{N}_{Pd}$-tail recursive for $x \geq y$.

3.3.3.2 Pre-images of the primitives of $\mathbf{B}_{\bar{x}}$

The primitives of $\mathbf{B}_{\bar{x}}$ are (1) the constant 0, (2) predecessor, (3) equality, (4) the constants $D_i - 1$, and (5) the constants $\pi_{\bar{x}}(x_i)$.

The pre-image of 0 is 0 and the pre-image of $\pi_{\bar{x}}(x_i)$ is x_i, which are uniformly explicit over $\mathbf{A}_{\bar{x}}$. The pre-image of equality is equality, which is a primitive.

The only nontrivial case is computing the pre-image of the predecessor function. However, notice that, since $\bar{z}$ is a quasi-basis for $\mathbf{A}_{\bar{x}}$, $A_{\bar{z}} = A_{\bar{x}}$, and thus $\prec_{\bar{z}}$ orders $A_{\bar{x}}$. Then I claim

Lemma 3.3.10. *The order $(A_{\bar{x}}, \prec_{\bar{z}})$ is exactly the pre-image under $\pi_{\bar{x}}$ of the order $(B_{\bar{x}}, <)$.*

Proof. For $a, b \in A_{\bar{x}}$, $a \prec_{\bar{z}} b$ is defined to hold exactly when $\bar{n} \prec \bar{m}$, where $\bar{n}$ and $\bar{m}$ are now the *unique* tuples such that $\sum n_i z_i = a$ and $\sum m_i z_i = b$, and $\prec$ is the lexicographical ordering on $\mathbb{N}^k$ with the least significant digit first. But then

$$\bar{n} \prec \bar{m} \iff \sum n_i D_{i-1} < \sum m_i D_{i-1} \iff \pi_{\bar{x}}(a) < \pi_{\bar{x}}(b)$$

□

By theorem 3.3.4, we can compute the predecessor function of $\prec_{\bar{z}}$ (which is the pre-image of Pd) uniformly over $\mathbf{A}_{\bar{z}}$, whose domain is $A_{\bar{x}}$, with a tail recursion. By theorem 3.3.9 we can compute $\bar{z}$ uniformly over $\mathbf{A}_{\bar{x}}$ with a tail

recursion. It follows that we can compute the pre-image of Pd uniformly over $\mathbf{A}_{\bar{x}}$ with a tail recursion.

Finally, we must compute the pre-images of the $D_i - 1$ for $1 \leq i \leq k$. But the pre-image of $D_i - 1$ is the $\prec_{\bar{z}'}$-maximal element of $A_{\bar{z}'} \subseteq A_{\bar{x}}$, where $\bar{z}' = (z_1, \ldots, z_i)$. This is uniformly computable over $\mathbf{A}_{\bar{z}}$, and hence over $\mathbf{A}_{\bar{x}}$, by a tail recursion.

Now we have finished with the construction of the uniform family of biintepretations.

3.3.4 Denouement

We now finish the proof of

Theorem 3.3.11. *Let$\{\mathbf{A}_i\}$ be the family m of all abelian groups. Then*

$$rec(\{\mathbf{A}_i\}) = tail(\{\mathbf{A}_i\}) \iff rec(\mathbf{N}_{Pd}) = tail(\mathbf{N}_{Pd})$$

By corollary 2.1.12, we have that there is a recursive not tail recursive family of partial constants over $\mathbf{A}_{\bar{x}}$ iff there is a recursive not tail recursive family of partial constants over $\mathbf{B}_{\bar{x}}$. It is easy to see that

Lemma 3.3.12. *There is a recursive not tail recursive family of partial functions over finite abelian groups iff for some k, there is a recursive not tail recursive family of partial constants over $\mathbf{A}_{\bar{x}}$.*

Proof. For a k-ary program E in the language Φ of groups, let the nullary $(\Phi, \bar{x})$-program $E^\star$ be obtained by replacing the k variables in the head by $\bar{x}$. Then if E computes a recursive not tail recursive family of functions over finite abelian groups, $E^\star$ computes a recursive not tail recursive family of constants over $\mathbf{A}_{\bar{x}}$.

Similarly, for a nullary $(\Phi, \bar{x})$-program E, let $E^\dagger$ be the nullary Φ-program obtained by replacing the constants x_i by additional variables. Then if E com-

putes a recursive not tail recursive family of constants over $\mathbf{A}_{\bar{x}}$, $E^\star$ computes a recursive not tail recursive family of functions over finite abelian groups. $\square$

It remains to show that there is a recursive not tail recursive family of partial constants over $\mathbf{B}_{\bar{x}}$ for some k iff $\mathrm{rec}(\mathbf{N}_{Pd}) \neq \mathrm{tail}(\mathbf{N}_{Pd})$. The forward direction is again easy: if $c_{\bar{x}}$ is such a family of partial constants over $\mathbf{B}_{\bar{x}}$ then the partial function

$$(D_1 - 1, \ldots, D_k - 1, \pi_{\bar{x}}(x_1), \ldots, \pi_{\bar{x}}(x_k)) \mapsto c_{\bar{x}}$$

is recursive not tail recursive over $\mathbf{N}_{Pd}$.

The converse, while not immediate, is much simpler than in the case of finite fields. Suppose there is a recursive not tail recursive partial function f over $\mathbf{N}_{Pd}$. Then by the results in 2.3.3, we may assume that f is unary. However, by the uniform family of bi-interpretations, a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is recursive (resp. tail recursive) iff the constant $f(d-1) \cdot x$ is uniformly computable over $\mathbf{A}_x$ (by a tail recursion), where d is the order of x. Hence, we obtain a recursive not tail recursive family of partial constants over $\mathbf{A}_x$, and we're done.

Chapter 4

Complexity Differences

We at last turn to Main Question 2

Is there a structure $\mathbf{A}$ *such that* $rec(\mathbf{A}) = tail(\mathbf{A})$*, some partial function* f *on* $\mathbf{A}$*, and some natural measure of efficiency on recursive programs such that there is a recursive program compuing* f *is more efficiently than any tail recursive program?*

In this chapter we give a positive answer to this question for a complexity measure on recursive programs that is "time-like" in that it is a *sequential* as opposed to *parallel* complexity measure.

Previous work To the best of our knowledge, there is only one previous example of an complexity difference between recursion and tail recursion in the literature. It originally appeared in [12] as an example of how the translation from linear recursion to tail recursion (see theorem 1.2.7) may necessarily come with a loss of complexity.

We feel that the most natural way to frame the problem is *printing a read-*

only linked list in reverse. In this discussion we use functions with *side-effects*, namely *print*. We trust that this will not cause confusion. The use of side effects is not essential.

Consider the structure of lists

$$\mathbf{L} = (\Sigma^{<\omega}, tail, eq_\varepsilon, printh)$$

over some alphabet Σ. Let ε be the empty list. If $\bar{u} = u_0 u_1 \dots u_{n-1}$, then

$$tail(\bar{u}) = \begin{cases} u_1 \dots u_{n-1} & n > 0 \\ \varepsilon & n = 0 \end{cases}$$

$$eq_\varepsilon(\bar{u}) \iff \bar{u} = \varepsilon$$

As for the function *printh*, it computes

$$printh(\bar{u}, \bar{v}) = \bar{v}$$

and prints u_0 to the output.

Notice there are no primitives that allow us to "build" lists, hence the term "read-only."

The following recursive equation computes the constant-ε function, but more importantly prints a given list in reverse

$$f(\bar{u}) = \text{if } eq_\varepsilon(\bar{u}) \text{ then } \varepsilon \text{ else } printh(\bar{u}, f(tail(\bar{u})))$$

Without yet having precise definitions, one can imagine what we mean by the statement "on a list $\bar{u}$ of length n, computing f on $\bar{u}$ takes at most Cn calls to the primitives of $\mathbf{L}$ for some constant C." It turns out that

Theorem. *For any tail recursive program E computing f, there is some k such that E makes at least $n^{1+1/k}$ calls to the primitives of* $\mathbf{L}$ *on lists of length n.*

This result appears in [12] without proof, though not stated in terms of printing a list. The fact that f can be computed with $n^{1+1/k}$ calls for any k can be found in [3], and a proof of the lower bound may be found in [2].

We should also mention the result of Colson in [4], which exhibits a complexity gap between *primitive recursion* and full recursion over the structure $\mathbf{N}_u$. We would ideally like to obtain a gap between recursion and tail recursion on this structure, and this remains one of the central open problems in this area. In fact, we have such a gap under assumption: the results of section 2.3 imply that there exists a partial function on the natural numbers computable over $\mathbf{N}_u$ in polynomial parameters but *not* by a tail recursion in polynomial parameters, iff exponential time is strictly greater than linear space.

Our Contributions We have improved this result in the following way. Most importantly, we have identified *arbitrarily large* gaps, in a way we make precise later. Secondly, all these gaps occur over structures in which tail recursion is equal to recursion in terms of computability.

4.1 Complexity measures on recursive programs

For a particular subset of the primitives, we measure the complexity of a recursive program by the number of times it calls a primitive in that subset on a given input. This is the type of complexity measure for which we will get a separation result.

Recall the definition 1.1.8 of an $(\mathbf{A}, E)$-term, where $\mathbf{A}$ is a Φ-structure and E is a Φ-recursive program. For an $(\mathbf{A}, E)$-term M, let $\bar{M}$ be its denotation, which is either undefined or an element of A.

Definition 4.1.1. For an $(\mathbf{A}, E)$-term M and $\Phi_0 \subseteq \Phi$, define $C_{\Phi_0}(M) = C_{\Phi_0}(\mathbf{A}, E, M)$ by recursion on M. In the following recursive equation, if any term, e.g. $\bar{M}$, on the right hand side diverges, then so does $C_{\Phi_0}(M)$. Hence C_{Φ_0} is a partial function from $(\mathbf{A}, E)$-terms to $\mathbb{N}$.

$$C_{\Phi_0}(M) =$$

$$\begin{cases} 0 & \text{if } M \text{ is a constant} \\ 1 + C_{\Phi_0}(M_1) + \cdots + C_{\Phi_0}(M_n) & \text{if } M \equiv \phi(M_1, \ldots, M_n) \text{ with } \phi \in \Phi_0 \\ C_{\Phi_0}(M_1) + \cdots + C_{\Phi_0}(M_n) + C_{\Phi_0}(E_i(\bar{M}_1, \ldots, \bar{M}_n)) & \text{if } M \equiv p_i(M_1, \ldots, M_n) \\ C_{\Phi_0}(M_0) + C_{\Phi_0}(M_1) & \text{if } M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2 \text{ and } \bar{M}_0 = 0 \\ C_{\Phi_0}(M_0) + C_{\Phi_0}(M_2) & \text{if } M \equiv \text{if } M_0 \text{ then } M_1 \text{ else } M_2 \text{ and } \neg\bar{M}_0 = 0 \end{cases}$$

Then $C_{\Phi_0}(M)$ is exactly the number of calls to the primitives in Φ_0 it takes to evaluate M, or undefined if the computation from M diverges.

The number of calls it takes to evaluate E on input x for a program E is $C_{\Phi_0}(\mathbf{A}, E, E_0(x))$, $E_0(x)$ being of course the first $(\mathbf{A}, E)$-term in the computation. Following [10], we define

$$c_{\Phi_0}(\mathbf{A}, E, x) := C_{\Phi_0}(\mathbf{A}, E, E_0(x))$$

4.2 Gaps in complexity

For the purposes of this section, define

$$\mathbf{N}^{\star}_{Pd} := (0, 1, Pd, eq_0)$$

and consider the expansion $(\mathbf{N}^\star_{Pd}, \gamma)$ of $\mathbf{N}^\star_{Pd}$ by an increasing unary function γ to be determined later. This is the structure on which we will obtain our complexity separation.

Notice that full equality is replaced by eq_0. In terms of tail recursive computability, $(\mathbf{N}_{Pd}, \gamma)$ and $(\mathbf{N}^\star_{Pd}, \gamma)$ are equivalent: we can compute whether $x = y$ by decrementing both until one is zero. In terms of number of calls complexity, they are *not* equivalent. Our result does not extend to the case with full equality.

On the structure $(\mathbf{N}^\star_{Pd}, \gamma)$ the total function

$$f(n, x) = \gamma^{2^n}(x) \tag{4.1}$$

can be computed the recursive program E:

$$f(n,x) = p(n,x) \text{ where}$$
$$p(n,x) = \begin{cases} \gamma(x) & \text{if } n = 0 \\ p(n-1, p(n-1,x)) & \text{otherwise} \end{cases}$$

It is easy to show:

Lemma 4.2.1. *$c_\gamma((\mathbf{N}^\star_{Pd}, \gamma), E, n, x) = 2^n$. In other words, E makes 2^n calls to γ on the input (n, x).*

For any function $g : \mathbb{N} \to \mathbb{N}$, we will define γ such that for infinitely many (n, x), computing $f(n, x)$ takes at least $g(n)$ calls to γ for any tail recursive program. This shows, in fact, that there can be arbitrarily large gaps in complexity between tail recursion and recursion in complete structures.

4.2.1 Combinatorics of $(\mathbf{N}^\star_{Pd}, \gamma)$-tail recursion

For this section fix a natural number k.

Definition 4.2.2. For tuples $\bar{m}, \bar{n} \in \mathbb{N}^k$ and $a \in \mathbb{N}$, define $\bar{m} \sim_a \bar{n}$ in case $\min(m_i, a) = \min(n_i, a)$ for each $1 \leq i \leq k$.

In other words, $\bar{m} \sim_a \bar{n}$ if the two tuples "agree on parameters less than a." It is clearly an equivalence relation.

It is straightforward to show that:

Lemma 4.2.3. *For some $f : \mathbb{N} \to \mathbb{N}$, and $a, b, c, c' \in \mathbb{N}$, suppose $c' - c$ is less than or equal to $f(a) - a$ and $f(b) - b$. Then*

$$\min(a, c) = \min(b, c) \implies \min(f(a), c') = \min(f(b), c')$$

There are two basic results governing how the relation $\sim_a$ interacts with explicit $(\mathbf{N}^\star_{Pd}, \gamma)$-terms:

Lemma 4.2.4. *Suppose F is an explicit $(\mathbf{N}^\star_{Pd}, \gamma)$-term of arity and co-arity k. Then there is a constant b depending only on F such that for all $a > b$,*

$$\bar{m} \sim_a \bar{n} \implies F(\bar{m}) \sim_{a-b} F(\bar{n})$$

Proof. Suppose that $F = (F_1, \ldots, F_k)$ and let b be the maximum length over all F_i's. Fix $\bar{m}$, $\bar{n}$, and a such that $\bar{m} \sim_a \bar{n}$ and $a > b$. We need to show that for each $1 \leq i \leq k$, $\min(F_i(\bar{m}), a - b) = \min(F_i(\bar{n}), a - b)$. We break into three cases depending on whether F_i contains none of $\{0, 1, eq_0\}$, contains eq_0 but not 0 or 1, or contains 0 or 1.

It's easy to see that for a $\{Pd, \gamma\}$-term f of length at most ℓ and $x \in \mathbb{N}$, $f(x) \geq x - \ell$, since γ is an increasing function. If F_i contains none of $\{0, 1, eq_0\}$, then $F_i(\bar{x})$ is $f(x_j)$ for some $\{Pd, \gamma\}$-term f and $1 \leq j \leq k$. Hence for these i,

$$\exists j \leq k \; \forall \bar{x} \in \mathbb{N}^k \; F_i(\bar{x}) \geq x_j - b$$

For these i, we have $b \leq F_i(\bar{m}) - m_j$ and $b \leq F_i(\bar{n}) - n_j$ for some j. Therefore by lemma 4.2.3,

$$\min(m_j, a) = \min(n_j, a) \implies \min(F_i(\bar{m}), a - b) = \min(F_i(\bar{n}), a - b)$$

and we're done.

In the second case, we consider i such that F_i is a $\{Pd, \gamma, eq_0\}$-term that contains at least one instance of eq_0. Then $F_i(\bar{x}) = F' \circ eq_0 \circ f(x_j)$ for some $1 \leq j \leq k$, $\{Pd, \gamma\}$-term f of length at most b (and hence less than a), and $\{Pd, \gamma, eq_0\}$-term F'. Notice that since f has depth less than a,

$$\min(m_j, a) = \min(n_j, a) \implies eq_0(f(m_j)) = eq_0(f(n_j))$$

Hence

$$\min(m_j, a) = \min(n_j, a) \implies F_i(\bar{m}) = F_i(\bar{n})$$

and we're done.

Finally, we consider i such that F_i contains 0 or 1. But then F_i is a variable-free term, so $F_i(\bar{m}) = F_i(\bar{n})$ automatically. This concludes the proof. □

Corollary 4.2.5. *Suppose F is an explicit $(\mathbf{N}^\star_{Pd}, \gamma)$-term of arity k. Then there is a constant c depending only on F such that for all $a > c$,*

$$\bar{m} \sim_a \bar{n} \implies eq_0(F(\bar{m})) = eq_0(F(\bar{n}))$$

Proof. Let c be the length of F and suppose $a > c$ and $\bar{m} \sim_a \bar{n}$. By the proof of lemma 4.2.4, we have that

$$\min(F(\bar{m}), a - c) = \min(F(\bar{n}), a - c)$$

Since $a - c > 0$, we conclude $eq_o(F(\bar{m})) = eq_0(F(\bar{n}))$. □

Now fix a tail recursive program

$$\begin{aligned} f(x_1, \ldots, x_n) &= p(G(x_1, \ldots, x_n)) \text{ where} \\ p(\bar{x}) &= \text{if } \tau(\bar{x}) \text{ then } o(\bar{x}) \text{ else } p(F(\bar{x})) \end{aligned} \tag{4.2}$$

where $\bar{x} = (x_1, \ldots, x_k)$, G, τ, o, and F are explicit $(\mathbf{N}^\star_{Pd}, \gamma)$-terms, and G and F have co-arity k. Let b be obtained from F as per lemma 4.2.4 and let c be obtained from τ as per corollary 4.2.5. Then we have:

Corollary 4.2.6. *Suppose $\bar{m} \sim_{ab+c} \bar{n}$ for some $a > 0$. Then $\tau(F^j(\bar{m})) = 0 \iff \tau(F^j(\bar{n})) = 0$ for $0 \leq j < a$.*

Proof. By lemma 4.2.4, $F^j(\bar{m}) \sim_{(a-j)b+c} F^j(\bar{n})$ for $0 \leq j < a$. Then by corollary 4.2.5, since $(a-j)b+c > c$, we have that $eq_0 \circ \tau \circ F^j(\bar{m}) = eq_0 \circ \tau \circ F^j(\bar{n})$. □

The consequence of this is that, assuming $\bar{m} \sim_{ab+c} \bar{n}$, if $p(\bar{m}) = o(F^j(\bar{m}))$ for some $j < a$ then $p(\bar{n}) = o(F^j(\bar{n}))$ and vice versa.

Definition 4.2.7. For $\bar{m} \in \mathbb{N}^k$ and $u < v \in \mathbb{N}$, define

$$\mathfrak{G}(\bar{m}, u, v) \iff \{m_1, \ldots, m_k\} \cap (u, v) = \emptyset$$

(Here (u, v) is an interval, i.e. the set $\{n \in \mathbb{N} : u < n < v\}$.)

Then we have immediately that if $\mathfrak{G}(\bar{m}, u, v)$, $\mathfrak{G}(\bar{n}, u, v)$ and $\bar{m} \sim_a \bar{n}$ then $\bar{m} \sim_b \bar{n}$.

Lemma 4.2.8. *Suppose that $\bar{m} \sim_{ab+c} F^i(\bar{m})$ for some $a, i > 0$. Then the least j such that $\tau(F^j(\bar{m})) = 0$, if it exists, cannot be in the interval $[i, i + a)$.*

Proof. By corollary 4.2.6, for $0 \leq j < a$,

$$\tau(F^j(\bar{m})) = 0 \iff \tau(F^{i+j}(\bar{m})) = 0$$

Therefore if for some $i \leq j < a$, $\tau(F^j(\bar{m})) = 0$, then $\tau(F^{j-i}(\bar{m})) = 0$. $\square$

Finally, we have

Lemma 4.2.9. *For some $a' \geq a > 0$, suppose $\mathfrak{G}(F^j(\bar{m}), a, a'b+c)$ and $\tau(F^j(\bar{m})) \neq 0$ for $0 \leq j < t$. Then either $t \leq (a+2)^k$ or $\tau(F^j(\bar{m})) \neq 0$ for $0 \leq j < a'$.*

Proof. If $F^{n_1}(\bar{m}) \sim_a F^{n_2}(\bar{m})$ for some $0 \leq n_1 < n_2 < t$, then $F^{n_1}(\bar{m}) \sim_{a'b+c} F^{n_2}(\bar{m})$, and we know by lemma 4.2.8 that the least j such that $\tau(F^j(\bar{m})) = 0$ cannot be in the interval $[n_2 - n_1, n_2 - n_1 + a')$. Since $\tau(F^j(\bar{m})) \neq 0$ for $\ell < n_2 - n_1$ by assumption, we conclude that the least such j such that $\tau(F^j(\bar{m})) = 0$ is at least a', and we're done.

Otherwise each tuple $\{F^j(\bar{m}) : 0 \leq j < t\}$ is in its own $\sim_a$-equivalence class, the number of which therefore must be at least t. How many $\sim_a$-equivalence classes are there? For each of the k indices, there are $u+2$ choices: one for each $\{0, 1, \ldots, a\}$, and one for numbers $\geq u$. $\square$

Corollary 4.2.10. *For some $a' \geq a > 0$, if $\mathfrak{G}(F^j(\bar{m}), a, a'b+c)$ and $\tau(F^j(\bar{m})) \neq 0$ for $0 \leq j \leq (a+2)^k$, then $\tau(F^j(\bar{m})) \neq 0$ for $0 \leq j < a'$.*

This result may seem very technical but the philosophy behind it is quite clear:

Suppose the elements of a tuple $\bar{m}$ contain very big or very small numbers. Then the computation of p on $\bar{m}$ halts in either a very short or very long amount of time.

4.2.2 An explicit complexity gap

Fix an increasing function g. Let $\gamma : \mathbb{N} \to \mathbb{N}$ be any increasing function for which for every function $L : x \mapsto ux + v$ for $u, v \in \mathbb{N}$, the interval $(n, 2^n + L(g(n)))$ is disjoint with the range of γ for arbitrarily large n. The fact that there exists such a functions requires proof.

Lemma 4.2.11. *For every increasing function $g : \mathbb{N} \to \mathbb{N}$ there is an increasing function $\gamma : \mathbb{N} \to \mathbb{N}$ such that for all $u, v \in \mathbb{N}$ there exists infinitely many n such that $(n, 2^n + g(n)u + v)$ is disjoint with the range of γ.*

Proof. We define γ by recursion. Let $\gamma(0)$ be 0, and define

$$\gamma(i+1) := 1 + \max\{2^{\gamma(i)+1} + g(\gamma(i)+1)u + v \mid u, v \leq i\}$$

so that for $n = \gamma(i+1)$, $\gamma(i) < n < 2^n + g(n)u + v < \gamma(i+1)$ for all $u, v \leq i$. Clearly γ is increasing.

Now fix $u, v \in \mathbb{N}$. Then for all $i \geq u, v$ if $n = \gamma(i)+1$, then $(n, 2^n+g(n)u+v)$ is disjoint from the range of γ. This is because $\gamma(i) < n$, $2^n+g(n)u+v < \gamma(i+1)$, and γ is increasing so there is no j such that $\gamma(i) < \gamma(j) < \gamma(i+1)$. □

Suppose that the tail recursion (4.2) computes the function f of (4.1). We shall show that the program (4.2) makes at least $g(n)$ calls to γ for infinitely many inputs (n, x). It's a trivial observation that the term F must contain some γ, otherwise for each (n, x) $\gamma^{2^n}(x)$ could be denoted by a term with a constant-in-n number of γ's. Since all the other primitives are decreasing, this is clearly false.

Let $L(n)$ be the linear function $nb+c$, where b and c are obtained from (4.2). By definition, $(n, 2^n+L(g(n)))$ is disjoint with the range of γ for arbitrarily large n. Let $n_1 < n_2 < \ldots$ be an infinite increasing sequence witnessing this, and let $x_i := 2^{n_i} + L(g(n_i))$.

For all i, j, let $\bar{m}_{i,j} := F^j(G(n_i, x_i))$. Then

Lemma 4.2.12. *For $0 \leq j < 2^{n_i}$, $\mathfrak{G}(\bar{m}_{i,j}, n_i, L(g(n_i)))$.*

Proof. Let $\bar{m}_{i,j} = (m_1^{(i,j)}, \ldots, m_k^{(i,j)})$. Let m range over $m_\ell^{(i,j)}$ for $i \in \mathbb{N}$, $0 \leq j < 2^{n_i}$ and $1 \leq \ell \leq k$. Then $m = b(y)$ where b is an algebraic $\{Pd, \gamma, eq_0\}$-term

of length less than 2^{n_i} and y is 0, 1, n_i, or x_i. I claim that all such m are contained in the set

$$[0, n_i] \cup [x_i - 2^{n_i}, x_i] \cup \bigcup_{\vartheta \in \gamma[\mathbb{N}]} [\vartheta - 2^{n_i}, \vartheta]$$

In case b is a $\{Pd\}$-term, this is obvious. Otherwise $b = Pd^j \circ \gamma \circ b'$ or $b = Pd^j \circ eq_0 \circ b'$ for some $j < 2^{n_i}$ and algebraic $\{Pd, \gamma, eq_0\}$-term b'. Define $m' := b'(y)$. In the first case, $m = \gamma(m') - j$, so $m \in [\vartheta - 2^{n_i}, \vartheta]$ for some ϑ in the range of γ. In the second case, m is 0 or 1.

Since $x_i - 2^{n_i} = L(g(n_i))$, each parameter m is contained in the set(2.3.3)

$$[0, n_i] \cup [L(g(n_i)), L(g(n_i)) + 2^{n_i}] \cup \bigcup_{\vartheta \in \gamma[\mathbb{N}]} [\vartheta - 2^{n_i}, \vartheta]$$

However, if we take the intersection with $(n_i, 2^{n_i} + L(g(n_i)))$,

$$[0, n_i] \cap (n_i, L(g(n_i))) = \emptyset$$

$$[L(g(n_i)), L(g(n_i)) + 2^{n_i}] \cap (n_i, L(g(n_i))) = \emptyset$$

If $z \in (n_i, L(g(n_i))) \cap [\vartheta - 2^{n_i}, \vartheta]$ for some $\vartheta \in \gamma[\mathbb{N}]$, then

$$n_i < z \leq \vartheta \leq z + 2^{n_i} < L(g(n_i)) + 2^{n_i}$$

This contradicts the assumption that the range of γ and $(n_i, L(g(n_i)) + 2^{n_i})$ are disjoint for all n_i. □

Theorem 4.2.13. *For sufficiently large i, the number of calls to γ made by the tail recursive program (4.2) on input (n_i, x_i) is at least $g(n_i)$.*

Proof. By lemma 4.2.12 and the definition of L, for all i, $\mathfrak{G}(\bar{m}_{i,j}, n_i, g(n_i)b + c)$

for $0 \leq j < 2^{n_i}$. Moreover, the least j such that $\tau(\bar{m}_{i,j}) = 0$ grows exponentially in n_i. This is because j is bounded below by a constant factor of the number of calls to γ made by the tail recursion (4.2), and the number of calls to γ made while computing $\gamma^{2^{n_i}}(x_i)$ is at least 2^{n_i}. This, in turn, is because the shortest algebraic $(\mathbf{N}^{\star}_{Pd}, \gamma)$-term defining $\gamma^{2^{n_i}}(x_i)$ in terms of n_i and x_i has length 2^{n_i}, and this so-called *value-depth complexity* is a lower bound for the sequential number-of-calls complexity (see chapter 4 in [10]).

Therefore, for sufficiently large i, the least j such that $\tau(\bar{m}_{i,j}) = 0$ is greater than $(n_i + 2)^k$. By corollary 4.2.10, we conclude that $\tau(\bar{m}_{i,j}) \neq 0$ for $0 \leq j < g(n_i)$. In other words, on input (n_i, x_i), the tail recursion (4.2) makes at least $g(n_i)$ recursive calls before halting. But since F contains at least one γ, it is easy to see that the number of calls to γ is at least $g(n_i)$ as well. □

Conclusion For each increasing function g, we have exhibited a structure $(\mathbf{N}^{\star}_{Pd}, \gamma)$ and a γ-recursive function f such that for each tail recursive program computing f, there is an infinite family of inputs (n_i, x_i), on which the number of calls to γ made by that program is at least $g(n_i)$. On the other hand, the recursive program E always makes 2^n calls to γ on all inputs (n, x), which is optimal.

Hence we have found structures with arbitrarily large complexity gaps. On the other hand, with an increasing function γ we can compute the successor by a tail recursion, so in terms of computability there is no difference between recursion and tail recursion over $(\mathbf{N}^{\star}_{Pd}, \gamma)$. Therefore, we have a positive answer to Main Question 2.

Bibliography

[1] XuHong Gao R. C. Mullin S. A. Vanstone T. Yaghoobian A. J. Menezes, I. F. Blake. *Applications of Finite Fields*. Springer US, 1993.

[2] Holger Petersen Amir M. Ben-Amram. Backing up in singly linked lists. *Journal of the ACM*, 53(4):681–705, 2006.

[3] Ashok K. Chandra. Efficient compliation of linear recursive programs. Technical report, Stanford University, 1972.

[4] L. Colson. About primitive recursive algorithms. *Theoretical Computer Science*, 83(1):57–69, 1991.

[5] Sheila A. Greibach. *Theory of Program Structures: Schemes, Semantics, Verification*. Springer-Verlag, 1975.

[6] Neil Jones. Logspace and ptime characterized by progprogram languages. *Theoretical Computer Science*, 228(1):151–174, 1999.

[7] A. J. Kfoury and A. P. Stolboushkin. An infinite pebble game and applications. *Information and Computation*, 136:53–66, 1997.

[8] N. A. Lynch and E. K. Blum. A difference in expressive power between flowcharts and recursion schemes. *Mathematical Systems Theory*, 12(1):205–211, 1979.

[9] John McCarthy. A basis for a mathematical theory of computation. In *Proceedings of the Western Joint Computer Conference*, 1961.

[10] Y. N. Moschovakis. *Recursion and Complexity.*

[11] Y. N. Moschovakis and L. van den Dries. Arithmetic complexity. *ACM Transactions on Computational Logic*, 10, 2009.

[12] M. S. Paterson and C. E. Hewitt. Comparative schematology. Technical report, MIT Artificial Intelligence lab, 1970.

[13] Paul Pollack. Revisiting gauss's analogue of the prime number theorem for polynomials over a finite field. *Finite Fields and Their Applications*, 16(4):290–299, 2010.

[14] Jerzy Tiuryn. A simplified proof of ddl $<$ dl. *Information and Computation*, 81(1):1–12, 1989.

[15] J. V. Tucker and J. I. Zucker. *Computable functions and semicomputable sets on many-sorted algebras*, chapter 5, pages 397–525. Oxford University Press, 2000.

[16] Pawel Urzyczyn. A necessary and sufficient condition in order that a herbrand interpretation be expressive relative to recursive programs. *Information and Control*, 56(3):212–219, 1983.

[17] S. A. Walker and H. R. Strong. Characterizations of flowchartable recursions. *Journal of Computer and System Sciences*, 7(4):404–477, 1973.