Universiteit Leiden
The Netherlands

Separating quantum and classical computing: rigorous proof and practical application

Marshall, S.C.

Citation

Marshall, S. C. (2025, May 27). *Separating quantum and classical computing: rigorous proof and practical application*. Retrieved from https://hdl.handle.net/1887/4247215

Version:	Publisher's Version
License:	Licence agreement concerning inclusion of doctoral thesis in the Institutional Repository of the University of Leiden
Downloaded from:	https://hdl.handle.net/1887/4247215

Note: To cite this publication please use the final published version (if applicable).

CHAPTER 3

On Bounded Advice Classes

3.1 Introduction

A common concept in complexity theory is 'advice', where a Turing machine is provided with an advice string alongside its input. This string can provide any conceivable advice on how to solve problems up to size n but must be the same advice string for all inputs of a given size. As this advice string could be anything, advice classes are often much more powerful than their unadvised counterparts, containing problems which are otherwise undecidable. This powerful advice string allows advice classes to act as useful models for problems with long/expensive preprocessing phases (modelled as advice) followed by faster/cheaper processing phases. However, in many of these situations, the advice is produced by a more powerful Turing machine, but not an infinitely more powerful Turing machine, as would be the case in standard advice classes. In such situations, it would be more appropriate to study a form of 'bounded advice', where we ask what a given Turing machine of some complexity can accomplish when given advice generated by a more powerful, but not unbounded, Turing machine. For many cases where we might want to use advice classes, this bounded advice captures a much tighter idea of what we are trying to model:

- Cryptography, when an adversary may be able to expend immense

resources if the information learnt allows them to break an encryption scheme for a much smaller cost during run-time [33, 34].

- Quantum computing, where the quantum computer is used to prepare an algorithm which will run classically [35–37], or where a classical machine learning algorithm attempts to replicate the quantum model just using data from the quantum model [38].

- Machine learning, where a large and expensive training run is used to compute a smaller number of weights for a machine learning model. Once these weights are computed the model is often comparatively quite cheap to run [39–41].

With bounded advice classes, we can address these scenarios, we can study how a cryptographic scheme performs against an attacker with 'only' an EXP generated cheat sheet, instead of an unbounded cheat sheet. Formalising bounded advice allows us to exactly extract these ideas, and to ask our key question: *When is advice generated by a given Turing machine useful?* In this context, we can informally say a certain complexity class is a useful advice generator to another complexity class when the former class can be used to create advice strings that allow the latter class to recognize strictly more languages.

In Section 3.2 we give common notation and define the class of problems solvable by advice generated by a certain complexity class, e.g. $\mathsf{P/poly}^{\mathsf{EXP}}$ is the class of problems solvable by a polynomial-time Turing machine with access to an exponential-time machine. We also review closely connected work on the complexity of advice functions [27, 39, 42–45]. This previous work typically studies the functional complexity of generating the advice for a given problem in $\mathsf{P/poly}$. While our results allow us to push the state of the art in this research direction (we are able to improve the strongest result), we primarily focus on a closely related, but different question: 'For a given complexity of advice generator, what problems can be solved?'.

In Section 3.3 we prove a connection between bounded advice and unary languages. Namely:

$$\mathsf{P/poly}^{\mathsf{B}} = \mathsf{P/poly}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})} = \mathsf{P}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})},$$

where $\mathsf{Un}(\mathsf{B})$ denotes the set of all unary languages in a class B.

This connection allows us to directly produce results on when given complexity classes generate useful advice in Section 3.4.

- $\mathsf{P} \subsetneq \mathsf{P/poly}^{\mathsf{EXP}}$

- $\mathsf{P} \subsetneq \mathsf{P/poly}^{\mathsf{NP}} \iff \mathsf{EXP} \neq \mathsf{EXP}^{\mathsf{NP}}$
- $\mathsf{P} \subsetneq \mathsf{P/poly}^{\mathsf{PSPACE}} \iff \mathsf{EXP} \neq \mathsf{EXPSPACE}$

We connect our framework to existing quantum classes, $\mathsf{BPP/samp}^{\mathsf{BQP}} \subseteq \mathsf{P/poly}^{\mathsf{Un}\left(\mathsf{ZPP}^{\mathsf{NP}^{\mathsf{BQP}}}\right)}$[1] and with randomised advice $\mathsf{BPP/samp}^{\mathsf{BQP}} \subseteq \mathsf{P/rpoly}^{\mathsf{BQP}}$ in Section 3.4.2. This section also connects bounded quantum advice (where the advice itself is a quantum state) into our framework, proving the bounded-advice equivalent of the well known result, $\mathsf{BQP/qpoly} \subseteq \mathsf{QMA/poly}$ [46]:

$$\mathsf{BQP/qpoly}^{\mathsf{B}} \subseteq \mathsf{QMA/poly}^{\mathsf{ZPP}^{\mathsf{QMA}^{\mathsf{B}}}}.$$

3.2 Background

3.2.1 General notation and definitions

This work makes use of notation or definitions that, while standard, may only be known to readers of specific backgrounds. To aid with readibility by a wide audience we provide a short list of notation, definitions of common complexity classes should be checked in the complexity zoo [47].

- **The symbol '#':** It is often useful to join two inputs together to pass them both to a Turing machine, e.g. if we want to calculate $x + y$ for $x = 010$ and $y = 11$ we will need to pass both x and y, but as both are written in binary a simple concatenation makes it unclear where x ends and y begins, $xy = 01011$. To solve this problem we introduce a special symbol to our alphabet, #, which will be used to simply demarcate where two strings meet, e.g. $x\#y = 010\#11$
- **Prefixes:** We say x *is a length* n *prefix of* y if x is length n and the first n letters of y are equal to x, e.g. $x = 100$ is a length 3 prefix of $y = 100010$.
- **Turing machine standardisation:** Unless otherwise specified a Turing machine is assumed to be a polynomial time deterministic Turing machine. Turing machine is often abbreviated to TM. Occasionally the TM will not be deterministic or not polynomial time, this will be specified or be clear from context (i.e. the previous line

[1] We will later define $\mathsf{BPP/samp}^{\mathsf{BQP}}$ as the class of problems which can be solved on a quantum computer or by a BPP machine with samples from the quantum computer

specified a non-deterministic Turing machine and the following line talks about 'this TM').

- **Unary:** A unary language is a language such that all elements $x \in L$ consist of a string of 1's, i.e. $\forall x \in L\ \exists m$ such that $x = 1^m$.

  The set of all unary languages is written TALLY, all unary languages in some complexity classes B could be written $\mathsf{TALLY} \cap \mathsf{B}$, but we find it is clearer to write $\mathsf{Un}(\mathsf{B}) := \mathsf{TALLY} \cap \mathsf{B}$.

- **Sparsity:** A sparse language is a language which has at most poly(n) elements of size n.

  The class of all sparse languages is written as SPARSE.

  The set of all sparse languages in a complexity class, B, can be written $\mathsf{SPARSE} \cap \mathsf{B}$ (the intersection of these two classes) but for notational clarity, we will write it as $\mathsf{SP}(\mathsf{B}) := \mathsf{SPARSE} \cap \mathsf{B}$. This significantly improves readability but may confuse readers familiar with an older form of relativisation notation: $\mathsf{B}(\mathsf{C}) := \mathsf{B}^{\mathsf{C}}$.

- **Which exponential time? E vs EXP.** There are two standard, but non-equivalent, definitions of exponential time decision problems: E, which is equal to $\mathsf{DTIME}(2^{\mathsf{O}(\mathsf{n})})$, and EXP, which is equal to $\bigcup_{\text{all polynomials } p} \mathsf{DTIME}(2^{\mathsf{p}(\mathsf{n})})$. This work will make it clear when our results apply to only one of these definitions or to both. Older works, some cited here, may not conform to this nomenclature.

- $\boldsymbol{L(x)}$**:** For a language L the function $L(x)$ is equal to 1 if $x \in L$ and 0 otherwise.

- $\mathbf{1^n}$**:** 1^n denotes a string of ones of length n. $1^4 = 1111$.

- $\mathsf{B} - \mathsf{C}$. For two complexity classes, B and C, the class $\mathsf{B} - \mathsf{C}$ is the set of all the languages in B that are not also in C. This notation comes from the definition of complexity classes as sets.

3.2.2 Oracle-machines and double oracles

The standard definition of an oracular complexity class is given as follows.

Definition 3.2.1 (Oracular classes)
The complexity class of problems solvable by an algorithm in B *with access*

to an oracle for a language L is B^{L}. This extends to define an oracular complexity class, C by taking the union:

$$\mathsf{B}^{\mathsf{C}} := \bigcup_{\mathsf{L} \in \mathsf{C}} \mathsf{B}^{\mathsf{L}}$$

Sometimes a single Turing machine will have to make Oracle calls to two different languages, while this can be defined within the existing framework by creating a third language that can answer oracular calls to either language, we find it much simpler to define a double oracle machine.

Definition 3.2.2 (Double oracles)
For complexity classes B, C and D, a double oracle machine is a Turing machine, T, with oracular access to two problems, L_0, and L_1. The complexity class is:

$$\mathsf{D}^{\mathsf{B},\mathsf{C}} = \bigcup_{\mathsf{L_B} \in \mathsf{B}, \mathsf{L_C} \in \mathsf{C}} \mathsf{D}^{\mathsf{L_B},\mathsf{L_C}}$$

In previous works the notation $\mathsf{L_B} \oplus \mathsf{L_C}$ is used for this purpose. While logically equivalent we find the comma notation to be much clearer, additionally it makes sparsity more apparent as either if either B or C (but not both) are sparse their set-addition may not be.

3.2.3 Bounded and unbounded advice classes

We can now move onto the meat of our definitions: advice classes.

Definition 3.2.3 (Advice classes)
A language L is in the advice class of B, B/poly, if there exists an integer, d, and a set of advice strings, $a = \{a_i : |a_i| < di^d\}_{i \in \mathbb{N}}$, such that the language

$$L' = \{x \# a_{|x|} : x \in L\}$$

is in B.

At this point the extension of this definition to bounded advice seems obvious, we just have to specify what it means for a complexity class to 'generate' a piece of advice. But it is not clear how a decision algorithm (which outputs a bit) 'in' C generates a piece of advice (a string of symbols). Fortunately, a lot of heavy lifting has already been done with the definition of 'transducers' [48], which, unlike the standard definition of a Turing machine, gives the entire final state of the tape as the output, where a regular Turing machine only outputs 'reject' or 'accept'.

Unfortunately, on closer inspection, transducers do not have the properties we want; consider polynomial-sized advice generated by an exponential time machine, which polynomially-sized piece of the exponentially long tape should we take? Or a non-deterministic Turing machine, which path do we accept? What about for even more artificial classes, like SPARSE, which is defined without reference to Turing machines at all! Previous works [27, 39, 42–45] have dealt with these problems by using functional complexity classes. While well-defined, functional complexity classes can have very different properties from their decision-class counterparts [49], they also lack a clear notion of unary languages, which is of key importance to this work.

To combat these issues we choose a definition of bounded advice that uses the tape from a polynomial-time Turing machine with oracular access to another class. This is equivalent to using a FP^{B} advice generator for some oracle class B. This definition solves our problem of defining the output tape but inherently transforms the advice generator into a P^{C} machine, which makes it difficult to study classes for which $\mathsf{P}^{\mathsf{C}} \neq \mathsf{C}$.

Definition 3.2.4 (Polynomial-sized bounded advice classes)
For arbitrary complexity classes, B *and* C*, a language* L *is in* $\mathsf{B}/\mathsf{poly}^{\mathsf{C}}$ *if there is an infinite set of advice strings,* $\{a_n\}_{n\in\mathbb{N}}$*, such that:*

- *(Advice Generation) There exists a deterministic polynomial-time 'advice generating' Turing machine,* T*, with oracular access to* C *which on input* 1^n *terminates in* poly(n) *time with* a_n *as the final state of its tape.*

- *(Advice Use) The language*

$$L' = \{x\#a_{|x|} : x \in L\}$$

  is in B

The notion of *unbounded* advice classes extends beyond classes of the form B/poly; logarithmic advice is possible with B/log, quantum advice, consisting of quantum states is possible with $\mathsf{B}/\mathsf{qpoly}$, or randomised advice can be captured with $\mathsf{B}/\mathsf{rpoly}$ [49]. Similarly, different flavours of bounded advice are possible; $\mathsf{B}/\mathsf{exp}^{\mathsf{C}}$ is the class of problems solvable by B given advice generated by a Turing machine running for exponentially long, or a quantum advice generator $\mathsf{B}/\mathsf{qpoly}^{\mathsf{C}}$, or even a non-deterministic advice generator $\mathsf{B}/\mathsf{npoly}^{\mathsf{C}}$. Exploring these definitions may allow future papers to work around the 'P-oracle problem' (how to study advice generated by

a class C such that $\mathsf{P}^{\mathsf{C}} \neq \mathsf{C}$). For our purposes, we will only need one of the possible extensions of bounded advice.

Definition 3.2.5 ($\mathsf{BQP/qpoly}^{\mathsf{C}}$)
For a fixed gate set and initial state $|0\rangle$, a language L is in $\mathsf{BQP/qpoly}^{\mathsf{C}}$ if the following two criteria are true:

- *(Advice Generation) There exists a polynomial-sized uniform family of quantum circuits with oracular access to C, $\{Q_n\}_{n\in\mathbb{N}}$ generating a set of advice states, $\{|\phi_n\rangle = Q_n\,|0\rangle\,|n \in \mathbb{N}\}$.*

- *(BQP Advice Use) There exists a polynomial-time quantum algorithm A such that for all n, $A(x, |\phi_n\rangle)$ outputs $L(x)$ with probability more than $2/3$ for all x up to length n.*

3.2.4 Previous work

We are not the first to consider bounded advice classes although a seemingly minor difference in our framing brings us to a novel set of questions and results. Much of the previous work on bounded advice has centred on the question 'If a given class A is in $\mathsf{P/poly}$, what is the complexity of generating the advice string?'. This naturally leads to studying the complexity of the 'advice function', a function which prints the advice used by the $\mathsf{P/poly}$ algorithm. The strongest answer to this question was noticed by Köbler and Watanabe [27] as a corollary to a result by Bshouty et al. [45], showing that the advice function for any $\mathsf{A} \in \mathsf{P/poly}$ is in $\mathsf{FZPP}^{\mathsf{NP}^{\mathsf{A}}}$.

While our bounded advice classes are equivalent to studying the advice function, the reframing leads to a different set of questions. Instead of asking 'for a given problem, what is the advice function to solve this?' we ask 'for a given advice function, which problems can this solve?'. This naturally leads to our central question of 'when are certain advice classes useful?'. It also forces us to standardise our notation and classes, which proves useful to deriving more flexible results. Finally, instead of studying functional classes to specify the advice function, we choose to stay within decision classes. This allows us to use the well-known connection between advice classes and unary languages.

For clarity, we will restate the state of the art result by Bshouty/Köbler using our terminology.

Theorem 3.2.1 ([27, 45])
For any set $\mathsf{A} \in \mathsf{P/poly}$: $\mathsf{A} \in \mathsf{P/poly}^{\mathsf{ZPP}^{\mathsf{NP}^{\mathsf{A}}}}$

Reexamining the proof of [45] it is easy to see this generalises to B/poly. We can also use the connection to unary languages we derive later to produce a stronger version of Bshouty et al.'s theorem.

Theorem 3.2.2

For complexity class A, if $\mathsf{A} \subseteq \mathsf{C}$ *and* $\mathsf{A} \subseteq \mathsf{B}/\mathsf{poly}$ *then* $\mathsf{A} \subseteq \mathsf{B}/\mathsf{poly}^{\mathsf{Un}\left(\mathsf{ZPP}^{\mathsf{NP}^{\mathsf{B},\mathsf{C}}}\right)}$

Many of our other results will rely on the well-known connection between advice classes and unary/sparse languages. As most research in advice functions has been functional languages this connection has scarcely been used. The only connection we are aware of is to a 1985 result by Ko and Schöning [43], showing that all sparse sets in Σ_i^p are also in P/poly with an advice function in $\mathsf{F}\Delta_{i+1}^p$.

3.3 Connection to sparse and unary languages

It is well known that any language in P/poly is Turing reducible to a unary language. Here we prove an analogous result, that any language in $\mathsf{P}/\mathsf{poly}^{\mathsf{B}}$ is Turing reducible to a unary language in P^{B}. We show this inclusion is tight, i.e. the set $\mathsf{P}/\mathsf{poly}^{\mathsf{B}}$ is exactly the set of languages which are Turing reducible to a unary language in P^{B}. We further show that all languages that are Turing reducible to sparse languages in P^{B} (i.e. in $\mathsf{P}^{\mathsf{SP}(\mathsf{P}^{\mathsf{B}})}$) are also in $\mathsf{P}/\mathsf{poly}^{\mathsf{NP}^{\mathsf{B}}}$.

To prove the main results we will need the following lemmas:

Lemma 3.3.1

For any complexity class B*:*

$$\mathsf{P}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})} \subseteq \mathsf{P}/\mathsf{poly}^{\mathsf{B}}.$$

Proof. As $\mathsf{Un}(\mathsf{P}^{\mathsf{B}})$ has at most n elements up to length n, the advice can simply be an n length string, a, whose m'th element is 1 iff $1^m \in L$. Obviously, this string can be generated by n queries to an P^{B} language. The advice-receiving Turing machine can then use this advice to simulate oracle calls to the P^{B} language. □

Lemma 3.3.2

For any complexity classes B *and* C*,*

$$\mathsf{C}/\mathsf{poly}^{\mathsf{B}} \subseteq \mathsf{P}^{\mathsf{C},\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$$

Proof. Take any $L \in \mathsf{C}/\mathsf{poly}^{\mathsf{B}}$, let a_n be the advice string generated by the advice generator, and let $p(n)$ be the polynomial which bounds the length of the advice string ($|a_n| \leq p(n)$), W.L.O.G. we may assume the advice is a bit string of exactly $p(n)$ bits. We will now describe an oracle in $\mathsf{Un}(\mathsf{P}^{\mathsf{B}})$ that can be used to generate a_n in polynomial time.

Define the language:

$$L_{advice} = \{1^{\Sigma_{k=1}^{n-1} p(k)} 1^m | \text{ the m'th bit of } a_n \text{ is } 1 \}$$

By the definition of bounded advice a_n can be generated in polynomial time with a B oracle, deciding if a particular bit is 1 or 0 is also polynomial time. Therefore L_{advice} is a unary language in P^{B}.

The language L_{advice} can be used to construct a_n one bit at a time. This advice string can then be passed to the C oracle to simulate the $\mathsf{C}/\mathsf{poly}^{\mathsf{B}}$ algorithm, showing that the language L is in $\mathsf{P}^{\mathsf{C},\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$. □

Combining lemma 3.3.1 and lemma 3.3.2 we get the central theorem of this section:

Theorem 3.3.1
For any complexity class B,

$$\mathsf{P}/\mathsf{poly}^{\mathsf{B}} = \mathsf{P}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$$

A simple corollary, $\mathsf{P}/\mathsf{poly}^{\mathsf{B}} = \mathsf{P}/\mathsf{poly}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$, provides us with a key insight: all bounded advice is unary advice.

The result of Theorem 3.3.1 also holds for advice receivers other than P. The choice of fixing P as the advice receiver was merely to ensure Turing reductions. Following the steps of the proof for other classes provides the following corollary.

Corollary 3.3.2

$$\mathsf{BPP}/\mathsf{poly}^{\mathsf{B}} = \mathsf{BPP}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$$

$$\mathsf{NP} \cap \mathsf{coNP}/\mathsf{poly}^{\mathsf{B}} = (\mathsf{NP} \cap \mathsf{coNP})^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$$

$$\mathsf{BQP}/\mathsf{poly}^{\mathsf{B}} = \mathsf{BQP}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$$

While the connection to unary languages is tighter, bounded advice can also be connected to sparsity, finding that sparse languages always sit inside some bounded advice class (Theorem 3.3.3), and bounded advice classes are contained in sparse languages (Corollary 3.3.4). Theorem 3.3.3

is similar to the work of Ko and Schöning [43] who show that all sparse sets in Σ_i^p are also in $\mathsf{P/poly}$ with an advice function in $\mathsf{F}\Delta_{i+1}^p$. Our result differs as it is both wider-reaching (applying to classes outside the polynomial hierarchy) and showing inclusions in both directions (advice inside sparsity, and sparsity inside advice).

Theorem 3.3.3
For any complexity class B*,*

$$\mathsf{P}^{\mathsf{SP}(\mathsf{P}^{\mathsf{B}})} \subseteq \mathsf{P/poly}^{\mathsf{SP}(\mathsf{NP}^{\mathsf{B}})}$$

Proof. Let K be a language in $\mathsf{P}^{\mathsf{SP}(\mathsf{P}^{\mathsf{B}})}$. Then there exists a sparse language, L in P^{B} such that $K \in P^L$. As L is sparse there are at most polynomially many strings $x \in L$ of any particular length, n. Our strategy will be to find all of these strings with the advice generating TM, to create a list, $A_n = \{x : x \in L, |x| \leq n\}$, and pass A_n as advice. The advice using TM can then simulate an oracle call to L by simply checking if x is on the list.

It is possible to generate A_n in polynomial time with access to an NP^{B} oracle, with access to the (sparse) language:

$$L'_{sparse} = \{1^n \# p : \exists x \text{ with } x \in L, \text{ and } p \text{ is a prefix of } x\}.$$

As L is in P^{B} existence of an $x \in L$ is in NP^{B}.

We can use L'_{sparse} to find a complete list of strings by beginning with an empty list, $A = \{\}$. We begin by using algorithm 1 starting from an empty string to find an initial element, $x \in L$, and put this element in A. If we try to nïavely reuse algorithm 1 to find more elements it may not return a new value of $x \in L$. We must force the algorithm to return a new value of x. We can solve this problem by beginning the search from some particular prefix, instead of the empty string.

Suppose our list, A, has m elements in it. This defines a partially explored trie, as shown in figure 3.1. At each node there are two children, one child must be an explored path that leads to at least one previously found element of $x \in A$, the other child may be unexplored. By beginning the search with a prefix from one of the unexplored children (highlighted in step b of figure 3.1) we will find any unfound elements beginning with that prefix. If a new element is found we add it to A and search its unexplored prefixes. If no new elements are found after all prefixes have been explored then we conclude the list A is complete. This list is our advice.

As A_n has at most polynomial elements and the tree is polynomially deep, finding each element takes at most polynomial time. Creating the

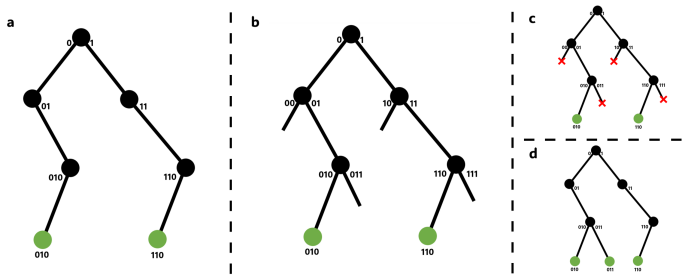

Figure 3.1: Step of the procedure for finding all the elements of a sparse language using L'. We begin with a possibly incomplete list, A, consisting of elements of L below to some given size, n. In step **a.** we represent A with a prefix tree, each layer representing a possible prefix of the string. If a string is a prefix for an element in A its branch continues to a green node. **b.** We add the possible unexplored children to the tree. We then use algorithm 1 starting from each unexplored child prefixes to find possible new elements of A. Either one is found and added to A (**d.**) or none are found, implying there are none left to find and A is complete (**c.**).

advice string A therefore takes polynomial time with an NP^{B} oracle, as L'_{sparse} is also sparse, we have shown the result:

$$\mathsf{P}^{\mathsf{SP}(\mathsf{P}^{\mathsf{B}})} \subseteq \mathsf{P/poly}^{\mathsf{SP}(\mathsf{NP}^{\mathsf{B}})}.$$

□

Theorem 3.3.3 shows sparse languages are in a form of bounded advice, the converse is also possible as a simple corollary of Theorem 3.3.1 as $\mathsf{Un}(\mathsf{B}) \subseteq \mathsf{Sp}(\mathsf{B})$.

Corollary 3.3.4
$\mathsf{P/poly}^{\mathsf{B}} \subseteq \mathsf{P}^{\mathsf{SP}(\mathsf{P}^{\mathsf{B}})}$

3.4 When is bounded advice useful?

This section will address this chapter's fundamental question: *"When is advice generated by a given complexity class useful?"* i.e. when advice from one class increases the amount of problems that can be solved by another class. In the first subsection, we will characterise when some major complexity classes generate advice useful to a polynomial time machine (i.e. when $\mathsf{P/poly}^{\mathsf{A}}$ is not simply equal to P). In the second

Algorithm 1 Binary search variant to find a_n from L'_{sparse}

Input: An integer n, a starting prefix, p
Output: The string a'

```
a' ← p                                  ▷ Initializing a' to the given prefix
advice_complete ← False                 ▷ Initializing advice_complete to False
while advice_complete is False
    if 1^n#a'0 ∈ L'_sparse              ▷ Find a symbol to append to the advice string
        a' ← a'0
    else if 1^n#a'1 ∈ L'_sparse
        a' ← a'1
    else
        advice_complete ← True          ▷ If no next symbol can be found, then the string is complete
```

subsection, we look at when quantum complexity classes are useful advice generators, this allows us to study a class of proposed uses of quantum computing that prepare classical algorithms. We connect $\mathsf{BPP/rpoly}^{\mathsf{BQP}}$ to the class of languages that can be decided with samples from a quantum computer through $\mathsf{BPP/samp}$[38]. We derive a result connecting quantum bounded polynomial advice (i.e. a quantum state) to non-quantum bounded polynomial advice (a classical bitstring).

3.4.1 Useful classical advice

As shown in the previous section $\mathsf{P/poly}^{\mathsf{B}} = \mathsf{P}^{\mathsf{Un}(\mathsf{P}^{\mathsf{B}})}$, thus bounded advice is useful if and only if unary languages are useful oracles. Much is known about the conditions for the existence of various unary languages, thus, this connection opens a variety of choices for grounding the hardness of various bounded advice classes. We provide a number of these results for the most famous complexity classes.

Theorem 3.4.1
$\mathsf{P} \neq \mathsf{P/poly}^{\mathsf{EXP}}$

Theorem 3.4.2
$\mathsf{P} \neq \mathsf{P/poly}^{\mathsf{PSPACE}}$ *if and only if* $\mathsf{EXPSPACE} \neq \mathsf{EXP}$.

Theorem 3.4.3
$\mathsf{P} \neq \mathsf{P/poly}^{\mathsf{NP}}$ *if and only if* $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{EXP}$.

Theorem 3.4.4
$\mathsf{BPP} \neq \mathsf{BPP/poly}^{\mathsf{BQP}}$ *if and only if* $\mathsf{BPEXP} \neq \mathsf{BQEXP}$

Many other results in unary languages can be used to prove results in advice classes. Such as showing the polynomial hierarchy gives useful advice if the exponential hierarchy doesn't collapse. However, for brevity, we have provided only these 4.

Proof of Theorem 3.4.1. We prove this result via showing $\mathsf{P} \neq \mathsf{P}^{\mathsf{Un(EXP)}}$. By the time hierarchy theorem, there exists an L such that,

$$L \in \mathsf{DTIME}(2^{\mathsf{n}^3}) - \mathsf{DTIME}(2^{\mathsf{n}^2}).$$

From L we define the unary language:

$$L_{unary} = \{1^x : x \in L\}$$

L_{unary} cannot be decided in $o(2^{log(n)^2})$, a quasipolynomial, thus $\mathsf{L_{unary}} \notin \mathsf{P}$.

Given a string from L_{unary}, 1^x, calculating x takes x steps, therefore L_{unary} can be decided by a deterministic Turing machine in $O(2^{log(n)^3})$ time. As $\mathsf{DTIME}(2^{\mathsf{log(n)}^3}) \subseteq \mathsf{EXP}$, $L_{unary} \in \mathsf{EXP}$. Therefore we have demonstrated there exists a unary language in $\mathsf{EXP} - \mathsf{P}$, by theorem 3.3.1 this implies $\mathsf{P/poly}^{\mathsf{EXP}} \neq \mathsf{P}$ □

Proof of Theorem 3.4.2. By Theorem 3.3.1 $\mathsf{P/poly}^{\mathsf{PSPACE}} = \mathsf{P}^{\mathsf{Un(P^{PSPACE})}}$, as $\mathsf{P}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$ [47] we derive the equality: $\mathsf{P/poly}^{\mathsf{PSPACE}} = \mathsf{P}^{\mathsf{Un(PSPACE)}}$. If $\mathsf{EXPSPACE} = \mathsf{EXP}$ there are no unary languages in $\mathsf{PSPACE} - \mathsf{P}$ [50], proving the 'only if' direction. For the other direction we notice that all unary languages in PSPACE are in $\mathsf{P}^{\mathsf{Un(P^{PSPACE})}}$, therefore if $\mathsf{P}^{\mathsf{Un(P^{PSPACE})}} = \mathsf{P}$ there are no unary languages in PSPACE and $\mathsf{EXPSPACE} = \mathsf{EXP}$ [50]. □

Proof of Theorem 3.4.3. There are unary languages in $\mathsf{P}^{\mathsf{NP}} - \mathsf{P}$ if and only $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{EXP}$. Therefore $\mathsf{P/poly}^{\mathsf{NP}} \neq \mathsf{P}$ if and only if $\mathsf{EXP}^{\mathsf{NP}} \neq \mathsf{EXP}$. □

Proof of Theorem 3.4.4. There are unary languages in $\mathsf{BQP} - \mathsf{BPP}$ if and only $\mathsf{BPEXP} \neq \mathsf{BQEXP}$, by a simple extension of the arguments in [50]. Therefore $\mathsf{P/poly}^{\mathsf{BQP}} \neq \mathsf{P}$ if and only if $\mathsf{BQEXP} \neq \mathsf{BPEXP}$. □

3.4.2 Useful Quantum advice

This subsection studies bounded quantum advice and when it is useful. First, we examine classical advice strings generated by quantum computers, this case contains many of the proposed algorithms to use quantum

computers to prepare algorithms for classical machines [35, 36, 38, 51]. One prominent example, using samples from some problem to produce an algorithm that can solve instances of the same problem, is formalised in the class $\mathsf{BPP/samp}$, if the samples are produced from a quantum computer, we denote it as the class $\mathsf{BPP/samp}^{\mathsf{BQP}}$ [38]. While it may be expected that $\mathsf{BPP/samp}^{\mathsf{BQP}}$ is contained in $\mathsf{P/poly}^{\mathsf{BQP}}$ this neglects the randomness in the sample selection, and we will show it is instead contained in $\mathsf{P/rpoly}^{\mathsf{BQP}}$. We also show a derandomised result, that $\mathsf{BPP/samp}^{\mathsf{BQP}}$ is contained in $\mathsf{P/poly}^{\mathsf{Un}\left(\mathsf{ZPP}^{\mathsf{NP}^{\mathsf{BQP}}}\right)}$. Second, we study the properties of bounded advice when the advice is itself a quantum state: $\mathsf{BQP/qpoly}^{\mathsf{B}}$, finding it is contained in a classical bounded advice state $\mathsf{QMA} \cap \mathsf{coQMA/poly}^{\mathsf{Un}(\mathsf{ZPP}^{\mathsf{QMA}^{\mathsf{B}}})}$.

Definition 3.4.1 ($\mathsf{BPP/samp}$ [38])
A language L is in $\mathsf{BPP/samp}$ if there exists probabilistic polynomial-time Turing machines M and D with the following properties: On input 1^n, D produces output distribution $\mathcal{D}_n$. M takes an input of size n along with 'samples' from L, $\mathcal{T} = \{(x_i, L(x_i))\}_{i=1}^{poly(n)}$, where x_i is sampled from $\mathcal{D}_n$. M outputs 1 with probability greater than $2/3$ if $x \in L$, and less than $1/3$ if $x \notin L$, where the probability taken is over the randomness in both sample selection and random coins.

The original definition of $\mathsf{BPP/samp}$ [38] is not explicit which domain the $2/3$ failure probability applies to, it could be that for most sets of samples the machine M must function 'according to the rules of BPP' (for all points there is a $2/3$ probability of failure over the coin flips), or it could be that for each point, a $2/3$ probability of failure applies over both the set of samples and the set of coin flips. While the former appears to be a much tighter restriction (the majority of all sets work on all points for most sets of coin flips), fortunately, these two definitions are equivalent via a simple boosting-and-majority-vote argument. Similarly, it is clear that the use of BPP in the definition of $\mathsf{BPP/samp}$ is superfluous as if there is randomness over the samples, this randomness is sufficient to use as randomness to simulate coin flips. If a given $\mathsf{BPP/samp}$ algorithm requires m random coins, we can ask for extra samples and use the random inputs x as random coins[2].

Theorem 3.4.5

[2]The random distribution of samples, $\mathcal{D}_n$, might not be uniform, infact there is no requirement for it to be non-deterministic. Fortunately, we can always append $m/|x|$ uniform random samples on the end of our set of samples to use as random coin flips.

$\mathsf{BPP/samp} = \mathsf{P/samp}$

For our purposes, we are interested in exactly the restriction of $\mathsf{BPP/samp}$ to samples that can be produced by a particular machine (i.e. a quantum machine/BQP). We define $\mathsf{BPP/samp^B}$ equivalently to $\mathsf{BPP/samp}$ but requiring that the labelling problem $(L(x))$ is in the complexity class B. Fortunately, this is simply $\mathsf{BPP/samp} \cap \mathsf{B}$.

Lemma 3.4.1
$\mathsf{BPP/samp^B} = \mathsf{BPP/samp} \cap \mathsf{B}$

From this definition, the following result is immediately clear:

Theorem 3.4.6
$\mathsf{BPP/samp^{BQP}} \subseteq \mathsf{BPP/rpoly^{BQP}}$

As noted by Watrous [52] randomised advice can be used to give random coins to a probabilistic algorithm, which extends to bounded advice to show $\mathsf{BPP/rpoly^{BQP}} = \mathsf{P/rpoly^{BQP}}$. This connection improves Theorem 3.4.6 to the following corollary.

Corollary 3.4.7
$\mathsf{BPP/samp^{BQP}} \subseteq \mathsf{P/rpoly^{BQP}}$

As noted above, the definition of $\mathsf{BPP/samp}$ seems to require randomised advice. Fortunately, we can use Theorem 3.4.6 to derandomise this advice, connecting $\mathsf{BPP/samp^{BQP}}$ to a deterministic bounded advice class.

Theorem 3.4.8

$$\mathsf{BPP/samp^{BQP}} \subseteq \mathsf{P/poly}^{\mathsf{Un}\left(\mathsf{ZPP}^{\mathsf{NP}^{\mathsf{BQP}}}\right)}$$

Proof. As $\mathsf{BPP/samp^{BQP}} \subseteq \mathsf{BQP}$ and $\mathsf{BPP/samp^{BQP}} \subseteq \mathsf{P/poly}$, applying Corollary 3.2.2 produces the desired result. □

Similar these techniques bound other 'quantum preparation classes', such as $\mathrm{CSIM_{QE}}$ [51].

We can now turn our attention to quantum states given as advice, the standard equivalent result in unbounded advice classes for connecting quantum advice to advice is $\mathsf{BQP/qpoly} \subseteq \mathsf{QMA} \cap \mathsf{coQMA/poly}$ [46], we show a close analogue exists for bounded advice classes:

Theorem 3.4.9

$$\mathsf{BQP/qpoly^A} \subseteq \mathsf{QMA} \cap \mathsf{coQMA/poly}^{\mathsf{ZPP}^{\mathsf{QMA}^{\mathsf{A}}}}$$

Proof. Aaronson [46] and Drucker showed:

$$\mathsf{BQP}/\mathsf{qpoly} \subseteq \mathsf{QMA} \cap \mathsf{coQMA}/\mathsf{poly}$$

As $\mathsf{BQP}/\mathsf{qpoly}^{\mathsf{A}} \subseteq \mathsf{BQP}^{\mathsf{A}}$ we can apply Theorem 3.2.1 to derive:

$$\mathsf{BQP}/\mathsf{qpoly}^{\mathsf{A}} \subseteq \mathsf{QMA} \cap \mathsf{coQMA}/\mathsf{poly}^{\mathsf{Un}(\mathsf{ZPP}^{\mathsf{QMA}^{\mathsf{QMA}\cap\mathsf{coQMA},\mathsf{BQP}^{\mathsf{A}}}})}$$

This result can be significantly simplified, first, we note that $\mathsf{QMA} \cap \mathsf{coQMA}$ is low for QMA

$$\mathsf{QMA}^{\mathsf{QMA}\cap\mathsf{coQMA},\mathsf{BQP}^{\mathsf{A}}} \subseteq \mathsf{QMA}^{\mathsf{BQP}^{\mathsf{A}}}$$

Then, we use $\mathsf{QMA}^{\mathsf{BQP}^{\mathsf{A}}} = \mathsf{QMA}^{\mathsf{A}}$ to derive the result

$$\mathsf{BQP}/\mathsf{qpoly}^{\mathsf{A}} \subseteq \mathsf{QMA} \cap \mathsf{coQMA}/\mathsf{poly}^{\mathsf{Un}(\mathsf{ZPP}^{\mathsf{QMA}^{\mathsf{A}}})}$$

□