Deterministic 0L Languages are of Very Low Complexity: D0L is in $\mathcal{AC}^0$

Carsten Damm*

FB IV – Informatik
Universität Trier
D–54286 Trier
Germany

Markus Holzer* Klaus-Jörn Lange* Peter Rossmanith†

Institut für Informatik
Technische Universität München
Arcisstr. 21, D–80290 München
Germany

Abstract

Languages defined by iterated homomorphisms, i.e., D0L systems, are closely related to the arithmetic of small numbers. Based on this interrelation we improve the upper bound $\mathcal{NC}^1$ on the complexity of the membership problem for D0L systems.

We prove that the membership problem is contained in a proper subclass of $\mathcal{AC}^0$. This will show that the D0L languages are easier to recognize than the aperiodic regular languages.

1 Introduction

We investigate languages defined by iterated homomorphisms or substitutions, i.e., context-free Lindenmayer languages. They provide a whole framework of classes which have properties similar to context-free (Chomsky) languages. The complexity of the membership problem for these languages is in most cases well determined, as it is for context-free languages. This can be seen by the following completeness-results‡:

*Partially supported by DFG-La 618/1-1

†Supported by SFB 0342 A4

‡Usually, only *extended* systems are considered, which select only generated words over a *terminal* alphabet. Obviously, this is not important with respect to membership problems.

- The class of T0L languages is $\mathcal{NP}$-complete,[12]
- the class of DT0L languages is $NSPACE(\log n)$-complete,[5] and
- the class of 0L languages is $\mathcal{LOGCFL}$-complete.[11]

An exception are D0L systems, i.e., *one* iterated homomorphism, where only a rather imprecise logarithmic space bound was provided by Sudborough in.[11]

This was recently improved by exhibiting close relations between the word problem of D0L systems and natural arithmetic problems, such as multiplication, polynom evaluation, iterated matrix multiplication, etc.[3] In the general case[4] the first two problems are not computable in $\mathcal{AC}^0$ and the latter is not known to be computable in $\mathcal{NC}^1$. By taking advantage of the fact that in the applications all numerical inputs as well as the results of the arithmetic computations are *small numbers*, i.e., polynomially bounded numbers,[1] this restriction led to upper bounds on the complexity in terms of very low circuit classes. In this way the large subfamily of polynomially growing D0L languages was shown to be contained in $\mathcal{AC}^0$. In general, the word problem is in a proper subfamily of $\mathcal{NC}^1$, which gave the surprising result that D0L languages are provably simpler to recognize than regular sets.

In this work we improve the upper bound $\mathcal{NC}^1$ for D0L membership to $\mathcal{AC}^0$, by improving the upper bound for iterated matrix multiplication (of small numbers), and hence for exponentially growing D0L systems. The idea for this improvement heavily relies on the well-known Chinese Remainder Theorem. Since we will prove the containment of D0L in a proper subclass of $\mathcal{AC}^0$, this will show that D0L languages are even easier to recognize than aperiodic regular languages.

In the next section we introduce the notions and facts about D0L languages. In Section 3 we develop the necessary background on uniform circuit families and introduce small number arithmetic. In Section 4 we show the containment of D0L in $\mathcal{AC}^0$ by improving the upper bound for iterated matrix multiplication.

2 Some facts about D0L systems

We assume the reader to be familiar with the standard facts about D0L systems, especially with growth functions. In addition we use the following notions. Most of the material is well-known.[3,8]

Definition 1 A D0L system is a triple $G = (\Sigma, h, \alpha)$ consisting of a finite alphabet Σ, a homomorphism $h : \Sigma^* \to \Sigma^*$, and the *axiom* $\alpha \in \Sigma^*$. The language L_G generated by G is the set of words obtained by iteratively applying the homomorphism h to α:

$$L_G = h^*(\alpha) = \{\alpha, h(\alpha), h^2(\alpha), h^3(\alpha), \ldots\}.$$

The class of D0L languages is denoted by $\text{D0L} = \{\, L_G \mid G \text{ is a D0L system}\,\}$.

The growth rate of the length of the words generated by a D0L system reveals much of the structure of the corresponding membership problem as we shall see. Therefore we define formally the growth function of a D0L system:

Definition 2 Given a D0L system $G = (\Sigma, h, \alpha)$, the function $f_G(n) := |h^n(\alpha)|$ is called the growth function of G.

Growth functions can be expressed in terms of matrix products. Given a D0L system (Σ, h, α) with $\Sigma = \{a_1, \ldots, a_s\}$, let A_G be the $s \times s$ matrix whose (i, j)th entry is the number of appearances of letter a_j in $h(a_i)$. Further let η be the all-one-vector of length s and let $\pi(\alpha)$ be the Parikh-vector of α, i.e., the vector of length s whose ith component is the number of appearances of letter a_i in α. Then

$$f_G(n) = \pi(\alpha)(A_G)^n \eta^T.$$

This formula is the Parikh-representation of the growth function.

Growth functions of D0L systems may be exponentially bounded from below or polynomially (or even constantly) bounded from above. In fact it is known that there is nothing "in between":

Lemma 3 For each D0L system G there is either a constant $c > 1$ such that $f_G(n) \geq c^n$ holds for all n or there is a polynomial $p(n)$ such that $f_G(n) \leq p(n)$ holds for all n.

Let us call the D0L systems with polynomial upper bound on the growth function *polynomial* and the other ones *exponential* D0L systems.

3 Uniform circuits and small number arithmetic

We assume the reader to be familiar with the definition of Boolean circuits.[2]

The resources of interest for us are *size*, *depth*, and *fan-in* of gates. We mainly use AND-, OR-, and NOT-gates. A circuit C_n on n inputs in the usual way computes an n-variable function $f_{C_n} : \{0,1\}^n \rightarrow \{0,1\}$, a circuit family $C = (C_n)$ accordingly computes a function $f_C : \{0,1\}^* \rightarrow \{0,1\}$, where the nth circuit processes inputs of length n. The language or problem decided by C is the set of words in $\{0,1\}^*$ mapped to 1 under f_C.

This definition covers only two-letter languages, since Boolean circuits are based on two-valued logic. By binary encodings circuit families can also recognize more general languages. We will tacitly make use of this as a convention, without specifying the encoding.

We further consider computation problems, e.g., given two numbers, compute their product. In this case the circuits to be considered have several output gates.

A circuit family $C = (C_n)$ is said to have

- logarithmic depth if $depth(C_n) = O(\log n)$, and constant depth if $depth(C_n) = O(1)$
- polynomial size if $size(C_n) = n^{O(1)}$
- bounded fan-in if $\textit{fan-in}(C_n) = O(1)$

To define uniformity we make use of the following terminology[9]: Let $C = (C_n)$ be a circuit family. The *direct connection language* of C is the set $L_{DC}(C)$ of tuples $\langle 1^n, g_1, g_2, \tau\rangle$ where g_1 and g_2 are names of gates in C_n, g_2 is a child of g_1 and τ is the type of g_1 (for gates g_1 without predecessors g_2 is dummy). If $\mathcal{A}$ is an arbitrary class of languages we call a circuit family C to be $\mathcal{A}$-uniform if $L_{DC}(C) \in \mathcal{A}$.

In order to get the circuits within the framework of low complexity classes, it is necessary to impose strong uniformity conditions. Thus, throughout this paper we consider $\mathcal{DLOGTIME}$-uniform circuits. Here the direct connection language is recognized by a deterministic Turing machines with logarithmic time bound.[1]

Definition 4 $\mathcal{AC}^0$ is the set of problems solvable by $\mathcal{DLOGTIME}$-uniform unbounded fan-in polynomial size circuits of constant depth. $\mathcal{NC}^1$ consists of all problems solvable by $\mathcal{DLOGTIME}$-uniform bounded fan-in polynomial size circuits of logarithmic depth.

Speaking informally $\mathcal{DLOGTIME}$-uniformity means to encode much of the local interconnection structure into the names of the gates. But when proving our upper bounds, we will only in a few cases explicitly indicate the full interconnection structure. Rather than doing this we exploit two properties of $\mathcal{DLOGTIME}$-uniform circuits: Once circuits have been proved to be $\mathcal{DLOGTIME}$-uniform they can be used as parts (modules) of new circuits, that turn out to be $\mathcal{DLOGTIME}$-uniform if the local interconnection structure between the modules can be recognized in $\mathcal{DLOGTIME}$. On the other hand we will, without explicitly stating it, make use of the characterization of $\mathcal{AC}^0$ in terms of first-order logic with BIT-predicate.[1]

The membership question for D0L systems was related to the computation of growth functions.[3] For polynomial D0L systems one has to evaluate polynomials* on small numbers. Since this arithmetic problem is solvable in $\mathcal{AC}^0$, the following upper bound holds[3]:

Theorem 5 The membership problem for polynomial D0L systems is solvable in $\mathcal{AC}^0$.

In the case of exponential D0L systems, one has to solve the problems iterated matrix multiplication (on small numbers) and the summation of $O(\log n)$ small numbers. Since the latter problem is in $\mathcal{AC}^0$, the complexity of the membership algorithm

*We remark that the circuit for multiplication of small numbers given by Damm et.al.[3] is not $\mathcal{DLOGTIME}$-uniform, for a correct solution see Barrington et.al.[1] Thus, all results hold es well.

depends only on the former one.[1] Unfortunately, for this problem only an $\mathcal{NC}^1$ upper bound was given.[3] In the following we will lower this bound to $\mathcal{AC}^0$.

Thus, in the next section we deal with arithmetic problems restricted to small numbers. "Small" means here to be polynomially bounded in n. Formally the restriction to small numbers can be achieved by supplying the numbers in unary notation: Input number $m \leq n$ is given in the form $1^m 0^{n-m}$. Further one considers 1^n to be part of the input. All numbers of the following problems are nonnegative integers:

1. POWER-TO-SMALL-NUMBERS: Given m, for a fixed integer $a > 0$ compute $bin(a^m)$ if $a^m \leq n$, otherwise return a fault signal.
2. MATRIX POWER-TO-SMALL-NUMBERS: Given m, for a fixed $s \times s$ matrix A and a fixed number $a > 1$ compute a binary representation of A^m if $a^m \leq n$ and all entries of A^m do not exceed n, otherwise return a fault signal.

Please note that 1 and 2 are *classes* of problems rather than single problems.

4 The containment of D0L in $\mathcal{AC}^0$

The algorithm for exponential D0L systems[3] heaviley relies on the arithmetic problem MATRIX-POWER-TO-SMALL-NUMBERS, resp. POWER-TO-SMALL-NUMBERS. In the following we show how to solve these two problems in $\mathcal{AC}^0$ and then we state the membership algorithm for exponential growing D0L systems.

Throughout this section we use the following notations: Let n be the length of the input and $r_b(x) := (x \bmod 1, x \bmod 2, \ldots, x \bmod b\log n)$ for some $b \geq 1$. By $x \bmod y$ we understand the remainder of x divided by y. We denote the natural logarithm by $\ln x$ and the binary logarithm by $\log x$.

The idea of the Chinese Remainder Theorem is to replace one big multiplication by several small ones that can be carried out in parallel. Before we come to the algorithm in detail, we need the following theorem about the product of primes. Rosser and Schoenfeld[7] showed the following:

Theorem 6 Let $p_1, p_2, \ldots, p_k$ be all primes less or equal than x. Then

$$\ln\left(\prod_{i=1}^{k} p_i\right) > x \cdot \left(1 - \frac{1}{\ln(x)}\right) \quad \text{for } x \geq 41.$$

With Theorem 6 one can easily verify that for every $a \geq 2$ there exists a b such that the product of all primes less or equal to $b\log n$ is greater or equal than $a^{\log n}$ for $n \geq 2$. As an immediate consequence of this and the Chinese Remainder Theorem we get that for $1 \leq k \leq \log n$ and $0 \leq y \leq n$ the equality $r_b(a^k) = r_b(y)$ already implies that $a^k = y$.

Now we are ready to state the algorithm for POWER-TO-SMALL-NUMBERS and MATRIX-POWER-TO-SMALL-NUMBERS.

Proposition 7 The problem POWER-TO-SMALL-NUMBERS is solvable in $\mathcal{AC}^0$.

Proof: To describe the algorithm for POWER-TO-SMALL-NUMBERS let $n \geq 2$. We obtain a^k in two stages.

First, we compute $r_b(a^k)$ by handling its components $a^k \bmod l$ in parallel for $1 \leq l \leq b \log n$. Actually, we compute $a^m \bmod l$ also for $1 \leq m \leq \log n$. This is a very small problem and can be done by the uniformity machine itself according to the next proposition. All what remains to be done is then to pick the right, i.e., kth, result.

Proposition 8 Given $a = O(1)$, $m = O(\log n)$, and $l = O(\log n)$ then a deterministic multi-tape Turing machine can compute $a^m \bmod l$ in $O((\log \log n)^3)$ time.

Proof: Raising a number a to the mth power can be solved by $m - 1$ multiplications in a straightforward way. By repeated squaring we can compute a^m with $\log(m)$ multiplications if m is a power of two. If m fails to be a power of two we can still compute a^s for all s that are powers of two and smaller than m. Another $\log(m)$ multiplications suffice to compute a^m from a, a^2, a^4, $\ldots$, a^s. During the computation of a^m by the method shown above we do not use a binary encoding of the whole intermediate results. After each multiplication with result r, we compute immediately $r \bmod l$. This guarantees that intermediate computations are carried out on *very small* numbers, i.e., $O(\log \log n)$ bits. Since all numbers involved have only $O(\log \log n)$ bits, multiplication takes $O((\log \log n)^2)$ time and division is possible within the same time bound by a technique due to Cook.[6] Thus the claim follows. □

Now we have to compute a^k from $r_b(a^k)$, which is done by computing $r_b(y)$ for $0 \leq y \leq n$ in parallel and choosing the y with $r_b(y) = r_b(a^k)$. Note that the computation of $r_b(y)$ involves establishing $y \bmod l$ for $1 \leq l \leq \log n$ which is no problem since division of small numbers is in $\mathcal{AC}^0$. Thus, the claim follows. □

Corollary 9 The problem MATRIX-POWER-TO-SMALL-NUMBERS is solvable in $\mathcal{AC}^0$.

Proof: It is easy to modify the method of the proof of Proposition 7 to solve the more general MATRIX-POWER-TO-SMALL-NUMBERS. Remember that only *constant size* matrixes are involved. For a matrix A we compute $r_b(A) := (A \bmod 1, A \bmod 2, \ldots, A \bmod b \log n)$. Here $A \bmod l = (A_{ij} \bmod l)$ if $A = (A_{ij})$, i.e., we compute remainders for each component.

Since each matrix multiplication takes only a constant factor more time than an integer multiplication, the above algorithm generalizes to matrixes. We conclude that MATRIX-POWER-TO-SMALL-NUMBERS can be computed just as the problem above, and thus it is solvable in $\mathcal{AC}^0$. □

As an immediate consequence we get:

Corollary 10 Given a D0L system G, the problem to decide upon 1^n, whether there is some k such that $n = f_G(k) := \pi(\alpha)(A_G)^k\eta^T$ and, if so, to compute the least such k, is computable in $\mathcal{AC}^0$. □

Sketch of Proof: By Lemma 3 the growth function is either exponentially bounded from below or is polynomially or constantly bounded form above. Further if $n = f_G(k)$ then $k = O(n)$. By Corollary 9 the growth function $f_G(k)$ can be computed in $\mathcal{AC}^0$ for all k in question. □

Now we use the algorithm for the membership problem for exponential D0L systems[3]: Let G be an exponential D0L system $G = (\Sigma, h, \alpha)$. First we introduce some notations. Let P be the predicate

$$P(k, i, a) \iff \text{the } i\text{th letter of } h^k(\alpha) \text{ is } a.$$

Observe that $w = b_1 b_2 \ldots b_n \in L_G$ if and only if there is a k such that $P(k, i, b_i)$ holds for all $i \in \{1, 2, \ldots, n\}$. The derivation length k is bounded by $c \cdot \log(f_G)$, because of the exponential growth, where c is a constant only depending on G.

We follow here the ideas of Sudborough.[11] The basic idea of his algorithm is, that if b_i is the appropriate letter on position i, then there exists a sequence of letters $a_0, a_1, \ldots, a_k \in \Sigma$ and a sequence of positions $i_0, i_1, \ldots, i_k$ such that

- a_0 is the i_0th letter in the initial string α,
- every a_j, with $1 \le j \le k$, is the i_jth letter in $h(a_{j-1})$ and
- a_k is equal to b_i.

Note that every i_j with $0 \le j \le k$ is bounded by the constant $m = \max_{a\in\Sigma}\{|h(a)|\}$. So the first and second conditions can be tested very easily once the a_j's and i_j's are chosen. Therefore we concentrate on the third condition—the test whether these sequences lead to the ith letter in w.

It is easy to see, that the global position number i is uniquely determined by the sequences $a_0, a_1, \ldots, a_k$ and $i_0, i_1, \ldots, i_k$. This dependence can be described in a very concise manner. To do this we introduce the following notation. Let $\mathit{pref}_i(v)$ be the prefix of length i of the word v. Using this, the position of a_0 in α is i_0, the position of a_1 in the word $h(\alpha)$ can be described by the formula

$$(\pi(\mathit{pref}_{i_0-1}(\alpha))(A_G) + \pi(\mathit{pref}_{i_1-1}(h(a_0))))\eta^T + 1.$$

Iterating this process, we get for the position i of a_k in the word $h^k(\alpha)$ the following formula:

$$\begin{gathered}(\pi(\mathit{pref}_{i_0-1}(\alpha))(A_G)^k + \pi(\mathit{pref}_{i_1-1}(h(a_0)))(A_G)^{k-1} + \cdots \\ \cdots + \pi(\mathit{pref}_{i_{k-1}-1}(h(a_{k-2})))(A_G) + \pi(\mathit{pref}_{i_k-1}(h(a_{k-1}))))\eta^T + 1.\end{gathered}$$

The first three conditions above are easy to be tested with the $\mathcal{DLOGTIME}$ circuit constructor. Since all parameters of the formula except k are constants, and k is bounded by $O(n)$ and is therefore a small number, the computation of the subformula $\pi(\mathit{pref}_{i_j-1}(h(a_{j-1})))(A_G)^{k-j}\eta^T$ for $0 \leq j \leq k$ with $a_{-1} := \alpha$, can be performed in $\mathcal{AC}^0$ by Corollary 10.

Since, by Barrington, Immerman, and Straubing,[1] the addition of $O(\log n)$ small numbers is in $\mathcal{AC}^0$, the whole computation of the above formula can be done by an $\mathcal{AC}^0$ circuit.

We describe the circuit for the predicate P. First model with OR-gates the existential branch for the sequences $a_0, a_1, \ldots, a_k$ and $i_0, i_1, \ldots, i_k$ and then test with an AND-gate whether i is equal to the value of the formula above by using the above $\mathcal{AC}^0$ circuit and test that the ith letter is a_k. Now it is easy to design $\mathcal{AC}^0$ circuits for the membership problem for G.

Theorem 11 The membership problem for exponential D0L systems is solvable in $\mathcal{AC}^0$.

By Theorem 5 and 11 we obtain:

Corollary 12 The membership problem for D0L systems is solvable in $\mathcal{AC}^0$.

5 Conclusions

A careful analysis shows that the depth of the $\mathcal{AC}^0$ circuits constructed in Theorems 5 and 11 is independent of the underlying D0L system. That is, the membership problem of D0L is contained in $\mathcal{AC}^0_k$ for some fixed depth k. Compare this to aperiodic (or star-free) regular sets: Although they are contained in $\mathcal{AC}^0$, it can be shown that there is no fixed depth k such that the aperiodic regular sets are contained in $\mathcal{AC}^0_k$.[1] This may be interpreted as saying that D0L languages are of provably smaller complexity than the aperiodic regular sets, since $\mathcal{AC}^0_k$ is a proper subset of $\mathcal{AC}^0_{k+1}$ for all k.[10]

D0L languages are one of the few natural examples of sparse sets. This seems to be one reason why it is so difficult to prove lower bounds for the D0L membership problem. Therefore, a promising approach could be to reduce appropriate families of sparse sets to D0L languages via projections. A natural candidate would be the family of bounded regular sets.

6 Acknowledgments

We would like to thank Uli Hertrampf, Neil Immerman, Rolf Niedermeier, and Denis Thérien for many hints and helpful discussions.

References

1. D. A. M. Barrington, N. Immerman, and H. Straubing. On uniformity within $\mathcal{NC}^1$. *Journal of Computer and System Sciences*, 41:274–306, 1990.

2. S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64(1):2–22, 1985.

3. C. Damm, M. Holzer, and K.-J. Lange. The parallel complexity of iterated morphisms and the arithmetic of small numbers. In *Proceedings of the 17th Conference on Mathematical Foundations of Computer Science*, number 629 in LNCS, pages 227–235. Springer, 1992.

4. M. Furst, J. B. Saxe, and M. Sipser. Parity circuits and the polynomial-time hierarchy. In *Proc. 22th IEEE Symposium on Foundations of Computer Science*, pages 260–270, 1981.

5. N. D. Jones and S. Skyum. Recognition of deterministic ET0L languages in logarithmic space. *Information and Computation*, 35:177–181, 1977.

6. D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, 2nd edition, 1969.

7. J. B. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

8. G. Rozenberg and A. Salomaa. *The Mathematical Theory of L Systems*, volume 90 of *Pure and Applied Mathematics*. Academic Press, 1980.

9. W. Ruzzo. On uniform circuit complexity. *Journal of Computer and System Sciences*, 22:365–338, 1981.

10. M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 61–69, 1983.

11. I. Sudborough. The time and tape complexity of developmental languages. In *Proceedings of 4th International Colloquim on Automata, Languages, and Programming*, number 52 in LNCS, pages 509–521. Springer, 1977.

12. J. van Leeuwen. The membership problem for ET0L-languages is polynomially complete. *Information Processing Letters*, 3(5):138–143, May 1975.