Electronic Colloquium on Computational Complexity, Report No. 352 (2026)

A note on input-oblivious unambiguous hierarchies

Edward A. Hirsch*

November 21, 2026

Abstract

Unambiguous hierarchies are defined similarly to the polynomial hierarchy; however, all witnesses must be unique. Their versions differ in the mode of using oracles. We consider input-oblivious unambiguous hierarchies built from the class **OUP**, in which all positive inputs of the same length have the same unique witness: the hierarchy **OUH** with language oracles and the hierarchy $\mathbf{OUH}_\bullet$ with loose access to promise oracles, which is the input-oblivious version of the loose unambiguous hierarchy of Hirsch and Sokolov (IPL, 2015). We show that languages in $\mathbf{OUH}_\bullet$ can be decided in deterministic polynomial time with an **ONP** oracle, and for languages in **OUH** oracle queries can be made nonadaptive.

Moreover, languages in **OUH** can be decided in deterministic polynomial time with a tally **NP** oracle, and languages in $\mathbf{OUH}_\bullet$ can be decided in **P/OptP**, that is, a subclass of **P/poly** where the advice is generated by a function that maximizes outputs of a nondeterministic machine on a unary input. We show that **P/OptP** can be characterized in terms of **OGTD** and **OUGTD** generator–tester–decoder classes recently considered by Hirsch and Volkovich (2026) and, in particular, is contained in their "ultimate" Karp–Lipton collapse class **OUM**.

We also observe several other facts related to **OUP**: (1) the hierarchy defined by input-oblivious unambiguous dot-operators collapses, (2) in **OUP** computations, a single loose query to **prOUP** can be replaced by a query to **OUP**, (3) **SOUP** = **OUP**∩**co-OUP**, where **SOUP** is the same-witness class recently considered by Hirsch, Volkovich, and Williams (2026).

AI usage statement. All notions and statements in this paper were formulated by the author, who also shared his proof sketches and discussed the proofs with the AI tools GPT6-Astra and Claude Opus 5.5. After that Opus 5.5 assisted with the literature search (the author admits that as a result he read the amount of *classical* literature like never before) and with drafting the text. The final version was written by the author in a long dialogue with Opus 5.5; in particular, the author verified and edited all parts written by the above tools and takes full responsibility for the content of the paper.

1 Introduction

Unambiguous hierarchies. Nondeterministic polynomial-time computations with at most one accepting path constitute the class **UP** defined[1] by Valiant. Unambiguous hierarchies [LR94, NR98] are defined similarly to the polynomial hierarchy; however, all witnesses must be unique. These hierarchies have subtle differences in the mode of using oracles. The hierarchy $\mathbf{UH} = \mathbf{UP} \cup \mathbf{UP}^{\mathbf{UP}} \cup \mathbf{UP}^{\mathbf{UP}^{\mathbf{UP}}} \cup \ldots$ [NR98] uses oracle access to languages, and the computation needs

*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.

[1]Here and in what follows, for the definitions of classical complexity classes, we refer the reader to the Complexity Zoo (complexityzoo.net) or to [HO02, Appendix A], or (for the most basic ones) to [AB09].

ISSN 1433-8092

to be unambiguous only for the correct oracle. The hierarchy $\mathcal{UH}$ [LR94, NR98] uses *guarded* oracle access to promise problems [CHV93]: queries outside the promise set are prohibited (Grollmann and Selman [GS88] called such reductions *smart*). The unambiguous alternation hierarchy **AUPH** [LR94] is defined by unambiguous alternating machines with a bounded number of alternations, via single-query oracle access, or by alternating unambiguous dot-operators [NR98] (Niedermeier and Rossmanith attribute the equivalence to Hemaspaandra). These hierarchies satisfy $\mathbf{AUPH} \subseteq \mathbf{UH} \subseteq \mathcal{UH}$ levelwise and $\mathcal{UH} \subseteq \mathbf{SPP}$ [LR94]; moreover, **UH** and $\mathcal{UH}$ are contained in the unambiguous alternating polynomial-time class **UAP** [CGRS04], which is contained in **SPP** [NR98]. Hemaspaandra and Rothe [HR97] related their structure to the existence of sparse Turing-complete sets for **UP**, and Spakowski and Tripathi [ST07, ST09] studied their relativized structure.

Hirsch and Sokolov [HS15] relaxed the definition of oracle access and considered the *loose* unambiguous hierarchy $\mathbf{prUH}_\bullet$: queries outside the promise set are allowed, but the computation must be correct (and unambiguous) for *all* possible answers to such queries. This mode of access to promise problems was introduced for deterministic machines by Cai, Hemachandra, and Vyskoč [CHV93] under the name "arbitrary failure access"; it was used by Chakaravarthy and Roy [CR11] for deterministic computations querying **prMA** and **prAM**, and it is also meant when the Valiant–Vazirani theorem (see [AB09, Theorem 17.18]) is formulated as $\mathbf{NP} \subseteq \mathbf{RP}^{\mathbf{prUP}}$. The class $\mathbf{UH}_\bullet$ (the class of languages in $\mathbf{prUH}_\bullet$) contains **UH** and $\mathcal{UH}$ and is still contained in **PH**, levelwise. Hirsch and Sokolov strengthened the first part of Toda's theorem $\mathbf{PH} \subseteq \mathbf{BP} \cdot \oplus\mathbf{P} \subseteq \mathbf{P}^{\mathbf{PP}}$ (see [AB09, Section 17.4]) to $\mathbf{PH} = \mathbf{BP} \cdot \mathbf{prUH}_\bullet$. They asked whether $\mathbf{prUH}_\bullet \subseteq \mathbf{prSPP}$ [HS15, Section 4], which has been very recently answered affirmatively by Miloschewsky, Podder, and Rudolph [MPR26, Lemma 3.6].

Input-oblivious computations. In an input-oblivious computation, the witness may depend on the length of the input but not on the input itself. Chakaravarthy and Roy [CR06] defined the input-oblivious version $\mathbf{O}_2^{\mathbf{P}}$ of $\mathbf{S}_2^{\mathbf{P}}$, Fortnow, Santhanam, and Williams [FSW09] defined the input-oblivious versions **ONP** and **OMA** of **NP** and **MA**, and Goldreich and Meir [GM15] studied input-oblivious proof systems as a uniform perspective on $\mathbf{P/poly}$. These classes are contained in $\mathbf{P/poly}$; for example, $\mathbf{NP} \subseteq \mathbf{P/poly}$ if and only if $\mathbf{NP} = \mathbf{ONP}$ [FSW09, Proposition 5]. Hirsch, Volkovich, and Williams [HVW26] recently introduced the class **OUP** of input-oblivious unambiguous computations, where all positive inputs of the same length have the same unique witness, and the class **OMIN**, which equals the input-oblivious version $\mathbf{OL}_2^{\mathbf{P}}$ [HV26a] of the class $\mathbf{L}_2^{\mathbf{P}}$ of Korten and Pitassi [KP24]. They proved $\mathbf{OUP} \subseteq \mathbf{OMIN}$ and considered the oblivious hierarchy $\mathbf{OH} = \mathbf{ONP} \cup \mathbf{ONP}^{\mathbf{ONP}} \cup \ldots \subseteq \mathbf{O}_2^{\mathbf{P}}$. It is natural to consider input-oblivious unambiguous hierarchies as well (and this is what we do in this paper): the hierarchy $\mathbf{OUH} = \mathbf{OUP} \cup \mathbf{OUP}^{\mathbf{OUP}} \cup \ldots$ with language oracles and the loose hierarchy $\mathbf{prOUH}_\bullet$; by $\mathbf{OUH}_\bullet$ we denote the class of languages in $\mathbf{prOUH}_\bullet$.

Several-stage computations. Hirsch and Volkovich [HV26b] introduced generator–tester–decoder computations. On inputs of length n, a *generator*, which is a polynomial-time transducer with an oracle, computes an advice string from 1^n; a *tester* certifies the advice without seeing the input; and a polynomial-time *decoder* decides the input using the advice. The class $\mathbf{OGTD}[\mathcal{A}, \mathcal{B}, \mathcal{C}]$ consists of the languages decided in this way by a generator from $\mathcal{A}$, a tester from $\mathcal{B}$, and a decoder from $\mathcal{C}$; in $\mathbf{OUGTD}[\mathcal{A}, \mathcal{B}, \mathcal{C}]$ the tester accepts exactly one advice string for each n (see Section 2.5). An important particular case, denoted $\mathbf{OGTD}[\mathcal{X}]$, is when the generator is in $\mathbf{FP}^{\mathcal{X}}$, the decoder is in **P**, and no tester is required. Clearly, $\mathbf{OGTD}[\mathcal{X}] \subseteq \mathbf{P}^{\mathcal{X}}$, since a polynomial-time machine with an oracle from $\mathcal{X}$ can compute the advice and then run the decoder.

Two-stage computations of this kind have been studied for decades. Karp and Lipton defined advice classes $\mathcal{C}/f$ that allowed computational models of class $\mathcal{C}$ to use the same trusted (and potentially uncomputable) advice string of length $f(n)$ on all inputs of length n. In particular, $\mathbf{P/poly} = \mathbf{OGTD}[\mathrm{poly}, *, \mathbf{P}]$, where poly stands for arbitrary polynomially bounded advice (a potentially uncomputable function $1^n \mapsto a_n$) and $*$ means there is no tester. Ko and Schöning [KS85], Köbler and Thierauf [KT94], and Gavaldà [Gav95] bounded the complexity of computing correct advice, that is, the complexity of the generator, and Kämper's nonuniform proof systems [Käm91] delimit the complexity of the sets of correct advice, that is, the sets accepted by a tester; for the usual classes $\mathcal{B}$ and $\mathcal{C}$, $\mathbf{OGTD}[\mathrm{poly}, \mathcal{B}, \mathcal{C}]$ is his class $\mathcal{C}$-$\mathcal{B}$-**PSL**. Köbler and Thierauf used the notation $\mathcal{C}/\mathcal{F}$ to require the advice strings to be computed by functions $1^n \mapsto a_n$ in a complexity class $\mathcal{F}$. In their notation, $\mathbf{OGTD}[A] = \mathbf{P/FP}^A$ for a language A. In fact, $\mathbf{OGTD}[\mathbf{NP}] = \mathbf{P/OptP}$ (see Corollary 3.2, which also follows from Krentel's theorem [Kre88, Theorem 3.1]); here **OptP** is Krentel's class of optimization functions.

Aaronson [Aar07] introduced the class **YP** of languages decidable in polynomial time with untrusted advice, and Aaronson and Drucker [AD14] introduced its variant $\mathbf{YP}^*$, where the advice is tested in polynomial time without access to the input; $\mathbf{YP}^*$ is $\mathbf{OGTD}[\mathrm{poly}, \mathbf{P}, \mathbf{P}]$, that is, Kämper's class **P-P-PSL** [Käm91]. Input-oblivious proof systems [FSW09, GM15] can be viewed as restricting **P/poly** to verifiable advice; in the class **OIP** of Goldreich and Meir [GM15], the interaction takes place on 1^n, and its result is subsequently used on the actual input. Buhrman, Fortnow, Hitchcock, and Loff [BFHL13, Theorem 4] showed that if $\mathbf{NP} \subseteq \mathbf{P/poly}$ and $\mathbf{PH} \subseteq \mathbf{P}^{\mathbf{NP}}$, then $\mathbf{PH} \subseteq \mathbf{P}/\alpha$ for an advice function α whose graph is in **co-NP**, that is, with unique advice certified by a **co-NP** tester. The main class of [HV26b] is $\mathbf{OM} = \mathbf{OGTD}[\mathbf{FP}^{\mathbf{sOprMA}}, \mathbf{co\text{-}ONP}, \mathbf{P}]$, where **sOprMA** is a restricted version of **prOMA** (still containing **ONP**). They also proved that a Karp–Lipton collapse holds down to its unique-advice subclass **OUM** (that is, $\mathbf{NP} \subseteq \mathbf{P/poly}$ implies $\mathbf{PH} = \mathbf{OUM}$), which is contained in three Karp–Lipton collapse classes that are pairwise incomparable as far as is known to date, $\mathbf{OUM} \subseteq \mathbf{OM} \subseteq \mathbf{OMIN} \cap \mathbf{P}^{\mathbf{sOprMA}} \cap \mathbf{ZPP}^{\mathbf{ONP}}$.

1.1 Our results

Input-oblivious unambiguous hierarchies. It is known that $\mathbf{OUP} \subseteq \mathbf{OMIN}$ and that **OMIN** is closed under polynomial-time Turing reductions [HVW26]; thus one can ask whether $\mathbf{OUH} \subseteq \mathbf{OMIN}$ and what is an upper bound on $\mathbf{OUH}_\bullet$. We answer the former question affirmatively; even more, we show that the languages of both hierarchies have polynomial-size circuits that can be constructed from 1^n with an **NP** oracle. For **OUH**, nonadaptive queries to a tally **NP** language suffice:

Theorem (Theorem 4.1)**.** $\mathbf{OUH} \subseteq \mathbf{P}^{\mathbf{TALLY} \cap \mathbf{NP}}$.

In particular, every language in **OUH** can be decided in polynomial time with nonadaptive queries to a language in **ONP**, i.e., $\mathbf{OUH} \subseteq \mathbf{P}_{\|}^{\mathbf{ONP}}$ (Corollary 4.2). For loose access to promise oracles, we construct an adaptive generator:

Theorem (Theorem 5.1)**.** $\mathbf{OUH}_\bullet \subseteq \mathbf{OGTD}[\mathbf{NP}]$.

For generators with an **NP** oracle, the advice can be made canonical, which gives a unique advice string certified by a **co-ONP** tester:

Lemma (Lemma 3.1 and Corollary 3.2)**.** $\mathbf{P/OptP} = \mathbf{OGTD}[\mathbf{NP}] = \mathbf{OGTD}[\mathbf{ONP}] = \mathbf{OUGTD}[\mathbf{FP}^{\mathbf{ONP}}, \mathbf{co\text{-}ONP}, \mathbf{P}]$.

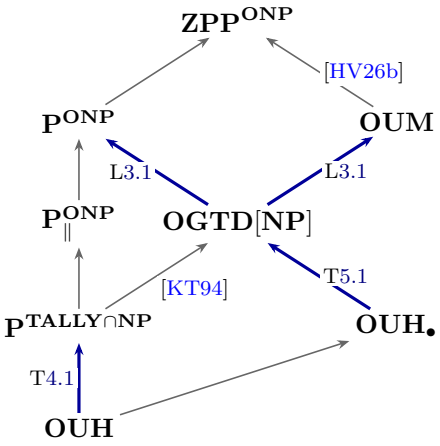


Figure 1: *Containments between the main classes considered in this paper; arrows point to larger classes. Thick arrows are proved in this paper; the other arrows are known or immediate. All these classes are contained in* $\mathbf{P/poly}$. *Moreover,* $\mathbf{OGTD[NP]} = \mathbf{OUGTD}[\mathbf{FP}^{\mathbf{ONP}}, \mathbf{co\text{-}ONP}, \mathbf{P}] = \mathbf{P/OptP}$ *by Lemma 3.1 and Corollary 3.2.*

It puts this class (as well as $\mathbf{OUH}_\bullet$) into both $\mathbf{P}^{\mathbf{ONP}}$ and $\mathbf{OUM}$. Consequently, every language in $\mathbf{OUH}_\bullet$ can be decided in polynomial time with adaptive queries to a language in $\mathbf{ONP}$: $\mathbf{OUH}_\bullet \subseteq \mathbf{P}^{\mathbf{ONP}}$ (Corollary 5.3).

See Fig. 1 for the containment diagram.

Dot-operators and single queries. One could try to define our hierarchies using dot-operators instead of oracles. Dot-operators were introduced by Wagner [Wag86] and studied further in [ZF87, Zac88, Sch89, BGM06]. The polynomial hierarchy can be defined using alternating existential and universal dot-operators (for example, $\mathbf{NP}^{\mathbf{NP}} = \exists \cdot \mathbf{co\text{-}NP}$), and so can $\mathbf{OH}$, since the proof of $\mathbf{ONP}^{\mathbf{ONP}} = \exists_{\mathbf{o}} \cdot \mathbf{co\text{-}ONP}$ [HVW26, Theorem 4.4] obviously relativizes; also the hierarchy $\mathbf{AUPH}$ can be equivalently defined using alternating unique dot-operators [LR94, NR98]. In contrast, the hierarchy defined by alternating input-oblivious unambiguous dot-operators collapses to $\mathbf{OUP} \cup \mathbf{co\text{-}OUP}$ (Theorem 6.2 and Corollary 6.3). We also observe that a single loose query to a promise problem in $\mathbf{prOUP}$ can be replaced by a query to a language in $\mathbf{OUP}$, that is, $\mathbf{OUP}^{\mathbf{prOUP}[1]} = \mathbf{OUP}^{\mathbf{OUP}[1]}$, and similarly $\mathbf{UP}^{\mathbf{prUP}[1]} = \mathbf{UP}^{\mathbf{UP}[1]}$ (Theorem 6.5). This is in contrast with $\mathbf{RP}^{\mathbf{prRP}[1]} = \mathbf{BPP}$ (by the proof of [BF99, Theorem 1]), whereas $\mathbf{P} = \mathbf{RP} \neq \mathbf{BPP}$ relative to an oracle [BF99, Theorem 3], and with $\mathbf{ONP}^{\mathbf{prONP}[1]} = \mathbf{O}_2^{\mathbf{p}}$ [HVW26, Theorem 4.7], whereas $\mathbf{ONP}^{\mathbf{ONP}[1]} = \mathbf{ONP}^{\mathbf{ONP}}$ is not known to equal $\mathbf{O}_2^{\mathbf{p}}$. Finally, the same-witness class $\mathbf{SOUP}$ of [HVW26] equals $\mathbf{OUP}\cap\mathbf{co\text{-}OUP}$; hence $\mathbf{OUP}^{\mathbf{SOUP}} = \mathbf{OUP}$ and $\mathbf{SOUP}^{\mathbf{SOUP}} = \mathbf{SOUP}$ (Proposition 6.6).

1.2 Our methods

Canonical advice. To better locate the class $\mathbf{OGTD[NP]}$, where we put our hierarchies, we discuss it in Section 3. Our Lemma 3.1 is based on Krentel's observation [Kre88, Theorem 3.1] that the correct sequence of answers of an $\mathbf{NP}$ oracle is the lexicographically largest sequence in which all positive answers are certified; with some effort, the lemma can also be inferred from his results. Buhrman, Fortnow, Hitchcock, and Loff [BFHL13, proof of Theorem 4] obtained unique advice with a $\mathbf{co\text{-}NP}$ graph in a similar way, appending to the run of an $\mathbf{FP}^{\mathbf{NP}}$ generator the lexicographically smallest witnesses for its positive answers. We encode a run of the generator

on 1^n by its sequence of oracle answers followed by certificates for the positive answers, and take the lexicographically largest feasible encoding a_n as the advice. This way, a_n is a common witness for all queries of the prefix search that finds it, so it is found in $\mathbf{FP}^{\mathbf{ONP}}$; it also refutes every other feasible encoding, so it is the only string accepted by a **co-ONP** tester. A minimum of a linear order is used as a common refutation in the same way in [HVW26, HV26b].

Language oracles. For **OUH** (Theorem 4.1) we show that $\mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}}$ is closed under **OUP** oracle computations. We use Mahaney's census technique (see [HO02, Section 1.2]), which was also used for computing advice from 1^n in [KT94] and for unambiguous computations with sparse oracles in [HR97]: knowing the number of strings of a tally **NP** oracle up to the relevant length, an **NP** machine can certify all its answers. In this way, Köbler and Thierauf [KT94, Theorem 3.2] proved $\mathbf{NP}^{\mathbf{SPARSE}\cap\mathbf{NP}} \subseteq \mathbf{NP}/\mathbf{OptP}[O(\log n)]$, where the advice is this number of strings. For a tally language in $\mathbf{NP}^{\mathbf{TALLY}\cap\mathbf{NP}}$, this number can be obtained by asking the tally oracle itself, and the remaining question is a query to another tally language in **NP**; hence every tally language in $\mathbf{NP}^{\mathbf{TALLY}\cap\mathbf{NP}}$ belongs to $\mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}}$ (Lemma 2.8; with some effort, it can also be inferred from the results of [KT94]). Our new ingredient is the obliviousness of the witness of the outer **OUP** computation: since all positive inputs of length n share the unique witness w_n, its bits are answers to oracle **NP** questions that do not depend on the input, and these questions can be answered by a tally language in $\mathbf{NP}^{\mathbf{TALLY}\cap\mathbf{NP}}$.

Loose access. For promise oracles, the census technique does not work because of unclear attribution of out-of-promise queries, and a search for an accepting witness may find a witness accepted on an input outside the promise. Our generator for $\mathbf{OUH}_\bullet$ (Theorem 5.1) follows the strategy of the proof by Hartmanis and Hemachandra [HH90, Theorem 2.1] that "robustly categorical" machines (nondeterministic machines with at most one accepting path for every oracle) accept, relative to every oracle A, languages decidable in polynomial time with queries to A and to an **NP** oracle; the result was extended to "robustly unambiguous" alternating machines (machines with a bounded number of alternations in which, for every oracle, every existential configuration has at most one accepting successor and every universal configuration has at most one rejecting successor) in [ST07, Theorem 5.1]. Their simulation repeatedly finds, with an **NP** oracle, an accepting path consistent with the answers of A learned so far and asks A all queries of this path; it stops when the path is correct or no such path exists. Since two accepting paths for different oracles must conflict on some query, each new path conflicts with every previous path on a query on which the latter was wrong; hence the number of rounds is bounded by the number of queries on a path. Spakowski and Tripathi [ST07] apply this argument level by level: by induction on the number k of levels, they simulate the lower levels in polynomial time with queries to A and to a $\mathbf{\Sigma}^{\mathbf{P}}_{\mathbf{k-1}}$ oracle and apply the argument of [HH90] to the top level, which gives the same with a $\mathbf{\Sigma}^{\mathbf{P}}_{\mathbf{k}}$ oracle.

Our setting differs in two respects. First, our machines are not robust: they are unambiguous only on queries in the promise set and only with oracles consistent with the promise problem one level below. Second, our generator has no access to the promise oracles (it has only an **NP** oracle), so the answers to promised queries are not available for checking a path. We therefore replace the learned answers by sets of candidate oblivious witnesses for every level and every query length. Relative to these sets, both answers of the oracles are certified: a positive answer by an accepting witness, and a negative answer by the rejection of all candidates. The generator looks for an input with two certified runs that give different outputs. By loose access, such a disagreement contains an incorrect answer to a query in the promise set; the uniqueness and the obliviousness of the witnesses on such queries take the place of the conflicts between accepting paths and let us trace it down the tower of oracles to a missing correct witness, which is

added. Since the witnesses are oblivious, the number of rounds is bounded by the number of pairs (level, query length), and the sizes of the sets are bounded by descending induction on the level. Consequently, one **NP** oracle serves all levels at once, without the level-by-level induction of [ST07], and the generator produces one advice string for all inputs of a given length instead of deciding a given input.

Dot-operators and single queries. The collapse in Corollary 6.3 uses the idea, applied in [HVW26] to prove that **SOUP** is closed under polynomial-time Turing reductions, to test a candidate oblivious witness on a fixed input: since the outer witness is unique, the inner **co-OUP** predicate is false on one of two fixed pairs, so the oblivious witness refuting it can be identified without the actual input, which gives $\exists_{\mathbf{ou}}\cdot\mathbf{co\text{-}OUP} = \mathbf{OUP} \cup \mathbf{co\text{-}OUP}$ (Theorem 6.2). In contrast, the non-oblivious dot-operator-based unambiguous hierarchy **AUPH** is infinite relative to an oracle [ST07].

To show that single queries to promise problems can be converted to language queries, we combine the "input-tagging" argument (a query q asked on input x is replaced by the pair (x, q), so that the oracle can check that q is indeed asked on x) of Cai, Hemachandra, and Vyskoč [CHV93, Proposition 2.6], who showed that for deterministic machines, guarded nonadaptive access to **prUP** is not more powerful than access to **UP**, with the observation that a single loose query of an unambiguous computation is guarded whenever its answer matters: otherwise, answering it in two ways would violate either correctness or independence of the computation result from the oracle's answer.

Organization of the paper. Section 2 contains definitions, notation, conventions, and known facts. Section 3 contains preparations concerning generators with an **NP** oracle. The next sections contain the main results of the paper: Sections 4 and 5 treat **OUH** and $\mathbf{OUH}_\bullet$, respectively, and Section 6 considers the first levels of the hierarchy (the hierarchy defined by dot-operators, single queries, and the class $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$).

2 Preliminaries

For rigorous definitions of standard notions of complexity theory, we refer to [AB09].

2.1 Conventions, injective pairing, promise problems, and oracle access

All machines are polynomial-time unless stated otherwise; n denotes the length of the input. In definitions based on polynomial-time predicates, all witnesses, random strings, and advice strings for inputs of length n have prescribed lengths given by polynomials in n, and quantifiers range over strings of these lengths. Pairs and tuples are encoded in a length-regular way: if the lengths of the components are determined by n, the length of the encoding determines n and the lengths of the components. A language T is *tally* if $T \subseteq \{1\}^*$; **TALLY** is the class of tally languages. We fix an injective pairing π of a constant number of natural numbers such that π is polynomially bounded in its arguments and π and its inverse are computable in time polynomial in the unary lengths; we use it to encode tuples of numbers by tally strings. By $<_{\ell ex}$ we denote the lexicographic order on strings of the same length.

A *promise problem* (introduced by Even, Selman, and Yacobi; see [AB09, Section 23.4]) is a pair $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$ of disjoint sets of strings; a language L is identified with $(L, \overline{L})$. We write $\Pi(x) = 1$ for $x \in \Pi_{\text{YES}}$ and $\Pi(x) = 0$ for $x \in \Pi_{\text{NO}}$. A language O is *consistent* with Π if $\Pi_{\text{YES}} \subseteq O$ and $O \cap \Pi_{\text{NO}} = \emptyset$. For a class $\mathcal{C}$ of languages defined by machines, $\mathbf{pr}\mathcal{C}$ is the corresponding class of promise problems: the correctness requirements, as well as the semantic requirements

such as the uniqueness of witnesses, are imposed only on the promise set $\Pi_{\text{YES}} \cup \Pi_{\text{NO}}$ [HS15]. For a class $\mathcal{D}$ of promise problems, the *class of languages in* $\mathcal{D}$ consists of the languages L such that $(L, \overline{L}) \in \mathcal{D}$.

Our default oracle access mode to promise problems is *loose*: queries outside the promise set are allowed and may be answered arbitrarily, and the computation must satisfy all requirements of its class for all such answers. These answers need not be consistent: the same query may receive different answers on different computation paths. We assume that a computation path never repeats a query. Then, for deterministic oracle machines, loose access is equivalent to the requirement of correctness for every language consistent with the oracle [CHV93, Proposition 5.9]. For unambiguous oracle machines, loose access is equivalent to the definition of [HS15, Definition 4]: the computation must be correct for every consistent language, and its accepting witness, if any, must be the same for all consistent languages. (Indeed, the answers received on a single computation path are given by some consistent language; and if two consistent languages produced different accepting witnesses, then combining their answers on the two corresponding paths would produce two accepting paths.) We use this formulation below.

Accordingly, $\mathbf{P}^{\Pi}$ is the class of languages decided by deterministic polynomial-time oracle machines M such that M^O gives the correct answer for every language O consistent with Π; an $\mathbf{FP}^{\Pi}$ computation may have several legal outputs (for different consistent languages O), and all of them must be correct but do not need to be the same string. The subscript $\|$ (as in $\mathbf{P}^{\Pi}_{\|}$ or $\mathbf{FP}^{\Pi}_{\|}$) means that all queries are computed before any answer is received. For a class $\mathcal{D}$ of promise problems, $\mathbf{P}^{\mathcal{D}} = \bigcup_{\Pi \in \mathcal{D}} \mathbf{P}^{\Pi}$, and similarly for other oracle classes; $\mathcal{C}^{\mathcal{D}[1]}$ means that at most one query is made on every computation path.

2.2 Unambiguous and input-oblivious classes

We define the main classes interesting to us in their general promise relativized versions.

Definition 2.1 (**UP**, **OUP** [HVW26], **ONP** [FSW09])**.** *Let* Π *be a promise problem. A promise problem* Γ *belongs to* $\mathbf{prUP}^{\Pi}$ *if there is a deterministic polynomial-time oracle machine* V *such that for every* $x \in \Gamma_{\text{YES}} \cup \Gamma_{\text{NO}}$*, the set*

$$W(x) = \{w : V^O(x, w) = 1\}$$

is the same for all languages O *consistent with* Π*,* $|W(x)| = 1$ *for* $x \in \Gamma_{\text{YES}}$*, and* $W(x) = \emptyset$ *for* $x \in \Gamma_{\text{NO}}$*. If, moreover, there are strings* w_n *such that* $W(x) = \{w_{|x|}\}$ *for every* $x \in \Gamma_{\text{YES}}$*, then* $\Gamma \in \mathbf{prOUP}^{\Pi}$*, and* w_n *are called* oblivious unambiguous witnesses*. The problem* Γ *belongs to* $\mathbf{prONP}^{\Pi}$ *if there are a deterministic polynomial-time oracle machine* V *and strings* w_n *such that for every language* O *consistent with* Π*,* $V^O(x, w_{|x|}) = 1$ *for every* $x \in \Gamma_{\text{YES}}$*, and* $V^O(x, w) = 0$ *for every* $x \in \Gamma_{\text{NO}}$ *and every* w*. The classes* $\mathbf{UP}^{\Pi}$*,* $\mathbf{OUP}^{\Pi}$*, and* $\mathbf{ONP}^{\Pi}$ *consist of the languages in these classes; for a language oracle* Π *(in particular, without an oracle) we obtain the usual definitions.*

2.3 Input-oblivious unambiguous hierarchies

Definition 2.2 (input-oblivious unambiguous hierarchies)**.** *Define* $\mathbf{OU\Sigma}^{\mathbf{P}}_{\mathbf{0}} = \mathbf{P}$*,* $\mathbf{prOU\Sigma}_{\bullet 0} = \mathbf{prP}$*, and, for* $k \geq 0$*,* $\mathbf{OU\Sigma}^{\mathbf{P}}_{\mathbf{k+1}} = \mathbf{OUP}^{\mathbf{OU\Sigma}^{\mathbf{P}}_{\mathbf{k}}}$ *and* $\mathbf{prOU\Sigma}_{\bullet k+1} = \mathbf{prOUP}^{\mathbf{prOU\Sigma}_{\bullet k}}$ *(recall that the access to promise oracles is loose). Let* $\mathbf{OUH} = \bigcup_k \mathbf{OU\Sigma}^{\mathbf{P}}_{\mathbf{k}}$ *and* $\mathbf{prOUH}_{\bullet} = \bigcup_k \mathbf{prOU\Sigma}_{\bullet k}$*. The classes* $\mathbf{OU\Sigma}_{\bullet k}$ *and* $\mathbf{OUH}_{\bullet}$ *consist of the languages in the respective classes of promise problems.*

Note that $\mathbf{P}^{\mathbf{prOUH}_\bullet} = \mathbf{OUH}_\bullet$: a deterministic polynomial-time machine with loose access to $\Pi \in \mathbf{prOU\Sigma}_{\bullet k}$ is an $\mathbf{OUP}^{\Pi}$ verifier with the empty witness, so its language belongs to $\mathbf{OU\Sigma}_{\bullet k+1}$; the converse inclusion needs one query.

2.4 Dot-operators

Definition 2.3 ([HVW26, Definition 2.8]). *Let $\mathcal{C}$ be a class of promise problems. A language L belongs to $\exists_{\mathbf{ou}\bullet}\mathcal{C}$ if there are $\Pi \in \mathcal{C}$, a polynomial p, and strings $w_n \in \{0,1\}^{p(n)}$ such that for every n and every $x \in \{0,1\}^n$,*

$$\begin{aligned} x \in L &\;\Rightarrow\; (x, w_n) \in \Pi_{\text{YES}} \text{ and } (x, w) \in \Pi_{\text{NO}} \text{ for all } w \neq w_n, \\ x \notin L &\;\Rightarrow\; (x, w) \in \Pi_{\text{NO}} \text{ for all } w. \end{aligned}$$

If the witness w_n may depend on x, we obtain the unique existential operator $\exists_{\mathbf{u}\bullet}$ (denoted $\exists!$ in [LR94, NR98, ST09]). If, instead, we drop the condition on the strings $w \neq w_n$, we obtain the oblivious existential operator $\exists_{\mathbf{o}\bullet}$ [HVW26], and if, moreover, the witness may depend on x, the existential operator $\exists_\bullet$. The dual operator is $\forall_{\mathbf{ou}}\cdot\mathcal{C} = \mathbf{co}\text{-}\exists_{\mathbf{ou}}\cdot\mathbf{co}\text{-}\mathcal{C}$, where $\mathbf{co}\text{-}$ exchanges Π_{YES} and Π_{NO}. We allow the witness length to be identically zero (identifying (x,ε) with x); thus the languages in $\mathcal{C}$ belong to $\exists_{\mathbf{ou}}\cdot\mathcal{C}$.

For $\exists_{\mathbf{ou}\bullet}$ and $\exists_{\mathbf{u}\bullet}$, all pairs (x,w) with $|w| = p(|x|)$ belong to the promise set of Π. Note that $\mathbf{OUP} = \exists_{\mathbf{ou}}\cdot\mathbf{P}$ and, for every oracle A, $\mathbf{OUP}^A = \exists_{\mathbf{ou}}\cdot\mathbf{P}^A$ (cf. [HVW26, Corollary 4.2]). The same holds for loose access to a promise problem Π: $\mathbf{OUP}^{\Pi} = \exists_{\mathbf{ou}}\cdot\mathbf{P}^{\Pi}$. Indeed, by Definition 2.1, the set of witnesses that an $\mathbf{OUP}^{\Pi}$ verifier accepts on an input x is the same for all languages consistent with Π, so the accepted pairs (x,w) form a language in $\mathbf{P}^{\Pi}$; the converse is immediate.

2.5 Generator–tester–decoder computations

Definition 2.4 (**OGTD**, **OUGTD** [HV26b]). *Let $\mathcal{X}$ be a class of promise problems. A* generator *in $\mathbf{FP}^{\mathcal{X}}$ is a deterministic polynomial-time oracle transducer G with an oracle $\Pi \in \mathcal{X}$ that outputs a string of length $p(n)$ on input 1^n, for a polynomial p. Its set of* legal outputs *is*

$$\mathrm{Out}_G(n) = \{G^O(1^n) : O \text{ is a language consistent with } \Pi\}.$$

Let $\mathcal{B}$ and $\mathcal{C}$ be classes of languages. A language L belongs to $\mathbf{OGTD}[\mathbf{FP}^{\mathcal{X}}, \mathcal{B}, \mathcal{C}]$ if there are a generator $G \in \mathbf{FP}^{\mathcal{X}}$, a tester *$T \in \mathcal{B}$ (a language of pairs $(1^n, a)$), and a* decoder *$D \in \mathcal{C}$ (a language of pairs (x,a)) such that, for every n,*

$$\mathrm{Out}_G(n) \subseteq \mathrm{Acc}_T(n) \subseteq \mathrm{Good}_D(n),$$

where $\mathrm{Acc}_T(n) = \{a : (1^n, a) \in T\}$ and $\mathrm{Good}_D(n) = \{a \in \{0,1\}^{p(n)} : \forall x \in \{0,1\}^n\ D(x,a) = L(x)\}$. The language L belongs to $\mathbf{OUGTD}[\mathbf{FP}^{\mathcal{X}}, \mathcal{B}, \mathcal{C}]$ if, moreover, $|\mathrm{Acc}_T(n)| = 1$ for every n. The symbol $*$ in place of $\mathcal{B}$ means that no tester is required, that is, only $\mathrm{Out}_G(n) \subseteq \mathrm{Good}_D(n)$ is required. Finally, $\mathbf{OGTD}[\mathcal{X}] = \mathbf{OGTD}[\mathbf{FP}^{\mathcal{X}}, *, \mathbf{P}]$.

Proposition 2.5 ([HV26b, Proposition 2.11]). *Let Π be a promise problem. Then $\mathbf{OGTD}[\Pi] \subseteq \mathbf{P}^{\Pi} \cap \mathbf{P/poly}$.*

We also use the notation of [KT94] for classes with bounded advice functions: for a class $\mathcal{C}$ of languages and a class $\mathcal{F}$ of advice functions, $\mathcal{C}/\mathcal{F}$ is the class of languages L such that $x \in L \Leftrightarrow (x, h(1^{|x|})) \in B$ for some $B \in \mathcal{C}$ and $h \in \mathcal{F}$.

Definition 2.6 (**sOprMA**, **OUM** [HV26b])**.** *A promise problem* Π *belongs to* **sOprMA** *if there are a polynomial-time predicate* $V(x,y,r)$ *and strings* y_n *such that for every* $x \in \{0,1\}^n$*: if* $x \in \Pi_{\text{YES}}$*, then* $V(x,y_n,r) = 1$ *for all* r*; if* $x \in \Pi_{\text{NO}}$*, then* $\Pr_r[V(x,y,r) = 1] < 1/4$ *for all* y*; and, for all* y *and* r*,* $V(x,y,r) = 1$ *implies* $V(x,y_n,r) = 1$*. Define* $\mathbf{OUM} = \mathbf{OUGTD}[\mathbf{FP}^{\mathbf{sOprMA}}, \mathbf{co\text{-}ONP}, \mathbf{P}]$.

Every language in **ONP** belongs to **sOprMA**: take $V(x,y,r) = V'(x,y)$ for an **ONP** verifier V' [HV26b]. Hence $\mathbf{FP}^{\mathbf{ONP}}$ generators are $\mathbf{FP}^{\mathbf{sOprMA}}$ generators.

2.6 Useful facts about computations with tally NP oracles

Queries to a tally oracle can be parallelized, because all of them can be computed during a preparation stage, as the (polynomial) range of all possible unary queries is known (cf., e.g., [KT94]). Since tally **NP** languages are in **ONP** (cf. [FSW09, GM15]), one can formulate the following useful fact.

Proposition 2.7. $\mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}} = \mathbf{P}_{\|}^{\mathbf{TALLY}\cap\mathbf{NP}} \subseteq \mathbf{P}_{\|}^{\mathbf{ONP}}$.

Lemma 2.8. *For a set* $C \in \mathbf{TALLY} \cap \mathbf{NP}$*, any tally set in* $\mathbf{NP}^C$ *belongs to* $\mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}}$.

Proof. Let $U \in \mathbf{NP}^C$ be a tally language, accepted by a nondeterministic polynomial-time oracle machine N with the oracle C. If U were in **NP**, our deterministic machine would just query U. However, the machine N uses the oracle C. We will get rid of this oracle by replacing it by an additional input to the **NP** machine (so our deterministic machine will query a different tally language D). We can compute the relevant portion of the truth table of C deterministically using the oracle C. However, we cannot use this truth table as the additional input, because it is too large to embed into a tally language. Instead, this additional input will be a much shorter "checksum": the number of strings of relevant lengths in C (which we can also compute using the oracle C).

Let $h(m)$ be the running time bound of N on input 1^m; it will bound the length of any query of N on this input. Let C_m contain all the strings in C of length at most $h(m)$, and let $c_m = |C_m|$. Knowing c_m, an **NP** machine can guess C_m (that is, c_m numbers $k \le h(m)$) together with **NP** witnesses for the elements of C_m; it can then deterministically verify these witnesses to make sure the guessed strings are indeed in C. After that this machine can answer queries to C itself.

Formally, consider the tally language D of all strings $1^{\pi(m,c)}$ (recall that π denotes an injective pairing) for which there are a c-size set S of numbers $\le h(m)$, correct **NP** witnesses of $1^k \in C$ for each $k \in S$, and an accepting computation of N on input 1^m in which the queries 1^k for $k \in S$ are answered positively and all other queries are answered negatively. Clearly, $D \in \mathbf{NP}$. For $c = c_m$ the only such set S is $\{k \in \mathbb{N} : 1^k \in C_m\}$: all elements of S must have correct witnesses, and the checksum makes sure all elements of C_m are listed. Hence the answers given to N are those of C, so $1^{\pi(m,c_m)} \in D$ if and only if $1^m \in U$. Thus, to decide whether $1^m \in U$, a deterministic polynomial-time machine asks the oracle C about 1^k for all $k \le h(m)$, which gives c_m, and then asks the oracle D about $1^{\pi(m,c_m)}$. The tally oracles C and D can be combined into a single oracle in $\mathbf{TALLY} \cap \mathbf{NP}$ using π. □

3 Generators with an NP oracle

Our special focus is on $\mathbf{OGTD}[\mathbf{NP}]$, which, as we show below (Corollary 3.2), is the class $\mathbf{P}/\mathbf{OptP}$ studied long ago by Köbler and Thierauf [KT94]. But first we show that in this class

the advice computed by the generator can be made "canonical" (that is, unambiguous) and testable in **co-ONP**.

We follow Krentel's proof [Kre88, Theorem 3.1]: the correct sequence of answers of an **NP** oracle is the lexicographically largest sequence in which all positive answers are certified. We need to add the witnesses for the positive answers to the sequence, to make a candidate checkable in polynomial time, and to use that on the input 1^n the largest feasible string a_n depends only on n, to make it a common witness for the queries of the generator and a common refutation of all other strings; this gives the **ONP** oracle and the **co-ONP** tester.

Lemma 3.1. $\mathbf{OGTD[NP]} = \mathbf{OGTD[ONP]} = \mathbf{OUGTD[FP^{ONP}, co\text{-}ONP, P]}$.

Proof. The containments $\mathbf{OUGTD[FP^{ONP}, co\text{-}ONP, P]} \subseteq \mathbf{OGTD[ONP]} \subseteq \mathbf{OGTD[NP]}$ are trivial. Let us replace the **NP** oracle of the generator by an **ONP** oracle and make the advice string unique by using the additional **co-ONP** tester.

Let $L \in \mathbf{OGTD}[A]$ with a generator $G \in \mathbf{FP}^A$ and a decoder $D \in \mathbf{P}$, where $A \in \mathbf{NP}$ has a polynomial-time verifier U. Our new generator will output something the old advice can be reconstructed from: the sequence a of oracle answers for the queries of $G(1^n)$, followed by the list of the corresponding witnesses for the positive answers (padded by zeroes to a fixed length sufficient in any possible case). It can be verified in polynomial time by simulating the run of G and verifying the witnesses with U. The only thing that cannot be verified is the negative answers. Therefore, we will require the lexicographically largest such string a_n: if we find it, then, in particular, all answers will be correct; also the largest string is guaranteed to be unique. The property that a formally correct candidate string is the largest can be tested in **co-ONP**: the correct answer a_n is the oblivious witness.

Let $R(1^n, a)$ be the polynomial-time deterministic procedure that checks that a is consistent with the run of $G(1^n)$ using the answers from a, and that all the witnesses are correct. We define our **co-ONP** tester as

$$T = \{(1^n, a) : R(1^n, a) \wedge \forall b >_{\ell ex} a \ \neg R(1^n, b)\},$$

thus it accepts solely $a = a_n$. Our new generator will use the oracle $Q = \{(1^n, c) : \exists a \geq_{\ell ex} c \ R(1^n, a)\} \in \mathbf{ONP}$ (where a_n plays again the role of the oblivious witness). It will find a_n by prefix search using this oracle. Namely, having found the first $j-1$ bits $b_1 \dots b_{j-1}$ of a_n, it asks whether $(1^n, c) \in Q$ for the string $c = b_1 \dots b_{j-1} 10 \dots 0$ of the same length as a_n. Since a_n is the largest string a with $R(1^n, a)$, we have $(1^n, c) \in Q$ if and only if $a_n \geq_{\ell ex} c$, that is, if and only if the j-th bit of a_n is 1.

The new decoder will reconstruct the old advice from a_n and run the old decoder. □

Corollary 3.2. $\mathbf{OGTD[NP]} = \mathbf{P/OptP}$.

Proof. This is a corollary of the proof rather than of the formulation of Lemma 3.1. We generate a_n as the maximum of all strings a with $R(1^n, a)$, which gives an **OptP** function computing a_n on input 1^n. In the other direction, every function in **OptP** can be computed in $\mathbf{FP^{NP}}$ by binary search [Kre88]. □

4 The oblivious unambiguous hierarchy

We show that the oblivious unambiguous hierarchy based on languages is contained in the second level of the oblivious hierarchy; moreover, its languages can be computed with nonadaptive queries to tally **NP** oracles.

Theorem 4.1. $\mathbf{OUH} \subseteq \mathbf{P^{TALLY \cap NP}}$.

Proof. It suffices to show that $\mathbf{OUP}^{\mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}}} \subseteq \mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}}$ and use it repeatedly.

Let $L \in \mathbf{OUP}^A$ with $A \in \mathbf{P}^T$ for some $T \in \mathbf{TALLY} \cap \mathbf{NP}$, and let V be a polynomial-time verifier for L with oblivious unambiguous witnesses w_n of length $p(n)$. Let V' be a version of V that simulates the oracle A itself using its $\mathbf{P}^T$ algorithm (so V' uses T as an oracle).

Since w_n is unique, we can compute it bit by bit (obliviously of the input) using queries to the following tally language

$$W = \{1^{\pi(n,i)} : 1 \le i \le p(n) \text{ and } V'^T(x,w) = 1 \text{ for some } x \in \{0,1\}^n \text{ and } w \text{ with } w_i = 1\}.$$

If $L \cap \{0,1\}^n \neq \emptyset$, then $w_n = W(1^{\pi(n,1)}) \dots W(1^{\pi(n,p(n))})$. If $L \cap \{0,1\}^n = \emptyset$, then this string is $0^{p(n)}$, which is as good as any other string in this case, so we can assume w.l.o.g. $w_n = 0^{p(n)}$. It is readily seen from that definition of W that $W \in \mathbf{NP}^T$. By Lemma 2.8, W can be computed in deterministic polynomial time with some oracle $O \in \mathbf{TALLY} \cap \mathbf{NP}$.

Now, L can be decided by first computing w_n and then simulating V'^T on the actual input with this witness. The resulting deterministic algorithm uses tally $\mathbf{NP}$ oracles T and O, which can be combined in a single oracle using π. □

Corollary 4.2. $\mathbf{OUH} \subseteq \mathbf{P}_{\|}^{\mathbf{ONP}}$.

Proof. By Theorem 4.1 and Proposition 2.7. □

5 The loose oblivious unambiguous hierarchy

The proof of Theorem 4.1 uses the fact that the oracle is a language: for a promise oracle, the search for an accepting witness may stumble upon a witness accepted on an input outside the promise. We will now overcome this difficulty at the expense of switching to the possibly larger class $\mathbf{OGTD}[\mathbf{NP}]$, whose generator makes adaptive $\mathbf{NP}$ (essentially, $\mathbf{ONP}$) queries (a nonadaptive one would give $\mathbf{P}^{\mathbf{TALLY}\cap\mathbf{NP}}$: its queries $q_{n,1}, q_{n,2}, \dots$ on 1^n do not depend on the oracle answers, so its oracle $A \in \mathbf{NP}$ can be replaced by the tally language $\{1^{\pi(n,j)} : q_{n,j} \in A\}$, which is in $\mathbf{NP}$), and accordingly our deterministic simulation with an $\mathbf{ONP}$ oracle becomes adaptive (compare Corollaries 4.2 and 5.3).

Theorem 5.1. $\mathbf{OUH}_\bullet \subseteq \mathbf{OGTD}[\mathbf{NP}]$.

Proof. Recall that $\mathbf{OUH}_\bullet = \mathbf{P}^{\mathbf{prOUH}_\bullet}$. Let $L \in \mathbf{P}^{\Pi_d}$, where $\Pi_d \in \mathbf{prOUP}^{\Pi_{d-1}}$, and let M be the corresponding polynomial-time deterministic oracle machine. The problem Π_d is defined inductively through promise problems at previous levels, that is, there are $\Pi_0 \in \mathbf{prP}$ and, for $1 \le i \le d$, $\Pi_i \in \mathbf{prOUP}^{\Pi_{i-1}}$, and there are oracle polynomial-time deterministic machines (verifiers) V_i for Π_i with oblivious unambiguous witnesses $w_{i,m}$. All the machines must work correctly for any answers to the oracle queries outside the promise set, in particular, for queries answered by any languages A_i consistent with Π_i. Let us define specific such languages. Fix a polynomial-time language A_0 consistent with Π_0, and define

$$A_i = \{q : \exists w\ V_i^{A_{i-1}}(q,w) = 1\} \qquad (1 \le i \le d).$$

By induction on i, the language A_i is consistent with Π_i; in particular, M^{A_d} must decide L. Note that we do not replace loose access with access via a consistent language in the definition of $\mathbf{OUH}_\bullet$; these languages are used only in the analysis.

We will replace these languages A_i with different languages B_i that have the same effect. Then queries to B_d will be handled within $\mathbf{OGTD}[\mathbf{NP}]$.

The advice string. Fix the input length n, and let $h = h(n)$ be a polynomial bounding the lengths of all queries and witnesses and the running times of M and of all V_i in the simulation

of M on inputs of length n, for all witnesses and all oracle answers. The advice that our $\mathbf{FP}^{\mathbf{NP}}$ generator will compute will contain all the witnesses we might use on all the lengths we can reach while running M on an input of length n. It will be a gradually constructed family $S = (S_{i,m})_{1 \le i \le d,\, 0 \le m \le h}$ of sets of witnesses, where $S_{i,m}$ consists of candidate witnesses for V_i on queries of length m; we put $S_{i,m} = \emptyset$ for $m > h$.

Restricting V_i to work with a subset of candidate witnesses; certified runs. A *run* of an oracle machine is a protocol of its behavior including the answers it received to its oracle queries (including those missing the intended promise set). We will call a run of $M(x)$ or of a $V_i(q, u)$ *certified* (w.r.t. specific S) if we can provide certificates for the answers to all its oracle queries in the following sense. We define certificates for the answers to level-i queries, i.e., queries to Π_i. At level 0, an answer to q is certified if it simply equals $A_0(q)$ (recall that $A_0 \in \mathbf{P}$). For $i \ge 1$, a certificate for a positive answer to q consists of a witness w and an accepting run of $V_i(q, w)$ together with certificates for all oracle answers in this run; a certificate for a negative answer consists, for every $s \in S_{i,|q|}$, of a rejecting run of $V_i(q, s)$ together with certificates for all oracle answers in these runs (if $S_{i,|q|} = \emptyset$, the empty certificate suffices for a negative answer). Note that, for a constant level d, certificates have size polynomial in n and the total number of witnesses in S, and they can be verified in polynomial time given 1^n and S.

New languages B_i and the decoder. If one uses the languages A_i to give the oracle answers, they are all certified w.r.t. every S: by induction on i, for $A_i(q) = 1$ use an accepting witness, and for $A_i(q) = 0$ use the rejecting runs of $V_i^{A_{i-1}}(q, s)$ for $s \in S_{i,|q|}$. Define $B_0 = A_0$ and

$$B_i = \{q : \exists s \in S_{i,|q|}\ V_i^{B_{i-1}}(q, s) = 1\} \qquad (1 \le i \le d);$$

the answers of B_i are certified similarly. The languages B_i need not be consistent with Π_i (their definition is restricted to witnesses in S), but, given S, $B_d(q)$ is computable in time polynomial in n and the total number of witnesses in S. So we define our decoder as $D(x, S) = M^{B_d}(x)$, and it can be implemented in deterministic polynomial time.

The generator. Start with $S = \emptyset$ and repeat the following round.

Ask the **NP** oracle whether M has two certified runs w.r.t. S with different outputs for some $x \in \{0,1\}^n$. If it has none, the process finishes and the generator outputs S, which is already good enough: since both runs $M^{A_d}(x)$ and $M^{B_d}(x)$ are certified, we have $D(x, S) = M^{B_d}(x) = M^{A_d}(x) = L(x)$ for every $x \in \{0,1\}^n$.

Otherwise, find this input x and two such runs together with all their certificates by repeatedly querying their prefixes with the **NP** oracle, and find the problem (which should be a missing correct witness — and indeed it will be found, as we show below). Namely, consider the two runs and check them recursively, examining the certificates of all their answers. For every negative answer to a level-i query q with $i \ge 1$, ask the **NP** oracle whether q has a certificate for a positive answer (with respect to the same S); if it does, find one, schedule its witness w for the addition to $S_{i,|q|}$, and check it as well. After this recursive process finishes, add all scheduled witnesses to S, and repeat the round.

We need to check how many rounds we need. To prove that a polynomial number of rounds suffices, it is enough to prove that each round adds some correct witness $w_{i,m}$ to some set $S_{i,m}$ — since we deal with unambiguous oblivious computations, there are at most $d(h+1)$ correct witnesses. Incorrect witnesses can be added too: a certificate for a positive answer to a level-i query q may contain an incorrect witness if q is outside the promise set of Π_i (where V_i may accept several witnesses) or if the certificate contains an incorrect answer to a promised query at a lower level (as in the third case of the claim below), which is possible only while some correct witness at a lower level is missing from S. Such witnesses stay in S, but they do not

harm our computations (we verify everything) and do not take too much time (the number of witnesses that a round adds at level i is bounded by a polynomial in h and the sizes of the sets at the levels above i: every level-j query checked in the round leads to checking at most $h(\max_m |S_{j,m}|+1)$ level-$(j-1)$ queries); hence by induction the sizes of all sets stay polynomial during all the rounds. In particular, the length we need to pad S to is polynomially bounded and the generator's running time is polynomial.

Remaining claim: each round adds at least one correct witness. Call an answer *wrong* if it is an incorrect answer to a query in the promise set of the respective Π_i. We claim that every round adds to some $S_{i,m}$ the witness $w_{i,m}$ that was not there before. One of the two runs of M has a wrong output; hence it contains a wrong answer, since otherwise it would be the run of $M^O(x)$ for some language O consistent with Π_d. Similarly, a run of V_i with all answers correct on promised queries is a run of V_i^O for a language O consistent with Π_{i-1}. Consider a wrong answer to a level-i query q discovered during the round ($i \geq 1$, because at level 0 all certified answers are always correct):

- If the answer is positive, then $q \in \Pi_{i,\mathrm{NO}}$, and the certificate contains an accepting run of $V_i(q,w)$, which must contain a wrong answer at level $i-1$.
- If the answer is negative and $w_{i,|q|} \in S_{i,|q|}$, then $q \in \Pi_{i,\mathrm{YES}}$, and the certificate contains a rejecting run of $V_i(q, w_{i,|q|})$, which must contain a wrong answer at level $i-1$.
- If the answer is negative and $w_{i,|q|} \notin S_{i,|q|}$, then a certificate for the positive answer exists (as $A_i(q)=1$), and the generator finds one with some witness w. If $w = w_{i,|q|}$, the claimed addition is made. Otherwise, by the uniqueness of the witness for $q \in \Pi_{i,\mathrm{YES}}$, the accepting run of $V_i(q,w)$ in this certificate contains a wrong answer at level $i-1$. □

Remark 5.2. *The generator follows the strategy of the simulation of robustly categorical machines by Hartmanis and Hemachandra [HH90, Theorem 2.1]: the sets $S_{i,m}$ are a more elaborate equivalent of the oracle answers learned there.*

Corollary 5.3. $\mathbf{OUH_\bullet} \subseteq \mathbf{OUM}$ *and* $\mathbf{OUH_\bullet} \subseteq \mathbf{P^{ONP}}$.

Proof. Theorem 5.1 puts $\mathbf{OUH_\bullet}$ into $\mathbf{OGTD}[\mathbf{NP}]$. Lemma 3.1 proves that this class equals $\mathbf{OGTD}[\mathbf{ONP}]$ (thus putting it into $\mathbf{P^{ONP}}$, cf. Proposition 2.5) and strengthens it to the class $\mathbf{OUGTD}[\mathbf{FP^{ONP}}, \mathbf{co\text{-}ONP}, \mathbf{P}]$ (thus putting it into $\mathbf{OUM}$). □

6 The first levels of the hierarchy

In this section we sort out what happens with even smaller classes. We check what happens for even more restricted constructions of the hierarchy. One can ask what happens if we attempt to define an oblivious unambiguous hierarchy using dot-operators — we will show that it surprisingly collapses. One can ask about applying our dot-operator to promise problems — it does not change the class. Also dot-operators are somewhat similar to single oracle queries — we will show that for such queries a promise oracle can be replaced in our context by a language (in contrast to what we know — or do not know — about promise $\mathbf{RP}$ and promise $\mathbf{ONP}$ queries). Finally, we remark that in an even smaller class $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$ the witness can in fact be made the same for both parts, making this class equal to the class $\mathbf{SOUP}$ recently defined by [HVW26].

6.1 A hierarchy defined via dot-operators collapses

We start with two simple properties of the operator $\exists_{\mathbf{ou}\bullet}$ (Definition 2.3) and then go straight to our first collapse.

Proposition 6.1.

1. $\exists_{\mathbf{ou}} \cdot \mathbf{pr}\mathcal{C} = \exists_{\mathbf{ou}} \cdot \mathcal{C}$ *for* $\mathcal{C} \in \{\mathbf{P}, \mathbf{OUP}, \mathbf{co\text{-}OUP}\}$.
2. $\exists_{\mathbf{ou}} \cdot \mathbf{OUP} = \mathbf{OUP}$.

Proof. 1. Restricting the machine for Π to the pairs (x, w) with $|w| = p(|x|)$, which are all in the promise set, gives a language in $\mathcal{C}$ that can replace Π.

2. Merge the two witnesses into one (taking care of possible length clashes). □

Theorem 6.2. $\exists_{\mathbf{ou}} \cdot \mathbf{co\text{-}OUP} = \mathbf{OUP} \cup \mathbf{co\text{-}OUP}$.

Proof. Zero witness length gives $\mathbf{co\text{-}OUP} \subseteq \exists_{\mathbf{ou}\bullet}\mathbf{co\text{-}OUP}$, and $\mathbf{OUP} = \exists_{\mathbf{ou}\bullet}\mathbf{P} \subseteq \exists_{\mathbf{ou}\bullet}\mathbf{co\text{-}OUP}$. Conversely, let $L \in \exists_{\mathbf{ou}\bullet} R$, where $R \in \mathbf{co\text{-}OUP}$, and let V be an oblivious unambiguous verifier for $\overline{R}$ with witnesses v_N. If $p \equiv 0$, then $L \in \mathbf{co\text{-}OUP}$. Otherwise, $p(n) \geq 1$ for all but finitely many n, and since finitely many input lengths can be handled separately, we may assume $p(n) \geq 1$ for all n. At most one w satisfies $(0^n, w) \in R$; hence at least one of the pairs $(0^n, 0^{p(n)})$ and $(0^n, 1^{p(n)})$ belongs to $\overline{R}$, and v_N, where N is the length of these pairs, is the only string v such that

$$V((0^n, 0^{p(n)}), v) = 1 \ \vee \ V((0^n, 1^{p(n)}), v) = 1$$

(testing a candidate oblivious witness on fixed inputs). On input $x \in \{0,1\}^n$, accept a pair (w, v) if and only if this condition holds and $V((x, w), v) = 0$. Then (w_n, v_N) is the only accepted pair for $x \in L$, and no pair is accepted for $x \notin L$. Thus $L \in \mathbf{OUP}$. □

Corollary 6.3. *The hierarchy obtained from* $\mathbf{P}$ *by alternating applications of* $\exists_{\mathbf{ou}\bullet}$ *and* $\forall_{\mathbf{ou}\bullet}$ *collapses to its second level* $\exists_{\mathbf{ou}\bullet}\mathbf{co\text{-}OUP} = \forall_{\mathbf{ou}\bullet}\mathbf{OUP} = \mathbf{OUP} \cup \mathbf{co\text{-}OUP}$*: this class is closed under* $\exists_{\mathbf{ou}\bullet}$ *and* $\forall_{\mathbf{ou}\bullet}$.

Proof. Clearly, $\exists_{\mathbf{ou}\bullet}(\mathcal{C} \cup \mathcal{D}) = \exists_{\mathbf{ou}\bullet}\mathcal{C} \cup \exists_{\mathbf{ou}\bullet}\mathcal{D}$ for all classes $\mathcal{C}$ and $\mathcal{D}$. Apply Proposition 6.1(2) and Theorem 6.2 and take complements. □

This is in contrast with the polynomial hierarchy and with the oblivious hierarchy that give the same classes when defined via oracles and via dot-operators (and are not known to collapse), as well as with the non-oblivious analogue **AUPH** of our collapsed unambiguous dot-operator hierarchy, which is infinite relative to an oracle [ST07]. Our collapse uses both obliviousness and unambiguity substantially: since the outer witness is unique, the inner **co-OUP** predicate is false on one of the fixed pairs $(0^n, 0^{p(n)})$ and $(0^n, 1^{p(n)})$, and since the witness refuting this predicate is oblivious, it can be tested on that pair.

6.2 The single query case

For deterministic machines, Cai, Hemachandra, and Vyskoč [CHV93, Proposition 2.6] observed that guarded nonadaptive access to **prUP** gives no more power than access to **UP**: tagging every query with the input yields a language in **UP**. We show that a single loose query of an unambiguous computation is guarded whenever its answer matters, which makes the tagging argument applicable.

Lemma 6.4. *Let Π be a promise problem, and let V be a polynomial-time oracle verifier that makes at most one query $q(x,w)$ on every input (x,w) and witnesses $L \in \mathbf{UP}^{\Pi}$. Let T be the set of pairs (x,w) such that the output of $V(x,w)$ depends on the answer to $q(x,w)$. Then $T \in \mathbf{P}$, and $q(x,w) \in \Pi_{\text{YES}} \cup \Pi_{\text{NO}}$ for every $(x,w) \in T$.*

Proof. To decide T, run $V(x,w)$ with both answers. Let $(x,w) \in T$, and suppose that $q = q(x,w)$ is outside the promise. Take languages O_0 and O_1 consistent with Π that differ only on q. Then $V^{O_b}(x,w) = 1$ for exactly one b. If $x \notin L$, this contradicts the correctness of V^{O_b}. If $x \in L$, then w is the accepting witness of V^{O_b} on x; since the accepting witness does not depend on the consistent language (Definition 2.1), $V^{O_{1-b}}(x,w) = 1$, a contradiction. □

Theorem 6.5. $\mathbf{UP}^{\mathbf{prUP}[1]} = \mathbf{UP}^{\mathbf{UP}[1]}$ *and* $\mathbf{OUP}^{\mathbf{prOUP}[1]} = \mathbf{OUP}^{\mathbf{OUP}[1]}$.

Proof. The containments "$\supseteq$" are trivial. Let V and Π be as in Lemma 6.4, where V witnesses $L \in \mathbf{OUP}^{\Pi}$ and $\Pi \in \mathbf{prOUP}$ via a verifier U (the case of $\mathbf{UP}$ is similar and simpler). Let $\langle x,w \rangle$ denote the pair (x,w) padded so that its length determines both $|x|$ and $|q(x,w)|$, and let

$$A = \{\langle x,w \rangle : (x,w) \in T \text{ and } q(x,w) \in \Pi_{\text{YES}}\}.$$

Then $A \in \mathbf{OUP}$: on input $\langle x,w \rangle$, reject if $(x,w) \notin T$, and otherwise run U on $q(x,w)$, which is in the promise set; all members of A of the same length have queries of the same length and hence the same witness. The new verifier computes $V(x,w)$ without the oracle if $(x,w) \notin T$, and otherwise answers the query $q(x,w)$ by querying $\langle x,w \rangle \in A$. On every input (x,w), it gives the same result as $V^{O}(x,w)$ for a fixed language O consistent with Π; thus it witnesses $L \in \mathbf{OUP}^{A}$ with the same witnesses. □

For comparison, a single loose query to $\mathbf{prRP}$ gives $\mathbf{RP}^{\mathbf{prRP}[1]} = \mathbf{BPP}$ (by the proof of [BF99, Theorem 1]), whereas $\mathbf{P} = \mathbf{RP} \neq \mathbf{BPP}$ relative to an oracle [BF99, Theorem 3]. Similarly, a single loose query to $\mathbf{prONP}$ gives $\mathbf{ONP}^{\mathbf{prONP}[1]} = \exists_{\mathbf{o}} \cdot \mathbf{pr\,co\text{-}ONP} = \mathbf{O}_2^{\mathbf{P}}$ [HVW26, Theorem 4.7]; whereas $\mathbf{ONP}^{\mathbf{ONP}[1]} = \mathbf{ONP}^{\mathbf{ONP}} = \exists_{\mathbf{o}} \cdot \mathbf{co\text{-}ONP}$ [HVW26, Theorem 4.4], and it is open whether these two classes differ.

6.3 A note on $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$

It turns out that $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$ coincides with the class $\mathbf{SOUP}$ [HVW26, Definition 2.7] of languages L such that L and $\overline{L}$ have oblivious unambiguous verifiers with the same oblivious unambiguous witnesses. Thus it inherits the polynomial-time Turing closure property from $\mathbf{SOUP}$; it also follows that it is self-low.

Proposition 6.6. $\mathbf{SOUP} = \mathbf{OUP} \cap \mathbf{co\text{-}OUP}$. *Consequently,* $\mathbf{OUP}^{\mathbf{SOUP}} = \mathbf{OUP}$ *and* $\mathbf{SOUP}^{\mathbf{SOUP}} = \mathbf{SOUP}$.

Proof. The inclusion $\mathbf{SOUP} \subseteq \mathbf{OUP} \cap \mathbf{co\text{-}OUP}$ is immediate. For the converse, let Y and Z be oblivious unambiguous verifiers for L and $\overline{L}$, respectively, with (w.l.o.g.) equally long witnesses. On input $x \in \{0,1\}^n$, guess a bit b and a string w and check

$$(b = 1 \wedge Y(0^n, w) = 1) \ \vee \ (b = 0 \wedge Z(0^n, w) = 1).$$

Since 0^n belongs either to L or to $\overline{L}$, exactly one pair passes this check. If $b = 1$, its second component is the oblivious witness for Y, so $Y(x,w) = L(x)$. If $b = 0$, similarly $1 - Z(x,w) = L(x)$. In any case, this pair (b,w) certifies either answer.

Let $A \in \mathbf{SOUP}$. Since $\mathbf{P}^{A} \subseteq \mathbf{SOUP}$ (as $\mathbf{SOUP}$ is closed under polynomial-time Turing reductions [HVW26]), Proposition 6.1(2) gives $\mathbf{OUP}^{A} = \exists_{\mathbf{ou}} \cdot \mathbf{P}^{A} \subseteq \exists_{\mathbf{ou}} \cdot \mathbf{OUP} = \mathbf{OUP}$; hence $\mathbf{co\text{-}OUP}^{A} \subseteq \mathbf{co\text{-}OUP}$ and $\mathbf{SOUP}^{A} \subseteq \mathbf{OUP}^{A} \cap \mathbf{co\text{-}OUP}^{A} \subseteq \mathbf{SOUP}$. □

The class **SOUP** is an unambiguous version of $\mathbf{YP}^*$ [HVW26], and $\mathbf{OUP} \cap \mathbf{co\text{-}OUP}$ is an unambiguous version of $\mathbf{YP} = \mathbf{ONP} \cap \mathbf{co\text{-}ONP}$ [AD14]; the proof of Proposition 6.6 relies on uniqueness and does not give $\mathbf{YP} = \mathbf{YP}^*$.

Acknowledgement

The author is indebted to Ilya Volkovich and Ryan Williams, with whom he started the study of unambiguous oblivious classes [HVW26, HV26b].

References

[Aar07] S. Aaronson. The learnability of quantum states. *Proc. R. Soc. A*, 463(2088):3089–3114, 2007.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

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

[BF99] H. Buhrman and L. Fortnow. One-sided versus two-sided error in probabilistic computation. In *STACS 1999*, LNCS 1563, pages 100–109, 1999.

[BFHL13] H. Buhrman, L. Fortnow, J. M. Hitchcock, and B. Loff. Learning reductions to sparse sets. In *MFCS 2013*, LNCS 8087, pages 243–253, 2013.

[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.

[CGRS04] M. Crâsmaru, C. Glaßer, K. W. Regan, and S. Sengupta. A protocol for serializing unique strategies. In *MFCS 2004*, LNCS 3153, pages 660–672, 2004.

[CHV93] J.-Y. Cai, L. A. Hemachandra, and J. Vyskoč. Promises and fault-tolerant database access. In K. Ambos-Spies, S. Homer, and U. Schöning, editors, *Complexity Theory: Current Research*, pages 101–146. Cambridge University Press, 1993. A preliminary version is available as Technical Report TR 447, Department of Computer Science, University of Rochester, 1993, https://hdl.handle.net/1802/599.

[CR06] V. T. Chakaravarthy and S. Roy. Oblivious symmetric alternation. In *STACS 2006*, LNCS 3884, pages 230–241, 2006.

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

[FSW09] L. Fortnow, R. Santhanam, and R. Williams. Fixed-polynomial size circuit bounds. In *CCC 2009*, pages 19–26, 2009.

[Gav95] R. Gavaldà. Bounding the complexity of advice functions. *J. Comput. Syst. Sci.*, 50(3):468–475, 1995.

[GM15] O. Goldreich and O. Meir. Input-oblivious proof systems and a uniform complexity perspective on P/poly. *ACM Trans. Comput. Theory*, 7(4):16:1–16:13, 2015.

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

[HH90] J. Hartmanis and L. A. Hemachandra. Robust machines accept easy sets. *Theor. Comput. Sci.*, 74(2):217–225, 1990.

[HO02] L. A. Hemaspaandra and M. Ogihara. *The Complexity Theory Companion*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2002.

[HR97] L. A. Hemaspaandra and J. Rothe. Unambiguous computation: Boolean hierarchies and sparse Turing-complete sets. *SIAM J. Comput.*, 26(3):634–653, 1997.

[HS15] E. A. Hirsch and D. Sokolov. On the probabilistic closure of the loose unambiguous hierarchy. *Inf. Process. Lett.*, 115(9):725–730, 2015.

[HV26a] E. A. Hirsch and I. Volkovich. Upper and lower bounds for the linear ordering principle. In *STACS 2026*, LIPIcs 364, pages 52:1–52:19, 2026. Full version: ECCC TR25-142, revision 2, 2026, https://eccc.weizmann.ac.il/report/2025/142/revision/2/download.

[HV26b] E. A. Hirsch and I. Volkovich. The ultimate Karp–Lipton collapse. arXiv:2611.16096, 2026.

[HVW26] E. A. Hirsch, I. Volkovich, and R. Williams. Input-oblivious complexity classes in the second level of the polynomial hierarchy. ECCC TR26-250, revision 2, 2026, https://eccc.weizmann.ac.il/report/2026/250/revision/2/download.

[Käm91] J. Kämper. Nonuniform proof systems: a new framework to describe nonuniform and probabilistic complexity classes. *Theor. Comput. Sci.*, 85(2):305–331, 1991.

[KP24] O. Korten and T. Pitassi. Strong vs. weak range avoidance and the linear ordering principle. In *FOCS 2024*, pages 1388–1407, 2024. Full version: ECCC TR24-076, 2024, https://eccc.weizmann.ac.il/report/2024/076/download.

[Kre88] M. W. Krentel. The complexity of optimization problems. *J. Comput. Syst. Sci.*, 36(3):490–509, 1988.

[KS85] K.-I. Ko and U. Schöning. On circuit-size complexity and the low hierarchy in NP. *SIAM J. Comput.*, 14(1):41–51, 1985.

[KT94] J. Köbler and T. Thierauf. Complexity-restricted advice functions. *SIAM J. Comput.*, 23(2):261–275, 1994.

[LR94] K.-J. Lange and P. Rossmanith. Unambiguous polynomial hierarchies and exponential size. In *Proceedings of the Ninth Annual Structure in Complexity Theory Conference*, pages 106–115. IEEE, 1994.

[MPR26] D. Miloschewsky, S. Podder, and D. Rudolph. Promises should be taken seriously: on relativization with promise problems. arXiv:2609.07945, 2026.

[NR98] R. Niedermeier and P. Rossmanith. Unambiguous computations and locally definable acceptance types. *Theor. Comput. Sci.*, 194(1–2):137–161, 1998.

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

[ST07] H. Spakowski and R. Tripathi. On the power of unambiguity in alternating machines. *Theory Comput. Syst.*, 41(2):291–326, 2007.

[ST09] H. Spakowski and R. Tripathi. Hierarchical unambiguity. *SIAM J. Comput.*, 38(5):2079–2112, 2009.

[Wag86] K. W. Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Inform.*, 23(3):325–356, 1986.

[Zac88] S. Zachos. Probabilistic quantifiers and games. *J. Comput. Syst. Sci.*, 36(3):433–451, 1988.

[ZF87] S. Zachos and M. Fürer. Probabilistic quantifiers vs. distrustful adversaries. In *FSTTCS 1987*, LNCS 287, pages 443–455, 1987.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il