HAL open science

Enumeration Classes Defined by Circuits

Nadia Creignou, Arnaud Durand, Heribert Vollmer

To cite this version:

Nadia Creignou, Arnaud Durand, Heribert Vollmer. Enumeration Classes Defined by Circuits. 47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022), Aug 2022, Vienne, Austria. pp.38, 10.4230/LIPIcs.MFCS.2022.38 . hal-04166123

HAL Id: hal-04166123

https://hal.science/hal-04166123v1

Submitted on 19 Jul 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Enumeration Classes Defined by Circuits

Nadia Creignou
Aix Marseille Univ, CNRS, LIS, Marseille, France

Arnaud Durand
Université Paris Cité, CNRS, IMJ-PRG, Paris, France

Heribert Vollmer
Leibniz Universität Hannover

Abstract

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.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Circuit complexity

Keywords and phrases Computational complexity, enumeration problem, Boolean circuit

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.38

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 [14], 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 [9].

In this paper we will look for notions of tractability for enumeration stricter than the above two. More specifically, we will introduce a refinement of the existing classes based on the computation model of Boolean circuits. The main new class in our framework is the class $\mathbf{Del}\cdot\mathbf{AC}^0$. An enumeration problem belongs to this class if there is a family of $\mathbf{AC}^0$ 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

© Nadia Creignou, Arnaud Durand and Heribert Vollmer;
licensed under Creative Commons License CC-BY 4.0
47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022).
Editors: Stefan Szeider, Robert Ganian, and Alexandra Silva; Article No. 38; pp. 38:1–38:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

input and the last solution, signals that no further solution exists. Still using $\mathbf{AC}^0$ 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 (from a constant to a polynomial number of bits)

By this, we obtain a hierarchy of classes within **DelayP**/**IncP** shown in Fig. 1.

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 $\mathbf{Del}\cdot\mathbf{AC}^0$, some cannot, but allowing precomputation or a certain number of bits of auxiliary memory they can. We would like to mention in particular the maybe algorithmically most interesting contribution of our paper, the case of enumeration for satisfiability of 2-CNF (Krom) formulas. While it is known that counting satisfying assignments for formulas from this fragment of propositional logic is #**P**-complete [18], we exhibit a $\mathbf{Del_P}\cdot\mathbf{AC}^0$ algorithm (i.e. $\mathbf{Del}\cdot\mathbf{AC}^0$ 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 $\mathbf{AC}^0$ is needed to produce the next solution) after a polynomial time precomputation before producing the first solution.

Building on well-known lower bounds (in particular for the parity function [12, 1]) 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 $\mathbf{CD}\circ\mathbf{lin}$ of problems that can be enumerated on RAMs with constant delay after linear time preprocessing [11] (see also the surveys [16, 10]). 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 $\mathbf{CD}\circ\mathbf{lin}$ [3], but we will see that it is in $\mathbf{Del}\cdot\mathbf{AC}^0$. On the other hand, the familiar lower bound for parity [1, 12] easily leads to an enumeration problem not in $\mathbf{Del}\cdot\mathbf{AC}^0$ but in $\mathbf{CD}\circ\mathbf{lin}$; hence we see that $\mathbf{CD}\circ\mathbf{lin}$ and $\mathbf{Del}\cdot\mathbf{AC}^0$ are incomparable classes; thus our approach provides a novel way to refine polynomial delay (see Section 3.3).

This paper is organized 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.

Because of space limitations, some of our proofs are only sketched or even omitted here; full proofs will appear in the final version of this paper.

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 $|y| \leq p(|x|)$. Now we 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 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 [17, 6]. 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., [20, 7]. We use $\mathbf{AC}^0$ to denote the class 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 $\mathbf{FAC}^0$, and for simplicity often again by $\mathbf{AC}^0$. 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 $\mathbf{NC}^1$.

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 [20].

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, however, we will see that our delay-classes with memory in a sense reflect incremental classes in the circuit model.

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 $\mathbf{AC}^0$ or $\mathbf{NC}^1$. 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 $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ 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 $\mathcal{K} = \mathbf{AC}^0$ or $\mathcal{K} = \mathbf{NC}^1$, but any circuit complexity class will do. Our definitions make sense both for uniform and for non-uniform classes.

▶ **Definition 1.** [$\mathcal{K}$-delay]
Let R be a polynomially balanced predicate. The enumeration problem Enum·R *is in* $\mathbf{Del}\cdot\mathcal{K}$ *if there exists a family of $\mathcal{K}$-circuits $\mathcal{C} = (C_n)_{n\in\mathbb{N}}$ 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 2.** [$\mathcal{K}$-delay with T-precomputation]
Let R be a polynomially balanced predicate and T be a complexity class. The enumeration problem Enum·R *is in* $\mathbf{Del}_T\cdot\mathcal{K}$ *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.** [$\mathcal{K}$-delay with auxiliary memory]
Let R be a polynomially balanced predicate. The enumeration problem Enum·R *is in* $\mathbf{Del}^*\cdot\mathcal{K}$ *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 any 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 $\mathbf{Del}^{\mathrm{P}}\cdot\mathcal{K}$, $\mathcal{K}$-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* $\mathbf{Del}^{\mathrm{c}}\cdot\mathcal{K}$, $\mathcal{K}$-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. When there is no restriction on memory i.e. when considering the class $\mathbf{Del}_T^* \cdot \mathcal{K}$, an incremental enumeration mechanism can be used. Indeed, the memory can then store all solutions produced so far which results in an increase of the expressive power.

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

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

3.3 Relation to Known Enumeration Classes

All classes we consider in this paper are subclasses of the well-known classes **DelayP** or **IncP**, resp., even if we allow our circuit to be of arbitrary depth (but polynomial size).

▶ **Theorem 5.** *If $\mathcal{K}, T \subseteq \mathbf{P}$, then $\mathbf{Del}_T^{\mathrm{P}} \cdot \mathcal{K} \subseteq \mathbf{DelayP}$ and $\mathbf{Del}_T^* \cdot \mathcal{K} \subseteq \mathbf{IncP}$.*

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 [11].

The algorithmic problem, given a graph, to enumerate all pairs of vertices that are connected by a path of length 2 has only a polynomial number of solutions and is trivially in $\mathbf{Del} \cdot \mathbf{AC}^0$. Since it is essentially the same as Boolean matrix multiplication, it is not in **CD∘lin**, assuming the beforementioned 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 $\mathbf{Del} \cdot \mathbf{AC}^0$, since the parity function is not in $\mathbf{AC}^0$ [12, 1]. However, since Parity can be computed in linear time, Enum-Parity is trivially in **CD∘lin**.

As we will show in detail in the full version of this paper, the computation of a constant number of time steps of a RAM can be simulated by $\mathbf{AC}^0$ 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 6.** *Assuming the BMM hypothesis, the classes $\mathbf{Del} \cdot \mathbf{AC}^0$ and $\mathbf{CD} \circ \mathbf{lin}$ are incomparable, and $\mathbf{CD} \circ \mathbf{lin} \subsetneq \mathbf{Del}_{lin}^{\mathrm{P}} \cdot \mathbf{AC}^0$.*

4 Examples

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

4.1 Graph Problems

We first consider the enumeration problem associated with the notion of reachability in a graph.

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

▶ **Theorem 7.** Enum-Reach $\in \mathbf{Del}^{\mathrm{P}} \cdot \mathbf{AC}^0$

Proof. At each step multiplication of Boolean matrices gives the set of vertices which are reachable from s with one more step. This can be done in $\mathbf{AC}^0$. The polynomial memory is used to remember all vertices that have been encountered to far. Observe that in each step, more vertices are being produced, until we reach convergence. ◀

Let us now turn to the enumeration of all transversals (i. e., vertex-sets intersecting every edge), not only the minimal ones.

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

▶ **Theorem 8.** Enum-Transversal $\in \mathbf{Del} \cdot \mathbf{AC}^0$.

Proof. Let $\mathcal{E}$ be a 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, $z^i = y_1 \ldots y_{i-1} 01 \ldots 1$.
  - 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 select the next transversal in the lexicographical order.

Now let us prove that this is an $\mathbf{AC}^0$-delay enumeration algorithm that does not require memory. The main observation is that one can check with an $\mathbf{AC}^0$ circuit whether a binary word corresponds to 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 $\mathbf{AC}^0$ circuit. ◀

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

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.

The binary reflected Gray code of length n, denoted G^n, 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 $n = 4$: $(0, 0000)$, $(1, 0001)$, $(2, 0011)$, $(3, 0010)$, $(4, 0110)$, $(5, 0111)$, $(6, 0101)$, $(7, 0100)$, $(8, 1100)$, $(9, 1101)$, $(10, 1111)$, $(11, 1110)$, $(12, 1010)$, $(13, 1011)$, $(14, 1001)$, $(15, 1000)$.

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 $j = 0, ..., n-1$,

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

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 lexicographic order, this is not so clear for Gray code order. Also, given a rank or a first word, to enumerate all words of higher Gray code rank (in arbitrary order) are interesting computational problems.

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, a very efficient enumeration is possible:

▶ **Theorem 9.** *Let n be an integer*

1. *Given 1^n, enumerating all words of length n even in lexicographic ordering is in* $\mathbf{Del}\cdot\mathbf{AC}^0$
2. Enum-Gray-Rank $\in \mathbf{Del}\cdot\mathbf{AC}^0$
3. Enum-Gray-Word $\in \mathbf{Del}\cdot\mathbf{AC}^0$

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.

▶ **Theorem 10.** *Given 1^n, enumerating all words of length n in a Gray code order is in* $\mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0 \backslash \mathbf{Del_P} \cdot \mathbf{AC}^0$

Proof. A classical method to enumerate gray code of length n is the following [15].

- 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 $\mathbf{AC}^0$-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 $\mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0$.

For the lower bound, suppose $\mathcal{C} = (C_n)_{n \in \mathbb{N}}$ is an $\mathbf{AC}^0$ circuit family enumerating the Gray code of length n after polynomial time precomputation produced by machine M. We will describe how to use $\mathcal{C}$ to construct an $\mathbf{AC}^0$-family computing the parity function, contradicting the lower bound given by [1, 12].

Given is an arbitrary word $w = w_{n-1} \dots w_0$ of length n, and 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 $\mathcal{C}$. Let w' be the next words after w. There exists r such that $G_r^n = w$ and $G_{r+1}^n = w'$. 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 [12, 1] that parity cannot even be computed by non-uniform $\mathbf{AC}^0$ circuit families. ◀

We also consider the problem of enumerating all words starting not from the first one but at a given position, but now in Gray code order. Surprisingly this time the complexity will depend on how the starting point is given, by rank or by word.

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 11.** 1. Enum-Gray-Rank$_{ord}$ $\in \mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0 \backslash \mathbf{Del_P} \cdot \mathbf{AC}^0$.
2. Enum-Gray-Word$_{ord}$ *is in the class* $\mathbf{Del}^{\mathrm{c}}_{\mathbf{P}} \cdot \mathbf{AC}^0$, *but neither in* $\mathbf{Del_P} \cdot \mathbf{AC}^0$ *nor* $\mathbf{Del}^{\mathrm{c}} \cdot \mathbf{AC}^0$.

4.3 Satisfiability Problems

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. For such classes we investigate the existence of an $\mathbf{AC}^0$-delay enumeration algorithm. First we consider monotone formulas.

Enum-Monotone-Sat

Input: A set of positive (resp. negative) clauses Γ over a set of variables V

Output: an enumeration of all assignments over V that satisfy Γ

The following positive result is an immediate corollary of Theorem 8.

▶ **Theorem 12.** Enum-Monotone-Sat $\in \mathbf{Del}\cdot\mathbf{AC}^0$.

If we allow polynomial precomputation, then we obtain an $\mathbf{AC}^0$-delay enumeration algorithm for a class of CNF-formulas, referred to as IHS in the literature (for Implicative Hitting Sets, see [8]), 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 $\mathcal{C}$ over a set of variables V, with $\mathcal{C} = \mathcal{M} \cup \mathcal{B}$, 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 of the form (x) or $(x \vee \neg x')$)

Output: an enumeration of all assignments over V that satisfy Γ

▶ **Theorem 13.** Enum-IHS-Sat $\in \mathbf{Del_P}\cdot\mathbf{AC}^0 \setminus \mathbf{Del}^*\cdot\mathbf{AC}^0$.

Proof sketch. 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, for each variable x we compute $\mathrm{tc}(x)$ the set of all variables that have to be set to 0 in any assignment satisfying Γ in which x is assigned 0. With this information we can use an algorithm that enumerates all truth assignments satisfying Γ in lexicographical order very similar to the one used for enumerating the transversals of a graph (see the proof of Theorem 8).

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. From G, s and t we build an instance of Enum-IHS-Sat as follows. We consider a set a clauses $\mathcal{C} = \mathcal{P} \cup \mathcal{B}$, where $\mathcal{P} = \{(s \vee t)\}$ and $\mathcal{B} = \{(\neg s)\} \cup \{(x \vee \neg y) \mid (x, y) \in A\}$. This is an $\mathbf{AC}^0$-reduction.

Observe that there exists a path from s to t if and only if Γ is unsatisfiable. Suppose that Enum-IHS-Sat $\in \mathbf{Del}\cdot\mathbf{AC}^0$, this means in particular that outputting a first assignment satisfying Γ or deciding there is none is in $\mathbf{AC}^0$. Thus the above reduction shows that st-connectivity is in $\mathbf{AC}^0$, thus contradicting the fact that st-connectivity is known not to be in $\mathbf{AC}^0$ (see [12, 1]). ◀

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 14.** Enum-Krom-Sat $\in \mathbf{Del_P}\cdot\mathbf{AC}^0 \setminus \mathbf{Del}^*\cdot\mathbf{AC}^0$.

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

Let Γ 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 $V \cup \{\bar{v} : v \in V\}$. For any 2-clause $(l \vee l')$ in Γ there are two arcs $\bar{l} \to l'$ and $\bar{l'} \to l$ in G.
- For each literal l compute $\mathrm{tc}(l)$ the set of vertices that are reachable from l in G.
- Compute the set of strongly connected components of G. If no contradiction is detected, that is if no strongly connected component contains both a variable x and its negation, then contract each strongly connected component into one vertex. The result of this operation is a DAG, which, by abuse of notation, we also call G.
- Compute a topological ordering of the vertices of G.
- In searching through this topological ordering, build an ordered sequence M of n literals corresponding to the first occurrences of each variable.

If the set of clauses is satisfiable, one can enumerate the satisfying assignments given as truth assignments on M in lexicographic order. The enumeration process is similar in spirit as the one developed in the preceding theorem.

For the lower bound, the proof given in Theorem 13 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 Γ over a set of variables V
Output: an enumeration of all assignments over V that satisfy Γ

If we allow a polynomial precomputation step, then we obtain an $\mathbf{AC}^0$-delay enumeration algorithm for this problem that uses constant memory. Interestingly this algorithm relies on the efficient enumeration of binary words in a Gray code order that we have seen in the previous section and contrary to the satisfiability problems studied so far does not provide an enumeration in lexicographic order.

▶ **Theorem 15.** Enum-XOR-Sat $\in \mathbf{Del}^{\mathrm{c}}_{\mathbf{P}} \cdot \mathbf{AC}^0 \setminus \mathbf{Del}^* \cdot \mathbf{AC}^0$.

Proof sketch. Observe that a set of XOR-clauses Γ 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 Γ comes down to enumerating all solutions of the corresponding linear system.

As a precomputation step we apply Gaussian elimination in order to obtain an equivalent triangular system. If the system has no solution, we indicate this as required by Definition 1. 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 $x_1, \dots, x_k$ are free variables, whose assignment determines the assignment of all other variables in the triangular system. We then compute a first solution s_0 corresponding to $x_1, \dots, x_k$ assigned $0 \dots 0$. Next, for each $i = 1, \dots, k$ compute the solution s_i corresponding to all variables in $x_1, , \dots x_k$ 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 we start our enumeration procedure, which uses the enumeration of binary prefixes of length k in a Gray code order as a subprocedure. ◀

5 Separations of Delay Classes

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 the exception of the class with unbounded 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.

5.1 Unconditional Separations for Classes without Precomputation

▶ **Theorem 16.** $\mathbf{Del}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0$

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:

- $A = \{\, y \in \{0,1\}^* \big| |y| = m, y \neq 0^m, y \neq 1^m \}$
- $B = \{\, 1^m \}$ 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 $R_L \in \mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0$, 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 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 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 $R_L \in \mathbf{Del}\cdot\mathbf{AC}^0$ 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 $\mathbf{AC}^0$ circuit family for parity, contradicting [12, 1]. ◀

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

▶ **Theorem 17.** $\mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0$

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 $\mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0$ (the fact there is only one solution makes memory useless). It is obviously in $\mathbf{DelayP}$. This implies that $\mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{DelayP}$. Putting all the previous results together, we conclude:

▶ **Corollary 18.** $\mathbf{Del}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{c}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{Del}^{\mathrm{P}}\cdot\mathbf{AC}^0 \subsetneq \mathbf{DelayP}$.

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 $\mathbf{Del_P} \cdot \mathbf{AC}^0$ and $\mathbf{Del^c_P} \cdot \mathbf{AC}^0$. Note that in Theorem 10 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 of the result below relies on a characterization of the class **PSPACE** in terms of regular leaf-languages or *serializable computation* [13, 19].

▶ **Theorem 19.** *If* $\mathbf{NP} \neq \mathbf{PSPACE}$*, then* $\mathbf{Del^c} \cdot \mathbf{AC}^0 \setminus \mathbf{Del_P} \cdot \mathbf{AC}^0 \neq \emptyset$.

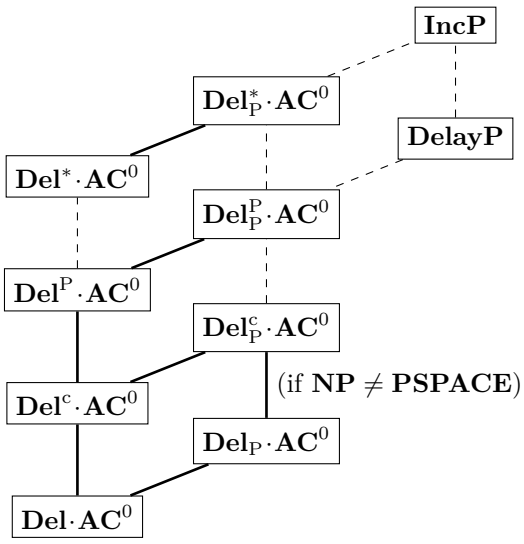

Figure 1 Diagram of the classes. Bold lines denote strict inclusions.

6 Conclusion

The obtained inclusion relations among the classes we introduced are summarized 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 [17, 6]. 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. 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 $\mathbf{Del_P} \cdot \mathbf{AC}^0$. This means satisfiable assignments can be enumerated very efficiently, i. e., by an $\mathbf{AC}^0$-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. Since HORN-SAT is **P**-complete, we conclude that ENUM-HORN-SAT $\notin \mathbf{Del}^* \cdot \mathbf{AC}^0$, and we conjecture that it is not in $\mathbf{Del_P} \cdot \mathbf{AC}^0$. In fact, we do not see any reasonable better bound than the known **DelayP**.

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 *Computer Science Logic (CSL)*, pages 208–222, 2007.
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. doi: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. doi:10.1007/978-3-662-04943-3.
8. 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.
9. 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. doi:10.1016/j.dam.2019.02.025.
10. 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.
11. 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.
12. 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.
13. 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.
14. David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988. doi:10.1016/0020-0190(88)90065-8.
15. Donald L. Kreher and Douglas Robert Stinson. *Combinatorial Algorithms: generation, enumeration, and search*. CRC Press, 1999.
16. 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.

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

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

19 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.

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