Formal Theories for Logspace Counting

Lila A. Fontes

Advisor: Stephen A. Cook

Research paper submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Department of Computer Science
University of Toronto

Toronto, Ontario
May 10, 2009

Contents

Acknowledgements

I would like to thank my advisor, Stephen A. Cook, for suggesting this project, and for his colossal help, enthusiasm, and advising over the course of my research. This project would not have been possible without the foundations he and Phuong Nguyen layed in their book *Logical Foundations of Proof Complexity* [9], and without months of conversations as I digested that material and began to see how I could build this small extension of it.

Thanks also to Eric Allender and Michael Soltys, who provided insights into the problem of AC^0-closure and reductions for the class DET, and pointed towards many useful references on the same.

Many thanks for a lifetime of support and encouragement are due to my parents and grandparents.

May 10, 2009 Lila A. Fontes

1 Introduction

This paper follows the framework of Chapter 9 of Cook and Nguyen's basic monograph on proof complexity [9]. Therein, the authors establish a general method for constructing a two-sorted logical theory that formalizes reasoning using concepts from a given complexity class. This logical theory extends their base theory V^0 for AC^0 by the addition of a single axiom. The axiom states the existence of a solution for a complete problem for a closed complexity class. Here, completeness and closure are with respect to AC^0-reductions, and in order to use the methods presented in Chapter 9 and earlier chapters, we will have to establish that our classes are closed under such reductions.

We focus on the classes $\#L$ and $\oplus L$, logspace counting and parity (counting mod 2), respectively. Obviously, $L \subseteq \oplus L \subseteq \#L$. In fact, $\#L$ contains the class $\mathrm{MOD}_k\, L$ for every k, and $\#L \subseteq NC^2 \subseteq P$. The logspace counting class $\#L$ is defined from L by analogy to $\#P$ and P. Both counting classes have nice complete problems: the problem of computing the permanent of a matrix is complete for $\#P$, and the problem of computing the determinant of a matrix is complete for $\#L$. Unlike $\#P$, however, $\#L$ is not known to be closed under several standard notions of reduction. The logspace counting hierarchy $\#LH$ is defined as $\#LH_1 = \#L$, $\#LH_{i+1} = L^{\#LH_i}$, and $\#LH = \bigcup_i \#LH_i$. There are several names for the closure under consideration:

$$AC^0(\#L) = \#LH = DET$$

It is unknown whether this hierarchy collapses, though Allender shows that $AC^0(\#L) = NC^1(\#L)$ is a sufficient condition for this collapse [1].

Since the class $\#L$ is not closed under AC^0-reductions, we consider its AC^0 closure $AC^0(\#L)$, denoted DET. In renaming this class, we follow the notation of [1], [12], and others. This class is suggestively named: one of its AC^0-complete problems is that of finding the determinant of an integer-valued matrix. (This should not be confused with the notation from Cook's survey [8], which considers NC^1-reductions.) $\oplus L$ is appropriately closed, and the fact that function values are mod 2 allows for a convenient notational shortcut not available in $\#L$: each number can be stored in just one bit of a bit string. Section 5 develops the theory for $\#L$. Section 4 develops the theory for $\oplus L$.

Following the format of Chapter 9 of [9], a class is defined in each section and shown to be closed under AC^0-reductions. Next, a complete problem is demonstrated and formalized as an axiom. (Because the classes are closely related, we use the same problem — matrix powering over the appropriate ring — for both.) VTC^0 extended by this axiom forms the theory $V\#L$ (respectively, $V^0(2) \subset V\oplus L$) with vocabulary $\mathcal{L}_A^2$, the basic language of 2-sorted arithmetic. Following this, we develop a universal conservative extension $\overline{V\#L}$ (resp. $\overline{V\oplus L}$), in which every string function has a symbol, with extended language $\mathcal{L}_{F\#L}$ (resp. $\mathcal{L}_{F\oplus L}$). By general results from Chapter 9 of [9], the provably total functions of $V\#L$ are exactly the functions of class $\#L$ (and similarly for $\oplus L$). These functions include many standard problems of linear algebra over the rings $\mathbb{Z}$ and $\mathbb{Z}_k$ ([7], [8], [6], [5]), as discussed in Section 6.

2 Notation

In this section, we restate useful number and string functions from [9]. The conventional functions with which we manipulate bit-strings and numbers are so pervasive in the discussion to follow that they merit their own notation. We also extend these functions with new notation, which will be helpful in the presentation of formal theories in later sections.

Chapter 4 of [9] defines the two-sorted first-order logic of our theories. There are two kinds of variables (and predicates and functions): *number* variables, indicated by lowercase letters x, y, z, $\ldots$, and *string* variables, indicated by uppercase letters X, Y, Z, $\ldots$. Strings can be interpreted as finite subsets of $\mathbb{N}$. Predicate symbols P, Q, R, $\ldots$, can take arguments of both sorts, as can function symbols. Function symbols are differentiated by case: f, g, h, $\ldots$, are number functions, and F, G, H, $\ldots$, are string functions.

2.1 Strings encoding matrices and lists

Section 5D of [9] provides useful notation for encoding a k-dimensional bit array in a string X. For our purposes below, it is useful to extend this notation, so that a string X may encode a 2-dimensional array of strings.

[9] defines the AC^0 functions *left*, *right*, *seq*, and $\langle \cdot, \cdot \rangle$, and the AC^0 relations *Pair* and *Row*, as follows.

The pairing function $\langle x, y \rangle$ is defined as

$$\langle x, y \rangle = (x + y)(x + y + 1) + 2y$$

This function can be chained to "pair" more than two numbers:

$$\langle x_1, x_2, \ldots, x_k \rangle = \langle \langle x_1, \ldots, x_{k-1} \rangle, x_k \rangle$$

Inputs to the pairing function can be recovered using the projection functions *left* and *right*:

$$y = \mathit{left}(x) \leftrightarrow \exists z \le x(x = \langle y, z \rangle) \qquad z = \mathit{right}(x) \leftrightarrow \exists y \le x(x = \langle y, z \rangle)$$

By definition, $\mathit{left}(x) = \mathit{right}(x) = 0$ when x is not a pair number, i.e., when $\neg \mathit{Pair}(x)$, where $\mathit{Pair}(x) \equiv \exists y, z \le x(x = \langle y, z \rangle)$.

Thus a k-dimensional bit array can be encoded in string X by:

$$X(x_1, \ldots, x_k) = X(\langle x_1, \ldots, x_k \rangle)$$

The *Row* function is bit-defined as:

$$\mathit{Row}(x, Z)(i) \leftrightarrow i < |Z| \wedge Z(x, i)$$

For notational convenience, we write $\mathit{Row}(x, Z) = Z^{[x]}$. This can be used to encode a 1-dimensional array of j strings $X_1, \ldots, X_j$ in a single string Z, where $X_i = Z^{[i]}$.

It will be useful in Section 4.3 to compose the *Row* function so that a single string encodes a 2-dimensional array of strings $X_{i,j}$. First encode each row i as a string Y_i, where $Y_i^{[j]} = X_{i,j}$. Then encode the list of strings Y_i as a 1-dimensional array of strings $Z^{[i]} = Y_i$. The resultant string encodes a 2-dimensional array of strings. Let $\mathit{Row}_2(x, y, Z) = Z^{[x][y]}$

represent the string in the $(x, y)^{\text{th}}$ position of the matrix of strings encoded in Z. Row_2 has bit definition:

$$Row_2(x, y, Z)(i) \leftrightarrow i < |Z| \wedge Row(x, Z)(y, i) \tag{1}$$

Using the *Row* Elimination Lemma (5.52 in [9]), we can obtain a $\Sigma_0^B(\mathcal{L}_A^2)$ formula provably equivalent in $V^0(Row)$ to (1). By Corollary 5.39 in [9], $V^0(Row_2)$ is a conservative extension of V^0. Also, using the Σ_0^B-Transformation Lemma (5.40 in [9]), we can obtain an analogous Row_2 elimination lemma.

Lemma 1 *For every $\Sigma_0^B(Row_2)$ formula φ, there is a $\Sigma_0^B(\mathcal{L}_A^2)$ formula φ^+ such that $V^0(Row_2) \vdash \varphi \leftrightarrow \varphi^+$.*

Just as *Row* is used to extract a list of strings $Z^{[0]}, Z^{[1]}, \ldots$, from Z, the number function *seq* enables string Z to encode a list of numbers y_0, y_1, y_2, $\ldots$, where $y_i = seq(i, Z)$. We denote $seq(i, Z)$ as $(Z)^i$. The number function $seq(x, Z)$ has the defining axiom

$$y = seq(x, Z) \leftrightarrow (y < |Z| \wedge Z(x, y) \wedge \forall z < y, \neg Z(x, z)) \vee (\forall z < |Z|, \neg Z(x, z) \wedge y = |Z|)$$

It will be useful in Section 3 to compose *Row* and *seq* so that a string encodes a matrix of numbers. Each row of the matrix is encoded as a string Y_i, where $(Y_i)^j = seq(j, Y_i)$ is the j^{th} number in the row. These rows are encoded as a string Z, where $Y_i = Z^{[i]} = Row(i, Z)$. The $(i, j)^{\text{th}}$ entry of the matrix is recoverable as $entry(i, j, Z)$. Number function *entry* has the definition

$$entry(i, j, Z) = y \leftrightarrow (Z^{[i]})^j = y \tag{2}$$

The next lemma follows from the *Row* Elimination Lemma (5.52 in [9]), the Σ_0^B-Transformation Lemma (5.40 in [9]), and (2).

Lemma 2 *For every $\Sigma_0^B(entry)$ formula φ, there is a $\Sigma_0^B(\mathcal{L}_A^2)$ formula φ^+ such that $V^0(entry) \vdash \varphi \leftrightarrow \varphi^+$.*

2.2 Terminology and conventions for defining string functions

For some string functions, we are only concerned with certain bits of the output; we ignore all other bits. However, every bit of output must be specified in order to define a function and argue about its uniqueness. The following convention allows us to define a string function by specifying only its "interesting" bits, and requiring that all other bits be zero.

For example, let $F(i, \vec{x}, \vec{X}) = Z$ be a string function, and let φ be its bit-graph:

$$F(i, \vec{x}, \vec{X})(b) \leftrightarrow \varphi(i, b, \vec{x}, \vec{X})$$

Let G be a string function such that $G(\vec{x}, \vec{X})^{[i]} = F(i, \vec{x}, \vec{X})$.

The technically correct bit-definition of G must specify every bit of the output:

$$G(\vec{x}, \vec{X})(b) \leftrightarrow \exists i, j < b, \langle i, j \rangle = b \wedge \varphi(i, j, \vec{x}, \vec{X}) \tag{3}$$

For conciseness, throughout this document, such bit-definitions will instead be written as:

$$G(\vec{x}, \vec{X})(i, j) \leftrightarrow \varphi(i, j, \vec{x}, \vec{X})$$

This has the intended meaning of (3), that is, the string function G is false at all bits b which are *not* pair numbers. Notice that this encoding of the list G of strings $F(0, \vec{x}, \vec{X})$, $F(1, \vec{x}, \vec{X})$,..., results in many "wasted" bits.

The following definitions are restated here for ease of reference.

Definition 3 ([9] 5.26, Two-Sorted Definability) *Let $\mathcal{T}$ be a theory with vocabulary $\mathcal{L} \supseteq \mathcal{L}_A^2$, and let Φ be a set of $\mathcal{L}$-formulas. A number function f not in $\mathcal{L}$ is Φ-definable in $\mathcal{T}$ if there is a formula $\varphi(y, \vec{x}, \vec{X})$ in Φ such that*

$$\mathcal{T} \vdash \forall\vec{x}\forall\vec{X}\exists!y, \varphi(y, \vec{x}, \vec{X})$$

and

$$y = f(\vec{x}, \vec{X}) \leftrightarrow \varphi(y, \vec{x}, \vec{X})$$

A string function F not in $\mathcal{L}$ is Φ-definable in $\mathcal{T}$ if there is a formula $\varphi(\vec{x}, \vec{X}, Y)$ in Φ such that

$$\mathcal{T} \vdash \forall\vec{x}\forall\vec{X}\exists!Y, \varphi(\vec{x}, \vec{X}, Y)$$

and

$$Y = F(\vec{x}, \vec{X}) \leftrightarrow \varphi(\vec{x}, \vec{X}, Y)$$

Definition 4 ([9] 5.31, Bit-Definable Function) *Let φ be a set of $\mathcal{L}$ formulas where $\mathcal{L} \supseteq \mathcal{L}_A^2$. We say that a string function symbol $F(\vec{x}, \vec{X})$ not in $\mathcal{L}$ is Φ-bit-definable from $\mathcal{L}$ if there is a formula $\varphi(i, \vec{x}, \vec{X})$ in Φ and an $\mathcal{L}_A^2$ number term $t(\vec{x}, \vec{X})$ such that the bit graph of F satisfies*

$$F(\vec{x}, \vec{X})(i) \leftrightarrow i < t(\vec{x}, \vec{X}) \wedge \varphi(i, \vec{x}, \vec{X})$$

The right-hand side of the above equation is the "bit-definition" of F.

Definition 5 ([9] 5.37, Σ_0^B-definable) *A number (resp., string) function is Σ_0^B-definable from a collection $\mathcal{L}$ of two-sorted functions and relations if it is p-bounded and its (bit) graph is represented by a $\Sigma_0^B(\mathcal{L})$ formula.*

The notion of Σ_0^B-definability is different from Σ_0^B-definability in a theory, which is concerned with provability. The two are related by the next result.

Corollary 6 ([9] 5.38) *Let $\mathcal{T} \supseteq V^0$ be a theory over $\mathcal{L}$ and assume that $\mathcal{T}$ proves the $\Sigma_0^B(\mathcal{L})$-COMP axiom scheme. Then a function which is Σ_0^B-definable from $\mathcal{L}$ is $\Sigma_0^B(\mathcal{L})$-definable in $\mathcal{T}$.*

3 Formalizing $\#L$ and $\oplus L$

The first step is to prove $\#STCON$ and matrix powering over $\mathbb{N}$ are AC^0-complete for $\#L$. The notion of an AC^0 reduction generalizes Σ_0^B-definability (Definition 5).

Definition 7 ([9] 9.1 AC^0-Reducibility) *A string function F (respectively, a number function f) is AC^0-reducible to $\mathcal{L}$ if there is a sequence of string functions $F_1, \ldots, F_n$, $(n \geq 0)$ such that*

$$F_i \text{ is } \Sigma_0^B\text{-definable from } \mathcal{L} \cup \{F_1, \ldots, F_{i-1}\} \text{ for } i = 1, \ldots, n;$$

and F (resp. f) is Σ_0^B-definable from $\mathcal{L} \cup \{F_1, \ldots, F_n\}$. A relation R is AC^0-reducible to $\mathcal{L}$ if there is a sequence of string functions $F_1, \ldots, F_n$ as above, and R is represented by a $\Sigma_0^B(\mathcal{L} \cup \{F_1, \ldots, F_n\})$-formula.

Notice that this is a semantic notion, separate from whether the function F (or f) is definable *in a theory* (in the style of Definition 3).

Definition 8 ([9] 5.15, Function Class) *If C is a two-sorted complexity class of relations, then the corresponding function class FC consists of all p-bounded number functions whose graphs are in C, together with all p-bounded string functions whose bit graphs are in C.*

This general definition yields the function classes $F\#L$ and $F\oplus L$ from the classes $\#L$ and $\oplus L$.

3.1 $\#L$ and its AC^0-complete problems

Definition 9 *$\#L$ is the class of functions F such that there is a non-deterministic logspace Turing machine, halting in polynomial time on all inputs, which on input X, has exactly $F(X)$ accepting computation paths.*[1]

The class $\#L$ is not known to be closed under AC^0-reductions, so we consider instead its closure $AC^0(\#L)$, denoted DET as in [1], [12], [2], and others.

Definition 10 *DET is the class of functions F which are AC^0-reducible to $\#L$.*

Since DET is a function class, and we want a relation class, we consider the class of relations with characteristic functions in DET.

The *characteristic function* $f_R(\vec{x}, \vec{X})$ of a relation $R(\vec{x}, \vec{X})$ is defined as

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

Definition 11 *$RDET$ is the class of relations whose characteristic functions are in DET.*

Definition 12 *Let $\#STCON$ be the functional counting analog of the decision problem $STCON$. That is, $\#STCON$ is a function which, given a directed graph*[2] *with two distinguished nodes $s \neq t$ among its n nodes, outputs the number of distinct paths of length $\leq p$ from s to t. Let G be the $n \times n$ Boolean matrix encoding the adjacency matrix of the graph. Then this number is represented (in binary) by the string $\#STCON(n, s, t, p, G)$.*

[1] In general, we expect that the value of a function in $\#L$ is exponential; since we limit ourselves to polynomially-bounded theories, the output of the function will be encoded as a binary string, and denoted with a capital letter: $F(X)$ instead of $f(X)$.

[2] A simple graph. Multiple edges are not allowed.

There are several simple ways to ensure that this output is finite. It suffices to require that the the input graph be loop-free, but for the reduction below it is more convenient to place an upper bound p on the number of edges in an s-t path. We consider the output of $\#STCON$ as a number given in binary notation (from least to most significant bit). Thus $\#STCON$ is a string function.

Claim 13 *$\#STCON$ is complete for DET under AC^0 reductions.*

Proof of claim 13: First, $\#STCON$ is AC^0-reducible to DET. We show this by proving the stronger fact that $\#STCON \in \#L$.

The graph input to Turing machine M is formatted specifically. Let the graph be represented by its Boolean adjacency matrix G, with s and t two listed vertices. This matrix is encoded as a binary string by means of the *Row* and pairing functions.

M maintains three numbers on its tape: the "current" vertex, the "next" vertex, and a count of the number of edges it has traversed. These numbers are stored in binary. The "current" vertex is initialized to s, and the count is initialized to 0.

When run, M traverses the graph by performing the following algorithm.

TRAVERSE(n, s, t, p, G)

$current \leftarrow s$
$count \leftarrow 0$
while $current \neq t \wedge count \leq p$
    do $next \leftarrow$ nondeterministically-chosen number $< n$
        if $G(current, next)$
            then $current \leftarrow next$
            else halt and reject
        $counter \leftarrow counter + 1$
if $current = t$
    then halt and accept
    else halt and reject

M simulates a traversal of the graph from s to t. Every accepting computation of M traces a path from s to t (of length $\leq p$), and for every path of length $\leq p$ from s to t there is an accepting computation of M. Thus $\#STCON \in DET$.

Next, $\#STCON$ is hard for DET with respect to AC^0-reducibility (Definition 7).

Let $F_M(X) =$ the number of accepting computation paths of nondeterministic logspace Turing machine M on input X.

Below, we show that the string function $\#STCON$ is complete for DET under AC^0-reductions. We Σ_0^B-bit-define an AC^0 function H such that

$$F_M(X) = \#STCON(n, s, t, p, H(X))$$

It follows that $\#STCON$ is Σ_0^B-definable from DET.

Defining H. H can be defined using prior knowledge of M. As usual, logspace M has two tapes, a read-only input tape and a read-write work tape. WLOG, let the work tape be infinite in both directions, and initialized to all zeroes; also, let M work with the alphabet

$\Sigma = \{0, 1, \$\}$,[3] have a "counter" which increments with each step of computation, and have a single accepting state. M runs in logarithmic space and *always* halts. Thus, on input X, M runs in time bounded by $|X|^k + 1$,[4] where k is some constant specific to M.

Configurations of M. Configurations of M are represented by 5-tuples (a, b, c, d, e) of numbers where

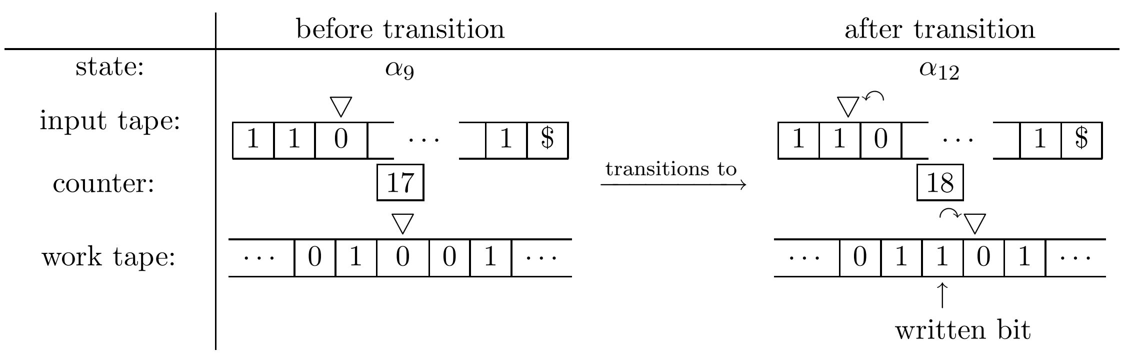
- a is the label of M's current state,
- b is the position of the head on the input tape,
- c is the value of the counter,
- d is the numerical value of the binary contents the work tape to the left of the head, and
- e is the numerical value of the contents of the rest of the work tape, in reverse (so the least significant bit of C is the bit currently being read).

M's configuration represented by (a, b, c, d, e) is encoded as the number $\langle a, b, c, d, e\rangle$ using the pairing function. This encoding and the *left* and *right* projection functions are used below in the Σ_0^B formula representing the bit-graph of H.

Σ_0^B-defining H. The string function $H(X)$ encodes the adjacency matrix of nodes in the graph. $H(X)(\ell, m)$ is true if there is a transition of M from the state encoded by ℓ to the state encoded by m.

$$H(X)(\ell, m) \leftrightarrow \big(\exists a, b, c, d, e \le (\ell + m), \psi\big) \vee$$
$$\exists a, b, c, d, e, a', b', c', d', e' \le \ell, \big(\ell = \langle a, b, c, d, e\rangle \wedge m = \langle a', b', c', d', e'\rangle \wedge \varphi\big)$$

Clauses of φ correspond to transitions of M. The formula φ is in disjunctive normal form. For each possible transition in M's transition relation Δ, φ has a clause specifying that there is an edge in the graph between the nodes representing M's configuration before and after the transition. For example, let M be in state α_9 reading a 0 on its input tape and 0 on its work tape, with counter value 17 (below its cutoff limit). Let one possible transition be to write a 1 to the work tape, move the work tape head right, move the input tape head left, and update the counter.



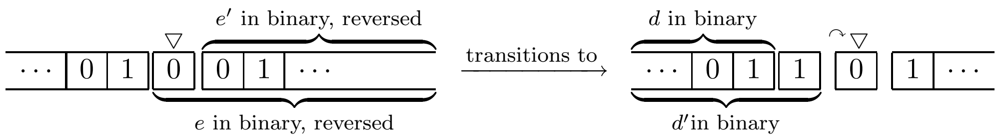
[3] $ is a special symbol which used only to indicate the right end of the input string. It is safe to assume that M keeps track of its input head position and the left end of the input string, e.g., by maintaining a counter of how many bits are to the left of the head.

[4] The $+1$ here ensures that, even on the empty input, M can at least read its input. If a tight time bound is actually, e.g., $5|X|^j$, then $k = j + 1$ will still yield $|X|^k + 1$ a bound. For inputs X short enough that M cannot perform any significant computation within the time bound, M can read the input and then look up the answers in a table.

Then φ contains the clause:

$$(a = 9 \wedge a' = 12 \wedge \neg X(b) \wedge b{+}1 = b' \wedge c \leq \underbrace{|X| \cdot |X| \cdots |X|}_{k \text{ times}} +1 \wedge c' = c{+}1 \wedge 2d{+}1 = d' \wedge 2e' = e)$$

Variables d, d', e, and e' store the numerical value of the binary contents of the work tape.



The right end of the input tape. In the special case when M reads \$ on its input tape, $b = |X|$. For $b \geq |X|$, by definition $X(b)$ is false; however, the intended interpretation is that $X(b)$ is false iff the b^{th} bit of X is zero. To amend this, any clauses of φ corresponding to transitions where M reads \$ from the input tape will include the condition $b = |X|$ instead of $X(b)$ or $\neg X(b)$. It is safe to assume that, having read \$, M *never* moves further right on its input tape.

Specifying s and t. The standard (above) for encoding adjacency matrices specifies that the distinguished nodes s and t be associated with row/column 0 and 1, respectively. Note that 0 and 1 are not in the correct form of encoded 5-tuples, requiring special treatment in ψ, a disjunction of two clauses. The first clause specifies that $H(X)(0, m)$ is true for m encoding M's starting configuration. Let α_0 be M's initial state:

$$a = b = c = d = e = 0 \wedge m = \langle a, b, c, d, e \rangle \wedge \ell = 0$$

The second clause specifies that $H(X)(\ell, 1)$ is true for any ℓ encoding a configuration where M is in its single accepting state α_r:

$$\ell = \langle a, b, c, d, e \rangle \wedge a = r \wedge c \leq \underbrace{|X| \cdot |X| \cdots |X|}_{k \text{ times}} +1) \wedge m = 1$$

Given φ and ψ as described above, we have a Σ_0^B formula representing the bit-graph of H, a function which takes input X to logspace Turing machine M and outputs the adjacency matrix of M's configurations (encoded as a binary string). By construction, the number $F_M(X)$ of accepting paths of machine M on input X is exactly equal to the number of paths from s (the node numbered 0) to t (node 1) in the adjacency matrix encoded in $H(X)$. By setting the bound p on path length to be sufficiently large, we see that $\#STCON$ is hard for $\#L$.

Since $\#STCON$ is both in DET and hard for DET with respect to AC^0 reductions, it is complete for DET. ■

Definition 14 (Matrix Powering) *Given matrix A and integer k, matrix powering is the problem of computing the matrix A^k.*

As a function mapping $(A, k) \longmapsto A^k$, matrix powering does not exactly fit the format required by Definition 9; it is neither a number function nor a bit-graph. Notationally, matrices are represented as bit-strings; thus matrix powering is a string function. In the context of $\#L$, we are interested in the bit-graph of this string function. Observe that the entries of the k^{th} power of even a matrix over $\{0,1\}$ can have k digits, so that the k^{th} power of a matrix must have entries encoded in binary.

Specifying matrices as strings requires an encoding scheme for input matrix entries; *unary matrix powering* and *binary matrix powering* are distinguished according to this number encoding. A unary input matrix is encoded by means of the *entry* function; a binary input matrix is encoded by means of the Row_2 function (defined in Section 2.1).

Remark 15 *Let A be the adjacency matrix of a graph; then $A[i,j]$ is 1 if there is an edge from vertex i to j, and 0 otherwise. Thus $A^1[i,j]$ is the number of edges from i to j, that is, the number of paths of length 1 from i to j. $A^2[i,j] = \sum_{\ell=1}^{n} A[i,\ell]A[\ell,j]$ is the number of paths from i to j passing through one intermediate vertex, i.e., of length exactly 2. Inductively, $A^k[i,j]$ is the number of paths from i to j of length exactly k.*

This observation provides the main insight for the next two lemmas.

Lemma 16 *$\#STCON$ is AC^0-reducible to matrix powering over $\mathbb{N}$.*

Proof: Recall from Definition 12 that $\#STCON$ counts the number of paths of length $\leq p$. Matrix powering counts paths of *exactly* a given length. It suffices to construct an AC^0 function to convert the adjacency matrix G of a graph into the adjacency matrix G' of a different graph, such that every s-t path of length $\leq p$ in G is converted into an s'-t' path of length[5] *exactly* $p+2$ in the graph G'. By notational convention, G and G' are be Boolean matrices; let A' be the number matrix over $\{0,1\}$ corresponding to G', i.e., $A'[i,j]=1$ iff $G'(i,j)$. Thus $\#STCON(n,s,t,p,G) = A'^{(p+2)}[s',t']$.

The new graph G' consists of $p+1$ "layers" of vertices; each layer contains a copy of the vertices of G (with no edges). G additionally has two vertices s' and t' not in any layer. A pair (v,ℓ) denotes the vertex v in layer ℓ; thus $(s,0)$ is the copy of vertex s in the first layer. There is an edge from vertex (v,ℓ) to $(v',\ell+1)$ if there is an edge from v to v' in G. There is an edge from s' to $(s,0)$, an edge from each (t,ℓ) to t', and a self-loop on vertex t'. Vertex (x,y) in G' is identified with the number $\langle x,y\rangle$; also, $s'=0$ and $t'=1$.

Adopting the same convention as above, let the first two nodes listed in A be $s=0$ and $t=1$. Let A be encoded as string X and A' be encoded as $X' = H(p,X)$. The conversion can be AC^0 bit-defined as follows:

$$
\begin{aligned}
H(p,X)(i,j) \quad \leftrightarrow \quad & (i=0 \wedge j=\langle 0,0\rangle) \vee && (4)\\
& (i=1 \wedge j=1) \vee && (5)\\
& \exists k \leq p(i=\langle 1,k\rangle \wedge j=1) \vee && (6)\\
& \exists k < p \exists v,v' \leq |X|(X(v,v') \wedge i=\langle v,k\rangle \wedge j=\langle v',k+1\rangle) && (7)
\end{aligned}
$$

The clause on line (4) ensures that there is an edge from s' to $(s,0)$. Line (5) defines the self-loop on vertex t'. Line (6) adds an edge from each (t,ℓ) to t'. By the condition on line (7), the new graph appropriately "inherits" all edges from the original graph.

This construction ensures that, for every s-t path in G of length k, there is a unique path in G' from $(s,0)$ to (t,k), which can be uniquely extended to a path from s' to t' (passing

[5]The added constant is an artifact of the reduction.

through $(s,0)$ and (t,k)) of length $k+2$. Vertex t' has only one outgoing edge, a self-loop. Thus for every s-t path in G of length $\leq p$, there is a unique path in G' from $(s,0)$ to t' of length exactly $p+2$.

This completes the reduction, as

$$\#STCON(n,s,t,p,G) = A'^{(p+2)}[s',t']$$

Thus $\#STCON$ is reducible to matrix powering. Note that this reduction only requires powering a matrix with entries in $\{0,1\}$. ■

Lemma 17 *Unary matrix powering over* $\mathbb{N}$ *is* AC^0*-reducible to* $\#STCON$*.*

Proof: Matrices above had entries from $\{0,1\}$. As a consequence, the *Row* and pairing functions sufficed to encode them as strings. However, when each matrix entry requires more than one bit to store, another layer of encoding is required. We will use the *entry* function, with definition given above by (2). This allows us to encode a matrix of numbers $A = (x_{ij})$ as a string X, recoverable as $x_{ij} = \mathit{entry}(i,j,X) = (X^{[i]})^j$.

Since A has entries from $\mathbb{N}$, it can be viewed as the adjacency matrix of a multigraph.[6] However, $\#STCON$ is defined only on graphs with at most one edge between an ordered pair of vertices. A multigraph can easily be converted into a standard graph by bisecting each edge with a new vertex. If the former graph G had E edges and V vertices, then the new graph G' will have $2|E|$ edges and $|V|+|E|$ vertices. The number of paths between any two vertices in G remains unchanged in G', and the length of every path is exactly doubled.

The following AC^0-defined string function *Convert* is designed to perform this transformation from multigraph to graph. Let X be the string encoding the unary $n \times n$ adjacency matrix A of the multigraph $G = (V,E)$, where $|V| = n$ and $|E| = \sum_i \sum_j A[i,j]$. Let $n' = n + |E|$. We construct the function *Convert* so that $\mathit{Convert}(X) = A'$ is the string encoding the $n' \times n'$ Boolean adjacency matrix of the graph G'. Let *Convert* be AC^0-bit-defined:

$$\begin{aligned}\mathit{Convert}(X)(k,\ell) \leftrightarrow \exists i,j,c < |X| \quad \big(\quad & (k = \langle i,j,c+n\rangle \wedge \ell = j \wedge c < \mathit{entry}(i,j,X)) \\ & \vee(\ell = \langle i,j,c+n\rangle \wedge k = i \wedge c < \mathit{entry}(i,j,X)) \ \big)\end{aligned}$$

By construction, there are $A(i,j)$ vertices in G' (with labels $\langle i,j,0+n\rangle,\ldots,\langle i,j,c-1+n\rangle$) such that, for each $c \in \{0,\ldots,c-1\}$, both $A'(i,\langle i,j,c+n\rangle)$ and $A'(\langle i,j,c+n\rangle,j)$ are true. (The added n ensures that the vertex labels are unique, that is, there is no new vertex labelled $\langle i,j,c+n\rangle = i'$ for some $i' < n$ a label of a vertex in the original graph.)

Every pair of vertices in G' shares at most one edge; hence a single bit suffices to store each entry of A', and it is not necessary to use the *seq* function. This definition of $\mathit{Convert}(X)$ takes advantage of that simplification by outputting a Boolean matrix.

By construction, for every $i,j \leq n$,

$$A^{k+1}[i,j] = A'^{2(k+1)}[i,j] = \#STCON(n',i,j,2(k+1),A') - \#STCON(n',i,j,2k,A')$$

The number n' is not obviously computable, but is used here for clarity. It can be replaced with a larger number bounding n' from above, e.g., $|A|$. This yields the same output of

[6] Entries of A that are > 1 can be viewed as duplicate edges in the graph, e.g., $A[i,i] = 4$ would mean that node i has four self-loops; $A[i,j] = 3$ would mean that there are three edges from node i to node j.

$\#STCON$, since the string $Convert(X)$ can be "interpreted" on larger numbers: any vertex of index $\geq n$ has no in- or out-edges.

By remark 15, $A^{k+1}[i, j]$ is the number of paths from i to j of length exactly $k + 1$. These paths map uniquely to paths in A' from i to j of length exactly $2(k + 1)$. (Notice that, by construction, paths between vertices of the original graph will always be of even length.) This is exactly the number of paths from i to j of length $\leq 2(k + 1)$ *minus* the number of paths from i to j of length $\leq 2k$. Thus unary matrix powering is AC^0-reducible to $\#STCON$. ■

Claim 18 *Unary matrix powering over* $\mathbb{N}$ *is* AC^0*-complete for* DET*.*

This follows from Lemma 16, Lemma 17, and Claim 13.

3.2 $\oplus L$ and its AC^0-complete problems

Definition 19 $\oplus L$ *is the class of decision problems* $R(\vec{x}, \vec{X})$ *such that, for some function* $F \in \#L$*,*

$$R(\vec{x}, \vec{X}) \leftrightarrow F(\vec{x}, \vec{X})(0)$$

That is, $R(\vec{x}, \vec{X})$ *holds depending on the value of* $F(\vec{x}, \vec{X}) \bmod 2$*.*

$\oplus L$ is also called $ParityL$ or MOD_2L. In general, MOD_kL has a similar definition[7].

Theorem 20 $\oplus L$ *is closed under* AC^0*-reductions.*

Beigel et al. prove that $\oplus L$ is closed under NC^1-reductions [4], which certainly implies it is closed under weaker AC^0-reductions. Chapter 9 of [9] provides a general criterion (Theorem 9.7) for such closure: closure under finitely many applications of composition and string comprehension.

By Definition 14, matrix powering is a function. For convenience in aligning with Definition 19, matrix powering mod 2 can also be considered as a decision problem on tuples $(n, k, \langle i, j\rangle, A)$ where A is a binary $n \times n$ matrix, and the answer is "yes" iff the $(i, j)^{\text{th}}$ entry of A^k is 1. The proof of the following claim is nearly identical to that of Claim 18.

Claim 21 *Matrix powering over* $\mathbb{Z}_2$ *is* AC^0*-complete for* $\oplus L$*.*

4 A theory for $\oplus L$

In this section, we develop a finitely axiomatized theory for the complexity class $\oplus L$. Matrix powering is complete for $\oplus L$ under AC^0 reductions, and has the convenient property that each matrix entry can be stored in a single bit.

[7] That is, MOD_kL is the class of decision problems $R(\vec{x}, \vec{X})$ such that, for some function $F \in \#L$, $R(\vec{x}, \vec{X})$ holds iff $F(\vec{x}, \vec{X}) \neq 0 \bmod k$. Using Fermat's little theorem, this can easily be modified to the two cases $F(x) = 1 \bmod k$ and $F(x) = 0 \bmod k$ for any prime k.

$V^0(2)$ provides the base theory for this section. It is associated with the complexity class $AC^0(2)$, which is AC^0 with the addition of mod 2 gates. Section 9D of [9] develops this theory, and the universal conservative extension $\overline{V^0(2)}$.

The theory $V{\oplus}L$ is obtained from the base theory $V^0(2)$ by adding an axiom which states the existence of a solution to matrix powering over $\mathbb{Z}_2$. $V{\oplus}L$ is a theory over the language $\mathcal{L}_A^2$; below, we detail two methods to obtain the added $\Sigma_1^B(\mathcal{L}_A^2)$ axiom, either by defining it explicitly, or by defining it in the universal conservative extension $\overline{V^0(2)}$ and using results of [9] to find a provably equivalent formula in the base language.

Throughout this section, matrices will be encoded using only the *Row* and pairing functions, taking advantage of the fact that a matrix entry from $\{0,1\}$ can be stored by a single bit of a bit string. For a string W encoding $n \times n$ matrix M, entries are recovered as $M[i,j] = W(i,j)$. (Or, more precisely, $M[i,j] = 1$ iff $W(i,j)$.)

4.1 The theory $V{\oplus}L$

By the nature of its construction, the theory $V{\oplus}L$ corresponds to $\oplus L$. As we will prove, the set of provably total functions of $V{\oplus}L$ exactly coincides with the functions of $F{\oplus}L$ (Definition 8) and the Δ_1^B-definable relations of $V{\oplus}L$ are exactly the relations in $\oplus L$.

Definition 22 (String Identity $ID(n)$**)** *Let the AC^0 string function $ID(n)$ have output the string that encodes the $n \times n$ identity matrix. $ID(n) = Y$ has the $\Sigma_0^B(\mathcal{L}_{FAC^0})$ bit definition:*

$$Y(b) \leftrightarrow \mathit{left}(b) < n \wedge \mathit{Pair}(b) \wedge \mathit{left}(b) = \mathit{right}(b)$$

Definition 23 (Pow_2**)** *Let X be a string representing an $n \times n$ matrix over $\{0,1\}$. Then the string function $\mathit{Pow}_2(n,k,X)$ has output X^k, the string representing the k^{th} power of the same matrix.*[8]

Definition 24 (PowSeq_2**)** *Let X be a string representing an $n \times n$ matrix over $\{0,1\}$, and let X^i be the string representing the i^{th} power of the same matrix. Then the string function $\mathit{PowSeq}_2(n,k,X)$ has output the list of matrices $[ID(n), X, X^2, \ldots, X^k]$.*

Although Definition 14 specifies that matrix powering is the problem of finding X^k, the function PowSeq_2 computes every entry of every power of X up to the k^{th} power. Lemmas 25 and 26 show that Pow_2 and PowSeq_2 are AC^0-reducible to each other. It is more convenient for us to develop the theory $V{\oplus}L$ with an axiom asserting the existence of a solution for PowSeq_2.

Lemma 25 *Pow_2 is AC^0-reducible to PowSeq_2.*

Pow_2 can be $\Sigma_0^B(\mathcal{L}_{FAC^0} \cup \{\mathit{PowSeq}_2\})$-defined by:

$$\mathit{Pow}_2(n,k,X)(i) \leftrightarrow i < \langle n,n \rangle \wedge \mathit{PowSeq}_2(n,k,X)^{[k]}(i) \qquad (8)$$

[8] This section features a slight, but innocuous, abuse of notation: in mathematical expressions, string X stands for the matrix it represents, e.g., X^3 and XY.

Lemma 26 *$PowSeq_2$ is AC^0-reducible to Pow_2.*

$PowSeq_2$ can be $\Sigma_0^B(\mathcal{L}_{FAC^0} \cup \{Pow_2\})$-defined by:

$$PowSeq_2(n,k,X)(i) \leftrightarrow i < \langle k, \langle n,n \rangle \rangle \wedge Pair(i) \wedge Pow_2(n, left(i), X)(right(i)) \qquad (9)$$

We detail two ways to define the relation $\delta_{PowSeq_2}(n,k,X,Y)$ representing the graph of $PowSeq_2(n,k,X) = Y$ in Sections 4.2 and 4.3, below.

Definition 27 *The theory $V \oplus L$ has vocabulary $\mathcal{L}_A^2$ and is axiomatized by $V^0(2)$ and a $\Sigma_1^B(\mathcal{L}_A^2)$ axiom PS_2 (formula 11) stating the existence of a string value for the function $PowSeq_2(n,k,X)$.*

The axiom is obtained below implicitly (formula (11) in Section 4.2) and explicitly (equation (17) in Section 4.3), and is roughly equivalent to the statement "there is some string Z that witnesses the fact that $Y = PowSeq_2(n,k,X)$." Thus it effectively states the existence of a solution to matrix powering over $\mathbb{Z}_2$. Notice that it actually asserts the existence of the entire series of matrices $X^1, X^2, \ldots, X^k$, not just the matrix X^k as specified by Definition 14.

4.2 Implicitly defining the new axiom

Using a series of intuitively "helper" functions, the relation $\delta_{PowSeq_2}(n,k,X,Y)$ can be defined in the language $\mathcal{L}_{FAC^0(2)} \supset \mathcal{L}_A^2$. This method requires the introduction of new function symbols, which can be used to express the axiom PS_2 in $\overline{V^0(2)}$, a universal conservative extension of $V^0(2)$.

Let $G(n,i,j,X_1,X_2)$ be the AC^0 string function which witnesses the computation of the $(i,j)^{\text{th}}$ entry of the $n \times n$ matrix product X_1X_2, bit-defined as:

$$G(n,i,j,X_1,X_2)(b) \leftrightarrow b < n \wedge X_1(i,b) \wedge X_2(b,j)$$

Each bit of $G(n,i,j,X_1,X_2)$ is the pairwise product of the bits in the i^{th} row of X_1 and the j^{th} column of X_2. Thus the $(i,j)^{\text{th}}$ entry of the matrix product X_1X_2 mod 2 is 1 if and only if $PARITY(G(n,i,j,X_1,X_2))$ holds. $Prod_2(n,X_1,X_2)$, the string function computing the product of two matrices, can be bit-defined as:

$$Prod_2(n,X_1,X_2)(i,j) \leftrightarrow i < n \wedge j < n \wedge PARITY(G(n,i,j,X_1,X_2))$$

That is, the $(i,j)^{\text{th}}$ bit mod 2 of the product matrix X_1X_2 is $\sum_{b=0}^{n-1} X_1(i,b)X_2(b,j)$ mod 2. Each bit of the string $G(n,i,j,X_1,X_2)$ is one of the terms of this sum; hence the parity of $G(n,i,j,X_1,X_2)$ is exactly the desired bit.

Observe that G and $Prod_2$, as well as $\langle \cdot, \cdot \rangle$ and $Pair$ (from Section 2), have $\Sigma_0^B(\mathcal{L}_{FAC^0(2)})$ definitions and are $AC^0(2)$ functions. Let $\delta_{PowSeq_2}(n,k,X,Y)$ be the $\Sigma_0^B(\mathcal{L}_{FAC^0(2)})$ formula

$$\forall b < |Y|, |Y| < \langle k, \langle n,n \rangle \rangle \wedge (Y(b) \supset Pair(b)) \wedge Y^{[0]} = ID(n) \wedge \\ \forall i < k(Y^{[i+1]} = Prod_2(n,X,Y^{[i]})) \qquad (10)$$

This formula asserts that the string Y is the output of $\mathit{PowSeq}_2(n, k, X)$. The "unimportant" bits – i.e., the bits of Y that do not encode a piece of the list – are required to be zero. Thus $\mathit{PowSeq}_2(n, k, X)$ is the lexographically first string that encodes the list of matrices $[X^1, X^2, \ldots, X^k]$.

$\overline{V^0(2)}$ is a conservative universal extension of $V^0(2)$, defined in Section 9D of [9]. Theorem 9.67(b) of [9] asserts that there is a $\mathcal{L}_A^2$ term t and a $\Sigma_0^B(\mathcal{L}_A^2)$ formula $\alpha_{\mathit{PowSeq}_2}$ such that

$$\exists Z < t, \alpha_{\mathit{PowSeq}_2}(n, k, X, Y, Z)$$

is provably equivalent to (10) in the theory $\overline{V^0(2)}$.

The axiom PS_2 used to define the theory $V\oplus L$ is

$$\exists Y < m, \exists Z < t, \alpha_{\mathit{PowSeq}_2}(n, k, X, Y, Z) \tag{11}$$

Formulas (10) and (11) each specify that Y witnesses the intermediate strings X^1, X^2, $\ldots$, X^k. String Y is *not* required to witness any of the work performed in calculating each entry of each product $X^j = X \times X^{j-1}$. In Section 4.3, Y is used to witness *all* of the intermediate work, by using the notation $Y^{[x][y]}$ from Section 2.

Lemma 28 *The matrix powering function PowSeq_2 is $\Sigma_1^B(\mathcal{L}_A^2)$-definable in $V\oplus L$.*

Proof: By construction,

$$\mathit{PowSeq}_2(n, k, X) = Y \leftrightarrow \exists Z < t, \alpha_{\mathit{PowSeq}_2}(n, k, X, Y, Z)$$

We need to show that

$$V\oplus L \vdash \forall n, k \forall X \exists! Y \exists Z < t, \alpha_{\mathit{PowSeq}_2}(n, k, X, Y, Z)$$

The axiom PS_2 guarantees that such Y and Z exist. (The function served by Z is made explicit in Section 4.3.) Formula (10) uniquely specifies every bit of Y using n, k, and X. Consider the Σ_0^B-formula stating that the first i bits of Y are unique. By induction on this formula, Y can be proved to be unique in the conservative extension $\overline{V^0(2)}$ together with the defining axiom for PowSeq_2. Thus $V\oplus L$ also proves that Y is unique. ■

Corollary 29 *Pow_2 is $\Sigma_1^B(\mathcal{L}_A^2)$-definable in $V\oplus L$.*

This corollary follows from Lemmas 28 and 25 above.

In Lemma 30 below, we show similarly that the aggregate function $\mathit{PowSeq}_2^\star$ is Σ_1^B-definable in $V\oplus L$. Recall from Chapter 8 of [9] (Definition 8.9) that the aggregate function $\mathit{PowSeq}_2^\star$ is the polynomially bounded string function that satisfies

$$|\,\mathit{PowSeq}_2^\star(b, W_1, W_2, X)| \leq \langle b, \langle |W_2|, \langle |W_1|, |W_1| \rangle\rangle\rangle$$

and

$$\mathit{PowSeq}_2^\star(b, W_1, W_2, X)(i, v) \leftrightarrow i < b \wedge \mathit{PowSeq}_2((W_1)^i, (W_2)^i, X^{[i]})(v)$$

The strings W_1, W_2, and X encode b-length lists of inputs to each place of PowSeq_2: W_1 encodes the list of numbers $n_0, n_1, n_2, \ldots, n_{b-1}$; W_2 encodes the list of numbers $k_0, k_1, \ldots, k_{b-1}$; and X encodes the list of strings $X_0, X_1, \ldots, X_{b-1}$; we are interested in raising the $n_i \times n_i$ matrix encoded in X_i to the powers $1, 2, \ldots, k_i$. The string function $\mathit{PowSeq}_2^\star$ computes *all* these lists of powers; it aggregates many applications of the function PowSeq_2.

Lemma 30 *The aggregate matrix powering function $PowSeq_2^\star$ is $\Sigma_1^B(\mathcal{L}_A^2)$-definable in $V{\oplus}L$.*

Proof: For $\delta_{PowSeq_2^\star}$ a Σ_1^B formula, we need to show both:

$$PowSeq_2^\star(b, W_1, W_2, X) = Y \leftrightarrow \delta_{PowSeq_2^\star}(b, W_1, W_2, X, Y) \tag{12}$$

and

$$V{\oplus}L \vdash \forall b \forall X, W_1, W_2 \exists! Y \delta_{PowSeq_2^\star}(b, W_1, W_2, X, Y)$$

We introduce the AC^0 functions max and S in order to simplify the definition of $\delta_{PowSeq_2^\star}$ over $\overline{V^0(2)}$, and then use Theorem 9.67(b) of [9] to obtain a provably equivalent $\Sigma_1^B(\mathcal{L}_A^2)$ formula. Since $\overline{V^0(2)}$ is a conservative extension of $V^0(2)$, this suffices to show that $PowSeq_2^\star$ is $\Sigma_1^B(\mathcal{L}_A^2)$-definable in $V{\oplus}L$. In order to do so, we must establish a $\Sigma_1^B(\mathcal{L}_{FAC^0(2)})$ formula equivalent to the desired formula $\delta_{PowSeq_2^\star}$.

Let the function $max(n, W)$ yield the maximum number from a list of n numbers encoded in string W:

$$max(n, W) = x \leftrightarrow \exists i < n \forall j < n, x = (W)^i \geq (W)^j \tag{13}$$

The string function S can be bit-defined as follows. Consider two strings W_1, representing a list of b numbers, and X, representing a list of b matrices as above. The function $S(b, W_1, X)$ returns the matrix with matrices X_i (appropriately padded with zeroes) on the diagonal, and all other entries zero. Let $m = max(b, W_1)$. Let X_i' be the $m \times m$ matrix X_i padded with columns and rows of zeroes:

$$\begin{array}{cc} & \begin{array}{cc} X_i & \overbrace{\begin{array}{ccc} 0 & \dots & 0 \end{array}}^{m-n_i} \end{array} \\ m - n_i \left\{ \begin{array}{c} 0 \\ \vdots \\ 0 \end{array} \right. & \begin{array}{ccc} \ddots & & \vdots \\ & \ddots & \\ \dots & & 0 \end{array} \end{array}$$

Then $S(b, Y_1, X)$ is the string encoding the matrix:

$$\begin{bmatrix} X_0' & & & 0 \\ & X_2' & & \\ & & \ddots & \\ 0 & & & X_{b-1}' \end{bmatrix}$$

All entries not in the matrices along the diagonal are 0.

The string function S can be bit-defined:

$$\begin{aligned} S(b, W_1, X)(i, j) \quad \leftrightarrow \quad & \exists a < b, \exists i', j' < (W_1)^a, i < m \wedge j < m \wedge \\ & i = i' + m \wedge j = j' + m \wedge X^{[a]}(i', j') \end{aligned}$$

By convention, the unspecified bits (i.e., bits b that are not pair numbers) are all zero. This bit-definition ensures that the string $S(b, W_1, X))$ is uniquely defined.

We will use this matrix $S(b, W_1, X)$ and the existence and uniqueness of its sequence of matrix powers to show the existence and uniqueness of the aggregate matrix powering function.

Let n_{max} denote $max(b, W_1)$ and k_{max} denote $max(b, W_2)$. Recall Definition 10 of $\delta_{PowSeq_2}(n, k, X)$. By convention, the aggregate function of $PowSeq_2$ is defined as:

$$\begin{aligned} PowSeq_2^\star(b, W_1, W_2, X) = Y \leftrightarrow |Y| < \langle b, \langle k_{max}, \langle n_{max}, n_{max}\rangle\rangle\rangle \wedge \\ \forall j < |Y|, \forall i < b, \left[(Y(j) \supset Pair(j)) \wedge \delta_{PowSeq_2}((W_1)^i, (W_2)^i, X^{[i]}, Y^{[i]})\right] \end{aligned} \quad (14)$$

The right-hand side of (14) is the relation $\delta_{PowSeq_2^\star}(b, W_1, W_2, X, Y)$; it has a provably equivalent $\Sigma_1^B(\mathcal{L}_A^2)$-formula $\exists Z < t, \alpha_{PowSeq_2^\star}(b, W_1, W_2, X, Y, Z)$, used as the definition for $PowSeq_2^\star$ over $V{\oplus}L$.

It remains to prove the existence and uniqueness for $PowSeq_2^\star(b, W_1, W_2, X)$. The functions max and S allow for a straightforward proof based on the existence and uniqueness of the string $A = PowSeq_2(b \cdot n_{max}, k_{max}, S(b, W_1, X))$, where $n_{max} = max(b, W_1)$ and $k_{max} = max(b, W_2)$.

We would like to define the string $B = PowSeq_2^\star(b, W_1, W_2, X)$ from A. Notice that B encodes a list of strings, each of which represents a power of the matrix $S(b, W_1, X)$. The string A encodes nearly the same information, but in a different format: A is a list of *lists*, each of which encodes the powers of a matrix from the list X of matrices.

Observe that:

$$S(b, W_1, X)^i = \begin{bmatrix} X_0' & & & 0 \\ & X_2' & & \\ & & \ddots & \\ 0 & & & X_{b-1}' \end{bmatrix}^i = \begin{bmatrix} X_0'^i & & & 0 \\ & X_2'^i & & \\ & & \ddots & \\ 0 & & & X_{b-1}'^i \end{bmatrix}$$

Also,

$$X_j'^i = \begin{array}{r} \\ m - n_j \left\{ \begin{array}{c} \\ \\ \\ \end{array} \right. \end{array} \begin{array}{cccc} X_j^i & \overbrace{\begin{array}{ccc} 0 & \dots & 0 \end{array}}^{m-n_j} \\ 0 & \ddots & & \vdots \\ \vdots & & \ddots & \\ 0 & \dots & & 0 \end{array}$$

Thus we can "look up" the required powers of each matrix. Let A and B be shorthand:

$$A = PowSeq_2(b \cdot n_{max}, k_{max}, S(b, W_1, X))$$
$$B = PowSeq_2^\star(b, W_1, W_2, X)$$

Then we can define $PowSeq_2^\star$ from $PowSeq_2$ as follows.

$$\begin{aligned} B^{[m][p]}(i, j) \quad \leftrightarrow \quad & m < b \wedge p \leq (W_2)^m \wedge i < (W_1)^m \wedge j < (W_1)^m \wedge \\ & \exists m' < m, m' + 1 = m \wedge A^{[p]}(n_{max} \cdot m' + i, n_{max} \cdot m' + j) \end{aligned} \quad (15)$$

Here, m represents the number of the matrix in the list X, p represents the power of matrix X_m, and i and j represent the row and column; thus the formula above defines the bit $(X_m^p)(i, j)$ for all matrices X_m in the list X. By shifting around the pieces of this formula and adding quantifiers for m, p, i, and j, it is clear that (15) can be translated into the appropriate form for a Σ_1^B-definition of the graph of $PowSeq_2^\star$.

Thus existence and uniqueness of $PowSeq_2^\star(b, W_1, W_2, X)$ follow from existence and uniqueness of $PowSeq_2(b \cdot n_{max}, k_{max}, S(b, W_1, X)$. Since max and S are AC^0 functions, they can be used without increasing the complexity of the definition (by use of Theorem 9.67(b) of [9], as above). ■

4.3 Explicitly defining the new axiom

The application of Theorem 9.67(b) in the previous section allowed for an easy conversion from a readable formula (10) expressing the graph $\delta_{PowSeq_2}(n,k,X,Y)$ to a $\Sigma_1^B(\mathcal{L}_A^2)$ formula (11) for PS_2. In this section, we explicitly write a $\Sigma_1^B(\mathcal{L}_A^2(PARITY))$ formula for PS_2 without helper functions. This formula clarifies the function of string Z in formula (11).

We can write an explicit $\Sigma_1^B(\mathcal{L}_{FAC^0(2)})$ formula for PS_2 using Z to witness *all* of the intermediate work done in computing each bit of each power of X: $X^1, X^2, \ldots, X^k$. Recall from Section 2 the notation $Z^{[i]}$ for the string in the i^{th} row of the list of strings encoded in Z, and $Z^{[i][j]}$ for the $(i,j)^{\text{th}}$ string entry of the two-dimensional matrix of strings encoded in string Z. In particular, the (very long!) string Z encodes a list of k strings $Z^{[\ell]}$ such that for for $\ell < k$, $Z^{[\ell][\langle i,j\rangle]}$ is the string witnessing the computation of the $(i,j)^{\text{th}}$ bit of $X^{\ell+1}$ for $i,j<n$.

With this organization of Z, $\delta_{PowSeq_2}(n,k,X,Y)$ can be $\Sigma_1^B(\mathcal{L}_{FAC^0(2)})$ defined as:

$$\exists Z < \langle k, \langle n,n\rangle\rangle, \big(Y^{[0]} = ID(n)\big) \wedge \Big(\forall \ell < k, \forall i,j < n, \big(Z^{[\ell+1][\langle i,j\rangle]}(b) \leftrightarrow X(i,b) \wedge Y^{[\ell]}(b,j)\big) \wedge Y^{[\ell+1]}(i,j) \leftrightarrow PARITY(Z^{[\ell+1][\langle i,j\rangle]})\Big) \quad (16)$$

For all $0 \le \ell \le k$, the string $Y^{[\ell]}$ encodes the ℓ^{th} power of matrix X. The sequence of matrices X^0, X^1, X^2, …, X^k are encoded as strings $Y^{[0]}, Y^{[1]}, \ldots, Y^{[k]}$.

We require a $\Sigma_1^B(\mathcal{L}_A^2)$ axiom for $\exists Y < m, \delta_{PowSeq_2}(n,k,X,Y)$. Unfortunately, formula (16) above includes $PARITY$, which is not a function in $\mathcal{L}_A^2$. The usage of $PARITY$ is most conveniently eliminated by application of Theorem 9.67(b), as in the previous section, obtaining a $\Sigma_1^B(\mathcal{L}_A^2)$ formula $\exists Z < t, \beta_{PowSeq_2}(n,k,X,Y,Z)$ which $\overline{V^0(2)}$ proves equivalent to (16). This formula can be made explicit by the construction in the proof of the First Elimination Theorem (9.17), which forms the basis of Theorem 9.67(b). This construction provides no special insight for the present case, however, and so we omit it here.

The $\Sigma_1^B(\mathcal{L}_A^2)$ axiom PS_2 for the theory $V\!\oplus\! L$ is:

$$\exists Y < m, \exists Z < t, \beta_{PowSeq_2}(n,k,X,Y,Z) \quad (17)$$

Notice that this appears identical to the axiom (11) obtained in Section 4.2 by less explicit means. The differences between (11) and (17) are obscured in the formulae α_{PowSeq_2} and β_{PowSeq_2}.

4.4 The theory $\overline{V\!\oplus\! L}$

Here we develop the theory $\overline{V\!\oplus\! L}$, a universal conservative extension of $V\!\oplus\! L$. Its language $\mathcal{L}_{F\oplus L}$ contains function symbols for all string functions in $F\!\oplus\! L$. The defining axioms for the functions in $\mathcal{L}_{F\oplus L}$ are based on their AC^0 reductions to the matrix powering function. Additionally, $\overline{V\!\oplus\! L}$ has a quantifier-free defining axiom for $PowSeq_2'$, a string function with inputs and output the same as $PowSeq_2$.

Formally, we leave the definition of $PowSeq_2$ unchanged; let $PowSeq_2'$ be the function with the same value as $PowSeq_2$ but the following quantifier-free defining axiom:

$$|Y| < \langle k, \langle n,n\rangle\rangle \wedge \big(b < |Y| \supset (\neg Y(b) \vee Pair(b))\big) \wedge Y^{[1]} = X \wedge \big(i < k \supset Y^{[i+1]} = Prod_2(n, X, Y^{[i]})\big) \quad (18)$$

Notice that this formula is similar to (10), but has new free variables b and i. The function $PowSeq_2$ satisfies this defining axiom for $PowSeq'_2$. $V^0(2)$, together with both defining axioms, proves $PowSeq_2(n,k,X) = PowSeq'_2(n,k,X)$.

We use the following notation. For a given formula $\varphi(z,\vec{x},\vec{X})$ and $\mathcal{L}^2_A$-term $t(\vec{x},\vec{X})$, let $F_{\varphi,t}(\vec{x},\vec{X})$ be the string function with bit definition

$$Y = F_{\varphi(z),t}(\vec{x},\vec{X}) \leftrightarrow z < t(\vec{x},\vec{X}) \wedge \varphi(z,\vec{x},\vec{X}) \tag{19}$$

Definition 31 **($\mathcal{L}_{F\oplus L}$)** *We inductively define the language $\mathcal{L}_{F\oplus L}$ of all functions with (bit) graphs in $\oplus L$. Let $\mathcal{L}^0_{F\oplus L} = \mathcal{L}_{FAC^0(2)} \cup \{PowSeq'_2\}$. Let $\varphi(z,\vec{x},\vec{X})$ be an open formula over $\mathcal{L}^i_{F\oplus L}$, and let $t = t(\vec{x},\vec{X})$ be a $\mathcal{L}^2_A$-term. Then $\mathcal{L}^{i+1}_{F\oplus L}$ is $\mathcal{L}^i_{F\oplus L}$ together with the string function $F_{\varphi(z),t}$ that has defining axiom:*

$$F_{\varphi(z),t}(\vec{x},\vec{X})(z) \leftrightarrow z < t(\vec{x},\vec{X}) \wedge \varphi(z,\vec{x},\vec{X}) \tag{20}$$

Let $\mathcal{L}_{F\oplus L}$ be the union of the languages $\mathcal{L}^i_{F\oplus L}$. Thus $\mathcal{L}_{F\oplus L}$ is the smallest set containing $\mathcal{L}_{FAC^0(2)} \cup \{PowSeq'_2\}$ and with the defining axioms for the functions $F_{\varphi(z),t}$ for every open $\mathcal{L}_{F\oplus L}$ formula $\varphi(z)$.

Notice that $\mathcal{L}_{F\oplus L}$ has a symbol for every string function in $F\oplus L$. By Exercise 9.2 of [9], for every number function f in $F\oplus L$, there is a string function F in $F\oplus L$ such that $f = |F|$. Thus there is a term in $\mathcal{L}_{F\oplus L}$ for every number function in $F\oplus L$.

Definition 32 **($\overline{V\oplus L}$)** *The universal theory $\overline{V\oplus L}$ over language $\mathcal{L}_{F\oplus L}$ has the axioms of $\overline{V^0(2)}$ together with the quantifier-free defining axiom (18) for $PowSeq'_2$ and the above defining axioms (20) for the functions $F_{\varphi(z),t}$.*

Theorem 33 *$\overline{V\oplus L}$ is a universal conservative extension of $V\oplus L$.*

Proof: Obviously, $\overline{V\oplus L}$ is a universal theory. It is an extension of $V\oplus L$ because all axioms of $V\oplus L$ are theorems of $\overline{V\oplus L}$.

To show that it is conservative, we build a sequence of theories $\mathcal{T}_i$ analogous to the languages $\mathcal{L}^i_{F\oplus L}$, and show that each $\mathcal{T}_{i+1}$ is a universal conservative extension of $\mathcal{T}_i$. By Theorem 5.27 in [9] (extension by definition), it suffices to show that F_{i+1} is definable in $\mathcal{T}_i$. (Alternatively, (20) provides a bit-definition of F_{i+1}, so Corollary 5.39 of [9] gives the same result.)

Let $\mathcal{T}_0 = V\oplus L$, and let $\mathcal{T}_{i+1}$ be the language obtained from T_i by adding a new function F_{i+1} of the form $F_{\varphi(z),t}$ and its defining axiom (20), where $\varphi(z)$ is a quantifier-free formula in the language $\mathcal{L}^i$ of $\mathcal{T}^i$. Thus $\overline{V\oplus L}$ extends every theory $\mathcal{T}^i$; it is their union.

$$\overline{V\oplus L} = \bigcup_{i\geq 0} \mathcal{T}^i$$

The rest of the proof consists of proving the claim that, for each $i \geq 0$, the function F_{i+1} is definable in $\mathcal{T}^i$. We proceed by induction, using results from [9].

As a base case, $V\oplus L$ proves $\Sigma^B_0(\mathcal{L}_{FAC^0(2)})$-COMP by Lemma 9.67(c), since it extends $V^0(2)$. Formula (10) gives a $\Sigma^B_0(\mathcal{L}_{FAC^0(2)})$ defining axiom for $PowSeq_2$ (and $PowSeq'_2$). By

Lemma 9.22, $V\oplus L(PowSeq_2, PowSeq_2^\star)$ proves (21) for $PowSeq_2$; also, both $PowSeq_2$ and $PowSeq_2^\star$ are $\Sigma_0^B(\mathcal{L}_{FAC^0(2)})$-definable in $V\oplus L$. (Notice that formulas (10) and (14) provide exactly these definitions for $PowSeq_2$ and $PowSeq_2^\star$, respectively.) By Lemma 9.22 and Theorem 8.15, $V\oplus L$ proves $\Sigma_0^B(\mathcal{L}_{FAC^0(2)} \cup \{PowSeq_2\})$-COMP.

Inductively, $\mathcal{T}^i$ proves $\Sigma_0^B(\mathcal{L}^{i-1})$-COMP, F_i and F_i^* are $\Sigma_0^B(\mathcal{L}^{i-1})$-definable in $\mathcal{T}^i$, and $\mathcal{T}^i(F_i, F_i^*)$ proves equation (165):

$$\forall i < b, F_i^*(b, \vec{Z}, \vec{X})^{[i]} = F_i((Z_1)^i, \ldots, (Z_k)^i, X_1^{[i]}, \ldots, X_n^{[i]}) \tag{21}$$

Thus by Theorem 8.15, $\mathcal{T}^i$ proves $\Sigma_0^B(\mathcal{L}^i)$-COMP.

By construction, $F_{i+1}(\vec{x}, \vec{X})$ has defining axiom (19) for φ some open $\mathcal{L}^i$ formula. Thus by Lemma 9.22, F_{i+1} is definable in $\mathcal{T}^i$. (Also, the next inductive hypothesis established: $\mathcal{T}^{i+1}$ extends $\mathcal{T}^i$, which can $\Sigma_0^B(\mathcal{L}^i)$-define F_i and F_i^* and prove (21) for F_i and F_i^*.)

■

4.5 Provably total functions of $V\oplus L$

The remaining results follow directly from [9]. We restate them here for convenience.

Claim 34 *The theory $\overline{V\oplus L}$ proves the axiom schemes $\Sigma_0^B(\mathcal{L}_{FAC^0(2)})$-COMP, $\Sigma_0^B(\mathcal{L}_{FAC^0(2)})$-IND, and $\Sigma_0^B(\mathcal{L}_{FAC^0(2)})$-MIN.*

See Lemma 9.23 [9].

Claim 35 *(a) A string function is in $F\oplus L$ if and only if it is represented by a string function symbol in $\mathcal{L}_{F\oplus L}$.*

(b) A relation is in $\oplus L$ if and only if it is represented by an open formula of $\mathcal{L}_{F\oplus L}$ if and only if it is represented by a $\Sigma_0^B(\mathcal{L}_{F\oplus L})$ formula.

See Lemma 9.24 [9].

Corollary 36 *Every $\Sigma_1^B(\mathcal{L}_{F\oplus L})$ formula φ^+ is equivalent in $\overline{V\oplus L}$ to a $\Sigma_1^B(\mathcal{L}_A^2)$ formula φ.*

See Corollary 9.25 [9].

Corollary 37 *(a) A function is in $F\oplus L$ iff it is $\Sigma_1^B(\mathcal{L}_{F\oplus L})$-definable in $\overline{V\oplus L}$ iff it is $\Sigma_1^B(\mathcal{L}_A^2)$-definable in $\overline{V\oplus L}$.*

(b) A relation is in $\oplus L$ iff it is $\Delta_1^B(\mathcal{L}_{F\oplus L})$-definable in $\overline{V\oplus L}$ iff it is $\Delta_1^B(\mathcal{L}_A^2)$-definable in $\overline{V\oplus L}$.

See Corollary 9.26 [9].

The next two theorems follow from Corollary 6 and Theorem 33. Their analogs are Theorem 9.10 and Corollary 9.11 [9].

Theorem 38 *A function is provably total (Σ_1^1-definable) in $V\oplus L$ iff it is in $F\oplus L$.*

Theorem 39 *A relation is in $\oplus L$ iff it is Δ_1^B-definable in $V\oplus L$ iff it is Δ_1^1-definable in $V\oplus L$.*

5 A theory for $\#L$

In this section, we develop a finitely axiomatized theory for the complexity class DET, the AC^0-closure of $\#L$. Claims 13 and 18 showed that $\#STCON$ and unary matrix powering over $\mathbb{N}$ are complete for DET.

By adopting the notation DET, we have invoked the fact that computing the determinant of an integer-valued matrix is AC^0-complete for $\#L$. However, the established notation has only two types: numbers $\in \mathbb{N}$ and binary strings. In this section it will be convenient to be able to capture negative numbers, so we introduce a method for encoding them as strings. This binary encoding will also be convenient in computing the determinant, which we expect to be a large value.

VTC^0 provides the base theory for this section. It is associated with the complexity class TC^0, which is AC^0 with the addition of threshold gates. Section 9C of [9] develops this theory, and its universal conservative extension $\overline{VTC^0}$. There, it is shown that the function *numones* is AC^0-complete for TC^0 (where $numones(y, X)$ is the number of elements of X that are $< y$, i.e., the number of 1 bits of X among its first y bits.) Thus $\mathcal{L}_{VTC^0}$ is $\mathcal{L}_A^2 \cup \{numones\}$. VTC^0 extends V^0 by the addition of an axiom for *numones*.

The theory $V\#L$ is obtained from the theory VTC^0 by the addition of an axiom which states the existence of a solution to matrix powering over $\mathbb{Z}$. It is a theory over the base language $\mathcal{L}_A^2$. The added $\Sigma_1^B(\mathcal{L}_A^2)$ axiom is obtained below by a method similar to the previous section. The universal conservative extension $\overline{V\#L}$ is obtained as before, and its language $\mathcal{L}_{F\#L}$ contains symbols for all string functions of $F\#L$.

5.1 Encoding integers in bit-strings

We encode an integer $x \in \mathbb{Z}$ as a bit-string X in the following way. The first bit $X(0)$ indicates the "negativeness" (sign) of x: $x < 0$ iff $X(0)$. The rest of X consists of a binary representation of x, from least to most significant bit. This section defines addition and multiplication for integers encoded in this way, and extends the similar functions given in [9] for binary encodings of natural numbers.

[9] includes notation for encoding a number $n \in \mathbb{N}$ as a binary string X.

$$bin(X) = \sum_i X(i) \cdot 2^i$$

Addditionally, for X and Y two strings, the string functions "binary addition" $X + Y$ and "binary multiplication" $X \times Y$ are defined (Sections 4C.2 and 5.2).

Our string X is shifted one bit to accomodate the sign of integer x. We define the number function *intsize* analogously, to map from a binary string X (representing an integer) to its size $|X|$. Thus the integer x can be recovered[9] as $(-1)^{X(0)} \cdot intsize(X)$.

$$intsize(X) = \sum_i X(i+1) \cdot 2^i$$

We write $X +_{\mathbb{Z}} Y = Z$ for the string function "integer addition" and $X \times_{\mathbb{Z}} Y = Z$ for

[9] This is a slight abuse of notation, since $X(i)$ is true/false, not one/zero-valued.

the string function "integer multiplication." Define the relations $R_{+_\mathbb{Z}}$ and $R_{\times_Z}$ by

$$R_{+_\mathbb{Z}}(X,Y,Z) \leftrightarrow (-1)^{Z(0)} \cdot \mathit{intsize}(Z) = (-1)^{X(0)} \cdot \mathit{intsize}(X) + (-1)^{Y(0)} \cdot \mathit{intsize}(Y)$$

$$R_{\times_\mathbb{Z}}(X,Y,Z) \leftrightarrow \mathit{intsize}(Z) = \mathit{intsize}(X) \times \mathit{intsize}(Y) \wedge (Z(0) \leftrightarrow (X(0) \oplus Y(0)))$$

Here, $\oplus$ represents exclusive or.

The bit-definitions for these functions will be similar to the definitions for binary numbers in [9] (binary addition in Chapter 4, binary multiplication in Chapter 9), and additionally handle the complication of having signed numbers.

We adopt the relation *Carry* from [9], and modify it to work with encoded integers. $\mathit{Carry}_\mathbb{Z}(i,X,Y)$ holds iff both integers represented by X and Y have the same sign, and there is a carry into bit i when computing $X+Y$.

$$\begin{aligned}\mathit{Carry}_\mathbb{Z}(i,X,Y) \leftrightarrow \quad & (X(0) \leftrightarrow Y(0)) \wedge \\ & \exists k < i, k > 0 \wedge X(k) \wedge Y(k) \wedge \forall j < i[k < j \supset (X(j) \vee Y(j))]\end{aligned}$$

Notice that *Carry* includes the check that X and Y have the same sign.

When the integers have different signs, we need to perform subtraction. The order of subtraction will depend on which of the two integers X and Y has larger size. The relation $\mathit{FirstIntDominates}(X,Y)$ holds iff $\mathit{intsize}(X) > \mathit{intsize}(Y)$.

$$\begin{aligned}\mathit{FirstIntDominates}(X,Y) \leftrightarrow |X| \geq |Y| \wedge \quad & \exists k \leq |X|, (X(k) \wedge \neg Y(k) \wedge \\ & (\forall j \leq |X|, (k < j \wedge Y(j)) \supset X(j))\end{aligned}$$

Suppose that $\mathit{intsize}(X) > \mathit{intsize}(Y)$ and the integers have different signs. Then we can think of the subtraction $\mathit{intsize}(X) - \mathit{intsize}(Y) = \mathit{intsize}(Z)$ as computing, bit by bit, the integer Z which adds to Y to obtain X. The relation $\mathit{Borrow}(i,X,Y)$ holds iff $\mathit{intsize}\, X > \mathit{intsize}\, Y$ and there is a carry from bit i when performing the addition $Z+Y$ (that is, the i^{th} bit of X is "borrowed from" in the subtraction).

$$\begin{aligned}\mathit{Borrow}(i,X,Y) \leftrightarrow \quad & (X(0) \leftrightarrow \neg Y(0))\, \mathit{FirstIntDominates}(X,Y) \wedge \\ & \exists k < i, k > 0 \wedge \neg X(k) \wedge Y(k) \wedge \forall j < i(k < j \supset (\neg X(j) \vee Y(j)))\end{aligned}$$

Notice that *Borrow* includes the check that X and Y have different signs, and $\mathit{intsize}(X) > \mathit{intsize}(Y)$.

Given these relations, it is now possible to bit-define integer addition:

$$\begin{aligned}(X +_\mathbb{Z} Y)(i) \quad \leftrightarrow \quad & \big[\big(i = 0 \wedge X(0) \wedge \mathit{FirstIntDominates}(X,Y)\big) \vee & (22)\\ & \big(i = 0 \wedge Y(0) \wedge \mathit{FirstIntDominates}(Y,X)\big)\big] & (23)\\ & \vee \big[i > 0 \wedge X(i) \oplus Y(i) \oplus & (24)\\ & \big(\mathit{Carry}_\mathbb{Z}(i,X,Y) \vee \mathit{Borrow}(i,X,Y) \vee \mathit{Borrow}(i,Y,X)\big)\big] & (25)\end{aligned}$$

Lines 22 and 23 ensure that the sign of the resulting integer is correct. The clauses on line 25 are mutually exclusive; at most one of them can be true. Notice that each of these clauses applies to a particular case:

- $\mathit{Carry}_\mathbb{Z}(i,X,Y)$ applies when X and Y have the same sign;

- $Borrow(i, X, Y)$ applies when $intsize(X) > intsize(Y)$ and X and Y have different signs; and
- $Borrow(i, Y, X)$ applies when $intsize(Y) > intsize(X)$ and X and Y have different signs.

In the special case when $intsize(X) = intsize(Y)$ and X and Y have different signs, neither of the *Borrow* clauses will apply. This has the desired effect: all bits of $X +_{\mathbb{Z}} Y$ will be zero. (Notice that there is only one valid encoding of zero, as the all-zero string +0.)

Binary multiplication is Σ_1^B-definable in VTC^0 by results from Section 9C.6 of [9]. It is easily adaptable to integer multiplication. Define the string function *BinaryPart* such that, for an integer encoded as string X, $bin(BinaryPart(X)) = intsize(X)$.

$$BinaryPart(X)(i) \leftrightarrow X(i+1)$$

This function simply extracts the part of the string encoding the number, and allows for an easy definition of integer multiplication.

$$\begin{aligned}(X \times_{\mathbb{Z}} Y)(i) \quad \leftrightarrow \quad & \big(i = 0 \wedge (X(0) \leftrightarrow \neg Y(0))\big) \vee \\ & \big(i > 0 \wedge \exists k < i, k+1 = i \wedge (BinaryPart(X) \times BinaryPart(Y))(k)\big) \quad (26)\end{aligned}$$

The formulas given by (22 – 25) and (26) define integer addition $+_{\mathbb{Z}}$ and integer multiplication $\times_{\mathbb{Z}}$. In order to use them in the next section, we will work in $\overline{VTC^0}$.

5.2 Additional complete problems for $\#L$

Eventually, we would like to justify the fact that $AC^0(\#L) = DET$ by formalizing the computation of integer determinants in $V\#L$. To that end, the theory $V\#L$ will be formed from the base theory VTC^0 by the addition of an axiom stating the existence of a solution for integer matrix powering. Computation of integer determinants is reducible to matrix powering [5].

The choice of matrix powering for our axiom is further justified by the fact that integer matrix powering is $\in \#L$ (as shown by Claim 42 below) and the easy reduction from matrix powering over $\mathbb{N}$ to matrix powering over $\mathbb{Z}$. Matrix powering over $\mathbb{N}$ is AC^0-complete for $\#L$ by Claim 18.

Let $\#STCON^m$ be the problem of $\#STCON$ on multigraphs. Notice that $\#STCON^m$ presents an additional layer of difficulty: since each entry of the adjacency matrix is $\in \mathbb{N}$ rather than $\in \{0, 1\}$, it requires more than one bit to store. This presents a choice of encodings: the input matrix entries can be encoded in unary or in binary. We will show that the binary version of $\#STCON^m$ is complete for $\#L$. This proof provides the necessary insight for Claim 42.

Claim 40 *Binary $\#STCON^m$ is complete for $\#L$ under AC^0-reductions.*

Proof: It is obvious that this problem is hard for $\#L$, since $\#STCON$ trivially reduces to $\#STCON^m$.

A few adjustments to the proof of claim 13 suffice to show that $\#STCON^m \in \#L$.

Again, let M be a logspace Turing machine with specific formatting of its input. Let the input graph be represented by a binary string encoding its adjacency matrix G, with s and t the first two listed vertices. Since multiple edges are allowed, this matrix has entries from $\mathbb{N}$. It is encoded as a binary string by means of the Row_2 function (see Section 2), with each matrix entry $G(u, v)$ written in binary notation as a string for numbers u and v referring to vertices.

Notice that there is some maximum m number of edges between any two vertices in the graph; hence every entry in the adjacency matrix has at most $log(m)$ bits. The entire adjacency matrix is encoded in at least $n^2 \log m$ bits.

As before, M maintains three (binary) numbers on its tape: the "current" vertex, the "next" vertex, and a count of the number of edges it has traversed. The "current" vertex is initialized to s (that is, the number 0, which indexes s in the encoded adjacency matrix), and the count is initialized to 0. M also maintains a bit "reachable", which is true iff the "next" vertex is reachable from the "current" vertex.

When run, M traverses the graph starting at s as follows:

```
TRAVERSE-MULTIGRAPH(n, s, t, p, G)
 1  counter ← 0
 2  current ← s
 3  while counter ≤ p and  current ≠ t
 4     do next ←  nondeterministically-chosen vertex from G
 5        reachable ← 0
 6        for i = |G(current, next)| to 0 do
 7           b ←  nondeterministally-chosen bit
 8           if reachable = 0
 9              then if b = 0 and G(current, next)[i] = 1
10                      then reachable ← 1
11                   if b = 1 and G(current, next)[i] = 0
12                      then  halt and reject
13        if reachable = 0
14           then  halt and reject
15        current ← next
16        counter ← counter + 1
17  if counter > p
18     then  halt and reject
19     else  halt and accept
```

We can think of the $g = G(current, next)$ edges between "current" and "next" as numbered $0, 1, \ldots, g - 1$. The loop on lines 6-12 implicitly selects a $(\log m)$-bit number n, one bit at a time, from most to least significant. It checks whether the "next" vertex is reachable from the "current" vertex along edge numbered n. If $n < g$, then the "next" vertex is reachable from the "current" vertex. The "reachable" bit is true if $n < g$ based on the already-seen bits.

M simulates a traversal of the graph from s to t by nondeterministically picking the next edge it traverses and the next vertex it visits. Every accepting computation of M traces a path from s to t (of length $\leq p$), and for every path of length $\leq p$ from s to t there is an accepting computation of M. Thus $\#STCON^m \in \#L$. ■

Claim 40 provides the insight for showing that matrix powering over $\mathbb{Z}$ is AC^0-reducible to $\#L$. In particular, it presents a technique whereby, using only two bits, a Turing machine can nondeterministically "pick" a natural number $< n$, where n is given in binary notation. Notice that this has the effect of causing the Turing machine to branch into n computational paths.

Remark 41 Branching into n_1 paths, then n_2 paths, …, then n_k paths has the effect of multiplication, resulting in $\prod_i n_i$ total computational paths.

Remark 41 and the proof of Claim 42 are inspired by the work of Vinay [14]. Lemma 6.2 of that paper proves a similar fact, though the conceptual framework, motivation, and notation are different.

Claim 42 *Matrix powering over $\mathbb{Z}$ is AC^0-reducible to $\#L$.*

Proof: The main idea of this proof is a combination of the reduction of $\#STCON$ to $\#L$ (Claim 13) and matrix powering over $\mathbb{N}$ to $\#STCON$ (Lemma 17), in order to show that matrix powering over $\mathbb{Z}$ can be defined entry-by-entry. We use the technique from Claim 40 to handle the binary encoding used for integers.

Let A be a matrix of integers, and let string X encode A via the Row_2 function. Ignoring the signs of integers, we can interpret A as the adjacency matrix of a directed multigraph. By definition, to show that matrix powering over $\mathbb{Z}$ is AC^0-reducible to $\#L$, we require a Σ_0^B formula for the bit graph of $Pow_{\mathbb{Z}}(n, k, X)$. We will demonstrate a stronger statement: in fact, $Pow_{\mathbb{Z}}(n, k, X)$ can be defined by whole entries $A^k[i, j]$.

We construct two Turing machines M^+ and M^- such that, given a particular entry (i, j) and integer matrix X as input, both simulate a traversal of the multigraph that X implicitly represents. For this purpose, the sign is ignored, so that $intsize(X^{[i][j]})$ is the number of edges of the multigraph from vertex i to vertex j. M^+ and M^- will use the signs of traversed edges to decide whether to accept or reject. The number of accepting paths of M^+ minus the number of accepting paths of M^- is the value of the $(i, j)^{\text{th}}$ entry of the matrix product A^k.

The machines M^+ and M^- have identical instructions except for their conditions for entering an accepting state. Each machine keeps track of two numbers, "current" and "next", indicating the index $(0, \ldots, n-1)$ of its simulated traversal, the "count" of how many edges it has traveled, and a single bit indicating the "sign" of the path traversed so far (when considered as the product of the signs of the edges traversed). The number "current" is initialized to i, "count" is initialized to 0, and "sign" is initialized to "+". When the machine needs to branch into b branches, where b is a number given in binary, we write "branch into b paths" in lieu of repeating the subroutine given in the Traverse-multigraph algorithm on page 23.

Positive-matrix-product(i, j, k, X)

$current \leftarrow i$
$sign \leftarrow 0$
$count \leftarrow 0$
while $counter < k$
    do $next \leftarrow$ nondeterministically-chosen number $< n$

6 branch into $intsize(X^{[current][next]})$ paths
7 $sign \leftarrow sign$ XOR $X^{[current][next]}(0)$
8 $count \leftarrow count + 1$
9 **if** $current = j$ and $sign = 0$
10 **then** halt and accept
11 **else** halt and reject

Given an input (i, j, k, X), M^+ traverses the implicit graph represented by X according to the algorithm POSITIVE-MATRIX-PRODUCT. The analogous algorithm NEGATIVE-MATRIX-PRODUCT for M^- will be identical, except that line 9 will require that $sign = 1$.

Observe that POSITIVE-MATRIX-PRODUCT is simply an adapted version of TRAVERSE-MULTIGRAPH. Rather than traversing a path between fixed nodes $s = 0$ and $t = 1$ of any length $\leq p$, it starts at vertex i and travels across *exactly* k edges. If the final vertex of that path is j, then we have traversed an i–j path of length exactly k. Remark 15 (which provided the insight for Lemma 17) and Remark 41 complete the proof.

Let f_{M^+} and f_{M^-} be functions of $\#L$, defined as the number of accepting paths of M^+ and M^-, respectively. The number $f_{M^+}(i, j, k, X)$ of accepting computations of M^+ is exactly the number of "positive-sign" paths from i to j, i.e., the sum of all positive terms in the computation of $A^k[i, j]$. The number $f_{M^-}(i, j, k, X)$ of accepting computations of M^- is exactly the number of "negative-sign" paths from i to j, i.e., the sum of all negative terms in the computation of $A^k[i, j]$. Thus the $(i, j)^{\text{th}}$ entry of A^k is given by

$$A^k[i, j] = f_{M^+}(i, j, k, X) - f_{M^-}(i, j, k, X)$$

■

5.3 The theory $V\#L$

By the nature of its construction, the theory $V\#L$ corresponds to the AC^0-closure of $\#L$. As we will prove, the set of provably total functions of $V\#L$ exactly coincides with the functions of $F\#L$, and the Δ_1^B-definable relations of $V\#L$ are exactly the relations in $\#L$.

Let $Pow_{\mathbb{Z}}(n, k, X)$ and $PowSeq_{\mathbb{Z}}(n, k, X)$ be defined as above (Definitions (23) and (24)), but for input X and outputs encoding a matrix (resp., list of matrices) of integers via the Row_2 function. Clearly there are analogs of Lemmas 25 and 26; $Pow_{\mathbb{Z}}$ and $PowSeq_{\mathbb{Z}}$ are AC^0-reducible to each other.

Definition 43 *The theory $V\#L$ has vocabulary $\mathcal{L}_A^2$ and is axiomatized by VTC^0 and a $\Sigma_1^B(\mathcal{L}_A^2)$ axiom $PS_{\mathbb{Z}}$ (formula 28) stating the existence of a string value for the function $PowSeq_{\mathbb{Z}}(n, k, X)$.*

We define our new axiom via a series of "helper" functions, just as (11) in Section 4.2. Let $\delta_{PowSeq_{\mathbb{Z}}}(n, k, X, Y)$ be the relation representing the graph of $PowSeq_{\mathbb{Z}}(n, k, X) = Y$. This relation will be defined below in the language $\mathcal{L}_{FTC^0} \supset \mathcal{L}_A^2$. This method requires the introduction of new function symbols, which can be used to express the axiom $PS_{\mathbb{Z}}$ in $\overline{VTC^0}$, a universal conservative extension of VTC^0.

Let $ID_{\mathbb{Z}}(n) = Y$ be the string function whose output encodes the $n \times n$ identity matrix over $\mathbb{Z}$, that is, $Y^{[i][j]}(c) \leftrightarrow i = j \wedge c = 1$. The extra layer of encoding necessary for integers makes this definition inelegant. (Notice that the previous equation defines the $\langle i, \langle j, c\rangle\rangle^{\text{th}}$ bit of Y.)

$$\begin{aligned} Y(b) \quad \leftrightarrow \quad & \mathit{left}(b) < n \wedge \mathit{right}(b) < n \wedge \mathit{Pair}(b) \wedge \mathit{Pair}(\mathit{right}(b)) \wedge \\ & \mathit{left}(b) = \mathit{left}(\mathit{right}(b)) \wedge \mathit{right}(\mathit{right}(b)) = 1 \end{aligned}$$

Let X_1 and X_2 be two strings encoding $n \times n$ integer matrices. Let $G(n, i, j, X_1, X_2)$ be the TC^0 string function that witnesses the computation of the $(i, j)^{\text{th}}$ entry of the matrix product X_1X_2, defined as:

$$G(n, i, j, X_1, X_2)(b) \leftrightarrow b < \langle |X_1|, |X_2|\rangle \wedge \mathit{Pair}(b) \wedge \big(X_1^{[i][\mathit{left}(b)]} \times_{\mathbb{Z}} X_2^{[\mathit{left}(b)][j]}\big)(\mathit{right}(b))$$

Here, the bound $b < \langle |X_1|, |X_2|\rangle$ is much larger than necessary. The function G serves as a witness; its output is a string encoding a list of integers Y_1, Y_2, …, Y_n, where $Y_k = X_1^{[i][k]} \times_{\mathbb{Z}} X_2^{[k][j]}$. The above definition specifies this exactly, as

$$G(n, i, j, X_1, X_2)^{[\ell]} = X_1^{[i][\ell]} \times_{\mathbb{Z}} X_2^{[\ell][j]}$$

Thus the $(i, j)^{\text{th}}$ entry of the matrix product X_1X_2 is given by the sum $Y_1 +_{\mathbb{Z}} \ldots +_{\mathbb{Z}} Y_n$. Chapter 9 of [9] defines the string function $\mathit{Sum}(n, m, Z)$ that takes the sum of a list of n binary numbers of length $\leq m$ stored as the first n rows of string Z. We adapt this definition to create string function $\mathit{Sum}_{\mathbb{Z}}$, which performs the same operaton on a list of n integers.

As a first step, we partition the list Z of integers into two lists: positive and negative numbers. Within each of these partitions, integers are stored as their *BinaryPart*, i.e., without a leading sign. Define the string functions *PosList* and *NegList* as:

$$\mathit{PosList}(Z)(i, j) \leftrightarrow \neg Z^{[i]} \wedge \mathit{BinaryPart}(Z^{[i]})(j)$$

$$\mathit{NegList}(Z)(i, j) \leftrightarrow Z^{[i]} \wedge \mathit{BinaryPart}(Z^{[i]})(j)$$

The function $\mathit{PosList}(Z)$ (respectively, $\mathit{NegList}(Z)$), outputs a list of the same length as Z, but only including positive (negative) elements of Z; all other entries are zero.

Thus $\mathit{Sum}(n, m, \mathit{PosList}(Z))$ is the natural number encoding the sum of the positive integers in list Z, and $\mathit{Sum}(n, m, \mathit{NegList}(Z))$ is the natural number encoding the *negation* of the sum of the negative integers in list Z. The output of each of these functions is a natural number. It is necessary to "shift" the bits by one place to re-insert the sign required by our encoding scheme for integers.

$$\mathit{PosSum}(n, m, Z)(b) \leftrightarrow \exists k < b, k + 1 = b \wedge \mathit{Sum}(n, m, \mathit{PosList}(Z))(k)$$

$$\mathit{NegSum}(n, m, Z)(b) \leftrightarrow (b = 0) \vee (\exists k < b, k + 1 = b \wedge \mathit{Sum}(n, m, \mathit{NegList}(Z))(b))$$

This allows for a simple definition of integer summation $\mathit{Sum}_{\mathbb{Z}}$:

$$\mathit{Sum}_{\mathbb{Z}}(n, m, Z)(i, j) \leftrightarrow= \mathit{PosSum}(n, m, Z) +_{\mathbb{Z}} \mathit{NegSum}(n, m, Z)$$

The string function $\mathit{Prod}_{\mathbb{Z}}$ computing the product of two integer matrices can be bit-defined as:

$$\mathit{Prod}_{\mathbb{Z}}(n, X_1, X_2)(i, j) \leftrightarrow i < n \wedge j < n \wedge \mathit{Sum}_{\mathbb{Z}}(n, |X_1| + |X_2|, G(n, i, j, X_1, X_2))$$

Again, the bound $|X_1| + |X_2|$ is much larger than necessary.

Given these functions, let the relation $\delta_{PowSeq_{\mathbb{Z}}}(n, k, X, Y)$ be the the $\Sigma_0^B(\mathcal{L}_{FTC^0})$-formula

$$\forall b < |Y|, |Y| < \langle k, \langle |X|, |X| \rangle \wedge [Y(b) \supset (Pair(b) \wedge Pair(right(b)))] \wedge \\ Y^{[0]} = ID_{\mathbb{Z}}(n) \wedge \forall i < k(Y^{[i+1]} = Prod_{\mathbb{Z}}(n, X, Y^{[i]})) \quad (27)$$

This formula asserts that the string Y is the output of $PowSeq_{\mathbb{Z}}(n, k, X)$. By our convention for defining string functions, the bits of Y that do not encode the list $[X^1, \ldots, X^k]$ are required to be zero, so $PowSeq_{\mathbb{Z}}(n, k, X)$ is the lexographically first string that encodes this list.

$\overline{VTC^0}$ is a conservative extension of VTC^0, defined in Section 9C of [9]. Theorem 9.33(b) of [9] asserts that there is a $\mathcal{L}_A^2$ term t and a $\Sigma_0^B(\mathcal{L}_A^2)$ formula $\alpha_{PowSeq_{\mathbb{Z}}}$ such that

$$\exists Z < t, \alpha_{PowSeq_{\mathbb{Z}}}(n, k, X, Y, Z)$$

is provably equivalent to (27) in $\overline{VTC^0}$.

The axiom $PS_{\mathbb{Z}}$ used to define the theory $V\#L$ is

$$\exists Y < m, \exists Z < t, \alpha_{PowSeq_{\mathbb{Z}}}(n, k, X, Y, Z) \quad (28)$$

Formulas (27) and (28) each require that Y witnesses the intermediate strings X^1, X^2, $\ldots$, X^k of the computation of the matrix power X^k. String Y does *not* witness any of the work performed in calculating these intermediate powers of X, just as in Section 4.2. That work, as before, is witnessed by the string Z, although this witnessing is obscured by the application of Theorem 9.33(b).

The next two lemmas are analogous to Lemmas 28 and 30, and proved in the same manner.

Lemma 44 *The integer matrix powering function* $PowSeq_{\mathbb{Z}}$ *is* $\Sigma_1^B(\mathcal{L}_A^2)$*-definable in* $V\#L$.

Lemma 45 *The aggregate integer matrix powering function* $PowSeq_{\mathbb{Z}}^{\star}$ *is* $\Sigma_1^B(\mathcal{L}_A^2)$*-definable in* $V\#L$.

5.4 The theory $\overline{V\#L}$

This section defines the theory $\overline{V\#L}$, a universal conservative extension of $V\#L$, in the same way as $\overline{V\oplus L} \supset V\oplus L$ (Section 4.4).

The theory $\overline{V\#L}$ is a universal conservative extension of $V\#L$. Its language $\mathcal{L}_{F\#L}$ contains function symbols for all string functions in $F\#L$. Note that $F\#L \neq \#L$, which can be observed from the fact that $F\#L$ contains string functions and number functions that take negative values, neither of which is in $\#L$. The defining axioms for the functions in $\mathcal{L}_{F\#L}$ are based on their AC^0 reductions to matrix powering. Additionally, $\overline{V\#L}$ has a quantifier-free defining axiom for $PowSeq'_{\mathbb{Z}}$, a string function with inputs and outputs the same as $PowSeq_{\mathbb{Z}}$.

As before, we leave the formal definition of $PowSeq_{\mathbb{Z}}$ unchanged. $PowSeq'_{\mathbb{Z}}$ has the quantifier-free defining axiom

$$|Y| < \langle k, \langle |X|, |X| \rangle \wedge (Y(b) \supset Pair(b) \wedge Pair(right(b))) \wedge \\ Y^{[0]} = ID_{\mathbb{Z}}(n) \wedge \big(i < k \supset (Y^{[i+1]} = Prod_{\mathbb{Z}}(n, X, Y^{[i]}))\big) \quad (29)$$

This formula is similar to (27). The function $PowSeq_{\mathbb{Z}}$ satisfies this defining axiom for $PowSeq'_{\mathbb{Z}}$. VTC^0, together with both axioms, proves $PowSeq_{\mathbb{Z}}(n,k,X) = PowSeq'_{\mathbb{Z}}(n,k,X)$.

Recall that for a given formula $\varphi(z,\vec{x},\vec{X})$ and $\mathcal{L}^2_A$-term $t(\vec{x},\vec{X})$, we let $F_{\varphi,t}(\vec{x},\vec{X})$ be the string function with bit definition (19)

$$Y = F_{\varphi(z),t}(\vec{x},\vec{X}) \leftrightarrow z < t(\vec{x},\vec{X}) \wedge \varphi(z,\vec{x},\vec{X})$$

Definition 46 ($\mathcal{L}_{F\#L}$) *We inductively define the language $\mathcal{L}_{F\#L}$ of all functions with (bit) graphs in $\#L$. Let $\mathcal{L}^0_{F\#L} = \mathcal{L}_{FTC^0} \cup \{PowSeq'_{\mathbb{Z}}\}$. Let $\varphi(z,\vec{x},\vec{X})$ be an open formula over $\mathcal{L}^i_{F\#L}$, and let $t = t(\vec{x},\vec{X})$ be a $\mathcal{L}^2_A$-term. Then $\mathcal{L}^{i+1}_{F\#L}$ is $\mathcal{L}^i_{F\#L}$ together with the string function $F_{\varphi(z),t}$ that has defining axiom:*

$$F_{\varphi(z),t}(\vec{x},\vec{X})(z) \leftrightarrow z < t(\vec{x},\vec{X}) \wedge \varphi(z,\vec{x},\vec{X}) \tag{30}$$

Let $\mathcal{L}_{F\#L}$ be the union of the languages $\mathcal{L}^i_{F\#L}$. Thus $\mathcal{L}_{F\#L}$ is the smallest set containing $\mathcal{L}_{FTC^0} \cup \{PowSeq'_{\mathbb{Z}}\}$ and with the defining axioms for the functions $F_{\varphi(z),t}$ for every open $\mathcal{L}_{F\oplus L}$ formula $\varphi(z)$.

As before, $\mathcal{L}_{F\#L}$ has a symbol for every string function in $F\#L$, and a term $|F|$ for every number function in $F\#L$, where F is a string function in $F\#L$.

Definition 47 ($\overline{V\#L}$) *The universal theory $\overline{V\#L}$ over language $\mathcal{L}_{F\#L}$ has the axioms of $\overline{VTC^0}$ together with the quantifier-free defining axiom (29) for $PowSeq'_{\mathbb{Z}}$ and the above defining axioms (30) for the functions $F_{\varphi(z),t}$.*

Theorem 48 *$\overline{V\#L}$ is a universal conservative extension of $V\#L$.*

The proof is the same as for Theorem 33 (page 18).

5.5 Provably total functions of $V\#L$

These results follow directly from [9]. We restate them here for convenience.

Claim 49 *The theory $\overline{V\#L}$ proves the axiom schemes $\Sigma^B_0(\mathcal{L}_{FTC^0})$-COMP, $\Sigma^B_0(\mathcal{L}_{FTC^0})$-IND, and $\Sigma^B_0(\mathcal{L}_{FTC^0})$-MIN.*

Claim 50 *(a) A string function is in $F\#L$ if and only if it is represented by a string function symbol in $\mathcal{L}_{F\#L}$.*

(b) A relation is in $\#L$ if and only if it is represented by an open formula of $\mathcal{L}_{F\#L}$ if and only if it is represented by a $\Sigma^B_0(\mathcal{L}_{F\#L})$ formula.

Corollary 51 *Every $\Sigma^B_1(\mathcal{L}_{F\#L})$ formula φ^+ is equivalent in $\overline{V\#L}$ to a $\Sigma^B_1(\mathcal{L}^2_A)$ formula φ.*

Corollary 52 *(a) A function is in $F\#L$ iff it is $\Sigma^B_1(\mathcal{L}_{F\#L})$-definable in $\overline{V\#L}$ iff it is $\Sigma^B_1(\mathcal{L}^2_A)$-definable in $\overline{V\#L}$.*

(b) *A relation is in $\#L$ iff it is $\Delta_1^B(\mathcal{L}_{F\#L})$-definable in $\overline{V\#L}$ iff it is $\Delta_1^B(\mathcal{L}_A^2)$-definable in $\overline{V\#L}$.*

Theorem 53 *A function is provably total (Σ_1^1-definable) in $V\#L$ iff it is in $F\#L$.*

Theorem 54 *A relation is in $\#L$ iff it is Δ_1^B-definable in $V\#L$ iff it is Δ_1^1-definable in $V\#L$.*

6 Future work

Due to the time constraints of this project, there are several results omitted above. These require work beyond what is contained in this paper.

The proof for Theorem 20 (page 11), stating the AC^0-closure of $\oplus L$, is omitted above, in lieu of which several references are given for papers proving the same closure, or a stronger version. For completeness, this theorem should be proven within the framework of Chapter 9, by use of Theorem 9.7.

Similarly, this work is incomplete without a proof that the closure of $\#L$ is the class DET. On page 5 the class DET is simply defined as this closure; the exposition would be more self-contained if this characterization of $AC^0(\#L)$ were proven. (It is proven or implied by results in [1], [12], [2], and others.)

By using the notation DET for the closure $AC^0(\#L)$, we implicitly rely upon the fact that the problem of computing the determinant of an integer-valued matrix is complete for $\#L$. The notation developed above is sufficient for a proof that the integer determinant can be captured by reasoning in $V\#L$. This proof would likely formalize Berkowitz's method, which provides a reduction from integer determinant to integer matrix powering [5].

There are a number of algebraic problems which [8], [7], [6], and [5] prove are NC^1-reducible to each other. These include: integer determinant, matix powering, iterated matrix product, computing the coefficients of characteristic polynomials, rank computation, choice of a linearly independent subset from a set of vectors, computing the basis of the kernel of a matrix, and solving a system of linear equations. Many of these reductions seem to be simple enough to be formalized as AC^0-reductions, and intuitively it seems that all of these problems should also be complete for $\oplus L$ and $\#L$ under AC^0-reductions. Since $\oplus L$ concerns elements over a field, finding matrix inverses can be added to the list (and similarly for every MOD_pL class for p prime).

Formalizing these reductions seems an arduous task, but a shortcut is possible. In [13], the authors construct formal theories for linear algebra over three sorts: indices ($\mathbb{N}$), field elements, and matrices. Several basic theorems of linear algebra, including many of the problems listed above, are provable in these theories. By interpreting these 2-sorted theories into the 3-sorted theories constructed in this paper, we can use convenient results without having to prove them again in a different framework.

References

[1] Eric Allender. Arithmetic Circuits and Counting Complexity Classes. *Complexity of Computations and Proofs* (ed. Jan Krajicek), 33–72, 2004.

[2] Eric Allender, Mitsunori Ogihara. Relationships Among PL, $\#L$, and the Determinant. *RAIRO - Theoretical Informatics and Applications*, 30:1–21, 1996.

[3] Carme Alvarez, Birgit Jenner. A Very Hard Log Space Counting Class. *Proc.* 5^{th} *Conference on Structure in Complexity Theory*, 154–168, 1990.

[4] Richard Beigel, John Gill, Ulrich Hertrampf. Counting classes: Thresholds, Parity, Mods, and Fewness. *Proc.* 7^{th} *STACS*, 49–57. Lecture notes in Computer Science, Vol. 415. Springer-Verlag, Berlin, 1990.

[5] S. J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

[6] A. Borodin, J. von zur Gathen, J. E. Hopcroft. Fast parallel matrix and gcd computations. *Inform. and Control*, 52:241–256, 1982.

[7] Gerhard Buntrock, Carsten Damm, Ulrich Hertrampf, and Christoph Meinel. Structure and importance of logspace-MOD class. *Mathematical Systems Theory*, 25:223–237, 1992.

[8] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

[9] Stephen Cook and Phuong Nguyen. Logical Foundations of Proof Complexity: Bounded Arithmetic and Propositional Translations. (Draft.)

[10] Carsten Damm. Problems complete for $\oplus L$. *Information Processing Letters* 36:247–250, 1990.

[11] Ulrich Hertrampf, Steffen Reith, Heribert Vollmer. A note on closure properties of logspace MOD classes. *Information Processing Letters* 75:91–93, 2000.

[12] Meena Mahajan, V. Vinay. Determinant: Combinatorics, Algorithms, and Complexity. *Chicago Journal of Theoretical Computer Science*, 1997.

[13] Michael Soltys and S. A. Cook. The Proof Complexity of Linear Algebra. *Annals of Pure and Applied Logic* 130:277–323, 2004.

[14] V. Vinay. Counting Auxiliary Pushdown Automata and Semi-Unbounded Arithmetic Circuits. *Proc.* 6^{th} *IEEE Structure in Complexity Theory Conference* 270–284, 1991.