Electronic Colloquium on Computational Complexity, Report No. 95 (2011)

Low uniform versions of NC^1

Christoph Behle[1], Andreas Krebs[1], Klaus-Jörn Lange[1], and Pierre McKenzie[2]

[1] Wilhelm-Schickard-Institut, Universität Tübingen
{behlec,krebs,lange}@informatik.uni-tuebingen.de
[2] DIRO, Université de Montréal
mckenzie@iro.umontreal.ca

Abstract. In the setting known as DLOGTIME-uniformity, the fundamental complexity classes $AC^0 \subset ACC^0 \subseteq TC^0 \subseteq NC^1$ have several robust characterizations. In this paper we refine uniformity further and examine the impact of these refinements on NC^1 and its subclasses. When applied to the logarithmic circuit depth characterization of NC^1, some refinements leave NC^1 unchanged while others collapse NC^1 to NC^0. Thus we study refinements of other circuit characterizations of NC^1. In the case of the $AC^0(A_5)$ characterization of NC^1, where A_5 is the NC^1-complete word problem of the group A_5, our refinements collapse NC^1 to a subset of the regular languages. For the $AC^0(\mathbb{D}_+)$ characterizations of NC^1, where $\mathbb{D}_+$ is the NC^1-complete language capturing the formula value problem, interestingly, these refinements scale down to circuits with linear fan-in. In particular, the latter refinements bring to the fore two classes, denoted FO[<]-uniform $AC^0(\mathbb{D}_+)_{LIN}$ and FO[<]-uniform TC^0_{LIN}, whose separation may be within the reach of current lower bound techniques, and whose separation would amount to distinguishing the power of a MAJ gate from that of a $\mathbb{D}_+$ gate.

1 Introduction

Uniformity conditions on Boolean circuits were introduced in order to exclude undecidable languages from circuit-based complexity classes, thus allowing a fair comparison between these and machine-based classes. In some cases, the circuit complexity of a language seems largely independent from the chosen uniformity. In other cases, our ability or inability to tighten circuit uniformity holds the key to long-standing open questions in complexity theory. In this paper we study strict uniformity notions and mainly examine their impact on circuit classes below and including NC^1.

Borodin [Bor77] and Cook [Coo79] first imposed circuit uniformity by means of space-bounded Turing machines computing entire circuit descriptions. This worked well for NC^2 and above. Inspired by Goldschlager [Gol78], Ruzzo [Ruz81] then tied uniformity to the ability to answer local circuit connectivity queries, defining an ALOGTIME-uniformity notion under which NC^1 meaningfully equals ALOGTIME. Yet tighter uniformities were needed to investigate $AC^0 \subset ACC^0 \subseteq TC^0 \subseteq NC^1$. Barrington, Immerman and Straubing [BIS90] thus developed

ISSN 1433-8092

DLOGTIME-uniformity, proving that Immerman's model-theoretic notion [Imm87] and Buss' Turing machine-based notion [Bus87] were equivalent to their own robust notion.

Barrington, Immerman and Straubing [BIS90] also showed how to translate back and forth between an extended first-order formula φ describing a language L and a FO[bit]-uniform family of generalized bounded-depth circuits recognizing L. (Here φ is an ExtFO[bit] formula, that is, a FO formula using the bit predicate and using ordinary quantifiers as well as other quantifiers such as "there exist r values modulo q" and "monoidal quantifiers" implementing the product operation in the transformation monoid of a finite automaton; for their part, generalized bounded-depth circuits are Boolean circuits with oracle gates that perform the product operations underlying the quantifiers used in φ).

Roy and Straubing [RS07] later triggered the need for an even stronger notion of uniformity than DLOGTIME. Loosely speaking, Roy and Straubing proved that any regular language described in ExtFO[$+$] can be described without $+$. This was an interesting step in light of the conjecture [STT95,BCST92,Pél92,MPT91], central to the structure of NC^1 and expressed in crisp model-theoretic terms by Straubing [Str94, IX.3.4], that the ExtFO description of a regular language does not require the use of nonregular numerical predicates. Answering Roy and Straubing, the first and third authors [BL06] provided a circuit interpretation for ExtFO[$+$] by proposing a new encoding of circuit connections. This made it possible to speak meaningfully of FO[$+$]-uniform and FO[$<$]-uniform circuit families. The new uniformity, yet finer than FO[bit]-uniformity, meant that Roy and Straubing had separated a very uniform variant of ACC^0 from NC^1.

Having the framework of [BL06] at hand for ascribing circuit interpretations to very tightly-uniform subclasses of NC^1, it is of interest to study how these classes change as the uniformity changes. It is also of interest to determine the impact of tightening the uniformity on the problems that were shown complete for DLOGTIME-uniform NC^1 under DLOGTIME-reductions. This is the purpose of the present paper.

In a nutshell, our definitions of uniformity refine the FO[bit]-uniformity of [BIS90] and the FO[$<$]-uniformity of [BL06] in two directions. First, different renderings of the direct connection language of a circuit family as a language over a fixed alphabet are considered: in the *binary shuffled encoding*, the parameters describing a circuit gate are expressed in binary notation (following [BCGR92]) and intertwined (following [BL06]); in the *unary shuffled encoding*, the parameters are expressed in unary notation prior to intertwining. Second, a range of weak numerical predicates, such as the successor predicate $+1$ and the doubling predicate $\times 2$, are considered as replacements for $<$ and bit. For instance, FO[$+1$]-uniform$_{\mathrm{bin}}$ refers to the uniformity resulting from replacing $<$ with $+1$ under the binary shuffled encoding. (See Section 3 for details and precise definitions.) We show:

- In the FO[$<$]-uniform$_{\mathrm{un}}$ world, NC^1 collapses to NC^0, thus trivialising the class of languages defined from logarithmic depth bounded fan-in circuits.

This is shown by proving that the length of maximal paths in polynomial size graphs represented by FO[<] formulas cannot be logarithmic.

- In the FO[<, $\times 2$]-uniform$_{\text{un}}$ world, NC^1 includes DLOGTIME-uniform NC^1. Hence adding $\times 2$, a weaker predicate than + or bit, to FO[<] under the unary shuffle encoding suffices to simulate FO[bit]-uniformity on logarithmic depth circuits.
- In the FO[+1]-uniform$_{\text{bin}}$ world, NC^1 includes DLOGTIME-uniform NC^1. Hence replacing $<$ with the provably weaker predicate $+1$ under the binary shuffle encoding also suffices to simulate FO[bit]-uniformity on logarithmic depth circuits.
- In the FO[+1]-uniform$_{\text{bin}}$ world, polynomial size circuits capture P. This is another instantiation of the qualitative statement that complexity classes defined from a single machine are very uniform.

Our delineated results concerning NC^1 above raise the question of how extreme uniformity interacts with the classes sitting between NC^0 and NC^1. Many such classes, including AC^0, TC^0 and NC^1 itself, are the closure of specific languages under AC^0-reducibility. Recall the two major NC^1-complete problems, namely the formula value problem FVP [Bus87,BCGR92] and the word problem, which we will simply denote A_5, over the alternating group A_5 [Bar89] (tree isomorphism is a third NC^1-complete problem [JKMT03] but it behaves in many ways like FVP). A robust uniformity for AC^0-reductions is FO[bit]-uniformity [BIS90]. We can thus apply our tighter uniformity notions to closures of FVP and of A_5. Instead of FVP per se, we will use the NC^1-complete variant $\mathbb{D}_+$ [BKL] defined as the subset of the Dyck language over one pair of parentheses that encodes the FVP over the basis NAND (see Definition 2). Our results are:

- In the FO[<]-uniform$_{\text{un}}$ world, $\text{AC}^0(A_5) \subset \text{ACC}^0(A_5) \subset$ REGULAR. This uniformity is too strict for $\text{AC}^0(A_5)$ to capture even uniform TC^0.
- In the FO[+1]-uniform$_{\text{bin}}$ world, AC^0 recognizes $\{0^{2^n} : n \geq 0\}$ and this heavily depends on marginal properties of the shuffled encoding. This suggests that the binary shuffled encoding as a basis for defining uniform AC^0 closures is inadequate.
- In the FO[<]-uniform$_{\text{un}}$ world, $\text{AC}^0(\mathbb{D}_+) \supseteq \text{TC}^0$ and the same inclusion holds when linear fan-in is imposed on both circuit classes. The proof is a tight reduction from the majority language MAJ to the Dyck language and then from the latter to $\mathbb{D}_+$.
- In the FO[<]-uniform$_{\text{un}}$ world, the bit numerical predicate cannot be expressed in $\text{AC}^0(\mathbb{D}_+)_{LIN}$.

The last two points make FO[<]-uniform$_{\text{un}}\text{AC}^0(\mathbb{D}_+)_{LIN}$ an interesting candidate for a separation from FO[<]-uniform$_{\text{un}}\text{TC}^0_{LIN}$. It is known that FO[<]-uniform$_{\text{un}}\text{TC}^0$ can already simulate the bit numerical predicate. Hence, by our tight reduction, its $\mathbb{D}_+$ counterpart, i.e. FO[<]-uniform$_{\text{un}}\text{AC}^0(\mathbb{D}_+)$ can, too, and is therefore equal to uniform NC^1. If one believes that $\text{TC}^0 \neq \text{NC}^1$ then one might also suspect that FO[<]-uniform$_{\text{un}}\text{TC}^0_{LIN}$ does not equal FO[<]-uniform$_{\text{un}}\text{AC}^0(\mathbb{D}_+)_{LIN}$. A separation of the latter two classes is perhaps within the reach of current techniques due to the fact that FO[<]-uniform$_{\text{un}}$ $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ does not contain the bit numerical predicate. Further, a separation of these two classes would be interesting even if $\mathrm{TC}^0 = \mathrm{NC}^1$, because this would shed light on the internal structure of NC^1.

2 Preliminaries

We assume the reader to be familiar with circuits and thus only recall some fundamental definitions. The class NC^0 is the set of languages recognized by circuit families of constant depth and polynomial size built from bounded fan-in AND, OR and NOT gates. The class AC^0 is obtained instead when the AND and OR gates have unbounded fan-in. The class ACC^0 is further obtained when unbounded fan-in MOD_q gates are also permitted. The class TC^0 is the set of languages recognized by circuit families of constant depth and polynomial size built from unbounded fan-in MAJORITY gates. The class NC^1 is the set of languages recognized by circuit families of depth $O(\log n)$ built from bounded fan-in AND, OR and NOT gates. We write $\mathrm{AC}^0(G)$ for the class AC^0 in which the circuits are additionally equipped with unbounded fan-in gates of type G.

The *direct connection language* of a circuit family consists of the set of all tuples $\langle t, a, b, y\rangle$, where t is the type of gate a, b is a predecessor of a and y equals the number of inputs n. If a has no inputs, then b may be arbitrary, if a is an input gate, then t tells the letter and b the position to question. We allow gates that are either in a non-connected component or connected to input gates but not connected to the output gate. The size of the circuit is the size of the underlying graph and the depth is the length of the longest path of that graph. Note that this includes also gates that lead not to the output gate and unconnected components. Hence, the uniformity language may produce unnecessary gates and wires for the computation, but these unnecessary gates and wires still add up to the size of the circuit.

Definition 1 (Semantics of the direct connection language). *Let C_n be a circuit with n inputs and size $\leq n^c$. We label the gates by c-tuples of numbers from* 1 *to n. Then direct connection language of C_n is*

$$L_{C_n} = \{\langle t, \boldsymbol{a}, \boldsymbol{b}, y\rangle \mid \text{the gate labeled by } \boldsymbol{a} \text{ has type } t \text{ and has gate } \boldsymbol{b} \text{ as input}\}.$$

For a sequence of circuits $C = (C_1, \dots)$ the direct connection language is $L_C = \bigcup_n L_{C_n}$. The predecessors of a gate are fed into the gate in ascending order of their numbers. The output gate is always numbered by $(1, \dots, 1)$.

We have yet to fix an encoding that will turn the above direct connection language into a set of words over a fixed alphabet. We will define two such encodings in Section 3, inspired by [BIS90,BL06].

The *unary encoding* of a number $1 \leq i \leq n$ over $\Sigma = \{a, b\}$ is defined by $i \mapsto a^i b^{n-i}$. Similarly, the *binary encoding* of a number $0 \leq i < 2^n$ is defined by $w_1 \dots w_n \in \{0,1\}^n$ such that $i = \sum_j 2^{j-1} w_j$.

Given a list of words $w_1, \ldots, w_c$ with common length n over a common alphabet Σ, the *shuffle* of these words is the unique word u of length n over Σ^c, defined by setting the i-th letter of u to $(\sigma_1, \ldots, \sigma_c)$ iff σ_j is the i-th letter of w_j for $1 \leq j \leq c$.

When dealing with the Dyck language $\mathbb{D}_1$ over one pair of parentheses we will use the letters $\{a, b\}$ instead of $\{(,)\}$ to improve readability.

Definition 2 ($\mathbb{D}_+$ language [BKL]). *Any word in the Dyck language $\mathbb{D}_1$ corresponds to a tree. If we let every leaf of the tree be a false node and every inner node be a* NAND*, the tree is a formula evaluating either to true or false. We let $\mathbb{D}_+ \subset \{a, b\}^*$ be the set of words that are in the Dyck language and whose corresponding formula evaluates to true.*

We assume familiarity with first order logic and its application to words. We follow the notations of [Str94] and recall only some facts used in this paper. We write FO to denote first order logic and write the set of allowed numerical predicates in brackets. We will use the binary predicates bit, $<$, $+1$, and $\mathsf{x2}$. Since the value of a numerical predicate depends only on the position of the variables and the word length, we will freely switch between variables and natural numbers denoting their positions. To make this transition clearer we write "$x = i$" for a variable x and a natural number i when x points to the i-th position. The predicate $\mathsf{bit}(i, j)$ is true if the i-th bit of the binary representation of j is a 1. The successor predicate $+1(i, j)$ is true $i = j + 1$. The double predicate $\mathsf{x2}(i, j)$ is true if $i = 2j$. Recall that FO[<] only describes regular languages (it in fact captures the aperiodic regular languages [MP71]). The class FO[+1] is a proper subclass of FO[<] that in fact captures the threshold testable languages [Tho78].

The class FO can also be extended by adding additional quantifiers. We write FO + MAJ for the class FO which is also equipped with the majority quantifier.

3 Definitions

We now define two different encodings for the direct connection language. The first is the unary shuffled encoding introduced in [BL06]:

Definition 3 (Unary shuffled encoding). *Let $a_1, \ldots a_c$, $b_1, \ldots, b_c$, t, n be numbers between 1 and n. We let $\langle t, \boldsymbol{a}, \boldsymbol{b}, y\rangle_u$ denote the shuffled unary encoding of this $(2c + 2)$-tuple. (Note: $|\langle t, \boldsymbol{a}, \boldsymbol{b}, y\rangle_u| = n$)*

Encoding numbers in binary instead of unary yields our second encoding. Note that this second encoding differs from the encoding in [BIS90] not only in the shuffling, but in the fact that here y is also given in binary:

Definition 4 (Binary shuffled encoding). *For $n \in \mathbb{N}$, let $a_1, \ldots a_c$, $b_1, \ldots, b_c$, t be numbers between 1 and n. We let $\langle t, \boldsymbol{a}, \boldsymbol{b}, y\rangle_b$ denote the shuffled binary encoding of this $(2c + 2)$-tuple. (Note: $|\langle t, \boldsymbol{a}, \boldsymbol{b}, y\rangle_b| = \lceil \log(n + 1)\rceil$)*

We say that a circuit family (C_n) is FO$[X]$-uniform$_{\text{un}}$ (FO$[X]$-uniform$_{\text{bin}}$) if the language formed by the unary (resp. binary) shuffled encoding of the tuples in its direct connection language can be described by an FO$[X]$ formula.

We now define two gate types based on languages that are complete for NC^1, even in the case of DLOGTIME-uniformity.

Definition 5 **(A_5 Gate).** *Let A_5 be the alternating group over 5 elements and $g_0, g_1 \in A_5$ be two 5 cycles that span the whole group. An A_5 gate with k Boolean inputs $x_1, \ldots x_k$ evaluates to 1 if $g_{x_1} \ldots g_{x_k} = 1$, otherwise to 0.*

Recall that we write the Dyck language over the alphabet $\{a, b\}$.

Definition 6 **($\mathbb{D}_+$ Gate).** *A $\mathbb{D}_+$ gate with k Boolean inputs $x_1, \ldots x_k$ evaluates to 1 if replacing every 0 by a and every 1 by b in the word $x_1 \ldots x_k$ yields a word in $\mathbb{D}_+$.*

We will encounter the following classes, both in their FO$[<]$-uniform$_{\text{un}}$ and their FO$[+]$-uniform$_{\text{bin}}$ versions: AC$^0(A_5)$, AC$^0(\mathbb{D}_+)$, NC^1.

4 Uniform versions of log depth circuits

In this section we consider tightly uniform versions of NC^1. Our first theorem shows that FO$[<]$-uniform$_{\text{un}}$ NC^1 is too restrictive as the class collapses into NC^0. If we add a simple numerical predicate like x2 we obtain full DLOGTIME-uniform NC^1.

To prove our first theorem we show that FO$[<]$ cannot express an edge relation such that the resulting graph has paths of length $\Theta(\log n)$. We first explain why we need only to consider the expressiveness of FO$[<]$ in terms of numerical predicates.

The proofs in [BL06] relied heavily on the fact that a word in unary shuffled encoding can be translated to a tuple of variables and vice versa: For any k-ary numerical predicate expressible in FO$[<]$ one can easily construct a FO$[<]$ formula that recognizes exactly the unary shuffled encodings of all $(i_1, \ldots, i_k)$ of length n such that the numerical predicate is true for the variables $x_1, \ldots, x_k$ with $x_j = i_j$ $1 \leq j \leq k$. Conversely, for each formula φ recognizing a subset L of all valid unary shuffled encodings of k numbers one can construct a formula without Q_a predicates and free variables $x_1, \ldots, x_k$ $\varphi'(x_1, \ldots, x_k)$, such that if the shuffled encoding of $(i_1, \ldots, i_k) \in L$, then $\varphi(x_1, \ldots, x_k)$ is true for $x_j = i_j$ $1 \leq j \leq k$.

The expressive power of FO$[<]$ is well understood. One important restriction of FO$[<]$ is that a fixed formula can only count up to a constant. Beyond this constant it can only check the relative ordering. It is for example known that for quantifier depth d a formula cannot distinguish between the words a^{2^d} and a^{2^d+1}. We first prove a similar technical result which says that for a fixed formula ϕ with k free variables and sufficient large n holds: If there is a $\mathcal{V}$-structure w such that $w \models \phi$ of length n then there is a $\mathcal{V}$-structure w' of length $n-1$ that is also a model for ϕ.

Lemma 1. *Let $\phi(x_1, \ldots, x_k)$ be a* FO[<] *formula of quantifier depth d. If there is a V-structure w with $|w| \geq (k+1) \cdot (2^d + 1)$ such that $w \models \phi$ then there is a V-structure w' of length $|w| - 1$ such that $w' \models \phi$.*

Proof. Consider the positions of all x_i on w. Without loss of generality we assume that $x_i < x_{i+1}$. These k variables split our word w in $k+1$ intervals. Since $|w| \geq (k+1) \cdot (2^d + 1)$ one of the intervals has to be greater than (2^d), i.e., of the following conditions must be true: (i) $x_1 \geq 2^d + 1$, (ii) $n - x_k \geq 2^d + 1$ (iii) there are two consecutive variables x_i, x_{i+1} with $x_{i+1} - x_i - 1 \geq 2^d + 1$. If all the intervals would have length $\leq 2^d$ and there are k variable that occupy k position, the word length would be bounded by $2^d \cdot (k+1) + k = (2^d + 1) \cdot (k+1) - 1$,

We construct a new word w' from w by removing one of the letters of w. If we are in the case (i) we remove the first letter from w, for (ii) the last letter. For case (iii) we remove one letter between x_{i+1} and x_i.

We have to show that in each case $w \models \phi$ if $w' \models \phi$.

It is known that a FO[<] formula of quantifier depth d cannot distinguish between words of length 2^d and $2^d + 1$, see Theorem IV.2.1 in [Str94]. This can be proved using Ehrenfeucht-Fraïssé games and we call duplicator's strategy A.

The strategy for duplicator is as follows: assume that spoiler puts a pebble in one of the intervals that was not modified then duplicator answers at the same position within this interval. In the case that spoiler puts a pebble in the interval that was modified duplicator plays the strategy A within this interval. □

Lemma 1 allows us to prove the following theorem:

Theorem 1. FO[<]-uniform$_{\mathrm{un}}$ NC^1 =FO[<]-uniform$_{\mathrm{un}}$ NC^0

Proof. We prove the theorem by showing that FO[<] cannot define graphs having the property that the length of maximal paths is sublinear but not constant. If one could define NC^1 circuits with logarithmic depth in FO[<]-uniform$_{\mathrm{un}}$ then the underlying graph of the circuit defined by the formula checking the uniformity language, could be used as a formula that defines the edge relation for such graphs.

Assume there exists a family of graphs (G_n) with vertices labeled by k-tuples whose edge relation $E(\boldsymbol{x}, \boldsymbol{y})$ is described by the formula $\phi(x_1, \ldots, x_k, y_1, \ldots y_k)$ of quantifier depth d. Let $l(n)$ be the length of a longest path in G_n. We will show that $l \in o(n)$ implies $l \in O(1)$. The idea is to show that for large enough n the following holds: For a path π_n of length $l(n)$ we can construct a path π' in G_{n-1} of length $l(n)$, so the longest path in G_{n-1} has length at least $l(n)$. This implies that $l(n) \leq l(n-1)$ for all n large enough and hence l is bounded for n large enough and therefore in $O(1)$. Since $l(n)$ is in $o(n)$ there is a value N_0 such that for all $n > N_0$ we have $n \geq (l(n) \cdot k + 1) \cdot (2^d + 1)$.

Choose any $n > N_0$. Let $\pi_n = (v^1, \ldots, v^l)$ be a path of length l with $l = l(n)$. Let $\boldsymbol{x}^i$ be the tuple of variables $(x^i_1, \ldots, x^i_k)$ denoting v^i. The formula $\bigwedge_{i=1}^{l-1} \phi(x^i, x^{i+1})$ is true iff π_n exists and the x^is are the labels of the vertices in π_n. So we will apply Lemma 1 and obtain as set of variables $\boldsymbol{x}'^1, \ldots, \boldsymbol{x}'^l$ on a word of length $n-1$. We can interpret the tuple $\boldsymbol{x}'^i$ as the label of a vertex v'_i in

G_{n-1} and by 1 we know that $\pi' = (v^1, \ldots, v^l)$ forms a path in G_{n-1}. It follows that $l(n-1) \geq l(n)$.

So we have shown that for all $n > N_0$ we have $l(n-1) \geq l(n)$, by induction we get that $l(N_0) \geq l(n)$. Since $l(n)$ is bounded by $l(N_0)$ it follows that l is bounded, and hence in $O(1)$. $\square$

While FO[<] cannot describe circuits of logarithmic length adding a simple binary predicate like $\mathsf{x2}$, which is much weaker than for example $+$, already allows to describe DLOGTIME uniform circuits:

Theorem 2. *The classes* FO[<, $\mathsf{x2}$]-uniform$_{\mathrm{un}}$NC^1 *and* FO[+1]-uniform$_{\mathrm{bin}}$NC^1 *each contain* DLOGTIME-*uniform* NC^1.

Proof. We let M be a ALOGTIME machine that recognizes the language of the DLOGTIME-uniform NC^1 circuit. Our model is similar to the one of Ruzzo [Ruz81], with the exception that we do not use an extra index tape. The index tape of the ALOGTIME machine are the first $\log n$ bits to the right of the head of the working tape and the machine queries the input only in the last step by switching into a state s_σ. Further we choose c such that the ALOGTIME machine uses at most $c \log n$ steps in every run.

We will build a FO[<, $\mathsf{x2}$]-uniform$_{\mathrm{un}}$ NC^1 circuit that recognizes the same language. We label the gates of the circuits by a tuple: $(t_1, \ldots, t_c, s, l, l_1, \ldots, l_c, r, r_1, \ldots, r_c)$. The idea is the following: $(t_1, \ldots, t_c)$ denotes the time step of the machine. We can count from 1 to $x \log n$ by starting from $(t_1, \ldots, t_e)$ and doubling t_1 each time, until $2 \cdot t_1$ would be greater than n, then we continue by doubling t_2 and so on. In the following, we write $\boldsymbol{x}$ to denote the vector $x_1, \ldots, x_c$. A tuple $s, \boldsymbol{l}, \boldsymbol{r}$ will denote the configuration of the machine, where s is the state of the machine and $\boldsymbol{l}$, $\boldsymbol{r}$ are the parts of the working tape left and right of the head. The auxiliary variables l and r will keep track of how many bits of l_1 and r_1 are used. Hence, the type of the gate depends directly on s, the only problem is to test if two gates are predecessors. Given two gates $(\boldsymbol{t}, s, l, \boldsymbol{l}, r, \boldsymbol{r})$, $(\boldsymbol{t}', s', l', \boldsymbol{l}', r', \boldsymbol{r}')$ we have to check the following conditions:

Time The formula has to ensure that $\boldsymbol{t}'$ describes one timestep after $\boldsymbol{t}$. We say that an entry t_i is maximal if there is no position z such that $z = \mathsf{x2}(t_i)$. By our encoding the formula has to check: There is an i such that $t_j = t'_j$ for $j < i$ and the t_j are maximal. For i we have $\mathsf{x2}(t_i) = t'_i$ and $t_k = t'_k$ are at the first position for $k > j$.

Tape We assume that the lowest bit of l is the bit under the working head. We can check if that bit is 0 by checking if l_1 is even, i.e. $\exists x x = \mathsf{x2}(l_1)$. We can write a 0 on that position by first shifting l to the right and then to the left, i.e. the formula first determines the largest x such that $\mathsf{x2}(x) \leq n$ and takes then $\mathsf{x2}(x)$. To write 1 we take $\mathsf{x2}(x) + 1$ of said x. The case of r_1 is handled analogously. Before writing a 1, the formula has to make sure that $\mathsf{x2}(x) + 1 \leq n$. If that is not the case, we copy all $l_i \neq 0$ to l_{i+1} and set $l_1 = 0$. This means, we just shift the vector $(l_1, \ldots, l_c)$ to the right.

State and head movement The transition from s to s' can be checked by reading the lowest bit of l_1. The movement of the head to the right is simulated by reading the lowest bit of l_1, shifting l_1 to the right, and writing that bit on r_1. Note, if l_1 was already zero, we have to shift $(l_1, \ldots, l_c)$ to the left, i.e. set l_i to l_{i+1}. We keep track of the bits stored in l_1 and r_1 by the variables l, r. This way we can distinguish between the vectors 1 and 10 for example. In both cases l_1 would equal 1, but l is 1 in the first case and 2 in the second.

Input The input gates are labeled by (σ, i), where i is given in unary. If the ATM makes a transition into state $READ(\sigma)$, we connect to the input gate (σ, l_j), where j is the maximal used block.

Note that the so constructed circuit also connects a lot of configurations that are not reachable by the ALOGTIME machine. Still any path in the circuit has a length of at most $c \log n$ because of t in our labeling. Hence, the circuit fulfills the required depth restrictions.

The same construction works for FO[+1]-uniform$_{\text{bin}}$ NC^1 circuits. We do not need a double predicate here since we only need to check if the binary numbers are shifted by one. □

We obtain a dichotomy for our uniformity definitions and circuits of logarithmic depth. The classes result either in subclasses of NC^0 or contain DLOGTIME-uniform NC^1. The fact that even low uniform circuit classes capture DLOGTIME-uniform NC^1 seems to stem from its equivalence to ALOGTIME. Turing machines are uniform and operate only locally. As noted in [BL06] this also allows strict uniform circuit characterizations for polynomial time. In Section 6 we discuss how this can be extended to a general framework.

5 Uniform versions of NC^1-complete problems

In this section we will not consider log depth circuits but constant depth circuits equipped with gates that compute NC^1 complete languages. We consider $\mathbb{D}_+$ and the word problem over A_5 as complete problems.

Extending AC^0 by A_5 gates gives a proper subclass of the regular languages.

Theorem 3. FO[<]-uniform$_{\text{un}}$ AC$^0(A_5) \subset$ REGULAR.

Proof. This follows by translating the circuit into a FO + A_5[<] formula and applying Theorem 11.6 from [BIS90]. □

Actually the previous proof does not only show that FO[<]-uniform$_{\text{un}}$ AC$^0(A_5)$ circuits are contained in the regular language but that even FO[<]-uniform$_{\text{un}}$ ACC$^0(A_5)$ recognize only a subset of the regular languages. The only way to obtain something containing an acceptably large subclass of TC^0 seems to be the bit predicate, but this immediately yields uniform NC^1. If we switch to binary encoding we do not obtain a suitable class either:

While FO[+1]-uniform$_{\text{bin}}$AC$^0(A_5)$ can compute if the word length is a power of 2, there is no indication that it can compute if the word length is a power of 3. The problem with binary encoding in this case is that the uniformity can exploit the representation. Hence, binary encoding differs strongly from ternary encoding. This does not matter for log depth circuits as in Section 4 where we immediately obtain ALOGTIME.

So instead of choosing gates based on finite non-solvable groups, we choose a gate type that corresponds to the Boolean formula value problem. Combining [BIS90], [LMSV01], and [BL06] yields that FO[<]-uniform$_{\text{un}}$TC^0 is separated from
FO[<]-uniform$_{\text{un}}$TC^0 with linear fan-in. While the former can simulate the bit predicate and is hence equal to DLOGTIME-uniform TC^0 the latter equals $\text{FO} + \text{MAJ}[<]$ and cannot simulate the bit predicate.

We can show that the inclusion chain under DLOGTIME uniformity remains valid for FO[<] uniformity in unary shuffled encoding.

Theorem 4. FO[<]-uniform$_{\text{un}}$ AC$^0(\mathbb{D}_+) \supseteq$ FO[<]-uniform$_{\text{un}}$TC^0
FO[<]-uniform$_{\text{un}}$ AC$^0(\mathbb{D}_+)_{LIN} \supseteq$ FO[<]-uniform$_{\text{un}}$TC$^0_{LIN}$.

Proof. We recall the definitions of the following languages: $MAJORITY = \{w \in \{0,1\}^* \mid \#_1(w) > \#_0(w)\}$, $EQUALITY = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$, and $\mathbb{D}_1 = \{w \in EQUALITY \mid \forall u, v \text{ with } uv = w : \#_a(u) \geq \#_b(u)\}$. We need to show that we can simulate a MAJ gate by a $\mathbb{D}_+$ gate in the given uniformity. To increase readability we use the alphabet $\{a,b\}$ instead of $\{0,1\}$ for the EQUALITY and Dyck languages. We reserve $\{0,1\}$ as the alphabet of the MAJORITY language. In order to do so we show how to reduce the majority language to the one sided Dyck language over a single pair of parentheses $\mathbb{D}_1$ and then reduce this to $\mathbb{D}_+$. Let us begin by reducing the language EQUALITY, i.e. $\{w \in \{0,1\}^* \mid \#_1(w) = \#_0(w)\}$, to $\mathbb{D}_1$. We define two morphisms $\pi_a, \pi_b : \{0,1\}^* \to \{a,b\}^*$ by letting $\pi_a(1) = aa, \pi_a(0) = ab$, $\pi_b(1) = ab, \pi_b(0) = bb$. We claim that the mapping $w \mapsto \pi_a(w)\pi_b(w)$ is a reduction from EQUALITY to $\mathbb{D}_1$. Observe that for any $w \in \{0,1\}^*$ $\pi_a(w)$ is always a valid prefix for a word in $\mathbb{D}_1$ and $\#_a(\pi_a(w)) - \#_b(\pi_a(w)) = 2(\#_1(w) - \#_0(w))$. Similarly, π_b only produces valid suffixes and we have $\#_b(\pi_a(w)) - \#_a(\pi_a(w)) = 2(\#_0(w) - \#_1(w))$. It follows that $\pi_a(w)\pi_b(w) \in \mathbb{D}_1 \Leftrightarrow \#_1(w) = \#_0(w)$.

To obtain a reduction from MAJORITY to $\mathbb{D}_1$ observe that

$$\#_a(\pi_a(w)\pi_b(w)) \geq 0 \Leftrightarrow \#_1(w) \geq \#_0(w)$$

(and $\pi_a(w)\pi_b(w)$ is a prefix of a Dyck word). Hence, if the number of 1's in w is less than half there is no suffix z such that $\pi_a(w)\pi_b(w)z \in \mathbb{D}_1$. We define two more morphisms that will build the Dyck inverse for $\pi_a(w)\pi_b(w)$: Let $\bar{\pi}_a(1) = bb, \bar{\pi}_a(0) = ab$, $\bar{\pi}_b(1) = ab, \bar{\pi}_b(0) = aa$. Now we have a reduction from MAJORITY to $\mathbb{D}_1$ by the mapping $w \mapsto \pi_a(w)\pi_b(w)\bar{\pi}_b(w)\bar{\pi}_a(w)$.

We sum up the idea: we open $\#_1(w)$ many parentheses, then close $\#_0(w)$ many parentheses, then open $\#_0(w)$ many parentheses, and finally close $\#_1(w)$ many parentheses. This expression is valid if no more parentheses in the middle are closed than are opened before, i.e. $\#_1(w) \geq \#_0(w)$. So actually this accepts words with an equal number of 1's and 0's. This can be fixed in the logic by excluding this case through testing on equality as shown above. We have that $w \in L_{MAJ} \Leftrightarrow \pi_a(w)\pi_b(w)\bar{\pi}_b(w)\bar{\pi}_a(w) \in \mathbb{D}_1$.

We reduce $\mathbb{D}_1$ to $\mathbb{D}_+$ by mapping w to $aabbw$. The reduction creates a tree, with a node, where the left subtree defined by $aabb$ evaluates to false and the right subtree is defined by w. Therefore since all gates of the tree are NAND gates, the whole tree always evaluates to true iff w is a correct word in $\mathbb{D}_1$. If w is not in $\mathbb{D}_1$ the whole word will be outside of $\mathbb{D}_1$ and hence not in $\mathbb{D}_+$.

To have this reduction work in FO[<], we need a reduction that has very limited computational power. We will generate a reduction that reduces a word of length n to a word of length $10n$. The position will be as tuples where (x, y) stands for position $x + ny$ with $x \in \{1, \ldots, n\}$ and $y \in \{1, \ldots, 10\}$. The position $x + ny$ of the reduced word will depend only on x and also either the input at position x, or will be a constant for all $x > 4$. We use the following observations:

First, the reduction from L_{MAJ} to $\mathbb{D}_1$ remains valid if we split the morphism π_a into two morphisms $\pi_a^1(a) = a, \pi_a^1(b) = a$, $\pi_a^2(a) = a, \pi_a^2(b) = b$. We split the other three morphisms in the same fashion and observe that still holds

$$w \in L_{MAJ} \Leftrightarrow \pi_a^1(w)\pi_a^2(w)\pi_b^1(w)\pi_b^2(w)\bar{\pi}_b^1(w)\bar{\pi}_b^2(w)\bar{\pi}_a^1(w)\bar{\pi}_a^2(w) \in \mathbb{D}_1.$$

Second, given a word w of length $n \geq 4$ we can reduce $\mathbb{D}_1$ to $\mathbb{D}_+$ by mapping w to $aabba^{n-4}aabbb^{n-4}w$. The word $a^{n-4}aabbb^{n-4}$ is in $\mathbb{D}_1$ so behaves neutral in the morphism.

Summarizing, we map a word of length n to a word of length $10n$ to obtain a reduction from L_{MAJ} to $\mathbb{D}_+$. Where

$$w \mapsto aabba^{n-4}aabbb^{n-4}\pi_a^1(w)\pi_a^2(w)\pi_b^1(w)\pi_b^2(w)\bar{\pi} : a^1(w)\bar{\pi}_b^2(w)\bar{\pi}_a^1(w)\bar{\pi}_a^2(w)$$

This construction can be carried out in FO[<]-uniformity by using a variable which is bound to the positions $\{1, \ldots, 10\}$. $\square$

We mention that MOD_q gates can be simulated in FO[<]-uniform$_{\mathrm{un}}$ TC^0_{LIN}. Together with the proof above it follows that

$$\text{FO}[<]\text{-uniform}_{\text{un}}\mathrm{AC}^0(\mathbb{D}_+) \supseteq \text{FO}[<]\text{-uniform}_{\text{un}}\mathrm{TC}^0 \supseteq \text{FO}[<]\text{-uniform}_{\text{un}}\mathrm{ACC}^0$$

These inclusions remain valid for the corresponding circuit classes with linear fan-in. The following theorem shows that $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ is strictly weaker than NC^1, but it is still not clear whether it is contained in (even non uniform) TC^0.

Theorem 5. *The predicate $*$ (and hence* bit*) cannot be expressed in* FO[<]-uniform$_{\mathrm{un}}\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$.

Proof. This follows from Theorem 4.16 in [LMSV01]. $\square$

Theorem 4 and Theorem 5 highlight the important difference between $\mathrm{AC}^0(\mathbb{D}_+)$ and $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$; indeed the former is able simulate bit and thus equals NC^1.

Note that superlinear fan-in of gates corresponds to quantifiers over tuples of variables in the logic world.[3]

Summarizing we are able to say: in contrast to A_5 gates, using $\mathbb{D}_+$ gates yields a class that is weaker than DLOGTIME uniform NC^1 but contains uniform subclasses of TC^0_{LIN}. Hence, FO[<]-uniform$_{\mathrm{un}}$ $\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ is a candidate subclass of NC^1 worthy of attempts to separate it from TC^0.

6 A guide to minimal Uniformity

In Section 4 we noticed that log depth circuits with very tight uniformity can already simulate circuits which are much more non-uniform. This phenomenon is much more general and we explore it in this section. We start by showing that polynomial time admits very uniform circuits over the standard Boolean gates.

Theorem 6. *The following circuit families define the same class of languages: PTIME-uniform polynomial size circuits,* FO[+1]-uniform$_{\mathrm{bin}}$ *polynomial size circuits,* FO[+1]-uniform$_{\mathrm{un}}$ *polynomial size circuits.*

Proof. It is known that the first circuit class equals P. The latter two classes are also easily seen to be in P, it remains to show that P is contained in those classes. To see how P can be translated into FO[+1]-uniform circuits recall how a TM M of running time n^k is simulated by circuits ([Lad75]). The basic layout of the circuit is a grid of $2n^k \times n^k$ subcircuits. A subcircuit at position (p, t), will compute the state of the tape at position p at time t. Since such a subcircuit has a constant size, we will enumerate the gates of the circuit by tuples of size $2k + k + c$, where c depends on the size of the subcircuit.

Such a subcircuit encodes the state of a tape cell at position p and time t, i.e. the symbol written on it and eventually the state of the TM, if the head is over the tape cell, by a constant number of, say c, output gates. Its inputs are connected to $3c$ input gates, namely the output gates of the subcircuits at positions $(p-1, t-1)$, $(p, t-1)$, and $(p+1, t)$.

To see why the resulting circuit is FO[+1]-uniform, we observe what has to be checked. To see if two gates, each encoded by a $3k + c$ tuple (p, t, g) are connected, the formula has to check two cases: (Let the gates be (p_1, t_1, g_1) and (p_2, t_2, g_2))

1. $p_1 = p_2, t_1 = t_2$, the wiring is within a subcircuit and hence a finite function ranging over the possible constant values of (g_1, g_2). All such functions are in FO[+1].
2. $t_1 = t_2 - 1$. In this case p_1 is either equal to p_2, $p_2 + 1$, or $p_2 - 1$. This is checkable in FO[+1]. Furthermore, the formula has to check, if g_1 and g_2 have the correct number, but the same argument as above applies.

[3] For the A_5 quantifier we do not have to distinguish between quantifiers over one variable and quantifiers over tuple of variables.

For the cases of $t = 1$ and $t = n^k$ we have to add special cases, one layer translating the input into tape cells, the other checking with a big OR gate, if one tape cell at the last time step has a head in an accepting state.

Each of these cases, as well as the cases at the border of the grid can be handled similar to above by a FO[+1]-formula.

It is also easy to see that we can switch from unary to binary shuffled encoding. The tests for the fixed cases are clearly in FO[+1], for the other tests the formula must compute ± 1 on binary numbers, but this can be done in FO[+1]. $\square$

The proof of Theorem 2 builds the circuit for an ALOGTIME machine from its configuration tree. Both theorems exploit the locality of a computational step of a Turing machine. We discuss possible extensions of these theorems and under which conditions might lead to a more general framework.

An obvious extension to Theorem 6 is to consider more general complexity classes of Turing machines. Consider a complexity class $\mathcal{M}$ defined by time and space bounds. When simulating a TM in $\mathcal{M}$ as a circuit then time translates to depth and space to width. If one equips FO[<] with unary predicates that allow to check the depth and width bounds, it is possible to perform the construction without much overhead. Therefore, a (deterministic) Turing machine can be simulated by very uniform circuits. If now conversely such a circuit can be evaluated by a TM in $\mathcal{M}$ then any $\mathcal{M}$-uniform $\mathcal{C}$ circuit can be converted to a FO[$<, \mathsf{p_B}$]-uniform $\mathcal{C}$ circuit. Here $\mathsf{p_B}$ stands for a set of unary predicates that allow to check the bounds on the circuit. This construction requires some minimal closure properties for the function for the functions giving the time and space bounds on the machine. The idea is to take the machine that evaluates a circuit in $\mathcal{M}$-uniform $\mathcal{C}$ and to construct the FO[$<, \mathsf{p_B}$]-uniform circuit for this machine. An example is polynomial time where P-uniform polynomial size circuits equal FO[<]-uniform polynomial size circuits. (Here, $\mathsf{p_B}$ can be directly expressed within FO[<].)

Similar observations hold for alternating Turing machines as exhibited in [Ruz81]: for $k > 1$ the different notions of uniformity define identical circuit families including LOGSPACE-uniform families. Then, it is easy to see how to extend the construction of Theorem 2 to circuits of depth $\log^k$.

We believe that the requirement for $\mathcal{C}$ to be contained in some Turing class can be omitted. Let M be the machine that would decide the uniformity language. The idea is the following: Let a be a gate and b be a possible candidate to be a predecessor. Instead of letting the uniformity language to decide whether there is a wire from b into a, we build a circuit, that will evaluate M and then either feed in b or not. (Note this requires to be able to feed in a neutral input.) This is done for all possible gates b for a. We call this construction a "switch gate". So we need to be able to simulate M in the circuit class $\mathcal{C}$. This idea is can be already found in [Ruz81].

This is an explanation why log-depth circuits seem to be always at least DLOGTIME-uniform as exhibited in Section 4.

7 Discussion

In this paper we considered uniform versions of NC^1 with stricter uniformity notions than DLOGTIME. Our motivation was to search for classes between TC^0 and NC^1 defined by uniformity that might be interesting candidates for separation attempts.

Considering logarithmic depth circuits proved unsuccessful since we obtained either the full power of ALOGTIME or NC^0. The fact that we can find strictly uniform circuit classes for ALOGTIME is based on the exploitation of uniformity and locality of the steps of a Turing machine. This also allows FO[<]-uniform$_{\mathrm{un}}$ characterizations of polynomial time as observed in [BL06]. We think that this could be extended to circuit classes that have characterizations in terms of Turing machines, or to circuit classes whose uniformity languages are defined by Turing machines as long as the circuit class is at least as powerful as the uniformity language.

So we examined other characterizations of NC^1, namely the AC^0 closures of A_5 and of the formula value improved problem in the form of the $\mathbb{D}_+$ language. While A_5 did not yield a satisfying subclass, we could show that the FO[<]-uniform AC^0 closure of $\mathbb{D}_+$ contains TC^0. While the FO[<]-uniform AC^0 closure of $\mathbb{D}_+$ equals NC^1, the version with linear fan-in is strictly weaker than NC^1 but contains its TC^0 counterpart, namely FO[<]-uniform$_{\mathrm{un}}\mathrm{TC}^0_{LIN}$. We leave open the question of separating the two equally uniform classes FO[<]-uniform$_{\mathrm{un}}\mathrm{AC}^0(\mathbb{D}_+)_{LIN}$ and FO[<]-uniform$_{\mathrm{un}}\mathrm{TC}^0_{LIN}$. Such a separation would amount to distinguishing the power of MAJ from the power of $\mathbb{D}_+$.

Perhaps one other research avenue would be to consider direct connection language encodings that are intermediate between the unary shuffled encoding and the binary shuffled encodings studied here.

References

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BCGR92] Samuel R. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992.

[BCST92] David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.

[BIS90] David A. Mix Barrington, Neil Immerman, and Howard Straubing. On Uniformity within NC^1. *J. Comput. Syst. Sci.*, 41(3):274–306, 1990.

[BKL] Christoph Behle, Andres Krebs, and Klaus-Jörn Lange. The Boolean Formula Value Problem as a Formal Language. In preperation.

[BL06] Christoph Behle and Klaus-Jörn Lange. FO[<]-Uniformity. In *IEEE Conference on Computational Complexity*, pages 183–189, 2006.

[Bor77] Allan Borodin. On relating time and space to size and depth. *SIAM J. Comput.*, 6(4):733–744, 1977.

[Bus87] Samuel R. Buss. The boolean formula value problem is in alogtime. In *STOC*, pages 123–131. ACM, 1987.

[Coo79] Stephen A. Cook. Deterministic CFL's are accepted simultaneously in polynomial time and log squared space. In *STOC*, pages 338–345. ACM, 1979.

[Gol78] Leslie M. Goldschlager. A unified approach to models of synchronous parallel machines. In *STOC*, pages 89–94, 1978.

[Imm87] Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.

[JKMT03] Birgit Jenner, Johannes Köbler, Pierre McKenzie, and Jacobo Torán. Completeness results for graph isomorphism. *J. Comput. Syst. Sci.*, 66(3):549–566, 2003.

[Lad75] Richard E. Ladner. The circuit value problem is log space complete for p. *SIGACT News*, 7:18–20, January 1975.

[LMSV01] Clemens Lautemann, Pierre McKenzie, Thomas Schwentick, and Heribert Vollmer. The descriptive complexity approach to LOGCFL. *J. Comput. Syst. Sci.*, 62(4):629–652, 2001.

[MP71] Robert McNaughton and Seymour Papert. *Counter-free automata. With an appendix by William Henneman.* Research Monograph No.65. Cambridge, Massachusetts, and London, England: The M. I. T. Press. XIX, 163 p., 1971.

[MPT91] Pierre McKenzie, Pierre Péladeau, and Denis Thérien. NC^1: The automata-theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991.

[Pél92] Pierre Péladeau. Formulas, regular languages and boolean circuits. *Theor. Comput. Sci.*, 101(1):133–141, 1992.

[RS07] Amitabha Roy and Howard Straubing. Definability of languages by generalized first-order formulas over N^+. *SIAM J. Comput.*, 37(2):502–521, 2007.

[Ruz81] Walter L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22(3):365–383, 1981.

[Str94] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, 1994.

[STT95] Howard Straubing, Denis Thérien, and Wolfgang Thomas. Regular languages defined with generalized quanifiers. *Inf. Comput.*, 118(2):289–301, 1995.

[Tho78] Wolfgang Thomas. The theory of successor with an extra predicate. *Math. Annalen*, 237:121–132, 1978.

ECCC ISSN 1433-8092

http://eccc.hpi-web.de