Uniform Diagonalization Theorem for Complexity Classes of Promise Problems 
including Randomized and Quantum Classes

Friederike Anna Dziemba
Institut für Theoretische Physik, Leibniz Universität Hannover, Germany
(Dated: December 19, 2017)

Diagonalization in the spirit of Cantor’s diagonal arguments is a widely used tool in theoretical computer sciences to obtain structural results about computational problems and complexity classes by indirect proofs. The Uniform Diagonalization Theorem allows the construction of problems outside complexity classes while still being reducible to a specific decision problem. This paper provides a generalization of the Uniform Diagonalization Theorem by extending it to promise problems and the complexity classes they form, e.g. randomized and quantum complexity classes. The theorem requires from the underlying computing model not only the decidability of its acceptance and rejection behaviour but also of its promise-contradicting indifferent behaviour – a property that we will introduce as “total decidability” of promise problems.

Implications of the Uniform Diagonalization Theorem are mainly of two kinds: 1. Existence of intermediate problems (e.g. between BQP and QMA) – also known as Ladner’s Theorem – and 2. Undecidability if a problem of a complexity class is contained in a subclass (e.g. membership of a QMA-problem in BQP). Like the original Uniform Diagonalization Theorem the extension applies besides BQP and QMA to a large variety of complexity class pairs, including combinations from deterministic, randomized and quantum classes.

I. INTRODUCTION

In the seventies two contrary results gave new insight into the structure of polynomial time reducibility and the hierarchy between the complexity class P of efficiently solvable problems and the broader class NP of efficiently verifiable problems. While Schaefer’s dichotomy theorem \[\text{[1]}\] states that every naturally restricted constraint-satisfaction problem outside P belongs immediately to the hardest problems of NP (so-called NP-complete), Ladner’s theorem \[\text{[2]}\] shows for every problem outside P the existence of an intermediate problem that is strictly simpler but still outside P.

In quantum complexity theory Local Hamiltonian problems assume the role that constraint-satisfaction problems play in classical complexity theory. The categorization of Local Hamiltonian problems by \[\text{[3]}\] can be regarded as a quantum analogue of Schaefer’s dichotomy result. In contrast, no quantum version of Ladner’s theorem has been formulated until today.

The strongest generalization of Ladner’s Theorem is nowadays known as “Uniform Diagonalization Theorem” \[\text{[4, 5]}\] and is applicable to a large variety of complexity classes. By generalizing the reduction notion the authors of \[\text{[6–8]}\] even realize a useful application of the theorem to complexity classes below P. Unfortunately, the formulation of the Uniform Diagonalization Theorem only covers classes of decision problems, whereas quantum complexity classes are formed by the broader concept of promise problems. While an algorithm for a decision problem has to work correctly on every input, an algorithm for a promise problem only has to work correctly on promised inputs. Because of this “untotal” property, promise problems are often avoided and therefore neglected in theoretical computer sciences. But with the introduction of randomized complexity classes and at the latest with the upcoming of quantum computing the concept of promise problems clearly deserves more respect.

Promise problems are natural for semantic complexity classes such as randomized and quantum classes due to their probabilistic nature. The main purpose of a promise is to guarantee that an algorithm of reasonable running time can differentiate yes- and no-instances well enough, i.e. adheres the probabilistic error allowed by the definition of the complexity class. Without promises many randomized and quantum classes would not contain any canonical problems. For example, neither the randomized classes BPP and MA of efficiently solvable and verifiable problems nor the quantum analogues BQP and QMA are known to contain any complete decision problems. But they all contain complete promise problems, including problems of high physical relevance like QMA-complete Local Hamiltonian problem.

The adaption of the Uniform Diagonalization Theorem to randomized and quantum complexity classes requires two steps: After the preliminary section we will first extend some necessary terminology originally defined in the context of decision problems to the context of promise problems like (total) decidability and recursive (re-)presentation and show that these properties are obeyed by standard randomized and quantum complexity classes. In the next section we can then adapt the proof of the Uniform Diagonalization Theorem to promise problems and their complexity classes.

Informally the Uniform Diagonalization Theorem states that for two complexity classes \(C\) and \(C'\) and two problems \(A \notin C\) and \(A' \notin C'\), there exists another problem \(B\) which inherits the property not to belong to any of the two complexity classes while still being reducible to the marked union of A and A’ (which basically implies reducibility to the more difficult problem of the two). The complexity bound on B provided by this reduction is cru-
cial, since just finding a problem outside two complexity classes is clearly trivial if it can be chosen arbitrarily more difficult.

In the implications section we will show that fixing one of the complexity classes and one of the problems leads to a simplified version of the theorem in the spirit of Ladner: Given a standard complexity class and a problem $A$ outside this class, one always finds another problem that lies outside the class but is strictly simpler than $A$.

Our versions of the Uniform Diagonalization Theorem and Ladner’s Theorem apply to all previously mentioned randomized and quantum classes and also to combinations from both kinds. This means for example that there exists an infinite hierarchy of intermediate problems between QMA and BQP as well as between BQP and BPP (understood as set of promise problems) under the assumption that these classes are unequal.

A second branch of implications from the Uniform Diagonalization Theorem will cover undecidability results for subclass membership, for example: Given a problem via a QMA-description it is undecidable if it is contained in BQP.

II. PRELIMINARIES

A. Deterministic Turing machines and Properties of Functions

Let $\Sigma^*$ denote the set of all strings over the binary alphabet $\Sigma = \{0, 1\}$. Theoretical computer sciences mainly deals with two kinds of computational tasks:

1. Either computing a function $\mathbb{N}_0 \to \mathbb{N}_0$ or $\Sigma^* \to \Sigma^*$ or

2. deciding a bipartite question in form of a promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ with

\[
A_{\text{yes}} \cap A_{\text{no}} = \emptyset, \\
A_{\text{yes}} \cup A_{\text{no}} \subseteq \Sigma^*.
\]

Problems for that the last line holds an equality are called decision problems and are a special case of promise problems. The fundamental computational model for both computational tasks is the deterministic Turing machine (DTM):

A deterministic Turing machine is an endless tape machine with a reading head starting on the first symbol of the input $x \in \Sigma^*$, which is padded at both sides with infinitely many blank symbols $\square$. Moreover, a DTM comes along with a finite set of states $S$ including an initial and possibly several final states. A computational step of the machine is described by the transition function

\[
\delta : S \times \{0, 1, \square\} \to S \times \{0, 1, \square\} \times \{L, R, N\}
\]

that maps the current state and the symbol at the head position to another state, overwrites the read symbol and moves the head at most by one cell (left / right / neutral). If a final state is reached, the machine halts and outputs the string that is written between the head position and the next blank symbol.

![FIG. 1. Initial configuration of a Turing machine on input $x = 110 \ldots 1$ and after one step according to transition function $\delta(z_0, 1) = (z_1, 0, R)$.](image)

Given input $x$ we denote the output of a DTM $M$ by $M(x)$. Note, that $M(x)$ doesn’t have to be defined for all inputs, since a DTM can also run into an infinite loop and never halt. The notion of an input can be extended to a multipartite input $x_1, x_2, \ldots \in \Sigma^*$. These inputs are then written onto the tape successively separated by a blank symbol and we denote the output accordingly by $M(x_1, x_2, \ldots)$.

There is a correspondence between all strings over the binary alphabet and Turing machines via the concept of Gödel numbers. A Gödel number is a binary encoding of the transition function together with a specification of the final states. Note, that the transition function can be encoded by a binary number of finite length since it is determined by finitely many transitions. If a binary number does not have the form of a valid Gödel number it is interpreted as the encoding of a trivial machine that always outputs 0. If we state in this paper that “a Turing machine is given” we mean that the Gödel number of the machine is supplied. In the same manner a computable function is always given via the Gödel number of the machine that computes the function.

Definition 1. A promise problem $A = (A_{\text{yes}}, A_{\text{no}})$ is decidable iff there exists a DTM $M$ such that

\[
\forall x \in A_{\text{yes}} M(x) = 1 \quad ("M accepts input x") \\
\forall x \in A_{\text{no}} M(x) = 0 \quad ("M rejects input x").
\]

The concept of Gödel numbers allows the construction of the most famous undecidable decision problem: The yes-instances of the Halting problem are exactly (the Gödel numbers of) those DTMs that don’t halt given their own Gödel number as input.

Computability of a function $f : \Sigma^* \to \Sigma^*$ means that there exists a DTM that for every input $x \in \Sigma^*$ outputs $f(x)$. For functions $f : \mathbb{N}_0 \to \mathbb{N}_0$ it means that there exists a DTM that for every input $x \in \mathbb{N}_0$ outputs $f(x)$, with $f(\cdot)$ denoting the binary representation of natural numbers without leading zeros.

For the later discussion of complexity classes and the proof of the central theorem we need some terminology
relating to the runtime of a DTM. Most important, we say that a DTM $M$ has a polynomial runtime iff there exists a polynomial $p$ over $\mathbb{N}_0$ such that for all inputs $x \in \Sigma^*$ $M$ halts in less or equal to $p(|x|)$ computational steps. Two more properties of functions will become relevant in this paper; the first will mainly be used to restrict resources (runtime, witness size, etc.) in complexity class definitions, the second is important for the proof of the central theorem:

**Definition 2.** A function $f : \mathbb{N}_0 \to \mathbb{N}_0$ is in the set poly iff $f$ is polynomially bounded and there exists a DTM $M$ that for all inputs of length $n$ outputs bin $(f(n))$ in polynomial time.

**Definition 3.** A function $f : \mathbb{N}_0 \to \mathbb{N}_0$ is called time-constructible, iff there exists a DTM that for each input of length $n$ halts exactly in $f(n)$ steps.

**Lemma 1.** The following holds for a function $f : \mathbb{N}_0 \to \mathbb{N}_0$:

1. $f$ is time-constructible $\Rightarrow$ $f$ is computable.
2. $f$ is computable $\Rightarrow$ $\exists$ time-constructible $f' \geq f$.

**Proof.**
1. Let $M_{T(f)}$ be a DTM that time-constructs $f$. Then the following machine computes $f$:
   First, change the input bin($x$) on the tape to its unary representation $1^x$. On this new input of length $x$ simulate $M_{T(f)}$ with the following adaptation: Interrupt after each original computational step (by changing into a new “interruption state”), let the head run to the end of the written tape and increment a counter there starting at 0. If the originally subsequent state is a final state, let the head remain on the counter and change into the final state (i.e. the new output is the counter), otherwise let the head run back to its original position and change into the subsequent state.

2. Let $M_f$ be a DTM that computes $f$. Then the following machine time-constructs a function $f'$ with $f' \geq f$:
   Replace the input bin($x$) by its length $n = |\text{bin}(x)|$. After simulating $M_f$, write its output in unary representation $1^{f(n)}$ and halt. Since writing onto $f(n)$ cells needs at least time $f(n)$, the new machine time-constructs a function $f'$ with $f' \geq f$. □

### B. Randomized and Quantum Models

The Church-Turing thesis asserts that every reasonable computational model can be simulated by the model of a deterministic Turing machine and no contradiction has been found until today. But especially when caring about complexity it is useful to consider as well other computational models with plausible implementations. These models aim at deciding decision or promise problems and are usually not used for the computation of functions, hence, they just come with a notion of acceptance and rejection but not with a notion of a function output.

The first, the probabilistic Turing machine, simply extends the model of the deterministic Turing machine by allowing a branching at every computational step, which is mathematically reflected by a transition function of the form

$$\delta : S \times \{0, 1, \square\} \to \mathcal{P}(S \times \{0, 1, \square\} \times \{L, R, N\})$$

with $\mathcal{P}$ denoting the power set.

All terminology of DTMs is used accordingly for each branch of a PTM. The formalism of Gödel numbers can also be easily extended to PTMs. A PTM as a whole has a probabilistic expression for accepting and rejecting an input $x$ determined by the fraction of accepting and rejecting branches:

$$P_{\text{acc}}(x) = \frac{\# \text{ accepting branches}}{\# \text{ all branches}}$$
$$P_{\text{rej}}(x) = \frac{\# \text{ rejecting branches}}{\# \text{ all branches}}.$$

The fundamental computing model in quantum information are families of quantum circuits $(C_x)_{x \in \Sigma^*}$ that are made up of a series of unitary quantum gates

$$U_x = U_{x,1}, U_{x,t-1} \ldots U_{x,2}U_{x,1}$$

acting on a Hilbert space of multiple qubits usually initialized in the state $|in\rangle = |0^k\rangle$ and closed by a final projective measurement $\Pi_{\text{acc}} := |1\rangle \langle 1|_{\text{out}}$ of a designated output qubit (e.g. the first one). The probability to measure $\Pi_{\text{acc}}$ is considered the acceptance probability for the input $x \in \Sigma^*$:

$$P_{\text{acc}}(x) = \langle in|U_x^\dagger \Pi_{\text{acc}} U_x |in\rangle.$$

A useful notion of complexity classes demands that the allowed gates are of an elementary kind. In this paper we assume the standard gates H, T and CNOT that act non-trivially only on one or two qubits. In the computational basis these gates have the following matrix representation:

**Hadamard:**

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

**T-gate:**

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

**Controlled NOT:**

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$
i.e. any unitary operation can be approximated arbitrarily well by these gates.

Until now we have just described the structure of a quantum circuit but not the structure of a family of quantum circuits \((C_x)_{x \in \Sigma^*}\), whose circuits can basically look very different for different inputs \(x\). The minimum requirement on a circuit family should be the existence of an algorithm that can compute a description of the quantum circuit \(C_x\) for each input \(x \in \Sigma^*\). Strictly speaking, a quantum circuit family is defined via a deterministic Turing machine that on input \(x \in \Sigma^*\) outputs an encoding of the quantum circuit \(C_x\) in a Gödel number style, e.g. encode

as the number

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0
\end{array}
\]

H on qubit 2

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 0 & 1 & 1 & 1 & 0
\end{array}
\]

T on 1

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

CNOT from 2 to 3

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
\]

CNOT from 1 to 3

Always be aware of the difference between the Gödel number of (the DTM representing) a quantum circuit family and the Gödel number of a specific circuit that it outputs. An output of a DTM representing a quantum circuit family that is not a valid circuit Gödel number is interpreted as an encoding of the trivial circuit that never accepts. Due to the Turing machine construction procedure we will call a quantum circuit family obeying the restrictions of a complexity class \(C\) also a \(C\)-machine, e.g. a BQP-machine, though BQP-circuit family might sound more intuitive.

A polynomial-time generated quantum circuit family \((U_x)_{x \in \Sigma^*}\) is a quantum circuit family whose constructing DTM has polynomial runtime. Since a circuit’s Gödel number is always longer than the circuit size (number of gates), the size of the constructed circuit is upper bounded by the same polynomial. The bound on the circuit size due to the non-compressive property of the encoding justifies to call polynomial-time generated quantum circuit families “efficient”. Instead of the Gödel scheme presented here one can of course also assume a different circuit encoding. The specifics of the encoding are irrelevant for the definition of usual complexity classes as long the the encoding is sensible, efficient but non-compressive and guarantees that “specific information about the structure of a circuit [is] computable in polynomial time from an encoding” [9].

C. Complexity Classes and Reductions

A complexity class is any set of promise problems. Usually such a set is defined by all problems that can be decided by a certain restricted machine model and can hence be considered of similar “complexity”. We will list here the definitions of the most important complexity classes, namely those of efficiently decidable and efficiently verifiable problems for deterministic, randomized and quantum computing.

In accordance with the literature, we define the classes \(P\) and \(NP\) based on deterministic Turing machines as sets of decision problems. Historically, the first complexity classes were defined as sets of decision problems (or, equivalently, of languages, which equal the yes-instances of decision problems), since it is guaranteed that each DTM with a restricted runtime decides a decision problem – in contrast to the later defined probabilistic and quantum machines.

**Definition 4.** The complexity class \(P\) is the set of all decision problems that can be decided by a deterministic Turing machine of polynomial runtime.

**Definition 5.** The complexity class \(NP\) is the set of all decision problems \(A\) for that there exists a deterministic Turing machine \(M\) of polynomial runtime and an \(m \in \text{poly} \) such that

\[
\forall x \in A_{\text{yes}} \exists y \in \Sigma^{|x|} : (x, y) = 1,
\]

\[
\forall x \in A_{\text{no}} \forall y \in \Sigma^{|x|} : (x, y) = 0.
\]

The \(y\) in the definition of \(NP\) is called a witness. In contrast to the problem input \(x\) we will call the input \((x, y)\) the machine actually acts on the computational input (here consisting of problem input and witness).

Next, we will define the randomized and quantum analogues of \(P\) and \(NP\). With the introduction of randomized classes and at the latest with the upcoming of quantum computing, the definition of complexity classes was broadened to promise problems. While in quantum information it is common sense to understand complexity classes as sets of promise problems, classical computer scientists are still debating if randomized classes such as \(BPP\) and \(MA\) should be considered as sets of decision problems or sets of promise problems. To avoid confusion we call the first \(BPP\) and \(MA\) and the later ones Promise-BPP and Promise-MA.

**Definition 6.** The complexity class \(BPP\) (Promise-BPP) is the set of all decision (promise) problems \(A\) for that there exists a probabilistic Turing machine \(M\) of polynomial runtime such that

\[
\forall x \in A_{\text{yes}} : P_{\text{acc}} \geq \frac{2}{3},
\]

\[
\forall x \in A_{\text{no}} : P_{\text{acc}} \leq \frac{1}{3}.
\]

**Definition 7.** The complexity class \(MA\) (Promise-MA) is the set of all decision (promise) problems \(A\) for that there exists a probabilistic Turing machine \(M\) of polyno-
mial runtime and an \( m \in \text{poly} \) such that

\[
\forall x \in A_{\text{yes}} \exists y \in \Sigma^m(|x|) : P_{\text{acc}} \geq \frac{2}{3},
\]

\[
\forall x \in A_{\text{no}} \forall y \in \Sigma^m(|x|) : P_{\text{acc}} \leq \frac{1}{3}.
\]

The probability thresholds \( c = \frac{2}{3} \) and \( s = \frac{1}{3} \) are called completeness and soundness parameter of the complexity classes.

Often literature remains unclear, if runtime-bounded probabilistic Turing machines like PromiseBPP- or PromiseMA-machines have to obey the specified runtime only on promised inputs or also for non-promised inputs. Since a machine of the first kind can easily be transformed into one of the second kind (by counting the computational steps and aborting after the specified runtime with a default value as output), we assume the second case. Accordingly we assume that the following bounded-runtime generated families of quantum circuits obey their runtime specification for all inputs:

**Definition 8.** The complexity class BQP is the set of all promise problems \( A \) for that there exists a polynomial-time generated family of quantum circuits \( (U_x)_{x \in \Sigma^*} \) on \( k \) qubits such that

\[
\forall x \in A_{\text{yes}} : \langle 0^k | U_x^\dagger \Pi_{\text{acc}} U_x | 0^k \rangle \geq \frac{2}{3},
\]

\[
\forall x \in A_{\text{no}} : \langle 0^k | U_x^\dagger \Pi_{\text{acc}} U_x | 0^k \rangle \leq \frac{1}{3}.
\]

**Definition 9.** The complexity class QCMA is the set of all promise problems \( A \) for that there exists a polynomial-time generated family of quantum circuits \( (U_x)_{x \in \Sigma^*} \) on \( k + m \in \text{poly} \) qubits such that

\[
\forall x \in A_{\text{yes}} \exists y \in \Sigma^m(|x|) : \langle y | \otimes \langle 0^k | U_x^\dagger \Pi_{\text{acc}} U_x (y) \otimes | 0^k \rangle \rangle \geq \frac{2}{3},
\]

\[
\forall x \in A_{\text{no}} \forall y \in \Sigma^m(|x|) : \langle y | \otimes \langle 0^k | U_x^\dagger \Pi_{\text{acc}} U_x (y) \otimes | 0^k \rangle \rangle \leq \frac{1}{3}.
\]

**Definition 10.** The complexity class QMA is the set of all promise problems \( A \) for that there exists a polynomial-time generated family of quantum circuits \( (U_x)_{x \in \Sigma^*} \) on \( k + m \in \text{poly} \) qubits such that

\[
\forall x \in A_{\text{yes}} \exists | \psi \rangle \in \mathbb{C}^{2^m(|x|)} : \langle \psi | \otimes \langle 0^k | U_x^\dagger \Pi_{\text{acc}} U_x ( | \psi \rangle \otimes | 0^k \rangle \rangle \geq \frac{2}{3},
\]

\[
\forall x \in A_{\text{no}} \forall | \psi \rangle \in \mathbb{C}^{2^m(|x|)} : \langle \psi | \otimes \langle 0^k | U_x^\dagger \Pi_{\text{acc}} U_x ( | \psi \rangle \otimes | 0^k \rangle \rangle \leq \frac{1}{3}.
\]

QCMA and QMA can both be considered as quantum analogues of NP. While QCMA is defined with a classical witness \( y \), QMA requires a quantum witness \( | \psi \rangle \). Note, that in case of QCMA and QMA the circuit-constructing DTM should not only output the encoding of the gates but also the number of witness qubits \( m \).

In extension of the notion “decidability”, we say that the “C-machine \( M \) decides the problem \( A \)” if \( M \) is the machine for that the membership of a problem \( A \) in one of the above complexity classes \( C \) is proven. From calling a machine either a DTM or a C-machine, it will always be clear if we mean the original notion of decidability or a machine deciding \( A \) according to the definition of the complexity class \( C \).

For abbreviation reasons we denote the set of the introduced classes of promise problems by

\[
\mathcal{C} = \{ \text{PromiseBPP}, \text{PromiseMA}, \text{BQP}, \text{QCMA}, \text{QMA} \}.
\]

These complexity classes together with the corresponding classical classes of decision problems form the following hierarchy (each class contains the connected classes below):

\[
\text{QMA} \\
\text{QCMA} \\
\text{BQP} \downarrow \text{PromiseMA} \downarrow \text{PromiseBPP} \downarrow \text{NP} \downarrow \text{P}
\]

Subset relations between classes of decision problems and classes of promise problems are trivially strict when their definitions are taken seriously. For studying hierarchy relations in a reasonable way, even quantum complexity classes are sometimes considered as restricted to decision problems, e.g. when the question “Is BQP equal to P?” is asked.

Reduction notions are a useful tool for a finer and also complexity class independent comparision of problems’ complexity. A problem \( A \) that can be reduced onto a problem \( B \) is considered simpler as \( B \) since it can be decided easily having knowledge about \( B \).
**Definition 12.** A promise problem $A$ is Cook- or T-reducible to a promise problem $B$ (notation: $A \leq_P^T B$) iff $A$ can be decided by a DTM of polynomial runtime with oracle $B$ (the DTM has access to an oracle state that upon entering replaces every $x \square$, $x \in B_{\text{yes}}$, at the head position instantenously by 1 and every $x \square$, $x \in B_{\text{no}}$, by 0).

Notice, that an oracle is only allowed to be queried for elements of $B_{\text{yes}}$ and $B_{\text{no}}$. In the case of promise problems one has to ensure that the DTM does not query the oracle for any non-promised inputs of $B$.

**Lemma 2.** It holds $A \leq_P^T B \Rightarrow A \leq_P^T B$.

*Proof.* Let $f$ be the polynomial-time computable function that reduces $A$ to $B$. Then $A$ can also be solved by a polynomial-time DTM first simulating the computation of $f$ and then querying the $B$-oracle on the function output. \hfill $\square$

Both introduced reduction notions (and any other reasonable reduction notion) form a pre-order obeying reflexivity and transitivity on the set of problems. For being a partial order the antisymmetric property is missing: Problems that can be reduced onto each other can still be different.

A special role is assumed by those problems that are the most difficult ones in a complexity class:

**Definition 13.** A promise problem $A$ is called $m$-hard (T-hard) for a complexity class $C$ iff all problems in $C$ can be Karp-reduced (Cook-reduced) to $A$. If $A$ is a problem of $C$ itself, $A$ is called $m$-complete (T-complete) for the complexity class $C$.

We denote by $C_{\text{-c}}$ and $C_{\text{-cT}}$, respectively, the set of all $m$- and $T$-complete problems for $C$.

The stricter notion of Karp-reducibility is the standard reduction notion for complexity classes above P, since $B \in C$ and $A \leq_P^T B$ implies $A \in C$ for all complexity classes $C$ that can compute a polynomial-time computable function as subroutine (like all previously defined classes). The representative nature of Karp-complete problems for complexity classes above P is the reason why Karp reductions are also simply called “complete” (without the addition “$m$” or “Karp”) and those that scientists like to find for complexity classes. For most of the introduced complexity classes complete problems are known:

The table just contains one examplary complete problem for each complexity class, though for most classes several complete problems are known. For QMA these are meanwhile several dozens; for NP several hundreds. Remarkably, no complete (decision) problems are known for BPP and MA. There seems to be a strong belief in the theoretical computer science community that BPP = P and MA = NP which may also be partly due to this fact. Moreover, we see here a good justification why rather PromiseBPP and PromiseMA should be considered as the proper complexity class definitions.

| Class   | Complete problem                                      | Ref. |
|---------|-------------------------------------------------------|------|
| P       | all problems with at least one yes- and no-instance   |      |
| NP      | $k$-Satisfiability, $k \geq 3$                        | [10] |
| BPP     | ?                                                     |      |
| MA      | ?                                                     |      |
| PromiseBPP | Acceptance Ratio of DTMs                            | [11] |
| PromiseMA | Stoquastic 6-Satisfiability                         | [12] |
| BQP     | Quadratically Signed Weight Enumerator                | [13] |
| QCMA    | Ground State Connectivity                            | [14] |
| QMA     | $k$-Local Hamiltonian, $k \geq 2$                    | [15] [16] |

Let us close this section by mentioning that some promise problems $A$ are considered of such high logical or physical relevance that it is worth defining all problems reducible to $A$ as a new complexity class with an own name. The complexity class $\text{TIM} \subseteq \text{QMA}$ is such an example [13], which consists of all problems that can be reduced onto a restricted Local Hamiltonian problem of transverse Ising model form. The notion of completeness hence allows an alternative, non-machine based approach to define complexity classes.

## III. A FRAMEWORK FOR PROMISE PROBLEMS

### A. Extremal Problems and Closure Properties of Complexity Classes

**Definition 14.** A complexity class $C$ that for every promise problem $(A_{\text{yes}}, A_{\text{no}}) \in C$ also contains every subproblem $(A'_{\text{yes}}, A'_{\text{no}})$ with

$$A'_{\text{yes}} \subseteq A_{\text{yes}}$$

$$A'_{\text{no}} \subseteq A_{\text{no}}$$

is called closed under promise extension.

According to definitions [6] - [10] the classes of $C$ are closed under promise extension. Since we like to uniquely identify only one problem with a machine like in the case of P and NP we introduce the notion of extremal problems:

**Definition 15.** The promise problem with the smallest promise decided by a DTM or a C-machine $M$, $C \subseteq C$, is called the extremal problem of $M$ and denoted by $P(M)$.

Accordingly, we call the decision problem decided by a P- or NP-machine $M$ extremal and denote it by $P(M)$.

**Definition 16.** We denote by $C^*$, $C \subseteq C$ the restriction of the according complexity class to its extremal problems.
Of course, the notion of extremal problems can easily be adapted to other “machine-based” complexity classes.

One might wonder why $C^* \subset C \in C$, is not used as the proper definition for the according randomized or quantum complexity class. Indeed, logically there is no reason to artificially demand a larger promise from a problem than necessary, but practically one usually starts by defining a logically or physically interesting problem and then aims at proving membership for this problem in a certain complexity class. These proofs often involve many implication arguments and approximations to finally show that an algorithm accepts with a sufficiently high or low probability. But this does usually not rule out that the algorithm also accepts some other, non-promised instances with similar high or low probability. Even in the case of the $k$-Local Hamiltonian problem, in which the fundamental algorithm accepts with a probability that trivially relates to the promise the final amplification to achieve the standard completeness and soundness parameters involves Chernoff’s bound. Hence, even the $k$-LH problem is probably not an extremal problem according to its usual definition. But the advantage of its usual definition is that the promise on the ground energy is simply physically describable.

We stress these subleties since structural results like the Uniform Diagonalization Theorem make actually statements about the structure of machine sets and hence about extremal classes. To formulate structural results correctly we use the above notations.

We will close this subsection by defining another closure property that even applies to all the complexity classes in this paper: the closure under finite variations (c.f.v.)

**Definition 18.** A complexity class $C$ (of decision problems) is closed under finite variations (c.f.v.) iff $A \in C$ implies $B \in C$ for every (decision) problem $B$ that equals $A$ almost everywhere.

**B. Total Decidability**

Many standard literature uses the notion of decidability only for decision problems (e.g. [18]). Those who use the notion in the context of promise problems (e.g. [19, 20]), agree on the one we gave in definition 1. The disadvantage of this definition is the arbitrary behaviour of the machine on non-promised inputs. It is therefore reasonable to introduce a stricter version of decidability that we will call total decidability:

**Definition 19.** A promise problem $A = (A_{yes}, A_{no})$ is totally decidable iff there exists a DTM $M$ such that

\[
\forall x \in A_{yes} \ M(x) = 1 \\
\forall x \in A_{no} \ M(x) = 0 \\
\forall x \in \Sigma^* \ (A_{yes} \cup A_{no}) \ M(x) = 10.
\]

Obviously total decidability implies decidability and the two notions are identical in the case of decision problems. There are reasonable examples for both promise problems that are totally decidable and promise problems that are decidable but not totally decidable:

**Example 1.** The promise problem $A = (A_{yes}, A_{no})$ with

\[
A_{yes} = \{ \text{DTM with even runtime on its Gödel number} \} \\
A_{no} = \{ \text{DTM with odd runtime on its Gödel number} \}
\]

is decidable but not totally decidable.

**Proof.** Clearly, $A$ is decidable by simply counting the running time of the given machine.

The set of non-promised inputs consists of exactly those DTMs that don’t halt on their own Gödel number as input. If $A$ was totally decidable, one could hence decide the Halting problem.

The purpose of the promise of the above problem is to avoid undecidable instances, which prevents the problem from being totally decidable. The original purpose
of promise problems was however to exclude instances for that checking membership up to a certainty / accuracy required by the randomized or quantum complexity class under consideration would exceed the runtime restriction of that class. Take for example the $k$-local Hamiltonian problem: In the standard protocol \cite{15} the promised gap on the ground energy of the Hamiltonian directly relates to the acceptance probability or – if this is improved to obey the standard completeness and soundness parameter – to the runtime overhead caused by amplification.

Luckily we don’t care about runtime when asking for total decidability, hence, the extremal problems of most standard complexity classes are totally decidable:

**Lemma 3.** The extremal problems of PromiseBPP- and PromiseMA-machines are totally decidable.

**Proof.** The extremal problem of a PromiseBPP-machine can be totally decided by a DTM that simulates all branches of the PTM (which always halt per definition) and checks its fraction of accepting branches. In case of a PromiseMA-machine this algorithm simply has to be repeated for each of the possible $2^m$ witnesses.

Note, that for the correctness of the above lemma the concrete form of the runtime bound $r$ and the concrete values of the completeness and soundness parameters $c$ and $s$ are irrelevant. Since the acceptance probability of a halting PTM is always an exactly computable rational number, the statement holds accordingly for any complexity class that replaces $r$ and $c > s$ by other computable functions and in addition if one or both of the criteria $P_{\text{acc}} \geq c$ and $P_{\text{acc}} \leq s$ are changed to strict inequalities. The statement also remains valid for computable functions $c = s$ with maximally one of the criteria containing a strict inequality. These generalizations hold as well for the quantum analogue of the lemma, which we will prove next:

**Lemma 4.** The extremal problems of BQP-, QCMA- and QMA-machines are totally decidable.

**Proof.** By simulating the circuit-constructing DTM we obtain the Gödel number of a quantum circuit with gate series $U_x$ on $k$ plus possibly $m$ witness qubits. We have to differentiate if the acceptance probability $P_{\text{acc}}$ is $\leq s$, $s$, $c$ or $\leq c$ with $s = \frac{1}{4}$, $c = \frac{3}{4}$ and

$$P_{\text{acc}} := \langle 0^k | U_x \Pi_{\text{acc}} U_x | 0^k \rangle$$

in the case of BQP,

$$P_{\text{acc}} := \max_{y \in \{0,1\}^m} \langle 0^k | \otimes (|y\rangle) U_x \Pi_{\text{acc}} U_x (|0^k\rangle \otimes |y\rangle)$$

in the case of QCMA and

$$P_{\text{acc}} := \text{highest eigenvalue of } Q := \langle 0^k | U_x \Pi_{\text{acc}} U_x | 0^k \rangle$$

in the case of QMA.

We recall that the generated quantum circuit just consists of H-, T- and CNOT-gates. In case of BQP the acceptance probability $P_{\text{acc}}$ equals hence a sum of products of elements from the field

$$Q(\frac{1}{\sqrt{2}}, i)$$

(notice that the phase $e^{i\pi/4}$ of the T-gate can be written as $\frac{1}{\sqrt{2}}(1 + i)$).

This finite field extension can be handled as 4-dimensional vector space $V$ over the rational numbers with the abstract basis vectors

$$1, \frac{1}{\sqrt{2}}, i, \frac{i}{\sqrt{2}}.$$

A DTM can compute operations on the coefficients exactly by storing two integers and a sign for each rational number and it can carry out the finitely many different products of basis vectors abstractly. There is consequently a DTM that can compute $P_{\text{acc}}$ as a rational linear combination of the abstract vectors $1$ and $\frac{1}{\sqrt{2}}$ (the imaginary vector obviously vanishes for the acceptance probability).

If the coefficient of the $\frac{1}{\sqrt{2}}$-vector vanishes and the coefficient of the 1-vector equals $c$ or $s$, the DTM can accept or reject directly. Otherwise, it is clear that $P_{\text{acc}}$unequals $c$ and $s$. Since there exists methods (e.g. the babylonian / Heron method) to compute monotonous sequences of converging upper and lower bounds of square roots, the DTM can compute improving lower and upper bounds on $P_{\text{acc}}$ until their exceeding of or falling below $c$ or $s$ discloses if $P_{\text{acc}} < s$, $P_{\text{acc}} > c$ or $P_{\text{acc}} \in [s,c]$. To totally decide the extremal problem of a QCMA-circuit family the above algorithm simply has to be run for all possible witnesses $y \in \{0,1\}^m$.

In the QMA-case a DTM can do the following to totally decide the extremal problem: The evaluation of the characteristic polynomial of $Q$ at the values $c$ and $s$ and the check of Sylvester’s criterion (positivity of all principal minors $= \text{determinants of upper-left submatrices}$) for the matrices $c I - Q$ and $s I - Q$ can both be handled abstractly in the vector space $V$ as described above. A coefficient check of the abstract vectors obtained in the first case tells with certainty if $c$ and $s$ are themselves eigenvalues of $Q$. After this is ruled out an improving lower and upper approximation of the abstract vectors in the second case will disclose after finite time, if the operators $c I - Q$ and $s I - Q$ are positive or negative definite. Together, this allows to distinguish the three relevant cases for the acceptance probability.

As we discussed before, the above two lemmata hold accordingly for randomized and quantum complexity classes with different runtime bounds and completeness and soundness parameters $c$ and $s$ as long as they are computable functions. In case of quantum complexity classes one might also ask if the last lemma still holds under the assumption of a different gate set providing the basic elements of a quantum circuit. From the proof
we can see that the answer is obviously yes as long as the matrix entries of these gates are from a field extension of \( \mathbb{Q} \) with countable degree such that the product of any two basis vectors is computable abstractly and a converging lower and upper bound for each basis vector is computable.

An example for such an infinite field extension is the field of all algebraic numbers \( \mathbb{A} \) which is made up by all roots of polynomials over \( \mathbb{Q} \). Instead of writing elements of \( \mathbb{A} \) as linear combinations of abstract basis vectors, one can also identify each element directly by a unique rational polynomial and a sufficiently small isolating rectangle in the complex plane. The author of [21] shows that summation, substraction, multiplication, division and powering in this representation only needs rational computations (including the computation of resultants of polynomials which is a new polynomial whose coefficients are an expression in the coefficients of the original ones) and can hence be done exactly. The final isolating rectangle can be narrowed enough to allow the approximation of the final number to arbitrary precision, while the exact root candidates \( c \) and \( s \) can be ruled out as before by simple insertion.

One case in which a complexity class crucially changes with the form of the gate set is a one-sided complexity class such as QMA_1 which equals QMA with \( c = 1 \). To achieve this perfect completeness a QMA_1-circuit family is usually allowed to contain any matrix element from the field the problem is formulated in. The authors of [22] advocate the algebraic numbers as largest reasonable field to define QMA_1. If one naively thought to allow any field that can be “described by words”, this would instead lead to an easy construction of circuits whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field containing a Chaitin’s number whose extremal problem is not decidable at all. Consider e.g. a field conta
lems, since classes of decision problems are not closed under promise extension by definition. Reversely, recursive presentability can not only apply to classes of decision problems but also to classes of promise problems, especially to those that are restricted to extremal problems.

**Lemma 6.** The complexity classes P, NP and any C* with C ∈ C are recursively presentable.

**Proof.** All deterministic Turing machines can be simulated given their Gödel number and all polynomials over N₀ form a computable series (pᵢ)ᵢ∈N₀ since polynomials whose coefficients add up to the same sum form a finite set and these are obviously enumerable. Hence, one can define a computable series (Mᵢ)ᵢ∈N₀ for all P-machines with Mᵢ, i interpreted as pair (j, k), the P-machine simulating the DTM with Gödel number j up to runtime pᵢ (with returning a default value, if the DTM doesn’t halt on 0 or 1 in time pᵢ).

Note that by dropping or adapting the restriction on the output values we can as well obtain a computable series for all polynomial-time computable functions Σ* → Σ*. If we consider the index i even as tuple (j, k, l) and simulate the DTM with Gödel number j up to runtime pᵢ and output the minimum of the obtained output and pᵢ, we even construct a computable series (fᵢ)ᵢ∈N₀ for all functions in the set poly.

A computable series (Mᵢ)ᵢ∈N₀ of all DTMs that decide NP-problems is realized by defining Mᵢ, i interpreted as tuple (j, k, l), as the DTM that checks the acceptance probability of the DTM with Gödel number j limited to time pᵢ for each witness of length fᵢ.

By interpreting Gödel numbers as encodings of probabilistic or quantum circuit generating Turing machines we can construct in a similar way computable series of all C*-machines, C ∈ C. Since the extremal problems of these machines are totally decidable according to lemmata [3] and [4], the machines in these series can be replaced by the DTMs that totally decide their extremal problems to obtain a recursive presentation of the complexity class.

**Corollary 1.** The complexity classes of C are recursively representable.

**Corollary 2.** For quantum channel families E whose Stinespring dilation over the field of algebraic numbers is computable, the complexity class QMA_E is recursively presentable and QMA_E hence recursively representable.

Note, that we do not know how to recursively present BPP or MA. The problem of the straightforward method is that we don’t know how to decide if the extremal problem of a polynomial-time PTM is a decision problem, i.e., if the machine accepts with probability ≥ 1/2 or ≤ 1/2 on all inputs. This relates to the missing knowledge of complete problems for BPP and MA, since a complete problem would provide another possibility to prove recursive presentation, namely by enumeration of all reduction functions:

**Lemma 7.** For a totally decidable problem A the set

\[ A^≤_m := \{ \text{promise problem } B \mid B \leq^P_m A \} \]

is recursively representable.

If A is a decision problem, then the series of DTMs that recursively represent \( A^≤_m \) recursively presents all decision problems that can be m-reduced to A.

**Proof.** Let \( M_A \) the DTM that totally decides A and \((fᵢ)ᵢ∈N₀\) the computable series of all polynomial-time computable functions \( \Sigma^* \to \Sigma^* \). Then \((Mᵢ)ᵢ∈N₀\) with

\[ Mᵢ(x) = M_A(fᵢ(x)) \]

is a recursive representation (presentation) of all (decision) problems B that are reducible to (the decision problem) A.

This approach allows to prove recursive (re)presentation for complexity classes that are defined via a complete problem like TIM, even if they are missing a machine-based definition.

The enumerability of reduction functions (and polynomial-time oracle machines) can also be used to prove a result the other way around: all problems of usual complexity classes that are more difficult than a problem A are recursively presentable as well. The proof basically resembles the proof for Cook-complete decision problems from [3]:

**Lemma 8.** Let C be a recursively presentable complexity class c.f.v., and A ∈ C be totally decidable. Then

\[ A_{≤m}^≤ := \{ B ∈ C \mid A ≤^P_m B \}, \]

\[ A_{≤P}^{C} := \{ B ∈ C \mid A ≤^P_B \} \]

are recursively presentable.

**Proof.** Let \((Mᵢ)ᵢ∈N₀\) be a recursive presentation of C, \((fᵢ)ᵢ∈N₀\) the computable series of all polynomial-time computable functions \( \Sigma^* \to \Sigma^* \) and \((Oᵢ)ᵢ∈N₀\) the computable series of all polynomial-time oracle Turing machines. Let \( M_A \) be the DTM that totally decides A.

For the recursive presentation of the set \( A_{≤m}^≤ \) we define the DTM \( Nᵢ \), \( i = (j, k) \), that on input x checks for all \(|y| ≤ |x|\) if

\[ y ∈ A_{yes} \Rightarrow f_j(y) ∈ P(M_k)_{yes} \]

\[ y ∈ A_{no} \Rightarrow f_j(y) ∈ P(M_k)_{no} \]

is fulfilled. If yes, it outputs \( M_k(x) \), otherwise \( M_A(x) \).

For the recursive presentation of the set \( A_{≤P}^C \) we define the DTM \( Nᵢ \), \( i = (j, k) \), that on input x checks for all
\[ |y| \leq |x| \text{ if} \]
\[ y \in A_{\text{yes}} \Rightarrow O_j \text{ with oracle } P(M_k) \text{ on input } y \]
\[ \text{only queries the oracle for promised inputs and accepts} \]
\[ y \in A_{\text{no}} \Rightarrow O_j \text{ with oracle } P(M_k) \text{ on input } y \]
\[ \text{only queries the oracle for promised inputs and rejects} \]
is fulfilled. If yes, it outputs \( M_k(x) \), otherwise \( M_A(x) \).
\[
(N_i)_{i \in \mathbb{N}} \text{ is a recursive presentation of } A_C^{ \leq_m } (A_C^P)
\]
since \( P(N_i) = P(M_k) \) if \( P(M_k) \) for \( i = (j, k) \) is a problem of \( C \) on that \( A \) can be \( m- \text{(T-)reduced while otherwise} \)
\( P(N_i) = A \) almost everywhere.

Corollary 3. Let \( C \) be a recursively presentable complexity class c.f.v. with at least one totally decidable \( m- \text{(T-)complete problem}. \) Then \( C-c_m \) \( (C-c_T) \) is recursively presentable.

IV. PROOF OF THE UNIFORM DIAGONALIZATION THEOREM

With the extended definitions and new notations introduced in the last section the proof of the extended Uniform Diagonalization Theorem resembles the original one limited to decision problems [4, 5].

Before we actually prove the Uniform Diagonalization Theorem we first restate the definition and an efficiency condition for the so-called gap language \( G[r] \), which allows us later to mix the two problems stated in the Uniform Diagonalization Theorem by restricting them to certain alternating intervals.

Definition 21. Let \( r : \mathbb{N}_0 \to \mathbb{N}_0 \) be a computable function with \( r(m) > m \) for all \( m \in \mathbb{N}_0 \). The gap language generated by \( r \) is defined as the set
\[ G[r] := \{ x \in \Sigma^* | r^n(0) \leq |x| < r^{n+1}(0) \text{ for } n \text{ even} \} \]
with \( r^n \) denoting the \( n \)-fold concatenation of \( r \).

Lemma 9. If \( r : \mathbb{N}_0 \to \mathbb{N}_0 \) with \( r(m) > m \) is time-constructible, then \( (G[r], \overline{G[r]}) \in P \).

Proof. Compute iteratively \( r(0), r^2(0), r^3(0) \ldots \) like in the proof of lemma 1. Abort the iteration if the counter during the computation of \( r^k(0) \) reaches \( |x| \). Accept if \( k - 1 \) is even, otherwise reject.

Revisiting the proof of lemma 1 shows that the computation of \( r(m) \) is efficient in \( r(m) \) and so is an aborted simulation in the final counter. Hence every iteration step is efficient in \( |x| \). Since \( r^n(0) \geq n \), the number of iteration steps is limited by \( |x|+1 \) and the above algorithm is an efficient decision algorithm for \( (G[r], \overline{G[r]}) \).

Now, we can finally prove the Uniform Diagonalization Theorem and construct a problem \( B \) from two problems \( A \notin C \) and \( A' \notin C' \) that inherits the property \( B \notin C \cup C' \) while being Karp-reducible to the marked union \( A \oplus A' \).

Definition 22. The marked union \( A \oplus A' \) of two promise problems \( A \) and \( A' \) is defined as the promise problem \( D \) with
\[ D_{\text{yes}} = \{ 0x | x \in A_{\text{yes}} \} \cup \{ 1x | x \in A'_{\text{yes}} \} \]
\[ D_{\text{no}} = \{ 0x | x \in A_{\text{no}} \} \cup \{ 1x | x \in A'_{\text{no}} \} \].

Theorem 1 (Uniform Diagonalization Theorem). Let \( C, C' \) be complexity classes closed under finite variations of which each is recursively presentable or recursively representable. Let \( A \notin C, A' \notin C' \) be totally decidable promise problems. Then there exists a totally decidable promise problem \( B \) such that
\[ B \notin C \cup C' \text{ and } B \leq_m A \oplus A' \].

If \( A \) and \( A' \) are extremal for one of the complexity classes from \( C \) or decision problems, then so is \( B \).

Proof. Let \( M_0, M_1, M_2, \ldots \) and \( M'_0, M'_1, M'_2, \ldots \) be recursive representations (presentations) for the complexity classes \( C \) and \( C' \), respectively. Due to \( A \notin C \), every \( M_i \) does not (totally) decide correctly some input with regard to the problem \( A \) (for a recursively representable class \( C \) such a “contradicting” element can only be from \( A_{\text{yes}} \cup A_{\text{no}} \); for a recursive presentable class also instances from \( A_{\text{yes}} \cup A_{\text{no}} \) can be contradicting, namely \( M_i \) accepts or rejects). The same holds for \( C' \) and \( A' \). The construction idea for the new problem \( B \) is to “mix” \( A \) and \( A' \) such that \( B \) inherits a “contradicting” element for each \( M_i \) and \( M'_i \).

To define a valid promise problem we mix \( A \) and \( A' \) by restricting them to alternating intervals via the previously defined gap language, i.e.
\[ B_{\text{yes}} = (G[r] \cap A_{\text{yes}}) \cup (\overline{G[r]} \cap A'_{\text{yes}}) \]
\[ B_{\text{no}} = (G[r] \cap A_{\text{no}}) \cup (\overline{G[r]} \cap A'_{\text{no}}) \]
with a properly chosen function \( r : \mathbb{N}_0 \to \mathbb{N}_0 \). Clearly, in this form \( B = (B_{\text{yes}}, B_{\text{no}}) \) is a valid promise problem. The function \( r \) has to be chosen such that for each \( M_i \) the union of all even intervals, i.e. \( G[r] \), contains an element contradicting \( A \) and that for all each \( M'_i \) the union of all odd intervals, i.e. \( \overline{G[r]} \), contains an element contradicting \( A' \) (see figure 2).

We achieve this by defining the function \( q : \mathbb{N}_0 \to \mathbb{N}_0 \),
\[ q(n) := \max_{i \leq n} \{ |z_{i,n}| \} + 1 \]
with \( z_{i,n} \in \Sigma^* \) the smallest word according to the usual binary order such that \( |z_{i,n}| > n \) and
\[ z_{i,n} \in \begin{cases} A \setminus P(M_i) & \text{if } C \text{ recursively representable} \\ A \Delta P(M_i) & \text{if } C \text{ recursively presentable} \end{cases} \]
Notice that \( z_{i,n} \) always exists. Otherwise,

\[
\begin{align*}
A & \subseteq P(M_i) \text{ a.e. if } C \text{ recursively representable} \\
A & = P(M_i) \text{ a.e. if } C \text{ recursively presentable}
\end{align*}
\]

and since \( C \) is closed under finite variations (and promise extension in the first case) this implies \( A \in C \), which contradicts the hypothesis of the theorem.

The total decidability of \( A \) and the existence of \( z_{i,n} \) imply the computability of \( q \). Analogously, the function

\[
q'(n) := \max_{i \leq n} \{ |z'_{i,n}| \} + 1
\]

is computable with \( z'_{i,n} \in \Sigma^* \) defined accordingly for \( A' \) and \( M'_i \).

We now choose our desired function \( r \) as the time-constructible function

\[
r(n) \geq \max\{q(n), q'(n)\}
\]

that exists according to lemma \[\ref{lemma:constructible}\] The definitions of \( q(n) \) and \( q'(n) \) imply \( r(n) > n \). Hence, the gap language \( G[r] \) is well-defined. And since it is decidable, \( B \) is totally decidable.

Notice that the most important step of the proof is indeed to choose the time-constructible function \( r \) instead of \( q \) and \( q' \) for defining the interval jumps of the gap language. For this function lemma \[\ref{lemma:construction}\] tells us that \( (G[r], G[r]) \) \( \in P \) and hence \( f : \Sigma^* \to \Sigma^* \) with

\[
f(x) := \begin{cases} 
0x & \text{if } x \in G[r] \\
1x & \text{if } x \in \overline{G[r]}
\end{cases}
\]

is a valid Karp-reduction from \( B \) to \( A \oplus A' \). This complexity bound on \( B \) is the essence of of the Uniform Diagonalization Theorem, since just finding a problem outside two complexity classes is trivial when it can be chosen arbitrarily more difficult.

\( G[r] \in P \) also implies that \( B \) is extremal if \( A \) and \( A' \) are extremal for a complexity class \( C \in C \), since every class \( C \in C \) is capable of performing the polynomial-time decision algorithm for \( G[r] \) as initial subroutine before simulating the algorithm for \( A \) or \( A' \).

If one had chosen the interval jumps not at \( r(n) \) but exactly at \( q(n) \) and \( q'(n) \), i.e. exactly at the highest necessary contradicting element, then the determination of the interval containing an input \( x \) can be far from efficient. For this one actually would have to compute the contradicting elements which means a simulation of all machines \( M_i \) with \( i \leq n \) for which we cannot fix a general polynomial runtime bound. The trick is that \( r(n) \) is defined larger than the maximum of these contradicting elements and the runtime that it needs to compute them (recall the proof of Lemma \[\ref{lemma:nonconstructible}\]). So usually \( r(n) \gg q(n) \) and the contradicting elements in figure \[\ref{fig:mixing}\] should be drawn cumulated at the lower interval limits.

This is also the reason why the proof technique is sometimes referred to as “delayed diagonalization” \[\ref{delayed_diagonalization}\]. The check if the next interval limit lies above the input \( x \) can now not only be answered positively by the output of the iterative computation of contradicting elements but also when this computation exceeds a runtime of \( |x| \), which is obviously efficient.

After we proved the reduction statement of the theorem, it only remains to show rigorously that the defined problem \( B \) lies indeed outside the two complexity classes \( C \) and \( C' \). Assume \( B \in C \). This means there exists an \( i \in \mathbb{N}_0 \) such that

\[
B \setminus \overline{P(M_i)} = \emptyset \quad \text{if } C \text{ recursively representable} \\
B \triangle P(M_i) = \emptyset \quad \text{if } C \text{ recursively presentable}.
\]

Let \( m \) be an even integer such that \( n := r^m(0) \geq i \). As we have seen above there exists \( z_{i,n} \in [n, r(n)] \) with \( z_{i,n} \in \Sigma^* \)

FIG. 2. Mixing of the two problems \( A \) and \( A' \) including all necessary elements to push it outside \( C \) and \( C' \).
The above results on their own might not seem very intriguing, but they reveal their whole power in the following implications:

**Corollary 6.** Given a QMA-machine it is undecidable if its extremal problem is in BQP assuming BQP ⊆ QMA.

**Proof.** Assume BQP ⊆ QMA and it is decidable whether the extremal problem of a QMA-machine is in BQP. Let $M_1, M_2, ...$ be a recursive presentation of QMA*. By substituting every $M_i$ whose extremal problem is in BQP by the QMA-machine deciding the $k$-LH problem, we obtain a recursive presentation of QMA* \ BQP* which is a contradiction to corollary 4. Hence, it is undecidable if the extremal problem of a QMA-machine is in BQP.

**Corollary 7.** Given a QMA-machine it is undecidable if its extremal problem is m-complete (T-complete) under the assumption that QMA does not equal the closure of P under promise extension.

**Proof.** Analogously.

Originally [4] proved the above corollaries for combinations of the complexity classes NP, PSPACE and PH as well as complement classes such as co-NP. We omit a corresponding version here, since complement classes are not really relevant for sets of promise problems. Complement classes are relevant for decision problems since results about them often imply structural consequences, e.g., if a problem in co-NP turned out to be NP-complete, then this would imply NP = co-NP [20, 25]. But there is no analogous implication known for classes of promise problems like for QMA and its complement.
Since Ladner’s Theorem constructs the intermediate problem as a mixture of the hard problem $A$ and the constant-no problem $A'$, the series of intermediate problems between $A = k$-LH* and BQP are variants of the Local Hamiltonian problem with more and more yes-instances turned into no-instances. This is why Ladner’s original proof is also called the method of “blowing holes into the complete problem” [24].

Does this descriptive property of the intermediate problems tell us something about the difficulty of specific Local Hamiltonian instances? Unfortunately, the criteria of kicking certain Local Hamiltonian instances out of the yes-instances is far from having any physical meaning. Instead it results from the fact that a binary string is interpreted as two unrelated encodings. If a Local Hamiltonian yes-instance remains an intermediate problem’s yes-instance depends on the behaviour of certain Turing machines on certain inputs, both determined by the encoding of the Local Hamiltonian instance and the specific Gödel numbering chosen for Turing machines. Due to the large degree of freedom in both encoding schemes, the holes blown into the Local Hamiltonian problem to make it easier are rather artificial than physically meaningful.

Since recursive presentability of extremal problems also holds for all other complexity classes of promise problems introduced in this paper, it follows:

Corollary 9. The above corollaries 4, 6, 8 and Ladner’s Theorem hold accordingly for every pair of the complexity classes P, NP, PromiseBPP, PromiseMA, BQP, QCMA, QMA and the noisy QMA-variant QMA_n with $E = (E_m)_{m \in \mathbb{N}}$ a quantum channel family whose Stinespring dilation over the field of algebraic numbers is computable.

Corollaries 4 and 6 also hold for any of the beforementioned classes with a complete problem instead of QMA.

VI. DISCUSSION

We adapted the Uniform Diagonalization Theorem and its most important implications to complexity classes of promise problems and showed that standard randomized and quantum complexity classes fulfill the recursive (re)presentability property required by the Theorem. In order not to overload the paper we explicitly showed this for the classes PromiseBPP, PromiseMA, BQP, QCMA and QMA, but the argumentation is easily adaptable to most natural randomized and quantum complexity classes including classes without probability gap such as PP and classes with different time- or space-restrictions and an interactive or multi-witness extension. We also argued that recursive (re)presentability is still fulfilled if the gate set of the quantum computing model is extended to all matrices over algebraic numbers, which is relevant for partly-deterministic classes such as QMA_1.

Besides the formulation of further corollaries for specific interesting pairs of classes like [4, 5] list for decision problems, it remains the non-trivial task to adapt the here presented form of the Uniform Diagonalization Theorem to more stricter reduction notions than Karp to make the theorem applicable to randomized and quantum classes with a runtime time below polynomial. For decision problems and log-space and log-time-reductions this has been achieved by [7, 8].

While it is clear that different reduction notions are needed for complexity classes below P, complexity classes above are usually considered within the framework of Karp reductions. That this might be too strict shows the the consistency problem for local density matrices [24] which is not known to be QMA-complete under Karp but under randomized Cook reductions. For classes above P it should be legitimate to consider weaker reduction notions (e.g., randomized or quantum) as long as they are “simulable” within this class. Since these reduction notions are implied by Karp, we can simply substitute them in the formulation of the Uniform Diagonalization Theorem. Ladner’s Theorem holds for these reductions as well as long as the complexity class problems harder than the supplied problem remain recursively presentable.

Another field of investigation arising from this paper is the structure of non-extremal problems. Several questions arise that do not seem trivial to answer. To list only some of them:

- The advantage of our extremality definition is that it allows the recursive presentation of the extremal problems for standard complexity classes. The disadvantage is that the notion only applies to classes with a machine-based definition (and hence e.g. not to TIM). Alternatively, a problem could be defined as extremal for a complexity class, if the class does not contain any supproblem. Are the such defined extremal problems recursively presentable?

- Does there exists a machine-based definition for TIM and hence a reasonable notion of extremality?

- Do there exist non-complete problems that are only decidable by machines whose extremal problem is complete? Notice, that if yes, it is indeed important to state in Ladner’s Theorem that $B \in \text{QMA}^* \setminus \text{BQP}^*$. The weaker statement $B \in \text{QMA} \setminus \text{BQP}$ would be meaningless, since $B$ could simply be a nonextremal problem of QMA whose extremal supproblem is as least as difficult as $A$.

- We proved recursive presentability of $A^{\leq P}_{C \emptyset}$ and $A^{\leq P}_{C C}$ for recursively presentable complexity classes $C$. Is the analogous set for recursively representable classes also recursively presentable?

- Is every complete problem extremal?

- Are all totally decidable problems of a class $C \in C$ extremal?
ACKNOWLEDGMENTS

I thank Tobias J. Osborne for our regular discussions and Alex Grilo for several useful remarks. This work was supported by the ERC grants QFTCMPS, and SIQS, and through the DFG by the cluster of excellence EXC 201 Quantum FQ Engineering and Space-Time Research, and the Research Training Group 1991.

[1] T. Schäfer, “The complexity of satisfiability problems,” 10th ACM Symposium on Theory of computing, pp. 216–226, ACM, 1978.
[2] R. E. Ladner, “On the structure of polynomial time reducibility,” JACM, vol. 22, pp. 155–171, Jan. 1975.
[3] T. Cubitt and A. Montanaro, “Complexity classification of local hamiltonian problems,” SIAM Journal on Computing, vol. 45, no. 2, pp. 268–316, 2016. arXiv:1311.3161.
[4] U. Schöning, “A uniform approach to obtain diagonal sets in complexity classes,” Theoretical Computer Science, vol. 18, pp. 95 – 103, 1982.
[5] J. L. Balcazar, J. Diaz, and J. Gabarró, Structural Complexity I. Springer, 1995.
[6] D. Schmidt, “The recursion-theoretic structure of complexity classes,” Theoretical Computer Science, vol. 38, pp. 143 – 156, 1985.
[7] H. Vollmer, “The Gap-Language-Technique Revisited,” 4th Workshop on Computer Science Logic, pp. 389–399, Springer, 1991.
[8] K. W. Regan and H. Vollmer, “Gap-languages and log-time complexity classes,” Theoretical Computer Science, vol. 188, no. 1, pp. 101 – 116, 1997.
[9] J. Watrous, Quantum Computational Complexity, pp. 7174–7201. Springer New York, 2009. arXiv:0804.3401.
[10] S. A. Cook, “The complexity of theorem-proving procedures,” 3rd STOC ’71, (New York, USA), pp. 151–158, ACM, 1971.
[11] O. Goldreich, Theoretical Computer Science, ch. On Promise Problems: A Survey, pp. 254–290. Springer, 2006.
[12] S. Bravyi, D. P. DiVincenzo, R. I. Oliveira, and B. M. Terhal, “The Complexity of Stoquastic Local Hamiltonian Problems,” Quantum Info. Comput., vol. 8, no. 5, pp. 0361–0385, 2008. arXiv:quant-ph/0606140.
[13] E. Knill and R. Laflamme, “Quantum computing and quadratically signed weight enumerators,” Information Processing Letters, vol. 79, no. 4, pp. 173 – 179, 2001. arXiv:quant-ph/9909094.
[14] S. Ghahramian and J. Sikora, “Ground state connectivity of local hamiltonians,” Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, pp. 617–628, Springer, 2015. arXiv:1409.3182.
[15] D. Aharonov and T. Naveh, “Quantum NP - A Survey.” arXiv:quant-ph/0210077, October 2002.
[16] J. Kempe, A. Kitaev, and O. Regev, “The complexity of the local hamiltonian problem,” SIAM Journal on Computing, vol. 35, no. 5, pp. 1070–1097, 2006. arXiv:quant-ph/0406180.
[17] A. Y. Kitaev, A. H. Shen, and M. N. Vyalyi, Classical and Quantum Computation (Graduate Studies in Mathematics), vol. 47. AMS, June 2002.
[18] C. H. Papadimitriou, Computational Complexity. Addison Wesley Longman, 1994.
[19] S. Arora and B. Barak, Computational Complexity – A Modern Approach. Cambridge University Press, 2009.
[20] O. Goldreich, Computational Complexity – A Conceptual Perspective. Cambridge University Press, 2008.
[21] A. W. Strzeboński, “Computing in the field of complex algebraic numbers,” Journal of Symbolic Computation, vol. 24, no. 6, pp. 647 – 656, 1997.
[22] S. Bravyi, “Efficient Algorithm for a Quantum Analogue of 2-SAT.” arXiv:quant-ph/0602108, February 2006.
[23] F. A. Dziembra, “Robustness of QMA against witness noise,” Quantum Info. Comput., vol. 17, no. 13&14, pp. 1167–1190. arXiv:1611.07332.
[24] R. Downey and L. Fortnow, “Uniformly hard languages,” Theor. Comput. Sci., vol. 298, pp. 303–315. Apr. 2003.
[25] S. Even, A. L. Selman, and Y. Yacobi, “The complexity of promise problems with applications to public-key cryptography,” Information and Control, vol. 61, no. 2, pp. 159 – 173, 1984.
[26] Y.-K. Liu, Consistency of Local Density Matrices is QMA-Complete, pp. 438–449. Springer Berlin Heidelberg, 2006.