$FO[<]$-Uniformity

Christoph Behle and Klaus-Jörn Lange
Wilhelm–Schickard–Institut für Informatik
Eberhard–Karls–Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
{behlec,lange}@informatik.uni-tuebingen.de

Abstract

Uniformity notions more restrictive than the usual $FO[<,+,*]$-uniformity = $FO[<,Bit]$-uniformity are introduced. It is shown that the general framework exhibited by Barrington et al. still holds if the fan-in of the gates in the corresponding circuits is considered.

1. Introduction

Barrington et al. showed that the uniform version of well-known complexity classes like AC^0, ACC^0, or TC^0 can be characterized in terms of languages described by logical formulae using the Bit-predicate. In this work we show that their framework essentially remains true when we restrict the uniformity formulae just to use the $<$ predicate.

One motivation for this investigation is to investigate classes like $\mathcal{L}(FO[<,+])$ or $\mathcal{L}(FO+Mod[<,+])$ where it is possible to find lower bounds ([2, 6]). We are able to provide the first characterizations in terms of circuits for these classes answering a question posed by Roy et al..

Going down to $FO[<]$-uniformity we can show that the resulting AC^0 and ACC^0-circuits can describe only regular languages.

Finally, we consider circuits built of majority gates where we can show that $FO[<]$- and $FO[<,+]$-uniformity coincide giving a first circuit characterization of the class $\mathcal{L}(Maj[<])$ ([4, 3]).

A main ingredient of our techniques is to shuffle vectors of numbers represented in unary coding. To handle majority quantifiers and gates the definability of counting quantifiers in $Maj[<]$ is an essential building block. As a result we can express the qualitative difference between $\mathcal{L}(Maj[<])$ and $\mathcal{L}(Maj[<,Bit])$ by the quantitative difference of allowing only linear or superlinear fan-in in TC^0-circuits.

2. Preliminaries

We will use essentially the logical notation as it is presented in the paper of Barrington, Immerman, and Straubing [1]. In the following we denote by $\mathbb{N}$ the positive integers and for a sequence $\vec{\alpha} \in \mathbb{N}^d$ we define $l(\vec{\alpha}) := d$.

2.1. Logic formulae over words

Throughout this paper we consider languages defined by logical formulae. In general, i, j, k, n will denote positive integers, while x, y, z will denote position variables with positive integer values. Thus, variables will range over $\{1, 2, \ldots, n\}$ where n is the length of the input word. The predicate $C_a(x)$ expresses that variable x is pointing to a position containing the symbol a.

As usual, a formula ϕ with set $\mathcal{V}$ of free variables is interpreted over words as structures $w = (a_1, \mathcal{V}_1)(a_2, \mathcal{V}_2) \cdots (a_n, \mathcal{V}_n)$ such that $\mathcal{V}$ is the union of the $\mathcal{V}_i$ and that the $\mathcal{V}_i$ are pairwise disjoint. Words of this kind are called $\mathcal{V}$*-structures*. A letter $(a, \emptyset)$ is simply denoted by a. We denote the set of all $\mathcal{V}$-structures over Σ by $\Sigma^* \otimes \mathcal{V}$ whereas $(\Sigma \times 2^{\mathcal{V}})^*$ contains also words with multiple occurrences of a variable. The set of $\mathcal{V}$-structures modeling ϕ is denoted by $L_{\phi,\mathcal{V}}$. If ϕ is a sentence, then $L_\phi = L_{\phi,\emptyset} = \{w \in \Sigma^* \mid w \models \phi\}$. We call this the set of words defined by ϕ.

If $w = (a_1, \mathcal{V}_1)(a_2, \mathcal{V}_2) \cdots (a_n, \mathcal{V}_n) \in \Sigma^* \otimes \mathcal{V}$ and $x \notin \mathcal{V}$ then $w_{x=i}$ denotes $(a_1, \mathcal{V}_1) \cdots (a_{i-1}, \mathcal{V}_{i-1})(a_i, \mathcal{V}_i \cup \{x\})(a_{i+1}, \mathcal{V}_{i+1}) \cdots (a_n, \mathcal{V}_n) \in \Sigma^* \otimes (\mathcal{V} \cup \{x\})$. As abbreviation for $w_{x=i} \models \phi$ we often write $w \models \phi(x = i)$, or, if x is understood, $w \models \phi(i)$.

All quantifiers in this paper like $\exists, \forall$ are also allowed to be quantified over tuples of variables which is indicated by a subscript, i.e. $\exists_2$ for existential quantification over pairs or $\forall_3$ for universal quantification over triplets of vaiables. Observe that for first order quantifiers quantification over tuples is equivalent to the iteration over single variables, i.e.: the formula $\exists_k\ (x_1, \cdots, x_k)\ \Phi$ is equivalent to

$\exists\, x_1 \cdots \exists\, x_k\ \Phi$. Let $\mathcal{Q}$ be a set of quantifier types (e.g. first-order) and $\mathcal{X}$ a set of numerical predicates (not necessarily containing the order predicate $<$). We denote by $\mathcal{Q}[\mathcal{X}]$ the set of all formulae built over the elements of $\mathcal{X} \cup C_a(\cdot)$ as atomic formulae by conjunction, negation and quantification using quantifier types from $\mathcal{Q}$. By $\mathcal{L}(\mathcal{Q}[\mathcal{X}])$ we denote the class of all languages definable by $\mathcal{Q}[\mathcal{X}]$ formulae. If the quantors in $\mathcal{Q}$ are allowed to quantifiy over tuples of variables we denote this by the subscript $tuple$, i.e.: using $\mathcal{Q}_{tuple}$.

Formulae not using the $C_a(\cdot)$ predicates describe *numerical predicates*. In the presence of the order predicate first-order quantifiers can define addition and multiplication by use of the BIT predicate and vice versa. First order quantifiers can express the order predicate with the help of several numerical predicates, for instance by addition and by the Bit predicate. To keep things clear we allways include the order predicate explicitly. A very comprehensive treatment of related results is given by Schweikardt [7].

$\exists^{r\ mod\ k}\, x$ denotes the modular counting quantifier which evaulates to true if the number satisfying positions of the input word is equal to $r\, mod\, k$ for $0 \leq r < k$. The modular counting quantifier over tuples can be simulated by the iteration of modular counting over single variables (but not vice versa). For instance we have

$$\exists^{r\ mod\ k}\ (x, y)\ \Phi \Leftrightarrow$$

$$\bigvee_{r=a_1+2a_2+\cdots(k-1)a_{k-1}\ mod\ k}\ \bigwedge_{\mu=0}^{k-1} \exists^{a_\mu\ mod\ k}\ x\ \exists^{\mu\ mod\ k}\ y\ \Phi.$$

We will use $Maj\, x$ to denote the majority quantifier (over single variables). $w \models Maj\, x\phi$ is fulfilled iff the number of all $1 \leq i \leq |w|$ such that $w_{x=i} \models \phi$ is larger than $|w|/2$. The majority quantifier rejects in case of a draw. If $\tilde{Maj}$ denotes the weak majority quantifier which accepts in case of a draw we have the deMorgan-like relation $\tilde{Maj}_x\Phi = \neg Maj_x\neg\Phi$. The quantifier $Maj\, x$ has to be distinguished from the majority quantifier over pairs $Maj_2\, x$ which can be simulated by the iteration of majorities over single variables in the presence of the Bit predicate ([1]), but not without it ([5]).

The (unary) counting quantifier is denoted by $\exists^{=y}x$, thus $w_{y=j} \models \exists^{=y}\, x\ \phi$ is fulfilled iff there are exactly j positions $1 \leq i \leq |w|$ such that $w_{x=i,y=j} \models \phi$ where j is the numerical value of variable y. The counting quantifier can take values in the range $\{0, 1, \ldots, n\}$ which is one more than there are positions available in inputs of size n. If we take care of the case $y = 0$ by the formula $\forall x \neg\phi$ and only treat the case $y \geq 1$ it is possible to show:

Lemma 2.1 ([4]) *First order quantifiers, addition, and the counting quantifier are definable in* $Maj[<]$.

For the ease of handling we will use the counting quantifier without this restriction and note that more formally in all formulae using the counting quantifier the exception handling of the case $y = 0$ should be added.

Barrington et al. exhibited the power of the majority quantifier over pairs and showed that in addition to lemma 2.1:

Lemma 2.2 ([1]) *Multiplication and the* Bit*-predicate are definable in* $Maj_2[<]$.

Combining these abilities it is easily seen that the counting quantifier over tuples of variables is definable in $Maj_2[<]$, as well.

2.2. Circuits

The reader is assumed to be acquainted with language classes defined by uniform circuits as they are presented by Barrington et al.. In particular we will use the relations ([1] [Theorem 10.2 and Proposition 10.3]):

$$\mathrm{TC}^0 = \mathcal{L}(FO + Maj[<, Bit]) = \mathcal{L}(FO + Maj_2[<]).$$

Here TC^0 refers to its $FO[<, Bit]$-uniform (or equivalently $FO[<, +, *]$-uniform) version.

If the fan-ins of all gates occurring in a circuit are bounded by the input size we denote this by the subscript LIN. This leads to the circuit classes $\mathrm{AC}^0{}_{LIN}$, $\mathrm{ACC}^0{}_{LIN}$, and $\mathrm{TC}^0{}_{LIN}$.

3. Expressing Uniformity by $FO[<]$

In this section we exhibit a more restrictive formalism expressing uniformity which will allow for a reasonable notion of $FO[<]$-uniformity, i.e.: without using the Bit predicate, multiplication or anything else outside of Presburger Arithmetic.

In a circuit C_n of polynomial size n^k with n inputs names of gates will be given as k-dimensional vectors over $\{1, \cdots, n\}$. In order to make those vectors addressable by $FO[<]$-formulae and hence by finite automata we encode a number $1 \leq i \leq n$ by the word $a^i b^{n-i}$ and shuffle k-dimensional vectors of such words into one word of length n over the base alphabet $\Gamma_k := \{a, b\}^k$. To help addressing the elements of a word $v \in \Gamma_k$ we define refinements of the $C_a(x)$ predicates as follows: we say that $v_{x=j} \models C_a(i.x)$ iff the $i. - th$ component of the $j. - th$ symbol of v is an a. Otherwise $C_b(i.x)$ holds. That is $C_a(i.x)$ is an abbreviation for $\bigvee_{c\in\{a,b\}^{i-1}a\{a.b\}^{n-i}} C_c(x)$.

We define for $1 \leq k \leq n$ the mapping $PACK_{n,k}$: $\bigcup_{1\leq d\leq k}\{1, \cdots, n\} \longrightarrow \Gamma_k^n$ by letting $PACK_{n,k}(\vec{\alpha})$ be that unique word $v \in \Gamma_k^n$ such that $v_x = j \models C_a(i.x)$ iff $i \leq l(\vec{\alpha})$ and $j \leq \alpha_i$.

By convention the output gates in our circuits will always be numbered by the empty vector $\vec{0}$ of length 0. Hence this gate in a circuit with n^k gates will be encoded by the all-b-word $(b^k)^n \in \Gamma_k^n$.

In order to make the first-order description of the connectivity language easier we split the description of the circuit into sublanguages. An essential point in our construction is that in the list of all predecessors $\vec{\beta}$ of a gate $\vec{\alpha}$ these predecessors are additionally numbered. Since this number might be superlinear (although still polynomial) we use a vector $\vec{\gamma}$ to number the predecessors. Hence, the set describing the connectivity will consists in triplets $(\vec{\alpha}, \vec{\beta}, \vec{\gamma})$ such that the gate named $\vec{\beta}$ is the $j.-th$ predecessor of a gate named $\vec{\alpha}$ where j is the numerical value of the vector $\vec{\gamma}$. To encode this in words we use the function $PACK^3_{n,k} : \bigcup_{1\leq d\leq k}\{1, \cdots, n\}^3 \longrightarrow \Gamma^n_{3k}$ to be defined in following way: $PACK_{n,k}(\vec{\alpha}, \vec{\beta}, \vec{\gamma})$ is that word v that fulfills $v_{x=j} \models C_a(i.x)$ iff $(i \leq l(\vec{\alpha}) \wedge \alpha_i \leq j)$, $(k < i \leq k + l(\vec{\beta}) \wedge \beta_i \leq j)$, or $(2k < i \leq 2k + l(\vec{\gamma}) \wedge \gamma_i \leq j)$. That is, $PACK^3_{n,k}(\vec{\alpha}, \vec{\beta}, \vec{\gamma})$ is the "vertical" composition of $PACK_{n,k}(\vec{\alpha})$, $PACK_{n,k}(\vec{\beta})$, and $PACK_{n,k}(\vec{\gamma})$.

Observe that the uniqueness of the $\vec{\gamma}$th predecessor, i.e. the property that for each $\vec{\alpha}, \vec{\gamma}$ there is at most one $\vec{\beta}$ such that $(\vec{\alpha}, \vec{\beta}, \vec{\gamma})$ is a valid predecessor triple is expressible in $FO[<]$.

Finally, for each possible type $\tau \in GT$ of a gate there will be corresponding language $L_\tau(C)$. Here $GT := QT \cup \{\wedge, \vee, true, false\} \cup \Sigma$ where $QT := \{\forall, \forall_2, \cdots, \exists, \exists_2, \cdots, Maj, Maj_2, \cdots, \tilde{M}aj, \tilde{M}aj_2, \cdots, \exists^{r \ mod \ k}, \exists_2^{r \ mod \ k}, \cdots\}$ denotes the set of all possible quantifier type. We explicitly include the weak majority quantifier $\tilde{M}aj$ since we will deal with formulae in prenex normal form for which we need also the negated form of the majority quantifier.

Thus, the connectivity information of a circuit family $C = (C_n)_{n\geq 1}$ of size n^k consists in the following languages[1]:

$$L_{pred}(C) \subset \Gamma^*_{3k}, \text{ and } L_\tau(C) \subset \Gamma^*_k \text{ for } \tau \in GT.$$

We say that a circuit family $C = (C_n)_{n\in\mathbb{N}}$ is $FO[<, \mathcal{X}]$-uniform if the uniformity languages $L_{pred}(C)$ and $L_\tau(C)$ are in $\mathcal{L}(FO[<, \mathcal{X}])$ for each $\tau \in GT$. For a family $\mathcal{Q}$ of gate types and their corresponding quantifier types we let $FO[<, \mathcal{X}] - \mathcal{Q}C^0$ be the class of all languages accepted by $FO[<, \mathcal{X}]$-uniform circuits of polynomial size and constant depth using $\mathcal{Q}$-gates. If the fan-in of all gates is bounded by the input size we get the classes $FO[<, \mathcal{X}] - \mathcal{Q}C^0_{LIN}$.

[1]Formally, we have to provide a formalism to name the elements of the infintite set GT and a mechanism to determine in a circuit description the finite subset of elements which are actually used.

4. Main Theorem

In the following let $\mathcal{X}$ denote a set of negation closed numerical predicates and $\mathcal{Q}$ a set of deMorgan closed[2] quantifier types containing the first-order quantifiers. We need that each quantifier in $\mathcal{Q}$ is monoidal (e.g. has an "identity element" in the sense of [1][page 294]). In addition we allow $\mathcal{Q}$ to contain the majority quantifiers.

Theorem 4.1

$$FO[<, \mathcal{X}] - \mathcal{Q}C^0_{LIN} = \mathcal{L}(\mathcal{Q}[<, \mathcal{X}]) \text{ and}$$

$$FO[<, \mathcal{X}] - \mathcal{Q}C^0 = \mathcal{L}(\mathcal{Q}_{tuple}[<, \mathcal{X}])$$

The proof of theorem 4.1 is given in the following two sections.

5. From Logic to Circuits

The following construction will only work for sufficiently long input words, i.e.: $|w| \geq \max\{k, m, m'\}$ k, m, m' dependent of the formulae. By reformulating the construction using more variables and expressing the *and* and *or* of the formulae through *and*s and *or*s with fan in 2 this could be improved to the (usual) requirement $|w| \geq 2$. In order to make the construction more readable, we skip this point.

We first indicate the construction for quantors quantifying over single variables resulting in circuits with gates of linear fan-in. The extension to treat the general case is sketched at the end of this section.

5.1. Quantification over a single variable

For each $\mathcal{Q}[<, \mathcal{X}]$ formula $\Psi(x_1, \cdots, x_k)$ recognizing a language $L \in \Sigma^*$ we want to construct a $FO[<, \mathcal{X}]$-uniform $\mathcal{Q}C^0$ circuit family C_Ψ accepting L. Since $\mathcal{X}$ is negation closed and $\mathcal{Q}$ is deMorgan closed we can demand Ψ to be given in normal form $Q_1x_1 \ldots Q_kx_k \bigvee^m_{i=1} \bigwedge^{m'}_{j=1} A_{ij}$ where Q_i is in $\mathcal{Q}$ and A_{ij} is either a built-in predicate in $\mathcal{X} \cup \{<, \geq\}$ or of the type $C_a(x_i)$ for some $a \in \Sigma$. (We don't need negations of the $C_a(x)$ since they are euqivalent to the disjuntion of the $C_b(X)$ over all $b \neq a$.)

As usual, when going from logical formulae to circuits, we build the circuit recursivly top down corresponding to the construction of the formula by adding new levels each of the same type of gates for each quantifier and $\wedge$ or $\vee$ in the formula Ψ. Level 1 contains only the output gate which corresponds to Q_1, which as mentioned above is named by the empty vector $\vec{0}$ of lenght 0. Now we start recursivly:

[2]This means that with $Q\,x\,\phi$ also $\neg Q\,x\,\neg\phi$ is a quantifier in $\mathcal{Q}$.

For each gate g in level i, where $i \geq 1$, we add n gates in level $i+1$ corresponding to the Quantifier Q_{i+1} and connect them with g. We name the gates by the following rule: if the name of gate g is $\vec{\alpha}$ of length $i-1$, then the j-th successor gate g' of g has the name $\vec{\beta}$ where $l(\vec{\beta}) = l(\vec{\alpha}) + 1 = i$, $\beta_j = \alpha_j$ for $1 \leq j \leq i-1$, and $\beta_i = j$. This we do until level k. After that there are two levels corresponding to propositional gates of fan-in m resp. m'.

That means that the connection language $L_{pred}(C_\Psi)$ consists of four sets: the predecessors of $\mathcal{Q}$-gates:

$$\{PACK^3_{n,k+2}(\vec{\alpha}, \vec{\beta}, \vec{\gamma}) \quad | \quad \begin{array}{l} n \geq k+3, \\ l(\vec{\alpha}) + 1 = l(\vec{\beta}) < k, \\ \alpha_i = \beta_i \text{ for } 1 \leq i \leq l(\vec{\alpha}), \\ l(\vec{\gamma}) = 1, \beta_{l(\vec{\beta})} = \gamma_1\}, \end{array} \quad (1)$$

the predecessors of disjunctions at level $k+1$:

$$\{PACK^3_{n,k+2}(\vec{\alpha}, \vec{\beta}, \vec{\gamma}) \quad | \quad \begin{array}{l} n \geq k+2, \\ \alpha_i = \beta_i \text{ for } 1 \leq i \leq k, \\ l(\vec{\alpha}) + 1 = l(\vec{\beta}) = k+1, \\ l(\vec{\gamma}) = 1, \\ \beta_{k+1} = \gamma_1 \leq m\}, \end{array} \quad (2)$$

the predecessors of conjunctions at level $k+2$:

$$\{PACK^3_{n,k+2}(\vec{\alpha}, \vec{\beta}, \vec{\gamma}) \quad | \quad \begin{array}{l} n \geq k+2, \\ l(\vec{\alpha}) + 1 = l(\vec{\beta}) = k+2, \\ \alpha_i = \beta_i \text{ for } 1 \leq i \leq k+1, \\ l(\vec{\gamma}) = 1, \alpha_{k+1} \leq m, \\ \beta_{k_1} = \gamma_1 \leq m'\}, \end{array} \quad (3)$$

and the Inputs:

$$\{PACK^3_{n,k+2}(\vec{\alpha}, \vec{\beta}, \vec{\gamma}) \quad | \quad \begin{array}{l} n \geq k+2, l(\vec{\alpha}) = k+2, \\ l(\vec{\beta}) = l(\vec{\gamma}) = 1, \gamma_1 = 1, \\ (\alpha_{k+1}, \alpha_{k+2}) \in J_a, \\ \beta_1 = \rho(\alpha_{k+1}, \alpha_{k+2})\}, \end{array} \quad (4)$$

Here J_a is the set of all $(i.j), i \leq m, j \leq m'$ such that $A_{i,j}$ is a $C_a(x_t)$ predicate, where we write $t = \rho(i,j)$ as a function of i and j.

Since k, m, m', J_a and the mapping ρ are constant the resulting set L_{pred} is easily seen to be a member of $\mathcal{L}(FO[<, \mathcal{X}])$.

The type languages are easy to describe as well. For $\tau \in \mathcal{Q}$ let J_τ be the set $\{1 \leq i \leq k \mid Q_i \text{ is of type } \tau\}$. Then $L_\tau(C_\Psi) = \{PACK_{n,k+2}(\vec{\alpha}) \mid l(\vec{\alpha}) + 1 \in J_\tau\}$. Further on we have

$$L_\vee(C_\Psi) = \{PACK_{n,k+2}(\vec{\alpha}) \mid n \geq k+2, l(\vec{\alpha}) = k\},$$

$$L_\wedge(C_\Psi) = \{PACK_{n,k+2}(\vec{\alpha}) \quad | \quad \begin{array}{l} n \geq k+2, \\ l(\vec{\alpha}) = k+1, \\ \alpha_{k+1} \leq m\}, \text{ and} \end{array}$$

$$L_{I_a}(C_\Psi) = \{PACK_{n,k+2}(\vec{\alpha}) \quad | \quad \begin{array}{l} n \geq k+2, \\ l(\vec{\alpha}) = k+2, \\ (\alpha_{k+1}, \alpha_{k+2}) \in J_a\} \end{array}$$

All these languages are obviously definable in $FO[<, \mathcal{X}]$. To compute $L_{true}(C_\Psi)$ and $L_{false}(C_\Psi)$ let J_P be the set of all (i,j) with $i \leq m, j \leq m'$ such that $A_{i,j}$ is a numerical predicate in $\mathcal{X}$ using some of the variable in $\{x_1, \cdots, x_k\}$. Then for each vector $\alpha \in \{1, \cdots, n\}^k$ the relation $w_{\vec{x}=\vec{\alpha}} \models A_{i,j}$ is definable in $FO[<, \mathcal{X}]$ for arbitrary $w \in \Sigma^*$ such that $|w| \geq k+2$.. Hence we have

$$L_{true}(C_\Psi) = \{PACK_{n,k+2}(\vec{\alpha}) \mid \begin{array}{l} n \geq k+2, \\ l(\vec{\alpha}) = k+2, \\ (\alpha_{k+1}, \alpha_{k+2}) \in J_P, \\ w_{\vec{x}=\vec{\alpha}} \models A_{i,j}\} \end{array} \quad (5)$$

and

$$L_{false}(C_\Psi) = \{PACK_{n,k+2}(\vec{\alpha}) \mid \begin{array}{l} n \geq k+2, \\ l(\vec{\alpha}) = k+2, \\ (\alpha_{k+1}, \alpha_{k+2}) \in J_P, \\ w_{\vec{x}=\vec{\alpha}} \not\models A_{i,j}\} \end{array} \quad (6)$$

Also these two sets are obviously members of $\mathcal{L}(FO[<, \mathcal{X}])$.

Observations The depth of the circuit C_n is constant and only depending on the number of quantifiers in the formula Ψ. The type of a gate g depends only on its level in the circuit except for the last level. The relations in $\mathcal{X}$ used in the formula are carried out by the uniformity formula. It could be mentioned that we don't need the closure under negation of the numerical predicates: the above construction shows how to evaluate both positive and negated predicates $A_{i,j}$ and how connect the outcoming result to a true or a false input.

5.2. Quantification over tuples

Here we shortly indicate the modifications in our construction in order to cope with quantification over tuples of variables. When dealing with such quantifiers we use the following extension in our uniformity languages: Instead of spending one new entry in the naming vector of a gate, we add d_i entries in the naming vector if we have to handle a quantifier Q_i over a d_i tuple of variables. Each new entry corresponds to a variable in the tuple. Thus the simulating gate has n^d predecessors in the circuit for inputs of length n. In this way we have to add the number d_i where we added a 1 in the type and in the predecessor language.

6. From Circuits to Logic

In this section we show how to simulate an $FO[<,\mathcal{X}]$-uniform $\mathcal{Q}C^0$ circuit by a $\mathcal{Q}[<,\mathcal{X}]$ formula. We are given languages $L_{pred}(C)$ and $L_\tau(C)$ for $\tau \in \mathcal{Q} \cup \Sigma \cup \{\wedge, \vee, true, false\}$ accepted by $FO[<,\mathcal{X}]$ formulae Ψ_{pred} and Ψ_τ.

As a first step we translate the uniformity information given by some $FO[<,\mathcal{X}]$ formula Ψ over words in $\Gamma^n_{(3)k}$ into a $FO[<,\mathcal{X}]$ formula $T(\Psi)$ over $\mathcal{V}$ structures. This translation converts the representations of numbers from unary words in a^*b^* into position values of first order variables. In a second step we will construct a formula simulating the circuit described by $T(\Psi)$.

6.1. Translating connectivity languages into $\Sigma^* \otimes \mathcal{V}$

We now convert the uniformity relations given by words in Γ^*_{3k} resp. Γ^*_k into formulae $\Phi_{pred}(\vec{x}, x', \vec{y}, y', \vec{z}, z') = T(\Psi_{pred})$ and $\Phi_\tau(\vec{x}, x') = T(\Psi_\tau)$. After the translation the connectivity is no longer stored in an input word in Γ^n_k but instead in three k-dimensional vectors of variables attached to an input word in Σ^n for the predecessor information. The type information can be stored in one vector of length k. More formally, we represent a vector $\vec{\alpha} \in \{1, \cdots, n\}^d, 0 \leq d \leq k$ by the following variable values: $x_1 = \alpha_1, \cdots, x_{l(\vec{\alpha})} = \alpha_{l(\vec{\alpha})}$, and the remaining $x_i, i > l(\vec{\alpha})$ arbitrary. This makes it necessary to store the value of $l(\vec{\alpha})$ in some variable x'. Since a variable can only store values between 1 and n, we represent this by attaching variable x' at position $l(\vec{\alpha})+1$. (Thus we increase the lower bound assumed to hold for the input words by one.) In the following translation we use the fact that if a word v is accepted by $\Psi_{something}$, then v fulfilles in every $i.-th$ component row that there is an $1 \leq j \leq |v|$ such that $C_a(i.y)$ for all $y \leq j$ and $C_b(i.y)$ for all $y > j$.

We now describe the translation $T(\cdot)$. Variables inside of $\Psi_{something}$ are denoted by capital letters, while $x, y, z \cdots$ stand for free variables, i.e.: parameters, of $T(\Psi_{something})$. The translation follows inductively the termstructure of Ψ:

1. $T(\Psi_1 \wedge \Psi_2) := T(\Psi_1) \wedge T(\Psi_2)$,
2. $T(\neg\Psi) := \neg T(\Psi)$,
3. $T(\exists X \Phi_1) := \exists X T(\Phi_1)$,
4. $T(X < Y) := X < Y$,
5. $T(C_a(i.X)) := X \leq x.i \wedge X < x'$ and
6. $T(C_b(i.X)) := X > x.i \vee X \geq x'$.

The essential translations of the C_a-predicates are due to the following observation: a vector $\vec{\alpha}$ of length $d \leq k$ of numbers between 1 and n is coded as a word $v \in (\{a,b\}^k)^n$ such the $i.-th$ row contains $a^{\alpha_i}b^{n-\alpha_i}$ for $1 \leq i \leq d$ while the remaining $k-d$ rows contain b^n.

6.2. The construction for gates of linear fan-in

We are now ready to indicate our construction which follows the usual approach of defining the acceptance of a gate at depth i in terms of gates describing the acceptance at depth $i-1$ ([8]). Since $\exists\, x_1, \cdots, x_k$ is equivalent to $\exists\, x_1 \cdots \exists x_k$ (and the same for universal quantifiers) we will use in the following the first order quantification over tuples also in this linear case to improve readability.

Let d be the depth of the $FO[<,\mathcal{X}]$ uniform circuit familiy $C = (C_n)_{n\in\mathbb{N}}$. Further on let $\Phi_{pred}(\vec{x}, x', \vec{y}, y', \vec{z}, z')$ and $\Phi_\tau(\vec{x}, x')$ for $\tau \in \mathcal{Q} \cup \Sigma \cup \{\wedge, \vee,\ true, false\}$ be the $FO[<,\mathcal{X}]$ obtained after the translation described in the previous subsection.

Again we start with the case of linear fan-in. In this case we can assume to have Φ_{pred} the free variables $\vec{x}, x', \vec{y}, y', z$ since the single variable z is now enough to count through all predecessors of the gate named by $\vec{x}, x'$.

For $i = 0, 1, \cdots, d$ we define now a formula $ACC_i(\vec{x}, x')$ which is true if the values of $\vec{x}, x'$ refer to an accepting gate at depth i.

For $i = 0$ this means to ask wether $\vec{x}, x'$ refers to an constant input $true$ or a positive input variable:

$$ACC_0(\vec{x}, x') = \Phi_{true}(\vec{x}, x') \vee$$

$$\left(\bigvee_{a\in\Sigma} \Phi_a(\vec{x}, x') \wedge \exists_y\, \Phi_{pred}(\vec{x}, x', y, 1) \wedge C_a(y) \right)$$

Here $\exists_y\, \Phi_{pred}(\vec{x}, x', y, 1)$ is an abbreviation to existentially quantify over all $\vec{y}, y', z$ such that $\Phi_{pred}(\vec{x}, x', \vec{y}, y', z)$ holds and in addition $z = 1, y' = 1$, and we then ask for $C_a(y_1)$ instead of $C_a(y)$.

The main part of the construction is now the the recursive definition of $ACC_i(\vec{x}, x')$. This is essentially a disjunction over all $\tau \in \mathcal{Q} \cup \Sigma \cup \{\wedge, \vee\}$ and of $ACC_0(\vec{x}, x')$. The later covers the case that the underlying circuit is not balanced. The cases $\wedge$ and $\vee$ can be treated like first order quantifiers. For the quantifier types we have three cases:

τ is a quantifier with $false$ as identity: this covers for instance existential and modulo counting quantifiers. In this case $ACC_i(\vec{x}, x')$ contains disjuntively the clause:

$$\Phi_\tau(\vec{x}, x') \wedge Q_\tau\, z\, \exists\, \vec{y}, y' :$$
$$(\Phi_{pred}(\vec{x}, x', \vec{y}, y', z) \wedge ACC_{i-1}(\vec{y}, y'))\,.$$

τ is a quantifier with $true$ as identity: this covers for instance the universal quantifier. In this case $ACC_i(\vec{x}, x')$ contains disjuntively the clause:

$$\Phi_\tau(\vec{x}, x') \wedge Q_\tau\, z \,\forall \vec{y}, y' :$$

$$(\Phi_{pred}(\vec{x}, x', \vec{y}, y', z) \Rightarrow ACC_{i-1}(\vec{y}, y'))\,.$$

τ is the majority quantifier: here we make use of the ability to define the counting quantifier $\exists^{=y} x$ in $Maj[<]$. In this case $ACC_i(\vec{x}, x')$ contains disjunctively the clause:

$$\begin{array}{c}\exists\, y_+ \,\exists\, y_- \,:(y_+ > y_-) \wedge \\ (\exists^{=y_+} z \,\exists\, \vec{y}, y'\, (\Phi_{pred}(\vec{x}, x', \vec{y}, y', z) \wedge ACC_{i-1}(\vec{y}, y')) \wedge \\ (\exists^{=y_-} z \,\exists\, \vec{y}, y'\, (\Phi_{pred}(\vec{x}, x', \vec{y}, y', z) \wedge \neg ACC_{i-1}(\vec{y}, y')).\end{array}$$

That is we express by y_+ and y_- the number of positive and negative predecessors and require that $y_+ > y_-$. The $\tilde{M}aj$ quantifier would be expressed by $y_+ \geq y_-$. Observe that the case "$y = 0$" needs some care when using the counting quantifier.

Finally, $w \models \exists_{\vec{x},x'} ACC_d(\vec{x}, x') \wedge x' = 1$ expresses $w \in L$ since the vector $\vec{x}, x'$ with $x' = 1$ denotes the empty vector $\vec{0}$ which is the name of the output gate.

□

6.3. Handling gates of superlinear fan-in

Working with superlinear fan-in makes necessary to use connectivity formulae Φ_{pred} with free variables $\vec{x}, x', \vec{y}, y', \vec{z}, z'$. The quantifier Q_τ simulating a gate of type τ now quantifies over a tuple $\vec{z}, z'$. In the case of majority quantifiers we now count sums of fulfilling and of rejecting predecessors in variable vectors $\vec{y_+}$ and $\vec{y_-}$. The rest of the construction goes through. It could be remarked that we could handle a fan-in larger than the input size as long as it is linear since we can add $O(1)$ many number bounded by n in $Maj[<]$.

7. Consequences

Concerning first order and modulo counting quantifiers we get that first order uniformity is very weak since it only recognize regular languages, namely the starfree regular sets and the solvable regular languages:

Corollary 7.1

$$FO[<] - AC^0 = \mathcal{L}(FO[<]) \textit{ and}$$

$$FO[<] - ACC^0 = \mathcal{L}(FO + Mod[<]).$$

Here we used the fact that first order and modular counting quantifiers ranging over tuples of variables can be simulated by a vector of quantifiers ranging over a single variable. If we allow addition in the uniformity description we come to circuit characterizations of classes of the form $\mathcal{L}(FO+\mathcal{Q}[<,+])$ answering a question of Roy and Straubing ([6]):

Corollary 7.2

$$FO[<,+] - AC^0 = \mathcal{L}(FO[<,+]) \textit{ and}$$

$$FO[<,+] - ACC^0 = \mathcal{L}(FO + Mod[<,+]).$$

When working with majority gates $FO[<]$-uniformity does not restrict the resulting circut class:

Corollary 7.3

$$FO[<] - TC^0{}_{LIN} = \mathcal{L}(Maj[<]) = FO[<,+] - TC^0{}_{LIN}$$

and

$$\begin{array}{rcl} FO[<] - TC^0 & = & \mathcal{L}(Maj_2[<]) \\ & = & FO[<, Bit] - TC^0{}_{LIN} \\ & = & FO[<, Bit] - TC^0. \end{array}$$

These relations which give the first circuit characterization of the class $\mathcal{L}(Maj[<])$ are obtained by combining results in [1] and [4]. $\mathcal{L}(Maj[<]$ is a proper subclass of (Bit-uniform) TC^0. Both classes seem to be unrelated to regular languages and hence lack a characterization in terms of finite monoids or groups. Nevertheless they have been described by certain varieties of soluble infinite groups ([3]).

8. Discussion

Our results extending the results of Barrington et al. ([1]) leave open when it is possible to remove the condition that $\mathcal{Q}$ contains first-order quantifiers. It should be noted that the use of first order quantifiers is quite restrictied. for instance the occurring existential quantifiers are unique, i.e.: could also be replaced by $Mod^{1(k)}$ for $k \geq 2$. It might be possible to get our results also for CC^0 circuits.

Another question is whether we need the condition that the quantifiers in $\mathcal{Q}$ need an identity element. The treatment of the majority quantifier indicates that a more general statement is possible.

Finally we mention that $FO[<]$-uniformity seems to be sufficient for higher complexity classes. It is very easy to see that Ladners construction of a polynomial size circuit simulation a polynomial time machine is $FO[<]$-uniform. It is possible to show this for all NC^k?

9. Acknowledgments

We thank Andreas Krebs and Stephanie Reifferscheid for very valuable discussions on this topic.

References

[1] D.A. Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *J. Comp. System Sci.*, 41:274–306, 1990.

[2] D. Barrington, N. Immerman, C. Lautemann, N. Schweickardt, and D. Therien. The Crane Beach Conjecture. In *Proc. of the 16th IEEE Symposium On Logic in Computer Science*, pages 187–196, 2001.

[3] A. Krebs, K.-J. Lange, and S. Reifferscheid. Characterizing TC^0 in terms of infinite groups. *proc. of the 22nd STACS 2005*, LNCS 3404, 496-507, 2005.

[4] K.-J. Lange. Some results on majority quantifiers over words. In *Proc. of the 19th IEEE Conference on Computational Complexity*, pages 123–129, 2004.

[5] C. Lautemann, P. McKenzie, T. Schwentick, and H. Vollmer. The descriptive complexity approach to LOGCFL. *J. Comp. System Sci.*, 62:629–652, 2001.

[6] A. Roy and H. Straubing. Definability of Languages by Generalized First-Order Formulas over (N,+). In *Proc. of the 23rd STACS 2006*, to appear.

[7] N. Schweikardt. On the Expressive Power of First-Order Logic with Built-In Predicates. Dissertation, Universität Mainz, 2001.

[8] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.