AND and/or OR:
Uniform Polynomial-Size Circuits

Niall Murphy[*]
Facultad de Informática
Universidad Politécnica de Madrid
CEI-Moncloa UCM-UPM
Madrid, Spain
niall.murphy@upm.es

Damien Woods[†]
Computer Science
Center for Mathematics of Information
California Institute of Technology
Pasadena, CA 91125, USA
woods@caltech.edu

We investigate the complexity of uniform OR circuits and AND circuits of polynomial-size and depth. As their name suggests, OR circuits have OR gates as their computation gates, as well as the usual input, output and constant $0/1$ gates. As is the norm for Boolean circuits, our circuits have multiple sink gates, which implies that an OR circuit computes an OR function on some subset of its input variables. Determining that subset amounts to solving a number of reachability questions on a polynomial-size directed graph (which input gates are connected to the output gate?), taken from a very sparse set of graphs. However, it is not obvious whether or not this (restricted) reachability problem can be solved, by say, uniform $\mathbf{AC}^0$ circuits (constant depth, polynomial-size, AND, OR, NOT gates). This is one reason why characterizing the power of these simple-looking circuits in terms of uniform classes turns out to be intriguing. Another is that the model itself seems particularly natural and worthy of study.

Our goal is the systematic characterization of uniform polynomial-size OR circuits, and AND circuits, in terms of known uniform machine-based complexity classes. In particular, we consider the languages reducible to such uniform families of OR circuits, and AND circuits, under a variety of reduction types. We give upper and lower bounds on the computational power of these language classes. We find that these complexity classes are closely related to **tallyNL**, the set of unary languages within **NL**, and to sets reducible to **tallyNL**. Specifically, for a variety of types of reductions (many-one, conjunctive truth table, disjunctive truth table, truth table, Turing) we give characterizations of languages reducible to OR circuit classes in terms of languages reducible to **tallyNL** classes. Then, some of these OR classes are shown to coincide, and some are proven to be distinct. We give analogous results for AND circuits. Finally, for many of our OR circuit classes, and analogous AND circuit classes, we prove whether or not the two classes coincide, although we leave one such inclusion open.

Keywords: Computational complexity; uniform Boolean circuits; AND circuits; OR circuits; NL; AC^0

1 Introduction

We look at the complexity of simple problems: those defined by uniform OR circuits and AND circuits of polynomial-size and depth. As their name suggests, OR circuits have only OR gates as their computation gates, as well as the usual input gates, constant $(0/1)$ gates, and an output gate. As is the norm for Boolean circuits, our circuits have multiple sink gates, which implies that an OR circuit computes an OR function on some *subset* of its input variables. Determining that subset amounts to solving a number of reachability

[*] N. Murphy was supported by the Campus de Excelencia Internacional Moncloa UCM-UPM PICATA Program and the Irish Research Council for Science, Engineering and Technology's EMBARK Initiative.

[†] D. Woods was supported by National Science Foundation (United States) grants CCF-1219274, CCF-1162589, and 0832824 (The Molecular Programming Project), and Junta de Andalucía (Spain) grant TIC-581.

T. Neary and M. Cook (Eds.):
Machines, Computations and Universality (MCU 2013)
EPTCS 128, 2013, pp. 150–166, doi:10.4204/EPTCS.128.20

questions on a polynomial-size directed graph (i.e. which input gates are connected to the output gate?), taken from a very sparse set of graphs. It is not obvious whether or not these reachability questions can be solved, in say, uniform $\mathbf{AC}^0$. Yet these problems are trivially in non-uniform-$\mathbf{AC}^0$. This is one reason why characterizing the power of these simple-looking circuits in terms of uniform classes turns out to be intriguing. Another is that the model itself seems particularly natural and worthy of study.

Our goal is the systematic characterization of polynomial-size uniform OR circuits, and AND circuits, in terms of known uniform machine-based complexity classes. In particular, we consider the languages reducible to such circuit classes, under a variety of reductions. We give upper and lower bounds on the computational power of these classes. We find that they are closely related to **tallyNL**, the set of unary languages within **NL**, and to sets reducible to **tallyNL**. Specifically, for a variety of types of reductions ($\mathbf{AC}^0$ many-one, conjunctive truth-table, disjunctive truth-table, truth-table, Turing) we give characterizations of languages reducible to OR circuit classes in terms of languages reducible to **tallyNL** classes. Two of the OR classes are shown to coincide, and others are proven to be distinct. We give analogous results for AND circuits. Finally, for many of our OR circuit classes, and analogous AND circuit classes, we prove whether or not the two classes coincide, although we leave one such inclusion open. These results are summarized in Figure 1.

We also look at a related notion called semi-uniformity where the uniformity function for a circuit family gets access to the input word (and not merely its length). For sufficiently weak uniformity functions, this notion is analogous to a reduction to a circuit value problem, and there is a very simple proof that uniformity is a strictly weaker notion than semi-uniformity. Although not covered in this paper, these ideas can be used in an analogous proof that semi-uniformity is strictly stronger than uniformity in a model called membrane systems [21], answering an open question in that field [24], but which is much simpler to state and prove here in the setting of Boolean circuits.

The paper is structured as follows. We begin with basic definitions and results in Sections 2 and 3. Section 4 contains our main results on characterizing the power of polynomial-size uniform OR circuits. We give lower and upper bounds, or characterizations, of the complexity classes defined by OR circuits under various kinds of reductions. Specifically, we show that polynomial-size uniform OR circuits contain **tallyNL** and are properly contained in $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$, i.e. the class of languages $\mathbf{AC}^0$ disjunctive truth-table reducible to **tallyNL**. We go on to show that the following three classes coincide: languages many-one reducible, and disjunctive truth-table reducible, to uniform OR circuits, and the class $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$. These results are shown on the left hand side of Figure 1. Analogous results for AND circuits are shown on the right of the same figure and are presented in Section 5. Results on semi-uniformity are given in Section 6.

Since we are working with extremely weak classes it is important to use appropriate reductions between problems and appropriate uniformity requirements on circuits. We use **DLOGTIME**-uniform $\mathbf{FAC}^0$ [8] for reductions [5, 4, 3] and circuit uniformity [1, 2], which is powerful enough to implement a variety of encoding/decoding functions, but yet suitable for use with our (weak) classes.

One way to think about uniform OR circuits is that they compute the OR function on a subset of n input variables, that subset being defined via a number of directed graph connectivity questions that are implicitly encoded by the uniformity condition. The seemingly simpler OR function on all n variables is trivially in depth 1 uniform $\mathbf{AC}^0$, yet there are unanswered questions there too. For example, it is not known if the OR function on all n variables (or indeed the AND function) is in $\mathbf{CC}^0[q]$, the class of problems accepted by constant depth polynomial-size circuits that use MOD_q gates [15].

Figure 1 suggests a number of open questions. Are there other classes that can be used to give a tighter characterization of the class of problems solved by polynomial-size uniform OR circuits ($\mathbf{FAC}^0$-uniform-**OR**)? Also, $\mathbf{FAC}^0$-uniform-**AND**? Is there a language in $\mathbf{FAC}^0$-uniform-**OR** that is

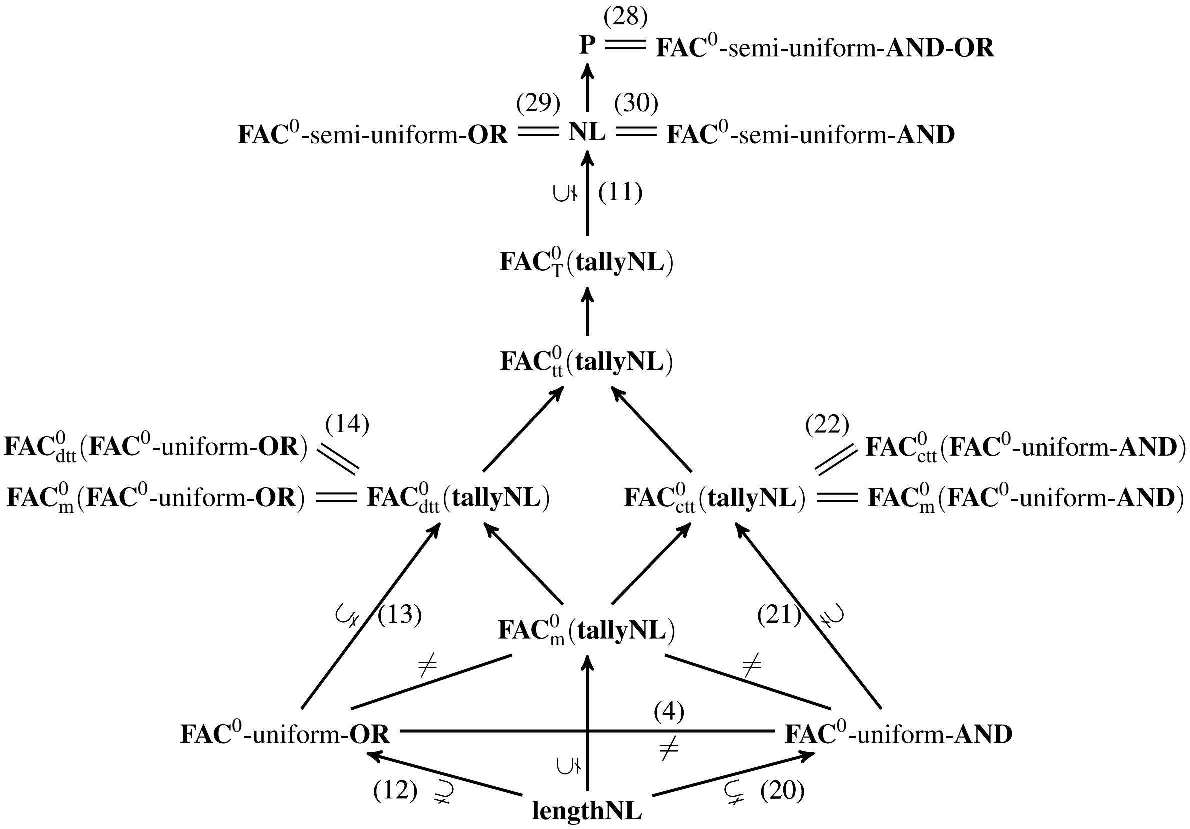
(28)
P = FAC⁰-semi-uniform-AND-OR
(29)
(30)
FAC⁰-semi-uniform-OR = NL = FAC⁰-semi-uniform-AND
(11)
FAC⁰_T(tallyNL)
FAC⁰_tt(tallyNL)
(14)
FAC⁰_dtt(FAC⁰-uniform-OR)
FAC⁰_m(FAC⁰-uniform-OR) = FAC⁰_dtt(tallyNL)
(22)
FAC⁰_ctt(FAC⁰-uniform-AND)
FAC⁰_ctt(tallyNL) = FAC⁰_m(FAC⁰-uniform-AND)
(13)
FAC⁰_m(tallyNL)
(21)
≠
≠
FAC⁰-uniform-OR
(4)
≠
FAC⁰-uniform-AND
(12)
lengthNL
(20)

Figure 1: Summary of results. The left side shows relationships between uniform polynomial-size OR circuit languages, **tallyNL** and sets reducible to these classes. The right side shows analogous relationships for AND circuit classes. $\mathbf{FAC}^0_{\mathrm{r}}(\mathbf{C})$ denotes $\mathbf{AC}^0$ computable reductions of type r to a class **C**. Numerical labels refer to theorem statements, and symbols are used to show inclusion type, with an unlabelled arrow denoting $\subseteq$. To save space, Theorems 18 and 19 are not shown.

not in $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{tallyNL})$? It would be interesting to look at the power of uniform polynomial-size circuits consisting of other, apparently weak, gates, such as XOR. Ko [19] shows that the classes of languages polynomial time disjunctive and conjunctive reducible to **tally** are distinct. If it is possible to apply Ko's technique, or something like it, to our much more restrictive setting (i.e. $\mathbf{AC}^0$ disjunctive/conjunctive reducible to **tallyNL**), this would show that the four classes $\mathbf{AC}^0$ many-one, disjunctive truth-table, conjunctive truth-table, and truth-table reducible to **tallyNL** are in fact distinct, which would in turn clarify the relationship between the OR and AND classes that we consider.

2 Definitions

We now give some basic definitions based on those in the literature [4, 6, 14]. For more details on Boolean circuits see [26].

For a function $f\colon \{0,1\}^* \to \{0,1\}^*$ and integers $m,n \geq 1$ let $f_n\colon \{0,1\}^n \to \{0,1\}^m$ be the restriction of f to domain and range consisting of strings of length n and m respectively (we consider only functions f where for each n there is an m where all length-n strings in f's domain are mapped to length-m strings, thus $f = \bigcup_n f_n$).

A *circuit* on n variables $w_0, \ldots, w_{n-1}$ is a directed acyclic multi-graph (there may be multiple edges, or wires, between vertices—useful for oracle gates). The vertices of the circuit are generally referred to as gates. The in-degree (out-degree) of a gate is called its fan-in (fan-out). Each source vertex (fan-in 0) is labelled either by one of the input variables $w_0, \ldots, w_{n-1}$ or by a constant "0" or "1" (false or true). Each non-source vertex is labelled by a function name, such as AND, OR, NOT, or ORACLE.

In this paper, we use ORACLE gates. For a given circuit C, it will be the case that all ORACLE gates in C compute exactly the same Boolean function $g\colon \{0,1\}^n \to \{0,1\}$ for $n > 1$, although of course their inputs may be different. We are using the following conventions for circuits with tally oracles. The tally alphabet is $\{1\}$. A *tally oracle gate* with n ordered input wires, takes a string of the form $0^{n-i}1^i$, $0 \leq i \leq n$ (encoding the unary word 1^i) as input, and outputs a single bit.

Gates with fan-out of 0 (called sinks) may or may not be designated as *output* gates.

Given an input $w \in \{0,1\}^n$, one can inductively assign a Boolean value to each vertex of a circuit as follows: each source (*input*) vertex labelled by an input variable gets the value of that variable, each source (*constant*) vertex labelled by a constant gets the value of that constant, and each other vertex gets the value of the function that labels it applied to the values of its children. Incoming and outgoing edges to a vertex are assumed to be ordered (for oracle gates).

The *depth* of a circuit is the length of the longest path from an input vertex to an output vertex. The *size* of a circuit is the number of wires it contains [4]. A circuit computes a function on a fixed number of Boolean variables. We consider functions of an arbitrary number of variables by defining (possibly infinite) families of circuits. We say a family of circuits $\mathcal{C} = \{C_n \mid n \in \mathbb{N}\}$ computes a function $f\colon \{0,1\}^* \to \{0,1\}^*$ if for all $n \in \mathbb{N}$, and for all $w \in \{0,1\}^n$ circuit C_n outputs the string $f(w)$ (we consider only functions f where for each n there is an m where all length-n strings in f's domain are mapped to length-m strings). We say a family of circuits $\mathcal{C}$ decides a language $L \subseteq \{0,1\}^*$ if for each $w \in \{0,1\}^n$ circuit $C_n \in \mathcal{C}$ on input w outputs 1 if $w \in L$ and 0 if $w \notin L$.

In a *non-uniform* family of circuits there is no required similarity between family members. In order to specify such a requirement we use a *uniformity function* that algorithmically specifies the similarity between members of a circuit family. Roughly speaking, a *uniform circuit family* $\mathcal{C}$ is an infinite sequence of circuits with an associated function $f : \{1\}^* \to \mathcal{C}$ that generates members of the family and is computable within some resource bound. More precisely:

Definition 1 (**C**-Uniform circuit family)**.** *Let* **C** *be a set of functions. A circuit family $\mathcal{C}$ is* **C***-uniform, if there is function $f \in$* **C***, $f : \{1\}^* \to \mathcal{C}$, where $f(1^n) = C_n$ for all $n \in \mathbb{N}$, and $C_n \in \mathcal{C}$ is a description of a circuit with n input gates (we use C_n to denote either a circuit or its encoding as a binary string).*

When dealing with uniformity for small complexity classes one of the preferred uniformity conditions is **DLOGTIME**-uniformity [8]. This definition uses an ordering on wires that leave and enter a given gate.

Definition 2 ([4])**.** *A circuit family $\mathcal{C}$ is* **DLOGTIME***-uniform if there is a procedure that on input (n,i,r,j,s,t), where $n,i,r,j,s \in \mathbb{N}$ are encoded in binary and t is a gate type (e.g.,* AND*,* OR*,* NOT*, input, 0, 1) encoded in binary, runs in time* linear *in its input size and accepts if and only if the gate of C_n having label i is of type t and its r-th child is the s-th output of the gate having label j. In the case where gate i is an input gate, the procedure accepts if gate i takes the value of the s-th input bit. Furthermore, the procedure accepts inputs of the form $(n,i,j,s,output)$ if and only if the s-th output wire of gate i is the j-th output gate of the circuit C_n. We also require that the procedure accepts the input (n,i,d) if and only if d is equal to the fan-in of the gate of C_n having label i.*

$\mathbf{AC}^0$ is the set of languages decidable by constant-depth polynomial-size (in input length n) **DLOGTIME**-uniform circuits built using unbounded fan-in AND and OR gates, and NOT gates with fan-in 1. $\mathbf{FAC}^0$ is the class of functions computable by polynomial-size constant-depth **DLOGTIME**-uniform circuits built using unbounded fan-in AND and OR gates, and NOT gates with fan-in 1.

An OR *circuit* is a circuit that uses only disjunctive logic, that is, a circuit that has only OR, constant, and input gates. One of the OR gates is denoted as the output gate. Similarly an AND *circuit* is a circuit that uses only conjunctive logic, that is, a circuit that has only AND, constant, and input gates. One of the AND gates is denoted as the output gate. Note that OR and AND circuits may have multiple non-output sinks. Let non-uniform-**OR** (non-uniform-**AND**) be the set of decision problems that solved by non-uniform families of OR (AND) circuits.

In this paper, we are concerned with $\mathbf{FAC}^0$-uniform-**OR**: the class of languages solved by uniform polynomial size OR circuits, formally defined as follows.

Definition 3. *Let* $\mathbf{FAC}^0$*-uniform-***OR** *be the set of decision problems over the alphabet* $\{0,1\}$ *that are solved by* $\mathbf{FAC}^0$ *uniform families of* OR *circuits.*

The class $\mathbf{FAC}^0$-uniform-**AND** is defined analogously, but using AND instead of OR circuits.

Lemma 4. $\mathbf{FAC}^0$*-uniform-***OR** $\neq$ $\mathbf{FAC}^0$*-uniform-***AND**.

Proof. An OR circuit computes an OR function on some subset of its inputs; in general there is no AND circuit that computes the same function, and vice-versa. □

NL is the class of languages accepted by non-deterministic logarithmic-space Turing machines. Such machines have a read-only input tape, a write-only output tape and a read-write work tape whose length is a logarithmic function of input length. The class of functions $f:\{0,1\}^*\to\{0,1\}^*$ computed by non-deterministic logarithmic-space Turing machines (with an additional write-only output tape) is denoted **FNL**. Let **tally** be the set of all languages over the one-letter alphabet $\{1\}$. Let **length** be the set of all languages $L\subseteq\{0,1\}^*$ such that if $w\in L$ then all words in $\{0,1\}^{|w|}$ are in L.

We define $\mathbf{tallyNL}=\mathbf{tally}\cap\mathbf{NL}$, i.e. the class of all tally languages and length encoded languages in **NL**. Let $\mathbf{tallycoNL}=\mathbf{tally}\cap\mathbf{coNL}$. The following lemma follows from $\mathbf{NL}=\mathbf{coNL}$, (i.e. let $L\in\mathbf{tallyNL}\subsetneq\mathbf{NL}=\mathbf{coNL}$, then $L\in\mathbf{coNL}$ implies $L\in\mathbf{tallycoNL}$; a similar argument holds for the converse):

Lemma 5. $\mathbf{tallyNL}=\mathbf{tallycoNL}$

Let $\mathbf{lengthNL}=\mathbf{length}\cap\mathbf{NL}$ and $\mathbf{lengthcoNL}=\mathbf{length}\cap\mathbf{coNL}$. Also $\mathbf{lengthNL}=\mathbf{lengthcoNL}$. We make use of functions from the class $\mathbf{tallyFAC}^0=\mathbf{tally}\cap\mathbf{FAC}^0$ which is contained in **tallyNL**.

Each language $L\subseteq\{0,1\}^*$ has an associated total *characteristic function* $\chi_L\colon\{0,1\}^*\to\{0,1\}$ defined by $\chi_L(w)=1$ if and only if $w\in L$.

Parity $\subseteq\{0,1\}^*$ is the set of binary strings that contain an odd number of 1s.

2.1 Reductions

For concreteness, we explicitly define some standard types of reductions. Let $A,B\subseteq\{0,1\}^*$.

Definition 6 (Many-one reducible)**.** *Set A is many-one reducible to set B, written* $A\leq_m^{\mathbf{C}}B$*, if there is a function f that is* **C***-computable with the property that for all w,* $w\in A$*, if and only if* $f(w)\in B$*.*

The following definition of truth table reductions comes from [9, 10], for a more formal definition see [20].

Definition 7 (Truth-table reduction)**.** *Set A is* $\mathbf{C}$ *truth-table reducible to B, written $A \leq^{\mathbf{C}}_{tt} B$, if there exists* $\mathbf{C}$*-computable functions τ and σ such that for all $w \in \{0,1\}^*$, $\tau(w)$ is a list of $\ell \in \mathbb{N}$ strings $a_1, \ldots, a_\ell$, also $\sigma(w)$ is a truth table (Boolean function) with ℓ variables, and $w \in A$ if and only if $\sigma(\chi_B(a_1), \ldots, \chi_B(a_\ell))$ evaluates to true, where χ_B is the characteristic function of B.*

A *disjunctive* truth table reduction (dtt) is one where at least one string generated by $\tau(w)$ is in B. Or equivalently, where $\sigma(w) = \bigvee_{1 \leq i \leq \ell} \chi_B(a_i)$. A *conjunctive* truth table reduction (ctt) is one where all the strings generated by $\tau(w)$ are in B. Or equivalently, where $\sigma(w) = \bigwedge_{1 \leq i \leq \ell} \chi_B(a_i)$.

Definition 8 (Turing reducible)**.** *Set A is* $\mathbf{C}$ *Turing reducible to B, written $A \leq^{\mathbf{C}}_{T} B$, if there is a* $\mathbf{C}$*-computable oracle circuit (or Turing machine) M such that $w \in A$ iff M accepts w with B as its oracle.*

The following implications follow directly from these definitions, for more details see [20].

$$\begin{tikzcd}
& A \leq^{\mathbf{C}}_{dtt} B \arrow[dr, Rightarrow] & & \\
A \leq^{\mathbf{C}}_{m} B \arrow[ur, Rightarrow] \arrow[dr, Rightarrow] & & A \leq^{\mathbf{C}}_{tt} B \arrow[r, Rightarrow] & A \leq^{\mathbf{C}}_{T} B \\
& A \leq^{\mathbf{C}}_{ctt} B \arrow[ur, Rightarrow] & &
\end{tikzcd}$$

Let $\mathbf{FAC}^0_{\mathrm{r}}(\mathbf{C})$ be the set of all languages that are $\mathbf{FAC}^0$ reducible to languages in $\mathbf{C}$ via some type of reduction $\mathrm{r} \in \{\mathrm{m, dtt, ctt, tt, T}\}$.

2.2 Some useful $\mathbf{FAC}^0$ functions

Pairing function We require a pairing function that is injective and extremely easy ($\mathbf{FAC}^0$) to compute. We use the pairing function that interleaves the bits of two binary string arguments a and b. For example, the binary strings $a = a_2a_1a_0$ and $b = b_2b_1b_0$ are paired as the interleaved string $\langle a, b\rangle = b_2a_2b_1a_1b_0a_0$. The circuits for interleaving and de-interleaving have only a single input gate layer and a single output gate layer (and so are 2-layer $\mathbf{AC}^0$ circuits). This circuit can be shown to be **DLOGTIME**-uniform.

Binary to Unary There is a constant depth circuit family where circuit C_n takes as input some word $w \in \{0,1\}^n$ and outputs 1^x where x is the positive integer encoded in the first $\lceil \log_2 n \rceil$ bits of w [11]. It can be shown that this circuit family is **DLOGTIME** uniform and so this conversion from short binary strings to unary is in $\mathbf{FAC}^0$.

Unary to Binary There is a constant depth circuit family where circuit C_n takes as input some word $w = 0^{n-x}1^x$ where $0 \leq x \leq n$, and outputs the binary encoding of x [11]. It can be shown that this circuit family is **DLOGTIME** uniform and so unary to binary conversion is in $\mathbf{FAC}^0$.

2.3 Configuration graphs

Definition 9 (Configuration Graph)**.** *Let $w \in \{0,1\}^*$ be the input to a halting Turing machine M. The* configuration graph *$C_{M,w}$ is a directed acyclic graph where each vertex encodes a configuration of M on inputs of length $|w|$. The graph $C_{M,w}$ has a directed edge from a vertex c to a vertex c' if the configuration encoded by c' can be reached from the configuration encoded by c in one step via M's transition function.*

A configuration graph $C_{M,w}$ has the property that there is a directed path from the vertex c_s representing the start configuration, to the accept vertex c_a if an only if M accepts input w. Lemma 10 follows from [16, 18].

Lemma 10. *Given the binary encoding of a Turing machine M, which has state set Q and has an* $\mathbf{FAC}^0$ *computable space bound $s = \mathcal{O}(\log |w|)$, and given an input w, the configuration graph $C_{M,w}$ is computable in* **DLOGTIME***-uniform-*$\mathbf{FAC}^0$ *and is of size $\mathcal{O}(2^s|w||Q|)$.*

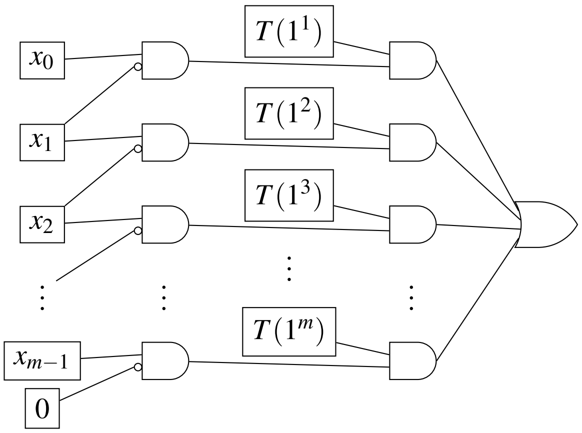

Figure 2: A gadget that simulates a single tally oracle gate. Gates of the form $T(1^i)$ are constant gates that simulate a Turing machine T: where $T(1^i) = 1$ if the Turing machine T accepts input $0^{m-i}1^i$, and $T(1^i) = 0$ otherwise.

3 Languages reducible to tallyNL

In this work we consider the class **tallyNL** as well as classes $\mathbf{AC}^0$ many-one, disjunctive truth-table, conjunctive truth-table, truth-table, and Turing reducible to **tallyNL**. Their containment relationships are shown in Figure 1. We prove the following for completeness.

Lemma 11. $\mathbf{FAC}^0_\mathrm{T}(\mathbf{tallyNL}) \subsetneq \mathbf{NL}$

Proof. ($\subseteq$) Let $L \in \mathbf{FAC}^0_\mathrm{T}(\mathbf{tallyNL})$. Since the circuit and the oracles compute functions in $\mathbf{NL}$, there is a non-deterministic logspace Turing machine that computes the composition of these functions.

($\neq$) Parity $\in \mathbf{NL}$. We know that Parity $\notin$ non-uniform-$\mathbf{AC}^0$ [12] and that **tally** $\subseteq$ non-uniform-$\mathbf{AC}^0$, hence it is sufficient to prove that $\mathbf{FAC}^0_\mathrm{T}(\mathbf{tallyNL}) \subseteq$ non-uniform-$\mathbf{AC}^0$.

Let $L \in \mathbf{FAC}^0_\mathrm{T}(\mathbf{tallyNL})$. Consider a family of circuits $\mathcal{C}_L$ that recognizes L and makes use of the Turing machine $\mathcal{M}$ as the tally oracle. Let $w \in \{0,1\}^*$, and consider the circuit $C_{|w|} \in \mathcal{C}_L$ that decides whether or not $w \in L$. There is some number $k \in \mathbb{N}$ of oracle gates in $C_{|w|}$. The ith such oracle gate, $i \in \{1,2,\ldots,k\}$, takes one of $m+1$ inputs where m is the number of wires into the gate (recall that inputs to the gate are of the form $0^{m-j}1^j$). We (non-uniformly) replace oracle gate i with the gadget shown in Figure 2. This gadget encodes tally machine answers as constants. The replacement can be done knowing $|w|$ (and not knowing w). We replace all k tally oracle gates with this gadget to get a new circuit that is a constant factor (i.e. 5 times) deeper than $C_{|w|}$ and polynomially (in $|w|$) larger. Applying this transformation to the entire family $\mathcal{C}$ results in a non-uniform $\mathbf{AC}^0$ circuit family that recognizes L. □

The same proof gives $\mathbf{FAC}^0_\mathrm{T}(\mathbf{tally}) \subseteq$ non-uniform-$\mathbf{AC}^0$ and hence $\mathbf{FAC}^0_\mathrm{T}(\mathbf{tally}) \neq \mathbf{NL}$, which holds for **tally** as opposed to **tallyNL**, and also for Turing reductions that are uniform-$\mathbf{FAC}^0$, or non-uniform-$\mathbf{FAC}^0$.

4 Uniform OR circuits

In this section we consider the relationship between uniform polynomial-size OR circuits and **tallyNL**. We also consider the classes of languages reducible to these classes by suitably weak reductions. We begin with a **lengthNL** lower bound on uniform polynomial size OR circuits. For this lower bound we consider **lengthNL** rather than **tallyNL** because OR circuits act on binary strings and **lengthNL** is a binary analogue of **tallyNL** (with almost the same proof we get an analogous **tallyNL** lower bound for $\mathbf{FAC}^0$-uniform-**OR** if we restrict to inputs from $\{1\}^*$).

Theorem 12. $\mathbf{lengthNL} \subsetneq \mathbf{FAC}^0$*-uniform-***OR***.*

Proof. Let $L \in \mathbf{lengthNL}$. L is accepted by a non-deterministic logspace Turing machine $\mathcal{M}$, for which one or more computation paths are accepting exactly for those words $w \in L \subseteq \{0,1\}^*$. The configuration graph $C_{\mathcal{M},w}$ for $\mathcal{M}$ on input $w \in \{0,1\}^*$ is $\mathbf{FAC}^0$ computable from $\mathcal{M}$ and w (see Lemma 10). We construct the configuration graph assuming that its input w is $1^{|w|}$ (recall that if $w \in L$ then all words in $\{0,1\}^{|w|}$ are in L). We modify the graph $C_{\mathcal{M},w}$ to create an OR circuit as follows. Each edge becomes a wire and each vertex becomes an OR gate, except the start vertex (representing the initial configuration of $\mathcal{M}$ on input $1^{|w|}$) which becomes a constant 1 gate. We add $|w|$ "dummy" input gates that are not wired to anything. We add a new OR gate that is the circuit's output gate, and a constant 0 is wired into the every OR gate in the circuit. All accept-vertices (representing the accepting configurations) are wired into this output gate. If $w \in L$ the circuit accepts since there is a path from 1 to the output gate. If $w \notin L$ the circuit rejects since there is no path from 1 to the output gate.

If we apply this transformation to the set of all configurations graphs for the fixed machine $\mathcal{M}$ over all inputs $w \in \{1\}^*$, we get a circuit family $\mathcal{C}$. Members of such a circuit family are computable by an $\mathbf{FAC}^0$ function $f_{\mathcal{M}} : \{1\}^* \to \mathcal{C}$.

Consider the language $L = \{w \mid w \text{ has at least one } 1\}$ which is easily seen to be in $\mathbf{FAC}^0$-uniform-**OR** but not in **lengthNL**, giving the required inequality for strict containment. □

Next we show that the languages accepted by uniform polynomial-size OR circuits are strictly contained in those disjunctive truth-table reducible to **tallyNL**.

Theorem 13. $\mathbf{FAC}^0$*-uniform-***OR** $\subsetneq \mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$

Proof. It is trivially the case that $\mathbf{FAC}^0$-uniform-**OR** $\subseteq \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0$-uniform-**OR**$)$. Then, by applying Theorem 14 (stated and proved below) we get that $\mathbf{FAC}^0$-uniform-**OR** $\subseteq \mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL}) = \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0$-uniform-**OR**$)$. To show strict containment, observe that $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$ contains languages in $\mathbf{AC}^0 \cap$ non-uniform-**AND** that are not accepted by any OR circuit family. □

Since the previously stated upper and lower bounds on $\mathbf{FAC}^0$-uniform-**OR** are both strict, it is natural to ask how $\mathbf{FAC}^0$-uniform-**OR** relates to the most obvious class that lies between these bounds, namely $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{tallyNL})$. In fact, we get an inequality: $\mathbf{FAC}^0$-uniform-**OR** $\neq \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{tallyNL})$, as $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{tallyNL})$ contains languages in $\mathbf{AC}^0 \cap$ non-uniform-**AND** that are not accepted by any OR circuit family.

The remainder of this section is concerned with the proof of Theorem 14, which was used in Theorem 13 to give an upper bound on $\mathbf{FAC}^0$-uniform-**OR**, and shows the equivalence of three complexity classes.

Theorem 14. *The following classes are equal:*

- $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0$*-uniform-***OR**$)$

- $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{FAC}^0\text{-}\textit{uniform}\text{-}\mathbf{OR})$
- $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$

This theorem is proven by the inclusion cycle in Lemmas 15, 16, and 17 below.

Lemma 15. $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\text{-}\textit{uniform}\text{-}\mathbf{OR}) \subseteq \mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{FAC}^0\text{-}\textit{uniform}\text{-}\mathbf{OR})$

Proof. The latter class is a generalization of the former. □

Lemma 16. $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{FAC}^0\text{-}\textit{uniform}\text{-}\mathbf{OR}) \subseteq \mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$

Proof. Let $L \in \mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{FAC}^0\text{-uniform-}\mathbf{OR})$ with oracle language $L' \in \mathbf{FAC}^0\text{-uniform-}\mathbf{OR}$. That is, there exists a function $\tau \in \mathbf{FAC}^0$ mapping from $\{0,1\}^*$ to the set of tuples of binary words where *at least one* word in the tuple $\tau(w) = (x_1, x_2, \ldots, x_m)$ is in L' iff $w \in L$.

To show that any of the binary words $\tau(w) = (x_1, x_2, \ldots, x_m)$ are in L' (i.e. are accepted by the OR circuit family) it is sufficient to show that there is a single bit 1 in a word from $\tau(w)$ such that the bit's assigned input gate is on a path to the output gate in the appropriate OR circuit (or that there is a constant 1 gate in some circuit that is on a path to the output gate).

With this in mind, we define the function $\tau' \in \mathbf{FAC}^0$, from $\{0,1\}^*$ to the set of tuples of unary words. $\tau'(w) = (u_1, \ldots, u_{q(|w|)})$, where $q(|w|)$ is polynomial in $|w|$, such that for each bit i in each word x_l in $\tau(w)$, there is a unary word $u_{l,i}$ in $\tau'(w)$ that encodes both $|x_l|$ (i.e. the length of x_l) and i, specifically:

$$u_{l,i} = \begin{cases} 1^{\langle |x_l|, |x_l| \rangle} & \text{if } i = |x_l|, \\ 1^{\langle i, |x_l| \rangle} & \text{if } 0 \le i \le |x_l| - 1 \text{ and bit } i \text{ of } x_l \text{ is } 1, \\ 1 & \text{if } 0 \le i \le |x_l| - 1 \text{ and bit } i \text{ of } x_l \text{ is } 0. \end{cases} \tag{1}$$

Here $u_{l,i}$ is the (l,i)th word in $\tau'(w)$, x_l is the lth word in $\tau(w)$ and $\langle \cdot, \cdot \rangle$ denotes the pairing function in Section 2.2. (Note that 0 bits are not uniquely encoded; our construction does not require it.)

Now we argue that $\tau' \in \mathbf{FAC}^0$. Each of the $q(|w|)$ unary words in $\tau'(w)$ are computed independently and in parallel. The (l,i)th unary word is computed as follows: First compute $x_l \in \{0,1\}^*$, which is the lth word in $\tau(w)$. If the ith bit of x_l is 0 then output the unary word 1. Otherwise compute the pairing $k = \langle i, |x_l| \rangle$ (Section 2.2), convert the binary number k to unary to give 1^k which is then output in an encoded form as $0^{z-k}1^k$ where $1 \le k < z$, $z = 2^{2\lceil \log |w| + 1 \rceil} \in \mathcal{O}(|w|^2)$. The (l,i)th sub-circuit of τ' is composed of a constant number of $\mathbf{FAC}^0$ computable routines from Section 2.2 along with the computation of τ which is, by hypothesis, in $\mathbf{FAC}^0$. The polynomial number $q(|w|)$ of such constant depth computations are done in parallel, hence $\tau' \in \mathbf{FAC}^0$.

Let $f \in \mathbf{FAC}^0$, $f : \{1\}^* \to \mathcal{C}$, be the uniformity function of the OR-circuit family that recognises L'. We next define a non-deterministic Turing machine $\mathcal{M}_f$ that takes unary input, and makes use of f. The machine $\mathcal{M}_f$ is defined to reject on input word 1 and accept input 1^k if $k > 1$ and if the un-pairing (see Section 2.2) of the binary encoding of k gives two binary numbers n and i, such that *there is a path* from the ith input gate to the output gate of circuit $f(1^n)$. $\mathcal{M}_f$ also accepts if $i = n$ and there is a path from some constant 1 gate to the output gate of circuit $f(1^n)$. $\mathcal{M}_f$ works as follows. $\mathcal{M}_f$ computes the unary to binary conversion and the un-pairing routine in logspace (see Section 2.2). By hypothesis, the uniformity function f is in $\mathbf{FAC}^0$ so, by using the standard re-computation trick [7, 22] for logspace Turing machines, $\mathcal{M}_f$ both computes f and tests reachability from input gate i to the output gate of circuit $f(1^n)$ in non-deterministic logspace. Hence, if there is a path from input gate i (or some constant 1 gate) to the output gate then $\mathcal{M}_f$ accepts, otherwise if no path is found then $\mathcal{M}_f$ rejects. Moreover, since $\mathcal{M}_f$ uses space $O(\log k)$, the language it accepts is in $\mathbf{tallyNL}$.

$\mathcal{M}_f$ will be our **tallyNL** oracle machine. We now prove that for any $w \in \{0,1\}^*$, at least one word in the tuple $\tau'(w)$ is accepted by at least one of the $\mathcal{M}_f$ oracle machines iff $w \in L$. If $w \in L$ then there exists a word x in the tuple $\tau(w)$ with at least one bit with value 1 that is assigned to an input gate that is on a path to the output gate in OR circuit $f(1^{|x|})$. This means that the tuple of words $\tau'(w)$ contains at least one unary word that encodes $|x|$ and i, where i is the bit position assigned to 1. By the construction in the previous paragraph, this word in $\tau'(w)$ is accepted by $\mathcal{M}_f$.

If $w \notin L$ then by hypothesis there are no words in $\tau(w)$ that are accepted by the uniform OR circuit family. Any 0's in words from $\tau(w)$ become encoded as the input 1 to $\mathcal{M}_f$, which is rejected by $\mathcal{M}_f$ since $k = 1$. While $\tau(w)$ may contain words x with bits set to 1 (or constant bits set to 1), these bits are assigned to input (or constant) gates that do not have a path to the output gate in the circuit $f(1^{|x|})$. Hence, none of these words in $\tau'(w)$ will be accepted by the oracle calls to $\mathcal{M}_f$.

Therefore τ' is a disjunctive truth-table reduction from L to a language in **tallyNL**. □

Lemma 17. $\mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL}) \subseteq \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{OR})$

Proof. Let $L \in \mathbf{FAC}^0_{\mathrm{dtt}}(\mathbf{tallyNL})$ with $T \in \mathbf{tallyNL}$ as the oracle language. That is, there exists a function $\tau \in \mathbf{FAC}^0$ that maps $\{0,1\}^*$ to the set of tuples of unary words, where at least one word in the tuple $\tau(w) = (x_1, x_2, \ldots, x_\ell)$ is in T iff $w \in L$.

Let $r : \{0,1\}^* \to \{0,1\}^*$. Let the notation $r(w)_k$ denote the kth bit of the word $r(w)$. The function r is defined in a bitwise fashion as follows:

$$r(w)_k = \begin{cases} 1 & \text{if } 1^k \text{ is in the tuple } \tau(w), \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

We claim that r is an $\mathbf{FAC}^0$ many-one reduction from L to a language in $\mathbf{FAC}^0$-uniform-**OR**.

First we prove that $r \in \mathbf{FAC}^0$. The circuit that computes $r(w)$ first computes the tuple $\tau(w)$, which is possible since $\tau \in \mathbf{FAC}^0$. Without loss of generality we say that $\tau(w)$ is a tuple of $\ell \in \mathbb{N}$ unary words, each of length $\leq q \in \mathbb{N}$, and each of which is padded up to length q with 0's (i.e. the unary word 1^k is padded to be $0^{q-k}1^k$; this technicality comes from the fact that the circuit has a fixed number q of wires used encode a unary string which is dependent on the circuit input). Then, in constant depth, the circuit translates each string of the form $0^{q-k}1^k$ into a string of the form $0^{q-k}10^{k-1}$. All ℓ such words are then bitwise ORed to give a single binary string of length q, that represents $r(w)$. This is all easily achieved in $\mathbf{FAC}^0$.

We now describe a uniform polynomial-size OR circuit family $\mathcal{C}$. Let $f_{\mathcal{M}} : \{1\}^* \to \mathcal{C}$ be the uniformity function of the circuit family $\mathcal{C}$. On 1^m, the function $f_{\mathcal{M}}$ creates m configuration graphs: one configuration graph $C_{\mathcal{M},k}$ of machine $\mathcal{M}$ (that accepts T) on input 1^k for each $k \in \{1, \ldots, m\}$ (a generalization of the technique used in the proof of Theorem 12). Then, each of the m graphs are modified and connected together to create a single OR circuit as follows. Each edge becomes a wire. The vertex in $C_{\mathcal{M},k}$ that represents the start configuration of $\mathcal{M}$ on input 1^k becomes the kth input gate of the OR circuit. All other vertices become an OR gate. For each k, all accept vertices of the graph $C_{\mathcal{M},k}$ (representing the accepting configurations) are wired into a new OR gate o_k. We add a single constant 0 gate which is wired into every OR gate in the circuit. Finally each of the o_k gates, where $1 \leq k \leq m$, are wired into a single OR gate which is the output gate. $\mathcal{C}$ is of polynomial size (each circuit $f_{\mathcal{M}}(1^m)$ is of size polynomial in m), and it is relatively straightforward to verify that $\mathcal{C}$ is $\mathbf{FAC}^0$ uniform.

We need to argue that the circuit family $\mathcal{C}$ accepts $r(w)$ iff $w \in L$. Suppose $w \in L$. This implies that the tuple $\tau(w)$ contains at least one word 1^j in the tally set T. In turn, this implies that bit j in $r(w)$ is 1 (formally, $r(w)_j = 1$). Let $|r(w)| = m$. The fact that $\mathcal{M}$ accepts 1^j implies that the circuit

$c_m = f_{\mathcal{M}}(1^m) \in \mathcal{C}$ is constructed in such a way that its jth input gate is on a path to its output gate. Input gate j is set to 1, therefore circuit c_m accepts $r(w)$.

Suppose $w \notin L$. Hence, no word in the tuple $\tau(w)$ is in the tally set T. Let 1^j be any unary word in the tuple $\tau(w)$. In turn, this implies that bit j in $r(w)$ is 1 (formally, $r(w)_j = 1$). Let $|r(w)| = m$. Consider the circuit $C_m = f_{\mathcal{M}}(1^m) \in \mathcal{C}$. Since the Turing machine $\mathcal{M}$ does not accept 1^j, this implies that there is no path from input gate j in C_m to the output gate of C_m. Since C_m is an OR circuit with no paths from the input gates that are set to 1 to the output gate, and where there are no constant 1 gates, it rejects $r(w)$.

Therefore r is a many-one reduction from L to a language in $\mathbf{FAC}^0$-uniform-**OR**. □

Section 5 contains our results on AND circuits, analogous to those shown here for OR circuits.

We omit the proofs of the following theorems, which can be obtained using the techniques in this section and those in Section 5.

Theorem 18. *The following classes are equal:*

- $\mathbf{FAC}^0_{\mathrm{T}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{OR})$
- $\mathbf{FAC}^0_{\mathrm{T}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND})$
- $\mathbf{FAC}^0_{\mathrm{T}}(\mathbf{tallyNL})$

Theorem 19. *The following classes are equal:*

- $\mathbf{FAC}^0_{\mathrm{tt}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{OR})$
- $\mathbf{FAC}^0_{\mathrm{tt}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND})$
- $\mathbf{FAC}^0_{\mathrm{tt}}(\mathbf{tallyNL})$

5 Uniform AND circuits

Here we give upper bounds and lower bounds on the power of uniform AND circuits in terms of **tallyNL** and problems reducible to **tallyNL**. The proofs have a similar flow to those for OR circuits in the Section 4, although in a number of cases different tricks are used.

We begin with an upperbound and lowerbound on polynomial-size uniform AND circuits: i.e. the class $\mathbf{FAC}^0$-uniform-**AND**.

Theorem 20. $\mathbf{lengthNL} \subsetneq \mathbf{FAC}^0$*-uniform-*$\mathbf{AND}$.

Proof. Let $L \in \mathbf{lengthNL}$. Since $\mathbf{lengthNL} = \mathbf{lengthcoNL}$, this implies that L is accepted by a co-nondeterministic logspace Turing machine $\mathcal{M}$, for which all computation paths are accepting exactly for those words $w \in L$. The configuration graph $C_{\mathcal{M},w}$ for $\mathcal{M}$ on input $w \in \{0,1\}^*$ is $\mathbf{FAC}^0$ computable from $\mathcal{M}$ and w (see Lemma 10). We construct the configuration graph assuming that its input w is $1^{|w|}$ (recall that if $w \in L$ then all words in $\{0,1\}^{|w|}$ are in L). We modify the graph $C_{\mathcal{M},w}$ to create an AND circuit as follows. Each edge becomes a wire and each vertex becomes an AND gate, except the start vertex (representing the initial configuration of $\mathcal{M}$ on input w) which becomes a constant 0 gate. We add $|w|$ "dummy" input gates that are not wired to anything. We add a new AND gate that is the circuit's output gate, and a constant 1 is wired into every AND gate in the circuit. All reject vertices (representing the rejecting configurations) are wired into the output gate. If $w \in L$ the circuit accepts since there is no path from 0 to the output gate. If $w \notin L$ the circuit rejects since there is a path from 0 to the output gate.

If we apply this transformation to the set of all configurations graphs for the fixed machine $\mathcal{M}$ over all inputs $w \in \{1\}^*$, we get a circuit family $\mathcal{C}$. Members of such a circuit family are computable by an $\mathbf{FAC}^0$ function $f_{\mathcal{M}} : \{1\}^* \to \mathcal{C}$.

Consider the language $L = \{1^n \mid n \in \mathbb{N}\}$ which is easily seen to be in $\mathbf{FAC}^0$-uniform-**AND** but not in **lengthNL**, giving the required inequality for strict containment. □

Next we show that languages accepted by uniform polynomial-size AND circuits are strictly contained in those conjunctive truth-table reducible to **tallyNL**.

Theorem 21. $\mathbf{FAC}^0$*-uniform-*$\mathbf{AND} \subsetneq \mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL})$

Proof. It is trivially the case that $\mathbf{FAC}^0$-uniform-$\mathbf{AND} \subseteq \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\text{-uniform-}\mathbf{AND})$. Then, by applying Theorem 22 (stated and proved below) we get that $\mathbf{FAC}^0$-uniform-$\mathbf{AND} \subseteq \mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL}) = \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\text{-uniform-}\mathbf{AND})$. To show strict containment, observe that $\mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL})$ contains languages in $\mathbf{AC}^0 \cap$ non-uniform-$\mathbf{OR}$ that are not accepted by any AND circuit family. □

We also get the following inequality: $\mathbf{FAC}^0$-uniform-$\mathbf{AND} \neq \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{tallyNL})$, as $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{tallyNL})$ contains languages in $\mathbf{AC}^0 \cap$ non-uniform-$\mathbf{OR}$ that are not accepted by any OR circuit family.

The remainder of this section is concerned with the proof of Theorem 22, which was used in Theorem 21 to give an upper bound on $\mathbf{FAC}^0$-uniform-$\mathbf{AND}$, and shows the equivalence of three complexity classes.

Theorem 22. *The following classes are equal:*

- $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND})$
- $\mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND})$
- $\mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL})$

This theorem is proven by the cycle of inclusions in Lemmas 23, 24, and 25 below.

Lemma 23. $\mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND}) \subseteq \mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND})$

Proof. The latter class is a generalization of the former. □

Lemma 24. $\mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND}) \subseteq \mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL})$

Proof. Let $L \in \mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{FAC}^0\text{-uniform-}\mathbf{AND})$ with oracle language $L' \in \mathbf{FAC}^0$-uniform-$\mathbf{AND}$. That is, there exists a function $\tau \in \mathbf{FAC}^0$ mapping from $\{0,1\}^*$ to the set of tuples of binary words where *all* words in the tuple $\tau(w) = (x_1, x_2, \dots, x_m)$ are in L' iff $w \in L$.

To show that any of the binary words $\tau(w) = (x_1, x_2, \dots, x_m)$ are not in L' (i.e. are rejected by the AND circuit family) it is sufficient to show that there is a single bit 0 in a word from $\tau(w)$ such that the bit's assigned input gate is on a path to the output gate in the appropriate AND circuit (or that there is a constant 0 gate in some circuit that is on a path to the output gate).

With this in mind, we define the function $\tau' \in \mathbf{FAC}^0$, from $\{0,1\}^*$ to the set of tuples of unary words. $\tau'(w) = (u_1, \dots, u_{q(|w|)})$, where $q(|w|)$ is polynomial in $|w|$, such that for each bit i in each word x_l in $\tau(w)$, there is a unary word $u_{l,i}$ in $\tau'(w)$ that encodes both $|x_l|$ (i.e. the length of x_l) and i, specifically:

$$u_{l,i} = \begin{cases} 1^{\langle |x_l|, |x_l| \rangle} & \text{if } i = |x_l|, \\ 1^{\langle i, |x_l| \rangle} & \text{if } 0 \leq i \leq |x_l| - 1 \text{ and bit } i \text{ of } x_l \text{ is } 0, \\ 1 & \text{if } 0 \leq i \leq |x_l| - 1 \text{ and bit } i \text{ of } x_l \text{ is } 1. \end{cases} \tag{3}$$

Here $u_{l,i}$ is the (l,i)th word in $\tau'(w)$, x_l is the lth word in $\tau(w)$ and $\langle \cdot, \cdot \rangle$ denotes the pairing function in Section 2.2. (Note that 1 bits are not uniquely encoded; our construction does not require it.)

Now we argue that $\tau' \in \mathbf{FAC}^0$. Each of the $q(|w|)$ unary words in $\tau'(w)$ are computed independently and in parallel. The (l,i)th unary word is computed as follows: First compute $x_l \in \{0,1\}^*$, which is the lth word in $\tau(w)$. If the ith bit of x_l is 1 then output the unary word 1. Otherwise compute the pairing $k = \langle i, |x_l| \rangle$ (Section 2.2), convert the binary number k to unary to give 1^k which is then output in an encoded form as $0^{z-k}1^k$ where $1 \leq k < z$, $z = 2^{2\lceil \log |w|+1 \rceil} \in \mathcal{O}(|w|^2)$. The (l,i)th sub-circuit of τ' is composed of a constant number of $\mathbf{FAC}^0$ computable routines from Section 2.2 along with the computation of τ which is, by hypothesis, in $\mathbf{FAC}^0$. The polynomial number $q(|w|)$ of such constant depth computations are done in parallel, hence $\tau' \in \mathbf{FAC}^0$.

Let $f \in \mathbf{FAC}^0$, $f : \{1\}^* \to \mathcal{C}$, be the uniformity function of the AND-circuit family that recognises L'. We next define a non-deterministic Turing machine $\mathcal{M}_f$ that takes unary input, and makes use of f. The machine $\mathcal{M}_f$ is defined to accept on input word 1 and reject input 1^k if $k > 1$ and if the un-pairing (see Section 2.2) of the binary encoding of k gives two binary numbers n and i, such that *there is a path* from the ith input gate to the output gate of circuit $f(1^n)$. $\mathcal{M}_f$ also accepts if $i = n$ and there is a path from some constant 0 gate to the output gate of circuit $f(1^n)$. $\mathcal{M}_f$ works as follows. $\mathcal{M}_f$ computes the unary to binary conversion and the un-pairing routine in logspace (see Section 2.2). By hypothesis, the uniformity function f is in $\mathbf{FAC}^0$ so, by using the standard re-computation trick [7, 22] for logspace Turing machines and the un-reachability algorithm [17, 25] $\mathcal{M}_f$ both computes f and tests non-reachability from input gate i to the output gate of circuit $f(1^n)$ in non-deterministic logspace. Hence, if there is a path from input gate i (or some constant 0 gate) to the output gate then $\mathcal{M}_f$ rejects, otherwise if no path is found then $\mathcal{M}_f$ accepts. Moreover, since $\mathcal{M}_f$ uses space $O(\log k)$, the language it accepts is in $\mathbf{tallyNL} = \mathbf{tallycoNL}$.

$\mathcal{M}_f$ will be our **tallyNL** oracle machine. We now prove that for any $w \in \{0,1\}^*$, all words in the tuple $\tau'(w)$ are accepted by the $\mathcal{M}_f$ oracle machines iff $w \in L$. If $w \notin L$ then there exists a word x in the tuple $\tau(w)$ with at least one bit with value 0 that is assigned to an input gate that is on a path to the output gate in AND circuit $f(1^{|x|})$. This means that the tuple of words $\tau'(w)$ contains at least one unary word that encodes $|x|$ and i, where i is the bit position assigned to 0. By the construction in the previous paragraph, this word in $\tau'(w)$ is rejected by $\mathcal{M}_f$.

If $w \in L$ then by hypothesis there are no words in $\tau(w)$ that are rejected by the uniform AND circuit family. Any 1's in words from $\tau(w)$ become encoded as the input 1 to $\mathcal{M}_f$, which is accepted by $\mathcal{M}_f$ since $k = 1$. While $\tau(w)$ may contain words x with bits set to 0 (or constant bits set to 0), these bits are not assigned to input (or constant) gates that have a path to the output gate in the circuit $f(1^{|x|})$. Hence, none of the words in $\tau'(w)$ will be rejected by the oracle calls to $\mathcal{M}_f$.

Therefore τ' is a conjunctive truth-table reduction from L to a language in **tallyNL**. □

Lemma 25. $\mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL}) \subseteq \mathbf{FAC}^0_{\mathrm{m}}(\mathbf{FAC}^0\textit{-uniform-}\mathbf{AND})$

Proof. Let $L \in \mathbf{FAC}^0_{\mathrm{ctt}}(\mathbf{tallyNL})$ with $T \in \mathbf{tallyNL}$ as the oracle language. That is, there exists a function $\tau \in \mathbf{FAC}^0$ that maps $\{0,1\}^*$ to the set of tuples of unary words, where all words in the tuple $\tau(w) = (x_1, x_2, \ldots, x_\ell)$ are in T iff $w \in L$.

Let $r : \{0,1\}^* \to \{0,1\}^*$. Let the notation $r(w)_k$ denote the kth bit of the word $r(w)$. The function r is defined in a bitwise fashion as follows:

$$r(w)_k = \begin{cases} 0 & \text{if } 1^k \text{ is in the tuple } \tau(w), \\ 1 & \text{otherwise.} \end{cases} \tag{4}$$

We claim that r is an $\mathbf{FAC}^0$ many-one reduction from L to a language in $\mathbf{FAC}^0$-uniform-**AND**.

First we prove that $r \in \mathbf{FAC}^0$. The circuit that computes $r(w)$ first computes the tuple $\tau(w)$, which is possible since $\tau \in \mathbf{FAC}^0$. Without loss of generality we say that $\tau(w)$ is a tuple of $\ell \in \mathbb{N}$ unary words, each of length $\leq q \in \mathbb{N}$, and each of which is padded up to length q with 0's (i.e. the unary word 1^k is padded to be $0^{q-k}1^k$; this technicality comes from the fact that the circuit has a fixed number q of wires used encode a unary string which is dependent on the circuit input). Then, in constant depth, the circuit translates each string of the form $0^{q-k}1^k$ into a string of the form $1^{q-k}0^1 1^{k-1}$. All ℓ such words are then bitwise ANDed to give a single binary string of length q, that represents $r(w)$. This is all easily achieved in $\mathbf{FAC}^0$.

We now describe a uniform polynomial-size AND circuit family $\mathcal{C}$. Let $f_{\mathcal{M}} : \{1\}^* \to \mathcal{C}$ be the uniformity function of the circuit family $\mathcal{C}$. On 1^m, the function $f_{\mathcal{M}}$ creates m configuration graphs: one configuration graph $C_{\mathcal{M},k}$ of machine $\mathcal{M}$ (that accepts T) on input 1^k for each $k \in \{1,\ldots,m\}$ (a generalization of the technique used in the proof of Theorem 20). Then, each of the m graphs are modified and connected together to create a single AND circuit as follows. Each edge becomes a wire. The vertex in $C_{\mathcal{M},k}$ that represents the start configuration of $\mathcal{M}$ on input 1^k becomes the kth input gate of the AND circuit. All other vertices become an AND gate. For each k, all reject vertices of the graph $C_{\mathcal{M},k}$ (representing the rejecting configurations) are wired into a new AND gate o_k. We add a single constant 1 gate which is wired into every AND gate in the circuit. Finally each of the o_k gates, where $1 \leq k \leq m$, are wired into a single AND gate which is the output gate. $\mathcal{C}$ is of polynomial size (each circuit $f_{\mathcal{M}}(1^m)$ is of size polynomial in m), and it is relatively straightforward to verify that $\mathcal{C}$ is $\mathbf{FAC}^0$ uniform.

We need to argue that the circuit family $\mathcal{C}$ accepts $r(w)$ iff $w \in L$. Suppose $w \notin L$. This implies that the tuple $\tau(w)$ contains at least one word 1^j not in the tally set T. In turn, this implies that bit j in $r(w)$ is 0 (formally, $r(w)_j = 0$). Let $|r(w)| = m$. The fact that $\mathcal{M}$ rejects 1^j implies that the circuit $c_m = f_{\mathcal{M}}(1^m) \in \mathcal{C}$ is constructed in such a way that its jth input gate is on a path to its output gate. Input gate j is set to 0, therefore circuit c_m rejects $r(w)$.

Suppose $w \in L$. Hence, all words in the tuple $\tau(w)$ are in the tally set T. Let 1^j be any unary word in the tuple $\tau(w)$. In turn, this implies that bit j in $r(w)$ is 0 (formally, $r(w)_j = 0$). Let $|r(w)| = m$. Consider the circuit $C_m = f_{\mathcal{M}}(1^m) \in \mathcal{C}$. Since the Turing machine $\mathcal{M}$ does not reject 1^j, this implies that there is no path from input gate j in C_m to the output gate of C_m. Since C_m is an AND circuit with no paths from the input gates that are set to 0 to the output gate, and where there are no constant 0 gates, it accepts $r(w)$.

Therefore r is a many-one reduction from L to a language in $\mathbf{FAC}^0$-uniform-**AND**. □

6 Semi-uniform circuit families

We introduce a definition of semi-uniform families of Boolean circuits. This definition is inspired by the concept in membrane systems [23]. Polynomial-size semi-uniform OR circuits, and AND circuits, are shown to characterize **NL**.

Definition 26 (Semi-uniform circuit family)**.** *A* semi-uniform circuit family $\mathcal{C}$ *is a set of Boolean circuits, each with a single output gate and no input gates, such that there is a function* $h : \{0,1\}^* \to \mathcal{C}$ *(computable within some resource bound) where* $h(x) = C_x$. *We say that a semi-uniform circuit family* $\mathcal{C}$ *decides a language* X *if for each* x, *the circuit* $h(x) = C_x \in \mathcal{C}$ *evaluates to* 1 *if* $x \in X$ *and* 0 *if* $x \notin X$.

Here, h is called the semi-uniformity function of $\mathcal{C}$. The intuition behind the definition is that the semi-uniformity function has access to the entire input word, whereas more standard uniformity functions access only the input word length (in unary).

Definition 27 ($\mathbf{FAC}^0$-semi-uniform-**OR**)**.** *Let* $\mathbf{FAC}^0$*-semi-uniform-***OR** *be the set of decision problems over a binary alphabet that are solved by* $\mathbf{FAC}^0$ *semi-uniform families of* OR *circuits.*

$\mathbf{FAC}^0$-semi-uniform-**AND** is defined analogously using AND circuits. Finally, the class $\mathbf{FAC}^0$-semi-uniform-**AND**-**OR** is defined analogously using circuits that have both AND and OR gates. The proof of the following lemma is straightforward.

Lemma 28. $\mathbf{FAC}^0$*-semi-uniform-***AND**-**OR** $= \mathbf{P}$

Proof. Any problem in **P** has a circuit family $\mathcal{C}$ with circuits using AND, OR, and NOT gates that is uniform by some function $f \in \mathbf{FAC}^0$, $f : \{1\}^* \to \mathcal{C}$. There is a semi-uniformity function $f' : \{0,1\}^* \to \mathcal{C}'$ for a semi-uniform circuit family $\mathcal{C}'$, that simulates f in the following way: For all $x \in \{0,1\}^*$, $f'(x)$ produces a circuit without input gates and where the string x and its bitwise complement are available as constants, and the circuit carries out a dual-rail logic simulation [13, 14] of the circuit $f(|x|)$. □

Lemma 29. $\mathbf{FAC}^0$*-semi-uniform-***OR** $=$ **NL**.

Proof. ($\mathbf{NL} \subseteq \mathbf{FAC}^0$-semi-uniform-**OR**) Let $L \in \mathbf{NL}$. L is accepted by a non-deterministic logspace Turing machine M, i.e. one or more computation paths are accepting exactly for those words $w \in L \subseteq \{0,1\}^*$. Consider the configuration graph $C_{M,w}$ for M on input $w \in \{0,1\}^*$, which is $\mathbf{FAC}^0$ computable from M and w (see Section 2.3). We modify the graph $C_{M,w}$ to create an OR circuit as follows. Each edge becomes a wire and each vertex becomes an OR gate, except the start vertex (which represents the initial configuration of M on w) which becomes a constant 1 gate. All *accepting vertices* (representing accepting configurations) are also wired to this output gate. We add a single constant 0 gate which is wired into every OR gate in the circuit. If $w \in L$ the circuit accepts since there is a path from 1 to the output gate. If $w \notin L$ the circuit rejects since there is no path from 1 to the output gate and a 0 feeds into that gate. These simple modifications can be made in $\mathbf{FAC}^0$.

Fixing the machine M, and then considering this transformation on the set of all configurations graphs, one for each input $w \in \{0,1\}^*$, we get a semi-uniform circuit family $\mathcal{C}$. Members of such a semi-uniform circuit family are computable by an $\mathbf{FAC}^0$ function $f_M : \{0,1\}^* \to \mathcal{C}$.

($\mathbf{FAC}^0$-semi-uniform-**OR** $\subseteq$ **NL**) Let $\mathcal{C}$ be a semi-uniform OR circuit family that recognizes $L \in \mathbf{FAC}^0$-semi-uniform-**OR**, we claim that there is a non-deterministic logspace Turing machine M that recognizes L. Let $h : \{0,1\}^* \to \mathcal{C}$ be the semi-uniformity function of $\mathcal{C}$. On input $x \in \{0,1\}^*$, M computes $h(x)$ and performs a simple reachability on the resulting OR circuit in the following way: M guesses a gate, if that gate is a constant 1-gate M then guesses a path from that gate, if the path ends at the output gate M accepts. □

Lemma 30. $\mathbf{FAC}^0$*-semi-uniform-***AND** $=$ **NL**.

Proof. ($\mathbf{NL} \subseteq \mathbf{FAC}^0$-semi-uniform-**AND**) Let $L \in$ **tallyNL**. Since **tallyNL** $=$ **tallycoNL** (Lemma 5), this implies that L is accepted by a co-non-deterministic logspace Turing machine M, for which all computation paths accept exactly for those words $w \in L \subseteq \{0,1\}^*$. Consider the configuration graph $C_{M,w}$ for M on input $w \in \{0,1\}^*$, which is $\mathbf{FAC}^0$ computable from M and w (see Section 2.3). We modify the graph $C_{M,w}$ to create an AND circuit as follows. Each edge becomes a wire and each vertex becomes an AND gate, except the start vertex (which represents the initial configuration on M on w) which becomes a constant 0 gate. We add a new AND gate that is the circuit's output gate. All reject vertex (representing the reject configurations) are wired into this output gate. We add a single constant 1 gate which is wired into every AND gate in the circuit. These modifications can be made in $\mathbf{FAC}^0$. If $w \in L$ the circuit accepts since there is no path from 0 to the output gate. If $w \notin L$ the circuit rejects since there is a path from 0 to the output gate.

Fixing the machine M, and then considering this transformation on the set of all configurations graphs, one for each input $w \in \{0,1\}^*$, we get a semi-uniform circuit family $\mathcal{C}$. Members of such a semi-uniform circuit family are computable by an $\mathbf{FAC}^0$ function $f_M : \{0,1\}^* \to \mathcal{C}$.

($\mathbf{FAC}^0$-semi-uniform-**AND** $\subseteq$ **NL**) Let $\mathcal{C}$ be a semi-uniform AND circuit family that recognizes $L \in \mathbf{FAC}^0$-semi-uniform-**AND**. We claim that there is a co-nondeterministic logspace Turing machine M that recognizes L and thus $L \in \mathbf{NL}$. Let $h : \{0,1\}^* \to \mathcal{C}$ be the semi-uniformity function of $\mathcal{C}$. On input $x \in \{0,1\}^*$, M computes $h(x)$ and performs a simple reachability on the resulting AND circuit in the following way. Starting at the output gate, M guesses a path along the reverse direction of the edges (wires) until the path terminates. If the path terminates at a constant 1 gate M accepts, otherwise M rejects (in the latter case the path terminates at a 0 gate, as by definition there are no AND gates with in-degree 0 in the circuit). M accepts x if and only if all of its computations accept, which is equivalent to saying that each path from an in-degree 0 gate to the circuit's output gate begins at a constant 1 gate, and so the circuit accepts. □

We have the following separation between uniform polynomial-size and semi-uniform OR circuits. The result also holds for AND circuits.

Theorem 31.

- $\mathbf{FAC}^0$-*uniform*-**OR** $\subsetneq$ $\mathbf{FAC}^0$-*semi-uniform*-**OR**
- $\mathbf{FAC}^0$-*uniform*-**AND** $\subsetneq$ $\mathbf{FAC}^0$-*semi-uniform*-**AND**

Proof. Follows from Theorem 11 and the containments in Figure 1. □

Acknowledgements

Many thanks to Eric Allender for valuable comments and discussion on uniform Boolean circuits and complexity classes within **P**. We also thank Antonio E. Porreca, David Doty, Jack Lutz and Dirk Walther for interesting discussions.

References

[1] Manindra Agrawal (2001): *The First-Order Isomorphism Theorem*. In: *FST TCS '01: Proc. of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 2245, Springer-Verlag, London, UK, pp. 70–82, doi:10.1007/3-540-45294-X_7.

[2] Manindra Agrawal (2011): *The Isomorphism Conjecture for constant depth reductions*. *Journal of Computer and System Sciences* 77(1), pp. 3–13, doi:10.1016/j.jcss.2010.06.003.

[3] Eric Allender (2012): *Investigations Concerning the Structure of Complete Sets*. In: *Workshop on Complexity and Logic*.

[4] Eric Allender & Michal Koucký (2010): *Amplifying lower bounds by means of self-reducibility*. *Journal of the ACM* 57, pp. 14:1–14:36, doi:10.1145/1706591.1706594.

[5] Eric Allender, David A. Mix Barrington, Tanmoy Chakraborty, Samir Datta & Sambuddha Roy (2009): *Planar and Grid Graph Reachability Problems*. *Theory of Computing Systems* 45(4), pp. 675–723, doi:10.1007/s00224-009-9172-z.

[6] Carme Álvarez & Birgit Jenner (1993): *A very hard log-space counting class*. *Theoretical Computer Science* 107(1), pp. 3–30, doi:10.1016/0304-3975(93)90252-O.

[7] Sanjeev Arora & Boaz Barak (2009): *Computational Complexity: A Modern Approach*. 978-0-511-53381-5, Cambridge University Press, doi:10.1017/CBO9780511804090.

[8] David A. Mix Barrington, Neil Immerman & Howard Straubing (1990): *On Uniformity within NC^1*. *Journal of Computer and System Sciences* 41(3), pp. 274–306, doi:10.1016/0022-0000(90)90022-D.

[9] Ronald V. Book & Ker-I Ko (1988): *On Sets Truth-Table Reducible to Sparse Sets*. *SIAM Journal of Computing* 17(5), pp. 903–919, doi:10.1137/0217056.

[10] Harry Buhrman, Edith Hemaspaandra & Luc Longpre (1995): *SPARSE Reduces Conjunctively to TALLY*. *SIAM Journal of Computing* 24, pp. 673–681, doi:10.1137/0224044.

[11] Ashok K. Chandra, Larry J. Stockmeyer & Uzi Vishkin (1984): *Constant Depth Reducibility*. SIAM Journal of Computing 13(2), pp. 423–439, doi:10.1137/0213028.

[12] Merrick L. Furst, James B. Saxe & Michael Sipser (1984): *Parity, circuits and the polynomial-time hierarchy*. Theory of Computing Systems (formerly Mathematical Systems Theory) 17(1), pp. 13–27, doi:10.1007/BF01744431.

[13] Leslie M. Goldschlager (1977): *The monotone and planar circuit value problems are log space complete for P*. SIGACT News 9(2), pp. 25–29, doi:10.1145/1008354.1008356.

[14] Raymand Greenlaw, H. James Hoover & Walter L. Ruzzo (1995): *Limits to parallel computation: P-completeness Theory*. Oxford University Press, New York, Oxford.

[15] Kristoffer Arnsfelt Hansen & Michal Koucký (2010): *A New Characterization of* ACC^0 *and Probabilistic* CC^0. Computational Complexity 19(2), pp. 211–234, doi:10.1007/s00037-010-0287-z.

[16] Neil Immerman (1987): *Languages that capture complexity classes*. SIAM Journal of Computing 16(4), pp. 760–778, doi:10.1137/0216051.

[17] Neil Immerman (1988): *Nondeterministic Space is Closed Under Complementation*. SIAM Journal of Computing 17(5), pp. 935–938, doi:10.1137/0217058.

[18] Neil Immerman (1999): *Descriptive Complexity*. Springer, doi:10.1007/978-1-4612-0539-5.

[19] Ker-I Ko (1989): *Distinguishing conjunctive and disjunctive reducibilities by sparse sets*. Information and Computation 81(1), pp. 62–87, doi:10.1016/0890-5401(89)90029-1.

[20] Richard E. Ladner, Nancy A. Lynch & Alan L. Selman (1975): *A comparison of polynomial time reducibilities*. Theoretical Computer Science 1(2), pp. 103–123, doi:10.1016/0304-3975(75)90016-X.

[21] Niall Murphy & Damien Woods (2010): *Uniformity conditions in natural computing*. In: The 16th International Conference on DNA Computing and Molecular Programming (DNA 16), Preproceedings, pp. 109–120. HKUST, Hong Kong, China.

[22] Christos H. Papadimitriou (1993): *Computational Complexity*. Addison Wesley.

[23] Mario J. Pérez-Jiménez, Agustín Riscos-Núñez, Alvaro Romero–Jiménez & Damien Woods (2009): *Handbook of Membrane systems*, chapter 12: Complexity – Membrane Division, Membrane Creation. Oxford University Press.

[24] Gheorghe Păun (2005): *Further twenty six open problems in membrane computing*. In: Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain), Fénix Editoria, pp. 249–262.

[25] Róbert Szelepcsényi (1988): *The Method of Forced Enumeration for Nondeterministic Automata*. Acta Informatica 26(3), pp. 279–284, doi:10.1007/BF00299636.

[26] Heribert Vollmer (1999): *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, doi:10.1007/978-3-662-03927-4.