Taming Quantum Time Complexity

Aleksandrs Belovs[1], Stacey Jeffery[2], and Duyal Yolcu[3]

[1]Center for Quantum Computing Science, Faculty of Science and Technology, University of Latvia
[2]QuSoft, CWI & University of Amsterdam
[3]https://github.com/qudent

Quantum query complexity has several nice properties with respect to composition. First, bounded-error quantum query algorithms can be composed without incurring log factors through error reduction (*exactness*). Second, through careful accounting (*thriftiness*), the total query complexity is smaller if subroutines are mostly run on cheaper inputs – a property that is much less obvious in quantum algorithms than in their classical counterparts. While these properties were previously seen through the model of span programs (alternatively, the dual adversary bound), a recent work by two of the authors (Belovs, Yolcu 2023) showed how to achieve these benefits without converting to span programs, by defining *quantum Las Vegas query complexity*. Independently, recent works, including by one of the authors (Jeffery 2022), have worked towards bringing thriftiness to the more practically significant setting of quantum *time* complexity.

In this work, we show how to achieve both exactness and thriftiness in the setting of time complexity. We generalize the quantum subroutine composition results of Jeffery 2022 so that, in particular, no error reduction is needed. We give a time complexity version of the well-known result in quantum query complexity, $Q(f \circ g) = \mathcal{O}(Q(f) \cdot Q(g))$, without log factors.

We achieve this by employing a novel approach to the design of quantum algorithms based on what we call *transducers*, and which we think is of large independent interest. While a span program is a completely different computational model, a transducer is a direct generalisation of a quantum algorithm, which allows for much greater transparency and control. Transducers naturally characterize general state conversion, rather than only decision problems; provide a very simple treatment of other quantum primitives such as quantum walks; and lend themselves well to time complexity analysis.

Contents

1 Introduction

Since the introduction of span programs into quantum query complexity by Reichardt and Špalek [36, 33][1], the quantum query world became a nicer place to be. A span program (alternatively, a feasible solution to the dual adversary bound) is an idealised computational model, in the sense that no real device corresponds to it. Nonetheless, it has a strong connection to quantum query complexity. We can turn any quantum query algorithm into a span program,[2] or we can construct one from scratch. They can be composed as usual quantum subroutines. At the end of the day, the span program can be transformed back into a quantum query algorithm. We identify two main points of advantage of span programs compared to usual quantum subroutines.

- We call the first one *exactness*. A span program evaluates a function exactly even if the quantum query algorithm it was converted from had a bounded error. Therefore, span programs can be composed without any error reduction. The conversion from the span program to a quantum query algorithm does introduce an error, and we do *not* get an exact quantum query algorithm at the end. However, the error is introduced only *once* per the whole algorithm, and does not accumulate even in the case of many non-precise subroutines.

  This idea was the major driving force behind this line of research starting with the algorithm for the iterated NAND function [24, 5] to general iterated Boolean functions [36, 35].

- We call the second one *thriftiness*. Span programs have a natural predisposition towards accurate bookkeeping. If an execution of a subroutine happens to be cheap on a particular input, this cheaper cost will contribute to the total complexity of the whole span program. This is in contrast to the usual execution of quantum subroutines, where the maximal cost is to be paid no matter how easy the input is in a particular execution. This is because we generally cannot measure a subroutine, which may be run only in some branch of a superposition, to see whether it has ended its work or not.[3] At the end, when converting to a quantum query algorithm, we still have to account for the maximal possible complexity, but this is done only *once* per the whole algorithm, thus amortising complexities of individual subroutines.

  This point has been emphasised to a lesser extent than the exactness property, but we find it equally important. In particular, this ability results in some rather interesting super-polynomial speed-ups [42]. We consider the thriftiness property in more detail in Section 2.

In a recent paper by a subset of the authors [17], it was shown that one does not even have to change the model to obtain these benefits. It suffices to keep the model of a quantum query algorithm, and only change the way query complexity is defined. Namely, one can count total squared norm of all the states processed by the input oracle (coined quantum Las Vegas query complexity in [17] as it can be interpreted as the expected number of queries) instead of

[1] However, this particular connection is attributed to Troy Lee in the first of these two papers.

[2] Originally, this construction was non-constructive as it came from the dual of a lower bound. Section 3 of [33] contains a constructive construction for algorithms with one-sided error. This was later extended to two-sided error in [27].

[3] In *specific* cases, Ambainis' variable-time framework [2, 3, 6] allows for this particular approach. We consider variable-time framework as a piece of motivation towards importance of thus defined thriftiness. See, e.g., the results mentioned in the introduction of [6].

the number of executions of the input oracle (which is the usual definition, called Monte Carlo in [17]). Note that quantum Las Vegas query complexity is an idealised complexity measure, but it has an operational meaning: at the end of the day, a quantum Las Vegas query algorithm can be turned into a Monte Carlo query algorithm with a constant increase in complexity and introducing a small error.

A major drawback of the above results is that while accounting of query complexity is very natural and important, analysis of the more meaningful *time complexity* of the resulting algorithm can be quite difficult. While the time complexity of some nicely structured span program algorithms has been successfully analyzed (e.g. [16, 36]), in other cases, time complexity analyses have proven much more challenging, perhaps most notably in [10], where a query algorithm was given, but a time complexity analysis remained elusive for more than a decade [29].

There has been a line of work attempting to extend some of the benefits of span programs to time complexity. In [23], it was shown how span programs could be made to capture time complexity of quantum algorithms, not only query complexity. Essentially what [23] did was to extend the algorithm-to-span-program conversion of [33] so that the span program still encodes information about the gate structure of the original algorithm. Thus, this structure can be recovered when the span program is converted back into an algorithm. The resulting span programs could be manipulated in a limited number of ways. For instance, this allowed for an alternative implementation of Ambainis' variable-time search algorithm [2]. This idea was further exploited in [26] by one of the authors (in the framework of multidimensional quantum walks [29]) to give a general composition result for time complexity of quantum algorithms, similar to the query results of [17], but lacking in generality. While [26] achieved thriftiness, it failed to achieve exactness. Instead, this work assumes that subroutines have had their success probability amplified through majority voting so that errors occur with inverse polynomial probability, which results in logarithmic factor overhead.

One of the main contributions of this paper is a framework that achieves both exactness and thriftiness for quantum *time* complexity in a systematic manner. In order to do so, we introduce *transducers*. While quantum Las Vegas query complexity keeps the model of a quantum algorithm the same but changes the definition of the query complexity, transducers still keep the same model but change the definition of computation. We consider two objects: a *transducer* that is a usual quantum algorithm in a larger Hilbert space, and its *transduction action* that is an idealised transformation obtained by "additively tracing out" a part of the Hilbert space of the transducer. A simple procedure converts a transducer into an algorithm implementing its transduction action, introducing a small error in the process. Monte Carlo query complexity of the resulting algorithm is Las Vegas query complexity of the transducer. Its time complexity is a product of two things: time complexity of the transducer (considered as a usual quantum algorithm) and its *transduction complexity*, which measures the extent of the "traced-out" part of the system. Compared to span programs, which most naturally model computations of decision problems, transducers naturally model arbitrary state conversion computations, and as a transducer is itself a quantum computation, its time complexity analysis is more immediate.

For the composition results, we follow the same overall strategy as for span programs. We can transform any quantum program into a transducer, and we can compose transducers in essentially the same way as we compose quantum programs. The advantage is that both query complexity and time complexity (in the guise of transduction complexity) are composed in a thrifty manner. At the end, we convert the composed transducer back into a quantum algorithm. In parcitular, we obtain the following results:

Exact and thrifty composition of quantum algorithms: We show how to compose quan-

tum algorithms in a way that is thrifty[4] in time complexity, but also exact (Theorem 3.10). This improves on [26] in several ways.

1. Exactness: We do not need to assume the composed algorithms have inverse polynomial errors, which saves log factors in the overall complexity.
2. The results apply to composing quantum algorithms that solve general state conversion problems, whereas [26] only applies to algorithms that decide Boolean functions.
3. In addition to achieving thrifty time complexity, the composed algorithm is also thrifty in its number of queries to the input oracle.

The one way in which our results, as stated, are not more general than [26] is that [26] considers quantum algorithms that work in the variable-time model, meaning their running times can be random variables, achieved by performing intermediate partial measurements that indicate if the algorithm is finished. While variable-time algorithms are also compatible with transducers, we omit their explicit treatment here, for the sake of simplicity, and leave it for future work. We also give results that apply purely to the circuit model (without quantum random access gates (QRAGs)), but with worse complexity (Theorem 3.8). We also explicitly consider multiple layers of composition (Theorem 3.11) in the QRAG model, and its special case of iterated functions in the circuit model (Theorem 3.12).

Quantum analogue of majority voting, and time-efficient function composition: It is known that the bounded-error quantum query complexity, Q, of a composed function $f \circ g$ is $Q(f \circ g) = \mathcal{O}(Q(f)Q(g))$. This statement would be obvious with log factors, by using the standard technique of majority voting to reduce the error of each call to g to inverse polynomial, but the fact that this holds without log factors is somewhat surprising. In Section 3.6, we shed some light on this surprising result, and show a similar result for time complexity. We extend a notion from [14], called a *purifier*, to show how to take a quantum algorithm with constant error, and convert it to a transducer with any error $\varepsilon > 0$ with only *constant* overhead on the query and time complexities. This, in turn, allows us to prove a log-factor-free composition result for time complexity in the QRAG model (Theorem 3.16). Specifically, if there is an algorithm for f that makes Q queries and takes T_f additional time, and an algorithm for g that takes T_g time, then there is an algorithm for $f \circ g$ that takes time $\mathcal{O}(T_f + QT_g)$. As with the aforementioned query result, this would be obvious with log factors on the second term, but the fact that it holds without log factor overhead is surprising. This implies log factor improvements in the time complexity of quantum algorithms obtained by composition, such as [29].

We stress that in addition to our concrete results, a major contribution is conceptual. Along with bringing the beautiful existing work on quantum query complexity to the real world of quantum time complexity, transducers achieve many existing results in a much simpler and cleaner way, and we feel that they are a novel and potentially instructive way of understanding quantum algorithms.

For example, discrete-time quantum walks are an important technical tool. However, understanding their internal workings requires some background: the notion of the spectral gap, phase estimation subroutine, non-trivial spectral analysis. In Section 3.2 we use transducers to devise a very simple implementation of a quantum walk that completely avoids all this background.

Let us end this section with a few remarks on the model of computation. The previous results towards thriftiness in time complexity, [23] and [26], assumed the QRAG model in a

[4]For an idea of what we mean by thrifty composition, see (2.2) and (2.4), and the surrounding discussion of the "gold standard" for composition.

fundamental way. It is based around the quantum random access gate (QRAG), which allows index access to an array of *quantum* registers in superposition. This should not be confused with quantum random access memory (QRAM), which assumes such access to an array of *classical* registers.[5] QRAG is a stronger model than QRAM, but it is utilised in a large number of time-efficient quantum algorithms, Ambainis' element distinctness algorithm [1] being a noticeable example.

Our general attitude towards the QRAG model is one of ambivalence. On the one hand, the QRAG model is a very natural quantization of the classical RAM machine, and, thus, makes a lot of sense to study theoretically. On the other hand, the assumption that we can swap a large number of qubits in superposition in essentially constant time seems far-fetched given the current state of the art in development of quantum computers. Because of that, we have chosen to pursue both directions. We prove results involving the QRAG model, as they tend to have more natural formulations, and continue the aforementioned line of research in [23, 26], but we also design algorithms in the circuit model, which, while not being as efficient as in the QRAG model, are of significant interest.

We now give a technical overview of our results in Section 2 and Section 3, before fleshing out full details in the remaining sections.

2 Conceptual Preliminaries: Quantum Las Vegas Complexity

Quantum Las Vegas query complexity [17], also known as the total query weight [26], is a cornerstone in understanding this paper for a number of reasons. First, the definition of the query complexity of a transducer is intrinsically the Las Vegas one. Second, when converting a quantum program into a transducer, we take into account its Las Vegas query complexity. Finally, we use composition results for the Las Vegas *query* complexity as a model that we strive to achieve for *time* complexity.

This section serves as a very brief overview of the composition properties for randomised and quantum Las Vegas complexity. The main goal of this section is to highlight the connection between quantum and randomised Las Vegas complexity, and to explain the composition properties we are interested in this paper. More technical exposition of these topics will be done in Sections 4 and 9, respectively.

The randomised results in this section are folklore, and the quantum ones are either from [17] or can be obtained similarly. Both are for purely illustrative purpose and will serve as reference points to the results obtained later in the paper. The reader may choose to skip to Section 3, and to return to this section if needed.

2.1 Types of Composition

The composition property we have in mind is as follows. Assume we have an algorithm A with an oracle O', and an algorithm B that implements the oracle O'. The functional composition of the two is an algorithm $A \circ B$, where the algorithm B is used as a subroutine to process the queries made by A to O'. The subroutine B has access to some oracle O, which is also the input to $A \circ B$. For simplicity we will now assume that A has no access to O, but we will drop this assumption shortly. The main question is a bound on the complexity of $A \circ B$ in terms of complexities of A and B.

[5] This nomenclature is not completely standardised and has been a source of confusion, as different authors use the same name to refer to different models.

One particular example studied extensively both classically and quantumly is given by composed functions, i.e., functions of the form

$$f\big(g_1(z_{1,1},\dots,z_{1,m}),g_2(z_{2,1},\dots,z_{2,m}),\dots,g_n(z_{n,1},\dots,z_{n,m})\big). \tag{2.1}$$

(For simplicity of notation, we assume all the functions g_i are on m variables, which is without loss of generality.) Then, O encodes the input string z, and A evaluates the function f. Concerning B, it is a parallel composition $B = \bigoplus_i B_i$ of algorithms evaluating g_i. On query i, B returns the output of B_i, which is $g_i(z_{i,1},\dots,z_{i,m})$.

In general, the parallel composition $B = \bigoplus_i B_i$, which we also call the direct sum, has queries of the form (i,j), on which it responds with the output of B_i on query j. We assume that all B and B_i have access to the same oracle O. In the example above, all j are absent. Also, although each B_i has access to O, it only uses the substring $z_i = (z_{i,1},\dots,z_{i,m})$ thereof.

2.2 Randomised Complexity

Recall the distinction between Las Vegas and Monte Carlo randomised algorithms. A Monte Carlo algorithm is a randomised algorithm that takes at most some fixed number T of time steps, and outputs the correct answer with bounded error. The complexity of such an algorithm is simply T. This is analogous to the usual *bounded-error* quantum query model. In contrast, a Las Vegas algorithm is a randomised algorithm whose number of steps is a random variable T, and that never outputs an incorrect answer (though it may run forever). The complexity of such an algorithm is $\mathbb{E}[T]$. More generally, one might consider the Las Vegas complexity, $\mathbb{E}[T]$, of an algorithm whose running time is a random variable T, even if the algorithm has some probability of erring (that is, it is not strictly a Las Vegas algorithm).

Randomised Las Vegas complexity behaves nicely with respect to composition. Here we sketch the corresponding notions and results in order to facilitate the forthcoming introduction of the related quantum notions, and to have a reference point against which we can gauge our quantum composition results.

Let us start with functional composition. Let $T_{\mathrm{rand}}(A,O')$ be the complexity of the algorithm A on the oracle O'. Let $B(O)$ denote the action of the algorithm B on the oracle O, and $T_{\mathrm{rand}}(B,O,i)$ the complexity of $B(O)$ on query i. Denote by $p^{(i)}_{\mathrm{rand}}(A,O')$ the probability the algorithm A will give i as a query to the oracle O' at some point during the execution of the algorithm (we assume each query is given at most once). Then, it is not hard to see that the total complexity of $A\circ B$ on oracle O is given by

$$T_{\mathrm{rand}}(A\circ B,O) = T_{\mathrm{rand}}\big(A,B(O)\big) + \sum_i p^{(i)}_{\mathrm{rand}}\big(A,B(O)\big)\cdot T_{\mathrm{rand}}(B,O,i). \tag{2.2}$$

Let us rewrite this formula to illuminate transition to the quantum case. Define the total query vector of the algorithm A as a formal linear combination

$$q_{\mathrm{rand}}(A,O') = \sum_i p^{(i)}_{\mathrm{rand}}(A,O')\; e_i$$

for e_i the elements of the standard basis. Let by definition the complexity of B on such a vector be

$$T_{\mathrm{rand}}\Big(B,O,\sum_i p_i e_i\Big) = \sum_i p_i\, T_{\mathrm{rand}}\big(B,O,i\big). \tag{2.3}$$

Then, we can rewrite (2.2) as

$$T_{\mathrm{rand}}(A\circ B,O) = T_{\mathrm{rand}}\big(A,B(O)\big) + T_{\mathrm{rand}}\big(B,O,q_{\mathrm{rand}}(A,B(O))\big), \tag{2.4}$$

which says that the complexity of the composed program on the oracle O equals the complexity of A by itself plus the complexity of B on its total query vector. This equation will serve as our "gold standard" of thriftiness in functional composition.

Similarly, thriftiness holds for parallel composition. Let $\xi = \sum_{i,j} \xi_{i,j} e_{i,j}$ be a query vector for $\bigoplus_i B_i$. We can break it down as

$$\xi = \bigoplus_i \xi_i, \tag{2.5}$$

where $\xi_i = \sum_j \xi_{i,j} e_{i,j}$ is the corresponding query vector for the constituent B_i. Then, we have

$$T_{\mathrm{rand}}\Big(\bigoplus_i B_i, O, \xi\Big) = \sum_i T_{\mathrm{rand}}\big(B_i, O, \xi_i\big). \tag{2.6}$$

(We assume here that relaying from B to the corresponding B_i is done instantly.)

These results can be combined in various ways. For instance, assume the oracle O' in the functional composition settings is a direct sum $O' = \bigoplus_{i=1}^{n} O^{(i)}$ of r independent oracles $O^{(i)}$. Let B_i implement $O^{(i)}$, so that $B = \bigoplus_i B_i$ implements O'. Similarly as in (2.5), we can decompose the corresponding total query vector

$$q_{\mathrm{rand}}(A, O') = \bigoplus_i q^{(i)}_{\mathrm{rand}}(A, O')$$

into partial query states corresponding to the i-th oracle. In this way, the multiple-oracle case differs from the single-oracle case only by this change of perspective. Combining (2.4) and (2.6), we obtain

$$T_{\mathrm{rand}}(A \circ B, O) = T_{\mathrm{rand}}\big(A, B(O)\big) + \sum_i T_{\mathrm{rand}}\big(B_i, O, q^{(i)}_{\mathrm{rand}}(A, B(O))\big). \tag{2.7}$$

2.3 Quantum Complexity

The usual definition of quantum query algorithms does not allow for different complexities on different inputs, hence, we cannot even properly define a meaningful analogue of (2.2). However, it is possible for *Las Vegas* complexity of quantum query algorithms defined in [17]. We will demonstrate here that it possesses properties essentially identical to those of the randomised case. For a more formal definition of the model of query algorithms and the Las Vegas complexity, refer to Section 4.1.

Let $A = A(O)$ be a quantum algorithm in space $\mathcal{H}$ with an oracle O in space $\mathcal{M}$. We denote by $A(O)$ the action of A on a specific input oracle O. We use $T(A)$ to denote time complexity of A, and $Q(A)$ to denote the number of queries made by A.

What is important, is that the queries to the input oracle O are conditional. This means that the query applies O to a subspace $\mathcal{H}^\bullet \subseteq \mathcal{H}$ of the workspace $\mathcal{H}$, and is the identity elsewhere. The total query state $q(A, O, \xi)$ records the history of all the queries given by the algorithm A to the input oracle O on the initial state ξ. In other words, it is the direct sum

$$q(A, O, \xi) = \bigoplus_{t=1}^{Q(A)} \psi_t^\bullet = \sum_{t=1}^{Q(A)} |t\rangle |\psi_t^\bullet\rangle,$$

where $\psi_t^\bullet \in \mathcal{H}^\bullet$ is the state given to the input oracle on the t-th query. We have $q(A, O, \xi) \in \mathcal{E} \otimes \mathcal{M}$ for some space $\mathcal{E}$. The Las Vegas query complexity is defined as $L(A, O, \xi) = \|q(A, O, \xi)\|^2$. This definition has an operational meaning: the algorithm can be modified to use $\mathcal{O}(L)$ queries, where L is the worst-case Las Vegas query complexity, by introducing some small error.

One convention of this paper is that we usually only allow *unidirectional* access to O. The algorithm can only execute O, but not its inverse O^*. This is without loss of generality, as *bidirectional access* to O, when the algorithm can execute both O and O^*, is equivalent to unidirectional access to $O \oplus O^*$.

As in the randomised case, r input oracles can be combined into one as follows:

$$O = O^{(1)} \oplus O^{(2)} \oplus \cdots \oplus O^{(r)}. \tag{2.8}$$

The *partial query state* $q^{(i)}(A, O, \xi)$ of the i-th input oracle is defined as the direct sum of all the states given to that particular oracle $O^{(i)}$. In particular, $q(A, O, \xi) = \bigoplus_i q^{(i)}(A, O, \xi)$. Similarly as before, the Las Vegas query complexity of the i-th input oracle is $L^{(i)}(A, O, \xi) = \|q^{(i)}(A, O, \xi)\|^2$.

Parallel Composition The parallel composition is straightforward. For programs $B_1, \ldots, B_n$, all on the input oracle O, its direct sum is $\bigoplus_i B_i$, which executes B_i on orthogonal parts of the space. It is not hard to show that

$$q\Big(\bigoplus_i B_i, O, \bigoplus_i \xi_i\Big) = \bigoplus_i q(B_i, O, \xi_i), \tag{2.9}$$

where we implicitly assume the correct arrangement of the entries in the corresponding direct sums. A direct consequence is the following counterpart of (2.6):

$$L\Big(\bigoplus_i B_i, O, \bigoplus_i \xi_i\Big) = \sum_i L(B_i, O, \xi_i).$$

Functional Composition Let us now derive a quantum query analogue of (2.4). First, though, we define a counterpart of (2.3). For $\xi' \in \mathcal{E} \otimes \mathcal{H}$, we can write $\xi' = \xi_1 \oplus \xi_2 \oplus \cdots \oplus \xi_m$ with $\xi_t \in \mathcal{H}$, and define

$$q(A, O, \xi') = \bigoplus_t q(A, O, \xi_t). \tag{2.10}$$

This is precisely the total query state we will get if we tensor-multiply A by the identity in the register $\mathcal{E}$ and execute it on ξ'. The corresponding Las Vegas query complexity is

$$L(A, O, \xi') = \|q(A, O, \xi')\|^2. \tag{2.11}$$

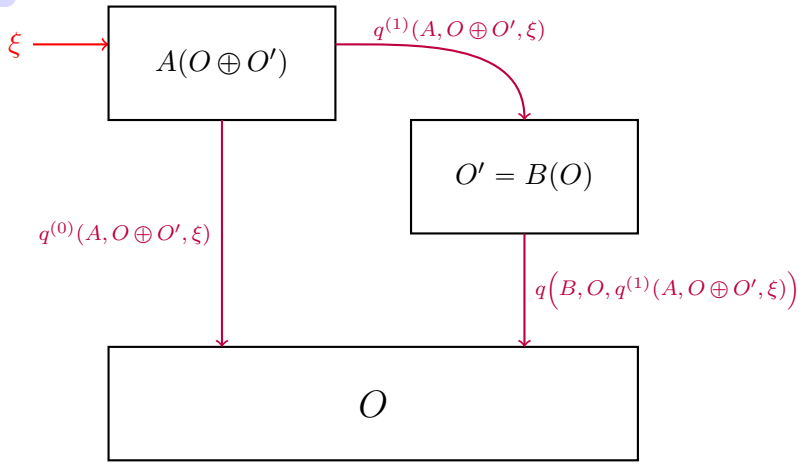
Now consider Figure 2.1, which depicts composition of two quantum programs, and which goes along the lines of our previously discussed randomised case. This time, however, we consider a more general case when A has access to the input oracle O as well.

The composed algorithm $A \circ B$ is implemented by replacing each execution of O' by an execution of B. Its action on the input oracle O is equal to $A\big(O \oplus B(O)\big)$. It is not hard to show that

$$q(A \circ B, O, \xi) = q^{(0)}\Big(A, O \oplus B(O), \xi\Big) \;\oplus\; q\Big(B, O, q^{(1)}\big(A, O \oplus B(O), \xi\big)\Big). \tag{2.12}$$

Similarly to (2.4), this slightly complicated expression represents a very intuitive observation that the total query state of $A \circ B$ on the input oracle O consists of the part of the query state of A given directly to O (denoted $q^{(0)}$), together with the query state of B on the initial state composed of the part of the query state of A given to O' (denoted $q^{(1)}$).

Figure 2.1



General case of functional composition of two programs. The outer program A has two oracles O and O', which we identify with the oracle $O \oplus O'$. The oracle O' is implemented by a program B with access to O. The diagram specifies the corresponding query states, where we use the upper indices $^{(0)}$ and $^{(1)}$ in relation to O and O', respectively.

It is quite often the case, that the oracle O' above is composed of several input oracles, each implemented by its own subroutine B_i, see Figure 2.2. In this case, we can use (2.9) to obtain an analogue of (2.7):

$$q(A \circ B, O, \xi) = q^{(0)}\Big(A, O \oplus B(O), \xi\Big) \oplus \bigoplus_i q\Big(B_i, O, q^{(i)}\big(A, O \oplus B(O), \xi\big)\Big). \tag{2.13}$$

It is often convenient to define $L_{\max}(B, O)$ as the worst-case complexity of $L(B, O, \xi)$ as ξ ranges over all unit vectors (or over all unit vectors in some admissible subspace of initial vectors). Then, using linearity, we can obtain from (2.12):

$$L(A \circ B, O, \xi) = L^{(0)}\Big(A, O \oplus B(O), \xi\Big) + L\Big(B, O, q^{(1)}\big(A, O \oplus B(O), \xi\big)\Big) \tag{2.14}$$

$$\leqslant L^{(0)}\Big(A, O \oplus B(O), \xi\Big) + L_{\max}(B, O) L^{(1)}\big(A, O \oplus B(O), \xi\big). \tag{2.15}$$

and from (2.13)

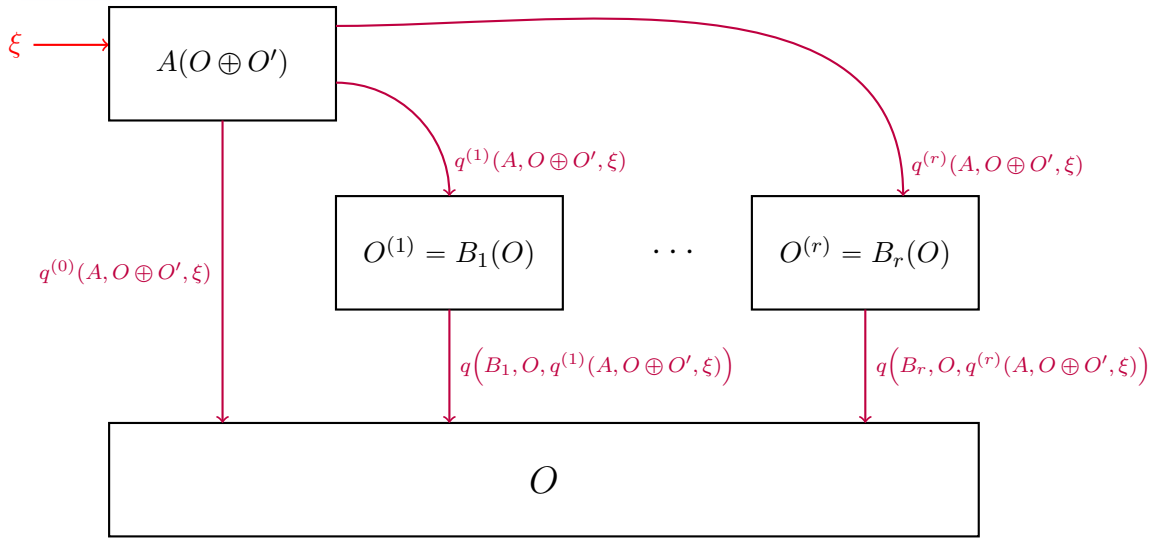
$$L(A \circ B, O, \xi) = L^{(0)}\Big(A, O \oplus B(O), \xi\Big) + \sum_i L\Big(B_i, O, q^{(i)}\big(A, O \oplus B(O), \xi\big)\Big) \tag{2.16}$$

$$\leqslant L^{(0)}\Big(A, O \oplus B(O), \xi\Big) + \sum_i L_{\max}(B_i, O) L^{(i)}\big(A, O \oplus B(O), \xi\big). \tag{2.17}$$

In (2.15) and (2.17), it is assumed that B and B_i are only executed on the admissible initial states.

To summarise, we see that quantum Las Vegas query complexity satisfies composition properties very similar to the "gold standard" of the randomised Las Vegas complexity. One of the goals of this paper is to approach these results for quantum *time* complexity. In [26], a result in this direction was obtained for evaluation of functions assuming the QRAG model of computation. In this paper, we consider more general state conversion settings, and also obtain partial results for the circuit model. We also think that the approach of this paper is less technical than the one taken in in [26].

Figure 2.2



A version of Figure 2.1 where the oracle O' is decomposed into r input oracles $O' = O^{(1)} \oplus \cdots \oplus O^{(r)}$ and the program B is accordingly decomposed into $B = B_1 \oplus \cdots \oplus B_r$ so that B_i implements $O^{(i)}$. The diagram specifies the corresponding query states, where we use the upper index $^{(0)}$ in relation to O. We note that it is without loss of generality to assume that A and all B_i use the same input oracle O. Indeed, if this is not the case, we can define O as the direct sum of the oracles used by A and B_i.

3 Overview of the Paper

This section serves as an informal version of the whole paper, where we introduce all the main concepts, ideas, and sketch the proofs of the main results. In the remaining paper, we fill in all the technical gaps.

3.1 Transducers

In the current paper, we take a different approach to time complexity than in the two papers [23, 26] mentioned in Section 1. Instead of using span programs or quantum walks, we build on the key technical primitive from [17], which we call a *transducer*[6] in this paper.

Transducers are based on the following mathematical observation we prove in Section 5.1:

Theorem 3.1 (Transduction)**.** *Let S be a unitary acting in a direct sum of two vectors spaces $\mathcal{H} \oplus \mathcal{L}$. For every $\xi \in \mathcal{H}$, there exist a unique $\tau = \tau(S, \xi) \in \mathcal{H}$ and in some sense unique $v = v(S, \xi) \in \mathcal{L}$ such that*

$$S\colon \xi \oplus v \mapsto \tau \oplus v. \tag{3.1}$$

We say in the setting of (3.1) that S *transduces* ξ into τ, denoted $\xi \overset{S}{\rightsquigarrow} \tau$ or $S\colon \xi \rightsquigarrow \tau$. This defines a mapping $\xi \mapsto \tau$ on $\mathcal{H}$, which turns out to be unitary. We call it the *transduction action* of S on $\mathcal{H}$, denoted by $S^{\curlywedge}_{\mathcal{H}}$. See Figure 3.1 for a schematic depiction. The motivation behind this terminology is that while S does *not* literally map ξ into τ, having S is a legitimate

[6] Not to be confused with transducers from the theory of finite automata, however there are some connections between the two, as we discuss in Section 4.7.

and fruitful way of implementing $S^{\xi}_{\mathcal{H}}$ on a quantum computer, as we show shortly. If a unitary S is designed primarily with this application in mind, we call it a *transducer*, and say that it *implements* $S^{\xi}_{\mathcal{H}}$.

Figure 3.1

Schematic depiction of transducers. To the left is the real action of S, which is interpreted as the action of $S^{\xi}_{\mathcal{H}}$ on $\mathcal{H}$.
Note that parallel wires here denote direct sum of the corresponding subspaces, not tensor product. The same applies to the other figures in this paper.

Let us note that this construction is not new. It has appeared before as an additive trace in the category of isometries in finite-dimensional Hilbert spaces [8, 7]. Here we demonstrate that this construction has an operational meaning. It would be interesting to understand the connection between our construction and the one taken in these two references.

We will stick to the following terminology. We call $\mathcal{H}$ the *public* and $\mathcal{L}$ the *private* space of S. We say that the transducer S is *on* the space $\mathcal{H}$, but works *in* the space $\mathcal{H} \oplus \mathcal{L}$. Also, we will call ξ the *initial state* of S, while $\xi \oplus v$ is the *initial coupling*. We call v the *catalyst* of the transduction (3.1) because it helps in the transformation of ξ into τ, but is not changed in the process. The role of the catalyst is similar to the role of the witness in span programs. In particular, the *transduction complexity* of the transducer S on an initial vector $\xi \in \mathcal{H}$ is given by its size:

$$W(S, \xi) = \|v(S, \xi)\|^2. \tag{3.2}$$

Let $T(S)$ denote the usual time complexity of implementing S as a unitary. As a rule of thumb, among various transducers S with the same transduction action, there is a trade-off between W and T so that the product

$$(1 + W(S, \xi)) \cdot T(S) \tag{3.3}$$

stays approximately the same. The importance of this product can be readily seen from the following result.

Theorem 3.2 (Implementation of Transducer)**.** *Let spaces $\mathcal{H}, \mathcal{L}$, and parameters $W, \varepsilon > 0$ be fixed. There exists a quantum algorithm that ε-approximately transforms ξ into $S^{\xi}_{\mathcal{H}}\xi$ for all transducers $S\colon \mathcal{H} \oplus \mathcal{L} \to \mathcal{H} \oplus \mathcal{L}$ and initial states $\xi \in \mathcal{H}$ such that $W(S, \xi) \leqslant W$. The algorithm conditionally executes S as a black box $K = \mathcal{O}(1 + W/\varepsilon^2)$ times, and makes $\mathcal{O}(K)$ other elementary operations.*

Since S generally takes at least one elementary operation, the complexity of the algorithm is dominated by the executions of S, which takes time (3.3) up to constant factors (assuming $\varepsilon = \Theta(1)$). The term 1 in the definition of K is required as we can have non-trivial transducers with $W = 0$, see, e.g., Section 3.4. Also, as follows from the discussion in [17], the dependence on ε is optimal.

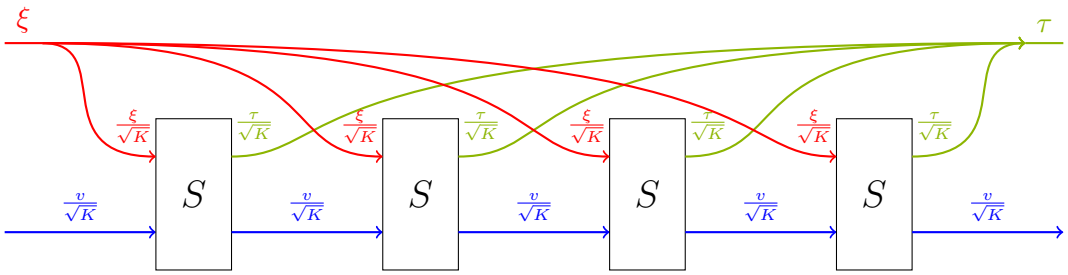
Proof sketch of Theorem 3.2. We are given a copy of ξ, and our goal is to transform it into $\tau = S^{\xi}_{\mathcal{H}}\xi$ using S as a black box. Assume we are additionally given a copy of v (in the sense of

direct sum, *cf.* Figure 3.1). Then, we can perform the required transformation $\xi \oplus v \mapsto \tau \oplus v$ using S.

There are two problems here. First, the algorithm is not given v, and, second, v can have a huge norm. The second problem is solved by breaking ξ down into K copies of $\xi/\sqrt{K}$, that is, performing the transformation $\xi \mapsto \sum_{t=0}^{K-1} |t\rangle|\xi\rangle/\sqrt{K}$. The key idea is that we can use the same scaled down catalyst $v/\sqrt{K}$ to perform K scaled down transductions $\xi/\sqrt{K} \rightsquigarrow \tau/\sqrt{K}$ as $v/\sqrt{K}$ does not change in the process. See Figure 3.2, for an illustration.

The first problem is solved by "guessing" $v/\sqrt{K}$, i.e., using that ξ is close to $\xi \oplus v/\sqrt{K}$ if K is sufficiently large. The larger the value of K, the smaller the error imposed by guessing, but the larger the number of executions of S. For a formal proof, see Section 5.2. □

Figure 3.2



A graphical illustration of the construction of Theorem 3.2. The initial state ξ is broken down into $K = 4$ copies of $\xi/\sqrt{K}$, which are sequentially transformed into $\tau/\sqrt{K}$ using only one copy of the scaled-down catalyst $v/\sqrt{K}$.

3.2 Connection to Quantum Walks

In this section, we take a short detour, and inspect the connection between transducers and quantum walks. Like transducers, quantum walks [39] replace the desired transformation with some other transformation that is easier to implement: one iteration of the quantum walk. See Figure 3.3 for an informal comparison between transducers and quantum walks, on which we will elaborate in this section.

Figure 3.3

Transducer	$\longleftrightarrow$	Quantum Walk
Execution of S	$\longleftrightarrow$	One Iteration R_2R_1
Transformation $S^{\xi}_{\mathcal{H}}$	$\longleftrightarrow$	Accept/reject of the initial state
$W(S, \xi)$	$\longleftrightarrow$	Spectral Gap
Theorem 3.2	$\longleftrightarrow$	Phase Estimation

An informal correspondence between transducers and quantum walks. The main point of this comparison is to give some intuition about the roles of different objects from Section 3.1 by linking them to objects with similar functions in the context of quantum walks. This comparison is purely indicative.

In this paper, we consider broadly interpreted discrete-time quantum walks. We identify the two characteristic properties of such algorithms, where the first one is essential, and the second one is usual, but not, strictly speaking, necessary.

The first, essential property is that one iteration of a quantum walk is a product of two reflections R_1 and R_2. The quantum walk either rejects the initial state ξ, when it is close to an eigenvalue-1 eigenvector of R_2R_1; or accepts it, when ξ is mostly supported on eigenvectors with eigenvalues far from 1. The most standard implementation of quantum walks is a phase estimation of the product R_2R_1 on the initial state ξ. The analysis of quantum walks involves spectral analysis, sometimes assisted by the effective spectral gap lemma [30].

The second, optional property is that each reflection, R_1 and R_2, is broken down as a product of local reflections that act on pairwise orthogonal subspaces. This allows for their efficient implementation. The walk is usually described using a bipartite graph (like in Figure 6.1), where each edge corresponds to a portion of the space. Local reflections are given by vertices, and they act on the direct sum of spaces corresponding to their incident edges. The reflection R_1 executes all the local reflections for one part of the bipartite graph, and R_2 for the second. The two reflections, interleaving, transcend locality and form an involved global transformation.

Because of the second property, quantum walks find a large number of algorithmic applications. This includes such basic primitives as Grover's algorithm [25] and amplitude amplification [19]; as well as the element distinctness algorithm [1], Szegedy quantum walks [41] and their various extensions, span programs [34], learning graphs [11], and others.

We will show that quantum walks are very often transducers of the following form. Let $\mathcal{H}$ and $\mathcal{L}$ be the public and the private spaces, so that the initial state $\xi \in \mathcal{H}$. The transducer is the iteration of the walk: $S = R_2R_1$, where we additionally assume that the second reflection R_2 acts trivially on $\mathcal{H}$. If ξ is negative, we have the following chain of transformations:

$$\xi \oplus v \xmapsto{R_1} \xi \oplus v \xmapsto{R_2} \xi \oplus v, \tag{3.4}$$

certifying that $\xi \overset{S}{\rightsquigarrow} \xi$. In the positive case, we have the following sequence of transformations:

$$\xi \oplus v \xmapsto{R_1} -\xi \oplus -v \xmapsto{R_2} -\xi \oplus v, \tag{3.5}$$

certifying that $\xi \overset{S}{\rightsquigarrow} -\xi$.

Both sequences follow the standard practice of designing quantum walks. In the negative case, $\xi \oplus v$ is a stationary vector of both R_1 and R_2, and, hence, R_2R_1. In the positive case, $\xi \oplus v$ is the witness for the Effective Spectral Gap Lemma. The transduction vantage point unites these asymmetric positive and negative analyses. An interesting artefact of this construction is that the transduction action of the corresponding quantum walk is exact: It transduces ξ to either ξ or $-\xi$ exactly.

We will illustrate this construction in more detail in Section 6 by re-proving the main result of [12] on electric quantum walks. However, the same applies to any quantum walk that adheres to the same design principles, including algorithms derived from span programs [13, Section 3.4], and more generally, algorithms of the type formally defined in [29, Section 3.1].

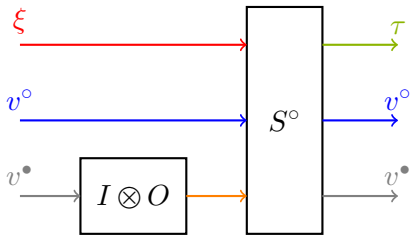
To epitomise, we keep the iteration of the quantum walk intact, but replace the wrapping phase estimation by the algorithm of Theorem 3.2. In this way, we significantly simplify the construction by abandoning *any* spectral analysis both from the implementation and the analysis of the algorithm. We believe this is worthy from the pedagogical point of view, as the corresponding algorithms now require very little background knowledge.

3.3 Input Oracle and the Canonical Form

Our previous discussion in Section 3.1 did not consider the input oracle. In this paper, we assume an approach similar to that of quantum query algorithms, where oracle executions and the remaining operations are separated. Moreover, unlike the algorithms, it suffices to have one query for a transducer.

We define the *canonical* form of a transducer $S = S(O)$ with the input oracle O in Figure 3.4. The private space $\mathcal{L} = \mathcal{L}^\circ \oplus \mathcal{L}^\bullet$ is decomposed into the work part $\mathcal{L}^\circ$ and the query part $\mathcal{L}^\bullet$, with the imposed decomposition $v = v^\circ \oplus v^\bullet$ of the catalyst v. The query is the very first operation, and it acts only on $v^\bullet$. It is followed by a unitary S° without queries.

Figure 3.4



A schematic depiction of a transducer in the canonical form. It consists of one application of the oracle O and an input-independent unitary S°. The catalyst $v \in \mathcal{L}$ is separated into two parts $v = v^\circ \oplus v^\bullet$ with $v^\circ \in \mathcal{L}^\circ$ and $v^\bullet \in \mathcal{L}^\bullet$. The first one is not processed by the oracle, and the second one is. Note that the input oracle is not applied to the public space.

Canonical transducers are easier to deal with, and every transducer can be converted into the canonical form (see Proposition 10.4). We will generally assume our transducers are canonical. We write $W(S, O, \xi)$ instead of $W\big(S(O), \xi\big)$ for the transduction complexity $W(S, O, \xi) = \|v\|^2 = \|v^\circ\|^2 + \|v^\bullet\|^2$. Also, the *total query state* is defined by $q(S, O, \xi) = v^\bullet$, and the *query complexity* by $L(S, O, \xi) = \|v^\bullet\|^2$.

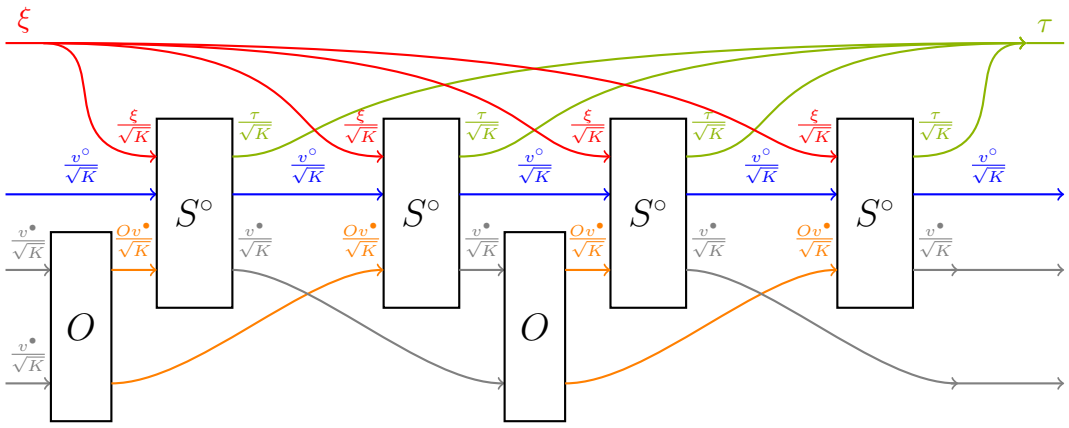
This definition is compatible with the case when O is combined of several input oracles like in (2.8). In this case, we define the *partial query state* $q^{(i)}(S, O, \xi)$ as the state processed by the oracle $O^{(i)}$, and $L^{(i)}(S, O, \xi) = \big\|q^{(i)}(S, O, \xi)\big\|^2$.

The following result, which justifies the name "query complexity", is proven in Section 7.3:

Theorem 3.3 (Query-Optimal Implementation of Transducer)**.** *Let spaces $\mathcal{H}, \mathcal{L} = \mathcal{L}^\circ \oplus \mathcal{L}^\bullet$ be fixed. Moreover, assume the transducer uses $r = \mathcal{O}(1)$ input oracles combined as in (2.8). Let $\varepsilon, W, L^{(1)}, \dots, L^{(r)} > 0$ be parameters. Then, there exists an algorithm that conditionally executes S° as a black box $K = \mathcal{O}(1 + W/\varepsilon^2)$ times, makes $\mathcal{O}(L^{(i)}/\varepsilon^2)$ queries to the i-th input oracle $O^{(i)}$, and uses $\mathcal{O}(K)$ other elementary operations. The algorithm ε-approximately transforms ξ into $\tau(S, O, \xi)$ for all S, $O^{(i)}$, and ξ such that $W(S, O, \xi) \leqslant W$ and $L^{(i)}(S, O, \xi) \leqslant L^{(i)}$ for all i.*

Proof Sketch. Let us first consider the special case of $r = 1$ for simplicity. The crucial new idea compared to Theorem 3.2 is that we guess not one but some D copies of the state $v^\bullet$. They are all processed by one oracle call, and then gradually given to S°, see Figure 3.5. Thus, the transduction complexity becomes $\|v^\circ\|^2 + D\|v^\bullet\|^2$, but we have to execute the input oracle only once in every D iterations. The correct choice of D is around $\|v^\circ\|^2/\|v^\bullet\|^2$, so that the norms of the query and the non-query parts of the catalyst become equalised. If there are $r = \mathcal{O}(1)$ input oracles, we perform the same procedure for all of them. The total transduction complexity grows by a factor of r, which is tolerable by our assumption of $r = \mathcal{O}(1)$. □

Figure 3.5



A graphical illustration of the construction of Theorem 3.3 with the same parameters as in Figure 3.2, and $D = 2$. We write O instead of $I \otimes O$ to save space.
Note that one oracle execution and D subsequent executions of S° form a transducer of its own.

This theorem does not hold for superconstant values of r, in which case a more technical Theorem 7.1 should be used. Note how canonicity of the transducer S is used here. Indeed, the input oracle is executed only once in each execution of the transducer, reducing the total number of oracle calls.

The definition of the canonical form is inspired by the implementation of the adversary bound for state conversion from [17]. Moreover, as we show in Section 8.1, the adversary bound is essentially equivalent to the above construction with $\mathcal{L}^\circ$ being empty. Also, we show in Section 8.2 how to implement the usual adversary bound for function evaluation:

Theorem 3.4 (Adversary Bound)**.** *For every function $f\colon D \to [p]$ with $D \subseteq [q]^n$, there exists a canonical transducer S_f with input oracle O_x encoding the input string x, such that, for every $x \in D$, S_f transduces $|0\rangle \rightsquigarrow |f(x)\rangle$ on the input oracle O_x, and*

$$W\big(S_f, O_x, |0\rangle\big) = L\big(S_f, O_x, |0\rangle\big) \leqslant \mathrm{Adv}^{\pm}(f),$$

where $\mathrm{Adv}^{\pm}(f)$ is the adversary bound of f, defined in Section 8.2.

We can draw the following parallels with span programs. It is known that the dual adversary bound for Boolean functions is equivalent to a very special case of span programs [33]. General span programs provide more flexibility, and thus are more suitable for time-efficient implementations [16, 36, 28, 23]. While it is possible to implement the dual adversary bound time-efficiently [4], the constructions are more complicated.

Span programs sometimes model more general function evaluation [9]; but the dual adversary has been extended much further to include arbitrary state conversion [30] with general unitary input oracles [14]. Transducers treat state conversion with unitary input oracles very naturally. Adding the non-query space $\mathcal{L}^\circ$ provides more flexibility compared to the dual adversary, which again is beneficial for time-efficient implementations. It is also of great help that transducers are quantum algorithms themselves, which makes time analysis especially straightforward.

Summarising, there are *three* basic complexity measures associated with a transducer:

- Time complexity $T(S)$, which is independent of the input oracle O and the initial state. Similarly to usual quantum algorithms, the precise value of $T(S)$ depends on the chosen model of quantum computation.
- Transduction complexity $W(S, O, \xi)$. It is defined mathematically, and does not depend on the model. On the other hand, it depends on both the input oracle and the initial state.
- Query complexity $L(S, O, \xi)$. It is also defined mathematically, and does not depend on the model. The query state $q(S, O, \xi)$ provides more information.

Let us finish this section with a few technicalities. Similarly to (2.10) and (2.11), we extend the above definitions to $\xi' = \xi_1 \oplus \cdots \oplus \xi_m \in \mathcal{E} \otimes \mathcal{H}$ via

$$q(S, O, \xi') = \bigoplus_t q(S, O, \xi_t), \quad L(S, O, \xi') = \sum_t L(S, O, \xi_t), \quad \text{and} \quad W(S, O, \xi') = \sum_t W(S, O, \xi_t). \tag{3.6}$$

Again, these can be interpreted as the complexities of the transducer $I_{\mathcal{E}} \otimes S$, see Corollary 9.7.

If for $\xi \in \mathcal{H}$ we use a catalyst v, for $c\xi$ with $c \in \mathbb{C}$, we can use the catalyst cv. This yields

$$W(S, O, c\xi) = |c|^2 W(S, O, \xi), \quad q^{(i)}(S, O, c\xi) = c\, q^{(i)}(S, O, \xi), \quad L^{(i)}(S, O, c\xi) = |c|^2 L^{(i)}(S, O, \xi). \tag{3.7}$$

It often makes sense to define the subspace $\mathcal{H}_{\mathrm{good}} \subseteq \mathcal{H}$ of admissible initial vectors to the transducer $S(O)$, and define $W_{\max}(S, O)$ as the supremum of $W(S, O, \xi)$ as ξ ranges over *unit* vectors in $\mathcal{H}_{\mathrm{good}}$. We define $L_{\max}$ and $L^{(i)}_{\max}$ similarly. We say that the initial state $\xi' \in \mathcal{E} \otimes \mathcal{H}$ is admissible if it lies in $\mathcal{E} \otimes \mathcal{H}_{\mathrm{good}}$. For any such state, by (3.6) and (3.7), we have

$$W(S, O, \xi') \leqslant W_{\max}(S, O) \|\xi'\|^2 \qquad \text{and} \qquad L(S, O, \xi') \leqslant L_{\max}(S, O) \|\xi'\|^2. \tag{3.8}$$

3.4 Transducers from Quantum Algorithms

In this section, we briefly explain how we achieve one of the points on our agenda: conversion of arbitrary quantum algorithms into transducers. We consider both the QRAG model mentioned at the end of Section 1, and the usual circuit model. While the stronger QRAG model allows for better and more intuitive exposition, we are still able to get some useful results in the circuit model.

Let

$$A(O) = G_T G_{T-1} \cdots G_2 G_1 \tag{3.9}$$

be a quantum algorithm, where G_i are individual elementary operations (gates), which also include queries to the input oracle. We call the mapping $i \mapsto G_i$ the *description* of A. The number of elementary operations $T = T(A)$ is the time complexity of the algorithm.

Trivial Transducer On the one extreme of the trade-off (3.3) is the trivial transducer $S = A$. In this case, the catalyst $v = 0$, hence, $W(S, O, \xi) = 0$, and we get that (3.3) equals $T(A)$, as expected. Of course, this does not require any change of the model.

QRAG Transducer The other extreme of the trade-off (3.3) is covered by the following construction, which assumes the QRAG model. Write the sequence of states the algorithm (3.9) goes through on the initial state ξ:

$$\xi = \psi_0 \xmapsto{G_1} \psi_1 \xmapsto{G_2} \psi_2 \xmapsto{G_3} \cdots \xmapsto{G_T} \psi_T = \tau. \tag{3.10}$$

We utilize the following history state:

$$v = \sum_{t=1}^{T-1} |t\rangle|\psi_t\rangle. \tag{3.11}$$

And define the transducer S_A as follows:

$$\xi \oplus v = \sum_{t=0}^{T-1} |t\rangle|\psi_t\rangle \xmapsto{S_A} \sum_{t=1}^{T} |t\rangle|\psi_t\rangle = \tau \oplus v, \tag{3.12}$$

where, in S_A, we first apply G_t conditioned on the first register having value t, and then increment t by one. The first operation is possible assuming the QRAG model and QRAM access to the description of A, see Section 4.5. The transduction action of $S_A(O)$ is exactly $A(O)$, and $W(S_A, O, \xi) = (T-1)\|\xi\|^2$, which we will simplify to $T\|\xi\|^2$ for brevity.

Comments on Time Complexity But what is the time complexity $T(S_A)$? In S_A, we perform two operations: increment a word-sided register and execute one instruction from the program specified by the address t. If this were a modern randomised computer, both operations would be elementary and took $\mathcal{O}(1)$ time. From the theoretical side, this is captured by the notion of RAM machine. If we consider the scale of individual qubits instead of word-sided registers, the increment operation takes time $\mathcal{O}(\log T)$, and the second operation at least as much. In order to simplify the following discussion, let us assume that both operations take some time we denote T_{R}. That is

$$T_{\mathrm{R}}\text{: time required to perform basic word operations, including random access.} \tag{3.13}$$

For simplicity, we do not discriminate between different word operations. We may think of T_{R} as being 1, or $\mathcal{O}(\log T)$, but either way, it is some fixed factor which denotes transition from the circuit model, where A operates,[7] to the QRAG model, where S_A is implemented.

Canonical Form Neither the first, nor the second transducer above are in the canonical form. The latter is given by the following result, which we prove in Section 10.

Theorem 3.5. *For a quantum program A on the input oracle O, there exists a canonical transducer $S_A = S_A(O)$, whose transduction action is identical to A, and whose complexity is given by Figure 3.6. In the QRAG model, we assume QRAM access to the description of the program A.*

Proof sketch. The constructions in both circuit and the QRAG models are already sketched above. We describe here how they can both be transformed into the canonical form so that the query state $v^\bullet$ from Figure 3.4 becomes equal to the total query state $q(A, O, \xi)$ of the algorithm.

For the trivial transducer, the catalyst is the total query state, which is all processed by one query to the oracle. Whenever the algorithm was making a query, the transducer switches the query state with the corresponding part of the catalyst, thus simulating a query.

For the QRAG transducer, the catalyst (3.11) already contains of all the intermediate states of the program. The transducer can then apply the oracle to all of them in one go, and then proceed as before without making a single additional query. □

[7] It is the usual assumption that A is a bona fide quantum circuit, but we may also assume that A uses QRAG gates.

Figure 3.6

		Circuit Model	QRAG model
Time	$T(S_A)$	$\mathcal{O}(T(A))$	$\mathcal{O}(T_{\mathrm{R}})$
Transduction	$W(S_A, O, \xi)$	$L(A, O, \xi)$	$T(A)\|\xi\|^2$
Query state	$q(S_A, O, \xi)$	$q(A, O, \xi)$	$q(A, O, \xi)$

Complexity of canonical transducers derived from a quantum algorithm in terms of complexities of the algorithm. Both the circuit and the QRAG model versions are considered.

Query Compression One consequence of the above results is that we can compress the number of queries of a quantum algorithm A to match its worst-case Las Vegas query complexity. The following result is an immediate corollary of Theorems 3.5 and 3.3.

Theorem 3.6 (Query Compression)**.** *Assume $A = A(O)$ is a quantum algorithm with $r = \mathcal{O}(1)$ input oracles as in (2.8). Let $\varepsilon, L^{(1)}, \dots, L^{(r)} > 0$ be parameters. There exists a quantum algorithm $A' = A'(O)$ with the following properties:*

- *It makes $\mathcal{O}(L^{(i)}/\varepsilon^2)$ queries to the i-th input oracle $O^{(i)}$.*
- *For every normalised initial state ξ and every input oracle $O = O^{(1)} \oplus O^{(2)} \oplus \cdots \oplus O^{(r)}$ as in (2.8), we have $\|A(O)\xi - A'(O)\xi\| \leqslant \varepsilon$ as long as $L^{(i)}(A, O, \xi) \leqslant L^{(i)}$ for all i.*
- *In the QRAG model and assuming QRAM access to the description of A, its time complexity is $\mathcal{O}(T_{\mathrm{R}} \cdot T(A)/\varepsilon^2)$.*
- *In the circuit model, its time complexity is $\mathcal{O}(L \cdot T(A)/\varepsilon^2)$, where $L = 1 + L^{(1)} + \cdots + L^{(r)}$.*

This improves over the analogous result from [17] in two respects. First, it essentially preserves the time complexity of the algorithm in the QRAG model, while [17] did not consider time complexity at all. Second, it allows multiple input oracles, as long as its number is bounded by a constant, while [17] only allowed for a single input oracle.

Proof of Theorem 3.6. First, obtain the transducer S_A as in Theorem 3.5. Then, apply Theorem 3.3 to S_A.

One key observation is that $q(S_A, O, \xi) = q(A, O, \xi)$ for all O and ξ. Hence, S_A and A have the same partial Las Vegas query complexities on all O and ξ. This yields the statement on the number of queries bounded by $L^{(i)}/\varepsilon^2$.

In the QRAG model, $W(S_A, O, \xi) = T(A)$ and $T(S_A) = \mathcal{O}(T_{\mathrm{R}})$. This gives the required runtime, as we can use that $T(A) \geqslant 1$ to remove the additive 1 factor.

In the circuit model, we use that $T(S_A) = \mathcal{O}\big(T(A)\big)$ and

$$W(S_A, O, \xi) = L(A, O, \xi) = \sum_i L^{(i)}(A, O, \xi) \leqslant \sum_i L^{(i)}.$$

We take care of the additive 1 factor by adding it explicitly to L. This covers the extreme case of L being too small, in particular, 0. □

3.5 Composition of Transducers

Transducers can be composed just like usual quantum algorithms: we consider parallel, sequential, and functional composition of transducers. Thus, from the design point of view, there is little difference between dealing with quantum algorithms and transducers. The advantage is

that in all these composition modes, the resources are more tightly accounted for than is the case for traditional quantum algorithms.

We will first sketch the composition results for transducers. We will completely omit the case of sequential composition from this section. It suffices to say, that it is very similar to the transformation of programs in Section 3.4. After that, we will give few applications both in the circuit and the QRAG models. Unlike other subsections of this section, we will be able to give complete proofs for most of the results.

Composition of Transducers The parallel composition of transducers is straightforward. They are just implemented in parallel as usual quantum algorithms. We have the following relations, akin to (2.6) and (2.9):

$$W\Big(\bigoplus_i S_i, O, \bigoplus_i \xi_i\Big) = \sum_i W(S_i, O, \xi_i) \qquad \text{and} \qquad q\Big(\bigoplus_i S_i, O, \bigoplus_i \xi_i\Big) = \bigoplus_i q(S_i, O, \xi_i). \tag{3.14}$$

For the time complexity of implementing $\bigoplus_i S_i^\circ$, we can say precisely as much as for usual quantum algorithms. In some cases it is easy: when all S_i° are equal, for example. Also, it can be efficiently implemented assuming the QRAG model, see Corollary 4.4. In general, however, direct sum in the circuit model can take as much time as the total complexity of all S_i° together.

The functional composition of transducers parallels Figure 2.1, where we replace the programs A and B with transducers S_A and S_B, respectively. The functional composition of the two is a transducer $S_A \circ S_B$ whose transduction action on the oracle O is equal to the transduction action of S_A on the oracle $O \oplus O'$, where O' is the transduction action of $S_B(O)$. The following result, proven in Section 9.5, parallels (2.12).

Proposition 3.7 (Functional Composition of Transducers)**.** *The functional composition $S_A \circ S_B$ can be implemented in the following complexity, where we use the extended notion of complexity from (3.6).*

- *Its transduction complexity satisfies*

$$W(S_A \circ S_B, O, \xi) = W(S_A, O \oplus O', \xi) + W\Big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\Big). \tag{3.15}$$

- *Its total query state is*

$$q(S_A \circ S_B, O, \xi) = q^{(0)}(S_A, O \oplus O', \xi) \oplus q\Big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\Big). \tag{3.16}$$

- *Its time complexity is the sum of the (conditional) time complexities of S_A and S_B.*

Here $q^{(0)}$ and $q^{(1)}$ denote the partial query states of S_A to the oracles O and O', respectively.

One can see that (3.15) and (3.16) meet the form of the "gold standard" of (2.4). The second one strongly resembles (2.12). One unfortunate deviation from this is the time complexity, which afterwards gets multiplied by the transduction complexity in (3.3). But since the dependence is additive, we can generally tolerate it, see, e.g., Theorem 3.12 below.

Let us state, for the ease of future referencing, a number of simple consequences of Proposition 3.7 similar to (2.13)–(2.17). First, if all the queries to S_B are admissible, we have by (3.15) and (3.8):

$$W(S_A \circ S_B, O, \xi) \leqslant W(S_A, O \oplus O', \xi) + W_{\max}(S_B, O) L^{(1)}\big(S_A, O \oplus O', \xi\big). \tag{3.17}$$

Also, from (3.16):

$$L(S_A \circ S_B, O, \xi) = L^{(0)}(S_A, O \oplus O', \xi) + L\Big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\Big) \tag{3.18}$$

$$\leqslant L^{(0)}(S_A, O \oplus O', \xi) + L_{\max}(S_B, O) L^{(1)}\big(S_A, O \oplus O', \xi\big). \tag{3.19}$$

In the case of multiple input oracles, similar to Figure 2.2, with $O' = \bigoplus_i O^{(i)}$ and $S_B = \bigoplus S_{B_i}$, so that S_{B_i} implements $O^{(i)}$, we have by (3.14):

$$W(S_A \circ S_B, O, \xi) = W(S_A, O \oplus O', \xi) + \sum_i W\Big(S_{B_i}, O, q^{(i)}(S_A, O \oplus O', \xi)\Big) \tag{3.20}$$

$$\leqslant W(S_A, O \oplus O', \xi) + \sum_i W_{\max}(S_{B_i}, O) L^{(i)}\big(S_A, O \oplus O', \xi\big), \tag{3.21}$$

$$q(S_A \circ S_B, O, \xi) = q^{(0)}(S_A, O \oplus O', \xi) \oplus \bigoplus_i q\Big(S_{B_i}, O, q^{(i)}(S_A, O \oplus O', \xi)\Big) \tag{3.22}$$

and

$$L(S_A \circ S_B, O, \xi) = L^{(0)}(S_A, O \oplus O', \xi) + \sum_i L\Big(S_{B_i}, O, q^{(i)}(S_A, O \oplus O', \xi)\Big) \tag{3.23}$$

$$\leqslant L^{(0)}(S_A, O \oplus O', \xi) + \sum_i L_{\max}(S_{B_i}, O) L^{(i)}\big(S_A, O \oplus O', \xi\big). \tag{3.24}$$

In (3.21) and (3.24), we used (3.8) in assumption that all the queries to S_B and S_{B_i} are admissible.

Composition of Programs Let us give a few examples of composition of quantum programs using transducers. We focus on the general state conversion here, evaluation of function postponed till Section 3.6. We consider both the circuit and the QRAG models.

Theorem 3.8 (Composition of Programs, circuit model)**.** *Assume the settings of Figure 2.2, where $r = \mathcal{O}(1)$ and all A and B_i are in the circuit model. Define $B = \bigoplus_i B_i$, and let $\varepsilon, L^{(1)}, \dots, L^{(r)} > 0$ be parameters.*

Then, there exists a quantum algorithm $A' = A'(O)$ such that $\big\|A'(O)\xi - A\big(O \oplus B(O)\big)\xi\big\| \leqslant \varepsilon$ for every normalised ξ as long as $L^{(i)}(A, O \oplus B(O), \xi) \leqslant L^{(i)}$ for all $i \geqslant 1$. The program A' can be implemented in the circuit model in time

$$\mathcal{O}\left(\frac{L \cdot T(A) + \sum_i T(B_i) L^{(i)}}{\varepsilon^2}\right), \tag{3.25}$$

where $L = 1 + L^{(1)} + \dots + L^{(r)}$.

Proof. Use the circuit version of Theorem 3.6 for the program A, where we treat calls to O as ordinary operations (in other words, we assume A has r input oracles $O^{(1)}, \dots, O^{(r)}$). After that, replace each call to $O^{(i)}$ by the execution of B_i. □

It turns out that it is not efficient to use Proposition 3.7 here as this would increase time complexity of the resulting transducer.

Remark 3.9. In Theorem 3.8, the emphasis is on the time complexity. If we want to simultaneously bound query complexity, we treat O as the input oracle in A as well. This gives time complexity (3.25) with $L = 1 + L^{(0)} + L^{(1)} + \dots + L^{(r)}$ and the total number of queries

$$\mathcal{O}\left(\frac{L^{(0)} + \sum_i Q(B_i) L^{(i)}}{\varepsilon^2}\right),$$

where $Q(B_i)$ is the number of queries made by B_i. We additionally require that $L^{(0)}\big(A, O \oplus B(O), \xi\big) \leqslant L^{(0)}$.

Theorem 3.10 (Composition of Programs, QRAG model)**.** *Assume the settings of Figure 2.2. Let $\varepsilon, T > 0$ be parameters. Assuming the QRAG model and QRAM access to an array with description of A and all B_i, there exists a quantum algorithm $A' = A'(O)$ with time complexity $\mathcal{O}(T_{\mathrm{R}} \cdot T/\varepsilon^2)$ such that, for every normalised ξ, we have $\|A'(O)\xi - A\big(O \oplus B(O)\big)\xi\| \leqslant \varepsilon$ as long as*

$$T(A) + \sum_{i=1}^{r} T(B_i) L^{(i)}(A, O \oplus B(O), \xi) \leqslant T. \tag{3.26}$$

The algorithm makes $\mathcal{O}(L/\varepsilon^2)$ queries to the input oracle O, where L is an upper bound on $L(A \circ B, O, \xi)$, given by (2.16).

Proof. Convert A and all the B_i into transducers S_A and S_{B_i} as in Theorem 3.5. We obtain the transducer $S_B = \bigoplus_i S_{B_i}$ for B. Then compose $S_A \circ S_B$ using Proposition 3.7. By definition, its transduction action is identical to $A(B)$.

Since the transduction complexity of S_A on a normalised initial state is bounded by $T(A)$ and that of S_{B_i} by $T(B_i)$, we get from (3.21) that for a normalised ξ:

$$W(S_A \circ S_B, O, \xi) \leqslant T(A) + \sum_i T(B_i) L^{(i)}\big(A, O \oplus B(O), \xi\big).$$

The main reason this construction is efficient in the QRAG model is that the time complexity of the transducers stays bounded by $\mathcal{O}(T_{\mathrm{R}})$ the whole time. Indeed, such is the time complexity of the individual transducers obtained from A and $B^{(i)}$. Parallel composition can be performed efficiently in the QRAG model (Proposition 10.6), and the time complexity of the functional composition is the sum of its constituents.

The transducers S_A and S_{B_i} have the same query states as A and B_i, respectively. By (3.22), we get that the total query state of $S_A \circ S_B$ is identical to that of $A \circ B$, which is given by (2.13). The statement of the theorem follows from Theorem 3.3. □

Observe that Theorem 3.10, while assuming a stronger model, gives a stronger result than Theorem 3.8. The differences are as follows. First, we do not have to assume that $r = \mathcal{O}(1)$. Second, the $L^{(i)}(A, O \oplus B(O), \xi)$ in (3.26) are the actual values of the Las Vegas query complexity, while $L^{(i)}$ in (3.25) are only upper bounds on them. This can be important if $L^{(i)}(A, O \oplus B(O), \xi)$ heavily fluctuates over different input oracles O. Finally, the $T(A)$ term is oddly multiplied by L in (3.25).

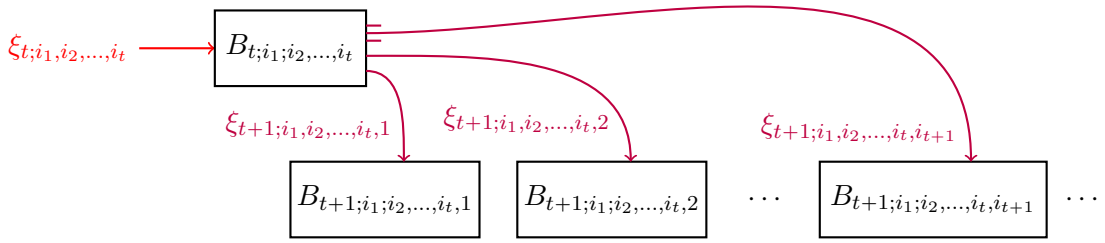
Altogether, the expression in (3.26) is more natural. It is also similar to the one in Section 1.2 of [26]. Our result is more general though, as it covers arbitrary state conversion, and not only function evaluation (we can assume $\varepsilon = \Omega(1)$ in Theorem 3.10 as it gives bounded-error evaluation of a function).

On the other hand, if the estimates $L^{(i)}$ are sufficiently precise, and $T(A)$ is smaller than average $T(B^{(i)})$, the expression in (3.25) is quite close to (3.26).

Multiple Layers of Composition Here we assume the QRAG model of computation. Theorems 3.8 and 3.10 considered one layer of composition. In the case of multiple layers, similarly as for the span programs, it is advantageous to perform all the compositions in the realm of transducers, and to transform the resulting transducer back into an actual algorithm only at the very end.

Consider a composition tree of quantum subroutines. The top layer is the algorithm B_0 that has several subroutines of the form $B_{1,i}$, like B_i used to be for A in Figure 2.2. We define the

Figure 3.7



One node $B_{t;i_1,i_2,\ldots,i_t}$ in the composition tree. Its initial state (in the general sense of (3.6)) is given by $\xi_{t;i_1,i_2,\ldots,i_t}$. It has several subroutines of the form $B_{t+1;i_1;i_2,\ldots,i_t,i_{t+1}}$ with the corresponding query state $\xi_{t+1;i_1;i_2,\ldots,i_t,i_{t+1}}$.

tree downwards so that, in general, a subroutine $B_{t;i_1,i_2,\ldots,i_t}$ has several subroutines of the form $B_{t+1;i_1,i_2,\ldots,i_t,i_{t+1}}$, see Figure 3.7. Let d be the maximal value of t in $B_{t;i_1,i_2,\ldots,i_t}$. It is the depth of the composition tree. We assume all the subroutines have access to some common oracle O. Define the composition $B = B(O)$ of the whole tree in the obvious inductive way, so that the initial state ξ of the composed algorithm is the initial state ξ of B_0.

It is not hard to get the Las Vegas query complexity of B using (2.13) inductively or from the general principles. Let $q_{t;i_1,i_2,\ldots,i_t}(O,\xi)$ be the query state given by $B_{t;i_1,i_2,\ldots,i_t}$ to the input oracle O when B is executed on the initial state ξ. Then,

$$q(B,O,\xi) = \bigoplus_{t;i_1,\ldots,i_t} q_{t;i_1,i_2,\ldots,i_t}(O,\xi). \tag{3.27}$$

We obtain a similar result for the time complexity. Let $\xi_{t;i_1,i_2,\ldots,i_t}(O,\xi)$ be the total query state given *to* the subroutine $B_{t;i_1,i_2,\ldots,i_t}$. In particular, $\xi_0(O,\xi) = \xi$.

Theorem 3.11 (Composition Tree)**.** *Assuming the QRAG model and QRAM access to the description of all $B_{t;i_1,\ldots,i_t}$ as above, there exists a quantum algorithm $B' = B'(O)$ with time complexity $\mathcal{O}(T_{\mathrm{R}}\cdot(d+1)\cdot T/\varepsilon^2)$ such that, for every ξ, we have $\|B'(O)\xi - B(O)\xi\| \leqslant \varepsilon$ as long as*

$$\sum_{t;i_1,\ldots,i_t} T(B_{t;i_1,i_2,\ldots,i_t})\|\xi_{t;i_1,i_2,\ldots,i_t}(O,\xi)\|^2 \leqslant T, \tag{3.28}$$

where the summation is over all the vertices of the composition tree. The algorithm makes $\mathcal{O}(L/\varepsilon^2)$ queries to O, where L is an upper bound on the Las Vegas query complexity of B as obtained from (3.27).

Proof. We use the induction on d to show that, under the assumptions of the theorem, there exists a transducer S_B whose transduction action is identical to B, whose transduction complexity is given by the left-hand side of (3.28), and whose time complexity is $\mathcal{O}\big((d+1)\cdot T_{\mathrm{R}}\big)$.

The base case is given by $d = 0$, where we only have B_0, which we transform into a transducer using Theorem 3.5. The transduction complexity of S_{B_0} on a normalised initial state is $T(B_0)$. Hence on the initial state ξ_0, the transduction complexity is $T(B_0)\|\xi_0\|^2$ by (3.7).

Assume the theorem is true for depth d. For the depth $d+1$, we treat the nodes $B_{t;i_1,i_2,\ldots,i_t}$ with $t \leqslant d$ as forming a composition tree A with depth d, and the nodes $B_{d+1;i_1,i_2,\ldots,i_d,i_{d+1}}$ as input oracles to A. In other words, A has an input oracle $O \oplus O'$ with

$$O' = \bigoplus_{i_1,\ldots,i_d,i_{d+1}} B_{d+1;i_1,\ldots,i_d,i_{d+1}}(O).$$

We use the induction assumption to obtain a transducer S_A for the composition tree A, whose transduction complexity on $O \oplus O'$ and ξ is given by the sum in (3.28), where we restrict the sum to $t \leqslant d$. We convert each $B_{d+1;i_1,\dots,i_d,i_{d+1}}$ into a transducer and join them via direct sum to obtain a transducer $S_{B_{d+1}}$ whose transduction action on the oracle O is identical to O'. Then, we apply Proposition 3.7 to get the transducer $S_B = S_A \circ S_{B_{d+1}}$. Its transduction complexity on O and ξ is given by the left-hand side of (3.28) with all the terms involved, where the term

$$T(B_{d+1;i_1,i_2,\dots,i_d,i_{d+1}}) \|\xi_{d+1;i_1,i_2,\dots,i_d,i_{d+1}}(O,\xi)\|^2$$

is the contribution of $B_{d+1;i_1,i_2,\dots,i_d,i_{d+1}}$.

All the transducers in $S_{B_{d+1}}$ can be implemented in parallel due to the QRAG assumption (see Proposition 10.6), hence, its time complexity is $\mathcal{O}(T_{\mathrm{R}})$. Thus, $T(S_B) = T(S_A) + T(S_{B_{d+1}}) = \mathcal{O}\big((d+1)T_{\mathrm{R}}\big)$. Finally, using (2.13) and (3.22), we get that S_B and B have the same query state. The statement of the theorem again follows from Theorem 3.3. □

Iterated Functions Due to the discussion after Theorem 3.8, one might think that several layers of composition are difficult for the circuit model. However, this is not the case if the composition tree is sufficiently homogenous. As an example, we consider evaluation of iterated functions in the circuit model.

Let $f\colon [q]^n \to [q]$ and $g\colon [q]^m \to [q]$ be total functions. The *composed function* $f \circ g\colon [q]^{nm} \to [q]$ is defined by

$$\begin{aligned}(f \circ g)(z_{1,1},\dots,z_{1,m},\ \ z_{2,1},\dots,z_{2,m},\ \ \dots\dots,\ \ z_{n,1},\dots,z_{n,m}) \\ = f\big(g(z_{1,1},\dots,z_{1,m}), g(z_{2,1},\dots,z_{2,m}),\dots,g(z_{n,1},\dots,z_{n,m})\big),\end{aligned} \tag{3.29}$$

which is equivalent to (2.1) with all the inner functions being equal. The function composed with itself several times is called *iterated function*. We use the following notation $f^{(1)} = f$ and $f^{(d+1)} = f^{(d)} \circ f = f \circ f^{(d)}$.

Iterated functions have been studied before both classically [40, 37, 38, 32, 31] and quantumly [24, 5, 35, 36]. For the case of Boolean functions, an essentially optimal algorithm was given by Reichardt and Špalek in [35, 36] using span programs.[8] It is based on the use of span programs. Similar results for the general case of non-Boolean functions can be easily obtained using composition of transducers. While it is true that the time complexity grows with each layer of iteration, it only does so *additively*, which is overshadowed by optimal *multiplicative* handling of the query complexity. We formally prove the following result in Section 11.

Theorem 3.12 (Iterated Functions)**.** *Let $f\colon [q]^n \to [q]$ be a total function. There exists a bounded-error quantum algorithm that evaluates the iterated function $f^{(d)}$ in query complexity $\mathcal{O}(\mathrm{Adv}^{\pm}(f)^d)$ and time complexity*[9] *$\mathcal{O}_f\big(d \cdot \mathrm{Adv}^{\pm}(f)^d\big)$, where $\mathrm{Adv}^{\pm}(f)$ is the adversary bound of f. The algorithm works in the circuit model.*

Proof sketch. We use induction to construct the transducer $S_{f^{(d)}}$ evaluating the function $f^{(d)}$ and having worst-case query complexity $\mathrm{Adv}^{\pm}(f)^d$, and transduction complexity $\mathcal{O}\big(\mathrm{Adv}^{\pm}(f)^d\big)$.

The base case is the transducer S_f from Theorem 3.4. For the inductive step, we use Proposition 3.7 with $S_A = S_{f^{(d)}}$ and $S_B = S_f$ to get $S_{f^{(d+1)}} = S_A \circ S_B$. In notations of that theorem, O encodes the input to $f^{(d+1)}$ and O' the input to $f^{(d)}$ obtained by evaluating the lowest level of the composition tree. First, S_A does not make direct queries to O. Second,

[8] Papers [35, 36] are mostly known for their *query* results, but they also contain statements on the time complexity of the resulting algorithms. It is these time complexity statements that we extend in Theorem 3.12.

[9] We use $\mathcal{O}_f(\cdot)$ to indicate that the suppressed constant may depend on the particular function f.

$S_B = S_f$ has worst-case Las Vegas query complexity $\mathrm{Adv}^{\pm}(f)$ on a unit admissible vector. Hence, by (3.19) and the induction assumption:

$$L\big(S_A \circ S_B, O, |0\rangle\big) \leqslant \mathrm{Adv}^{\pm}(f) L\big(S_A, O', |0\rangle\big) \leqslant \mathrm{Adv}^{\pm}(f)^{d+1}.$$

Similarly, the worst-case transduction complexity of S_B on a unit admissible vector is $\mathrm{Adv}^{\pm}(f)$, hence by (3.17):

$$\begin{aligned} W\big(S_A \circ S_B, O, |0\rangle\big) &\leqslant W\big(S_A, O, |0\rangle\big) + \mathrm{Adv}^{\pm}(f) L(S_A, O', |0\rangle) \\ &\leqslant \mathcal{O}\big(\mathrm{Adv}^{\pm}(f)^d\big) + \mathrm{Adv}^{\pm}(f)^{d+1} = \mathcal{O}\big(\mathrm{Adv}^{\pm}(f)^{d+1}\big) \end{aligned}$$

for a sufficiently large constant behind the big-Oh.

For the time complexity, we have by induction that $S_{f^{(d)}}$ has time complexity $d \cdot T(S_f)$. The theorem follows from Theorem 3.3. □

Note that the transducer $S_{f^{(d)}}$ in the proof of Theorem 3.12 is different from the transducer $S'_{f^{(d)}}$ we would get by applying Theorem 3.4 to the adversary bound of $f^{(d)}$ obtained using the composition results for the adversary bound. Indeed, for $S'_{f^{(d)}}$, its transduction and query complexities are equal, which is not the case for $S_{f^{(d)}}$. Also, for $S'_{f^{(d)}}$, we have no guarantees on its running time. In Theorem 3.12, we use the non-query part v° of the catalyst as a "scaffolding" to build a time-efficient iterative algorithm.

3.6 Purifiers and Composition of Functions

Although we have studied thriftiness from Section 1, we have so far not touched much on exactness. True, in most cases, like in Section 3.2 on quantum walks, or Theorem 3.4 on the adversary bound, the transduction action of the corresponding transducer is exact. In this section, we will show how to get very close to general exactness starting from an arbitrary algorithm evaluating a function with bounded error.

We consider both Boolean and non-Boolean functions. In the Boolean case, we abstract the action of the function-evaluating algorithm as an input oracle performing the following state generation:

$$O_\psi \colon |0\rangle_{\mathcal{M}} \mapsto |\psi\rangle_{\mathcal{M}} = |0\rangle_{\mathcal{B}}|\psi_0\rangle_{\mathcal{N}} + |1\rangle_{\mathcal{B}}|\psi_1\rangle_{\mathcal{N}} \tag{3.30}$$

in some space $\mathcal{M} = \mathcal{B} \otimes \mathcal{N}$ with $\mathcal{B} = \mathbb{C}^2$. The action of the purifier only depends on the state ψ in (3.30), hence the notation O_ψ. We allow the gap to be at any position c between 0 and 1. In other words, we assume there exist constants $0 \leqslant c - d < c + d \leqslant 1$ such that

$$\text{either} \qquad \|\psi_1\|^2 \leqslant c - d \qquad \text{or} \qquad \|\psi_1\|^2 \geqslant c + d. \tag{3.31}$$

The first case is negative, the second one positive, or, $f(\psi) = 0$ and $f(\psi) = 1$, respectively.

In the non-Boolean case, the range is some $[p]$. For simplicity, we assume $p = \mathcal{O}(1)$ here. An input oracle has the form

$$O_\psi \colon |0\rangle_{\mathcal{M}} \mapsto |\psi\rangle_{\mathcal{M}} = \sum_{j=0}^{p-1} |j\rangle_{\mathcal{B}}|\psi_j\rangle_{\mathcal{N}}, \tag{3.32}$$

with $\mathcal{B} = \mathbb{C}^p$, and we assume there exists a (unique) $f(\psi) \in [p]$ such that

$$\|\psi_{f(\psi)}\|^2 \geqslant \frac{1}{2} + d \tag{3.33}$$

for some constant $d > 0$.

The traditional way of performing error reduction is via majority voting. The following result is folklore.

Theorem 3.13 (Majority Voting)**.** *For any $\varepsilon > 0$, there exists an algorithm with bidirectional access to an oracle like in (3.32) that has the following properties. Assuming the oracle satisfies (3.31) or (3.33), the algorithm evaluates $f(\psi)$ with error at most ε. The query complexity of the algorithm is $\mathcal{O}\big(\log \frac{1}{\varepsilon}\big)$ and its time complexity in the circuit model is polynomial in $\log \frac{1}{\varepsilon}$.*

The majority-voting construction is a direct quantisation of a purely classical technique. The logarithmic query complexity in the above theorem results in extra logarithmic factors that can be found in the analyses of a large variety of quantum algorithms. In this paper, we develop an alternative, genuinely quantum approach to error reduction, that we call a *purifier*. The main feature is that, unlike majority voting, the query complexity of a purifier stays bounded by a constant *no matter how small the error ε is*. This is effectively equivalent to having an errorless algorithm (although, we cannot obtain an exact algorithm with a finite overhead in general). We prove the following theorem in Section 13.

Theorem 3.14 (Purifier)**.** *For any $\varepsilon > 0$, there exists a canonical transducer S_{pur} with bidirectional access to an oracle like in (3.32) that has the following properties. Assuming the oracle satisfies (3.31) or (3.33), the purifier transduces $|0\rangle$ into $|f(\psi)\rangle$ with error at most ε (in the sense to be made exact in Section 12). Both its transduction and query complexities are bounded by a constant. Its time complexity is $\mathcal{O}\big(s \log \frac{1}{\varepsilon}\big)$ in the circuit model, and $\mathcal{O}(T_{\mathrm{R}})$ in the QRAG model, where s is the number of qubits used by $\mathcal{M}$.*

The purifier is inspired by the corresponding construction in [14], which used the dual adversary bound. It had the same characteristic property of query complexity being bounded by a constant, but there are some differences.

- The purifiers in [14] worked solely in the *query* complexity settings. The resulting dual adversary bound was for *exact* function evaluation. Also, by the nature of the adversary bound, it was implicitly assumed that there is only a *finite* collection of possible input oracles, and, technically, for different collections, we obtain different purifiers.
- The purifiers in the current paper are constructed keeping both query and *time* complexity in mind. Also, the same purifier works for all (*infinitely many*) possible input oracles.
- Due to these improvements, the purifier ceases to be exact, but introduces a small error.

Proof sketch of Theorem 3.14. We consider the Boolean case, as the general case can be easily obtained using the Bernstein-Vazirani algorithm [18] as in, e.g., [29, Section 4]. Our construction is a quantum walk in the sense of Section 3.2. In particular, it transduces $|0\rangle$ into $(-1)^{f(\psi)}|0\rangle$.

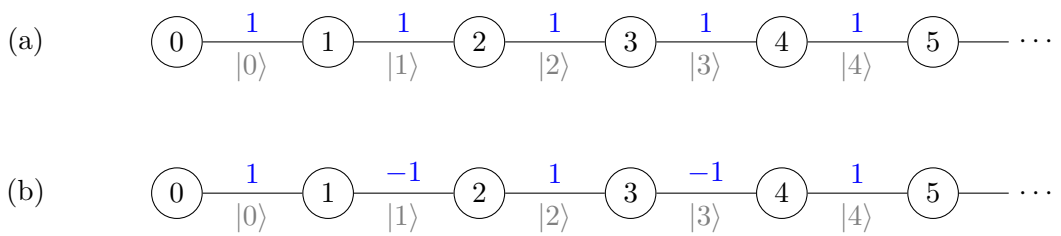
The overall structure is given by the following toy transducer S_{toy} in Figure 3.8. It is a quantum walk on the one-sided infinite line. That is, $S_{\mathrm{toy}} = R_2 R_1$, where the reflection R_1 is the product of the local reflections about the odd vertices $1, 3, 5, \cdots$, and R_2 about the positive even vertices $2, 4, 6, \cdots$. The local reflection at the vertex $i > 0$ is given by the X operation, which maps

$$|i-1\rangle + |i\rangle \mapsto |i-1\rangle + |i\rangle \qquad \text{and} \qquad |i-1\rangle - |i\rangle \mapsto -|i-1\rangle + |i\rangle. \tag{3.34}$$

Now, it is not hard to see that in the negative and the positive case, we have the following mappings, which follow the general case of (3.4) and (3.5):

$$|0\rangle + \sum_{i=1}^{+\infty} |i\rangle \overset{S_{\mathrm{toy}}}{\longmapsto} |0\rangle + \sum_{i=1}^{+\infty} |i\rangle \qquad \text{and} \qquad |0\rangle + \sum_{i=1}^{+\infty} (-1)^i |i\rangle \overset{S_{\mathrm{toy}}}{\longmapsto} -|0\rangle + \sum_{i=1}^{+\infty} (-1)^i |i\rangle. \tag{3.35}$$

Figure 3.8

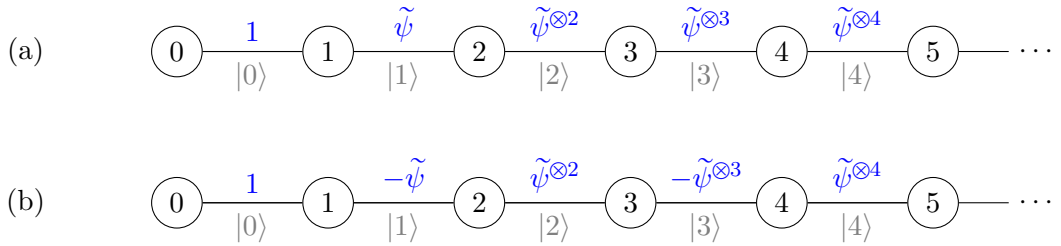


A toy transducer illustrating the overall construction of a purifier. It is a quantum walk on the one-sided infinite line. Each edge correspond to an element of the standard basis written below it. The public space is given by $|0\rangle$. We have two different catalysts (a) and (b) for the same initial state, the numbers above the edge giving the corresponding coefficients.

This formally gives us both $|0\rangle \overset{S_{\text{toy}}}{\rightsquigarrow} |0\rangle$ and $|0\rangle \overset{S_{\text{toy}}}{\rightsquigarrow} -|0\rangle$. Of course, this does not contradict Theorem 3.1, because the corresponding space has infinite dimension and both catalysts in (3.35) have infinite norm.

Nonetheless, we will be able to utilise this general construction. The first order of business is to reduce the norm of the catalysts. Our next step towards a purifier will be a multidimensional quantum walk on the line, in the sense that each edge corresponds to a multidimensional subspace. See Figure 3.9.

Figure 3.9



An improved transducer. It is a multidimensional quantum walk. Each edge corresponds to the element of the standard basis beneath it tensor multiplied by $\mathcal{M}^{\otimes\infty}$. The public space is given by $|0\rangle$. For the negative and positive case, we have the catalysts like in (a) and (b), respectively, where the vector above the edge is placed in the corresponding subspace.

We will define a vector $\widetilde{\psi}$ that depends on ψ and satisfies $\|\widetilde{\psi}\| = 1 - \Omega(1)$. The initial coupling is given by

$$\sum_{i=0}^{\infty} (-1)^{i \cdot f(\psi)} |i\rangle |\widetilde{\psi}\rangle^{\otimes i},$$

and the transduction complexity is bounded by

$$\sum_{i=1}^{\infty} \|\widetilde{\psi}\|^{2i} = \mathcal{O}(1).$$

Let us now explain how to implement the local reflections (3.34) for this modified transducer. Up to a sign, the content of the space incident to a vertex $i > 0$ is given by

$$|i-1\rangle|\widetilde{\psi}\rangle^{\otimes i-1} + (-1)^{f(\psi)}|i\rangle|\widetilde{\psi}\rangle^{\otimes i}. \tag{3.36}$$

We use the input oracle to obtain the state

$$|i-1\rangle|\widetilde{\psi}\rangle^{\otimes i-1}|\psi\rangle + (-1)^{f(\psi)}|i\rangle|\widetilde{\psi}\rangle^{\otimes i}. \tag{3.37}$$

Later we can get back from (3.37) to (3.36) by uncomputing the ψ. Therefore, bringing the term $|i\rangle|\widetilde{\psi}\rangle^{\otimes i-1}$ outside the brackets, it suffices to implement the mapping

$$\begin{cases} |0\rangle|\psi\rangle + |1\rangle|\widetilde{\psi}\rangle \mapsto |0\rangle|\psi\rangle + |1\rangle|\widetilde{\psi}\rangle, & \text{if } f(\psi) = 0; \text{ and} \\ |0\rangle|\psi\rangle - |1\rangle|\widetilde{\psi}\rangle \mapsto -|0\rangle|\psi\rangle + |1\rangle|\widetilde{\psi}\rangle, & \text{if } f(\psi) = 1. \end{cases} \tag{3.38}$$

This is where the condition (3.31) comes into play. We use the same rescaling idea as in [14], and define the state

$$\widetilde{\psi} = \begin{cases} \frac{1}{a}|0\rangle|\psi_0\rangle + b|1\rangle|\psi_1\rangle, & \text{if } f(\psi) = 0; \\ a|0\rangle|\psi_0\rangle + \frac{1}{b}|1\rangle|\psi_1\rangle, & \text{if } f(\psi) = 1. \end{cases}$$

with

$$a = \sqrt[4]{\frac{1-c+d}{1-c-d}} \qquad \text{and} \qquad b = \sqrt[4]{\frac{c+d}{c-d}}.$$

It is not hard to check that $\|\widetilde{\psi}\|^2 = 1 - \Omega(1)$. Now the operation in (3.38) reads as

$$\begin{cases} \big(|0\rangle + \frac{1}{a}|1\rangle\big)|0\rangle|\psi_0\rangle + \big(|0\rangle + b|1\rangle\big)|1\rangle|\psi_1\rangle \mapsto \big(|0\rangle + \frac{1}{a}|1\rangle\big)|0\rangle|\psi_0\rangle + \big(|0\rangle + b|1\rangle\big)|1\rangle|\psi_1\rangle, \\ \big(|0\rangle - a|1\rangle\big)|0\rangle|\psi_0\rangle + \big(|0\rangle - \frac{1}{b}|1\rangle\big)|1\rangle|\psi_1\rangle \mapsto \big(-|0\rangle + a|1\rangle\big)|0\rangle|\psi_0\rangle + \big(-|0\rangle + \frac{1}{b}|1\rangle\big)|1\rangle|\psi_1\rangle, \end{cases}$$

which can be implemented as the 2-qubit reflection about the span of the states

$$\big(a|0\rangle + |1\rangle\big)|0\rangle \qquad \text{and} \qquad \big(|0\rangle + b|1\rangle\big)|1\rangle.$$

Following the same logic as in the toy example, we obtain that $|0\rangle \rightsquigarrow (-1)^{f(\psi)}|0\rangle$. However, this time, both the transduction and the query complexities are bounded by a constant. The problem is that this construction still requires infinite space. We solve this by truncating the line after some vertex D. This introduces an error, but since the norms of the vectors decrease exponentially with i, it suffices to take $D = \mathcal{O}\big(\log \frac{1}{\varepsilon}\big)$. Finally, the transducer can be converted into a canonical form by the standard technique of Proposition 10.4. □

The main purpose of purifiers is to reduce error. Let us give few examples. First, recall the definition of the composed function $f \circ g$ from (3.29):

$$\begin{aligned} (f \circ g)(z_{1,1}, \dots, z_{1,m}, \;\; z_{2,1}, \dots, z_{2,m}, \;\; \dots\dots, \;\; z_{n,1}, \dots, z_{n,m}) \\ = f\big(g(z_{1,1}, \dots, z_{1,m}), g(z_{2,1}, \dots, z_{2,m}), \dots, g(z_{n,1}, \dots, z_{n,m})\big). \end{aligned}$$

We use notation

$$\vec{y}_i = (z_{i,1}, \dots, z_{i,m}) \qquad \text{and} \qquad x = \big(g(\vec{y}_1), g(\vec{y}_2), \dots g(\vec{y}_n)\big)$$

so that $f(x) = (f \circ g)(z)$. In the circuit model, we have the following result on the evaluation of this function.

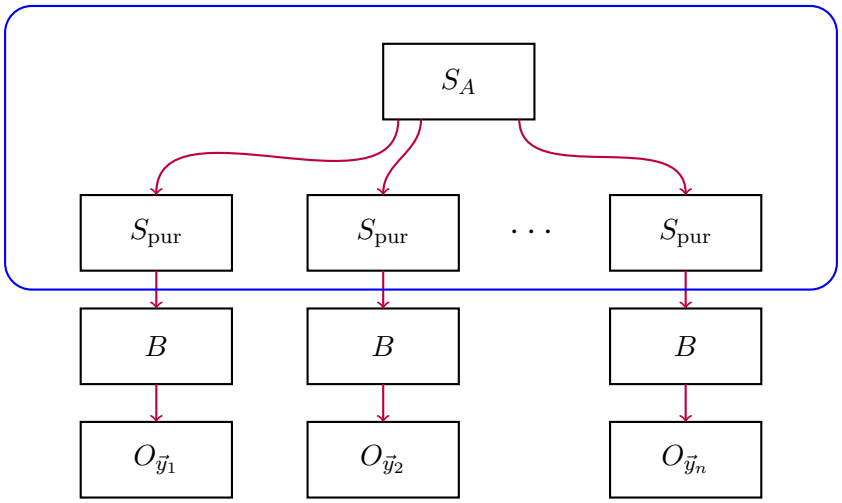
Theorem 3.15. *Let A and B be quantum algorithms in the circuit model that evaluate the functions f and g, respectively, with bounded error. Then, there exists a quantum algorithm in the circuit model that evaluates the function $f \circ g$ with bounded error in time complexity*

$$\mathcal{O}(L)\big(T(A) + T(B) + s \log L\big) \tag{3.39}$$

where L is the worst-case Las Vegas query complexity of A, and s is the space complexity of B.

Proof sketch. We obtain the transducer S_A in the circuit model using Theorem 3.5. The transduction and query complexities of S_A are bounded by L. Let S_{pur} be the purifier for the input oracle B. Consider the transducer $S = S_A \circ \bigoplus_i S_{\mathrm{pur}}$, see Figure 3.10. The transducer S_{pur} on the input oracle $B(O_{\vec{y}_i})$ evaluates $g(y_i)$ with diminished error. It suffices to make error somewhat smaller than $1/L$. Then, S on the input oracle $\bigoplus_i B(O_{\vec{y}_i})$ evaluates $f \circ g$ with bounded error.

Figure 3.10



A composition scheme for Theorem 3.15. The input oracle O_z is broken down as $\bigoplus_i O_{\vec{y}_i}$. The composed transducer contains the elements inside the blue box. The program B is executed as is, serving as an input oracle to the composed transducer.

The transduction complexity of S_{pur} on a unit admissible vector is $\mathcal{O}(1)$. Hence, by (3.17)

$$W(S, O_z, |0\rangle) \leqslant W(S_A, O_x, |0\rangle) + \mathcal{O}(1) \cdot L(S_A, O_x, |0\rangle) = \mathcal{O}(L).$$

All the purifiers can be implemented in parallel, hence by Proposition 3.7,

$$T(S) = T(S_A) + T(S_{\mathrm{pur}}) = \mathcal{O}\big(T(A)\big) + \mathcal{O}(s \log L).$$

The theorem follows from Theorem 3.2 applied to S, where we replace execution of the input oracle by the execution of B. Again, all the B can be executed in parallel. □

In the QRAG model, we can easily deal with different functions g_i, like in the following function, which we already considered in (2.1):

$$f\big(g_1(z_{1,1}, \ldots, z_{1,m}), g_2(z_{2,1}, \ldots, z_{2,m}), \ldots, g_n(z_{n,1}, \ldots, z_{n,m})\big). \tag{3.40}$$

We again use notation

$$\vec{y}_i = (z_{i,1}, \dots, z_{i,m}) \qquad \text{and} \qquad x = \big(g_1(\vec{y}_1), g_2(\vec{y}_2), \dots g_n(\vec{y}_n)\big).$$

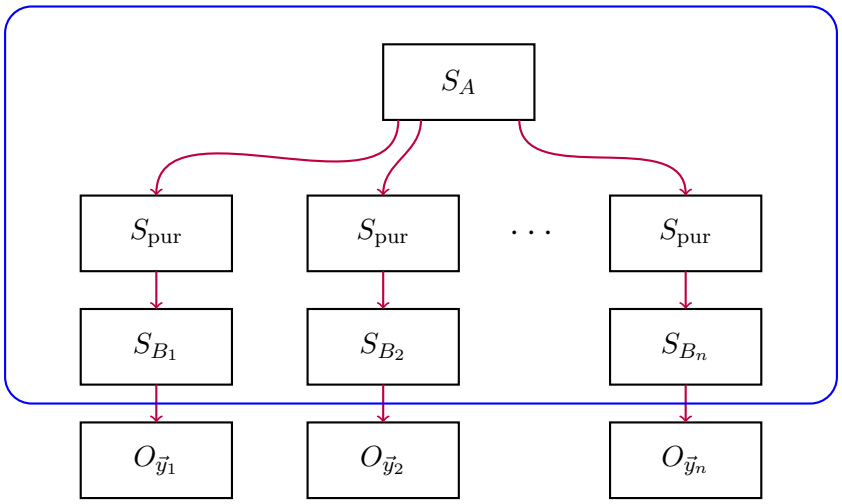
Theorem 3.16. *Consider the function as in (3.40). Let A and $B_1, \dots, B_n$ be quantum algorithms that evaluate the functions f and $g_1, \dots, g_n$, respectively, with bounded error. Assuming the QRAG model and QRAM access to the description of A and $B_1, \dots, B_n$, there exists a quantum algorithm that evaluates the function (3.40) with bounded error in time complexity*

$$\mathcal{O}(T_{\mathrm{R}}) \max_x \Big(T(A) + \sum_{i=1}^{n} T(B_i) L_x^{(i)}(A) \Big). \tag{3.41}$$

Here $L_x^{(i)}(A)$ is the i-th partial Las Vegas complexity $L^{(i)}\big(A, O_x, |0\rangle\big)$ of the algorithm A on the input oracle encoding x.

Proof sketch. We obtain the transducer S_A in the QRAG model using Theorem 3.5. Its transduction complexity is $T(A)$. We assume all B_i have the same range and the same error. Let S_{pur} be the corresponding purifier. Finally, let S_{B_i} be the transducer in the QRAG model corresponding to B_i. Consider the composed transducer S as in Figure 3.11. Similarly to Theorem 3.15, the transducer S on the input oracle $O_z = \bigoplus_i O_{\vec{y}_i}$ evaluates the function $f \circ g$ with bounded error provided that the error of the purifier is sufficiently smaller than $1/L$.

Figure 3.11



A composition scheme for Theorem 3.16. The input oracle O_z is again broken down as $\bigoplus_i O_{\vec{y}_i}$. The composed transducer S contains the elements inside the blue box. This time every B_i is turned into a transducer and partakes in the composition.

Using that the transduction and the query complexity of S_{pur} are $\mathcal{O}(1)$, we obtain that $W_{\max}(S_{\mathrm{pur}} \circ S_{B_i}, O_{\vec{y}_i}) = \mathcal{O}\big(T(B_i)\big)$. Therefore, by (3.21) and using that A and S_A have the same query state:

$$W\big(S, O_z, |0\rangle\big) \leqslant W(S_A, O_x, |0\rangle) + \sum_i \mathcal{O}\big(T(B_i)\big) L^{(i)}\big(A, O_x, |0\rangle\big).$$

For the time complexity, S is a composition of three transducers, where the last two are direct sums. All three of them have time complexity $\mathcal{O}(T_{\mathrm{R}})$, hence, this is also the time complexity of S. The theorem follows from Theorem 3.2. □

Comparison between Theorems 3.15 and 3.16 is similar to the comparison between Theorems 3.8 and 3.10 in Section 3.5. The second theorem considers a more general case (3.40), and its formulation is close to (2.4). On the other hand, in (3.39), $T(B)$ will most likely dominate $s \log L$, so the latter can be removed. Also, if $T(A)$ is smaller than $T(B)$, the whole expression is dominated by $L \cdot T(B)$, which is what we would like to have.

Finally, in the QRAG model, any bounded-error quantum algorithm A can be turned into an essentially exact transducer S_A such that, up to constant factors, its transduction complexity is $T(A)$ and its query complexity is the query complexity of A. The latter transducers can be composed in multiple layers as in Theorem 3.11.

4 Preliminaries

If not said otherwise, a *vector space* is a finite-dimensional complex inner product space. They are denoted by calligraphic letters. We assume that each vector space has a fixed orthonormal basis, and we often identify an operator with the corresponding matrix. $I_{\mathcal{X}}$ stands for the identity operator in $\mathcal{X}$. The inner product is denoted by $\langle \cdot, \cdot \rangle$. A^* stands for the adjoint linear operator. All projectors are orthogonal projectors. We use ket-notation to emphasise that a vector is a state of a quantum register, or to denote the elements of the computational basis.

We use $\mathcal{O}$ for the big-Oh notation in order to distinguish from O, which we use for input oracles. $\mathcal{O}_A$ means that the constant may depend on A. If P is a predicate, we use 1_P to denote the corresponding indicator variable; which is equal to 1 if P is true, and to 0 otherwise.

4.1 Query Algorithms

In this section, we briefly describe the model of quantum query algorithms, and define quantum Las Vegas query complexity. The query model itself is essentially standard [20, 22], with the main difference that we do not restrict ourselves to the evaluation of functions, and also allow for multiple input oracles, which can be arbitrary unitaries. The notion of quantum Las Vegas query complexity is relatively new [17].

A quantum query algorithm A works in space $\mathcal{H}$, which we call the *workspace* of the algorithm. The algorithm is given an oracle O, which is a unitary[10] in some space $\mathcal{M}$. The interaction between the algorithm and the oracle is in the form of queries, which we are about to define. The workspace is decomposed as $\mathcal{H} = \mathcal{H}^\circ \oplus \mathcal{H}^\bullet$, where the oracle is only applied to the second half. Also, $\mathcal{H}^\bullet = \mathcal{H}^\uparrow \otimes \mathcal{M}$ for some $\mathcal{H}^\uparrow$, and the *query* is

$$\widetilde{O} = I^\circ \oplus I \otimes O, \tag{4.1}$$

where I° and I are the identities in $\mathcal{H}^\circ$ and $\mathcal{H}^\uparrow$, respectively.

In terms of registers, we assume the decomposition $\mathcal{H} = \mathcal{H}^\circ \oplus \mathcal{H}^\bullet$ is marked by a register $\mathcal{R}$ so that $|0\rangle_{\mathcal{R}}$ corresponds to $\mathcal{H}^\circ$ and $|1\rangle_{\mathcal{R}}$ to $\mathcal{H}^\bullet$. Then, the query $\widetilde{O}$ is an application of O, controlled by $\mathcal{R}$, where O acts on some subset of the registers (which constitute $\mathcal{M}$).

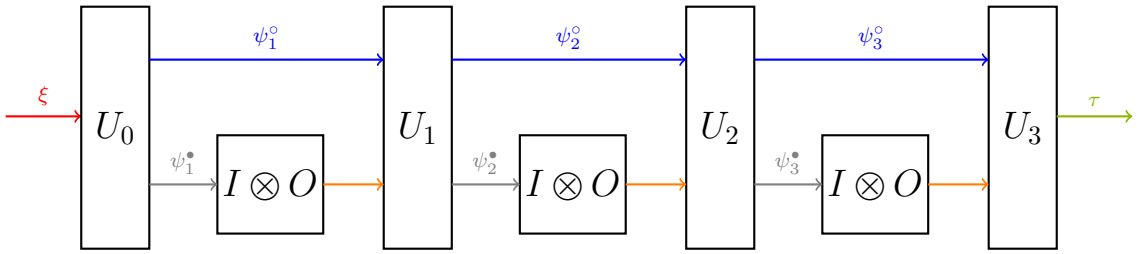
The *quantum query algorithm* $A = A(O)$ is a sequence of linear transformations in $\mathcal{H}$:

$$A(O) = U_Q \, \widetilde{O} \, U_{Q-1} \, \widetilde{O} \cdots U_1 \, \widetilde{O} \, U_0, \tag{4.2}$$

where U_t are some input-independent unitaries in $\mathcal{H}$. See Figure 4.1. Thus, the algorithm implements a transformation $O \mapsto A(O)$: from the input oracle O in $\mathcal{M}$ to the linear operator (4.2) in $\mathcal{H}$.

[10] While [17] define more general input oracles, we, for simplicity, consider only unitary input oracles in this paper.

Figure 4.1



A graphical illustration of a quantum query algorithm with $Q = 3$ queries. The algorithm interleaves input-independent unitaries U_t with queries $\widetilde{O} = I^\circ \oplus I \otimes O$. The intermediate state ψ_t after U_{t-1} and before the t-th query is decomposed as $\psi_t^\circ \oplus \psi_t^\bullet$, where only the second half is processed by the oracle.

We will generally work with the state conversion formalism. We say that A transforms ξ into τ on oracle O, if $A(O)\xi = \tau$. We say that A does so ε-approximately if $\|\tau - A(O)\xi\| \leqslant \varepsilon$. In this context, we will often call ε the *error* of the algorithm.

Let us make two important remarks on the structure of thus defined query algorithms.

Remark 4.1 (Alignment)**.** Note that all the queries in (4.2) are identical, i.e., the oracle O is always applied to the same registers and is always controlled by $\mathcal{R}$. To acknowledge this, we say that all the queries in A are *aligned*. Usually, this is not important, but the alignment property will play a significant role in this paper, in particular in Sections 9 and 10. The main reason is that for the aligned program we can perform all the queries in parallel (assuming we have the intermediate states ψ_t from Figure 4.1 somehow).

Remark 4.2 (Unidirectionality)**.** The input oracle is unidirectional: the algorithm only has access to O. This suffices for most of our results. Quite often, however, bidirectional access to the input oracle is allowed, where the algorithm can query both O and O^*. The latter is a special case of the former, as bidirectional access to O is equivalent to unidirectional access to $O \oplus O^*$. As this situation will be common in some sections of the paper, we utilise the following piece of notation:

$$\overleftrightarrow{O} = O \oplus O^*. \tag{4.3}$$

The standard complexity measure of a quantum query algorithm is $Q = Q(A)$: the total number of times the query $\widetilde{O}$ is executed. It was called Monte Carlo complexity in [17] in order to distinguish it from Las Vegas complexity defined next.

Let $\Pi^\bullet$ be the projector onto $\mathcal{H}^\bullet$. The state processed by the oracle on the t-th query is $\psi_t^\bullet = \Pi^\bullet U_{t-1} \widetilde{O} U_{t-2} \cdots \widetilde{O} U_0$, and the *total query state* is

$$q(A, O, \xi) = \bigoplus_{t=1}^{Q} \psi_t^\bullet. \tag{4.4}$$

This is the most complete way of specifying the work performed by the input oracle O in the algorithm A on the initial state ξ. It is a member of $\mathcal{E} \otimes \mathcal{M}$ for some space $\mathcal{E}$ (the latter actually being equal to $\mathbb{C}^Q \otimes \mathcal{H}^\uparrow$). The simplest way to gauge the total query state is by defining the corresponding *quantum Las Vegas query complexity*:

$$L(A, O, \xi) = \|q(A, O, \xi)\|^2. \tag{4.5}$$

As mentioned in Section 2.3, we extend the definitions (4.4) and (4.5) for the case $\xi' \in \mathcal{E} \otimes \mathcal{H}$ for some $\mathcal{E}$ using identities (2.10) and (2.11).

4.2 Multiple Input Oracles

It is also possible for an algorithm to have access to several input oracles $O^{(1)}, \dots, O^{(r)}$. Las Vegas query complexity can naturally accommodate such a scenario. Indeed, access to several input oracles is equivalent to access to the one combined oracle

$$O = O^{(1)} \oplus O^{(2)} \oplus \dots \oplus O^{(r)}. \tag{4.6}$$

Consequently, the space of the oracle has a similar decomposition $\mathcal{M} = \mathcal{M}^{(1)} \oplus \dots \oplus \mathcal{M}^{(r)}$, where $O^{(i)}$ acts in $\mathcal{M}^{(i)}$. The total query state can also be decomposed into *partial query states*

$$q(A, O, \xi) = q^{(1)}(A, O, \xi) \oplus q^{(2)}(A, O, \xi) \oplus \dots \oplus q^{(r)}(A, O, \xi),$$

where $q^{(i)}(A, O, \xi)$ is processed by $O^{(i)}$. This gives partial Las Vegas query complexities

$$L^{(i)}(A, O, \xi) = \left\| q^{(i)}(A, O, \xi) \right\|^2.$$

In terms of registers, it can be assumed that the input oracle is controlled by some register $\mathcal{R}$, where the value $|0\rangle_{\mathcal{R}}$ indicates no application of the input oracle, and $|i\rangle_{\mathcal{R}}$ with $i > 0$ indicates the i-th input oracle $O^{(i)}$. We note that we use i in $|i\rangle_{\mathcal{R}}$ only as a label. In particular, we do not perform any arithmetical operations on them. Therefore, $|i\rangle_{\mathcal{R}}$ can have a complicated internal encoding that can facilitate the application of the oracle.

The assumption on the register $\mathcal{R}$ in this section is not necessarily in contradiction with the assumptions of Section 4.1, as $\mathcal{R}$ can have a separate qubit that indicates whether i in $|i\rangle_{\mathcal{R}}$ is non-zero. This qubit can serve as $\mathcal{R}$ in the sense of Section 4.1.

The case of usual Monte Carlo query complexity requires additional changes, as per now it turns out that all the oracles are queried the same number of times, Q. One way to allow for different number of queries is as follows. Similarly to (4.1), define the query to the i-th input oracle as

$$\widetilde{O}^{(i)} = I^{\circ} \oplus I \otimes \Big(I^{(1)} \oplus \dots \oplus I^{(i-1)} \oplus O^{(i)} \oplus I^{(i+1)} \oplus \dots \oplus I^{(r)} \Big),$$

where the decomposition in the brackets is the same as in (4.6). In other words, $\widetilde{O}^{(i)}$ is just the application of $O^{(i)}$ controlled by $|i\rangle_{\mathcal{R}}$. The query algorithm is then defined as

$$A(O) = U_Q \, \widetilde{O}^{(s_Q)} \, U_{Q-1} \, \widetilde{O}^{(s_{Q-1})} \, \dots \, U_2 \, \widetilde{O}^{(s_2)} \, U_1 \, \widetilde{O}^{(s_1)} \, U_0, \tag{4.7}$$

where $s_1, s_2, \dots, s_Q \in [r]$. The number of invocations of the i-th oracle, $Q^{(i)}$, is defined as the number of s_j equal to i in (4.7).

4.3 Evaluation of Functions

The *standard* settings for quantum algorithms evaluating a (partial) function $f \colon D \to [p]$ with $D \subseteq [q]^n$ are as follows. For an input $x \in [q]^n$, the corresponding input oracle acts in $\mathbb{C}^n \otimes \mathbb{C}^q$ as

$$O_x \colon |i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle \tag{4.8}$$

for all $i \in [n]$ and $b \in [q]$. Here $\oplus$ stands for the bitwise XOR, and we assume that q is a power of 2 (we can ignore the inputs outside of the domain).

The algorithm A itself has a special output register isomorphic to $\mathbb{C}^p$. After finishing the algorithm, measuring the output register should yield the value $f(x)$. This either happens with probability 1 (for exact algorithms), or with probability at least $1/2 + d$ for some constant $d > 0$ (bounded error).

This definition is nice because there is one well-defined input oracle O_x for each input. Also $O_x^2 = I$, which makes uncomputing very easy. Unfortunately, the standard definition has a drawback that the input oracle of the algorithm has a more restricted form than the one required for the algorithm itself. This is problematic if the algorithm is expected to be used as an input oracle for another algorithm. This issue is solved by noting that $|b\rangle \mapsto |b \oplus f(x)\rangle$ can be implemented by evaluating $f(x)$, performing the XOR operation, and uncomputing $f(x)$. This increases the complexity by a factor of 2, which is fine if we ignore constant factors. We call it *robust* evaluation of function, as the action of the algorithm is specified for all input states, not just $|0\rangle$.

However, constant factors can be important, for instance, in iterated functions, where such factors appear as a base of the exponent, or in the settings of Las Vegas complexity in [17], where precise complexity is sought for. In this case, a more homogenous definition would be appreciated.

We follow one such approach from [14], which we call *state-generating*. We say that the input oracle O_x encodes the input string $x \in [q]^n$ if it performs the transformation

$$O_x \colon |i\rangle|0\rangle \mapsto |i\rangle|x_i\rangle \tag{4.9}$$

for all $i \in [n]$. The action of this oracle on $|i\rangle|0\rangle$ is identical to that of (4.8), but we do not require anything for other states. In other words, the admissible subspace of O_x consists of vectors having $|0\rangle$ in the second register. The admissible subspace of the algorithm itself is spanned by $|0\rangle$. On that, it has to perform the transformation $|0\rangle \mapsto |f(x)\rangle$. The algorithm has bidirectional access to O_x, which we treat as unidirectional access to $\overleftrightarrow{O_x}$ defined in (4.3). It is possible to assume the input oracle O_x is a direct sum of n unitaries acting in $\mathbb{C}^q$ in order to apply the multiple input oracle settings from Section 7.2.

As the initial state is always $|0\rangle$, and O_x is essentially determined by x, we will write

$$L_x(A) = L\big(A, \overleftrightarrow{O_x}, |0\rangle\big) \qquad \text{and} \qquad L(A) = \max_{x \in D} L_x(A). \tag{4.10}$$

More precisely, we define $L_x(A)$ as the supremum over all input oracles O_x that encode the input x. We use similar notation for $L_x^{(i)}$ and $L^{(i)}$.

This approach has a number of advantages. First, it casts function evaluation as a special case of state conversion with state-generating input oracles [14]. Second, the algorithm can be directly used as a part of the input oracle for another algorithm without any uncomputation. Finally, this definition does not involve the somewhat arbitrary XOR operation and may be, thus, regarded as being more pure. In particular, it makes sense to ask for the precise value of its quantum Las Vegas query complexity (and not just only up to constant factors).

This approach has a number of disadvantages. First, we have to explicitly allow bidirectional access to O_x in order to allow uncomputing, as it is no longer the case that O_x is its own inverse. More importantly, though, neither the action nor the Las Vegas complexity of the algorithm is specified for the initial states orthogonal to $|0\rangle$. For our own algorithms, we can design them so that O_x is only executed with $|0\rangle$ in the second register (maybe after some perturbation, see Section 4.6). But, if we are dealing with an arbitrary algorithm, we have no such guarantees.

4.4 Circuit Model

We assume that the space of the algorithm is embedded into a product of qubits, $(\mathbb{C}^2)^{\otimes s}$, for some s called the space complexity of the algorithm. A quantum program is a product of elementary operations called gates:

$$A = G_T G_{T-1} \cdots G_1. \tag{4.11}$$

In the circuit model, each gate G_i is usually a 1- or a 2-qubit operation that can be applied to any qubit or a pair of qubits. The number of elementary operations, T, is called the time complexity of A, and is denoted by $T(A)$. We assume a universal gate set, so that every unitary can be written as a quantum program. We do not focus too much on a particular model, as they are all essentially equivalent.

In a query algorithm like in Section 4.1, each execution of the input oracle $\widetilde{O}$ also traditionally counts as one elementary operation. In other words, each unitary in (4.2) can be decomposed into elementary gates as in (4.11) to give a corresponding program in the circuit model. We use T to denote its time complexity, and Q to denote its Monte Carlo query complexity.

We will often require an algorithm like in (4.11) to be executed conditionally, that is, controlled by the value of some external qubit. In other words, we would like to perform an operation A^{c} of the form

$$|0\rangle|\xi\rangle \overset{A^{\mathrm{c}}}{\longmapsto} |0\rangle|\xi\rangle, \qquad |1\rangle|\xi\rangle \overset{A^{\mathrm{c}}}{\longmapsto} |1\rangle|A\xi\rangle,$$

where the first qubit is the control qubit. We will denote time complexity of this procedure by $T_{\mathrm{C}}(A)$.

Since it is possible to substitute each G_i in (4.11) by its controlled version, we have that $T_{\mathrm{C}}(A) = \mathcal{O}(T(A))$. But it is often possible to do better. For instance, assume that A is of the form $A_2 A_1^{\mathrm{c}}$, i.e., a large chunk of A is already conditioned. We have that $T(A) = T(A_2) + T_{\mathrm{C}}(A_1)$. On the other hand, $T_{\mathrm{C}}(A) = T_{\mathrm{C}}(A_2) + T_{\mathrm{C}}(A_1) + \mathcal{O}(1)$, because we can calculate the AND of the two control qubits of A_1 into a fresh qubit, execute A_1 controlled by this fresh qubit, and uncompute the new qubit afterwards. In other words, the constant factor of $T_{\mathrm{C}}(A) = \mathcal{O}(T(A))$ is not getting accumulated with each new conditioning, but is, in a way, paid only once.

4.5 QRAG model

The QRAG model extends the circuit model by allowing the following Quantum Random Access Gate:

$$\mathrm{QRAG}\colon |i\rangle|b\rangle|x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_m\rangle \mapsto |i\rangle|x_i\rangle|x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_m\rangle,$$

where the first register is an m-qudit, and the remaining ones are quantum words (i.e., quantum registers large enough to index all the qubits in the program). We assume the QRAG takes time T_{R} as specified in (3.13). Note that this gate would require time $\Omega(m)$ to implement in the usual circuit model, as it depends on all $m + 2$ registers.

This should not be confused with the QRAM model, which allows oracle access to an array of classical registers $x_1, x_2, \ldots, x_m$:

$$\mathrm{QRAM}\colon |i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle,$$

where $\oplus$ stands for the bit-wise XOR. The difference is that x_is are being fixed during the execution of the quantum procedure (but they may be changed classically between different executions). The QRAG model is more powerful than the QRAM model.

The main reason we need the QRAG is the following result (see, e.g., [29] for a formal statement, although the same construction has been used elsewhere including [2, 23, 26]):

Theorem 4.3 (Select Operation)**.** *Assume the QRAG model and that we have QRAM access to a description of a quantum program as in (4.11) in some space $\mathcal{H}$, where each gate G_i either comes from a fixed set of 1- or 2-qubit operations (which can be applied to different qubits each time), or is a QRAG. Let $\mathcal{I}$ be a T-qudit. Then, the following Select operation*

$$\sum_i |i\rangle_{\mathcal{I}} |\psi_i\rangle_{\mathcal{H}} \mapsto \sum_i |i\rangle_{\mathcal{I}} |G_i \psi_i\rangle_{\mathcal{H}} \tag{4.12}$$

can be implemented in time $\mathcal{O}(T_{\mathrm{R}})$.

Proof sketch. We add a number of scratch registers to perform the following operations. Conditioned on i, we use the QRAM to read the description of G_i. We switch the arguments of G_i into the scratch space using the QRAG. We apply the operation G_i on the scratch space. We switch the arguments back into memory, and erase the description of G_i from the scratch memory. All the operations besides applying G_i take time $\mathcal{O}(T_{\mathrm{R}})$. Application of a usual gate G_i takes time $\mathcal{O}(1)$ as the gate set is fixed, or $\mathcal{O}(T_R)$ if G_i is a QRAG. □

It is also possible to not store the whole program in memory, but compute it on the fly, in which case the complexity of this computation should be added to the complexity stated in Theorem 4.3.

Corollary 4.4 (Parallel execution of programs)**.** *Assume that in the settings of Theorem 4.3 we have QRAM access to an array storing descriptions of m quantum programs $A^{(1)}, \dots, A^{(m)}$. Let $\mathcal{I}$ be a m-qudit. Then, the following operation*

$$\sum_i |i\rangle_{\mathcal{I}} |\psi_i\rangle_{\mathcal{H}} \mapsto \sum_i |i\rangle_{\mathcal{I}} |A^{(i)} \psi_i\rangle_{\mathcal{H}},$$

can be implemented in time $\mathcal{O}\big(T_{\mathrm{R}} \cdot \max_i T(A^{(i)})\big)$.

Proof sketch. Use Theorem 4.3 to implement the first gate in all of $A^{(i)}$ in parallel, then the second one, and so on until the time mark $\max_i T(A^{(i)})$. □

Another important primitive is the random access (RA) input oracle. If $O\colon \mathcal{M} \to \mathcal{M}$ is an input oracle, then its RA version acts on $\mathcal{J} \otimes \mathcal{M}^{\otimes K}$, where $\mathcal{J}$ is a K-qudit. If the register $\mathcal{J}$ contains value i, the input oracle is applied to the i-th copy of $\mathcal{M}$ in $\mathcal{M}^{\otimes K}$.

The idea behind this is that if O is implemented as a subroutine, then the RA input oracle is a special case of Corollary 4.4, where all $A^{(i)}$ are the same, but act on different sets of registers (which is easy to define using $|i\rangle_{\mathcal{J}}$ as an offset). Therefore, it makes sense to define the RA input oracle as an elementary operation in the QRAG model.

4.6 Perturbed Algorithms

The following lemma is extremely useful in quantum algorithms, but for some reason has seldom experienced the honour of being explicitly stated.

Lemma 4.5. *Assume we have a collection of unitaries $U_1, \dots, U_m$ all acting in the same vector space $\mathcal{H}$. Let $\psi'_0, \dots, \psi'_m$ be a collection of vectors in $\mathcal{H}$ such that*

$$\psi'_t = U_t \psi'_{t-1}$$

for all t. Let $\psi_0, \dots, \psi_m$ be another collection of vectors in $\mathcal{H}$ such that $\psi_0 = \psi'_0$ and

$$\|\psi_t - U_i \psi_{t-1}\| \leqslant \varepsilon_i \tag{4.13}$$

for all i. Then,

$$\|\psi_m - \psi'_m\| \leqslant \sum_{t=1}^{m} \varepsilon_t. \tag{4.14}$$

Proof. Denote by V_t the product $U_m U_{m-1} \cdots U_{t+1}$. In particular, $V_m = I$. Then,

$$\psi_m - \psi'_m = V_m\psi_m - V_0\psi_0 = \sum_{t=1}^{m} \big(V_t\psi_t - V_{t-1}\psi_{t-1}\big) = \sum_{t=1}^{m} V_t\big(\psi_t - U_t\psi_{t-1}\big).$$

We obtain (4.14) from the triangle inequality using (4.13) and the unitarity of V_t. □

In the application of this lemma, U_t stands for sequential sections of a quantum algorithm. The vectors ψ'_t form the sequence of states the algorithm goes through during its execution. The vectors ψ_t is an idealised sequence, which is used instead of ψ'_t in the analysis.

We call the difference between ψ_t and $U_t\psi_{t-1}$ a (conceptual) perturbation. The expression in (4.13) is the size of the perturbation. Therefore, the Eq. (4.14) can be stated as the total perturbation of the algorithm does not exceed the sum of the perturbations of individual steps. If this sum is small, the final state ψ_m of the analysis is not too far away from the actual final state ψ'_m of the algorithm.

This lemma is implicitly used every time an approximate version of a quantum subroutine is used, which happens in pretty much every non-trivial quantum algorithm.

4.7 Efficient Implementation of Direct-Sum Finite Automata

We will repeatedly use the following construction in this paper, for which we describe a time-efficient implementation. We call it direct-sum quantum finite automata due to its superficial similarity to quantum finite automata.

The space of the algorithm is $\mathcal{K} \otimes \mathcal{P} \otimes \mathcal{H}$, where $\mathcal{K}$ is a K-qudit, $\mathcal{P}$ is a qubit, and $\mathcal{H}$ is an arbitrary space. Additionally, we have black-box access to K unitaries $S_0, \ldots, S_{K-1}$ in $\mathcal{P} \otimes \mathcal{H}$. The algorithm is promised to start in the state of the form

$$|0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\phi\rangle_{\mathcal{H}} + \sum_{t=0}^{K-1} |t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|\psi_t\rangle_{\mathcal{H}}, \tag{4.15}$$

and it has to perform the following transformation

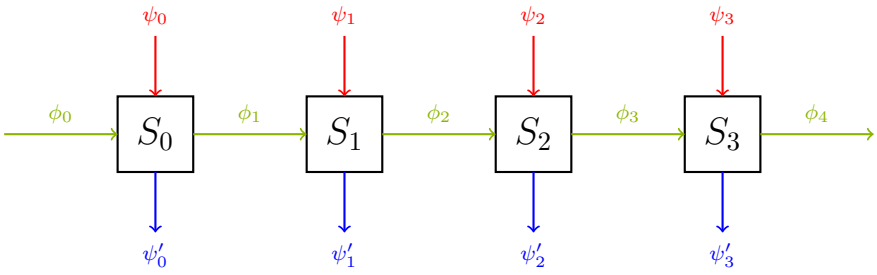
- For $t = 0, 1, \ldots, K-1$:
  (a) Execute S_t on $\mathcal{P} \otimes \mathcal{H}$ conditioned on $|t\rangle_{\mathcal{K}}$. (4.16)
  (b) Conditioned on $|0\rangle_{\mathcal{P}}$, replace $|t\rangle_{\mathcal{K}}$ by $|t+1\rangle_{\mathcal{K}}$.

Let us elaborate on the "replace" in point (b). It is not hard to show by induction that the $|0\rangle_{\mathcal{P}}$-part of the state contains a vector of the form $|t\rangle_{\mathcal{K}}|\phi_{t+1}\rangle_{\mathcal{H}}$. This vector has to be replaced by $|t+1\rangle_{\mathcal{K}}|\phi_{t+1}\rangle_{\mathcal{H}}$. We also assume that $|K\rangle_{\mathcal{K}}$ is identical with $|0\rangle_{\mathcal{K}}$. For a graphical illustration refer to Figure 4.2.

It is trivial to implement the transformation in (4.16) in $\mathcal{O}(K \log K)$ elementary operations besides the executions of S_t. It is a technical observation that we can remove the logarithmic factor.

Lemma 4.6. *The transformation in (4.16) can be implemented in time $\mathcal{O}(K) + \sum_t T_{\mathrm{C}}(S_t)$, where T_{C} is defined in Section 4.4.*

Figure 4.2



A graphical illustration of the action of a direct-sum finite automaton. States ϕ_t represent the internal state of the automaton, as it processes a "string" of quantum vectors $\psi_0, \dots, \psi_{K-1}$ into $\psi'_0, \dots, \psi'_{K-1}$. Unlike the usual quantum automata, the internal state of the automaton and the current "letter" of the "string" are joined via the direct sum. Similarity with Figure 3.2 is apparent. It is an interesting question, whether such finite automata can find other applications.

Proof. The register $\mathcal{K}$ uses $\ell = \log K$ qubits. We introduce an additional register $\mathcal{C}$ that also consists of ℓ qubits. For $i, t \in [K]$, we denote

$$|i \curlyvee t\rangle_{\mathcal{C}} = |1\rangle^{\otimes c}|0\rangle^{\otimes \ell - c}$$

if the binary representations of i and t, considered as elements of $\{0,1\}^\ell$, agree on the first c most significant bits, and disagree on the $(c+1)$-st one (or $c = \ell$).

We modify the algorithm so that before the t-th iteration of the loop, the algorithm is in a state of the form

$$|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|t \curlyvee t\rangle_{\mathcal{C}}|\phi_t\rangle_{\mathcal{H}} + \sum_{i=0}^{t-1}|i\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|i \curlyvee t\rangle_{\mathcal{C}}|\psi'_t\rangle_{\mathcal{H}} + \sum_{i=t}^{K-1}|i\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|i \curlyvee t\rangle_{\mathcal{C}}|\psi_t\rangle_{\mathcal{H}} \tag{4.17}$$

In particular, in the $|0\rangle_{\mathcal{P}}$-part, the register $\mathcal{C}$ contains ℓ ones.

The state (4.17) with $t = 0$ can be obtained from (4.15) in $\mathcal{O}(\ell)$ elementary operations by computing $\mathcal{C}$ starting from the highest qubit. Execution of S_t on Step (a) can be conditioned on the lowest qubit of $\mathcal{C}$, which is equal to 1 if and only if $\mathcal{K}$ contains t.

It remains to consider Step (b) and the update of the register $\mathcal{C}$ during the increment from t to $t+1$. Assume that $t+1$ is divisible by 2^d, but not by 2^{d+1}. Then, Step (b) takes $d+1$ controlled 1-qubit operations. Similarly, the change from t to $t+1$ in $|i \curlyvee t\rangle_{\mathcal{C}}$ in (4.17) takes time $\mathcal{O}(d)$ by first uncomputing the $d+1$ lowest qubits for t, and then computing them for $t+1$. Finally, after the loop, the register $\mathcal{C}$ can be uncomputed in $\mathcal{O}(\ell)$ operations. Therefore, the total number of elementary operations is $\mathcal{O}(K)$. □

5 Transducers

In this section, we define transducers and give their basic properties. This is an initial treatment and will be extended in Section 7 to include query complexity.

5.1 Definition

Mathematically, our approach is based on the following construction.

Theorem 5.1. *Let $\mathcal{H} \oplus \mathcal{L}$ be a direct sum of two vector spaces, and S be a unitary on $\mathcal{H} \oplus \mathcal{L}$. Then, for every $\xi \in \mathcal{H}$, there exist $\tau \in \mathcal{H}$ and $v \in \mathcal{L}$ such that*

$$S\colon \xi \oplus v \mapsto \tau \oplus v. \tag{5.1}$$

Moreover,

(a) The vector $\tau = \tau(S, \xi) = \tau_{\mathcal{H}}(S, \xi)$ is uniquely defined by ξ and S.

(b) The vector $v = v(S, \xi) = v_{\mathcal{H}}(S, \xi)$ is also uniquely defined if we require that it is orthogonal to the 1-eigenspace of $\Pi S \Pi$, where Π is the projection on $\mathcal{L}$.

(c) The mapping $\xi \mapsto \tau$ is unitary and $\xi \mapsto v$ is linear.

We will prove the theorem at the end of this section.

In the setting of Theorem 5.1, we will say that S *transduces* ξ into τ, and write $\xi \overset{S}{\rightsquigarrow} \tau$. The mapping $\xi \mapsto \tau$ on $\mathcal{H}$ will be called the *transduction action* of S on $\mathcal{H}$ and denoted by $S_{\mathcal{H}}^{\{\}}$.

We call any v satisfying $S(\xi \oplus v) = \tau \oplus v$ a *catalyst* for $\xi \overset{S}{\rightsquigarrow} \tau$. The condition in Point (b) of v to be orthogonal to the 1-eigenspace of $\Pi S \Pi$ is crucial for uniqueness, as adding such a vector to v does not affect (5.1). Clearly, the vector v as defined in Point (b) has the smallest possible norm. Therefore, we can define transduction complexity of S on ξ as $W(S, \xi) = \|v(S, \xi)\|^2$ for the latter v. We write $W_{\mathcal{H}}(S, \xi)$ if the space $\mathcal{H}$ might not be clear from the context. The above discussion can be formulated as the following claim.

Claim 5.2. *For any catalyst v of the transduction $\xi \overset{S}{\rightsquigarrow} \tau$, we have $W(S, \xi) \leqslant \|v\|^2$.*

On the other hand, checking orthogonality to $\Pi S \Pi$ is complicated and unnecessary, and we avoid doing it. We usually couple a transducer with some chosen catalyst v for all the ξ of interest, which need not have the smallest possible norm. In this case, we somewhat sloppily write $W(S, \xi) = \|v\|^2$ even for this catalyst v. This agreement will become especially important when we add query complexity into the picture in Section 7; see in particular the note towards the end of Section 7.1.

Other important notions related to the transducer are its time and query complexity. Its time complexity, $T(S)$, is defined as its time complexity as an algorithm. For $\xi \in \mathcal{H}$, its query state, $q_{\mathcal{H}}(S, O, \xi)$, and query complexity, $L_{\mathcal{H}}(S, O, \xi)$, are defined as those of S as an algorithm on the initial state $\xi \oplus v$. Note that for transducers with input oracles, we will adopt a special canonical form defined in Section 7, until then we mostly ignore the oracle-related notions.

Example 5.3. Let us a give a simple concrete example illustrating the above notions. Assume that $\mathcal{H}$ is one-dimensional and spanned by $|0\rangle$, and $\mathcal{L}$ is two-dimensional and spanned by $|1\rangle$ and $|2\rangle$. Let S be the reflection of the vector $|0\rangle - |1\rangle - |2\rangle$, so that its orthogonal complement stays intact.

The transduction action of S on $\mathcal{H}$ is the identity, which is certified by

$$S\colon |0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle \mapsto |0\rangle + \frac{1}{2}|1\rangle + \frac{1}{2}|2\rangle.$$

Hence, we have $v(S, |0\rangle) = (|1\rangle + |2\rangle)/2$, and $W(S, |0\rangle) = 1/2$. This is not the only catalyst, as one can also take $v = |1\rangle$ or $v = |2\rangle$. However, $(|1\rangle + |2\rangle)/2$ is the only catalyst orthogonal to the 1-eigenspace of $\Pi S \Pi$, which is spanned by $|1\rangle - |2\rangle$, and also has the smallest norm.

Proof of Theorem 5.1. The vector v can be found from the equation

$$\Pi v = \Pi\big(\tau + v\big) = \Pi\big(S(\xi + v)\big) = \Pi S \xi + \Pi S v = \Pi S \xi + \Pi S \Pi v.$$

From this we would like to argue that v can be expressed as

$$v = (\Pi - \Pi S \Pi)^{+} \Pi S \xi, \tag{5.2}$$

where $(\cdot)^{+}$ stands for the Moore-Penrose pseudoinverse. Let us show that this is indeed the case.

Denote by $\mathcal{K}$ the kernel of $\Pi - \Pi S \Pi$ in $\mathcal{L}$, and by $\mathcal{K}^{\perp}$ its orthogonal complement in $\mathcal{L}$. The subspace $\mathcal{K}$ equals the 1-eigenspace of $\Pi S \Pi$. But since S is a unitary, a 1-eigenvector of $\Pi S \Pi$ is necessarily a 1-eigenvector of S. Hence, S is a direct sum of the identity on $\mathcal{K}$ and a unitary on $\mathcal{H} \oplus \mathcal{K}^{\perp}$. Thus, $\Pi - \Pi S \Pi$ is a direct sum of the zero operator in $\mathcal{H} \oplus \mathcal{K}$ and some operator in $\mathcal{K}^{\perp}$. Moreover, the latter operator is invertible in $\mathcal{K}^{\perp}$ as its kernel is empty. Since $\xi \in \mathcal{H}$ is orthogonal to $\mathcal{K}$, we get that $\Pi S \xi \in \mathcal{K}^{\perp}$. Hence, (5.2) indeed uniquely specifies v. This proves (b) and the second half of (c).

The uniqueness of τ and the linearity of $\xi \mapsto \tau$ now follow from (5.1) and the linearity of S. Finally, unitarity of S implies $\|\xi\| = \|\tau\|$, hence, the map $\xi \mapsto \tau$ is also unitary. □

Proposition 5.4 (Transitivity of Transduction)**.** *Assume $\mathcal{H} \subseteq \mathcal{H}_1 \subseteq \mathcal{H}_2$ are vector spaces, and S is a unitary in $\mathcal{H}_2$. Then,*

$$S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}}} = (S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}})^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}}},$$

and, for every $\xi \in \mathcal{H}$, a possible catalyst is

$$v_{\mathcal{H}}(S, \xi) = v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi) + v_{\mathcal{H}_1}\big(S, \xi \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi)\big). \tag{5.3}$$

Proof. By definition,

$$S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}} : \xi \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi) \mapsto \tau \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi)$$

for some $\tau \in \mathcal{H}$. The latter means that

$$S : \xi \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi) \oplus v_{\mathcal{H}_1}\big(S, \xi \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi)\big) \mapsto \tau \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi) \oplus v_{\mathcal{H}_1}\big(S, \xi \oplus v_{\mathcal{H}}(S^{\circlearrowleft}_{\circlearrowleft_{\mathcal{H}_1}}, \xi)\big),$$

proving (5.3). □

5.2 Implementation

The key point we will now make is that given a transducer S, there exists a very simple quantum algorithm that approximately implements its transduction action on $\mathcal{H}$. This algorithm is a generalisation of the one from [17], which was used for implementation of the adversary bound.

Before we describe this algorithm, let us establish a few conventions. We call $\mathcal{H}$ the *public* and $\mathcal{L}$ the *private* space of S. We indicate this separation of $\mathcal{H}$ and $\mathcal{L}$ by a privacy qubit $\mathcal{P}$. The value 0 of $\mathcal{P}$ will indicate the public space $\mathcal{H}$, and the value 1 the private space $\mathcal{L}$. This means that both $\mathcal{H}$ and $\mathcal{L}$ are embedded into the same register during implementation. Thus,

$$\xi \oplus v = |0\rangle_{\mathcal{P}} |\xi\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} |v\rangle_{\mathcal{L}} \tag{5.4}$$

explicitly specifying the public and the private spaces. We will extend this notation in Section 7.1.

As it can be understood from the name, the algorithm does not have direct access to the private space $\mathcal{L}$ of the transducer. All the interaction between the transducer and its surrounding is through the public space $\mathcal{H}$.

Theorem 5.5. *Let spaces $\mathcal{H}$, $\mathcal{L}$, and a positive integer K be fixed. There exists a quantum algorithm that transforms ξ into τ' such that*

$$\|\tau' - \tau(S,\xi)\| \leqslant 2\sqrt{\frac{W(S,\xi)}{K}}$$

for every transducer $S\colon \mathcal{H}\oplus\mathcal{L} \to \mathcal{H}\oplus\mathcal{L}$ and initial state $\xi \in \mathcal{H}$. The algorithm conditionally executes S as a black box K times, and uses $\mathcal{O}(K)$ other elementary operations.

Theorem 3.2 is a direct corollary of Theorem 5.5. A sketch of the proof of Theorem 5.5 was already given in the same section, see, in particular, Figure 3.2.

Proof of Theorem 5.5. The space of the algorithm is $\mathcal{K}\otimes(\mathcal{H}\oplus\mathcal{L})$, where $\mathcal{K}$ is a K-qudit. The register $\mathcal{H}\oplus\mathcal{L}$ contains the privacy qubit $\mathcal{P}$ as described above. The algorithm starts in the state $\xi = |0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\xi\rangle_{\mathcal{H}}$, and performs the following transformations:

1. Map $|0\rangle_{\mathcal{K}}$ into the uniform superposition $\frac{1}{\sqrt{K}}\sum_{t=0}^{K-1}|t\rangle_{\mathcal{K}}$.
2. For $t = 0, 1, \ldots, K-1$:
   (a) Execute S on $\mathcal{H}\oplus\mathcal{L}$ conditioned on $|t\rangle_{\mathcal{K}}$.
   (b) Conditioned on $|1\rangle_{\mathcal{P}}$, replace $|t\rangle_{\mathcal{K}}$ by $|t+1\rangle_{\mathcal{K}}$ (where $|K\rangle_{\mathcal{K}}$ is equal to $|0\rangle_{\mathcal{K}}$).
3. Run Step 1 in reverse.

Clearly, the algorithm conditionally executes S exactly K times. As described now, the algorithm takes time $\mathcal{O}(K\log K)$, but it is implementable in time $\mathcal{O}(K)$ using Lemma 4.6.

Let us prove correctness. We write $v = v(S,\xi)$ and $\tau = \tau(S,\xi)$. After Step 1, the algorithm is in the state

$$\frac{1}{\sqrt{K}}\sum_{i=0}^{K-1}|i\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\xi\rangle_{\mathcal{H}}. \tag{5.5}$$

We perform a perturbation in the sense of Lemma 4.5 and assume the algorithm is instead in the state

$$\frac{1}{\sqrt{K}}\sum_{i=0}^{K-1}|i\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\xi\rangle_{\mathcal{H}} + \frac{1}{\sqrt{K}}|0\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}}. \tag{5.6}$$

On the t-th iteration of the loop, the transducer S on Step 2(a) transforms the part of the state

$$\frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\xi\rangle_{\mathcal{H}} + \frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}} \longmapsto \frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\tau\rangle_{\mathcal{H}} + \frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}}, \tag{5.7}$$

and on Step 2(b) the following transformation of the part of the state is performed:

$$\frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}} \longmapsto \frac{1}{\sqrt{K}}|t+1\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}}.$$

Therefore, after the execution of the loop in Step 2, we get the state

$$\frac{1}{\sqrt{K}}\sum_{i=0}^{K-1}|i\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\tau\rangle_{\mathcal{H}} + \frac{1}{\sqrt{K}}|0\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}}. \tag{5.8}$$

We perturb the state to

$$\frac{1}{\sqrt{K}}\sum_{i=0}^{K-1}|i\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\tau\rangle_{\mathcal{H}}. \tag{5.9}$$

After Step 3, we get the state $\tau = |0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\tau\rangle_{\mathcal{H}}$.

Note that the difference between the states in (5.5) and (5.6) has norm $\|v\|/\sqrt{K}$. The same is true for the difference between the states in (5.8) and (5.9). Therefore, by Lemma 4.5, the actual final state τ' of the algorithm satisfies

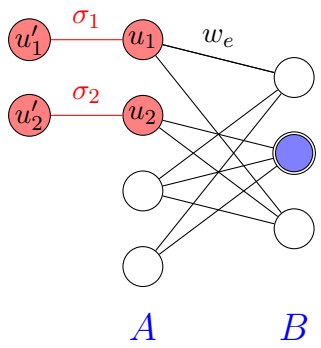
$$\|\tau' - \tau\| \leqslant 2\frac{\|v\|}{\sqrt{K}}$$

as required. □

6 Example I: Quantum Walks

In this section, we implement the electric quantum walk from [12] using the construction outlined in Section 3.2. See also a subsequent paper [15], where similar ideas are applied to search and to the Welded Tree problem [21].

Figure 6.1



An example of the extension of a graph for a quantum walk. The original graph contains two parts A and B of 4 and 3 vertices, respectively. The initial probability distribution is supported on two vertices $\{u_1, u_2\} \subseteq A$. The original edges E of the graph are black, the new ones E' are red. One marked vertex in B is coloured blue.

A quantum walk is described by a bipartite graph G, whose parts we denote by A and B. Let E be the set of edges of G. Each edge e of the graph is given a non-negative real *weight* w_e. We have some set $M \subseteq A \cup B$ of *marked* vertices. There is a subroutine Check that, for every vertex u, says whether it is marked. The goal of the quantum walk is to detect whether M is empty or not.

In the framework of electric quantum walks, the graph is extended as follows, see Figure 6.1. The quantum walk is tailored towards a specific initial probability distribution σ on A. Let $A_\sigma \subseteq A$ be the support of σ. For each $u \in A_\sigma$, we add a new vertex u' and a new dangling edge $u'u$ to the graph. The newly added vertices are *not* contained in B. Let E' be the set of newly added edges. For edges if E' we assume the weight $w_{u'u} = \sigma_u$.

We treat this construction as a transducer. The private space $\mathcal{L}$ of the quantum walk is $\mathbb{C}^E$. The public space $\mathcal{H}$ is $\mathbb{C}^{E'}$. The initial state ξ is given by

$$\xi = \sum_{u \in A_\sigma} \sqrt{\sigma_u}|u'u\rangle \in \mathcal{H}.$$

For a vertex u, let $\mathcal{L}_u$ denote the space spanned by all the edges incident to u (including the ones in E'). Define

$$\psi_u = \sum_{e : e \sim u} \sqrt{w_e}|e\rangle \in \mathcal{L}_u \tag{6.1}$$

where the sum is over all the edges incident to u.

For $U \subseteq A$ or $U \subseteq B$, let R_U denote the reflection of all ψ_u for $u \in U$, i.e., R_U acts as negation on the span of all these ψ_u and as identity on its orthogonal complement. We define the transducer, which depends on the set of marked vertices M, as

$$S_M = R_{B\setminus M} R_{A\setminus M}.$$

Each of $R_{A\setminus M}$ and $R_{B\setminus M}$ is decomposable into products of local reflections in $\mathcal{L}_u$ as u ranges over A and B, respectively. The corresponding local reflections are either identities for $u \in M$, or reflections of ψ_u for $u \notin M$. The implementation of S_M can be done using local reflections controlled by the Check subroutine.

Let

$$W = \sum_{e\in E} w_e$$

be the total weight of the graph. For $M \neq \varnothing$, let $R_{\sigma,M}$ be the minimum of

$$\sum_{e\in E} \frac{p_e^2}{w_e}, \tag{6.2}$$

over all flows $p = (p_e)$ on the graph where σ_u units of flow are injected in $u \in A_\sigma$, and the flow is collected at the vertices in M. The minimum is attained by the electrical flow, and $R_{\sigma,M}$ is the corresponding effective resistance.

Theorem 6.1. *The transducer S_M defined above transduces $\xi \rightsquigarrow -\xi$ if $M = \varnothing$, and $\xi \rightsquigarrow \xi$ otherwise. Its transduction complexity is*

$$W\big(S_\varnothing, \xi\big) = W \qquad \text{and} \qquad W\big(S_M, \xi\big) = R_{\sigma,M} \tag{6.3}$$

respectively.

Thus, the transduction action of the quantum walk encodes the answer to the detection problem in the phase, and this is done exactly. Let R denote the maximal effective resistance over all possible choices of $M \neq \varnothing$. We can rescale all w_i by the same factor so that the maximal transduction complexity in (6.3) becomes equal to $\sqrt{RW}$. By Theorem 3.2, the presence of marked elements can be detected, with bounded error, in $O(\sqrt{RW})$ executions of S_M. This coincides with the complexity estimate established in [12].

Proof of Theorem 6.1. Let us start with the case $M = \varnothing$. The initial coupling is

$$\xi \oplus v_\varnothing = \sum_{e\in E\cup E'} \sqrt{w_e}|e\rangle.$$

We have $\xi \oplus v_\varnothing = \sum_{u\in A} \psi_u$ and $v_\varnothing = \sum_{u\in B} \psi_u$. Hence, R_A reflects all $\xi \oplus v_\varnothing$, and R_B only reflects $v_\varnothing$. This gives us the following chain of transformations

$$\xi \oplus v_\varnothing \overset{R_A}{\longmapsto} -\xi \oplus -v_\varnothing \overset{R_B}{\longmapsto} -\xi \oplus v_\varnothing,$$

giving $\xi \overset{S_\varnothing}{\rightsquigarrow} -\xi$. Note that this chain of transformation adheres to (3.5).

Now assume $M \neq \varnothing$. Let p_e be a flow on the graph where σ_u units of flow are injected in u', and the flow is collected at M. To solve the sign ambiguity, we assume that all the edges are oriented towards A. This time, we define the catalyst v_M so that

$$\xi \oplus v_M = \sum_{e\in E\cup E'} \frac{p_e}{\sqrt{w_e}}|e\rangle. \tag{6.4}$$

Recall that $p_{u'u} = w_{u'u} = \sigma_u$, hence the above equation is satisfied in $\mathcal{H}$.

The projection of (6.4) onto $\mathcal{L}_u$ is given by

$$\sum_{e:e\sim u} \frac{p_e}{\sqrt{w_e}}|e\rangle. \tag{6.5}$$

This state is not changed by the corresponding local reflection in $\mathcal{L}_u$. Indeed if $u \in M$, the corresponding local reflection is the identity. If $u \notin M$, then, by the flow condition, the state in (6.5) is orthogonal to ψ_u in (6.1). Thus, neither $R_{A\setminus M}$ nor $R_{B\setminus M}$ change $\xi \oplus v_M$, hence, $\xi \overset{S_M}{\rightsquigarrow} \xi$. Note that in this case we adhere to (3.4).

The corresponding transduction complexities are as given in (6.3). The catalyst in the second case uses the flow p_e through the graph, which is not unique. We choose the minimal one as per Claim 5.2. □

7 Canonical Transducers

In this section, we define a specific form of transducers we will be using in this paper. The main point is in the application of the input oracle. Inspired by the construction in [17], we assume that the transducer first executes the input oracle, and then performs some input-independent unitary. Moreover, the input oracle is always applied to the private space of the transducer. These assumptions simplify many constructions, and any transducer can be transformed into the canonical form with a small overhead as shown later in Proposition 10.4.

7.1 Definition

Concerning the input oracle, the assumptions are similar to those in Section 4.1. The input oracle is a unitary O in some space $\mathcal{M}$, and we have unidirectional access to O. The oracle only acts on the local space $\mathcal{L}$ of the transducer. Let us decompose the latter in two parts $\mathcal{L} = \mathcal{L}^\circ \oplus \mathcal{L}^\bullet$, which stand for the work (non-query) and query parts. We also have $\mathcal{L}^\bullet = \mathcal{L}^\uparrow \otimes \mathcal{M}$ for some space $\mathcal{L}^\uparrow$. We denote the identity on $\mathcal{L}^\uparrow$ simply by I.

A canonical transducer $S = S(O)$ performs the following transformations, see also Figure 3.4:

- It executes the input oracle $I \otimes O$ on $\mathcal{L}^\bullet$. Similarly to (4.1), we call it a query and denote it by
$$\widetilde{O} = I_{\mathcal{H}} \oplus I^\circ \oplus I \otimes O, \tag{7.1}$$
where $I_{\mathcal{H}}$ and I° are identities on $\mathcal{H}$ and $\mathcal{L}^\circ$, respectively.
- It performs an input-independent work unitary S° on $\mathcal{H} \oplus \mathcal{L}$.

The decomposition $\mathcal{L} = \mathcal{L}^\circ \oplus \mathcal{L}^\bullet$ yields the decomposition $v = v^\circ \oplus v^\bullet$ of the catalyst. Thus, the action $S(O)$ of the transducer S on the input oracle O is given by the following chain of transformations:

$$S(O)\colon \xi \oplus v^\circ \oplus v^\bullet \overset{\widetilde{O}}{\longmapsto} \xi \oplus v^\circ \oplus (I \otimes O)v^\bullet \overset{S^\circ}{\longmapsto} \tau \oplus v^\circ \oplus v^\bullet. \tag{7.2}$$

Now we can make the following complexity-related definitions. The catalyst is $v = v(S, O, \xi)$.[11] The transduction complexity is

$$W(S, O, \xi) = \|v(S, O, \xi)\|^2. \tag{7.3}$$

[11] It is the same as $v(S(O), \xi)$ in the previous notation, however, we opted to $v(S, O, \xi)$ to reduce the number of brackets and to keep notation synchronised with [17].

The query state is $q(S,O,\xi) = v^\bullet = \Pi^\bullet v(S,O,\xi)$, where $\Pi^\bullet$ denotes the orthogonal projector onto $\mathcal{L}^\bullet$. The (Las Vegas) query complexity of the transducer is

$$L(S,O,\xi) = \big\|q(S,O,\xi)\big\|^2. \tag{7.4}$$

Note that formally the definitions $q(S,O,\xi)$ and $L(S,O,\xi)$ are in conflict with the same definitions (4.4) and (4.5) if S is considered as a *program* and not as a transducer. However, this should not cause a confusion. If the space $\mathcal{H}$ is not clear from the context, we will add it as a subscript as in Section 5.1.

Time complexity $T(S)$ of the transducer is the number of elementary operations required to implement the unitary S°. Note that we do not count the query towards time complexity of the transducer.

Finally, the definitions (7.3) and (7.4) can be extended to $\xi' \in \mathcal{E} \otimes \mathcal{H}$ using (3.6).

Note on Non-Uniqueness of Catalyst It is important to note that in this setting the non-uniqueness of the catalyst v discussed in Section 5.1 becomes very important. To understand why, consider again Example 5.3 from that section. This time, assume that $\mathcal{L}^\circ$ is spanned by $|1\rangle$ and $\mathcal{L}^\bullet$ by $|2\rangle$, the input oracle is $O = I$, and $S^\circ = S$ as defined previously.

The "right" catalyst $v = (|1\rangle + |2\rangle)/2$ for the initial state $\xi = |0\rangle$ suggests that $W(S,O,|0\rangle) = 1/2$ and $L(S,O,|0\rangle) = 1/4$. However, if we take $v = |1\rangle$, we get that $W(S,O,|0\rangle) = 1$ and $L(S,O,|0\rangle) = 0$. Therefore, there is no longer a single catalyst that minimises both the transduction and the query complexity. This is similar to usual algorithms, where time and query complexity can be minimised by different algorithms.

We solve this complication by implicitly assigning a specific catalyst $v(S,O,\xi)$ for every O and ξ of interest, that gives *both* $W(S,O,\xi)$ and $L(S,O,\xi)$ simultaneously. Of course, neither of the two are guaranteed to be minimal. It is possible to study the trade-off between the transduction and the query complexity for a fixed O and ξ, but we will not explicitly pursue that in this paper.

Implementation Details In terms of registers, as in Section 4.1, the separation $\mathcal{L} = \mathcal{L}^\circ \oplus \mathcal{L}^\bullet$ is indicated by the qubit $\mathcal{R}$. Now it makes sense to assume that the registers $\mathcal{H}$, $\mathcal{L}^\circ$ and $\mathcal{L}^\bullet$ are the same, the distinction being given by the values of the registers $\mathcal{P}$ and $\mathcal{R}$. We will usually place this common register as the unnamed last register in our expressions. In particular,

$$\xi \oplus v = |0\rangle_\mathcal{P}|0\rangle_\mathcal{R}|\xi\rangle + |1\rangle_\mathcal{P}|0\rangle_\mathcal{R}|v^\circ\rangle + |1\rangle_\mathcal{P}|1\rangle_\mathcal{R}|v^\bullet\rangle. \tag{7.5}$$

Note that the space $\mathcal{H}$ is indicated by $|0\rangle_\mathcal{P}|0\rangle_\mathcal{R}$ meaning that it is not acted on by the oracle. This allows us to implement the query as an application of O controlled by $|1\rangle_\mathcal{R}$, which is in accord with the convention established in Section 4.1.

We will also use registers $\mathcal{H}$ and $\mathcal{L}$ in the sense of Section 5.2, that is, containing $\mathcal{R}$. In particular, we can write the action of a canonical transducer (7.2) in registers like

$$S(O)\colon |0\rangle_\mathcal{P}|\xi\rangle_\mathcal{H} + |1\rangle_\mathcal{P}|v\rangle_\mathcal{L} \xmapsto{\widetilde{O}} |0\rangle_\mathcal{P}|\xi\rangle_\mathcal{H} + |1\rangle_\mathcal{P}\big|\widetilde{O}v\big\rangle_\mathcal{L} \xmapsto{S^\circ} |0\rangle_\mathcal{P}|\tau\rangle_\mathcal{H} + |1\rangle_\mathcal{P}|v\rangle_\mathcal{L}, \tag{7.6}$$

where we used shorthand

$$\widetilde{O}v = |0\rangle_\mathcal{R}|v^\circ\rangle + |1\rangle_\mathcal{R}\big|(I\otimes O)v^\bullet\big\rangle. \tag{7.7}$$

7.2 Multiple Input Oracles

Following [17], we can also allow multiple input oracles joined by direct sum:

$$O = O^{(1)} \oplus O^{(2)} \oplus \cdots \oplus O^{(r)}, \tag{7.8}$$

where the i-th input oracle $O^{(i)}$ acts in space $\mathcal{M}^{(i)}$ and $\mathcal{M} = \mathcal{M}^{(1)} \oplus \cdots \oplus \mathcal{M}^{(r)}$. We get the corresponding decomposition of the query state:

$$q(S, O, \xi) = q^{(1)}(S, O, \xi) \oplus \cdots \oplus q^{(r)}(S, O, \xi), \tag{7.9}$$

where $q^{(i)}(S, O, \xi)$ is the *partial query state* of the i-th input oracle. This also gives query complexities of the individual oracles:

$$L^{(i)}(S, O, \xi) = \|q^{(i)}(S, O, \xi)\|^2.$$

It makes sense to tweak the notation assumed earlier in Section 7.1 to make it in line with Section 4.2. We assume the register $\mathcal{R}$ can hold an integer from 0 to r, where $|i\rangle_{\mathcal{R}}$ with $i > 0$ indicates the space of the i-th input oracle. Thus, in place of (7.5), we have

$$\xi \oplus v = |0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\xi\rangle + |1\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|v^\circ\rangle + |1\rangle_{\mathcal{P}} \sum_{i=1}^{r} |i\rangle_{\mathcal{R}}|v^{(i)}\rangle \tag{7.10}$$

with $v^{(i)} = q^{(i)}(S, O, \xi)$. To apply the i-th input oracle $O^{(i)}$, it suffices to condition it on $|i\rangle_{\mathcal{R}}$. The action of a canonical transducer stays given by (7.6), where, this time,

$$\widetilde{O}v = |0\rangle_{\mathcal{R}}|v^\circ\rangle + \sum_{i=1}^{r} |i\rangle_{\mathcal{R}} \Big| \big(I \otimes O^{(i)}\big) v^{(i)} \Big\rangle. \tag{7.11}$$

For notational convenience, we will assume a single input oracle in most of the paper. The case of multiple oracles can be obtained using the decomposition of the query state in (7.9), which contains all the necessary information.

7.3 Reducing the Number of Oracle Calls

One problem with the algorithm in Theorem 5.5 is that, when applied to a canonical transducer, the input oracle O is executed the same number of times as the work unitary S°. This is suboptimal as the transduction complexity can be much larger than the query complexity. In this section, we describe a query-efficient implementation, which can also handle multiple input oracles.

Let S be a canonical transducer with r input oracles joined into one oracle O via direct sum as in (7.8). In the following theorem, we assume the spaces $\mathcal{H}$, $\mathcal{L}^\circ$, $\mathcal{L}^\uparrow$, $\mathcal{M}^{(1)}, \ldots, \mathcal{M}^{(r)}$ are fixed, while the operators S° and $O^{(1)}, \ldots, O^{(r)}$ can vary.

Theorem 7.1 (Query-Optimal Implementation of Transducers)**.** *Let $K \geqslant K^{(1)}, \ldots, K^{(r)}$ be positive integers, which we assume to be powers of 2 for simplicity. There exists a quantum algorithm that conditionally executes S° as a black box K times, makes $K^{(i)}$ queries to the i-th input oracle $O^{(i)}$, and uses $\mathcal{O}(K + K^{(1)} + \cdots + K^{(r)}) \log r$ other elementary operations. For each S°, $O^{(i)}$, and initial state ξ, the algorithm transforms ξ into τ' such that*

$$\|\tau' - \tau(S, O, \xi)\| \leqslant \frac{2}{\sqrt{K}} \sqrt{W(S, O, \xi) + \sum_{i=1}^{r} \left(\frac{K}{K^{(i)}} - 1 \right) L^{(i)}(S, O, \xi)}. \tag{7.12}$$

Theorem 5.5 is a special case of this theorem with all $K^{(i)}$ equal to K. Observe that the number of elementary operations is equal to the total number of invocations of S° and $O^{(i)}$ times $\log r$. It is highly unlikely that this part of the algorithm would dominate its time complexity.

Proof of Theorem 7.1. The proof is an extension of that of Theorem 5.5. Its outline was already given in Section 3.3; see in particular Figure 3.5.

Let us define $D^{(i)} = K/K^{(i)}$, which is a power of 2. The space of the algorithm is $\mathcal{K}\otimes(\mathcal{H}\oplus\mathcal{L})$, where $\mathcal{K}$ is a K-qudit. The register $\mathcal{H}\oplus\mathcal{L}$ contains the registers $\mathcal{P}$ and $\mathcal{R}$ as described above. The algorithm starts its work in the state $\xi = |0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\xi\rangle$. Its steps are as follows.

1. Map $|0\rangle_{\mathcal{K}}$ into the uniform superposition $\frac{1}{\sqrt{K}}\sum_{t=0}^{K-1}|t\rangle_{\mathcal{K}}$.
2. For $t = 0, 1, \ldots, K-1$:
   (a) For each $i = 1, \ldots, r$:
      - if t is divisible by $D^{(i)}$, execute the input oracle $O^{(i)}$ conditioned on $|i\rangle_{\mathcal{R}}$.
   (b) Execute the work unitary S° on $\mathcal{H}\oplus\mathcal{L}$ conditioned on $|t\rangle_{\mathcal{K}}$.
   (c) Conditioned on $|1\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}$, replace $|t\rangle_{\mathcal{K}}$ by $|t+1\rangle_{\mathcal{K}}$.
   (d) For each $i = 1, \ldots, r$:
      - if $t+1$ is divisible by $D^{(i)}$, add $D^{(i)}$ to $\mathcal{K}$ conditioned on $|i\rangle_{\mathcal{R}}$. The operation is performed modulo K.
3. Run Step 1 in reverse.

The analysis is similar to that in the proof of Theorem 5.5. Now we assume that after Step 1, instead of the state $\frac{1}{\sqrt{K}}\sum_{t=0}^{K-1}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\xi\rangle$, we are in the state

$$\frac{1}{\sqrt{K}}\sum_{t=0}^{K-1}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\xi\rangle + \frac{1}{\sqrt{K}}|1\rangle_{\mathcal{P}}\Big[|0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{R}}|v^\circ\rangle + \sum_{i=1}^{r}\sum_{t=0}^{D^{(i)}-1}|t\rangle_{\mathcal{K}}|i\rangle_{\mathcal{R}}|v^{(i)}\rangle\Big]. \tag{7.13}$$

Since $t = 0$ is divisible by all $D^{(i)}$, after Step 2(a) of the first iteration of the loop, we have the state

$$\frac{1}{\sqrt{K}}\sum_{t=0}^{K-1}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\xi\rangle + \frac{1}{\sqrt{K}}|1\rangle_{\mathcal{P}}\Big[|0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{R}}|v^\circ\rangle + \sum_{i=1}^{r}\sum_{t=0}^{D^{(i)}-1}|t\rangle_{\mathcal{K}}|i\rangle_{\mathcal{R}}|(I\otimes O^{(i)})v^{(i)}\rangle\Big].$$

The crucial observation is that on each iteration of the loop on step 2(b), the following transformation is performed. The part of the state

$$\frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}\Big[|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\xi\rangle + |1\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|v^\circ\rangle + |1\rangle_{\mathcal{P}}\sum_{i=1}^{r}|i\rangle_{\mathcal{R}}|(I\otimes O^{(i)})v^{(i)}\rangle\Big] \tag{7.14}$$

gets mapped by S° into

$$\frac{1}{\sqrt{K}}|t\rangle_{\mathcal{K}}\Big[|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\tau\rangle + |1\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|v^\circ\rangle + |1\rangle_{\mathcal{P}}\sum_{i=1}^{r}|i\rangle_{\mathcal{R}}|v^{(i)}\rangle\Big], \tag{7.15}$$

where we used (7.6) with (7.11).

If $t = cD^{(i)} - 1$ for some integer c, then on Step 2(d), we perform the transformation of the part of the state

$$\frac{1}{\sqrt{K}}|1\rangle_{\mathcal{P}}|i\rangle_{\mathcal{R}} \sum_{t=0}^{D^{(i)}-1} \Big|(c-1)D^{(i)} + t\Big\rangle_{\mathcal{K}}|v^{(i)}\rangle \longmapsto \frac{1}{\sqrt{K}}|1\rangle_{\mathcal{P}}|i\rangle_{\mathcal{R}} \sum_{t=0}^{D^{(i)}-1} \Big|cD^{(i)} + t\Big\rangle_{\mathcal{K}}|v^{(i)}\rangle, \quad (7.16)$$

which is then mapped on Step 2(a) of the next iteration into

$$\frac{1}{\sqrt{K}}|1\rangle_{\mathcal{P}}|i\rangle_{\mathcal{R}} \sum_{t=0}^{D^{(i)}-1} \Big|cD^{(i)} + t\Big\rangle_{\mathcal{K}}\Big|(I \otimes O^{(i)})v^{(i)}\Big\rangle.$$

Therefore, after all the K iterations of the loop in Step 2, we result in the state

$$\frac{1}{\sqrt{K}} \sum_{t=0}^{K-1} |t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\tau\rangle + \frac{1}{\sqrt{K}}|1\rangle_{\mathcal{P}}\Big[|0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{R}}|v^{\circ}\rangle + \sum_{i=1}^{r} \sum_{t=0}^{D^{(i)}-1} |t\rangle_{\mathcal{K}}|i\rangle_{\mathcal{R}}|v^{(i)}\rangle\Big].$$

After that, we finish as in Theorem 5.5, by assuming we are in the state $\frac{1}{\sqrt{K}}\sum_{t=0}^{K-1}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|\tau\rangle$ instead, and applying Step 3.

The total perturbation of the algorithm is

$$\frac{2}{\sqrt{K}}\Big\||0\rangle_{\mathcal{K}}|0\rangle_{\mathcal{R}}|v^{\circ}\rangle + \sum_{i=1}^{r} \sum_{t=0}^{D^{(i)}-1} |t\rangle_{\mathcal{K}}|i\rangle_{\mathcal{R}}|v^{(i)}\rangle\Big\|, \quad (7.17)$$

which is equal to the right-hand side of (7.12).

The claim on the number of executions of S° and $O^{(i)}$ in the algorithm is obvious. Besides that, it is trivial to implement the algorithm in $\mathcal{O}(K + K^{(1)} + \cdots + K^{(r)}) \log K \log r$ elementary operations, where the $\log r$ factors comes from the necessity to index one of the r input oracles. Using a slight modification of Lemma 4.6, the algorithm can be implemented in $\mathcal{O}(K + K^{(1)} + \cdots + K^{(r)}) \log r$ elementary operations. One crucial point here is that when $t = cD^{(i)} - 1$, addition of $D^{(i)}$ on Step 2(d) is equivalent to the replacement of $c-1$ by c in the highest qubits of the register $\mathcal{K}$ as indicated by (7.16). We omit the details. □

The following corollary, which is equivalent to Theorem 3.3, is easier to apply.

Corollary 7.2. *Assume $r = \mathcal{O}(1)$, and let $\varepsilon, W, L^{(1)}, \ldots, L^{(r)} > 0$ be parameters. There exists a quantum algorithm that conditionally executes S° as a black box $K = \mathcal{O}(1 + W/\varepsilon^2)$ times, makes $\mathcal{O}(L^{(i)}/\varepsilon^2)$ queries to the i-th input oracle $O^{(i)}$, and uses $\mathcal{O}(K)$ other elementary operations. The algorithm ε-approximately transforms ξ into $\tau(S, O, \xi)$ for all S, $O^{(i)}$, and ξ such that $W(S, O, \xi) \leqslant W$ and $L^{(i)}(S, O, \xi) \leqslant L^{(i)}$ for all i.*

Proof of Corollary 7.2. We first prove a relaxed version of the corollary, where we allow each input oracle to be called $\mathcal{O}\big(1 + L^{(i)}/\varepsilon^2\big)$ times. Afterwards, we show how to remove this assumption. We allow error $\varepsilon/2$ in this relaxed version.

If $W < \varepsilon^2/16$, the (relaxed) corollary follows from Theorem 5.5, as the algorithm then executes S° and each input oracle $K = 1$ times. Therefore, we will assume $W \geqslant \varepsilon^2/16$. Also, by definition we have that $W(S, O, \xi) \geqslant L^{(i)}(S, O, \xi)$. Therefore, we may assume that $W \geqslant L^{(i)}$, reducing $L^{(i)}$ otherwise.

We intend to use Theorem 7.1. We take K as the smallest power of 2 exceeding $32(r + 1)W/\varepsilon^2$. In particular, $K = \Theta(W/\varepsilon^2)$ by our assumption on W and $r = \mathcal{O}(1)$. We take $K^{(i)}$ as the largest power of 2 that does not exceed $\max\big\{1, KL^{(i)}/W\big\}$. It satisfies $K^{(i)} \leqslant K$ as

required. On the other hand, $K^{(i)} \geqslant KL^{(i)}/(2W)$ implying $K/K^{(i)} \leqslant 2W/L^{(i)}$, which gives the error estimate (7.12) at most

$$\frac{2}{\sqrt{K}}\sqrt{2(r+1)W} \leqslant \frac{2\sqrt{2(r+1)W}}{\sqrt{32(r+1)W/\varepsilon^2}} \leqslant \varepsilon/2.$$

If $K^{(i)} > 1$, we get $K^{(i)} \leqslant KL^{(i)}/W = \mathcal{O}(L^{(i)}/\varepsilon^2)$, which finishes the proof of the relaxed statement of the corollary.

In the following, we assume we used Theorem 7.1 in the proof, the case of Theorem 5.5 being similar. Consider all the values of i such that $K^{(i)} = 1$. By the proof of Theorem 7.1, the input oracles are applied to the perturbation added in (7.13), also the norm of the perturbation is at most $\varepsilon/4$ (*cf.* (7.17)).

To get the original statement of the corollary, we do not apply all the input oracles $O^{(i)}$ with $K^{(i)} = 1$. As they are applied once in the algorithm, this gives additional perturbation of size at most $\varepsilon/2$. Combining with the perturbation $\varepsilon/2$ of the relaxed algorithm itself, we get an estimate of the total error of at most ε. □

8 Example II: Adversary Bound

As mentioned in the introduction, the construction of transducers is based on the implementation of the adversary bound in [17]. The quantum adversary bound was first developed as a powerful tool for proving quantum query lower bounds. However, it was later extended to include upper bounds as well, and we consider the latter in this paper. For more detail on the adversary bound, refer to the introduction of [17] and the references therein. We first consider the general case of state conversion with unidirectional unitary input oracles, and then move on to more usual function evaluation problems.

8.1 State Conversion

We consider the adversary bound for state conversion from [17]. In the state conversion problem, we have a collection of pairs $\xi_x \mapsto \tau_x$ of states in $\mathcal{H}$ and input oracles $O_x \colon \mathcal{M} \to \mathcal{M}$, where x ranges over some finite set D. The task is to develop an algorithm A such that $A(O_x)\xi_x = \tau_x$ for all x. The goal is to minimise $L(A, O_x, \xi_x)$. The corresponding adversary bound is the following multi-objective optimisation problem:

$$\begin{aligned}
&\text{minimise} && \big(\|v_x\|^2\big)_{x\in D} && && (8.1a)\\
&\text{subject to} && \langle \xi_x, \xi_y\rangle - \langle \tau_x, \tau_y\rangle = \big\langle v_x,\ (I_{\mathcal{W}} \otimes (I_{\mathcal{M}} - O_x^* O_y))v_y\big\rangle && \text{for all } x, y \in D; && (8.1b)\\
& && \mathcal{W} \text{ is a vector space}, && v_x \in \mathcal{W} \otimes \mathcal{M}. && (8.1c)
\end{aligned}$$

A canonical transducer S_v can be obtained from any feasible solution $v = (v_x)$ to this problem. It works as follows (with $I = I_{\mathcal{W}}$):

$$\xi_x \oplus v_x \overset{I\otimes O_x}{\longmapsto} \xi_x \oplus (I \otimes O_x)v_x \overset{S_v^\circ}{\longmapsto} \tau_x \oplus v_x,$$

where S_v° is an input-independent unitary whose existence is assured by (8.1b), as the latter can be rewritten as

$$\langle \xi_x, \xi_y\rangle + \big\langle (I \otimes O_x)v_x,\ (I \otimes O_y)v_y\big\rangle = \langle \tau_x, \tau_y\rangle + \langle v_x, v_y\rangle,$$

and two state collections with the same combination of inner products always admit such a state-independent transforming unitary. Thus we have a transduction $\xi_x \overset{S_v(O_x)}{\rightsquigarrow} \tau_x$ with the transduction and the query complexities satisfying

$$W(S_v, O_x, \xi_x) = L(S_v, O_x, \xi_x) = \|v_x\|^2, \qquad q(S_v, O_x, \xi_x) = v_x.$$

As shown in [17], this perfectly captures Las Vegas query complexity of state conversion. Note that canonical transducers with empty non-query space $\mathcal{L}^\circ$ are essentially equivalent to this construction.

8.2 Function Evaluation

Now we describe the usual case of function evaluation. These results can be derived from [17] and [14]. First, we define the formalism behind function-evaluating transducers, and then move on to the adversary bound.

Let $f\colon D \to [p]$ be a function with domain $D \subseteq [q]^n$. We want to construct a transducer S_f that evaluates f. We assume the state-generating settings from Section 4.3, which means that, for every $x \in D$, with bidirectional access to the input oracle $O_x\colon |i\rangle|0\rangle \mapsto |i\rangle|x_i\rangle$, the transducer S_f has to perform the transduction $|0\rangle \rightsquigarrow |f(x)\rangle$. Again, we cast this as unidirectional access to $\overleftrightarrow{O_x}$ from (4.3).

Similarly to (4.10), we write

$$W_x(S) = W\big(S, \overleftrightarrow{O_x}, |0\rangle\big) \qquad \text{and} \qquad W(S) = \max_{x\in D} W_x(S). \tag{8.2}$$

We use similar notation for L and $L^{(i)}$.

There is a slight discrepancy between various existing definitions of the adversary bound $\mathrm{Adv}^{\pm}(f)$ for non-Boolean functions, in the sense that they differ by a factor of at most 2 (see, e.g., Section 3 of [30].) We adopt the formulation from [14], which reads in notation of that paper as

$$\mathrm{Adv}^{\pm}(f) = \gamma_2\Big(1_{f(x)\neq f(y)} \,\Big|\, \bigoplus\nolimits_{i\in[n]} 1_{x_i\neq y_i}\Big)_{x,y\in D},$$

and which is equivalent to $\gamma_2(J - F|\Delta)$ in notations of [30]. An explicit definition of $\mathrm{Adv}^{\pm}(f)$ is:[12]

$$\text{minimise} \quad \max_{x\in D} \frac{1}{2}\sum_{i=1}^{n}\Big(\|u_{x,i}\|^2 + \|v_{x,i}\|^2\Big) \tag{8.3a}$$

$$\text{subject to} \quad 1_{f(x)\neq f(y)} = \sum_{i:x_i\neq y_i} \langle u_{x,i}, v_{y,i}\rangle \qquad \text{for all } x, y \in D; \tag{8.3b}$$

$$\mathcal{W} \text{ is a vector space}, \qquad u_{x,i}, v_{x,i} \in \mathcal{W}. \tag{8.3c}$$

Let us now describe the canonical transducer $S_{u,v}$ corresponding to a feasible solution $u_{x,i}, v_{x,i}$ of (8.3). Its local space $\mathcal{L} = \mathcal{L}^\bullet$ is of the form $\mathcal{W}\otimes\mathcal{R}\otimes\mathcal{B}\otimes\mathcal{Q}$. Here $\mathcal{W}$ acts as $\mathcal{L}^\uparrow$ in notation of Section 7.1, $\mathcal{R}$ is an n-qudit indicating the index of the input variable, $\mathcal{B}$ is a qubit indicating direction of the query, and $\mathcal{Q}$ is a q-qudit storing the output of the query. We

[12] The usual definition has $\max\{\sum_{i=1}^n \|u_{x,i}\|^2, \sum_{i=1}^n \|v_{x,i}\|^2\}$ in the objective instead of $\frac{1}{2}\Big(\sum_{i=1}^n \|u_{x,i}\|^2 + \|v_{x,i}\|^2\Big)$. The two formulations are equivalent [17]. But even a priori, our formulation does not exceed the usual formulation, and, since we are interested in upper bounds, supersedes the latter.

use $\uparrow$ and $\downarrow$ to denote the basis states of $\mathcal{B}$. The first one stands for the direct, and the second one for the inverse query. The input oracle acts on $\mathcal{R} \otimes \mathcal{B} \otimes \mathcal{Q}$ as

$$\overleftrightarrow{O_x} = \bigoplus_{i \in [n]} \overleftrightarrow{O_{x,i}}, \tag{8.4}$$

where

$$O_{x,i}\colon \quad \mathcal{Q} \to \mathcal{Q}, \quad |0\rangle \mapsto |x_i\rangle \tag{8.5}$$

is the ith constituent of the input oracle.

Define the following vectors in $\mathcal{W}$:

$$v^{\uparrow}_{x,i} = \frac{u_{x,i} + v_{x,i}}{2} \qquad \text{and} \qquad v^{\downarrow}_{x,i} = \frac{u_{x,i} - v_{x,i}}{2}.$$

They possess the following important property:

$$\langle v^{\uparrow}_{x,i}, v^{\uparrow}_{y,i} \rangle - \langle v^{\downarrow}_{x,i}, v^{\downarrow}_{y,i} \rangle = \frac{\langle u_{x,i}, v_{y,i} \rangle + \langle v_{x,i}, u_{y,i} \rangle}{2}. \tag{8.6}$$

The catalyst for the input x is

$$v_x = \sum_{i \in [n]} |i\rangle_{\mathcal{R}} \Big[|\uparrow\rangle_{\mathcal{B}} |0\rangle_{\mathcal{Q}} |v^{\uparrow}_{x,i}\rangle_{\mathcal{W}} + |\downarrow\rangle_{\mathcal{B}} |x_i\rangle_{\mathcal{Q}} |v^{\downarrow}_{x,i}\rangle_{\mathcal{W}} \Big]. \tag{8.7}$$

The transducer starts in $\xi_x \oplus v_x = |0\rangle \oplus |v_x\rangle$. It applies the input oracle (8.4), which gives the state

$$\psi_x = |0\rangle \oplus \sum_{i \in [n]} |i\rangle_{\mathcal{R}} \Big[|\uparrow\rangle_{\mathcal{B}} |x_i\rangle_{\mathcal{Q}} |v^{\uparrow}_{x,i}\rangle_{\mathcal{W}} + |\downarrow\rangle_{\mathcal{B}} |0\rangle_{\mathcal{Q}} |v^{\downarrow}_{x,i}\rangle_{\mathcal{W}} \Big]. \tag{8.8}$$

The construction then follows from the following claim.

Claim 8.1. *There exists an input-independent unitary $S^{\circ}_{u,v}$ that maps the state ψ_x from (8.8) into $|f(x)\rangle \oplus |v_x\rangle$ for all x.*

Proof. Indeed, for a pair of $x, y \in D$, we have

$$\langle \psi_x, \psi_y \rangle = 1 + \sum_{i \in [n]} \big[1_{x_i = y_i} \langle v^{\uparrow}_{x,i}, v^{\uparrow}_{y,i} \rangle + \langle v^{\downarrow}_{x,i}, v^{\downarrow}_{y,i} \rangle \big]. \tag{8.9}$$

On the other hand, the inner product between $|f(x)\rangle \oplus |v_x\rangle$ and $|f(y)\rangle \oplus |v_y\rangle$ is

$$1_{f(x)=f(y)} + \sum_{i \in [n]} \big[\langle v^{\uparrow}_{x,i}, v^{\uparrow}_{y,i} \rangle + 1_{x_i = y_i} \langle v^{\downarrow}_{x,i}, v^{\downarrow}_{y,i} \rangle \big]. \tag{8.10}$$

To establish the existence of the unitary $S^{\circ}_{u,v}$ it suffices to show that (8.9) and (8.10) are equal for all $x, y \in D$. Subtracting the latter from the former gives us

$$1_{f(x) \neq f(y)} - \sum_{i : x_i \neq y_i} \big[\langle v^{\uparrow}_{x,i}, v^{\uparrow}_{y,i} \rangle - \langle v^{\downarrow}_{x,i}, v^{\downarrow}_{y,i} \rangle \big] = 1_{f(x) \neq f(y)} - \frac{1}{2} \sum_{i : x_i \neq y_i} \big[\langle u_{x,i}, v_{y,i} \rangle + \langle v_{x,i}, u_{y,i} \rangle \big] = 0$$

using (8.6) and (8.3b). □

Thus, we have that $S_{u,v}$ on the input oracle $\overleftrightarrow{O_x}$ transduces $|0\rangle$ into $|f(x)\rangle$. If we consider $\overleftrightarrow{O_x}$ as a direct sum of n input oracles as in (8.4), we get the partial query states

$$q_x^{(i)}(S_{u,v}) = v_{x,i}^{\uparrow} \oplus v_{x,i}^{\downarrow},$$

with

$$L_x^{(i)}(S_{u,v}) = \|v_{x,i}^{\uparrow}\|^2 + \|v_{x,i}^{\downarrow}\|^2 = \frac{\|u_{x,i}\|^2 + \|v_{x,i}\|^2}{2}$$

by the parallelogram identity. Finally, the transduction complexity and the total query complexity is

$$W_x(S_{u,v}) = L_x(S_{u,v}) = \frac{1}{2}\left(\sum_{i=1}^{n} \|u_{x,i}\|^2 + \|v_{x,i}\|^2\right)$$

which is in the objective of (8.3a). This can be summarised as

Theorem 8.2. *For every function $f\colon D \to [p]$ with $D \subseteq [q]^n$, there exists a canonical transducer S_f evaluating the function f and whose transduction complexity is bounded by $\mathrm{Adv}^{\pm}(f)$. In more detail, the admissible subspace of S_f is $|0\rangle$; for every $x \in D$, S_f transduces $|0\rangle \rightsquigarrow |f(x)\rangle$ with bidirectional access to the input oracle O_x encoding the input string x; and $W_x(S_f) = L_x(S_f) \leqslant \mathrm{Adv}^{\pm}(f)$. Moreover, the catalyst of the transducer S_f is as in (8.7). In particular, it executes the input oracle only on its admissible subspace.*

9 Composition of Transducers

In this section, we describe basic properties of transducers. In particular, we show how to combine simple transducers in order to obtain more complex ones. This is akin to quantum algorithms being build out of elementary operations and subroutines. In this section, we mostly focus on the circuit model of computation.

9.1 Basic Properties

From Theorem 5.1, it follows that, for a fixed S and O, the mappings $\xi \mapsto v(S, O, \xi)$ and $\xi \mapsto q(S, O, \xi)$ are linear. In particular, for $c \in \mathbb{C}$, we have

$$W(S, O, c\xi) = |c|^2 W(S, O, \xi) \qquad \text{and} \qquad L^{(i)}(S, O, c\xi) = |c|^2 L^{(i)}(S, O, \xi). \tag{9.1}$$

Notice, however, that it is *not* necessarily true that $W(S, O, \xi_1 + \xi_2) = W(S, O, \xi_1) + W(S, O, \xi_2)$ even for orthogonal ξ_1 and ξ_2. On the other hand, it *is* the case that for $\xi_1 \in \mathcal{E}_1 \otimes \mathcal{H}$ and $\xi_2 \in \mathcal{E}_2 \otimes \mathcal{H}$, we have

$$W(S, O, \xi_1 \oplus \xi_2) = W(S, O, \xi_1) + W(S, O, \xi_2),$$

using the extended definition of (3.6).

Proposition 9.1 (Inverse)**.** *For a canonical transducer S, the inverse transducer S^{-1} satisfies $\tau \overset{S^{-1}(O^*)}{\rightsquigarrow} \xi$ whenever $\xi \overset{S(O)}{\rightsquigarrow} \tau$. Moreover, S^{-1} can be implemented in the canonical form, it has the same time complexity as S,*

$$W(S^{-1}, O^*, \tau) = W(S, O, \xi), \qquad \textit{and} \qquad q(S^{-1}, O^*, \tau) = (I \otimes O) q(S, O, \xi).$$

Proof. Let $v = v^\circ \oplus v^\bullet$ be the catalyst of the transduction $\xi \overset{S}{\rightsquigarrow} \tau$. From (5.1), it is clear that if $S(O)$ maps $\xi \oplus v \mapsto \tau \oplus v$, then $S(O)^*$ maps $\tau \oplus v \mapsto \xi \oplus v$, hence, transduces τ into ξ with the same catalyst. One problem is that its action, as the inverse of (7.2),

$$\tau \oplus v^\circ \oplus v^\bullet \overset{(S^\circ)^*}{\longmapsto} \xi \oplus v^\circ \oplus (I \otimes O)v^\bullet \overset{\widetilde{O}^*}{\longmapsto} \xi \oplus v^\circ \oplus v^\bullet.$$

is not in the canonical form. But we can take the following transducer S^{-1} in its stead:

$$\tau \oplus v^\circ \oplus (I \otimes O)v^\bullet \overset{\widetilde{O}^*}{\longmapsto} \tau \oplus v^\circ \oplus v^\bullet \overset{(S^\circ)^*}{\longmapsto} \xi \oplus v^\circ \oplus (I \otimes O)v^\bullet.$$

It is in the canonical form, and satisfies all the conditions. □

9.2 Alignment

In the remaining part of this section, we will study different ways of combining transducers $S_1, \dots, S_m$. First, we consider parallel composition of transducers $\bigoplus_i S_i$, where individual S_i act on orthogonal parts of the workspace. Then we move onto sequential composition $S_m * S_{m-1} * \cdots * S_1$, where they act on the same space one after another. Finally, we consider functional composition of two transducers, where the second transducer acts as an oracle for the first one.

We generally assume that all S_i use the same oracle O. This is without loss of generality since if they use different sets of input oracles, we can assume they all use the oracle O which is the direct sum of the union of these sets of oracles. Individual S_i will then just ignore the input oracles they are not using.

In principle, the spaces $\mathcal{H}_i$ and $\mathcal{L}_i$ can differ between different S_i, but we assume they are all embedded into some larger register $\mathcal{H}$, which also serves as $\mathcal{L}$ per our convention of Section 5.2. What is crucial, though, is that all S_i use the same privacy qubit $\mathcal{P}$, the same query register $\mathcal{R}$, and, most importantly, the oracle O is applied to the same subset of registers in all S_i. This is summarised by the following definition, *cf.* Remark 4.1:

Definition 9.2 (Alignment)**.** We say that canonical transducers $S_1, \dots, S_m$ are *aligned in the oracle* $O^{(i)}$ if the query of $O^{(i)}$ is conditioned on the same value $|i\rangle_{\mathcal{R}}$ of the same register and acts on the same subset of registers in all of them. We say that $S_1, \dots, S_m$ are *aligned* if they are aligned in all their oracles, and, additionally, use the same privacy qubit $\mathcal{P}$.

If the alignment condition is not satisfied, the transducers have to explicitly move their registers around to meet it. This might take time if the registers are lengthy.

9.3 Parallel Composition

Parallel composition of two or several quantum programs is their execution as a direct sum on orthogonal parts of the space of the algorithm. For transducers, parallel composition can be implemented in a straightforward way.

Definition 9.3 (Direct Sum of Transducers)**.** Let $S_1, \dots, S_m$ be canonical transducers. We assume they all use the same space $\mathcal{H} \oplus \mathcal{L}$, the same input oracle $O \colon \mathcal{M} \to \mathcal{M}$, and are aligned. In particular, the query $\widetilde{O} = I_{\mathcal{H}} \oplus I^\circ \oplus I \otimes O$ is given by (7.1) and is controlled by $|1\rangle_{\mathcal{R}}$ in all of them.

Let the register $\mathcal{J} = \mathbb{C}^m$. The direct sum $\bigoplus_i S_i$ is a canonical transducer in the space $\mathcal{J} \otimes (\mathcal{H} \oplus \mathcal{L}) = (\mathcal{J} \otimes \mathcal{H}) \oplus (\mathcal{J} \otimes \mathcal{L})$. It has the same input oracle O as all S_i. The query is again controlled by $|1\rangle_{\mathcal{R}}$. The work unitary of $\bigoplus_i S_i$ is $\bigoplus_i S_i^\circ$, where S_i° is the work unitary of S_i.

Proposition 9.4 (Parallel Composition)**.** *The canonical transducer $S = \bigoplus_i S_i$ from Definition 9.3 satisfies the following conditions. Assume $\xi_i \overset{S_i(O)}{\rightsquigarrow} \tau_i$ for all i. Then, $S(O)$ transduces $\xi = \bigoplus_i \xi_i$ into $\tau = \bigoplus_i \tau_i$. Moreover,*

$$W(S,O,\xi) = \sum_i W(S_i,O,\xi_i) \qquad \text{and} \qquad q(S,O,\xi) = \bigoplus_i q(S_i,O,\xi_i). \tag{9.2}$$

The time complexity of S is equal to the time complexity of implementing $\bigoplus_i S_i^\circ$.

Proof. Recall that all S_i are aligned, and, hence, use the same space, $\mathcal{P}$ is their common privacy qubit, and $\mathcal{R}$ their common query register. The privacy and the query registers of S will still be $\mathcal{P}$ and $\mathcal{R}$.

Let v_i be the catalyst of transduction $\xi_i \overset{S_i(O)}{\rightsquigarrow} \tau_i$. The initial coupling of S is

$$\xi \oplus v = \sum_{i=1}^{m} |i\rangle_{\mathcal{J}} \big[|0\rangle_{\mathcal{P}} |\xi_i\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} |v_i\rangle_{\mathcal{L}} \big]. \tag{9.3}$$

As required by canonicity, we first apply the input oracle O controlled by $|1\rangle_{\mathcal{R}}$. This has the effect that O is applied to all S_i in parallel. Then, conditioned on the value i in $\mathcal{J}$, we apply the work unitary S_i° to the last two registers. By the assumption $\xi_i \overset{S_i(O)}{\rightsquigarrow} \tau_i$, this gives

$$\sum_{i=1}^{m} |i\rangle_{\mathcal{J}} \big[|0\rangle_{\mathcal{P}} |\tau_i\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} |v_i\rangle_{\mathcal{L}} \big] = \tau \oplus v.$$

Eq. (9.2) follows from (9.3). □

Remark 9.5 (Different input oracles in S_i)**.** In Definition 9.3, we assume all S_i use the same input oracle O. It is, however, possible to allow each S_i to use its own input oracle O_i so that the total input oracle of $\bigoplus_i S_i$ is $\bigoplus_i O_i$, and $\xi_i \overset{S_i(O_i)}{\rightsquigarrow} \tau_i$. This is used, for instance, in iterated and composed functions. We account for this possibility by allowing $\mathcal{M}$ to contain $\mathcal{J}$, so that the input oracle of S_i has read-only access to the value of $|i\rangle_{\mathcal{E}}$. This does not interfere with the proof of Proposition 9.4, and we get the following identities instead of (9.2):

$$W(S,O,\xi) = \sum_i W(S_i,O_i,\xi_i) \qquad \text{and} \qquad q(S,O,\xi) = \bigoplus_i q(S_i,O_i,\xi_i). \tag{9.4}$$

The time complexity of implementing $\bigoplus_i S_i$ greatly depends on the structure of S_i and the computational model. For instance, if we assume the QRAG model, implementation of $\bigoplus S_i$ can be done in time essentially $\max_i T(S_i)$ as per Corollary 4.4. The important special case when all S_i are the same can be efficiently handled in the circuit model as well. We write it out explicitly for ease of referencing.

Definition 9.6 (Multiplication by Identity)**.** Let S be a canonical transducer with space $\mathcal{H} \oplus \mathcal{L}$, input oracle $O\colon \mathcal{M} \to \mathcal{M}$, and query given by (7.1). Let also $\mathcal{E}$ be a space, and $I_{\mathcal{E}}$ be the identity on $\mathcal{E}$. We define $I_{\mathcal{E}} \otimes S$ as a canonical transducer in the space $(\mathcal{E} \otimes \mathcal{H}) \oplus (\mathcal{E} \otimes \mathcal{L})$, with the work unitary $I_{\mathcal{E}} \otimes S^\circ$. In other words, $I_{\mathcal{E}} \otimes S = \bigoplus_i S$ where the summation is over the basis of $\mathcal{E}$.

Corollary 9.7. *In the settings of Definition 9.6, assume $\xi_i \overset{S(O)}{\rightsquigarrow} \tau_i$ as i ranges over the basis of $\mathcal{E}$. Then, $I_{\mathcal{E}} \otimes S(O)$ transduces $\xi = \bigoplus_i \xi_i$ into $\tau = \bigoplus_i \tau_i$. The complexities are given by (9.2) or (9.4) with all $S_i = S$. The time complexity is $T(I_{\mathcal{E}} \otimes S) = T(S)$.*

Note that complexities in (3.6) are the same as in this corollary with Eq. (9.2) used.

9.4 Sequential Composition

A quantum program (4.11) is a sequence of gates applied one after the other. Therefore, sequential composition is of prime importance in quantum algorithms. Let us formally define it for transducers.

Definition 9.8 (Sequential Composition)**.** Let $S_1, \dots, S_m$ be an aligned family of canonical transducers, all on the same public space $\mathcal{H}$ and the same input oracle O. Its sequential composition is a canonical transducer $S = S_m * S_{m-1} * \cdots * S_1$ on the same public space with the following property. For every sequence of transductions

$$\xi = \psi_1 \overset{S_1(O)}{\rightsquigarrow} \psi_2 \overset{S_2(O)}{\rightsquigarrow} \psi_3 \overset{S_3(O)}{\rightsquigarrow} \cdots \overset{S_{m-1}(O)}{\rightsquigarrow} \psi_m \overset{S_m(O)}{\rightsquigarrow} \psi_{m+1} = \tau, \tag{9.5}$$

$S(O)$ transduces the initial state ξ into the final state τ.

The above definition does not specify the implementation as we give two different implementations in this section. The first one is tailored towards the circuit model, and the second one towards the QRAG model. This division is not strict though.

Proposition 9.9 (Sequential Composition, Sequential Implementation)**.** *A sequential composition $S = S_m * S_{m-1} * \cdots * S_1$ as in Definition 9.8 can be implemented by a canonical transducer with the following parameters:*

$$W(S, O, \xi) = \sum_{t=1}^{m} W(S_t, O, \psi_t), \qquad q(S, O, \xi) = \bigoplus_{t=1}^{m} q(S_t, O, \psi_t), \tag{9.6}$$

and

$$T(S) = \mathcal{O}(m) + \sum_{t=1}^{m} T_{\mathrm{C}}(S_t), \tag{9.7}$$

where T_{C} is defined in Section 4.4.

Proof. The general idea is simple. We first apply the input oracle to make the queries in all S_t in parallel. After the query, we execute all S_t° one after the other, see Figure 9.1. Some care must be taken, however, so that transducers act inside their respective private spaces and do not interfere with private spaces of other transducers.

The transducer S uses the same privacy and query registers $\mathcal{P}$ and $\mathcal{R}$ as the family $S_1, \dots, S_m$. We additionally use an m-qudit $\mathcal{K}$, so that the local space of the transducer S_t is marked by the value $|t\rangle_{\mathcal{K}}$. Therefore, the space of S is of the form $\mathcal{K} \otimes (\mathcal{H} \oplus \mathcal{L})$. Its public space is still $\mathcal{H}$ embedded as $|0\rangle_{\mathcal{K}} \otimes \mathcal{H}$.

Let ξ and ψ_t be as in (9.5). For each t, let v_t be the catalyst of the transduction $\psi_t \overset{S_t(O)}{\rightsquigarrow} \psi_{t+1}$ from (9.5). For the transduction $\xi \overset{S(O)}{\rightsquigarrow} \tau$, we have the following initial coupling:

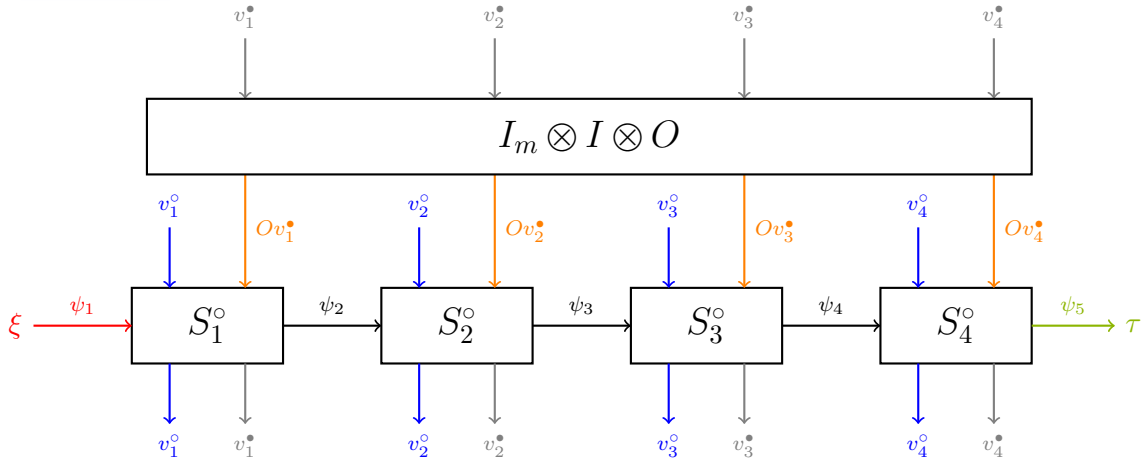
$$\xi \oplus v = |0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{K}}|\xi\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} \sum_{t=1}^{m} |t\rangle_{\mathcal{K}}|v_t\rangle_{\mathcal{L}}. \tag{9.8}$$

This already implies (9.6).

As required by the definition, the first operation is the application of the input oracle conditioned on $|1\rangle_{\mathcal{R}}$. This gives the state

$$|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{K}}|\psi_1\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} \sum_{t=1}^{m} |t\rangle_{\mathcal{K}}|\widetilde{O}v_t\rangle_{\mathcal{L}},$$

Figure 9.1



A graphical illustration to the construction of Proposition 9.9 with $m = 4$. For convenience of representation, we draw the catalyst states by vertical lines, not horizontal ones like in Figure 3.4. We also write $Ov_t^\bullet$ instead of $(I \otimes O)v_t^\bullet$ to save space.
We first apply the query $I_m \otimes I \otimes O$, which implements the queries in all S_t°. After that, we apply all of S_t° one after the other.

where $\widetilde{O}v_t$ is defined as in (7.7), and we used that $\xi = \psi_1$.

The work part of the transducer S is as follows. We assume the indexing of $\mathcal{K}$ is done modulo m.

- For $t = 1, \ldots, m$:
  1. Controlled by $|0\rangle_{\mathcal{P}}$, replace the value $|t-1\rangle_{\mathcal{K}}$ by $|t\rangle_{\mathcal{K}}$.
  2. Apply the work unitary S_t° controlled by $|t\rangle_{\mathcal{K}}$.

By induction and using (7.6), after ℓ iterations of the loop, we have the state

$$|0\rangle_{\mathcal{P}}|\ell\rangle_{\mathcal{K}}|\psi_{\ell+1}\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} \sum_{t=1}^{\ell} |t\rangle_{\mathcal{K}}|v_t\rangle_{\mathcal{L}} + |1\rangle_{\mathcal{P}} \sum_{t=\ell+1}^{m} |t\rangle_{\mathcal{K}}|\widetilde{O}v_t\rangle_{\mathcal{L}}.$$

And after execution of the whole transducer S, we have the state

$$|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{K}}|\tau\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}} \sum_{t=1}^{m} |t\rangle_{\mathcal{K}}|v_t\rangle_{\mathcal{L}} = \tau \oplus v,$$

where we used that $\psi_{m+1} = \tau$. The time complexity in (9.7) can be achieved using the direct-sum finite automaton of Lemma 4.6. □

The second implementation of the sequential composition is slightly less intuitive. We move all the intermediate states in (9.5) to the catalyst. This increases the catalyst size, but now all the operations S_t° can be implemented in parallel.

Proposition 9.10 (Sequential Composition, Parallel Implementation)**.** *A sequential composition $S = S_m * S_{m-1} * \cdots * S_1$ as in Definition 9.8 can be implemented by a transducer with the*

following parameters:

$$W(S,O,\xi)=\sum_{t=2}^{m}\|\psi_t\|^2+\sum_{t=1}^{m}W(S_t,O,\psi_t),\qquad q(S,O,\xi)=\bigoplus_{t=1}^{m}q(S_t,O,\psi_t)\tag{9.9}$$

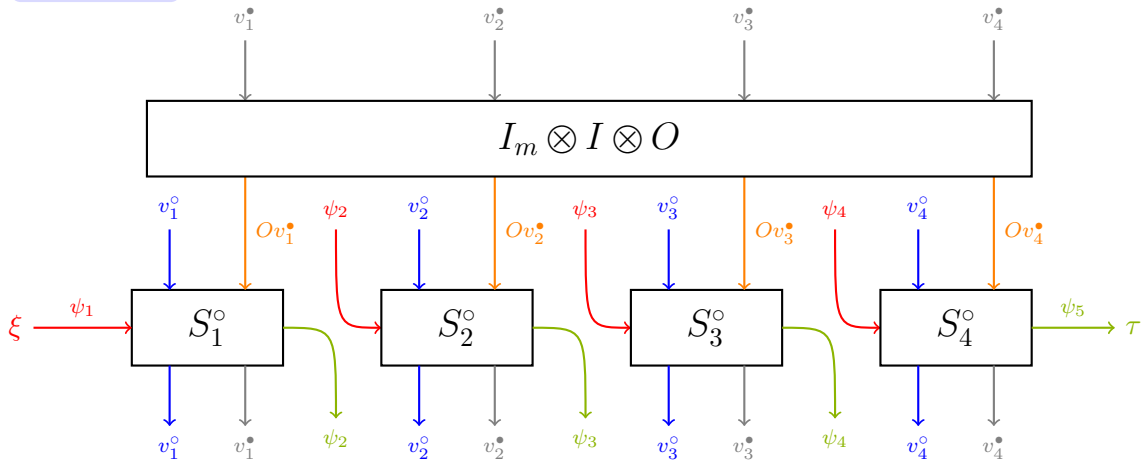
and

$$T(S)=\mathcal{O}(\log m)+T\Big(\bigoplus_t S_t^\circ\Big).\tag{9.10}$$

In the QRAG model, we usually replace $\mathcal{O}(\log m)$ in (9.10) by $\mathcal{O}(T_{\mathrm{R}})$.

Proof. The main new idea compared to Proposition 9.9 is that we do not compute the intermediate ψ_t from (9.5), but store the sequence $\psi_2,\dots,\psi_m$ in the catalyst. We apply all of S_t° in parallel, thus moving this sequence forward by one position, see Figure 9.2.

Figure 9.2



A graphical illustration to the construction of Proposition 9.10 with $m=4$. For convenience of representation, we draw the catalyst states by vertical lines, not horizontal ones like in Figure 3.4. We also write $Ov_t^\bullet$ instead of $(I\otimes O)v_t^\bullet$ to save space.
We first apply the query $I_m\otimes I\otimes O$, which implements the queries in all S_t°. After that, we apply all of S_t° in parallel. The initial state ξ becomes the input for S_1°. The input of S_t° for $t>1$ is taken from the catalyst, and the output becomes the catalyst for $t+1$, except for $t=m$, which yields the terminal state τ.

Again, we assume all S have space $\mathcal{H}\oplus\mathcal{L}$, have privacy qubit $\mathcal{P}$, and query register $\mathcal{R}$. The transducer S uses the same query register $\mathcal{R}$, but we introduce a new privacy qubit $\mathcal{P}'$. We also use an m-qudit $\mathcal{K}$, so that the space of S is of the form $\mathcal{K}\otimes\mathcal{P}'\otimes(\mathcal{H}\oplus\mathcal{L})$.

Let ξ and ψ_t be as in (9.5). For each t, let v_t be the catalyst of transduction $\psi_t \overset{S_t(O)}{\rightsquigarrow} \psi_{t+1}$. We start with the following initial coupling

$$\xi\oplus v=|0\rangle_{\mathcal{P}'}|\xi\rangle_{\mathcal{H}}+|1\rangle_{\mathcal{P}'}\Bigg[\sum_{t=2}^{m}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\psi_t\rangle_{\mathcal{H}}+\sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v_t\rangle_{\mathcal{L}}\Bigg].$$

This already gives (9.9). As required by the canonicity assumption, we first apply the input oracle. This gives

$$|0\rangle_{\mathcal{P}'}|\xi\rangle_{\mathcal{H}}+|1\rangle_{\mathcal{P}'}\Bigg[\sum_{t=2}^{m}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\psi_t\rangle_{\mathcal{H}}+\sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}\Big|\widetilde{O}v_t\Big\rangle_{\mathcal{L}}\Bigg],$$

where $\widetilde{O}$ is defined in (7.7). Here we use that $\mathcal{H}$ has value 0 in the register $\mathcal{R}$ and is not affected by the input oracle. Next, we exchange $|0\rangle_{\mathcal{P}'} \otimes \mathcal{H}$ with $|1\rangle_{\mathcal{P}'}|1\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}} \otimes \mathcal{H}$. Since $\xi = \psi_1$, this gives

$$|1\rangle_{\mathcal{P}'}\left[\sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\psi_t\rangle_{\mathcal{H}} + \sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}\left|\widetilde{O}v_t\right\rangle_{\mathcal{L}}\right].$$

Now we apply S_t° to the last two registers, controlled by the value in the register $\mathcal{K}$. In other words, we apply $\bigoplus_t S_t^{\circ}$. By (7.6), this gives

$$|1\rangle_{\mathcal{P}'}\left[\sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\psi_{t+1}\rangle_{\mathcal{H}} + \sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v_t\rangle_{\mathcal{L}}\right]. \tag{9.11}$$

Now we increment the value in the register $\mathcal{K}$ conditioned on $|0\rangle_{\mathcal{P}}$, and exchange $|0\rangle_{\mathcal{P}'} \otimes \mathcal{H}$ with $|1\rangle_{\mathcal{P}'}|m+1\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}} \otimes \mathcal{H}$. Since $\tau = \psi_{m+1}$, we get

$$|0\rangle_{\mathcal{P}'}|\tau\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}'}\left[\sum_{t=2}^{m}|t\rangle_{\mathcal{K}}|0\rangle_{\mathcal{P}}|\psi_t\rangle_{\mathcal{H}} + \sum_{t=1}^{m}|t\rangle_{\mathcal{K}}|1\rangle_{\mathcal{P}}|v_t\rangle_{\mathcal{L}}\right] = \tau \oplus v. \tag{9.12}$$

The time complexity estimate is obvious. □

9.5 Functional Composition

We defined functional composition in Section 2.1 as an algorithm where a subroutine is used to implement an oracle call. In the case of transducers, this falls under the transitivity of transduction, Proposition 5.4, as part of the transducer (the oracle call) is implemented as a transduction action of another transducer. In this section, we give an explicit construction for canonical transducers.

We assume the settings similar to Figure 2.1, except we use transducers S_A and S_B instead of programs A and B. The outer transducer S_A has two input oracles $O \oplus O'$ joined as in Section 7.2. The first one, O, is the global input oracle, and the second one, O', is the oracle realised as the transduction action of the inner transducer S_B. We assume both transducers are in the canonical form and aligned in the oracle O. We denote their public spaces by $\mathcal{H}_A$ and $\mathcal{H}_B$, respectively. If there are several subroutines implemented by different transducers, then, as in Figure 2.2, they can be joined via parallel composition of Proposition 9.4.

Proposition 9.11 (Functional Composition)**.** *Under the above assumptions, there exists a canonical transducer $S_A \circ S_B$ with the public space $\mathcal{H}_A$ and the oracle O, which satisfies the following properties.*

- *Its transduction action on input oracle O is equal to the transduction action of S_A on oracle $O \oplus O'$, where $O' = S_B(O)^{\circlearrowleft}_{\mathcal{H}_B}$.*
- *Its transduction complexity satisfies*

$$W(S_A \circ S_B, O, \xi) = W(S_A, O \oplus O', \xi) + W\Big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\Big). \tag{9.13}$$

- *Its total query state is*

$$q(S_A \circ S_B, O, \xi) = q^{(0)}(S_A, O \oplus O', \xi) \oplus q\Big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\Big). \tag{9.14}$$

- *Its time complexity is $T_{\mathrm{C}}(S_A) + T_{\mathrm{C}}(S_B)$.*

Here $q^{(0)}$ and $q^{(1)}$ denote the partial query states of S_A to the oracles O and O', respectively. We also used the extended versions of W and q from (3.6) for the transducer S_B.

Note that both Equations (9.13) and (9.14) are reminiscent of the "gold standard" (2.4).

Proof. Let $\mathcal{M}$ be the space of the input oracle O. The space of the transducer S_A is of the form $\mathcal{H}_A \oplus \mathcal{L}_A^\circ \oplus \mathcal{L}_A^\bullet \oplus \mathcal{L}'_A$, where the last two spaces are for the queries to O and O', respectively. In particular, $\mathcal{L}'_A = \mathcal{L}_A^\uparrow \otimes \mathcal{H}_B$ by the assumption that $S_B(O)$ implements O'.

Let $v = v^\circ \oplus v^\bullet \oplus v'$ be the catalyst for the transduction $\xi \rightsquigarrow \tau$ by $S_A(O \oplus O')$. That is, $v^\bullet = q^{(0)}(S_A, O \oplus O', \xi)$ and $v' = q^{(1)}(S_A, O \oplus O', \xi)$. By (7.6), the transducer S_A imposes the following chain of transformations in $\mathcal{H}_A \oplus \mathcal{L}_A^\circ \oplus \mathcal{L}_A^\bullet \oplus \mathcal{L}'_A$:

$$\xi \oplus v^\circ \oplus v^\bullet \oplus v' \overset{\tilde{O}}{\longmapsto} \xi \oplus v^\circ \oplus (I \otimes O)v^\bullet \oplus v' \overset{\widetilde{O'}}{\longmapsto} \xi \oplus v^\circ \oplus (I \otimes O)v^\bullet \oplus (I \otimes O')v' \overset{S_A^\circ}{\longmapsto} \tau \oplus v^\circ \oplus v^\bullet \oplus v', \tag{9.15}$$

where we separated the applications of O and O'. Recall that the identity I acts on $\mathcal{L}_A^\uparrow$.

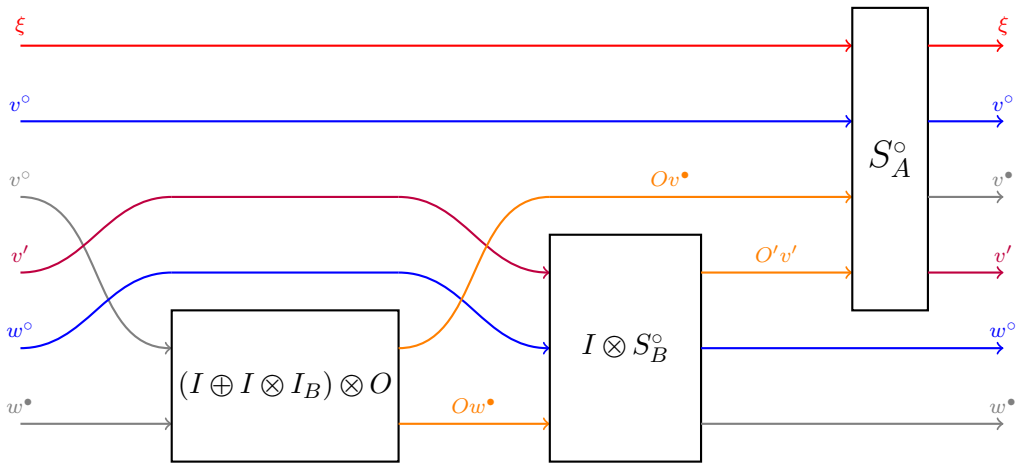
The space of S_B is $\mathcal{H}_B \oplus \mathcal{L}_B^\circ \oplus \mathcal{L}_B^\bullet$, where $\mathcal{L}_B^\bullet = \mathcal{L}_B^\uparrow \otimes \mathcal{M}$. We obtain the transducer $I \otimes S_B$ as in Definition 9.6, whose space is $\mathcal{L}_A^\uparrow \otimes (\mathcal{H}_B \oplus \mathcal{L}_B^\circ \oplus \mathcal{L}_B^\bullet)$.

Let $w = w^\circ \oplus w^\bullet$ be the catalyst for the transduction $v' \rightsquigarrow (I \otimes O')v'$ by $I \otimes S_B(O)$. In particular, $w^\bullet = q\big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\big)$. By (7.6) again, we have the chain of transformations in $\mathcal{L}_A^\uparrow \otimes (\mathcal{H}_B \oplus \mathcal{L}_B^\circ \oplus \mathcal{L}_B^\bullet)$:

$$v' \oplus w^\circ \oplus w^\bullet \overset{\tilde{O}}{\longmapsto} v' \oplus w^\circ \oplus (I \otimes I_B \otimes O)w^\bullet \overset{I \otimes S_B^\circ}{\longmapsto} (I \otimes O')v' \oplus w^\circ \oplus w^\bullet, \tag{9.16}$$

where I_B is the identity on $\mathcal{L}_B^\uparrow$.

Figure 9.3



A graphical representation of a function composition of transducers S_A and S_B. We omit the tensor multipliers of O and O' on the arrows to save space. The vector v' is simultaneously a non-queried catalyst for $S_A \circ S_B$ and the input to $I \otimes S_B$.

The transducer $S_A \circ S_B$ works as follows; see Figure 9.3. It starts in

$$\xi \oplus v^\circ \oplus v^\bullet \oplus v' \oplus w^\circ \oplus w^\bullet. \tag{9.17}$$

It applies the input oracle O to the third and the last terms, which corresponds to the first steps in both (9.15) and (9.16). This gives

$$\xi \oplus v^{\circ} \oplus (I \otimes O)v^{\bullet} \oplus v' \oplus w^{\circ} \oplus (I \otimes I_B \otimes O)w^{\bullet}. \tag{9.18}$$

Then it performs $I \otimes S_B^{\circ}$, which is the second operation in (9.16), and which gives

$$\xi \oplus v^{\circ} \oplus (I \otimes O)v^{\bullet} \oplus (I \otimes O')v' \oplus w^{\circ} \oplus w^{\bullet}. \tag{9.19}$$

After that, S_A° is performed, which is the last operation of (9.15), and we get the final state

$$\tau \oplus v^{\circ} \oplus v^{\bullet} \oplus v' \oplus w^{\circ} \oplus w^{\bullet}. \tag{9.20}$$

The transduction (9.13) and the query (9.14) complexities follow from (9.17). In order to get the time complexity, we have to elaborate on the placement of all these transducers in registers. Since S_A and S_B are aligned in O, we assume they use the same value $|1\rangle_{\mathcal{R}}$ of the same register to denote its execution. Concerning O', which is an oracle only for S_A, we assume it is indicated by another qubit $\mathcal{R}'$. Let $\mathcal{P}_A$ and $\mathcal{P}_B$ be the privacy qubits of S_A and S_B, respectively; we assume they are different.

Let us write down the values of these indicator qubits for the various subspaces of the composed transducer:

	$\mathcal{H}_A$	$\mathcal{L}_A^{\circ}$	$\mathcal{L}_A^{\bullet}$	$\mathcal{L}'_A$ and $\mathcal{H}_B$	$\mathcal{L}_B^{\circ}$	$\mathcal{L}_B^{\bullet}$
$\mathcal{P}_A$	0	1	1	1	1	1
$\mathcal{R}$	0	0	1	0	0	1
$\mathcal{R}'$	0	0	0	1	1	1
$\mathcal{P}_B$	0	0	0	0	1	1

Other than that, we assume we can embed all these subspaces into registers also preserving $\mathcal{L}'_A = \mathcal{L}^{\uparrow} \otimes \mathcal{H}_B$. The privacy qubit of $S_A \circ S_B$ is $\mathcal{P}_A$ and its query register is $\mathcal{R}$.

Let us go through the steps of $S_A \circ S_B$. The application of the query to O to get to (9.18) is conditioned on $|1\rangle_{\mathcal{R}}$ as required, and it is possible because we assume S_A and S_B are aligned in O. The application of $I \otimes S_B^{\circ}$ to get to (9.19) is conditioned on $|1\rangle_{\mathcal{R}'}$. Finally, the application of S_A° to get to the final state (9.20) is conditioned on $|0\rangle_{\mathcal{P}_B}$. This gives the required time complexity estimate. □

10 Transducers from Programs

In this section, we describe how to convert a quantum program A like in (4.11) into a canonical transducer S_A, and give the corresponding corollaries. We consider both the circuit and the QRAG models. The general idea is similar in both cases. First, we obtain transducers for individual gates, and then compose them using either Proposition 9.9 for the circuit model, or Proposition 9.10 for the QRAG model.

10.1 General Assumptions

We are given a program A, and our goal is to construct a canonical transducer S_A whose transduction action is identical to the action of A.

One thing to observe is that we have to modify our assumptions on the execution of input oracles by introducing an additional register related to $\mathcal{R}$. There are two main reasons for that. First, the initial state ξ of S_A can be any state in $\mathcal{H}$. In particular, it can use $\mathcal{H}^{\bullet}$ which is in

contradiction with the assumption of Section 7.1 that ξ must be contained in $\mathcal{H}^\circ$. Second, as described in Section 3.4, the catalyst in the QRAG case is the history state (3.11). Various ψ_t can use $\mathcal{H}^\bullet$, and we would like to protect them from the application of the input oracle in S_A.

Because of that, we introduce an additional qubit $\widetilde{\mathcal{R}}$. The input oracle in the canonical transducer S_A is applied conditioned by $|1\rangle_{\widetilde{\mathcal{R}}}$. More precisely, the i-th input oracle is controlled by $|1\rangle_{\widetilde{\mathcal{R}}}|i\rangle_{\mathcal{R}}$.

Additionally, we assume that all the queries made by the program A are aligned, see Definition 9.2, and, in case of several programs, they are aligned by their shared input oracles. This is necessary since the composition results of Section 9 require the transducers to be aligned.

10.2 Building Blocks

Here we describe transducers corresponding to elementary operations. The following proposition is trivial.

Proposition 10.1 (Trivial Transducer)**.** *Let A be a quantum algorithm (without oracle calls) that implements some unitary in $\mathcal{H}$. It can be considered as a transducer with $\mathcal{L}$ being empty, $T(A)$ being the time complexity of A, and both $W(A,\xi)$ and $q(A,\xi)$ equal to zero.*

Oracle execution is more tricky because of the canonicity assumption. Recall the query $\widetilde{O} = I^\circ \oplus I \otimes O$ from (4.1) taking place in $\mathcal{H} = \mathcal{H}^\circ \oplus \mathcal{H}^\bullet$ with $\mathcal{H}^\bullet = \mathcal{H}^\uparrow \otimes \mathcal{M}$. Let us divide $\xi = \xi^\circ \oplus \xi^\bullet$ accordingly.

Proposition 10.2 (Oracle)**.** *For a fixed embedding $\mathcal{H} = \mathcal{H}^\circ \oplus \mathcal{H}^\uparrow \otimes \mathcal{M}$ as above, there exists a canonical transducer S_{Q} such that $\xi \overset{S_{\mathrm{Q}}(O)}{\rightsquigarrow} \widetilde{O}\xi$ for all $\xi \in \mathcal{H}$ and unitaries $O\colon \mathcal{M} \to \mathcal{M}$. The transducer satisfies $W(S_{\mathrm{Q}},O,\xi) = \|\xi^\bullet\|^2$, $q(S_{\mathrm{Q}},O,\xi) = \xi^\bullet$, and $T(S_{\mathrm{Q}}) = O(1)$.*

Proof. The private space of S_{Q} will be $\mathcal{L} = \mathcal{L}^\bullet = \mathcal{L}^\uparrow \otimes \mathcal{M}$ with $\mathcal{L}^\uparrow$ equal to $\mathcal{H}^\uparrow$. The catalyst is $v = v^\bullet = \xi^\bullet$. The transducer S_{Q} first applies the input oracle to $\mathcal{L}^\bullet$, which is achieved by conditioning on $|1\rangle_{\widetilde{\mathcal{R}}}$:

$$\begin{aligned}|0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}|\xi\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}}|1\rangle_{\widetilde{\mathcal{R}}}|\xi^\bullet\rangle_{\mathcal{L}} &\overset{\widetilde{O}}{\longmapsto} |0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}|\xi\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}}|1\rangle_{\widetilde{\mathcal{R}}}\big|(I\otimes O)\xi^\bullet\big\rangle_{\mathcal{L}}\\ &= |0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}|0\rangle_{\mathcal{R}}|\xi^\circ\rangle + |0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}|1\rangle_{\mathcal{R}}|\xi^\bullet\rangle + |1\rangle_{\mathcal{P}}|1\rangle_{\widetilde{\mathcal{R}}}|1\rangle_{\mathcal{R}}|(I\otimes O)\xi^\bullet\rangle.\end{aligned}$$

Now apply C-NOT to both $\mathcal{P}$ and $\widetilde{\mathcal{R}}$ controlled by $|1\rangle_{\mathcal{R}}$. This gives

$$|0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}|0\rangle_{\mathcal{R}}|\xi^\circ\rangle + |0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}|1\rangle_{\mathcal{R}}|(I\otimes O)\xi^\bullet\rangle + |1\rangle_{\mathcal{P}}|1\rangle_{\widetilde{\mathcal{R}}}|1\rangle_{\mathcal{R}}|\xi^\bullet\rangle = |0\rangle_{\mathcal{P}}|0\rangle_{\widetilde{\mathcal{R}}}\big|\widetilde{O}\xi\big\rangle_{\mathcal{H}} + |1\rangle_{\mathcal{P}}|1\rangle_{\widetilde{\mathcal{R}}}\big|\xi^\bullet\big\rangle_{\mathcal{L}}.$$

□

10.3 Circuit Model

For the circuit model, we have the following result, which is a first half of Theorem 3.5.

Theorem 10.3 (Program to Transducer, Circuit Model)**.** *Let $A = A(O)$ be a quantum program in some space $\mathcal{H}$ assuming the circuit model. We assume all the queries in A are aligned, see Definition 9.2. Then, there exists a canonical transducer $S_A(O)$ with the following properties.*

- *$S_A(O)^{\xi}_{\circlearrowleft\mathcal{H}} = A(O)$ for all input oracles O.*
- *For any input oracle O and the initial state ξ, we have $q(S_A,O,\xi) = q(A,O,\xi)$. In particular, $L(S_A,O,\xi) = L(A,O,\xi)$.*

- *The catalyst $v(S_A, O, \xi)$ is equal to the query state $q(S_A, O, \xi)$. In particular, $W(S_A, O, \xi) = L(A, O, \xi)$.*
- *Finally, the transducer S_A can be implemented in the circuit model, and $T(S_A) = T_{\mathrm{C}}(A) + \mathcal{O}(Q(A)) = \mathcal{O}(T(A))$, where the complexity measures of A are as in Section 4.4.*

Proof. Take the representation of the query algorithm as in (4.2). We interpret each unitary U_t as a transducer using Proposition 10.1, and transform each query into an independent copy of the transducer S_{Q} from Proposition 10.2.

Now, we apply sequential composition of Proposition 9.9 with $m = 2Q(A) + 1$. The total query state $q(A, O, \xi)$ is defined as the direct sum of all the queries given to the input oracle, hence, the third and the second points follow from (9.6). The time complexity follows from (9.7) using that $T(A) = Q(A) + \sum_t T(U_t)$. □

While canonical transducers are nice from the theoretical point of view, designing one from scratch is usually inconvenient. The following result shows that we can convert any transducer into a canonical form with a slight increase in complexity.

Let $S = S(O)$ be a transducer in $\mathcal{H} \oplus \mathcal{L}$ not in the canonical form. Recall from Section 5.1 that we defined its total query state $q_{\mathcal{H}}(S, O, \xi)$, and Las Vegas query complexity $L_{\mathcal{H}}(S, O, \xi)$ on $\xi \in \mathcal{H}$ as that of S, considered as a usual quantum algorithm in $\mathcal{H} \oplus \mathcal{L}$, on the initial state $\xi \oplus v\big(S(O), \xi\big)$, where $v\big(S(O), \xi\big)$ is as in Theorem 5.1. We assume all the queries made in S are aligned.

Proposition 10.4 (Transforming Transducers into Canonical Form)**.** *For a non-canonical transducer $S = S(O)$ as above, there exists a canonical transducer $S' = S'(O)$ with the same transduction action and such that, for all O and ξ,*

$$q(S', O, \xi) = q_{\mathcal{H}}(S, O, \xi), \qquad W(S', O, \xi) = W(S(O), \xi) + L_{\mathcal{H}}(S, O, \xi), \tag{10.1}$$

and $T(S') = \mathcal{O}(T(S))$.

Proof. Let $S_S = S_S(O)$ be a canonical transducer as obtained in Theorem 10.3 from $S = S(O)$ considered as a quantum program. Its public space is $\mathcal{H} \oplus \mathcal{L}$, and $S_S(O)_{\{\mathcal{H} \oplus \mathcal{L}}} = S(O)$.

Let $\xi' = \xi \oplus v\big(S(O), \xi\big)$. By Theorem 10.3, we have $T(S_S) = \mathcal{O}\big(T(S)\big)$, and

$$v_{\mathcal{H} \oplus \mathcal{L}}(S_S, O, \xi') = q_{\mathcal{H} \oplus \mathcal{L}}(S_S, O, \xi') = q_{\mathcal{H} \oplus \mathcal{L}}(S, O, \xi') = q_{\mathcal{H}}(S, O, \xi), \tag{10.2}$$

where, in the third expression, we consider S as a usual quantum program in $\mathcal{H} \oplus \mathcal{L}$.

We obtain $S' = S'(O)$ as the transduction action of $S_S(O)$ on $\mathcal{H}$, using the transitivity of transduction, Proposition 5.4. First, $S_S(O)$ is in canonical form when considered as a transducer with the public space $\mathcal{H} \oplus \mathcal{L}$. A fortiori, it is canonical also on the public space $\mathcal{H}$. Next, its time complexity does not change, hence, $T(S') = T(S_S) = \mathcal{O}\big(T(S)\big)$. For the query state, from (5.3) and (10.2), we get

$$q_{\mathcal{H}}(S', O, \xi) = q_{\mathcal{H} \oplus \mathcal{L}}(S_S, O, \xi') = q_{\mathcal{H}}(S, O, \xi).$$

Finally, for the transduction complexity, we can utilise (5.3) in the following way:

$$\begin{aligned} W(S', O, \xi) = W_{\mathcal{H}}\big(S_S(O), \xi\big) &= W_{\mathcal{H}}\big(S_S(O)_{\{\mathcal{H} \oplus \mathcal{L}}, \xi\big) + W_{\mathcal{H} \oplus \mathcal{L}}\big(S_S(O), \xi'\big) \\ &= W_{\mathcal{H}}(S(O), \xi) + L_{\mathcal{H}}(S, O, \xi), \end{aligned}$$

where we used (10.2) in the last equality. □

The remaining corollaries of Theorem 10.3 were already proven in Section 3: Theorems 3.6 and 3.8.

Let us note that Theorem 10.3 might not always be the best way to get a transducer S_A from a quantum program A in the circuit model. One can use additional structure of A to get better transducers. For instance, if A repeatedly uses the same sequence of gates, one can define it as a new oracle, thus reducing the time complexity of the transducer, see, e.g., Proposition 11.3. If A contains a large loop, one can use Proposition 9.10. Obtaining efficient transducers in the circuit model for specific cases seems like an interesting research direction.

10.4 QRAG Model

Assuming the QRAG model, we get the remaining half of Theorem 3.5.

Theorem 10.5. *Let $A = A(O)$ be a quantum program in some space $\mathcal{H}$, and assume we have QRAM access to the description of A. Then, there exists a canonical transducer $S_A(O)$ with the following properties.*

- *$S_A(O)^{\xi}_{\mathcal{H}} = A(O)$ for all input oracles O.*
- *For any input oracle O and initial state ξ, we have $q(S_A, O, \xi) = q(A, O, \xi)$. In particular, $L(S_A, O, \xi) = L(A, O, \xi)$.*
- *Also, $W(S_A, O, \xi) \leqslant T(A)\|\xi\|^2$.*
- *Finally, in the QRAG model, we have $T(S_A) = \mathcal{O}(T_{\mathrm{R}})$ as defined in (3.13).*

Proof. The proof parallels that of Theorem 10.3 using Proposition 9.10 instead of Proposition 9.9, but since this special case might be of interest, we give an explicit construction here, slightly simplifying it along the way.

It is convenient to assume a different but essentially equivalent form of a quantum algorithm. Namely, we assume that the algorithm A is given as

$$A(O) = G_{m-1}\widetilde{O}^{b_{m-1}}G_{m-2}\cdots\widetilde{O}^{b_2}G_1\widetilde{O}^{b_1}G_0, \tag{10.3}$$

where each G_i is a gate (not a query), $\widetilde{O}$ is the query operator (4.1), and each b_t is a bit that indicates whether there is a query before the t-th gate (b_0 is always 0). A program like in (4.11) can be transformed into (10.3) with $m \leqslant T + 1$ adding identity G_t if necessary. We assume we have QRAM access to an array specifying the gates G_t as well as to the array of b_t.

Let the algorithm A go through the following sequence of states on the initial state ξ and the oracle O:

$$\xi = \psi_0 \overset{G_0}{\longmapsto} \psi_1 \overset{G_1\widetilde{O}^{b_1}}{\longmapsto} \psi_2 \overset{G_2\widetilde{O}^{b_2}}{\longmapsto} \cdots \overset{G_{m-1}\widetilde{O}^{b_{m-1}}}{\longmapsto} \psi_m = \tau. \tag{10.4}$$

At high level, the action of S_A is

$$\xi \oplus v = \sum_{t=0}^{m-1} |t\rangle|\psi_t\rangle \overset{\widetilde{O}^{b_t}}{\longmapsto} \sum_{t=0}^{m-1} |t\rangle|\widetilde{O}^{b_t}\psi_t\rangle \overset{G_t}{\longmapsto} \sum_{t=0}^{m-1} |t\rangle|\psi_{t+1}\rangle \longmapsto \sum_{t=1}^{m} |t\rangle|\psi_t\rangle = \tau \oplus v, \tag{10.5}$$

where, we first apply the input oracle conditioned on b_t, then G_t conditioned on t using Theorem 4.3, and then increment t by 1 modulo m.

Recall that we have a decomposition $\mathcal{H} = \mathcal{H}^{\circ} \oplus \mathcal{H}^{\bullet}$ of the space of A, which is indicated by the query register $\mathcal{R}$. As mentioned in Section 10.1, the transducer S_A uses a different query qubit $\widetilde{\mathcal{R}}$. We will use notation $\widetilde{\mathcal{H}} = \widetilde{\mathcal{R}} \otimes \mathcal{H}$. Let D denote the C-NOT on $\widetilde{\mathcal{R}}$ controlled by $\mathcal{R}$. Then, for $\psi = \psi^{\circ} \oplus \psi^{\bullet} \in \mathcal{H}$, we have

$$|\psi\rangle_{\widetilde{\mathcal{H}}} = |0\rangle_{\widetilde{\mathcal{R}}}|0\rangle_{\mathcal{R}}|\psi^{\circ}\rangle + |0\rangle_{\widetilde{\mathcal{R}}}|1\rangle_{\mathcal{R}}|\psi^{\bullet}\rangle \qquad \text{and} \qquad |D\psi\rangle_{\widetilde{\mathcal{H}}} = |0\rangle_{\widetilde{\mathcal{R}}}|0\rangle_{\mathcal{R}}|\psi^{\circ}\rangle + |1\rangle_{\widetilde{\mathcal{R}}}|1\rangle_{\mathcal{R}}|\psi^{\bullet}\rangle.$$

Let $\mathcal{T}$ be an m-qudit with operations modulo m, and $\mathcal{P}$ be the privacy qubit of S_A.

Let us go through the steps of (10.5). The initial coupling is given by

$$\xi \oplus v = |0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{T}}|\psi_0\rangle_{\widetilde{\mathcal{H}}} + \sum_{t=1}^{m-1} |1\rangle_{\mathcal{P}}|t\rangle_{\mathcal{T}}|D^{b_t}\psi_t\rangle_{\widetilde{\mathcal{H}}}.$$

First, as required by the canonical form and our assumptions in Section 10.1, we apply the input oracle O conditioned on $|1\rangle_{\widetilde{\mathcal{R}}}$. This acts as $\widetilde{O}$ on $D\psi_t$, but does not change ψ_t. Then, we apply the operation D controlled on b_t (which can be accessed using the QRAM). After that, we apply C-NOT to $\mathcal{P}$ controlled by $|0\rangle_{\mathcal{T}}$. This gives the second state in the sequence (10.5):

$$|1\rangle_{\mathcal{P}} \sum_{t=0}^{m-1} |t\rangle_{\mathcal{T}}|\widetilde{O}^{b_t}\psi_t\rangle_{\widetilde{\mathcal{H}}}.$$

Now, we apply the last two operations from (10.5): G_t conditioned on $|t\rangle_{\mathcal{T}}$, and increment of $\mathcal{T}$ by 1 modulo m. This gives

$$|1\rangle_{\mathcal{P}} \sum_{t=1}^{m} |t\rangle_{\mathcal{T}}|\psi_t\rangle_{\widetilde{\mathcal{H}}}.$$

Now we apply the operation D controlled on b_t and apply C-NOT to the register $\mathcal{P}$ controlled by $|0\rangle_{\mathcal{T}} = |m\rangle_{\mathcal{T}}$. This gives the final state

$$|0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{T}}|\psi_m\rangle_{\widetilde{\mathcal{H}}} + \sum_{t=1}^{m-1} |1\rangle_{\mathcal{P}}|t\rangle_{\mathcal{T}}|D^{b_t}\psi_t\rangle_{\widetilde{\mathcal{H}}} = \tau \oplus v.$$

Since $\|\psi_t\| = \|\xi\|$, we have that $W(S_A, O, \xi) = (m-1)\|\xi\|^2 \leqslant T(A)\|\xi\|^2$. It is clear that $q(S_A, O, \xi) = q(A, O, \xi)$, and the time complexity of S_A is $\mathcal{O}(T_{\mathrm{R}})$ thanks in particular to Theorem 4.3. □

In applications like in Section 3.5, we often have to apply this construction in parallel. The following easy modification of the proof takes care of that.

Proposition 10.6. *Let $A_1, \dots, A_n$ be quantum programs in some space $\mathcal{H}$, all using the same oracle O. Assume we have QRAM access to their joint description. Then, the direct sum $\bigoplus_{i=1}^n S_{A_i}$ of transducers defined in Theorem 10.5 can be implemented in time $\mathcal{O}(T_{\mathrm{R}})$.*

By the joint access, we mean that we have access to the list $m_1, \dots, m_n$ of the parameters m in (10.3), as well as access to b_t and G_t of A_i as a double array with indices i and t.

Proof of Proposition 10.6. We execute the transducers S_{A_i} in parallel using a register $\mathcal{J}$ to store the value of i. Accessing b_t and G_t now is double-indexed by i and t, and by our general assumption of (3.13) it takes time $\mathcal{O}(T_{\mathrm{R}})$ to access them. For incrementation of $\mathcal{T}$ modulo m, we use the array containing m_i. Other than that, it is a standard word-sided operation and takes time $\mathcal{O}(T_{\mathrm{R}})$. □

The consequences of this result were already considered in Section 3: Theorems 3.6, 3.10, and 3.11.

11 Example III: Iterated Functions

In this section, we give a more detailed proof of Theorem 3.12. We will use notation of Section 8.2 for the transducer S_f built from the adversary bound. However, we assume a more general variant of a canonical transducer S_f for evaluation of $f\colon [q]^n \to [q]$. Its public space is $\mathbb{C}^q$, its admissible space is spanned by $|0\rangle$, and $|0\rangle \overset{S_f(\overleftrightarrow{O_x})}{\rightsquigarrow} |f(x)\rangle$ for all $x \in [q]^n$. We assume the initial coupling of S_f on $\overleftrightarrow{O_x}$ is given by

$$|0\rangle \oplus v_x = |0\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|0\rangle + |1\rangle_{\mathcal{P}}|0\rangle_{\mathcal{R}}|v_x^{\circ}\rangle + |1\rangle_{\mathcal{P}} \sum_{i\in[n]} |i\rangle_{\mathcal{R}}\Big[|{\uparrow}\rangle_{\mathcal{B}}|0\rangle_{\mathcal{Q}}|v_{x,i}^{\uparrow}\rangle_{\mathcal{W}} + |{\downarrow}\rangle_{\mathcal{B}}|x_i\rangle_{\mathcal{Q}}|v_{x,i}^{\downarrow}\rangle_{\mathcal{W}}\Big] \tag{11.1}$$

for some vectors v_x°, $v_{x,i}^{\uparrow}$, and $v_{x,i}^{\downarrow}$, where the registers are as in (8.7). The difference between (11.1) and (8.7), however, is addition of the term v_x° that is not processed by the input oracle. Note that (11.1) gives the general form of a canonical transducer that executes the input oracle on the admissible subspace: the constituent $O_{x,i}$ from (8.5) on $|0\rangle$ and $O_{x,i}^*$ on $|x_i\rangle$.

Recall the definition of the composed function $f \circ g$ from (3.29):

$$\begin{aligned}(f\circ g)(z_{1,1},\dots,z_{1,m},\ \ z_{2,1},\dots,z_{2,m},\ \ \dots\dots,\ \ z_{n,1},\dots,z_{n,m})\\ = f\big(g(z_{1,1},\dots,z_{1,m}), g(z_{2,1},\dots,z_{2,m}),\dots,g(z_{n,1},\dots,z_{n,m})\big).\end{aligned} \tag{11.2}$$

We define

$$\vec{y}_i = (z_{i,1},\dots,z_{i,m}) \qquad \text{and} \qquad x = \big(g(\vec{y}_1), g(\vec{y}_2),\dots g(\vec{y}_n)\big) \tag{11.3}$$

so that $f(x) = (f\circ g)(z)$. Recall also notation $W_x(S) = W\big(S, \overleftrightarrow{O_x}, |0\rangle\big)$ and $W(S) = \max_x W_x(S)$ from (8.2).

Proposition 11.1. *Let S_f and S_g be canonical transducers for the functions f and g of the form described above. Then, there exists a canonical transducer $S_{f\circ g}$ for the composed function $f \circ g$ from (11.2) that has the same form, and such that*

$$W_z(S_{f\circ g}) = W_x(S_f) + \sum_{i=1}^{n} L_x^{(i)}(S_f)\cdot W_{\vec{y}_i}(S_g) \leqslant W(S_f) + L(S_f)\cdot W(S_g) \tag{11.4}$$

and

$$L_z(S_{f\circ g}) = \sum_{i=1}^{n} L_x^{(i)}(S_f)\cdot L_{\vec{y}_i}(S_g) \leqslant L(S_f)\cdot L(S_g), \tag{11.5}$$

for every $z \in [q]^{nm}$. Time complexity satisfies $T(S_{f\circ g}) = T_{\mathrm{C}}(S_f) + 2T_{\mathrm{C}}(S_g)$.

Proof. As the first step, we create a bidirectional version $\overleftrightarrow{S_g}$ of S_g. Its public space is $\mathcal{B}\otimes\mathcal{Q}$, and its admissible space on the input oracle $\overleftrightarrow{O_y}$ is spanned by $|{\uparrow}\rangle_{\mathcal{B}}|0\rangle_{\mathcal{Q}}$ and $|{\downarrow}\rangle_{\mathcal{B}}|g(y)\rangle_{\mathcal{Q}}$. Its corresponding transduction action is

$$|{\uparrow}\rangle_{\mathcal{B}}|0\rangle_{\mathcal{Q}} \rightsquigarrow |{\uparrow}\rangle_{\mathcal{B}}|g(y)\rangle_{\mathcal{Q}} \qquad \text{and} \qquad |{\downarrow}\rangle_{\mathcal{B}}|g(y)\rangle_{\mathcal{Q}} \rightsquigarrow |{\downarrow}\rangle_{\mathcal{B}}|0\rangle_{\mathcal{Q}}.$$

Its time complexity is $2T_{\mathrm{C}}(S_g)$ and, in notation at the end of Section 3.3:

$$W_{\max}\big(\overleftrightarrow{S_g}, \overleftrightarrow{O_y}\big) = W_y(S_g) \qquad \text{and} \qquad L_{\max}\big(\overleftrightarrow{S_g}, \overleftrightarrow{O_y}\big) = L_y(S_g).$$

We obtain $\overleftrightarrow{S_g}$ as follows. By swapping $\uparrow$ and $\downarrow$ if necessary, we may identify $\overleftrightarrow{O_y}^*$ and $\overleftrightarrow{O_y}$. Then from Proposition 9.1, we obtain a transducer S_g^{-1} with transduction action $|g(y)\rangle \rightsquigarrow |0\rangle$ on $\overleftrightarrow{O_y}$, whose complexity is identical to S_g. We may assume S_g and S_g^{-1} are aligned. We get

$\overleftrightarrow{S_g}$ as $S_g \oplus S_g^{-1}$, where the direct sum is done via the register $\mathcal{B}$. By the definition of $W_{\max}$, we have that for some unit vector $(\alpha, \beta) \in \mathbb{C}^2$:

$$\begin{aligned} W_{\max}(\overleftrightarrow{S_g}, \overrightarrow{O_y}) &= W\Big(\overleftrightarrow{S_g}, \overrightarrow{O_y}, \alpha|\uparrow\rangle_{\mathcal{B}}|0\rangle_{\mathcal{Q}} + \beta|\downarrow\rangle_{\mathcal{B}}|g(y)\rangle_{\mathcal{Q}}\Big) \\ &= |\alpha|^2 W\big(S_g, \overrightarrow{O_y}, |0\rangle\big) + |\beta|^2 W\big(S_g^{-1}, \overrightarrow{O_y}, |g(y)\rangle\big) = W_y(S_g), \end{aligned}$$

where we used Propositions 9.4 and 9.1, as well as (9.1). Query complexity derivation is similar.

As the next step, we construct the transducer $I_n \otimes \overleftrightarrow{S_g}$, where I_n acts on the span of $|i\rangle_{\mathcal{R}}$ with $i > 0$. Moreover, as in Remark 9.5, we assume the input oracle uses the register $\mathcal{R}$. Therefore, the transduction action of $I_n \otimes \overleftrightarrow{S_g}$ on the input oracle

$$\overleftrightarrow{O_z} = \bigoplus_{i=1}^{n} \overleftrightarrow{O_{\vec{y}_i}}$$

is identical to the action of $\overleftrightarrow{O_x}$ on its admissible subspace.

Finally, we get $S_{f\circ g}$ as $S_f \circ \big(I_n \otimes \overleftrightarrow{S_g}\big)$. The time complexity estimate follows from Proposition 9.11. For the transduction complexity, we obtain from (3.21), using that S_f makes only admissible queries to $\overleftrightarrow{O_x}$ and does not have direct access to $\overleftrightarrow{O_z}$:

$$\begin{aligned} W_z(S_{f\circ g}) &= W\big(S_{f\circ g}, \overrightarrow{O_z}, |0\rangle\big) \\ &\leqslant W\big(S_f, \overrightarrow{O_x}, |0\rangle\big) + \sum_{i=1}^{n} W_{\max}\big(\overleftrightarrow{S_g}, \overrightarrow{O_{\vec{y}_i}}\big) L^{(i)}(S_f, \overrightarrow{O_x}, |0\rangle) \\ &= W_x(S_f) + \sum_{i=1}^{n} W_{\vec{y}_i}(S_g) \cdot L_x^{(i)}(S_f). \end{aligned}$$

The last inequality in (11.4) follows from $L_x(S_f) = \sum_i L_x^{(i)}(S_f)$. The estimate (11.5) is similar, where we again use that S_f does not make direct queries to $\overleftrightarrow{O_z}$. Finally, since $\overleftrightarrow{S_g}$ makes only admissible queries to O_y, we get that $S_{f\circ g}$ makes only admissible queries to O_z. □

Now we can prove a more detailed version of Theorem 3.12.

Theorem 11.2. *Let S_f be a canonical transducer for a function f of the form as in (11.1), and such that $L = L(S_f) = 1 + \Omega(1)$. Let $W = W(S_f)$ and $T = T(S_f)$, assuming the circuit model. Then, for each d, there exists a bounded-error quantum algorithm evaluating the iterated function $f^{(d)}$ in Monte Carlo query complexity $\mathcal{O}(L^d)$ and time complexity $\mathcal{O}\big(d \cdot TWL^{d-1}\big) = \mathcal{O}_f(d \cdot L^d)$ in the circuit model.*

Proof. Define $S_{f^{(d)}}$ as S_f composed with itself d times using Proposition 11.1. By induction, we have that $L\big(S_{f^{(d)}}\big) \leqslant L^d$, and

$$W\big(S_{f^{(d)}}\big) \leqslant \big(1 + L + \cdots + L^{d-1}\big)W = \mathcal{O}\big(L^{d-1}W\big).$$

Similarly, its time complexity is at most $2d \cdot T_{\mathrm{C}}(S_f) = \mathcal{O}(d \cdot T)$. The theorem follows from Theorem 7.1. □

The time complexity of the previous theorem can be slightly improved. Let s denote the initial space complexity of the transducer S_f, i.e, the number of qubits used to encode the initial coupling $|0\rangle \oplus v_x$. It can be much smaller than the time complexity of S_f, as well as its space complexity, which is the total number of qubits used by S_f.

Proposition 11.3. *The time complexity of Theorem 11.2 can be improved to* $\mathcal{O}\big((sd+T)WL^{d-1}\big)$, *where* s *is the initial space complexity of the transducer* S_f.

We only sketch the proof, as it does not improve the overall asymptotic $\mathcal{O}_f(d \cdot L^d)$.

Proof sketch of Proposition 11.3. By studying the proof of Theorem 11.2, we can observe that the work unitary of $S_{f^{(d)}}$ consists of repeated applications of $\overleftrightarrow{S_f}$ to different registers. We can treat $\overleftrightarrow{S_f}$ as an oracle. Then, the work unitary of $S_{f^{(d)}}$ becomes a non-canonical transducer with the oracle $\overleftrightarrow{S_f}$. It is non-aligned, but we can make it aligned in additional time $s \cdot d$ by switching the registers before each execution of $\overleftrightarrow{S_f}$. We can convert it into the canonical form using Proposition 10.4, which increases the transduction complexity by at most a constant factor. However, now it takes only one execution of $\overleftrightarrow{S_f}$ and $\mathcal{O}(sd)$ other operations to implement the transducer. The statement again follows from Theorem 7.1. □

12 Perturbed Transducers

It is quite common in quantum algorithm to use subroutines that impose some error. The total correctness of the algorithm then follows from a variant of Lemma 4.5, assuming that the error of each constituent is small enough.

In most cases in this paper, like in Sections 6 and 8.2, the transduction action is the exact implementation of the required transformation. However, it is not always feasible, and it makes sense to study what happens if a transducer satisfies the condition (5.1) only approximately.

It is worth noting that small errors in a transducer can result in large errors in the corresponding transduction action. In other words, it is possible that $\xi \oplus v \overset{S}{\longmapsto} \tau' \oplus v'$ with v close to v', but $\xi \overset{S}{\rightsquigarrow} \tau$ with τ being far from τ'. For example, let both $\mathcal{H}$ and $\mathcal{L}$ be 1-dimensional, and assume S acts as

$$S\colon \begin{pmatrix} 0 \\ v \end{pmatrix} \mapsto \begin{pmatrix} a \\ \sqrt{v^2 - a^2} \end{pmatrix}$$

for some positive real v and a. Observe that

$$v - \sqrt{v^2 - a^2} = \frac{a^2}{v + \sqrt{v^2 - a^2}}$$

can be very small for large v, even if a is substantial. Thus, on the basis of $v \approx v - \sqrt{v^2 - a^2}$ we may be tempted to assume that S approximately transduces 0 into a, but this is very far from the true transduction action $0 \overset{S}{\rightsquigarrow} 0$.

12.1 Definition

In order to solve this issue, we incorporate a perturbation, in the sense of Lemma 4.5, into the transducer S. Let S be a unitary in $\mathcal{H} \oplus \mathcal{L}$ that maps

$$\widetilde{S}\colon \xi \oplus v \mapsto \widetilde{\tau} \oplus \widetilde{v}.$$

We assume its idealised version maps

$$S\colon \xi \oplus v \mapsto \tau \oplus v$$

with the perturbation

$$\delta(S, \xi) = \big\|(\tau \oplus v) - (\widetilde{\tau} \oplus \widetilde{v})\big\|.$$

We call S the idealised or perturbed transducer, and $\widetilde{S}$ the approximate transducer. We say that S transduces ξ into τ, and we keep notation $\xi \overset{S}{\rightsquigarrow} \tau$, $v(S,\xi) = v$, $W(S,\xi) = \|v\|^2$, and $\tau(S,\xi) = \tau$.

This also extends to the canonical form of transducer in Section 7.1, where we decompose $v = v^\circ \oplus v^\bullet$, and let $q(S,O,\xi) = v^\bullet$ and $L(S,O,\xi) = \|v^\bullet\|^2$. As before, we write $\delta(S,O,\xi)$ instead of $\delta(S(O),\xi)$, and similarly for other pieces of notation.

As for usual quantum algorithms, we can use approximate transducers instead of the idealised ones, as long as we carefully keep track of the perturbations. In the remaining part of this section, we will briefly study the main results of this paper under the perturbation lenses.

12.2 Implementation

We have the following variant of Theorem 5.5.

Theorem 12.1. *Under the assumptions of Section 12.1, for every positive integer K, there exists a quantum algorithm that transforms ξ into τ' such that*

$$\|\tau' - \tau(S,\xi)\| \leqslant 2\sqrt{\frac{W(S,\xi)}{K}} + \sqrt{K}\delta(S,\xi)$$

for every $\widetilde{S}\colon \mathcal{H}\oplus\mathcal{L} \to \mathcal{H}\oplus\mathcal{L}$, perturbed version S, and initial state $\xi \in \mathcal{H}$. The algorithm conditionally executes $\widetilde{S}$ as a black box K times, and uses $\mathcal{O}(K)$ other elementary operations.

Proof. The algorithm is identical to that of Theorem 5.5. The analysis is similar with the only difference that, on Step 2(a), in order to obtain the mapping

$$\frac{1}{\sqrt{K}}|t\rangle_{\mathcal{T}}|0\rangle_{\mathcal{P}}|\xi\rangle_{\mathcal{H}} + \frac{1}{\sqrt{K}}|t\rangle_{\mathcal{T}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}} \longmapsto \frac{1}{\sqrt{K}}|t\rangle_{\mathcal{T}}|0\rangle_{\mathcal{P}}|\tau\rangle_{\mathcal{H}} + \frac{1}{\sqrt{K}}|t\rangle_{\mathcal{T}}|1\rangle_{\mathcal{P}}|v\rangle_{\mathcal{L}}$$

as in (5.7), we introduce a perturbation of size $\delta(S,\xi)/\sqrt{K}$. By Lemma 4.5, the total perturbation is then

$$2\frac{\|v\|}{\sqrt{K}} + K\cdot\frac{\delta(S,\xi)}{\sqrt{K}},$$

which gives the required estimate. □

Again, this theorem can be reformulated as follows.

Corollary 12.2. *For all $W,\varepsilon > 0$, there exists a quantum algorithm that executes $\widetilde{S}$ as a black box $K = \mathcal{O}(1 + W/\varepsilon^2)$ times, uses $\mathcal{O}(K)$ other elementary operations, and ε-approximately transforms ξ into $\tau(S,\xi)$ for all S, $\widetilde{S}$, and the initial state ξ that satisfy $W(S,\xi) \leqslant W$ and $\delta(S,\xi) \leqslant \frac{\varepsilon}{2\sqrt{K}}$.*

We get a version of Theorem 7.1 in a similar fashion.

Proposition 12.3. *Theorem 7.1 works assuming a perturbed transducer S. The estimate is*

$$\|\tau' - \tau(S,O,\xi)\| \leqslant \frac{2}{\sqrt{K}}\sqrt{W(S,O,\xi) + \sum_{i=1}^{r}\left(\frac{K}{K^{(i)}} - 1\right)L^{(i)}(S,O,\xi)} + \sqrt{K}\delta(S,O,\xi).$$

Corollary 7.2 also works assuming a perturbed transducer S under the additional assumption of $\delta(S,O,\xi) \leqslant \frac{\varepsilon}{2\sqrt{K}}$.

Proof. The proof is analogous to Theorem 12.1: we use perturbation of size $\delta(S,O,\xi)/\sqrt{K}$ to get from (7.14) to (7.15). □

12.3 Composition

The composition of perturbed transducers exactly follows the corresponding constructions of Section 9, where we use Lemma 4.5 to evaluate the total perturbation. This gives us the following proposition.

Proposition 12.4. *Propositions 9.4, 9.9, 9.10 and 9.11 work assuming perturbed transducers. We get the following estimates:*

$$\delta(S, O, c\xi) = |c|\delta(S, O, \xi), \tag{12.1}$$

$$\delta(S, O, \xi) \leqslant \sqrt{\sum_{i=1}^{m} \delta(S_i, O, \xi_i)^2} \tag{12.2}$$

for Proposition 9.4, and

$$\delta(S, O, \xi) \leqslant \sum_{t=1}^{m} \delta(S_t, O, \psi_t) \tag{12.3}$$

for Propositions 9.9 and 9.10. For Proposition 9.11, we have

$$\delta(S_A \circ S_B, O, \xi) \leqslant \delta\big(S_A, O \oplus O', \xi\big) + \delta\big(S_B, O, q^{(1)}(S_A, O \oplus O', \xi)\big). \tag{12.4}$$

Proof. Eq. (12.1) follows by linearity. All the remaining estimates follow from the corresponding proofs in Section 9 replacing each transducer with its approximate version and using Lemma 4.5. The estimate (12.2) follows from the observation that perturbations in terms of the direct sum act on orthogonal subspaces. □

13 Purifiers

In this section, we formulate and prove the formal version of Theorem 3.14, as well as draw some consequences of it for composition of bounded-error algorithms.

13.1 Boolean Case

Recall our settings from Section 3.6. The input oracle performs the transformation

$$O_\psi \colon |0\rangle_{\mathcal{M}} \mapsto |\psi\rangle_{\mathcal{M}} = |0\rangle_{\mathcal{B}}|\psi_0\rangle_{\mathcal{N}} + |1\rangle_{\mathcal{B}}|\psi_1\rangle_{\mathcal{N}} \tag{13.1}$$

for some unit vector ψ in some space $\mathcal{M} = \mathcal{B} \otimes \mathcal{N}$ with $\mathcal{B} = \mathbb{C}^2$, and it is promised that there exist constants $0 \leqslant c - d < c + d \leqslant 1$ such that

$$\text{either} \qquad \|\psi_1\|^2 \leqslant c - d \qquad \text{or} \qquad \|\psi_1\|^2 \geqslant c + d, \tag{13.2}$$

which corresponds to $f(\psi) = 0$ and $f(\psi) = 1$, respectively. Let

$$\mu = \sqrt{(1-c)^2 - d^2} + \sqrt{c^2 - d^2} < 1. \tag{13.3}$$

We say the input oracle O_ψ is *admissible* if it satisfies (13.1) and (13.2).

Theorem 13.1. *Let D be a positive integer. In the above assumptions, there exists a perturbed transducer S_{pur} on the 1-dimensional public space and with bidirectional access to O_ψ which satisfies the following conditions:*

- *It transduces* $|0\rangle$ *into* $(-1)^{f(\psi)}|0\rangle$ *for all admissible input oracles* O_ψ*. In particular, every vector in its public space is admissible.*
- *On every admissible input oracle and normalised input state, its perturbation is at most* $\delta_{\mathrm{pur}} = 2\mu^{D-1}$*.*
- *Its transduction and query complexities,* $W_{\max}(S_{\mathrm{pur}})$ *and* $L_{\max}(S_{\mathrm{pur}})$*, are* $\mathcal{O}\big(1/(1-\mu)\big) = \mathcal{O}(1)$*.*
- *It executes the input oracle on the admissible subspace only:* O_ψ *on* $|0\rangle_{\mathcal{M}}$ *and* O_ψ^* *on* $|\psi\rangle_{\mathcal{M}}$*.*

In the circuit model, the purifier can be implemented in time $\mathcal{O}(D \cdot s)$*, where* s *is the number of qubits used in* $\mathcal{M}$*. In the QRAG model and assuming the RA input oracle, the purifier can be implemented in time* $\mathcal{O}(T_{\mathrm{R}})$*.*

Here we used $W_{\max}(S_{\mathrm{pur}})$ to denote maximal $W(S_{\mathrm{pur}}, \overleftrightarrow{O_\psi}, |0\rangle)$ over all admissible input oracles O_ψ. $L_{\max}(S_{\mathrm{pur}})$ is defined similarly.

Proof. The outline of the proof was already given in Section 3.6. We will assume that D is even for concreteness, the case of odd D being analogous. Recall the parameters

$$a = \sqrt[4]{\frac{1-c+d}{1-c-d}} \qquad \text{and} \qquad b = \sqrt[4]{\frac{c+d}{c-d}}.$$

and the following vector in $\mathcal{M}$:

$$\widetilde{\psi} = \begin{cases} \frac{1}{a}|0\rangle|\psi_0\rangle + b|1\rangle|\psi_1\rangle, & \text{if } f(\psi) = 0; \\ a|0\rangle|\psi_0\rangle + \frac{1}{b}|1\rangle|\psi_1\rangle, & \text{if } f(\psi) = 1. \end{cases}$$

We have the following important estimate. If $f(\psi) = 0$:

$$\|\widetilde{\psi}\|^2 = \frac{1}{a^2}\|\psi_0\|^2 + b^2\|\psi_1\|^2 \leqslant \frac{1}{a^2}(1-c+d) + b^2(c-d) = \mu, \tag{13.4}$$

where we used that $1/a^2 < 1 < b^2$. Similarly, for $f(\psi) = 1$:

$$\|\widetilde{\psi}\|^2 = a^2\|\psi_0\|^2 + \frac{1}{b^2}\|\psi_1\|^2 \leqslant a^2(1-c-d) + \frac{1}{b^2}(c+d) = \mu. \tag{13.5}$$

We describe the transducer as non-canonical, and we will turn it into the canonical form later. The space of the transducer is $\mathcal{D} \otimes \mathcal{M}^{\otimes D-1}$, where $\mathcal{D}$ is a D-qudit. The one-dimensional public space is spanned by $\xi = |0\rangle_{\mathcal{D}}|0\rangle^{\otimes D-1}$. The initial coupling is given by

$$\xi \oplus v = \sum_{i=0}^{D-1} (-1)^{i \cdot f(\psi)} |i\rangle_{\mathcal{D}} |\widetilde{\psi}\rangle^{\otimes i} |0\rangle^{\otimes D-i-1}. \tag{13.6}$$

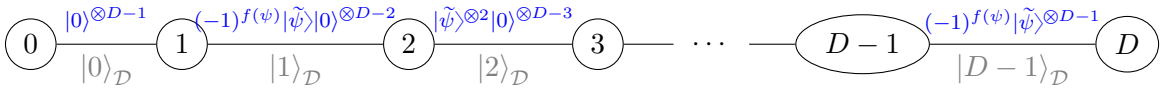
The transduction complexity is

$$\|v\|^2 \leqslant \|\xi \oplus v\|^2 \leqslant \sum_{i=0}^{\infty} \|\widetilde{\psi}\|^{2i} = \frac{1}{1 - \|\widetilde{\psi}\|^2} \leqslant \frac{1}{1-\mu}. \tag{13.7}$$

Let us now describe the action of the transducer. The transducer is a multidimensional quantum walk on the line graph, see Figure 13.1. It is a product of two reflections R_1 and R_2. The reflection R_1 is the product of the local reflections on the odd vertices, $i = 1, 3, 5, \ldots, D-1$. The reflection R_2 is the product of the local reflections on the even vertices, $i = 2, 4, \ldots, D-2$.

The local reflection for the vertex $i = 1, \ldots, D-1$ is as follows. It acts in $\mathrm{span}\{|i-1\rangle_{\mathcal{D}}, |i\rangle_{\mathcal{D}}\} \otimes \mathcal{M}^{\otimes D-1}$. Define a qubit $\mathcal{A}$ whose value 0 corresponds to $|i-1\rangle_{\mathcal{D}}$ and 1 to $|i\rangle_{\mathcal{D}}$. If i is odd, this could be the least significant qubit of $\mathcal{D}$. Let $\mathcal{M}^{(i)} = \mathcal{B}^{(i)} \otimes \mathcal{N}^{(i)}$ be the i-th multiplier in the tensor product $\mathcal{M}^{\otimes D-1}$. The reflection is as follows:

Figure 13.1



A purifier is a multidimensional quantum walk on the line graph seen above. The edge between the vertices i and $i+1$ corresponds to the subspace $|i\rangle_{\mathcal{D}} \otimes \mathcal{M}^{\otimes D-1}$ as indicated by the state below the edge. The local reflections on the vertices 0 and D are identities. The local reflection at the vertex $i = 1, \dots, D-1$ acts on the subspace $\operatorname{span}\{|i-1\rangle_{\mathcal{D}}, |i\rangle_{\mathcal{D}}\} \otimes \mathcal{M}^{\otimes D-1}$. The expressions above the edges give the initial coupling from (13.6).

1. Execute the input oracle O_ψ on $\mathcal{M}^{(i)}$ conditioned on $|0\rangle_{\mathcal{A}} = |i-1\rangle_{\mathcal{D}}$.
2. Execute a two-qubit unitary on $\mathcal{A} \otimes \mathcal{B}^{(i)}$, which is the reflection about the span of the states
$$\big(a|0\rangle_{\mathcal{A}} + |1\rangle_{\mathcal{A}}\big)|0\rangle_{\mathcal{B}} \qquad \text{and} \qquad \big(|0\rangle_{\mathcal{A}} + b|1\rangle_{\mathcal{A}}\big)|1\rangle_{\mathcal{B}}.$$
3. Execute the inverse oracle O_ψ^* on $\mathcal{M}^{(i)}$ conditioned on $|0\rangle_{\mathcal{A}} = |i-1\rangle_{\mathcal{D}}$.

Claim 13.2. *The local reflection for the vertex i multiplies the corresponding part of the state in (13.6)*
$$|i-1\rangle_{\mathcal{D}}|\widetilde{\psi}\rangle^{\otimes i-1}|0\rangle^{\otimes D-i} + (-1)^{f(\psi)}|i\rangle_{\mathcal{D}}|\widetilde{\psi}\rangle^{\otimes i}|0\rangle^{\otimes D-i-1}$$
by the phase $(-1)^{f(\psi)}$.

Proof. After application of the oracle in Step 1, we get the state
$$|\widetilde{\psi}\rangle^{\otimes i-1} \otimes \Big[|i-1\rangle_{\mathcal{D}}|\psi\rangle_{\mathcal{M}} + (-1)^{f(\psi)}|i\rangle_{\mathcal{D}}|\widetilde{\psi}\rangle_{\mathcal{M}}\Big] \otimes |0\rangle^{\otimes D-i-1}.$$
The local reflection acts on the state in the square brackets, which can be rewritten as
$$\Big(|0\rangle_{\mathcal{A}} + \frac{1}{a}|1\rangle_{\mathcal{A}}\Big)|0\rangle_{\mathcal{B}}|\psi_0\rangle_{\mathcal{N}} + \Big(|0\rangle_{\mathcal{A}} + b|1\rangle_{\mathcal{A}}\Big)|1\rangle_{\mathcal{B}}|\psi_1\rangle_{\mathcal{N}} \tag{13.8}$$
if $f(\psi) = 0$, and
$$\Big(|0\rangle_{\mathcal{A}} - a|1\rangle_{\mathcal{A}}\Big)|0\rangle_{\mathcal{B}}|\psi_0\rangle_{\mathcal{N}} + \Big(|0\rangle_{\mathcal{A}} - \frac{1}{b}|1\rangle_{\mathcal{A}}\Big)|1\rangle_{\mathcal{B}}|\psi_1\rangle_{\mathcal{N}} \tag{13.9}$$
if $f(\psi) = 1$. It is easy to see that the operation on Step 2 does not change the state in (13.8) and negates the one in (13.9), from which the claim follows. □

From the claim, it immediately follows that, if $f(\psi) = 0$, the transducer does not change the state (13.6). Therefore, in this case $|0\rangle \overset{S_{\text{pur}}}{\rightsquigarrow} |0\rangle$.

On the other hand, if $f(\psi) = 1$, then R_1 reflects the whole state $\xi \oplus v$, and R_2 reflects all the terms in the sum except for $i = 0$ and $i = D-1$. Thus, the final state is
$$-\xi \oplus v - 2(-1)^{(D-1)f(\psi)}|D-1\rangle_{\mathcal{D}}|\widetilde{\psi}\rangle^{\otimes D-1}.$$
This can be interpreted as transducing $|0\rangle$ into $-|0\rangle$ with a perturbation of size at most $2\mu^{D-1}$ by (13.4) and (13.5).

One can see that the local reflection for the vertex i applies the input oracle and its inverse on the part of the state v in the subspace $|i-1\rangle_{\mathcal{D}} \otimes \mathcal{M}^{\otimes D-1}$. Hence, the query complexity is at most $2\|\xi \oplus v\|^2 \leqslant 2/(1-\mu)$ by (13.7). The transformation into canonical form, Proposition 10.4, adds the query complexity to the transduction complexity, hence, the latter stays $\mathcal{O}\big(1/(1-\mu)\big)$.

Let us estimate time complexity. We start with the circuit model. The queries in transducer of Theorem 13.1 are not aligned, as they are applied to different copies of $\mathcal{M}$. In order to make them aligned, as required by Proposition 10.4, the register $\mathcal{M}^{(i)}$ should be moved to some specific array of qubits shared by all the local reflections. This takes time $\mathcal{O}(s)$ per each local reflection. Step 2 of the local reflection can be implemented in constant time. Thus, each local reflection takes time $\mathcal{O}(s)$. There are $D-1$ local reflections performed. By Lemma 4.6 with all ϕ_i being absent, the whole transducer can be implemented in time $\mathcal{O}(D \cdot s)$. Transformation into canonical form in Proposition 10.4 keeps the time complexity of the transducer essentially the same.

Now consider the QRAG model. All local reflections in R_1 can be performed in parallel, and the same is true for R_2. The first and the third operation in the local reflection are implemented by the RA input oracle. The second operation can be performed in $\mathcal{O}(T_{\mathrm{R}})$ time by Theorem 4.3. □

13.2 Non-Boolean Case

This time let $\mathcal{M} = \mathcal{B} \otimes \mathcal{N}$ be a space with $\mathcal{B} = \mathbb{C}^p$. Let O_ψ be an oracle that performs the following state generation:

$$O_\psi \colon |0\rangle_{\mathcal{M}} \mapsto |\psi\rangle_{\mathcal{M}} = \sum_{j=0}^{p-1} |j\rangle_{\mathcal{B}} |\psi_j\rangle_{\mathcal{N}}. \tag{13.10}$$

Let $d > 0$ be a constant. We assume that for every ψ there exists (unique) $f(\psi) \in [p]$ such that

$$\|\psi_{f(\psi)}\|^2 \geqslant \frac{1}{2} + d. \tag{13.11}$$

Define

$$\mu = 2\sqrt{1/4 - d^2} < 1,$$

which is the same as in (13.3) for $c = 1/2$. We treat $\mathcal{B} = \mathbb{C}^p$ as composed out of $\ell = \log p$ qubits. We do *not* assume that $\ell = \mathcal{O}(T_{\mathrm{R}})$ here, as functions, in principle, can have very long output. Again, we call every input oracle O_ψ satisfying (13.10) and (13.11) admissible.

Theorem 13.3. *Let D be a positive integer. Under the above assumptions, there exists a perturbed transducer S_{pur} on the public space $\mathbb{C}^p$ and with bidirectional access to O_ψ which satisfies the following conditions:*

- *For all $b \in \{0,1\}^\ell$, and admissible O_ψ, it transduces $|b\rangle$ into $\big|b \oplus f(\psi)\big\rangle$, where $\oplus$ stands for the bit-wise XOR. In particular, every initial vector in $\mathbb{C}^p$ is admissible.*
- *On any admissible input oracle and unit initial vector, its perturbation is at most $\delta_{\mathrm{pur}} = 2\mu^{D-1}$.*
- *Its query complexity satisfies $L_{\max}(S_{\mathrm{pur}}) = \mathcal{O}\big(1/(1-\mu)\big) = \mathcal{O}(1)$.*
- *It executes the input oracle on the admissible subspace only: O_ψ on $|0\rangle_{\mathcal{M}}$ and O_ψ^* on $|\psi\rangle_{\mathcal{M}}$.*

For the time and the transduction complexity, we have the following estimates:

- *In the circuit model,* $W_{\max}(S_{\mathrm{pur}}) = \mathcal{O}\big(1/(1-\mu)\big) = \mathcal{O}(1)$ *and* $T(S_{\mathrm{pur}}) = \mathcal{O}(s \cdot D)$*, where* s *is the number of qubits used in* $\mathcal{M}$*.*
- *In the QRAG model, assuming the RA input oracle, we have* $W_{\max}(S_{\mathrm{pur}}) = \mathcal{O}(\log p/(1-\mu)) = \mathcal{O}(\log p)$ *and* $T(S_{\mathrm{pur}}) = \mathcal{O}(T_{\mathrm{R}})$*.*

Here we use $W_{\max}(S_{\mathrm{pur}})$ to denote maximal $W(S_{\mathrm{pur}}, \overleftrightarrow{O_\psi}, \xi)$ over all admissible input oracles O_ψ and admissible normalised initial states ξ. $L_{\max}(S_{\mathrm{pur}})$ is defined similarly.

Proof. We reduce the non-Boolean case to the Boolean case of Theorem 13.1 by encoding the value into the phase and using the Bernstein-Vazirani algorithm [18] to decode it back.

For simplicity of notation, we will assume $p = 2^\ell$ so that O_ψ in (13.10) just does not use the extra dimensions. For $a, b \in [p]$, we denote by $a \odot b \in \{0,1\}$ their inner product when considered as elements of $\mathbb{F}_2^\ell$.

Denote the public space $\mathbb{C}^p$ of S_{pur} by $\mathcal{J}$. We first apply the Hadamard $H^{\otimes \ell}$ to perform the following transformation

$$H^{\otimes \ell} \colon |b\rangle_{\mathcal{J}} \longmapsto \frac{1}{\sqrt{p}} \sum_{i=0}^{p-1} (-1)^{i \odot b} |i\rangle_{\mathcal{J}}. \tag{13.12}$$

Consider the following procedure E that evaluates the inner product between i and the output of the oracle into an additional qubit $\mathcal{Z}$:

$$E(O_\psi) \colon |i\rangle_{\mathcal{J}} |0\rangle_{\mathcal{M}} |0\rangle_{\mathcal{Z}} \overset{O_\psi}{\longmapsto} |i\rangle_{\mathcal{J}} \sum_{j=0}^{p-1} |j\rangle_{\mathcal{B}} |\psi_j\rangle_{\mathcal{N}} |0\rangle_{\mathcal{Z}} \longmapsto |i\rangle_{\mathcal{J}} \sum_{j=0}^{p-1} |j\rangle_{\mathcal{B}} |\psi_j\rangle_{\mathcal{N}} |i \odot j\rangle_{\mathcal{Z}}.$$

We consider it as a direct sum $E = \bigoplus_{i \in [p]} E^{(i)}$ with

$$E^{(i)}(O_\psi) \colon |0\rangle_{\mathcal{M}} |0\rangle_{\mathcal{Z}} \longmapsto \sum_{j=0}^{p-1} |j\rangle_{\mathcal{B}} |\psi_j\rangle_{\mathcal{N}} |i \odot j\rangle_{\mathcal{Z}}. \tag{13.13}$$

We convert them into canonical transducers S_E and $S_E^{(i)}$. We have $L_{\max}\big(S_E^{(i)}\big) = 1$, where the admissible subspace is $|0\rangle_{\mathcal{M}} |0\rangle_{\mathcal{Z}}$. Using Propositions 9.1 and 9.4, we get transducers

$$\overleftrightarrow{S_E}\big(\overleftrightarrow{O_\psi}\big) = S_E(O_\psi) \oplus S_E^{-1}(O_\psi^*) \qquad \text{and} \qquad \overleftrightarrow{S_E}^{(i)}\big(\overleftrightarrow{O_\psi}\big) = S_E^{(i)}(O_\psi) \oplus (S_E^{(i)})^{-1}(O_\psi^*).$$

Again, $L_{\max}\big(\overleftrightarrow{S_E}^{(i)}\big) = 1$. We still have $\overleftrightarrow{S_E} = \bigoplus_i \overleftrightarrow{S_E}^{(i)}$ with the help of Remark 9.5. It is also clear that $\overleftrightarrow{S_E}$ only executes the input oracle $\overleftrightarrow{O_\psi}$ on the admissible subspace.

We treat $E^{(i)}(O_\psi)$ as an oracle encoding a Boolean value into the register $\mathcal{Z}$. By (13.11), we get that $E^{(i)}(O_\psi)$ evaluates $i \odot f(\psi)$ with bounded error. Take the purifier S'_{pur} from Theorem 13.1 with $c = 1/2$ and the same values of d and D. This purifier satisfies

$$S'_{\mathrm{pur}}\Big(\overleftrightarrow{E^{(i)}(O_\psi)}\Big) \colon |0\rangle \rightsquigarrow (-1)^{i \odot f(\psi)} |0\rangle.$$

Taking direct sum over all $i \in [p]$, we get that a transducer

$$(I_{\mathcal{J}} \otimes S'_{\mathrm{pur}})\Big(\overleftrightarrow{E(O_\psi)}\Big) = \bigoplus_{i=0}^{p-1} S'_{\mathrm{pur}}\Big(\overleftrightarrow{E^{(i)}(O_\psi)}\Big) \tag{13.14}$$

performs the following transduction:

$$\frac{1}{\sqrt{p}} \sum_{i=0}^{p-1} (-1)^{i \odot b} |i\rangle_{\mathcal{J}} \rightsquigarrow \frac{1}{\sqrt{p}} \sum_{i=0}^{p-1} (-1)^{i \odot b + i \odot f(\psi)} |i\rangle_{\mathcal{J}}. \tag{13.15}$$

Finally, we again apply $H^{\otimes\ell}$ to get

$$H^{\otimes\ell}\colon \frac{1}{\sqrt{p}}\sum_{i=0}^{p-1}(-1)^{i\odot b+i\odot f(\psi)}|i\rangle_{\mathcal{J}} \longmapsto |b\oplus f(\psi)\rangle_{\mathcal{J}}. \tag{13.16}$$

Combining (13.12), (13.15) and (13.16), we see that we can use sequential composition of Proposition 9.9 to get

$$S_{\mathrm{pur}} = S_{H^{\otimes\ell}} * \big((I_{\mathcal{J}}\otimes S'_{\mathrm{pur}})\circ \overleftrightarrow{S_E}\big) * S_{H^{\otimes\ell}},$$

where $S_{H^{\otimes\ell}}$ is a transducer from Proposition 10.1 with transduction action $H^{\otimes\ell}$ and no input oracle.

Let us estimate query complexity on a unit vector $\xi\in\mathcal{J}$. We have the following estimate, where we explain individual lines after the equation.

$$\begin{aligned}
L(S_{\mathrm{pur}},\overleftrightarrow{O_\psi},\xi) &= L\big((I_{\mathcal{J}}\otimes S'_{\mathrm{pur}})\circ\overleftrightarrow{S_E},\overleftrightarrow{O_\psi},H^{\otimes\ell}\xi\big)\\
&= \sum_{i=0}^{p-1} L\big(S'_{\mathrm{pur}}\circ\overleftrightarrow{S_E}^{(i)},\overleftrightarrow{O_\psi},\phi_i\big)\\
&\leqslant \sum_{i=0}^{p-1} L_{\max}\big(S'_{\mathrm{pur}}\big)L_{\max}\big(\overleftrightarrow{S_E}^{(i)}\big)\|\phi_i\|^2\\
&\leqslant \mathcal{O}(1/(1-\mu))\sum_{i=0}^{p-1}\|\phi_i\|^2 = \mathcal{O}(1/(1-\mu)).
\end{aligned}$$

On the first line, we used sequential composition of Proposition 9.9 and that the transducer $S_{H^{\otimes\ell}}$ does not use the input oracle. On the second line, we decomposed $H^{\otimes\ell}\xi = \bigoplus_i \phi_i$, and used (13.14) and Proposition 9.4. On the third line, we used functional composition of Proposition 9.11, Eq. (9.1), and that S'_{pur} does not have direct access to O_ψ and executes its input oracle $E^{(i)}$ on the admissible subspace only.

In a similar way, we have

$$\begin{aligned}
W(S_{\mathrm{pur}},\overleftrightarrow{O_\psi},\xi) &= 2W_{\max}(S_{H^{\otimes\ell}}) + W\big((I_{\mathcal{J}}\otimes S'_{\mathrm{pur}})\circ\overleftrightarrow{S_E},\overleftrightarrow{O_\psi},H^{\otimes\ell}\xi\big)\\
&= 2W_{\max}(S_{H^{\otimes\ell}}) + \sum_{i=0}^{p-1} W\big(S'_{\mathrm{pur}}\circ\overleftrightarrow{S_E}^{(i)},\overleftrightarrow{O_\psi},\phi_i\big)\\
&\leqslant 2W_{\max}(S_{H^{\otimes\ell}}) + \sum_{i=0}^{p-1}\Big[W_{\max}\big(S'_{\mathrm{pur}}\big) + L_{\max}\big(S'_{\mathrm{pur}}\big)W_{\max}\big(\overleftrightarrow{S_E}^{(i)}\big)\Big]\|\phi_i\|^2\\
&\leqslant 2W_{\max}(S_{H^{\otimes\ell}}) + W_{\max}(S'_{\mathrm{pur}}) + L_{\max}(S'_{\mathrm{pur}})\max_i W_{\max}\big(\overleftrightarrow{S_E}^{(i)}\big)\\
&\leqslant 2W_{\max}(S_{H^{\otimes\ell}}) + \mathcal{O}\big(1/(1-\mu)\big)\max_i\Big(1+W_{\max}\big(\overleftrightarrow{S_E}^{(i)}\big)\Big).
\end{aligned}\tag{13.17}$$

For the perturbation, we have using Proposition 12.4:

$$\begin{aligned}
\delta(S_{\mathrm{pur}},\overleftrightarrow{O_\psi},\xi) &= \delta\big((I_{\mathcal{J}}\otimes S'_{\mathrm{pur}})\circ\overleftrightarrow{S_E},\overleftrightarrow{O_\psi},H^{\otimes\ell}\xi\big)\\
&= \sqrt{\sum_{i=0}^{p-1}\delta\big(S'_{\mathrm{pur}},E^{(i)}(O_\psi),\phi_i\big)^2} \leqslant \sqrt{\sum_{i=0}^{p-1}\big(\delta_{\mathrm{pur}}\|\phi_i\|\big)^2} = \delta_{\mathrm{pur}}.
\end{aligned}$$

Finally,

$$T(S_{\text{pur}}) = 2T_{\text{C}}(S_{H^{\otimes \ell}}) + T_{\text{C}}(S'_{\text{pur}}) + T_{\text{C}}(\overleftrightarrow{S_E}) + \mathcal{O}(1). \tag{13.18}$$

In the circuit model, the transduction complexities of $S_{H^{\otimes \ell}}$ and $\overleftrightarrow{S_E}^{(i)}$ are 0 and 1, respectively, and we get the required estimate from (13.17). Also, both $T(S_{H^{\otimes \ell}})$ and $T(\overleftrightarrow{S_E})$ are $\mathcal{O}(\log p)$, and $T(S'_{\text{pur}}) = \mathcal{O}(s \cdot D)$. Since $s \geqslant \log p$, we get the required estimate from (13.18).

In the QRAG model, we have both $W_{\max}(S_{H^{\otimes \ell}})$ and $W_{\max}(\overleftrightarrow{S_E}^{(i)})$ bounded by $\mathcal{O}(\log p)$, which gives the required estimate on the transduction complexity. For the time complexity, all the terms in (13.18) are $\mathcal{O}(T_{\text{R}})$, which shows that $T(S_{\text{pur}}) = \mathcal{O}(T_{\text{R}})$. □

13.3 Composition of Bounded-Error Algorithms

Purifier can be composed with algorithms evaluating functions with bounded error to reduce the error. In this section, we mention some examples.

Since we are ignoring the constant factors, we will assume the standard version of the input oracle: $O_x \colon |i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$. In particular, it is its own inverse, and we use O_x instead of $\overleftrightarrow{O_x}$ everywhere in this section.

Let us again recall the composed function $f \circ g$ from (11.2):

$$\begin{aligned}(f \circ g)(z_{1,1}, \dots, z_{1,m},\ \ z_{2,1}, \dots, z_{2,m},\ \ \dots\dots,\ \ z_{n,1}, \dots, z_{n,m}) \\ = f\big(g(z_{1,1}, \dots, z_{1,m}), g(z_{2,1}, \dots, z_{2,m}), \dots, g(z_{n,1}, \dots, z_{n,m})\big).\end{aligned} \tag{13.19}$$

and

$$\vec{y}_i = (z_{i,1}, \dots, z_{i,m}) \qquad \text{and} \qquad x = \big(g(\vec{y}_1), g(\vec{y}_2), \dots g(\vec{y}_n)\big) \tag{13.20}$$

so that $f(x) = (f \circ g)(z)$. The following result is essentially Theorem 3.15.

Theorem 13.4. *Let A and B be quantum algorithms in the circuit model that evaluate functions f and g, respectively, with bounded error. Then, there exists an algorithm in the circuit model that evaluates the function $f \circ g$ with bounded error in time complexity*

$$\mathcal{O}(L)\big(T(A) + T(B) + s \log L\big) \tag{13.21}$$

where L is the worst-case Las Vegas query complexity of A, and s is the space complexity of B. The algorithm makes $\mathcal{O}\big(L \cdot Q(B)\big)$ queries, where $Q(B)$ is the usual Monte Carlo query complexity of B.

Proof. First, use Theorem 10.3 to get a transducer S_A whose transduction action is identical to the execution of A. Its time complexity $T(S_A) = \mathcal{O}\big(T(A)\big)$ and its transduction and query complexities are bounded by L.

The algorithm B on the input oracle O_y evaluates $g(y)$ with bounded error. We obtain the algorithm B^{-1} with the input oracle $O_y^* = O_y$ whose action is the inverse of B. Combining the two via direct sum, we get the algorithm $\overleftrightarrow{B}(O_y) = B(O_y) \oplus B^{-1}(O_y)$. Let S_{pur} be the corresponding purifier from Theorem 13.3, and D and δ_{pur} be the parameters therein. The transduction action of S_{pur} on the input oracle $\overleftrightarrow{B}(O_y)$ is $|b\rangle \rightsquigarrow |b \oplus g(y)\rangle$.

By (13.20), we have $O_z = \bigoplus_i O_{\vec{y}_i}$. Let us denote by $\overleftrightarrow{B}(O_z)$ the algorithm $(I_n \otimes \overleftrightarrow{B})(O_z) = \bigoplus_i \overleftrightarrow{B}(O_{\vec{y}_i})$. By Corollary 9.7 with Remark 9.5, the transducer $I_n \otimes S_{\text{pur}}$ on the input oracle $\overleftrightarrow{B}(O_z)$ performs the transduction $|i\rangle|b\rangle \rightsquigarrow |i\rangle|b \oplus g(\vec{y}_i)\rangle$ for every $i \in [n]$. In other words, its transduction action is O_x.

Now consider the transducer

$$S = S_A \circ (I_n \otimes S_{\text{pur}})$$

with the input oracle $\overleftarrow{B}(O_z)$. By the definition of functional composition, it transduces $|0\rangle$ into $(g \circ f)(z)$. By (3.21), its transduction complexity is at most

$$W(S_A, O_x, |0\rangle) + \sum_i L^{(i)}(S_A, O_x, |0\rangle) \cdot W_{\max}\big(S_{\text{pur}}, \overleftarrow{B}(O_{\vec{y}_i})\big) = \mathcal{O}(L).$$

We obtain the required algorithm by using Corollary 12.2 with $\varepsilon = \Theta(1)$ on the above transducer S. The transducer S is executed $\mathcal{O}(L)$ times. Each execution takes times $\mathcal{O}\big(T(A) + sD\big)$ to execute the transducer and $\mathcal{O}\big(T(B)\big)$ to execute the input oracle. Therefore, the total time complexity is

$$\mathcal{O}(L)\big(T(A) + T(B) + sD\big)$$

and the query complexity is $\mathcal{O}(L)Q(B)$.

It remains to estimate D. We may assume the error of A is a small enough constant. By Corollary 12.2, in order to get a bounded-error algorithm from the transducer $S_A \circ (I_n \otimes S_{\text{pur}})$, we should have

$$\delta\big(I_n \otimes S_{\text{pur}}, \overleftarrow{B}(O_z), q(S_A, O_x, |0\rangle)\big) = \mathcal{O}\big(1/\sqrt{L}\big).$$

for a small enough constant. Using (12.1) and that $\|q(S_A, O_x, |0\rangle)\| \leqslant \sqrt{L}$, we get that it suffices to have $\delta_{\text{pur}} = \mathcal{O}(1/L)$. Therefore, we can take $D = \mathcal{O}(\log L)$, which finishes the proof. □

Let us now proceed with QRAG case, Theorem 3.16. Recall the function from (3.40):

$$f\big(g_1(z_{1,1}, \dots, z_{1,m}), g_2(z_{2,1}, \dots, z_{2,m}), \dots, g_n(z_{n,1}, \dots, z_{n,m})\big). \tag{13.22}$$

with notation

$$\vec{y}_i = (z_{i,1}, \dots, z_{i,m}) \qquad \text{and} \qquad x = \big(g_1(\vec{y}_1), g_2(\vec{y}_2), \dots g_n(\vec{y}_n)\big).$$

For simplicity, we assume all functions f and g use the same input and output alphabet q. Let A and B_i be quantum algorithms that evaluate f and g_i, respectively. To simplify expressions, we assume that $T(B_i) \geqslant \log q$. In other words, we spend at least 1 iteration per bit of the output. Also, all the algorithms have the same upper bound on permissible error. We use an approach similar to Section 11 on iterated functions, so that we are able to compose several layer of functions.

As in Section 8.2, we denote $L_x(A) = L(A, O_x, |0\rangle)$ and similarly for other notation. This time, however the input oracle $O_x\colon |i\rangle|b\rangle \mapsto |i\rangle|b \oplus x_i\rangle$ is uniquely defined. As we can make the perturbation of a purifier as small as necessary without increasing complexity, we ignore the perturbations in the following implicitly assuming they are small enough.

Theorem 13.5. *Let S_f be a perturbed transducer evaluating the function f, and $B_1, \dots, B_n$ be algorithms evaluating the functions $g_1, \dots, g_n$ with bounded-error. Assuming the QRAG model with RA input oracle, and QRAM access to the description of $B_1, \dots, B_n$, there exists a perturbed transducer S evaluating the function (13.22) with the following parameters. Its transduction complexity is*

$$W(S, O_z, |0\rangle) = W(S_f, O_x, |0\rangle) + \sum_{i=1}^{n} \mathcal{O}\big(L_x^{(i)}(S_f) T(B_i)\big) \tag{13.23}$$

its query complexity is

$$L^{(i,j)}(S, O_z, |0\rangle) = \mathcal{O}\big(L_x^{(i)}(S_f) L_{\vec{y}_i}^{(j)}(B_i)\big) \tag{13.24}$$

and its time complexity is $T(S) = T_{\text{C}}(S_f) + \mathcal{O}(T_{\text{R}})$.

Proof. From Theorem 10.5, for each i, we obtain a transducer S_{B_i} whose transduction action is identical to the action of B_i. Its transduction complexity $W_{\max}(S_{B_i}) = \mathcal{O}\big(T(B_i)\big)$ and query state is identical to that of B_i. Also, we obtain $\overleftrightarrow{S_{B_i}} = S_{B_i} \oplus S_{B_i}^*$ whose complexity is identical to S_{B_i}. Let S_{pur} be the corresponding purifier. We have that the transduction action of $S_{\mathrm{pur}} \circ \overleftrightarrow{S_{B_i}}$ on an input oracle O_y is $|b\rangle \rightsquigarrow |b \oplus g_i(y)\rangle$.

Using direct sum, we get that the transducer

$$S_g = (I_n \otimes S_{\mathrm{pur}}) \circ \bigoplus_i \overleftrightarrow{S_{B_i}} = \bigoplus_i S_{\mathrm{pur}} \circ \overleftrightarrow{S_{B_i}}$$

on the input oracle $O_z = \bigoplus_i O_{\vec{y}_i}$ transduces $|i\rangle|b\rangle \rightsquigarrow |i\rangle|b \oplus g_i(\vec{y}_i)\rangle$ for every $i \in [n]$. Thus, its transduction action is O_x, and the transducer $S = S_f \circ S_g$ evaluates the function in (13.22).

Let us estimate its transduction complexity. First by (3.17):

$$W_{\max}(S_{\mathrm{pur}} \circ \overleftrightarrow{S_{B_i}}, O_{\vec{y}_i}) \leqslant W_{\max}(S_{\mathrm{pur}}) + L_{\max}(S_{\mathrm{pur}}) W_{\max}(\overleftrightarrow{S_{B_i}}) = \mathcal{O}(\log p) + \mathcal{O}(T(B_i)) = \mathcal{O}(T(B_i))$$

using our assumption on $T(B_i) \geqslant \log p$. Therefore, by (3.21):

$$\begin{aligned} W\big(S, O_z, |0\rangle\big) &\leqslant W(S_f, O_x, |0\rangle) + \sum_i L_x^{(i)}(S_f) W_{\max}\big(S_{\mathrm{pur}} \circ \overleftrightarrow{S_{B_i}}, O_{\vec{y}_i}\big) \\ &\leqslant W(S_f, O_x, |0\rangle) + \sum_i L_x^{(i)}(S_f) \mathcal{O}(T(B_i)). \end{aligned}$$

For the query complexity, we use a partial-query variant of (3.19) and that the purifier only executes the subroutine on the admissible initial states to obtain:

$$L_{\max}^{(j)}\big(S_{\mathrm{pur}} \circ \overleftrightarrow{S_{B_i}}, O_{\vec{y}_i}\big) \leqslant L_{\max}(S_{\mathrm{pur}}) L^{(j)}\big(S_{B_i}, O_{\vec{y}_i}, |0\rangle\big) = \mathcal{O}\big(L_{\vec{y}_i}^{(j)}(B_i)\big).$$

Hence, by (3.24):

$$L^{(i,j)}\big(S, O_z, |0\rangle\big) \leqslant L_x^{(i)}(A) L_{\max}^{(j)}\big(S_{\mathrm{pur}} \circ \overleftrightarrow{S_{B_i}}, O_{\vec{y}_i}\big) \leqslant L_x^{(i)}(A) \mathcal{O}\big(L_{\vec{y}_i}^{(j)}(B_i)\big).$$

The time complexity of S_{pur} is $\mathcal{O}(T_{\mathrm{R}})$. Also, the direct sum $\bigoplus_i \overleftrightarrow{S_{B_i}}$ can be implemented in $\mathcal{O}(T_{\mathrm{R}})$ by Proposition 10.6. Thus, the time complexity of S is $T_{\mathrm{C}}(S_f) + \mathcal{O}(T_{\mathrm{R}})$. □

We get Theorem 3.16 from this theorem by using a transducer S_f obtained from the program A using Theorem 10.5, and then applying Proposition 12.3 to the resulting transducer. Moreover, we get that the algorithm executes the input oracle O_z

$$\mathcal{O}\Big(\max_z \sum_{i=1}^n L_x^{(i)}(A) L_{\vec{y}_i}(B_i)\Big)$$

times.

Theorem 13.5 can be used multiple times in a row to obtain a composed transducer for a tree of functions similar to the one in Theorem 3.11. If d is the depth of the tree, the query complexity grows by the factor of C^d, where C is the constant in (13.24). This growth is completely analogous to what one obtains using span programs for the query complexity. The contribution to the time complexity from the subroutines on layer ℓ also grows by the factor of C^ℓ because they are multiplied by the corresponding query complexity in (13.23). The time complexity of the final transducer is $\mathcal{O}(dT_{\mathrm{R}})$.

Acknowledgements

We would like to thank Titouan Carette for bringing references [8, 7] to our attention. We are grateful to anonymous referees for their useful suggestions on the presentation of this paper.

AB is supported by the Latvian Quantum Initiative under European Union Recovery and Resilience Facility project no. 2.3.1.1.i.0/1/22/I/CFLA/001 and the QuantERA project QOPT.

SJ is supported by NWO Klein project number OCENW.Klein.061; and ARO contract no W911NF2010327. SJ is funded by the European Union (ERC, ASC-Q, 101040624). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. SJ is supported by the project Divide & Quantum (with project number 1389.20.241) of the research programme NWA-ORC which is (partly) financed by the Dutch Research Council (NWO). SJ is a CIFAR Fellow in the Quantum Information Science Program.

References

[1] A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. Earlier: *FOCS'04*, arXiv:quant-ph/0311001. 7, 15

[2] A. Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3):786–807, 2010. Earlier: *STACS'08*, arXiv:quant-ph/0609188. 4, 5, 36

[3] A. Ambainis. Variable time amplitude amplification and quantum algorithms for linear algebra problems. In *Proc. of 29th STACS*, volume 14 of *LIPIcs*, pages 636–647. Dagstuhl, 2012. arXiv:1010.4458. 4

[4] A. Ambainis, A. Belovs, O. Regev, and R. de Wolf. Efficient Quantum Algorithms for (Gapped) Group Testing and Junta Testing. In *Proc. of 27th ACM-SIAM SODA*, pages 903–922, 2016. arXiv:1507.03126. 17

[5] A. Ambainis, A. M. Childs, B. W. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. *SIAM Journal on Computing*, 39(6):2513–2530, 2010. Earlier: *FOCS'07*. 4, 25

[6] A. Ambainis, M. Kokainis, and J. Vihrovs. Improved algorithm and lower bound for variable time quantum search. In *Proc. of 18th TQC*, volume 266 of *LIPIcs*, pages 7:1–7:18, 2023. arXiv:2302.06749. 4

[7] P. Andrés-Martínez. *Unbounded loops in quantum programs: categories and weak while loops*. PhD thesis, University of Edinburgh, 2022. arXiv:2212.05371. 13, 79

[8] M. Bartha. Quantum Turing automata. In *Proc. of 8th DCM*, volume 143 of *EPTCS*, pages 17–31, 2014. arXiv:1404.0074. 13, 79

[9] S. Beigi and L. Taghavi. Span program for non-binary functions. *Quantum Information & Computation*, 19(9):760—792, 2019. arXiv:1805.02714. 17

[10] A. Belovs. Learning-graph-based quantum algorithm for k-distinctness. In *Proc. of 53rd IEEE FOCS*, pages 207–216, 2012. arXiv:1205.1534. 5

[11] A. Belovs. Span programs for functions with constant-sized 1-certificates. In *Proc. of 44th ACM STOC*, pages 77–84, 2012. arXiv:1105.4024. 15

[12] A. Belovs. Quantum walks and electric networks. arXiv:1302.3143, 2013. 15, 43, 44

[13] A. Belovs. *Applications of the Adversary Method in Quantum Query Algorithms*. PhD thesis, University of Latvia, 2014. arXiv:1402.3858. 15

[14] A. Belovs. Variations on quantum adversary. arXiv:1504.06943, 2015. 6, 17, 27, 29, 35, 51

[15] A. Belovs. Global phase helps in quantum search: Yet another look at the welded tree problem. arXiv:2404.19476, 2024. 43

[16] A. Belovs and B. W. Reichardt. Span programs and quantum algorithms for *st*-connectivity and claw detection. In *Proc. of 20th ESA*, volume 7501 of *LNCS*, pages 193–204. Springer, 2012. arXiv:1203.2603. 5, 17

[17] A. Belovs and D. Yolcu. One-way ticket to Las Vegas and the quantum adversary. arXiv:2301.02003, 2023. 4, 5, 7, 9, 12, 13, 17, 20, 32, 33, 35, 41, 45, 47, 50, 51

[18] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on Computing*, 26(5):1411–1473, 1997. Earlier: *STOC'93*. 27, 74

[19] G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74, 2002. arXiv:quant-ph/0005055. 15

[20] H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science*, 288:21–43, 2002. 32

[21] A. M. Childs, R. Cleve, E. Deotto, E. Farhi, S. Gutmann, and D. A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proc. of 35th ACM STOC*, pages 59–68, 2003. arXiv:quant-ph/0209131. 43

[22] R. Cleve. An introduction to quantum complexity theory. arXiv:quant-ph/9906111, 1999. 32

[23] A. Cornelissen, S. Jeffery, M. Ozols, and A. Piedrafita. Span programs and quantum time complexity. In *Proc. of 45th MFCS*, pages 26:1–26:14, 2020. arXiv:2005.01323. 5, 6, 7, 12, 17, 36

[24] E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the Hamiltonian NAND tree. *Theory of Computing*, 4:169–190, 2008. arXiv:quant-ph/0702144. 4, 25

[25] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proc. of 28th ACM STOC*, pages 212–219, 1996. arXiv:quant-ph/9605043. 15

[26] S. Jeffery. Quantum subroutine composition. arXiv:2209.14146, 2022. 5, 6, 7, 11, 12, 23, 36

[27] S. Jeffery. Span programs and quantum space complexity. *Theory of Computing*, 18(11):1–49, 2022. arXiv:1908.04232. 4

[28] S. Jeffery and S. Kimmel. Quantum algorithms for graph connectivity and formula evaluation. *Quantum*, 1:26, 2017. arXiv:1704.00765v3. 17

[29] S. Jeffery and S. Zur. Multidimensional quantum walks, with application to k-distinctness. In *Proc. of 55th ACM STOC*, pages 1125–1130, 2023. arXiv:2208.13492. 5, 6, 15, 27, 36

[30] T. Lee, R. Mittal, B. W. Reichardt, R. Špalek, and M. Szegedy. Quantum query complexity of state conversion. In *Proc. of 52nd IEEE FOCS*, pages 344–353, 2011. arXiv:1011.3020. 15, 17, 51

[31] N. Leonardos. An improved lower bound for the randomized decision tree complexity of recursive majority. In *Proc. of 40th ICALP, Part I*, volume 7965 of *LNCS*, pages 303–314. Springer, 2013. ECCC:2012/099. 25

[32] F. Magniez, A. Nayak, M. Santha, and D. Xiao. Improved bounds for the randomized decision tree complexity of recursive majority. In *Proc. of 38th ICALP*, volume 6755 of *LNCS*, pages 317–329. Springer, 2011. ECCC:2010/192. 25

[33] B. W. Reichardt. Span programs and quantum query complexity: The general adversary bound is nearly tight for every Boolean function. In *Proc. of 50th IEEE FOCS*, pages 544–551, 2009. arXiv:0904.2759. 4, 5, 17

[34] B. W. Reichardt. Reflections for quantum query algorithms. In *Proc. of 22nd ACM-SIAM SODA*, pages 560–569, 2011. arXiv:1005.1601. 15

[35] B. W. Reichardt. Span-program-based quantum algorithm for evaluating unbalanced formulas. In *Proc. of 6th TQC*, volume 6745 of *LNCS*, pages 73–103. Springer, 2014. arXiv:0907.1622. 4, 25

[36] B. W. Reichardt and R. Špalek. Span-program-based quantum algorithm for evaluating formulas. *Theory of Computing*, 8:291–319, 2012. Earlier: *STOC'08*, arXiv:0710.2630. 4, 5, 17, 25

[37] M. Saks and A. Wigderson. Probabilistic Boolean decision trees and the complexity of evaluating game trees. In *Proc. of 27th IEEE FOCS*, pages 29–38, 1986. 25

[38] M. Santha. On the Monte Carlo boolean decision tree complexity of read-once formulae. *Random Structures and Algorithms*, 6(1):75–87, 1995. 25

[39] M. Santha. Quantum walk based search algorithms. In *Proc. of 5th TAMC*, volume 4978 of *LNCS*, pages 31–46. Springer, 2008. arXiv:0808.0059. 14

[40] M. Snir. Lower bounds for probabilistic linear decision trees. *Theoretical Computer Science*, 38:69–82, 1985. 25

[41] M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *Proc. of 45th IEEE FOCS*, pages 32–41, 2004. 15

[42] B. Zhan, S. Kimmel, and A. Hassidim. Super-polynomial quantum speed-ups for Boolean evaluation trees with hidden structure. In *Proc. of 3rd ACM ITCS*, pages 249–265, 2012. arXiv:1101.0796. 4