Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 250 (2026)

Input-oblivious complexity classes in the second level of the Polynomial Hierarchy*

Edward A. Hirsch† Ilya Volkovich‡ Ryan Williams§

October 21, 2026

Abstract

We continue to investigate input-oblivious computations and build a map of known and new complexity classes. In particular,

- We prove a new upper bound for $\mathbf{OL_2^P}$ defined by Hirsch and Volkovich (STACS, 2026), which is a natural input-oblivious version of the $\mathbf{L_2^P}$ class of Korten and Pitassi (FOCS, 2024): it is captured by symmetric (shared-witness) input-oblivious unambiguous computations with one query to a nondeterministic input-oblivious computation. We introduce the corresponding class **SOUP** and prove that it is closed under polynomial-time Turing reductions.
- We give an equivalent characterization of $\mathbf{OL_2^P}$: a new symmetric input-oblivious class **OMIN**. We prove that **OMIN** contains hard functions against fixed-polynomial size circuits.
- We prove a number of query aggregation results showing a number of containments and equivalences between input-oblivious complexity classes. In particular, we prove that $\mathbf{BPP^{OL_2^P}} = \mathbf{OL_2^P}$ and $\mathbf{O_2^P} = \mathbf{ONP^{prONP}}$. We also characterize a number of input-oblivious classes using the dot-operator.
- We extract a number of derandomization corollaries from our results: an upper bound for $\mathbf{BPP^{ONP}}$, as well as the new characterization $\mathbf{OMA} = \mathbf{BP \cdot prONP}$, and new upper bounds for **MA**, **OMA**, $\mathbf{P^{prMA}}$, $\mathbf{P^{OprMA}}$ (we define the latter class and prove a Karp–Lipton-style collapse for it). These improve and extend earlier bounds of Chakaravarthy and Roy (STACS, 2006) and Hirsch and Volkovich (STACS, 2026).

AI Usage Statement

No generative AI tools (including LLMs) have been used in proving the results or writing the first revision of this report, except for using ChatGPT for proofreading a preliminary version of Theorem 3.4. After the first revision was published, Claude found a discrepancy between a definition and a proof in a *previous paper*, which resulted in updating Definition 2.4 and Proposition 6.9 (the authors remain responsible for this material as well).

*The authors would like to thank the Isaac Newton Institute for Mathematical Sciences, Cambridge, for support and hospitality during the programme *Logical foundations of complexity theory*, where a part of work on this paper was undertaken. This programme was supported by EPSRC grant EP/Z000580/1.

†Department of Computer Science, Ariel University, Israel. Email: edwardh@ariel.ac.il . This research was conducted with the support of the State of Israel, the Ministry of Immigrant Absorption, and the Center for the Absorption of Scientists. This research was supported by The Israel Science Foundation (grant No. 1645/26).

‡Computer Science Department, Boston College, Chestnut Hill, MA. Email: ilya.volkovich@bc.edu

§EECS and CSAIL, MIT. Work supported in part by NSF CCF-2420092.

ISSN 1433-8092

Contents

1 Introduction **3**
- 1.1 Motivation . . . 3
- 1.2 Background . . . 3
- 1.3 Our contribution . . . 6
- 1.4 Overview of our paper . . . 8
  - 1.4.1 The Korten–Pitassi linear order and a new single-minimum class . . . 8
  - 1.4.2 Dot-operators, query aggregation, and the oblivious hierarchy . . . 9
  - 1.4.3 Same-witness symmetric input-oblivious unambiguous classes and derandomization . . . 10
  - 1.4.4 Trading $\mathbf{OL}_2^{\mathrm{P}}$ for **ONP**, and tighter upper bounds for **MA** and **OMA** . . . 10
  - 1.4.5 Derandomization of **BPP**-based classes . . . 11

2 Definitions and Basics **11**
- 2.1 Notation agreement and a technical lemma about linear orders . . . 11
- 2.2 Input-oblivious nondeterministic and symmetric computations . . . 12
- 2.3 Input-oblivious unambiguous computations . . . 13
- 2.4 Dot operators . . . 14
- 2.5 Computations based on the Linear Ordering Principle . . . 15
  - 2.5.1 Known classes (basic and input-oblivious) . . . 15
  - 2.5.2 "Half-oblivious" linear ordering . . . 16
  - 2.5.3 A new input-oblivious linear order class . . . 16

3 $\mathbf{OL}_2^{\mathrm{P}} = \mathbf{OMIN}$ and contains hard functions **17**

4 Query aggregation **19**
- 4.1 $\mathbf{OL}_2^{\mathrm{P}}$ is polynomial-time Turing-closed . . . 19
- 4.2 **SOUP** is polynomial-time Turing-closed . . . 20
- 4.3 Aggregating **ONP** queries using the dot-operator . . . 20
- 4.4 Aggregating **prONP** queries, $\mathbf{O}_2^{\mathrm{P}}$ and $\mathbf{NO}_2^{\mathrm{P}}$ characterization, and quantifier swapping . . . 22

5 Unique witnesses with an NP (or ONP) oracle capture the Linear Ordering Principle **24**
- 5.1 Warm-up: The non-oblivious case . . . 24
- 5.2 Symmetric input-oblivious unambiguous computations . . . 24
  - 5.2.1 Input-oblivious unique witnesses: the main theorem . . . 24
  - 5.2.2 Corollaries for $\mathbf{OL}_2^{\mathrm{P}}$ and **ONP** oracles . . . 25

6 New upper bounds for MA- and BPP-based classes **25**
- 6.1 Basic containments of **MA** . . . 26
- 6.2 Corollaries for classes that use Merlin–Arthur oracles . . . 29
- 6.3 Corollaries for probabilistic oracle computations . . . 30

1 Introduction

1.1 Motivation

Two types of computational models have been central to complexity theory, which we may roughly classify as "uniform" models and "non-uniform" models. Uniform models are constant-sized programs that must compute correctly on every input; non-uniform allow for a separate "computer" C_n for each input length n. When the size of C_n and its computation time is $\mathrm{poly}(n)$, the decision problems we can capture with such a model is precisely $\mathbf{P/poly}$. As the choice of computer C_n *only* depends on the input length n and no further details about the input, we say that the computational model is "input-oblivious".

A long line of work has studied the question of what *other* aspects of computation may be input-oblivious, such as data movement (namely, oblivious Turing machines and RAMs [PF79, GO96, BP25]) and the lengths of proofs in various types of proof systems [CR06, Aar07, FSW09, AD14, GM15, GLV24, HV26]. For example, how feasibly can we give short proofs of statements, when the proof does not depend on the specific claim being proved, but only the *length* of the claim? The class of languages with such proofs is modeled by the class **ONP** [FSW09], which is easily seen to be contained in $\mathbf{P/poly}$.

Input-obliviousness provides a useful bridge between uniform and non-uniform models; this is an important bridge, since non-uniform models are generally incredibly more powerful than their uniform counterparts.[1] We can often restate non-uniform questions purely in terms of uniform ones, via input-obliviousness: for example, we know $\mathbf{NP} \subset \mathbf{P/poly}$ if and only if $\mathbf{NP} = \mathbf{ONP}$. As a result, input-oblivious classes naturally arise when we consider the following fundamental question:

> *What are the* **minimal** *uniform complexity classes containing functions that do not have small non-uniform circuits?*

In this paper, we study (and define) a wide range of new input-oblivious complexity classes, where the modes of computation depend only on input length. As a result, we uncover new fixed-polynomial circuit lower bounds for uniform classes, and make new connections with (and between) other well-known complexity classes (oblivious and not).

1.2 Background

Fixed-polynomial lower bounds, Karp–Lipton–style collapses, and symmetric alternation. Kannan [Kan82] proved that for every k, there are languages in the polynomial hierarchy ($\mathbf{PH}$) that cannot be computed by Boolean circuits of size $O(n^k)$: in symbols, $\Sigma_4^p \cap \Pi_4^p \not\subseteq \mathbf{Size}[n^k]$. This statement was strengthened using a collapse result proved by Karp and Lipton [KL80]: if $\mathbf{NP} \subseteq \mathbf{P/poly}$, then $\mathbf{PH} = \Sigma_2^p \cap \Pi_2^p$, and thus $\Sigma_2^p \cap \Pi_2^p \not\subseteq \mathbf{Size}[n^k]$ follows by a "win-win" argument (if $\mathbf{NP} \subseteq \mathbf{P/poly}$, then a collapse occurs and the new bounds follow from Kannan's theorem, otherwise there is nothing to do). A long chain of works [BCG+96, KW98, CR06, Cai07, CR11] followed, proving new *Karp–Lipton–style collapses* and thus eventually improving these bounds to $\mathbf{P}^{\mathbf{prMA}}$ [CR11], where **prMA** is a promise[2] version of the Merlin–Arthur class.

[1] For instance, non-uniform models can compute certain *undecidable* languages, by encoding their answers into the computers C_n.

[2] A promise problem $\Pi = (\Pi_{\mathrm{YES}}, \Pi_{\mathrm{NO}})$ is a relaxation of a language where $\Pi_{\mathrm{YES}} \cap \Pi_{\mathrm{NO}} = \emptyset$ and one needs to solve the problem correctly on the inputs in the promise set $\Pi_{\mathrm{YES}} \cup \Pi_{\mathrm{NO}}$. In a promise problem complexity class, the definition of the computational device has to be respected on the promise set only.

An important class on that list is $\mathbf{S_2^P}$, defined in [RS98, Can96]. This class captures the so-called "symmetric alternation". More specifically, a language L belongs to $\mathbf{S_2^P}$ if there exists a polynomial-time "referee" $V(x, y, z)$ such that every input x has *irrefutable certificates*. That is, $x \in L \iff \exists y_* \forall z\ V(x, y_*, z) = 1$ and $x \notin L \iff \exists z_* \forall y\ V(x, y, z_*) = 0$. Note that the order of quantifiers here is different from that in Σ_2^p. The class is positioned between $\mathbf{P^{prMA}} \subseteq \mathbf{S_2^P} \subseteq \mathbf{ZPP^{NP}} \subseteq \Sigma_2^p \cap \Pi_2^p$.

However, this "win-win" approach could not be extended to input-oblivious classes, since they typically do not contain **NP** (as otherwise **PH** would collapse). Indeed, for about two decades, there was a gap between the best Karp–Lipton-style collapse and the smallest class for which fixed-polynomial lower bounds were known. In particular, in [CR06], Chakaravarthy and Roy defined the class $\mathbf{O_2^P}$ as the input-oblivious version of $\mathbf{S_2^P}$. That is, rather than having a (potentially different) irrefutable certificate for every input, there is a pair of irrefutable "yes" and "no" certificates y_n, z_n *for each input length* n. In the same paper, it was shown that the collapse goes all the way to $\mathbf{O_2^P} \subseteq \mathbf{S_2^P}$. Yet, for the reason mentioned above, the best fixed-polynomial lower bound remained at $\mathbf{S_2^P}$ until the recent progress on *the Range Avoidance problem* (that was originally known under the name *the dual weak pigeon-hole principle*, dWPHP), which eventually led to $\mathbf{O_2^P} \not\subseteq \mathbf{Size}[n^k]$ [GLV24]. More recently, [HV26] deepened the collapse further to $\mathbf{P^{prOMA}} \subseteq \mathbf{O_2^P}$, yet without matching fixed-polynomial lower bounds.

In a related earlier line of work, Santhanam [San09] established fixed-polynomial lower bounds for **prMA**. This result is somewhat different in nature, since all the classes discussed above (including $\mathbf{P^{prMA}}$) consist of *languages*, whereas **prMA** is a class of *promise problems*. In particular, we note there is no *immediate* way to transfer a lower bound against **prMA** to a seemingly "nearby" language class, such as **MA**, **AM** or even $\mathbf{P^{prMA}}$. For further discussion of promise problems and how they differ from languages, see [GS88, BF99a, Gol06].

Range Avoidance and the Linear Ordering Principle. The Range Avoidance problem (Avoid for short) is a total search problem, where we are given a Boolean circuit $C\colon \{0,1\}^n \to \{0,1\}^m$ for $m > n$ (one can think about popular parameters $m = n + 1$ or $m = 2n$), and we are interested in finding a (certainly existing) point outside its image. Recently, Avoid has received much attention [KKMP21, Kor22, CHR24, CHLR23, Li24, CHLR26, GLS26, FGJ+26] partially rediscovering earlier works in logic (see e.g. [PWW88, Jeř04] and the survey [Kra25]). A number of important problems have been reduced to it, most notably computing the truth table of a hard function (given 1^{2^n}, output a truth table of a function $\{0,1\}^n \to \{0,1\}$ of circuit complexity at least $2^{n/2 \log n}$) and PRG (given 1^n, output a pseudorandom generator against n-input circuits of size n) [Kor22].

Another relevant problem, previously known as MIN in the bounded arithmetic community [CK98] (find the smallest element of a linear order), has also gained renewed attention recently following the introduction of the class $\mathbf{L_2^P}$ by Korten and Pitassi [KP24], based on the *Linear Ordering Principle* (LOP for short). The LOP problem asks to find either the smallest element or a breach of the linear order axioms in an implicitly described ordering relation, typically given by a Boolean circuit. This class can be viewed as a particular case of $\mathbf{S_2^P}$: now the "referee" compares not just a *yes-certificate* against a *no-certificate*, but it must be able to compare *any two certificates* according to a linear order (a certificate now starts with the answer bit, certificates of the same polarity are also compared between themselves). As a result, we obtain that $\mathbf{L_2^P} \subseteq \mathbf{S_2^P}$: the irrefutable certificate is the minimum of the linear order. Recently, this class was shown to lie between two known classes, $\mathbf{P^{prMA}} \subseteq \mathbf{L_2^P} \subseteq \mathbf{P^{prSBP}}$ [HV26], and was placed on the map of total functions by showing that $\mathbf{L_2^P} \subseteq \mathbf{UEOPL^{NP}}$ [GGLS25].

Input-obliviousness and derandomization. One particular significance of LOP and $\mathbf{L}_2^{\mathbf{P}}$ is that Avoid and PRG reduce to them, as was shown by Korten and Pitassi [KP24]. On the other hand, many prominent applications of Avoid actually require solving the uniform, or input-oblivious, version of the problem. For instance, a single PRG or a single hard truth table suffices for all inputs of the same length. In [HV26], it was observed that the above argument carries over to the oblivious setting. Motivated by this observation, Hirsch and Volkovich defined a new class $\mathbf{OL}_2^{\mathbf{P}}$ as the input-oblivious version of $\mathbf{L}_2^{\mathbf{P}}$. Namely, for each input length n, it requires two input-oblivious minima y_n, z_n of the linear order — for positive instances and negative instances, respectively. They further observed that PRG reduces to $\mathbf{OL}_2^{\mathbf{P}}$. This observation, in turn, led to new containment results: $\mathbf{P}^{\mathbf{prMA}} \subseteq \mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{NP}}$ and $\mathbf{P}^{\mathbf{prOMA}} \subseteq \mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{prONP}}$.

Hierarchies, dot-operators, and half-oblivious classes. A number of other input-oblivious classes have been defined [CR06, FSW09, GM15, Aar07, AD14]. Yet, our understanding of their computational power remains limited. Even the most basic question — what problems can be solved in **ONP**? — remains open. All we know is that **ONP** contains **RP** [GM15], as well as problems arising from computational Ramsey theory [GLV24].

Similarly to the polynomial hierarchy that is based on **NP**, one can define the *random hierarchy* [Zac88] based on **RP** and the *oblivious hierarchy* (call it **OH**) based on **ONP**:

$$\begin{aligned}
\mathbf{RH} &\triangleq \quad \mathbf{RP} \cup \mathbf{RP}^{\mathbf{RP}} \cup \mathbf{RP}^{\mathbf{RP}^{\mathbf{RP}}} \ldots \quad \subseteq \mathbf{BPP}, \\
\mathbf{OH} &\triangleq \mathbf{ONP} \cup \mathbf{ONP}^{\mathbf{ONP}} \cup \mathbf{ONP}^{\mathbf{ONP}^{\mathbf{ONP}}} \ldots \subseteq \mathbf{O}_2^{\mathbf{P}}.
\end{aligned}$$

It is unknown whether either hierarchy collapses or is equal to its natural upper bound, **BPP** and $\mathbf{O}_2^{\mathbf{P}}$, respectively. These upper bounds follow from the containments $\mathbf{RP} \subseteq \mathbf{BPP}$ and $\mathbf{ONP} \subseteq \mathbf{O}_2^{\mathbf{P}}$, together with the self-lowness of **BPP** [Zac82] and $\mathbf{O}_2^{\mathbf{P}}$ [CR06].

Partial progress regarding the random hierarchy was made in [BF99a]. They showed that replacing the **RP** oracle with an oracle for the corresponding promise problem class, **prRP**, suffices to recover **BPP**. More precisely, a single oracle query suffices: $\mathbf{BPP} = \mathbf{RP}^{\mathbf{prRP}[1]}$. On the other hand, they also established an oracle separation between $\mathbf{RP}^{\mathbf{RP}}$ and **BPP**. One can ask the same questions about **OH**.

While defining a hierarchy based on a self-low class would be pointless, as such a hierarchy would collapse, Chakaravarthy and Roy [CR06] introduced the *oblivious symmetric alternation hierarchy* (**OSH**), based on a "half-oblivious" class $\mathbf{NO}_2^{\mathbf{P}}$ that they defined. In $\mathbf{NO}_2^{\mathbf{P}}$, the obliviousness property is required to hold only for negative instances, while the yes-certificates may be arbitrary. They also proved that $\mathbf{MA} \subseteq \mathbf{NO}_2^{\mathbf{P}}$.

In fact, [CR06] introduces $\mathbf{O}_2^{\mathbf{P}}$ and $\mathbf{NO}_2^{\mathbf{P}}$ using *dot-operators* $\mathbf{O}_2^{\mathbf{P}\bullet}$ and $\mathbf{NO}_2^{\mathbf{P}\bullet}$. The history of dot-operators started with classical nondeterministic computation. Classes of the (classical) polynomial hierarchy have complete problems that completely characterize them in terms of quantifiers. Wagner [Wag86] defined the first dot-operators as "quantifiers for classes": for example, for a class of languages $\mathcal{C}$, one can define a new class $\exists \cdot \mathcal{C}$ consisting of languages $\{x : \exists w \in \{0,1\}^{p(|x|)}\ (x,w) \in R\}$ for some language $R \in \mathcal{C}$ and a polynomial p. In particular, all oracle queries in a class of the polynomial hierarchy can be aggregated into a single query of the polarity opposite to the polarity of the basic machine, for example, $\mathbf{NP}^{\mathbf{NP}} = \exists \cdot \mathbf{co\text{-}NP}$. Later a number of dot-operators for different types of computations have been defined [ZF87, Zac88, BS01, BGM06, CR06].

Untrusted advice and two-stage computations. Aaronson and Drucker [Aar07, AD14] studied the notion of "untrusted" advice. To this end, they defined the class "Yaroslav–Percival" **YP**, consisting of languages L decided by a "zero-error" algorithm with advice. That is, for

every input length n, there exists an advice string s_n such that, given s_n, the algorithm outputs the correct answer on *all* inputs of length n (*both positive and negative*). Yet, for *any* advice string, the algorithm either outputs the correct answer or returns "I don't know". It is not hard to see that this class is equal to $\mathbf{ONP} \cap \mathbf{co\text{-}ONP}$. In particular, s_n will serve as an oblivious witness for both L and its complement! They also defined $\mathbf{YP}^*$, a subclass of $\mathbf{YP}$ in which a purported advice string s can be verified efficiently and obviously, that is, without access to the input x; then the second stage gives the final answer using both s and x.

1.3 Our contribution

In this paper, we continue to investigate input-oblivious classes, with the goal of identifying and mapping the classes within the second level of the polynomial hierarchy and, in particular, the exact position of $\mathbf{OL_2^P}$. Our contribution is threefold:

- We give a new equivalent definition to $\mathbf{OL_2^P}$, identify this new class $\mathbf{OMIN} = \mathbf{OL_2^P}$ as the currently smallest (input-oblivious) class possessing fixed-polynomial lower bounds, and investigate its structural properties.
- We define a new symmetric witness-sharing class **SOUP**, investigate it and use it for proving an upper bound $\mathbf{OL_2^P} \subseteq \mathbf{SOUP}^{\mathbf{ONP}[1]}$ and similar-spirit upper bounds for several other previously known classes.
- Last but not least, we give new characterizations and containment results for a number of input-oblivious classes and also for a few related half-oblivious (such as $\mathbf{NO_2^P}$) and non-oblivious (such as **MA**) classes.

(See the overview of class containments in Fig. 1, and a more detailed overview for classes based on **ONP** and **NP** is given later in Fig. 2.)

Input-oblivious class(es) based on the Linear Ordering Principle (LOP) and their new upper bound class. We begin by proposing a stricter yet equivalent reformulation for the class $\mathbf{OL_2^P}$ introduced in [HV26]. We denote this class by **OMIN** (Theorem 3.4). As our first result, we establish fixed-polynomial lower bounds for this class (Theorem 3.3).

More broadly, this new reformulation helps characterizing this class in terms of symmetric input-oblivious unambiguous computations equipped with nondeterministic input-oblivious oracles. To this end, we introduce the class of input-oblivious unambiguous computations (**OUP**) by requiring a nondeterministic polynomial-time machine to have the same unique witness for all positive instances of the same length (thus, for each input length, there can only be one witness that this machine accepts). Since $\mathbf{OL_2^P} = \mathbf{OMIN}$ is a symmetric class, the upper bounds we consider naturally take the form of intersections with their complements, such as $(\mathbf{OUP} \cap \mathbf{co\text{-}OUP})^{\mathbf{ONP}}$ to begin with. We then show that the unique oblivious witness can be made the same for both positive and negative instances: we introduce the class **SOUP** as a same-witness version of $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$ and prove that $\mathbf{OL_2^P} \subseteq \mathbf{SOUP}^{\mathbf{ONP}[1]}$ (Theorem 5.2).

Observe that the newly introduced class **OMIN** and the class $\mathbf{YP}^*$ of [AD14] share a common feature: the computation is divided into two stages: first, an input-oblivious procedure verifies a witness/minimum (for **OMIN**, a comparison algorithm that depends only on the input length identifies the minimum, whereas in $\mathbf{YP}^*$, the validity of the advice string is verified based only on the input length). The verified witness/minimum is then used to produce the answer. Additionally, one can also view **SOUP** as an *unambiguous* version of **YP** and, in fact, of $\mathbf{YP}^*$. To see this, consider a variant of **YP** in which, for each input length n, there is a *unique* correct advice string s_n. By uniqueness, a purported advice string s can be verified without access to

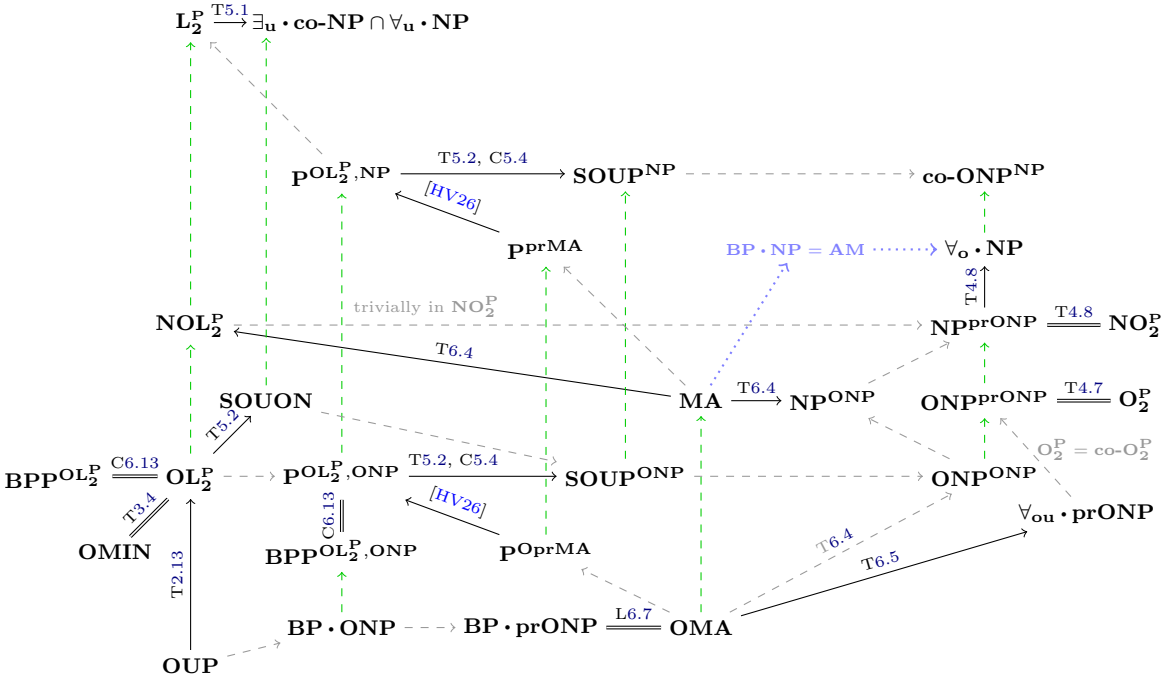


Figure 1: Overview of containments. Legend: solid arrow or equivalence — interesting containment, dashed arrow — this containment is trivial or follows from other results, dotted arrow — not discussed in the paper, shown to give a better perspective. The full version contains more details.

the input x: simply run the procedure on $x = 0^n$. The output will be different from "I don't know" if and only if s is the correct advice string. Indeed, since there is only one correct advice string for each input length, any advice string that passes this verification must be that unique string.

Further structural properties of $\mathbf{OL_2^P}$ and SOUP and their applications to derandomization. We begin by proving that $\mathbf{OL_2^P}$ and $\mathbf{SOUP}^A$ are closed under polynomial-time Turing reductions (Theorems 4.1 and 4.3), where the latter holds for any oracle A. It implies upper bounds for two intermediate classes that appear in several derandomization results: $\mathbf{P^{OL_2^P,ONP}} \subseteq \mathbf{SOUP^{ONP}}$ and $\mathbf{P^{OL_2^P,NP}} \subseteq \mathbf{SOUP^{NP}}$.

Subsequently, in Corollary 6.13 we prove that $\mathbf{OL_2^P}$ is also closed under randomized reductions, that is, $\mathbf{BPP^{OL_2^P}} = \mathbf{OL_2^P}$. Note that prior to our work, it was known that $\mathbf{L_2^P}$ is closed under deterministic polynomial-time Turing reductions [KP24], but whether the analogous closure property holds for $\mathbf{OL_2^P}$ was open. Indeed, establishing the stronger statement $\mathbf{BPP^{L_2^P}} = \mathbf{L_2^P}$ would imply that $\mathbf{AM} \subseteq \Sigma_2^p$, thereby resolving a longstanding open problem. Going back to our results, we actually prove a stronger statement: $\mathbf{BPP}^{A,\mathbf{OL_2^P}} = \mathbf{P}^{A,\mathbf{OL_2^P}}$ for every oracle $A \in \mathbf{P/poly}$. In particular, this implies that $\mathbf{BPP^{OL_2^P,ONP}} = \mathbf{P^{OL_2^P,ONP}}$. Therefore, $\mathbf{SOUP^{NP}}$ contains $\mathbf{P^{prMA}}$ while $\mathbf{SOUP^{ONP}}$ contains $\mathbf{BPP^{OL_2^P,ONP}}$ and $\mathbf{P^{OprMA}}$, where **OprMA** is a restricted version of **prOMA** that we define. We also prove that $\mathbf{P^{OprMA}}$ is the smallest Karp–Lipton-style collapse class.

Symmetric input-oblivious and half-oblivious classes and the power of input-oblivious promise oracles. To understand better the relative position of $\mathbf{SOUP^{ONP}}$

(and thus $\mathbf{OL}_2^{\mathbf{P}}$) in the complexity landscape, we introduce the *input-oblivious hierarchy* **OH** by replacing the role of **NP** in the polynomial hierarchy with **ONP**. Note that it is not even clear whether $\mathbf{O}_2^{\mathbf{P}}$, defined in [CR06] as the input-oblivious version of $\mathbf{S}_2^{\mathbf{P}}$, is contained in **OH**. Motivated by these questions, we investigate some of the natural complexity classes that sit at the second level of **OH** and their relationships with "half-oblivious" classes.

Our results characterize the classes $\mathbf{O}_2^{\mathbf{P}}$ and $\mathbf{NO}_2^{\mathbf{P}}$ by proving that $\mathbf{O}_2^{\mathbf{P}} = \mathbf{ONP}^{\mathbf{prONP}}$ (Theorem 4.7) and $\mathbf{NO}_2^{\mathbf{P}} = \mathbf{NP}^{\mathbf{prONP}}$ (Theorem 4.8). Along the way, we recover a lowness result that is interesting in its own right: $\mathbf{prO}_2^{\mathbf{P}}$ is low for $\mathbf{O}_2^{\mathbf{P}}$, $\mathbf{NO}_2^{\mathbf{P}}$, and $\mathbf{S}_2^{\mathbf{P}}$ (Theorem 4.6).

We also prove several dot-operator versions of our results. In particular, we show that oracle queries to **ONP**, and even to **prONP**, can be aggregated, provided that the base class is "sufficiently powerful". It is worth noting that while query aggregation within the levels of the polynomial hierarchy is a classical result, it does not immediately carry over to input-oblivious setting. Finally, we establish the containment $\mathbf{NO}_2^{\mathbf{P}} = \mathbf{NP}^{\mathbf{prONP}} \subseteq \mathbf{co\text{-}ONP}^{\mathbf{NP}}$ (and an even stronger containment in the dot-operator notation), which demonstrates a surprising quantifier-swapping phenomenon for the input-oblivious universal quantifier (dot operator).

Tighter containments for Merlin-Arthur classes. Even though $\mathbf{OL}_2^{\mathbf{P}}$ helps derandomize large classes, such as $\mathbf{P}^{\mathbf{prMA}}$, we can obtain tighter results for **MA** and **OMA**. Chakaravarthy and Roy [CR06] showed that $\mathbf{MA} \subseteq \mathbf{NO}_2^{\mathbf{P}}$ and $\mathbf{OMA} \subseteq \mathbf{O}_2^{\mathbf{P}}$. In Theorem 6.4, we strengthen the former containment in two incomparable ways. First, we show that $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{OL}_2^{\mathbf{P}}} \subseteq \mathbf{NP}^{\mathbf{ONP}}$ (recall that we already established $\mathbf{NO}_2^{\mathbf{P}} = \mathbf{NP}^{\mathbf{prONP}}$). Second, we show that $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{OL}_2^{\mathbf{P}}} \subseteq \mathbf{NOL}_2^{\mathbf{P}}$, where $\mathbf{NOL}_2^{\mathbf{P}} \subseteq \mathbf{NO}_2^{\mathbf{P}}$ is a half-oblivious analogue of $\mathbf{OL}_2^{\mathbf{P}}$, introduced in a similar fashion to $\mathbf{NO}_2^{\mathbf{P}}$.

For the oblivious class **OMA**, we characterize it as $\mathbf{OMA} = \mathbf{BP} \cdot \mathbf{prONP}$. This stands in sharp contrast with the non-oblivious case where $\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP}$ [ZF87, Sch89]. Even though our result that $\mathbf{P}^{\mathbf{OprMA}} \subseteq \mathbf{SOUP}^{\mathbf{ONP}}$ already puts **OMA** in $\mathbf{OUP}^{\mathbf{ONP}}$, we prove a tighter containment in terms of the dot-operator (Theorem 6.5). This, in turn, leads to a surprising outcome (Corollary 6.8): when applied to promise **co-ONP** problems, the randomized dot-operator $\mathbf{BP}\cdot$ can be replaced by the universal unambiguous input-oblivious operator!

1.4 Overview of our paper

1.4.1 The Korten–Pitassi linear order and a new single-minimum class

Korten and Pitassi [KP24] defined a polynomial-time decidable linear ordering such that its minimum contains a solution to Avoid. Since [Li24, Kor22] reduced the problem of finding a hard truth table to Avoid, we can take the relevant instance of Avoid and then its solution will be the minimum; one can easily extract the hard truth table from this string (for our purpose of lower bounds of size-n^k circuits, this truth table will list the answers for the first n^{2k} inputs, so it won't be too large). We want our language to correspond to a specific hard function, thus this truth table is the same for every input of the same length n, so the Avoid instance and the minimum of the linear order are input-oblivious. It gives rise to the following class **OMIN**: its computational model consists of

1. A polynomial-time decidable linear order that depends on the length of the input (but not on the input itself).

2. A polynomial-time deterministic "decoder" that, given both the input and the minimum of this linear order, outputs the accept/reject decision.

Formalizing the above argument shows fixed-polynomial lower bounds for **OMIN**:

Theorem (Theorem 3.3). *For every $k \geq 1$, there is an $L_k \in \mathbf{OMIN}$ that does not have circuits of size kn^k.*

We further prove that this class is in fact an equivalent simplified definition of $\mathbf{OL_2^P}$:

Theorem (Theorem 3.4). $\mathbf{OMIN} = \mathbf{OL_2^P}$.

The difference between the two definitions is that in $\mathbf{OL_2^P}$ only the minimum is input-oblivious, while the ordering may depend on the input; moreover, there are two different input-oblivious minima: the minimum for yes- and the minimum for no-instances. We then set up a *single* linear ordering in which the minimum will contain *both* minima; more exactly it will contain the minimum R_n for the input 0^n and the minimum S_n for inputs that have the answer opposite to that for 0^n.

A subtle difficulty is that $\mathbf{OL_2^P}$ uses the input in order to compare the elements; it would be OK to use a different distinguished input of the same polarity (thus making the comparison input-oblivious), but what if it does not exist? We overcome this obstacle by including a possible input string y (this is *not* our real input string) to the elements that we compare (thus, we compare triples: a candidate R_n, a possible input y, and a candidate S_n), and the order will prefer lexicographically smaller possible inputs. As a result, in the case where the language has the same answer for every input of length n, our minimum will still contain R_n.

We prove that this class has a useful property: it is polynomial-time Turing-closed.

Theorem (Theorem 4.1). $\mathbf{P^{OMIN}} = \mathbf{OMIN}$.

We later repeat this proof idea for other input-oblivious classes: a witness (an element of the order) for the new machine will include all potential oblivious witnesses for the oracle machine (since the basic machine is polynomial time, it can make queries for a polynomial number of lengths, thus a polynomial number of witnesses suffices). Since by definition of **OMIN** the linear order is input-oblivious, we do not need actual queries in order to compare each pair of the respective witnesses, thus the new order will be oblivious as well, and its minimum will provide the minima for the oracle for every potential query length.

We will strengthen this theorem to randomized reductions later (but the new proof will use this theorem as "the basic case").

1.4.2 Dot-operators, query aggregation, and the oblivious hierarchy

Let us take a small detour before proceeding to the applications of **OMIN** and our characterizations of other input-oblivious classes. We first settle several structural questions inside $\mathbf{O_2^P}$, $\mathbf{NO_2^P}$, and the oblivious hierarchy classes contained in $\mathbf{O_2^P}$.

We show that $\mathbf{O_2^P} = \mathbf{ONP^{prONP[1]}}$! In fact, we prove a stronger result generalizing the lowness of the class of *languages* $\mathbf{O_2^P}$ for $\mathbf{O_2^P}$ and $\mathbf{S_2^P}$ [CR06]. Our proof builds on and extends a result of [HV26], which shows that $\mathbf{P^{prO_2^P}} = \mathbf{O_2^P}$:

Theorem (Theorem 4.6). $\mathbf{prO_2^P}$ *is low for* $\mathbf{O_2^P}$, $\mathbf{NO_2^P}$, *and* $\mathbf{S_2^P}$.

This theorem helps us to characterize not only $\mathbf{O_2^P}$ as $\mathbf{O_2^P} = \mathbf{ONP^{prONP}}$ but also $\mathbf{NO_2^P}$ as $\mathbf{NO_2^P} = \mathbf{NP^{prONP}}$. Moreover, we prove that, similarly to the classical polynomial hierarchy, queries to oblivious classes can be aggregated and formulated using the corresponding dot operator. To this end, we introduce the *oblivious dot-operators* $\exists_{\mathbf{o}\bullet}$ and $\forall_{\mathbf{o}\bullet}$ by requiring that the witness be input-oblivious. In addition to proving the aggregation results, we observe a somewhat surprising phenomenon: given two alternating "quantifiers" $\exists y\, \forall_{\mathrm{o}} z$, where the second quantifier is oblivious and the first one is not, one can swap them, yielding a class of the opposite polarity (one can find here also an analogy with $\mathbf{MA} \subseteq \mathbf{AM}$). We prove:

Theorem (Theorems 4.4, 4.7, 4.8)**.**

- $\mathbf{ONP}^{\mathbf{ONP}} = \exists_{\mathbf{o}} \cdot \mathbf{co\text{-}ONP}$ *and* $\mathbf{ONP}^{\mathbf{prONP}} = \exists_{\mathbf{o}} \cdot \mathbf{pr\,co\text{-}ONP} = \mathbf{O}_{\mathbf{2}}^{\mathbf{P}}$.
- $\mathbf{NP}^{\mathbf{ONP}} = \exists \cdot \mathbf{co\text{-}ONP}$ *and* $\mathbf{NP}^{\mathbf{prONP}} = \exists \cdot \mathbf{pr\,co\text{-}ONP} = \mathbf{NO}_{\mathbf{2}}^{\mathbf{P}} \subseteq \forall_{\mathbf{o}} \cdot \mathbf{NP}$.

We remark that since $\mathbf{ONP} \subseteq \mathbf{NP}$ and **NP** is a "syntactic" class, any promise problem in **prONP** can be "completed" to an **NP** language and, as a result, an **prONP** oracle can be replaced with an **NP** oracle.

This is where our detour ends and we return to the applications of **OMIN**.

1.4.3 Same-witness symmetric input-oblivious unambiguous classes and derandomization

The minimum element m of a linear order (see Section 1.4.1) is uniquely determined and can therefore serve as an unambiguous witness. To verify that m is indeed the minimum element, it suffices to check that there is no smaller element, which is a **co-NP** query. Furthermore, m itself can serve as an oblivious proof that a different element m' is not the minimum. This allows us to turn the query into a **co-ONP** query. This results in $\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}} = \mathbf{OMIN} \subseteq \mathbf{OUP}^{\mathbf{ONP}[1]}$. (Note that while one might hope to strengthen this further to $\mathbf{OUP}^{\mathbf{OUP}}$, there is no immediate way to achieve it: if m' is not the minimum, then any element smaller than m' can serve as a witness to its non-minimality and there is no obvious way to select a single distinguished witness.)

Since the classes we are dealing with are symmetric (i.e., closed under complement), we actually end up in $(\mathbf{OUP} \cap \mathbf{co\text{-}OUP})^{\mathbf{ONP}[1]}$. However, there is more to it: since our class **OMIN** has the same minimum for both positive and negative instances, we can assume that the unambiguous witness is *the same single witness* on both sides. This is where our **SOUP** (*same-witness* oblivious unique) class comes into play. It also has a useful property: it is closed under polynomial-time Turing reductions, and this closure property relativizes.

Theorem (Theorem 4.3)**.** *For any oracle A,* $\mathbf{P}^{\mathbf{SOUP}^A} = \mathbf{SOUP}^A$.

The reason is that one can test a candidate unambiguous witness of length i on the input $x = 0^i$: either the **OUP** or the **co-OUP** computation must accept this input. In total,

Theorem (Theorem 5.2 (part))**.** $\mathbf{OMIN} \subseteq \mathbf{SOUP}^{\mathbf{ONP}[1]}$.

Even though in this same theorem we put **OMIN** even deeper by formulating this upper bound using an appropriate dot-operator, the classes $\mathbf{SOUP}^{\mathbf{ONP}}$ and $\mathbf{SOUP}^{\mathbf{NP}}$ turn out to be useful because of the Turing-closure property. We substitute our upper bound for $\mathbf{OMIN} = \mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}$ into the intermediate classes $\mathbf{P}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}},\mathbf{ONP}}$ and $\mathbf{P}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}},\mathbf{NP}}$. Thus we get

$$\mathbf{P}^{\mathbf{OprMA}} \subseteq \mathbf{P}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}},\mathbf{ONP}} \subseteq \mathbf{P}^{\mathbf{SOUP}^{\mathbf{ONP}},\mathbf{ONP}} = \mathbf{P}^{\mathbf{SOUP}^{\mathbf{ONP}}} = \mathbf{SOUP}^{\mathbf{ONP}},$$

and similarly $\mathbf{P}^{\mathbf{prMA}} \subseteq \mathbf{SOUP}^{\mathbf{NP}}$.

1.4.4 Trading $\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}$ for ONP, and tighter upper bounds for MA and OMA

The $\mathbf{SOUP}^{\mathbf{ONP}}$ upper bound for $\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}$ and the closure properties of our classes result in a possibility to replace an $\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}$ oracle by an **ONP** oracle in many cases:

Corollary (Corollary 5.3 (part))**.** $\mathcal{C}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}} \subseteq \mathcal{C}^{\mathbf{ONP}}$*, where* $\mathcal{C} = \mathbf{NP}$*,* **ONP***,* **UP***, or* **OUP**.

The derandomization results of [HV26] mentioned above were obtained by observing that Korten's PRG is input-oblivious and thus can be constructed by querying $\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}$:

Lemma (Lemma 6.2 ([HV26, Proposition 5.5])). PRG *can be computed in deterministic polynomial time, given oracle access to* $\mathbf{OL_2^P}$.

By using it we obtain an upper bound $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{OL_2^P}}$. Then by the oracle replacement corollary above, we continue to $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{ONP}}$ and $\mathbf{MA} \subseteq \mathbf{NOL_2^P}$, which gives two incomparable ways of improving the $\mathbf{MA} \subseteq \mathbf{NO_2^P}$ bound of [CR06].

Similar upper bounds for **OMA** follow from our more general derandomization results for $\mathbf{P}^{\mathbf{OprMA}}$. However, we can get a tighter derandomization. By following the same idea of replacing the minimum of the oblivious version of the Korten–Pitassi linear order by oblivious unambiguous computations with subsequent **co-ONP** verification (and thus using Korten's PRG in this framework), we prove a tighter upper bound for **co-OMA**. The reason is that another oracle query is to a promise problem in **co-ONP** for the derandomized version, and since both oracle queries are of the same polarity (**co-ONP**), we can aggregate them into a single query and subsequently replace this oracle by a special dot-operator. To this end we introduce one more dot-operator $\exists_{\mathbf{ou}\bullet}$ (along with its complementary dot-operator $\forall_{\mathbf{ou}\bullet}$): its witnesses are both oblivious and unambiguous (i.e. for all the positive inputs of the same length there exist a unique witness).

Theorem (Theorem 6.5). $\mathbf{co\text{-}OMA} \subseteq \exists_{\mathbf{ou}} \cdot \mathbf{pr\,co\text{-}ONP}$.

On the other hand, we fully characterize **OMA** by observing that switching to promise problems is enough to define it via the $\mathbf{BP}\bullet$ operator (in contrast to $\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP}$!).

Lemma (Lemma 6.7). $\mathbf{OMA} = \mathbf{BP} \cdot \mathbf{prONP}$.

Thus we can trade $\mathbf{BP}\bullet$ for $\exists_{\mathbf{ou}\bullet}$ in this case: $\mathbf{BP} \cdot \mathbf{pr\,co\text{-}ONP} \subseteq \exists_{\mathbf{ou}} \cdot \mathbf{pr\,co\text{-}ONP}$!

1.4.5 Derandomization of BPP-based classes

By definition, PRGs are designed to "fool" computations performed by small circuits (and, consequently, by polynomial-time Turing machines). However, their guarantees may no longer hold when we allow oracle queries or oracle gates in our circuits or computations. On the other hand, if the oracle itself can be implemented by small circuits, then the entire computation can be simulated by small, oracle-free circuits.

In [HV26], it was shown that a PRG can be computed in polynomial time with oracle access to $\mathbf{OL_2^P}$. The above observation allows us to extend this argument to **BPP**-based classes with oracles computable in $\mathbf{P/poly}$. Formally, we prove the following:

Lemma (Lemma 6.12). *Suppose* $A \in \mathbf{P/poly}$. *Then* $\mathbf{BPP}^A \subseteq \mathbf{P}^{A,\mathbf{OL_2^P}}$.

The proof idea is first to use the $\mathbf{OL_2^P}$ oracle access in order to construct a PRG and then use the PRG to derandomize $\mathbf{BPP}^A$. As a consequence, we obtain that $\mathbf{OL_2^P}$ is closed under randomized reductions: $\mathbf{BPP}^{\mathbf{OL_2^P}} = \mathbf{OL_2^P}$. It also follows that for such oracles, $\mathbf{BPP}^{A,\mathbf{OL_2^P}} = \mathbf{P}^{A,\mathbf{OL_2^P}}$, thus by using $A = \mathbf{ONP}$ we prove that the important intermediate class mentioned before is closed under randomized reductions: $\mathbf{BPP}^{\mathbf{ONP},\mathbf{OL_2^P}} = \mathbf{P}^{\mathbf{ONP},\mathbf{OL_2^P}}$.

2 Definitions and Basics

2.1 Notation agreement and a technical lemma about linear orders

We assume that all nondeterministic witnesses for the same input length are of the same length either, therefore even when a computation requires more than a single witness, we switch freely

between concatenations like xuv (i.e., $x \circ u \circ v$) and tuples like (x, u, v), because know the length of each element of the tuple. We assume that witness length is a monotone function of the input length growing at least as fast as $f(n) = n$, thus we never hit a length collision (if $|x_1| \neq |x_2|$, then $|(x_1, w_1)| \neq |(x_2, w_2)|$).

In what follows, we denote by $<_{lex}$ the lexicographic order on $\{0,1\}^*$. The following simple lemma will be useful in some of our proofs.

Lemma 2.1. *Let $\phi : X \to Y$ be an injective map such that $<_Y$ is a strict linear order on Y. Then the relation $a <_X b \iff \phi(a) <_Y \phi(b)$ forms a strict linear order on X.*

2.2 Input-oblivious nondeterministic and symmetric computations

We remind the definitions of **ONP** and **OMA** [FSW09] (the input-oblivious versions of **NP** and **MA**), as well as of $\mathbf{O_2^P}$ [CR06] (the input-oblivious version of $\mathbf{S_2^P}$). We define the promise versions of these classes; the classes of languages are defined as particular cases of the respective promise classes.

Definition 2.2 (**prONP**, **ONP**). *A promise problem $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ belongs to* **prONP** *if there is a polynomial p, a deterministic polynomial-time verifier V computing a binary relation in $\{0,1\}^n \times \{0,1\}^{p(n)}$, and a sequence of $p(n)$-bit strings $(y_n)_{n\in\mathbb{N}}$ such that*

- $\forall x \in \Pi_{\text{YES}} \cap \{0,1\}^n \; V(x, y_n) = 1$,
- $\forall x \in \Pi_{\text{NO}} \; \forall y \; V(x, y) = 0$.

The class of languages **ONP** *corresponds to languages in* **prONP** *(i.e., $\Pi_{\text{YES}} = \overline{\Pi_{\text{NO}}}$).*

Remark 2.3. *By definition,* **ONP** $\subseteq$ **NP**. *Moreover, since* **ONP** *is a "semantic" class, a* **prONP** *machine may behave arbitrarily outside the promise set. This is not the case for* **NP**, *since every* **NP** *machine decides some language. Consequently, any promise problem in* **prONP** *can be "completed" to an* **NP** *language. In particular, this means that an* **prONP** *oracle can be replaced with an* **NP** *oracle.*

A promise problem equivalent of **OMA** comes in two versions, a more general version **prOMA** and a restricted version **OprMA** [3]:

Definition 2.4 (**prOMA**, **OprMA**, **OMA**). *A promise problem $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ belongs to* **prOMA** *if there is a polynomial p, a deterministic polynomial-time Turing machine V computing a ternary relation in $\{0,1\}^n \times \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$, and a sequence of $p(n)$-bit strings $(y_n)_{n\in\mathbb{N}}$ such that*

- $\forall x \in \Pi_{\text{YES}} \cap \{0,1\}^n \; \forall r \in \{0,1\}^{p(n)} \; V(x, y_n, r) = 1$,
- $\forall x \in \Pi_{\text{NO}} \; \forall y \; \Pr_r\{V(x, y, r) = 1\} < \frac{1}{4}$.

The problem belongs to **OprMA** *if additionally y_n is oblivious even beyond the promise set separately for every random string, i.e.,*

- *for every $x \in \{0,1\}^n \setminus \Pi_{\text{NO}}$, every $y \in \{0,1\}^{p(n)}$, and every $r \in \{0,1\}^{p(n)}$: if $V(x, y, r) = 1$, then $V(x, y_n, r) = 1$.*

[3] The initial version of [HV26] defined a different version of this class; since it was removed from it in later revisions, we keep using the same abbreviation.

The class of languages **OMA** *corresponds to languages in* **prOMA** *(i.e.,* $\Pi_{\text{YES}} = \overline{\Pi_{\text{NO}}}$*).*

One can also define a two-sided version of **OMA***, which is equivalent to the one-sided version similarly to the non-oblivious case.*

Chakaravarthy and Roy defined $\mathbf{O}_2^{\mathbf{P}}$ [CR06]. We give this definition in the promise version:

Definition 2.5 ($\mathbf{prO}_2^{\mathbf{P}}$, $\mathbf{O}_2^{\mathbf{P}}$)**.** *A promise problem* $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ *belongs to* $\mathbf{prO}_2^{\mathbf{P}}$ *if there is a polynomial* p*, a polynomial-time deterministic Turing machine* V *computing a ternary predicate* $\{0,1\}^n \times \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$ *(we use the notation* $V_x(u,w)$ *to denote its result), and two sequences of* $p(n)$*-bit strings* $(y_n)_{n\in\mathbb{N}}$*,* $(z_n)_{n\in\mathbb{N}}$ *such that*

- $\forall x \in \Pi_{\text{YES}} \cap \{0,1\}^n \ \forall z \ V_x(y_n, z) = 1$,
- $\forall x \in \Pi_{\text{NO}} \cap \{0,1\}^n \ \forall y \ V_x(y, z_n) = 0$.

These strings y_n *and* z_n *are called input-oblivious* irrefutable certificates *of the yes- and no-answer, respectively.*

The class of languages $\mathbf{O}_2^{\mathbf{P}}$ *corresponds to languages in* $\mathbf{prO}_2^{\mathbf{P}}$ *(i.e.,* $\Pi_{\text{YES}} = \overline{\Pi_{\text{NO}}}$*).*

2.3 Input-oblivious unambiguous computations

We define the input-oblivious version of **UP** (Unambiguous Polynomial-Time), the class introduced by Valiant [Val76]. Recall that **UP** is a subclass of **NP** in which every "yes" input x has a *unique* accepting witness. However, this witness may depend on the input. We define an oblivious variant by requiring that, for every input length, the unique accepting witness is the same for all "yes" inputs of a given length.

Definition 2.6 (**OUP**)**.** *A language* L *belongs to* **OUP** *if there is a polynomial* p*, polynomial-time deterministic Turing machine* M *computing a binary relation in* $\{0,1\}^n \times \{0,1\}^{p(n)}$*, and a sequence of* $p(n)$*-bit strings* $(w_n)_{n\in\mathbb{N}}$ *(oblivious unambiguous witnesses) such that:*

- *(uniqueness) for every* $x \in \{0,1\}^n$*, for every* $u \neq w_n$*,* $M(x,u) = 0$.
- $\forall x \in L \cap \{0,1\}^n \quad M(x, w_n) = 1$,
- $\forall x \in \overline{L} \cap \{0,1\}^n \quad \forall u \in \{0,1\}^{p(n)} \quad M(x,u) = 0$.

Note that if $L \cap \{0,1\}^n = \emptyset$, then *any* w_n would do the job, in particular $w_n = 0^{p(n)}$.

We also introduce a symmetric version of the unambiguous oblivious computations. It is similar to $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$, but the two unique witnesses must be equal.

Definition 2.7 (**SOUP**)**.** *A language* L *belongs to* **SOUP** *if there is a polynomial* p*, polynomial-time deterministic Turing machines* Y *and* Z *computing binary relations in* $\{0,1\}^n \times \{0,1\}^{p(n)}$*, and a sequence of* $p(n)$*-bit strings* $(w_n)_{n\in\mathbb{N}}$ *such that:*

- *(uniqueness) for every* $x \in \{0,1\}^n$*, for every* $u \neq w_n$*,* $Y(x,u) = 0$ *and* $Z(x,u) = 0$.
- $\forall x \in L \cap \{0,1\}^n \quad Y(x, w_n) = 1$,
- $\forall x \in \overline{L} \cap \{0,1\}^n \quad \forall u \in \{0,1\}^{p(n)} \quad Y(x,u) = 0$,
- $\forall x \in \overline{L} \cap \{0,1\}^n \quad Z(x, w_n) = 1$,
- $\forall x \in L \cap \{0,1\}^n \quad \forall u \in \{0,1\}^{p(n)} \quad Z(x,u) = 0$.

Obviously, $\mathbf{SOUP} \subseteq \mathbf{OUP} \cap \mathbf{co\text{-}OUP}$.

2.4 Dot operators

The first dot-operator was introduced by Wagner [Wag86], and then more of them were defined by by Zachos and Fürer [ZF87, Zac88] and then by Schöning [Sch89] and Böhler, Glaßer, and Meister [BGM06]. They have been used extensively in the literature afterwards, sometimes applied to promise problems as well; Borchert and Silvestri [BS01] wrote on a general notion of such and operator. To avoid ambiguity, in this section we give formal definitions in this context (dot-operators for languages are a particular case of these definitions).

Definition 2.8 (dot operators)**.**

1. *We define the dot-operator* $\exists\bullet$ *for promise classes as follows:* $L \in \exists \cdot \mathbf{pr}\,\mathcal{C}$ *if there is a promise problem* $(\Pi_{\text{YES}}, \Pi_{\text{NO}}) \in \mathcal{C}$ *and a polynomial* p *such that for every* $x \in \{0,1\}^*$*,*

$$\begin{aligned} x \in L &\Rightarrow \exists w \in \{0,1\}^{p(|x|)}\ (x,w) \in \Pi_{\text{YES}}, \\ x \notin L &\Rightarrow \forall w \in \{0,1\}^{p(|x|)}\ (x,w) \in \Pi_{\text{NO}}. \end{aligned}$$

   We remark that "queries" that are made outside the promise set (that is, by the quantified $\exists w$ *when* w *is unsuccessful) do not influence the answer, thus it intuitively corresponds to "loose" oracle access (like, for example, the model used in [CR11]). We will also use the notation* $L = \exists \cdot \Pi$*.*

   The dot-operator $\forall\bullet$ *for promise classes is defined analogously. These generalize the corresponding notations for languages.*

2. *The dot-operators* $\exists_{\mathbf{o}}\bullet$*,* $\exists_{\mathbf{u}}\bullet$*, and* $\exists_{\mathbf{ou}}\bullet$ *are defined similarly, when the corresponding witness* w *must be input-oblivious, unique, or unique and input-oblivious, respectively.*

   For instance, in the latter case there is a single witness for each and every positive input of the same length, and this is the same witness for all of them.)

3. *The four opposite dot-operators* $\forall\bullet$*,* $\forall_{\mathbf{o}}\bullet$*,* $\forall_{\mathbf{u}}\bullet$*, and* $\forall_{\mathbf{ou}}\bullet$ *are defined by symmetry:*

$$\begin{aligned} x \in L &\Rightarrow \forall w \in \{0,1\}^{p(|x|)}\ (x,w) \in \Pi_{\text{YES}}, \\ x \notin L &\Rightarrow \exists w \in \{0,1\}^{p(|x|)}\ (x,w) \in \Pi_{\text{NO}}. \end{aligned}$$

   Witnesses (that need to be input-oblivious or unique) are now for the negative instances.

   In particular, $\exists \cdot (\Pi_{\text{YES}}, \Pi_{\text{NO}}) = \overline{\forall(\Pi_{\text{NO}}, \Pi_{\text{YES}})}$ *(thus,* $\mathbf{co}\text{-}\exists \cdot \mathcal{C} = \forall \cdot \mathbf{co}\text{-}\mathcal{C}$*, where* $\mathbf{co}\text{-}$ *for promise problems exchanges* Π_{YES} *and* Π_{NO} *sets) and the same holds for the other three pairs.*

4. *Let* $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$ *be a promise problem. We say that* $L \in \mathbf{BP} \cdot \Pi$*, if there exists a polynomial* $p(n)$ *such that:*

   - $x \in L \implies \Pr_{r \in \{0,1\}^{p(|x|)}}[(x,r) \in \Pi_{\text{YES}}] \geq 3/4.$
   - $x \notin L \implies \Pr_{r \in \{0,1\}^{p(|x|)}}[(x,r) \in \Pi_{\text{NO}}] \geq 3/4.$

Dot-operators for languages are particular cases of dot-operators for promise classes. Therefore, in particular, $\mathbf{ONP} = \exists_{\mathbf{o}} \cdot \mathbf{P}$ and $\mathbf{OUP} = \exists_{\mathbf{ou}} \cdot \mathbf{P}$. Additionally, we define a special restricted version of second-order co-cap classes:

Definition 2.9 (**SOUON**, **SOUprON**)**.** *Define* **SOUON** *as a restricted version of* $\exists_{\mathbf{ou}} \cdot \mathbf{co\text{-}ONP} \cap \forall_{\mathbf{ou}} \cdot \mathbf{ONP}$ *where the two unique oblivious witnesses (for "yes" and for "no") coincide. Define similarly* **SOUprON** *based on* $\exists_{\mathbf{ou}} \cdot \mathbf{pr\,co\text{-}ONP} \cap \forall_{\mathbf{ou}} \cdot \mathbf{prONP}$*.*

2.5 Computations based on the Linear Ordering Principle

2.5.1 Known classes (basic and input-oblivious)

We begin by recalling the definitions of the regular and the input-oblivious versions of $\mathbf{L}_2^{\mathbf{P}}$.

Definition 2.10 ($\mathbf{L}_2^{\mathbf{P}}$ [KP24]). *A language L belongs to $\mathbf{L}_2^{\mathbf{P}}$ if there is a polynomial p, polynomial-time deterministic Turing machine V computing a ternary predicate $\{0,1\}^n \times \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$ (we use the notation $V_x(u,w)$ to denote its result) such that:*

- *for every x, V_x is a strict linear order (define $u <_x w$ iff $V_x(u,w) = 1$),*
- $\forall x \in L \cap \{0,1\}^n \quad \min <_x$ *starts with* 1,
- $\forall x \in \overline{L} \cap \{0,1\}^n \quad \min <_x$ *starts with* 0.

The following lemma generalizes $\mathbf{NP} \subseteq \mathbf{L}_2^{\mathbf{P}}$ [KP24] and states specific features of the constructed linear ordering that will be used later:

Lemma 2.11. *Let $L \in \mathbf{NP}$ and let $M(x,w)$ be the corresponding polynomial-time verifier with $|w| = p(|x|)$ for some polynomial p. Consider the following relation on strings bw when $b \in \{0,1\}$ and $w \in \{0,1\}^{p(n)}$:*

$$bw <_x b'w' \iff \big(b \oplus M(x,w), \bar{b}, w\big) <_{\ell ex} \big(b' \oplus M(x,w'), \bar{b}', w'\big).$$

Then for every x, the relation $<_x$ forms a strict linear order. Furthermore,

- *If $x \in L$ then $\min <_x = 1w_{x\text{-}min}$, when $w_{x\text{-}min} = \min_{<_{\ell ex}} \Big\{ w \in \{0,1\}^{p(|x|)} \mid M(x,w) = 1 \Big\}$.*
- *If $x \in \overline{L}$ then $\min <_x = 0^{p(|x|)+1}$.*

Proof. Since the map $\phi(b,w) = \big(b \oplus M(x,w), \bar{b}, w\big)$ is injective, by Lemma 2.1, the relation $<_x$ forms a strict linear order. We now establish the claim about the respective minima.

- Let $x \in L$. Assume for a contradiction that $b'w' <_x 1w_{x\text{-min}}$. Since $\phi(1, w_{x\text{-min}}) = (0, 0, w_{x\text{-min}})$, it must be the case that $b' \oplus M(x,w') = 0, \bar{b}' = 0$ and $w' <_{\ell ex} w_{x\text{-min}}$. However, this in turn implies that $M(x,w') = 1$, contradicting the definition of $w_{x\text{-min}}$.
- Let $x \in \overline{L}$. As before, assume for a contradiction that $b'w' <_x 0^{p(n)+1}$. Since $\phi(0, 0^{p(n)}) = (0, 1, 0^{p(n)})$, it must be the case that $b' \oplus M(x,w') = 0$ and $\bar{b}' = 0$. Consequently, $M(x,w') = 1$. However, recall that by definition in this case $\forall w \; M(x,w) = 0$. □

Definition 2.12 ($\mathbf{OL}_2^{\mathbf{P}}$ [HV26]). *A language L belongs to $\mathbf{OL}_2^{\mathbf{P}}$ if there is a polynomial p, polynomial-time deterministic Turing machine V computing a ternary predicate $\{0,1\}^n \times \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$ (we use the notation $V_x(u,w)$ to denote its result), and two sequences of $p(n)$-bit strings $(y_n)_{n\in\mathbb{N}}$, $(z_n)_{n\in\mathbb{N}}$ such that:*

- *for every x, V_x is a strict linear order (define $u <_x w$ iff $V_x(u,w) = 1$),*
- $\forall x \in L \cap \{0,1\}^n \quad \min <_x = 1y_n$,
- $\forall x \in \overline{L} \cap \{0,1\}^n \quad \min <_x = 0z_n$.

Despite $\mathbf{NP} \subseteq \mathbf{L}_2^{\mathbf{P}}$ (and even $\mathbf{P}^{\mathbf{NP}} \subseteq \mathbf{L}_2^{\mathbf{P}}$) [KP24], it is unclear whether $\mathbf{ONP} \subseteq \mathbf{OL}_2^{\mathbf{P}}$. However, we can at least prove that $\mathbf{OUP} \subseteq \mathbf{OL}_2^{\mathbf{P}}$.

Theorem 2.13. $\mathbf{OUP} \subseteq \mathbf{OL}_2^{\mathbf{P}}$.

Proof. Consider a polynomial-time verifier $M(x, w)$ with input-oblivious and unique witnesses $(w_n)_{n\in\mathbb{N}}$. Consider the linear order $<_x$ given by Lemma 2.11.

We now show that the minima of this order are input-oblivious. For $x \in \overline{L} \cap \{0,1\}^n$ this is trivial as the minimum is $0^{p(n)+1}$. For $x \in L \cap \{0,1\}^n$, we claim that $\min <_x = 1w_n$. Assume for a contradiction that $b'w' <_x 1w_n$. Since $\phi(1, w_n) = (0, 0, w_n)$, it must be the case that $b' \oplus M(x, w') = 0, \overline{b}' = 0$ and $w' <_{\ell ex} w_n$. However, this in turn implies that $M(x, w') = 1$. Yet, by uniqueness $w' = w_n$. □

2.5.2 "Half-oblivious" linear ordering

Chakaravarthy and Roy [CR06] defined a "half-oblivious" class $\mathbf{NO}_2^{\mathbf{P}}$ where witnesses are input-oblivious on one side only (thus this class contains **NP**). We do the same for a similar class based on the Linear Order Principle.

Definition 2.14 ($\mathbf{NO}_2^{\mathbf{P}}$ [CR06])**.** *A language L belongs to $\mathbf{NO}_2^{\mathbf{P}}$ if there is a polynomial p, a polynomial-time deterministic Turing machine V computing a ternary predicate $\{0,1\}^n \times \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$ (we use the notation $V_x(u, w)$ to denote its result), and a sequence of $p(n)$-bit strings $(z_n)_{n\in\mathbb{N}}$ such that*

- $\forall x \in L\ \exists y\ \forall z\ V_x(y, z) = 1$,
- $\forall x \in \overline{L} \cap \{0,1\}^n\ \forall y\ V_x(y, z_n) = 0$.

Definition 2.15 ($\mathbf{NOL}_2^{\mathbf{P}}$)**.** *A language L belongs to $\mathbf{NOL}_2^{\mathbf{P}}$ if there is a polynomial p, a polynomial-time deterministic Turing machine V computing a ternary predicate $\{0,1\}^n \times \{0,1\}^{p(n)} \times \{0,1\}^{p(n)}$ (we use the notation $V_x(u, w)$ to denote its result), and a sequence of $p(n)$-bit strings $(z_n)_{n\in\mathbb{N}}$ such that:*

- *for every x, V_x is a strict linear order (define $u <_x w$ iff $V_x(u, w) = 1$),*
- $\forall x \in L \quad \min <_x$ *starts with* 1,
- $\forall x \in \overline{L} \cap \{0,1\}^n \quad \min <_x = 0z_n$.

Even though the following proposition is strengthened in what follows, it is very natural and it is good to note it now for its simplicity:

Proposition 2.16. $\mathbf{NP} \subseteq \mathbf{NOL}_2^{\mathbf{P}}$.

Proof. Observe that in Lemma 2.11 the "no"-minimum of $<_x$ is 0^{n+1} and thus input-oblivious. □

2.5.3 A new input-oblivious linear order class

We define a class of languages such that acceptance and rejection can be determined in polynomial time from the minimum of a polynomial-time linear order, in which the linear order is *oblivious*: it is a function only of the input length.

Definition 2.17. *We say that $L \in$ **OMIN** if there is a polynomial p, a deterministic polynomial-time Turing machine D, and a deterministic Turing machine $V(1^n, y, z)$ working in time polynomial in n that, for every $n \in \mathbb{N}$, computes a linear order $<_n$ on $p(n)$-bit strings such that, for every $x \in \{0,1\}^n$, we have*

$$x \in L \iff D\big(x, min_{<_n}\big) = 1$$

where $min_{<_n}$ is the unique minimum of the order. That is, every input of length n can be determined in polynomial time once we have access to the minimum element of this oblivious linear ordering.

3 $\mathbf{OL}_2^{\mathrm{P}} = \mathbf{OMIN}$ and contains hard functions

In this section, we show that $\mathbf{OL}_2^{\mathrm{P}}$ has a simplified "normal form", as the class **OMIN**. We will need the following theorem of Korten and Pitassi.

Lemma 3.1 ([KP24, Theorem 1])**.** *There is a polynomial $p(n)$ and a polynomial-time algorithm $V(C, u, v)$ that defines a linear ordering $u <_C v \iff V(C, u, v) = 1$, where $u, v \in \{0,1\}^{p(|C|)}$ and C is a Boolean circuit that outputs twice as many bits as its input bits, and there is a polynomial-time algorithm $D(1^n, C, m)$ that outputs $y \notin \mathrm{Im}(C)$, given the minimum element m of the ordering $<_C$.*

We will also need the following simple "truth table generator" [Kor21].

Proposition 3.2. *For every positive k, there is a polynomial time algorithm E^k which expects an input C of length $N = \lfloor n^{2k}/2 \rfloor$ (for some parameter n), interprets C as the encoding of a circuit of size kn^k with $2k \log n$ input bits, and outputs the n^{2k}-bit truth table of C: the outputs of C over all possible inputs.*

This algorithm can be seen as an instance of Range Avoidance (as defined in the introduction), as its output lengths always exceed its input lengths.

Theorem 3.3. *For every $k \geq 1$, there is an $L_k \in$ **OMIN** that does not have circuits of size kn^k.*

In fact, the hard language only ever accepts on the lexicographically first n^{2k} inputs of length n (it is sparse) and the lower bound holds almost everywhere.

Proof. Fix a constant k. Let E_n^k be the *truth table generator* (from Proposition 3.2) that takes in an input of length $n^{2k}/2$, construes it as a circuit of size at most kn^k with $2k \log n$ input bits, then outputs a n^{2k}-bit truth table (the values of the circuit on the lexicographically first n^{2k} inputs). As in prior works (e.g. [Kor21]), we note that since the length of the output of E^k is greater than the length of its input, we have that for all n there is a string of length n^{2k} that is not in the range of E^k. Our task is to construct such a string, as it will not have kn^k-size circuits.

We start with the linear ordering of Korten and Pitassi [KP24] given by Lemma 3.1; since E_n^k can be generated in time polynomial in n this lemma gives a polynomial-time algorithm $D(1^n, m_n)$ that outputs $y_n = D(1^n, m_n)$ not in $\mathrm{Im}(E_n^k)$, where m_n is the minimum element of their ordering. By construction, y_n is the truth table of a function that does not have kn^k-size circuits. For simplicity let $|y_n| = q(n)$ and $|m_n| = p(n)$ in the following. We set up an input-oblivious polynomial-time computable linear ordering $<_n$ on $q(n) + p(n)$ length strings with minimum element $y_n m_n$.

In particular, we define the ordering as follows: given yz and $y'z'$, we say $yz <_n y'z'$ iff either

- $z < z'$ according to the Korten–Pitassi order, or
- $z = z'$ and $y = D(1^n, z)$ and $y' \neq D(1^n, z)$, or
- $z = z'$, $y \neq D(1^n, z)$, $y' \neq D(1^n, z)$, and y is less than y' in the lexicographic order.

We observe that the minimum element of this linear ordering is precisely $y_n m_n$. Since y_n is the truth table of a hard function, our polynomial-time decoder $D(x, y_n)$ can just index into the bits of y_n (i.e., $D(x, y_n)$ outputs the x-th bit of y_n if x is sufficiently small, else $D(x, y_n)$ outputs 0). We have therefore constructed a language $L \in$ **OMIN** that does not have kn^k-size circuits. □

Our next result shows that the newly defined class **OMIN** coincides with the class based on the oblivious linear ordering principle introduced in [HV26]. In particular, this equivalence shows that requiring the same oblivious minima is equivalent to requiring the same oblivious linear order.

Theorem 3.4. $\mathbf{OMIN} = \mathbf{OL_2^P}$.

Proof. First we show $\mathbf{OMIN} \subseteq \mathbf{OL_2^P}$. Let $L \in \mathbf{OMIN}$, and let D be its decoder. For $x \in \{0,1\}^n$, we define a linear ordering $<_x$ as follows on strings of length $p(n)+1$. For bits a, a' and bit strings u, u' of length $p(n)$, we define $au <_x a'u'$ if and only if either $(u <_n u')$ or $(u = u'$ and $a = D(x,u)$ and $a' \neq D(x,u))$. We observe that the minimum of $<_x$ is $L(x)\min_{<_n}$. This construction works on both yes and no inputs; therefore $L \in \mathbf{OL_2^P}$.

Next we show the harder direction, $\mathbf{OL_2^P} \subseteq \mathbf{OMIN}$. Let $L \in \mathbf{OL_2^P}$. On every input x of length n, there is a polynomial-time computable linear ordering $<_x$ on $(t(n)+1)$-bit strings, with two possible minima: $1y_n$ and $0z_n$ for yes- and no-instances of length n, respectively, where y_n and z_n are $t(n)$-bit strings.

Fix an input length n. We will design a single oblivious linear ordering that will include y_n and z_n in the minimum, thus we will be able to decide $L(x)$ on all strings of length n.

First, let $R_n = b_n r_n$ be the minimum with respect to the input 0^n. By definition of $\mathbf{OL_2^P}$, either $R_n = 1y_n$ or $R_n = 0z_n$. Also by definition, $L(0^n) = b_n$.

Let $S_n = (1-b_n)s_n$ be the *other* possible minimum, namely the string distinct from R_n. Observe that

$$L(x) \neq b_n \iff S_n <_x R_n.$$

Consequently,

$$L(x) = b_n \oplus \big[S_n <_x R_n\big].$$

Thus, if we could set up a linear ordering in which we have access to both R_n and S_n in the minimum element of that ordering, we could decide $L(x)$ with some decoder D.

Our goal is to construct an oblivious order such that:

1. If some input of length n has an answer *different* from $L(0^n) = b_n$, then the minimum element contains both R_n and S_n, and
2. If all inputs of length n have the same answer, then the minimum contains $R_n = L(0^n)r_n$. In this case, the answer $L(0^n)$ already suffices.

This will allow us to decode the answer $L(x)$ for all x, from the minimum element.

The new ordering is over triples (r', y, s'), where y is an n-bit string and r', s' are $(t(n)+1)$-bit strings. To compare two triples, we first compare their r' strings with respect to $<_{0^n}$, and prefer the triple with smaller r' wrt this ordering. Thus, the minimum triple must have $r' = R_n$. Among those triples with the same component r', the ordering then prefers triples such that $s' <_y r'$ (over triples for which this property does not hold). If a pair of triples either both satisfy $s' <_y r'$ or both fail to satisfy it, then the ordering prefers the triple with the smaller y in lex order. Finally, for those remaining pairs of triples whose ordering has not yet been determined, the ordering prefers the triple with the smaller s' under $<_y$ (they have the same y).

If $L(x) = L(0^n)$ for every x of length n, then we claim that the minimum of the ordering is $(R_n, 0^n, R_n)$. Indeed, in this case, the string R_n is the minimum element for $<_y$ over all y. Hence there is no y such that $s' <_y r'$, so the ordering prefers $y = 0^n$ and then $s' = R_n$.

If $L(x) \neq L(0^n)$ for some x of length n, then the minimum is (R_n, y, S_n) where y is the lexicographically minimum string such that $L(y) \neq L(0^n)$. Indeed, in this case there is a y and

s' such that $s' <_y R_n$; our ordering selects the lexicographically first such y, and the minimum s' under $<_y$ which is S_n.

In either case, the correct value of $L(x)$ can be easily decoded from the minimum element. We conclude that $\mathbf{OL_2^P} \subseteq \mathbf{OMIN}$. □

Corollary 3.5. $\exists \cdot \mathbf{OL_2^P} \subseteq \mathbf{NOL_2^P}$.

Proof. By Theorem 3.4 we can use the definition of **OMIN** for the $\mathbf{OL_2^P}$ language L, let us stay in the notation of Definition 2.17.

Then our language $K \in \exists \cdot \mathbf{OL_2^P}$ can be defined as $K = \{(x, w) \mid (x, w) \in L\}$, where $|w| = p(|x|)$ for some polynomial p (w.l.o.g., the same polynomial as in Definition 2.17).

Similarly to the first part of the theorem proof, for $x \in \{0,1\}^n$, we define a linear ordering $<_x$ as follows. Let $m = |(x, w)|$. For bits b, b', bit strings w, w' of length $p(n)$, and bit strings u, u' of length $p(m)$, we define $buw \prec_x b'u'w'$ as

$$buw \prec_x b'u'w' \iff \big(b \oplus D((x,w),u), u, \bar{b}, w\big) <_{lex+m} \big(b' \oplus D((x,w'),u'), u', \bar{b}', w'\big),$$

where $<_{lex+m}$ compares tuples lexicographically, all their components are also compared lexicographically except for the comparison $u <_m u'$. Since the map $\phi(buw) = \big(b \oplus D((x,w),u), u, \bar{b}, w\big)$ is injective and $<_{lex+m}$ is a linear order, by Lemma 2.1 $\prec_x$ is a linear order.

Denote $u_* = \min_{<_m}$.

If $x \in K$, then the minimum of $\prec_x$ is $1u_*w_*$, where w_* is the lexicographically smallest witness w such that $(x, w) \in L$. (Indeed, all the components of $\phi(1u_*w_*)$ are the minimums of the respective orders.)

If $x \notin K$, then the minimum of $\prec_x$ is $0u_*0^{p(n)}$ (and thus it is input-oblivious). Indeed, the only string buw that could have a chance to beat it must use $u = u_*$, but then $D((x, w), u) = 0$ since $(x, w) \notin L$, and the xor component dictates $b = 0$, thus only the last component w could be different, but it's already all-zero in our case. □

4 Query aggregation

In this section we consider replacing multiple oracle queries by a single query, which either collapses the class or leads to a definition using a certain version of the dot-operator.

In Sections 4.1 and 4.2 we prove that $\mathbf{P^{OL_2^P}} = \mathbf{OL_2^P}$ and $\mathbf{P^{SOUP}} = \mathbf{SOUP}$.

In the remaining Sections 4.3, 4.4 we deal with **ONP** and **prONP** queries, the resulting diagram is given in Fig. 2 (implied containments not shown, as well as trivial containments replacing **prONP** by **NP**).

4.1 $\mathbf{OL_2^P}$ is polynomial-time Turing-closed

Korten and Pitassi [KP24] prove that $\mathbf{P^{L_2^P}} = \mathbf{L_2^P}$. It turns out that similarly to $\mathbf{L_2^P}$ queries, oblivious ($\mathbf{OL_2^P}$) queries can be aggregated and, moreover, $\mathbf{OL_2^P}$ is closed under polynomial-time Turing reductions. We use the equivalent definition (**OMIN**) due to Theorem 3.4.

Theorem 4.1 (**OMIN** query aggregation)**.** $\mathbf{P^{OMIN}} = \mathbf{OMIN}$.

Proof. Let $A \in \mathbf{P^{OMIN}}$ be defined by a deterministic kn^k-time oracle Turing machine M^L with an oracle $L \in \mathbf{OMIN}$. We assume the notation like in Definition 2.17, and the verifier V_L for the linear ordering $<_n$ is also kn^k-time.

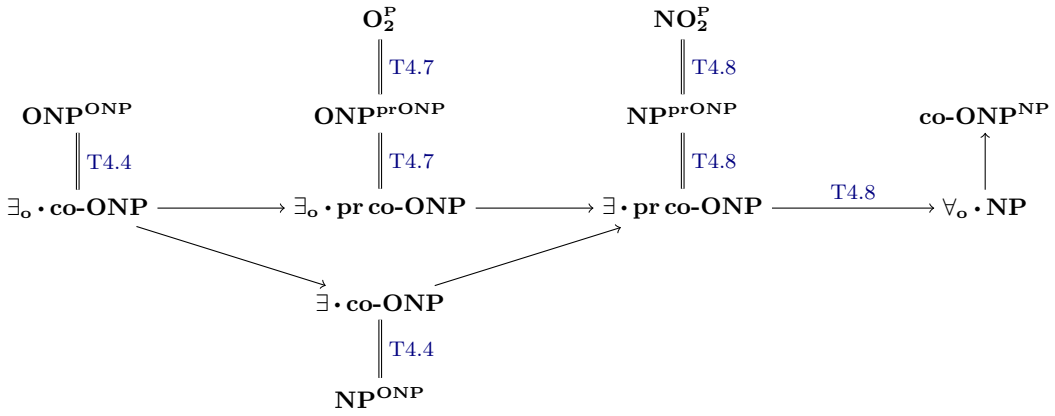


Figure 2: Diagram of equivalences and containments for classes with **ONP** and **prONP** oracles

Our new **OMIN** verifier will use witnesses in the format $w = w_1 \circ \ldots \circ w_{kn^k}$, where w_ℓ is a $k\ell^k$-length candidate witness (the intention is that w_ℓ is the V_L-minimum for length-ℓ queries).

Our new linear ordering $w \prec_n w'$ will compare each witness w_i against w'_i w.r.t. $<_i$, starting from $i = 1$. As soon as it finds $w_i \neq w'_i$, it returns the result of comparison. Thus $\prec_n$ is a linear order (it is a lexicographic order over the answers of $<_i$, it is oblivious, and its minimum contains exactly the minimum elements w.r.t. the old order.

To decode the answer from w, we simulate $M^\bullet$ without using the oracle L. When this machine needs to query $q \in L$, it uses $D(q, w_{|q|})$ instead of the oracle's answer. □

Corollary 4.2. $\mathbf{NP^{OL_2^P}} = \exists \cdot \mathbf{OL_2^P}$. *Moreover, the same is true for* **ONP**, **UP**, **OUP** *and the respective dot-operators.*

Proof. Observe that for any oracle O, $\mathbf{NP}^O = \exists \cdot \mathbf{P}^O$ (and similarly for **ONP**, **UP**, **OUP**). However, $\mathbf{P^{OL_2^P}} = \mathbf{OL_2^P}$ by Theorem 4.1. □

4.2 SOUP is polynomial-time Turing-closed

It turns out that symmetric input-oblivious unambiguous computations are closed under polynomial-time Turing reductions. This statement is relativizable (it will be useful later when these classes will naturally as oracles).

Theorem 4.3. *For any oracle A,* $\mathbf{P^{SOUP}}^A = \mathbf{SOUP}^A$.

Proof. Let us guess all the unique oblivious witnesses w_i for the oracle for all relevant input lengths i. We can test them on the input 0^i: w_i must work either for the positive or for the negative part of the $\mathbf{SOUP}^A$ language definition. If any of the checks fail, we reject (both in the positive and in the negative part of our new computation). Then we can simulate our oracle machine replacing the oracle using these witnesses and get the same result as the original oracle machine. □

4.3 Aggregating ONP queries using the dot-operator

It is known that the two classical definitions of the polynomial hierarchy (using oracles and using quantifiers) are equivalent, for example, $\mathbf{NP^{NP}} = \exists \cdot \mathbf{co\text{-}NP}$. It matches the fact that respective versions of the language of true quantified Boolean formulas with the corresponding

quantifiers is complete for these classes. It turns out that even though **ONP** is not known to have complete problems, with some effort one can aggregate queries to a specific **ONP** oracle into a single **ONP** query and then reformulate the use of an oracle using the corresponding dot-operator.

We first prove it for **ONP** queries made by **NP**- and **ONP**-type computations and then continue to **prONP** queries. In particular, it turns out that **ONP**-type computations with a **prONP** oracle capture exactly the $\mathbf{O}_2^{\mathbf{P}}$ class, which gives it also a characterization in terms of the oblivious existential dot-operator: $\mathbf{O}_2^{\mathbf{P}} = \exists_{\mathbf{o}} \cdot \mathbf{pr\,co\text{-}ONP}$. We then prove (in Section 4.4) that $\mathbf{NO}_2^{\mathbf{P}}$ defined by [CR06] has a similar characterization in terms of the non-oblivious existential dot-operator: $\mathbf{NO}_2^{\mathbf{P}} = \exists \cdot \mathbf{pr\,co\text{-}ONP}$. It leads to a surprising phenomenon: the non-oblivious existential and the oblivious universal dot-operators can be swapped, that is, $\exists \cdot \mathbf{pr\,co\text{-}ONP} \subseteq \forall_{\mathbf{o}} \cdot \mathbf{NP}$.

Theorem 4.4 (**ONP** query aggregation)**.**

1. $\mathbf{ONP}^{\mathbf{ONP}} = \exists_{\mathbf{o}} \cdot \mathbf{co\text{-}ONP}$.
2. $\mathbf{NP}^{\mathbf{ONP}} = \exists \cdot \mathbf{co\text{-}ONP}$.

Proof. 1. We prove the first claim; the second claim is similar, the difference is that in the second claim the "external" witnesses are not input-oblivious.

Let $L \in \mathbf{ONP}^{\mathbf{ONP}}$ be accepted by an input-oblivious polynomial-time oracle machine $M^{\bullet}$ with an input-oblivious oracle O (defined by a polynomial-time deterministic machine $M_O(q,u)$ so that $O = \{q \mid \exists u\ M_O(q,u) = 1\}$). Suppose that both machines make at most n^k queries, given an input of length n, and each query is of size at most n^k. Denote by z_m and o_m the corresponding oblivious witnesses of M and M_O at length m, respectively. Assume w.l.o.g. that $|z_m| = |o_m| = m^k$ and $\varepsilon \notin L$.

Consider an execution of $M^{\bullet}$ on an input of size n. Let a witness w be parsed as follows: $w = (w[0], w[1], \ldots, w[n^k])$. In particular, we define a new witness w_n as $w_n = (z_n, o_1, \ldots, o_{n^k})$. Here, for each n, the length $\ell(n)$ of w is defined so that $|(x, w_{|x|})| \neq |(y, w_{|y|})|$ if $|x| \neq |y|$.

Given that, denote by $M^*(x,w)$ be the result of the execution of $M^{\bullet}$ on x with $w[0]$ used as the witness (i.e. the original witness of $M^{\bullet}$) and where each oracle query q is answered as $M_O(q, w[|q|])$ (instead of calling the oracle). Define W to be the language consisting (for all $n \in \mathbb{N}$) of all pairs $(x,w) \in \{0,1\}^n \times \{0,1\}^{\ell(n)}$ satisfying *either* $M^*(x,w) = 0$ *or* $\exists w' = (w'[1], w'[2], \ldots, w'[n^k])$ s.t.

$$\bigvee_{1 \leq j \leq n^k} M_O(q_j, w[|q_j|]) = 0 \wedge M_O(q_j, w'[|q_j|]) = 1, \tag{4.1}$$

where $q_1, \ldots, q_{n^k}$ is the list of queries made by $M^*(x,w)$. (The latter condition means that some $w[i]$ in w is "bad": by using it we miss a correct witness $w'[i]$ for one of the queries.)

We first prove that $W \in \mathbf{ONP}$ and then that $L = \{x \mid \exists w\ (x,w) \in \overline{W}\}$ with the oblivious witness w_n, thus completing the proof of our claim.

Proof that $W \in \mathbf{ONP}$. Indeed, let $w'_n[i] = o_i$ for every $1 \leq i \leq n^k$. We prove that it is an oblivious witness for W. Let $(x,w) \in W$. We have two cases to consider:

- $M^*(x,w) = 0$. In this case w'_n (as well as *any* witness) leads to acceptance.
- $\exists w'$ satisfying (4.1). Let j be the smallest index s.t. $M_O(q_j, w[|q_j|]) = 0 \wedge M_O(q_j, w'[|q_j|]) = 1$. We obtain that $q_j \in O$ and hence $M_O(q_j, w'_n[|q_j|]) = M_O(q_j, o_{|q_j|}) = 1$. Consequently, w'_n leads to acceptance as well.

Proof that $L = \{x \mid \exists w\ (x,w) \in \overline{W}\}$ *with the oblivious witness* w_n. Note that an error in the execution of $M^*(x,w)$ might occur only when the machine makes a query q such that $M_O(q, w[|q|]) = 0$, yet $q \in O$. The purpose of W is to ensure that this kind of an error never occurs. In particular, if $(x,w) \notin W$, then $M^*(x,w) = 1$ and all queries are answered correctly ((4.1) is false for every w' including w'_n). If $(x,w) \in W$, then either $M^*(x,w) = 0$ or there is an error in answering the queries.

Therefore, more formally:

- If $x \in L$, then by definition $M^O(x) = 1$ and $M^*(x, w_n) = 1$. At the same time, Condition (4.1) is false since $w_n[i] = w'_n[i]$ for all $1 \le i \le n^k$. As a result, $(x, w_n) \notin W$.
- If $x \notin L$, then no w would do: if the simulation computation accepts while M^O would not accept, it means that at least one query was answered as 0 instead of 1, but then the second condition will be satisfied for w'_n (because of this query), thus $(x,w) \in W$. If the simulated computation rejects, then also $(x,w) \in W$.

2. For $\mathbf{NP}^{\mathbf{ONP}}$, the same proof works: note that the oblivious witness w'_n in (4.1) consists of oblivious witnesses for every possible query length. The only place where the existence of oblivious z_n is used is the obliviousness of w_n; we do not need it for our claim in this part of the theorem. □

Remark 4.5. *Note that we use here the fact that the basic machine is powerful enough to simulate the oracle machine. For example, for computations that cannot simulate* **NP** *themselves, even aggregating* **NP** *queries may not be possible: if* $\mathbf{P}^{\mathbf{NP}[1]} = \mathbf{P}^{\mathbf{NP}[2]}$ *or* $\mathbf{O}_2^{\mathbf{P}^{\mathbf{NP}[1]}} = \mathbf{O}_2^{\mathbf{P}^{\mathbf{NP}[2]}}$, *then* **PH** *collapses [BF99b, CR06].*

4.4 Aggregating prONP queries, $\mathbf{O}_2^{\mathbf{P}}$ and $\mathbf{NO}_2^{\mathbf{P}}$ characterization, and quantifier swapping

Theorem 4.6. $\mathbf{prO}_2^{\mathbf{P}}$ *is low for* $\mathbf{O}_2^{\mathbf{P}}$, $\mathbf{NO}_2^{\mathbf{P}}$, *and* $\mathbf{S}_2^{\mathbf{P}}$.

Proof. Let $\Gamma \in \{\mathbf{O}_2^{\mathbf{P}}, \mathbf{NO}_2^{\mathbf{P}}, \mathbf{S}_2^{\mathbf{P}}\}$ and let $L \in \Gamma^{\mathbf{prO}_2^{\mathbf{P}}}$. Let p be a polynomial and $M^\bullet(x,y,z)$ be a $p(n)$-time deterministic Turing machine that correctly decides L according to the definition of the respective class $\Gamma^\bullet$, given loose oracle access to $\Pi \in \mathbf{prO}_2^{\mathbf{P}}$, and let A be a verifier for Π. We construct a new verifier $A'\big(x, (y,U), (z,V)\big)$ that shows that $L \in \Gamma$.

For $n \in \mathbb{N}$, let $1, \ldots, p(n)$ be all possible lengths of oracle queries that could be made by $M^\bullet$ on inputs of length n. Define

$$W_n \stackrel{\Delta}{=} \big(w_1^{(0)}, \ldots, w_{p(n)}^{(0)}, w_1^{(1)}, \ldots, w_{p(n)}^{(1)}\big).$$

Theorem 4.3 in [HV26, ECCC version] shows that $\mathbf{P}^{\mathbf{prO}_2^{\mathbf{P}}} \subseteq \mathbf{O}_2^{\mathbf{P}}$. Its proof shows specifically that there is a deterministic polynomial-time procedure $S(q,U,V)$ that correctly computes the label (yes/no) of a query q in the promise set (i.e., $q \in \Pi_{\text{YES}} \cup \Pi_{\text{NO}}$) made on input x, as long as either $U = W_{|x|}$ or $V = W_{|x|}$, regardless of the value of the other variable (this is simply because the corresponding irrefutable certificate $w_{|q|}^{(b)}$ will win the game judged by A over any possible certificate supplied for the answer $1-b$, whether in the same string or the other one). It must be enough for $M^\bullet$ to do its job: loose oracle access guarantees that its answer (according to the definition of the respective class! for these classes it means that a certain [oblivious, where applicable] irrefutable certificate remains irrefutable) will be correct irrespectively of the oracle answers outside the promise set. Given this, we consider two cases:

- Suppose $x \in L$. There exists an irrefutable yes-certificate y^* for x in M^{Π}. Consequently, $(y^*, W_{|x|})$ is an irrefutable yes-certificate for x in A, which is oblivious if y^* is.
- Suppose $x \notin L$. There exists an irrefutable no-certificate z^* for x in M^{Π}. Consequently, $(z^*, W_{|x|})$ is an irrefutable no-certificate for x in A, which is oblivious if z^* is. □

Theorem 4.7. $\exists_{\mathbf{o}} \cdot \mathbf{pr\,co\text{-}ONP} = \mathbf{ONP}^{\mathbf{prONP}} = \mathbf{O_2^P}^{\mathbf{prO_2^P}} = \mathbf{O_2^P}$.

Proof. The first three containments follow trivially from the definitions and the last equality from Theorem 4.6:

$$\exists_{\mathbf{o}} \cdot \mathbf{pr\,co\text{-}ONP} \subseteq \mathbf{ONP}^{\mathbf{prONP}} \subseteq \mathbf{O_2^P}^{\mathbf{prO_2^P}} = \mathbf{O_2^P}.$$

We now show the remaining containment.

$\boxed{\mathbf{O_2^P} \subseteq \exists_{\mathbf{o}} \cdot \mathbf{pr\,co\text{-}ONP}.}$ Consider $L \in \mathbf{O_2^P}$ and let $V(x, y, z)$ be its verifier with oblivious witnesses y_n and z_n (in what follows we denote $n = |x|$). We will guess y with the existential quantifier and then query the following promise problem Π:

$$\begin{aligned} \Pi_{\text{YES}} &= \{(x, y) : x \notin L \wedge \exists z\ V(x, y, z) = 0\}, \\ \Pi_{\text{NO}} &= \{(x, y) : x \in L \wedge \forall z\ V(x, y, z) = 1\}. \end{aligned}$$

This problem $\Pi \in \mathbf{prONP}$ with oblivious witnesses z_n and the verification procedure "guess z; if $V(x, y, z) = 0$, then accept, otherwise reject".

Observe that $L = \exists_{\mathbf{o}} \cdot (\Pi_{\text{NO}}, \Pi_{\text{YES}})$:

- $\forall x \in L\ \exists y$ (namely, the oblivious witness y_n that remains oblivious) that beats every z for V, that is, $(x, y) \in \Pi_{\text{NO}}$;
- $\forall x \notin L$ every witness y must be beaten by the oblivious witness z_n for V, that is, $(x, y) \in \Pi_{\text{YES}}$.

□

In the same vein, we can also characterize $\mathbf{NO_2^P}$ in terms of the dot-operator and observe that it provides quantifier swapping, namely, $\exists \cdot \mathbf{co\text{-}ONP} \subseteq \forall_{\mathbf{o}} \cdot \mathbf{NP}$ (actually, even a slightly stronger statement about promise **co-ONP** "oracles" follows).

Theorem 4.8. $\exists \cdot \mathbf{pr\,co\text{-}ONP} = \mathbf{NP}^{\mathbf{prONP}} = \mathbf{NO_2^P}^{\mathbf{prO_2^P}} = \mathbf{NO_2^P} \subseteq \forall_{\mathbf{o}} \cdot \mathbf{NP}$.

Proof. As before, the first three containments follow trivially from the definitions and the last equality from Theorem 4.6:

$$\exists \cdot \mathbf{pr\,co\text{-}ONP} \subseteq \mathbf{NP}^{\mathbf{prONP}} \subseteq \mathbf{NO_2^P}^{\mathbf{prO_2^P}} = \mathbf{NO_2^P}.$$

We now show the remaining containments.

$\boxed{\mathbf{NO_2^P} \subseteq \exists \cdot \mathbf{pr\,co\text{-}ONP}.}$ The same proof as in Theorem 4.7 works, but the dot-operator will be non-oblivious.

$\boxed{\mathbf{NO_2^P} \subseteq \forall_{\mathbf{o}} \cdot \mathbf{NP}.}$ To prove the last containment, consider a language L in $\mathbf{NO_2^P}$ with a verifier $V(x, y, z)$ and oblivious witnesses z_n. Then define

$$O = \{(x, z) : \exists y\ V(x, y, z) = 1\} \in \mathbf{NP}.$$

Then $\overline{L} = \exists_{\mathbf{o}} \cdot \overline{O}$, where z_n is still playing the role of an oblivious witness for $\exists_{\mathbf{o}}\cdot$. □

5 Unique witnesses with an NP (or ONP) oracle capture the Linear Ordering Principle

The classes $\mathbf{L}_2^{\mathbf{P}}$ and $\mathbf{OL}_2^{\mathbf{P}}$ certainly belong to the second level of the polynomial hierarchy (and its input-oblivious analogue, respectively). In this section we show tighter bounds by observing that the witness for the basic machine can be made unique. Furthermore, we characterize LOP-based classes using the dot-operators with unique witnesses (input-oblivious or not).

We start with proving an upper bound on $\mathbf{OMIN} = \mathbf{OL}_2^{\mathbf{P}}$ using symmetric unambiguous input-oblivious computations with an input-oblivious nondeterministic oracle. We also reformulate it using the dot-operator, which implies that **NP**, **ONP**, **UP**, **OUP** can use an **ONP** oracle instead of an $\mathbf{OL}_2^{\mathbf{P}}$ oracle. Another corollary is an upper bound for $\mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{ONP}}$ and $\mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{NP}}$ classes, which we will later use to characterize multiple classes using symmetric unambiguous input-oblivious computations with an oracle.

We start, however, with a warm-up for the non-oblivious case ($\mathbf{L}_2^{\mathbf{P}}$).

5.1 Warm-up: The non-oblivious case

Theorem 5.1. $\mathbf{L}_2^{\mathbf{P}} \subseteq \exists_{\mathbf{u}} \cdot \mathbf{co\text{-}NP} \cap \forall_{\mathbf{u}} \cdot \mathbf{NP}$.

Proof. Consider $L \in \mathbf{L}_2^{\mathbf{P}}$ defined by a linear order $<_x$ computable by a polynomial-time function $V_x(y,z)$ on bit strings of a certain polynomial length $\ell(n)$. Consider

$$O = \{(x,w) \colon \exists u, u' \ (0u <_x 1w \vee 1u' <_x 1w)\}$$

Obviously, $O \in \mathbf{NP}$.

For every x, there exists a unique minimal element of the order $<_x$. In the case $x \in L$ this is some string $1w_x$, and it is the only w such that $(x,w) \notin O$, because $u' = w_x$ will beat all other candidates. In the case $x \notin L$ there is no w such that $(x,w) \in O$, because the corresponding minimal element $0u$ will beat them all.

Thus $\mathbf{L}_2^{\mathbf{P}} \in \exists_{\mathbf{u}} \cdot \mathbf{co\text{-}NP}$. The theorem statement follows by the symmetry of $\mathbf{L}_2^{\mathbf{P}}$. □

5.2 Symmetric input-oblivious unambiguous computations

5.2.1 Input-oblivious unique witnesses: the main theorem

Theorem 5.2. $\mathbf{OMIN} \subseteq \mathbf{SOUP}^{\mathbf{ONP}[1]}$. *Moreover,* $\mathbf{OMIN} \subseteq \mathbf{SOUON}$.

Proof. Consider $L \in \mathbf{OMIN}$. Let $(m_n)_{n\in\mathbb{N}}$ be the oblivious minima of the corresponding linear order. Define

$$O = \{(x,w) : \exists w' \ w' <_{|x|} w\} \tag{5.1}$$

answering the question that w is *not* the minimum of $<_{|x|}$. Observe that $O \in \mathbf{ONP}$, because $w' = m_{|x|}$ serves as an oblivious witness.

Also in terms of Definition 2.17

$$x \in L \iff D(x, m_n) = 1 \iff \exists w \ ((x,w) \notin O \ \wedge \ D(x,w) = 1)$$

and by symmetry

$$x \notin L \iff D(x, m_n) = 0 \iff \exists w \ ((x,w) \notin O \ \wedge \ D(x,w) = 0)\,.$$

Note that $w = m_n$ serves as an oblivious witness for all positive instances and also serves as an oblivious witness for all negative instances. It is unique, because if $(x,w) \notin O$, then w is the minimum of $<_{|x|}$.

This proves $\mathbf{OMIN} \subseteq \mathbf{SOUP}^{\mathbf{ONP}[1]}$. In order to prove $\mathbf{OMIN} \subseteq \mathbf{SOUON}$, one needs to incorporate the condition $D(x,w) = 1$ into the "oracle" language while keeping the same (identical) witnesses for "yes" and "no" answers. The price for it is two different oracles instead of just one. Define

$$O_b = \{(x,w) : \exists w' \ \big(w' <_{|x|} w \ \vee \ D(x,w) = 1-b\big)\}.$$

Note that $O_0, O_1 \in \mathbf{ONP}$, because their difference from $O \in \mathbf{ONP}$ is in the case where $D(x,w) = 1-b$, but in this case any witness w' does the job (including $w' = m_n$).

Observe that $L = \exists_{\mathbf{ou}} \cdot \overline{O_1} \cap \forall_{\mathbf{ou}} \cdot O_0$. Indeed, if $x \in L$, then $(x, m_n) \notin O_1$, because $D(x, m_n) = 1$ and there is no $w' <_n m_n$. No other w can do the job for any y of the same length: if $(y,w) \notin O_1$, then in particular there is no $w' <_n w$, but it means $w = m_n$. The same thing happens for the second part. □

5.2.2 Corollaries for $\mathbf{OL}_2^{\mathbf{P}}$ and ONP oracles

In what follows we will use two simple corollaries of Theorem 5.2.

We start with observing that it now follows that an $\mathbf{OL}_2^{\mathbf{P}}$ oracle is no more powerful than **ONP** if the base machine can manage at least a little bit of nondeterminism.

Corollary 5.3. *$\mathcal{C}^{\mathbf{OL}_2^{\mathbf{P}}} \subseteq \mathcal{C}^{\mathbf{ONP}}$, where $\mathcal{C} = \mathbf{NP}$, **ONP**, **UP**, or **OUP**. Moreover, this class is contained in $\exists_* \cdot \mathbf{co\text{-}ONP}$, where $\exists_* \cdot$ is the corresponding dot-operator.*

Proof. By Corollary 4.2 and Theorem 5.2, $\mathbf{NP}^{\mathbf{OL}_2^{\mathbf{P}}} = \exists \cdot \mathbf{OL}_2^{\mathbf{P}} \subseteq \exists \cdot \exists_{\mathbf{ou}} \cdot \mathbf{co\text{-}ONP} \cap \forall_{\mathbf{ou}} \cdot \mathbf{ONP} \subseteq \exists \cdot \exists_{\mathbf{ou}} \cdot \mathbf{co\text{-}ONP} = \exists \cdot \mathbf{co\text{-}ONP}$ and similarly for other classes (replacing $\exists \cdot$ with the corresponding dot-operator). □

The following classes have been used in [HV26], upper bounds for them are now simple corollaries of Theorem 5.2.

Corollary 5.4.

1. $\mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{ONP}} \subseteq \mathbf{SOUP}^{\mathbf{ONP}}$.
2. $\mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{NP}} \subseteq \mathbf{SOUP}^{\mathbf{NP}}$.

Proof. By Theorem 5.2, $\mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{ONP}} \subseteq \mathbf{P}^{\mathbf{SOUP}^{\mathbf{ONP}[1]},\mathbf{ONP}}$. Then for $L \in \mathbf{SOUP}^{\mathbf{ONP}[1]}$ and $L' \in \mathbf{ONP}$ one can use $0 \cdot L \cup 1 \cdot L'$ as an $\mathbf{SOUP}^{\mathbf{ONP}}$ oracle instead (the unique witness for $1 \cdot L'$ can be set to 1^t for appropriate t and then the query to L' can be answered by the oracle). Then $\mathbf{P}^{\mathbf{SOUP}^{\mathbf{ONP}}} = \mathbf{SOUP}^{\mathbf{ONP}}$ by Theorem 4.3.

One can do the same with the **NP** oracle. □

6 New upper bounds for MA- and BPP-based classes

The results we proved in previous sections yield several new upper bounds.

In this section we will recall the derandomization scheme used in [HV26] (and based on [Kor22, KP24]) and extend it to derandomizing probabilistic oracle machines, which will also allow us to strengthen Theorem 4.1 to $\mathbf{BPP}^{\mathbf{OL}_2^{\mathbf{P}}} = \mathbf{OL}_2^{\mathbf{P}}$.

As we advance on these matters, we discuss new upper bounds for **MA**, **OMA**, $\mathbf{P}^{\mathbf{prMA}}$, $\mathbf{P}^{\mathbf{OprMA}}$, and another related class $\mathbf{BPP}^{\mathbf{ONP},\mathbf{OL}_2^{\mathbf{P}}}$.

6.1 Basic containments of MA

Chakaravarthy and Roy [CR06] prove that $\mathbf{MA} \subseteq \mathbf{NO}_2^{\mathbf{P}}$ and $\mathbf{OMA} \subseteq \mathbf{O}_2^{\mathbf{P}}$. We now strengthen these containments.

Definition 6.1 ([Kor22, Definitions 6, 7], [Kor21, Definitions 7, 8])**.** PRG *is the following search problem: given* 1^n*, output a pseudorandom generator*[4] $R = (x_1, \ldots, x_m)$*, that is, an array of strings* $x_i \in \{0,1\}^n$ *such that for every* n*-input circuit* C *of size* n*:*

$$\left| \Pr_{x \leftarrow U(R)} \{C(x) = 1\} - \Pr_{y \leftarrow U(\{0,1\}^n)} \{C(y) = 1\} \right| \leq \frac{1}{n}.$$

In [HV26], it was observed that such a generator can be computed in deterministic polynomial time, given oracle access to $\mathbf{OL}_2^{\mathbf{P}}$.

Lemma 6.2 ([HV26, Proposition 5.5])**.** PRG *can be computed in deterministic polynomial time, given oracle access to* $\mathbf{OL}_2^{\mathbf{P}}$*.*

Proof (Sketch). Korten [Kor22] constructs a PRG using a single (and, of course, input-oblivious) query to Avoid. Korten and Pitassi [KP24] construct a deterministic polynomial-time truth-table reduction of Avoid to a language in $\mathbf{L}_2^{\mathbf{P}}$. More details in the input-oblivious setting are given in [HV26, Proposition 5.5]. □

While we know that $\mathbf{P}^{\mathbf{prMA}} \subseteq \mathbf{L}_2^{\mathbf{P}}$ [HV26], using this lemma one can give a tighter derandomization for **MA** and **OMA**.

Lemma 6.3. $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{OL}_2^{\mathbf{P}}}$ *and* $\mathbf{OMA} \subseteq \mathbf{ONP}^{\mathbf{OL}_2^{\mathbf{P}}}$*.*

Proof. The nondeterministic machine will guess Merlin's witness and then derandomize Arthur's circuit using a PRG computed using the $\mathbf{OL}_2^{\mathbf{P}}$ oracle by Lemma 6.2.

Given such a PRG, the final result of the computation can be as follows:

1. The input is a negative instance, all guesses are bad, then PRG forces the circuit to reject.
2. The input is a positive instance, and the guess is good, then PRG forces the circuit to accept.
3. The input is a positive instance and the guess is not good, but then we do not care about the answer, because the $\mathbf{NP}^{\mathbf{OL}_2^{\mathbf{P}}}$ machine will accept for the good guess (case 2 above) and that's enough.

The upper bound for **OMA** is analogous. □

In the following theorem the containment $\mathbf{OMA} \subseteq \mathbf{ONP}^{\mathbf{ONP}}$ is superseded by a stronger upper bound for $\mathbf{P}^{\mathbf{OprMA}}$ we prove in what follows, but the other two containments are incomparable to our further upper bound for $\mathbf{P}^{\mathbf{prMA}}$ (the latter bound has **NP** as the oracle class).

[4] Korten does not say that m has a polynomial dependence on n though we think it is assumed, and anyway his theorem provides a construction with $m = n^6$.

Theorem 6.4.

1. $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{ONP}}$ *and* $\mathbf{OMA} \subseteq \mathbf{ONP}^{\mathbf{ONP}}$.

2. $\mathbf{MA} \subseteq \mathbf{NOL}_{\mathbf{2}}^{\mathbf{P}}$.

Proof. 1. By Lemma 6.3 and Corollary 5.3, $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}} \subseteq \mathbf{NP}^{\mathbf{ONP}}$ and $\mathbf{OMA} \subseteq \mathbf{ONP}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}} \subseteq \mathbf{ONP}^{\mathbf{ONP}}$.

2. By Lemma 6.3, $\mathbf{MA} \subseteq \mathbf{NP}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}}$, then by Corollary 4.2 $\mathbf{NP}^{\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}} = \exists \cdot \mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}$. Finally, by Corollary 3.5 $\exists \cdot \mathbf{OL}_{\mathbf{2}}^{\mathbf{P}} \subseteq \mathbf{NOL}_{\mathbf{2}}^{\mathbf{P}}$. □

The next statement is also incomparable to our further upper bound for $\mathbf{P}^{\mathbf{OprMA}}$, because for **co-OMA** we can formulate it with the dot-operator (yet at the price of using a *promise* version of **co-ONP**). The reason we can use the dot-operator is that both queries to **ONP** have the same polarity and thus can be merged. For **OMA** (instead of **co-OMA**) the two oracle queries would have opposite polarities, which prevents us from strengthening the upper bound $\mathbf{SOUP}^{\mathbf{ONP}[2]}$ that we could potentially obtain to an unambiguous oblivious dot-operator applied to an oblivious nondeterministic class.

Theorem 6.5. $\mathbf{co\text{-}OMA} \subseteq \exists_{\mathbf{ou}} \cdot \mathbf{pr}\,\mathbf{co\text{-}ONP}$.

Proof.

Proof plan and our linear order $<_n$. Our plan is as follows. Lemma 6.2 computes PRG using an $\mathbf{OL}_{\mathbf{2}}^{\mathbf{P}}(= \mathbf{OMIN})$ oracle; thus similarly to Theorem 4.1 there is a deterministic polynomial-time procedure D that can extract this PRG from the minimum m_n of a polynomial-time decidable input-oblivious linear order $<_n$ (given 1^n on the input). Since the queries to the oracle in Lemma 6.2 are solving Avoid instances generated on input 1^n in polynomial time, D even does not need to know them, it can construct them itself if needed. Similarly to Theorem 5.2, we observe that the minimality of this unique string m_n can be verified using a single **co-ONP** query to the language (5.1).

We then use this PRG to derandomize **co-OMA** to **pr co-ONP** (namely, we guess obliviously Merlin's proof and then use Arthur on every string in PRG to verify it by computing the probability of acceptance). This is a query to a promise problem, because it is conditioned on the success of the first query. In total, we have two **co-ONP**-type queries; it remains to merge them into a single one while preserving the unambiguous obliviousness of m_n on the promise set that we are about to define. Further details follow.

The definition of our promise problem Q. Let the **OMA** language L be defined by a polynomial-time verifier $A(x, y, r)$ with perfect completeness, random string r, and oblivious witnesses $(y_n)_{n\in\mathbb{N}}$. We can assume that its error probability is less than $1/4$. We define the following promise problem $Q = (Q_{\text{YES}}, Q_{\text{NO}})$, where u, u' are elements of the order $<_n$:

$$(x,u) \in Q_{\text{NO}} \iff x \in L \wedge \exists u'\ \exists y\ (u' <_n u \vee \Pr_{r\leftarrow D(u)} A(x,y,r) = 1 > 1/2).$$

$$(x,u) \in Q_{\text{YES}} \iff \forall u'\ \forall y\ ((u <_n u' \vee u = u') \wedge \Pr_{r\leftarrow D(u)} A(x,y,r) = 1 \leq 1/2).$$

The defining condition of the set Q_{NO} says that, for $x \in L$, (x, u) must be rejected if a wrong minimum u is supplied as well as if A has a witness y with a good acceptance probability ($> 1/2$) on the PRG strings (and thus overall). Informally it means that there is no supporting evidence for $x \notin L$.

Such a supporting evidence exists if $u = m_n$ and for every witness y, A has a small acceptance probability ($\leq 1/2$) on the PRG strings (and thus overall the probability is not 1, which implies $x \notin L$ by the perfect completeness of A). This is exactly the condition defining the set Q_{YES}. In particular, $Q_{\text{YES}} \cap Q_{\text{NO}} = \emptyset$ (as $(x,u) \in Q_{\text{NO}} \Rightarrow x \notin L$).

The promise problem is useful: $\overline{L} \in \exists_{\mathbf{ou}} \cdot (Q_{\text{YES}}, Q_{\text{NO}})$. If $x \in \overline{L}$, then there exists u (namely, $u = m_n$, thus it is oblivious) such that $(x,u) \in Q_{\text{YES}}$. No other u would do, because it would fail the minimality check $\forall u'((u <_n u') \vee (u = u'))$; thus $u = m_n$ is the unique solution.

On the other hand, if $x \in L$, then for every u, $(x,u) \in Q_{\text{NO}}$. Indeed, if $u \neq m_n$, then there exists $u' < u$ (namely, $u' = m_n$), and if $u = m_n$, then the PRG will force the probability of acceptance to be close to 1.

Overall, our language $\overline{L} \in$ **co-MA** is defined by the existence of the oblivious unambiguous witness $u = m_n$ that satisfies $(q,u) \in Q_{\text{YES}}$, that is, a query to $(Q_{\text{YES}}, Q_{\text{NO}})$.

Our promise problem is in **pr co-ONP**. It remains to check that the complement of our promise problem, $(Q_{\text{NO}}, Q_{\text{YES}})$, is in **prONP**. We define the verification procedure as the machine that on input $((x,u),(u',y))$ outputs "$u' <_n u$ or $\Pr > 1/2$", where the probability is checked by running $A(x,y,r)$ on every r in the PRG and counting the answers.

We prove that queries to Q_{NO} are answered correctly and, moreover, $(u',y) = (u_n, y_n)$ is an oblivious witness. Indeed, if $(x,u) \in Q_{\text{NO}}$, then either u is not the minimum and then $u' = u_n$ makes the condition $u' <_n u$ true, or $u = u_n$ and then, for $y = y_n$, by the pseudorandomness property the condition $x \in L$ guarantees that the probability of acceptance must be much more than $1/2$. There may be other witnesses that the verification procedure accepts (if $u \neq u_n$ or Merlin has another proof $y \neq y_n$), but it does not matter as no uniqueness is required from (u',y).

If $(x,u) \in Q_{\text{YES}}$, then the verification procedure cannot accept any (u',y): by the definition of Q_{YES}, $u = m_n$, so there is no $u' <_n m_n$; the second possibility is $\Pr > 1/2$, but by the definition of Q_{YES} the probability is at most $1/2$. □

The last two theorems yield:

Corollary 6.6. **OMA** $\cap$ **co-OMA** $\subseteq$ **SOUprON**.

Note that in contrast to **BP** $\cdot$ **NP** = **AM** [ZF87, Sch89], it is not difficult to see that the input-oblivious version **BP**$\cdot$**ONP** is contained in **OMA**; more precisely, **BP**$\cdot$**prONP** = **OMA**.

Lemma 6.7. **OMA** = **BP** $\cdot$ **prONP**.

Proof. $\boxed{\textbf{BP} \cdot \textbf{prONP} \subseteq \textbf{OMA}.}$ Let $L \in \textbf{BP} \cdot \Pi$ for some $\Pi \in$ **prONP** and let $A(x,r,w)$ (formally, $A((x,r),w)$) be the corresponding deterministic polynomial-time machine for Π with input-oblivious witnesses $(w_m)_{m \in \mathbb{N}}$ (we are interested in $m = |(x,r)|$ and can assume that this mapping of the input lengths is injective). That is:

- $(x,r) \in \Pi_{\text{YES}} \implies A(x,r,w_{|(x,r)|}) = 1$,
- $(x,r) \in \Pi_{\text{NO}} \implies \forall w\ A(x,r,w) = 0$.

We will now show that $L \in$ **OMA** and $w'_n = w_{|(x,r)|}$ will serve as input-oblivious Merlin's proofs for $x \in \{0,1\}^n \cap L$. Indeed,

- $x \in \{0,1\}^n \cap L \implies \Pr_r[A(x,r,w'_n) = 1] \geq \Pr_r[(x,r) \in \Pi_{\text{YES}}] \geq 3/4$,
- $x \in \{0,1\}^n \cap \bar{L} \implies \Pr_r[\forall w\ A(x,r,w) = 0] \geq \Pr_r[(x,r) \in \Pi_{\text{NO}}] \geq 3/4$, therefore $\forall w\ \Pr_r[A(x,r,w) = 1] \leq \Pr_r[\exists u\ A(x,r,u) = 1] \leq 1/4$.

$\mathbf{OMA} \subseteq \mathbf{BP} \cdot \mathbf{prONP}$. Now, suppose that $L \in \mathbf{OMA}$ with perfect completeness (one-sided error). By definition, there exists a deterministic polynomial-time verification machine $M(x, r, w)$ and $(w_n)_{n\in\mathbb{N}}$ such that:

- $x \in \{0,1\}^n \cap L \implies \forall r\ M(x, r, w_n) = 1$,
- $x \in \{0,1\}^n \cap \bar{L} \implies \forall w\ \Pr_r[M(x, r, w) = 1] \leq 1/4$.

Let $q(n) = |w_n|$. Consider the following promise problem $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$:

$$\Pi_{\text{YES}} = \left\{(x, r_1, r_2, \ldots, r_{q(|x|)}) \mid \forall i\ M(x, r_i, w_n) = 1\right\}.$$

$$\Pi_{\text{NO}} = \left\{(x, r_1, r_2, \ldots, r_{q(|x|)}) \mid \forall w\ \exists i\ M(x, r_i, w) = 0\right\}.$$

Then, we have the following for every $x \in \{0,1\}^n$ and every $\bar{r} = (r_1, r_2, \ldots, r_{q(|x|)})$:

- $x \in L \implies \forall \bar{r}\ M(x, r_i, w_n) = 1 \implies \forall \bar{r}\ (x, \bar{r}) \in \Pi_{\text{YES}}$.
- $x \notin L \implies \Pr_{\bar{r}}[\exists w\ \forall i\ M(x, r_i, w) = 1] \leq 2^{q(n)} \cdot \frac{1}{4^{q(n)}} \ll 1/4$. Thus $\Pr_{\bar{r}}[(x, \bar{r}) \in \Pi_{\text{NO}}] \geq 3/4$.

The **ONP** algorithm for Π has witness w and simply checks $\bigwedge_{i=1}^{q(|x|)} M(x, r_i, w)$ on input $(x, \bar{r})$. This check returns true for $(x, \bar{r}) \in \Pi_{\text{YES}}$ (with witness $w = w_n$) and false for $(x, \bar{r}) \in \Pi_{\text{NO}}$ (with any witness). By definition of w_n, is an oblivious witness for Π_{YES}, thus indeed $\Pi \in \mathbf{prONP}$.

To finish the proof we observe that **OMA** with two-sided error is equal to **OMA** with one-sided error. □

Since $\mathbf{BP} \cdot \mathbf{pr\,co\text{-}ONP} = \mathbf{co\text{-}(BP} \cdot \mathbf{prONP)} \overset{\text{L6.7}}{=\!=} \mathbf{co\text{-}OMA} \overset{\text{T6.5}}{\subseteq} \exists_{\mathbf{ou}} \cdot \mathbf{pr\,co\text{-}ONP}$, we get

Corollary 6.8. $\mathbf{BP} \cdot \mathbf{pr\,co\text{-}ONP} \subseteq \exists_{\mathbf{ou}} \cdot \mathbf{pr\,co\text{-}ONP}$.

That is, the *two-sided bounded probability* dot-operator can be replaced here by an input-oblivious *unambiguous* dot-operator!

6.2 Corollaries for classes that use Merlin–Arthur oracles

$\mathbf{P}^{\mathbf{prOMA}}$ and $\mathbf{P}^{\mathbf{prMA}}$ classes have been the smallest Karp–Lipton collapse classes to the date. In [HV26] it was conjectured that the [CR11] collapse proof goes through for an even smaller class. We now prove that this is indeed so for $\mathbf{P}^{\mathbf{OprMA}}$. Thus, these are the currently smallest non-input-oblivious and input-oblivious classes, respectively, and previous containments from [HV26] together with Corollary 5.4 put these into the corresponding classes of symmetric unambiguous oblivious computations with an oracle.

Proposition 6.9.

1. $\mathbf{P}^{\mathbf{prMA}} \subseteq \mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{NP}}$ *[HV26, Corollary 5.6].*
2. $\mathbf{P}^{\mathbf{OprMA}} \subseteq \mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}},\mathbf{ONP}}$.
3. *If* $\mathbf{NP} \subseteq \mathbf{P/poly}$, *then* $\mathbf{prMA} \subseteq \mathbf{OprMA}$ *and* $\mathbf{PH} = \mathbf{P}^{\mathbf{OprMA}}$.

Proof.

2. *The* $\mathbf{OL}_2^{\mathbf{P}}$ *oracle* L_0. Note that we can store a PRG in the minimum of an oblivious linear order. It is easy to see from the proof of Lemma 6.2 that a polynomial-time Turing machine with some $\mathbf{OL}_2^{\mathbf{P}}$ oracle L_0 provides not just PRG, but a polynomial-time decidable linear order

so that PRG G_S can be decoded in polynomial time from the minimum m_S of this order by some decoder D, where $S = S(n)$ is the bit-size of the circuits that we intend to derandomize.

The **ONP** *oracle* L_1. Let $(\Pi_{\text{YES}}, \Pi_{\text{NO}}) \in \mathbf{OprMA}$ with a randomized polynomial-time verifier V, and input-oblivious witnesses y_n. Let $C_r(x, y)$ be the corresponding circuit of polynomial size $S(n)$, where the random string r is fixed. Let $q(n)$ be the (polynomial) length of pairs (m, y), where m is an element in the Korten–Pitassi linear order computing the PRG G_S, and y is Merlin's proof. Define

$$L_1 = \Big\{(x, m) : m \in \{0,1\}^{q(|x|)} \quad \text{and} \quad \Big(\exists m'\ m' <_{S(|x|)} m \quad \text{or} \quad \exists y \Pr_{g \in D(m)}[V(x, y, g) = 1] \geq \frac{1}{2}\Big)\Big\},$$

where g is uniformly distributed on the list of strings in $D(m)$. Clearly, $L_1 \in \mathbf{NP}$.

This oracle is in **ONP**. Indeed, $u_n = (m_{S(n)}, y_n)$ is an oblivious witness: any other successful witness $u = (m', y)$ must either provide a smaller $m' <_{S(|x|)} m$ (but then the even smaller $m_{S(n)}$ would also work) or provide an acceptable Merlin's proof y if the acceptance probability is computed according to PRG $G_S = D(m_{S(n)})$ (which means $(x,) \notin \Pi_{\text{NO}}$ because of the derandomization properties of G_S).

The simulation. The new oracle machine simulates the old one; given a query q, it computes $m_{S(|q|)}$ using the oracle L_0; then it queries L_1 for $(q, m_{S(|q|)})$ and uses its answer as the oracle's answer to q.

For $x \in \{0,1\}^n$, if $x \in \Pi_{\text{YES}}$, then by perfect completeness $V(x, y_n, g) = 1$ for every g, thus $(x, m_{S(n)}) \in L_1$. If $x \in \Pi_{\text{NO}}$, then the first condition $(\exists m' \ldots)$ fails for $m = m_{S(n)}$ and no y reaches the threshold $1/2$ on $G_{S(n)}$.

Therefore, all queries in $\Pi_{\text{YES}} \cup \Pi_{\text{NO}}$ are answered correctly; the answers to the other queries do not matter by the definition of loose access. Thus $L \in \mathbf{P}^{L_0, L_1} \subseteq \mathbf{P}^{\mathbf{OL}_2^{\mathbf{P}}, \mathbf{ONP}}$.

3. Assume $\mathbf{NP} \subseteq \mathbf{P/poly}$. We show that $\mathbf{prMA} \subseteq \mathbf{OprMA}$; then $\mathbf{PH} = \mathbf{P}^{\mathbf{prMA}} \subseteq \mathbf{P}^{\mathbf{OprMA}}$ follows immediately from the collapse to $\mathbf{P}^{\mathbf{prMA}}$ due [CR11].

Similarly to the classical proof of $\mathbf{MA} \subseteq \mathbf{AM}$, we amplify the probability of success by repeating Arthur's verification algorithm enough times; let $C_{x,\bar{r}}(w)$ be the resulting circuit accepting Merlin's proofs that passed all the checks.

Then we use the assumption ($\mathbf{NP} \subseteq \mathbf{P/poly}$) to make Merlin's proofs oblivious (namely, Merlin will provide circuits E_n for the CircuitSAT search problem). Our new Arthur will compute a candidate satisfying assignment w for $C_{x,\bar{r}}$ using Merlin's circuits, and output $C_{x,\bar{r}}(w)$.

Incorrect circuits won't harm the obliviousness: since the satisfying assignment they found has been verified, the "good" search circuits E_n will also found one, and thus will work as a witness. □

Corollary 6.10.

1. $\mathbf{P}^{\mathbf{prMA}} \subseteq \mathbf{SOUP}^{\mathbf{NP}}$.
2. $\mathbf{P}^{\mathbf{OprMA}} \subseteq \mathbf{SOUP}^{\mathbf{ONP}}$.

6.3 Corollaries for probabilistic oracle computations

We now proceed to derandomizing computations based on probabilistic oracle machines.

First, observe that, as any deterministic Turing machine can be transformed into a Boolean circuit, one can use a PRG to derandomize **BPP** algorithms. We observe that this derandomization can be extended to **BPP** algorithms with oracle access to problems that admit small

(i.e., polynomial-size) circuits. An A-oracle circuit is a Boolean circuit that has additionally gates or arbitrary arity that query A. We begin with the following observation:

Observation 6.11. *Let language A be computed by Boolean circuits of size $s_A(n)$ and let $C(y_1, \ldots, y_t)$ be an A-oracle circuit of size $s_C \geq n$. Then there exists a (non-oracle) Boolean circuit $C'(y_1, \ldots, y_t)$ of size at most $s_A(s_C)$ such that for all $y \in \{0,1\}^t : C'(y) = C(y)$.*

Given this observation, we can use an $\mathbf{OL_2^P}$ oracle to construct a PRG and thereby derandomize $\mathbf{BPP}^A$, extending the previous results.

Lemma 6.12. *Suppose $A \in \mathbf{P/poly}$. Then $\mathbf{BPP}^A \subseteq \mathbf{P}^{A,\mathbf{OL_2^P}}$.*

The proof goes along the lines of Theorem 5.4 and Corollary 5.6 from [HV26].

Proof. Let $L \in \mathbf{BPP}^A$, and let $M^A(x, r)$ be the corresponding deterministic Turing machine that runs in time $p(|x|)$ for some polynomial $p(n)$ and uses random string r. Given an input x of size n, let $C(y) = M^A(x, y)$. That is, C is an A-oracle circuit obtained by hard-coding x into the machine $M^A(x, y)$ and subsequently converting it into a circuit. By Observation 6.11, there exists a non-oracle circuit $C'(y)$ that computes the same function as $C(y)$. Since the size of $C(y)$ is poly(n), the size of $C'(y)$ is also poly(n), perhaps with a different polynomial degree. Let $h(n)$ denote the size of $C'(y)$.

We now invoke Lemma 6.2 with $1^{h(n)}$ to construct the appropriate pseudorandom strings $g_1, \ldots, g_m \in \{0,1\}^{h(n)}$. Finally, for every $i \in [m]$, we run $M^A(x, g_i)$ deterministically, using the oracle A to answer its queries, and accept if and only if $M^A(x, g_i)$ accepts for at least half of the g_i's. □

Corollary 6.13.

1. *For any $A \in \mathbf{P/poly}$,* $\mathbf{BPP}^{A,\mathbf{OL_2^P}} = \mathbf{P}^{A,\mathbf{OL_2^P}}$.
2. $\mathbf{BPP}^{\mathbf{OL_2^P}} = \mathbf{OL_2^P}$.
3. $\mathbf{BPP}^{\mathbf{ONP},\mathbf{OL_2^P}} = \mathbf{P}^{\mathbf{ONP},\mathbf{OL_2^P}} \subseteq \mathbf{SOUP}^{\mathbf{ONP}}$.

Proof. We first prove claim 1 and then use it to prove claims 2 and 3.

1. Let $A \in \mathbf{P/poly}$ and $L \in \mathbf{OL_2^P}$. Define $A' = 0 \cdot A \cup 1 \cdot L$. Observe that $A' \in \mathbf{P/poly}$. By Lemma 6.12,

$$\mathbf{BPP}^{A,L} = \mathbf{BPP}^{A'} \subseteq \mathbf{P}^{A',\mathbf{OL_2^P}} = \mathbf{P}^{A,L,\mathbf{OL_2^P}} = \mathbf{P}^{A,\mathbf{OL_2^P}} \subseteq \mathbf{BPP}^{A,\mathbf{OL_2^P}}.$$

Note that we are relying on the fact that $\mathbf{OL_2^P} = \mathbf{OMIN}$ is closed under union, thus two oracles $L, L' \in \mathbf{OL_2^P}$ can be replaced by $0 \cdot L \cup 1 \cdot L'$.

2. $\mathbf{BPP}^{\mathbf{OL_2^P}} \overset{\text{claim 1}}{=\!=} \mathbf{P}^{\mathbf{OL_2^P}} \overset{\text{T4.1}}{=\!=} \mathbf{OL_2^P}$.
3. $\mathbf{BPP}^{\mathbf{ONP},\mathbf{OL_2^P}} \overset{\text{claim 1}}{=\!=} \mathbf{P}^{\mathbf{ONP},\mathbf{OL_2^P}} \overset{\text{C5.4}}{\subseteq} \mathbf{SOUP}^{\mathbf{ONP}}$. □

References

[Aar07] S. Aaronson. The learnability of quantum states. In *Proceedings of the Royal Society A*, volume 463, pages 3089–3114, 2007.

[AD14] S. Aaronson and A. Drucker. A full characterization of quantum advice. *SIAM J. Comput.*, 43(3):1131–1183, 2014.

[BCG+96] N. H. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon. Oracles and queries that are sufficient for exact learning. *J. Comput. Syst. Sci.*, 52(3):421–433, 1996.

[BF99a] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *STACS*, pages 100–109, 1999.

[BF99b] H. Buhrman and L. Fortnow. Two queries. *J. Comput. Syst. Sci.*, 59(2):182–194, 1999.

[BGM06] E. Böhler, C. Glaßer, and D. Meister. Error-bounded probabilistic computations between MA and AM. *J. Comput. Syst. Sci.*, 72(6):1043–1076, 2006.

[BP25] A. Barenghi and G. Pelosi. *Oblivious RAM (ORAM)*, pages 1718–1721. Springer Nature Switzerland, Cham, 2025.

[BS01] B. Borchert and R. Silvestri. Dot operators. *Theoretical Computer Science*, 262:501–523, 2001.

[Cai07] J.-Y. Cai. $S_2P \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences*, 73(1):25–35, 2007.

[Can96] R. Canetti. More on BPP and the polynomial-time hierarchy. *Inf. Process. Lett.*, 57(5):237–241, 1996.

[CHLR23] Y. Chen, Y. Huang, J. Li, and H. Ren. Range avoidance, remote point, and hard partial truth table via satisfying-pairs algorithms. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, STOC 2023, pages 1058–1066, 2023.

[CHLR26] L. Chen, S. Hirahara, Z. Li, and H. Ren. Symmetric exponential time requires near-maximum circuit size. *J. ACM*, 73(1):4:1–4:21, 2026.

[CHR24] L. Chen, S. Hirahara, and H. Ren. Symmetric exponential time requires near-maximum circuit size. In *STOC 2024*, 2024.

[CK98] M. Chiari and J. Krajíček. Witnessing functions in bounded arithmetic and search problems. *The Journal of Symbolic Logic*, 63(3):1095–1115, 1998.

[CR06] V. T. Chakaravarthy and S. Roy. Oblivious symmetric alternation. In *STACS*, pages 230–241, 2006.

[CR11] V. T. Chakaravarthy and S. Roy. Arthur and Merlin as oracles. *Comput. Complex.*, 20(3):505–558, 2011.

[FGJ+26] N. Fleming, S. Grosser, S. Jain, J. Li, H. Ren, M. Shirley, and W. Yuan. Total search problems in ZPP. In *17th Innovations in Theoretical Computer Science Conference, ITCS 2026*, volume 362 of *LIPIcs*, pages 60:1–60:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026.

[FSW09] L. Fortnow, R. Santhanam, and R. Williams. Fixed-polynomial size circuit bounds. In *Proceedings of the 24th Annual IEEE Conference on Computational Complexity*, pages 19–26, 2009.

[GGLS25] K. Gajulapalli, S. Ghentiyala, Z. Li, and S. Saraogi. Downward self-reducibility in the total function polynomial hierarchy. *Electron. Colloquium Comput. Complex.*, TR25-121, 2025.

[GLS26] S. Ghentiyala, Z. Li, and N. Stephens-Davidowitz. Range avoidance, arthur-merlin, and TFNP. In *Proceedings of the 58th Annual ACM Symposium on Theory of Computing, STOC*, pages 1925–1936. ACM, 2026.

[GLV24] K. Gajulapalli, Z. Li, and I. Volkovich. Oblivious complexity classes revisited: Lower bounds and hierarchies. In *FSTTCS 2024*, volume 323 of *LIPIcs*, pages 23:1–23:19, 2024. Full version at https://eccc.weizmann.ac.il/report/2024/049.

[GM15] O. Goldreich and O. Meir. Input-oblivious proof systems and a uniform complexity perspective on P/poly. *ACM Transactions on Computation Theory (TOCT)*, 7(4):1–13, 2015.

[GO96] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, 1996.

[Gol06] O. Goldreich. On promise problems: A survey. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer, 2006.

[GS88] J. Grollmann and A. L. Selman. Complexity measures for public-key cryptosystems. *SIAM J. Comput.*, 17(2):309–335, 1988.

[HV26] E. A. Hirsch and I. Volkovich. Upper and Lower Bounds for the Linear Ordering Principle. In *STACS 2026*, volume 364 of *LIPIcs*, pages 52:1–52:19, 2026. Full version: ECCC TR25-142, https://eccc.weizmann.ac.il/report/2025/142/.

[Jeř04] E. Jeřábek. Dual weak pigeonhole principle, boolean complexity, and derandomization. *Annals of Pure and Applied Logic*, 129:1–37, 2004.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55(1-3):40–56, 1982.

[KKMP21] R. Kleinberg, O. Korten, D. Mitropolsky, and C. Papadimitriou. Total Functions in the Polynomial Hierarchy. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*, volume 185 of *LIPIcs*, pages 44:1–44:18, 2021.

[KL80] R. M. Karp and R. J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing*, pages 302–309, 1980.

[Kor21] O. Korten. The hardest explicit construction. *CoRR*, abs/2106.00875, 2021.

[Kor22] O. Korten. The hardest explicit construction. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 433–444. IEEE, 2022.

[KP24] O. Korten and T. Pitassi. Strong vs. Weak Range Avoidance and the Linear Ordering Principle. In *Proceedings of the 65th IEEE Annual Symposium on Foundations of Computer Science, (FOCS)*, pages 1388–1407. IEEE, 2024. Full version at https://eccc.weizmann.ac.il/report/2024/076.

[Kra25] J. Krajíček. *Proof Complexity Generators*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2025.

[KW98] J. Köbler and O. Watanabe. New collapse consequences of NP having small circuits. *SIAM J. Comput.*, 28(1):311–324, 1998.

[Li24] Z. Li. Symmetric exponential time requires near-maximum circuit size: Simplified, truly uniform. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024*, pages 2000–2007. ACM, 2024.

[PF79] N. Pippenger and M. J. Fischer. Relations among complexity measures. *Journal of the ACM (JACM)*, 26(2):361–381, 1979.

[PWW88] J. Paris, A. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *J. Symb. Log.*, 53(4):1235–1244, 1988.

[RS98] A. Russell and R. Sundaram. Symmetric alternation captures BPP. *Comput. Complex.*, 7(2):152–162, 1998.

[San09] R. Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

[Sch89] U. Schöning. Probabilistic complexity classes and lowness. *J. Comput. Syst. Sci.*, 39(1):84–100, 1989.

[Val76] L. G. Valiant. Relative complexity of checking and evaluating. *Inf. Process. Lett.*, 5(1):20–23, 1976.

[Wag86] K. Wagner. The complexity of combinatorial problems with succinct input representations. *Acta Inform.*, 23:325–356, 1986.

[Zac82] S. Zachos. Robustness of probabilistic computational complexity classes under definitional perturbations. *Inf. Control.*, 54(3):143–154, 1982.

[Zac88] S. Zachos. Probabilistic quantifiers and games. *J. Comput. Syst. Sci.*, 36:433–451, 1988.

[ZF87] S. Zachos and M. Fürer. Probabilistic quantifiers vs. distrustful adversaries. In *Foundations of Software Technology and Theoretical Computer Science*, volume 287, pages 443–455, 1987.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il