Enumeration Classes Defined by Circuits

NADIA CREIGNOU, Aix Marseille Univ, CNRS, LIS, France
ARNAUD DURAND, Université Paris Cité, CNRS, IMJ-PRG, France
HERIBERT VOLLMER, Leibniz Universität, Germany

We refine the complexity landscape for enumeration problems by introducing very low classes defined by using Boolean circuits as enumerators. We locate well-known enumeration problems, e.g., from graph theory, Gray code enumeration, and propositional satisfiability in our classes. In this way we obtain a framework to distinguish between the complexity of different problems known to be in DelayP, for which a formal way of comparison was not possible to this day.

CCS Concepts: • **Theory of computation** → Complexity classes; Computational complexity and cryptography; Circuit complexity.

Additional Key Words and Phrases: Computational complexity, enumeration problem, Boolean circuit

1 INTRODUCTION

In computational complexity theory, most often decision problems are studied that ask for the existence of a solution to some problem instance, e.g., a satisfying assignment of a given propositional formula. In contrast, enumeration problems ask for a list of all solutions, e.g., all satisfying assignments. In many application areas these are the more "natural" kind of problems—let us just mention database queries, web search, diagnosis, data mining, bioinformatics, etc.

The notion of *tractability* for enumeration problems requires a new approach, simply because there may be a large number of solutions, exponential in the input size. Widely studied is the class DelayP ("polynomial delay"), containing all enumeration problems where, for a given instance x, (i) the time to compute the first solution, (ii) the time between producing any two consecutive solutions, and (iii) the time to detect that no further solution exists, are all polynomially bounded in the length of x. Also the class IncP ("incremental polynomial time"), where we allow the time to produce the next solution and to signal that no further solution exists to grow by a polynomial bounded in the size of the input *plus* the number of already computed solutions. These classes were introduced in 1988 in [17], and since then, an immense number of membership results have been obtained. Recently, also intractable enumeration problems have received some attention. Reducibilities, a completeness notion and a hierarchy of intractable enumeration problems, analogous to the well-known polynomial hierarchy, were defined and studied in [10].

In this paper we will look for notions of tractability for enumeration stricter than DelayP. More specifically, we will introduce a refinement of DelayP based on the computation model of Boolean circuits. The main new class in our framework is the class $\mathrm{Del}\cdot\mathrm{AC}^0$. An enumeration problem belongs to this class if there is a family of AC^0 circuits, i.e., a family of Boolean circuits of constant depth and polynomial size with unbounded fan-in gates, that (i) given the input computes the first solution, (ii) given input and a solution computes the next solution (in any fixed order of solutions), and (iii) given input and the last solution, signals that no further solution exists. Still using AC^0 circuits we then consider extended classes by allowing

- precomputation of different complexity (typically, polynomial time precomputation), and/or
- memory to be passed on from the computation of one solution to the next (a constant, a polynomial or an unbounded number of bits)

Authors' addresses: Nadia Creignou, Aix Marseille Univ, CNRS, LIS, Marseille, France, Nadia.Creignou@lis-lab.fr; Arnaud Durand, Université Paris Cité, CNRS, IMJ-PRG, Paris, France, durand@imj-prg.fr; Heribert Vollmer, Leibniz Universität, Hannover, Germany, vollmer@thi.uni-hannover.de.

By this, we obtain a hierarchy of classes, most of which are subclasses of DelayP, as shown in Fig. 1 on page 7.

The main motivation behind our work is the wish to be able to compare the complexity of different tractable enumeration problems by classifying them in a fine hierarchy within DelayP, and to obtain lower bounds for enumeration tasks. From different application areas such as graph problems, Gray code enumeration and satisfiability, we identify natural problems, all belonging to DelayP, some of which can be enumerated in $\mathrm{Del}\cdot\mathrm{AC}^0$, some cannot, but allowing precomputation or a certain number of bits of auxiliary memory they can. We would like to mention in particular the case of enumeration for satisfiability of Krom (i.e., 2-CNF) formulas. While it is known that counting satisfying assignments for formulas from this fragment of propositional logic is #P-complete [23], we exhibit a $\mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$ algorithm (i.e., $\mathrm{Del}\cdot\mathrm{AC}^0$ with polynomial time precomputation but no memory), for enumeration, thus placing the problem in one of the lowest class in our framework. This means that surprisingly satisfying assignments of Krom formulas can be enumerated very efficiently (only AC^0 is needed to produce one solution after the other) after a polynomial time precomputation before producing the first solution.

Building on well-known lower bounds (in particular for the parity function [1, 15]) we prove (unconditional) separations among (some of) our classes and strict containment in DelayP, and building on well-known completeness results we obtain conditional separations, leading to the inclusions and non-inclusions depicted in Fig. 1.

Another refinement of DelayP that has received considerable attention in the past, in particular in the database community, is the class $\mathsf{CD}\circ\mathsf{lin}$ of problems that can be enumerated on RAMs with constant delay after linear time preprocessing [12] (see also the surveys [11, 21]). A consequence of the widely believed assumption that Boolean matrix multiplication cannot be computed in time linear in the number m of non-zero entries of the matrices A, B and AB (the so called BMM conjecture, see e.g. [5]), is that enumerating the one-entries in AB is not in $\mathsf{CD}\circ\mathsf{lin}$ [3], but we will see that it is in $\mathrm{Del}\cdot\mathrm{AC}^0$. On the other hand, the familiar lower bound for parity [1, 15] easily leads to an enumeration problem not in $\mathrm{Del}\cdot\mathrm{AC}^0$ but in $\mathsf{CD}\circ\mathsf{lin}$; hence we see that $\mathsf{CD}\circ\mathsf{lin}$ and $\mathrm{Del}\cdot\mathrm{AC}^0$ are incomparable classes; thus our approach provides a novel way to refine polynomial delay (see Section 3.3).

This paper is organised as follows. After some preliminaries, we introduce our new classes in Sect. 3. In Sect. 4 we present a number of upper and lower bounds for example enumeration problems from graph theory, Gray code enumeration and propositional satisfiability. Depending whether we allow or disallow precomputation steps, we obtain further conditional or unconditional separation results between classes in Sect. 5. Finally we conclude with a number of open problems.

This work is an extension of the conference article [8].

2 PRELIMINARIES

Since our main computational model will be Boolean circuits, we fix the alphabet $\Sigma = \{0, 1\}$, and use this alphabet to encode graphs, formulas, etc., as usual. Any reasonable encoding will do for all of our results.

Let $R \subseteq \Sigma^* \times \Sigma^*$ be a computable predicate. We say that R is polynomially balanced, if there is a polynomial p such that for all pairs $(x, y) \in R$, we have $|y| \leq p(|x|)$. Given such a predicate, we now define the enumeration problem associated to R as follows.

ENUM·R
Input: $x \in \Sigma^*$
Output: an enumeration of elements in $\mathrm{Sol}_R(x) = \{y : R(x, y)\}$

We require that R is computable (and polynomially balanced) but do not make any complexity assumptions on R. In the enumeration context, it is sometimes stipulated that R is polynomial-time checkable, i.e., membership of (x, y) in R is decidable in time polynomial in the length of the pair [6, 22]. Generally, we do not require this, but we will come back to this point later.

We assume basic familiarity of the reader with the model of Boolean circuits, see, e.g., [7, 25]. We use AC^0 to denote the class of languages that can be decided by uniform families of Boolean circuits of polynomial size and constant depth with gates of unbounded fan-in. The class of functions computed by such circuit families is denoted by FAC^0, and for simplicity often again by AC^0. The notation for the corresponding class of languages/functions defined by uniform families of circuits of polynomial size and logarithmic depth with gates of bounded fan-in is NC^1.

The actual type of uniformity used is of no importance for the results of the present paper. However, for concreteness, all circuit classes in this paper are assumed to be uniform using the "standard" uniformity condition, i. e., DLOGTIME-uniformity/$\mathrm{U_E}$-uniformity [4]; the interested reader may also consult the textbook [25].

It should be noted that every language in NP admits an AC^0 verifier, that is, for every $A \subseteq \{0,1\}^*$, $A \in \mathsf{NP}$, there is a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ checkable in AC^0 and a polynomial p such that for all x, $x \in A$ iff there is some y, $|y| \leq p(|x|)$, such that $(x, y) \in R$. Thus, enumerating $\mathrm{Sol}_R(x)$ is trivial, since given x we can simply loop over all candidates $y \in \{0,1\}^{p(|x|)}$ and check if $(x, y) \in R$. However, this enumeration algorithm does not necessarily have polynomial delay, highlighting the motivation for considering *delay* as an important resource in the area of enumeration problems.

3 DELAY CLASSES WITH CIRCUIT GENERATORS

In this section we present the formal definition of our new enumeration classes. As we already said, we will restrict our definition to usual delay classes; classes with incremental delay can be defined analogously.

The main idea is that the generation of a *next* solution will be done by a circuit from a family; in the examples and lower and upper bounds in the upcoming sections, these families are usually of low complexity like AC^0 or NC^1. The generator will receive the original input word plus the previous solution. Parameters in the definition will be first the complexity of any precomputation before the first solution is output, and second the amount of information passed from the generation of one solution to the next.

3.1 Delay Classes with no Memory

For a family $C = (C_n)_{n\in\mathbb{N}}$ of Boolean circuits, circuit C_i will be the circuit in the family with i input gates. When the length of the circuit input is clear from the context, we will usually simply write $C_{|\cdot|}$ to refer to the circuit with appropriate number of input gates. In the subsequent definitions we use $\mathcal{K}$ to denote a complexity classes defined by families of Boolean circuits obeying certain complexity restrictions. Examples are the above mentioned cases $\mathcal{K} = \mathsf{AC}^0$ or $\mathcal{K} = \mathsf{NC}^1$, but any circuit complexity class will do. Our definitions make sense both for uniform and for non-uniform classes.

Definition 3.1 ($\mathcal{K}$-delay). Let R be a polynomially balanced predicate. The enumeration problem Enum·R is in Del·$\mathcal{K}$ if there exists a family of $\mathcal{K}$-circuits $C = (C_n)_{n\in\mathbb{N}}$ such that, for all inputs x, there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:

- $C_{|\cdot|}(x) = y_1 \in \mathrm{Sol}_R(x)$,
- for all $i < k$: $C_{|\cdot|}(x, y_i) = y_{i+1} \in \mathrm{Sol}_R(x)$
- $C_{|\cdot|}(x, y_k) = y_k$

Note that by the last requirement, the circuit family signals there is no further solution if the input solution is given again as output. Moreover, we point out that, in the definition above, if x is an input and $y \in \mathrm{Sol}_R(x)$, then $C_{|x|+|y|}$ produces a $z \in \mathrm{Sol}_R(x)$. However, if $y \notin \mathrm{Sol}_R(x)$, nothing is specified about the output z.

Next we consider classes where a precomputation before outputting the first solution is allowed. The ressource bounds of the precomputation are specified by an arbitrary complexity class.

Definition 3.2 ($\mathcal{K}$-delay with T-precomputation). Let R be a polynomially balanced predicate and T be a complexity class. The enumeration problem $\textsc{Enum}{\cdot}R$ is in $\mathrm{Del}_T{\cdot}\mathcal{K}$ if there exists an algorithm M working with resource T and a family of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ such that, for all input x there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:

- M computes some value x^*, i.e., $M(x) = x^*$
- $C_{|\cdot|}(x^*) = y_1 \in \mathrm{Sol}_R(x)$,
- for all $i < k$: $C_{|\cdot|}(x^*, y_i) = y_{i+1} \in \mathrm{Sol}_R(x)$
- $C_{|\cdot|}(x^*, y_k) = y_k$

3.2 Delay Classes with Memory

Extending the above model, we now allow each circuit to produce slightly more than the next solution. These additional information is then passed as extra input to the computation of the next solution, in other words, it can serve as an auxiliary memory.

Definition 3.3 ($\mathcal{K}$-delay with auxiliary memory). Let R be a polynomially balanced predicate. The enumeration problem $\textsc{Enum}{\cdot}R$ is in $\mathrm{Del}^*{\cdot}\mathcal{K}$ if there exist two families of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ such that, for all input x there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:

- $C_{|\cdot|}(x) = y_1^*$ and $D_{|\cdot|}(y_1^*) = y_1 \in \mathrm{Sol}_R(x)$,
- for all $i < k$: $C_{|\cdot|}(x, y_i^*) = y_{i+1}^*$ and $D_{|\cdot|}(y_{i+1}^*) = y_{i+1} \in \mathrm{Sol}_R(x)$,
- $C_{|\cdot|}(x, y_k^*) = y_k^*$,
- for $1 \le i \le k$, y_i is a prefix of y_i^*; i. e., the information y_i^* passed on to the next round consists of the previous solution y_i plus some additional information.

When there exists a polynomial $p \in \mathbb{N}[x]$ such that $|y_i^*| \le p(|x|)$, for all $i \le k$, the class is called $\mathrm{Del}^{\mathrm{p}}{\cdot}\mathcal{K}$, *$\mathcal{K}$-delay with polynomial auxiliary memory*. When there exists a constant $c \in \mathbb{N}$ such that $|y_i^*| \le |y_i| + c$, for all $i \le k$, the class is called $\mathrm{Del}^{\mathrm{c}}{\cdot}\mathcal{K}$, *$\mathcal{K}$-delay with constant auxiliary memory*.

The idea is that the y_i^* will contain the previous solution plus the additional memory. Hence the superscript "c" indicates a bounded auxiliary memory size.

By abuse of expression, we will sometimes say that a problem in some of these classes above can be enumerated with a delay in $\mathcal{K}$ or with a $\mathcal{K}$-delay.

Also in the case of memory, we allow possibly precomputation before the first output is made:

Definition 3.4 ($\mathcal{K}$-delay with T-precomputation and auxiliary memory). Let R be a polynomially balanced predicate and T be a complexity class. The enumeration problem $\textsc{Enum}{\cdot}R$ is in $\mathrm{Del}_T^*{\cdot}\mathcal{K}$ if there exists an algorithm M working with resource T and two families of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$, $\mathcal{D} = (D_n)_{n\in\mathbb{N}}$ such that, for all input x there is an enumeration $y_1, ..., y_k$ of $\mathrm{Sol}_R(x)$ and:

- M computes some value x^*, i.e., $M(x) = x^*$,
- $C_{|\cdot|}(x^*) = y_1^*$ and $D_{|\cdot|}(y_1^*) = y_1 \in \mathrm{Sol}_R(x)$,
- for all $i < k$: $C_{|\cdot|}(x^*, y_i^*) = y_{i+1}^*$ and $D_{|\cdot|}(y_{i+1}^*) = y_{i+1} \in \mathrm{Sol}_R(x)$,
- $C_{|\cdot|}(x^*, y_k^*) = y_k^*$,
- for $1 \le i \le k$, y_i is a prefix of y_i^*.

When there exists a polynomial $p \in \mathbb{N}[x]$ such that $|y_i^*| \leq p(|x|)$, for all $i \leq k$, the class is called $\mathrm{Del}_T^{\mathrm{p}} \cdot \mathcal{K}$, *$\mathcal{K}$-delay with T-precomputation and polynomial auxiliary memory*. When there exists a constant $c \in \mathbb{N}$ such that $|y_i^*| \leq |y_i| + c$, for all $i \leq k$, the class is called $\mathrm{Del}_T^{\mathrm{c}} \cdot \mathcal{K}$, *$\mathcal{K}$-delay with T-precomputation and constant auxiliary memory.*

Remark 1. *In this paper, we focus on four different levels of memory size (none, constant, polynomial and unbounded) and two modes of precomputation (none and polynomial time). It would make sense to consider alternative the memory model (using a logarithmically bounded number of bits, for example) and precomputation (linear time as it is used in the* CD∘lin *class - see [12] and next section).*

3.3 Relation to Known Enumeration Class

All classes we consider in this paper with at most polynomial precomputation and polynomial memory are subclasses of the well-known classe DelayP, even if we allow our circuit to be of arbitrary depth (but polynomial size).

Theorem 3.5. *If $\mathcal{K}, T \subseteq \mathsf{P}$, then $\mathrm{Del}_T^{\mathrm{p}} \cdot \mathcal{K} \subseteq$ DelayP .*

Let us briefly clarify the relation between our classes and the class CD∘lin of enumeration problems that have a constant delay on a RAM after linear-time precomputation. This class was introduced in [12].

The algorithmic problem, given a graph with n vertices, of enumerating all pairs of vertices that are connected by a path of length 2 has only a quadratic number of solutions and is trivially in $\mathrm{Del}\cdot\mathsf{AC}^0$: one first compute the length n^2 word w representing the adjacency matrix of the distance 2 vertices (this can be done in depth 2); from this, enumerating the suitable pairs of vertices requires to use an AC^0 routine which, given the last solution (i, j), find the position $p_{i,j}$ in w, finds the next position $p_{i',j'}$ in w with a 1 and generate the binary representation of positions i' and j'. Since it is essentially the same as Boolean matrix multiplication, it is not in CD∘lin, assuming the before mentioned BMM hypothesis.

On the other hand, note that the enumeration problem Enum-Parity, given as input a sequence of bits with the solution set consisting only of one solution, the parity of the input, is not in $\mathrm{Del}\cdot\mathsf{AC}^0$, since the parity function is not in AC^0 [1, 15]. However, since Parity can be computed in linear time, Enum-Parity is trivially in CD∘lin.

As shown below, the computation of a constant number of time steps of a RAM can be simulated by AC^0 circuits. Hence if we add linear precomputation and polynomial memory to save the configuration of the RAM, we obtain an upper bound for CD∘lin. To summarize:

Theorem 3.6. *Assuming the BMM hypothesis, the classes* $\mathrm{Del}\cdot\mathsf{AC}^0$ *and* CD∘lin *are incomparable, and* $\mathsf{CD}\circ\mathsf{lin} \subsetneq \mathrm{Del}_{lin}^{\mathrm{p}} \cdot \mathsf{AC}^0$.

Proof. Only the inclusion $\mathsf{CD}\circ\mathsf{lin} \subseteq \mathrm{Del}_{lin}^{\mathrm{p}} \cdot \mathsf{AC}^0$ remains to be proved. In the CD∘lin context, one considers the following RAM model. It has a series of different purpose registers : N, the input size register; $I[j]$, $j \geq 0$ the read-only input registers; $O[j]$, $j \geq 0$ the write-only output registers; $R[j]$, $j \geq 0$ the memory registers. We also use two special registers A and B to retrieve or update information from the memory. At any moment of the computation, the input registers $I[0], ..., I[N-1]$ contain the input. The model considers uniform cost for operations but any number manipulated during the computation (register values and addresses) is bounded by $c \times N$, for some fixed $c \in \mathbb{N}$ (actually, for our purpose here, the weaker constraint that every number is bounded in length by some $c \times \log N$, hence polynomially in N in values, would be sufficient). So, after a linear time precomputation and during all intermediate constant time steps of the enumeration process, no register contains an integer or is at an address that is beyond, say, $c \times N$. It means that at each step, the meaningful part of the configuration of the RAM is the sequence:

$$N, I[0], ..., I[N-1], A, B, R[0], ..., R[c \cdot N - 1], O[0], ..., O[c \cdot N - 1]$$

where $N, I[0], ..., I[N-1]$ is a representation of input x of the enumeration problem. In other words, no other registers than those listed here will be accessed or modified.

The RAM will use $+, -$ as basic operations together with constants 0, 1, etc. Each program is an index set of instructions l_0,..., l_r. The instruction set of the RAM is classical: it contains affectation between registers (including by constant values), such as $A := B$ or $A := N$; operations $A := A + B$, $A := A - B$; indirect addressing: $R[A] := B, A := R[A], A := I[A], O[A] := B$; conditional goto statements: *if* $A = 0$ *then goto* l_0 *else goto* l_1; some halting instruction: *Output*.

Because of the existence of conditional statements, the possible labels that may appear during a constant time computation form a tree of root I_0 (the first instruction) and of constant depth. The actual path of the computation is determined by the value of the test $A = 0$.

The simulation of a constant number d of RAM instructions by a polynomial size constant depth circuit works as follows. Let x^* be the input of the circuit after precomputation i.e., x^* for the circuit is the sequence $N, I[0], ..., I[N-1], A, B, R[0], ..., R[c \cdot N - 1], O[0], ..., O[c \cdot N - 1]$ where each register is encoded over, say, $c \cdot \log N$ bits. Such a sequence describes a configuration of the RAM. Each instruction transforms a configuration into another one.

If the current instruction of the RAM is $A := A + B$, then we use a constant depth sub-circuit for addition that works on the adequate part of the input. The values of the output gates of the sub-circuit will hence be $N, I[0], ..., I[N-1], A + B, B, R[0], ..., R[c \cdot N - 1], O[0], ..., O[c \cdot N - 1]$. If the current instruction is an affectation between registers such as $A := B$, $A := N$ or $A := i$, for some constant i, the simulation is similar.

If the current instruction of the RAM is an indirect addressing instruction such as $R[A] := B$, we plug the sub-circuit that selects, in constant depth and polynomial size, which of registers $R[0], ..., R[c \cdot N - 1]$ has index equal to A, say $R[i]$, and, once found replace $R[i]$ by the value of B. The rest of the configuration is unchanged.

If the current instruction of the RAM is *if* $A = 0$ *then goto* l_0 *else goto* l_1 then we plug in parallel the sub-circuit for instruction l_0 and for instruction l_1 and fork the simulation to continue from these two branches in parallel.

After d steps, it remains to determine which of the branches is the correct one. Let $t_1, ..., t_e$, $e \leq d$ be the levels of the circuit where a conditional statement was simulated and $A_1, ..., A_{t_e}$ be the value of A at these instants. We know how to select the correct branch by just sequentially testing whether $A_1 = 0$,..., $A_{t_e} = 0$ and output the configuration that corresponds to the satisfying tests.

The *Output* instruction of the RAM just mark the end of the sub-circuit simulation and forces the output of an intermediate solution. □

In the paper, as illustrated in Figure 1, we will prove inclusions and separations among the classes described in this section.

4 LOWER AND UPPER BOUNDS FOR NATURAL PROBLEMS

In this section we show that many natural problems, ranging from graph and hypergraph problems, enumeration of Gray codes and satisfiability problems lie in our circuit classes.

4.1 Graph Problems

A *finite graph* $G = (V, E)$ is given by a finite set of vertices V and a set E of pairs of vertices called edges. Two vertices in a graph are *adjacent* if $\{x, y\}$ is an edge. A vertex t is *reachable* from s if there exists a sequence of adjacent vertices that starts with s and ends with t. A *hypergraph* $H = (V, \mathcal{E})$ is a generalisation of a graph where V denotes the set of vertices and $\mathcal{E} \subseteq 2^V$ is a collection of sets

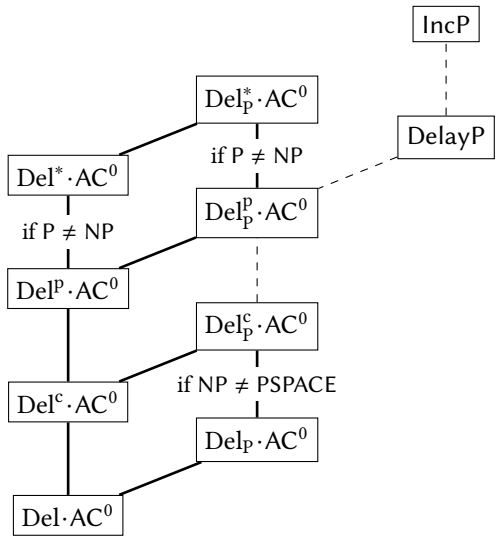

Fig. 1. Diagram of the classes. Bold lines denote strict inclusions.

of vertices called *hyperedges*. Thus, a graph is a particular hypergraph where all hyperedges are of size two. A *transversal* of a hypergraph is a set of vertices that intersects all hyperedges, i.e., is a subset $I \subseteq V$ such that, for all $S \in \mathcal{E}$, $S \cap I \neq \emptyset$.

We first consider the enumeration problem associated with the notion of reachability in a graph. Below, a given finite graph $G = (V, E)$ with $|V| = n$ will be represented as a $\{0,1\}$ word of length n^2 representing its adjacency matrix.

Enum-Reach
Input: a graph $G = (V, E)$, $s \in V$
Output: an enumeration of vertices reachable from s

Theorem 4.1. Enum-Reach $\in \mathrm{Del}^{\mathrm{p}}\cdot\mathrm{AC}^0$

Proof. Let M^0 be the all-zero square matrix and M^1 the adjacency matrix of the graph G. By means of multiplication of Boolean matrices we successively compute M^k, $1 \leq k \leq n$, the matrix such that $M^k[i,j] = 1$ if there exists a path of length less than or equal to k between the two vertices i and j. Then we output all j such that $M^k[s,j] = 1$ while $M^{k-1}[s,j] = 0$. We do this until we reach convergence, i.e., until $M^k = M^{k-1}$. This can be done in AC^0 if we pass the last two computed matrices M^k and M^{k-1} as additional information to the last output vertex from one step to the next. Therefore, polynomial memory is required.

□

A finite hypergraph with $|V| = n$ can be represented as a sequence of words of length n over $\{0,1\}$ listing its hyperedges. Let us consider the following enumeration problem.

Enum-Transversal
Input: a hypergraph $H = (V, \mathcal{E})$
Output: an enumeration of all transversals of H

Enum-Transversal is known to be in DelayP. The exact complexity of enumerating minimal transversals (i.e., transversals which are minimal for inclusion), in particular whether it belongs to

DelayP or even IncP, is a well-known open problem [13, 14]. Let us now turn to the enumeration of all transversals in our context.

Theorem 4.2. Enum-Transversal $\in$ Del$\cdot$AC^0.

Proof. Let H be a hypergraph and $\mathcal{E}$ be its set of hyperedges over a set of n vertices. Every binary word $y = y_1 \cdots y_n \in \{0,1\}^n$ can be interpreted as a subset of vertices. We propose an algorithm that enumerates each of these words that corresponds to a transversal of H in lexicographical order with the convention $1 < 0$. The algorithm is as follows:

- As a first step output $1 \ldots 1$, the trivial solution.
- Let H be the input and y be the last output solution.
  - For each prefix $y_1 \ldots y_i$ of y with $y_i = 1$ and $i \leq n$ consider the word of length n, $z^i = y_1 \ldots y_{i-1} 0 1 \ldots 1$.
  - Check whether at least one of these words z^i is a transversal of H.
  - If yes select the one with the longest common prefix with y, that is the transversal z^i with the largest i and output it as the next solution.
  - Else stop.

First we prove that the algorithm is correct. The transversal that is the successor of y in our lexicographical order (where $1 < 0$), if it exists, has a common prefix with y, then a bit flipped from 1 to 0, and finally is completed by 1's only. Indeed, a successor of y necessarily starts this way, and by monotonicity the first extension of such a prefix into a solution is the one completed by 1's only. As a consequence our algorithm explores all possible candidates and selects the next transversal in the lexicographical order.

Now let us prove that this is an AC^0-delay enumeration algorithm that does not require memory. The main observation is that one can check with an AC^0 circuit whether a binary word corresponds to a transversal of H. Now, for each i we can use a sub-circuit, which on input (H, y) checks whether $y_i = 1$ and if yes whether z^i is a transversal of H. This circuit can output $(z^i, 1)$ if both tests are positive, and $(y_i, 0)$ otherwise. All these sub-circuits can be wired in parallel. Finally it suffices to use a selector to output z^i with the largest i for which $(z^i, 1)$ is output at the previous step. Such a selector can be implemented by an AC^0 circuit. □

It is then easy to show (in a similar way) that enumeration of all dominating sets of a graph can be done in Del$\cdot$AC^0.

4.2 Gray Code

Given $n \in \mathbb{N}$, a Gray n-code is a ranked list of elements of Σ^n such that between two successive words x, y there exists only one bit such that $x_i \neq y_i$. Since we deal with Boolean circuits, we have to fix $\Sigma = \{0,1\}$, but Gray codes are defined for arbitrary alphabets.

Throughout this paper, when we refer to Gray codes, we mean the binary reflected Gray code of length n, denoted G^n, which is made of 2^n words: $G^n = [G^n_0, G^n_1, \ldots, G^n_{2^n-1}]$. It is defined recursively as follows: $G^1 = [0, 1]$ and, for $n \geq 1$

$$G^n = [0G^{n-1}_0, 0G^{n-1}_1, \ldots, 0G^{n-1}_{2^{n-1}-1}, 1G^{n-1}_{2^{n-1}-1}, \ldots, 1G^{n-1}_1, 1G^{n-1}_0].$$

As an example let us consider the list of pairs $(rank, word)$ for $n = 4$:

0 :	0000	8 :	1100
1 :	0001	9 :	1101
2 :	0011	10 :	1111
3 :	0010	11 :	1110
4 :	0110	12 :	1010
5 :	0111	13 :	1011
6 :	0101	14 :	1001
7 :	0100	15 :	1000

Given n and $r < 2^n$, let $b_{n-1} \cdots b_1 b_0$ be the binary decomposition of r and $G^n_r = a_{n-1} \cdots a_1 a_0 \in \Sigma^n$ be the rth word in the binary reflected code of length n. It is well-known that, for all $j = 0, ..., n-1$,

$$b_j = \sum_{i=j}^{n-1} a_i \bmod 2 \text{ and } a_j = (b_j + b_{j+1}) \bmod 2,$$

setting $b_n = 0$, i.e., $a_{n-1} = b_{n-1}$. Hence computing the rank of a word in the binary reflected code amounts to be able to compute parity. On the other side, computing the word from its rank can easily be done by a circuit.

While it is trivial to enumerate all words of length n in arbitrary or lexicographical order, the computational complexity is higher for Gray code order.

Theorem 4.3. *(1) Given 1^n, enumerating all words of length n even in lexicographical ordering is in* $\mathrm{Del}\cdot\mathrm{AC}^0$

(2) Given 1^n, enumerating all words of length n in a Gray code order is in $\mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0 \setminus \mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$

The role of the memory bit and its apparent necessity in the enumeration of binary reflected Gray code was already raised by Knuth in [18, p. 286].

Proof. The proof of the first item is immediate.

A classical method to enumerate Gray code of length n is the following [19].

- Step 0 : produce the word $0 \cdots 0$ of length n.
- Step $2k + 1$: switch the bit at position 0.
- Step $2k + 2$: find minimal position i where there is a 1 and switch bit at position $i + 1$.

This method can be turned into an AC^0-delay enumeration without precomputation using one bit of memory (to keep trace if the step is an even or odd one all along the computation). This proves the membership in $\mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$.

For the lower bound, suppose $C = (C_n)_{n\in\mathbb{N}}$ is an AC^0 circuit family enumerating the Gray code of length n after polynomial time precomputation produced by machine M. We will describe how to use C to construct an AC^0 family computing the parity function, contradicting the well-known lower bound given by [1, 15].

Given an arbitrary word $w = w_{n-1} \dots w_0$ of length n, we want to compute its parity $\left(\sum_{i=0}^{n-1} w_i\right) \bmod 2$. Let $x^* = M(1^n)$. Then, w will appear as a solution somewhere in the enumeration defined by C. Let w' be the next word after w. There exists r such that $G^n_r = w$ and $G^n_{r+1} = w'$. By comparing w and w', one can decide which transformation step has been applied to w to obtain w' and thus if r is odd or even. Note that the parity of w is 1 if and only if r is odd. Hence, one can compute parity by a constant depth circuit operating as follows:

Input w:
  $n := |w|$;

$x^* := M(1^n)$;
$w' := C_{|\cdot|}(x^*, w)$;
if last bits of w and w' differ then $v := 1$ else $v := 0$;
output v.

Note that the computation of x^* does not depend on w but only on the length of w; hence x^* can be hardwired into the circuit family, which, since M runs in polynomial time, will then be P-uniform. But we know from [1, 15] that parity cannot even be computed by non-uniform AC^0 circuit families. □

Also, given a rank or a first word, to enumerate all words of higher Gray code rank are interesting computational problems. Let us first consider these problems where the enumeration can be done in an arbitrary order.

Enum-Gray-Rank
Input: a binary word r of length n interpreted as an integer in $[0, 2^n[$
Output: an enumeration of words of G^n that are of rank at least r

Enum-Gray-Word
Input: a word x of length n
Output: an enumeration of words of G^n, that are of rank at least the rank of x

It turns out that for those problems where the order of solutions is not important, an efficient enumeration is possible [1]:

Theorem 4.4. *(1)* Enum-Gray-Rank $\in \mathrm{Del}\cdot\mathrm{AC}^0$
(2) Enum-Gray-Word $\in \mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$

Proof. (1) Let r be an input of Enum·Gray-Rank. From r one can easily compute $y_0 = G^n_r$ and $y_1 = G^n_{r+1}$ but also $z_0 = 10\cdots01$ and $z_1 = 10\cdots00$ the last two words of the binary reflected Gray code. Suppose first that r is even. The enumeration step is the following:

- As a first step, output y_0.
- Let r be the input and y be the last output solution.
  - Compute (again) z_0 and y_1
  - If $y = y_1$, stop
  - If $y = z_0$, then output z_1
  - Else,
    (i) switch the bit of y at position 0, then
    (ii) find the minimal position i where there is a 1, and switch bit at position $i+1$.
    Output this word as the new solution.

The above process does not require memory. Starting from y_0, it will start enumerating binary words of rank $G^n_{r+2}, G^n_{r+4}, G^n_{r+6}$... until it reaches $z_0 = 10\cdots01 = G^n_{2^n-2}$. It then outputs $z_1 = 10\cdots00 = G^n_{2^n-1}$ and, applying the same rules, enumerates successively $G^n_{2^n-3}, G^n_{2^n-5}$, ... until $y_1 = G^n_{r+1}$.
If r is odd we proceed in a similar way, exchanging the rôles of z_0 and z_1, and swapping the order of the switches (i) and (ii) in the process.

(2) The enumeration for Enum-Gray-Word with one memory bit proceeds along the same lines as the enumeration of words of length n in Gray code order (see the proof of Theorem

[1] In the conference version of this paper [8] we claimed that Enum-Gray-Word $\in \mathrm{Del}\cdot\mathrm{AC}^0$ but we were not able to produce a proof that it is or that it is not in this class.

4.3) in starting from the bottom, i.e., from $10\ldots0$, and in stopping when the enumeration reaches x.

□

We next turn to those versions of the above problems, where we require that solutions are given one after the other in Gray code order. For each of them, the computational complexity is provably higher than in the above cases.

Enum-Gray-Rank$_{ord}$
Input: a binary word r of length n interpreted as an integer in $[0, 2^n[$
Output: an enumeration of words of G^n in increasing number of ranks starting from rank r

Enum-Gray-Word$_{ord}$
Input: a word x of length n
Output: an enumeration of words of G^n in Gray code order that are of rank at least the rank of x

Theorem 4.5. *(1)* Enum-Gray-Rank$_{ord} \in \mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0 \setminus \mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$.
(2) Enum-Gray-Word$_{ord}$ *is in the class* $\mathrm{Del}^{\mathrm{c}}_{\mathrm{P}}\cdot\mathrm{AC}^0$, *but neither in* $\mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$ *nor* $\mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$.

Proof. (1) For the upper bound, we use the method described in Theorem 4.3. Since the starting word is given by its rank r, one needs to slightly modify the approach above by first computing G^n_r, check what the parity of its last bit is to determine what kind of step needs to be performed first. Then we go on as in the above proof.
Suppose Enum·Gray-Rank$_{ord}$ is in $\mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$. Then by choosing $x = 0^n$ we can enumerate all words of length n in a Gray code order in $\mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$, which, by Theorem 4.3, is not possible.
(2) For the membership in $\mathrm{Del}^{\mathrm{c}}_{\mathrm{P}}\cdot\mathrm{AC}^0$ one just computes the rank r of x during the precomputation and uses the preceding theorem.
The first lower bound is proven exactly as above.
The second lower bound follows by an easy modification: Suppose Enum·Gray-Word$_{ord} \in \mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$. Given word w of length n, we can compute its parity in AC^0 as follows: Start the enumeration of G^n to compute the first solution w and next solution w'. Even with constant memory, this can be done by a circuit of constant-depth. Then decide the parity as in Theorem 4.3.

□

4.3 Satisfiability Problems

Let $V = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. We denote by $\neg x_j$ the negation of variable x_j. A *literal* is a variable (*positive literal*) or its negation (*negative literal*). A *clause* is a disjunction of literals. A *CNF-formula* is a conjunction of clauses, seen as a set of clauses. A *truth assignment* $I\colon V \to \{0,1\}$, extended to literals in setting $I(\neg x_j) = 1 - I(x_j)$, *satisfies* a clause if it assigns 1 to at least one of its literals. A CNF-formula, or a clause set, is satisfied by an assignment if all its clauses are satisfied. A clause is positive (resp., negative) if it contains only positive (resp., negative) literals. A *Horn clause* is a clause that contains at most one positive literal. A *Krom clause* (or a 2-clause) is a clause that contains at most two literals. An *XOR-clause* is a clause in which the binary disjunction has been replaced by an exclusive-or disjunction. Such a clause can be seen as a linear equation over the two elements field $\mathbb{F}_2$. A Horn (resp., Krom, XOR) formula is a set of Horn (resp. Krom, XOR) clauses.

Deciding the satisfiability of a CNF-formula is well-known to be NP-complete. Nevertheless the problem becomes tractable for some restricted classes of formulas, e.g., Horn, Krom and XOR-formulas. For such classes we investigate the existence of an AC^0-delay enumeration algorithm. First we consider monotone formulas.

Enum-Monotone-Sat
Input: a set of positive (resp. negative) clauses C over a set of variables V
Output: an enumeration of all assignments over V that satisfy C

The following positive result is an immediate corollary of Theorem 4.2.

Theorem 4.6. Enum-Monotone-Sat $\in$ Del$\cdot AC^0$.

If we allow polynomial precomputation, then we obtain an AC^0-delay enumeration algorithm for a class of CNF-formulas, referred to as IHS in the literature (for Implicative Hitting Sets, see [9]), which is larger than the monotone class. A formula in this class consists of monotone clauses (either all positive or all negative) together with implicative clauses.

Enum-IHS-Sat
Input: a set of clauses C over a set of variables V, with $C = \mathcal{M} \cup \mathcal{B}$, where $\mathcal{M}$ is a set of positive clauses (resp. negative clauses) and $\mathcal{B}$ a set of clauses of the form $(\neg x)$ or $(x \vee \neg x')$ (resp. of the form (x) or $(x \vee \neg x')$)
Output: an enumeration of all assignments over V that satisfy C

Theorem 4.7. Enum-IHS-Sat $\in$ Del$_P\cdot AC^0 \setminus$ Del$^*\cdot AC^0$.

Proof. Let C be a set of clauses over a set of n variables $V = \{x_1, \ldots x_n\}$, with $C = \mathcal{P} \cup \mathcal{B}$, where $\mathcal{P}$ is a set of positive clauses and $\mathcal{B}$ a set of binary clauses of the form $(\neg x)$ or $(x \vee \neg x')$. Any truth assignment can be seen as a binary word of length n.

Observe that contrary to the monotone case 1....1 is not a trivial solution. Indeed a negative unary clause $(\neg x)$ in $\mathcal{B}$ forces x to be assigned 0, and this truth value can be propagated to other variables by the implicative clauses of the form $(x \vee \neg x')$. For this reason as a precomputation step we propose the following procedure:

- Build a directed graph G whose set of vertices is V. For any 2-clause $(x \vee \neg x')$ in $\mathcal{B}$ there is an arc (x, x') in G.
- For each variable x compute $\mathrm{tc}(x)$ the set of vertices that are reachable from x in G.

Intuitively $\mathrm{tc}(x)$ contains all variables that have to be assigned 0 in any satisfying assignment in which x is assigned 0. Observe that any variable x such that $(\neg x) \in \mathcal{B}$ has to be assigned 0, and so have to be all the variables in $\mathrm{tc}(x)$. We replace all these variables by their value and simplify the set of clauses accordingly. If the empty clause occurs, then C is not satisfiable, otherwise it is satisfiable by the $1 \ldots 1$ assignment.

So in the following w.l.o.g we suppose that C is satisfiable and has no negative unary clause.

Once the precomputation has been performed, we propose an algorithm that enumerates all truth assignments satisfying C in lexicographical order with the convention $1 < 0$. The algorithm is as follows:

- As a first step output $1 \ldots 1$ the trivial solution.
- Let the set of clauses, C, together with the set of lists of vertices reachable from each vertex x in the digraph G, $\{\mathrm{tc}(x) | x \in V\}$, be the input and y be the last output solution.
  - For each prefix $y_1 \ldots y_i$ of y with $y_i = 1$ and $i \leq n$ consider the word of length n, $z^i = y_1 \ldots y_{i-1} 0 w_{i+1} \ldots w_n$, where for $j \geq i+1$, $w_j = 0$ if $x_j \in \mathrm{tc}(x_i) \cup \bigcup_{\{k | k<i, y_k=0\}} \mathrm{tc}(x_k)$, and $w_j = 1$ otherwise.

- Check whether at least one of these words z^i is an assignment satisfying C.
- If yes select the one with the longest common prefix with y, that is the satisfying assignment z^i with the largest i and output it as the next solution.
- Else stop.

Observe that $y_1 \dots y_{i-1}0$ is the prefix of a solution if and only if $y_1 \dots y_{i-1}0w_{i+1} \dots w_n$ as it is defined is a solution. Moreover if $y_1 \dots y_{i-1}0w_{i+1} \dots w_n$ is a solution, then it is the first one in the considered lexicographical order with prefix $y_1 \dots y_{i-1}0$. The proof that the algorithm is correct is then similar to the one of Theorem 4.2. The precomputation can be done in polynomial time, and when done allows an implementation of the enumeration algorithm with constant-depth circuits, thus proving that Enum-IHS-Sat $\in \mathrm{Del_P} \cdot \mathrm{AC}^0$.

For the lower bound, consider the st-connectivity problem: given a directed graph $G = (V, A)$ with two distinguished vertices s and t, decide whether there exists a path from s to t in G. From G, s and t we build an instance of Enum-IHS-Sat as follows. We consider a set a clauses $C = \mathcal{P} \cup \mathcal{B}$, where $\mathcal{P} = \{(s \vee t)\}$ and $\mathcal{B} = \{(\neg s)\} \cup \{(x \vee \neg y) \mid (x, y) \in A\}$. This is an AC^0-reduction.

Observe that there exists a path from s to t if and only if C is unsatisfiable. Suppose that Enum-IHS-Sat $\in \mathrm{Del}^* \cdot \mathrm{AC}^0$, this means in particular that outputting a first assignment satisfying C or deciding there is none is in AC^0. Thus the above reduction shows that st-connectivity is in AC^0, thus contradicting the fact that st-connectivity is known not to be in AC^0 (see [1, 15]). □

Surprisingly the enumeration method used so far for satisfiability problems presenting a kind of monotonicity can be used for the enumeration of all assignments satisfying a Krom set of clauses (i.e., a 2-CNF formula) as soon as the literals are considered in an appropriate order.

Theorem 4.8. Enum-Krom-Sat $\in \mathrm{Del_P} \cdot \mathrm{AC}^0 \setminus \mathrm{Del}^* \cdot \mathrm{AC}^0$.

Proof. The proof builds on the algorithm in [2] that decides whether a set of Krom clauses is satisfiable in linear time.

Let C be a set of 2-clauses over a set of n variables V. We perform the following precomputation steps:

- Build the associated implication graph, i.e., the directed graph G whose set of vertices is the set of literals $V \cup \{\overline{v} : v \in V\}$ where $\overline{v}$ represents the literal $\neg v$. For any 2-clause $(l \vee l')$ in C there are two arcs $\overline{l} \to l'$ and $\overline{l'} \to l$ in G. Observe that G has a duality property, i.e., if $l \to l'$ is an arc of G, then so is $\overline{l'} \to \overline{l}$.
- For each literal l compute $\mathrm{tc}(l)$ the set of vertices that are reachable from l in G.

Intuitively $\mathrm{tc}(l)$ contains all literals that have to be assigned 1 in any satisfying assignment in which l is assigned 1. Moreover, a given truth assignment is satisfying if and only if there is no arc $1 \to 0$ in the graph in which every literal has been replaced by its truth value.

From this precomputation we can decide whether C is satisfiable. Indeed, it is proven in [2] that C is satisfiable if and only if no strongly connected component of G contains both a variable x and its negation, i.e., there is no variable x such that $x \in \mathrm{tc}(\overline{x})$ and $\overline{x} \in \mathrm{tc}(x)$. We can also detect equivalent literals.

So in the following w.l.o.g we suppose that C is satisfiable and has no equivalent literals. In particular G is then a directed acyclic graph.

We then go on with two additional precomputation steps:

- Compute a topological ordering of the vertices of G, which is denoted $\leq$ in the following.
- In searching through this topological ordering build an ordered sequence $M = (l_1, \dots, l_n)$ of n literals corresponding to the first occurrences of all variables i.e., for all $i, j \leq n$ s.t. $i \neq j$: $l_i \neq l_j$ and $l_i \neq \overline{l_j}$.

Once the precomputation has been performed, we propose an algorithm that enumerates all truth assignments satisfying C. Note that these truth assignments will be given as truth assignments on M (that is, on literals, not variables), in lexicographical order. In particular the all-zero assignment does not assign all variables to 0, but all literals in M to 0. The algorithm is as follows:

- As a first step output $0 \dots 0$ the first solution.
- Let the set of clauses, C, together with the set of lists of vertices reachable from each vertex l in the implication graph G be the input and y be the last output solution.
  - For each prefix $y_1 \dots y_i$ of y with $y_i = 0$ and $i \leq n$ consider the word of length n, $z^i = y_1 \dots y_{i-1} 1 w_{i+1} \dots w_n$, where for $j \geq i+1$, $w_j = 1$ if $l_j \in \mathrm{tc}(l_i) \cup \bigcup_{\{k \mid k<i, y_k=1\}} \mathrm{tc}(l_k)$, $w_j = 0$ otherwise.
  - Check whether at least one of these words z^i is an assignment satisfying C.
  - If yes select the one with the longest common prefix with y, that is the satisfying assignment z^i with the largest i and output it as the next solution.
  - Else stop.

To prove that the algorithm is correct we have to prove the following:

- The assignment $l_1 = 0, \dots, l_n = 0$ is satisfying.
- For all $i \leq n-1$, $y_1 \dots y_i \in \{0,1\}^i$ is the prefix of a solution if and only if $y_1 \dots y_i w_{i+1} \dots w_n$ where for $j \geq i+1$, $w_j = 1$ if $l_j \in \bigcup_{\{k \mid k \leq, y_k=1\}} \mathrm{tc}(l_k)$, and $w_j = 0$ otherwise, is a solution.

Observe that if $y_1 \dots y_i w_{i+1} \dots w_n$ is a solution, then it is the first solution with prefix $y_1 \dots y_i$ in our lexicographical order.

The fact that the assignment $l_1 = 0, \dots, l_n = 0$ is satisfying follows from the proof in [2]. In seek of completeness let us reprove it. In order to get a contradiction suppose it is not the case. This means that there are l_i and l_j in M such that $\overline{l_j} \to l_i$. By duality one can suppose w.l.o.g that $i \leq j$. On the one hand $i \leq j$ implies that in the topological order $l_i \leq l_j$, while on the other hand $\overline{l_j} \to l_i$ implies $\overline{l_j} \leq l_i$. So we have $\overline{l_j} \leq l_i \leq l_j$, which contradicts the fact that l_j (and not $\overline{l_j}$) is in M.

Now let us prove that $y_1 \dots y_i$ is the prefix of a solution if and only if $y_1 \dots y_i w_{i+1} \dots w_n$ where for $j \geq i+1$, $w_j = 1$ if $l_j \in \bigcup_{\{k \mid k \leq i, y_k=1\}} \mathrm{tc}(l_k)$, and $w_j = 0$ otherwise, is a solution.

One implication is trivial. So, let us suppose that $y_1 \dots y_i w_{i+1} \dots w_n$ is not a solution. Then, by replacing the literals with their truth values in the graph, an arc $1 \to 0$ appears. There is a discussion on the variables underlying this arc.

If the observed contradiction involves two variables whose truth values are fixed by the prefix, then $y_1 \dots y_i$ is not the prefix of any solution.

Suppose now that the observed contradiction involves two variables such that one has its truth value fixed by the prefix, the other not. Then they are two literals l_h and l_j with $h \leq i$ and $j \geq i+1$, such that by duality either:

- $l_h \to l_j$, $y_h = 1$ and $w_j = 0$, or
- $l_h \to \overline{l_j}$, $y_h = 1$ and $w_j = 1$, or
- $\overline{l_h} \to l_j$, $y_h = 0$ and $w_j = 0$, or
- $\overline{l_h} \to \overline{l_j}$, $y_h = 0$ and $w_j = 1$.

Suppose that $l_h \to l_j$, $y_h = 1$ and $w_j = 0$. The arc $l_h \to l_j$ implies that $l_j \in \mathrm{tc}(l_h)$, which together with $y_h = 1$ implies $w_j = 1$ by definition, a contradiction.

Suppose now that $l_h \to \overline{l_j}$, $y_h = 1$ and $w_j = 1$. On the one hand $l_h \to \overline{l_j}$ implies that any satisfying assignment that assigns 1 to l_h, assigns 0 to l_j. On the other hand $w_j = 1$ means any satisfying assignment that starts by $y_1 \dots y_i$ assigns 1 to l_j. Hence, since $y_h = 1$ we get a contradiction.

An arc $\overline{l_h} \to l_j$ cannot occur. Indeed, by duality we have then also $\overline{l_j} \to l_h$, which implies that $\overline{l_j} \leq l_h$ in the topological order. The fact that $h \leq j$ implies $l_h \leq l_j$. Thus we have $\overline{l_j} \leq l_h \leq l_j$. Hence contradicting the fact that l_j (and not $\overline{l_j}$) is in M.

Finally an arc $\overline{l_h} \to \overline{l_j}$ cannot occur either. Indeed by duality there is then also the arc $l_j \to l_h$, which implies $l_j \leq l_h$ in the topological order. But, since $h \leq j$, we have also $l_h \leq l_j$, a contradiction.

It remains to deal with the case where the observed contradiction involves two variables whose truth values are not fixed by the prefix. Then they are two literals l_j and l_k with $i + 1 \leq j \leq k$, such that by duality either:

- $l_j \to l_k$, $w_j = 1$ and $w_k = 0$, or
- $l_j \to \overline{l_k}$, $w_j = 1$ and $w_k = 1$, or
- $\overline{l_j} \to l_k$, $w_j = 0$ and $w_k = 0$, or
- $\overline{l_j} \to \overline{l_k}$, $w_j = 0$ and $w_k = 1$.

Suppose $l_j \to l_k$, $w_j = 1$ and $w_k = 0$. By definition $w_j = 1$ means that there is an $h \leq i$ such that $y_h = 1$ and $l_j \in \mathrm{tc}(l_h)$. But then, the arc $l_j \to l_k$ implies that also $l_k \in \mathrm{tc}(l_h)$, thus contradicting the fact that $w_k = 0$.

Suppose now that $l_j \to \overline{l_k}$, $w_j = 1$ and $w_k = 1$. On the one hand $l_j \to \overline{l_k}$ implies that any satisfying assignment that assigns 1 to l_j, assigns 0 to l_k. Since $w_j = 1$ this means in particular that any satisfying assignment that starts by $y_1 \dots y_i$ assigns 0 to l_k. On the contrary $w_k = 1$ means that any satisfying assignment that starts by $y_1 \dots y_i$ assigns 1 to l_k, a contradiction.

The last two cases cannot occur, for the same reasons as in the discussion above: the existence of such arcs either contradicts the definition of M or the definition of a topological order.

The precomputation can be done in polynomial time, and when done allows an implementation of the enumeration algorithm with constant-depth circuits, thus proving that Enum-Krom-Sat $\in \mathrm{Del_P} \cdot \mathrm{AC}^0$.

For the lower bound, the proof given in Theorem 4.7 applies. □

We next turn to the special case where clauses are XOR-clauses, i.e., clauses in which the usual "or" connective is replaced by the exclusive-or connective, $\oplus$. Such a clause can be seen as a linear equation over the two elements field $\mathbb{F}_2$.

Enum-XOR-Sat
Input: a set of XOR-clauses C over a set of variables V
Output: an enumeration of all assignments over V that satisfy C

If we allow a polynomial precomputation step, then we obtain an AC^0-delay enumeration algorithm for this problem. Interestingly this algorithm relies on an efficient enumeration of binary words in a Gray code order style (in which two consecutive outputs differ on one or two digits only) that we have seen in the previous section and contrary to the satisfiability problems studied so far does not provide an enumeration in lexicographical order.

Theorem 4.9. Enum-XOR-Sat $\in \mathrm{Del_P} \cdot \mathrm{AC}^0 \setminus \mathrm{Del}^* \cdot \mathrm{AC}^0$.

Proof. Observe that a set of XOR-clauses C over a set of variables $V = \{x_1, \dots x_n\}$ can be seen as a linear system over V on the two elements field $\mathbb{F}_2$. As a consequence enumerating all assignments over V that satisfy C comes down to enumerating all solutions of the corresponding linear system. As a precomputation step we propose the following procedure:

- Apply Gaussian elimination in order to obtain an equivalent triangular system. If the system has no solution stop. Otherwise we can suppose that the linear system is of rank $n-k$ for some $0 \leq k \leq n-1$, and without loss of generality that $x_1, \ldots, x_k$ are free variables, whose assignment determines the assignment of all other variables in the triangular system.
- Compute a first solution s_0 corresponding to $x_1, \ldots, x_k$ assigned $0 \ldots 0$.
- For each $i = 1, \ldots, k$ compute the solution s_i corresponding to all variables in $x_1, , \ldots x_k$ assigned 0 except x_i which is assigned 1. Compute then the influence list of x_i, $L(x_i) = \{j \mid k+1 \leq j \leq n, s_0(x_j) \neq s_i(x_j)\}$. The influence list of x_i gives the bits that will be changed when going from a solution to another one in flipping only the bit x_i in the prefix corresponding to the free variables. Observe that this list does not depend on the solution (s_0 in the definition) we start from.

With this precomputation one can decide whether the system has solutions and if yes easily output a first solution, corresponding to the prefix $x_1 = \ldots = x_k = 0$. Then the enumeration procedure uses a memoryless enumeration of binary prefixes of length k such that two successive prefixes differ only in one or two bits similar to that of Theorem 4.4 (for Enum-Gray-Rank).

The algorithm is as follows:

- As a first step compute a first solution s_0 corresponding to $x_1, \ldots, x_k$ assigned $0 \ldots 0$. If it exists, output it, else stop.
- Let the triangular system obtained from Gaussian elimination together with the set of influence lists of all free variables $\{L(x_i) | 1 \leq i \leq k\}$ be the input and $s = ww_{k+1} \ldots w_n$, where w is a prefix of length k, be the last output solution.
  - If $w = 0 \cdots 01$, stop
  - If $w = 10 \cdots 01$, then $w' := 10 \cdots 0$ (w and w' are then the last two words of the binary reflected Gray code and differ only at position 0, i.e., on the value assigned to x_k). Compute $s' = w'w'_{k+1} \ldots w'_n$ where for $j \geq k+1$, $w'_j = 1 - w_j$ if $j \in L(x_k)$, and $w'_j = w_j$ otherwise. Output s' as the next solution.
  - Else,
    (i) switch the bit of w at position 0, then
    (ii) find the minimal position i where there is a 1, and switch bit at position $i+1$.
    Let w' the word so obtained. Then w' and w differ only on two positions, say the i_1th and the i_2th. Compute $s' = w'w'_{k+1} \ldots w'_n$ where for $j \geq k+1$, $w'_j = 1 - w_j$ if $j \in (L(x_{i_1}) \cup L(x_{i_2})) \setminus (L(x_{i_1}) \cap L(x_{i_2}))$, and $w'_j = w_j$ otherwise. Output s' as the next solution.

The precomputation runs in polynomial time. The system has 2^k solutions, one for each possible prefix on $x_1, \ldots, x_k$. The enumeration of these prefixes as proposed can be done with AC^0-delay as already seen in the proof of Theorem 4.4. In this enumeration, two successive prefixes differ exactly on two indices, except once when we reach the last but one word in the Gray code and then move to the last one (in which case, the two words differ only on the bit at position 0). When two successive prefixes differ exactly on two indices i_1 and i_2, then we can then go from one solution to the next one in flipping in the former solution the variables which are either in the influence list of x_{i_1} or in the influence list of x_{i_2} but not in both. When the two prefixes differ only on the index k, then we go from one solution to the next one in flipping in the former solution the variables in the influence list of x_k. Since the influence lists have been computed in the precomputation step, this can be done with constant depth circuits with no additional memory, thus concluding the proof.

To show that Enum-XOR-Sat $\notin \mathrm{Del}^* \cdot \mathrm{AC}^0$ we remark that a word $w = w_1 \ldots w_n \in \{0,1\}^n$ has an even number of ones iff the following set of XOR-clauses C has a solution:

$$C = \{x_1 = w_1, x_2 = w_2, \ldots, x_n = w_n, x_1 \oplus x_2 \oplus \cdots \oplus x_n = 0\}.$$

Moreover, all words of length n can be mapped to such systems of the same length. □

Above we have covered most of the tractable enumeration problems for satisfiability of clause sets except Enum-Horn-Sat, defined as follows:

Enum-Horn-Sat
Input: a set of Horn clauses C over a set of variables V
Output: an enumeration of all assignments over V that satisfy C

For this problem we only have a DelayP upper bound. But we can show that with unlimited memory and polynomial-time precomputation we can enumerate Enum-Horn-Sat in AC^0.

Theorem 4.10. Enum-Horn-Sat $\in \mathrm{Del}^*_{\mathrm{P}} \cdot \mathsf{AC}^0 \setminus \mathrm{Del}^* \cdot \mathsf{AC}^0$.

Proof. Let C be a set of clauses over a set of variables V with $|V| = n$, given as an instance of the satisfiability problem. The clause set C can be identified with its variable-literal incidence relation, which can be encoded by a word $x \in \{0,1\}^*$ of length $|C| \times 2n$.

The enumerating circuit family works as follows:

(1) We use the preprocessing phase to compute n satisfying assignments $s_1, \ldots, s_n$ of C.
(2) We build in n steps a memory y containing all the 2^n possible assignments for variables in V. During this generation, the circuit enumerates the assignment s_i, $1 \le i \le n$. To produce y we use a recursive definition of all binary words of size n than can be generated by doubling the memory size at each step. We make use of the definition of the binary reflected Gray code in Section 4.2 to realize this recursive procedure. In detail, the algorithm works as described next.
As before, we use brackets and separators for sequences of binary words to ease the reading. Let us recall the notation: $G^1 = [0, 1]$ and, more generally, G^n is the list of the words of the binary reflected Gray code of length n; we saw that G^n is defined recursively for $n > 1$ by
$$G^n = [0G_0^{n-1}, 0G_1^{n-1}, \ldots, 0G_{2^{n-1}-1}^{n-1}, 1G_{2^{n-1}-1}^{n-1}, \ldots, 1G_1^{n-1}, 1G_0^{n-1}].$$
The circuit families $(C_{|\cdot|})$ and $(D_{|\cdot|})$, computing the memory and the output from step to step are defined as follows:
   - First $C_{|\cdot|}(x)$ computes $y_1^* = s_1 G^1$. We set $D_{|\cdot|}(y_1^*) = s_1$.
   - For $1 < i \le n$, given input x and the previous memory y_{i-1}, the circuit $C_{|\cdot|}$ computes $C_{|\cdot|}(x, y_{i-1}^*) = y_i^* = s_i G^i$. Here, G^i is easily computed by a circuit of size polynomial in $|x|$ and $|G^{i-1}| = 2^{i-1}$ in constant depth. According to the above definition, it just has to output G^{i-1} followed by a mirror image of itself and append 0 to all words of G^{i-1} and 1 to all words of the mirror image. We set $D_{|\cdot|}(y_i^*) = s_i$.
   - After n steps, the memory is $y = G^n = [G_0^n, G_1^n, \ldots, G_{2^n-1}^n]$.
(3) We view y as a sequence of all candidate assignments of C. In y we mark all candidates which are actual solutions that have not been output yet.
(4) In AC^0 we can find the first marked candidate, output it, and unmark it. We repeat this step until all solutions have been output.

That Enum-Horn-Sat $\notin \mathrm{Del}^* \cdot \mathsf{AC}^0$ follows from the fact that the satisfiability problem for Horn formulas is P-complete [20] and therefore not in AC^0. □

Observe that the above proof relies on the two facts that we can compute polynomial number of solutions in polynomial time and that the underlying relation is AC^0-checkable. Therefore we conclude:

Corollary 4.11. *Let R be a polynomially balanced relation such that* Enum$\cdot R \in$ IncP *and R is AC^0-checkable. Then* Enum$\cdot R \in \mathrm{Del}^*_{\mathrm{P}} \cdot \mathsf{AC}^0$.

5 SEPARATIONS OF DELAY CLASSES DEFINED BY CIRCUITS

In the previous results we already presented a few lower bounds, but now we will systematically strive to separate the studied classes.

As long as no precomputation is allowed, we are able to separate all delay classes with bounded auxiliary memory. With precomputation, the situation seems to be more complicated. We obtain only a conditional separation of the class with constant memory from the one without memory at all. Finally we obtain a conditional separation of the classes with unbounded memory from the ones with polynomially bounded memory.

5.1 Unconditional Separations for Classes without Precomputation

THEOREM 5.1. $\mathrm{Del}\cdot\mathrm{AC}^0 \subsetneq \mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$

PROOF. Let $x \in \{0,1\}^*$, $|x| = n \in \mathbb{N}^*$, $x = x_1 \dots x_n$. We denote by $m = \lceil \log n \rceil + 1$. Let R_L be defined for all $x \in \{0,1\}^*$ as the union of the two following sets A and B:

- $A = \left\{ y \in \{0,1\}^* \middle| |y| = m, y \neq 0^m, y \neq 1^m \right\}$
- $B = \{ 1^m \}$ if x has an even number of ones, else $B = \{ 0^m \}$.

We denote by $z_1, ..., z_t$ an enumeration of elements of A. Clearly, $|R_L(x)| = t + 1$ and $t \geq n$. To show that $R_L \in \mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$, we use the enumeration of elements of A (which is easy) and one additional memory bit that is transferred from one step to the other to compute PARITY. Indeed, we build families of circuits (C_n) and (D_n) according to Definition 3.3 as follows.

- First $C_{|\cdot|}(x)$ computes $y_1^* = z_1 b^1$ where $b^1 = x_1$, and $D_{|\cdot|}(y_1^*) = z_1$.
- For $1 < i \leq t$, the circuit $C_{|\cdot|}(x, y_{i-1}^*)$ computes y_i^*, where $y_i^* = z_i b^i$ with $b^i = b^{i-1} \oplus x_i$ if $i \leq n$, and $b^i = b^{i-1}$ else, and $D_{|\cdot|}(y_i^*) = z_i$.
- After $t \geq n$ steps, the memory bit b^t contains a 0 if and only if the number of ones in x is even. According to this, we either output 1^m or 0^m as last solution.

Note that the size of the solutions is m, the size of the memory words above is $m + 1$, hence we need constant amount of additional memory. The circuit families (C_n) and (D_n) are obviously DLOGTIME-uniform.

Suppose now that $R_L \in \mathrm{Del}\cdot\mathrm{AC}^0$ and let (C_n) be the associated family of enumeration circuits. We construct a circuit family as follows: We compute in parallel all $C_{|\cdot|}(x)$ and $C_{|\cdot|}(x, z_i)$ for $1 \leq i \leq t$. In this way, we will obtain among other solutions either 0^m or 1^m. We accept in the first case. Note that the z_i are the same for all inputs x of the same length. Thus, we obtain an AC^0 circuit family for parity, contradicting [1, 15]. □

By extending the above approach, one can prove the following separation:

THEOREM 5.2. $\mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0 \subsetneq \mathrm{Del}^{\mathrm{p}}\cdot\mathrm{AC}^0$

PROOF. For a given $x \in \{0,1\}^*$, $n = |x|$, we denote by $m = \lceil \log \log n \rceil + 1$. Let R_L be defined for all $x \in \{0,1\}^*$ as the union of the two following sets A and B:

- $A = \left\{ y \in \{0,1\}^* \middle| |y| = m, y \neq 0^m, y \neq 1^m \right\}$
- $B = \{ 1^m \}$ if x has an even number of ones, else $B = \{ 0^m \}$.

We denote by $z_1, ..., z_t$ an enumeration of elements of A. Clearly, $|R_L(x)| = t + 1$ and $t \geq \log n$.

To show that $R_L \in \mathrm{Del}^{\mathrm{p}}\cdot\mathrm{AC}^0$, we mimic the proof of the computation of the parity of n input bits with a tree of binary parity-gates. Indeed, we build families of circuits (C_n) and (D_n) according to Definition 3.3 as follows.

- First $C_{|\cdot|}(x)$ computes $y_1^* = z_1 b^1$ where $b_j^1 = x_{2j-1} \oplus x_{2j}$; if n is odd then $b_{\lceil n/2 \rceil}^1 = x_n$. We set $D_{|\cdot|}(y_1^*) = z_1$.

- For $1 < i \leq t$, the circuit $C_{|\cdot|}(x, y^*_{i-1}) = y^*_i$, where $y^*_i = z_i b^i$ where $b^i_j = b^{i-1}_{2j-1} \oplus b^{i-1}_{2j}$ for $1 \leq j \leq \lceil n/2 \rceil$; if $|b^i|$ is odd then the last bit of b^i is the last bit of b^{i-1}_n. We set $D_{|\cdot|}(y^*_i) = z_i$.
- After $t \geq \log n$ steps, the bit b^t_1 contains a 0 if and only if the number of ones in x is even. According to this, we either output 1^m or 0^m as last solution.

Note that the memory words above are of linear size.

Suppose now that $R_L \in \mathrm{Del}^{\mathrm{c}} \cdot \mathrm{AC}^0$. Let (C_n), (D_n) be the associated families of enumeration circuits and let $c \in \mathbb{N}$ be the space allowed for additional memory. Fix an input x, $|x| = n$. Let t be as above. Now, for any sequence $b^1, \ldots, b^{t+1}$ of additional memory, where one b^i is empty and all the others have length exactly c, we can check in AC^0 that this is a correct sequence: First, in parallel, we compute for every b^i, the value $C_{|\cdot|}(x, z_i b^i)$. Then, we check in AC^0 that this value is of the form $z_j b^j$ for some j and that each $z_j b^j$ appears only once as an image (as a parallel pairwise check of inequalities). Here, for the empty memory word ε we let $z\varepsilon = \varepsilon$. For the correct sequence, we check as in the proof of Theorem 5.1 if the solution 0^m or 1^m will appear.

Since the length of a sequence of additional memory words is at most $O(ct) = O(c \log n)$, their number is polynomial in n and we can do all the above checks in parallel. Hence we obtain an AC^0 circuit for parity which does not exist. □

The PARITY problem can be seen as an enumeration problem: given x, one output the unique solution 1 if the number of ones in x is even. One outputs 0 if it is odd. Since as a function problem, PARITY can not be in $\mathrm{Del}^{\mathrm{p}} \cdot \mathrm{AC}^0$ (the fact there is only one solution makes memory useless). It is obviously in DelayP. This implies that $\mathrm{Del}^{\mathrm{p}} \cdot \mathrm{AC}^0 \subsetneq$ DelayP. Putting all the previous results together, we conclude:

COROLLARY 5.3. $\mathrm{Del} \cdot \mathrm{AC}^0 \subsetneq \mathrm{Del}^{\mathrm{c}} \cdot \mathrm{AC}^0 \subsetneq \mathrm{Del}^{\mathrm{p}} \cdot \mathrm{AC}^0 \subsetneq$ DelayP.

5.2 Conditional Separation for Classes with Precomputation

If precomputation is allowed, the separation proofs of the previous subsection no longer work; in fact we do not know if the corresponding separations hold. However, under reasonable complexity-theoretic assumptions we can at least separate the classes $\mathrm{Del}_{\mathrm{P}} \cdot \mathrm{AC}^0$ and $\mathrm{Del}^{\mathrm{c}}_{\mathrm{P}} \cdot \mathrm{AC}^0$. Note that in Theorem 4.3 we already proved a separation of just these two classes, but this concerns only the special case of ordered enumeration, and does not say anything about the general case. We find it interesting that the proof below relies on a characterization of the class PSPACE in terms of regular leaf-languages or *serializable computation* [16, 24].

THEOREM 5.4. *If* NP ≠ PSPACE, *then* $\mathrm{Del}^{\mathrm{c}} \cdot \mathrm{AC}^0 \setminus \mathrm{Del}_{\mathrm{P}} \cdot \mathrm{AC}^0 \neq \emptyset$.

PROOF. Hertrampf *et al.* [16] proved that PSPACE is AC^0-serialisable, i. e., every PSPACE decision algorithm can be divided into an exponential number of slices, each requiring only the power of AC^0 and passing only a constant number of bits to the next slice. More formally, for every language $L \in$ PSPACE there is an AC^0-circuit family $C = (C_n)_{n \in \mathbb{N}}$, numbers $k, l \in \mathbb{N}$ such that every C_n has k output bits and for every input x, $|x| = n$, $N = n + n^l + k$, $x \in L$ if and only if $c_{2^{n^l}} = \underbrace{1 \ldots 1}_{k}$ where

- y_i is the i-th string in $\{0,1\}^{n^l}$ in lexicographical order,
- $c_1 = C_N(x, y_1, \underbrace{0 \ldots 0}_{k})$, and
- $c_i = C_N(x, y_i, c_{i-1})$ for $1 < i \leq 2^{n^l}$.

Depending on L we now define the relation

$$R_L(x) = \left\{ 0y \in \{0,1\}^* \,\middle|\, |y| = n^l \right\} \cup \{\, 1^{n^l+1} \mid x \in L \,\} \cup \{\, 10^{n^l} \mid x \notin L \,\}.$$

To enumerate R_L with constant auxiliary memory, we construct circuit families $C' = (C'_n)_{n\in\mathbb{N}}$ and $\mathcal{D}' = (D'_n)_{n\in\mathbb{N}}$ as follows:

- $C'_{|\cdot|}(x) = (y_0, c)$ where $c = C_N(x, y_0, 0\ldots0)$
- $C'_{|\cdot|}(x,(y_i,c)) = (y_{i+1}, c')$ for $0 \le i < 2^{n^l}$ and $c' = C_N(x, y_i, c)$,
- $C'_{|\cdot|}(x,(y_{2^{n^l}},c)) = (1^{n^l+1}, c)$
- $C'_{|\cdot|}(x,(1^{n^l+1},c)) = (1^{n^l+1}, c)$
- $D'_{|\cdot|}(y_i, c) = 0y_i$ for $1 \le i \le 2^{n^l}$
- $D'_{|\cdot|}(1^{n^l+1}, c) = 1^{n^l+1}$ if $C_N(x, y_{2^{n^l}}, c) = \underbrace{1\ldots1}_{k}$, 10^{n^l} else.

Thus we see that R_L can be enumerated with k bits of auxiliary memory, i.e, $\textsc{Enum}\cdot R_L \in \mathrm{Del}^{\mathrm{c}}\cdot\mathrm{AC}^0$.

Now suppose $\textsc{Enum}\cdot R_L \in \mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$ via AC^0-circuit family $C = (C_n)_{n\in\mathbb{N}}$. Then $L \in \mathsf{NP}$ by the following algorithm.

- Given x, use precomputation to produce x^*.
- Check if $C_{|\cdot|}(x^*) = 1^{n^k+1}$. If yes, accept.
- Guess some output $y \in R_L(x)$.
- Check that $C_{|\cdot|}(x^*, y) = 1^{n^k+1}$.
- If yes, then accept.

□

5.3 Conditional Separation for Classes with Unbounded Memory

In this section we separate polynomial memory classes from those with unbounded memory under condition. In doing so we prove that AC^0-delay classes with unbounded auxiliary memory are atypical classes that are included in none of the well-known classes, neither in DelayP nor in IncP.

Theorem 5.5. *If* $\mathsf{P} \neq \mathsf{NP}$, *then* $\mathrm{Del}^*\cdot\mathrm{AC}^0 \setminus \mathrm{Del}^{\mathrm{p}}_{\mathrm{P}}\cdot\mathrm{AC}^0 \neq \emptyset$.

Proof. Let C be a set of propositional clauses over a set of variables V with $|V| = n$. As before, we identify the clause set C with its variable-literal incidence relation $x \in \{0,1\}^*$ of length $|C| \times 2n$.

Let $m = \lceil \log n \rceil + 1$ and $R_L(x)$ be defined for all $x \in \{0,1\}^*$ as the union of the two following sets A and B:

- $A = \left\{ y \in \{0,1\}^* \middle| |y| = m, y \neq 0^m, y \neq 1^m \right\}$,
- $B = \{1^m\}$ if C is satisfiable, else $B = \{0^m\}$.

Let $z_1, ..., z_t$ be an enumeration of elements of A. It holds that $t \ge n$.

We show that $\textsc{Enum}\cdot R_L \in \mathrm{Del}^*\cdot\mathrm{AC}^0$. We use an approach similar to the one in the proof of Theorem 4.10 and proceed by building in at most $O(n) = O(2^m)$ steps a memory y containing all the 2^n possible assignments for variables in V. During this generation, the circuit enumerates all elements in A. Once this is done, we check in constant depth if at least one of the possible assignments in y is satisfying.

The circuit families $(C_{|\cdot|})$ and $(D_{|\cdot|})$, computing the memory and the output from step to step are defined as follows:

- First $C_{|\cdot|}(x)$ computes $y_1^* = z_1 G^1$, and $D_{|\cdot|}(y_1^*) = z_1$.
- For $1 < i \le n \le t$, given input x and the previous memory y_{i-1}, $C_{|\cdot|}$ computes $C_{|\cdot|}(x, y_{i-1}^*) = y_i^* = z_i G^i$, and $D_{|\cdot|}(y_i^*) = z_i$.
- Once y_n^* has been produced, just enumerate the remaining elements of A if any.

– After t steps, the memory is $G^n = [G^n_0, G^n_1, \ldots, G^n_{2^n-1}]$. We just check in constant depth if one of G^n_j, $j \leq 2^{n-1}$, is a satisfying assignment for C.

Suppose now that $\textsc{Enum}\cdot R_L \in \mathrm{Del}^{\mathrm{p}}_{\mathrm{p}}\cdot\mathrm{AC}^0$. The whole process, including precomputation and generating the elements of A, is polynomial time since $t = O(n)$. Hence, we can decide in polynomial time if C is satisfiable. Contradiction with the fact that $\mathsf{P} \neq \mathsf{NP}$. □

The following corollary is obvious.

Corollary 5.6. *If* $\mathsf{P} \neq \mathsf{NP}$, *then* $\mathrm{Del}^{\mathrm{p}}_{\mathrm{p}}\cdot\mathrm{AC}^0 \subsetneq \mathrm{Del}^{*}_{\mathrm{p}}\cdot\mathrm{AC}^0$ *and* $\mathrm{Del}^{\mathrm{p}}\cdot\mathrm{AC}^0 \subsetneq \mathrm{Del}^{*}\cdot\mathrm{AC}^0$.

Coming back to the relation betweeen $\mathrm{Del}^{*}_{\mathrm{p}}\cdot\mathrm{AC}^0$ and the classes DelayP and IncP, we now can state the following.

Corollary 5.7. *If* $\mathsf{P} \neq \mathsf{NP}$, *then* $\mathrm{Del}^{*}\cdot\mathrm{AC}^0 \not\subseteq \mathsf{IncP}$.

Proof. Consider the relation $R_L \in \mathrm{Del}^{*}\cdot\mathrm{AC}^0$ from the proof of Theorem 5.5. Suppose now that $R_L \in \mathsf{IncP}$. As above, the whole process of generating the elements of A is then polynomial time since $t = O(n)$. Hence, we can decide in polynomial time if C is satisfiable. Contradiction with the fact that $\mathsf{P} \neq \mathsf{NP}$. □

6 CONCLUSION

In this paper we introduced complexity classes for enumeration based on circuits that we regard as subclasses of DelayP corresponding to efficient parallel enumeration. Considering different bounds for precomputation and for space we obtain a hierarchy of classes and prove its usefulness by placing natural problems in different classes from our framework. The obtained inclusion relations among the classes we introduced are summarised in Fig. 1.

We noted earlier that in our context, enumeration problems are defined without a complexity assumption concerning the underlying relation. We should remark that quite often, a polynomial-time upper bound is required, see [6, 22]. All of our results, with the exception of the conditional separations in Sect. 5 also hold under the stricter definition; however, the relation R_L used in the lower bounds in Subsect. 5.2 is based on a PSPACE-complete set and therefore, to check whether $y \in R_L(x)$ requires polynomial space w.r.t. the length of x. Similarly for Theorem 5.5, the argument is based on a relation that is NP-complete. It would be nice to be able to base these separations on polynomial-time checkable relations, or even better, to separate the classes unconditionally, but this remains open. Moreover, some further inclusions in Fig. 1 are still not known to be strict.

In Subsect. 4.3, we proved that, for several fragments of propositional logic, among them the Krom and the affine fragments, the enumeration of satisfiable assignments is in the class $\mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$. This means satisfiable assignments can be enumerated very efficiently, i. e., by an AC^0-circuit family, after some precomputation, which is also efficiently doable (in polynomial time). For another important and very natural fragment of propositional logic, namely the Horn fragment, a DelayP-algorithm is known, but it is not at all clear how polynomial-time precomputation can be of any help to produce more than one solution. An incomparable upper bound that we proved in this paper is $\textsc{Enum-Horn-Sat} \in \mathrm{Del}^{*}_{\mathrm{p}}\cdot\mathrm{AC}^0$ (Theorem 4.10). We know that $\textsc{Enum-Horn-Sat} \notin \mathrm{Del}^{*}\cdot\mathrm{AC}^0$, and we conjecture that it is not in $\mathrm{Del}_{\mathrm{P}}\cdot\mathrm{AC}^0$. The exact status of Enum-Horn-Sat remains open.

Introducing unbounded memory allows to generate all candidate solutions and check them in parallel, which makes our classes too powerful. Actually, under the assumption $\mathsf{P} \neq \mathsf{NP}$ we know that $\mathrm{Del}^{*}\cdot\mathrm{AC}^0$ is incomparable with both DelayP and IncP, since $\mathrm{Del}^{*}\cdot\mathrm{AC}^0 \not\subseteq \mathsf{IncP}$ (Corollary 5.7) and $\mathsf{DelayP} \not\subseteq \mathrm{Del}^{*}\cdot\mathrm{AC}^0$ (witnessed by Enum-Horn-Sat). Note that when using preprocessing, we still know that $\mathrm{Del}^{*}_{\mathrm{p}}\cdot\mathrm{AC}^0 \not\subseteq \mathsf{IncP}$, but the other direction and even the inclusion $\mathsf{DelayP} \subseteq \mathrm{Del}^{*}_{\mathrm{p}}\cdot\mathrm{AC}^0$ is open.

REFERENCES

[1] Miklós Ajtai. First-order definability on finite structures. *Annals of Pure and Applied Logic*, 45:211–225, 1989.

[2] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

[3] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007. URL: https://doi.org/10.1007/978-3-540-74915-8_18.

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

[5] Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.

[6] Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discret. Appl. Math.*, 268:179–190, 2019. URL: https://doi.org/10.1016/j.dam.2018.06.038.

[7] Peter Clote and Evangelos Kranakis. *Boolean Functions and Computation Models*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002. URL: https://doi.org/10.1007/978-3-662-04943-3.

[8] Nadia Creignou, Arnaud Durand, and Heribert Vollmer. Enumeration classes defined by circuits. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022, August 22-26, 2022, Vienna, Austria*, volume 241 of *LIPIcs*, pages 38:1–38:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.4230/LIPIcs.MFCS.2022.38.

[9] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*, volume 7 of *SIAM monographs on discrete mathematics and applications*. SIAM, 2001.

[10] Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. A complexity theory for hard enumeration problems. *Discret. Appl. Math.*, 268:191–209, 2019. URL: https://doi.org/10.1016/j.dam.2019.02.025.

[11] Arnaud Durand. Fine-grained complexity analysis of queries: From decision to counting and enumeration. In Dan Suciu, Yufei Tao, and Zhewei Wei, editors, *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 331–346. ACM, 2020. doi:10.1145/3375395.3389130.

[12] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007.

[13] Thomas Eiter, Georg Gottlob, and Kazuhisa Makino. New results on monotone dualization and generating hypergraph transversals. *SIAM J. Comput.*, 32(2):514–537, 2003. doi:10.1137/S009753970240639X.

[14] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21(3):618–628, 1996. doi:10.1006/jagm.1996.0062.

[15] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Syst. Theory*, 17(1):13–27, 1984. doi:10.1007/BF01744431.

[16] Ulrich Hertrampf, Clemans Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. On the power of polynomial time bit-reductions. In *Proceedings 8th Structure in Complexity Theory*, pages 200–207. IEEE Computer Society Press, 1993.

[17] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. URL: https://doi.org/10.1016/0020-0190(88)90065-8.

[18] Donald E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*, volume 4A. Addison-Wesley Professional, 1st edition, 2011.

[19] Donald L. Kreher and Douglas Robert Stinson. *Combinatorial Algorithms: generation, enumeration, and search*. CRC Press, 1999.

[20] Richard E. Ladner. The circuit value problem is log space complete for *P*. *SIGACT News*, 7(1):18–20, 1975. doi:10.1145/990518.990519.

[21] Luc Segoufin. A glimpse on constant delay enumeration (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPIcs*, pages 13–27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.STACS.2014.13.

[22] Yann Strozecki. Enumeration complexity. *Bull. EATCS*, 129, 2019. URL: http://bulletin.eatcs.org/index.php/beatcs/article/view/596/605.

[23] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979. doi:10.1137/0208032.

[24] Heribert Vollmer. A generalized quantifier concept in computational complexity theory. In Jouko A. Väänänen, editor, *Generalized Quantifiers and Computation, 9th European Summer School in Logic, Language, and Information,*

ESSLLI'97 Workshop, Revised Lectures, volume 1754 of *Lecture Notes in Computer Science*, pages 99–123. Springer, 1999. doi:10.1007/3-540-46583-9_5.

[25] Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Springer, Heidelberg, 1999. URL: https://doi.org/10.1007/978-3-662-03927-4.