Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ana Isabel Carvalho Neri

Towards Quantum Program Calculation

October 2018

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Ana Isabel Carvalho Neri

Towards Quantum Program Calculation

Master dissertation
Mestrado Integrado em Engenharia Física - Física da Informação

Dissertation supervised by
 José Nuno Oliveira (Univ. Minho)
 Rui Soares Barbosa (Univ. Oxford)

October 2018

ACKNOWLEDGEMENTS

ABSTRACT

Based on the similarity between the categorial derivation of classical programs from their specification and the category theory approach to quantum physics, this dissertation aims at extending the laws of classical program algebra to quantum programming.

In this context, the principles of the algebra of classical programs are applied to quantum programming, in order to verify the feasibility of creating correct-by-construction quantum circuits that can run on quantum devices available in the IBM Q Experience.

The reversibility restrictions of quantum circuits are ensured by minimal complements. Moreover, measurements are postponed to the end of recursive computations called "quantamorphisms" to avoid the collapse of quantum states. Quantamorphisms are classical catamorphisms extended to ensure quantum reversibility.

The derived quantamorphisms implement quantum cycles (vulg. for-loops) and quantum folds on lists. By Kleisli correspondence, quantamorphisms can be written as monadic functional programs with quantum parameters. This enables the use of Haskell, a monadic functional programming language, to perform the experimental work. The examples of the calculated quantum programs are simulated in Haskell, Quipper and QISKit and run on the quantum computers of the IBM Q Experience.

The main conclusions of this work are that, while all the simulations produced correspond to the predicted results, running these programs on real quantum devices results in a significant amount of errors. As quantum devices are constantly evolving, it is likely that in the near future these devices will increase their reliability, allowing programs to run more accurately.

The extension of the quantamorphism concept to more general input structures, such as finite trees, remains a challenge that is left for future work. Also relevant will be the study of conditional quantum control without measurements, which will extend the scope of quantamorphisms as quantum circuit specifications.

RESUMO

Tendo como base a similaridade entre a matemática categorial para derivar programas a partir da sua especificação e a teoria categorial usada na física quântica, esta dissertação pretende estender as leis da álgebra de programas clássicos à programação quântica.

Nesse contexto, a dissertação trata de explorar o significado desses princípios e suas construções na programação quântica e verificar a viabilidade de as aplicar à criação de programas quânticos correctos que possam correr em dispositivos quânticos disponíveis no IBM Q Experience.

As restrições de reversibilidades exigidas pela programação quântica são asseguradas por complemento mínimo, e para evitar o colapso dos estados quânticos a medição é adiada através de "quantamorfismos", nome dado à extensão reversível do conceito clássico de catamorfismo.

Os quantamorfismos que se implementaram permitem correr ciclos-*for* quânticos e *folds* quânticos sobre listas. Com base na correspondência de Kleisli é possível escrevê-los como programas funcionais monádicos com parâmetros quânticos. Para esse efeito recorre-se à linguagem de programação funcional Haskell como base para o trabalho experimental. Os exemplos dos programas quânticos calculados foram simulados em Haskell, Quipper e QISKit e correram nos computadores quânticos da IBM Q Experience.

Constata-se que, enquanto todas as simulações produzidas correspondem ao previsto, correr estes programas em máquinas reais resulta numa quantidade significativa de erros. Como os dispositivos quânticos estão em constante evolução, é provável que num futuro próximo estes dispositivos aumentem a sua fiabilidade, permitindo que os programas corram de forma mais precisa.

Entre as questões que esta tese levanta inclui-se a extensão dos seus resultados a estruturas de entrada mais gerais, como por exemplo árvores, e estruturas de controlo condicionais que não efectuem medidas e que assim possam estender o âmbito do quantamorfismo como veículo de especificação de circuitos quânticos.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1.1 THE RISE OF QUANTUM COMPUTING

At the time of writing this dissertation, most of the industry sector worldwide is still finding out reasons to invest in quantum computing. By contrast, companies at the vanguard of quantum technology (e.g. IBM, Google, Microsoft) are already exploring it. There is nevertheless a growing interest from specific companies in the potential advantages offered by quantum technologies, namely in the automotive and telecommunication sectors (Simon et al., 2006; 2018, 2017; Team, 2018).

Meanwhile, academic institutions all over the world are making visible progress, for example researchers in China who have recently broken the record for the number of entangled qubits (Wang et al., 2018). Such increasing research activity around quantum technologies is also leading to new consortia between industry and academia. An example of this is the IBM Q Network, which involves a number of companies (e.g. Mitsubishi Chemical) as well as academic institutions (e.g. the University of Oxford) aiming primarily at sharing know-how.

Figure 1.1.: IBM quantum computer processor.

As part of this increasing concern for quantum computing, the IBM Quantum Network has recently added QuantaLab as their Academic Partner in Portugal (Shifter, 2018). QuantaLab is a consortium involving the University of Minho (UM), the International Iberian Nanotechnology Laboratory (INL), the Centre of Engineering and Product Development (CEiiA) and INESC TEC (Institute for Systems and Computer Engineering, Technology and Science). The main goal of this academic hub is to explore how quantum computing can be used to make simulations and tests in several resource-demanding fields (Shifter, 2018).

It should be mentioned that most of the tools used in these IBM Q hubs are quantum devices based on quantum gates available online to anyone interested in the subject.[1] Anyway, quantum computation devices such as those available from the IBM Q Experiment are still relying on rudimentary machinery that gives room to scepticism.

Nevertheless, due to the accumulation of problems that cannot be (efficiently) solved by classic computers and the physical limitation brought on classic systems due to the decreasing of the size of circuits, quantum computation appears as a prime candidate to be thoroughly studied as a viable option.

1.2 FROM CLASSICAL TO QUANTUM PROGRAMMING

Those familiar with the history of classic computation will have a feeling of *déjà vu* when looking at the (more recent) history of quantum computation.

Classical computing blends knowledge of philosophy, linguistics, mathematics and physics. The story of classic computers started from mathematical abstractions which led in particular to the so-called *Turing machine* (Turing, 1936), which is still regarded as the canonical, abstract notion of a programmable computer, and to the λ-calculus (Church, 1936), a mathematical system that provided a basis for functional programming.

The step from abstraction to reality was possible due to advances in physics like the invention of triodes (1912) and then of transistors (1948), leading to the integrated circuits that are the basis of the *in silico* technology of today (Nebeker, 2009; Routray, 2004; Laws, 2013). Once such devices were first employed to store information in realistic situations it became clear that further abstraction was required. This led to the explicit adoption of formal logic, an abstraction still in use today — as the aphorism says, *"Logic is the language of computing"*.

Analogously to classical computing, but several decades later, the quantum computing trend was born out of mathematical abstractions again, this with time the description of the first universal quantum computer by Deutsch (1985). And the parallel goes on: nowadays, physicists are testing methods to implement such abstract concepts, linking theory to reality once again.

In a similar fashion to what happened in classical computation, software is getting into the quantum computation history. The birth of software as an independent technology took place in the 1950s. (The first programming language, Fortran, appeared in 1953.) But soon it was faced with a crisis because an effective discipline of programming was lacking. The term *software engineering* appeared in the 1960s and was the subject of a conference supported by NATO, in 1968, in Garmisch, Germany. People at the conference expressed concerns and called for theoretical foundations. This resulted in the birth of the principles of *structured*

---

[1]For details visit the IBM Q Experiment website, https://quantumexperience.ng.bluemix.net/qx/experience.

programming that became popular in the 1970s. But, in a sense, the 1968 crisis is not over yet: the complexity of the software that the IT sector requires programmers to build every day leads to unsafe code due to the widespread use of informal, ad hoc methods, instead of the mathematically sound methods anticipated by its founding fathers.

The problem of software engineering is that quality control is based on *testing* the software *after* it is built, and not on ensuring quality in a stepwise manner, as advocated by academia since the 1970s.

Some believe that the problem is lack of mathematical abstraction once again (Kramer, 2007). Stepping back to the initial computation abstractions of the 1930s, lambda calculus was developed with the aim of creating a model of computation based exclusively on function abstraction and application. This led to a mathematically robust style of programming known as functional programming (FP), which has become a reliable paradigm for producing software. The *correct-by-construction* programming techniques proposed by this field have shown a significant impact on software theory. Such techniques promise significant reduction in development costs by avoiding dependence on testing and debugging.

Correct-by-construction advocates the calculation of programs from problem specifications. In the functional setting, such formal methods are based on a so-called *"Algebra of Programming"* (AoP) which is the subject of a textbook by Bird and de Moor (1997). The branch of mathematics used to formalise the abstractions in AoP is category theory (MacLane, 1971).

Despite its strong algebraic basis — cf. Hilbert spaces, linear algebra, etc. (Nielsen and Chuang, 2011) — quantum mechanics is still a counter-intuitive field, and one that will require further abstractions for programmers. In quantum mechanics, every measure implies a destruction of the superposition state, spoiling the quantum advantage altogether. This renders current *debugging* strategies obsolete and nearly impossible: one needs to *get it right* from the very beginning!

Because testing and debugging won't apply to the specific case of quantum programming, at least in their current standards, the traditional life-cycle based on *edit-compile-run* is not an option. This further increases the need for correct-by-construction methods in quantum programming.

1.3 OVERVIEW OF THE LITERATURE

The literature on quantum computing is vast. Two standard textbooks written by Nielsen and Chuang (2011) and Yanofsky and Mannucci (2008) thoroughly present the quantum world to programmers. These books start from basic notions of quantum mechanics and lead the topic to quantum computing, giving insights on quantum circuits, on quantum algorithms and even on how the hardware of a quantum computer works.

The related topic of reversible computing (Bennett, 1973) was triggered by the so-called *Landauer Principle* (Landauer, 1961), stating that, with increased reversibility it is possible to reduce the energy consumption of software systems. Although recent papers dismiss the thermodynamic motivation (López-Suárez et al., 2016), reversibility is a well-developed research theme in the intersection of computing with quantum physics (Mu et al., 2004).

The mathematical basis of this dissertation owes much to books such as (Coecke, 2011) and (Bird and de Moor, 1997), which supply definitions, examples and exercises that allow an easy introduction to category theory and categorial logic. Coecke (2011) stresses on the importance of these theories to computer science and physics, showing how powerful and flexible categories can be in a small and large scale. These textbooks provide a large part of the background of this dissertation.

Coecke and Kissinger (2017) explore diagrams common in category theory to explain some of the most interesting properties of quantum computing. Concerning quantum programming languages, Selinger (2004) goes further and outlines the syntax and semantic rules of a functional quantum programming language with some high-level features, and Altenkirch and Grattage (2005) move a step forward by aiming at quantum control and quantum data. These works result in several quantum language proposals. One of these, named Quipper (Green et al., 2013a), will play an important role in the work described in this dissertation.

1.4 AIMS OF THE DISSERTATION

The main research questions this thesis wishes to address are the following:

1. Is it possible to extend the MPC[2] culture, principles and constructions – which have been so effective in disciplining the whole field of recursive functional programming and data structuring – to quantum programming?

2. Is it viable to apply such constructions to derive programs down to the level of actually *running* them on the experimental quantum machines of today?

An important requirement to take into account when scaling the classic paradigms to quantum level is *reversibility*, because quantum programs are limited to so-called unitary transformations (Yanofsky and Mannucci, 2008). Therefore, this research intersects with that on reversible computation.

It must be mentioned that a similar extension of the MPC paradigm to *probabilistic programming* has shown to be viable in practice (Murta and Oliveira, 2015), although in a very different context: that of reasoning about program reliability in presence of faulty hardware. The laws of such an approach require *typed* linear algebra rather than just relational algebra[3]

---

[2]MPC stands for "Mathematics of Program Construction", a branch of applied mathematics proposed by Backhouse (2004) for program calculation based on logic and relational algebra.

[3]This has been referred to by the acronym LAoP ("linear algebra of programming") (Oliveira, 2012).

and apply to probabilistic functions (Markov chains) which, on the experimental side, require programming over the *distribution monad* as implemented by Erwig and Kollmannsberger (2006). The recursive programming construction at target was the so-called *catamorphism* (Bird and de Moor, 1997). The idea now is to generalize from probabilistic catamorphisms to unitary transformations over a vector space monad implementing finite-dimensional Hilbert spaces (Coecke, 2011). The corresponding extension of the catamorphism concept, which will be studied and implemented in this dissertation, will be termed "quantamorphism" — a form of *recursive* classic control of *quantum* data (Oliveira, 2018).

A half-way concept between classical functions and unitary transformations is that of a reversible function, also known as isomorphism or bijection. The dissertation will contribute to the current investment in reversible computing by extending a technique known as *complementation* (Matsuda et al., 2007) to recursive programs. The background of all this research is also enhanced by studies in quantum functional programming (QFP) (Altenkirch and Grattage, 2005; Green et al., 2013a) and extensive research in categorial quantum physics (Coecke, 2011).

1.5 STRUCTURE OF THE DISSERTATION

To make the presentation self-contained, the next chapter introduces the basic mathematical concepts essential to understanding this dissertation and some background on how these apply to computer science.

Chapter 3 introduces quantum computing. It aims at exposing the reader to the new possibilities offered by the quantum paradigm.

Chapter 4 develops the quantamorphism concept, an extension of classical catamorphisms to the quantum context.

The implementation of quantamorphisms can be found in chapter 5. Some examples are given to illustrate how to derive a reversible program from such specifications, upon which quantum circuits are generated. Such circuits or their decompositions are later implemented in IBM Q Experience devices. Details of the outcome of each experiment and of the tooling involved are given in the appendices.

Finally, chapter 6 draws conclusions and points to a number of questions raised by the work developed in this master's project, i.e. topics that are left as suggestions for future work.

BACKGROUND

The work reported in this dissertation relies on a mild application of *category theory* (Awodey, 2010) to describing computations in general, and quantum computations in particular. The standard text on category theory was written by MacLane (1971) more than forty years ago. Other textbooks have meanwhile appeared in which such a generic theory is presented bearing computing in mind, notably written by Freyd and Scedrov (1990), by Bird and de Moor (1997) and by Coecke (2011).

This chapter gives an overview of basic notions of category theory and of other techniques rooted on such foundations, namely MPC and correct-by-construction programming techniques, which the reader should be acquainted with, in order to follow the chapters that follow.

2.1 CATEGORIES

Before providing any definitions from category theory, it is essential to understand why all of it is so important.

The abstract concept of a category captures a most important aspect of *any* system or theory, be it purely mathematical, physical, computational or otherwise: the fact that *compositionality* is at the heart of any successful piece of science or theory. This is very much in line with current concerns in the computing sector. More and more complex systems are on demand by the IT sector which cannot be designed and implemented from scratch — they have to arise by composing existing systems. In other words, the laws behind *composing* existing systems have become more important than the laws behind building the individual systems themselves.

In this context, is no wonder that the main operation in a category is called *composition*. Such an operation has to be associative, in order to cater for arbitrarily complex system construction. As the reader will soon verify, matrix multiplication, function / quantum-gate composition, various kinds of orderings, etc. shall be instances of the generic concept of composition.
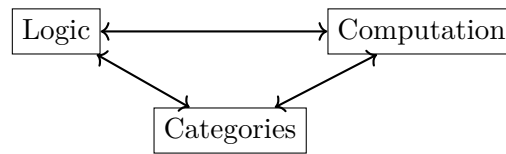
Figure 2.1.: Curry-Howard-Lambek.

Categories give rise to interesting and powerful constructs that find application in many fields, including mathematics, computer science and physics (Coecke, 2011). Such constructs offer an "internal", abstract language that can be used to design generic systems that behave alike. Quite often, expanding an existing formula written in such a language towards a more complex domain (such as e.g. quantum physics) is achievable not by reworking the formula but rather by keeping it unchanged and merely changing the category in which it is to be interpreted.[1]

The connection between categories, logic and computation was first developed by Lambek and is included in the so-called Curry–Howard–Lambek correspondence represented in figure 2.1 (Abramsky and Tzevelekos, 2011).

CATEGORIES AND DIAGRAMS   The definition of a category follows:

Definition 2.1. Category — A category C is a mathematical system made of:[2]

1. A class of objects, $obj(C)$.

2. For each pair of objects A, B, there is a class of arrows, $C(A, B)$, from object A to object B (these are known as domain and codomain, respectively); writing $a \in C(A, B)$ can be written $a : A \to B$ or $A \xrightarrow{a} B$ with less symbols.

3. Usually referred to as *morphisms*, arrows compose provided the codomain of the source arrow is the domain of the target arrow.

4. Arrow composition is associative.

5. For every A, $C(A, A)$ includes the *identity* arrow, which is the unit of composition (Figure 2.2).

☐

To illustrate these ideas it is important to visualise them through *diagrams*. Indeed, diagrams are regarded as perhaps the most useful tool for understanding categories (MacLane, 1971; Bird and de Moor, 1997; Abramsky and Tzevelekos, 2011). A recent approach by Coecke

---

[1]Thus the lemma: *"Keep definition, change category"* (Oliveira and Miraldo, 2016).
[2]See e.g. (MacLane, 1971; Awodey, 2010; Bird and de Moor, 1997; Manes and Arbib, 1986).
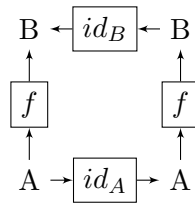
Figure 2.2.: Diagram portraying the identity function.
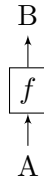
Figure 2.3.: A pair of objects A (domain) and B (codomain) in some category C linked by an arrow $f : A \to B$. As an example, f could be a function mapping an arbitrary group of students (set A) to their student numbers (set B).

and Kissinger (2017) elaborates on the diagrammatic language of categories in a way such that (it is claimed) triumphs over the actual (traditional) algebraic syntax.[3]

Figure 2.3 depicts an arrow $f : A \to B$ and the diagram of figure 2.4 illustrates arrow composition $g \cdot f$. The identity is also characterised in this diagram, returning the same value as its argument. Figure 2.5 gives the diagram of the unit law. This law shows similarity to $a + 0 = 0 + a = a$ and consequently, the identity is called the *unit* of the composition.

Morphism (arrow) composition is not necessarily sequential. Figure 2.6 shows three morphisms independently "running in parallel". So it makes sense to have a parallel composition operator $f \otimes g$ in which nothing is connected. There is an implicit associative law which gives intuition on the importance of connectivity. To understand such laws one needs the concept that follows.

Definition 2.2. Isomorphism (Abramsky and Tzevelekos, 2011) — A morphism $f : A \to B$ in a category C is said to be an *isomorphism* iff there exists a unique morphism $g : B \to A$ such that

$$g \cdot f = id_A$$
$$f \cdot g = id_B$$

Morphism g is often referred to as the *converse* of f. It follows that g is also an isomorphism, with converse f. Then objects A and B are said to be *isomorphic* (or abstractly identical) and one writes $A \cong B$ to tell this.

$\square$

---

[3]Therefore, both notations will be present in this chapter.
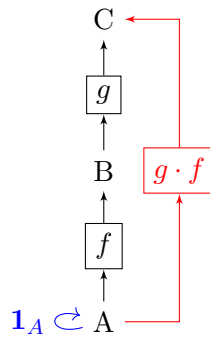
Figure 2.4.: Composition (in red) of arrows f and g and the identity (in blue) on object A. Continuing with the example of the previous figure, suppose set C represents the set of email addresses of students and that g is a function that receives the student's number and returns the corresponding email. $g \cdot f$ is the composition of g and f that to a given student's name gives its email address.
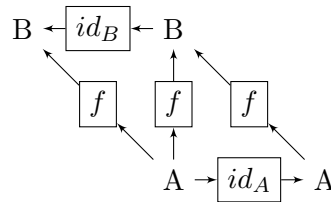
Figure 2.5.: Diagram of the unit law ($f \cdot id_A = f = id_B \cdot f$).
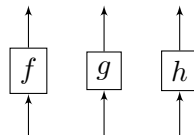
Figure 2.6.: Three parallel (independent) processes running in parallel. Such parallel processes can be anything, e.g. f may be a program playing music in the background, g may be a program displaying something on a screen, and g may be a program accepting inputs from the user. The diagram shows implicitly the associative law of parallel composition: $(f \otimes g) \otimes h = f \otimes (g \otimes h)$.
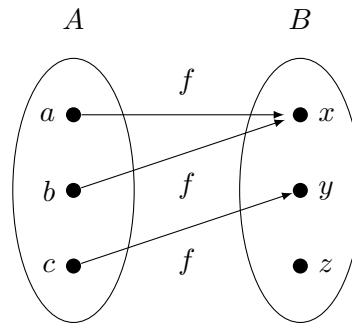
Figure 2.7.: Example of a non-injective function f.

FUNCTIONS (SETS)   So far, objects A, B and C have been left uninterpreted, as the genericity of the category concept implied. One particular instantiation is to regard them as sets. At school, one gets used to defining functions $f : A \to B$ where A and B are sets. Such sets and functions form a category which is often regarded as the "mother" of all other categories. Sequential composition is the expected: $(f \cdot g)(a) = f(g(a))$. The identity function is the (unique) function that "does nothing": for each object A there is an identity function $id_A : A \to A$ that is the unit of composition: $f \cdot id = id \cdot f = f$.

In this category, parallel composition is written $(f \times g)(a, b) = (f(a), g(b))$ (Abramsky and Tzevelekos, 2011) assuming that sets of pairs exist in the category. This indeed happens and such sets of pairs are usually known as Cartesian products:

$$A \times B = \{(a, b) \mid a \in A, b \in B\}$$

For more than two sets, the pairs of Cartesian products generalize to tuples:

$$A_1 \times ... \times A_n := \{(a_1, ...a_n) | a_i \in A_i\}$$

A function f is said to be *injective* iff $f(a) = f(b) \Leftrightarrow a = b$ and *surjective* iff for every $b \in B$ there is some $a \in A$ such that $b = f(a)$. Figure 2.7 depicts a function that is neither injective nor surjective. Functions that are injective and surjective are said to be *bijective*. The isomorphisms of this category are exactly the bijections. An example of isomorphism (bijection) that will play a role in the sequel is

$$assocr((a, b), c) = (a, (b, c))$$

This isomorphism witnesses the associative law of parallel composition,

$$(A \times B) \times C \cong A \times (B \times C)$$

which in this category is *not* an equality on objects.

RELATIONS (REL)   Like the previous category, the objects of this category are also sets. The difference therefore is on the morphisms $R : A \to B$, which now are binary relations instead of functions: $R \subseteq A \times B$ (Coecke and Kissinger, 2017). Thus composition is defined in a different (in fact, more general) way:

$$(a,c) \in (S \cdot R) \iff \exists_{b \in B}(a,b) \in R \wedge (b,c) \in S \tag{2.1}$$

That is, the sequential composition of two relations is a set of pairs such that there is an element b connecting the relations. (Figure 2.8.) Note that every function (f) can be regarded as a special case of a relation (R):

$$(a,b) \in R \iff b = f(a)$$

In general, the symbol f denotes *both* the function and the corresponding relation (often referred to as the *graph* of the function).

This means that relations may happen to be functions, but in general relations add to functions a *non-deterministic* feature (Abramsky and Tzevelekos, 2011). On the other hand, this non-deterministic characteristic allows for the existence of converse relations in any case, denoted by R°:

$$(a,b) \in R \iff (b,a) \in R^\circ$$

For instance, $f^\circ = \{(x,a),(x,b),(y,c)\}$ in Figure 2.7.

In summary, **Set** $\subseteq$ **Rel** (Abramsky et al., 2017) and the composition of functions portrayed in Figure 2.4 is a specific case of relation composition (Coecke and Kissinger, 2017). The identity for relations is the identity function.

It is a significant remark that relations may be more interesting than functions. In fact, relations admit a matrix calculus like linear maps, which is pertinent for computation and quantum processing (Coecke and Kissinger, 2017). Furthermore, the category of relations has proved to be a solid basis for the mathematics of program construction in classic and quantum programming (Zeng, 2015).

MATRICES (MAT$_K$)   This category is defined over a field K that may be for instance that of the real or complex numbers. The objects are natural numbers (n). Each morphism $n \xrightarrow{M} m$ in the category describes a matrix with n columns and m rows. If the input n or the output m is 1 then M is called a column vector or row vector, respectively (Coecke and Kissinger, 2017).

Composition in this category corresponds to matrix-matrix multiplication (MMM). Furthermore, identity morphisms are the diagonal matrices ($n \times n$) (Abramsky and Tzevelekos, 2011).
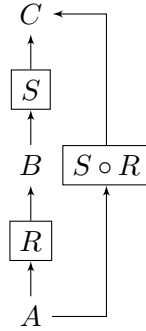
Figure 2.8.: Composition $S \circ R$.

Categories of matrices have been used in probabilistic programming. In this context, Murta and Oliveira (2015) study programs with probabilistic behaviour caused by faulty hardware. Like relations, matrices are commonly associated with linear maps[4]. However, these are more general (Coecke and Kissinger, 2017).

INITIAL AND TERMINAL OBJECTS   Some objects in a category may have special properties. Such is the case of so-called *initial* and *terminal* objects.

Definition 2.3. Initial and Terminal Objects (Abramsky and Tzevelekos, 2011) — Object 0 is said to be an initial object of category C if there is a unique arrow from 0 to every other object A, which is written as $?_A : 0 \to A$. Object 1 is said to be a terminal object of category C if there is a unique arrow from every other object A to 1, which is written as $!_A : A \to 1$
□

The initial and terminal objects are represented by the diagrams of figure 2.9. These concepts are dual, meaning that an initial object in a category C represents a terminal object in the opposite category C^{op}. The opposite category C^{op} is defined by simply inverting the direction of the arrows in C: $C^{op}(A, B) := C(B, A)$.

The initial object of category **Set** is the empty set while the terminal object is any singleton set (a set with only one element) (Abramsky and Tzevelekos, 2011). In the category **Rel** of relations, the empty set is both the initial and terminal object (Bird and de Moor, 1997). In **Mat$_k$** the natural number 0 is also initial and terminal.

Moreover, initial and terminal objects can be seen as the limit of the datatype spectrum (see figure 2.10) (Oliveira, 2008).

PRODUCTS AND COPRODUCTS   Composition has been presented as the standard way to combine morphisms in a category. However, some categories offer other ways to connect morphisms, known as products and coproducts. These constructs perform the pairing of two

---

[4]This category is also used to describe states and effects, as described in the following chapter.
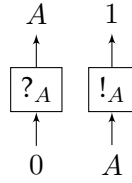
Figure 2.9.: Diagrams of the initial object and terminal object, respectively (Bird and de Moor, 1997).
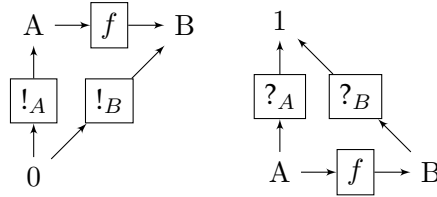
Figure 2.10.: Initial datatype function fusion (left) or final datatype (right) fusion.

arrows, with the same domain (in the case of products) or the same codomain (in the case of coproducts).

As already seen, in the **Set** category, the Cartesian product of sets A and B is the set of ordered pairs,

$$A \times B = \{(a,b) | a \in A \wedge b \in B\}$$

wherefrom each element can be retrieved:

$$\pi_1(a,b) = a$$
$$\pi_2(a,b) = b$$

Functions π_1 and π_2 are called *projections*:

$$A \xleftarrow{\pi_1} A \times B \xrightarrow{\pi_2} B$$

In general:

Definition 2.4. Product (Awodey, 2010) — In a category C, the product of object A and object B is the object $A \times B$ with the morphisms $\pi_1 : A \times B \to A$ and $\pi_2 : A \times B \to B$ satisfying the following universal property:

To any given object X in $f : X \to A$ and $g : X \to B$, there is a unique arrow from X to $A \times B$, $f \triangledown g$ such that the diagram in figure 2.11 commutes.

In symbols:

$$k = f \triangledown g \Leftrightarrow \begin{cases} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{cases} \tag{2.2}$$
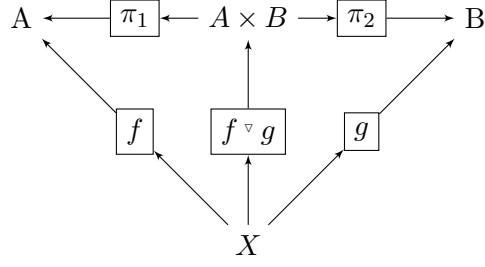
Figure 2.11.: Product diagram illustrates the cancellation properties $\pi_1 \cdot (f \triangledown g) = f$ and $\pi_2 \cdot (f \triangledown g) = g$.

$f \triangledown g$ is read "f split g" or the pairing of f with g.

□

Therefore, each morphism $k : X \to A \times B$ splits uniquely into a pair of morphisms, of types $X \to A$ and $X \to B$, expressing the homset isomorphism $(X \to A \times B) \cong (X \to A) \times (X \to B)$. This captures the well-known bijection, in sets, between pair-valued functions and pairs of functions.

$\mathbf{Mat_k}$ is an example of category with products, in which the universal property is written as

$$X = \left[\frac{M}{N} \right] = \begin{cases} \pi_1 \cdot X = M \\ \pi_2 \cdot X = N \end{cases}$$

(Oliveira, 2017). The notation $\left[\dfrac{M}{N} \right]$ represents the vertical stacking of the matrices M and N, with the same number of columns, say p. Matrices π_1 and π_2 are the fragments of the identity matrix such that $\left[\dfrac{\pi_1}{\pi_2} \right] = id$.

Definition 2.5. Coproducts(Abramsky and Tzevelekos, 2011) — In a category C, the coproduct of the A and B is the object $A + B$ and the pair of morphisms i_1 e i_2, called injections,

$$A \xrightarrow{i_1} A + B \xleftarrow{i_2} B$$

such that, for any X and

$$A \xrightarrow{f} X \xleftarrow{g} B$$

there exists a unique morphism $[f, g] : A + B \to X$ that commutes the diagram of figure 2.12. Thus the universal property:

$$k = [f, g] \Leftrightarrow \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases} \tag{2.3}$$
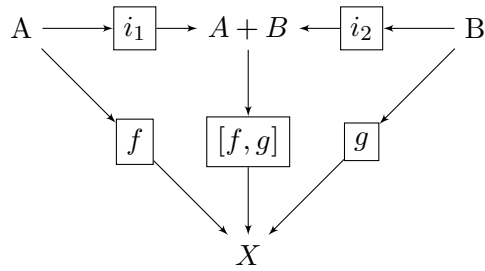
Figure 2.12.: Coproduct. This diagram illustrates the cancellation properties $[f,g] \cdot i_1 = f$ and $[f,g] \cdot i_2 = g$.

□

The morphism $[f,g]$ is sometimes termed the *copairing* of f and g. Like initial and terminal objects, the product and coproducts are unique up to isomorphism, recall definition 2.2.

2.2 FUNCTORS

Because the type of the objects is not the only thing to take into account, the lemma of category theory is:

The morphisms are what really matters.

Therefore, it is possible to have homomorphisms between categories (Awodey, 2010).

Definition 2.6. Functor (Abramsky and Tzevelekos, 2011) — A functor $\mathsf{F} : C \to D$ between categories C and D is defined by two mappings:

1. object-map: this assigns an object $\mathsf{F}\,A$ of target category D to every object A of source category C.

2. arrow-map: this assigns an arrow $\mathsf{F}\,f : \mathsf{F}\,A \to \mathsf{F}\,B$ of category D to every arrow $f : A \to B$, in such a way that identity arrow and composition are preserved:

$$\mathsf{F}\,(id_A) = id_{\mathsf{F}\,A} \tag{2.4}$$

$$\mathsf{F}\,(g \cdot f) = \mathsf{F}\,g \cdot \mathsf{F}\,f \tag{2.5}$$

Since the two component mappings of functor F preserve domain and codomain of every pair of objects A and B of C, one has:

$$\mathsf{F}_{A,B} : C(A,B) \to D(\mathsf{F}\,A, \mathsf{F}\,B)$$

□
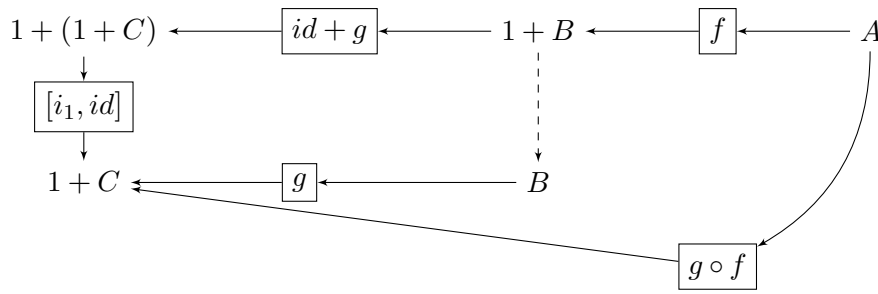
Figure 2.13.: Diagram with composition $g \cdot f$ and extension of g.
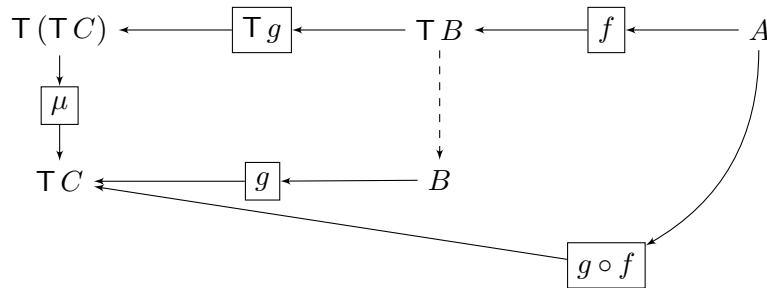
Figure 2.14.: Monadic composition — general case.

As any other homomorphism, functors can be composed — $(\mathsf{F} \circ \mathsf{G})f = \mathsf{F}(\mathsf{G}\,f)$, and for any category C there exists an identity functor: $id : C \to C$ (Bird and de Moor, 1997).

2.3 MONADS

So far we have handled morphism that map from objects A to B in the same category, and morphisms that map objects from one category to another (functors). Besides these, in computing one also has to handle situations where the output of one morphism (the source) is more elaborate than the input of another (the consumer) in a sequential composition. For instance, when testing the head of a list one needs to handle the exceptional case raised by the empty list. An abstract way to represent exceptions of this kind is

$$f : 1 + A \leftarrow B$$

where the unique inhabitant of object 1 signals the exception.

The sequential composition of functions with type $A \to 1 + B$ has a problem, cf. eg. $f : 1 + B \leftarrow A$ and $g : 1 + C \leftarrow B$. The composition $g \cdot f$ calls for the *extension* of g. This is depicted in figure 2.13.

To achieve generality, define functor $\mathsf{T}\,X = 1 + X$ and redraw the diagram using T, as displayed in figure 2.14, where the composition is given by:

$$g \circ f = \mu \cdot \mathsf{T}\,g \cdot f$$

$$A \leftarrow \boxed{f} \leftarrow \mathsf{F}\,A$$
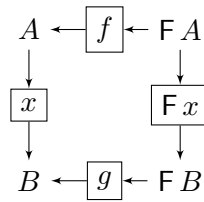
Figure 2.15.: Program structure.

Functor T is a special case of a *monad*, and this monadic composition is called *Kleisli composition*. A *monadic arrow* $f : \mathsf{T}\,Y \leftarrow X$ represents a function with an output of type Y wrapped by the (computational) "effect" embodied in T.

The morphism $\mu \cdot \mathsf{T}\,g$ is referred as the (monadic) extension of g. The important role is played by $\mu : \mathsf{T}\,X \leftarrow \mathsf{T}^2 X$, a suitable polymorphic function called *multiplication* of monad T. Arrow $u : \mathsf{T}\,X \leftarrow X$ is the monad's *unit*, which injects data inside the effect T.

Monads play a major role in today's computer programming. Kleisli arrows of type $A \xrightarrow{f} \mathsf{T}\,B$ correspond to particular morphisms in other categories, depending on T. In particular, binary relations, Markov chains and matrices, in general, can be regarded as Kleisli arrows for suitable monads T. This correspondence, labelled *Kleisli* correspondence, will be very helpful in the sequel.

2.4 RECURSION

One of the main advantages of ordinary mathematics is the ability to formulate problems by equations and obtain solutions for the problems by solving the corresponding equations. For instance, the problem "*is there a natural number that is the successor of its half?*" can be captured by the equation $x = 1 + \frac{x}{2}$. It has solution $x = 2$ since $2 = 1 + \frac{2}{2}$.

In any category-based approach, morphisms play the major role. So one may wonder what the categorial analogous to some equation $x = f\,x$ would be. Since x is a morphism, f has to be a functor, say F in figure 2.15. To express the equation one needs to relate A, the source of x, with $\mathsf{F}\,A$, the source of $\mathsf{F}\,x$, and similarly for B. Let $f : \mathsf{F}\,A \to A$ and $g : \mathsf{F}\,B \to B$ be morphisms that establish such relationships. Then the category counterpart of equation $x = f\,x$ will be

$$x \cdot f = g \cdot \mathsf{F}\,x \tag{2.6}$$

Given a functor F, a morphism with type $X \leftarrow \mathsf{F}\,X$ is called an F-algebra. So f and g above are F-algebras and the diagram shows that the role of x is to transform F-algebra f into F-algebra g. The branch $\mathsf{F}\,x$ expresses *recursion* in the sense that x has to run "inside" the structure of F to carry out such a transformation.

As often happens in mathematics in general, equation (2.6) may happen to have no solution at all. There are, however, situations where solutions are guaranteed. This is the case when algebra f is an isomorphism. Take for instance $\mathsf{F}\,X = 1 + X$ and $A = \mathbb{N}_0$, the natural numbers. Then define $f : \mathsf{F}\,\mathbb{N}_0 \to \mathbb{N}_0$ by

$$f = [zero, succ]$$

where $zero\_ = 0$ and $succ\,n = n + 1$. Algebra f is an isomorphism (bijection) because (a) every natural number either is 0 or the successor of another natural number (f is surjective); (b) 0 is never the successor of another natural number (f is injective). Algebra f can be recognized as the well-known Peano algebra supporting the axiomatic definition of the natural numbers.

Let α denote one such isomorphism f regarded as standard, say $\alpha = [zero, succ]$, and let this be fixed. Then the standard solutions to equation (2.6) are parametric on algebra g. It turns out that, for each g, there is a unique solution to the equation, which is denoted by $(\!|g|\!)$ and called the *catamorphism* of algebra g (figure 2.16):

$$(\!|g|\!) \cdot \alpha = g \cdot \mathsf{F}\,(\!|g|\!)$$

The uniqueness of this solution is captured by the universal property:

$$x = (\!|g|\!) \quad \Leftrightarrow \quad x \cdot \alpha = g \cdot \mathsf{F}\,x \tag{2.7}$$

from which the equality just above is obtained by letting $x = (\!|g|\!)$ and simplifying. This property is known as the *cata-cancellation* law.

The categorial model for recursion explained just above is very powerful and generic (Bird and de Moor, 1997). Catamorphisms are just one particular class of solutions to categorial recursive equations such as (2.6). In general, any x that is a solution to (2.6) is said to be a *homomorphism* from algebra f to algebra g, in fact a morphism $g \xleftarrow{\ x\ } f$ in the category whose objects are F-algebras and whose morphisms are F-homomorphisms. Some such algebras α may happen to be *initial* objects in such category, meaning that there is a unique homomorphism from α to any given algebra g. Such unique homomorphism is written $(\!|g|\!)$. As an immediate consequence of such uniqueness, one has:

$$(\!|\alpha|\!) = id \tag{2.8}$$

since id is an omnipresent morphism. Therefore, the "genetic material" of the identity catamorphism is the initial algebra α. This corresponds to the *reflection* property of catamorphisms. Because α is an isomorphism, law (2.6) can be written with its inverse (α°):

$$(\!|g|\!) = g \cdot \mathsf{F}\,(\!|g|\!) \cdot \alpha^\circ$$
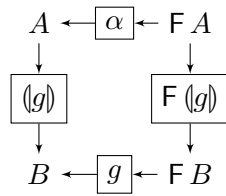
Figure 2.16.: Catamorphisms.
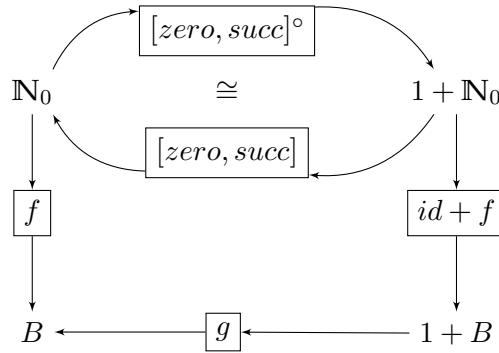
Figure 2.17.: Catamorphism representing the Peano Algebra.

Returning to the Peano algebra example, these laws result in the catamorphism of figure 2.17.

In this example the catamorphism is:

$$f = (\!|g|\!) \Leftrightarrow f \cdot [zero, succ] = g \cdot (id + f)$$

The morphism g is a copairing $[g_1, g_2]$. When writing a program following this diagram, $g_1 : 1 \to B$ corresponds to the program instruction to stop and $g_2 : B \to B$ is the program's instruction to continue iterating.

2.5 ALLEGORIES

RELATIONS AND ALLEGORIES   Relations increase the expressive power of functions, in the sense that they can be used to describe nondeterministic computations and define problems in terms of converses of other problems. While functions can be compared for equality only, relations can be tested for inclusion. This means that the homsets of the **Rel** category are more than just *sets*: they are *partially ordered sets*. Categories whose homsets are enriched by some algebraic structure are called *enriched categories*. One particular branch of enriched categories abstracts the ordered features of **Rel**. Such categories have become known as *allegories.*

Definition 2.7. Allegory (Freyd and Scedrov, 1990) — An allegory A is a category enriched with extra operators inspired by the category **Rel**. While in a standard category the homsets

are sets, in an allegory the homsets are partially ordered sets. This means that besides target, source, composition, and identity, in allegory morphisms can be compared by a partial order $\subseteq$. Moreover, allegories also allow for converse $(\_)^\circ$ morphisms and morphism intersection $(\cap)$ and union $(\cup)$. $\square$

PARTIAL ORDER   Partial orders in allegories compare homomorphisms with the same source and target. In the **Rel** allegory, where a relation $R : A \leftarrow B$ is a subset $R \in A \times B$, the concept of inclusion in relations is defined by:

$$R \subseteq S \Leftrightarrow (\forall y, x : yRx \Rightarrow ySx) \tag{2.9}$$

INTERSECTION   In the **Rel** allegory, for every homomorphism pair $R, S : A \leftarrow B$, their intersection, $R \cap S : A \leftarrow B$, is defined by:

$$X \subseteq (R \cap S) \Leftrightarrow (X \subseteq R) \wedge (X \subseteq S)$$

Similarly,

$$(R \cup S) \subseteq X \Leftrightarrow (R \subseteq X) \wedge (S \subseteq X)$$

CONVERSE   In an allegory, the converse of a morphism $R : A \rightarrow B$ is the morphism $R^\circ : A \rightarrow B$ such that the following properties hold (Bird and de Moor, 1997):

1. $(R^\circ)^\circ = R$

2. $R \subseteq S \Leftrightarrow R^\circ \subseteq S^\circ$

3. $(R \cdot S)^\circ = S^\circ \cdot R^\circ$

In consequence:

$$id^\circ = id \tag{2.10}$$

In the **Rel** allegory, converse is defined by:

$$yRx \Leftrightarrow xR^\circ y \tag{2.11}$$

So converse in **Rel** implements the passive voice of natural language. A tangible example could be the converse of *John loves Mary*, which is *Mary loves° John*.

BINARY RELATION TAXONOMY   Similar to functions, relations may be divided into different classes. For instance, a *total function* [5] and *partial functions* [6] can be transposed to the relational algebra by the transpose operator Λ and Γ, respectively, which are defined by the universal properties:[7]

$$f = \Lambda R \Leftrightarrow (bRa \Leftrightarrow b \in fa) \tag{2.12}$$

$$f = \Gamma R \Leftrightarrow (bRa \Leftrightarrow (fa = \mathbf{Just}\ b)) \tag{2.13}$$

In order to avoid confusion with partial order and total orders, relation algebra uses *entire* for *total* and *simple* for *partial* relations.

Recall the identity morphisms *id*, from which some terminology arises:

Definition 2.8. An (endo)relation $R : A \to A$ is reflexive iff $id_A \subseteq R$ and coreflexive iff $R \subseteq id_A$ $\square$

The transitive concept defines that R is transitive iff $R \cdot R \subseteq R$ hold. Combining these notions results in preorders: reflexive and transitive relations. Partial orders are anti-symmetric preorders, i.e. $R \cap R^\circ \subseteq id$ holds.

To understand the whole taxonomy of binary relations given in figure 2.18 it is important to define the kernel and image of a relation.

Definition 2.9. The kernel of a relation is defined by

$$\ker R = R^\circ \cdot R$$

and the image is kernel's dual, that is:

$$\mathsf{img}\ R = \ker(R^\circ) \text{ or } \mathsf{img}\ R = R \cdot R^\circ$$

$\square$

These two notions lead to some terminology: a relation is *entire* if its kernel is reflexive, *simple* iff its image is coreflexive. By duality, a relation is *surjective* iff R° is entire and *injective* iff R° is simple. This terminology is summarized in table 2.1.

In this context *function* describes a relation simple and entire. On other hand, f is a *bijection* iff $\mathsf{img}\ f = id \wedge id = \ker f$.

It is also useful to mention that these relations may be seen as boolean matrices. In this case a *function* has exactly one *1* in every column, and a *bijection* has one *1* in every column

---

[5]By total is meant a function where each value of the domain is assigned to a value (Abramsky and Tzevelekos, 2011).

[6]By partial functions is meant functions which are undefined for some arguments (Oliveira and Rodrigues, 2004) as the head function example seen in section 2.3.

[7]Details in (Oliveira and Rodrigues, 2004).
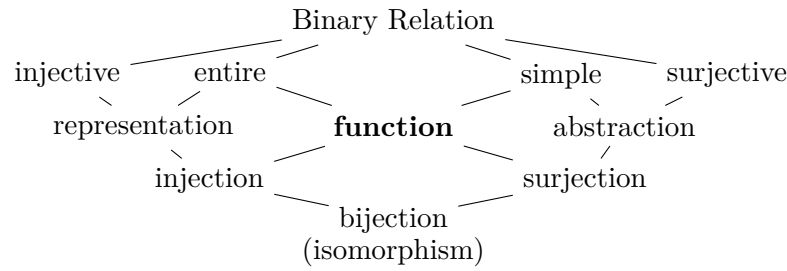
Figure 2.18.: Binary Relations taxonomy.

	Reflexive	Coreflexive
ker R	entire R	injective R
img R	surjective R	simple R

Table 2.1.: Terminology summary.

and in every row. Functions are a special case of relations that satisfy the so-called shunting rules:

$$f \cdot R \subseteq S \Leftrightarrow R \subseteq f^\circ \cdot S \tag{2.14}$$

$$R \cdot f^\circ \subseteq S \Leftrightarrow R \subseteq S \cdot f \tag{2.15}$$

from which the following equivalences hold:

$$f \subseteq g \Leftrightarrow f = g \Leftrightarrow g \subseteq f \tag{2.16}$$

2.6 SUMMARY

This chapter presents a minimal collection of mathematical notions that provide foundations for the rest of the dissertation. The main notion is that of a *category*, which provides an abstract way to think of and reason about programs. Diagrams play a central role in categorial reasoning. These notions were accompanied by examples that help to realise their power and to observe the freedom of dealing with different types of data.

Another concept introduced in this chapter is that of a *functor*. Functors connect categories among themselves. This concept is followed by that of a monad, which enables programs that manage inputs and outputs with different levels of complexity.

The chapter also introduced catamorphisms, which are a categorial device for describing recursion. The final section analysed allegories, a brand of enriched categories that generalize the category **Rel** of relations.

The next chapter addresses quantum computing. Because this model of computation requires further abstraction and of a different nature, such foundations are not part of this chapter.

3

QUANTUM COMPUTING

This chapter introduces the quantum computing paradigm on the basis of the terminology already presented in the previous chapter. Having in mind the basic laws of the quantum computing will allow the reader to easily debunk some of the widespread myths about quantum computing, while understanding the constraints of quantum programming.

It is also important to settle some conventions, notations and concepts of quantum computing. Among these, one of the major postulates is the request for reversibility, a notion already introduced in the background provided by the previous chapter.

3.1 OVERVIEW OF QUANTUM THEORY

In the beginning of the twentieth-century physicists studied the dichotomy between matter and wave, assuming they were two different concepts that would not overlap. However, throughout the century considerable evidence piled up exposing situations within this line of thought, e.g. the photoelectric effect, black-body radiation and the Compton effect.[1]

By that time, it was vital to find a new theory in which light and matter presented both light-like behaviour and matter-like behaviour. Setting matter and waves apart wasn't acceptable at the quantum level (Yanofsky and Mannucci, 2008).

QUANTUM STATE   Yanofsky and Mannucci (2008) start the study of quantum states by considering an arbitrary subatomic particle, that cannot be detected on a particular position but rather within the range of some set of positions $\{x_1, x_2, ..., x_n\}$, all equally spaced by some δx. For simplicity of presentation, such a set of possible states can be regarded as finite.[2] This means that one can only measure the *probability* of the particle being found in a particular such position.

---

[1] For relatively recent accounts of these advances see e.g. the textbooks by Yanofsky and Mannucci (2008) and Gasiorowicz (2003).

[2] In reality, this set is infinite and so $\delta x \rightarrow 0$.
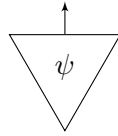
Figure 3.1.: Diagram representing a state.

The next step is to describe such a probabilistic state by an n-dimensional (column) vector. A particle with 100% probability of being in position x_i is described by the Dirac so-called *"ket"* notation $|x_i\rangle$, and so to each n state there is a column vector:

$$|x_1\rangle \rightarrow \begin{bmatrix} 1 \\ 0 \\ ... \\ 0 \end{bmatrix} \quad |x_2\rangle \rightarrow \begin{bmatrix} 0 \\ 1 \\ ... \\ 0 \end{bmatrix} \quad (...) \quad |x_n\rangle \rightarrow \begin{bmatrix} 0 \\ 0 \\ ... \\ 1 \end{bmatrix}$$

These are eigenvectors that constitute the canonical base $\mathbb{C}^n$ of the system.

Note that this example handles a quantum particle and such particles do not only have the probabilistic behaviour but also a wave-like behaviour. This duality could be represented by real numbers but is not efficient. A better option is to write the states with complex numbers, where the modulus of the complex number calculates the probabilities and its argument expresses the phase difference.

While this mathematical system would be enough to describe a classic system, this is where these systems diverge from quantum systems. In a quantum system, *any* vector in $\mathbb{C}^n$ is a valid physical state of the particle. Therefore, let $|\psi\rangle$ be an arbitrary state with the linear combination of the eigenvectors $|x_1\rangle, |x_2\rangle, ..., |x_n\rangle$ with their complex amplitude, $\{c_1, c_2, ..., c_n\}$.

$$|\psi\rangle = c_1|x_1\rangle + c_2|x_2\rangle + ... + c_n|x_n\rangle$$

Assuming such a standard basis, such an arbitrary state may be represented by the vector:

$$|\psi\rangle \rightarrow \begin{bmatrix} c_1 \\ c_2 \\ ... \\ c_n \end{bmatrix}$$

This is called a *superposition* of eigenstates (Gasiorowicz, 2003) or generally called superposition of states (Yanofsky and Mannucci, 2008).

In the example of the position of a particle, this superposition describes the many places a particle can be at the same time. The complex amplitudes, c_i, represent the "degree" in which the particle is in which state.
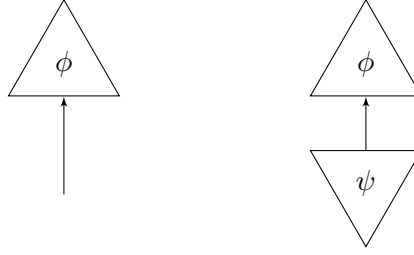
Figure 3.2.: Diagram representing an effect (left) and a bracket (right).

It is easy to imagine these vectors framed in a category of matrices and consequently described by diagrams (Coecke and Kissinger, 2017), as for instance in figure 3.1. This shows not only a *ket* but also its dual process, called *effect* by Coecke and Kissinger (2017) and written $\langle \phi |$ in standard quantum notation — a "*bra*". A *braket*, i.e. the merging of the two, $\langle \phi | \psi \rangle$, corresponds to the inner product of two vectors (see figure 3.2).

An inner product space is a vector space equipped with inner products (Nielsen and Chuang, 2011). A *Hilbert Space* is an inner product space of finite-dimensional complex vector spaces. In terms of process theory, Hilbert spaces are the objects of the category of linear maps (Coecke and Kissinger, 2017).

OBSERVABLE AND MEASUREMENT   It is possible to take the probability of finding the particle in a state x_i after observing it:

$$p(x_i) = \frac{|c_i|^2}{||\psi\rangle|^2} = \frac{|c_i|^2}{\sum_j |c_j|^2} \quad , \quad p(x_i) \in [0, 1]$$

A so-called *observable* gives the possibility to ask the state of a system. This question will admit a set of answers and corresponding probabilities. On the other hand, measuring corresponds to running an observable. In this case, not only the question is made to the system but also the system gives a definite answer. Both action, observe and measure, follow a set of axioms (Yanofsky and Mannucci, 2008):

Axiom 3.1. *A physical observable is analogue to a Hermitian operator:*[3]
□

This axiom forces the observable to be a linear operator. As a result of this, the observable Ω applied to a state $|\psi\rangle$ modifies the state to $\Omega|\psi\rangle$. Furthermore, the eigenvalues of a Hermitian operator are all real, which leads to:

Axiom 3.2. *The only acceptable values to be observed and that can be a result of measuring a state are eigenvalues of the Hermitian operator associated with a physical observable. The*

---

[3]A square matrix is called Hermitian if is self-adjoint: $A^\dagger = A$ (Yanofsky and Mannucci, 2008), where $A^\dagger$ is the conjugate transpose $\overline{A}^T$.

eigenvectors of the Hermitian operator also form the basis for the state space.
□

With this, it is reasonable to conclude that the set of values that can answer the observable question are provided by the eigenvalues of the observable.

In terms a measure there is a clear contrast between classical and quantum systems. In classic systems, a measurement does not interfere with what is being measured and the result is predictable. As seen above, the states of a quantum system are modified by measuring and this is a non-deterministic process of which it's only possible to predict a probability.

Finally, one last axiom:

Axiom 3.3. *Let Ω be an observable and ψ be an arbitrary state. If the measurement result of Ω is an eigenvalue λ, then the state after measuring will always be the eigenvector corresponding to λ.*
□

DYNAMICS   All systems considered above are not time-dependent. Concerning quantum dynamics the axiom to keep in mind is:

Axiom 3.4. *(Yanofsky and Mannucci, 2008) — The evolution of a quantum system (that is not an observable nor a measurement) is given by unitary operators or transformations.*
□

Unitary transformations[4] are a group of transformation that are preserved by converse and composition, i.e. the product of two unitary matrices is also unitary and the inverse of the unitary matrix is still a unitary matrix.

QUANTUM SYSTEM ASSEMBLY   Systems always have more than one particle. Putting particles together is achieved via *tensor products*:

Axiom 3.5. *(Yanofsky and Mannucci, 2008) Let $\mathbb{Q}$ and $\mathbb{Q}'$ be two quantum systems, characterised by the vectors $\mathbb{V}$ and $\mathbb{V}'$, respectively. When systems $\mathbb{Q}$ and $\mathbb{Q}'$ are merged the resulting system is described by the tensor product $\mathbb{V} \otimes \mathbb{V}'$.*
□

Extending the example of the position of a particle to the positions of two particles, the set $\{x_1, x_2, ..., x_n\}$ represents possible positions of the first particle and the set $\{y_1, y_2, ..., y_m\}$ represents possible positions of the second particle.

The arbitrary superposition state is now written as:

$$|\psi\rangle = c_{1,1}|x_1\rangle \otimes |y_1\rangle + ... + c_{i,j}|x_i\rangle \otimes |y_j\rangle + ... + c_{n,m}|x_n\rangle \otimes |y_m\rangle$$

---

[4]The definition of the unitary matrix only appears when in applied in chapter 4 (i.e. definition 4.2).

Notation	Description	Example
z*	complex conjugate of the complex number z	(1+i)* = (1-i)
$\lvert\psi\rangle$	ket	
$\langle\phi\rvert$	bra	
$\langle\psi\vert\phi\rangle$	inner product	
$\lvert\psi\rangle \otimes \lvert\phi\rangle$ or $\lvert\psi\rangle\,\lvert\phi\rangle$	tensor product	$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} =$ $= \begin{bmatrix} a\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & b\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \\ c\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & d\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \end{bmatrix}$
A^*	Complex conjugate of the A matrix	
A^T	Transpose of the A matrix	
$A^\dagger$	Hermitian or adjoint of the A matrix	
$\langle\phi\vert A\vert\psi\rangle$	inner product of ϕ amd $A\lvert\psi\rangle$ or inner product of $A^\dagger\langle\phi\rvert$ and $\lvert\psi\rangle$	

Table 3.1.: Useful information regarding quantum mechanics (Nielsen and Chuang, 2011; Selinger, 2004).

This can be scaled to n systems:

$$\mathbb{V}_1 \otimes \mathbb{V}_2 \otimes ... \otimes \mathbb{V}_n$$

3.2 BIT VS QUBIT

The previous section gives good intuitions on the differences between the classic and the quantum computer systems. Standard computers are based on a minimal, indivisible unit of information, called the *bit*.

Definition 3.1. (Coecke and Kissinger, 2017; Yanofsky and Mannucci, 2008) A *bit* is the unit of information, characterized by:

1. it accepts two states, 0 and 1;

2. it can be subject to any function;

3. it can be read without any obstruction and the act of reading does not produce any change to the state.

□
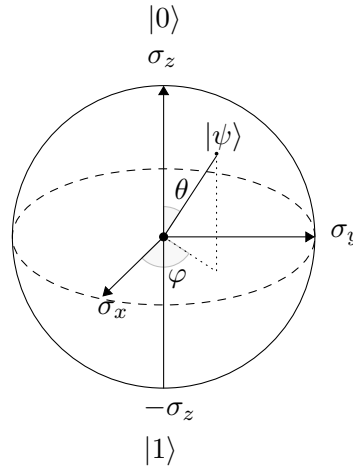
Figure 3.3.: Bloch Sphere representing a qubit (Nielsen and Chuang, 2011).

Definition 3.2. (Nielsen and Chuang, 2011; Coecke and Kissinger, 2017) The quantum bit, or *qubit*, is characterized by:

1. it accepts every state in the Bloch sphere (see figure 3.3);

2. it can only be subject to rotation on the sphere, i.e. an operator applied to a qubit must be a unitary transformation;

3. to measure a quantum bit requires an invasive process that destroys the previous state.

□

The traditional way to represent a bit holding 0 in a quantum system is the ket notation $|0\rangle$, which is characterised by the vector $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$; so the bit holding 1 is represented by $|1\rangle$ and characterised by the vector $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$. A qubit is a superposition of such two states:

$$c_1|0\rangle + c_2|1\rangle \tag{3.1}$$

Pairing qubits may result in states that are either *separable* or *entangled*. Tensor product always produces separable qubit pairs. For instance, if qubit q_1 in the state $c_1|0\rangle + c_2|1\rangle$ and qubit q_2 in the state $c_1'|0\rangle + c_2'|1\rangle$ are paired up, then the combined state is:

$$q_1 \otimes q_2 = c_1c_1'|00\rangle + c_1c_2'|01\rangle + c_2c_1'|10\rangle + c_2c_2'|11\rangle$$

However, a qubit-pair may happen to be in an *entangled* state (Selinger, 2004), meaning that it cannot be written as above. Entanglement is one of the most interesting aspects of quantum systems, and one that cannot be found in classical systems. The representation of the two

(a) Separable states.

(b) Entangled states - cup (the dual of cup is referred as cap.).
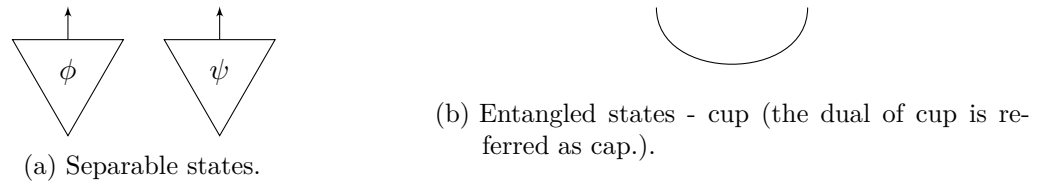
Figure 3.4.: Combination of states.

situations, separable and entangled, in the notation of (Coecke and Kissinger, 2017) is given in figure 3.4.

3.3 OPERATIONS

As stated in definition 3.2, a qubit can only be operated by a *unitary transformation*, and the same happens concerning multiple qubit systems. Classical isomorphisms, now regarded as transformations between two Hilbert spaces, form a special case of unitary (bijective) transformation that is both quantum and classic (Zeng, 2015).

Just like classic computation has its standard gates, e.g. AND, NOT and XOR (Yanofsky and Mannucci, 2008), quantum computation also has its own set of standard gates (Selinger, 2004):

$$\sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}; \qquad \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}; \qquad \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}; \qquad S = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}; \qquad T = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{i} \end{bmatrix}$$

$$CNOT = \begin{bmatrix} I & 0 \\ 0 & X \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

σ_x and $CNOT$ are classic isomorphisms, the latter obtained from XOR by copying one of its inputs to the output. As will be seen in chapters to follow, this is a standard way of converting non-reversible operators into reversible ones, a topic that has received considerable attention (Mu et al., 2004).

$$c_{1,1}|00\rangle + c_{1,2}|01\rangle + c_{2,1}|10\rangle + c_{2,2}|11\rangle$$

measure $q_1 = 0$ measure $q_1 = 1$

$$c_{1,1}|00\rangle + c_{1,2}|01\rangle \qquad c_{2,1}|10\rangle + c_{2,2}|11\rangle$$

measure $q_2 = 0$ measure $q_2 = 1$ measure $q_2 = 0$ measure $q_2 = 1$

$$c_{1,1}|00\rangle \qquad c_{1,2}|01\rangle \qquad c_{2,1}|10\rangle \qquad c_{2,2}|11\rangle$$
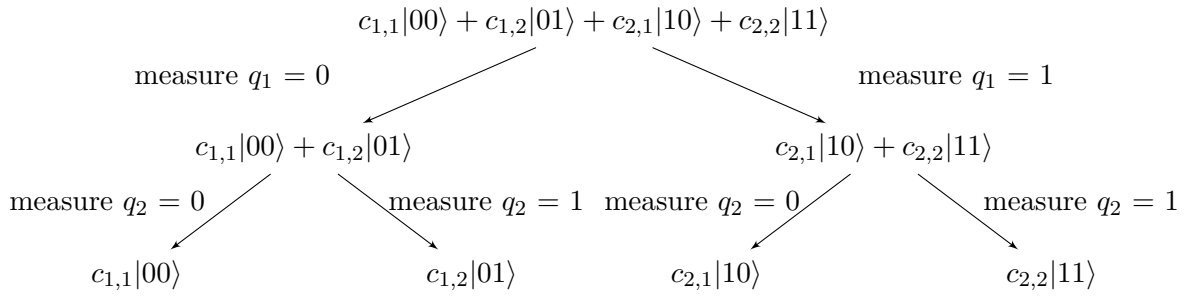
Figure 3.5.: Effect of measuring two (entangled) qubits.

3.4 DENSITY MATRIX

Selinger (2004) draws the picture shown in figure 3.5 to explain the effect of *measuring* two normalised, entangled qubits. Assuming normalised states, the probability of having the initial state collapsing to $c_{1,1}|00\rangle + c_{1,2}|01\rangle$ when the first bit is measured is $|c_{1,1}|^2 + |c_{1,2}|^2$. The other probabilities are achieved in a similar way. Finally, the probability of observing each state is:

$$p(00) = |c_{1,1}|^2$$
$$p(01) = |c_{1,2}|^2$$
$$p(10) = |c_{2,1}|^2$$
$$p(11) = |c_{2,2}|^2$$

Not all information in the system is accessible to the observers.

When a quantum state is exactly known it is called *pure state*. Otherwise, it is called a *mixed state* (Nielsen and Chuang, 2011) and it corresponds to a (classical) probability distribution on quantum states (Selinger, 2004). Since *pure* and *mixed states* are descriptions of the observers' knowledge, it is important to note that:

- a system may be in different mixed states, depending on the viewpoint of the observers;

- a system is always in a pure state; however, this state may be unknown.

While pure states can be represented by vectors, as seen already, *mixed* states call for a matrix representation of a special kind, called *density* matrix:

Definition 3.3. (Selinger, 2004) A hermitian matrix A is a density matrix iff $\text{tr } A = 1$. [5]
□

---
[5]

Definition 3.4. (Weisstein, 2018) trace is the sum of the diagonal elements: $\text{tr} A \Leftrightarrow \sum_{i=1}^{n} a_{ii}$

Let a pure state be represented by the usual column vector u. The corresponding density matrix representation will be $uu^{\dagger}$. Different mixed states can be represented by the same density matrix. Among normalised pure quantum states the representation is, however, unique. Density matrix notation provides a real advantage when representing mixed states, since a mixed state is simply characterised by a linear combination of the density matrices of the pure components.

3.5 PECULIARITIES OF QUANTUM COMPUTING

The importance of quantum computing relies upon its possible applications. As is known, an increasing number of problems are known to be impossible to solve using classical computing. These problems are not impossible to compute in theory but the resources given by a classic computer are not enough to solve these problems (Nielsen and Chuang, 2011).

Quantum computing offers new algorithms to solve such problems. Two well-known examples are (Nielsen and Chuang, 2011):

SHOR'S ALGORITHM — this algorithm, also called the *Shor's quantum Fourier transform*, is used to solve factoring and discrete logarithm problems.

GROVER'S ALGORITHM for quantum search.

These two algorithms quantum algorithms increase in an impressive way the speed with which some problems are solved e.g. breaking RSA (Nielsen and Chuang, 2011).

Besides quantum algorithms, the quantum computing model offers additional mechanisms or procedures, namely (Vicary, 2013):

TELEPORTATION — the state of an arbitrary qubit is transferred from one location to another; (Yanofsky and Mannucci, 2008)

DENSE CODING allowing to send two bits of information in a single qubit (Nielsen and Chuang, 2011).

If used properly, these properties may have a huge impact in several fields such as cryptography or medicine.

A striking peculiarity of quantum systems is the *no-cloning* property. This is an example of a property that is usually taken as granted in any classic physical system and is no longer admitted in quantum systems. In a classical program or circuit one can always copy bits or any other kind of information (e.g. this document can be copied and sent to other machines). In quantum systems this is no longer possible. This quantum particularity is named the *no-cloning theorem*. Proofs of this theorem can be found in the literature, e.g. (Nielsen and Chuang, 2011) or (Coecke and Kissinger, 2017), using string diagrams in the latter case.

Not being able to clone information may seem like an inadequate feature, but in fact it turns out to be the basis of quantum cryptography. In quantum computing, stealing a secret erases the original, meaning that a thefts or intruders can be detected. Quantum programming languages have been studying ways to add this feature to their semantics (Selinger, 2004).

3.6 COMPUTING VERSUS PHYSICS

Quantum computing is a clear intersection between computer science and (quantum) physics. It is not, however, the first time the two disciplines intersect each other. One such other intersection is reversible computing, bringing thermodynamics and information theory together. Reversible computing is in some sense pre-quantum computing, taking into account that every reversible process is unitary, but still classical.

Reversibility promises to reduce energy costs in computing by increasing the injectivity of computations. A program is reversible if the input may be recovered by the information in the output. According to Landauer law (Landauer, 1961) it is possible to decrease the cost of computation by using reversibility. The core of the Landauer principle is a study of the computational processes which reveal that energy and heat are lost due to erasing information, and not as a result of writing information.

Following this line of thought, Bennett (1973) explores the possibility of a reversible computer which does not erase information and consequently does not lose energy. This lead to the study of reversible circuits and programs (Yanofsky and Mannucci, 2008). However, such low-power designs have been struggling with increased complexity associated with reversibility and the need to temporarily store a substantial amount of data.

This raises obstacles to the implementation of such computation principles at an industrial level because industry requirements prioritise size over energy efficiency. Although there is already an extensive study to avoid these problems when increasing injectivity (Dueck and Maslov, 2003), and consequently creating reversible programs, the general feeling is that the main usage of reversible computation will be in quantum computation.

3.7 SUMMARY

This short chapter is aimed at introducing the quantum model of computation and its peculiarities. First of all, it is clear that quantum states, quantum gates and measures can be seen as matrices, and so quantum programs can be easily described through categories of matrices.

Such ideas are not new, as there have been studies like those by Zeng (2015) and Vicary (2013) that use categorial mathematics (with categories of sets, relation and topology) to analyse the semantic of known quantum algorithms.

It is also important to mention that the despite the marvellous capacities of quantum computing, superposition does not mean that every state can be tested at the same time. Moreover, there are quantum constraints of reversibility and the impossibility to copy a state that must be taken into consideration (Ying, 2010).

CALCULATING QUANTUM PROGRAMS

Standard (classic) programming theory relies on a notion of program *refinement*. The starting point typically is a so-called *specification*, which indicates the expected behaviour of the program to be developed with no indication of *how* outputs are computed from the inputs. So, "vagueness" is a chief ingredient of a good specification, giving freedom to the programmer to choose a particular algorithmic solution.

Refinement is the process of deriving one such algorithmic solution from the specification. A program refines another insofar as it increases its "definedness", measured not only in terms of reducing the vagueness of the specification but also increasing the domain of definition (Oliveira and Rodrigues, 2006). In the limit situation, a refinement is a function f (Figure 4.1) such that

$$S \vdash f \quad \Leftrightarrow \quad f \cdot \delta\, S \subseteq S \tag{4.1}$$

holds.[1] In words: *"specification S is refined by implementation f iff function f, once restricted to the domain of definition of S, is a part of S".*

From (4.1) one clearly sees that (nondeterministic) program *inclusion* ($P \subseteq Q$) is the ordering underlying program refinement calculi, notably in the Algebra of Programming (Bird and de Moor, 1997). However, programs derived in this way show no concern for reversibility, as can be seen from the basic principles described above.
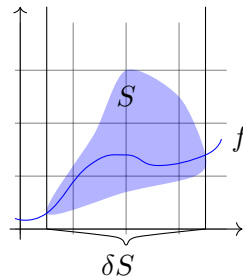
Figure 4.1.: Program from specification.

---

[1] $\delta\, S$ denotes the domain of the specification, that is, the range of inputs for which the program has to deliver an output.

Figure 4.2.: Injectivity (pre)order.

Because quantum programs are always reversible (recall section 3.3), any method for deriving quantum programs has to be concerned, in some way or another, with function *reversibility*. The following section addresses this topic.

4.1 INCREASING INJECTIVITY

Recall that a *bijection* (Figure 2.18) is a function that is both *injective* and *surjective*. Bijections and reversible functions coincide: they are the isomorphisms of the category of sets. Therefore, one needs to find an ordering on functions capturing its "injectiveness" (figure 4.2).[2] As can be found in any school book on discrete maths, a function is injective iff

$$f \ x = f \ x' \Rightarrow x = x'$$

which abbreviates to

$$f^\circ \cdot f \subseteq id$$

as seen in chapter 2. Moreover, g is *less injective* than f is captured by the pre-order

$$g \leq f \quad \Leftrightarrow \quad f \ x = f \ x' \Rightarrow g \ x = g \ x'$$

which in turn simplifies to:

$$g \leq f \quad \Leftrightarrow \quad f^\circ \cdot f \subseteq g^\circ \cdot g$$

In the sequel, the commitment is in exploiting the *injectivity* preorder,[3]

$$R \leq S \quad \Leftrightarrow \quad \mathsf{ker} \ S \subseteq \mathsf{ker} \ R$$

---

[2]Surjectiveness does not need to be handled explicitly because our implementations will be endo-functions $(f : A \to A)$ where A is finite and an injective endo-function a finite set is surjective.

[3]Recall from definition 2.9 that $\mathsf{ker} \ R = R^\circ \cdot R$ is called the *kernel* of relation R.

as a *refinement ordering* guiding programmers towards more and more *injective* computations. This ordering is rich in properties.[4] For instance, it is upper-bounded by relation *pairing*,

$$R \,{\triangledown}\, S \leq X \quad \Leftrightarrow \quad R \leq X \wedge S \leq X \tag{4.2}$$

an operator defined in the expected way:

$$(b, c)\,(R \,{\triangledown}\, S)\,a \quad \Leftrightarrow \quad b\,R\,a \wedge c\,S\,a$$

In the case of functions:

$$(f \,{\triangledown}\, g)\,a \quad = \quad (f\,a, g\,a) \tag{4.3}$$

as seen already. From (4.2) one infers that pairing always increases injectivity:

$$R \leq R \,{\triangledown}\, S \text{ and } S \leq R \,{\triangledown}\, S \tag{4.4}$$

This unfolds to

$$\mathsf{ker}\,(R \,{\triangledown}\, S) \subseteq (\mathsf{ker}\,R) \cap (\mathsf{ker}\,S)$$

which turns out to be an equality:

$$\mathsf{ker}\,(R \,{\triangledown}\, S) = \mathsf{ker}\,R \cap \mathsf{ker}\,S \tag{4.5}$$

This equality is a corollary of the more general:[5]

$$(R \,{\triangledown}\, S)^{\circ} \cdot (Q \,{\triangledown}\, P) = (R^{\circ} \cdot Q) \cap (S^{\circ} \cdot P) \tag{4.6}$$

Similarly to the usual "shunting laws" (2.14, 2.15), injectivity shunting laws also arise as Galois connections, for instance:

$$R \cdot g \leq S \Leftrightarrow R \leq S \cdot g^{\circ} \tag{4.7}$$

Restricted to functions, $(\leq)$ is universally bounded by

$$! \leq f \leq id \tag{4.8}$$

where ! is the only function of its type, $A \to 1$, where 1 is the terminal object (cf. section 2.1).

From (4.8) it is clear that any function f more injective than the identity function is injective. From (4.4) one concludes that $f \,{\triangledown}\, id$ is always injective, for all f.

---

[4]See e.g. (Oliveira, 2014) for a comprehensive account.
[5]See e.g. (Bird and de Moor, 1997).

MINIMAL COMPLEMENTS   As written above, a way of increasing the injectivity of an arbitrary function it to pair it with another function. As a refinement strategy, this corresponds to running non-injective functions inside injective *"envelopes"* and delaying their observation as much as possible. But this cannot be done arbitrarily, as explained below.

Optimal envelopes can be calculated via *complementation*. Two arbitrary functions, f and g, are said to be complementary iff $id \leq (f \triangledown g)$ holds, that is, they are jointly injective.[6] A simple example of complementary functions are the projections

$$\pi_1(a, b) = a \quad , \quad \pi_2(a, b) = b \tag{4.9}$$

which once paired yield the identity itself: $\pi_1 \triangledown \pi_2 = id$ (A.8).[7]

Definition 4.1. Minimal complements (Bancilhon and Spyratos, 1981) Let a function f be given and suppose, for some g:

- $id \leq f \triangledown g$;

- for any other h such that $id \leq f \triangledown h$ and $h \leq g$, then $g \leq h$.

Then g is said to be a minimal complement of f.

□

In other words, a minimal complement is a function (non-unique in general) that captures "what is missing" from the original function to become an injection.

Example 4.1. Let f be the exclusive-or ($\veebar$) function, which is such that:

$$\mathsf{ker}\,(\veebar) = \mathsf{ker} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

The objective is to find a minimal complement g for $\veebar$. Looking at (4.5), $\mathsf{ker}\,g$ has to cancel all 1's outside the diagonal. Function *id* could be used for this but it won't be minimal. Clearly, other options are allowed provided their kernel adds 1s where the kernel of the exclusive-or has 0's. One possibility could be:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

---

[6]Cf. (Matsuda et al., 2007). Other terminologies are *monic pair* (Freyd and Scedrov, 1990) or *jointly monic* (Bird and de Moor, 1997).

[7]Note that, if π_1 is a complement of f, i.e. $id \leq \pi_1 \triangledown f$, this means that f is injective on the second arguments once the first argument is fixed: $f(a, b) = f(a, b') \Rightarrow b = b'$.

However, this matrix is not the kernel of a function because it fails to be an equivalence relation. The matrix describes a symmetric, reflexive relation, but transitivity does not hold.

The easy way to ensure transitivity is to ensure that the matrix is *difunctional*.[8] Difunctionality is easy to check when relations are represented by Boolean matrices: columns either do not intersect or they are the same. Therefore it is possible to reach the kernels of minimal complements of $\dot{\vee}$ by cancelling zeros symmetrically, outside the diagonal:

$$\begin{bmatrix} 1 & 1 & \mathbf{1} & 0 \\ 1 & 1 & 0 & 1 \\ \mathbf{1} & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 1 & 1 \\ 0 & \mathbf{1} & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \mathsf{ker}\ \pi_1$$

or

$$\begin{bmatrix} 1 & \mathbf{1} & 1 & 0 \\ \mathbf{1} & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & \mathbf{1} \\ 0 & 1 & \mathbf{1} & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \mathsf{ker}\ \pi_2$$

The minimal complements of $\dot{\vee}$ are π_1 and π_2. Taking the former, by complementing $\dot{\vee}$ with π_1 one obtains the bijection:

$$\mathbf{2} \times \mathbf{2} \xleftarrow{\ \pi_1 \triangledown \dot{\vee}\ } \mathbf{2} \times \mathbf{2} \ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The outcome is nothing but the classical **CNOT** gate:

$$\begin{cases} cnot(0, b) = (0, b) \\ cnot(1, b) = (1, \neg b) \end{cases}$$

---

[8]R is a difunctional relation iff $R \cdot R^\circ \cdot R \subseteq R$. Suppose R is reflexive ($id \subseteq R$), symmetric ($R^\circ = R$) and difunctional. Then R is transitive, cf.

$$R \cdot R \subseteq R$$
$$\Leftrightarrow \qquad \{\ R^\circ = R,\ R \cdot id = R\ \}$$
$$R \cdot R^\circ \cdot id \subseteq R$$
$$\Leftarrow \qquad \{\ id \subseteq R\ \}$$
$$R \cdot R^\circ \cdot R \subseteq R$$
$$\Leftrightarrow \qquad \{\ \text{definition}\ \}$$
$$R \text{ is difunctional}$$
$$\square$$

.

$\square$

The example just given, started from a *specification* — the original logic gate ($\lor$) — and obtained a version of it — the *cnot* gate — as reversible *implementation*. This illustrates a constructive approach to the creation of quantum gates which is more general than it appears to be. Noting that *cnot* can be expressed by

$$
\begin{array}{c}
x \quad \rule[0.5ex]{2em}{0.4pt}\boxed{cnot}\rule[0.5ex]{2em}{0.4pt} \quad x \\[1em]
y \quad \rule[0.5ex]{2em}{0.4pt}\rule[0.5ex]{2em}{0.4pt} \quad (id\,x)\lor y
\end{array}
\tag{4.10}
$$

let us generalize id to some function f and $\lor$ to the binary operator θ of some monoid $(B; \theta, 0)$ such that

$$x \, \theta \, x = 0 \tag{4.11}$$

holds, and define the following generic gate, parametric on f and θ:

$$
\begin{array}{c}
x \quad \rule[0.5ex]{2em}{0.4pt}\boxed{Uf}\rule[0.5ex]{2em}{0.4pt} \quad x \\[1em]
y \quad \rule[0.5ex]{2em}{0.4pt}\rule[0.5ex]{2em}{0.4pt} \quad (fx)\,\theta\,y
\end{array}
\tag{4.12}
$$

Gate (4.12) can be expressed as:

$$Uf : (A \to B) \to (A \times B) \to (A \times B)$$
$$Uf = \pi_1 \,^\triangledown (\theta \cdot (f \times id))$$

Below is proven that this is a reversible gate for *any* $f : A \to B$:

$$
\begin{aligned}
& (Uf) \cdot (Uf) = id \\
\Leftrightarrow \quad & \qquad \{Uf(x, y) = (x, (f\,x)\,\theta\,y)\} \\
& Uf(x, (f\,x)\,\theta\,y) = (x, y) \\
\Leftrightarrow \quad & \qquad \{\text{again } Uf(x, y) = (x, (f\,x)\,\theta\,y)\} \\
& (x, (f\,x)\,\theta\,((f\,x)\,\theta\,y)) = (x, y) \\
\Leftrightarrow \quad & \qquad \{\theta \text{ is associative and } x\,\theta\,x = 0\} \\
& (x, 0\,\theta\,y) = (x, y) \\
\Leftrightarrow \quad & \qquad \{0\,\theta\,x = x\} \\
& (x, y) = (x, y) \\
& \square
\end{aligned}
$$
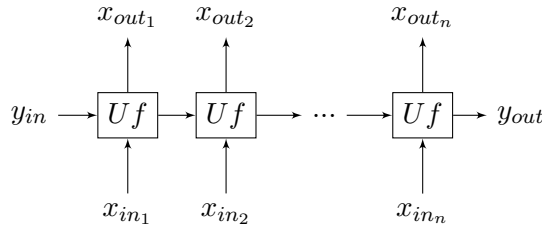
Figure 4.3.: Chaining n π_1-complement computation.

Noting that (4.12) can be drawn in another 2D layout,
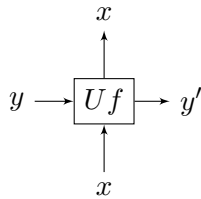
we observe that such π_1-complemented computations can be chained in the way pictured in figure 4.3, which is reminiscent of the accumulative map (recursive) pattern $mapAccumR$ in Haskell. This provides some intuition about our next target — extend π_1 complementation to recursive computations.

4.2 RECURSIVE PROGRAMS

The *cnot* example given above shows $A \times B \xrightarrow{\pi_1} A$ being paired with some function of type $A \times B \to B$ resulting in a bijection of type $A \times B \to A \times B$ in case of complementation. The goal now is to extend this technique to recursive (non-injective) functions, for instance,

$$k : A^* \to B$$
$$k\,[\,] = b$$
$$k(a : x) = f(a, k\,x)$$

so that $\pi_1 \triangledown k$ is injective at a minimal cost. Note the type $f : A \times B \to B$. As is well-known, k can be expressed by the "fold" combinator, $k = \mathsf{foldr}\ \overline{f}\ b$. Let us define

$$(\!|f|\!) : A^* \times B \to B$$
$$(\!|f|\!)\,(x, b) = \mathsf{foldr}\ \overline{f}\ b$$
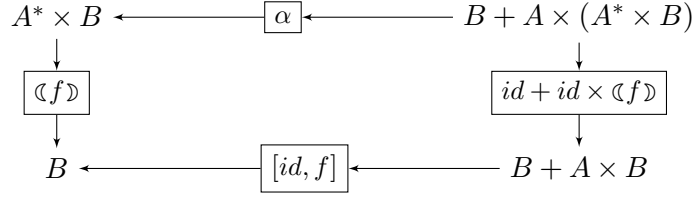
Figure 4.4.: Diagram of generic fold.

that is

$$(\!(f)\!)\ ([\ ], b) = b$$

$$(\!(f)\!)\ (a : x, b) = f\ (a, (\!(f)\!)\ (x, b))$$

The diagram describing $(\!(f)\!)$ is given in figure 4.4, where the recursive path is governed by functor $\mathsf{F}\,X = B + A \times X$.[9] Our research question at this point is: suppose $\pi_1 \triangledown f$ is injective — is $\pi_1 \triangledown (\!(f)\!)$ injective? We shall handle a simpler situation first, as preparation for the answer.

FOR-LOOPS   The following recursive function,

$$\mathsf{for}\ f\ i\ 0 = i$$

$$\mathsf{for}\ f\ i\ (n+1) = f\ (\mathsf{for}\ f\ i\ n)$$

is such that $\mathsf{for}\ f\ i\ n = f^n\ i$. That is, it iterates a function $f : B \to B$ as many times as n, with starting input i. In programming, this is known as a for-loop. Note the type $\mathsf{for} : (B \to B) \to B \to \mathbb{N}_0 \to B$, that is, $\mathsf{for}\ f : B \to \mathbb{N}_0 \to B$. As we did above for finite lists, we define a similar combinator over natural numbers, $(\!(f)\!)(n, i) = \mathsf{for}\ f\ i\ n$,

$$(\!(f)\!)(0, i) = i$$

$$(\!(f)\!)(n+1, i) = f((\!(f)\!)(n, i))$$

whose diagram is shown in figure 4.5, for functor $\mathsf{F}\,X = B + X$. Isomorphism α is[10]

$$\alpha = [\underline{0} \triangledown id, succ \times id] = [\underline{0}, succ \cdot \pi_1] \triangledown [id, \pi_2] \tag{4.13}$$

It will be convenient to generalize $[id, f]$ in figure 4.5 to g in figure 4.6, leading to the universal property

$$k = (\!(g)\!) \quad \Leftrightarrow \quad k \cdot \alpha = g \cdot \mathsf{F}\,k \tag{4.14}$$

---

[9]Isomorphism α will be discussed later.

[10]Recall that constant functions are denoted by $\underline{k}\ x = k$.
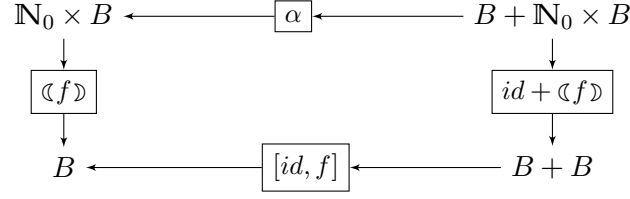
Figure 4.5.: Fold over natural numbers.

where $\mathsf{F}\,k = id + k$. From (4.14) one draws the usual reflexion-property:

$$(\!(\alpha)\!) = id \qquad\qquad (4.15)$$

Then:

$$\pi_1 \cdot \alpha = [\underline{0}, succ \cdot \pi_1]$$

$$\Leftrightarrow \qquad \{\ +\text{-absorption}\ \}$$

$$\pi_1 \cdot \alpha = [\underline{0}, succ] \cdot (id + \pi_1)$$

$$\Leftrightarrow \qquad \{\ \text{universal property (4.14)}\ \}$$

$$\pi_1 = (\!([\underline{0}, succ])\!)$$

$$\Leftrightarrow \qquad \{\ \text{definition of the initial algebra } \mathsf{in}_{\mathbb{N}_0} = [\underline{0}, succ]\ \}$$

$$(\!(\mathsf{in}_{\mathbb{N}_0})\!) = \pi_1 \qquad\qquad (4.16)$$

From (4.14) we also get a "banana-split" law:

$$(\!(g)\!) \mathbin{\triangledown} (\!(h)\!) = (\!((g \cdot (id + \pi_1)) \mathbin{\triangledown} (h \cdot (id + \pi_2)))\!) \qquad\qquad (4.17)$$

This is all we need to study the π_1-complement of $(\!([id, f])\!)$, which we shall denote by

$$\mathbb{N}_0 \times B \xleftarrow{\quad \Psi f \quad} \mathbb{N}_0 \times B \ = \pi_1 \mathbin{\triangledown} (\!([id, f])\!) \qquad\qquad (4.18)$$

for an arbitrary $f : B \leftarrow B$. We reason:

$$\Psi f$$

$$= \qquad \{\ \text{definition of } \Psi f \text{ (4.18) and (4.16)}\ \}$$

$$(\!([\underline{0}, succ])\!) \mathbin{\triangledown} (\!([id, f])\!)$$

$$= \qquad \{\ \text{banana-split (4.17)}\ \}$$

$$(\!([\underline{0}, succ \cdot \pi_1] \mathbin{\triangledown} [id, f \cdot \pi_2])\!)$$

$$= \qquad \{\ \text{exchange law (A.28)}\ \}$$

$$(\!([\underline{0} \mathbin{\triangledown} id, succ \times f])\!)$$
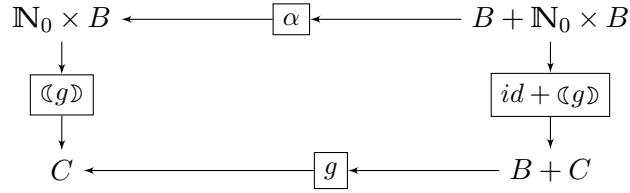
Figure 4.6.: Generalized for-loop ('fold' over the natural numbers).

From $\Psi f = (\!\![\underline{0} \triangledown id, succ \times f]\!\!)$ we draw, via the universal property:

$$\Psi f \cdot (\underline{0} \triangledown id) = \underline{0} \triangledown id$$
$$\Psi f \cdot (succ \times id) = (succ \times f) \cdot \Psi f$$

Recall that $\Psi f = \pi_1 \triangledown (\!\![id, f]\!\!)$ and so $\Psi f(n, b) = (n, f^n\ b)$ — a for-loop that preserves its input:

(4.19)

From the two equations defining Ψf above one can straightforwardly derive the following implementation in the Haskell programming language, where `qfor f` is concrete syntax for Ψf:

```
qfor :: (b -> b) -> (Int, b) -> (Int, b)
qfor f (0,b) = (0,b)
qfor f (n+1,b) = let (m, b') = qfor f (n, b)
                 in  (m+1,f b')
```

This implementation will be used in chapter 5.

Finally, one needs to check how injective Ψf is be knowing that f is so. By the rule

$$[R, S] \text{ is injective iff both } R \text{ and } S \text{ are injective and } R^\circ \cdot S \subseteq \bot$$

$[\underline{0} \triangledown id, succ \times f]$ is injective, because $succ \times f$ and $\underline{0} \triangledown id$ are so, and:

$$(\underline{0} \triangledown id)^\circ \cdot (succ \times f) \subseteq \bot$$

$\Leftrightarrow$ { converses }

$$(succ^\circ \times f^\circ) \cdot (\underline{0} \triangledown id) \subseteq \bot$$

$\Leftrightarrow$ { $\times$-absorption }

$$(succ^\circ \cdot \underline{0}) \triangledown f^\circ \subseteq \bot$$

$\Leftarrow$ { $\bot \triangledown R = \bot$ }

$$succ^\circ \cdot \underline{0} = \bot$$

$\Leftrightarrow$ { since $n + 1 \neq 0$ for all n }

true

Since $\Psi f = \Cap[\underline{0} \triangledown id, succ \times f]\Cup$ and $[\underline{0} \triangledown id, succ \times f]$ is injective for injective f, Ψf will be injective provided $\Cap f \Cup$ preserves injectivity. Let us abbreviate $\Cap f \Cup$ by k and $\mathsf{ker}\, k$ by K:

$$K = k^\circ \cdot k$$

$\Leftrightarrow$ { unfold $k = f \cdot \mathsf{F}\, k \cdot \alpha^\circ$ by (4.14) }

$$K = \alpha \cdot \mathsf{F}\, (k^\circ) \cdot f^\circ \cdot f \cdot \mathsf{F}\, k \cdot \alpha^\circ$$

$\Leftrightarrow$ { assumption: f is injective, $f^\circ \cdot f = id$ }

$$K = \alpha \cdot \mathsf{F}\, (k^\circ) \cdot \mathsf{F}\, k \cdot \alpha^\circ$$

$\Leftrightarrow$ { $\mathsf{F}\,(R \cdot S) = (\mathsf{F}\, R) \cdot (\mathsf{F}\, S)$ and $\mathsf{F}\,(R^\circ) = \mathsf{F}\, R^\circ$ }

$$K = \alpha \cdot \mathsf{F}\, (k^\circ \cdot k) \cdot \alpha^\circ$$

$\Leftrightarrow$ { $K = k^\circ \cdot k$; UP (for relations) }

$$K = \Cap \alpha \Cup$$

$\Leftrightarrow$ { Reflexion: $\Cap \alpha \Cup = id$ (4.15) }

$$K = id \qquad\qquad\qquad (4.20)$$

$\square$

FOLDS OVER FINITE LISTS   Back to figure 4.4, the same steps done for for-loops can now be repeated for "folds'" over finite lists, recall:

$$\Cap f \Cup ([\,], b) = b$$
$$\Cap f \Cup (a : x, b) = f\, (a, \Cap f \Cup (x, b))$$

The isomorphism α of the generic fold over finite lists (figure 4.4) is:

$$\alpha = [nil \triangledown id, (cons \times id) \cdot \mathsf{a}] \qquad\qquad\qquad (4.21)$$
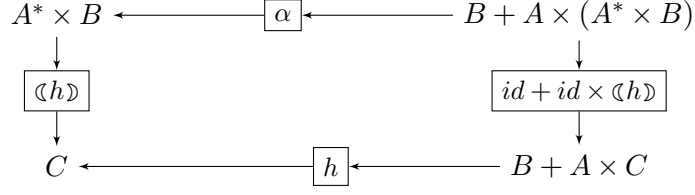
Figure 4.7.: Generalized fold over lists.

This involves the components

$$nil\_\_ = []$$
$$cons(a, x) = a : x$$

of the initial algebra of lists $in^* = [nil, cons]$ and the isomorphism:

$$(A \times B) \times C \xleftarrow{\quad \mathsf{a} \quad} A \times (B \times C) \; = (id \times \pi_1) \,^\triangledown (\pi_2 \cdot \pi_2) \qquad (4.22)$$

Following the same line of reasoning of the for-loop, the next step is to generalize figure 4.4 to figure 4.7, the former being obtained from the latter by making $h = [id, f]$.

The universal property of such a generalization is

$$k = (\!|h|\!) \quad \Leftrightarrow \quad k \cdot \alpha = h \cdot \mathsf{F}\, k \qquad (4.23)$$

where, this time, $\mathsf{F}\, k = id + id \times k$. Note how the two universal properties "look the same", modulo a different functor F and a different isomorphism α. It is, therefore, no wonder that a similar result is obtained concerning π_1:

$$\pi_1 \cdot \alpha = [nil, cons \cdot \pi_1 \cdot \mathsf{a}]$$

$\Leftrightarrow \qquad \{ \text{ definition (4.22) } \}$

$$\pi_1 \cdot \alpha = [nil, cons \cdot (id \times \pi_1)]$$

$\Leftrightarrow \qquad \{ \text{ +-absorption (A.22) } \}$

$$\pi_1 \cdot \alpha = [nil, cons] \cdot (id + id \times \pi_1)$$

$\Leftrightarrow \qquad \{ \; in^* = [nil, cons]; \text{ universal property (4.23) } \}$

$$A^* \xleftarrow{\quad \pi_1 \quad} A^* \times B \; = (\!|in^*|\!) \qquad (4.24)$$

$\square$

From this point onwards the parallelism with for-loops is not so close because, while $f : B \to B$ in the case of for-loops could be assumed injective, $f : A \times B \to B$ in the case of folds can never be injective for A not a singleton type. However, should f be π_1-complemented,

$$A^* \times B \longleftarrow \boxed{\alpha} \longleftarrow B + A \times (A^* \times B)$$

$$\downarrow \qquad\qquad\qquad\qquad\qquad \downarrow$$

$$\boxed{(\![h]\!)} \qquad\qquad\qquad \boxed{id + id \times (\![h]\!)}$$

$$\downarrow \qquad\qquad\qquad\qquad\qquad \downarrow$$

$$A^* \times B \longleftarrow \boxed{h} \longleftarrow B + A \times (A^* \times B)$$

$$\downarrow$$

$$\boxed{\alpha} \longleftarrow \boxed{\Phi(\pi_1 \,{}^{\triangledown} f)}$$

$$\downarrow$$

$$B + A \times (A^* \times B)$$
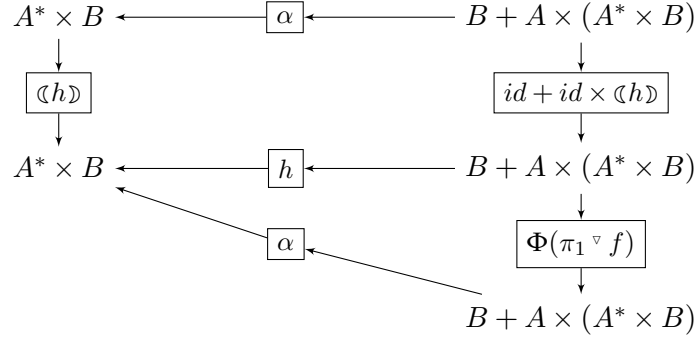
Figure 4.8.: Complement π_1 with f to build the envelop for foldr $\overline{f}b$.

$\pi_1 \,{}^{\triangledown} f$ will be injective, and there are chances of $(\![\pi_1 \,{}^{\triangledown} f]\!)$ also being π_1-complemented. This plan is shown in the diagram of figure 4.8, where

$$\Phi\, x = id + \mathsf{xl} \cdot (id \times x) \cdot \mathsf{xl} \tag{4.25}$$

assumes the isomorphism $A \times (B \times C) \xrightarrow{\ \mathsf{xl}\ } B \times (A \times C)$.

Recall that the complement $\pi_1 \,{}^{\triangledown} (\![[id, f]]\!)$ has the type $A^* \times B \leftarrow A^* \times B$. Therefore the following step is to compare $\pi_1 \,{}^{\triangledown} (\![[id, f]]\!)$ with $(\![\alpha \cdot \Phi(\pi_1 \,{}^{\triangledown} f)]\!)$. The equality $\pi_1 = (\![\mathsf{in}^*]\!)$ (4.24) hints the application of "banana-split",

$$(\![g]\!) \,{}^{\triangledown} (\![h]\!) = (\![(g \cdot \mathsf{F}\, \pi_1) \,{}^{\triangledown} (h \cdot \mathsf{F}\, \pi_2)]\!)$$

(for $\mathsf{F}\, k = id + id \times k$) once again:

$$\pi_1 \,{}^{\triangledown} (\![[id, f]]\!)$$

$$= \qquad \{\text{ banana-split (above) }\}$$

$$(\![(\mathsf{in}^* \times [id, f]) \cdot (\mathsf{F}\, \pi_1 \,{}^{\triangledown} \mathsf{F}\, \pi_2)]\!)$$

$$= \qquad \{\text{ pairing laws (products) ; } \mathsf{F}\, k = id + id \times k \ \}$$

$$(\![[nil, cons \cdot (id \times \pi_1)] \,{}^{\triangledown} [id, f \cdot (id \times \pi_2)]]\!)$$

$$= \qquad \{\text{ exchange law A.28 }\}$$

$$(\![[nil \,{}^{\triangledown} id, (cons \cdot (id \times \pi_1)) \,{}^{\triangledown} (f \cdot (id \times \pi_2))]]\!)$$

$$= \qquad \{\text{ products A.6 ; } \mathsf{a} \cdot \mathsf{a}^\circ = id \ \}$$

$$(\![\alpha \cdot (\mathsf{a}^\circ \cdot (id \times \pi_1) \,{}^{\triangledown} (f \cdot (id \times \pi_2)))]\!)$$

Note that

$$(\mathsf{a}^\circ \cdot (id \times \pi_1) \,{}^{\triangledown} (f \cdot (id \times \pi_2))) = \Phi(\pi_1 \,{}^{\triangledown} f)$$

and so:

$$\pi_1 {}^\triangledown ⦅[id, f]⦆ = ⦅\alpha \cdot (\Phi(\pi_1 {}^\triangledown f))⦆ \tag{4.26}$$

The meaning of $\Phi(\pi_1 {}^\triangledown f)$ is easy to spell out:

$$\mathsf{a}^\circ((id \times \pi_1) {}^\triangledown (f \cdot (id \times \pi_2))(a, (x, b)))$$

$$= \qquad \{\text{ composition (A.55) }; \pi_1 \text{ and } \pi_2 \text{ projections (4.9) }\}$$

$$\mathsf{a}^\circ((a, x), f(a, b))$$

$$= \qquad \{\text{associate the right isomorphism } a^\circ\}$$

$$(a, (x, f(a, b)))$$

□

Φ preserves injectivity as it is made of operators all of which preserve injectivity. Interestingly, the proof that $⦅\_⦆$ preserves injectivity is exactly the same as (4.20), which is parametric on F and isomorphism α. That is: $\pi_1 {}^\triangledown f$ injective $\Rightarrow ⦅\alpha \cdot (\Phi(\pi_1 {}^\triangledown f))⦆$ injective. Altogether, the moral is:

The π_1-complementation of f in foldr $\overline{f}$ b *in promoted to the π_1-complementation of the fold itself. That is, π_1-complementation propagates inductively.*

Defining Φ $f = ⦅\alpha \cdot (\Phi(\pi_1 {}^\triangledown f))⦆$, we obtain the reversible gate

$$\tag{4.27}$$

provided f is π_1-complemented — the construction corresponding to Ψ f in the case of for-loops.

4.3 QUANTAMORPHISM

Section 2.5 points out that functions can be represented by Boolean $(0, 1)$-matrices. It is not difficult to regard 0 and 1 as, respectively, 0% and 100% probability of the function delivering the corresponding output. This leads to probabilistic functions, also known as Markov-chains.

As a prototypical model of computations, the function concept suffers a further extension in quantum mechanics, and a rather interesting one — the probabilities of Markov-chains generalize to complex numbers, named *amplitudes*, which, as mentioned in section 3.1, capture information of quantum bits.

As all quantum computations have to be reversible, only the isomorphisms of classical function theory can be regarded as quantum gates. Indeed, they altogether make up the important
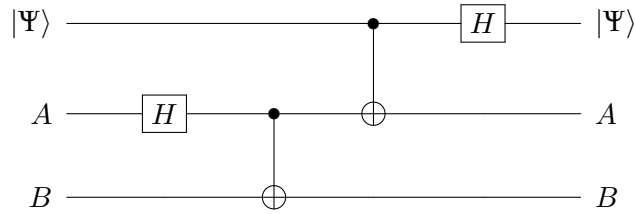
Figure 4.9.: Teleportation protocol (Alice).

class of so-called *classical* gates. What happens, then, to this concept of isomorphism when matrices are allowed to contain complex amplitudes and not just 0s and 1s? Such a concept suffers a dramatic generalization, leading to so-called *unitary matrices* — the building blocks of quantum programming.

Definition 4.2. (Ivanova, 2011) A complex matrix $U : A \to A$ is a unitary matrix iff the Hermitian conjugate is also its inverse, i.e. :

$$U^\dagger \cdot U = U \cdot U^\dagger = id \tag{4.28}$$

where $U^\dagger = \overline{U}^\circ$ is the conjugate transpose of U.

$\square$

Clearly, every classical isomorphism f is unitary because $f^\dagger$ is the converse f° and $f^\circ \cdot f = id = f \cdot f^\circ$. Let us look at a concrete quantum circuit, trying to identify where the unitary transformations are. We take the first part (The Alice part) of the well-known *teleportation* protocol, depicted in figure 4.9. Two instances of *cnot* can be seen, one involving lines A and B and the other involving the lines $|\Psi\rangle$ and A. These are unitary, classic gates. Then we see two instances of the shape
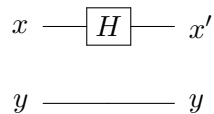
Let us refer to this shape using the symbol h. Then it is clear that the lower *cnot* is preceded by h, while the upper *cnot* is followed by h. Defining

$$bell = cnot \cdot h$$

and

$$unbell = h \cdot cnot$$

we reach figure 4.10. The question is: what does the "composition" operator, $(\cdot)$ mean, in this context?
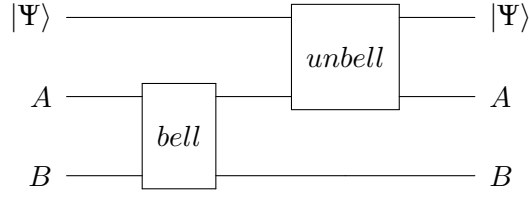
Figure 4.10.: Symmetry in the teleportation protocol (Alice part).

Since sequential composition is no longer between functions but rather between matrices, the answer is that composition is the matrix-matrix multiplication (MMM) mentioned in section 2.1. In a category of matrices, such compositions are characterised by: $bell = \mathbf{4} \xrightarrow{h} \mathbf{4} \xrightarrow{cnot} \mathbf{4}$ and $unbell = \mathbf{4} \xrightarrow{cnot} \mathbf{4} \xrightarrow{h} \mathbf{4}$.

Note that h is not the Hadamard gate (of type $\mathbf{2} \to \mathbf{2}$) — it is actually the tensor product of the Hadamard gate with the identity (this corresponds to the line below, in the diagram). This leads to the next question: what does the product of matrices mean? By extending pairing of relations to pairing of matrices one is led to the so-called *Khatri-Rao product*. The Khatri-Rao product of matrices M and N is defined by:

$$(x, y)(M \triangledown N)a = (xMa)(yNa)$$

As for relations, the product of two matrices arises as the bifunctor associated to pairing[11],

$$M \otimes N = M \cdot \pi_1 \triangledown N \cdot \pi_2$$

which is popularly known as the *Kronecker product*.

In the category of relations, we had the notion of join $R \cup S$ and meet $R \cap S$ of two relations R and S. Moving to matrices, these operations translate to (cell-wise) addition and multiplication, respectively:

$$R \cup S \text{ becomes } M + N$$
$$R \cap S \text{ becomes } M \times N$$

Moreover, categories of matrices are Abelian categories. This means that every homset forms an Abelian group such that composition is bilinear relative to $+$:

$$Q \cdot (M + N) = Q \cdot M + Q \cdot N$$
$$(M + N) \cdot Q = M \cdot Q + N \cdot Q$$

---

[11]A bifunctor is a binary functor of type: $\mathbf{F} : C \times D \to E$. When $C = E = D$, $\mathbf{F}$ is referred to as an endo-bifunctor.

Thanks to matrix product and composition, gates *bell* and *unbell* above can be expressed by $bell = cnot \cdot (H \otimes id)$ and $unbell = (H \otimes id) \cdot cnot$[12], respectively. Moreover, the Alice part of the teleportation protocol is the circuit captured by

$$telep = (unbell \otimes id) \cdot \mathbf{a} \cdot (id \otimes bell) \tag{4.29}$$

where isomorphism **a** — recall (4.22) — is grants the "shift" between the qubits involved.

   In the same way relational terms can be expressed by *monadic* functions over the powerset monad, and Markov chain terms involving probabilistic matrices can be performed by monadic functions over the distribution monad[13], the linear algebra expression *telep* can be formulated as a monadic program too:

$$
\begin{aligned}
telep \quad &(c, (a, b)) = \\
\mathbf{do} \quad &\{ \\
&(a', b') \leftarrow bell(a, b) \\
&(c', a'') \leftarrow unbell(c, a') \\
&\mathbf{return}(c', (a'', b')) \\
&\}
\end{aligned}
$$

Note the type $telep : Vec\ 2 \to Vec\ (2 \times (2 \times 2))$ of the monadic program, where Vec is a (parametric) vector space monad. In this line of thought, the Hadamard gate (H) itself (previously defined as a matrix in section 3.3) can be written pointwise as the Vec-valued function,

$$
\begin{aligned}
H &:: \mathbf{2} \to Vec\ \mathbf{2} \\
H\ 0 &= \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \\
H\ 1 &= \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix}
\end{aligned}
$$

and even as

$$
\begin{aligned}
H &:: \mathbf{2} \to Vec\ \mathbf{2} \\
H\ 0 &= \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\
H\ 1 &= \frac{|0\rangle - |1\rangle}{\sqrt{2}}
\end{aligned}
$$

---

[12]The symbol H is normally used to denote the Hadamard gate.

[13]This requires finite support distributions in the latter case and finitely bounded non-determinism in the relations of the former case (Hasuo et al., 2007; Oliveira and Miraldo, 2016).

where $|0\rangle$ and $|1\rangle$ are the standard basis vectors defined in section 3.2.

Note how the vector monad $Vec\ A$ is captured, in our setting, (finitely supported) complex-valued vectors with base A. Therefore, $A \to Vec\ B$ is a function representing a matrix of type $A \to B$. In fact, every such function represents a matrix and vice-versa. Technically, one says that a category of matrices is the Kleisli category of the corresponding vector-space monad. This allows us to interpret diagrams in the Kleisli category while implementing them as monadic functions. Because we are studying quantum programming, such matrices have to be unitary, recall axiom (3.4).[14]

The question now is: how does one express the recursive, reversible functions of section 4.2 in the situation where the involved operations become unitary computations and not just functions? All that needs to be done is to extend the recursive patterns of section 4.2 (for's and folds, recall) monadically, as explained next.

Starting from the *for*-combinator implementing standard recursive programs over natural numbers, we head for a quantum for-combinator, `qfor`, as is the outcome of the π_1-complement of $(\![id, f]\!)$. Again we represent this by Ψf, recall $\Psi f \cdot \alpha = [\underline{0} \triangledown id, succ \times f] \cdot (id + \Psi f)$ and follow the lemma *"keep definition, change category"*[15]:

$$\Psi M \cdot \alpha = \left[\ \underline{0} \triangledown id\ \middle|\ succ \otimes M\ \right] \cdot (id \oplus \Psi M) \tag{4.30}$$

where f has given place to a unitary matrix M and the product became the Kronecker product. Also, note that relational junction $[R, S]$ becomes the coproduct of two matrices $\left[\ M\ \middle|\ N\ \right]$ (this collates M and N horizontally). The corresponding bifunctor is usually known as *direct sum*:

$$M \oplus N = \left[\ i_1 \cdot M\ \middle|\ i_2 \cdot N\ \right] \tag{4.31}$$

Therefore, we can further calculate:

$$\Psi M \cdot \alpha = \left[\ \underline{0} \triangledown id\ \middle|\ succ \otimes M\ \right] \cdot (id \oplus \Psi M)$$

$\Leftrightarrow$ $\qquad$ { re-arranging }

$$\Psi M = \left[\ \underline{0} \triangledown id\ \middle|\ succ \otimes M)\cdot \Psi M\ \right] \cdot \alpha^{\circ}$$

$\Leftrightarrow$ $\qquad$ { unfold α }

$$\Psi M = \left[\ \underline{0} \triangledown id\ \middle|\ (succ \otimes M)\cdot \Psi M\ \right] \cdot \left[\ \underline{0} \triangledown id\ \middle|\ succ \otimes id\ \right]^{\circ}$$

---

[14]By definition, in an orthogonal basis a unitary transformation corresponds to a unitary matrix, see definition 4.2. A unitary transformation can be regarded as an isomorphism between two Hilbert spaces, meaning that unitary transformations generalize bijective functions.

[15](Oliveira and Miraldo, 2016).

Relations and matrices are so close the relational property $[R, S] \cdot [P, Q]^\circ = R \cdot P^\circ \cup S \cdot Q^\circ$ also holds for matrices, changing the category:[16]

$$\left[\begin{array}{c|c} M & N \end{array} \right] \cdot \left[\begin{array}{c|c} P & Q \end{array} \right]^\circ = M \cdot P^\circ + N \cdot Q^\circ \tag{4.32}$$

This allows writing ΨM as

$$\Psi M = (\underline{0} \triangledown id)(\underline{0} \triangledown id)^\circ + (succ \otimes M) \cdot \Psi M \cdot (succ^\circ \otimes id) \tag{4.33}$$

Therefore we obtain the recursive matrix definition whose least fixpoint is:

$$\Psi M = \mu X.((\underline{0} \triangledown id) \cdot (\underline{0} \triangledown id)^\circ + (succ \otimes M) \cdot X \cdot (succ^\circ \otimes id)) \tag{4.34}$$

This corresponds to a quantum gate that iterates M over two inputs, the target (second input) and the control (first input):

$$\tag{4.35}$$

The monadic function corresponding to this matrix is as simple as the monadic evolution of what we had before,

```
qfor :: (b -> b) -> (Int, b) -> (Int, b)
qfor f (0,b) = (0,b)
qfor f (n+1,b) = let (m, b') = qfor f (n, b)
                 in  (m+1,f b')
```

which we label `mqfor`:[17]

```
mqfor :: (Monad m) => (b -> m b) -> (Int, b) -> m (Int, b)
mqfor f (0,b) = return (0,b)
mqfor f (n+1,b) = do { b' <- f b ;
                       (m,b'') <- mqfor f (n, b');
                       return (m+1,b'')
                     }
```

This implementation will be used in chapter 5.

The question is now: is ΨM unitary for M unitary? Recall that the functor underlying these definitions is $\mathsf{F}\, X = id \oplus X$. Since $\oplus$ is a bifunctor, it commutes with composition (matrix multiplication) and preserves unitary matrices. Moreover, $(\mathsf{F}\, X)^\dagger = \mathsf{F}\, (X^\dagger)$ and, of

---

[16]This equality is known as the *divide & conquer* rule of matrix multiplication.

[17]This definition is generic on monad `m`. For the identity monad, `mqfor` coincides with `qfor`.

course, $\alpha^\dagger = \alpha^\circ$. So we can proceed with the usual proof, abbreviating $\Psi M = X$ and $K = X^\dagger \cdot X$:

$$K = X^\dagger \cdot X$$

$$\Leftrightarrow \quad \{ \text{ unfold } X = M \cdot \mathsf{F} X \cdot \alpha^\circ \}$$

$$K = \alpha \cdot \mathsf{F}(X^\dagger) \cdot M^\dagger \cdot M \cdot \mathsf{F} X \cdot \alpha^\circ$$

$$\Leftrightarrow \quad \{ \text{ assumption: } M \text{ is unitary, definition (4.2): } M^\dagger \cdot M = id \}$$

$$K = \alpha \cdot \mathsf{F}(X^\dagger) \cdot \mathsf{F} X \cdot \alpha^\circ$$

$$\Leftrightarrow \quad \{ \mathbf{F}(M \cdot N) = (\mathbf{F} \cdot M)(\mathbf{F} \cdot N) \text{ and } \mathbf{F}M^\dagger = (\mathbf{F}M)^\dagger \}$$

$$K = \alpha \cdot \mathsf{F}(X^\dagger \cdot X) \cdot \alpha^\circ$$

$$\Leftrightarrow \quad \{ K = X^\dagger \cdot X; \text{ UP for catamorphisms (4.14) } \}$$

$$K = (\!(\alpha)\!)$$

$$\Leftrightarrow \quad \{ \text{ Reflexion: } (\!(\alpha)\!) = id \text{ (4.15) } \}$$

$$K = id \tag{4.36}$$

$\square$

QUANTUM FOLDS This paragraph finally addresses quantamorphisms which are folds of lists over unitary matrices. In this case, function $f : A \times B \to B$ complemented by π_1 gives room to some unitary matrix $U : A \times B \to A \times B$. The strategy is to repeat what we have done before, for functor $\mathsf{F} X = id \oplus id \times X$, which preserves unitary matrices too. This particular case of quantamorphism is depicted in figure 4.11. The corresponding monadic encoding is as follows:

```
mqfold :: ((a, b) -> Vec (c, b)) -> ([a], b) -> Vec ([c], b)
mqfold f ([],b) = return ([],b)
mqfold f (h:t,b) =
  do {
    (t',b') <- mqfold f (t,b) ;
    (h'',b'') <- f(h,b');
    return(h'':t', b'')
    }
```
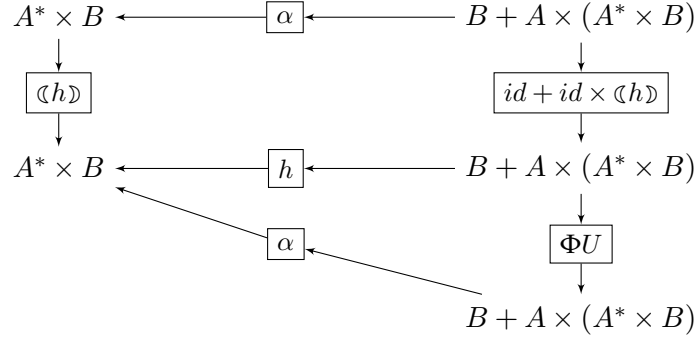
Figure 4.11.: Quantum foldr over unitary matrix U.

This is actually a more generic program, parametric on the underlying monad, that can be pretty-printed as follows:

$$
\begin{aligned}
&(\![.]\!) \quad :: Monad\ \mathrm{T}\ \Rightarrow ((a,b) \to \mathrm{T}(c,b)) \to ([a],b) \to \mathrm{T}([c],b) \\
&(\![f]\!) \quad ([],b) = \mathrm{return}([],b) \\
&(\![f]\!) \quad (h:t,b) = \mathbf{do}\{ \\
&\qquad (t',b') \leftarrow (\![\ f\ ]\!)(t,b); \\
&\qquad (h'',b'') \leftarrow f(h,b') \\
&\qquad \mathrm{return}(h'':t',b'') \\
&\quad \}
\end{aligned}
$$

$$(4.37)$$

The generic program uses a list of classical bits to control qubit b (the target), where f is assumed unitary.

4.4   RUNNING QUANTAMORPHISMS

The monadic encodings of quantamorphisms given above are, as we have seen, in one-to-one correspondence with unitary matrices describing quantum computations. Running such monadic functions is a form of simulating such computations. Take for instance the quantum fold $Q = (\![H \times id]\!)$, where H is the Hadamard gate. Using GHCi, the Haskell standard interpreter, by evaluating $q([0,1,1,1],0)$, where q is the monadic function encoding Q, one obtains the vector

$$q([0, 1, 1, 1], 0) = \begin{array}{r|r}
([0,0,0,0],0) & 0.24999997 \\
([1,0,0,0],0) & -0.24999997 \\
([0,1,0,0],0) & -0.24999997 \\
([1,1,0,0],0) & 0.24999997 \\
([0,0,1,0],0) & -0.24999997 \\
([1,0,1,0],0) & 0.24999997 \\
([0,1,1,0],0) & 0.24999997 \\
([1,1,1,0],0) & -0.24999997 \\
([0,0,0,1],0) & 0.24999997 \\
([1,0,0,1],0) & -0.24999997 \\
([0,1,0,1],0) & -0.24999997 \\
([1,1,0,1],0) & 0.24999997 \\
([0,0,1,1],0) & -0.24999997 \\
([1,0,1,1],0) & 0.24999997 \\
([0,1,1,1],0) & 0.24999997 \\
([1,1,1,1],0) & -0.24999997 \\
\end{array}$$

expressing the superposition of all possible outputs and the corresponding amplitudes.

However, further than simulation, the aim of this master project is to generate actual quantum programs and run these on quantum hardware, namely on IBM Q Experience devices. The strategy devised for achieving this aim is displayed in figure 4.12. Its work-flow consists of four main steps, as follows:

- GHCi - depending on the number of resources available in the target hardware (i.e. number of qubits), the monadic quantamorphisms are used to generate the finite, unitary matrices that describe the intended (recursive) quantum computations;

- Quipper - this tool generates the quantum circuit from the unitary matrix;

- QISKit[18] - the quantum circuit generated by Quipper is passed to this Python interface, which adds error-correction extra circuitry;

- IBM-Q - the actual code generated by QISKit runs on the actual, physical quantum device.

Note that Quipper and QISKit can also both be used to simulate the circuits locally, cross-checking with the behaviour observed in GHCi.

Chapter 5 shall devote itself to describing all the experiments carried out. All of them are benchmarks of recursive quantum programs generated using the principles described in the current chapter.

---

[18]QISKit version earlier than 0.6.
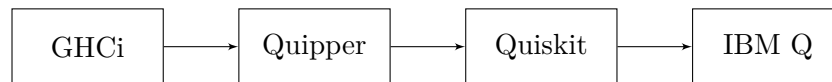
Figure 4.12.: Work-flow.

4.5 SUMMARY

The short version of this chapter goes as follows. Programs can be seen as morphisms of categories and, consequently, obey some mathematical laws. It turns out that, in many situations, it is possible to start creating a program from its specification and refine this specification up to a bijective implementation.

Bijections are reversible, which is a requirement of quantum programs. In this work, the method used to acquire reversibility is the minimal complement (defined in 4.1). Once reversibility is established, at quantum level one tries to assure that the program runs as long as possible without any measure. This is possible for a particular class of recursive programs termed *quantamorphisms*. Two examples of quantamorphism are given, quantum for-loops (where the control is a natural number) and quantum folds (where the control is a finite list).

To make sure that such programs extend to quantum, one somehow needs to reach the standard semantics of quantum programs, usually expressed in linear algebra. In other words, instead of achieving a bijective function, the new goal is to achieve a unitary matrix. For this, the *"keep definition, change category"* principle is followed. Since the new category is the Kleisli category of a particular monad, for the finite case one can write monadic, functional programs to calculate the unitary matrices. The Haskell functional programming language is used for this purpose, paving the way to experimentation.

Although the quantum computers of today are still in an embryonic stage, the quantamorphism concept does not have to linger in a theoretical context. Besides the many ways to simulate quantum computer behaviour, there are quantum computers of IBM Q Experience that can run our programs. The following chapter details the experiments of running the quantum programs devised in the current chapter on real quantum devices.

APPLICATION - CASE STUDIES AND EXPERIMENTS

The previous chapter, chiefly devoted to explaining the theory behind recursive quantum programming combinators called *quantamorphisms*, ended with a prospect of implementing such ideas not only on a classical computer but also, and foremost, on IBM Q Experience quantum devices.

This chapter will describe all the steps needed to carry out such an implementation through examples. Each example starts with a specification, written as a quantamorphism. By encoding this into a (recursive, monadic) function in Haskell, it is possible to recover the matrix that describes the semantics of the program (section 5.2), tuned to the particular resources available. Then Quipper is used to extract a quantum circuit from this matrix (section 5.3). Finally, we resort to QISKit to run the generated circuit on the actual quantum device (section 5.4).

Running the circuits in real quantum devices has raised challenges not anticipated in the theoretical perspective. For better comprehension of the implementation scheme, this chapter starts by introducing typical error sources of IBM Q Experience devices (section 5.1).

5.1 CHALLENGES

DECOHERENCE (Sandberg et al., 2018) A quantum bit is a system that can be accessed and manipulated through quantum gates. Therefore, it will exchange energy with the environment and will not work exactly as predicted by the theory. In particular, it will not maintain its state for an arbitrary time. *Relaxation* derives from energy exchanges between the system and the environment that relax (or excite) the quantum system to a state with a different energy. Experiments reported in the QISKit tutorial (Breitweiser et al., 2018) already show that increasing the time of running a circuit results in exponential decaying of the expected measures.

Further to relaxation, a working quantum computer is expected to interact with its environment. Increasing the number of interaction with the environment (gates) will result in *dephasing*, i.e. in a process that transforms quantum coherent states into classical mixed states and does not result in energy exchange.
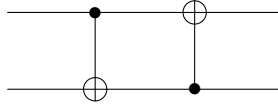
Figure 5.1.: An example of a gate without direct implementation in IBM Q Experience.

LIMITATIONS OF IBM Q EXPERIENCE   IBM Q Experience is not yet ready to directly implement some gates or some combinations of gates, see e.g. figure 5.1. As the quantum circuits in Quipper resulting from our matrices are not always limited to the standard gates available in IBM Q Experience, it is necessary to decompose some gates. Besides, the QISKit program adds extra changes to the circuit in order to adapt to the actual device used.

Such modifications of the initial circuit increase circuit size, not only making circuits more susceptible to error (like decoherence) but also increasing the possibility of problems in QASM [1] (e.g. bugs). In other words, the circuit may have a higher error rate or even be impossible to run a real device.

5.2 GHCI

The applications examined in the chapter follow the general encoding of monadic quantamorphisms, recall e.g. (4.37). As a result, the programs tested always control a target qubit with other qubits, testing classic scenarios first and gradually adding superposition. Experiments involving programs with more qubits follow.

QUANTAMORPHISM for X As seen in section 3.3, the X gate is equivalent to the classical NOT gate. The loop for X accepts two arguments, a number and a boolean. When a number is even nothing happens to the boolean, otherwise the boolean is negated. That is to say, for X tests the parity of the number given as input. In Haskell, this calls for the `qfor` combinator, recall:

```
qfor :: (b -> b) -> (Int, b) -> (Int, b)
qfor f (0,b) = (0,b)
qfor f (n+1,b) = let (m,b') = qfor f (n, f b) in (m+1,b')
```

This experiment tested two different cardinalities, corresponding to 2 (resp. 3) control qubits enabling numbers from 0 to 3 (resp. 7), that is, to the range of inputs $\{00, 01, 10, 11\}$ (resp. $\{000, 001, 010, 011, 100, 101, 110, 111\}$). The corresponding implementations in Haskell result in the matrices (5.1) and (5.2). To better understand the meaning of these matrices,

---

[1] QASM is a simple text language that describes the generic quantum circuit (Cross et al., 2017).

the truth table 5.1 makes it simple to comprehend what the outputs expected for each input are.

	(0,False)	(0,True)	(1,False)	(1,True)	(2,False)	(2,True)	(3,False)	(3,True)
(0,False)	1	0	0	0	0	0	0	0
(0,True)	0	1	0	0	0	0	0	0
(1,False)	0	0	0	1	0	0	0	0
(1,True)	0	0	1	0	0	0	0	0
(2,False)	0	0	0	0	1	0	0	0
(2,True)	0	0	0	0	0	1	0	0
(3,False)	0	0	0	0	0	0	0	1
(3,True)	0	0	0	0	0	0	1	0

(5.1)

	(0,False)	(0,True)	(1,False)	(1,True)	(2,False)	(2,True)	(3,False)	(3,True)	(4,False)	(4,True)	(5,False)	(5,True)	(6,False)	(6,True)	(7,False)	(7,True)
(0,False)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(0,True)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(1,False)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
(1,True)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
(2,False)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
(2,True)	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
(3,False)	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
(3,True)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
(4,False)	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
(4,True)	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
(5,False)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
(5,True)	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
(6,False)	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
(6,True)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
(7,False)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
(7,True)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

(5.2)

Clearly, when the control qubits (q_0 and q_1) of the input represent an even number the target qubit (q_2) value holds, otherwise the control qubits represent an odd number and the target changes.

In the following experiments, we shall drop the matrix label descriptions for the economy of space.

QUANTAMORPHISM for Y The Pauli gates (X gate, Y gate and Z gate) stand out as some of the most used. While the previous experiment used the X gate (a completely

input			output		
q0	q1	q2	q0	q1	q2
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Table 5.1.: Truth table of circuit for-loop quantamorphism over X gate.

classical gate), in the next we want to observe truly quantum outcomes. For this purpose, we use the Y gate.

In this case, wherever the number represented by the control bits is even or not, the target preserves its state. Otherwise, the state rotates π around of the Y-axis of the Bloch sphere; in other words, $|0\rangle$ is changed to $i|1\rangle$ and $|1\rangle$ to $-i|0\rangle$. Since this gate is truly quantum, we have to use the monadic implementation of for-loops, recall `mqfor`:

```
mqfor :: (Monad m) => (b -> m b) -> (Int, b) -> m (Int, b)
mqfor f (0,b) = return (0,b)
mqfor f (n+1,b) = do { b' <- f b ; (m,b'') <- mqfor f (n, b'); return (m+1,b'') }
```

The matrix corresponding to for Y for 2 control qubits (numbers from 0 to 3) is given by (5.3).

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -i & 0 & 0 & 0 & 0 \\
0 & 0 & i & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & i \\
0 & 0 & 0 & 0 & 0 & 0 & i & 0
\end{bmatrix}
\tag{5.3}
$$

Although this program involves complex numbers in its matrix, the measure will have the same result as in the truth table 5.1.

QUANTAMORPHISM for H (CREATING SUPERPOSITION)   To assure quantum superposition in the target when the control is an odd number we use Hadamard gate . This quantum rotation gate is the most commonly used to create superposition. H gate maps

inputs		output
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

Table 5.2.: Truth table of the XOR gate.

the state $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. This means that once the control qubis trigger the Hadamard gate, the target (assuming it is in the default state $|0\rangle$) will have $50/50$ chances to measure 0 or 1. The outcome of quantamorphism `mqfor` passing H as parameter is the matrix (5.4).

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{5.4}$$

QUANTAMORPHISM foldr XOR Recall from section 3.3 that XOR (not reversible, see the truth table 5.2) becomes $CNOT$ (reversible) once π_1-complemented. So the quantamorphism folding over XOR is a classical circuit controlled by a list of qubits, recall:

```
qfold :: ((a,b)->(c,b))->([a],b)->([c],b)
qfold f ([], b) = return ([],b)
qfold f (h:t,b) =
    do {
        (t',b') <- qfold f (t, b);
        (h'',b'') <- f (h,b');
        return (h'':t',b'')
    }
```

Using lists of at most 4 qubits the outcome of this quantamorphism folding over XOR is the matrix given in (5.5).

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix} \tag{5.5}
$$

5.3 QUIPPER

Quipper is a scalable functional programming language for quantum computation. It does not depend on any particular hardware but is adjusted to the computation model consisting of a classical computer (the master, for control) and a quantum device (the slave) (Green et al., 2013a,b).

The main purpose of using Quipper in our experiments is to generate quantum circuits from the matrices produced in section 5.2. Quipper has mechanisms to perform this in an automatic way.

Furthermore, there is interest in simulating these circuits in order to verify if approximations play a significant part in this part of the tool-chain. For a detailed analysis, printing the circuits and its description is necessary.

All the programs described in this section can be found in appendix B. The following information regards the implementation.

FOR-LOOP QUANTAMORPHISM OVER X GATE   Back to this example, the aim is to produce a quantum circuit from matrix (5.2). Of the various methods for generating circuits in Quipper, the selected alternative is the function `exact_synthesis` from `QuipperLib.Synthesis`. This function receives a matrix and generates a circuit without ancillas. Listing 5.1 exhibits

how to use function `exact_synthesis` and how to use the result to define a circuit with $[a, b]$ as control qubits and c as target qubit.

```
1  -- From a matrix
   mymatrix :: Matrix Eight Eight (Integer)
3  mymatrix = matrix_of_function f
     where
5      f i j
         | i == 0 && j == 0 = 1
7        | i == 1 && j == 1 = 1
         | i == 2 && j == 3 = 1
9        | i == 3 && j == 2 = 1
         | i == 4 && j == 4 = 1
11       | i == 5 && j == 5 = 1
         | i == 6 && j == 7 = 1
13       | i == 7 && j == 6 = 1
         | otherwise  = 0
15
   synthesized = exact_synthesis mymatrix
17
   circuit ::([Qubit], Qubit)-> Circ ([Qubit], Qubit)
19 circuit ([a, b], c) = do
     synthesized [a,b,c]
21   return ([a,b],c)
```

Listing 5.1: Quipper `exact_synthesis` implementation.

The next step is to embed the original circuit into one main circuit. The main circuit receives the original circuit as an oracle and includes the qubit state preparation and measurements. (See listing 5.2.)

```
1  -- declare circuit function
   circuit_function :: Oracle -> Circ ([Bit], Bit)
3  circuit_function oracle = do
       -- initialize string of qubits
5      top_qubits <- qinit (replicate (qubit_num oracle) False)
       bottom_qubit <- qinit True
7      label (top_qubits, bottom_qubit) ("|0>","|1>")

9      -- set the initial states of the qubits
       mapUnary hadamard top_qubits
11
       comment "before oracle"
13     -- call oracle
       function oracle (top_qubits, bottom_qubit)
15     comment "after oracle"

17     -- measure qubits
```

input		
q0	**q1**	**q2**
$|0\rangle$	$|0\rangle$	$|0\rangle$
$|0\rangle$	$|0\rangle$	$|1\rangle$
$|0\rangle$	$|0\rangle$	$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
$|1\rangle$	$|1\rangle$	$|0\rangle$
$|1\rangle$	$|1\rangle$	$|1\rangle$
$|1\rangle$	$|1\rangle$	$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$	$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$	$|0\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$	$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$	$|1\rangle$
$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$	$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$	$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$

Table 5.3.: Input for circuit for-loop quantamorphism over X gate.

```
        (top_qubits, bottom_qubit) <- measure (top_qubits, bottom_qubit)
19      -- discard unnecessary output and return result
        return (top_qubits, bottom_qubit)
```

Listing 5.2: Quipper main circuit function implementation.

This change in the qubits can be implemented by adding the lines of listing 5.3. In particular, the `gate_X` is added to shift states from 0 to 1^2 and `Hadamard` to create superposition. The sets of inputs tested can be found in table 5.3.

```
        -- set the initial states of the qubits
2        mapUnary Hadamard top_qubits
        -- and/or
4        mapUnary gate_X bottom_qubits
```

Listing 5.3: Set Quipper inputs.

When adding additional control qubits, the matrix implemented is also different — compare the matrix as implemented in listing 5.4 with matrix (5.2).

```
  -- From a matrix
2 type Sixteen = Ten_and Six

4

  mymatrix :: Matrix Sixteen Sixteen (Integer)
6 mymatrix = matrix [[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                     [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
8                    [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
```

---

[2]Quipper allows initializing qubits with 0 and 1, hence we can choose not to use X gate if it is the first gate to operate a qubit.

```
                    [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
10                  [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
12                  [0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
14                  [0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0],
16                  [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],
18                  [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0],
20                  [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
                    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]]

22


24  synthesized = exact_synthesis mymatrix

26  circuit ::([Qubit], Qubit)-> Circ ([Qubit], Qubit)
    circuit ([a, b, c], d) = do
28    synthesized [a,b,c,d]
      return ([a,b,c],d)
```

Listing 5.4: Quipper `exact_synthesis` implementation of quantamorphism for X with 3 control qubits.

While Quipper accepts a large number of quantum gates, QISKit is much more limited in terms of gate diversity. This results in the need to decompose circuits. The IBM Q Experience tutorial provides some explanation on how to decomposed such gates but Quipper goes further and provides a function that decomposes circuits for us, see e.g. listing 5.5.

```
1  -- Decompose the circuit in the standard gates
   rand = RandomSource(fst(split(mkStdGen 10)))
3  prec = 20 * bits

5  circuit_decompose = decompose_generic (Standard prec rand) circuit
```

Listing 5.5: Quipper `decompose_generic` implementation.

To ensure that the circuit of quantamorphism `for` X, for numbers up to 7, is easy for QISKit to translate, the type of decomposition chosen is *Standard*. This means that the decomposed circuit is limited to the essential gates: Pauli Gates (X, Y, Z), Hadamard gate, $CNOT$, S, $S^\dagger$, T and $T^\dagger$.

Compared to other decomposition types, *Standard* has the advantage of reducing the decomposition made by QISKit.[3] The downside of applying the *Standard* decomposition are the approximations, the increase in the number of gates and the addition of auxiliary qubits, all of which contribute for a larger error rate when experimenting on a real device.

Because of this decomposition, this experiment offers results in two data sets (the original and the decomposed), each of comprising a circuit, a circuit description and simulation tests.

FOR-LOOP QUANTAMORPHISM OVER Y GATE   The implementation of the matrix generated by `mqfor` Y is similar to the experiment over X gate with the maximum number equal to 3. However, the type of matrix is different, for its elements are no longer `Integer`, neither are such elements of type `Complex`, as in Haskell. Quipper asks for a specific type: `Cplx Integer`. The implementation can be found in listing 5.6.

```
   -- From a matrix
2  mymatrix :: Matrix Eight Eight (Cplx Integer)
   mymatrix = matrix [[Cplx (1)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
       (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
4                     [Cplx (0)(0), Cplx (1)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                          (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
                      [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(-1), Cplx
                          (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
6                     [Cplx (0)(0), Cplx (0)(0), Cplx (0)(1), Cplx (0)(0), Cplx
                          (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
                      [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                          (1)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
8                     [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                          (0)(0), Cplx (1)(0), Cplx (0)(0), Cplx (0)(0) ],
                      [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                          (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(-1) ],
10                    [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                          (0)(0), Cplx (0)(0), Cplx (0)(1), Cplx (0)(0)]]

12
   synthesized = exact_synthesis mymatrix
14
   circuit :: ([Qubit], Qubit) -> Circ ([Qubit],Qubit)
16 circuit ([a, b], c) = do
     synthesized [a,b,c]
18   return ([a,b],c)
```

Listing 5.6: Quipper `exact_synthesis` implementation of `mqfor` Y.

---

[3]QISKit is still work in progress and suffers from a considerable amount of bugs. Thus our choice of relying on Quipper decompositions rather than on QISKit decompositions.

A short analysis of the circuit generated for this matrix (figure 5.13) exposes gates with complex implementation in QISKit. The function for decomposing this circuit is equal to the one previously used to decompose `mqfor X`, with inputs ranging over $(\{0, ..., 7\},$ boolean$)$, see listing 5.5. Furthermore, observe how the code in listings 5.7 and 5.8 is applied to call the simulations, where the function `simulate` is implemented in listing 5.9.

```
1   simulate (circuit_function my_oracle)
    where
3    -- declare empty_oracle's data type
    my_oracle :: Oracle
5    my_oracle = Oracle {
        -- set the length of qubit string
7        qubit_num = 2,
        function = circuit
9    }
```

Listing 5.7: Call simulation of original circuit.

```
    simulate (circuit_function my_oracle)
2   where
     -- declare empty_oracle's data type
4    my_oracle :: Oracle
    my_oracle = Oracle {
6        -- set the length of qubit string
        qubit_num = 2,
8        function = circuit_decompose
    }
```

Listing 5.8: Call simulation of the decomposed circuit.

```
    -- simulate function
2   simulate :: Circ ([Bit],Bit) -> IO ()
    simulate circuit = print (sim_generic (1.0::Float) circuit)
```

Listing 5.9: Simulation function.

CREATING SUPERPOSITION VIA H As expected, the experimental setup of the for-loop over Hadamard gate is almost identical to the aforementioned experiments. The main change is in the matrix type. The type corresponding to $\frac{1}{\sqrt{2}}$ is `Cplx (RootTwo (Ratio Integer))`. To make this matrix legible $\frac{1}{\sqrt{2}}$ was defined as `invsq2` (listing 5.10). This experiment has the same requirements for decomposition as the previous experiment with the Y gate. However, this simulation could not run with the simulation of the decomposed circuit, because of the assertion of the auxiliary qubit.

```
   -- * From a matrix
2
   invsq2 :: Cplx (RootTwo (Ratio Integer))
4  invsq2 = Cplx (RootTwo 0 (1 % 2)) 0    :: Cplx (RootTwo (Ratio Integer))

6  mymatrix :: Matrix Eight Eight (Cplx (RootTwo (Ratio Integer)))
   mymatrix = matrix [[1, 0, 0, 0, 0, 0, 0, 0],
8                      [ 0, 1, 0, 0, 0, 0, 0, 0],
                       [ 0, 0, invsq2, invsq2, 0, 0, 0, 0],
10                     [ 0, 0, invsq2, -invsq2, 0, 0, 0, 0],
                       [ 0, 0, 0, 0, 1, 0, 0, 0],
12                     [ 0, 0, 0, 0, 0, 1, 0, 0],
                       [ 0, 0, 0, 0, 0, 0, invsq2, invsq2],
14                     [ 0, 0, 0, 0, 0, 0, invsq2, -invsq2]]


16
   synthesized = exact_synthesis mymatrix
18
   circuit ::([Qubit], Qubit)-> Circ ([Qubit], Qubit)
20 circuit ([a, b], c) = do
     synthesized [a,b,c]
22   return ([a,b],c)
```

Listing 5.10: Quipper `exact_synthesis` implementation of matrix H.

Remark 5.1. The authors of Quipper know that, in a real quantum computer, when a qubit concludes with an assertion there is no way to verify the assertion. So, they admit the best option is to measure the qubit and throw an error when the assertion is incorrect. Although they also acknowledge that a quantum state could be split and the program could throw an error if the state existed in the incorrect state with probability different of zero, the authors chose not to apply this because of the errors arising from rounding (Silk, 2016).

Consequently, the simulation was limited to the initial circuit. The simulation of quantamorphism `mqfor` H ran with the same inputs used in the first quantamorphism `qfor` X gate.

FOLD QUANTAMORPHISM OVER XOR GATE   The experiment in the quantamorphism of XOR gate does not increase the superposition of the qubits. In fact, it is entirely classical. The changes from this experiment compared to `qfor` X are mainly limited to the number of qubits. This example exhibits 4 control qubits and 1 target qubit. This (relatively small) increase in the number of qubits results in an exponential growth of the matrix size. Consequently, defining a new type `thirty_two` was required. Due to its size, the Quipper implementation of this matrix is deferred to appendix B.11.
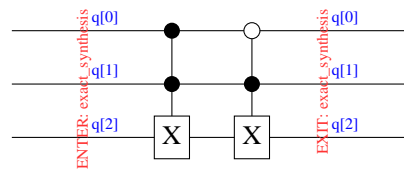
Figure 5.2.: Circuit from quantamorphism `qfor` X working with 2 qubits as controls.

Despite being completely classical, this circuit needed decomposition (similar to the second example of quantamorphism `qfor` X gate). The simulation corresponding to the decomposed circuit did not run either.

Finally, is also worth mentioning that the function to print circuits and circuit descriptions can be found in listing 5.11 and 5.12.

```
1    print_generic Preview (circuit_function my_oracle)
```

Listing 5.11: Function to print pdf of original circuit.

```
print_generic ASCII (circuit_function my_oracle)
```

Listing 5.12: Function to print description of the circuit.

Having made the implementation more clear, we proceed to the analysis of the results.

FOR-LOOP QUANTAMORPHISM OVER X GATE   In the first case of a `qfor` over X gate, the implementation gives 3 outputs:

- The circuit;

- The description of the circuit;

- And the simulation.

The circuits can be found in figure 5.2. This shows two Toffoli gates, where the second Toffoli gate has a negated control (the white dot).

The description of this circuit was later used to translate this circuit to QISKit. It can be found in listing 5.13.

```
1  Inputs: none
   QInit1(0)
3  QInit1(1)
   QInit1(2)
5  Comment[""](0:"|1>[0]", 1:"|1>[1]", 2:"|1>")
   Comment["before oracle"]()
7  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
```

Initial State Preparation			Output Measure			Probability (%)
q_0	q_1	q_2	q_0	q_1	q_2	
0	0	0	0	0	0	100
0	0	1	0	0	1	100
0	0	Hadamard	0	0	1	50
			0	0	0	50
1	1	0	1	1	1	100
1	1	1	1	1	0	100
1	1	Hadamard	1	1	1	50
			1	1	0	50
Hadamard	Hadamard	0	1	1	1	25
			0	1	1	25
			1	0	0	25
			0	0	0	25
Hadamard	Hadamard	1	1	0	1	25
			0	0	1	25
			1	1	0	25
			0	1	0	25
Hadamard	Hadamard	Hadamard	1	1	1	12.499999
			0	1	1	12.499999
			1	0	1	12.499999
			0	0	1	12.499999
			1	1	0	12.5
			0	1	0	12.5
			1	0	0	12.5
			0	0	0	12.5

Table 5.4.: Results of Quipper simulation in the circuit from matrix `qfor` X working with 2 qubits as control.

```
   QGate["X"](2) with controls=[+0, +1]
 9 QGate["X"](2) with controls=[-0, +1]
   Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
11 Comment["after oracle"]()
   QMeas(2)
13 QMeas(1)
   QMeas(0)
15 Outputs: 0:Cbit, 1:Cbit, 2:Cbit
```

Listing 5.13: Description of circuit of the `qfor` of X gate working with 2 qubits as controls.

The simulation outputs are expressed in terms of "True" or "False". "True" means to have the qubit in state 1 and "False" means to have the qubits in state 0. In table 5.4 the simulation results are translated to the conventional notation in this dissertation.

The second case of X gate quantamorphism has a second group of outputs corresponding to the decomposition.
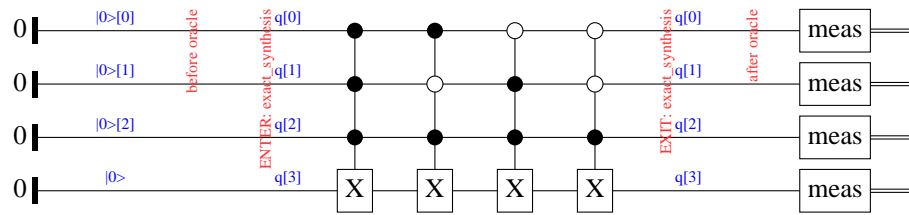
Figure 5.3.: Original circuit generated by `qfor` X working with 3 qubits as controls.
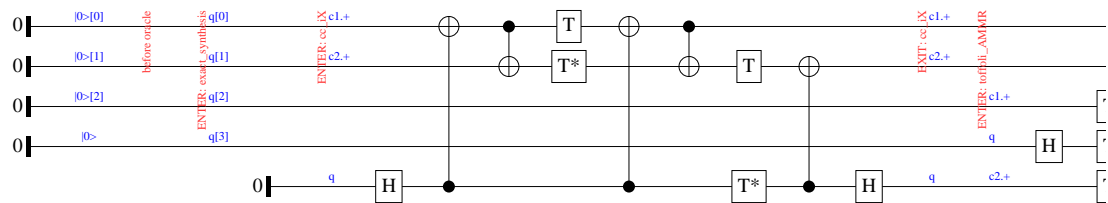
Figure 5.4.: Decomposed circuit generated by `qfor` X working with 3 qubits as controls (section 1).

The circuits generated by this matrix are in figure 5.3 (original circuit) and figures 5.4 to 5.12 (decomposed circuit).

The initial circuit does not use standard gates but its analysis is still clear, the top qubits are the controls and when the output is 111, 101, 011, or 001(corresponding to numbers 7, 5, 3 and 1) activate the X gate in the target qubit. The decomposed circuit behaves in the same way (this can be verified in table 5.5 where the outcomes of the simulation of both circuits are portrayed) but has a lot more gates, an additionally auxiliary qubit, and is difficult to read and to analyse.

Since the results of the simulation and of the original circuit are the same, in the experiments below the second is not given.

Moreover, this experiment also returns the description of the circuits. The description of the original circuit can be shown in here (listing 5.14) but the description of the decomposed circuit is in the appendix (listing C.10).

```
1  Inputs: none
```
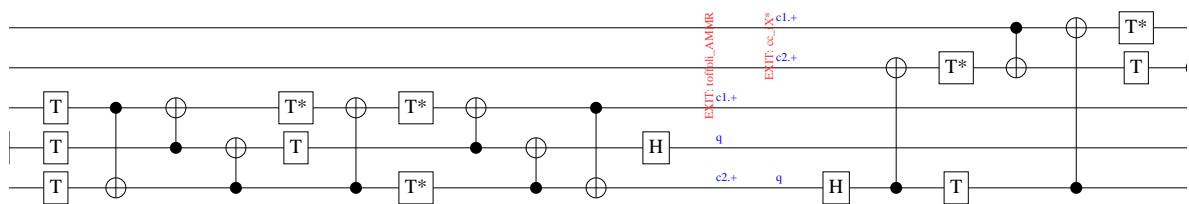

Figure 5.5.: Decomposed circuit generated by `qfor` X working with 3 qubits as controls (section 2).
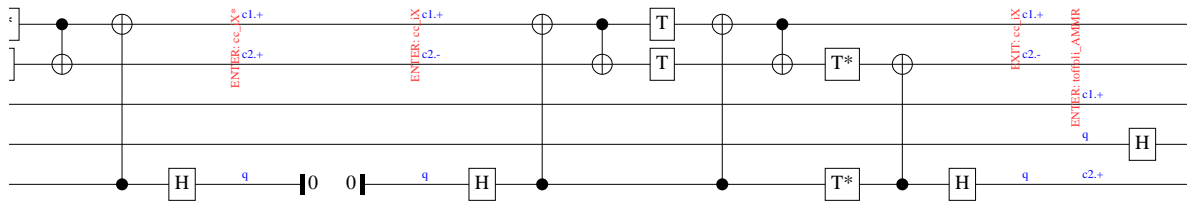
Figure 5.6.: Decomposed circuit generated by qfor X working with 3 qubits as controls (section 3).
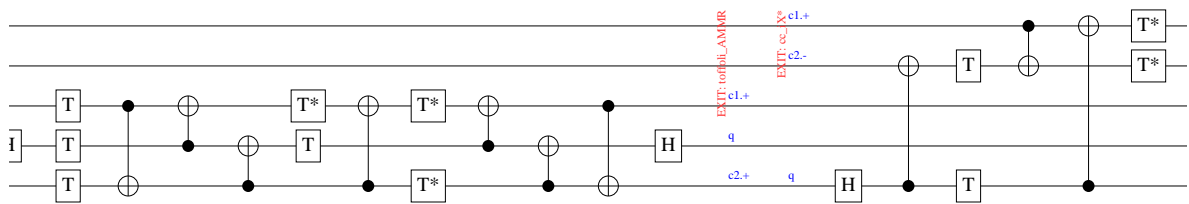
Figure 5.7.: Decomposed circuit generated by qfor X working with 3 qubits as controls (section 4).
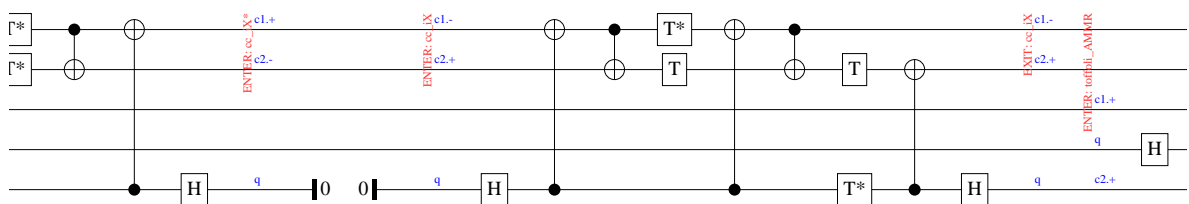
Figure 5.8.: Decomposed circuit generated by qfor X working with 3 qubits as controls (section 5).
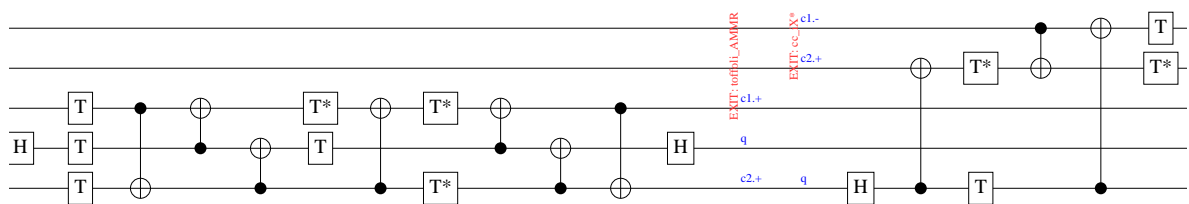
Figure 5.9.: Decomposed circuit generated by qfor X working with 3 qubits as controls (section 6).
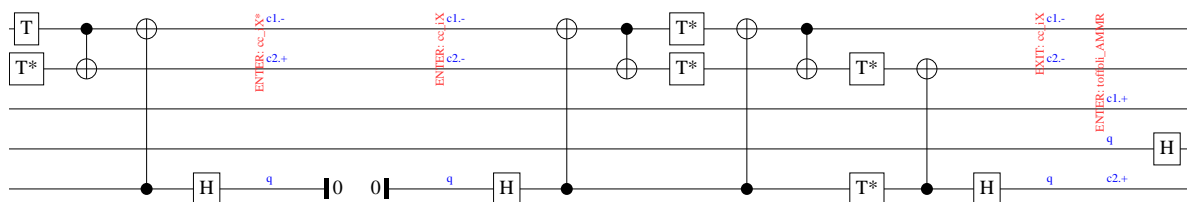
Figure 5.10.: Decomposed circuit generated by qfor X working with 3 qubits as controls (section 7).

Circuit	Initial State Preparation				Output Measure				Probability (%)
	q_0	q_1	q_2	q_3	q_0	q_1	q_2	q_3	
O	0	0	0	0	0	0	0	0	100
D	0	0	0	0	0	0	0	0	100
O	0	0	0	1	0	0	0	1	100
D	0	0	0	1	0	0	0	1	100
O	1	1	1	0	1	1	1	1	100
D	1	1	1	0	1	1	1	1	100
O	1	1	1	1	1	1	1	0	100
D	1	1	1	1	1	1	1	0	100
O	Hadamard	Hadamard	Hadamard	0	1	1	1	1	12.499999
					0	1	1	1	12.499999
					1	0	1	1	12.499999
					0	0	1	1	12.499999
					1	1	0	0	12.5
					0	1	0	0	12.5
					1	0	0	0	12.5
					0	0	0	0	12.5
D	Hadamard	Hadamard	Hadamard	0	1	1	1	1	12.499999
					0	1	1	1	12.499999
					1	0	1	1	12.499999
					0	0	1	1	12.499999
					1	1	0	0	12.5
					0	1	0	0	12.5
					1	0	0	0	12.5
					0	0	0	0	12.5
O	Hadamard	Hadamard	Hadamard	1	1	1	0	1	12.499999
					0	1	0	1	12.499999
					1	0	0	1	12.499999
					0	0	0	1	12.499999
					1	1	1	0	12.5
					0	1	1	0	12.5
					1	0	1	0	12.5
					0	0	1	0	12.5
D	Hadamard	Hadamard	Hadamard	1	1	1	0	1	12.499999
					0	1	0	1	12.499999
					1	0	0	1	12.499999
					0	0	0	1	12.499999
					1	1	1	0	12.5
					0	1	1	0	12.5
					1	0	1	0	12.5
					0	0	1	0	12.5

Table 5.5.: Results of Quipper simulation both circuits of `qfor` with 3 qubits. The letter O stands for Original circuit and the letter D stands for the Decomposed circuit.
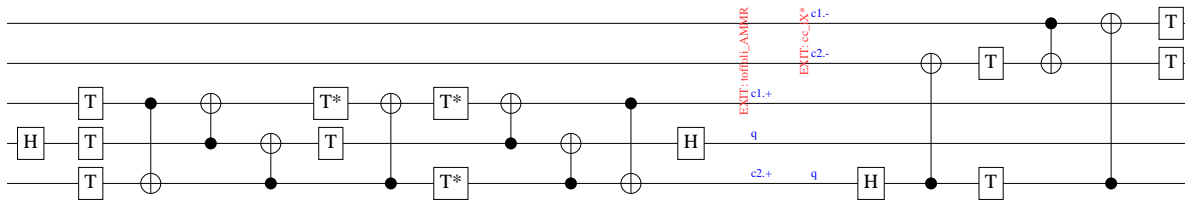
Figure 5.11.: Decomposed circuit generated by `qfor` X working with 3 qubits as controls (section 8).
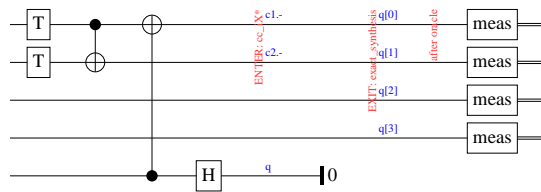
Figure 5.12.: Decomposed circuit generated by `qfor` X working with 3 qubits as controls (section 9).

```
    QInit0(0)
 3  QInit0(1)
    QInit0(2)
 5  QInit0(3)
    Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|0>[2]", 3:"|0>")
 7  Comment["before oracle"]()
    Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]")
 9  QGate["X"](3) with controls=[+0, +1, +2]
    QGate["X"](3) with controls=[+0, -1, +2]
11  QGate["X"](3) with controls=[-0, +1, +2]
    QGate["X"](3) with controls=[-0, -1, +2]
13  Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]")
    Comment["after oracle"]()
15  QMeas(3)
    QMeas(2)
17  QMeas(1)
    QMeas(0)
19  Outputs: 0:Cbit, 1:Cbit, 2:Cbit, 3:Cbit
```

Listing 5.14: Description of circuit of the `qfor` X gate working with 3 qubits as controls.

FOR-LOOP QUANTAMORPHISM OVER Y GATE   In the quantamorphism for Y the required results were:

- The circuit;

- The description of the circuit;
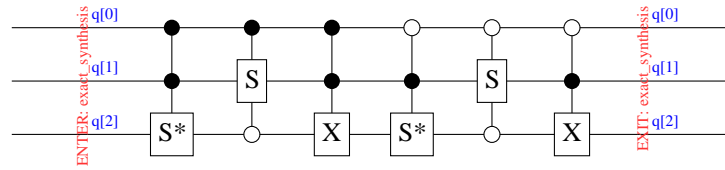
Figure 5.13.: Circuit from for-loop quantamorphism over Y gate.
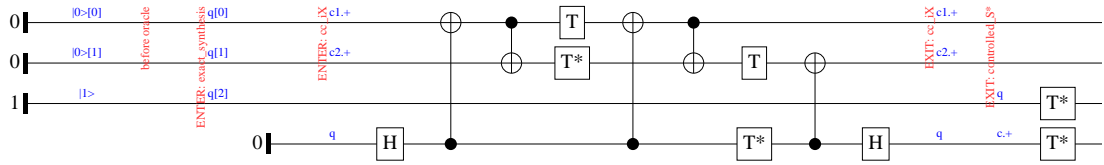
Figure 5.14.: Decomposed circuit of for-loop quantamorphism over Y gate (section 1).

- The simulation of the circuit;

- The decomposed circuit;

- The description of the decomposed circuit;

- And the simulation of the decomposed circuit.

The initial circuit generated from the matrix is in figure 5.13. This circuit utilises the phase rotation gate S and is much more complex to analyse. Nevertheless, it is clear that the target qubit will only be altered when the controls represent odd numbers.

The description of the circuit is in appendix C.23 [4], and the simulation results can be found in table 5.6.

The gates of the type control-control-S do not have a trivial implementation on QISKit. The result of this, was the need for decomposition (figures 5.14 to 5.21).

Obviously, the decomposition increases the number of gates considerably and adds the additional ancilla.
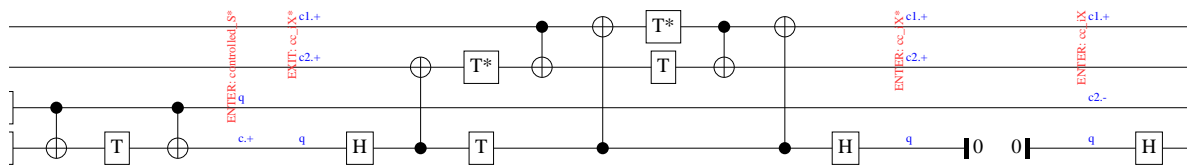
Figure 5.15.: Decomposed circuit of for-loop quantamorphism over Y gate (section 2).

---

[4]this description was not translated to QISKit.

Initial State Preparation			Output Measure			Probability (%)
q_0	q_1	q_2	q_0	q_1	q_2	
0	0	0	0	0	0	100
0	0	1	0	0	1	100
0	0	Hadamard	0	0	1	50
			0	0	0	50
1	1	0	1	1	1	100
1	1	1	1	1	0	100
1	1	Hadamard	1	1	1	50
			1	1	0	50
Hadamard	Hadamard	0	1	1	1	25
			0	1	1	25
			1	0	0	25
			0	0	0	25
Hadamard	Hadamard	1	1	0	1	25
			0	0	1	25
			1	1	0	25
			0	1	0	25
Hadamard	Hadamard	Hadamard	1	1	1	12.499999
			0	1	1	12.499999
			1	0	1	12.499999
			0	0	1	12.499999
			1	1	0	12.5
			0	1	0	12.5
			0	1	1	12.5
			0	0	0	12.5

Table 5.6.: Results of Quipper simulation in the circuit from for-loop quantamorphism over Y gate. These values in the original circuit match the decomposed circuit.
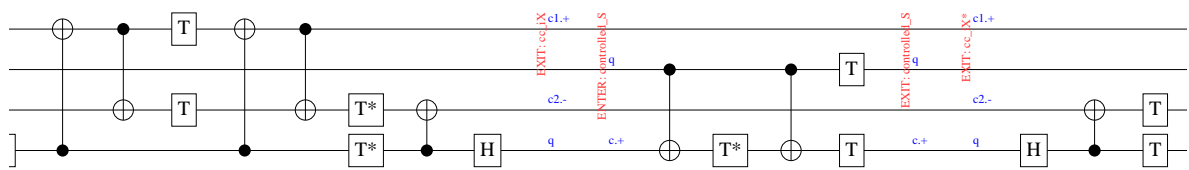
Figure 5.16.: Decomposed circuit of for-loop quantamorphism over Y gate (section 3).
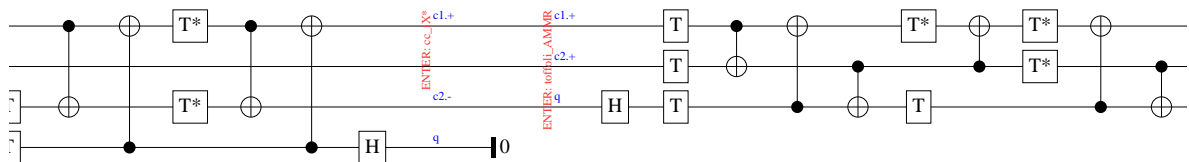
Figure 5.17.: Decomposed circuit of for-loop quantamorphism over Y gate (section 4).
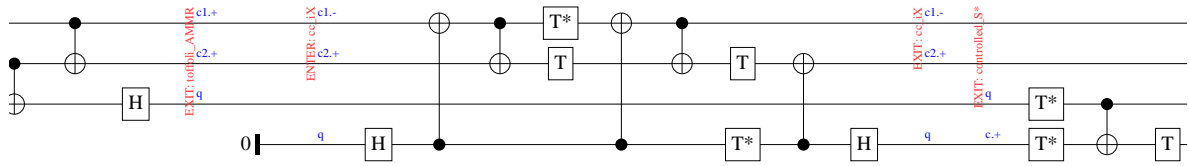
Figure 5.18.: Decomposed circuit of for-loop quantamorphism over Y gate (section 5).
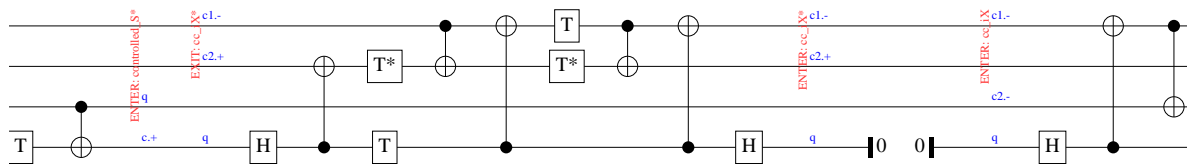
Figure 5.19.: Decomposed circuit of for-loop quantamorphism over Y gate (section 6).

Figure 5.20.: Decomposed circuit of for-loop quantamorphism over Y gate (section 7).

Figure 5.21.: Decomposed circuit of for-loop quantamorphism over Y gate (section 8).

Figure 5.22.: Circuit from for-loop quantamorphism over Hadamard gate .
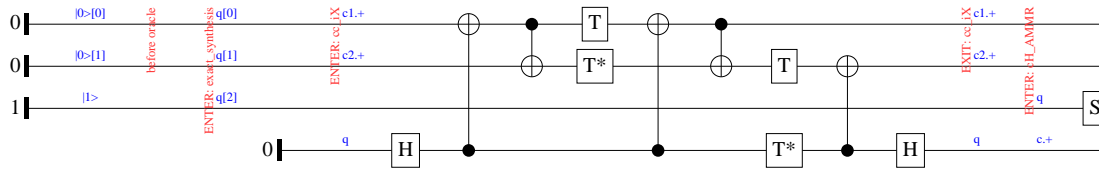
Figure 5.23.: Decomposed circuit of for-loop quantamorphism over Hadamard gate (section 1).

The simulations of the circuit of for-loop quantamorphism over Y gate result in simulations with the exact same output values as the ones presented by the previous circuit (table 5.6).

Finally, the description of the decomposed circuit can be found in appendix C.24.

CREATING SUPERPOSITION VIA H The required results for the quantamorphism of Hadamard gate are the same as the quantamorphism of Y gate, in this case, the last objective end up being unachievable [5].

The other results were obtained as expected. Figure 5.22 shows the circuit generated by the matrix. As anticipated this circuit adds superposition in a target qubit initial set to 0 if the control qubits represent the numbers 1 or 3. Simulation test to this circuits verifies this observation (table 5.7).   The description of this circuits was not translated to QISKit because of the control-control-Hadamard gate, but can be found in appendix C.34. This conditional gate also called for decomposition, see figures 5.23 to 5.26.
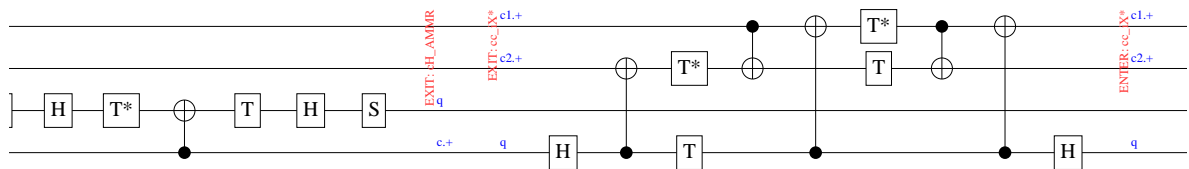
Figure 5.24.: Decomposed circuit of for-loop quantamorphism over Hadamard gate (section 2).

---

[5] recall remark 5.1.

Initial State Preparation			Output Measure			Probability (%)
q_0	q_1	q_2	q_0	q_1	q_2	
0	0	0	0	0	0	100
0	0	1	0	0	1	100
0	0	Hadamard	0	0	1	50
			0	0	0	50
1	1	0	1	1	1	50
			1	1	0	50
1	1	1	1	1	1	50
			1	1	0	50
1	1	Hadamard after 1	1	1	1	100
1	1	Hadamard after 0	1	1	0	100
Hadamard	Hadamard	0	1	1	1	12.499999
			0	1	1	12.499999
			1	1	0	12.499999
			0	1	0	12.499999
			1	0	0	25
			0	0	0	25
Hadamard	Hadamard	1	1	1	1	12.499999
			0	1	1	12.499999
			1	0	1	24.999999
			0	0	1	24.999999
			1	1	0	12.500003
			0	1	0	12.500003
Hadamard	Hadamard	Hadamard after 0	1	0	1	12.5
			0	0	1	12.5
			1	1	0	24.999999
			0	1	0	24.999999
			1	0	0	12.500001
			0	0	0	12.500001
Hadamard	Hadamard	Hadamard after 1	1	1	1	24.999999
			0	1	1	24.999999
			1	0	1	12.500001
			0	0	1	12.500001
			1	0	0	12.5
			0	0	0	12.5

Table 5.7.: Results of Quipper simulation in the circuit from for-loop quantamorphism over Hadamard gate .
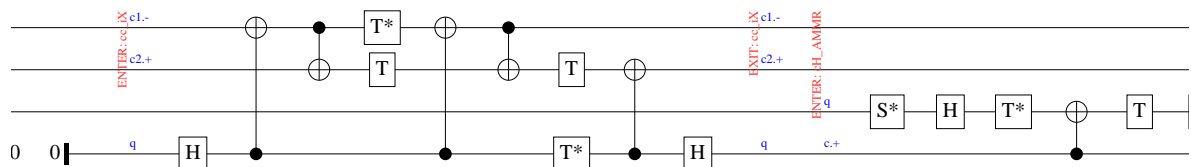
Figure 5.25.: Decomposed circuit of for-loop quantamorphism over Hadamard gate (section 3).
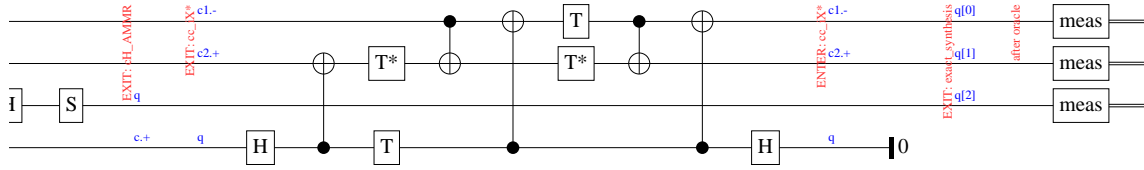
Figure 5.26.: Decomposed circuit of for-loop quantamorphism over Hadamard gate (section 4).
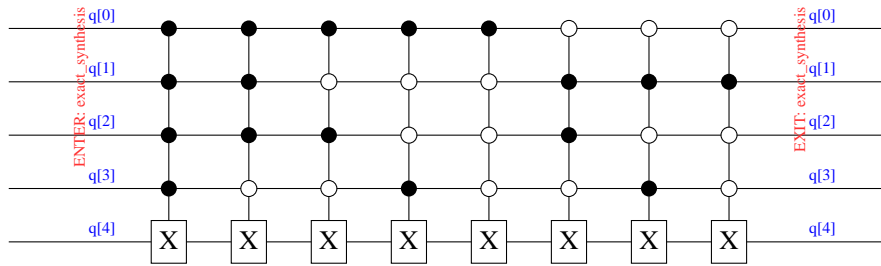
Figure 5.27.: Circuit from quantamorphism foldr XOR .

Identical to the former example, this decomposition adds a large number of gates (mainly $CNOT$, Hadamard gate and rotation gates of the type T and $T^\dagger$) and an auxiliary qubit. The description of this circuits is in appendix C.35.

FOLDR QUANTAMORPHISM OVER XOR GATE   Like the previous example, this circuit fails to test the simulation of its decomposition.

On the other hand, this circuit differs from the previous due to use 5 qubits - 4 control qubits (q_0, q_1, q_2 and q_3) and 1 target qubits (q_4), its aspect is notably distinct from what has been seen so far (figure 5.27). The simulation of the circuits generated is in table 5.8.

The control-control-control-control-X gate needs undoubtedly decomposition. Such decomposition generates the circuit in figure 5.28 to 5.53. The description of this circuit is in appendix C.48.
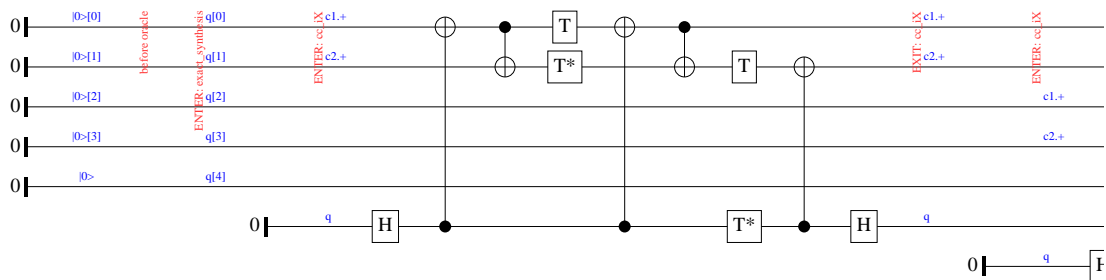
Figure 5.28.: Decomposed circuit from fold quantamorphism over XOR gate (section 1).

Initial State Preparation					Output Measure					Probability (%)
q_0	q_1	q_2	q_3	q_4	q_0	q_1	q_2	q_3	q_4	
0	0	0	0	0	0	0	0	0	0	100
0	0	0	0	1	0	0	0	0	1	100
1	1	1	1	0	1	1	1	1	1	100
1	1	1	1	1	1	1	1	1	0	100
					1	1	1	1	1	6.249999
					0	1	0	1	1	6.249998
					1	0	0	1	1	6.249998
					1	1	1	0	1	6.2499996
					0	1	1	0	1	6.2499996
					1	0	1	0	1	6.2499993
					0	1	0	0	1	6.250001
H	H	H	H	0	1	0	0	0	1	6.250001
					0	1	1	1	0	6.249999
					1	0	1	1	0	6.2499985
					0	0	1	1	0	6.2499985
					1	1	0	1	0	6.25
					0	0	0	1	0	6.25
					0	0	1	0	0	6.2500015
					1	1	0	0	0	6.250001
					0	0	0	0	0	6.250001
					0	1	1	1	1	6.2499985
					1	0	1	1	1	6.249998
					0	0	1	1	1	6.249998
					1	1	0	1	1	6.2499996
					0	0	0	1	1	6.2499996
					0	0	1	0	1	6.250001
					1	1	0	0	1	6.25
H	H	H	H	1	0	0	0	0	1	6.25
					1	1	1	1	0	6.2499993
					0	1	0	1	0	6.2499985
					1	0	0	1	0	6.2499985
					1	1	1	0	0	6.25
					0	1	1	0	0	6.25
					1	0	1	0	0	6.2499996
					0	1	0	0	0	6.2500015
					1	0	0	0	0	6.2500015
					1	1	1	1	1	3.1249996
					0	1	1	1	1	3.1249996
					1	0	1	1	1	3.1249996
					0	0	1	1	1	3.1249996
					1	1	0	1	1	3.1249998
					0	1	0	1	1	3.1249998
					1	0	0	1	1	3.1249998
					0	0	0	1	1	3.1249998
					1	1	1	0	1	3.1249998
					0	1	1	0	1	3.1249998
					1	0	1	0	1	3.1249998
					0	0	1	0	1	3.1249998
					1	1	0	0	1	3.125
					0	1	0	0	1	3.125
					1	0	0	0	1	3.125
H	H	H	H	H	0	0	0	0	1	3.125
					1	1	1	1	0	3.1249996
					0	1	1	1	0	3.1249996
					1	0	1	1	0	3.1249996
					0	0	1	1	0	3.1249996
					1	1	0	1	0	3.1249998
					0	1	0	1	0	3.1249998
					1	0	0	1	0	3.1249998
					0	0	0	1	0	3.1249998
					1	1	1	0	0	3.1249998
					0	1	1	0	0	3.1249998
					1	0	1	0	0	3.1249998
					0	0	1	0	0	3.1249998
					1	1	0	0	0	3.125
					0	1	0	0	0	3.125
					1	0	0	0	0	3.125
					0	0	0	0	0	3.125

Table 5.8.: Results of Quipper simulation in the circuit from quantamorphism foldr XOR .
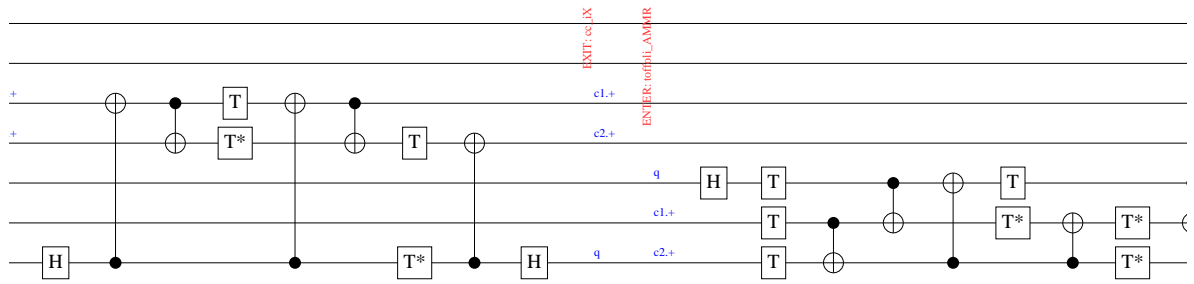
Figure 5.29.: Decomposed circuit from fold quantamorphism over XOR gate (section 2).
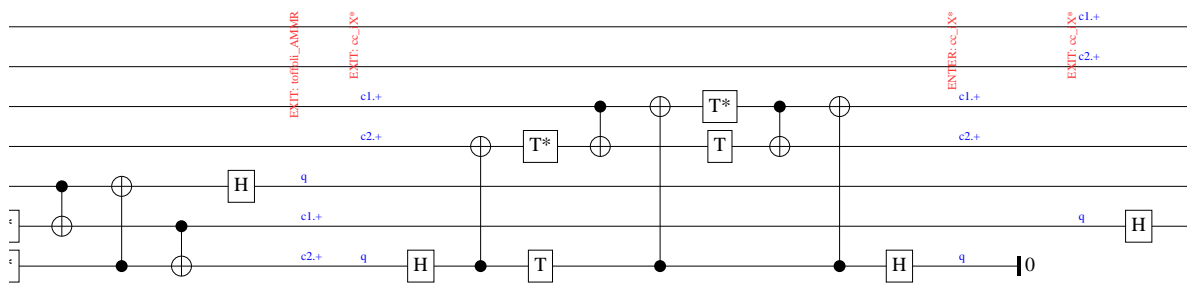
Figure 5.30.: Decomposed circuit from fold quantamorphism over XOR gate (section 3).
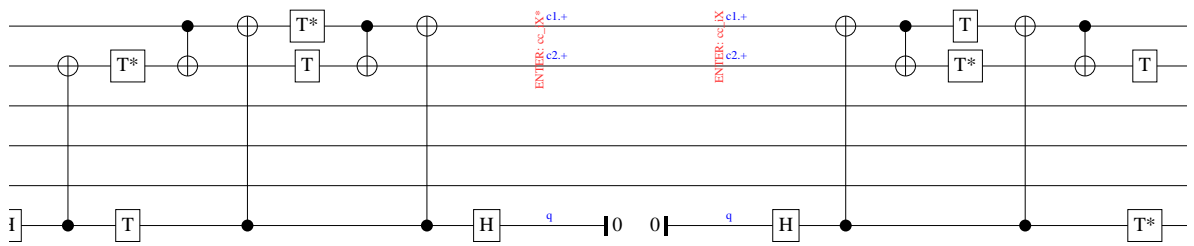
Figure 5.31.: Decomposed circuit from fold quantamorphism over XOR gate (section 4).
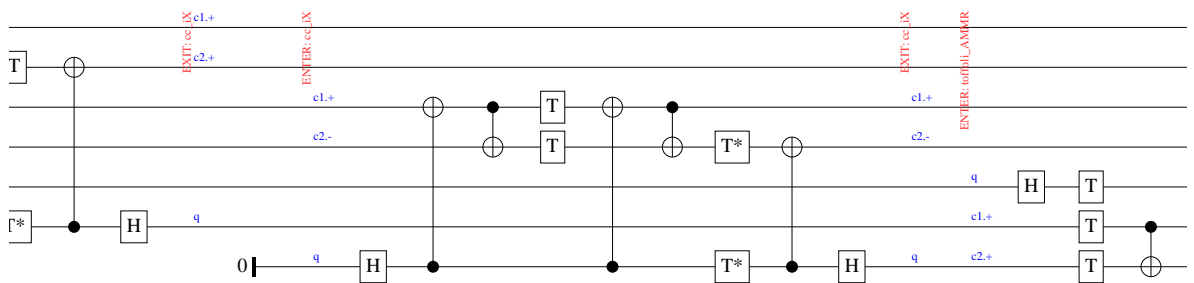
Figure 5.32.: Decomposed circuit from fold quantamorphism over XOR gate (section 5).
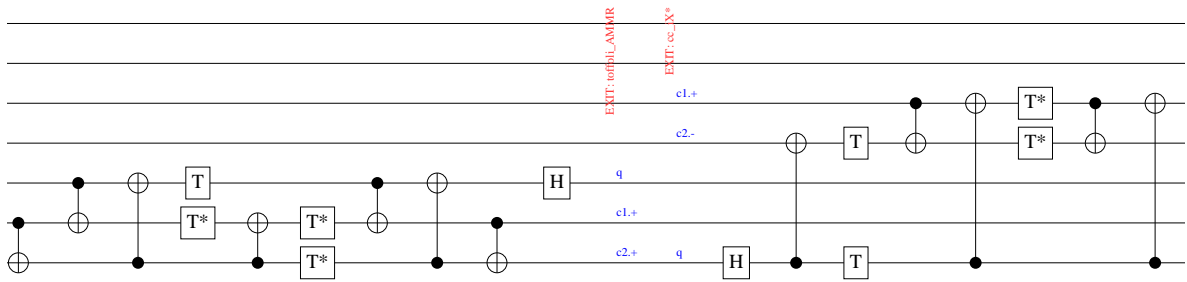
Figure 5.33.: Decomposed circuit from fold quantamorphism over XOR gate (section 6).
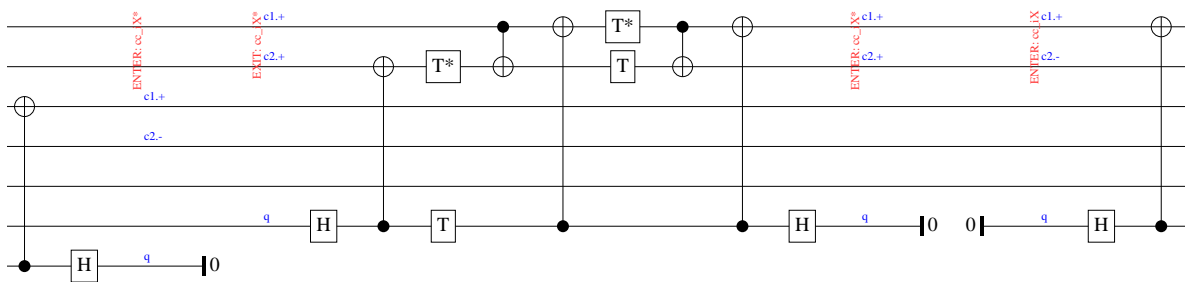
Figure 5.34.: Decomposed circuit from fold quantamorphism over XOR gate (section 7).
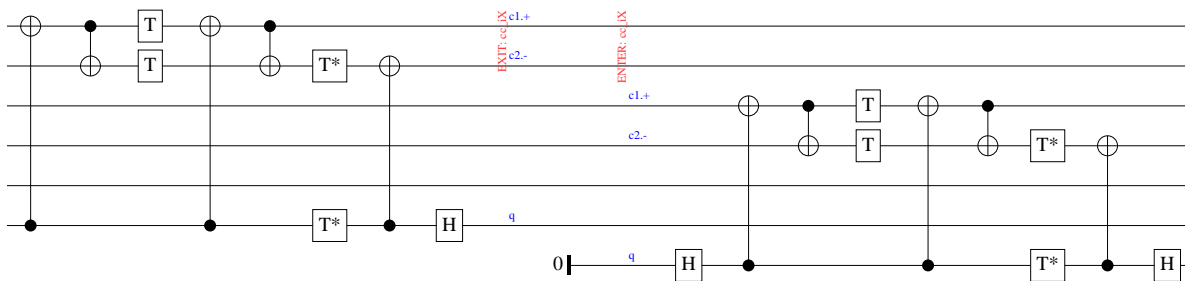
Figure 5.35.: Decomposed circuit from fold quantamorphism over XOR gate (section 8).
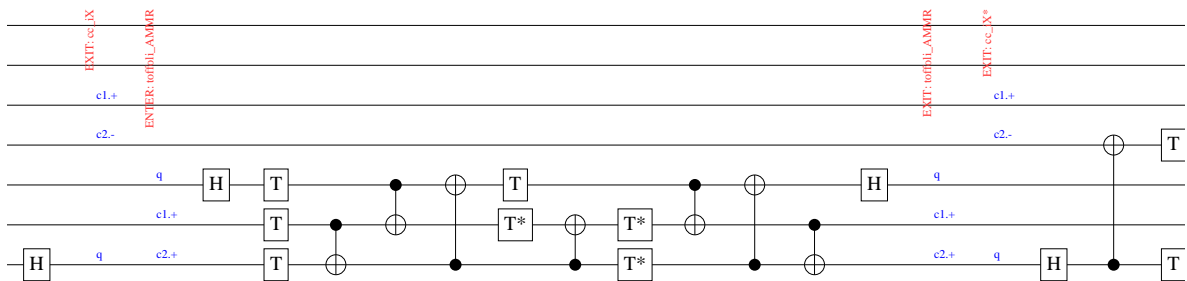
Figure 5.36.: Decomposed circuit from fold quantamorphism over XOR gate (section 9).
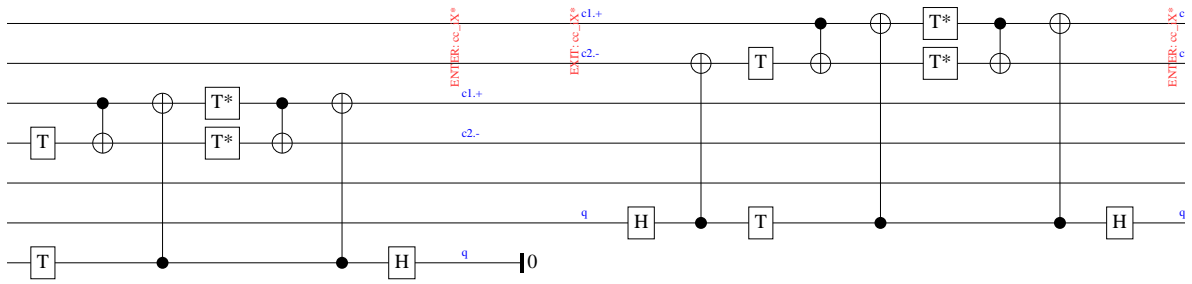
Figure 5.37.: Decomposed circuit from fold quantamorphism over XOR gate (section 10).
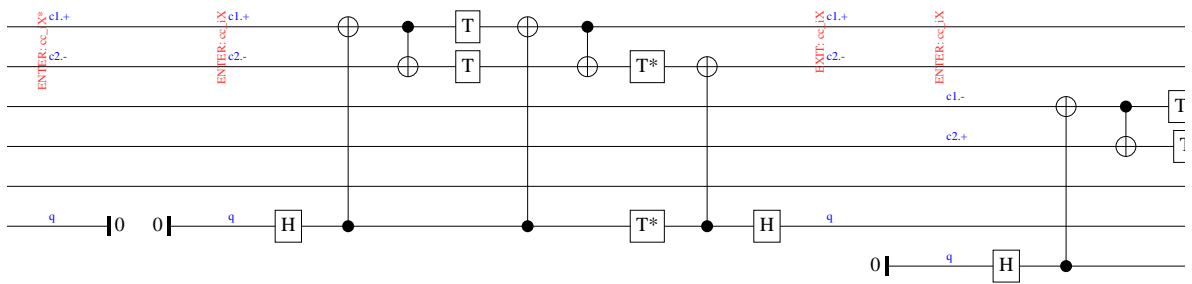
Figure 5.38.: Decomposed circuit from fold quantamorphism over XOR gate (section 11).
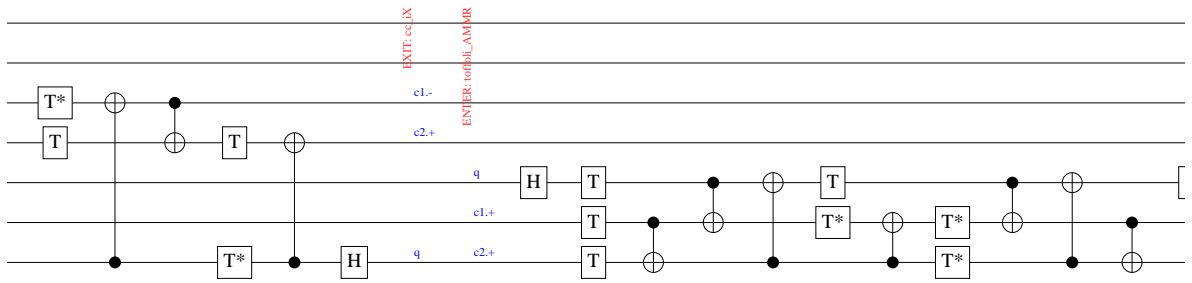
Figure 5.39.: Decomposed circuit from fold quantamorphism over XOR gate (section 12).
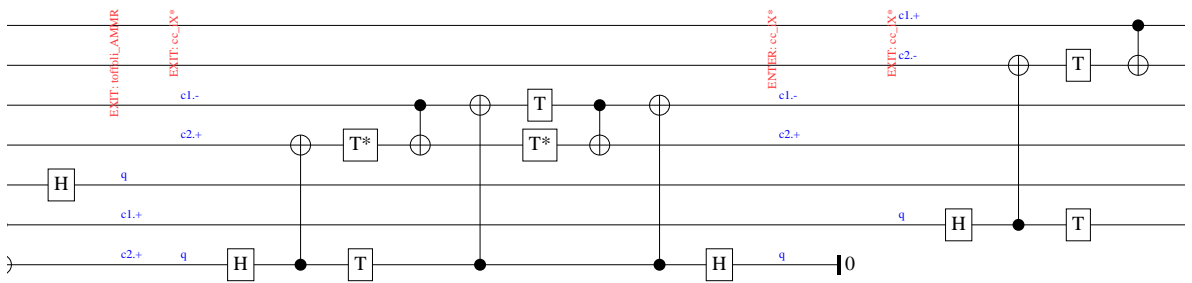
Figure 5.40.: Decomposed circuit from fold quantamorphism over XOR gate (section 13).
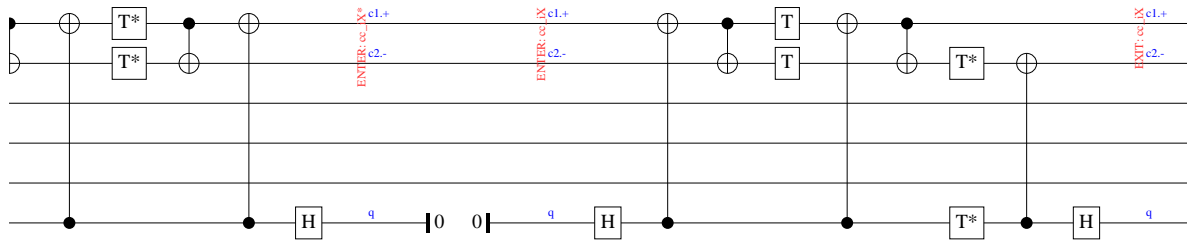
Figure 5.41.: Decomposed circuit from fold quantamorphism over XOR gate (section 14).
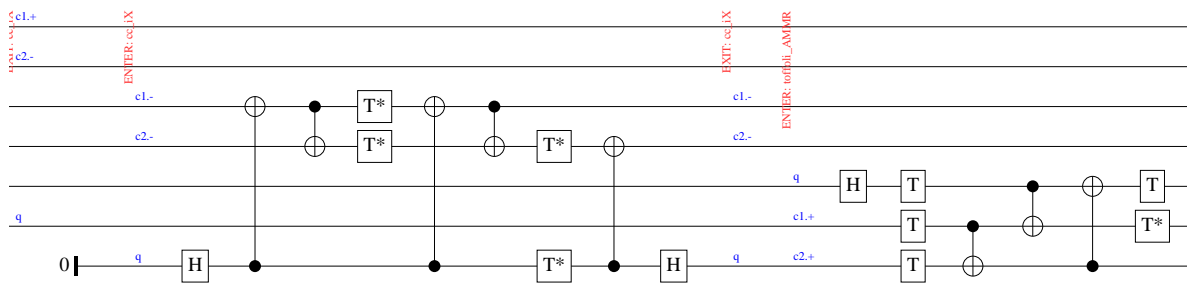
Figure 5.42.: Decomposed circuit from fold quantamorphism over XOR gate (section 15).
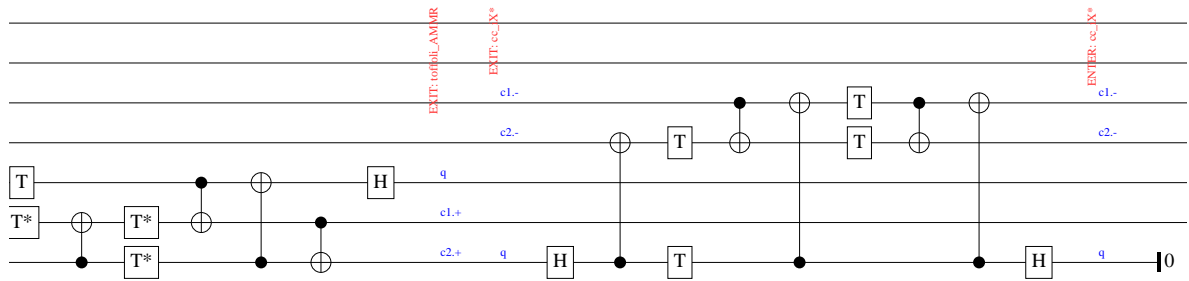
Figure 5.43.: Decomposed circuit from fold quantamorphism over XOR gate (section 16).
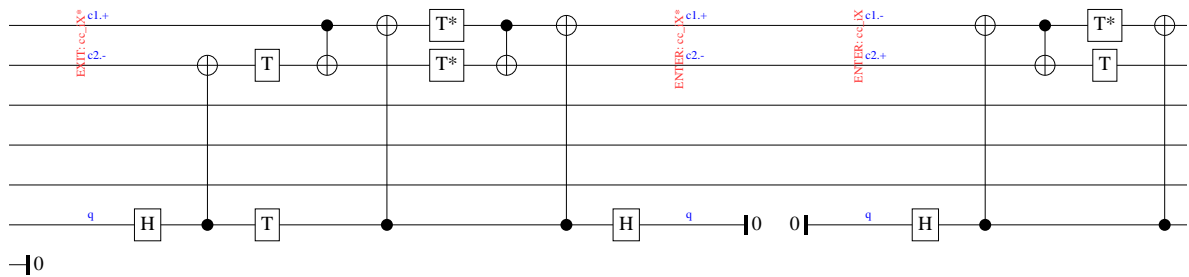
Figure 5.44.: Decomposed circuit from fold quantamorphism over XOR gate (section 17).
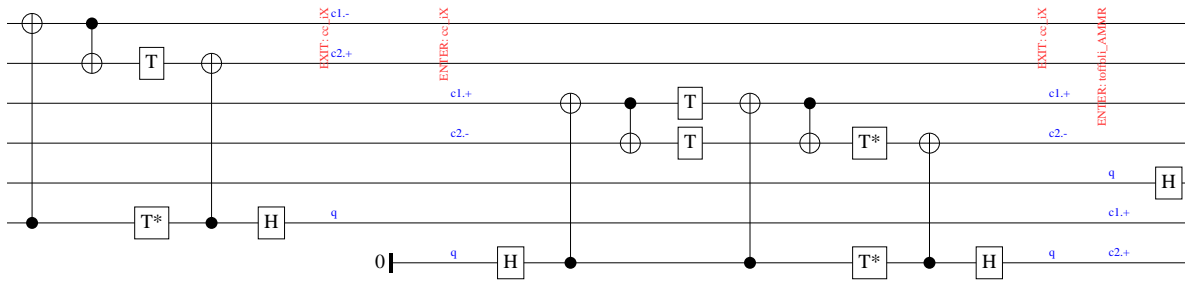
Figure 5.45.: Decomposed circuit from fold quantamorphism over XOR gate (section 18).
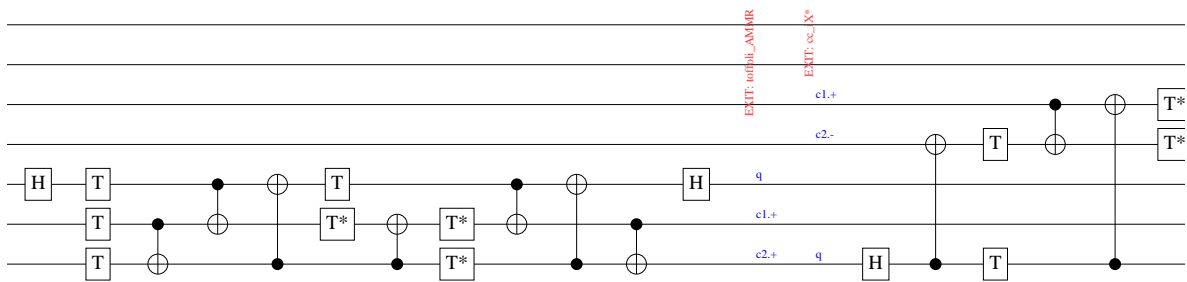
Figure 5.46.: Decomposed circuit from fold quantamorphism over XOR gate (section 19).
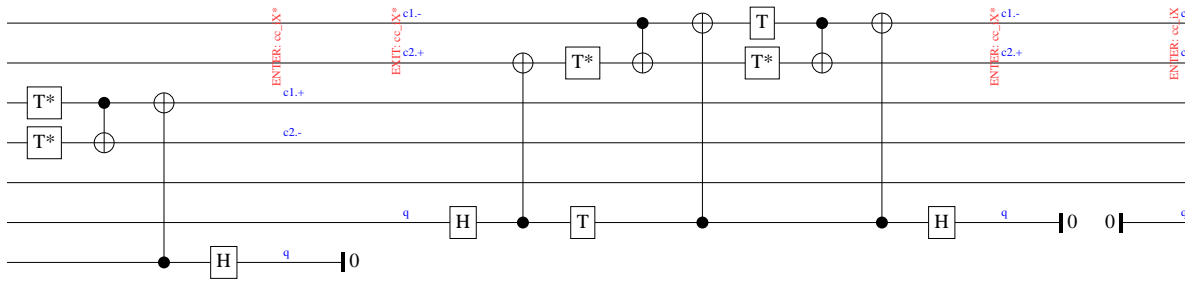
Figure 5.47.: Decomposed circuit from fold quantamorphism over XOR gate (section 20).
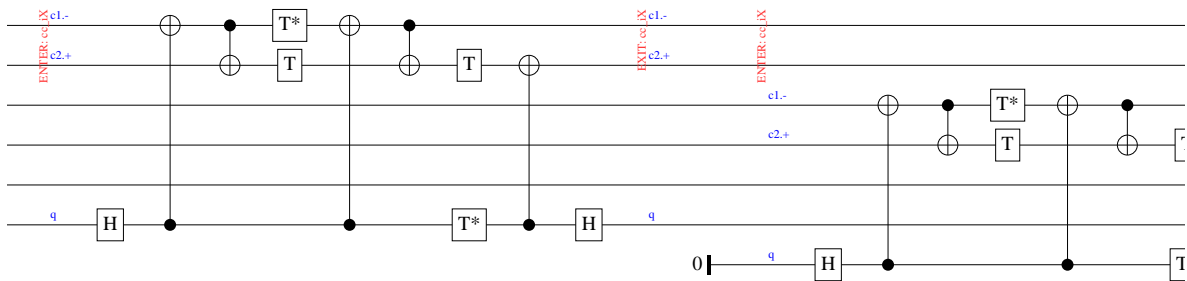
Figure 5.48.: Decomposed circuit from fold quantamorphism over XOR gate (section 21).
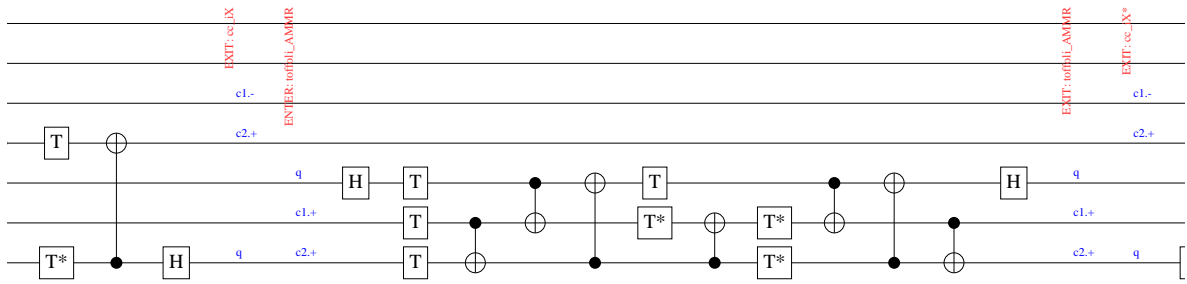
Figure 5.49.: Decomposed circuit from fold quantamorphism over XOR gate (section 22).
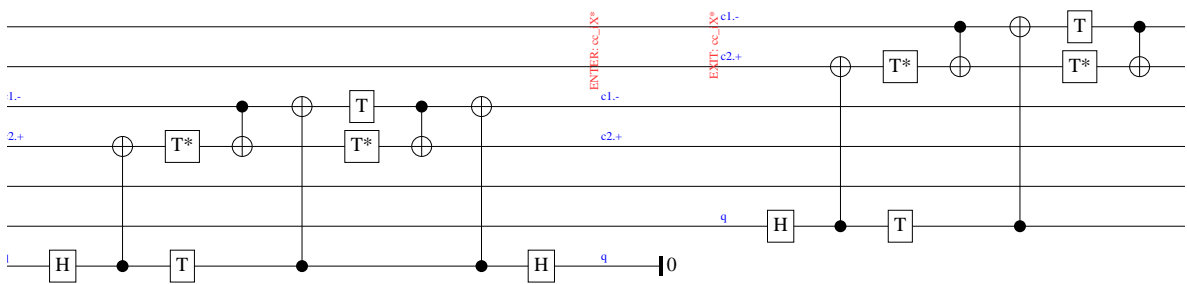
Figure 5.50.: Decomposed circuit from fold quantamorphism over XOR gate (section 23).
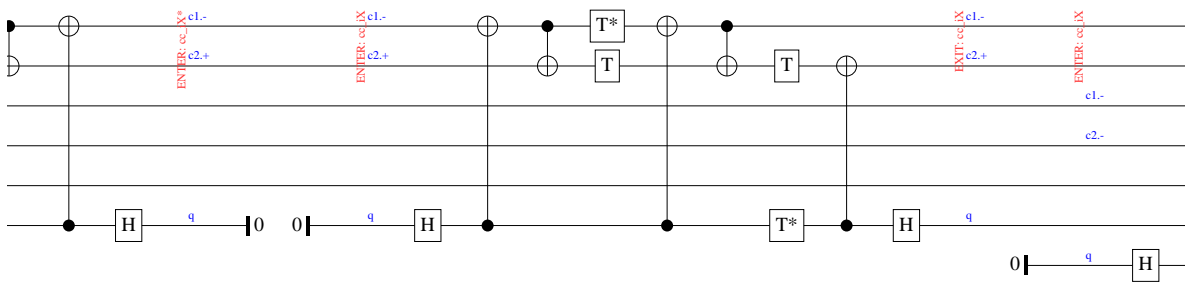
Figure 5.51.: Decomposed circuit from fold quantamorphism over XOR gate (section 24).
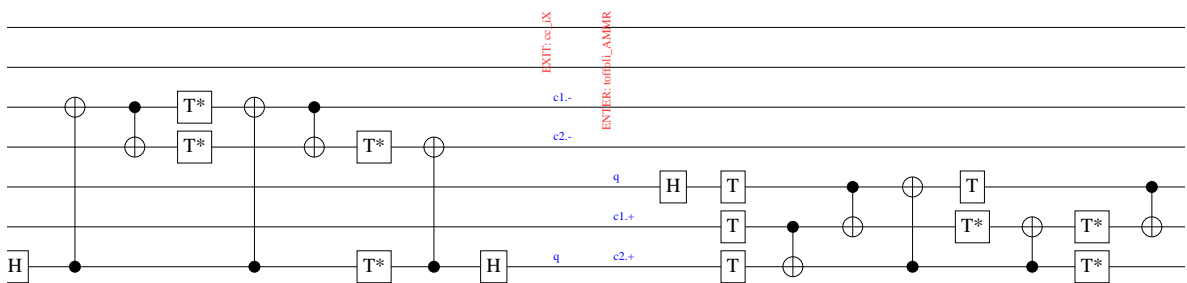
Figure 5.52.: Decomposed circuit from fold quantamorphism over XOR gate (section 25).
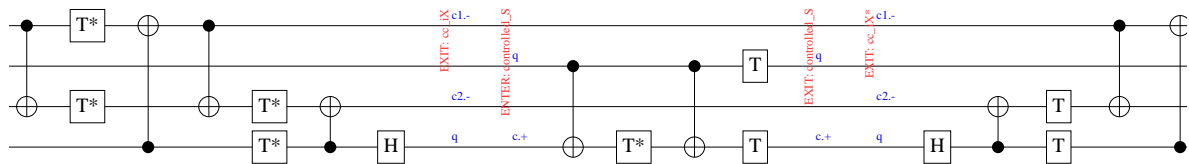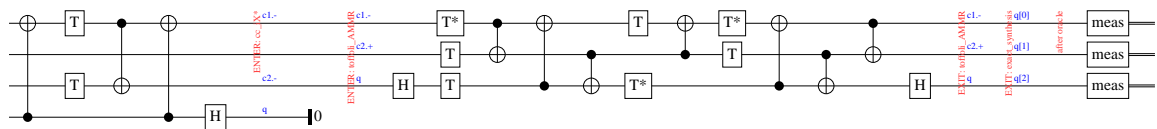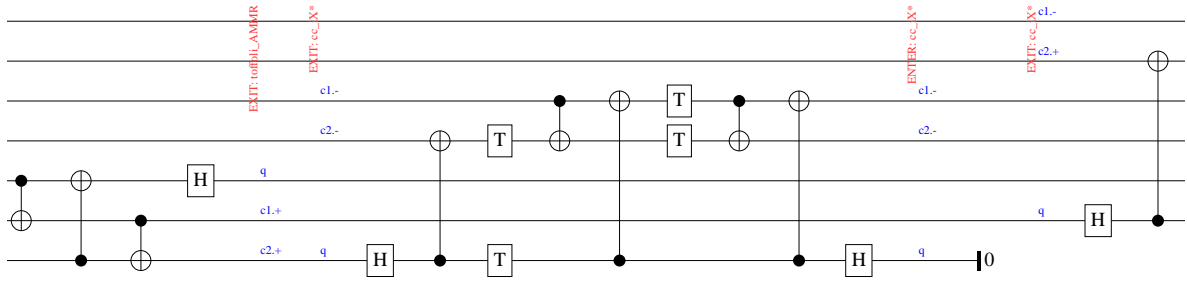
Figure 5.53.: Decomposed circuit from fold quantamorphism over XOR gate (section 26).
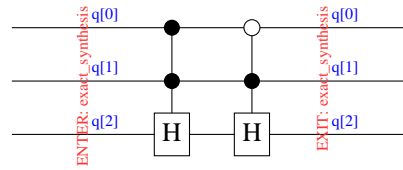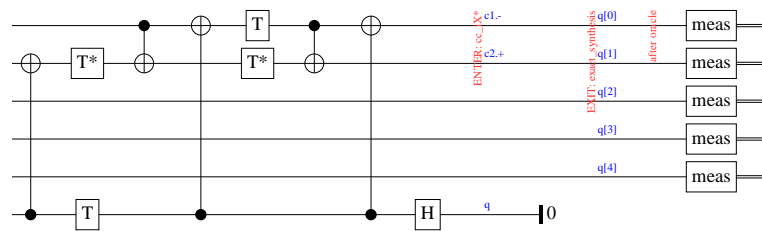

Figure 5.54.: Decomposed circuit from fold quantamorphism over XOR gate (section 27).

It is also important to notice that this circuit adds two ancillary qubits. This gives the circuit a total of 7 qubits, this was significant when applying this circuit in section 5.4.

5.4   QISKIT AND IBM Q

IBM Q Experience is a cloud-based platform that allows users from all over the world to interact with real quantum devices available from IBM Research labs. Everyone interested in working with a quantum computer can do so using the quantum composer (this helps users to create and understand quantum programs) or the QISKit interface (Quantum Information Science Kit).

The results of Quipper make the need to use QISKit clear. QISKit is a Pythons interface that lets one execute multiple circuits in devices with 5 or 16 qubits. The QISKit documentation has a collection of useful tutorials. The Jupyter Notebooks of the tutorials were adopted as guidelines for the programs presented in this chapter.

The first step in every experiment is to verify the connection and identify the machines and simulators available. It is also advisable to keep track of the number of credits available. The maximum number of credits is 15, and to recover takes 24 hours or the job has to run off the queue. The code used to accomplish this task can be found in appendices D.1 and D.2.

FOR-LOOP QUANTAMORPHISM OVER X GATE   The goal of this experiment is to implement and test circuits generated from for-loop quantamorphism over the X by Quipper (figure 5.2). Recall the first circuit which employs two Toffoli gates, one of which has a negated control.

QISKit has a function to write simple Toffoli gates (`ccx`). The implementation of negated control can be done by applying a X gate before and after the control. This corresponds to a small number of gates that can be written easily (see listing 5.15). However, a short analysis of the following circuits prompted the creation of a tool, "QuipperToQISKit.gawk" (Neri and Rodrigues, 2018). Because of this example, this tool makes the translation of Toffoli to `ccx`. Moreover, the translation tool pays attention to the case of negated control and adds the corresponding X gates.

```
1 # the circuit we want:
  qc.ccx(qr[0],qr[1],qr[2])
3 qc.x(qr[0])
  qc.ccx(qr[0],qr[1],qr[2])
5 qc.x(qr[0])
```

Listing 5.15: QISKit circuit for the matrix of the quantamorphism for X working with 2 control qubits.

In listing 5.15, `qc` stands for quantum circuit, `x` and `ccx` are quantum gates and `qr` is a quantum register[6].

One of the drawbacks of this tool is not to be programmed to update the number of qubits and bits registered. This circuit works with 3 quantum registers 3 bits registers. Note that listing 5.15 has to run before the definition of circuit seen in listing 5.16.

```
1 # create Quantum Register called "qr" with 3 qubits
  qr = qp.create_quantum_register('qr', 3)
3 # create Classical Register  called "cr" with 3 bits
  cr = qp.create_classical_register('cr', 3)
```

Listing 5.16: QISKit registers for matrix of the for-loop quantamorphism over X gate

In the first implementation of this circuit, the qubits were maintained in the default state ($|0\rangle$). As seen in the simulation of Quipper (table 5.4) the output expected is $|000\rangle$.

To reach the goal of running the program in a real device there are some steps that have to be taken first. Namely, produce the matrix that QISKit associates with the implemented circuit and simulate the result in QISKit simulation.

---

[6]The initialization of qc and qr is in appendix D.3.

These results and the main goal (running circuit in real device) can be achieved by applying the function execution[7]. The difference between the three tasks resides in the type of simulator or computer given.

Starting with the verification of matrix, the execution of the function runs in the local unitary simulator. This returns the equivalent unitary transformation of the circuit (Javadi-Abhari et al., 2018). The local unitary simulator has some restrictions, namely, it has only one shot and can not be used with measurements or resets. Thus, the measurement gates were only added later on the Jupyter Notebook.

The second goal is a local simulation, accomplished by implementing backend `local_qasm_simulator`. In this simulator, one is allowed to choose the number of shots (from 1 to 8192). This experiment ran with 1024 shots, i.e. the measures were taken 1024 times. The results of this execution can be displayed in a histogram (a bar-graphic with the percentage of getting each measure) or have the specific number of times a measure was obtained (this information can be acquired with the functions `get_count` or `get_data`).

When the circuit is compiled to `local_qasm_simulator` the system begins to do adjustments to the circuit initially defined (mainly to assure the gates used are standard). In order to record these alterations, the image and corresponding QASM [8] of the decomposed circuit were both printed.

Finally, running in a real device the first point to consider is how to select a device. If the goal is just to make a quick test, it is advisable to check which is the device with fewer jobs in the queue (the definition of a function that does exactly this can be found in the QISKit tutorial). This function was used to know which experiments to execute first.

The experiments ran in these case-studies have a considerable number of conditional gates which increased the error rate. Therefore, in order to execute these experiments with fewer errors, it is important to choose the device with the best behaviour in terms of Multiqubit gate error. Whenever possible, programs are executed on the `ibmqx4` device.

It is no surprise that running this circuit adds further modifications to it. This circuit has to be adapted to the selected device and make an effort to decrease the error rate in this specific machine. In this way, it is not only relevant to get the data about the output, but also collect the information related to the device used (like its characteristics by the time of the experiment) and the circuit generated by compiling in the selected device.

The following goals were to verify if there were ways to decrease the error rate and then test the circuit with different state preparation. The best option to decrease the error rate is to make an adaptation to the programs written in QISKit. In QISKit, the least significant bit (LSB) and the most significant bit (MSB) are not in the usual order. While most of the scientific use MSB on the rightmost position and the LSB on the leftmost, QISKit does

---

[7]The function execution can also be replaced by the function compile and run.
[8]QASM is a simple text "assembly" language for describing quantum circuits in general (Cross et al., 2017).
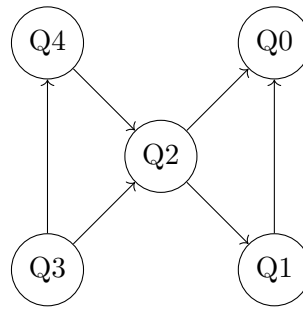
Figure 5.55.: Coupling map of IBM Q 5 Tenerife V1.x.x (`ibmqx4`). The direction of the arrows reads from control to target e.g. the qubit Q2 controls Q0 and Q1 and can be controlled by the qubits Q4 and Q3, and the Q4 has to direct influence in Q0 or Q1.

exactly the opposite. When adapting circuits to QISKit, not only are better results expected in the local unitary simulator but also fewer errors in the execution on the real device.

To clarify why lesser error rates are expected recall that in the original circuit, q_0 and q_1 control q_2 and this is the exact opposite of what IBM Q 5 Tenerife is programmed to do (see figure 5.55 where the coupling of this device is defined). Moreover, the alteration of the MSB and LSB transforms q_2 in another control and q_0 in the target and this equals the coupling map (q_0 can be controlled by q_1 and q_2). Ultimately, this corresponds to fewer gates used to decompose the circuit and fewer problems with relaxation and decoherence.

Implementing the adapted circuit corresponds to make a new input circuit where functions find and replace were used to change "2" to "0" and "0" to "2" (listing 5.17). This was the only alteration needed on the original circuit to get the circuit adapted to QISKit. All the other steps were repeated with the same inputs.

```
1  # the circuit we want:
   qc.ccx(qr[2],qr[1],qr[0])
3  qc.x(qr[2])
   qc.ccx(qr[2],qr[1],qr[0])
5  qc.x(qr[2])
```

Listing 5.17: QISKit circuit for matrix of quantamorphism for X .

Finally, tests can have diverse state preparations. In this situation, there is no interest in seeing the matrix because adding the gates to prepare the state changes it. The intent here is to test if the circuit really does what is expected or if by default tends to have more outputs in the state $|000\rangle$. These tests were made by:

- adding an X gate to all qubits the initial state was set to $q_2 = 1$, $q_1 = 1$ and $q_0 = 1$;

- adding an Hadamard gate to q_0 the initial state was set to $q_2 = 0$, $q_1 = 0$ and $q_0 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$;

- adding Hadamard gates to q_1 and q_2 the initial state was set to $q_2 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$, $q_1 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ and $q_0 = 0$;

- by entangling q_1 and q_2, the initial state was set to Bell state in q_1 and q_2 and $q_0 = 0$.

Besides the aforementioned alterations, these experiments were executed like the original circuit.

This example has yet another matrix to implement. The *qfor of X gate with 3 qubits as controls* needs decomposition and to avoid errors it is also crucial to use the tool Quipper-ToQISKit (Neri and Rodrigues, 2018). As already mentioned, this tool does not update the number of circuits in the register. This means that the differences start with the number of qubits and bits registered, in this case 5 (3 controls, 1 target and 1 auxiliary qubit).

This experiment follows the first example. After the register there is the definition of the circuit. Since this circuit has a huge number of gates its definition is shown in appendix D.18. As can be seen in this appendix the first implementation of this matrix is done with the initial default state $|000\rangle$.

After executing in a local unitary simulator, the measures are added. Note that the auxiliary qubit (q_3) does not need to be measured, thus resulting in the measure implementation seen in listing 5.18.

```
1  qc.measure(qr[3],cr[3])
   qc.measure(qr[2],cr[2])
3  qc.measure(qr[1],cr[1])
   qc.measure(qr[0],cr[0])
```

Listing 5.18: QISKit measurement of `qfor` of X gate with 3 qubits to control

The implementation of the QASM simulator, the execution in real devices and the information associated to the compilations match the previous implementation. Recall that, although this program runs with more qubits it is still possible to run it on the `ibmqx4` device.

Another implementation detail associated to this `qfor` is the adaptation to improve results. The difference here lays in the circuit implemented (appendix D.18) and the measures (listing 5.19).

```
1  qc.measure(qr[1],cr[1])
   qc.measure(qr[2],cr[2])
3  qc.measure(qr[3],cr[3])
   qc.measure(qr[4],cr[4])
```

Listing 5.19: QISKit measurement of `qfor` of X gate with 3 qubits to control adapted to QISKit

Since the differences between the direct translation and the adaptation are not significant, it was decided to continue the experiments with the circuit adapted. The experiments where the state preparation is changed were examined with the initial state at:

- $|1110\rangle$ achieved by adding an X gate to all the control qubits;

- $q_4 = 0$, $q_3 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$, $q_2 = 1$ and $q_1 = 1$, this is achieved by letting the target q_4 untouched, adding Hadamard gate to the least significant bit of the controls and adding X gates to the rest of the controls;

- $q_4 = 0$, $q_3 = 1$, $q_2 = 0$ and $q_1 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ this is achieved by adding X gate to the least significant qubit of the controls and adding a Hadamard gate to the most significant qubits.

Aside from the state preparation, nothing changes in the rest of the implementation of this example.

FOR-LOOP QUANTAMORPHISM OVER Y GATE   This example aims at running the decomposed circuit seen in section 5.3. Similar to what has been seen in a previous example with the decomposed circuit, it is clear that making the translation by hand, could result in multiple mistakes. Circuits this large call for the QuipperToQISKit tool (Neri and Rodrigues, 2018). Once again, recall that this tool does not update the number of circuits registered, so the initial difference between the circuit of the quantamorphism of Y and the circuits seen in the quantamorphism of X examples is the initialization of quantum and classical registers (appendix D.19).

The initial experiment executed for the quantamorphism of Y is made with the initial states in the default state ($|0\rangle$), and so the outputs should maintain this states unaltered. Recall from the experiments in Quipper that q_3 is the ancilla and there is no interest to measure this result. Also, in this directly translated circuit, q_2 is the target qubit.

Executing in the backend `local_unitary_simulator` can be seen in listing 5.20. From this experiment is expected a different matrix from the one implemented in Quipper due to the LSB/MSB issue. And similar to previous examples the measure has to be added later.

```
1  job = execute(qc, backend='local_unitary_simulator')
   np.round(job.result().get_data(qc)['unitary'], 3)
```

Listing 5.20: QISKit implementation of `local_unitary_simulator`.

The following execution, portrayed in the listing 5.21, shows how to use the backend `local_qasm_simulator` and different techniques to print the results. Moreover, the new QASM circuit and description are both printed.

```
   job = execute(qc, backend='local_qasm_simulator', shots=1024, max_credits=3)
2
   lapse = 0
4  interval = 5
   while not job.done:
6      print('Status @ {} seconds'.format(interval * lapse))
       print(job.status)
8      time.sleep(interval)
       lapse += 1
10 print(job.status)


12 print(job.result().get_counts(qc))
   plot_histogram(job.result().get_counts(qc))
```

Listing 5.21: QISKit implementation of `local_qasm_simulator`.

This circuit still operates with less than 6 qubits and so it can run on the `ibmqx4` device (implementation in listing 5.22). This implementation enables one to know the status of the experiment, it gives a notion of how long the job stays in the queue and it informs the user when the job is finally running.

```
1  shots=1024
   max_credits=3
3  job_exp = execute(qc, backend=backend, shots=shots, max_credits=max_credits)

5  lapse = 0
   interval = 10
7  while not job_exp.done:
       print('Status @ {} seconds'.format(interval * lapse))
9      print(job_exp.status)
       time.sleep(interval)
11     lapse += 1
   print(job_exp.status)
```

Listing 5.22: QISKit implementation of the circuit in backend `ibmqx4`

Facing tremendous errors in this case, three different strategies were tested to decrease the error rate:

1. The adaptation to QISKit (i.e. dealing with the LBS/MSB conflict);

2. Test the circuit in a different device (namely, `ibmqx5`);

3. Test with the maximum number of shots (8192 shots).

The advantages of the first strategy were already expressed in the `qfor` of X gate but the improvements may not be so visible as seen in the first example.
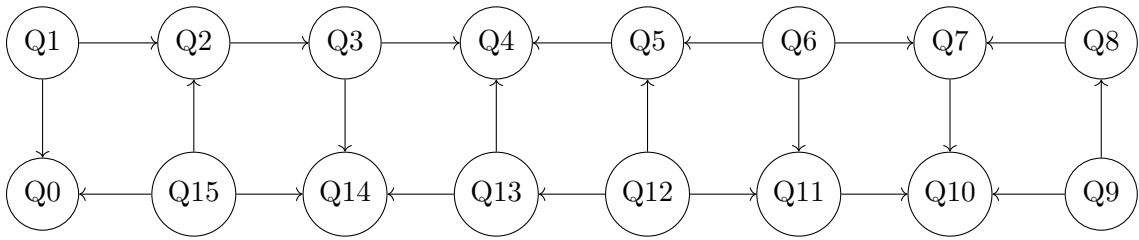
Figure 5.56.: Coupling map of IBM Q 16 Rueschlikon V1.x.x (`ibmqx5`). The direction of the arrows reads from control to target.

The second strategy aims mostly to compare the different devices. Although `ibmqx5` has a lot more qubits than `ibmqx4` the coupling is still very limited (compare the figures 5.55 to 5.56).

In the circuit of the `mqfor` of Y gate, it is likely that the error rate problems derive from the big number of gates. If this is the case, none of the strategies will be very helpful and decoherence will be present in all the cases.

The last strategy was to increase the number of shots. However, increasing the number of shots also increases the number of credits spent in the experiments (listing D.23). Consequently, this may not be the best option.

After a simple analysis of these results, adaptation to QISKit was the selected method to continue the experiments.

Since the circuits of the quantamorphism over Y produce a considerable amount of errors, this circuit was only tested for cases with one or two possible acceptable answers. The states were prepared with:

- control qubits (q_3 and q_2) were left in the default state ($|0\rangle$) and the target (q_1) was prepared with a Hadamard gate.

- all qubits (q_3, q_2 and q_1) were prepared with a X gate. The input state was $|111\rangle$

- the control qubits (q_3 and q_2) were prepared to be in a Bell state (maximum entanglement), this was achieved by sending q_3 to a superposition state through an Hadamard gate and the using a $CNOT$ gate where q_3 was the control and q_2 was the target. The target q_1 was left in the default state.

Note that q_0 in the adaptation case is the auxiliary qubit.

CREATING SUPERPOSITION VIA H As we already know, this example has a lot in common with the quantamorphism of Y gate. Consequently, this experiment follows the same procedure:

1. translation of the circuit in Quipper to QISKit using the tool;

2. run the original circuits with initial state $|000\rangle$ in the simulators and in the backend;

3. test methods to decrease the error rate;

4. use the best option to implement different initial states.

The first step is to translate the circuit using, the only command line need to implement QuipperToQISKit is in listing 5.23.

```
gawk -f quipperToQISKit.gawk circuit_h_quipper.txt > circuit_h_QISKit.txt
```

Listing 5.23: Bash command to run the tool QuipperToQISKit.

The initial circuit was executed in the backends `local_unitary_simulator`, `local_qasm_simulator` and `ibmqx4`. In this case, the only strategy used to improve the data was the adaptation to QISKit, but it did not accomplish its goal[9].

Therefore, the following experiments used the original circuit. Like the quantamorphism of Y gate, these tests were only made in situations where the output did not accept more than two results, in particular:

- $q_0 = 1$, $q_1 = 1$ and $q_2 = 0$ produced by adding X gates to the control qubits;

- $q_0 = 1$, $q_1 = 1$ and $q_2 = 1$ produced by adding X gates to all qubits;

- $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$ produced by adding X gates to the control qubits and a Hadamard gate to the target qubit;

- $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$ produced by adding X gates to all qubits and then a Hadamard gate to the target qubit.

QUANTAMORPHISM OVER XOR GATE   This experiment has some significant differences from the previous ones since this is the only circuit that runs with more than 5 qubits. Actually, the number of registers has to be updated to support the 7 qubits and so the circuit has to run on the `ibmqx5` device.

When this simulation was run for the first time the result was "INVALID QASM". Meanwhile, updates were made in the system and this circuit run with no problems (besides the large error rate). This confirms the need to register the information of the devices used in the experiments. Such data is obtained with the functions in listings D.7 and D.8.[10]

The attempts to improve this circuit included adaptation and adaptation with 8192 shots. These two improvement efforts were used with the initial state set to $|00000\rangle$. Thereupon it was decided to continue the experiments with the adapted circuit with 1024 shots.

---

[9]In section 5.5 further analysis will be made to understand why adaptation improves some situations and not others.

[10]Note that these functions were used in all the experiments.
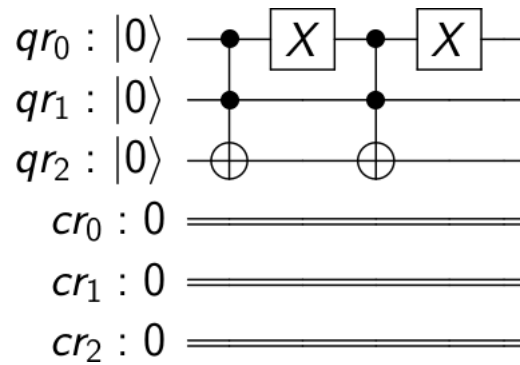
Figure 5.57.: Circuit from for-loop quantamorphism over X gate.

The following experiments tested with the input states:

- Hadamard gate in target qubit ($q_2 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$, $q_3 = 0$, $q_4 = 0$, $q_5 = 0$ and $q_6 = 0$);

- and X gate in control qubits($q_2 = 0$, $q_3 = 1$, $q_4 = 1$, $q_5 = 1$ and $q_6 = 1$);

- and X gate in all qubits ($q_2 = 1$, $q_3 = 1$, $q_4 = 1$, $q_5 = 1$ and $q_6 = 1$).

To conclude the implementation of QISKit its worth mentioning that everything relevant to implement or repeat in these experiments can be found in appendix D.

FOR-LOOP QUANTAMORPHISM OVER X GATE   As already seen in section 5.3, the results follow the implementations of the experiments. Recall that the outputs required are:

1. the circuit as it was written;

2. the matrix;

3. the simulation output;

4. the circuit that was simulated;

5. the output of running the experiment;

6. and the circuit of running the experiments.

Starting with the initial circuit (figure 5.57) it is easy to verify that this circuit is the one expected, with two Toffoli gates and X gates to negate one of the controls.

The output of the unitary matrix corresponding to this circuit (matrix 5.6) does not seem to agree to with the results of GHCi implementation (matrix 5.2). This is a direct result of
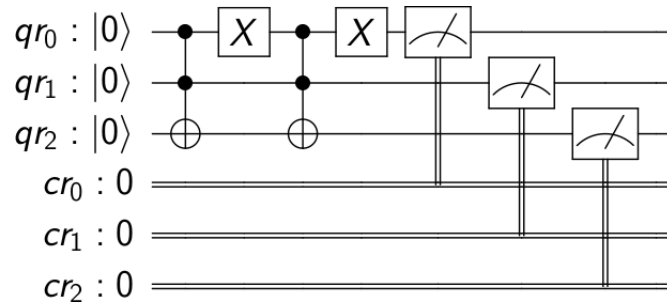
Figure 5.58.: Circuit from for-loop quantamorphism over X gate with measurement gate.

having the LSB and the MSB in the opposite order. Section 5.5 explains in detail how this works.

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\tag{5.6}
$$

Despite the differences in the matrix, both programs (the circuit running in Quipper and the circuit running in QISKit) perform the same task, i.e both of them use q_0 and q_1 to control an alteration to qubit 2.

The following step consists of adding the measurement, leading to the circuit of figure 5.58. Both proceedings compile and run, resulting in the same output (listing 5.24).

```
1        {'000': 1024}
```

Listing 5.24: `Get_count` output.

Listing 5.24 reads as: all the 1024 shots have the same result. Another way to present this output can be seen in figure 5.59.

Because this simulator includes all the Open QASM commands and works in a way comparable to the devices in IBM Q Experience (Javadi-Abhari et al., 2018) the gates allowed in the devices are more restricted than the ones that can be described. For instance, Toffoli gates cannot be implemented in the device and have to be decomposed following a method similar to the one by Cross et al. (2017). Knowing this information, it is no surprise that the QASM resulting from compiling the initial circuit has these gates decomposed (see figure 5.60 and listing 5.25). This decomposition is quite generic because it has to be adaptable to multiple devices.
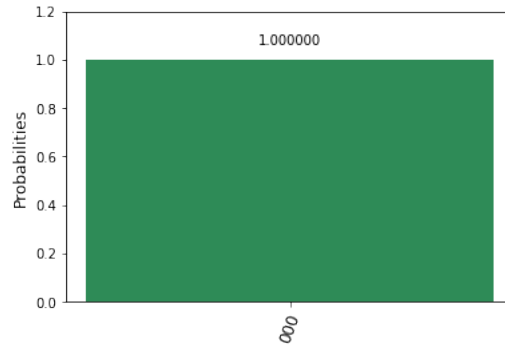
Figure 5.59.: Histogram of result.
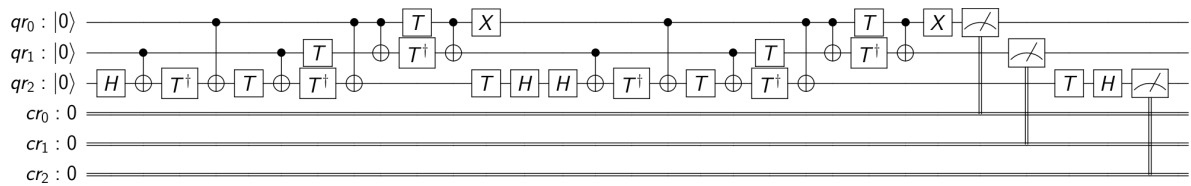
Figure 5.60.: Decomposed circuit from matrix quantamorphism for X.

```
1  OPENQASM 2.0;
   include "qelib1.inc";
3  qreg qr[3];
   creg cr[3];
5  ccx qr[0],qr[1],qr[2];
   x qr[0];
7  ccx qr[0],qr[1],qr[2];
   x qr[0];
9  measure qr[0] -> cr[0];
   measure qr[1] -> cr[1];
11 measure qr[2] -> cr[2];
```

Listing 5.25: QASM corresponding to the circuit in figure 5.60.

Since this experiment only requires 3 qubits there is no need to use the `ibmqx5` (team, 2018) which is the device ready for 16 qubits. At the time this thesis was written the `ibmqx5` was the public device with the bigger number of qubits available.

Running the circuit in a real device needs another "decomposition" this time is an adaptation of the required circuit to the device itself (figures 5.61 and 5.62).

Due to its size, this circuit is illegible, and for this reason, printing the QASM string is useful for further analyse (listing 5.26). In the QASM, the unitary gates are perfectly clear. These unitary gates (Research and the IBM QX team, 2017) are the general case of the gates implemented so far. Their function in here is analyzed in the section 5.5.
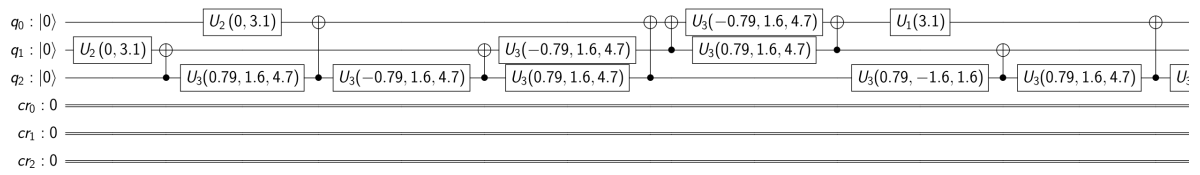
Figure 5.61.: Automatic adaptation to the device `ibmqx4` of the circuit from matrix quantamorphism `qfor` X (section 1).
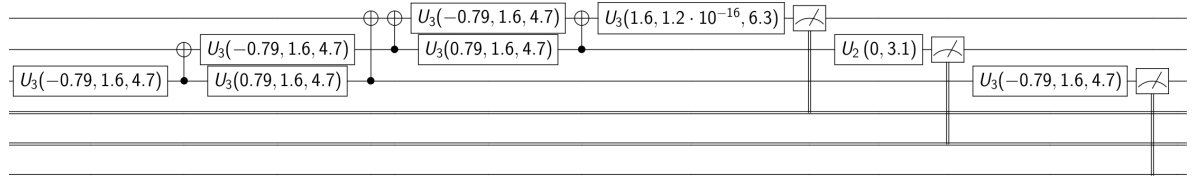
Figure 5.62.: Automatic adaptation to the device `ibmqx4` of the circuit from matrix quantamorphism `qfor` X (section 2).

```
1  OPENQASM 2.0;
   include "qelib1.inc";
3  qreg q[3];
   creg cr[3];
5  u2(0,3.14159265358979) q[1];
   cx q[2],q[1];
7  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
   u2(0,3.14159265358979) q[0];
9  cx q[2],q[0];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
11 cx q[2],q[1];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
13 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
   cx q[2],q[0];
15 cx q[1],q[0];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[0];
17 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
   cx q[1],q[0];
19 u1(3.14159265358979) q[0];
   u3(0.785398163397448,-1.57079632679490,1.57079632679490) q[2];
21 cx q[2],q[1];
   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
23 cx q[2],q[0];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
25 cx q[2],q[1];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
27 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
   cx q[2],q[0];
29 cx q[1],q[0];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[0];
```
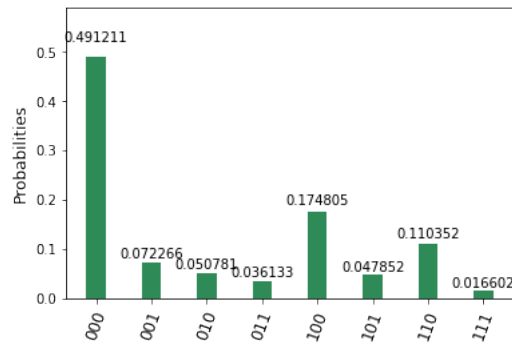
Figure 5.63.: Histogram: results of circuit for-loop quantamorphism over X gate with state preparation $|000\rangle$ in `ibmqx4`.

```
31  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
    cx q[1],q[0];
33  u3(1.57079632679490,1.22464679914735e-16,6.28318530717959) q[0];
    measure q[0] -> cr[0];
35  u2(0,3.14159265358979) q[1];
    measure q[1] -> cr[1];
37  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
    measure q[2] -> cr[2];
```

Listing 5.26: QASM corresponding to the circuit in figures 5.61 and 5.62.

Finally, the main goal of all the experiments, the information about running the program in a real device show a vast number of errors (nearly 50.87% of outputs failing the expected measure). Nevertheless, they still have a tendency to the right measure as can be seen in figure 5.63 of listing 5.27.

```
    {'counts': {'000': 503,
2          '001': 74,
           '010': 52,
4          '011': 37,
           '100': 179,
6          '101': 49,
           '110': 113,
8          '111': 17},
       'date': '2018-07-20T23:28:41.510Z',
10     'time': 24.603150844573975}
```

Listing 5.27: `Get_data` output.

Moreover, to better analyze these results it is also important to hold information about the device used at the time the experiment ran. These outputs can be found in appendix E.

$$qr_0 : |0\rangle$$
$$qr_1 : |0\rangle$$
$$qr_2 : |0\rangle$$
$$cr_0 : 0$$
$$cr_1 : 0$$
$$cr_2 : 0$$
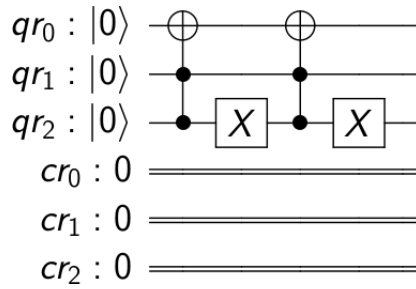
Figure 5.64.: Initial circuit of matrix quantamorphism `qfor` X adapted to LSB/MSB change and to the coupling map.

$$qr_0 : |0\rangle$$
$$qr_1 : |0\rangle$$
$$qr_2 : |0\rangle$$
$$cr_0 : 0$$
$$cr_1 : 0$$
$$cr_2 : 0$$
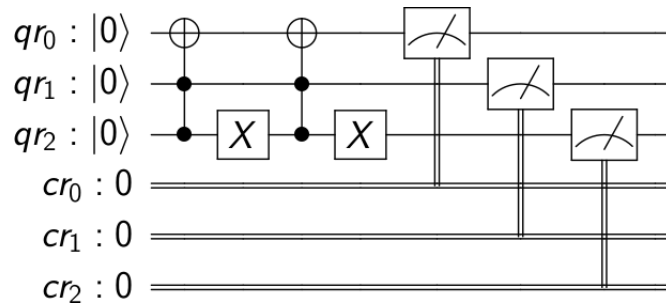
Figure 5.65.: Initial circuit of matrix quantamorphism `qfor` X adapted and with measurement gates.

The following results correspond to the circuit adapted to QISKit. Recall that in this case, the order of the qubits was inverted, the result was the circuit of figure 5.64.

The unitary matrix from this circuit (matrix 5.7) no doubt matches the initial (5.1). But in this case, qubit 0 is the target and 1 and 2 are the controls. [11]

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \tag{5.7}$$

The simulation required measurement gates as in figure 5.58, and result in the circuit of figure 5.65. This circuit has the qubits 1 and 2 controlling the qubit 0 which suits the coupling map of the device used.

Although this circuit has already an adaptation, it still operates with Toffoli gates that have to be decomposed by QISKit (figure 5.66). The circuit of figure 5.66 clearly shows that

---

[11] The real output of the program is an array of complex numbers, the notation used is efficient but not easy to read, this outputs can be found in appendix E.
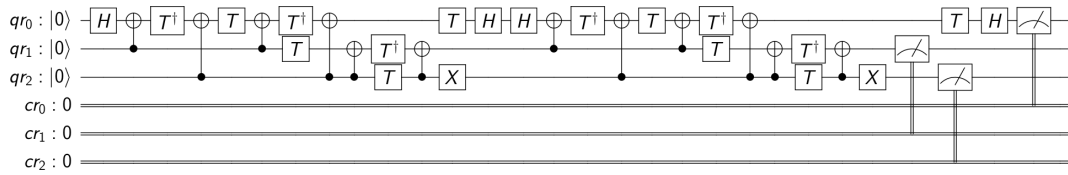
Figure 5.66.: Decomposed circuit of matrix quantamorphism `qfor` X adapted.
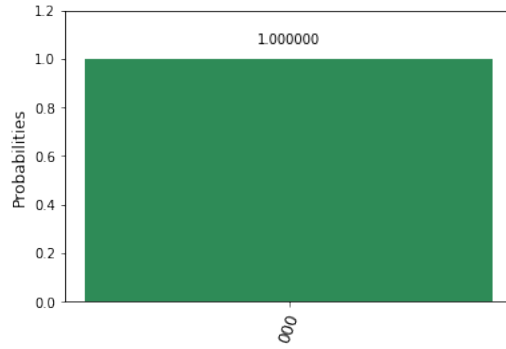
Figure 5.67.: Simulation output of circuit matrix quantamorphism `for` X adapted.

qubit 0 and 1 are controlled by qubit 2, but never control qubit 2. This is a precise match to the coupling map of the real device (figure 5.55).

The execution in a local simulator gives the same result, as can be seen by comparing figures 5.67 and 5.59.

Following the previous pattern, the execution in the real device not only displays a distinct output from the simulation, see figure 5.69.[12], but also has a new decomposition resulting in the circuit of figure 5.68. (This circuit is depicted in the QASM listing 5.28.)

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[3];
creg cr[3];
u2(0,3.14159265358979) q[0];
cx q[1],q[0];
u1(-0.785398163397448) q[0];
cx q[2],q[0];
u1(0.785398163397448) q[0];
```
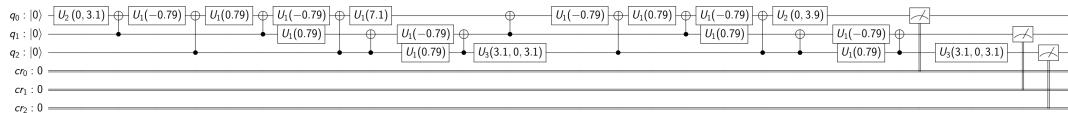

Figure 5.68.: Automatic adaptation to the device `ibmqx4` of the adapted circuit from matrix quantamorphism for X.

---

[12]The detailed data of this experiment can be found in appendix E.
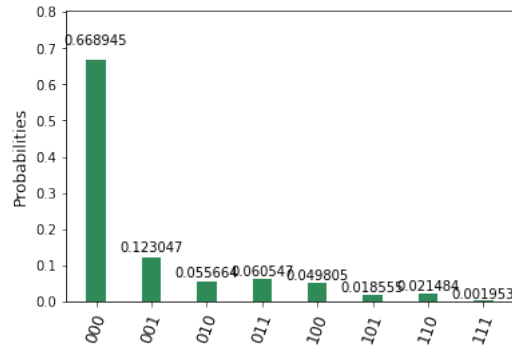
Figure 5.69.: Results of running the circuit matrix quantamorphism `qfor` X adapted `ibmqx4`.

```
10  cx q[1],q[0];
    u1(0.785398163397448) q[1];
12  u1(-0.785398163397448) q[0];
    cx q[2],q[0];
14  cx q[2],q[1];
    u1(7.06858347057703) q[0];
16  u1(0.785398163397448) q[2];
    u1(-0.785398163397448) q[1];
18  cx q[2],q[1];
    u3(3.14159265358979,0,3.14159265358979) q[2];
20  cx q[1],q[0];
    u1(-0.785398163397448) q[0];
22  cx q[2],q[0];
    u1(0.785398163397448) q[0];
24  cx q[1],q[0];
    u1(0.785398163397448) q[1];
26  u1(-0.785398163397448) q[0];
    cx q[2],q[0];
28  cx q[2],q[1];
    u2(0,3.92699081698724) q[0];
30  u1(0.785398163397448) q[2];
    u1(-0.785398163397448) q[1];
32  cx q[2],q[1];
    measure q[0] -> cr[0];
34  u3(3.14159265358979,0,3.14159265358979) q[2];
    measure q[1] -> cr[1];
36  measure q[2] -> cr[2];
```

Listing 5.28: QASM from compiling in the real device the adapted circuit.

Moreover, it is crucial to notice that in spite of being different from the simulation, this execution has considerably less errors. In fact, the measures only fail 33.12% of the shots. (This was the reason why the adaptation was chosen in the rest of the experiments).

(a) Simulation with state preparation $q_2 = 0$ $q_1 = 0$ and $q_0 = Hadamard$.

(b) Simulation with state preparation $q_2 = Hadamard$ $q_1 = Hadamard$ and $q_0 = 0$.

(c) Simulation with state preparation $q_2 = 1$ $q_1 = 1$ and $q_0 = 1$.

(d) Simulation with state preparation q_2 and q_1 with Bell state and $q_0 = 0$.
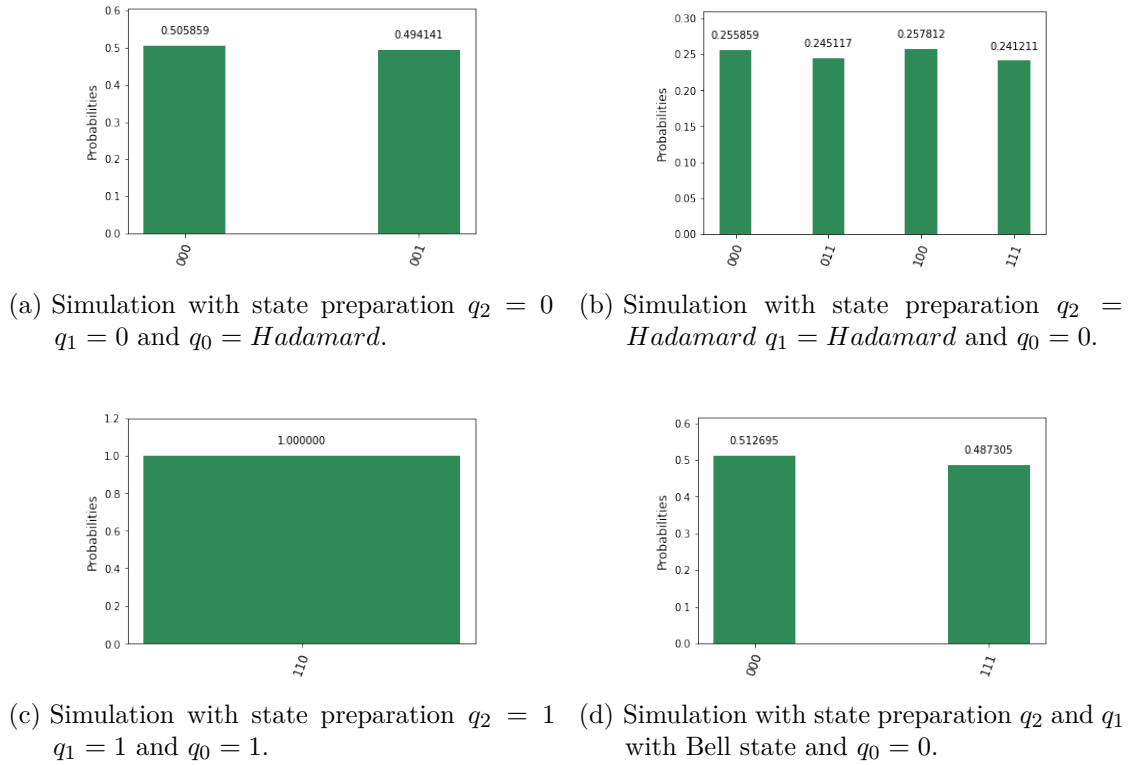
Figure 5.70.: Simulation of program circuit quantamorphism `qfor` X with different input states.

Recall that in the experimentations where the initial state change the only notable outputs are the simulation in local QASM (figures 5.70) and the execution in the real device (figure 5.71). There was a previous mention that the simulation behaves similarly to the real device. To obtain a better approximation, this simulation relies on the coin tossing method to add a random factor to the results. In figures 5.70a and 5.70d the graphics don't have a mathematical 50/50 probability due to this simulation property.

As a final note to these experiments, it is possible to see that, although the execution in the real device is not perfect, the results correspond to the expected in most cases.

Finally, *the even or odd test* was repeated with a larger number of control qubits. In previous sections (5.2 and 5.3) the alteration of the number of qubits forces to add the decomposition, which drastically changes the results when running in the real device.

Starting with the matrix of the original translation (appendix E.7), clearly, QISKit is not ready to deal with such a big matrix (32×32) [13]. Previously we have reasoned that the alteration to the matrix happens because of the LSB and MSB but in this case, there is an extra problem, namely, the auxiliary qubit. A detail analysis can be found in section 5.5. As expected, the adapted circuit has the same problem with the matrix resulting in the appendix E.7.

_____

[13]recall that this matrix was initially a 16×16 matrix.

(a) Execute with state preparation $q_2 = 0$ $q_1 = 0$ and $q_0 = Hadamard$.

(b) Execute with state preparation $q_2 = Hadamard$ $q_1 = Hadamard$ and $q_0 = 0$.

(c) Execute with state preparation $q_2 = 1$ $q_1 = 1$ and $q_0 = 1$.

(d) Execute with state preparation q_2 and q_1 with Bell state and $q_0 = 0$.
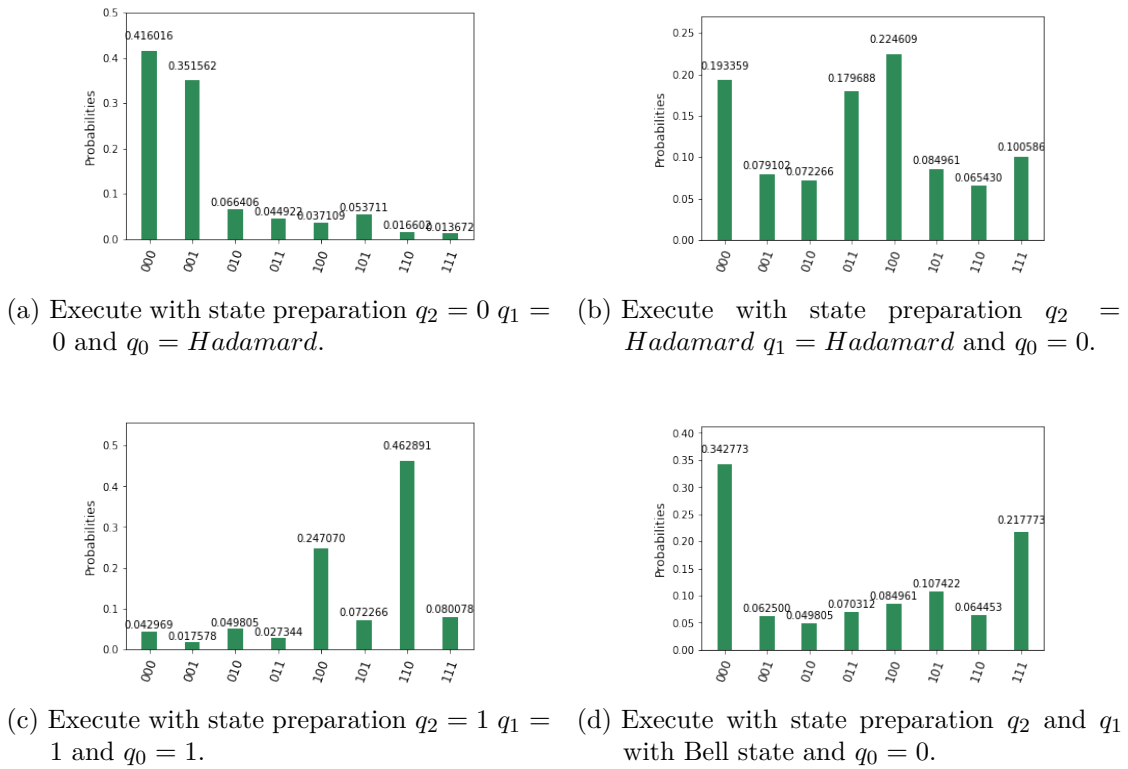
Figure 5.71.: Execute of program circuit quantamorphism `qfor` X with different input states in `ibmqx4`.

(a) Simulation of the original circuit.

(b) Simulation of the circuit adapted to QISKit.
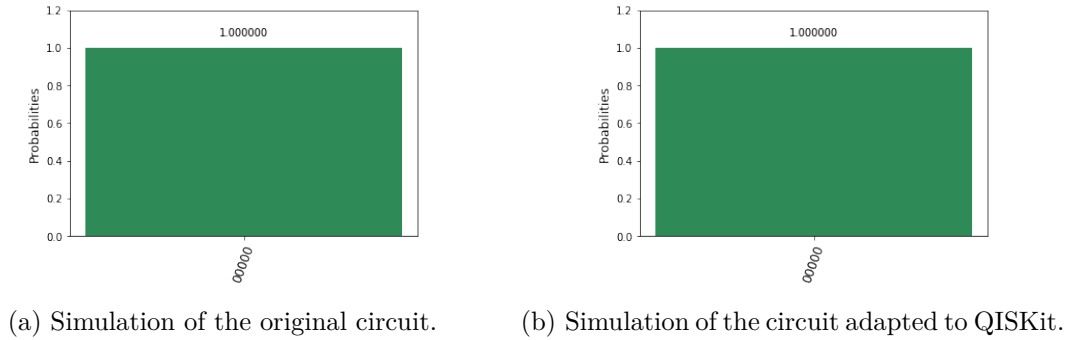
Figure 5.72.: Simulation of circuit `qfor` X gate with 3 control qubits.

(a) Execution in the real device of the original circuit.

(b) Execution in the real device of the circuit adapted to QISKit.
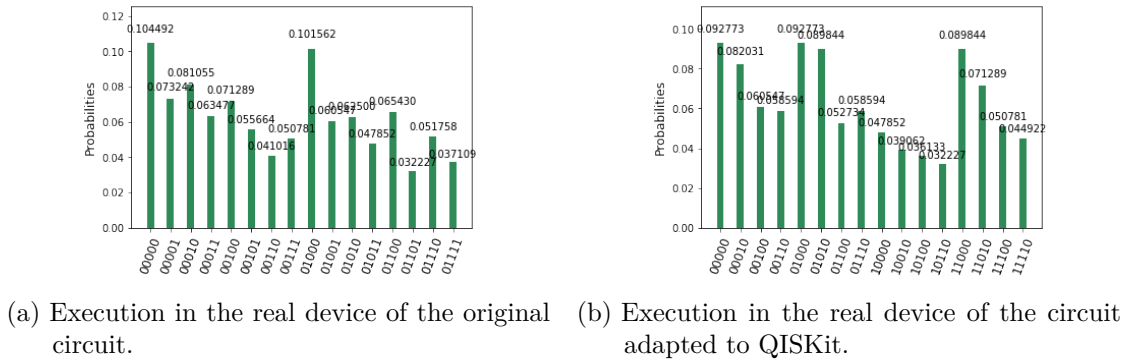
Figure 5.73.: Execution in the real device `ibmqx4` of circuit `qfor` X gate with 3 control qubits.

The simulations of the circuit adapted to QISKit and the original translation are equal the figure 5.72.

Moreover, the results of the execution in the real device do not differ substantially, as can be observed from the figure 5.73 both of this results have approximately the same percentage of accurate results (roughly 10%). This hints even worse results for the rest of the experiments.

The experiments with distinctive initial states resulted in the simulation of figure 5.74 and the execution in the real device `ibmqx4`, see figure 5.75.

The differences within this results are explored in section 5.5.

FOR-LOOP QUANTAMORPHISM OVER Y GATE   The result of implementing the direct translation of the Quipper's decomposed circuit is the circuit shown in figures 5.76, 5.77 and 5.78.

Recall the previous example, in which the ancillary qubit increases the size of the matrix. This effect is also present in this example, despite still being possible to print the matrix (matrix 5.8). As should be expected by the previous results this matrix does not precisely
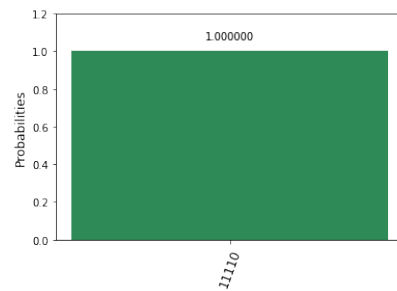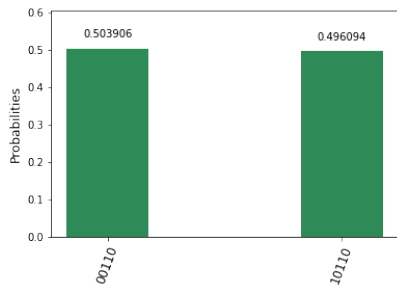
(a) Simulation of circuit with MSB prepared with Hadamard gate and the LSB of the controls prepared with X gate

(b) Simulation of circuit with control qubits prepared with X gate.
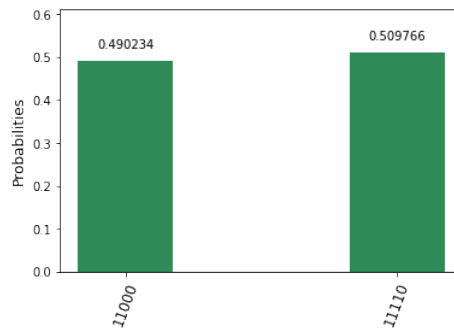
(c) Simulation of circuit with control prepared as Bell state.

Figure 5.74.: Simulations of experiments.

(a) Results of running the circuit with MSB prepared with Hadamard gate and the LSB of the controls prepared with X gate

(b) Results of running the circuit with with control qubits prepared with X gate.
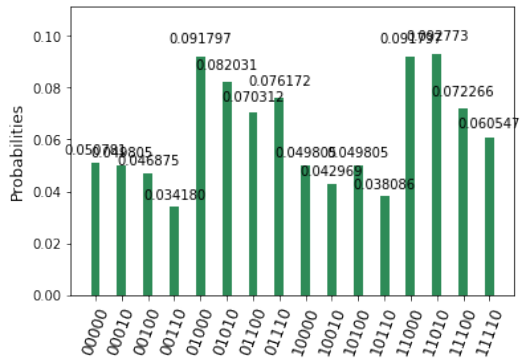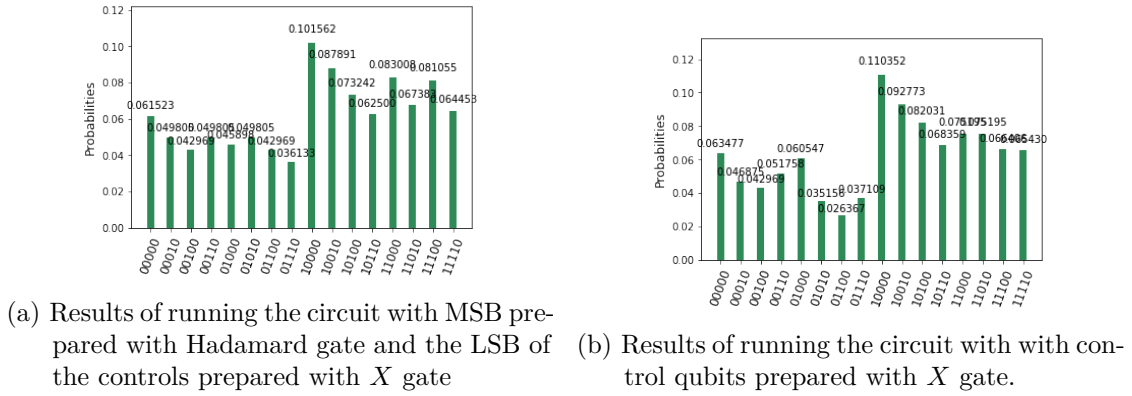
(c) Results of running the circuit with control prepared as Bell state.

Figure 5.75.: Execution in the real device `ibmqx4`.
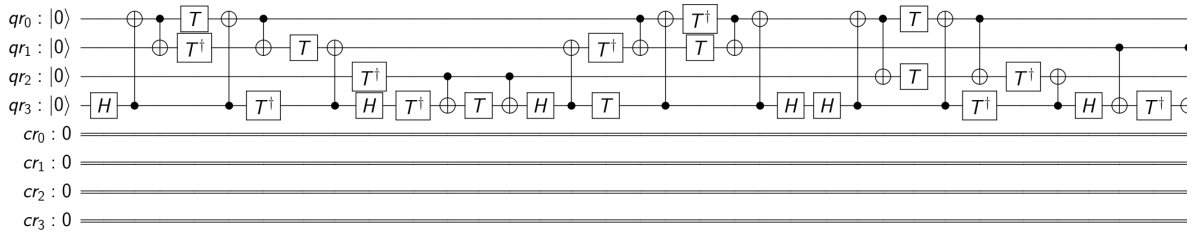
Figure 5.76.: Initial circuit of gate quantamorphism `qfor` Y in QISKit implementation (section 1).
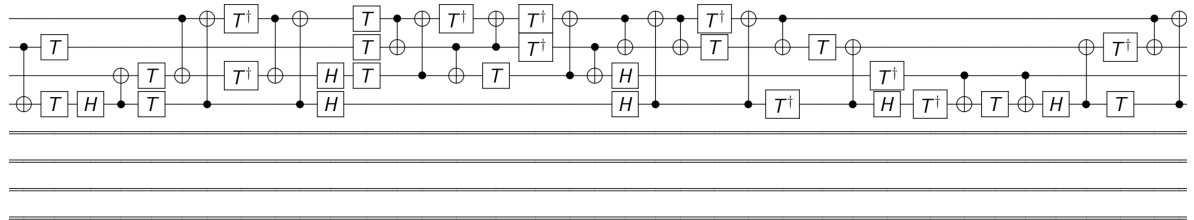
Figure 5.77.: Initial circuit of gate quantamorphism `qfor` Y in QISKit implementation (section 2).
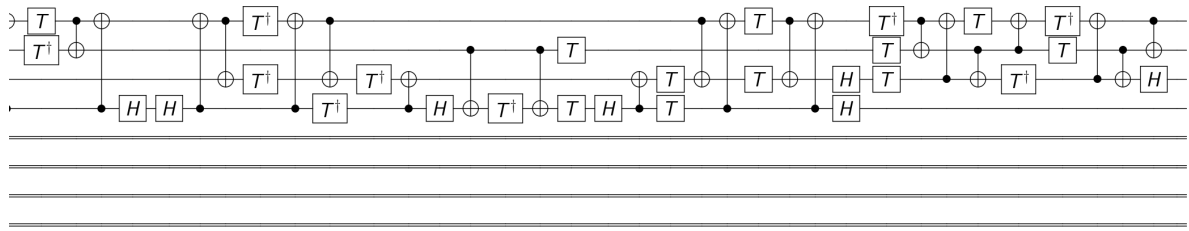
Figure 5.78.: Initial circuit of gate quantamorphism `qfor` Y in QISKit implementation (section 3).
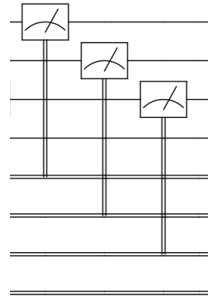
Figure 5.79.: Circuit of matrix quantamorphism `qfor` Y with measurement gates.

resemble the matrix implemented in Quipper. The reason to achieve this matrix is detailed in section 5.5.

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & i & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix}
\tag{5.8}
$$

With measures, the previous circuit ends with gates in figure 5.79.

The QASM simulation output is given in figure 5.80, but keep in mind that such simulation corresponds to the first decomposed circuit (figures 5.81 to 5.85). Because all QASM scripts printed in this experiment are very long, they are deferred to appendix E.

To run the circuit in the real device the circuit has yet another decomposition (figures 5.86 to 5.89) which results in the data seen in figure 5.90.

Remind the attempts to improve this outputs started with the adaptation. This adaptation to QISKit gives the circuits figures 5.91, 5.92 and 5.93. Which produces the matrix (5.9).
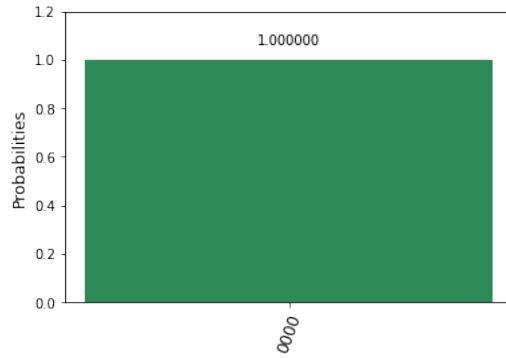
Figure 5.80.: Simulation of circuit matrix quantamorphism for Y.
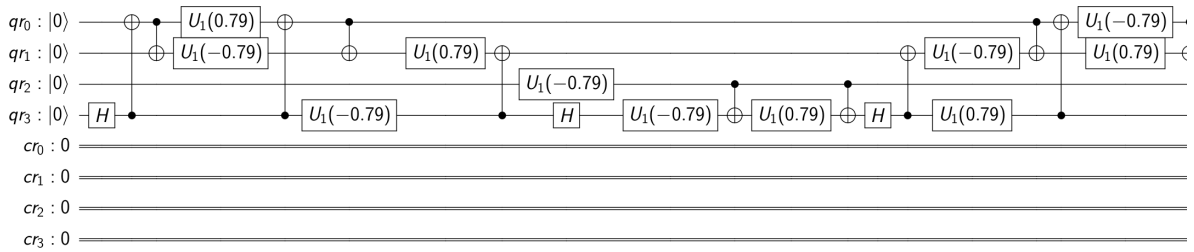
Figure 5.81.: Decomposed circuit of matrix quantamorphism `qfor` Y (section 1).
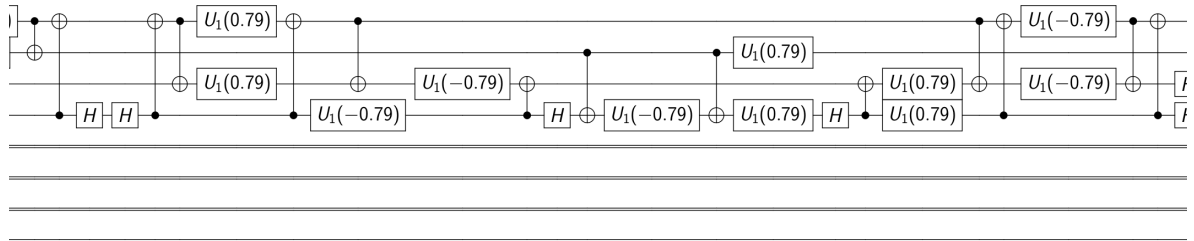
Figure 5.82.: Decomposed circuit of matrix quantamorphism `qfor` Y (section 2).
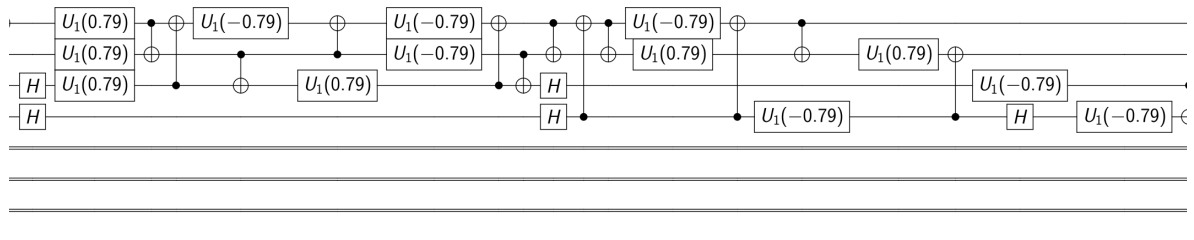
Figure 5.83.: Decomposed circuit of matrix quantamorphism `qfor` Y (section 3).
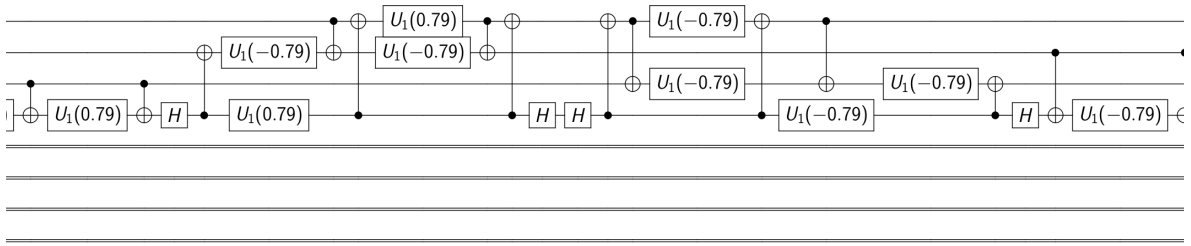
Figure 5.84.: Decomposed circuit of matrix quantamorphism `qfor` Y (section 4).
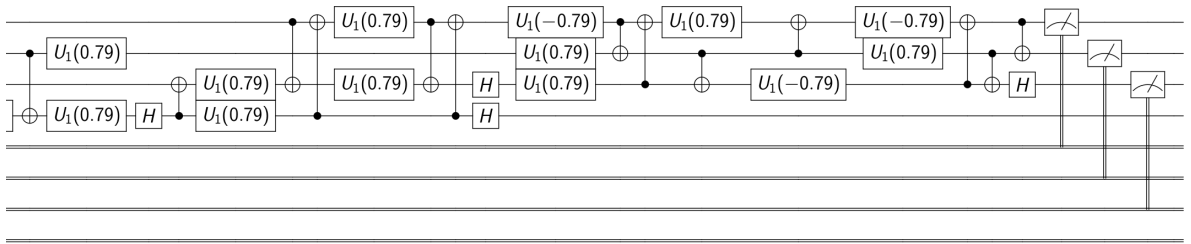
Figure 5.85.: Decomposed circuit of matrix quantamorphism `qforY` (section 5).
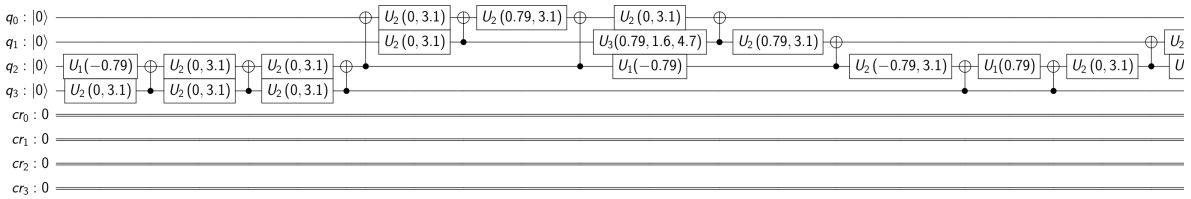
Figure 5.86.: Circuit of matrix quantamorphism `qfor` Y (section 1).
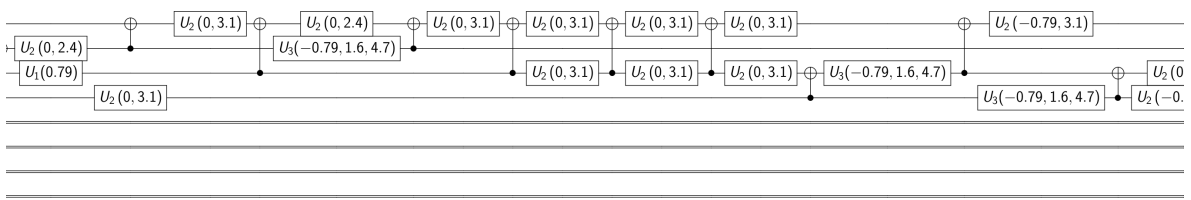
Figure 5.87.: Circuit of matrix quantamorphism `qfor` Y (section 2).
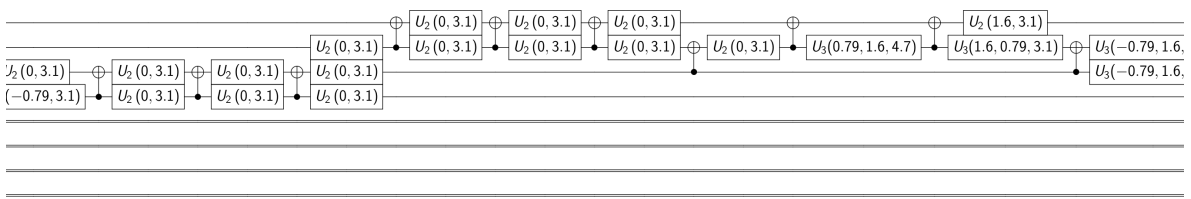
Figure 5.88.: Circuit of matrix quantamorphism `qfor` Y (section 3).
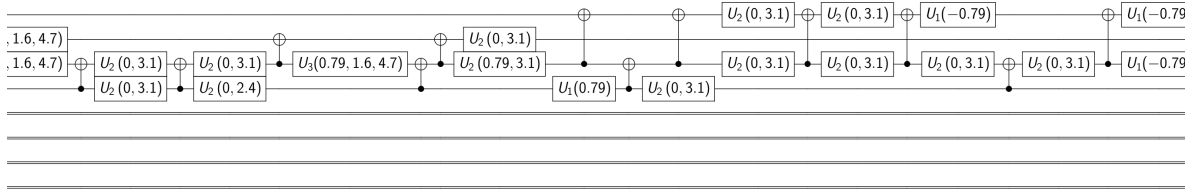
Figure 5.89.: Circuit of matrix quantamorphism `qfor` Y (section 4).
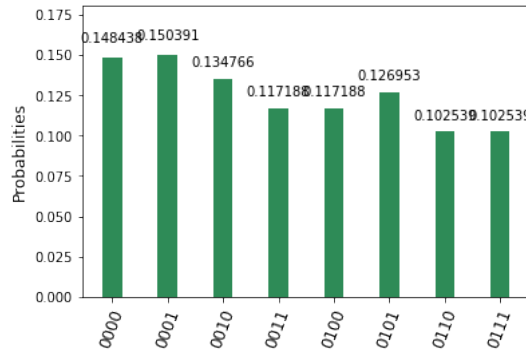
Figure 5.90.: Output of running the circuit matrix quantamorphism `qfor` Y in `ibmqx4` device.
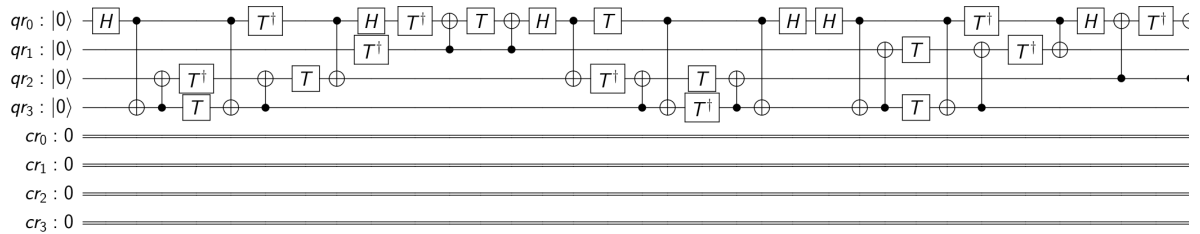
Figure 5.91.: Initial circuit of matrix quantamorphism `qfor` Y adapted to LSB/MSB change (section 1).
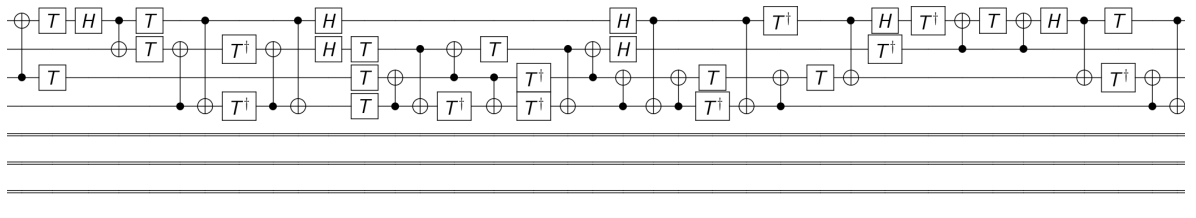
Figure 5.92.: Initial circuit of matrix quantamorphism `qfor` Y adapted to LSB/MSB change (section 2).
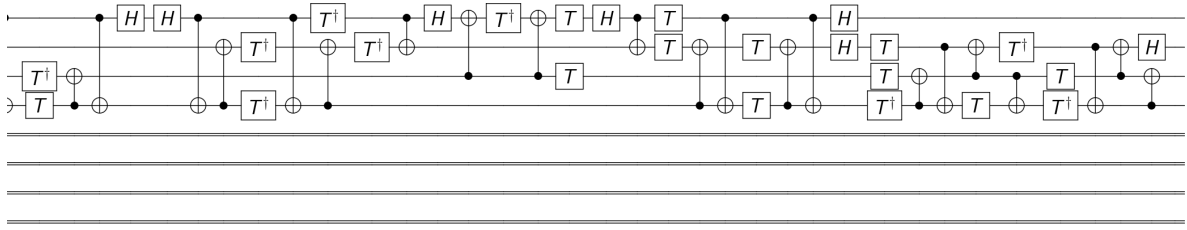
Figure 5.93.: Initial circuit of matrix quantamorphism `qfor` Y adapted to LSB/MSB change (section 3).
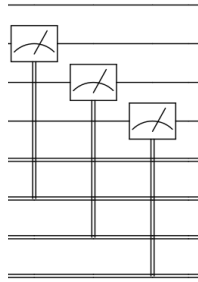
Figure 5.94.: Measurement gates of the circuit adapted.

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -i & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & i & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0
\end{bmatrix} \tag{5.9}$$

The measures are different (figure 5.94) from the previous example because there is no interest in the measure of the auxiliary qubit.

The simulation of this circuit matches the outputs previously seen in figure 5.80. It is given by the circuit split over figures 5.95 to 5.100.

Because the circuit that runs in the real device exceeds the maximum size to print in LaTeX, only its QASM description is given, see appendix E. This results in the graphic of figure 5.101.
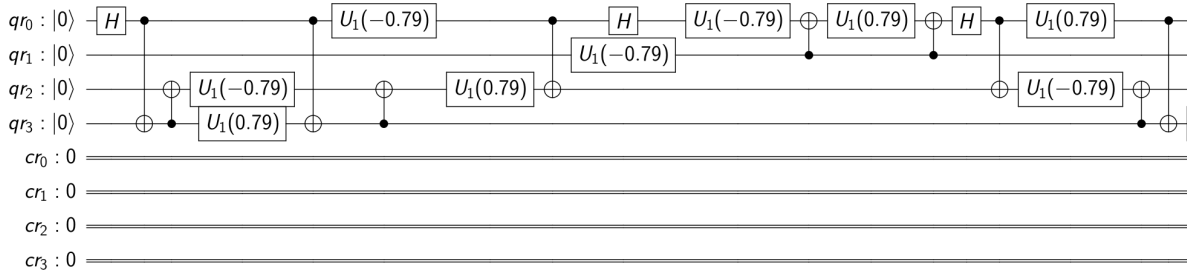
Figure 5.95.: Decomposed circuits of quantamorphism for Y to execute in QASM simulator (section 1).
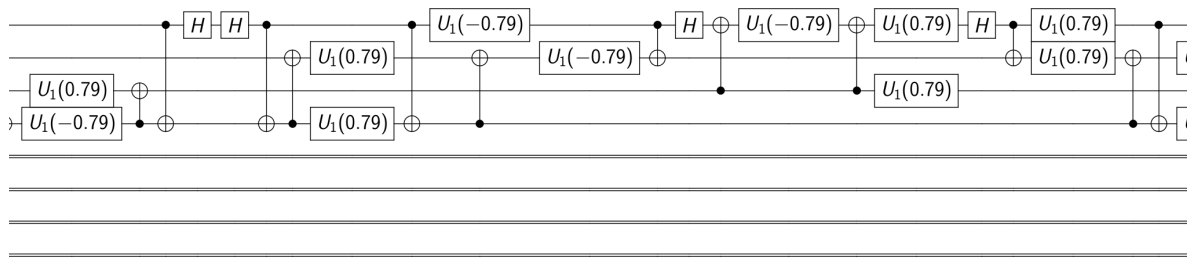
Figure 5.96.: Decomposed circuits of quantamorphism for Y to execute in QASM simulator (section 2).
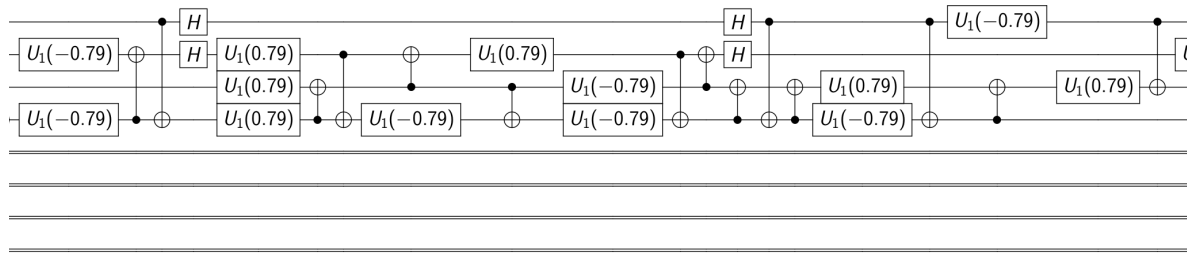
Figure 5.97.: Decomposed circuits of quantamorphism for Y to execute in QASM simulator (section 3).
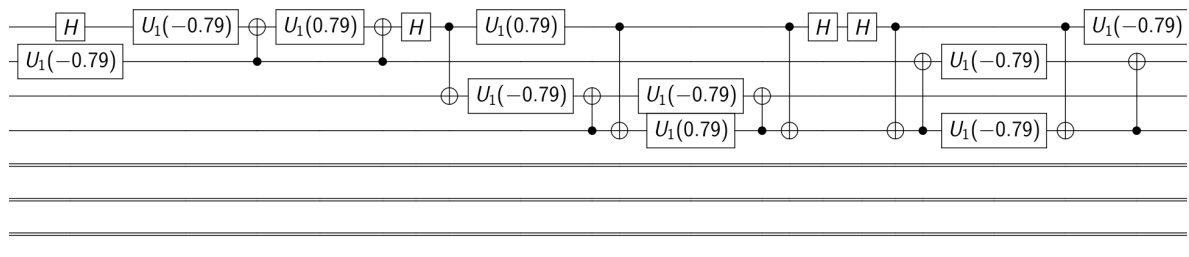
Figure 5.98.: Decomposed circuits of quantamorphism for Y to execute in QASM simulator (section 4).
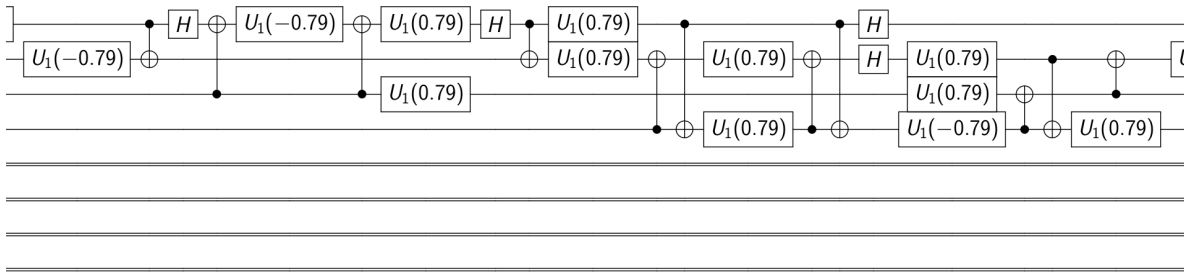
Figure 5.99.: Decomposed circuits of quantamorphism for Y to execute in QASM simulator (section 5).
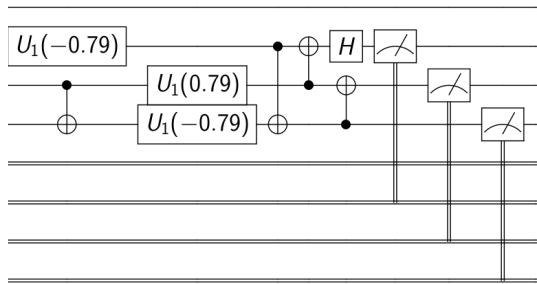
Figure 5.100.: Decomposed circuits of quantamorphism for Y to execute in QASM simulator (section 6).

In figure 5.101 it is possible to see that the improvements are poor, nevertheless, the following improvements work upon this adaptation.

As previously mentioned, the only alterations are in running the program in the real devices. Therefore, the only outputs shown in this section are the final graphics 5.102a and 5.102b, these outputs show an insignificant improvement, specifically, the difference between the best output (obtained by increasing the number of shots to 8192) and the simple adaptation is only 0.7569%.
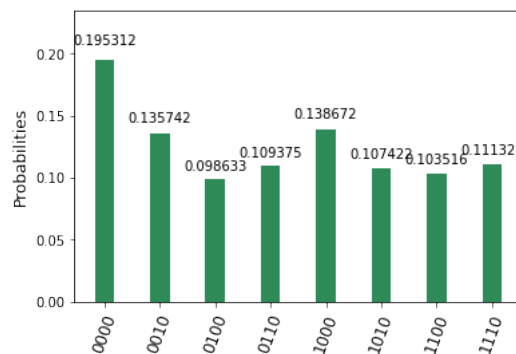
Figure 5.101.: Running adapted circuit of quantamorphism for Y in `ibmqx4`.

(a) Results of running the circuit `qfor` Y adapted to QISKit in `ibmqx5`, with 1024.

(b) Results of running the circuit `qfor` Y adapted to QISKit in `ibmqx4`, with 8192 shots.
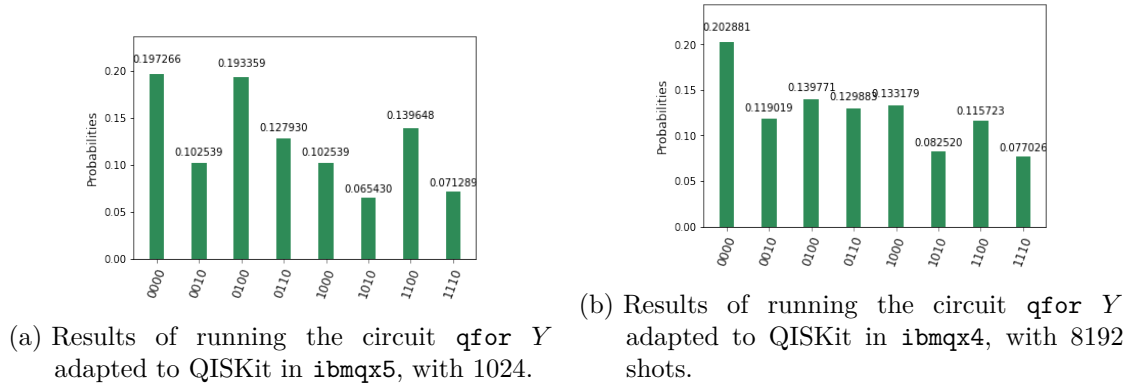
Figure 5.102.: Results of the improvement experiments.

Such results lead to the use of the simple adapted QISKit to make other tests. Namely, the tests with diverse states. In these cases, the inputs tested are in the figure 5.103 and despite the simulation matches the prediction from the Quipper (5.104) the execution in the real device gives an output with clear decoherence issues (5.105).

CREATING SUPERPOSITION VIA H Finally, a study of a quantum program that handles real superposition. The original translation of this program gives the circuit of figure 5.106. This can be represented by the unitary matrix in (5.10).

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}}
\end{bmatrix}
\tag{5.10}
$$

The measures were added later, giving to the circuit of figure 5.106 the termination seen in figure 5.79. By adding the measures it was possible to test the simulation (figure 5.107), recall that this needed the decomposition (figures 5.108 and 5.109).
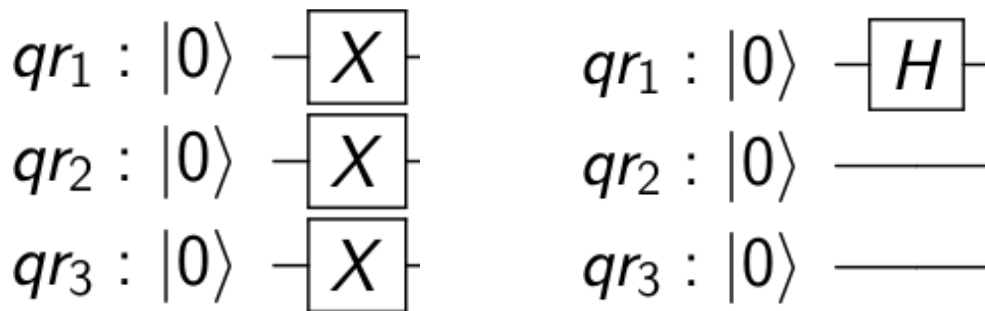
(a) Gates to prepares the circuit quantamorphism for Y gate with the initial states: $q_1 = 1$, $q_2 = 1$ and $q_3 = 1$.

(b) Gates to prepare the circuit quantamorphism for Y gate with the target in the superposition state.
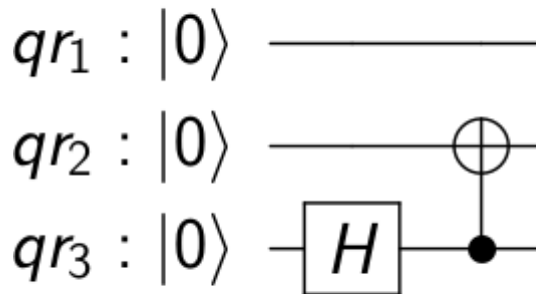
(c) Gates to prepare the circuit quantamorphism with control prepared as Bell state.

Figure 5.103.: Different state preparations of quantamorphism for Y .
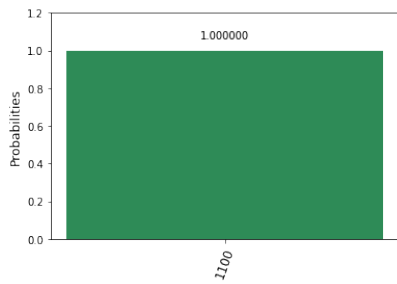
(a) Simulation of circuit quantamorphism for Y gate with all qubits prepared with X gates, the initial states: $q_1 = 1$, $q_2 = 1$ and $q_3 = 1$.
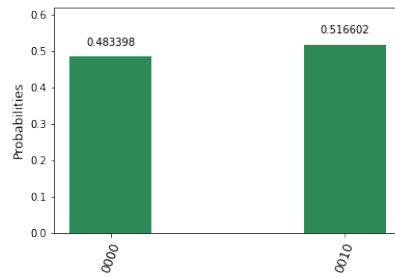
(b) Simulation of circuit quantamorphism for Y gate with target qubit prepared with Hadamard gate.
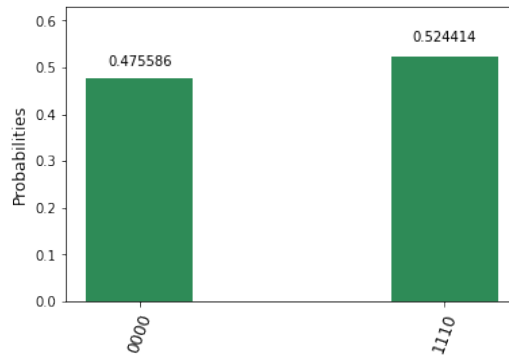
(c) Simulation of circuit with control prepared as Bell state.

Figure 5.104.: Simulations of experiments of quantamorphism for Y.
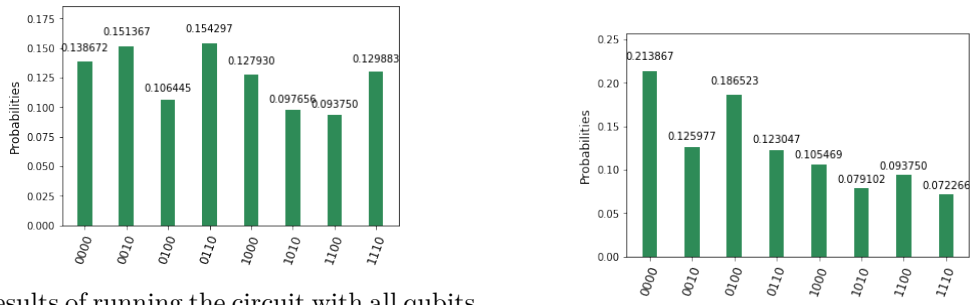
(a) Results of running the circuit with all qubits prepared with X gate, the initial states: $q_1 = 1$, $q_2 = 1$ and $q_3 = 1$.

(b) Results of running the circuit with target qubit prepared with Hadamard gate.
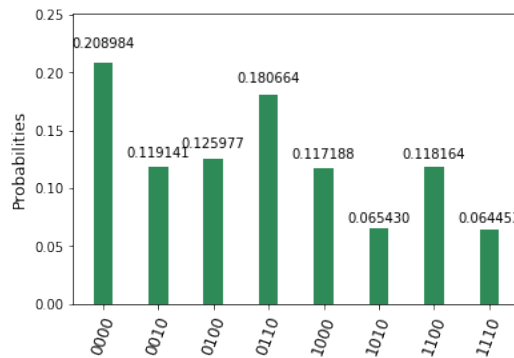
(c) Results of running the circuit with control prepared as Bell state.

Figure 5.105.: Execution in the real device `ibmqx4` of quantamorphism for Y .
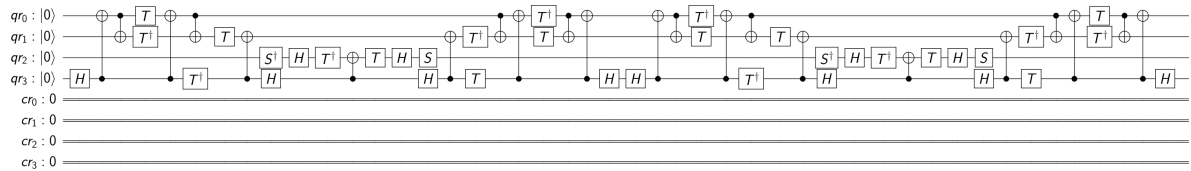
Figure 5.106.: Initial circuit of for-loop quantamorphism over Hadamard gate in QISKit implementation.
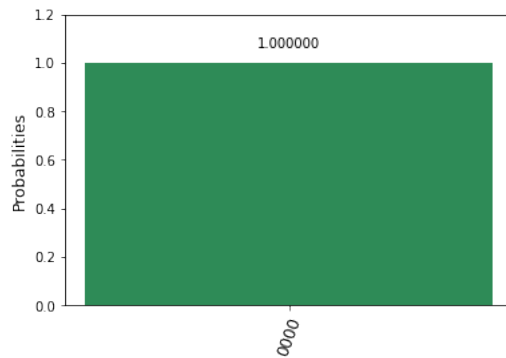
Figure 5.107.: Simulation of quantamorphism over Hadamard gate.
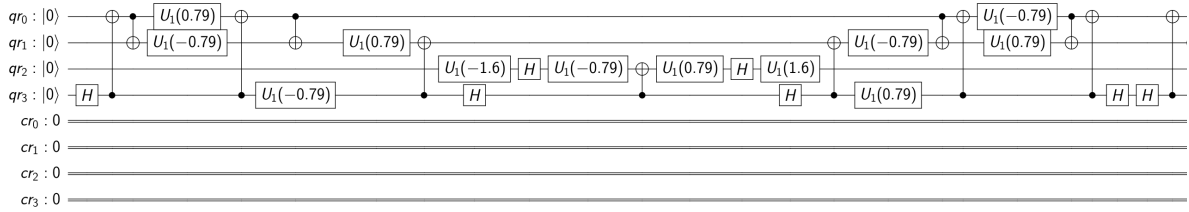
Figure 5.108.: Decomposed circuit of for-loop quantamorphism over Hadamard gate to execute in QASM simulator (section 1).
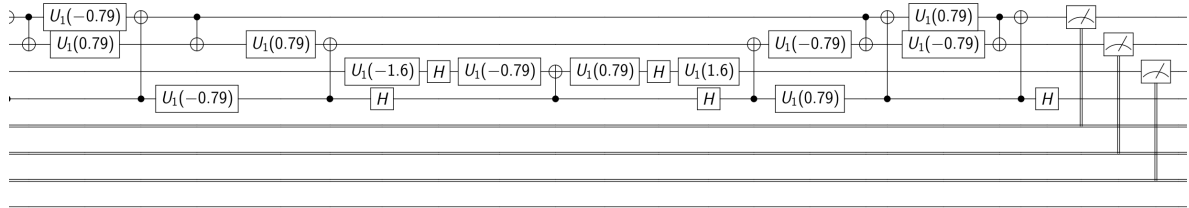
Figure 5.109.: Decomposed circuit of for-loop quantamorphism over Hadamard gate to execute in QASM simulator (section 2).

At this point, it is predictable that running the circuit increases the errors and, in fact, the outputs measured show no surprises (fig 5.110). The last decomposition needed to run this experiment is in figures 5.111 and 5.112.

As previously done, there were efforts to improve this result and the attempts started specifically with the adaptation. This resulted in the new circuit of figure 5.113 and in a new unitary matrix (5.11).
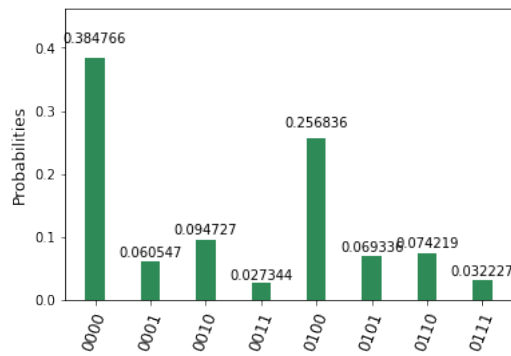
Figure 5.110.: Output of running the circuit of for-loop quantamorphism over Hadamard gate in `ibmqx4` device.

Figure 5.111.: Decomposed circuit of for-loop quantamorphism over Hadamard gate to execute in `ibmqx4` (section 1).

Figure 5.112.: Decomposed circuit of for-loop quantamorphism over Hadamard gate to execute in `ibmqx4` (section 2).

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}}
\end{bmatrix}
\tag{5.11}
$$

Simulation matched the already seen in figure 5.107 and was achieved with the decomposed circuit in figures 5.114 and 5.115.

Figure 5.113.: Circuit of quantamorphism over Hadamard gate adapted to QISKit.

Figure 5.114.: Decomposed circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit to execute in QASM simulator (section 1).
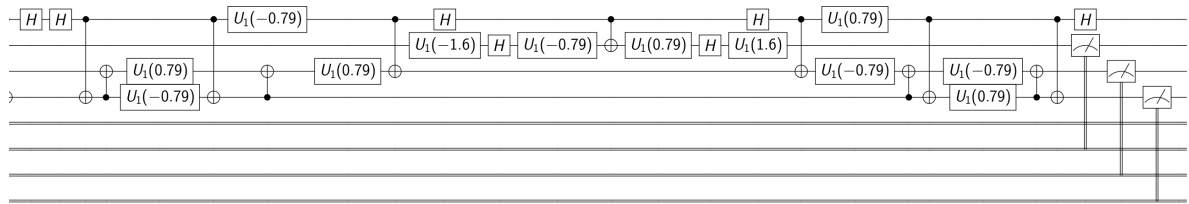
Figure 5.115.: Decomposed circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit to execute in QASM simulator (section 2).

Finally, the last decomposition was made to run this circuit in a real device (figure from 5.116 to 5.119) but contrary to what has been seen in the previous cases the adaptation did not reveal an improvement (figure 5.120).

Therefore the following experiments with different initial states were made with the original circuit. Here the outputs expected (seen in the simulation of figure 5.121) were quite different from the outputs in the real device (seen in the graphic of figure 5.122). Note the tendency to the right solution.

QUANTAMORPHISM OF XOR GATE   In this last example, the circuit is essentially different in the execution of the real device due to the fact that can only run `ibmqx5`. Moreover, due to the number of controls, the circuits had to be decomposed in a circuit so large that can not be printed. All the QASM scripts are in appendix E.

It comes with no surprise that the matrix of this circuit could not print either (since it would be 128×128 matrix). The simulation result taken can be found in figure 5.123 and the
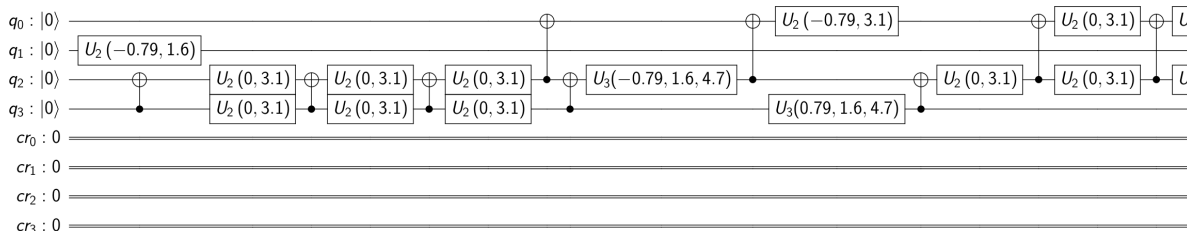
Figure 5.116.: Decomposed circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit to execute in `ibmqx4` (section 1).
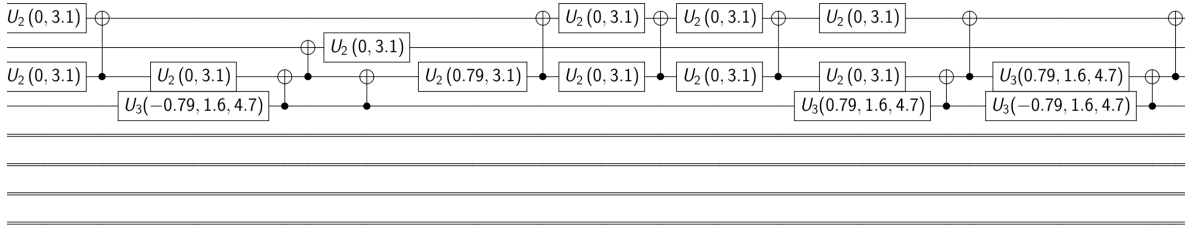
Figure 5.117.: Decomposed circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit to execute in `ibmqx4` (section 2).
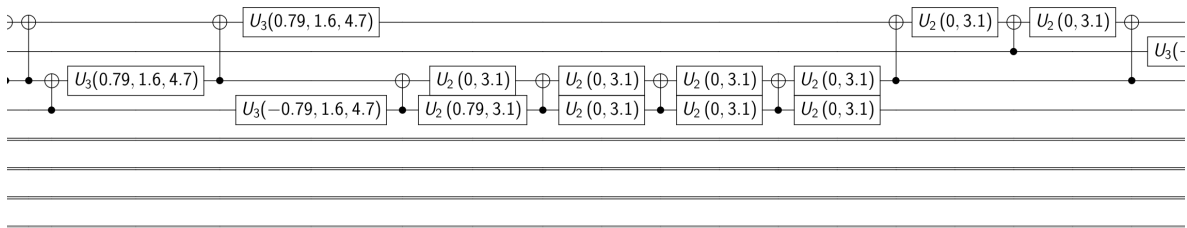
Figure 5.118.: Decomposed circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit to execute in `ibmqx4` (section 3).
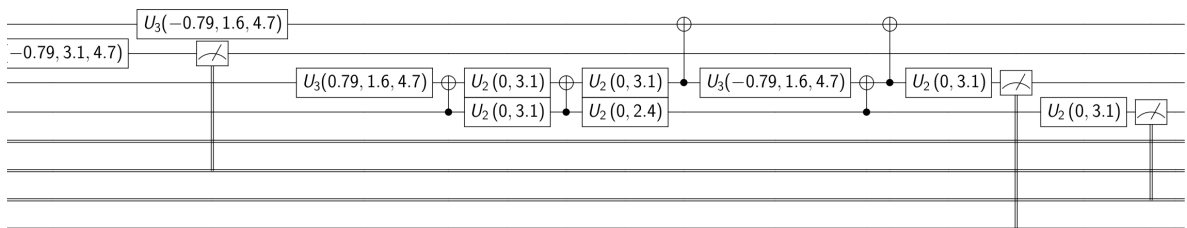
Figure 5.119.: Decomposed circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit to execute in `ibmqx4` (section 4).
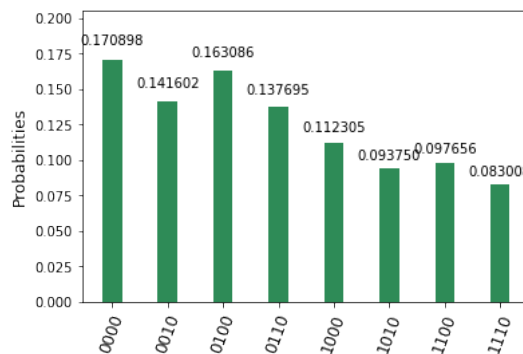
Figure 5.120.: Output of running the circuit of for-loop quantamorphism over Hadamard gate adapted to QISKit in `ibmqx4` device.

(a) Results of the simulation where the initial state was prepared with a X gate in the control qubits.

(b) Results of the simulation where the initial state was prepared with X gate in all qubits followed by a Hadamard gate in the target qubit.

(c) Results of the simulation where the initial state was prepared with X gate in the control qubits and a Hadamard gate in the target qubit.

(d) Results of the simulation where the initial state was prepared with X gate in all qubits.

Figure 5.121.: Simulation of the experiments of quantamorphism over Hadamard gate.

(a) Results of running the circuit where the initial state was prepared with a X gate in the the control qubits.

(b) Results of running the circuit where the initial state was prepared with X gate in all qubits followed by a Hadamard gate in the target qubit.

(c) Results of running the circuit where the initial state was prepared with X gate in the control qubits and a Hadamard gate in the target qubit.

(d) Results of running the circuit where the initial state was prepared with X gate in all qubits.

Figure 5.122.: Execution in the real device `ibmqx4` of quantamorphism over Hadamard gate.
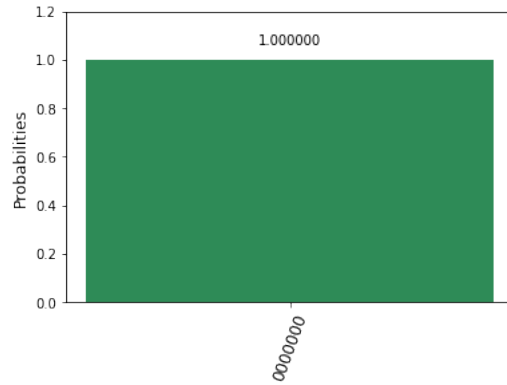
Figure 5.123.: Simulation of quantamorphism foldr XOR with input $|00000\rangle$.
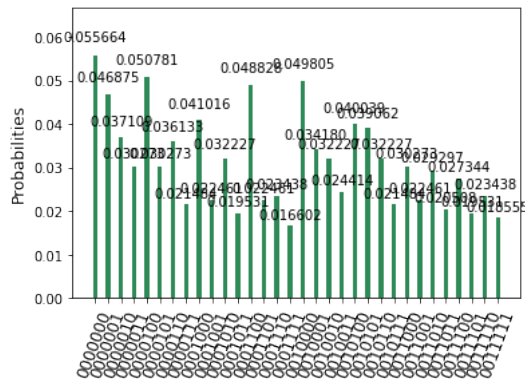
Figure 5.124.: Executing circuit of quantamorphism foldr XOR in ibmqx5.

execution on the real device in figure 5.124. The execution is mostly random, not showing any real tendency to any of the values.

This called for improvement trials, which were the adaptation and the adaptation with 8192 shots. These trials yield simulation outputs similar to those of figure 5.123 and the execution outputs of figure 5.125.

By now one could have some belief in a tendency to the right results. However, such tendency is disproved by the examples with a different initial state (figures 5.126 and 5.127).

In case of any omission, every circuit aforementioned tested in real or simulated has its QASM printed in appendix E.

5.5 DISCUSSION

OUTPUTS FROM EXECUTIONS IN THE LOCAL UNITARY SIMULATOR   There are two main factors influencing the results of this simulator, one being the LSB/MSB problem. In

(a) Results of running the circuit adapted to QISKit.

(b) Results of running the adapted circuit with 8192 shots.
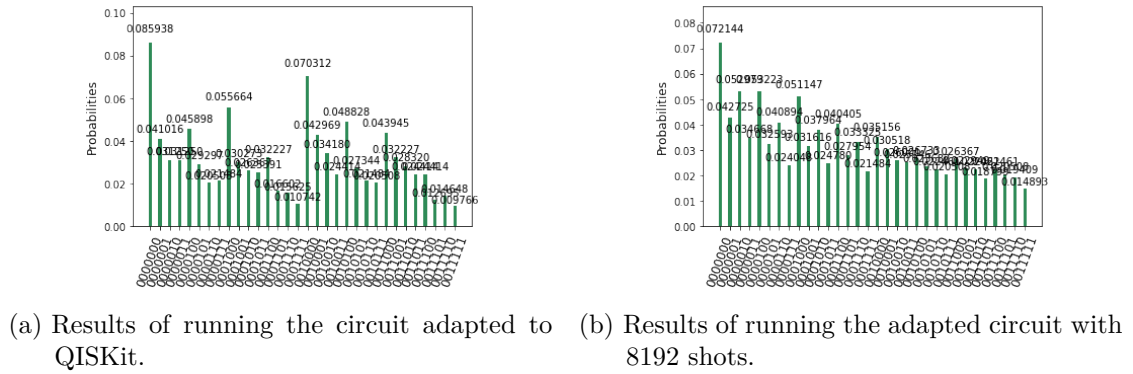
Figure 5.125.: Execution in the real device `ibmqx5` of attempts to improve the results of quantamorphism over Hadamard gate.

(a) Results of the simulation where the initial state was prepared with Hadamard gate to the target qubit.

(b) Results of the simulation where the initial state was prepared with X gate in the control qubits.

(c) Results of the simulation where the initial state was prepared with X gate in all qubits.
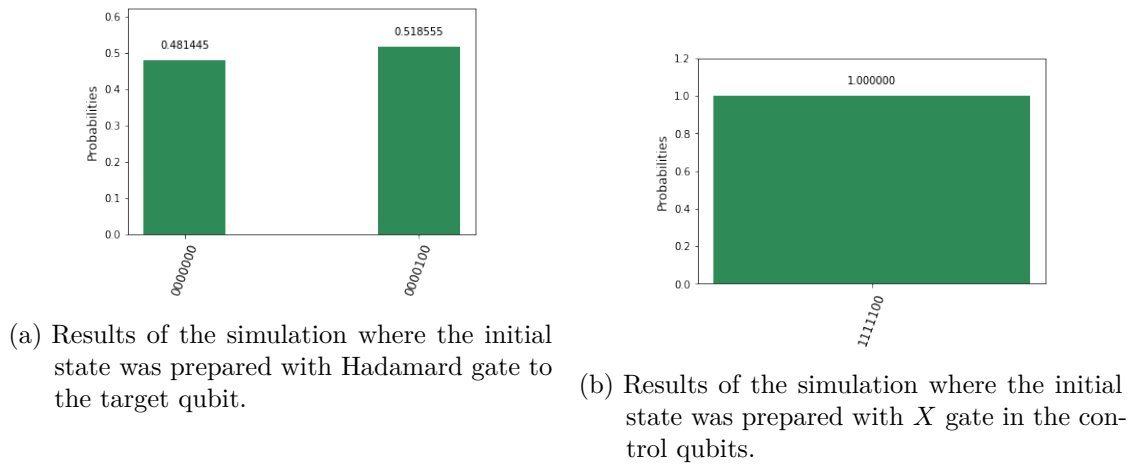
Figure 5.126.: Simulation of the experiments of quantamorphism over XOR gate adapted to QISKit.
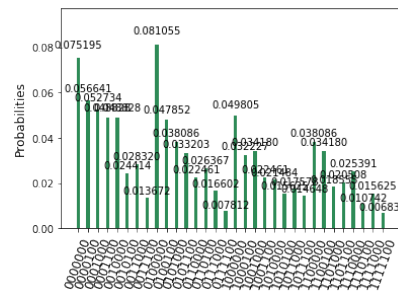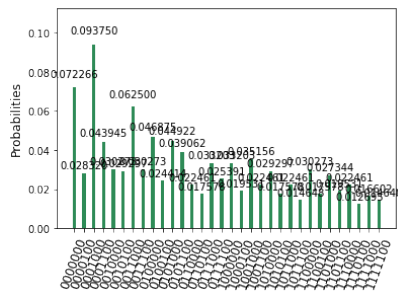
(a) Results of running the circuit where the initial state was prepared with Hadamard gate to the target qubit.

(b) Results of running the circuit where the initial state was prepared with X gate in the control qubits.
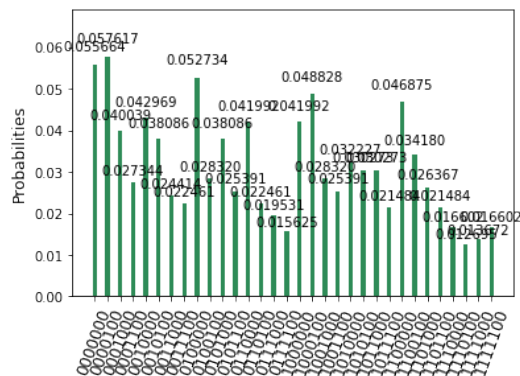
(c) Results of running the circuit where the initial state was prepared with X gate in all qubits.

Figure 5.127.: Execution in the real device `ibmqx5` of quantamorphism over XOR gate adapted to QISKit.

input			output		
q2	**q1**	**q0**	**q2**	**q1**	**q0**
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	1	0	1	1

Table 5.9.: Truth table of quantamorphism over X gate where q_2 is the LSB.

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{bmatrix} \tag{5.12}
$$

the first example (quantamorphism of X gate with two control qubits) the LSB/MSB problem is responsible for the initial unexpected matrix.

Recall the first case where we have the standard definition of the MSB and the LSB, i.e. q_2 is the least significant bit and q_0 is the most significant bit. This corresponds to having q_2 as a target qubit and q_1 and q_0 as control qubits. Truth table 5.9 and the unitary matrix (5.12) express such a circuit.

As can be seen in this first example, the number 0 — represented by the controls $q_1 = 0$ and $q_0 = 0$ — holds the value of the target, cf. the first and fifth row of table 5.9 and the first and fifth columns of matrix (5.12). Then number 1, represented by $q_1 = 1$ and $q_0 = 0$, inverts the state of q_2; since QISKit perceives qubit 2 has the most significant bit, it gives the nonstandard results in the third and seventh column of the matrix (5.12).

Otherwise, we should consider the QISKit definition of the most and least significant bit, i.e. q_2 is the MSB and q_0 is the LSB. For instance, the truth table 5.10 and the unitary matrix (5.13) correspond a circuit where q_2 and q_1 control the qubit q_0.

The other matrices cannot be entirely justified by the LSB and MSB problem. This leads to the second major element in the local unitary simulator results: the unitary matrix takes into consideration the auxiliary qubits. This means that in the cases where there is one auxiliary qubit the matrix is twice the size of the initial matrix used in Quipper.

Let us analyse the first matrix of the quantamorphism of `mqfor` of Y gate. In this case, q_0 and q_1 are the control qubits, q_2 is the target and q_3 is the ancillary. Despite the matrix corresponding to the unitary gate that leads with all possible inputs, this example assumes that the ancillary is always at 0. Since the ancillary is the most significant qubit, there is only interest in analysing the first eight columns. Rewriting the matrix with the first columns gives the matrix (5.14) which is the expected matrix for `mqfor` Y with no adaptation.

input			output		
q2	**q1**	**q0**	**q2**	**q1**	**q0**
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

Table 5.10.: Truth table of quantamorphism over X gate where q_2 is the MSB.

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix}
\tag{5.13}
$$

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \\
0 & 0 & 0 & 0 & 0 & 0 & i & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & -i & 0 & 0 & 0 & 0 \\
0 & 0 & i & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\tag{5.14}
$$

The adaptation case is even less obvious. It happens because the LSB of the second circuit is the auxiliary qubit. Every time the unitary matrix of this circuit is modifying an odd number this means that the ancillary initial state is 1. We can ignore this initial state and rewrite the matrix (5.8) obtaining the matrix (5.15) and this is an exact match to the matrix generated by GHCi.

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -i & 0 & 0 & 0 & 0 \\
0 & 0 & i & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -i \\
0 & 0 & 0 & 0 & 0 & 0 & i & 0
\end{bmatrix}
\tag{5.15}
$$

Applying this knowledge, it is clear that the matrices issued from the quantamorphism of Hadamard gate can be rewritten as (5.16) and (5.17) which corresponds to the expected.

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\
0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & \frac{1}{\sqrt{2}} \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}} & 0 \\
0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & -\frac{1}{\sqrt{2}}
\end{bmatrix}
\tag{5.16}
$$

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}
\end{bmatrix}
\tag{5.17}
$$

The matrix for the quantamorphism of XOR gate could not be handled. Note that this matrix has the initial size 32×32 and with the 2 extra qubits it would have to be a 128×128 matrix. This is too large to print in QISKit.

FROM SIMULATION TO THE REAL DEVICE   Concerning simulations, in terms of results it is clear that even in the cases where the simulation of the decomposed circuits did not work in Quipper the decomposition does not add a significant error rate. Moreover, it is important to point out that this simulation seeks to work in a similar fashion wrt. the quantum device. It adds the coin tossing effect that grants random output when measuring superposition and adds the first decomposition.

The experimental quantum computers of today are not ready to handle most of the unitary gates. The decomposition performed while compiling simulations tries to decompose the circuit in the minimal possible number of gates (Research and the IBM QX team, 2017). Some decompositions include U gates, which are advanced single gates (Research and the IBM QX team, 2017). The phase rotation gates (T, $T^\dagger$, S, $S^\dagger$ and Z) are implemented using the gate $u1$. The Hadamard gate is made with gate $u2$. The $u3$ gates are an extension of these gates and correspond to:

$$
\begin{bmatrix}
\cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\
e^{i\phi}\sin\frac{\theta}{2} & e^{i\lambda+i\phi}\cos\frac{\theta}{2}
\end{bmatrix}
$$

These initial decompositions can be adapted later to the implementation of any IBM Q Experience device. The Quipper circuit is re-written to the gate that will be likely used in the implementation.

To understand why there are differences from the circuit simulated to the one executed in real devices it is important to notice that besides the restriction in terms of gate types, there are constraints in terms of control. This demands the presence of swaps between qubits and since swaps cannot be implemented directly, they call for extra gates.

Finally, this dissertation reached the analyses of the conflicting outputs from execution in the quantum machines.

When testing the programs the overall goal is to use some gates to control other gates, and this may include entanglement. Entanglement between qubits is a primary feature of quantum computing but if the quantum system lacks isolation the quantum effect disappears. On the other hand, a perfectly isolated qubit could not be manipulated by any unitary gate. Such incompatibility makes qubit states very fragile. Chow et al. (2014) states that parameters that make the performance more vulnerable include the connectivity between qubits, the gates available and the number of gates that can be applied without error or decoherence, masking real outputs.

The programs handled in our examples are always of multiple controls over another qubit. Not only this class needs connectivity between qubits but also deals with many gates that are not directly implemented in QISKit. Both issues increase the number of gates, which in its turn increases the number of errors and the probability of decoherence. (Mind that decoherence can occur in qubits working in a classic circuit, e.g. quantamorphisms over X and XOR).

Recall that one the first decoherence problems is relaxation, whereby qubits lose their excited states (i.e. pass from 1 to 0 after some time). This problem explains why some cases tend to the right values when tested in 0000 (output expected to be 0000) but prove completely wrong in tests with different outputs.

IBM Q aims at achieving universal fault-tolerant quantum computing. Recently, the strategy has been to work with surface code with superconducting qubits (Moll et al., 2018). This is clearly a matter already studied, pointing to the need of decompositions with the minimum number of gates. However, the results show that such improvements are not enough.

The adaptation to QISKit is the most used attempt of decreasing errors throughout these experiments. This method is significant in the initial quantamorphism of X gate, irrelevant in cases like Y and prejudicial in the case of the Hadamard gate. Such differences in outcome lay in the coupling. If the circuit coupling matches the coupling in the device the decomposition to the device needs fewer gates and this entails fewer errors.

Looking back to the coupling map of figure 5.55 and comparing to figures 5.128, 5.129 and 5.130, it is clear which are the cases where the adaptation works.

(a) Coupling map of original circuit of the for-loop of X gate.

(b) Coupling map of adapted circuit of the for-loop of X gate.

Figure 5.128.: Coupling maps of the experiments in quantamorphism over X gate.
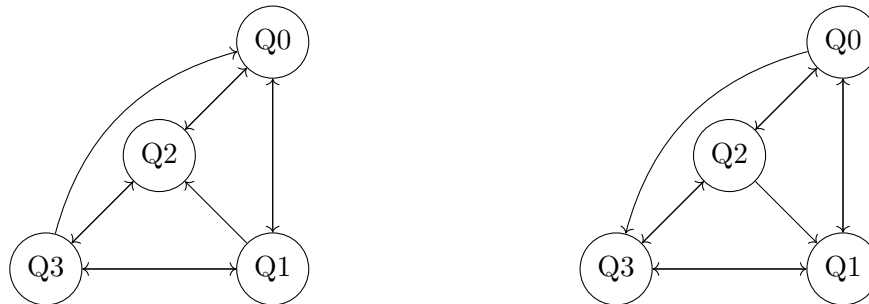
(a) Coupling map of original circuit of the for-loop of Y gate.

(b) Coupling map of adapted circuit of the for-loop of Y gate.

Figure 5.129.: Coupling maps of the experiments in quantamorphism over Y gate.

In figure 5.128 the coupling map obtained with the adaptation is precisely the coupling map of the device (figure 5.55). This is the reason to have a better improvement in the quantamorphisms over X gate with two control qubits.

In the case of quantamorphisms over Y gate (figure 5.129), the only connections that may have a considerable influence in changing the results are $Q3 \rightarrow Q0$ and $Q1 \rightarrow Q2$. The first joint does not exist in the real device and has to be decomposed in other connections. The original circuit may seem better but due to the decomposition needed the errors may not decrease in a significant way. The second link benefits from the adaptation, and thus explain the results.

In the last example (figure 5.130), the adaptation changes the connection $Q3 \rightarrow Q1$ to $Q1 \rightarrow Q3$ and the connection $Q3 \rightarrow Q1$ to $Q0 \rightarrow Q2$, these alterations result in less error rate when maintaining the original circuit.
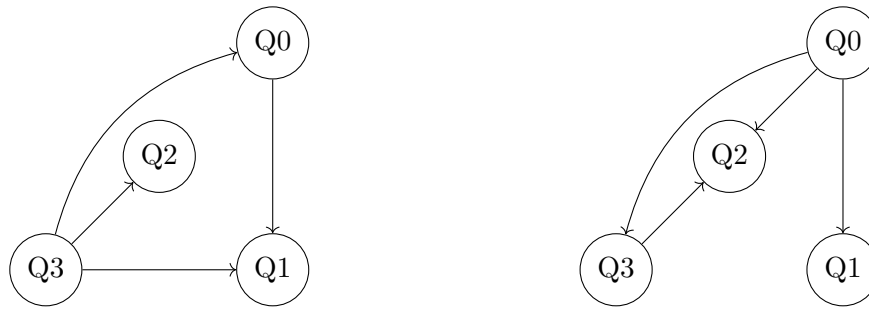
(a) Coupling map of original circuit of the for-
loop of Hadamard gate .

(b) Coupling map of adapted circuit of the for-
loop of Hadamard gate .

Figure 5.130.: Coupling maps of the experiments in quantamorphism over Hadamard gate .

It is also possible to see that the circuits over gates H and Y do not correspond to the exact coupling map of the real device and consequently will have more errors than the quantamorphism over X gate.

5.6 SUMMARY

After the description of the process of calculating recursive quantum programs (circuits) from their specifications given in the previous chapter, this chapter proves that such programs work.

The calculated programs are simulated in various ways (Quipper and QISKit) and the simulations give the expected result. Furthermore, it is possible to run such quantum programs in real quantum devices. However, most of the experimental results obtained thus, throughout this dissertation, present a big amount of errors.

Most of the faulty outputs can be explained by the problems described in section 5.1 (decoherence issues and coupling difficulties). This and other subproducts of QISKit are analysed in section 5.5.

<div align="right">6</div>

CONCLUSIONS AND FUTURE WORK

The main aim of this dissertation is to extend the standard mathematics of program construction (MPC) to quantum programming. In this context, the concept of a *quantamorphism* is proposed and elaborated. Quantamorphisms are regarded as high-level, generic specifications of quantum circuits that perform recursive computations without measurements.

Another aim of the dissertation is to actually test the quantum circuits produced from quantamorphisms on physical quantum devices available from IBM Q Experience. A toolchain is devised to fulfil the overall plan involving GHCi and Quipper (Haskell) and then the QISKit Python interface to IBM Q Experience devices.

The strategy works well for cycles (for-loops controlled by natural numbers) and *folds* over lists. However, scaling quantamorphisms beyond these input control structures remains a challenge, as mentioned in the prospect for future work given later in this chapter.

All programs calculated from quantamorphisms have been simulated and shown to return the expected outputs, confirming that runtime errors produced in the later stages of the toolchain are not part of translation from a quantum language to another (namely, from Quipper to QISKit). Regarding the quantamorphism experiments in QISKit real devices, the simpler quantum programs were successfully implemented, but the larger circuits could not be tested on real IBM Q Experience devices.

The error rate of these experiments is still notably high. It is believed that this is a result of not only the number of control gates used in the non-trivial circuits generated from quantamorphisms (which originate approximations when the qubits are not directly linked) but also of the size of the circuits. This is because qubit states are vulnerable to decoherence and relaxation, as explained in section 5.1.

Quantamorphisms are proposed as a form of recursive classic control of quantum data. This is the same principle behind the quantum programming language Quipper (Selinger, 2017) built upon work by Knill (1996). Quipper quantum software solutions typically consist of a classic device that controls a quantum device, the latter working as a *slave* that produces quantum results.

This architecture contrasts with the strategy proposed in this dissertation of delaying measurements of quantum states as much as possible, taking more advantage of quantum effects
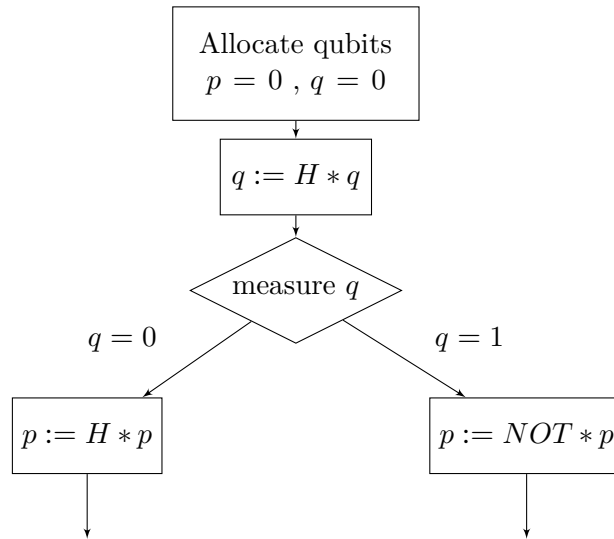
Figure 6.1.: The example of from (Yanofsky and Mannucci, 2008) page 236 there is quantum control

but raising the question: "How long can measurements be delayed?". This is one of several research questions that aroused from the work reported in this dissertation, each of them pointing to a direction for future work, as described next.

6.1 PROSPECT FOR FUTURE WORK

QUANTUM CONTROL   Altenkirch and Grattage (2005) are among the first to mention quantum control flow. Since then, others have explored this concept. For instance, Ying (2014) studies the semantics of quantum recursion with quantum control flow using quantum case statements and quantum choices; Bădescu and Panangaden (2015) include quantum alteration in a quantum programming language semantics to express *if-statements*; and recently, Sabry et al. (2018) apply pattern-matching to capture not only *if-then-else* but also quantum loops.

A simple example with classical control *if-then-else* of quantum computation can be found in the flow graph diagram of figure 6.1, extracted from (Nielsen and Chuang, 2011). Note that the diagram performs a choice between two alternative quantum computations after a measurement of the outcome of q.

Initial State Preparation		Output Measure		Probability original circuit (%)	Probability decomposed circuit (%)
q_0	q_1	q_0	q_1		
		1	1	50.000006	50
0	0	1	0	24.999997	25
		0	0	24.999997	25

Table 6.1.: Results of Quipper simulation of *quantum_if* circuit.
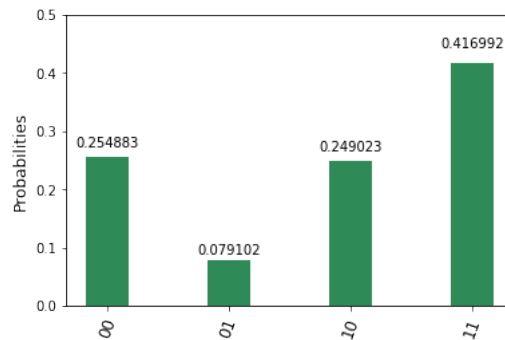
Figure 6.2.: Output of the *quantum_if* circuit adapted to QISKit with $q = 0$ and $p = 0$ in the `ibmqx4` device.

This contrasts with the way the same conditional control is expressed by the following monadic program, written in Haskell:

$$quantum\_if \ h \quad (p, q) = \mathbf{do}\{$$
$$q' \leftarrow h \ q;$$
$$p' \leftarrow \mathbf{if} \ q' \ \mathbf{then} \ \text{return} \ (\neg \ p) \ \mathbf{else} \ h \ p;$$
$$\text{return} \ (p', q')$$
$$\}$$

Note that the conditional statement in this program is classic. What is happening here is a *superposition* of many such classic conditional controls, originating from $h \ q$.

Similarly to what has been done concerning quantamorphisms in chapter 5, such conditional monadic program corresponds to a unitary matrix that can be simulated (6.1) and tested (6.2). Both circuits (original and decomposed) have been simulated, see the outcome in table 6.1. However, tests didn't cover the circuit with measurements, so no comparison between the two has been performed.

Although some still regard a quantum computer without a classical master (Selinger, 2004) unlikely in the future, a calculus for quantum programming should consider this kind of control structures.

TOWARDS MORE GENERIC QUANTAMORPHISMS   Besides extending quantamorphisms so as to cope with conditional quantum control, there is a need for an extension to inductive types other than natural numbers and finite lists. This generalization is a challenge for future work, possibly inspired by algorithmic structures found in other areas of programming. Take for instance (section 4.1) the generalisation of π_1-complementation that leads to the chaining of such computations. This can be regarded as a special use of a device known as *accumulating map* in the functional programming community (Jaskelioff and Rypacek, 2012). This device can be found, as conspicuously noted by Olah (2015), in neural networks (Hermida and Jacobs, 1998). In Haskell, function `mapAccumR` can handle so-called *traversable* structures and these may offer a path to generalising π_1-complementation.

TOOL CHAIN   The tool-chain used in our experimental setup uses GHCi and Quipper. As both run Haskell programs, merging these two first blocks of the toolchain seems viable and interesting to explore.

Achieving this will require a thorough analyse of Quipper recursive circuit implementation (Eisenberg et al., 2016). It is likely that implementing circuits with these methods will result in a larger initial circuit than the circuit implemented through a matrix. Similar to classical reversible programs, quantum programs (reversible by definition) tend to add a substantial amount of garbage.

ACADEMIC PARTNERSHIP BETWEEN QUANTALAB AND IBM   This partnership, started last September, will open new opportunities, with respect to the Q Experience resources available for experimentation. Researchers at the University of Minho, INESC TEC, CEiiA and INL are now able to test the circuits of the IBM Q 20 Tokyo device. These devices are more powerful not only due to the larger number of qubits but also due to the way qubits are coupled to each other. Despite the number of qubits and the connections among them, IBM researchers claim fewer errors in these devices (Miller, 2017).

Recall the quantamorphisms over the Hadamard gate. Running the generated circuits in a real device shows evidence of decoherence problems, but still tending to the correct values. A quick test of one of these circuits in the 20 qubit machine already shows progress, see figure 6.3.[1]

These machines are still in an initial stage and significant enhancements to the systems took place while writing this dissertation. Some important functions in QISKit were altered

---

[1]Other examples, e.g. quantamorphisms over Y gate, do not show any relevant improvements.
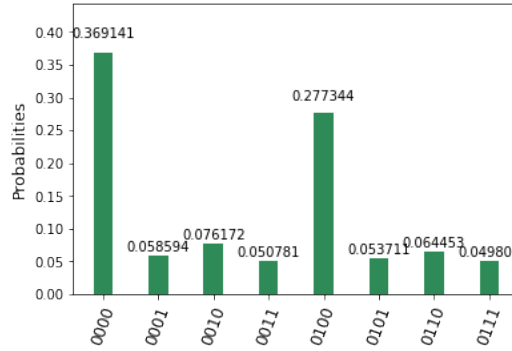
(a) Results of the circuit in figure 5.110. Correct measures improved from 38.48% to 36.91%.
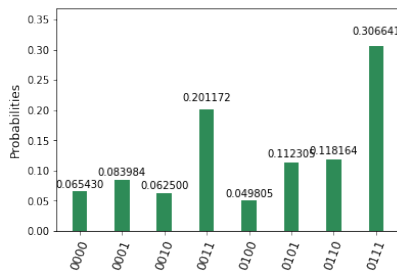
(b) Results the circuit in figure 5.122a. Correct measures improved from 29.69% to 50.78%.
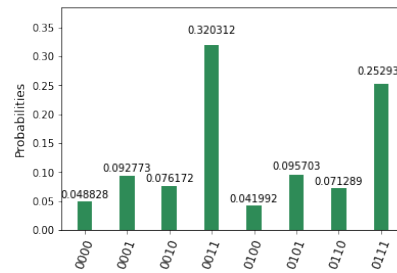
(c) Results of the circuit in figure 5.122d. Correct measures improved from 37.89% to 57.32%.
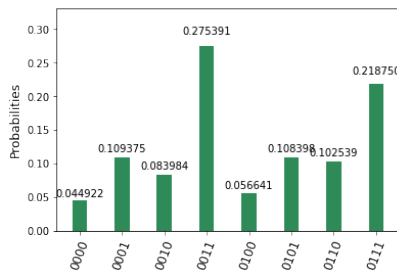
(d) Results of the circuit in figure 5.122c. Correct measures improved from 24.41% to 32.03%.
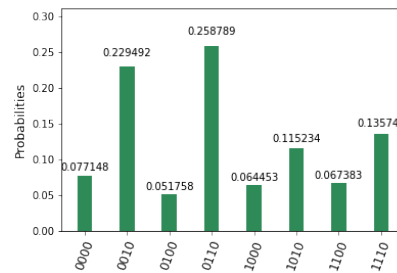
(e) Results of the circuit in figure 5.122b. The number of correct measure went from 15.92% to 25.88%.

Figure 6.3.: Experimenting a quantamorphism over the Hadamard gate in the IBM device with 20 qubits.

too, bugs were removed, and the whole system had a huge update. Consequently, some of the circuits could be re-tested near the end of this research work and show lesser error rates. Moreover, some circuits that did not run previously, namely quantamorphisms over XOR, now do so and can be tested.

Such fast advances in such a short time increase confidence with respect to the follow up of this work — better results are expected by re-testing the work already reported, while possibly encountering other unforeseen limitations of quantum devices.

BIBLIOGRAPHY

© Volkswagen AG 2018. Volkswagen group and google work together on quantum computer. `https://www.volkswagenag.com/en/news/2017/11/quantum-computing.html`, 11 2017. (Accessed on 08/03/2018).

S. Abramsky and N. Tzevelekos. Introduction to categories and categorical logic. *Lecture Notes in Physics*, 813:3–94, 2011. ISSN 00758450. doi: 10.1007/978-3-642-12821-9_1.

S. Abramsky, R.S. Barbosa, N. Silva, and O. Zapata. The quantum monad on relational structures. *CoRR*, abs/1705.07310, 2017. URL `http://arxiv.org/abs/1705.07310`.

T. Altenkirch and J. Grattage. A functional quantum programming language. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 249–258, 2005. doi: 10.1109/LICS.2005.1. URL `https://doi.org/10.1109/LICS.2005.1`.

S. Awodey. *Category Theory*. Oxford University Press, Inc., New York, NY, USA, 2nd edition, 2010. ISBN 0199237182, 9780199237180.

R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.

F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981. doi: 10.1145/319628.319634. URL `http://doi.acm.org/10.1145/319628.319634`.

C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, November 1973. ISSN 0018-8646. doi: 10.1147/rd.176.0525. URL `http://dx.doi.org/10.1147/rd.176.0525`.

R. S. Bird and O. de Moor. *Algebra of programming*. Prentice Hall International series in computer science. Prentice Hall, 1997. ISBN 978-0-13-507245-5.

A. Breitweiser, J. Chow, A. Córcoles, A. Cross, A. Cross, V. Dwyer, M. Everitt, I. Faro, A. Frisch, A. Fuhrer, J. Gambetta, T. Imamichi, A. Javadi, A. Mezzacapo, R. Movassagh, A. D. Perruzzi, A. Phan, R. Raymond, R. Rundle, N. Sathaye, K. Temme, T. Tilma, C. Wood, and J. Wootton. Github - qiskit/qiskit-tutorial: A collection of jupyter notebooks using qiskit. `https://github.com/QISKit/qiskit-tutorial`, 2018. (Accessed on 08/08/2018).

C. Bădescu and P. Panangaden. Quantum Alternation: Prospects and Problems. *Electronic Proceedings in Theoretical Computer Science*, 195(Qpl):33–42, 2015. ISSN 2075-2180. doi: 10.4204/EPTCS.195.3. URL http://arxiv.org/abs/1511.01567.

J. M. Chow, J. M. Gambetta, E. Magesan, D. W. Abraham, A. W. Cross, B. R. Johnson, N. A. Masluk, C. A. Ryan, J. A. Smolin, S. J. Srinivasan, and M. Steffen. Implementing a strand of a scalable fault-tolerant quantum computing fabric. *Nature Communications*, 5: 1–9, 2014. ISSN 20411723. doi: 10.1038/ncomms5015.

A. Church. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics*, 58(2):345, 1936. ISSN 00029327. doi: 10.2307/2371045. URL http://www.jstor.org/stable/2371045?origin=crossref.

B. Coecke, editor. *New Structures for Physics.* Number 831 in Lecture Notes in Physics. Springer-Verlag, 2011.

B. Coecke and A. Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.* Cambridge University Press, 2017. ISBN 9781316219317. doi: 10.1017/9781316219317. URL https://doi.org/10.1017/9781316219317.

A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open Quantum Assembly Language. *ArXiv e-prints*, July 2017.

D. Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 400(1818):97–117, 1985. ISSN 0080-4630. doi: 10.1098/rspa.1985.0070. URL http://rspa.royalsocietypublishing.org/content/400/1818/97.

G W Dueck and D Maslov. Reversible Function Synthesis with Minimum Garbage Outputs. *International Symposium on Representations and Methodology of Future Computing Technologies*, pages 154–161, 2003.

R. Eisenberg, A. S. Green, P. L. Lumsdaine, K. Kim (ACS), S. Mau (ACS), B. Mohan, W. Ng (ACS), J. Ravelomanantsoa-Ratsimihah, N. J. Ross, A. Scherer (ACS), P. Selinger, B. Valiron, A. Virodov (ACS), and S. A. Zdancewic. The quipper language. https://www.mathstat.dal.ca/~selinger/quipper/, 2016. (Accessed on 08/09/2018).

M. Erwig and S. Kollmannsberger. Functional pearls: Probabilistic functional programming in Haskell. *J. Funct. Program.*, 16:21–34, January 2006.

P.J. Freyd and A. Scedrov. *Categories, allegories.* North-Holland mathematical library 39. North-Holland, 1 edition, 1990. ISBN 9780444703682,0444703683,9780080887012,0444703683-:,0444703675.

S. Gasiorowicz. *Quantum Physics*. John Wiley & Sons, 2003. ISBN 9780471057000. URL https://books.google.pt/books?id=NXMtCwAAQBAJ.

A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. An introduction to quantum programming in quipper. In *Reversible Computation - 5th International Conference, RC 2013, Victoria, BC, Canada, July 4-5, 2013. Proceedings*, pages 110–124, 2013a. doi: 10.1007/978-3-642-38986-3_10. URL https://doi.org/10.1007/978-3-642-38986-3_10.

A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron. Quipper: A scalable quantum programming language. *CoRR*, abs/1304.3390, 2013b. URL http://arxiv.org/abs/1304.3390.

I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4):1–36, 2007.

C. Hermida and B. Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation*, 145(2):107–152, 1998. ISSN 08905401. doi: 10.1006/inco.1998.2725.

O.A. Ivanova. Unitary matrix - encyclopedia of mathematics. https://www.encyclopediaofmath.org/index.php/Unitary_matrix, 02 2011. (Accessed on 08/10/2018).

M. Jaskelioff and O. Rypacek. An Investigation of the Laws of Traversals. *Electronic Proceedings in Theoretical Computer Science*, 76(Msfp):40–49, 2012. ISSN 2075-2180. doi: 10.4204/EPTCS.76.5.

A. Javadi-Abhari, J. M. Gambetta, and A. Cross. Program quantum computers more easily with qiskit 0.5. https://medium.com/qiskit/program-quantum-computers-more-easily-with-qiskit-0-5-802d1e4a338d, 2018. Accessed: 2018-07-30.

E Knill. Conventions for quantum pseudocode. *Quantum*, 2724(LAUR-96-2724):1–13, 1996. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.1328{&}rep=rep1{&}type=pdf.

J. Kramer. Is abstraction the key to computing? *Commun. ACM*, 50(4):36–42, 2007. doi: 10.1145/1232743.1232745. URL http://doi.acm.org/10.1145/1232743.1232745.

R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961. doi: 10.1147/rd.53.0183. URL https://doi.org/10.1147/rd.53.0183.

D. Laws. Who invented the transistor? | computer history museum. `http://www.computerhistory.org/atchm/who-invented-the-transistor/`, December 2013. (Accessed on 09/05/2018).

M. López-Suárez, I. Neri, and L. Gammaitoni. Sub-kBT micro-electromechanical irreversible logic gate. *Nature Communications*, 7(May), 2016. ISSN 20411723. doi: 10.1038/ncomms12068.

S. MacLane. *Categories for the Working Mathematician.* Springer-Verlag, 1971.

E.G. Manes and M.A. Arbib. *Algebraic Approaches to Program Semantics.* Texts and Monographs in Computer Science. Springer-Verlag, 1986. D. Gries, series editor.

K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming, ICFP 2007, Freiburg, Germany, October 1-3, 2007*, pages 47–58, 2007. doi: 10.1145/1291151.1291162. URL `http://doi.acm.org/10.1145/1291151.1291162`.

R. Miller. Ibm makes 20 qubit quantum computing machine available as a cloud service | techcrunch. `https://techcrunch.com/2017/11/10/ibm-passes-major-milestone-with-20-and-50-qubit-quantum-computers-as-a-service/`, Nov 2017. (Accessed on 08/18/2018).

N. Moll, P. Barkoutsos, L. S. Bishop, J. M. Chow, A. Cross, D. J. Egger, S. Filipp, A. Fuhrer, J. M. Gambetta, M. Ganzhorn, A. Kandala, A. Mezzacapo, P. Müller, W. Riess, G. Salis, J. Smolin, I. Tavernelli, and K. Temme. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):1–30, 2018. ISSN 20589565. doi: 10.1088/2058-9565/aab822.

S. Mu, Z. Hu, and M. Takeichi. An injective language for reversible computation. In *Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings*, pages 289–313, 2004. doi: 10.1007/978-3-540-27764-4_16. URL `https://doi.org/10.1007/978-3-540-27764-4_16`.

D. Murta and J.N. Oliveira. A study of risk-aware program transformation. *SCP*, 110:51–77, 2015.

F. Nebeker. *Dawn of the Electronic Age: Electrical Technologies in the Shaping of the Modern World, 1914 to 1945.* Wiley-IEEE Press, 2009. ISBN 0470260653, 9780470260654.

A. Neri. Qiskit experiments, 2018. GitHub project, `https://github.com/arcalab/quantamorphisms/tree/master/Qiskit_tests`. (Accessed on 02/10/2018).

A. Neri and A. Rodrigues. Quippertoqiskit tool, 2018. GitHub project, `https://github.com/arcalab/quantamorphisms/tree/master/Tool-QuipperToQiskit`. (Accessed on 02/10/2018).

M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, New York, NY, USA, 10th edition, 2011. ISBN 1107002176, 9781107002173.

C. Olah. Neural networks, types, and functional programming – colah's blog. `http://colah.github.io/posts/2015-09-NN-Types-FP/`, 02 2015. (Accessed on 08/10/2018).

J. N. Oliveira. Towards a linear algebra of programming. *Formal Asp. Comput.*, 24(4-6):433–458, 2012. doi: 10.1007/s00165-012-0240-9. URL `https://doi.org/10.1007/s00165-012-0240-9`.

J.N. Oliveira. Program design by calculation. `http://www4.di.uminho.pt/~jno/ps/pdbc_part.pdf`, 2008. Draft of textbook in preparation, current version: April 2018.

J.N. Oliveira. A relation-algebraic approach to the "Hoare logic" of functional dependencies. *JLAP*, 83(2):249–262, 2014.

J.N. Oliveira. 1st Module : Category theory for the software sciences, 2017. Course *Algebraic and Coalgebraic Methods in Software Development*, MAPi PhD Programme, lectured by A. Madeira, L. Barbosa, D. Hofmann, A. Martins and J. Oliveira.

J.N. Oliveira. Compiling quantamorphisms for the IBM Q-experience, July 2018. Talk at the IFIP WG 2.1 #77 Meeting, Brandenburg (Germany). Joint work with A. Neri and R.S. Barbosa.

J.N. Oliveira and V.C. Miraldo. "Keep definition, change category" — a practical approach to state-based system calculi. *JLAMP*, 85(4):449–474, 2016.

J.N. Oliveira and C.J. Rodrigues. Transposing relations: From maybe functions to hash tables. In *Mathematics of Program Construction, 7th International Conference, MPC 2004, Stirling, Scotland, UK, July 12-14, 2004, Proceedings*, pages 334–356, 2004. URL `https://doi.org/10.1007/978-3-540-27764-4_18`.

J.N. Oliveira and C.J. Rodrigues. Pointfree Factorization of Operation Refinement. In *FM'06*, volume 4085 of *LNCS*, pages 236–251. Springer-Verlag, 2006.

IBM Research and the IBM QX team. User guide — experience documentation 2.0 documentation. `https://qiskit.github.io/ibmqx-user-guides/full-user-guide/introduction.html`, 2017. (Accessed on 09/10/2018).

S. K. Routray. History of Electronics. *The 2004 IEEE Conference on the History of Electronics (CHE2004)*, 2004. URL http://www.ieeeghn.org/wiki/images/8/87/Routray.pdf.

A. Sabry, B. Valiron, and J. K. Vizzotto. From symmetric pattern-matching to quantum control. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, pages 348–364, Cham, 2018. Springer International Publishing. ISBN 978-3-319-89366-2.

M. Sandberg, H. Paik, A. Córcoles, D. McClure, and J. Gambetta. qiskit-tutorial/relaxation_and_decoherence.ipynb at master · qiskit/qiskit-tutorial · github. https://github.com/Qiskit/qiskit-tutorial/blob/master/reference/qcvv/relaxation_and_decoherence.ipynb, 07 2018. (Accessed on 08/09/2018).

P. Selinger. Towards a quantum programming language. *Mathematical. Structures in Comp. Sci.*, 14(4):527–586, August 2004. ISSN 0960-1295. URL https://doi.org/10.1017/S0960129504004256.

P. Selinger. Quantum programming. http://alfa.di.uminho.pt/~nevrenato/probprogschool_slides/Peter.pdf, 2017.

Shifter. Universidade do Minho é a primeira a integrar a rede de computação quântica da IBM. https://shifter.pt/2018/06/universidade-do-minho-rede-de-computacao-quantica-ibm/, 06 2018. (Accessed on 08/02/2018).

N. Silk. Github: quipper/quantumsimulation.hs at master · silky/quipper. https://github.com/silky/quipper/blob/master/QuipperLib/Simulation/QuantumSimulation.hs, Nov 2016. (Accessed on 01/09/2018).

S. H. Simon, N. E. Bonesteel, M. H. Freedman, N. Petrovic, and L. Hormozi. Topological quantum computing with only one mobile quasiparticle. *Physical Review Letters*, 96(7):1–4, 2006. ISSN 00319007. doi: 10.1103/PhysRevLett.96.070503.

IBM Q Team. Ibm q network - quantum computing. https://www.research.ibm.com/ibm-q/network/, 2018. (Accessed on 08/14/2018).

IBM Q team. Ibm q 16 rueschlikon v1.x.x, 2018. URL https://github.com/Qiskit/qiskit-backend-information/tree/master/backends/rueschlikon/V1.

A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936. URL http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf.

J. Vicary. Topological structure of quantum algorithms. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 93–102, 2013. doi: 10.1109/LICS.2013.14. URL https://doi.org/10.1109/LICS.2013.14.

X. L. Wang, Y. H. Luo, H. L. Huang, M. C. Chen, Z. E. Su, C. Liu, C. Chen, W. Li, Y. Q. Fang, X. Jiang, J. Zhang, L. Li, N. L. Liu, C. Y. Lu, and J. W. Pan. 18-Qubit Entanglement with Six Photons' Three Degrees of Freedom. *Physical Review Letters*, 120(26), jan 2018. ISSN 10797114. doi: 10.1103/PhysRevLett.120.260502. URL http://arxiv.org/abs/1801.04043http://dx.doi.org/10.1103/PhysRevLett.120.260502.

E. W. Weisstein. Matrix trace – from mathworld– a wolfram web resource. http://mathworld.wolfram.com/MatrixTrace.html, 2018. (Accessed on 09/26/2018).

N. S. Yanofsky and M. A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008. doi: 10.1017/CBO9780511813887.

M. Ying. Foundations of quantum programming (extended abstract). In *Programming Languages and Systems - 8th Asian Symposium, APLAS 2010, Shanghai, China, November 28 - December 1, 2010. Proceedings*, pages 16–20, 2010. doi: 10.1007/978-3-642-17164-2_2. URL https://doi.org/10.1007/978-3-642-17164-2_2.

M. Ying. Quantum recursion and second quantisation: Basic ideas and examples. *CoRR*, abs/1405.4443, 2014. URL http://arxiv.org/abs/1405.4443.

W. Zeng. Models of quantum algorithms in sets and relations. *CoRR*, abs/1503.05857, 2015. URL http://arxiv.org/abs/1503.05857.

LAWS OF THE ALGEBRA OF PROGRAMMING

FUNCTIONS

Natural-id:	$f \cdot id = id \cdot f = f$	(A.1)
Associative composition:	$(f \cdot g) \cdot h = f \cdot (g \cdot h)$	(A.2)
Natual-Constante:	$\underline{k} \cdot f = \underline{k}$	(A.3)
Fusion-Constante:	$f \cdot \underline{k} = \underline{fk}$	(A.4)
Leibniz:	$f \cdot h = g \cdot h \Leftarrow f = g$	(A.5)

PRODUCT

Universal-×:
$$k = f \triangledown g \Leftrightarrow \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases} \tag{A.6}$$

Cancellation-×:
$$\begin{cases} \pi_1 \cdot f \triangledown g = f \\ \pi_2 \cdot f \triangledown g = g \end{cases} \tag{A.7}$$

Reflection-×:	$\pi_1 \triangledown \pi_2 = id_{A \times B}$	(A.8)
Fusion-×:	$g \triangledown h \cdot f = g \cdot f \triangledown h \cdot f$	(A.9)
Definition-×:	$f \times g = f \cdot \pi_1 \triangledown g \cdot \pi_2$	(A.10)
Absorption-×:	$(i \times j) \cdot g \triangledown h = i \cdot g \triangledown j \cdot h$	(A.11)
Natural-π_1:	$\pi_1 \cdot (f \times g) = f \cdot \pi_1$	(A.12)
Natural-π_2:	$\pi_2 \cdot (f \times g) = g \cdot \pi_2$	(A.13)
Functor-×:	$(g \cdot h) \times (i \cdot j) = (g \times i) \cdot (h \times j)$	(A.14)
Functor-id-×:	$id_A \times id_B = id_{A \times B}$	(A.15)

Eq-×:
$$f \triangledown g = h \triangledown k \Leftrightarrow \begin{cases} f = h \\ g = k \end{cases} \tag{A.16}$$

COPRODUCT

Universal-+:	$k = [f, g] \Leftrightarrow \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases}$	(A.17)
Cancellation-+:	$\begin{cases} [f, g] \cdot i_1 = f \\ [f, g] \cdot i_2 = g \end{cases}$	(A.18)
Reflection-+:	$[i_1, i_2] = id_{A+B}$	(A.19)
Fusion-+:	$f \cdot [g, h] = [g \cdot f, h \cdot f]$	(A.20)
Definition-+:	$f + g = [i_1 \cdot f, i_2 \cdot g]$	(A.21)
Absorption-+:	$[g, h] \cdot (i + j) = [i \cdot g, j \cdot h]$	(A.22)
Natural-i_1:	$(i + j) \cdot i_1 = i_1 \cdot i$	(A.23)
Natural-i_2:	$(i + j) \cdot i_2 = i_2 \cdot j$	(A.24)
Functor-+:	$(g \cdot h) + (i \cdot j) = (g + i) \cdot (h + j)$	(A.25)
Functor-id-+:	$id_A + id_B = id_{A+B}$	(A.26)
Eq-+:	$[f, g] = [h, k] \Leftrightarrow \begin{cases} f = h \\ g = k \end{cases}$	(A.27)

PRODUCT AND COPRODUCT

Exchange law:	$[f \triangledown g, h \triangledown k] = [f, h] \triangledown [g, k]$	(A.28)

FUNCTORS

Functor-F:	$\mathrm{F}(g \cdot h) = (\mathrm{F}g) \cdot (\mathrm{F}h)$	(A.29)
Functor-id-F:	$\mathrm{F}id_A = id_{\mathrm{F}A}$	(A.30)

CATAMORPHISMS

Universal-cata:	$k = (\!|g|\!) \Leftrightarrow k \cdot \mathbf{in} = g \cdot \mathrm{F}k$	(A.31)
Cancellation-cata:	$(\!|g|\!) \cdot \mathbf{in} = g\mathrm{F} \cdot (\!|g|\!)$	(A.32)
Reflection-cata:	$(\!|\mathbf{in}|\!) = id_{\mathrm{T}}$	(A.33)
Fusion-cata:	$f \cdot (\!|g|\!) = (\!|h|\!) \Leftarrow f \cdot g = h \cdot \mathrm{F}f$	(A.34)
Definition-map-cata:	$\mathrm{T}f = (\!|\mathbf{in} \cdot \mathrm{B}(f, id)|\!)$	(A.35)
Absorption-cata:	$(\!|g|\!) \cdot \mathrm{T}f = (\!|g \cdot \mathrm{B}(id, f)|\!)$	(A.36)
Base-cata:	$\mathrm{F}f = \mathrm{B}(id, f)$	(A.37)

MUTUAL RECURSION

Fokkinga:
$$\begin{cases} f \cdot \mathbf{in} = h \cdot \mathrm{F}f \mathbin{\triangledown} g \\ g \cdot \mathbf{in} = k \cdot \mathrm{F}f \mathbin{\triangledown} g \end{cases} \Leftrightarrow f \mathbin{\triangledown} g = (\!| h \mathbin{\triangledown} k |\!) \tag{A.38}$$

Banana-split: $(\!| i |\!) \mathbin{\triangledown} (\!| j |\!) = (\!| (i \times j) \cdot \mathrm{F}\pi_1 \mathbin{\triangledown} \mathrm{F}\pi_2 |\!)$ $\tag{A.39}$

MONADS

Multiplication:	$\mu \cdot \mu = \mu \cdot \mathrm{T}\mu$	(A.40)
Unity:	$\mu \cdot u = \mu \cdot \mathrm{T}u = id$	(A.41)
Natual-u:	$u \cdot f = \mathrm{T}f \cdot u$	(A.42)
Natural-μ:	$\mu \cdot \mathrm{T}(\mathrm{T}f) = \mathrm{T}f \cdot \mu$	(A.43)
Monodic Composition:	$f \bullet g = \mu \mathrm{T}f \cdot g$	(A.44)
Associativity-$\bullet$:	$(f \bullet g) \bullet h = f \bullet (g \bullet h)$	(A.45)
Identity-$\bullet$:	$u \bullet f = f = f \bullet u$	(A.46)
Associativity-$\bullet/\cdot$:	$(f \bullet g) \cdot h = f \bullet (g \cdot h)$	(A.47)
Associativity-$\cdot/\bullet$:	$(f \cdot g) \bullet h = f \bullet (\mathrm{T}g \bullet h)$	(A.48)
μ **versos** $\bullet$:	$id \bullet id = \mu$	(A.49)
"μ **as binding** ":	$\mu x = x \ggg= id$	(A.50)
"**Binding as** μ":	$x \ggg= f = x \ggg= \underline{y}$	(A.51)
Sequencing:	$x \gg y = x \ggg= \underline{y}$	(A.52)
do-notation:	$\mathbf{do}\{x \leftarrow a; b\}a \ggg= (\lambda x \to b)$	(A.53)

OTHER DEFINITIONS

Equality:	$f = g \Leftrightarrow \langle \forall x :: fx = gx \rangle$	(A.54)
Definition Composition:	$(f \cdot g)x = f(gx)$	(A.55)
Definition Identity:	$id\ x = x$	(A.56)
Definition Constante:	$\underline{k}x = k$	(A.57)
λ-Notation:	$fa = b \Leftrightarrow f = \lambda a \leftarrow b$	(A.58)
Definition Split:	$f \mathbin{\triangledown} gx = (fx, gx)$	(A.59)
Definition $\times$:	$(f \times g)(a, b) = (fa, gb)$	(A.60)
Definition Conditional:	$(p \rightarrow f, g)x = \textbf{if } p\ x \textbf{ then } f\ x \textbf{ else } g\ x$	(A.61)
Definition Projection:	$\pi_1(x, y) = x \wedge \pi_2(x, y) = y$	(A.62)
let-Elim:	$\textbf{let } x = a \textbf{ in } b = b[x, a]$	(A.63)
pair-Elim:	$t = t[(x, y)/z, x/\pi_1 z, y/\pi_2 z]$	(A.64)
Definition ap:	$ap(f, x) = f\ x$	(A.65)
Curry:	$\overline{f}ab = f(a, b)$	(A.66)
Uncurry:	$\widehat{f}(a, b) = fab$	(A.67)

B

QUANTAMORPHISM OVER X GATE

```
   import System.Exit
2
   import Quipper
4
   import Quantum.Synthesis.Matrix
6  import Quantum.Synthesis.Ring
   import QuipperLib.Synthesis
8
   import Data.Complex
10
   -- import modules for simulations
12 import qualified Data.Map as Map
   import QuipperLib.Simulation
14 import System.Random
16 -- declare sample_oracle's data type
   data Oracle = Oracle {
18    qubit_num :: Int ,
      function :: ([Qubit], Qubit) -> Circ ([Qubit], Qubit)
20 }
22 -- declare circuit function
   circuit_function :: Oracle -> Circ ([Bit], Bit)
24 circuit_function oracle = do
        -- initialize string of qubits
26      top_qubits <- qinit (replicate (qubit_num oracle) False)
        bottom_qubit <- qinit True
28      label (top_qubits, bottom_qubit) ("|0>","|1>")
30      -- set the initial states of the qubits
        mapUnary hadamard top_qubits
32
        comment "before oracle"
34      -- call oracle
```

```
        function oracle (top_qubits, bottom_qubit)
36      comment "after oracle"

38      -- measure qubits
        (top_qubits, bottom_qubit) <- measure (top_qubits, bottom_qubit)
40      -- discard unnecessary output and return result
        return (top_qubits, bottom_qubit)
42

   -- From a matrix
44 mymatrix :: Matrix Eight Eight (Integer)
   mymatrix = matrix_of_function f
46   where
     f i j
48     | i == 0 && j == 0 = 1
       | i == 1 && j == 1 = 1
50     | i == 2 && j == 3 = 1
       | i == 3 && j == 2 = 1
52     | i == 4 && j == 4 = 1
       | i == 5 && j == 5 = 1
54     | i == 6 && j == 7 = 1
       | i == 7 && j == 6 = 1
56     | otherwise  = 0


58 synthesized = exact_synthesis mymatrix

60 circuit ::([Qubit], Qubit)-> Circ ([Qubit], Qubit)
   circuit ([a, b], c) = do
62   synthesized [a,b,c]
     return ([a,b],c)
64

   -- simulate function
66 simulate :: Circ ([Bit],Bit) -> IO ()
   simulate circuito = print (sim_generic (1.0::Float) circuito)
68

   main1 :: IO ()
70 main1 = do
     print_generic Preview (circuit_function my_oracle)
72      where
       -- declare empty_oracle's data type
74     my_oracle :: Oracle
       my_oracle = Oracle {
76       -- set the length of qubit string
         qubit_num = 2,
78       function = circuit
       }
80

   main2 :: IO ()
```

```
82   main2 = do
       print_generic ASCII (circuit_function my_oracle)
84         where
           -- declare empty_oracle's data type
86         my_oracle :: Oracle
           my_oracle = Oracle {
88             -- set the length of qubit string
             qubit_num = 2,
90           function = circuit
           }
92

     main3 :: IO ()
94   main3 = do
       simulate (circuit_function my_oracle)
96     where
        -- declare empty_oracle's data type
98     my_oracle :: Oracle
       my_oracle = Oracle {
100       -- set the length of qubit string
         qubit_num = 2,
102       function = circuit
       }
104

     main = do
106
       putStrLn "\n \n choose an option:"
108    putStrLn " 1 - PDF circuit\n"
       putStrLn " 2 - text description of circuit\n"
110    putStrLn " 3 - simulation\n"
       putStrLn " 4 - exit\n"
112    line <- getLine
       case line of
114      "1" -> do main1
                   main
116      "2" -> do main2
                   main
118      "3" -> do main3
                   main
120      "4" -> do exitSuccess
         _ -> do main
122
       -- if you want the circuit description a .txt file
124    -- comment the menu above
       -- replacing it by:
126    -- main2
       -- and run:
128    -- $ ./quanta_x.hs > matrix_x_quipper.txt
```

Listing B.1: Quipper program for quantamorphism over X gate with n = 3.

The following examples are very similar to the first case, hence the following examples only exhibit the section of the script that requires changes.

```
1  import QuipperLib.Decompose.GateBase
   import QuipperLib.Decompose
3
   import Libraries.RandomSource
```

Listing B.2: Quipper program for quantamorphism over X gate with n = 7 - import modules for decomposition.

```
import Control.Monad (replicateM)
```

Listing B.3: Quipper program for quantamorphism over X gate with n = 7 - import extra modules for simulation.

```
   type Sixteen = Ten_and Six
2

4  mymatrix :: Matrix Sixteen Sixteen (Integer)
   mymatrix = matrix [[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
6                     [0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0],
                      [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0],
8                     [0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0],
                      [0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0],
10                    [0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0],
                      [0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0],
12                    [0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0],
                      [0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0],
14                    [0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0],
                      [0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0],
16                    [0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0],
                      [0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0],
18                    [0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0],
                      [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
20                    [0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0]]

22
   synthesized = exact_synthesis mymatrix
24
   circuit ::([Qubit], Qubit)-> Circ ([Qubit], Qubit)
26 circuit ([a, b, c], d) = do
     synthesized [a,b,c,d]
```

```
28      return ([a,b,c],d)
```

Listing B.4: Quipper program for quantamorphism over X gate with n = 7 - matrix definition.

```
1  -- Decompose the circuit in the standard gates
   rand = RandomSource(fst(split(mkStdGen 10)))
3  prec = 20 * bits

5  circuit_decompose = decompose_generic (Standard prec rand) circuit
```

Listing B.5: Quipper program for quantamorphism over X gate with n = 7 - decompose the circuit in the standard gates.

```
1  main1 :: IO ()
   main1 = do
3    print_generic Preview (circuit_function my_oracle)
       where
5      -- declare empty_oracle's data type
       my_oracle :: Oracle
7      my_oracle = Oracle {
         -- set the length of qubit string
9        qubit_num = 3,
         function = circuit
11     }

13 main2 :: IO ()
   main2 = do
15   print_generic ASCII (circuit_function my_oracle)
          where
17         -- declare empty_oracle's data type
           my_oracle :: Oracle
19         my_oracle = Oracle {
             -- set the length of qubit string
21           qubit_num = 3,
             function = circuit
23         }

25 main3 :: IO ()
   main3 = do
27   simulate (circuit_function my_oracle)
     where
29     -- declare empty_oracle's data type
       my_oracle :: Oracle
31     my_oracle = Oracle {
         -- set the length of qubit string
33       qubit_num = 3,
         function = circuit
```

```haskell
35      }

37  main4 :: IO ()
    main4 = do
39    print_generic Preview (circuit_function my_oracle)
        where
41      -- declare empty_oracle's data type
      my_oracle :: Oracle
43    my_oracle = Oracle {
          -- set the length of qubit string
45        qubit_num = 3,
          function = circuit_decompose
47    }
     -- print mymatrix
49
    main5 :: IO ()
51  main5 = do
      print_generic ASCII (circuit_function my_oracle)
53      where
        -- declare empty_oracle's data type
55    my_oracle :: Oracle
      my_oracle = Oracle {
57        -- set the length of qubit string
          qubit_num = 3,
59        function = circuit_decompose
      }
61
    main6 :: IO ()
63  main6 = do
      simulate (circuit_function my_oracle)
65    where
        -- declare empty_oracle's data type
67    my_oracle :: Oracle
      my_oracle = Oracle {
69        -- set the length of qubit string
          qubit_num = 3,
71        function = circuit_decompose
      }
73
    main = do
75    putStrLn "\n \n choose an option:"
      putStrLn " 1 - PDF original circuit\n"
77    putStrLn " 2 - text description of original circuit\n"
      putStrLn " 3 - original simulation\n"
79    putStrLn " 4 - PDF decomposed circuit\n"
      putStrLn " 5 - text description of decomposed circuit\n"
81    putStrLn " 6 - simulation of the decomposed circuit \n"
```

```
      putStrLn " 7 - exit\n"
83    line <- getLine
      case line of
85      "1" -> do main1
                  main
87      "2" -> do main2
                  main
89      "3" -> do main3
                  main
91      "4" -> do main4
                  main
93      "5" -> do main5
                  main
95      "6" -> do main6
                  main
97      "7" -> do exitSuccess
        _ -> do main
99

      -- if you want the circuit description a .txt file
101   -- comment the menu above
      -- replacing it by:
103   -- main5
      -- and run:
105   -- $ ./ok_fold_X_7 > matrix_x_7_quipper.txt
```

Listing B.6: Quipper program for quantamorphism over X gate with n = 7 - main function.

QUANTAMORPHISM OVER Y GATE   Quantamorphisms over Y gate import all the modules needed in the quantamorphism over X gate with 3 control qubits. In addition to this, the following example also needs to import modules to handle complex numbers and the module Quipper.Internal.

```
import Quipper.Internal
```

Listing B.7: Quipper program for quantamorphism over Y gate - import extra module.

```
1  import Data.Complex
   import Data.Ratio
3  import Data.Tuple
```

Listing B.8: Quipper program for quantamorphism over Y gate - import module for complex numbers.

```
   mymatrix :: Matrix Eight Eight (Cplx Integer)
2  mymatrix = matrix [[Cplx (1)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
       (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
```

```
                              [Cplx (0)(0), Cplx (1)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                                  (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
4                             [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(-1), Cplx
                                  (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
                              [Cplx (0)(0), Cplx (0)(0), Cplx (0)(1), Cplx (0)(0), Cplx
                                  (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
6                             [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                                  (1)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0) ],
                              [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                                  (0)(0), Cplx (1)(0), Cplx (0)(0), Cplx (0)(0) ],
8                             [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                                  (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(-1) ],
                              [Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx (0)(0), Cplx
                                  (0)(0), Cplx (0)(0), Cplx (0)(1), Cplx (0)(0)]]
10


12  synthesized = exact_synthesis mymatrix

14  circuit :: ([Qubit], Qubit) -> Circ ([Qubit],Qubit)
    circuit ([a, b], c) = do
16    synthesized [a,b,c]
      return ([a,b],c)
```

Listing B.9: Quipper program for quantamorphism over Y gate - matrix definition.

QUANTAMORPHISM OVER HADAMARD GATE   The only change needed to convert the implementation of quantamorphism over Y gate into that of the quantamorphism over the Hadamard gate is the matrix definition.

```
    invsq2 :: Cplx (RootTwo (Ratio Integer))
2   invsq2 = Cplx (RootTwo 0 (1 % 2)) 0    :: Cplx (RootTwo (Ratio Integer))

4   mymatrix :: Matrix Eight Eight (Cplx (RootTwo (Ratio Integer)))
    mymatrix = matrix [[1, 0, 0, 0, 0, 0, 0, 0],
6                      [ 0, 1, 0, 0, 0, 0, 0, 0],
                       [ 0, 0, invsq2, invsq2, 0, 0, 0, 0],
8                      [ 0, 0, invsq2, -invsq2, 0, 0, 0, 0],
                       [ 0, 0, 0, 0, 1, 0, 0, 0],
10                     [ 0, 0, 0, 0, 0, 1, 0, 0],
                       [ 0, 0, 0, 0, 0, 0, invsq2, invsq2],
12                     [ 0, 0, 0, 0, 0, 0, invsq2, -invsq2]]


14
    synthesized = exact_synthesis mymatrix
16
    circuit ::([Qubit], Qubit)-> Circ ([Qubit], Qubit)
```

```
18  circuit ([a, b], c) = do
      synthesized [a,b,c]
20    return ([a,b],c)
```

Listing B.10: Quipper program for quantamorphism over the Hadamard gate - matrix definition.

QUANTAMORPHISM OVER XOR GATE

```
    -- natural number 32 as a type
2   type Thirty_two = Ten_and (Ten_and (Ten_and Two))

4   --32x32
    mymatrix :: Matrix Thirty_two Thirty_two Integer
6   mymatrix = matrix [[ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
8                     [ 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
10                    [ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
12                    [ 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
14                    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
16                    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
18                    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
20                    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                      [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
22                    [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
                          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
```

```
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
24                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
26                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
28                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 ],
30                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ],
32                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0 ],
34                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0 ],
36                        [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ],
                          [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 ]]
38

40  synthesized = exact_synthesis mymatrix

42  circuit :: ([Qubit], Qubit) -> Circ ([Qubit], Qubit)
    circuit ([a,b,c,d], e) = do
44    synthesized [a,b,c,d,e]
      return ([a,b,c,d],e)
```

Listing B.11: Quipper program for quantamorphism over the XOR gate - matrix definition.

QUIPPER RESULTS

FOR-LOOP QUANTAMORPHISM OVER X GATE   The outputs of the simulation of the circuit for the quantamorphism over the X gate with n = 3 are given in listings C.1 to C.9.

```
1          [((([False,False],False),1.0)]
```

Listing C.1: The input state is $q_0 = 0$, $q_1 = 0$ and $q_2 = 0$.

```
1          [((([False,False],True),1.0)]
```

Listing C.2: Adding a X gate to q_2 changes the input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = 1$.

```
1          [((([False,False],True),0.5),((([False,False],False),0.5)]
```

Listing C.3: Adding a Hadamard gate to q_2 changes the input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

```
1          [((([True,True],True),1.0)]
```

Listing C.4: Adding X gates to q_0 and q_1 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = 0$.

```
1          [((([True,True],False),1.0)]
```

Listing C.5: Adding X gates to all qubits changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = 1$.

```
1          [((([True,True],True),0.5),((([True,True],False),0.5)]
```

Listing C.6: Adding X gates to q_0 and q_1 and Hadamard gate to q_2 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

```
1          [((([True,True],True),0.25),((([False,True],True),0.25),((([True,False],
               False),0.25),((([False,False],False),0.25)]
```

Listing C.7: Adding Hadamard gates to q_0 and q_1 changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = 0$.

```
1           [(([True,False],True),0.25),(([False,False],True),0.25),(([True,True],
               False),0.25),(([False,True],False),0.25)]
```

Listing C.8: Adding Hadamard gates to q_0 and q_1 and X gate to q_2 changes the input state to $q_0 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = 1$.

```
1           [(([True,True],True),0.12499999),(([False,True],True),0.12499999),((([
               True,False],True),0.12499999),(([False,False],True),0.12499999)
               ,(([True,True],False),0.125),(([False,True],False),0.125),(([True,
               False],False),0.125),(([False,False],False),0.125)]
```

Listing C.9: Adding Hadamard gates to all qubits changes the input state to $q_0 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

In the case where the previous function `qfor` works with $n = 7$, the description of the decomposition can be found in listing C.10.

```
1  Inputs: none
   QInit0(0)
3  QInit0(1)
   QInit0(2)
5  QInit1(3)
   Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|0>[2]", 3:"|1>")
7  Comment["before oracle"]()
   Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]")
9  QInit0(4) with nocontrol
   Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.+", 4:"q")
11 QGate["H"](4) with nocontrol
   QGate["not"](0) with controls=[+4] with nocontrol
13 QGate["not"](1) with controls=[+0] with nocontrol
   QGate["T"](0) with nocontrol
15 QGate["T"]*(1) with nocontrol
   QGate["not"](0) with controls=[+4] with nocontrol
17 QGate["not"](1) with controls=[+0] with nocontrol
   QGate["T"]*(4) with nocontrol
19 QGate["T"](1) with nocontrol
   QGate["not"](1) with controls=[+4] with nocontrol
21 QGate["H"](4) with nocontrol
   Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.+", 4:"q")
23 Comment["ENTER: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
   QGate["H"](3)
25 QGate["T"](3)
```

```
   QGate["T"](2)
27 QGate["T"](4)
   QGate["not"](4) with controls=[+2]
29 QGate["not"](2) with controls=[+3]
   QGate["not"](3) with controls=[+4]
31 QGate["T"]*(2)
   QGate["T"](3)
33 QGate["not"](2) with controls=[+4]
   QGate["T"]*(2)
35 QGate["T"]*(4)
   QGate["not"](2) with controls=[+3]
37 QGate["not"](3) with controls=[+4]
   QGate["not"](4) with controls=[+2]
39 QGate["H"](3)
   Comment["EXIT: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
41 Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.+", 4:"q")
   QGate["H"](4) with nocontrol
43 QGate["not"](1) with controls=[+4] with nocontrol
   QGate["T"]*(1) with nocontrol
45 QGate["T"](4) with nocontrol
   QGate["not"](1) with controls=[+0] with nocontrol
47 QGate["not"](0) with controls=[+4] with nocontrol
   QGate["T"](1) with nocontrol
49 QGate["T"]*(0) with nocontrol
   QGate["not"](1) with controls=[+0] with nocontrol
51 QGate["not"](0) with controls=[+4] with nocontrol
   QGate["H"](4) with nocontrol
53 Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.+", 4:"q")
   QTerm0(4) with nocontrol
55 QInit0(4) with nocontrol
   Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.-", 4:"q")
57 QGate["H"](4) with nocontrol
   QGate["not"](0) with controls=[+4] with nocontrol
59 QGate["not"](1) with controls=[+0] with nocontrol
   QGate["T"](0) with nocontrol
61 QGate["T"](1) with nocontrol
   QGate["not"](0) with controls=[+4] with nocontrol
63 QGate["not"](1) with controls=[+0] with nocontrol
   QGate["T"]*(4) with nocontrol
65 QGate["T"]*(1) with nocontrol
   QGate["not"](1) with controls=[+4] with nocontrol
67 QGate["H"](4) with nocontrol
   Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.-", 4:"q")
69 Comment["ENTER: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
   QGate["H"](3)
71 QGate["T"](3)
   QGate["T"](2)
```

```
73  QGate["T"](4)
    QGate["not"](4) with controls=[+2]
75  QGate["not"](2) with controls=[+3]
    QGate["not"](3) with controls=[+4]
77  QGate["T"]*(2)
    QGate["T"](3)
79  QGate["not"](2) with controls=[+4]
    QGate["T"]*(2)
81  QGate["T"]*(4)
    QGate["not"](2) with controls=[+3]
83  QGate["not"](3) with controls=[+4]
    QGate["not"](4) with controls=[+2]
85  QGate["H"](3)
    Comment["EXIT: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
87  Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.-", 4:"q")
    QGate["H"](4) with nocontrol
89  QGate["not"](1) with controls=[+4] with nocontrol
    QGate["T"](1) with nocontrol
91  QGate["T"](4) with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
93  QGate["not"](0) with controls=[+4] with nocontrol
    QGate["T"]*(1) with nocontrol
95  QGate["T"]*(0) with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
97  QGate["not"](0) with controls=[+4] with nocontrol
    QGate["H"](4) with nocontrol
99  Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.-", 4:"q")
    QTerm0(4) with nocontrol
101 QInit0(4) with nocontrol
    Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.+", 4:"q")
103 QGate["H"](4) with nocontrol
    QGate["not"](0) with controls=[+4] with nocontrol
105 QGate["not"](1) with controls=[+0] with nocontrol
    QGate["T"]*(0) with nocontrol
107 QGate["T"](1) with nocontrol
    QGate["not"](0) with controls=[+4] with nocontrol
109 QGate["not"](1) with controls=[+0] with nocontrol
    QGate["T"]*(4) with nocontrol
111 QGate["T"](1) with nocontrol
    QGate["not"](1) with controls=[+4] with nocontrol
113 QGate["H"](4) with nocontrol
    Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.+", 4:"q")
115 Comment["ENTER: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
    QGate["H"](3)
117 QGate["T"](3)
    QGate["T"](2)
119 QGate["T"](4)
```

```
     QGate["not"](4) with controls=[+2]
121  QGate["not"](2) with controls=[+3]
     QGate["not"](3) with controls=[+4]
123  QGate["T"]*(2)
     QGate["T"](3)
125  QGate["not"](2) with controls=[+4]
     QGate["T"]*(2)
127  QGate["T"]*(4)
     QGate["not"](2) with controls=[+3]
129  QGate["not"](3) with controls=[+4]
     QGate["not"](4) with controls=[+2]
131  QGate["H"](3)
     Comment["EXIT: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
133  Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.+", 4:"q")
     QGate["H"](4) with nocontrol
135  QGate["not"](1) with controls=[+4] with nocontrol
     QGate["T"]*(1) with nocontrol
137  QGate["T"](4) with nocontrol
     QGate["not"](1) with controls=[+0] with nocontrol
139  QGate["not"](0) with controls=[+4] with nocontrol
     QGate["T"]*(1) with nocontrol
141  QGate["T"](0) with nocontrol
     QGate["not"](1) with controls=[+0] with nocontrol
143  QGate["not"](0) with controls=[+4] with nocontrol
     QGate["H"](4) with nocontrol
145  Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.+", 4:"q")
     QTerm0(4) with nocontrol
147  QInit0(4) with nocontrol
     Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.-", 4:"q")
149  QGate["H"](4) with nocontrol
     QGate["not"](0) with controls=[+4] with nocontrol
151  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["T"]*(0) with nocontrol
153  QGate["T"]*(1) with nocontrol
     QGate["not"](0) with controls=[+4] with nocontrol
155  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["T"]*(4) with nocontrol
157  QGate["T"]*(1) with nocontrol
     QGate["not"](1) with controls=[+4] with nocontrol
159  QGate["H"](4) with nocontrol
     Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.-", 4:"q")
161  Comment["ENTER: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
     QGate["H"](3)
163  QGate["T"](3)
     QGate["T"](2)
165  QGate["T"](4)
     QGate["not"](4) with controls=[+2]
```

```
167  QGate["not"](2) with controls=[+3]
     QGate["not"](3) with controls=[+4]
169  QGate["T"]*(2)
     QGate["T"](3)
171  QGate["not"](2) with controls=[+4]
     QGate["T"]*(2)
173  QGate["T"]*(4)
     QGate["not"](2) with controls=[+3]
175  QGate["not"](3) with controls=[+4]
     QGate["not"](4) with controls=[+2]
177  QGate["H"](3)
     Comment["EXIT: toffoli_AMMR"](2:"c1.+", 3:"q", 4:"c2.+")
179  Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.-", 4:"q")
     QGate["H"](4) with nocontrol
181  QGate["not"](1) with controls=[+4] with nocontrol
     QGate["T"](1) with nocontrol
183  QGate["T"](4) with nocontrol
     QGate["not"](1) with controls=[+0] with nocontrol
185  QGate["not"](0) with controls=[+4] with nocontrol
     QGate["T"](1) with nocontrol
187  QGate["T"](0) with nocontrol
     QGate["not"](1) with controls=[+0] with nocontrol
189  QGate["not"](0) with controls=[+4] with nocontrol
     QGate["H"](4) with nocontrol
191  Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.-", 4:"q")
     QTerm0(4) with nocontrol
193  Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]")
     Comment["after oracle"]()
195  QMeas(3)
     QMeas(2)
197  QMeas(1)
     QMeas(0)
199  Outputs: 0:Cbit, 1:Cbit, 2:Cbit, 3:Cbit
```

Listing C.10: Description of circuit generated by quantamorphism for X with 3 control qubits.

The simulations of the circuits (original and decomposed) turnout equal as can be seen in listings C.11 to C.22.

```
     [((([False,False,False],False),1.0)]
```

Listing C.11: The input state is $q_0 = 0$, $q_1 = 0$, $q_2 = 0$, $q_3 = 0$ - original circuit.

```
1    [((([False,False,False],False),1.0)]
```

Listing C.12: The input state is $q_0 = 0$, $q_1 = 0$, $q_2 = 0$, $q_3 = 0$ - decomposed circuit.

```
1        [(([False,False,False],True),1.0)]
```

Listing C.13: Adding a X gate to the target qubit changes the input states to $q_0 = 0$, $q_1 = 0$, $q_2 = 0$, $q_3 = 1$ - original circuit.

```
1        [(([False,False,False],True),1.0)]
```

Listing C.14: Adding a X gate to the target qubit changes the input states to $q_0 = 0$, $q_1 = 0$, $q_2 = 0$, $q_3 = 1$ - decomposed circuit.

```
1        [(([True,True,True],True),1.0)]
```

Listing C.15: Adding X gates to the control qubits changes the input states to $q_0 = 1$, $q_1 = 1$, $q_2 = 1$, $q_3 = 0$ - original circuit.

```
1        [(([True,True,True],True),1.0)]
```

Listing C.16: Adding X gates to the control qubits changes the input states to $q_0 = 1$, $q_1 = 1$, $q_2 = 1$, $q_3 = 0$ - decomposed circuit.

```
1        [(([True,True,True],False),1.0)]
```

Listing C.17: Adding X gates to all qubits changes the input states to $q_0 = 1$, $q_1 = 1$, $q_2 = 1$, $q_3 = 1$ - original circuit.

```
1        [(([True,True,True],False),1.0)]
```

Listing C.18: Adding X gates to all qubits changes the input states to $q_0 = 1$, $q_1 = 1$, $q_2 = 1$, $q_3 = 1$ - decomposed circuit.

```
1        [(([True,True,True],True),0.12499999),(([False,True,True],True)
         ,0.12499999),(([True,False,True],True),0.12499999),(([False,False,True
         ],True),0.12499999),(([True,True,False],False),0.125),(([False,True,
         False],False),0.125),(([True,False,False],False),0.125),(([False,False
         ,False],False),0.125)]
```

Listing C.19: Adding Hadamard gates to the controls qubits changes the input states to $q_0 = \frac{1}{\sqrt{2}}$, $q_1 = \frac{1}{\sqrt{2}}$, $q_2 = \frac{1}{\sqrt{2}}$, $q_3 = 0$ - original circuit.

```
1        [(([True,True,True],True),0.12499999),(([False,True,True],True)
         ,0.12499999),(([True,False,True],True),0.12499999),(([False,False,True
         ],True),0.12499999),(([True,True,False],False),0.125),(([False,True,
         False],False),0.125),(([True,False,False],False),0.125),(([False,False
         ,False],False),0.125)]
```

Listing C.20: Adding Hadamard gates to the controls qubits changes the input states to $q_0 = \frac{1}{\sqrt{2}}$, $q_1 = \frac{1}{\sqrt{2}}$, $q_2 = \frac{1}{\sqrt{2}}$, $q_3 = 0$ - decomposed circuit.

```
1    [((([True,True,False],True),0.12499999),(([False,True,False],True)
       ,0.12499999),(([True,False,False],True),0.12499999),(([False,False,
       False],True),0.12499999),(([True,True,True],False),0.125),(([False,
       True,True],False),0.125),(([True,False,True],False),0.125),(([False,
       False,True],False),0.125)]
```

Listing C.21: Adding Hadamard gates to the control qubits and a X gate to the target qubit the input states changes to $q_0 = \frac{1}{\sqrt{2}}$, $q_1 = \frac{1}{\sqrt{2}}$, $q_2 = \frac{1}{\sqrt{2}}$, $q_3 = 1$ - original circuit.

```
1    [((([True,True,False],True),0.12499999),(([False,True,False],True)
       ,0.12499999),(([True,False,False],True),0.12499999),(([False,False,
       False],True),0.12499999),(([True,True,True],False),0.125),(([False,
       True,True],False),0.125),(([True,False,True],False),0.125),(([False,
       False,True],False),0.125)]
```

Listing C.22: Adding Hadamard gates to the control qubits and a X gate to the target qubit the input states changes to $q_0 = \frac{1}{\sqrt{2}}$, $q_1 = \frac{1}{\sqrt{2}}$, $q_2 = \frac{1}{\sqrt{2}}$, $q_3 = 1$ - decomposed circuit.

FOR-LOOP QUANTAMORPHISM OVER Y GATE   The description of the circuit Y:

```
1  Inputs: none
   QInit0(0)
3  QInit0(1)
   QInit0(2)
5  Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|0>")
   Comment["before oracle"]()
7  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
   QGate["S"]*(2) with controls=[+0, +1]
9  QGate["S"](1) with controls=[+0, -2]
   QGate["X"](2) with controls=[+0, +1]
11 QGate["S"]*(2) with controls=[-0, +1]
   QGate["S"](1) with controls=[-0, -2]
13 QGate["X"](2) with controls=[-0, +1]
   Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
15 Comment["after oracle"]()
   QMeas(2)
17 QMeas(1)
   QMeas(0)
19 Outputs: 0:Cbit, 1:Cbit, 2:Cbit
```

Listing C.23: Description of circuit generated by quantamorphism for Y .

The same circuit after decomposition:

```
 1  Inputs: none
    QInit0(0)
 3  QInit0(1)
    QInit0(2)
 5  Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|0>")
    Comment["before oracle"]()
 7  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
    QInit0(3) with nocontrol
 9  Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.+", 3:"q")
    QGate["H"](3) with nocontrol
11  QGate["not"](0) with controls=[+3] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
13  QGate["T"](0) with nocontrol
    QGate["T"]*(1) with nocontrol
15  QGate["not"](0) with controls=[+3] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
17  QGate["T"]*(3) with nocontrol
    QGate["T"](1) with nocontrol
19  QGate["not"](1) with controls=[+3] with nocontrol
    QGate["H"](3) with nocontrol
21  Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.+", 3:"q")
    Comment["EXIT: controlled_S"]*(2:"q", 3:"c.+")
23  QGate["T"]*(2)
    QGate["T"]*(3)
25  QGate["not"](3) with controls=[+2]
    QGate["T"](3)
27  QGate["not"](3) with controls=[+2]
    Comment["ENTER: controlled_S"]*(2:"q", 3:"c.+")
29  Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.+", 3:"q")
    QGate["H"](3) with nocontrol
31  QGate["not"](1) with controls=[+3] with nocontrol
    QGate["T"]*(1) with nocontrol
33  QGate["T"](3) with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
35  QGate["not"](0) with controls=[+3] with nocontrol
    QGate["T"](1) with nocontrol
37  QGate["T"]*(0) with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
39  QGate["not"](0) with controls=[+3] with nocontrol
    QGate["H"](3) with nocontrol
41  Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.+", 3:"q")
    QTerm0(3) with nocontrol
43  QInit0(3) with nocontrol
    Comment["ENTER: cc_iX"](0:"c1.+", 2:"c2.-", 3:"q")
45  QGate["H"](3) with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
```

```
47  QGate["not"](2) with controls=[+0] with nocontrol
    QGate["T"](0) with nocontrol
49  QGate["T"](2) with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
51  QGate["not"](2) with controls=[+0] with nocontrol
    QGate["T"]*(3) with nocontrol
53  QGate["T"]*(2) with nocontrol
    QGate["not"](2) with controls=[+3] with nocontrol
55  QGate["H"](3) with nocontrol
    Comment["EXIT: cc_iX"](0:"c1.+", 2:"c2.-", 3:"q")
57  Comment["ENTER: controlled_S"](1:"q", 3:"c.+")
    QGate["not"](3) with controls=[+1]
59  QGate["T"]*(3)
    QGate["not"](3) with controls=[+1]
61  QGate["T"](3)
    QGate["T"](1)
63  Comment["EXIT: controlled_S"](1:"q", 3:"c.+")
    Comment["EXIT: cc_iX"]*(0:"c1.+", 2:"c2.-", 3:"q")
65  QGate["H"](3) with nocontrol
    QGate["not"](2) with controls=[+3] with nocontrol
67  QGate["T"](2) with nocontrol
    QGate["T"](3) with nocontrol
69  QGate["not"](2) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
71  QGate["T"]*(2) with nocontrol
    QGate["T"]*(0) with nocontrol
73  QGate["not"](2) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
75  QGate["H"](3) with nocontrol
    Comment["ENTER: cc_iX"]*(0:"c1.+", 2:"c2.-", 3:"q")
77  QTerm0(3) with nocontrol
    Comment["ENTER: toffoli_AMMR"](0:"c1.+", 1:"c2.+", 2:"q")
79  QGate["H"](2)
    QGate["T"](2)
81  QGate["T"](0)
    QGate["T"](1)
83  QGate["not"](1) with controls=[+0]
    QGate["not"](0) with controls=[+2]
85  QGate["not"](2) with controls=[+1]
    QGate["T"]*(0)
87  QGate["T"](2)
    QGate["not"](0) with controls=[+1]
89  QGate["T"]*(0)
    QGate["T"]*(1)
91  QGate["not"](0) with controls=[+2]
    QGate["not"](2) with controls=[+1]
93  QGate["not"](1) with controls=[+0]
```

```
    QGate["H"](2)
 95 Comment["EXIT: toffoli_AMMR"](0:"c1.+", 1:"c2.+", 2:"q")
    QInit0(3) with nocontrol
 97 Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.+", 3:"q")
    QGate["H"](3) with nocontrol
 99 QGate["not"](0) with controls=[+3] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
101 QGate["T"]*(0) with nocontrol
    QGate["T"](1) with nocontrol
103 QGate["not"](0) with controls=[+3] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
105 QGate["T"]*(3) with nocontrol
    QGate["T"](1) with nocontrol
107 QGate["not"](1) with controls=[+3] with nocontrol
    QGate["H"](3) with nocontrol
109 Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.+", 3:"q")
    Comment["EXIT: controlled_S"]*(2:"q", 3:"c.+")
111 QGate["T"]*(2)
    QGate["T"]*(3)
113 QGate["not"](3) with controls=[+2]
    QGate["T"](3)
115 QGate["not"](3) with controls=[+2]
    Comment["ENTER: controlled_S"]*(2:"q", 3:"c.+")
117 Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.+", 3:"q")
    QGate["H"](3) with nocontrol
119 QGate["not"](1) with controls=[+3] with nocontrol
    QGate["T"]*(1) with nocontrol
121 QGate["T"](3) with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
123 QGate["not"](0) with controls=[+3] with nocontrol
    QGate["T"]*(1) with nocontrol
125 QGate["T"](0) with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
127 QGate["not"](0) with controls=[+3] with nocontrol
    QGate["H"](3) with nocontrol
129 Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.+", 3:"q")
    QTerm0(3) with nocontrol
131 QInit0(3) with nocontrol
    Comment["ENTER: cc_iX"](0:"c1.-", 2:"c2.-", 3:"q")
133 QGate["H"](3) with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
135 QGate["not"](2) with controls=[+0] with nocontrol
    QGate["T"]*(0) with nocontrol
137 QGate["T"]*(2) with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
139 QGate["not"](2) with controls=[+0] with nocontrol
    QGate["T"]*(3) with nocontrol
```

```
141  QGate["T"]*(2) with nocontrol
     QGate["not"](2) with controls=[+3] with nocontrol
143  QGate["H"](3) with nocontrol
     Comment["EXIT: cc_iX"](0:"c1.-", 2:"c2.-", 3:"q")
145  Comment["ENTER: controlled_S"](1:"q", 3:"c.+")
     QGate["not"](3) with controls=[+1]
147  QGate["T"]*(3)
     QGate["not"](3) with controls=[+1]
149  QGate["T"](3)
     QGate["T"](1)
151  Comment["EXIT: controlled_S"](1:"q", 3:"c.+")
     Comment["EXIT: cc_iX"]*(0:"c1.-", 2:"c2.-", 3:"q")
153  QGate["H"](3) with nocontrol
     QGate["not"](2) with controls=[+3] with nocontrol
155  QGate["T"](2) with nocontrol
     QGate["T"](3) with nocontrol
157  QGate["not"](2) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+3] with nocontrol
159  QGate["T"](2) with nocontrol
     QGate["T"](0) with nocontrol
161  QGate["not"](2) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+3] with nocontrol
163  QGate["H"](3) with nocontrol
     Comment["ENTER: cc_iX"]*(0:"c1.-", 2:"c2.-", 3:"q")
165  QTerm0(3) with nocontrol
     Comment["ENTER: toffoli_AMMR"](0:"c1.-", 1:"c2.+", 2:"q")
167  QGate["H"](2)
     QGate["T"](2)
169  QGate["T"]*(0)
     QGate["T"](1)
171  QGate["not"](1) with controls=[+0]
     QGate["not"](0) with controls=[+2]
173  QGate["not"](2) with controls=[+1]
     QGate["T"](0)
175  QGate["T"]*(2)
     QGate["not"](0) with controls=[+1]
177  QGate["T"]*(0)
     QGate["T"](1)
179  QGate["not"](0) with controls=[+2]
     QGate["not"](2) with controls=[+1]
181  QGate["not"](1) with controls=[+0]
     QGate["H"](2)
183  Comment["EXIT: toffoli_AMMR"](0:"c1.-", 1:"c2.+", 2:"q")
     Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
185  Comment["after oracle"]()
     QMeas(2)
187  QMeas(1)
```

```
      QMeas(0)
189   Outputs: 0: Cbit, 1: Cbit, 2: Cbit
```

Listing C.24: Description of decomposed circuit generated by quantamorphism for Y .

The simulation of this circuit results in the outputs C.25 to C.33.

```
              [((([False,False],False),1.0)]
```

Listing C.25: The input state $q_0 = 0$, $q_1 = 0$ and $q_2 = 0$.

```
1             [((([False,False],True),1.0)]
```

Listing C.26: Adding X gates to q_2 changes the input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = 1$.

```
1             [((([False,False],True),0.5),(([False,False],False),0.5)]
```

Listing C.27: Adding Hadamard gate to q_2 changes the input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

```
1             [((([True,True],True),1.0)]
```

Listing C.28: Adding X gates to q_0 and q_1 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = 0$.

```
1             [((([True,True],False),1.0)]
```

Listing C.29: Adding X gates to all qubits changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = 1$.

```
1             [((([True,True],True),0.5),(([True,True],False),0.5)]
```

Listing C.30: Adding X gates to q_0 and q_1 and Hadamard gate to q_2 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

```
1             [((([True,True],True),0.25),(([False,True],True),0.25),(([True,False],
                  False),0.25),(([False,False],False),0.25)]
```

Listing C.31: Adding Hadamard gates to q_0 and q_1 changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = 0$.

```
1             [((([True,False],True),0.25),(([False,False],True),0.25),(([True,True],
                  False),0.25),(([False,True],False),0.25)]
```

Listing C.32: Adding Hadamard gates to q_0 and q_1 and X gate to q_2 changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = 1$.

```
1            [((([True,True],True),0.12499999),(([False,True],True),0.12499999),((([
                True,False],True),0.12499999),(([False,False],True),0.12499999)
                ,((([True,True],False),0.125),(([False,True],False),0.125),((([True,
                False],False),0.125),(([False,False],False),0.125)]
```

Listing C.33: Adding Hadamard gates to all qubits changes the input state to $q_0 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

FOR-LOOP QUANTAMORPHISM OVER HADAMARD GATE   The description of the original circuit:

```
1  Inputs: none
   QInit0(0)
3  QInit0(1)
   QInit1(2)
5  Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|1>")
   Comment["before oracle"]()
7  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
   QGate["H"](2) with controls=[+0, +1]
9  QGate["H"](2) with controls=[-0, +1]
   Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
11 Comment["after oracle"]()
   QMeas(2)
13 QMeas(1)
   QMeas(0)
15 Outputs: 0:Cbit, 1:Cbit, 2:Cbit
```

Listing C.34: Description of circuit generated by the for-loop quantamorphism over Hadamard gate .

After the decomposition of the circuit:

```
1  Inputs: none
   QInit0(0)
3  QInit0(1)
   QInit1(2)
5  Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|1>")
   Comment["before oracle"]()
7  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
   QInit0(3) with nocontrol
9  Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.+", 3:"q")
   QGate["H"](3) with nocontrol
11 QGate["not"](0) with controls=[+3] with nocontrol
   QGate["not"](1) with controls=[+0] with nocontrol
13 QGate["T"](0) with nocontrol
   QGate["T"]*(1) with nocontrol
15 QGate["not"](0) with controls=[+3] with nocontrol
```

```
   QGate["not"](1) with controls=[+0] with nocontrol
17 QGate["T"]*(3) with nocontrol
   QGate["T"](1) with nocontrol
19 QGate["not"](1) with controls=[+3] with nocontrol
   QGate["H"](3) with nocontrol
21 Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.+", 3:"q")
   Comment["ENTER: cH_AMMR"](2:"q", 3:"c.+")
23 QGate["S"]*(2)
   QGate["H"](2)
25 QGate["T"]*(2)
   QGate["not"](2) with controls=[+3]
27 QGate["T"](2)
   QGate["H"](2)
29 QGate["S"](2)
   Comment["EXIT: cH_AMMR"](2:"q", 3:"c.+")
31 Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.+", 3:"q")
   QGate["H"](3) with nocontrol
33 QGate["not"](1) with controls=[+3] with nocontrol
   QGate["T"]*(1) with nocontrol
35 QGate["T"](3) with nocontrol
   QGate["not"](1) with controls=[+0] with nocontrol
37 QGate["not"](0) with controls=[+3] with nocontrol
   QGate["T"](1) with nocontrol
39 QGate["T"]*(0) with nocontrol
   QGate["not"](1) with controls=[+0] with nocontrol
41 QGate["not"](0) with controls=[+3] with nocontrol
   QGate["H"](3) with nocontrol
43 Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.+", 3:"q")
   QTerm0(3) with nocontrol
45 QInit0(3) with nocontrol
   Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.+", 3:"q")
47 QGate["H"](3) with nocontrol
   QGate["not"](0) with controls=[+3] with nocontrol
49 QGate["not"](1) with controls=[+0] with nocontrol
   QGate["T"]*(0) with nocontrol
51 QGate["T"](1) with nocontrol
   QGate["not"](0) with controls=[+3] with nocontrol
53 QGate["not"](1) with controls=[+0] with nocontrol
   QGate["T"]*(3) with nocontrol
55 QGate["T"](1) with nocontrol
   QGate["not"](1) with controls=[+3] with nocontrol
57 QGate["H"](3) with nocontrol
   Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.+", 3:"q")
59 Comment["ENTER: cH_AMMR"](2:"q", 3:"c.+")
   QGate["S"]*(2)
61 QGate["H"](2)
   QGate["T"]*(2)
```

```
63  QGate["not"](2) with controls=[+3]
    QGate["T"](2)
65  QGate["H"](2)
    QGate["S"](2)
67  Comment["EXIT: cH_AMMR"](2:"q", 3:"c.+")
    Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.+", 3:"q")
69  QGate["H"](3) with nocontrol
    QGate["not"](1) with controls=[+3] with nocontrol
71  QGate["T"]*(1) with nocontrol
    QGate["T"](3) with nocontrol
73  QGate["not"](1) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
75  QGate["T"]*(1) with nocontrol
    QGate["T"](0) with nocontrol
77  QGate["not"](1) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+3] with nocontrol
79  QGate["H"](3) with nocontrol
    Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.+", 3:"q")
81  QTerm0(3) with nocontrol
    Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]")
83  Comment["after oracle"]()
    QMeas(2)
85  QMeas(1)
    QMeas(0)
87  Outputs: 0:Cbit, 1:Cbit, 2:Cbit
```

Listing C.35: Description of decomposed circuit generated by the for-loop quantamorphism over Hadamard gate .

```
        [((([False,False],False),1.0)]
```

Listing C.36: The input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = 0$.

```
1       [((([False,False],True),1.0)]
```

Listing C.37: Adding a X gate to q_2 changes the input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = 1$.

```
1       [((([False,False],True),0.5),(([False,False],False),0.5)]
```

Listing C.38: Adding Hadamard gate to q_2 changes the input state to $q_0 = 0$, $q_1 = 0$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

```
1       [((([True,True],True),0.5),(([True,True],False),0.5)]
```

Listing C.39: Adding X gates to q_0 and q_1 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = 0$.

```
1           [(([True,True],True),0.5),(([True,True],False),0.5)]
```

Listing C.40: Adding X gates to all qubits changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = 1$.

```
1           [(([True,True],True),1.0)]
```

Listing C.41: Adding X gates to all qubits and Hadamard gate to q_2 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

```
1             [(([True,True],False),1.0)]
```

Listing C.42: Adding X gates to q_0 and q_1 and Hadamard gate to q_2 changes the input state to $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

```
1           [(([True,True],True),0.12499999),(([False,True],True),0.12499999),(([
               True,True],False),0.12499999),(([False,True],False),0.12499999),(([
               True,False],False),0.25),(([False,False],False),0.25)]
```

Listing C.43: Adding Hadamard gates to q_0 and q_1 changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = 0$.

```
1           [(([True,True],True),0.124999985),(([False,True],True),0.124999985),(([
               True,False],True),0.24999999),(([False,False],True),0.24999999),(([
               True,True],False),0.12500003),(([False,True],False),0.12500003)]
```

Listing C.44: Adding Hadamard gates to q_0 and q_1 and X gate to q_2 changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = 1$.

```
1           [(([True,False],True),0.125),(([False,False],True),0.125),(([True,True],
               False),0.24999999),(([False,True],False),0.24999999),(([True,False],
               False),0.12500001),(([False,False],False),0.12500001)]
```

Listing C.45: Adding Hadamard gates to all qubits changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

hhh(1)

```
1           [(([True,True],True),0.24999999),(([False,True],True),0.24999999),(([
               True,False],True),0.12500001),(([False,False],True),0.12500001),(([
               True,False],False),0.125),(([False,False],False),0.125)]
```

Listing C.46: Adding X gates to q_2 and Hadamard gates to all qubits changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_2 = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

FOLD QUANTAMORPHISM OVER XOR GATE   The description of the circuit can be found in listing C.47

```
1  Inputs: none
   QInit0(0)
3  QInit0(1)
   QInit0(2)
5  QInit0(3)
   QInit0(4)
7  Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|0>[2]", 3:"|0>[3]", 4:"|0>")
   Comment["before oracle"]()
9  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]", 4:"q
       [4]")
   QGate["X"](4) with controls=[+0, +1, +2, +3]
11 QGate["X"](4) with controls=[+0, +1, +2, -3]
   QGate["X"](4) with controls=[+0, -1, +2, -3]
13 QGate["X"](4) with controls=[+0, -1, -2, +3]
   QGate["X"](4) with controls=[+0, -1, -2, -3]
15 QGate["X"](4) with controls=[-0, +1, +2, -3]
   QGate["X"](4) with controls=[-0, +1, -2, +3]
17 QGate["X"](4) with controls=[-0, +1, -2, -3]
   Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]", 4:"q
       [4]")
19 Comment["after oracle"]()
   QMeas(4)
21 QMeas(3)
   QMeas(2)
23 QMeas(1)
   QMeas(0)
25 Outputs: 0:Cbit, 1:Cbit, 2:Cbit, 3:Cbit, 4:Cbit
```

Listing C.47: Description of circuit generated by fold quantamorphism over XOR gate.

After the decomposition of the circuit:

```
1  Inputs: none
   QInit0(0)
3  QInit0(1)
   QInit0(2)
5  QInit0(3)
   QInit1(4)
7  Comment[""](0:"|0>[0]", 1:"|0>[1]", 2:"|0>[2]", 3:"|0>[3]", 4:"|1>")
   Comment["before oracle"]()
9  Comment["ENTER: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]", 4:"q
       [4]")
   QInit0(5) with nocontrol
11 Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.+", 5:"q")
   QGate["H"](5) with nocontrol
```

```
13  QGate["not"](0) with controls=[+5] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
15  QGate["T"](0) with nocontrol
    QGate["T"]*(1) with nocontrol
17  QGate["not"](0) with controls=[+5] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
19  QGate["T"]*(5) with nocontrol
    QGate["T"](1) with nocontrol
21  QGate["not"](1) with controls=[+5] with nocontrol
    QGate["H"](5) with nocontrol
23  Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.+", 5:"q")
    QInit0(6) with nocontrol
25  Comment["ENTER: cc_iX"](2:"c1.+", 3:"c2.+", 6:"q")
    QGate["H"](6) with nocontrol
27  QGate["not"](2) with controls=[+6] with nocontrol
    QGate["not"](3) with controls=[+2] with nocontrol
29  QGate["T"](2) with nocontrol
    QGate["T"]*(3) with nocontrol
31  QGate["not"](2) with controls=[+6] with nocontrol
    QGate["not"](3) with controls=[+2] with nocontrol
33  QGate["T"]*(6) with nocontrol
    QGate["T"](3) with nocontrol
35  QGate["not"](3) with controls=[+6] with nocontrol
    QGate["H"](6) with nocontrol
37  Comment["EXIT: cc_iX"](2:"c1.+", 3:"c2.+", 6:"q")
    Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
39  QGate["H"](4)
    QGate["T"](4)
41  QGate["T"](5)
    QGate["T"](6)
43  QGate["not"](6) with controls=[+5]
    QGate["not"](5) with controls=[+4]
45  QGate["not"](4) with controls=[+6]
    QGate["T"]*(5)
47  QGate["T"](4)
    QGate["not"](5) with controls=[+6]
49  QGate["T"]*(5)
    QGate["T"]*(6)
51  QGate["not"](5) with controls=[+4]
    QGate["not"](4) with controls=[+6]
53  QGate["not"](6) with controls=[+5]
    QGate["H"](4)
55  Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
    Comment["EXIT: cc_iX"]*(2:"c1.+", 3:"c2.+", 6:"q")
57  QGate["H"](6) with nocontrol
    QGate["not"](3) with controls=[+6] with nocontrol
59  QGate["T"]*(3) with nocontrol
```

```
    QGate["T"](6) with nocontrol
61  QGate["not"](3) with controls=[+2] with nocontrol
    QGate["not"](2) with controls=[+6] with nocontrol
63  QGate["T"](3) with nocontrol
    QGate["T"]*(2) with nocontrol
65  QGate["not"](3) with controls=[+2] with nocontrol
    QGate["not"](2) with controls=[+6] with nocontrol
67  QGate["H"](6) with nocontrol
    Comment["ENTER: cc_iX"]*(2:"c1.+", 3:"c2.+", 6:"q")
69  QTerm0(6) with nocontrol
    Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.+", 5:"q")
71  QGate["H"](5) with nocontrol
    QGate["not"](1) with controls=[+5] with nocontrol
73  QGate["T"]*(1) with nocontrol
    QGate["T"](5) with nocontrol
75  QGate["not"](1) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+5] with nocontrol
77  QGate["T"](1) with nocontrol
    QGate["T"]*(0) with nocontrol
79  QGate["not"](1) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+5] with nocontrol
81  QGate["H"](5) with nocontrol
    Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.+", 5:"q")
83  QTerm0(5) with nocontrol
    QInit0(5) with nocontrol
85  Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.+", 5:"q")
    QGate["H"](5) with nocontrol
87  QGate["not"](0) with controls=[+5] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
89  QGate["T"](0) with nocontrol
    QGate["T"]*(1) with nocontrol
91  QGate["not"](0) with controls=[+5] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
93  QGate["T"]*(5) with nocontrol
    QGate["T"](1) with nocontrol
95  QGate["not"](1) with controls=[+5] with nocontrol
    QGate["H"](5) with nocontrol
97  Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.+", 5:"q")
    QInit0(6) with nocontrol
99  Comment["ENTER: cc_iX"](2:"c1.+", 3:"c2.-", 6:"q")
    QGate["H"](6) with nocontrol
101 QGate["not"](2) with controls=[+6] with nocontrol
    QGate["not"](3) with controls=[+2] with nocontrol
103 QGate["T"](2) with nocontrol
    QGate["T"](3) with nocontrol
105 QGate["not"](2) with controls=[+6] with nocontrol
    QGate["not"](3) with controls=[+2] with nocontrol
```

```
107  QGate["T"]*(6) with nocontrol
     QGate["T"]*(3) with nocontrol
109  QGate["not"](3) with controls=[+6] with nocontrol
     QGate["H"](6) with nocontrol
111  Comment["EXIT: cc_iX"](2:"c1.+", 3:"c2.-", 6:"q")
     Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
113  QGate["H"](4)
     QGate["T"](4)
115  QGate["T"](5)
     QGate["T"](6)
117  QGate["not"](6) with controls=[+5]
     QGate["not"](5) with controls=[+4]
119  QGate["not"](4) with controls=[+6]
     QGate["T"]*(5)
121  QGate["T"](4)
     QGate["not"](5) with controls=[+6]
123  QGate["T"]*(5)
     QGate["T"]*(6)
125  QGate["not"](5) with controls=[+4]
     QGate["not"](4) with controls=[+6]
127  QGate["not"](6) with controls=[+5]
     QGate["H"](4)
129  Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
     Comment["EXIT: cc_iX"]*(2:"c1.+", 3:"c2.-", 6:"q")
131  QGate["H"](6) with nocontrol
     QGate["not"](3) with controls=[+6] with nocontrol
133  QGate["T"](3) with nocontrol
     QGate["T"](6) with nocontrol
135  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
137  QGate["T"]*(3) with nocontrol
     QGate["T"]*(2) with nocontrol
139  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
141  QGate["H"](6) with nocontrol
     Comment["ENTER: cc_iX"]*(2:"c1.+", 3:"c2.-", 6:"q")
143  QTerm0(6) with nocontrol
     Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.+", 5:"q")
145  QGate["H"](5) with nocontrol
     QGate["not"](1) with controls=[+5] with nocontrol
147  QGate["T"]*(1) with nocontrol
     QGate["T"](5) with nocontrol
149  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
151  QGate["T"](1) with nocontrol
     QGate["T"]*(0) with nocontrol
153  QGate["not"](1) with controls=[+0] with nocontrol
```

```
       QGate["not"](0) with controls=[+5] with nocontrol
155  QGate["H"](5) with nocontrol
       Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.+", 5:"q")
157  QTerm0(5) with nocontrol
       QInit0(5) with nocontrol
159  Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.-", 5:"q")
       QGate["H"](5) with nocontrol
161  QGate["not"](0) with controls=[+5] with nocontrol
       QGate["not"](1) with controls=[+0] with nocontrol
163  QGate["T"](0) with nocontrol
       QGate["T"](1) with nocontrol
165  QGate["not"](0) with controls=[+5] with nocontrol
       QGate["not"](1) with controls=[+0] with nocontrol
167  QGate["T"]*(5) with nocontrol
       QGate["T"]*(1) with nocontrol
169  QGate["not"](1) with controls=[+5] with nocontrol
       QGate["H"](5) with nocontrol
171  Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.-", 5:"q")
       QInit0(6) with nocontrol
173  Comment["ENTER: cc_iX"](2:"c1.+", 3:"c2.-", 6:"q")
       QGate["H"](6) with nocontrol
175  QGate["not"](2) with controls=[+6] with nocontrol
       QGate["not"](3) with controls=[+2] with nocontrol
177  QGate["T"](2) with nocontrol
       QGate["T"](3) with nocontrol
179  QGate["not"](2) with controls=[+6] with nocontrol
       QGate["not"](3) with controls=[+2] with nocontrol
181  QGate["T"]*(6) with nocontrol
       QGate["T"]*(3) with nocontrol
183  QGate["not"](3) with controls=[+6] with nocontrol
       QGate["H"](6) with nocontrol
185  Comment["EXIT: cc_iX"](2:"c1.+", 3:"c2.-", 6:"q")
       Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
187  QGate["H"](4)
       QGate["T"](4)
189  QGate["T"](5)
       QGate["T"](6)
191  QGate["not"](6) with controls=[+5]
       QGate["not"](5) with controls=[+4]
193  QGate["not"](4) with controls=[+6]
       QGate["T"]*(5)
195  QGate["T"](4)
       QGate["not"](5) with controls=[+6]
197  QGate["T"]*(5)
       QGate["T"]*(6)
199  QGate["not"](5) with controls=[+4]
       QGate["not"](4) with controls=[+6]
```

```
201  QGate["not"](6) with controls=[+5]
     QGate["H"](4)
203  Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
     Comment["EXIT: cc_iX"]*(2:"c1.+", 3:"c2.-", 6:"q")
205  QGate["H"](6) with nocontrol
     QGate["not"](3) with controls=[+6] with nocontrol
207  QGate["T"](3) with nocontrol
     QGate["T"](6) with nocontrol
209  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
211  QGate["T"]*(3) with nocontrol
     QGate["T"]*(2) with nocontrol
213  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
215  QGate["H"](6) with nocontrol
     Comment["ENTER: cc_iX"]*(2:"c1.+", 3:"c2.-", 6:"q")
217  QTerm0(6) with nocontrol
     Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.-", 5:"q")
219  QGate["H"](5) with nocontrol
     QGate["not"](1) with controls=[+5] with nocontrol
221  QGate["T"](1) with nocontrol
     QGate["T"](5) with nocontrol
223  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
225  QGate["T"]*(1) with nocontrol
     QGate["T"]*(0) with nocontrol
227  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
229  QGate["H"](5) with nocontrol
     Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.-", 5:"q")
231  QTerm0(5) with nocontrol
     QInit0(5) with nocontrol
233  Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.-", 5:"q")
     QGate["H"](5) with nocontrol
235  QGate["not"](0) with controls=[+5] with nocontrol
     QGate["not"](1) with controls=[+0] with nocontrol
237  QGate["T"](0) with nocontrol
     QGate["T"](1) with nocontrol
239  QGate["not"](0) with controls=[+5] with nocontrol
     QGate["not"](1) with controls=[+0] with nocontrol
241  QGate["T"]*(5) with nocontrol
     QGate["T"]*(1) with nocontrol
243  QGate["not"](1) with controls=[+5] with nocontrol
     QGate["H"](5) with nocontrol
245  Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.-", 5:"q")
     QInit0(6) with nocontrol
247  Comment["ENTER: cc_iX"](2:"c1.-", 3:"c2.+", 6:"q")
```

```
    QGate["H"](6) with nocontrol
249 QGate["not"](2) with controls=[+6] with nocontrol
    QGate["not"](3) with controls=[+2] with nocontrol
251 QGate["T"]*(2) with nocontrol
    QGate["T"](3) with nocontrol
253 QGate["not"](2) with controls=[+6] with nocontrol
    QGate["not"](3) with controls=[+2] with nocontrol
255 QGate["T"]*(6) with nocontrol
    QGate["T"](3) with nocontrol
257 QGate["not"](3) with controls=[+6] with nocontrol
    QGate["H"](6) with nocontrol
259 Comment["EXIT: cc_iX"](2:"c1.-", 3:"c2.+", 6:"q")
    Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
261 QGate["H"](4)
    QGate["T"](4)
263 QGate["T"](5)
    QGate["T"](6)
265 QGate["not"](6) with controls=[+5]
    QGate["not"](5) with controls=[+4]
267 QGate["not"](4) with controls=[+6]
    QGate["T"]*(5)
269 QGate["T"](4)
    QGate["not"](5) with controls=[+6]
271 QGate["T"]*(5)
    QGate["T"]*(6)
273 QGate["not"](5) with controls=[+4]
    QGate["not"](4) with controls=[+6]
275 QGate["not"](6) with controls=[+5]
    QGate["H"](4)
277 Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
    Comment["EXIT: cc_iX"]*(2:"c1.-", 3:"c2.+", 6:"q")
279 QGate["H"](6) with nocontrol
    QGate["not"](3) with controls=[+6] with nocontrol
281 QGate["T"]*(3) with nocontrol
    QGate["T"](6) with nocontrol
283 QGate["not"](3) with controls=[+2] with nocontrol
    QGate["not"](2) with controls=[+6] with nocontrol
285 QGate["T"]*(3) with nocontrol
    QGate["T"](2) with nocontrol
287 QGate["not"](3) with controls=[+2] with nocontrol
    QGate["not"](2) with controls=[+6] with nocontrol
289 QGate["H"](6) with nocontrol
    Comment["ENTER: cc_iX"]*(2:"c1.-", 3:"c2.+", 6:"q")
291 QTerm0(6) with nocontrol
    Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.-", 5:"q")
293 QGate["H"](5) with nocontrol
    QGate["not"](1) with controls=[+5] with nocontrol
```

```
295   QGate["T"](1) with nocontrol
      QGate["T"](5) with nocontrol
297   QGate["not"](1) with controls=[+0] with nocontrol
      QGate["not"](0) with controls=[+5] with nocontrol
299   QGate["T"]*(1) with nocontrol
      QGate["T"]*(0) with nocontrol
301   QGate["not"](1) with controls=[+0] with nocontrol
      QGate["not"](0) with controls=[+5] with nocontrol
303   QGate["H"](5) with nocontrol
      Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.-", 5:"q")
305   QTerm0(5) with nocontrol
      QInit0(5) with nocontrol
307   Comment["ENTER: cc_iX"](0:"c1.+", 1:"c2.-", 5:"q")
      QGate["H"](5) with nocontrol
309   QGate["not"](0) with controls=[+5] with nocontrol
      QGate["not"](1) with controls=[+0] with nocontrol
311   QGate["T"](0) with nocontrol
      QGate["T"](1) with nocontrol
313   QGate["not"](0) with controls=[+5] with nocontrol
      QGate["not"](1) with controls=[+0] with nocontrol
315   QGate["T"]*(5) with nocontrol
      QGate["T"]*(1) with nocontrol
317   QGate["not"](1) with controls=[+5] with nocontrol
      QGate["H"](5) with nocontrol
319   Comment["EXIT: cc_iX"](0:"c1.+", 1:"c2.-", 5:"q")
      QInit0(6) with nocontrol
321   Comment["ENTER: cc_iX"](2:"c1.-", 3:"c2.-", 6:"q")
      QGate["H"](6) with nocontrol
323   QGate["not"](2) with controls=[+6] with nocontrol
      QGate["not"](3) with controls=[+2] with nocontrol
325   QGate["T"]*(2) with nocontrol
      QGate["T"]*(3) with nocontrol
327   QGate["not"](2) with controls=[+6] with nocontrol
      QGate["not"](3) with controls=[+2] with nocontrol
329   QGate["T"]*(6) with nocontrol
      QGate["T"]*(3) with nocontrol
331   QGate["not"](3) with controls=[+6] with nocontrol
      QGate["H"](6) with nocontrol
333   Comment["EXIT: cc_iX"](2:"c1.-", 3:"c2.-", 6:"q")
      Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
335   QGate["H"](4)
      QGate["T"](4)
337   QGate["T"](5)
      QGate["T"](6)
339   QGate["not"](6) with controls=[+5]
      QGate["not"](5) with controls=[+4]
341   QGate["not"](4) with controls=[+6]
```

```
    QGate["T"]*(5)
343 QGate["T"](4)
    QGate["not"](5) with controls=[+6]
345 QGate["T"]*(5)
    QGate["T"]*(6)
347 QGate["not"](5) with controls=[+4]
    QGate["not"](4) with controls=[+6]
349 QGate["not"](6) with controls=[+5]
    QGate["H"](4)
351 Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
    Comment["EXIT: cc_iX"]*(2:"c1.-", 3:"c2.-", 6:"q")
353 QGate["H"](6) with nocontrol
    QGate["not"](3) with controls=[+6] with nocontrol
355 QGate["T"](3) with nocontrol
    QGate["T"](6) with nocontrol
357 QGate["not"](3) with controls=[+2] with nocontrol
    QGate["not"](2) with controls=[+6] with nocontrol
359 QGate["T"](3) with nocontrol
    QGate["T"](2) with nocontrol
361 QGate["not"](3) with controls=[+2] with nocontrol
    QGate["not"](2) with controls=[+6] with nocontrol
363 QGate["H"](6) with nocontrol
    Comment["ENTER: cc_iX"]*(2:"c1.-", 3:"c2.-", 6:"q")
365 QTerm0(6) with nocontrol
    Comment["EXIT: cc_iX"]*(0:"c1.+", 1:"c2.-", 5:"q")
367 QGate["H"](5) with nocontrol
    QGate["not"](1) with controls=[+5] with nocontrol
369 QGate["T"](1) with nocontrol
    QGate["T"](5) with nocontrol
371 QGate["not"](1) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+5] with nocontrol
373 QGate["T"]*(1) with nocontrol
    QGate["T"]*(0) with nocontrol
375 QGate["not"](1) with controls=[+0] with nocontrol
    QGate["not"](0) with controls=[+5] with nocontrol
377 QGate["H"](5) with nocontrol
    Comment["ENTER: cc_iX"]*(0:"c1.+", 1:"c2.-", 5:"q")
379 QTerm0(5) with nocontrol
    QInit0(5) with nocontrol
381 Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.+", 5:"q")
    QGate["H"](5) with nocontrol
383 QGate["not"](0) with controls=[+5] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
385 QGate["T"]*(0) with nocontrol
    QGate["T"](1) with nocontrol
387 QGate["not"](0) with controls=[+5] with nocontrol
    QGate["not"](1) with controls=[+0] with nocontrol
```

```
389  QGate["T"]*(5) with nocontrol
     QGate["T"](1) with nocontrol
391  QGate["not"](1) with controls=[+5] with nocontrol
     QGate["H"](5) with nocontrol
393  Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.+", 5:"q")
     QInit0(6) with nocontrol
395  Comment["ENTER: cc_iX"](2:"c1.+", 3:"c2.-", 6:"q")
     QGate["H"](6) with nocontrol
397  QGate["not"](2) with controls=[+6] with nocontrol
     QGate["not"](3) with controls=[+2] with nocontrol
399  QGate["T"](2) with nocontrol
     QGate["T"](3) with nocontrol
401  QGate["not"](2) with controls=[+6] with nocontrol
     QGate["not"](3) with controls=[+2] with nocontrol
403  QGate["T"]*(6) with nocontrol
     QGate["T"]*(3) with nocontrol
405  QGate["not"](3) with controls=[+6] with nocontrol
     QGate["H"](6) with nocontrol
407  Comment["EXIT: cc_iX"](2:"c1.+", 3:"c2.-", 6:"q")
     Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
409  QGate["H"](4)
     QGate["T"](4)
411  QGate["T"](5)
     QGate["T"](6)
413  QGate["not"](6) with controls=[+5]
     QGate["not"](5) with controls=[+4]
415  QGate["not"](4) with controls=[+6]
     QGate["T"]*(5)
417  QGate["T"](4)
     QGate["not"](5) with controls=[+6]
419  QGate["T"]*(5)
     QGate["T"]*(6)
421  QGate["not"](5) with controls=[+4]
     QGate["not"](4) with controls=[+6]
423  QGate["not"](6) with controls=[+5]
     QGate["H"](4)
425  Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
     Comment["EXIT: cc_iX"]*(2:"c1.+", 3:"c2.-", 6:"q")
427  QGate["H"](6) with nocontrol
     QGate["not"](3) with controls=[+6] with nocontrol
429  QGate["T"](3) with nocontrol
     QGate["T"](6) with nocontrol
431  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
433  QGate["T"]*(3) with nocontrol
     QGate["T"]*(2) with nocontrol
435  QGate["not"](3) with controls=[+2] with nocontrol
```

```
      QGate["not"](2) with controls=[+6] with nocontrol
437   QGate["H"](6) with nocontrol
      Comment["ENTER: cc_iX"]*(2:"c1.+", 3:"c2.-", 6:"q")
439   QTerm0(6) with nocontrol
      Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.+", 5:"q")
441   QGate["H"](5) with nocontrol
      QGate["not"](1) with controls=[+5] with nocontrol
443   QGate["T"]*(1) with nocontrol
      QGate["T"](5) with nocontrol
445   QGate["not"](1) with controls=[+0] with nocontrol
      QGate["not"](0) with controls=[+5] with nocontrol
447   QGate["T"]*(1) with nocontrol
      QGate["T"](0) with nocontrol
449   QGate["not"](1) with controls=[+0] with nocontrol
      QGate["not"](0) with controls=[+5] with nocontrol
451   QGate["H"](5) with nocontrol
      Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.+", 5:"q")
453   QTerm0(5) with nocontrol
      QInit0(5) with nocontrol
455   Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.+", 5:"q")
      QGate["H"](5) with nocontrol
457   QGate["not"](0) with controls=[+5] with nocontrol
      QGate["not"](1) with controls=[+0] with nocontrol
459   QGate["T"]*(0) with nocontrol
      QGate["T"](1) with nocontrol
461   QGate["not"](0) with controls=[+5] with nocontrol
      QGate["not"](1) with controls=[+0] with nocontrol
463   QGate["T"]*(5) with nocontrol
      QGate["T"](1) with nocontrol
465   QGate["not"](1) with controls=[+5] with nocontrol
      QGate["H"](5) with nocontrol
467   Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.+", 5:"q")
      QInit0(6) with nocontrol
469   Comment["ENTER: cc_iX"](2:"c1.-", 3:"c2.+", 6:"q")
      QGate["H"](6) with nocontrol
471   QGate["not"](2) with controls=[+6] with nocontrol
      QGate["not"](3) with controls=[+2] with nocontrol
473   QGate["T"]*(2) with nocontrol
      QGate["T"](3) with nocontrol
475   QGate["not"](2) with controls=[+6] with nocontrol
      QGate["not"](3) with controls=[+2] with nocontrol
477   QGate["T"]*(6) with nocontrol
      QGate["T"](3) with nocontrol
479   QGate["not"](3) with controls=[+6] with nocontrol
      QGate["H"](6) with nocontrol
481   Comment["EXIT: cc_iX"](2:"c1.-", 3:"c2.+", 6:"q")
      Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
```

```
483  QGate["H"](4)
     QGate["T"](4)
485  QGate["T"](5)
     QGate["T"](6)
487  QGate["not"](6) with controls=[+5]
     QGate["not"](5) with controls=[+4]
489  QGate["not"](4) with controls=[+6]
     QGate["T"]*(5)
491  QGate["T"](4)
     QGate["not"](5) with controls=[+6]
493  QGate["T"]*(5)
     QGate["T"]*(6)
495  QGate["not"](5) with controls=[+4]
     QGate["not"](4) with controls=[+6]
497  QGate["not"](6) with controls=[+5]
     QGate["H"](4)
499  Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
     Comment["EXIT: cc_iX"]*(2:"c1.-", 3:"c2.+", 6:"q")
501  QGate["H"](6) with nocontrol
     QGate["not"](3) with controls=[+6] with nocontrol
503  QGate["T"]*(3) with nocontrol
     QGate["T"](6) with nocontrol
505  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
507  QGate["T"]*(3) with nocontrol
     QGate["T"](2) with nocontrol
509  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
511  QGate["H"](6) with nocontrol
     Comment["ENTER: cc_iX"]*(2:"c1.-", 3:"c2.+", 6:"q")
513  QTerm0(6) with nocontrol
     Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.+", 5:"q")
515  QGate["H"](5) with nocontrol
     QGate["not"](1) with controls=[+5] with nocontrol
517  QGate["T"]*(1) with nocontrol
     QGate["T"](5) with nocontrol
519  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
521  QGate["T"]*(1) with nocontrol
     QGate["T"](0) with nocontrol
523  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
525  QGate["H"](5) with nocontrol
     Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.+", 5:"q")
527  QTerm0(5) with nocontrol
     QInit0(5) with nocontrol
529  Comment["ENTER: cc_iX"](0:"c1.-", 1:"c2.+", 5:"q")
```

```
      QGate["H"](5) with nocontrol
531   QGate["not"](0) with controls=[+5] with nocontrol
      QGate["not"](1) with controls=[+0] with nocontrol
533   QGate["T"]*(0) with nocontrol
      QGate["T"](1) with nocontrol
535   QGate["not"](0) with controls=[+5] with nocontrol
      QGate["not"](1) with controls=[+0] with nocontrol
537   QGate["T"]*(5) with nocontrol
      QGate["T"](1) with nocontrol
539   QGate["not"](1) with controls=[+5] with nocontrol
      QGate["H"](5) with nocontrol
541   Comment["EXIT: cc_iX"](0:"c1.-", 1:"c2.+", 5:"q")
      QInit0(6) with nocontrol
543   Comment["ENTER: cc_iX"](2:"c1.-", 3:"c2.-", 6:"q")
      QGate["H"](6) with nocontrol
545   QGate["not"](2) with controls=[+6] with nocontrol
      QGate["not"](3) with controls=[+2] with nocontrol
547   QGate["T"]*(2) with nocontrol
      QGate["T"]*(3) with nocontrol
549   QGate["not"](2) with controls=[+6] with nocontrol
      QGate["not"](3) with controls=[+2] with nocontrol
551   QGate["T"]*(6) with nocontrol
      QGate["T"]*(3) with nocontrol
553   QGate["not"](3) with controls=[+6] with nocontrol
      QGate["H"](6) with nocontrol
555   Comment["EXIT: cc_iX"](2:"c1.-", 3:"c2.-", 6:"q")
      Comment["ENTER: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
557   QGate["H"](4)
      QGate["T"](4)
559   QGate["T"](5)
      QGate["T"](6)
561   QGate["not"](6) with controls=[+5]
      QGate["not"](5) with controls=[+4]
563   QGate["not"](4) with controls=[+6]
      QGate["T"]*(5)
565   QGate["T"](4)
      QGate["not"](5) with controls=[+6]
567   QGate["T"]*(5)
      QGate["T"]*(6)
569   QGate["not"](5) with controls=[+4]
      QGate["not"](4) with controls=[+6]
571   QGate["not"](6) with controls=[+5]
      QGate["H"](4)
573   Comment["EXIT: toffoli_AMMR"](4:"q", 5:"c1.+", 6:"c2.+")
      Comment["EXIT: cc_iX"]*(2:"c1.-", 3:"c2.-", 6:"q")
575   QGate["H"](6) with nocontrol
      QGate["not"](3) with controls=[+6] with nocontrol
```

```
577  QGate["T"](3) with nocontrol
     QGate["T"](6) with nocontrol
579  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
581  QGate["T"](3) with nocontrol
     QGate["T"](2) with nocontrol
583  QGate["not"](3) with controls=[+2] with nocontrol
     QGate["not"](2) with controls=[+6] with nocontrol
585  QGate["H"](6) with nocontrol
     Comment["ENTER: cc_iX"]*(2:"c1.-", 3:"c2.-", 6:"q")
587  QTerm0(6) with nocontrol
     Comment["EXIT: cc_iX"]*(0:"c1.-", 1:"c2.+", 5:"q")
589  QGate["H"](5) with nocontrol
     QGate["not"](1) with controls=[+5] with nocontrol
591  QGate["T"]*(1) with nocontrol
     QGate["T"](5) with nocontrol
593  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
595  QGate["T"]*(1) with nocontrol
     QGate["T"](0) with nocontrol
597  QGate["not"](1) with controls=[+0] with nocontrol
     QGate["not"](0) with controls=[+5] with nocontrol
599  QGate["H"](5) with nocontrol
     Comment["ENTER: cc_iX"]*(0:"c1.-", 1:"c2.+", 5:"q")
601  QTerm0(5) with nocontrol
     Comment["EXIT: exact_synthesis"](0:"q[0]", 1:"q[1]", 2:"q[2]", 3:"q[3]", 4:"q
        [4]")
603  Comment["after oracle"]()
     QMeas(4)
605  QMeas(3)
     QMeas(2)
607  QMeas(1)
     QMeas(0)
609  Outputs: 0:Cbit, 1:Cbit, 2:Cbit, 3:Cbit, 4:Cbit
```

Listing C.48: Description of decomposed circuit generated by fold quantamorphism over gate XOR.

```
         [(([False,False,False,False],False),1.0)]
```

Listing C.49: The input state was $q_0 = 0$, $q_1 = 0$, $q_2 = 0$, $q_3 = 0$ and $q_4 = 0$.

```
1        [(([False,False,False,False],True),1.0)]
```

Listing C.50: Adding X gates to q_4 changes the input state to $q_0 = 0$, $q_1 = 0$, $q_2 = 0$, $q_3 = 0$ and $q_4 = 1$.

```
1        [((([True,True,True,True],True),1.0)]
```

Listing C.51: Adding X gates to q_0 , q_1, q_2 and q_3 changes the input state to $q_0 = 1$, $q_1 = 1$, $q_2 = 1$, $q_3 = 1$ and $q_2 = 0$.

```
1        [((([True,True,True,True],False),1.0)]
```

Listing C.52: Adding X gates to all qubits changes the input state to $q_0 = 1$, $q_1 = 1$, $q_2 = 1$, $q_3 = 1$ and $q_4 = 1$.

```
1        [((([True,True,True,True],True),6.249999e-2),((([False,True,False,True],True
            ),6.249998e-2),((([True,False,False,True],True),6.249998e-2),((([True,
            True,True,False],True),6.2499996e-2),((([False,True,True,False],True)
            ,6.2499996e-2),((([True,False,True,False],True),6.2499993e-2),((([False,
            True,False,False],True),6.250001e-2),((([True,False,False,False],True)
            ,6.250001e-2),((([False,True,True,True],False),6.249999e-2),((([True,
            False,True,True],False),6.2499985e-2),((([False,False,True,True],False)
            ,6.2499985e-2),((([True,True,False,True],False),6.25e-2),((([False,False
            ,False,True],False),6.25e-2),((([False,False,True,False],False)
            ,6.2500015e-2),((([True,True,False,False],False),6.250001e-2),((([False,
            False,False,False],False),6.250001e-2)]
```

Listing C.53: Adding Hadamard gates the control qubits changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_3 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_4 = 0$.

```
1        [((([False,True,True,True],True),6.2499985e-2),((([True,False,True,True],
            True),6.249998e-2),((([False,False,True,True],True),6.249998e-2),((([
            True,True,False,True],True),6.2499996e-2),((([False,False,False,True],
            True),6.2499996e-2),((([False,False,True,False],True),6.250001e-2),((([
            True,True,False,False],True),6.25e-2),((([False,False,False,False],True
            ),6.25e-2),((([True,True,True,True],False),6.2499993e-2),((([False,True,
            False,True],False),6.2499985e-2),((([True,False,False,True],False)
            ,6.2499985e-2),((([True,True,True,False],False),6.25e-2),((([False,True,
            True,False],False),6.25e-2),((([True,False,True,False],False),6.2499996
            e-2),((([False,True,False,False],False),6.2500015e-2),((([True,False,
            False,False],False),6.2500015e-2)]
```

Listing C.54: Adding Hadamard gate the control qubits and X gate to the target qubit changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_3 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_4 = 1$.

```
1        [((([True,True,True,True],True),3.1249996e-2),((([False,True,True,True],True
            ),3.1249996e-2),((([True,False,True,True],True),3.1249996e-2),((([False,
            False,True,True],True),3.1249996e-2),((([True,True,False,True],True)
```

```
           ,3.1249998e-2),(([False,True,False,True],True),3.1249998e-2),(([True,
       False,False,True],True),3.1249998e-2),(([False,False,False,True],True)
       ,3.1249998e-2),(([True,True,True,False],True),3.1249998e-2),(([False,
       True,True,False],True),3.1249998e-2),(([True,False,True,False],True)
       ,3.1249998e-2),(([False,False,True,False],True),3.1249998e-2),(([True,
       True,False,False],True),3.125e-2),(([False,True,False,False],True)
       ,3.125e-2),(([True,False,False,False],True),3.125e-2),(([False,False,
       False,False],True),3.125e-2),(([True,True,True,True],False),3.1249996e
       -2),(([False,True,True,True],False),3.1249996e-2),(([True,False,True,
       True],False),3.1249996e-2),(([False,False,True,True],False),3.1249996e
       -2),(([True,True,False,True],False),3.1249998e-2),(([False,True,False,
       True],False),3.1249998e-2),(([True,False,False,True],False),3.1249998e
       -2),(([False,False,False,True],False),3.1249998e-2),(([True,True,True,
       False],False),3.1249998e-2),(([False,True,True,False],False),3.1249998
       e-2),(([True,False,True,False],False),3.1249998e-2),(([False,False,
       True,False],False),3.1249998e-2),(([True,True,False,False],False)
       ,3.125e-2),(([False,True,False,False],False),3.125e-2),(([True,False,
       False,False],False),3.125e-2),(([False,False,False,False],False),3.125
       e-2)]
```

Listing C.55: Adding Hadamard gate to all qubits changes the input state to $q_0 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_1 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_2 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, $q_3 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $q_4 = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.

QISKIT IMPLEMENTATION

This annex lists a number of important functions that (omitted from the main text for improved readability) are needed to run the experiments reported in the dissertation. Readers wishing to see a more detailed analysis of the case studies are referred to the repository set up by Neri (2018).

QUANTAMORPHISM OVER X GATE

```python
1  # Checking the version of PYTHON; we only support > 3.5
   import sys
3  sys.path.append('../../../Transferencias/qiskit-sdk-py-master')

5  if sys.version_info < (3,5):
       raise Exception('Please use Python version 3.5 or greater.')
7

   import qiskit
9  import numpy

11 # Import the QISKit
   from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister,
       QuantumProgram, Result
13 from qiskit import available_backends, execute, register, get_backend
   import getpass, time
15 import Qconfig
   from qiskit import compile

17


19 # import basic plot tools
   from qiskit.tools.visualization import plot_histogram, circuit_drawer,
       plot_state
21

   #information of API
23 from IBMQuantumExperience import IBMQuantumExperience

25 # Import tomography tools
   import qiskit.tools.qcvv.tomography as tomo
```

```
27
   # Aditional packages
29 from qiskit.tools.qi.qi import *
```

Listing D.1: Imports needed.

```
1 api = IBMQuantumExperience(Qconfig.APItoken)
  myCredits = api.get_my_credits()
3
  print(myCredits)
5
  # set the APIToken and API url
7 try:
      register(Qconfig.APItoken,Qconfig.config['url'])
9
      print('\nYou have access to great power!')
11     print(available_backends({'local': False, 'simulator': False}))
      print('Available simulators:')
13     print(available_backends({'simulator':True}))
  except:
15     print('Something went wrong.\nDid you enter a correct token?')
```

Listing D.2: Verification of credits and devices available.

```
1 # Creating Programs
  # create QuantumProgram object instance.
3 qp = QuantumProgram()
5 # Creating Registers
  # create Quantum Register called "qr" with 3 qubits
7 qr = qp.create_quantum_register('qr', 3)
  # create Classical Register  called "cr" with 3 bits
9 cr = qp.create_classical_register('cr', 3)
11 # Creating Circuits
  # create Quantum Circuit called "qc" involving your Quantum Register "qr"
13 # and your Classical Register "cr"
  qc = qp.create_circuit('Circuit', [qr], [cr])
```

Listing D.3: Creating the quantum program, circuit and corresponding quantum and classical registers.

```
1 QASM_source = qp.get_qasm('Circuit')
3 print(QASM_source)
```

Listing D.4: Get QASM of original circuit.

```
  #the circuit running is not the one we write
2 qobj = compile(qc, backend='local_qasm_simulator')

4 qasm_str = qp.get_compiled_qasm(qobj, 'Circuit')

6 qasm_circ = qiskit.load_qasm_string(qasm_str);
  circuit_drawer(qasm_circ)
```

Listing D.5: Get QASM compile in the local simulator.

Running in a real device.

```
   def lowest_pending_jobs():
2      """Returns the backend with lowest pending jobs."""
       list_of_backends = available_backends(
4          {'local': False, 'simulator': False})
       device_status = [get_backend(backend).status
6                         for backend in list_of_backends]

8      best = min([x for x in device_status if x['operational'] is True],
                   key=lambda x: x['pending_jobs'])
10     return best['name']

12 backend = lowest_pending_jobs()
```

Listing D.6: Lowest pending jobs function.

```
1 api.backend_calibration(backend)
```

Listing D.7: The information about the calibration of the chosen backend.

```
  api.backend_parameters(backend)
```

Listing D.8: The information about the parameters of the backend.

```
1 qobj = compile(qc, backend=backend)

3 qs_str = qp.get_compiled_qasm(qobj, 'Circuit')

5 qs_circ = qiskit.load_qasm_string(qs_str);
  circuit_drawer(qs_circ)
```

Listing D.9: Get the circuit that runs in the real device.

```
1  print(qs_str)
```

Listing D.10: Print QASM of circuit that runs in a real device.

```
   plot_histogram(job_exp.result().get_counts(qc))
```

Listing D.11: Obtain histogram.

```
1  job_exp.result().get_data('Circuit')
```

Listing D.12: Obtain information about the experiment - counts, time to run and date.

To prepare the inital states of the experiments done in the adapted circuit, the code of listings D.13 to D.16 has to be added before the circuit of the for-loop quantamorphism over the X gate with 2 control qubits.

```
1  qc.h(qr[0])
```

Listing D.13: State preparation - Hadamard gate in a the target qubit.

```
1  qc.x(qr[0])
   qc.x(qr[1])
3  qc.x(qr[2])
```

Listing D.14: State preparation - X gate in every qubit.

```
1  qc.h(qr[2])
   qc.cx(qr[2],qr[1])
```

Listing D.15: State preparation - Bell in control qubits.

```
   qc.h(qr[2])
2  qc.h(qr[1])
```

Listing D.16: State preparation - Hadarmd gates in control qubits.

When working with 3 control qubits the first change is concerned with register numbers (see listing D.17).

```
  # create Quantum Register called "qr" with 3 qubits
2 qr = qp.create_quantum_register('qr', 5)
  # create Classical Register  called "cr" with 3 bits
4 cr = qp.create_classical_register('cr', 5)
```

Listing D.17: Creating the program.

```
   qc.h(qr[4])
2  qc.cx(qr[4],qr[0])
   qc.cx(qr[0],qr[1])
4  qc.t(qr[0])
   qc.tdg(qr[1])
6  qc.cx(qr[4],qr[0])
   qc.cx(qr[0],qr[1])
8  qc.tdg(qr[4])
   qc.t(qr[1])
10 qc.cx(qr[4],qr[1])
   qc.h(qr[4])
12 qc.h(qr[3])
   qc.t(qr[3])
14 qc.t(qr[2])
   qc.t(qr[4])
16 qc.cx(qr[2],qr[4])
   qc.cx(qr[3],qr[2])
18 qc.cx(qr[4],qr[3])
   qc.tdg(qr[2])
20 qc.t(qr[3])
   qc.cx(qr[4],qr[2])
22 qc.tdg(qr[2])
   qc.tdg(qr[4])
24 qc.cx(qr[3],qr[2])
   qc.cx(qr[4],qr[3])
26 qc.cx(qr[2],qr[4])
   qc.h(qr[3])
28 qc.h(qr[4])
   qc.cx(qr[4],qr[1])
30 qc.tdg(qr[1])
   qc.t(qr[4])
32 qc.cx(qr[0],qr[1])
   qc.cx(qr[4],qr[0])
34 qc.t(qr[1])
   qc.tdg(qr[0])
36 qc.cx(qr[0],qr[1])
   qc.cx(qr[4],qr[0])
38 qc.h(qr[4])
   qc.barrier(qr[4])
```

```
40  qc.h(qr[4])
    qc.cx(qr[4],qr[0])
42  qc.cx(qr[0],qr[1])
    qc.t(qr[0])
44  qc.t(qr[1])
    qc.cx(qr[4],qr[0])
46  qc.cx(qr[0],qr[1])
    qc.tdg(qr[4])
48  qc.tdg(qr[1])
    qc.cx(qr[4],qr[1])
50  qc.h(qr[4])
    qc.h(qr[3])
52  qc.t(qr[3])
    qc.t(qr[2])
54  qc.t(qr[4])
    qc.cx(qr[2],qr[4])
56  qc.cx(qr[3],qr[2])
    qc.cx(qr[4],qr[3])
58  qc.tdg(qr[2])
    qc.t(qr[3])
60  qc.cx(qr[4],qr[2])
    qc.tdg(qr[2])
62  qc.tdg(qr[4])
    qc.cx(qr[3],qr[2])
64  qc.cx(qr[4],qr[3])
    qc.cx(qr[2],qr[4])
66  qc.h(qr[3])
    qc.h(qr[4])
68  qc.cx(qr[4],qr[1])
    qc.t(qr[1])
70  qc.t(qr[4])
    qc.cx(qr[0],qr[1])
72  qc.cx(qr[4],qr[0])
    qc.tdg(qr[1])
74  qc.tdg(qr[0])
    qc.cx(qr[0],qr[1])
76  qc.cx(qr[4],qr[0])
    qc.h(qr[4])
78  qc.barrier(qr[4])
    qc.h(qr[4])
80  qc.cx(qr[4],qr[0])
    qc.cx(qr[0],qr[1])
82  qc.tdg(qr[0])
    qc.t(qr[1])
84  qc.cx(qr[4],qr[0])
    qc.cx(qr[0],qr[1])
86  qc.tdg(qr[4])
```

```
     qc.t(qr[1])
 88  qc.cx(qr[4],qr[1])
     qc.h(qr[4])
 90  qc.h(qr[3])
     qc.t(qr[3])
 92  qc.t(qr[2])
     qc.t(qr[4])
 94  qc.cx(qr[2],qr[4])
     qc.cx(qr[3],qr[2])
 96  qc.cx(qr[4],qr[3])
     qc.tdg(qr[2])
 98  qc.t(qr[3])
     qc.cx(qr[4],qr[2])
100  qc.tdg(qr[2])
     qc.tdg(qr[4])
102  qc.cx(qr[3],qr[2])
     qc.cx(qr[4],qr[3])
104  qc.cx(qr[2],qr[4])
     qc.h(qr[3])
106  qc.h(qr[4])
     qc.cx(qr[4],qr[1])
108  qc.tdg(qr[1])
     qc.t(qr[4])
110  qc.cx(qr[0],qr[1])
     qc.cx(qr[4],qr[0])
112  qc.tdg(qr[1])
     qc.t(qr[0])
114  qc.cx(qr[0],qr[1])
     qc.cx(qr[4],qr[0])
116  qc.h(qr[4])
     qc.barrier(qr[4])
118  qc.h(qr[4])
     qc.cx(qr[4],qr[0])
120  qc.cx(qr[0],qr[1])
     qc.tdg(qr[0])
122  qc.tdg(qr[1])
     qc.cx(qr[4],qr[0])
124  qc.cx(qr[0],qr[1])
     qc.tdg(qr[4])
126  qc.tdg(qr[1])
     qc.cx(qr[4],qr[1])
128  qc.h(qr[4])
     qc.h(qr[3])
130  qc.t(qr[3])
     qc.t(qr[2])
132  qc.t(qr[4])
     qc.cx(qr[2],qr[4])
```

```
134  qc.cx(qr[3],qr[2])
     qc.cx(qr[4],qr[3])
136  qc.tdg(qr[2])
     qc.t(qr[3])
138  qc.cx(qr[4],qr[2])
     qc.tdg(qr[2])
140  qc.tdg(qr[4])
     qc.cx(qr[3],qr[2])
142  qc.cx(qr[4],qr[3])
     qc.cx(qr[2],qr[4])
144  qc.h(qr[3])
     qc.h(qr[4])
146  qc.cx(qr[4],qr[1])
     qc.t(qr[1])
148  qc.t(qr[4])
     qc.cx(qr[0],qr[1])
150  qc.cx(qr[4],qr[0])
     qc.t(qr[1])
152  qc.t(qr[0])
     qc.cx(qr[0],qr[1])
154  qc.cx(qr[4],qr[0])
     qc.h(qr[4])
```

Listing D.18: Circuit of `qfor` X with 3 control qubits.

QUANTAMORPHISM OVER Y GATE

```
   # create Quantum Register called "qr" with 4 qubits
2  qr = qp.create_quantum_register('qr', 4)
   # create Classical Register  called "cr" with 4 bits
4  cr = qp.create_classical_register('cr', 4)
```

Listing D.19: Creating the program.

```
   qc.h(qr[3])
2  qc.cx(qr[3],qr[0])
   qc.cx(qr[0],qr[1])
4  qc.t(qr[0])
   qc.tdg(qr[1])
6  qc.cx(qr[3],qr[0])
   qc.cx(qr[0],qr[1])
8  qc.tdg(qr[3])
   qc.t(qr[1])
10 qc.cx(qr[3],qr[1])
   qc.h(qr[3])
12 qc.tdg(qr[2])
```

```
   qc.tdg(qr[3])
14 qc.cx(qr[2],qr[3])
   qc.t(qr[3])
16 qc.cx(qr[2],qr[3])
   qc.h(qr[3])
18 qc.cx(qr[3],qr[1])
   qc.tdg(qr[1])
20 qc.t(qr[3])
   qc.cx(qr[0],qr[1])
22 qc.cx(qr[3],qr[0])
   qc.t(qr[1])
24 qc.tdg(qr[0])
   qc.cx(qr[0],qr[1])
26 qc.cx(qr[3],qr[0])
   qc.h(qr[3])
28 qc.barrier(qr[3])
   qc.h(qr[3])
30 qc.cx(qr[3],qr[0])
   qc.cx(qr[0],qr[2])
32 qc.t(qr[0])
   qc.t(qr[2])
34 qc.cx(qr[3],qr[0])
   qc.cx(qr[0],qr[2])
36 qc.tdg(qr[3])
   qc.tdg(qr[2])
38 qc.cx(qr[3],qr[2])
   qc.h(qr[3])
40 qc.cx(qr[1],qr[3])
   qc.tdg(qr[3])
42 qc.cx(qr[1],qr[3])
   qc.t(qr[3])
44 qc.t(qr[1])
   qc.h(qr[3])
46 qc.cx(qr[3],qr[2])
   qc.t(qr[2])
48 qc.t(qr[3])
   qc.cx(qr[0],qr[2])
50 qc.cx(qr[3],qr[0])
   qc.tdg(qr[2])
52 qc.tdg(qr[0])
   qc.cx(qr[0],qr[2])
54 qc.cx(qr[3],qr[0])
   qc.h(qr[3])
56 qc.h(qr[2])
   qc.t(qr[2])
58 qc.t(qr[0])
   qc.t(qr[1])
```

```
60  qc.cx(qr[0],qr[1])
    qc.cx(qr[2],qr[0])
62  qc.cx(qr[1],qr[2])
    qc.tdg(qr[0])
64  qc.t(qr[2])
    qc.cx(qr[1],qr[0])
66  qc.tdg(qr[0])
    qc.tdg(qr[1])
68  qc.cx(qr[2],qr[0])
    qc.cx(qr[1],qr[2])
70  qc.cx(qr[0],qr[1])
    qc.h(qr[2])
72  qc.barrier(qr[3])
    qc.h(qr[3])
74  qc.cx(qr[3],qr[0])
    qc.cx(qr[0],qr[1])
76  qc.tdg(qr[0])
    qc.t(qr[1])
78  qc.cx(qr[3],qr[0])
    qc.cx(qr[0],qr[1])
80  qc.tdg(qr[3])
    qc.t(qr[1])
82  qc.cx(qr[3],qr[1])
    qc.h(qr[3])
84  qc.tdg(qr[2])
    qc.tdg(qr[3])
86  qc.cx(qr[2],qr[3])
    qc.t(qr[3])
88  qc.cx(qr[2],qr[3])
    qc.h(qr[3])
90  qc.cx(qr[3],qr[1])
    qc.tdg(qr[1])
92  qc.t(qr[3])
    qc.cx(qr[0],qr[1])
94  qc.cx(qr[3],qr[0])
    qc.tdg(qr[1])
96  qc.t(qr[0])
    qc.cx(qr[0],qr[1])
98  qc.cx(qr[3],qr[0])
    qc.h(qr[3])
100 qc.barrier(qr[3])
    qc.h(qr[3])
102 qc.cx(qr[3],qr[0])
    qc.cx(qr[0],qr[2])
104 qc.tdg(qr[0])
    qc.tdg(qr[2])
106 qc.cx(qr[3],qr[0])
```

```
    qc.cx(qr[0],qr[2])
108 qc.tdg(qr[3])
    qc.tdg(qr[2])
110 qc.cx(qr[3],qr[2])
    qc.h(qr[3])
112 qc.cx(qr[1],qr[3])
    qc.tdg(qr[3])
114 qc.cx(qr[1],qr[3])
    qc.t(qr[3])
116 qc.t(qr[1])
    qc.h(qr[3])
118 qc.cx(qr[3],qr[2])
    qc.t(qr[2])
120 qc.t(qr[3])
    qc.cx(qr[0],qr[2])
122 qc.cx(qr[3],qr[0])
    qc.t(qr[2])
124 qc.t(qr[0])
    qc.cx(qr[0],qr[2])
126 qc.cx(qr[3],qr[0])
    qc.h(qr[3])
128 qc.h(qr[2])
    qc.t(qr[2])
130 qc.tdg(qr[0])
    qc.t(qr[1])
132 qc.cx(qr[0],qr[1])
    qc.cx(qr[2],qr[0])
134 qc.cx(qr[1],qr[2])
    qc.t(qr[0])
136 qc.tdg(qr[2])
    qc.cx(qr[1],qr[0])
138 qc.tdg(qr[0])
    qc.t(qr[1])
140 qc.cx(qr[2],qr[0])
    qc.cx(qr[1],qr[2])
142 qc.cx(qr[0],qr[1])
    qc.h(qr[2])
```

Listing D.20: Circuit qfor Y implemented in QISKit.

```
   qc.h(qr[0])
2  qc.cx(qr[0],qr[3])
   qc.cx(qr[3],qr[2])
4  qc.t(qr[3])
   qc.tdg(qr[2])
6  qc.cx(qr[0],qr[3])
   qc.cx(qr[3],qr[2])
```

```
 8  qc.tdg(qr[0])
    qc.t(qr[2])
10  qc.cx(qr[0],qr[2])
    qc.h(qr[0])
12  qc.tdg(qr[1])
    qc.tdg(qr[0])
14  qc.cx(qr[1],qr[0])
    qc.t(qr[0])
16  qc.cx(qr[1],qr[0])
    qc.h(qr[0])
18  qc.cx(qr[0],qr[2])
    qc.tdg(qr[2])
20  qc.t(qr[0])
    qc.cx(qr[3],qr[2])
22  qc.cx(qr[0],qr[3])
    qc.t(qr[2])
24  qc.tdg(qr[3])
    qc.cx(qr[3],qr[2])
26  qc.cx(qr[0],qr[3])
    qc.h(qr[0])
28  qc.barrier(qr[0])
    qc.h(qr[0])
30  qc.cx(qr[0],qr[3])
    qc.cx(qr[3],qr[1])
32  qc.t(qr[3])
    qc.t(qr[1])
34  qc.cx(qr[0],qr[3])
    qc.cx(qr[3],qr[1])
36  qc.tdg(qr[0])
    qc.tdg(qr[1])
38  qc.cx(qr[0],qr[1])
    qc.h(qr[0])
40  qc.cx(qr[2],qr[0])
    qc.tdg(qr[0])
42  qc.cx(qr[2],qr[0])
    qc.t(qr[0])
44  qc.t(qr[2])
    qc.h(qr[0])
46  qc.cx(qr[0],qr[1])
    qc.t(qr[1])
48  qc.t(qr[0])
    qc.cx(qr[3],qr[1])
50  qc.cx(qr[0],qr[3])
    qc.tdg(qr[1])
52  qc.tdg(qr[3])
    qc.cx(qr[3],qr[1])
54  qc.cx(qr[0],qr[3])
```

```
   qc.h(qr[0])
56 qc.h(qr[1])
   qc.t(qr[1])
58 qc.t(qr[3])
   qc.t(qr[2])
60 qc.cx(qr[3],qr[2])
   qc.cx(qr[1],qr[3])
62 qc.cx(qr[2],qr[1])
   qc.tdg(qr[3])
64 qc.t(qr[1])
   qc.cx(qr[2],qr[3])
66 qc.tdg(qr[3])
   qc.tdg(qr[2])
68 qc.cx(qr[1],qr[3])
   qc.cx(qr[2],qr[1])
70 qc.cx(qr[3],qr[2])
   qc.h(qr[1])
72 qc.barrier(qr[0])
   qc.h(qr[0])
74 qc.cx(qr[0],qr[3])
   qc.cx(qr[3],qr[2])
76 qc.tdg(qr[3])
   qc.t(qr[2])
78 qc.cx(qr[0],qr[3])
   qc.cx(qr[3],qr[2])
80 qc.tdg(qr[0])
   qc.t(qr[2])
82 qc.cx(qr[0],qr[2])
   qc.h(qr[0])
84 qc.tdg(qr[1])
   qc.tdg(qr[0])
86 qc.cx(qr[1],qr[0])
   qc.t(qr[0])
88 qc.cx(qr[1],qr[0])
   qc.h(qr[0])
90 qc.cx(qr[0],qr[2])
   qc.tdg(qr[2])
92 qc.t(qr[0])
   qc.cx(qr[3],qr[2])
94 qc.cx(qr[0],qr[3])
   qc.tdg(qr[2])
96 qc.t(qr[3])
   qc.cx(qr[3],qr[2])
98 qc.cx(qr[0],qr[3])
   qc.h(qr[0])
100 qc.barrier(qr[0])
   qc.h(qr[0])
```

```
102  qc.cx(qr[0],qr[3])
     qc.cx(qr[3],qr[1])
104  qc.tdg(qr[3])
     qc.tdg(qr[1])
106  qc.cx(qr[0],qr[3])
     qc.cx(qr[3],qr[1])
108  qc.tdg(qr[0])
     qc.tdg(qr[1])
110  qc.cx(qr[0],qr[1])
     qc.h(qr[0])
112  qc.cx(qr[2],qr[0])
     qc.tdg(qr[0])
114  qc.cx(qr[2],qr[0])
     qc.t(qr[0])
116  qc.t(qr[2])
     qc.h(qr[0])
118  qc.cx(qr[0],qr[1])
     qc.t(qr[1])
120  qc.t(qr[0])
     qc.cx(qr[3],qr[1])
122  qc.cx(qr[0],qr[3])
     qc.t(qr[1])
124  qc.t(qr[3])
     qc.cx(qr[3],qr[1])
126  qc.cx(qr[0],qr[3])
     qc.h(qr[0])
128  qc.h(qr[1])
     qc.t(qr[1])
130  qc.tdg(qr[3])
     qc.t(qr[2])
132  qc.cx(qr[3],qr[2])
     qc.cx(qr[1],qr[3])
134  qc.cx(qr[2],qr[1])
     qc.t(qr[3])
136  qc.tdg(qr[1])
     qc.cx(qr[2],qr[3])
138  qc.tdg(qr[3])
     qc.t(qr[2])
140  qc.cx(qr[1],qr[3])
     qc.cx(qr[2],qr[1])
142  qc.cx(qr[3],qr[2])
     qc.h(qr[1])
```

Listing D.21: Circuit of adapted qfor Y implemented in QISKit.

```
1  qc.measure(qr[1],cr[1])
   qc.measure(qr[2],cr[2])
```

```
3  qc.measure(qr[3],cr[3])
```

Listing D.22: Measurement gates of adapted `qfor` *Y* implemented in QISKit.

Running with 8192 shots and in `ibmqx5` was made with the adapted circuit and:

```
   shots=8192
2  max_credits=5
```

Listing D.23: Alteration needed to run with 8192 shots.

```
1  qc.x(qr[1])
   qc.x(qr[2])
3  qc.x(qr[3])
```

Listing D.24: State preparation - *X* gate in every qubit.

```
1  qc.h(qr[1])
```

Listing D.25: State preparation - Hadamard gate in target qubit.

```
1  qc.h(qr[3])
   qc.cx(qr[3],qr[2])
```

Listing D.26: State preparation - Bell in control qubits.

QUANTAMORPHISM OVER HADAMARD GATE

```
   qc.h(qr[3])
2  qc.cx(qr[3],qr[0])
   qc.cx(qr[0],qr[1])
4  qc.t(qr[0])
   qc.tdg(qr[1])
6  qc.cx(qr[3],qr[0])
   qc.cx(qr[0],qr[1])
8  qc.tdg(qr[3])
   qc.t(qr[1])
10 qc.cx(qr[3],qr[1])
   qc.h(qr[3])
12 qc.sdg(qr[2])
   qc.h(qr[2])
14 qc.tdg(qr[2])
   qc.cx(qr[3],qr[2])
16 qc.t(qr[2])
   qc.h(qr[2])
```

```
18  qc.s(qr[2])
    qc.h(qr[3])
20  qc.cx(qr[3],qr[1])
    qc.tdg(qr[1])
22  qc.t(qr[3])
    qc.cx(qr[0],qr[1])
24  qc.cx(qr[3],qr[0])
    qc.t(qr[1])
26  qc.tdg(qr[0])
    qc.cx(qr[0],qr[1])
28  qc.cx(qr[3],qr[0])
    qc.h(qr[3])
30  qc.barrier(qr[3])
    qc.h(qr[3])
32  qc.cx(qr[3],qr[0])
    qc.cx(qr[0],qr[1])
34  qc.tdg(qr[0])
    qc.t(qr[1])
36  qc.cx(qr[3],qr[0])
    qc.cx(qr[0],qr[1])
38  qc.tdg(qr[3])
    qc.t(qr[1])
40  qc.cx(qr[3],qr[1])
    qc.h(qr[3])
42  qc.sdg(qr[2])
    qc.h(qr[2])
44  qc.tdg(qr[2])
    qc.cx(qr[3],qr[2])
46  qc.t(qr[2])
    qc.h(qr[2])
48  qc.s(qr[2])
    qc.h(qr[3])
50  qc.cx(qr[3],qr[1])
    qc.tdg(qr[1])
52  qc.t(qr[3])
    qc.cx(qr[0],qr[1])
54  qc.cx(qr[3],qr[0])
    qc.tdg(qr[1])
56  qc.t(qr[0])
    qc.cx(qr[0],qr[1])
58  qc.cx(qr[3],qr[0])
    qc.h(qr[3])
```

Listing D.27: Circuit quantamorphism for H implemented in QISKit.

The measurements in the original circuit are equal to 5.18.

```
1  qc.h(qr[0])
```

```
   qc.cx(qr[0],qr[3])
 3 qc.cx(qr[3],qr[2])
   qc.t(qr[3])
 5 qc.tdg(qr[2])
   qc.cx(qr[0],qr[3])
 7 qc.cx(qr[3],qr[2])
   qc.tdg(qr[0])
 9 qc.t(qr[2])
   qc.cx(qr[0],qr[2])
11 qc.h(qr[0])
   qc.sdg(qr[1])
13 qc.h(qr[1])
   qc.tdg(qr[1])
15 qc.cx(qr[0],qr[1])
   qc.t(qr[1])
17 qc.h(qr[1])
   qc.s(qr[1])
19 qc.h(qr[0])
   qc.cx(qr[0],qr[2])
21 qc.tdg(qr[2])
   qc.t(qr[0])
23 qc.cx(qr[3],qr[2])
   qc.cx(qr[0],qr[3])
25 qc.t(qr[2])
   qc.tdg(qr[3])
27 qc.cx(qr[3],qr[2])
   qc.cx(qr[0],qr[3])
29 qc.h(qr[0])
   qc.barrier(qr[0])
31 qc.h(qr[0])
   qc.cx(qr[0],qr[3])
33 qc.cx(qr[3],qr[2])
   qc.tdg(qr[3])
35 qc.t(qr[2])
   qc.cx(qr[0],qr[3])
37 qc.cx(qr[3],qr[2])
   qc.tdg(qr[0])
39 qc.t(qr[2])
   qc.cx(qr[0],qr[2])
41 qc.h(qr[0])
   qc.sdg(qr[1])
43 qc.h(qr[1])
   qc.tdg(qr[1])
45 qc.cx(qr[0],qr[1])
   qc.t(qr[1])
47 qc.h(qr[1])
   qc.s(qr[1])
```

```
49  qc.h(qr[0])
    qc.cx(qr[0],qr[2])
51  qc.tdg(qr[2])
    qc.t(qr[0])
53  qc.cx(qr[3],qr[2])
    qc.cx(qr[0],qr[3])
55  qc.tdg(qr[2])
    qc.t(qr[3])
57  qc.cx(qr[3],qr[2])
    qc.cx(qr[0],qr[3])
59  qc.h(qr[0])
```

Listing D.28: Circuit quantamorphism for H adapted to QISKit.

The measurements in the adapted circuit are equal to D.22.

The experiments are made with the original circuit.

```
1  qc.x(qr[0])
   qc.x(qr[1])
```

Listing D.29: State preparation - X gate in control qubits.

```
   qc.x(qr[0])
2  qc.x(qr[1])
   qc.x(qr[2])
```

Listing D.30: State preparation - X gate in all qubits.

```
1  qc.x(qr[0])
   qc.x(qr[1])
3  qc.h(qr[2])
```

Listing D.31: State preparation - X gate in control qubits and Hadamard gate in target qubit.

```
1  qc.x(qr[0])
   qc.x(qr[1])
3  qc.x(qr[2])
   qc.h(qr[2])
```

Listing D.32: State preparation - X gate in all qubits and Hadamard gate in target qubit.

```
   qc.h(qr[0])
2  qc.cx(qr[0],qr[1])
```

Listing D.33: State preparation - Bell in control qubits.

```
  # create Quantum Register called "qr" with 7 qubits
2 qr = qp.create_quantum_register('qr', 7)
  # create Classical Register  called "cr" with 7 bits
4 cr = qp.create_classical_register('cr', 7)
```

Listing D.34: Creating the program.

```
   qc.h(qr[5])
2  qc.cx(qr[5],qr[0])
   qc.cx(qr[0],qr[1])
4  qc.t(qr[0])
   qc.tdg(qr[1])
6  qc.cx(qr[5],qr[0])
   qc.cx(qr[0],qr[1])
8  qc.tdg(qr[5])
   qc.t(qr[1])
10 qc.cx(qr[5],qr[1])
   qc.h(qr[5])
12 qc.h(qr[6])
   qc.cx(qr[6],qr[2])
14 qc.cx(qr[2],qr[3])
   qc.t(qr[2])
16 qc.tdg(qr[3])
   qc.cx(qr[6],qr[2])
18 qc.cx(qr[2],qr[3])
   qc.tdg(qr[6])
20 qc.t(qr[3])
   qc.cx(qr[6],qr[3])
22 qc.h(qr[6])
   qc.h(qr[4])
24 qc.t(qr[4])
   qc.t(qr[5])
26 qc.t(qr[6])
   qc.cx(qr[5],qr[6])
28 qc.cx(qr[4],qr[5])
   qc.cx(qr[6],qr[4])
30 qc.tdg(qr[5])
   qc.t(qr[4])
32 qc.cx(qr[6],qr[5])
   qc.tdg(qr[5])
34 qc.tdg(qr[6])
   qc.cx(qr[4],qr[5])
36 qc.cx(qr[6],qr[4])
   qc.cx(qr[5],qr[6])
38 qc.h(qr[4])
   qc.h(qr[6])
```

```
40  qc.cx(qr[6],qr[3])
    qc.tdg(qr[3])
42  qc.t(qr[6])
    qc.cx(qr[2],qr[3])
44  qc.cx(qr[6],qr[2])
    qc.t(qr[3])
46  qc.tdg(qr[2])
    qc.cx(qr[2],qr[3])
48  qc.cx(qr[6],qr[2])
    qc.h(qr[6])
50  qc.h(qr[5])
    qc.cx(qr[5],qr[1])
52  qc.tdg(qr[1])
    qc.t(qr[5])
54  qc.cx(qr[0],qr[1])
    qc.cx(qr[5],qr[0])
56  qc.t(qr[1])
    qc.tdg(qr[0])
58  qc.cx(qr[0],qr[1])
    qc.cx(qr[5],qr[0])
60  qc.h(qr[5])
    qc.barrier(qr[5])
62  qc.h(qr[5])
    qc.cx(qr[5],qr[0])
64  qc.cx(qr[0],qr[1])
    qc.t(qr[0])
66  qc.tdg(qr[1])
    qc.cx(qr[5],qr[0])
68  qc.cx(qr[0],qr[1])
    qc.tdg(qr[5])
70  qc.t(qr[1])
    qc.cx(qr[5],qr[1])
72  qc.h(qr[5])
    qc.barrier(qr[6])
74  qc.h(qr[6])
    qc.cx(qr[6],qr[2])
76  qc.cx(qr[2],qr[3])
    qc.t(qr[2])
78  qc.t(qr[3])
    qc.cx(qr[6],qr[2])
80  qc.cx(qr[2],qr[3])
    qc.tdg(qr[6])
82  qc.tdg(qr[3])
    qc.cx(qr[6],qr[3])
84  qc.h(qr[6])
    qc.h(qr[4])
86  qc.t(qr[4])
```

```
    qc.t(qr[5])
88  qc.t(qr[6])
    qc.cx(qr[5],qr[6])
90  qc.cx(qr[4],qr[5])
    qc.cx(qr[6],qr[4])
92  qc.tdg(qr[5])
    qc.t(qr[4])
94  qc.cx(qr[6],qr[5])
    qc.tdg(qr[5])
96  qc.tdg(qr[6])
    qc.cx(qr[4],qr[5])
98  qc.cx(qr[6],qr[4])
    qc.cx(qr[5],qr[6])
100 qc.h(qr[4])
    qc.h(qr[6])
102 qc.cx(qr[6],qr[3])
    qc.t(qr[3])
104 qc.t(qr[6])
    qc.cx(qr[2],qr[3])
106 qc.cx(qr[6],qr[2])
    qc.tdg(qr[3])
108 qc.tdg(qr[2])
    qc.cx(qr[2],qr[3])
110 qc.cx(qr[6],qr[2])
    qc.h(qr[6])
112 qc.h(qr[5])
    qc.cx(qr[5],qr[1])
114 qc.tdg(qr[1])
    qc.t(qr[5])
116 qc.cx(qr[0],qr[1])
    qc.cx(qr[5],qr[0])
118 qc.t(qr[1])
    qc.tdg(qr[0])
120 qc.cx(qr[0],qr[1])
    qc.cx(qr[5],qr[0])
122 qc.h(qr[5])
    qc.barrier(qr[5])
124 qc.h(qr[5])
    qc.cx(qr[5],qr[0])
126 qc.cx(qr[0],qr[1])
    qc.t(qr[0])
128 qc.t(qr[1])
    qc.cx(qr[5],qr[0])
130 qc.cx(qr[0],qr[1])
    qc.tdg(qr[5])
132 qc.tdg(qr[1])
    qc.cx(qr[5],qr[1])
```

```
134  qc.h(qr[5])
     qc.barrier(qr[6])
136  qc.h(qr[6])
     qc.cx(qr[6],qr[2])
138  qc.cx(qr[2],qr[3])
     qc.t(qr[2])
140  qc.t(qr[3])
     qc.cx(qr[6],qr[2])
142  qc.cx(qr[2],qr[3])
     qc.tdg(qr[6])
144  qc.tdg(qr[3])
     qc.cx(qr[6],qr[3])
146  qc.h(qr[6])
     qc.h(qr[4])
148  qc.t(qr[4])
     qc.t(qr[5])
150  qc.t(qr[6])
     qc.cx(qr[5],qr[6])
152  qc.cx(qr[4],qr[5])
     qc.cx(qr[6],qr[4])
154  qc.tdg(qr[5])
     qc.t(qr[4])
156  qc.cx(qr[6],qr[5])
     qc.tdg(qr[5])
158  qc.tdg(qr[6])
     qc.cx(qr[4],qr[5])
160  qc.cx(qr[6],qr[4])
     qc.cx(qr[5],qr[6])
162  qc.h(qr[4])
     qc.h(qr[6])
164  qc.cx(qr[6],qr[3])
     qc.t(qr[3])
166  qc.t(qr[6])
     qc.cx(qr[2],qr[3])
168  qc.cx(qr[6],qr[2])
     qc.tdg(qr[3])
170  qc.tdg(qr[2])
     qc.cx(qr[2],qr[3])
172  qc.cx(qr[6],qr[2])
     qc.h(qr[6])
174  qc.h(qr[5])
     qc.cx(qr[5],qr[1])
176  qc.t(qr[1])
     qc.t(qr[5])
178  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
180  qc.tdg(qr[1])
```

```
      qc.tdg(qr[0])
182   qc.cx(qr[0],qr[1])
      qc.cx(qr[5],qr[0])
184   qc.h(qr[5])
      qc.barrier(qr[5])
186   qc.h(qr[5])
      qc.cx(qr[5],qr[0])
188   qc.cx(qr[0],qr[1])
      qc.t(qr[0])
190   qc.t(qr[1])
      qc.cx(qr[5],qr[0])
192   qc.cx(qr[0],qr[1])
      qc.tdg(qr[5])
194   qc.tdg(qr[1])
      qc.cx(qr[5],qr[1])
196   qc.h(qr[5])
      qc.barrier(qr[6])
198   qc.h(qr[6])
      qc.cx(qr[6],qr[2])
200   qc.cx(qr[2],qr[3])
      qc.tdg(qr[2])
202   qc.t(qr[3])
      qc.cx(qr[6],qr[2])
204   qc.cx(qr[2],qr[3])
      qc.tdg(qr[6])
206   qc.t(qr[3])
      qc.cx(qr[6],qr[3])
208   qc.h(qr[6])
      qc.h(qr[4])
210   qc.t(qr[4])
      qc.t(qr[5])
212   qc.t(qr[6])
      qc.cx(qr[5],qr[6])
214   qc.cx(qr[4],qr[5])
      qc.cx(qr[6],qr[4])
216   qc.tdg(qr[5])
      qc.t(qr[4])
218   qc.cx(qr[6],qr[5])
      qc.tdg(qr[5])
220   qc.tdg(qr[6])
      qc.cx(qr[4],qr[5])
222   qc.cx(qr[6],qr[4])
      qc.cx(qr[5],qr[6])
224   qc.h(qr[4])
      qc.h(qr[6])
226   qc.cx(qr[6],qr[3])
      qc.tdg(qr[3])
```

```
228  qc.t(qr[6])
     qc.cx(qr[2],qr[3])
230  qc.cx(qr[6],qr[2])
     qc.tdg(qr[3])
232  qc.t(qr[2])
     qc.cx(qr[2],qr[3])
234  qc.cx(qr[6],qr[2])
     qc.h(qr[6])
236  qc.h(qr[5])
     qc.cx(qr[5],qr[1])
238  qc.t(qr[1])
     qc.t(qr[5])
240  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
242  qc.tdg(qr[1])
     qc.tdg(qr[0])
244  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
246  qc.h(qr[5])
     qc.barrier(qr[5])
248  qc.h(qr[5])
     qc.cx(qr[5],qr[0])
250  qc.cx(qr[0],qr[1])
     qc.t(qr[0])
252  qc.t(qr[1])
     qc.cx(qr[5],qr[0])
254  qc.cx(qr[0],qr[1])
     qc.tdg(qr[5])
256  qc.tdg(qr[1])
     qc.cx(qr[5],qr[1])
258  qc.h(qr[5])
     qc.barrier(qr[6])
260  qc.h(qr[6])
     qc.cx(qr[6],qr[2])
262  qc.cx(qr[2],qr[3])
     qc.tdg(qr[2])
264  qc.tdg(qr[3])
     qc.cx(qr[6],qr[2])
266  qc.cx(qr[2],qr[3])
     qc.tdg(qr[6])
268  qc.tdg(qr[3])
     qc.cx(qr[6],qr[3])
270  qc.h(qr[6])
     qc.h(qr[4])
272  qc.t(qr[4])
     qc.t(qr[5])
274  qc.t(qr[6])
```

```
     qc.cx(qr[5],qr[6])
276  qc.cx(qr[4],qr[5])
     qc.cx(qr[6],qr[4])
278  qc.tdg(qr[5])
     qc.t(qr[4])
280  qc.cx(qr[6],qr[5])
     qc.tdg(qr[5])
282  qc.tdg(qr[6])
     qc.cx(qr[4],qr[5])
284  qc.cx(qr[6],qr[4])
     qc.cx(qr[5],qr[6])
286  qc.h(qr[4])
     qc.h(qr[6])
288  qc.cx(qr[6],qr[3])
     qc.t(qr[3])
290  qc.t(qr[6])
     qc.cx(qr[2],qr[3])
292  qc.cx(qr[6],qr[2])
     qc.t(qr[3])
294  qc.t(qr[2])
     qc.cx(qr[2],qr[3])
296  qc.cx(qr[6],qr[2])
     qc.h(qr[6])
298  qc.h(qr[5])
     qc.cx(qr[5],qr[1])
300  qc.t(qr[1])
     qc.t(qr[5])
302  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
304  qc.tdg(qr[1])
     qc.tdg(qr[0])
306  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
308  qc.h(qr[5])
     qc.barrier(qr[5])
310  qc.h(qr[5])
     qc.cx(qr[5],qr[0])
312  qc.cx(qr[0],qr[1])
     qc.tdg(qr[0])
314  qc.t(qr[1])
     qc.cx(qr[5],qr[0])
316  qc.cx(qr[0],qr[1])
     qc.tdg(qr[5])
318  qc.t(qr[1])
     qc.cx(qr[5],qr[1])
320  qc.h(qr[5])
     qc.barrier(qr[6])
```

```
322  qc.h(qr[6])
     qc.cx(qr[6],qr[2])
324  qc.cx(qr[2],qr[3])
     qc.t(qr[2])
326  qc.t(qr[3])
     qc.cx(qr[6],qr[2])
328  qc.cx(qr[2],qr[3])
     qc.tdg(qr[6])
330  qc.tdg(qr[3])
     qc.cx(qr[6],qr[3])
332  qc.h(qr[6])
     qc.h(qr[4])
334  qc.t(qr[4])
     qc.t(qr[5])
336  qc.t(qr[6])
     qc.cx(qr[5],qr[6])
338  qc.cx(qr[4],qr[5])
     qc.cx(qr[6],qr[4])
340  qc.tdg(qr[5])
     qc.t(qr[4])
342  qc.cx(qr[6],qr[5])
     qc.tdg(qr[5])
344  qc.tdg(qr[6])
     qc.cx(qr[4],qr[5])
346  qc.cx(qr[6],qr[4])
     qc.cx(qr[5],qr[6])
348  qc.h(qr[4])
     qc.h(qr[6])
350  qc.cx(qr[6],qr[3])
     qc.t(qr[3])
352  qc.t(qr[6])
     qc.cx(qr[2],qr[3])
354  qc.cx(qr[6],qr[2])
     qc.tdg(qr[3])
356  qc.tdg(qr[2])
     qc.cx(qr[2],qr[3])
358  qc.cx(qr[6],qr[2])
     qc.h(qr[6])
360  qc.h(qr[5])
     qc.cx(qr[5],qr[1])
362  qc.tdg(qr[1])
     qc.t(qr[5])
364  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
366  qc.tdg(qr[1])
     qc.t(qr[0])
368  qc.cx(qr[0],qr[1])
```

```
    qc.cx(qr[5],qr[0])
370 qc.h(qr[5])
    qc.barrier(qr[5])
372 qc.h(qr[5])
    qc.cx(qr[5],qr[0])
374 qc.cx(qr[0],qr[1])
    qc.tdg(qr[0])
376 qc.t(qr[1])
    qc.cx(qr[5],qr[0])
378 qc.cx(qr[0],qr[1])
    qc.tdg(qr[5])
380 qc.t(qr[1])
    qc.cx(qr[5],qr[1])
382 qc.h(qr[5])
    qc.barrier(qr[6])
384 qc.h(qr[6])
    qc.cx(qr[6],qr[2])
386 qc.cx(qr[2],qr[3])
    qc.tdg(qr[2])
388 qc.t(qr[3])
    qc.cx(qr[6],qr[2])
390 qc.cx(qr[2],qr[3])
    qc.tdg(qr[6])
392 qc.t(qr[3])
    qc.cx(qr[6],qr[3])
394 qc.h(qr[6])
    qc.h(qr[4])
396 qc.t(qr[4])
    qc.t(qr[5])
398 qc.t(qr[6])
    qc.cx(qr[5],qr[6])
400 qc.cx(qr[4],qr[5])
    qc.cx(qr[6],qr[4])
402 qc.tdg(qr[5])
    qc.t(qr[4])
404 qc.cx(qr[6],qr[5])
    qc.tdg(qr[5])
406 qc.tdg(qr[6])
    qc.cx(qr[4],qr[5])
408 qc.cx(qr[6],qr[4])
    qc.cx(qr[5],qr[6])
410 qc.h(qr[4])
    qc.h(qr[6])
412 qc.cx(qr[6],qr[3])
    qc.tdg(qr[3])
414 qc.t(qr[6])
    qc.cx(qr[2],qr[3])
```

```
416  qc.cx(qr[6],qr[2])
     qc.tdg(qr[3])
418  qc.t(qr[2])
     qc.cx(qr[2],qr[3])
420  qc.cx(qr[6],qr[2])
     qc.h(qr[6])
422  qc.h(qr[5])
     qc.cx(qr[5],qr[1])
424  qc.tdg(qr[1])
     qc.t(qr[5])
426  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
428  qc.tdg(qr[1])
     qc.t(qr[0])
430  qc.cx(qr[0],qr[1])
     qc.cx(qr[5],qr[0])
432  qc.h(qr[5])
     qc.barrier(qr[5])
434  qc.h(qr[5])
     qc.cx(qr[5],qr[0])
436  qc.cx(qr[0],qr[1])
     qc.tdg(qr[0])
438  qc.t(qr[1])
     qc.cx(qr[5],qr[0])
440  qc.cx(qr[0],qr[1])
     qc.tdg(qr[5])
442  qc.t(qr[1])
     qc.cx(qr[5],qr[1])
444  qc.h(qr[5])
     qc.barrier(qr[6])
446  qc.h(qr[6])
     qc.cx(qr[6],qr[2])
448  qc.cx(qr[2],qr[3])
     qc.tdg(qr[2])
450  qc.tdg(qr[3])
     qc.cx(qr[6],qr[2])
452  qc.cx(qr[2],qr[3])
     qc.tdg(qr[6])
454  qc.tdg(qr[3])
     qc.cx(qr[6],qr[3])
456  qc.h(qr[6])
     qc.h(qr[4])
458  qc.t(qr[4])
     qc.t(qr[5])
460  qc.t(qr[6])
     qc.cx(qr[5],qr[6])
462  qc.cx(qr[4],qr[5])
```

```
    qc.cx(qr[6],qr[4])
464 qc.tdg(qr[5])
    qc.t(qr[4])
466 qc.cx(qr[6],qr[5])
    qc.tdg(qr[5])
468 qc.tdg(qr[6])
    qc.cx(qr[4],qr[5])
470 qc.cx(qr[6],qr[4])
    qc.cx(qr[5],qr[6])
472 qc.h(qr[4])
    qc.h(qr[6])
474 qc.cx(qr[6],qr[3])
    qc.t(qr[3])
476 qc.t(qr[6])
    qc.cx(qr[2],qr[3])
478 qc.cx(qr[6],qr[2])
    qc.t(qr[3])
480 qc.t(qr[2])
    qc.cx(qr[2],qr[3])
482 qc.cx(qr[6],qr[2])
    qc.h(qr[6])
484 qc.h(qr[5])
    qc.cx(qr[5],qr[1])
486 qc.tdg(qr[1])
    qc.t(qr[5])
488 qc.cx(qr[0],qr[1])
    qc.cx(qr[5],qr[0])
490 qc.tdg(qr[1])
    qc.t(qr[0])
492 qc.cx(qr[0],qr[1])
    qc.cx(qr[5],qr[0])
494 qc.h(qr[5])
    # don't forget update the number of qubits and bits registered
```

Listing D.35: Circuit quantamorphism foldr XOR implemented in QISKit.

```
    qc.barrier(qr[4])
 2  qc.barrier(qr[3])
    qc.barrier(qr[2])
 4  qc.barrier(qr[1])
    qc.barrier(qr[0])
 6
    qc.measure(qr[4],cr[4])
 8  qc.measure(qr[3],cr[3])
    qc.measure(qr[2],cr[2])
10  qc.measure(qr[1],cr[1])
    qc.measure(qr[0],cr[0])
```

Listing D.36: Measurement gates of circuit quantamorphism foldr XOR implemented in QISKit.

The adaptation only changes $0 \to 4$, $1 \to 3$, $3 \to 1$ and $4 \to 0$ as ilustrated by the measures in listing D.37.

```
1  qc.measure(qr[4],cr[4])
   qc.measure(qr[3],cr[3])
3  qc.measure(qr[2],cr[2])
   qc.measure(qr[1],cr[1])
5  qc.measure(qr[0],cr[0])
```

Listing D.37: Measurement gates of circuit quantamorphism foldr XOR adapted in QISKit.

```
1  qc.x(qr[2])
   qc.x(qr[3])
3  qc.x(qr[4])
   qc.x(qr[5])
5  qc.x(qr[6])
```

Listing D.38: State preparation - X gate in all qubits.

```
1  qc.x(qr[3])
   qc.x(qr[4])
3  qc.x(qr[5])
   qc.x(qr[6])
```

Listing D.39: State preparation - X gate in control qubits.

```
qc.h(qr[2])
```

Listing D.40: State preparation - Hadamard gate in target qubit.

RUNNING CIRCUITS IN `IBMQ_20_TOKYO`   The only problems ran in this devices were the for-loop quantamorphism over Hadamard gate . The alteration needed start in the `register` function:

```
register(Qconfig.APItoken,Qconfig.config['url'],Qconfig.config['hub'],Qconfig.
    config['group'],Qconfig.config['project'])
```

Listing D.41: Register in account with access to `ibmq_20_tokyo`.

Besides this the only alteration need is to change the backend:

```
1  backend='ibmq_20_tokyo'
```

Listing D.42: Defining the backend.

QISKIT RESULTS

Similarly to annex D, further details about the information in this annex can be found in (Neri, 2018).

```
1  {'promotional': 0, 'remaining': 15, 'maxUserType': 15}

3  You have access to great power!
   ['ibmq_16_rueschlikon', 'ibmq_5_tenerife', 'ibmq_5_yorktown']
5  Available simulators:
   ['ibmq_qasm_simulator', 'local_qasm_simulator', 'local_statevector_simulator',
        'local_unitary_simulator']
```

Listing E.1: The output of D.2.

FOR-LOOP QUANTAMORPHISM OVER X GATE

```
   array([[ 1.-0.j,  0.+0.j,  0.+0.j,  0.+0.j, -0.+0.j, -0.-0.j,  0.+0.j,
2           0.+0.j],
        [-0.+0.j,  1.-0.j,  0.+0.j,  0.+0.j,  0.-0.j, -0.+0.j,  0.+0.j,
4           0.+0.j],
        [ 0.+0.j,  0.+0.j,  0.+0.j, -0.+0.j,  0.+0.j,  0.+0.j,  1.-0.j,
6           0.-0.j],
        [ 0.+0.j,  0.+0.j,  0.-0.j,  0.+0.j,  0.+0.j,  0.+0.j, -0.+0.j,
8           1.-0.j],
        [ 0.+0.j, -0.-0.j,  0.+0.j,  0.+0.j,  1.-0.j,  0.+0.j,  0.+0.j,
10          0.+0.j],
        [ 0.-0.j,  0.+0.j,  0.+0.j,  0.+0.j, -0.+0.j,  1.-0.j,  0.+0.j,
12          0.+0.j],
        [ 0.+0.j,  0.+0.j,  1.-0.j,  0.-0.j,  0.+0.j,  0.+0.j, -0.+0.j,
14          -0.+0.j],
        [ 0.+0.j,  0.+0.j, -0.+0.j,  1.-0.j,  0.+0.j,  0.+0.j,  0.-0.j,
16          -0.+0.j]])
```

Listing E.2: Output of 5.20.

```
   OPENQASM 2.0;
2  include "qelib1.inc";
   qreg qr[3];
4  creg cr[3];
   h qr[2];
6  cx qr[1],qr[2];
   tdg qr[2];
8  cx qr[0],qr[2];
   t qr[2];
10 cx qr[1],qr[2];
   t qr[1];
12 tdg qr[2];
   cx qr[0],qr[2];
14 cx qr[0],qr[1];
   tdg qr[1];
16 t qr[0];
   cx qr[0],qr[1];
18 x qr[0];
   t qr[2];
20 h qr[2];
   h qr[2];
22 cx qr[1],qr[2];
   tdg qr[2];
24 cx qr[0],qr[2];
   t qr[2];
26 cx qr[1],qr[2];
   t qr[1];
28 tdg qr[2];
   cx qr[0],qr[2];
30 cx qr[0],qr[1];
   tdg qr[1];
32 t qr[0];
   cx qr[0],qr[1];
34 x qr[0];
   measure qr[0] -> cr[0];
36 measure qr[1] -> cr[1];
   t qr[2];
38 h qr[2];
   measure qr[2] -> cr[2];
```

Listing E.3: After running in local_qasm_simulator.

```
1  {'backend': 'ibmqx4',
    'lastUpdateDate': '2018-07-31T09:56:43.000Z',
3   'multiQubitGates': [{'gateError': {'date': '2018-07-31T09:56:43Z',
       'value': 0.030172186236362503},
```

```
 5       'name': 'CX1_0',
         'qubits': [1, 0],
 7       'type': 'CX'},
       {'gateError': {'date': '2018-07-31T09:56:43Z',
 9        'value': 0.022706065613666004},
         'name': 'CX2_0',
11       'qubits': [2, 0],
         'type': 'CX'},
13     {'gateError': {'date': '2018-07-31T09:56:43Z', 'value':
           0.04570350848648097},
         'name': 'CX2_1',
15       'qubits': [2, 1],
         'type': 'CX'},
17     {'gateError': {'date': '2018-07-31T09:56:43Z', 'value':
           0.07328729043000415},
         'name': 'CX3_2',
19       'qubits': [3, 2],
         'type': 'CX'},
21     {'gateError': {'date': '2018-07-31T09:56:43Z',
          'value': 0.041993419696321255},
23       'name': 'CX3_4',
         'qubits': [3, 4],
25       'type': 'CX'},
       {'gateError': {'date': '2018-07-31T09:56:43Z', 'value':
           0.05004433614140752},
27       'name': 'CX4_2',
         'qubits': [4, 2],
29       'type': 'CX'}],
     'qubits': [{'gateError': {'date': '2018-07-31T09:56:43Z',
31       'value': 0.0006867731322012238},
         'name': 'Q0',
33       'readoutError': {'date': '2018-07-31T09:56:43Z', 'value': 0.048}},
       {'gateError': {'date': '2018-07-31T09:56:43Z',
35        'value': 0.002232583111384412},
         'name': 'Q1',
37       'readoutError': {'date': '2018-07-31T09:56:43Z', 'value': 0.039}},
       {'gateError': {'date': '2018-07-31T09:56:43Z',
39        'value': 0.001545458810288558},
         'name': 'Q2',
41       'readoutError': {'date': '2018-07-31T09:56:43Z', 'value': 0.02}},
       {'gateError': {'date': '2018-07-31T09:56:43Z', 'value':
43        'value': 0.001631340796924452},
         'name': 'Q3',
45       'readoutError': {'date': '2018-07-31T09:56:43Z', 'value': 0.036}},
       {'gateError': {'date': '2018-07-31T09:56:43Z', 'value':
           0.00128782749692391},
47       'name': 'Q4',
```

```
      'readoutError': {'date': '2018-07-31T09:56:43Z', 'value': 0.113}}]]}
```

Listing E.4: Calibration information of backend `ibmqx4` when this experiment run.

```
   {'backend': 'ibmqx4',
2   'fridgeParameters': {'Temperature': {'date': '-', 'unit': '-', 'value': []},
    'cooldownDate': '2017-09-07'},
4   'lastUpdateDate': '2018-07-31T09:56:43.000Z',
    'qubits': [{'T1': {'date': '2018-07-31T09:56:43Z',
6      'unit': 'μs',
       'value': 50.4},
8     'T2': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 55.5},
      'buffer': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 10},
10    'frequency': {'date': '2018-07-31T09:56:43Z',
       'unit': 'GHz',
12     'value': 5.24985},
      'gateTime': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 60},
14    'name': 'Q0'},
    {'T1': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 52.4},
16    'T2': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 22.9},
      'buffer': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 10},
18    'frequency': {'date': '2018-07-31T09:56:43Z',
       'unit': 'GHz',
20     'value': 5.29575},
      'gateTime': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 60},
22    'name': 'Q1'},
    {'T1': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 33.1},
24    'T2': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 27.9},
      'buffer': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 10},
26    'frequency': {'date': '2018-07-31T09:56:43Z',
       'unit': 'GHz',
28     'value': 5.35322},
      'gateTime': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 60},
30    'name': 'Q2'},
    {'T1': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 47.9},
32    'T2': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 32.7},
      'buffer': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 10},
34    'frequency': {'date': '2018-07-31T09:56:43Z',
       'unit': 'GHz',
36     'value': 5.43498},
      'gateTime': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 60},
38    'name': 'Q3'},
    {'T1': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 57.1},
40    'T2': {'date': '2018-07-31T09:56:43Z', 'unit': 'μs', 'value': 13.8},
      'buffer': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 10},
42    'frequency': {'date': '2018-07-31T09:56:43Z',
       'unit': 'GHz',
```

```
44        'value': 5.17584},
        'gateTime': {'date': '2018-07-31T09:56:43Z', 'unit': 'ns', 'value': 60},
46        'name': 'Q4'}]}
```

Listing E.5: The parameters of the backend.

```
   Status @ 0 seconds
2  {'job_id': None, 'status': <JobStatus.INITIALIZING: 'job is being initialized'
       >, 'status_msg': 'Job is initializing. Please, wait a moment.'}
   Status @ 10 seconds
4  {'job_id': None, 'status': <JobStatus.INITIALIZING: 'job is being initialized'
       >, 'status_msg': 'Job is initializing. Please, wait a moment.'}
   Status @ 20 seconds
6  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 5}
   Status @ 30 seconds
8  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 5}
   Status @ 40 seconds
10 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 5}
   Status @ 50 seconds
12 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 4}
   Status @ 60 seconds
14 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 4}
   Status @ 70 seconds
16 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 4}
   Status @ 80 seconds
18 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 3}
   Status @ 90 seconds
20 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 3}
   Status @ 100 seconds
22 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 2}
   Status @ 110 seconds
24 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
   Status @ 120 seconds
26 {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
       queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
   Status @ 130 seconds
```

```
28  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 140 seconds
30  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 150 seconds
32  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 160 seconds
34  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 170 seconds
36  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 180 seconds
38  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 190 seconds
40  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 200 seconds
42  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 210 seconds
44  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 220 seconds
46  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 230 seconds
48  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 240 seconds
50  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 250 seconds
52  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.QUEUED: 'job is
        queued'>, 'status_msg': 'job is queued', 'queue_position': 1}
    Status @ 260 seconds
54  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.RUNNING: 'job is
        actively running'>, 'status_msg': 'job is actively running'}
    Status @ 270 seconds
56  {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.RUNNING: 'job is
        actively running'>, 'status_msg': 'job is actively running'}
    {'job_id': '5b60c0b0a98e23003a62274a', 'status': <JobStatus.DONE: 'job has
        successfully run'>, 'status_msg': 'job has successfully run'}
```

Listing E.6: The output of running the programs in the real device.

```
1  array([[ 1.-0.j,   0.+0.j,   0.+0.j,  ...,   0.+0.j,   0.+0.j,   0.+0.j],
          [ 0.+0.j,   1.-0.j,   0.+0.j,  ...,   0.+0.j,   0.+0.j,   0.+0.j],
3         [ 0.+0.j,   0.+0.j,   1.-0.j,  ...,   0.+0.j,   0.+0.j,   0.+0.j],
          ...,
5         [ 0.+0.j,   0.+0.j,   0.+0.j,  ...,  -0.+0.j,   0.+0.j,   0.+0.j],
          [ 0.+0.j,   0.+0.j,   0.+0.j,  ...,   0.+0.j,   0.+0.j,   0.+0.j],
7         [ 0.+0.j,   0.+0.j,   0.+0.j,  ...,   0.+0.j,   0.+0.j,   0.+0.j]])
```

Listing E.7: Output of 5.20 in the original and adapted circuits with 3 control qubits.

FOR-LOOP QUANTAMORPHISM OVER Y GATE

```
1  OPENQASM 2.0;
   include "qelib1.inc";
3  qreg qr[4];
   creg cr[4];
5  h qr[3];
   cx qr[3],qr[0];
7  cx qr[0],qr[1];
   u1(0.785398163397448) qr[0];
9  u1(-0.785398163397448) qr[1];
   cx qr[3],qr[0];
11 cx qr[0],qr[1];
   u1(-0.785398163397448) qr[3];
13 u1(0.785398163397448) qr[1];
   cx qr[3],qr[1];
15 h qr[3];
   u1(-0.785398163397448) qr[2];
17 u1(-0.785398163397448) qr[3];
   cx qr[2],qr[3];
19 u1(0.785398163397448) qr[3];
   cx qr[2],qr[3];
21 h qr[3];
   cx qr[3],qr[1];
23 u1(-0.785398163397448) qr[1];
   u1(0.785398163397448) qr[3];
25 cx qr[0],qr[1];
   cx qr[3],qr[0];
27 u1(0.785398163397448) qr[1];
   u1(-0.785398163397448) qr[0];
29 cx qr[0],qr[1];
   cx qr[3],qr[0];
31 h qr[3];
```

```
   barrier qr[3];
33 h qr[3];
   cx qr[3],qr[0];
35 cx qr[0],qr[2];
   u1(0.785398163397448) qr[0];
37 u1(0.785398163397448) qr[2];
   cx qr[3],qr[0];
39 cx qr[0],qr[2];
   u1(-0.785398163397448) qr[3];
41 u1(-0.785398163397448) qr[2];
   cx qr[3],qr[2];
43 h qr[3];
   cx qr[1],qr[3];
45 u1(-0.785398163397448) qr[3];
   cx qr[1],qr[3];
47 u1(0.785398163397448) qr[3];
   u1(0.785398163397448) qr[1];
49 h qr[3];
   cx qr[3],qr[2];
51 u1(0.785398163397448) qr[2];
   u1(0.785398163397448) qr[3];
53 cx qr[0],qr[2];
   cx qr[3],qr[0];
55 u1(-0.785398163397448) qr[2];
   u1(-0.785398163397448) qr[0];
57 cx qr[0],qr[2];
   cx qr[3],qr[0];
59 h qr[3];
   h qr[2];
61 u1(0.785398163397448) qr[2];
   u1(0.785398163397448) qr[0];
63 u1(0.785398163397448) qr[1];
   cx qr[0],qr[1];
65 cx qr[2],qr[0];
   cx qr[1],qr[2];
67 u1(-0.785398163397448) qr[0];
   u1(0.785398163397448) qr[2];
69 cx qr[1],qr[0];
   u1(-0.785398163397448) qr[0];
71 u1(-0.785398163397448) qr[1];
   cx qr[2],qr[0];
73 cx qr[1],qr[2];
   cx qr[0],qr[1];
75 h qr[2];
   barrier qr[3];
77 h qr[3];
   cx qr[3],qr[0];
```

```
79  cx qr[0],qr[1];
    u1(-0.785398163397448) qr[0];
81  u1(0.785398163397448) qr[1];
    cx qr[3],qr[0];
83  cx qr[0],qr[1];
    u1(-0.785398163397448) qr[3];
85  u1(0.785398163397448) qr[1];
    cx qr[3],qr[1];
87  h qr[3];
    u1(-0.785398163397448) qr[2];
89  u1(-0.785398163397448) qr[3];
    cx qr[2],qr[3];
91  u1(0.785398163397448) qr[3];
    cx qr[2],qr[3];
93  h qr[3];
    cx qr[3],qr[1];
95  u1(-0.785398163397448) qr[1];
    u1(0.785398163397448) qr[3];
97  cx qr[0],qr[1];
    cx qr[3],qr[0];
99  u1(-0.785398163397448) qr[1];
    u1(0.785398163397448) qr[0];
101 cx qr[0],qr[1];
    cx qr[3],qr[0];
103 h qr[3];
    barrier qr[3];
105 h qr[3];
    cx qr[3],qr[0];
107 cx qr[0],qr[2];
    u1(-0.785398163397448) qr[0];
109 u1(-0.785398163397448) qr[2];
    cx qr[3],qr[0];
111 cx qr[0],qr[2];
    u1(-0.785398163397448) qr[3];
113 u1(-0.785398163397448) qr[2];
    cx qr[3],qr[2];
115 h qr[3];
    cx qr[1],qr[3];
117 u1(-0.785398163397448) qr[3];
    cx qr[1],qr[3];
119 u1(0.785398163397448) qr[3];
    u1(0.785398163397448) qr[1];
121 h qr[3];
    cx qr[3],qr[2];
123 u1(0.785398163397448) qr[2];
    u1(0.785398163397448) qr[3];
125 cx qr[0],qr[2];
```

```
    cx  qr[3],qr[0];
127 u1(0.785398163397448) qr[2];
    u1(0.785398163397448) qr[0];
129 cx  qr[0],qr[2];
    cx  qr[3],qr[0];
131 h  qr[3];
    h  qr[2];
133 u1(0.785398163397448) qr[2];
    u1(-0.785398163397448) qr[0];
135 u1(0.785398163397448) qr[1];
    cx  qr[0],qr[1];
137 cx  qr[2],qr[0];
    cx  qr[1],qr[2];
139 u1(0.785398163397448) qr[0];
    u1(-0.785398163397448) qr[2];
141 cx  qr[1],qr[0];
    u1(-0.785398163397448) qr[0];
143 u1(0.785398163397448) qr[1];
    cx  qr[2],qr[0];
145 cx  qr[1],qr[2];
    cx  qr[0],qr[1];
147 h  qr[2];
    measure qr[0] -> cr[0];
149 measure qr[1] -> cr[1];
    measure qr[2] -> cr[2];
```

Listing E.8: QASM of original circuit for-loop quantamorphism over Y gate after the simulation.

```
    OPENQASM 2.0;
 2  include "qelib1.inc";
    qreg q[4];
 4  creg cr[4];
    u2(0,3.14159265358979) q[3];
 6  u1(-0.785398163397448) q[2];
    cx q[3],q[2];
 8  u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[3];
10  cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
12  u2(0,3.14159265358979) q[3];
    cx q[3],q[2];
14  cx q[2],q[0];
    u2(0,3.14159265358979) q[1];
16  u2(0,3.14159265358979) q[0];
    cx q[1],q[0];
18  u2(0.785398163397448,3.14159265358979) q[0];
    cx q[2],q[0];
```

```
20  u1(-0.785398163397448) q[2];
    u2(0,3.14159265358979) q[0];
22  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
    cx q[1],q[0];
24  u2(0.785398163397448,3.14159265358979) q[1];
    cx q[2],q[1];
26  u2(-0.785398163397448,3.14159265358979) q[2];
    cx q[3],q[2];
28  u1(0.785398163397448) q[2];
    cx q[3],q[2];
30  u2(0,3.14159265358979) q[2];
    cx q[2],q[1];
32  u1(0.785398163397448) q[2];
    u2(0,2.35619449019234) q[1];
34  cx q[1],q[0];
    u2(0,3.14159265358979) q[3];
36  u2(0,3.14159265358979) q[0];
    cx q[2],q[0];
38  u2(0,2.35619449019234) q[0];
    u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
40  cx q[1],q[0];
    u2(0,3.14159265358979) q[0];
42  cx q[2],q[0];
    u2(0,3.14159265358979) q[2];
44  barrier q[2];
    u2(0,3.14159265358979) q[0];
46  cx q[2],q[0];
    u2(0,3.14159265358979) q[0];
48  u2(0,3.14159265358979) q[2];
    cx q[2],q[0];
50  u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[2];
52  cx q[3],q[2];
    u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
54  cx q[2],q[0];
    u2(-0.785398163397448,3.14159265358979) q[0];
56  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
    cx q[3],q[2];
58  u2(0,3.14159265358979) q[2];
    u2(-0.785398163397448,3.14159265358979) q[3];
60  cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
62  u2(0,3.14159265358979) q[3];
    cx q[3],q[2];
64  u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[3];
66  cx q[3],q[2];
```

```
    u2(0 ,3.14159265358979) q[3];
68  u2(0 ,3.14159265358979) q[2];
    u2(0 ,3.14159265358979) q[1];
70  cx q[1],q[0];
    u2(0 ,3.14159265358979) q[0];
72  u2(0 ,3.14159265358979) q[1];
    cx q[1],q[0];
74  u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[1];
76  cx q[1],q[0];
    u2(0 ,3.14159265358979) q[0];
78  u2(0 ,3.14159265358979) q[1];
    cx q[2],q[1];
80  u2(0 ,3.14159265358979) q[1];
    cx q[1],q[0];
82  u3(0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[1];
    cx q[1],q[0];
84  u2(1.57079632679490 ,3.14159265358979) q[0];
    u3(1.57079632679490 ,0.785398163397448 ,3.14159265358979) q[1];
86  cx q[2],q[1];
    u3(-0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[1];
88  u3(-0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[2];
    cx q[3],q[2];
90  u2(0 ,3.14159265358979) q[2];
    u2(0 ,3.14159265358979) q[3];
92  cx q[3],q[2];
    u2(0 ,2.35619449019234) q[3];
94  u2(0 ,3.14159265358979) q[2];
    cx q[2],q[1];
96  u3(0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[2];
    cx q[3],q[2];
98  cx q[2],q[1];
    barrier q[1];
100 u2(0 ,3.14159265358979) q[1];
    u2(0.785398163397448 ,3.14159265358979) q[2];
102 cx q[2],q[0];
    u1(0.785398163397448) q[3];
104 cx q[3],q[2];
    cx q[2],q[0];
106 u2(0 ,3.14159265358979) q[3];
    u2(0 ,3.14159265358979) q[0];
108 u2(0 ,3.14159265358979) q[2];
    cx q[2],q[0];
110 u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[2];
112 cx q[2],q[0];
    u1(-0.785398163397448) q[0];
```

```
114  u2(0 ,3.14159265358979) q[2];
     cx  q[3] ,q[2];
116  u2(0 ,3.14159265358979) q[2];
     cx  q[2] ,q[0];
118  u1(-0.785398163397448) q[2];
     u1(-0.785398163397448) q[0];
120  cx  q[2] ,q[0];
     u3(-0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[3];
122  u2(0 ,3.14159265358979) q[0];
     u2(0 ,3.14159265358979) q[2];
124  cx  q[2] ,q[0];
     u2(0 ,3.14159265358979) q[0];
126  u2(0 ,3.14159265358979) q[2];
     cx  q[2] ,q[0];
128  cx  q[1] ,q[0];
     u2(0 ,3.14159265358979) q[2];
130  cx  q[3] ,q[2];
     u2(0 ,3.14159265358979) q[2];
132  u2(0 ,3.14159265358979) q[3];
     cx  q[3] ,q[2];
134  u2(0 ,3.14159265358979) q[3];
     u2(0 ,3.14159265358979) q[2];
136  u2(0 ,3.14159265358979) q[0];
     u2(0 ,3.14159265358979) q[1];
138  cx  q[1] ,q[0];
     u2(0 ,3.14159265358979) q[0];
140  u2(0 ,3.14159265358979) q[1];
     cx  q[1] ,q[0];
142  u2(0 ,3.14159265358979) q[0];
     u2(0 ,3.14159265358979) q[1];
144  cx  q[2] ,q[1];
     u2(0 ,3.14159265358979) q[1];
146  u2(0 ,3.14159265358979) q[2];
     cx  q[2] ,q[1];
148  u2(0 ,3.14159265358979) q[1];
     u2(0 ,3.14159265358979) q[2];
150  cx  q[2] ,q[1];
     u2(0 ,3.14159265358979) q[1];
152  u2(0 ,3.14159265358979) q[2];
     cx  q[2] ,q[1];
154  u2(-0.785398163397448 ,3.14159265358979) q[1];
     u2(0 ,3.14159265358979) q[2];
156  cx  q[3] ,q[2];
     u2(0 ,3.14159265358979) q[2];
158  u2(0 ,3.14159265358979) q[3];
     cx  q[3] ,q[2];
160  u2(0 ,3.14159265358979) q[3];
```

```
    u2(0,3.14159265358979) q[2];
162 cx q[2],q[0];
    cx q[3],q[2];
164 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
    cx q[2],q[0];
166 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[0];
    u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
168 cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
170 u2(0.785398163397448,3.14159265358979) q[3];
    cx q[3],q[2];
172 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[3];
174 cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
176 u2(0,3.14159265358979) q[3];
    cx q[3],q[2];
178 u2(0,3.14159265358979) q[2];
    cx q[2],q[0];
180 u1(-0.785398163397448) q[0];
    cx q[1],q[0];
182 u1(0.785398163397448) q[0];
    cx q[1],q[0];
184 cx q[2],q[0];
    u2(0.785398163397448,3.14159265358979) q[0];
186 u2(-0.785398163397448,3.14159265358979) q[2];
    cx q[3],q[2];
188 cx q[2],q[0];
    u2(0,3.14159265358979) q[3];
190 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[2];
192 cx q[2],q[0];
    u2(0,3.14159265358979) q[0];
194 u2(0,3.14159265358979) q[2];
    cx q[2],q[0];
196 u1(-0.785398163397448) q[0];
    u2(0,3.14159265358979) q[2];
198 cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
200 u2(0.785398163397448,3.14159265358979) q[3];
    cx q[3],q[2];
202 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[3];
204 cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
206 u2(0,3.14159265358979) q[3];
    cx q[3],q[2];
```

```
208  cx q[2],q[0];
     cx q[3],q[2];
210  u2(0,3.14159265358979) q[3];
     barrier q[3];
212  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
214  cx q[2],q[1];
     u1(-0.785398163397448) q[2];
216  u1(-0.785398163397448) q[1];
     cx q[3],q[2];
218  u1(-0.785398163397448) q[3];
     cx q[2],q[1];
220  u1(-0.785398163397448) q[1];
     cx q[2],q[1];
222  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[2];
224  cx q[2],q[1];
     u2(0,3.14159265358979) q[1];
226  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
228  cx q[3],q[2];
     u2(0,3.14159265358979) q[3];
230  cx q[1],q[0];
     cx q[3],q[2];
232  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
234  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
236  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
238  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[2];
240  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
242  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
244  cx q[3],q[2];
     u2(0,3.14159265358979) q[3];
246  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
248  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[2],q[1];
250  u2(1.57079632679490,3.14159265358979) q[1];
     u3(1.57079632679490,0.785398163397448,3.14159265358979) q[2];
252  cx q[3],q[2];
     u2(0.785398163397448,3.14159265358979) q[2];
254  cx q[2],q[0];
```

```
      u2(0,3.14159265358979) q[0];
256   u2(0,3.14159265358979) q[2];
      cx q[2],q[0];
258   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[2];
260   cx q[2],q[0];
      u2(0,3.14159265358979) q[0];
262   u2(0,3.14159265358979) q[2];
      u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
264   cx q[3],q[2];
      cx q[2],q[0];
266   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
      u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
268   cx q[3],q[2];
      cx q[2],q[0];
270   u2(-0.785398163397448,3.14159265358979) q[2];
      cx q[2],q[1];
272   u1(0.785398163397448) q[3];
      cx q[3],q[2];
274   cx q[2],q[1];
      u2(0,3.14159265358979) q[3];
276   u2(0,3.14159265358979) q[1];
      u2(0,3.14159265358979) q[2];
278   cx q[2],q[1];
      u2(0,3.14159265358979) q[1];
280   u2(0,3.14159265358979) q[2];
      cx q[2],q[1];
282   u1(0.785398163397448) q[1];
      u2(0,3.14159265358979) q[2];
284   cx q[3],q[2];
      u2(0,3.14159265358979) q[2];
286   cx q[2],q[1];
      u1(0.785398163397448) q[2];
288   u1(-0.785398163397448) q[1];
      u2(-0.785398163397448,3.14159265358979) q[3];
290   cx q[3],q[2];
      u2(0,3.14159265358979) q[2];
292   u2(0,3.14159265358979) q[3];
      cx q[3],q[2];
294   u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[3];
296   cx q[3],q[2];
      cx q[2],q[1];
298   u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
300   cx q[3],q[2];
      u2(0,3.14159265358979) q[2];
```

```
302  u2(0 ,3.14159265358979) q[3];
     cx q[3],q[2];
304  u2(0 ,3.14159265358979) q[3];
     measure q[3] -> cr[2];
306  u2(0 ,3.14159265358979) q[2];
     u2(0 ,3.14159265358979) q[1];
308  cx q[2],q[1];
     u2(0 ,3.14159265358979) q[1];
310  measure q[1] -> cr[0];
     u2(0 ,3.14159265358979) q[2];
312  measure q[2] -> cr[1];
```

Listing E.9: QASM of original circuit for-loop quantamorphism over Y gate after running in quantum device.

```
   OPENQASM 2.0;
2  include "qelib1.inc";
   qreg q[4];
4  creg cr[4];
   u2(0 ,2.35619449019234) q[1];
6  u2(0 ,3.14159265358979) q[0];
   cx q[2],q[0];
8  u2(0 ,3.14159265358979) q[3];
   u2(0 ,3.14159265358979) q[0];
10 u2(0 ,3.14159265358979) q[2];
   cx q[2],q[0];
12 u2(0 ,3.14159265358979) q[0];
   u2(0 ,3.14159265358979) q[2];
14 cx q[2],q[0];
   u2(0 ,3.14159265358979) q[2];
16 cx q[3],q[2];
   u2(0 ,3.14159265358979) q[2];
18 u2(0 ,3.14159265358979) q[3];
   cx q[3],q[2];
20 u2(0 ,3.14159265358979) q[2];
   u2(0 ,3.14159265358979) q[3];
22 cx q[3],q[2];
   u2(0 ,3.14159265358979) q[2];
24 u2(0 ,3.14159265358979) q[3];
   cx q[3],q[2];
26 cx q[2],q[0];
   u1(0.785398163397448) q[2];
28 u1(-0.785398163397448) q[0];
   cx q[3],q[2];
30 u1(-0.785398163397448) q[3];
   cx q[2],q[0];
32 u1(0.785398163397448) q[0];
```

```
   cx q[3],q[2];
34 u2(0,3.14159265358979) q[2];
   u2(0,3.14159265358979) q[3];
36 cx q[3],q[2];
   u2(0,3.14159265358979) q[2];
38 u2(0,3.14159265358979) q[3];
   cx q[3],q[2];
40 cx q[2],q[0];
   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
42 cx q[2],q[1];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
44 cx q[2],q[1];
   cx q[2],q[0];
46 u1(0.785398163397448) q[2];
   u1(-0.785398163397448) q[0];
48 cx q[2],q[0];
   u2(0,3.14159265358979) q[1];
50 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[2];
52 cx q[2],q[0];
   u2(0,3.14159265358979) q[0];
54 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
56 cx q[3],q[2];
   cx q[2],q[0];
58 u2(0,3.14159265358979) q[3];
   u2(0,3.14159265358979) q[0];
60 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
62 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[2];
64 cx q[2],q[0];
   u1(0.785398163397448) q[0];
66 u2(0,3.14159265358979) q[2];
   cx q[3],q[2];
68 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
70 u2(-0.785398163397448,3.14159265358979) q[3];
   u2(0,3.14159265358979) q[0];
72 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
74 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[2];
76 cx q[2],q[0];
   cx q[3],q[2];
78 cx q[2],q[0];
   u2(0,3.14159265358979) q[3];
```

```
80  u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[2];
82  cx q[2],q[0];
    u2(0,3.14159265358979) q[0];
84  u2(0,3.14159265358979) q[2];
    cx q[2],q[0];
86  u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[2];
88  cx q[3],q[2];
    barrier q[2];
90  cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
92  cx q[2],q[1];
    u2(0,3.14159265358979) q[1];
94  u2(0,3.14159265358979) q[2];
    cx q[2],q[1];
96  u2(0,3.14159265358979) q[1];
    u2(0,3.14159265358979) q[2];
98  cx q[2],q[1];
    u2(0,3.14159265358979) q[3];
100 cx q[3],q[2];
    u2(0,3.92699081698724) q[3];
102 u1(0.785398163397448) q[2];
    cx q[2],q[1];
104 u2(0,3.14159265358979) q[1];
    u2(0,3.14159265358979) q[2];
106 cx q[2],q[1];
    u2(0,3.14159265358979) q[1];
108 u2(0,3.14159265358979) q[2];
    cx q[2],q[1];
110 u2(0,3.14159265358979) q[2];
    cx q[3],q[2];
112 u2(0,3.14159265358979) q[2];
    cx q[2],q[1];
114 u2(0,3.14159265358979) q[1];
    u2(0,3.14159265358979) q[2];
116 cx q[2],q[1];
    u2(0,3.14159265358979) q[1];
118 u2(0,3.14159265358979) q[2];
    cx q[2],q[1];
120 u2(0,2.35619449019234) q[1];
    u2(0,3.14159265358979) q[3];
122 cx q[3],q[2];
    u2(0,2.35619449019234) q[2];
124 cx q[2],q[1];
    u2(0,3.14159265358979) q[1];
126 cx q[1],q[0];
```

```
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
128   cx q[1],q[0];
      u2(1.57079632679490,3.14159265358979) q[0];
130   u3(1.57079632679490,0.785398163397448,3.14159265358979) q[1];
      cx q[2],q[1];
132   u2(0.785398163397448,3.14159265358979) q[1];
      u2(0.785398163397448,3.14159265358979) q[2];
134   cx q[3],q[2];
      cx q[2],q[1];
136   u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[1];
138   u2(0,3.14159265358979) q[2];
      cx q[2],q[1];
140   u2(0,3.14159265358979) q[1];
      u2(0,3.14159265358979) q[2];
142   cx q[2],q[1];
      u1(-0.785398163397448) q[1];
144   u2(0,3.14159265358979) q[2];
      cx q[3],q[2];
146   u2(0,3.14159265358979) q[2];
      cx q[2],q[1];
148   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
      u2(0,3.14159265358979) q[1];
150   u2(0,3.14159265358979) q[2];
      cx q[2],q[1];
152   u2(0,3.14159265358979) q[1];
      u2(0,3.14159265358979) q[2];
154   cx q[2],q[1];
      u2(0,3.14159265358979) q[2];
156   cx q[3],q[2];
      u2(0,3.14159265358979) q[2];
158   u2(0,3.14159265358979) q[3];
      cx q[3],q[2];
160   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
      u2(0,3.14159265358979) q[2];
162   u2(0,3.14159265358979) q[1];
      cx q[2],q[1];
164   barrier q[1];
      u2(0.785398163397448,3.14159265358979) q[2];
166   cx q[2],q[0];
      u2(0,3.14159265358979) q[2];
168   cx q[3],q[2];
      u2(0,3.14159265358979) q[2];
170   u2(0,3.14159265358979) q[3];
      cx q[3],q[2];
172   u1(-0.785398163397448) q[3];
      u2(0,3.14159265358979) q[2];
```

```
174  u2(0,3.14159265358979) q[0];
     cx q[2],q[0];
176  u2(0,3.14159265358979) q[2];
     cx q[3],q[2];
178  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
180  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
182  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
184  u2(0,3.92699081698724) q[3];
     u2(0,3.14159265358979) q[2];
186  cx q[2],q[0];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[0];
188  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[3],q[2];
190  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
192  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
194  cx q[2],q[0];
     u2(0,3.14159265358979) q[0];
196  u2(0,3.14159265358979) q[2];
     cx q[3],q[2];
198  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
200  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
202  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
204  u2(-0.785398163397448,3.14159265358979) q[3];
     cx q[2],q[0];
206  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
208  u2(0,3.14159265358979) q[2];
     cx q[2],q[0];
210  u2(0,2.35619449019234) q[2];
     u1(0.785398163397448) q[0];
212  cx q[2],q[1];
     u2(-0.785398163397448,3.14159265358979) q[1];
214  u2(0,3.14159265358979) q[2];
     cx q[2],q[0];
216  u1(0.785398163397448) q[0];
     cx q[1],q[0];
218  u2(-0.785398163397448,3.14159265358979) q[1];
     cx q[3],q[2];
220  u2(0,3.14159265358979) q[2];
```

```
     u2(0,3.14159265358979) q[3];
222  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
224  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
226  cx q[2],q[1];
     u1(0.785398163397448) q[1];
228  cx q[2],q[1];
     u2(0,3.14159265358979) q[1];
230  cx q[1],q[0];
     u2(0,3.92699081698724) q[1];
232  u1(-0.785398163397448) q[0];
     cx q[3],q[2];
234  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
236  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
238  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
240  cx q[2],q[0];
     u1(-0.785398163397448) q[0];
242  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
244  cx q[2],q[1];
     u2(0.785398163397448,3.14159265358979) q[2];
246  cx q[2],q[0];
     u2(0,3.14159265358979) q[0];
248  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
250  barrier q[1];
     cx q[2],q[1];
252  cx q[3],q[2];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
254  cx q[2],q[1];
     u2(-0.785398163397448,3.14159265358979) q[1];
256  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
     cx q[3],q[2];
258  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
260  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[2];
262  cx q[2],q[1];
     u2(0,3.14159265358979) q[1];
264  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
266  u2(0,3.14159265358979) q[2];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
```

```
268  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
270  cx q[2],q[0];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
272  cx q[2],q[0];
     u2(1.57079632679490,3.14159265358979) q[0];
274  u3(1.57079632679490,0.785398163397448,3.14159265358979) q[2];
     cx q[3],q[2];
276  u2(0.785398163397448,3.14159265358979) q[2];
     cx q[2],q[1];
278  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[2];
280  cx q[2],q[1];
     u2(0,3.14159265358979) q[1];
282  u2(0,3.14159265358979) q[2];
     cx q[2],q[1];
284  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[2];
286  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
     cx q[3],q[2];
288  cx q[2],q[1];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
290  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
     cx q[3],q[2];
292  cx q[2],q[1];
     u2(-0.785398163397448,3.14159265358979) q[2];
294  cx q[2],q[0];
     u2(0,3.14159265358979) q[2];
296  u2(0,3.14159265358979) q[0];
     u2(0,3.92699081698724) q[3];
298  cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
300  u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
302  u2(0,3.92699081698724) q[3];
     u2(0,3.14159265358979) q[2];
304  cx q[2],q[0];
     u2(0,3.14159265358979) q[0];
306  u2(0,3.14159265358979) q[2];
     cx q[2],q[0];
308  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[2];
310  cx q[2],q[0];
     u2(0,3.14159265358979) q[0];
312  u2(0,3.14159265358979) q[2];
     cx q[2],q[0];
314  u2(0,2.35619449019234) q[0];
```

```
    u2(0 ,3.14159265358979) q[2];
316 cx q[3],q[2];
    u2(0.785398163397448 ,3.14159265358979) q[2];
318 u2(-0.785398163397448 ,3.14159265358979) q[3];
    cx q[3],q[2];
320 u2(0 ,3.14159265358979) q[2];
    u2(0 ,3.14159265358979) q[3];
322 cx q[3],q[2];
    u2(0 ,3.14159265358979) q[2];
324 u2(0 ,3.14159265358979) q[3];
    cx q[3],q[2];
326 u2(0 ,3.14159265358979) q[2];
    cx q[2],q[0];
328 u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[2];
330 cx q[3],q[2];
    u2(0 ,3.14159265358979) q[2];
332 u2(0 ,3.14159265358979) q[3];
    cx q[3],q[2];
334 u2(0 ,3.14159265358979) q[2];
    u2(0 ,3.14159265358979) q[3];
336 cx q[3],q[2];
    cx q[2],q[0];
338 cx q[3],q[2];
    u2(0 ,3.14159265358979) q[0];
340 measure q[3] -> cr[3];
    measure q[2] -> cr[2];
342 measure q[0] -> cr[1];
```

Listing E.10: QASM of adapted circuit for-loop quantamorphism over Y gate after running in quantum device.

FOR-LOOP QUANTAMORPHISM OVER HADAMARD GATE

```
    OPENQASM 2.0;
 2 include "qelib1.inc";
    qreg qr[4];
 4 creg cr[4];
    h qr[3];
 6 cx qr[3],qr[0];
    cx qr[0],qr[1];
 8 u1(0.785398163397448) qr[0];
    u1(-0.785398163397448) qr[1];
10 cx qr[3],qr[0];
    cx qr[0],qr[1];
12 u1(-0.785398163397448) qr[3];
    u1(0.785398163397448) qr[1];
```

```
14  cx qr[3],qr[1];
    h qr[3];
16  u1(-1.57079632679490) qr[2];
    h qr[2];
18  u1(-0.785398163397448) qr[2];
    cx qr[3],qr[2];
20  u1(0.785398163397448) qr[2];
    h qr[2];
22  u1(1.57079632679490) qr[2];
    h qr[3];
24  cx qr[3],qr[1];
    u1(-0.785398163397448) qr[1];
26  u1(0.785398163397448) qr[3];
    cx qr[0],qr[1];
28  cx qr[3],qr[0];
    u1(0.785398163397448) qr[1];
30  u1(-0.785398163397448) qr[0];
    cx qr[0],qr[1];
32  cx qr[3],qr[0];
    h qr[3];
34  barrier qr[3];
    h qr[3];
36  cx qr[3],qr[0];
    cx qr[0],qr[1];
38  u1(-0.785398163397448) qr[0];
    u1(0.785398163397448) qr[1];
40  cx qr[3],qr[0];
    cx qr[0],qr[1];
42  u1(-0.785398163397448) qr[3];
    u1(0.785398163397448) qr[1];
44  cx qr[3],qr[1];
    h qr[3];
46  u1(-1.57079632679490) qr[2];
    h qr[2];
48  u1(-0.785398163397448) qr[2];
    cx qr[3],qr[2];
50  u1(0.785398163397448) qr[2];
    h qr[2];
52  u1(1.57079632679490) qr[2];
    h qr[3];
54  cx qr[3],qr[1];
    u1(-0.785398163397448) qr[1];
56  u1(0.785398163397448) qr[3];
    cx qr[0],qr[1];
58  cx qr[3],qr[0];
    u1(-0.785398163397448) qr[1];
60  u1(0.785398163397448) qr[0];
```

```
   cx qr[0],qr[1];
62 cx qr[3],qr[0];
   h qr[3];
64 measure qr[0] -> cr[0];
   measure qr[1] -> cr[1];
66 measure qr[2] -> cr[2];
```

Listing E.11: QASM of original circuit for-loop quantamorphism over Hadamard gate after the simulation.

```
   OPENQASM 2.0;
2  include "qelib1.inc";
   qreg q[4];
4  creg cr[4];
   u2(0,3.14159265358979) q[3];
6  u1(-1.57079632679490) q[2];
   cx q[3],q[2];
8  cx q[1],q[0];
   u2(0,3.14159265358979) q[2];
10 u2(0,3.14159265358979) q[3];
   cx q[3],q[2];
12 u2(0,3.14159265358979) q[2];
   u2(0,3.14159265358979) q[3];
14 cx q[3],q[2];
   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
16 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[1];
18 cx q[1],q[0];
   u2(0,3.14159265358979) q[0];
20 u2(0,3.14159265358979) q[1];
   cx q[1],q[0];
22 cx q[2],q[1];
   cx q[1],q[0];
24 u1(0.785398163397448) q[1];
   u1(-0.785398163397448) q[0];
26 cx q[2],q[1];
   u1(-0.785398163397448) q[2];
28 cx q[1],q[0];
   u1(0.785398163397448) q[0];
30 cx q[2],q[0];
   cx q[3],q[2];
32 cx q[2],q[0];
   u1(0.785398163397448) q[2];
34 u1(-0.785398163397448) q[0];
   cx q[1],q[0];
36 cx q[2],q[1];
   u1(0.785398163397448) q[0];
```

```
38  u1(-0.785398163397448) q[1];
    cx q[1],q[0];
40  cx q[2],q[1];
    u2(0,3.14159265358979) q[2];
42  barrier q[2];
    u2(0,3.14159265358979) q[2];
44  cx q[2],q[1];
    cx q[1],q[0];
46  u1(-0.785398163397448) q[1];
    u1(0.785398163397448) q[0];
48  cx q[2],q[1];
    u1(-0.785398163397448) q[2];
50  cx q[1],q[0];
    u1(0.785398163397448) q[0];
52  cx q[2],q[0];
    u1(6.28318530717959) q[3];
54  cx q[3],q[2];
    cx q[2],q[0];
56  u1(0.785398163397448) q[2];
    u1(-0.785398163397448) q[0];
58  cx q[1],q[0];
    cx q[2],q[1];
60  u1(-0.785398163397448) q[0];
    u1(0.785398163397448) q[1];
62  cx q[1],q[0];
    cx q[2],q[1];
64  measure q[0] -> cr[1];
    u2(0,3.14159265358979) q[2];
66  measure q[1] -> cr[0];
    u3(-0.785398163397448,3.14159265358979,4.71238898038469) q[3];
68  measure q[3] -> cr[2];
```

Listing E.12: QASM of original circuit for-loop quantamorphism over Hadamard gate after running in quantum device.

```
    OPENQASM 2.0;
2   include "qelib1.inc";
    qreg q[4];
4   creg cr[4];
    u2(-0.785398163397448,1.57079632679490) q[1];
6   cx q[3],q[2];
    u2(0,3.14159265358979) q[2];
8   u2(0,3.14159265358979) q[3];
    cx q[3],q[2];
10  u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[3];
12  cx q[3],q[2];
```

```
   u2(0,3.14159265358979) q[3];
14 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
16 cx q[3],q[2];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
18 cx q[2],q[0];
   u2(-0.785398163397448,3.14159265358979) q[0];
20 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
   cx q[3],q[2];
22 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
24 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[2];
26 cx q[2],q[0];
   u2(0,3.14159265358979) q[0];
28 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
30 u2(0,3.14159265358979) q[2];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
32 cx q[3],q[2];
   cx q[2],q[1];
34 u2(0,3.14159265358979) q[1];
   cx q[3],q[2];
36 u2(0.785398163397448,3.14159265358979) q[2];
   cx q[2],q[0];
38 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[2];
40 cx q[2],q[0];
   u2(0,3.14159265358979) q[0];
42 u2(0,3.14159265358979) q[2];
   cx q[2],q[0];
44 u2(0,3.14159265358979) q[0];
   u2(0,3.14159265358979) q[2];
46 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
   cx q[3],q[2];
48 cx q[2],q[0];
   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
50 u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
   cx q[3],q[2];
52 cx q[2],q[0];
   barrier q[0];
54 cx q[2],q[0];
   cx q[3],q[2];
56 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
   cx q[2],q[0];
58 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[0];
   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
```

```
60   cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
62   u2(0.785398163397448,3.14159265358979) q[3];
     cx q[3],q[2];
64   u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
66   cx q[3],q[2];
     u2(0,3.14159265358979) q[2];
68   u2(0,3.14159265358979) q[3];
     cx q[3],q[2];
70   u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
72   cx q[2],q[0];
     u2(0,3.14159265358979) q[0];
74   cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
76   cx q[2],q[0];
     u3(-0.785398163397448,3.14159265358979,4.71238898038469) q[1];
78   measure q[1] -> cr[1];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[0];
80   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[3],q[2];
82   u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[3];
84   cx q[3],q[2];
     u2(0,2.35619449019234) q[3];
86   u2(0,3.14159265358979) q[2];
     cx q[2],q[0];
88   u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[3],q[2];
90   cx q[2],q[0];
     u2(0,3.14159265358979) q[2];
92   measure q[2] -> cr[3];
     u2(0,3.14159265358979) q[3];
94   measure q[3] -> cr[2];
```

Listing E.13: QASM of adapted circuit for-loop quantamorphism over Hadamard gate after running in quantum device.

FOLD QUANTAMORPHISM OVER XOR GATE

```
     OPENQASM 2.0;
2    include "qelib1.inc";
     qreg qr[7];
4    creg cr[7];
     h qr[5];
6    cx qr[5],qr[0];
```

```
   cx qr[0],qr[1];
 8 u1(0.785398163397448) qr[0];
   u1(-0.785398163397448) qr[1];
10 cx qr[5],qr[0];
   cx qr[0],qr[1];
12 u1(-0.785398163397448) qr[5];
   u1(0.785398163397448) qr[1];
14 cx qr[5],qr[1];
   h qr[5];
16 h qr[6];
   cx qr[6],qr[2];
18 cx qr[2],qr[3];
   u1(0.785398163397448) qr[2];
20 u1(-0.785398163397448) qr[3];
   cx qr[6],qr[2];
22 cx qr[2],qr[3];
   u1(-0.785398163397448) qr[6];
24 u1(0.785398163397448) qr[3];
   cx qr[6],qr[3];
26 h qr[6];
   h qr[4];
28 u1(0.785398163397448) qr[4];
   u1(0.785398163397448) qr[5];
30 u1(0.785398163397448) qr[6];
   cx qr[5],qr[6];
32 cx qr[4],qr[5];
   cx qr[6],qr[4];
34 u1(-0.785398163397448) qr[5];
   u1(0.785398163397448) qr[4];
36 cx qr[6],qr[5];
   u1(-0.785398163397448) qr[5];
38 u1(-0.785398163397448) qr[6];
   cx qr[4],qr[5];
40 cx qr[6],qr[4];
   cx qr[5],qr[6];
42 h qr[4];
   h qr[6];
44 cx qr[6],qr[3];
   u1(-0.785398163397448) qr[3];
46 u1(0.785398163397448) qr[6];
   cx qr[2],qr[3];
48 cx qr[6],qr[2];
   u1(0.785398163397448) qr[3];
50 u1(-0.785398163397448) qr[2];
   cx qr[2],qr[3];
52 cx qr[6],qr[2];
   h qr[6];
```

```
54  h qr[5];
    cx qr[5],qr[1];
56  u1(-0.785398163397448) qr[1];
    u1(0.785398163397448) qr[5];
58  cx qr[0],qr[1];
    cx qr[5],qr[0];
60  u1(0.785398163397448) qr[1];
    u1(-0.785398163397448) qr[0];
62  cx qr[0],qr[1];
    cx qr[5],qr[0];
64  h qr[5];
    barrier qr[5];
66  h qr[5];
    cx qr[5],qr[0];
68  cx qr[0],qr[1];
    u1(0.785398163397448) qr[0];
70  u1(-0.785398163397448) qr[1];
    cx qr[5],qr[0];
72  cx qr[0],qr[1];
    u1(-0.785398163397448) qr[5];
74  u1(0.785398163397448) qr[1];
    cx qr[5],qr[1];
76  h qr[5];
    barrier qr[6];
78  h qr[6];
    cx qr[6],qr[2];
80  cx qr[2],qr[3];
    u1(0.785398163397448) qr[2];
82  u1(0.785398163397448) qr[3];
    cx qr[6],qr[2];
84  cx qr[2],qr[3];
    u1(-0.785398163397448) qr[6];
86  u1(-0.785398163397448) qr[3];
    cx qr[6],qr[3];
88  h qr[6];
    h qr[4];
90  u1(0.785398163397448) qr[4];
    u1(0.785398163397448) qr[5];
92  u1(0.785398163397448) qr[6];
    cx qr[5],qr[6];
94  cx qr[4],qr[5];
    cx qr[6],qr[4];
96  u1(-0.785398163397448) qr[5];
    u1(0.785398163397448) qr[4];
98  cx qr[6],qr[5];
    u1(-0.785398163397448) qr[5];
100 u1(-0.785398163397448) qr[6];
```

```
    cx qr[4],qr[5];
102 cx qr[6],qr[4];
    cx qr[5],qr[6];
104 h qr[4];
    h qr[6];
106 cx qr[6],qr[3];
    u1(0.785398163397448) qr[3];
108 u1(0.785398163397448) qr[6];
    cx qr[2],qr[3];
110 cx qr[6],qr[2];
    u1(-0.785398163397448) qr[3];
112 u1(-0.785398163397448) qr[2];
    cx qr[2],qr[3];
114 cx qr[6],qr[2];
    h qr[6];
116 h qr[5];
    cx qr[5],qr[1];
118 u1(-0.785398163397448) qr[1];
    u1(0.785398163397448) qr[5];
120 cx qr[0],qr[1];
    cx qr[5],qr[0];
122 u1(0.785398163397448) qr[1];
    u1(-0.785398163397448) qr[0];
124 cx qr[0],qr[1];
    cx qr[5],qr[0];
126 h qr[5];
    barrier qr[5];
128 h qr[5];
    cx qr[5],qr[0];
130 cx qr[0],qr[1];
    u1(0.785398163397448) qr[0];
132 u1(0.785398163397448) qr[1];
    cx qr[5],qr[0];
134 cx qr[0],qr[1];
    u1(-0.785398163397448) qr[5];
136 u1(-0.785398163397448) qr[1];
    cx qr[5],qr[1];
138 h qr[5];
    barrier qr[6];
140 h qr[6];
    cx qr[6],qr[2];
142 cx qr[2],qr[3];
    u1(0.785398163397448) qr[2];
144 u1(0.785398163397448) qr[3];
    cx qr[6],qr[2];
146 cx qr[2],qr[3];
    u1(-0.785398163397448) qr[6];
```

```
148  u1(-0.785398163397448) qr[3];
     cx qr[6],qr[3];
150  h qr[6];
     h qr[4];
152  u1(0.785398163397448) qr[4];
     u1(0.785398163397448) qr[5];
154  u1(0.785398163397448) qr[6];
     cx qr[5],qr[6];
156  cx qr[4],qr[5];
     cx qr[6],qr[4];
158  u1(-0.785398163397448) qr[5];
     u1(0.785398163397448) qr[4];
160  cx qr[6],qr[5];
     u1(-0.785398163397448) qr[5];
162  u1(-0.785398163397448) qr[6];
     cx qr[4],qr[5];
164  cx qr[6],qr[4];
     cx qr[5],qr[6];
166  h qr[4];
     h qr[6];
168  cx qr[6],qr[3];
     u1(0.785398163397448) qr[3];
170  u1(0.785398163397448) qr[6];
     cx qr[2],qr[3];
172  cx qr[6],qr[2];
     u1(-0.785398163397448) qr[3];
174  u1(-0.785398163397448) qr[2];
     cx qr[2],qr[3];
176  cx qr[6],qr[2];
     h qr[6];
178  h qr[5];
     cx qr[5],qr[1];
180  u1(0.785398163397448) qr[1];
     u1(0.785398163397448) qr[5];
182  cx qr[0],qr[1];
     cx qr[5],qr[0];
184  u1(-0.785398163397448) qr[1];
     u1(-0.785398163397448) qr[0];
186  cx qr[0],qr[1];
     cx qr[5],qr[0];
188  h qr[5];
     barrier qr[5];
190  h qr[5];
     cx qr[5],qr[0];
192  cx qr[0],qr[1];
     u1(0.785398163397448) qr[0];
194  u1(0.785398163397448) qr[1];
```

```
    cx qr[5],qr[0];
196 cx qr[0],qr[1];
    u1(-0.785398163397448) qr[5];
198 u1(-0.785398163397448) qr[1];
    cx qr[5],qr[1];
200 h qr[5];
    barrier qr[6];
202 h qr[6];
    cx qr[6],qr[2];
204 cx qr[2],qr[3];
    u1(-0.785398163397448) qr[2];
206 u1(0.785398163397448) qr[3];
    cx qr[6],qr[2];
208 cx qr[2],qr[3];
    u1(-0.785398163397448) qr[6];
210 u1(0.785398163397448) qr[3];
    cx qr[6],qr[3];
212 h qr[6];
    h qr[4];
214 u1(0.785398163397448) qr[4];
    u1(0.785398163397448) qr[5];
216 u1(0.785398163397448) qr[6];
    cx qr[5],qr[6];
218 cx qr[4],qr[5];
    cx qr[6],qr[4];
220 u1(-0.785398163397448) qr[5];
    u1(0.785398163397448) qr[4];
222 cx qr[6],qr[5];
    u1(-0.785398163397448) qr[5];
224 u1(-0.785398163397448) qr[6];
    cx qr[4],qr[5];
226 cx qr[6],qr[4];
    cx qr[5],qr[6];
228 h qr[4];
    h qr[6];
230 cx qr[6],qr[3];
    u1(-0.785398163397448) qr[3];
232 u1(0.785398163397448) qr[6];
    cx qr[2],qr[3];
234 cx qr[6],qr[2];
    u1(-0.785398163397448) qr[3];
236 u1(0.785398163397448) qr[2];
    cx qr[2],qr[3];
238 cx qr[6],qr[2];
    h qr[6];
240 h qr[5];
    cx qr[5],qr[1];
```

```
242  u1(0.785398163397448) qr[1];
     u1(0.785398163397448) qr[5];
244  cx qr[0],qr[1];
     cx qr[5],qr[0];
246  u1(-0.785398163397448) qr[1];
     u1(-0.785398163397448) qr[0];
248  cx qr[0],qr[1];
     cx qr[5],qr[0];
250  h qr[5];
     barrier qr[5];
252  h qr[5];
     cx qr[5],qr[0];
254  cx qr[0],qr[1];
     u1(0.785398163397448) qr[0];
256  u1(0.785398163397448) qr[1];
     cx qr[5],qr[0];
258  cx qr[0],qr[1];
     u1(-0.785398163397448) qr[5];
260  u1(-0.785398163397448) qr[1];
     cx qr[5],qr[1];
262  h qr[5];
     barrier qr[6];
264  h qr[6];
     cx qr[6],qr[2];
266  cx qr[2],qr[3];
     u1(-0.785398163397448) qr[2];
268  u1(-0.785398163397448) qr[3];
     cx qr[6],qr[2];
270  cx qr[2],qr[3];
     u1(-0.785398163397448) qr[6];
272  u1(-0.785398163397448) qr[3];
     cx qr[6],qr[3];
274  h qr[6];
     h qr[4];
276  u1(0.785398163397448) qr[4];
     u1(0.785398163397448) qr[5];
278  u1(0.785398163397448) qr[6];
     cx qr[5],qr[6];
280  cx qr[4],qr[5];
     cx qr[6],qr[4];
282  u1(-0.785398163397448) qr[5];
     u1(0.785398163397448) qr[4];
284  cx qr[6],qr[5];
     u1(-0.785398163397448) qr[5];
286  u1(-0.785398163397448) qr[6];
     cx qr[4],qr[5];
288  cx qr[6],qr[4];
```

```
    cx qr[5],qr[6];
290 h qr[4];
    h qr[6];
292 cx qr[6],qr[3];
    u1(0.785398163397448) qr[3];
294 u1(0.785398163397448) qr[6];
    cx qr[2],qr[3];
296 cx qr[6],qr[2];
    u1(0.785398163397448) qr[3];
298 u1(0.785398163397448) qr[2];
    cx qr[2],qr[3];
300 cx qr[6],qr[2];
    h qr[6];
302 h qr[5];
    cx qr[5],qr[1];
304 u1(0.785398163397448) qr[1];
    u1(0.785398163397448) qr[5];
306 cx qr[0],qr[1];
    cx qr[5],qr[0];
308 u1(-0.785398163397448) qr[1];
    u1(-0.785398163397448) qr[0];
310 cx qr[0],qr[1];
    cx qr[5],qr[0];
312 h qr[5];
    barrier qr[5];
314 h qr[5];
    cx qr[5],qr[0];
316 cx qr[0],qr[1];
    u1(-0.785398163397448) qr[0];
318 u1(0.785398163397448) qr[1];
    cx qr[5],qr[0];
320 cx qr[0],qr[1];
    u1(-0.785398163397448) qr[5];
322 u1(0.785398163397448) qr[1];
    cx qr[5],qr[1];
324 h qr[5];
    barrier qr[6];
326 h qr[6];
    cx qr[6],qr[2];
328 cx qr[2],qr[3];
    u1(0.785398163397448) qr[2];
330 u1(0.785398163397448) qr[3];
    cx qr[6],qr[2];
332 cx qr[2],qr[3];
    u1(-0.785398163397448) qr[6];
334 u1(-0.785398163397448) qr[3];
    cx qr[6],qr[3];
```

```
336  h qr[6];
     h qr[4];
338  u1(0.785398163397448) qr[4];
     u1(0.785398163397448) qr[5];
340  u1(0.785398163397448) qr[6];
     cx qr[5],qr[6];
342  cx qr[4],qr[5];
     cx qr[6],qr[4];
344  u1(-0.785398163397448) qr[5];
     u1(0.785398163397448) qr[4];
346  cx qr[6],qr[5];
     u1(-0.785398163397448) qr[5];
348  u1(-0.785398163397448) qr[6];
     cx qr[4],qr[5];
350  cx qr[6],qr[4];
     cx qr[5],qr[6];
352  h qr[4];
     h qr[6];
354  cx qr[6],qr[3];
     u1(0.785398163397448) qr[3];
356  u1(0.785398163397448) qr[6];
     cx qr[2],qr[3];
358  cx qr[6],qr[2];
     u1(-0.785398163397448) qr[3];
360  u1(-0.785398163397448) qr[2];
     cx qr[2],qr[3];
362  cx qr[6],qr[2];
     h qr[6];
364  h qr[5];
     cx qr[5],qr[1];
366  u1(-0.785398163397448) qr[1];
     u1(0.785398163397448) qr[5];
368  cx qr[0],qr[1];
     cx qr[5],qr[0];
370  u1(-0.785398163397448) qr[1];
     u1(0.785398163397448) qr[0];
372  cx qr[0],qr[1];
     cx qr[5],qr[0];
374  h qr[5];
     barrier qr[5];
376  h qr[5];
     cx qr[5],qr[0];
378  cx qr[0],qr[1];
     u1(-0.785398163397448) qr[0];
380  u1(0.785398163397448) qr[1];
     cx qr[5],qr[0];
382  cx qr[0],qr[1];
```

```
      u1(-0.785398163397448) qr[5];
384   u1(0.785398163397448) qr[1];
      cx qr[5],qr[1];
386   h qr[5];
      barrier qr[6];
388   h qr[6];
      cx qr[6],qr[2];
390   cx qr[2],qr[3];
      u1(-0.785398163397448) qr[2];
392   u1(0.785398163397448) qr[3];
      cx qr[6],qr[2];
394   cx qr[2],qr[3];
      u1(-0.785398163397448) qr[6];
396   u1(0.785398163397448) qr[3];
      cx qr[6],qr[3];
398   h qr[6];
      h qr[4];
400   u1(0.785398163397448) qr[4];
      u1(0.785398163397448) qr[5];
402   u1(0.785398163397448) qr[6];
      cx qr[5],qr[6];
404   cx qr[4],qr[5];
      cx qr[6],qr[4];
406   u1(-0.785398163397448) qr[5];
      u1(0.785398163397448) qr[4];
408   cx qr[6],qr[5];
      u1(-0.785398163397448) qr[5];
410   u1(-0.785398163397448) qr[6];
      cx qr[4],qr[5];
412   cx qr[6],qr[4];
      cx qr[5],qr[6];
414   h qr[4];
      h qr[6];
416   cx qr[6],qr[3];
      u1(-0.785398163397448) qr[3];
418   u1(0.785398163397448) qr[6];
      cx qr[2],qr[3];
420   cx qr[6],qr[2];
      u1(-0.785398163397448) qr[3];
422   u1(0.785398163397448) qr[2];
      cx qr[2],qr[3];
424   cx qr[6],qr[2];
      h qr[6];
426   h qr[5];
      cx qr[5],qr[1];
428   u1(-0.785398163397448) qr[1];
      u1(0.785398163397448) qr[5];
```

```
430  cx qr[0],qr[1];
     cx qr[5],qr[0];
432  u1(-0.785398163397448) qr[1];
     u1(0.785398163397448) qr[0];
434  cx qr[0],qr[1];
     cx qr[5],qr[0];
436  h qr[5];
     barrier qr[5];
438  h qr[5];
     cx qr[5],qr[0];
440  cx qr[0],qr[1];
     u1(-0.785398163397448) qr[0];
442  u1(0.785398163397448) qr[1];
     cx qr[5],qr[0];
444  cx qr[0],qr[1];
     u1(-0.785398163397448) qr[5];
446  u1(0.785398163397448) qr[1];
     cx qr[5],qr[1];
448  h qr[5];
     barrier qr[6];
450  h qr[6];
     cx qr[6],qr[2];
452  cx qr[2],qr[3];
     u1(-0.785398163397448) qr[2];
454  u1(-0.785398163397448) qr[3];
     cx qr[6],qr[2];
456  cx qr[2],qr[3];
     u1(-0.785398163397448) qr[6];
458  u1(-0.785398163397448) qr[3];
     cx qr[6],qr[3];
460  h qr[6];
     h qr[4];
462  u1(0.785398163397448) qr[4];
     u1(0.785398163397448) qr[5];
464  u1(0.785398163397448) qr[6];
     cx qr[5],qr[6];
466  cx qr[4],qr[5];
     cx qr[6],qr[4];
468  u1(-0.785398163397448) qr[5];
     u1(0.785398163397448) qr[4];
470  cx qr[6],qr[5];
     u1(-0.785398163397448) qr[5];
472  u1(-0.785398163397448) qr[6];
     cx qr[4],qr[5];
474  cx qr[6],qr[4];
     cx qr[5],qr[6];
476  h qr[4];
```

```
    h qr[6];
478 cx qr[6],qr[3];
    u1(0.785398163397448) qr[3];
480 u1(0.785398163397448) qr[6];
    cx qr[2],qr[3];
482 cx qr[6],qr[2];
    u1(0.785398163397448) qr[3];
484 u1(0.785398163397448) qr[2];
    cx qr[2],qr[3];
486 cx qr[6],qr[2];
    h qr[6];
488 h qr[5];
    cx qr[5],qr[1];
490 u1(-0.785398163397448) qr[1];
    u1(0.785398163397448) qr[5];
492 cx qr[0],qr[1];
    cx qr[5],qr[0];
494 u1(-0.785398163397448) qr[1];
    u1(0.785398163397448) qr[0];
496 cx qr[0],qr[1];
    cx qr[5],qr[0];
498 h qr[5];
    barrier qr[4];
500 barrier qr[3];
    barrier qr[2];
502 barrier qr[1];
    barrier qr[0];
504 measure qr[4] -> cr[4];
    measure qr[3] -> cr[3];
506 measure qr[2] -> cr[2];
    measure qr[1] -> cr[1];
508 measure qr[0] -> cr[0];
```

Listing E.14: QASM of original circuit fold quantamorphism over XOR gate after simulation.

```
    OPENQASM 2.0;
 2  include "qelib1.inc";
    qreg q[16];
 4  creg cr[7];
    u2(0,3.14159265358979) q[14];
 6  u2(0,3.14159265358979) q[13];
    u2(0,3.14159265358979) q[3];
 8  cx q[15],q[14];
    cx q[2],q[3];
10  cx q[1],q[0];
    u2(0,3.14159265358979) q[14];
12  u2(0,3.14159265358979) q[15];
```

```
   cx  q[15],q[14];
14 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[15];
16 cx  q[15],q[14];
   u2(0,3.92699081698724) q[15];
18 cx  q[13],q[14];
   u2(0,3.14159265358979) q[14];
20 u2(0,3.14159265358979) q[3];
   u2(0,3.14159265358979) q[2];
22 cx  q[2],q[3];
   u2(0,3.14159265358979) q[3];
24 u2(0,3.14159265358979) q[2];
   cx  q[2],q[3];
26 u2(0,3.14159265358979) q[3];
   cx  q[3],q[14];
28 u2(0.785398163397448,3.14159265358979) q[14];
   cx  q[13],q[14];
30 u1(-0.785398163397448) q[13];
   u2(0,3.14159265358979) q[14];
32 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
   cx  q[3],q[14];
34 u2(0,3.14159265358979) q[14];
   u2(0.785398163397448,3.14159265358979) q[3];
36 cx  q[3],q[14];
   u2(0,3.14159265358979) q[14];
38 u2(0,3.14159265358979) q[3];
   cx  q[3],q[14];
40 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[3];
42 cx  q[3],q[14];
   cx  q[13],q[14];
44 u2(0.785398163397448,3.14159265358979) q[13];
   cx  q[13],q[14];
46 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[13];
48 cx  q[13],q[14];
   u2(0,3.14159265358979) q[14];
50 u2(0,3.14159265358979) q[13];
   cx  q[13],q[14];
52 cx  q[3],q[14];
   u2(0,3.14159265358979) q[14];
54 u2(0,3.14159265358979) q[3];
   cx  q[3],q[14];
56 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[3];
58 cx  q[3],q[14];
   u2(0,3.14159265358979) q[2];
```

```
60  u2(0 ,3.14159265358979)  q[0];
    u2(0 ,3.14159265358979)  q[1];
62  cx  q[1] ,q[0];
    u2(0 ,3.14159265358979)  q[0];
64  u2(0 ,3.14159265358979)  q[1];
    cx  q[1] ,q[0];
66  u2(0 ,3.14159265358979)  q[1];
    cx  q[1] ,q[2];
68  u2(0 ,3.14159265358979)  q[1];
    cx  q[1] ,q[0];
70  u2(0 ,3.92699081698724)  q[1];
    u1(-0.785398163397448)  q[0];
72  cx  q[1] ,q[2];
    u2(-0.785398163397448 ,3.14159265358979)  q[2];
74  u2(0 ,3.14159265358979)  q[1];
    cx  q[1] ,q[0];
76  u1(0.785398163397448)  q[0];
    cx  q[1] ,q[2];
78  u2(0 ,3.14159265358979)  q[2];
    u2(0 ,3.14159265358979)  q[1];
80  cx  q[1] ,q[2];
    u2(0 ,3.14159265358979)  q[2];
82  u2(0 ,3.14159265358979)  q[1];
    cx  q[1] ,q[2];
84  cx  q[1] ,q[0];
    u2(0.785398163397448 ,3.14159265358979)  q[1];
86  cx  q[1] ,q[2];
    u2(0 ,3.14159265358979)  q[2];
88  u2(0 ,3.14159265358979)  q[1];
    cx  q[1] ,q[2];
90  u2(0 ,3.14159265358979)  q[2];
    u2(0 ,3.14159265358979)  q[1];
92  cx  q[1] ,q[2];
    cx  q[2] ,q[3];
94  u2(0 ,3.14159265358979)  q[2];
    cx  q[15] ,q[2];
96  u2(0 ,3.14159265358979)  q[2];
    u2(0 ,3.14159265358979)  q[15];
98  cx  q[15] ,q[2];
    u1(-0.785398163397448)  q[15];
100 cx  q[15] ,q[14];
    u2(0 ,3.14159265358979)  q[2];
102 u2(0 ,3.14159265358979)  q[14];
    u2(0 ,3.14159265358979)  q[15];
104 cx  q[15] ,q[14];
    u2(0 ,3.14159265358979)  q[14];
106 u2(0 ,3.14159265358979)  q[15];
```

```
     cx q[15],q[14];
108  u2(0,3.14159265358979) q[3];
     cx q[2],q[3];
110  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
112  u1(-0.785398163397448) q[14];
     u2(0,2.35619449019234) q[3];
114  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
116  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
118  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
120  cx q[15],q[14];
     cx q[13],q[14];
122  u2(0,3.14159265358979) q[15];
     u2(0,3.14159265358979) q[14];
124  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
126  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
128  cx q[13],q[14];
     u2(0,3.14159265358979) q[13];
130  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[15],q[2];
132  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
134  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
136  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
138  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
140  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
142  cx q[15],q[2];
     cx q[2],q[3];
144  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
146  u2(0,2.35619449019234) q[14];
     u1(0.785398163397448) q[3];
148  cx q[15],q[0];
     cx q[13],q[14];
150  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
152  cx q[13],q[14];
     u2(0,3.92699081698724) q[13];
```

```
154  cx q[3],q[14];
     u2(0,2.35619449019234) q[14];
156  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
158  cx q[3],q[14];
     u2(0,3.14159265358979) q[3];
160  barrier q[3];
     u2(0,3.14159265358979) q[3];
162  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
164  cx q[13],q[14];
     u2(0.785398163397448,3.14159265358979) q[14];
166  cx q[3],q[14];
     u1(-0.785398163397448) q[3];
168  u2(0,3.14159265358979) q[14];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[13];
170  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
172  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
174  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
176  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
178  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
180  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[13];
     cx q[13],q[14];
182  u1(0.785398163397448) q[14];
     u2(0,3.14159265358979) q[0];
184  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
186  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
188  cx q[15],q[0];
     u1(0.785398163397448) q[0];
190  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
192  u2(0.785398163397448,3.14159265358979) q[2];
     u2(-0.785398163397448,3.14159265358979) q[15];
194  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
196  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
198  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
200  cx q[15],q[2];
```

```
    cx q[1],q[2];
202 u1(0.785398163397448) q[2];
    cx q[15],q[0];
204 u2(0,3.14159265358979) q[1];
    u2(0,3.14159265358979) q[0];
206 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
208 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
210 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
212 cx q[1],q[0];
    u2(-0.785398163397448,3.14159265358979) q[1];
214 cx q[1],q[2];
    u2(0,3.14159265358979) q[1];
216 cx q[1],q[0];
    barrier q[0];
218 cx q[1],q[0];
    u2(0,3.14159265358979) q[1];
220 cx q[1],q[2];
    u1(-0.785398163397448) q[2];
222 u2(0,3.92699081698724) q[1];
    cx q[1],q[0];
224 u2(-0.785398163397448,3.14159265358979) q[0];
    cx q[15],q[0];
226 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
228 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
230 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
232 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[1];
234 cx q[1],q[2];
    u1(0.785398163397448) q[2];
236 cx q[15],q[2];
    u2(0.785398163397448,3.14159265358979) q[15];
238 cx q[15],q[14];
    u2(0,3.14159265358979) q[1];
240 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
242 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
244 cx q[15],q[14];
    u2(0,3.14159265358979) q[14];
246 u2(0,3.14159265358979) q[15];
    cx q[15],q[14];
```

```
248  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
250  cx q[15],q[14];
     u1(-0.785398163397448) q[14];
252  cx q[15],q[0];
     cx q[15],q[14];
254  u1(0.785398163397448) q[0];
     u1(-0.785398163397448) q[15];
256  u1(-0.785398163397448) q[14];
     cx q[15],q[0];
258  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
260  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
262  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
264  cx q[15],q[14];
     u2(0,3.14159265358979) q[0];
266  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
268  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
270  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
272  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
274  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
276  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
278  u1(0.785398163397448) q[0];
     u2(0,3.14159265358979) q[15];
280  cx q[15],q[14];
     cx q[15],q[14];
282  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
284  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
286  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
288  cx q[15],q[2];
     u1(0.785398163397448) q[15];
290  u2(0,2.35619449019234) q[2];
     cx q[1],q[2];
292  u2(0,3.14159265358979) q[14];
     cx q[13],q[14];
294  u2(0.785398163397448,3.14159265358979) q[14];
```

```
    u2(0.785398163397448,3.14159265358979) q[13];
296 cx q[13],q[14];
    u2(0,3.14159265358979) q[14];
298 u2(0,3.14159265358979) q[13];
    cx q[13],q[14];
300 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[13];
302 cx q[13],q[14];
    cx q[3],q[14];
304 cx q[13],q[14];
    u2(0,3.14159265358979) q[3];
306 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[13];
308 cx q[13],q[14];
    u2(0,3.14159265358979) q[14];
310 u2(0,3.14159265358979) q[13];
    cx q[13],q[14];
312 u1(-0.785398163397448) q[13];
    u2(0,3.14159265358979) q[14];
314 cx q[3],q[14];
    u2(0,3.14159265358979) q[14];
316 cx q[13],q[14];
    u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
318 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[13];
320 cx q[13],q[14];
    u2(0,3.14159265358979) q[14];
322 u2(0,3.14159265358979) q[13];
    cx q[13],q[14];
324 u2(0,3.14159265358979) q[14];
    cx q[3],q[14];
326 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[3];
328 cx q[3],q[14];
    cx q[13],q[14];
330 u2(0,3.14159265358979) q[13];
    barrier q[13];
332 u2(0,3.14159265358979) q[13];
    cx q[13],q[14];
334 u2(0,3.14159265358979) q[3];
    u2(0,3.14159265358979) q[14];
336 cx q[3],q[14];
    u2(0.785398163397448,3.14159265358979) q[14];
338 cx q[13],q[14];
    u1(-0.785398163397448) q[13];
340 u2(0,3.14159265358979) q[14];
    u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
```

```
342  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
344  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
346  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
348  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
350  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
352  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
     cx q[3],q[14];
354  u1(0.785398163397448) q[14];
     u2(0,3.14159265358979) q[3];
356  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
358  cx q[1],q[2];
     cx q[15],q[2];
360  u2(0,3.92699081698724) q[1];
     u2(0,2.35619449019234) q[2];
362  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
364  cx q[15],q[2];
     u2(0,3.14159265358979) q[15];
366  barrier q[15];
     u2(0,3.14159265358979) q[15];
368  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
370  cx q[1],q[2];
     u2(0.785398163397448,3.14159265358979) q[2];
372  cx q[15],q[2];
     u1(-0.785398163397448) q[15];
374  cx q[15],q[0];
     u2(0,3.14159265358979) q[2];
376  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     cx q[1],q[2];
378  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
380  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
382  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
384  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
386  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     u2(0,3.14159265358979) q[0];
388  u2(0,3.14159265358979) q[15];
```

```
    cx q[15],q[0];
390 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
392 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
394 cx q[1],q[0];
    u1(0.785398163397448) q[0];
396 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
398 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
400 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
402 cx q[15],q[0];
    cx q[15],q[14];
404 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
406 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
408 u2(0,3.14159265358979) q[15];
    cx q[15],q[14];
410 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[15];
412 cx q[15],q[14];
    u2(0,3.14159265358979) q[14];
414 u2(0,3.14159265358979) q[15];
    cx q[15],q[14];
416 u1(-0.785398163397448) q[14];
    cx q[15],q[0];
418 cx q[15],q[14];
    u2(0,3.92699081698724) q[0];
420 u1(-0.785398163397448) q[15];
    u1(-0.785398163397448) q[14];
422 cx q[15],q[14];
    u2(0,3.14159265358979) q[14];
424 u2(0,3.14159265358979) q[15];
    cx q[15],q[14];
426 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[15];
428 cx q[15],q[14];
    u2(0,3.14159265358979) q[14];
430 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
432 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
434 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
```

```
436   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
438   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
440   cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
442   u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
444   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
446   cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
448   u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
450   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
452   cx q[15],q[14];
      u1(0.785398163397448) q[14];
454   cx q[13],q[14];
      u2(0,3.14159265358979) q[15];
456   cx q[15],q[0];
      cx q[15],q[2];
458   u1(0.785398163397448) q[15];
      u1(0.785398163397448) q[2];
460   cx q[2],q[3];
      u2(0,3.14159265358979) q[0];
462   cx q[1],q[0];
      u2(0,3.14159265358979) q[3];
464   u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
466   u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
468   cx q[2],q[3];
      u2(0,3.14159265358979) q[2];
470   u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[14];
472   u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
474   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
476   cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
478   cx q[3],q[14];
      u2(0,3.14159265358979) q[14];
480   cx q[15],q[14];
      u2(0,2.35619449019234) q[14];
482   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
```

```
     cx q[3],q[14];
484  u2(0,3.14159265358979) q[14];
     u2(0.785398163397448,3.14159265358979) q[0];
486  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     cx q[1],q[2];
488  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
490  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
492  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
494  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
496  cx q[1],q[0];
     cx q[1],q[2];
498  u1(-0.785398163397448) q[0];
     u1(-0.785398163397448) q[2];
500  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
502  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
504  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
506  cx q[15],q[2];
     cx q[15],q[0];
508  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
510  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
512  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
514  cx q[15],q[2];
     cx q[15],q[14];
516  cx q[1],q[2];
     u2(0,3.14159265358979) q[15];
518  u2(0,3.14159265358979) q[1];
     barrier q[15];
520  barrier q[1];
     u2(0,3.14159265358979) q[15];
522  u2(0,3.14159265358979) q[1];
     cx q[15],q[14];
524  cx q[1],q[2];
     cx q[15],q[2];
526  u2(0,3.14159265358979) q[14];
     cx q[3],q[14];
528  u2(-0.785398163397448,3.14159265358979) q[14];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
```

```
530  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
532  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
534  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
536  cx q[15],q[0];
     u1(0.785398163397448) q[15];
538  u1(0.785398163397448) q[0];
     cx q[15],q[2];
540  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
542  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
544  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
546  cx q[15],q[14];
     cx q[1],q[2];
548  u1(-0.785398163397448) q[1];
     cx q[15],q[0];
550  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[14];
552  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
554  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
556  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
558  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
560  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
562  u2(0.785398163397448,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[0];
564  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
566  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
568  cx q[15],q[0];
     u1(-0.785398163397448) q[0];
570  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
572  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
574  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
576  cx q[2],q[3];
```

```
    u2(0,3.14159265358979) q[3];
578 u2(0,3.14159265358979) q[2];
    cx q[2],q[3];
580 u2(-0.785398163397448,3.14159265358979) q[15];
    cx q[15],q[0];
582 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
584 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
586 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
588 cx q[1],q[0];
    cx q[15],q[2];
590 u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
    u2(0.785398163397448,3.14159265358979) q[1];
592 cx q[1],q[2];
    u2(0,3.14159265358979) q[2];
594 u2(0,3.14159265358979) q[1];
    cx q[1],q[2];
596 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[1];
598 cx q[1],q[2];
    u2(0,3.14159265358979) q[2];
600 cx q[15],q[2];
    u2(0,3.14159265358979) q[2];
602 cx q[2],q[3];
    u2(0,3.14159265358979) q[3];
604 u2(0,3.14159265358979) q[2];
    cx q[2],q[3];
606 u2(0,3.14159265358979) q[3];
    u2(0,3.14159265358979) q[2];
608 cx q[2],q[3];
    cx q[1],q[2];
610 u2(0,3.14159265358979) q[3];
    cx q[3],q[14];
612 u2(0,3.14159265358979) q[14];
    u2(-0.785398163397448,3.14159265358979) q[3];
614 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[1];
616 cx q[1],q[2];
    u2(0,3.14159265358979) q[2];
618 u2(0,3.14159265358979) q[1];
    cx q[1],q[2];
620 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[15];
622 cx q[15],q[14];
    cx q[3],q[14];
```

```
624  u2(0 ,3.14159265358979) q[14];
     u2(0 ,3.14159265358979) q[3];
626  cx q[3],q[14];
     u2(0 ,3.14159265358979) q[14];
628  u2(0 ,3.14159265358979) q[3];
     cx q[3],q[14];
630  u1(0.785398163397448) q[3];
     cx q[15],q[14];
632  u1(-0.785398163397448) q[15];
     u1(-0.785398163397448) q[14];
634  cx q[3],q[14];
     cx q[15],q[14];
636  u2(0 ,3.14159265358979) q[3];
     u2(0 ,3.14159265358979) q[14];
638  u2(0 ,3.14159265358979) q[15];
     cx q[15],q[14];
640  u2(0 ,3.14159265358979) q[14];
     u2(0 ,3.14159265358979) q[15];
642  cx q[15],q[14];
     u2(0 ,3.14159265358979) q[14];
644  cx q[3],q[14];
     u2(0 ,3.14159265358979) q[14];
646  cx q[15],q[14];
     u2(0 ,3.14159265358979) q[15];
648  u2(0 ,3.14159265358979) q[14];
     cx q[15],q[0];
650  u1(0.785398163397448) q[15];
     u1(0.785398163397448) q[0];
652  cx q[1],q[0];
     u1(-0.785398163397448) q[0];
654  u2(0 ,3.14159265358979) q[1];
     u2(0 ,3.14159265358979) q[3];
656  cx q[3],q[14];
     u2(0 ,3.14159265358979) q[14];
658  u2(0 ,3.14159265358979) q[3];
     cx q[3],q[14];
660  u2(0 ,3.14159265358979) q[14];
     u2(0 ,3.14159265358979) q[3];
662  cx q[3],q[14];
     u1(0.785398163397448) q[14];
664  cx q[13],q[14];
     u2(0 ,3.14159265358979) q[3];
666  cx q[2],q[3];
     u2(0.785398163397448 ,3.14159265358979) q[3];
668  u2(-0.785398163397448 ,3.14159265358979) q[2];
     cx q[2],q[3];
670  u2(0 ,3.14159265358979) q[3];
```

```
     u2(0,3.14159265358979) q[2];
672  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
674  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
676  cx q[15],q[2];
     u2(0,3.14159265358979) q[3];
678  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
680  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
682  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
684  u2(0,3.14159265358979) q[2];
     cx q[1],q[2];
686  u2(-0.785398163397448,3.14159265358979) q[1];
     cx q[1],q[0];
688  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
690  barrier q[2];
     cx q[1],q[2];
692  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
694  u2(0,3.92699081698724) q[1];
     u1(0.785398163397448) q[0];
696  cx q[1],q[2];
     u2(-0.785398163397448,3.14159265358979) q[2];
698  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
700  u1(-0.785398163397448) q[0];
     u2(0,3.14159265358979) q[14];
702  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
704  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
706  cx q[13],q[14];
     u2(0,3.14159265358979) q[13];
708  u2(0,3.14159265358979) q[14];
     cx q[3],q[14];
710  u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
712  u2(0,3.92699081698724) q[14];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
714  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
716  cx q[15],q[14];
     u2(0,3.14159265358979) q[15];
```

```
718  barrier q[15];
     u2(0,3.14159265358979) q[15];
720  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
722  cx q[3],q[14];
     u2(-0.785398163397448,3.14159265358979) q[14];
724  cx q[15],q[14];
     u1(-0.785398163397448) q[15];
726  cx q[15],q[2];
     u2(0,3.14159265358979) q[14];
728  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
730  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
732  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
734  cx q[15],q[0];
     u2(0.785398163397448,3.14159265358979) q[15];
736  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[3];
     cx q[3],q[14];
738  u2(0,3.14159265358979) q[14];
     u2(-0.785398163397448,3.14159265358979) q[3];
740  cx q[2],q[3];
     u2(0.785398163397448,3.14159265358979) q[2];
742  cx q[15],q[2];
     cx q[2],q[3];
744  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
746  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
748  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
750  cx q[15],q[14];
     cx q[15],q[0];
752  u2(0,3.14159265358979) q[14];
     cx q[13],q[14];
754  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
756  cx q[13],q[14];
     u2(0,2.35619449019234) q[13];
758  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[0];
760  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
762  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
764  cx q[15],q[0];
```

```
    cx q[1],q[0];
766 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[1];
768 cx q[1],q[0];
    u2(0,3.14159265358979) q[0];
770 u2(0,3.14159265358979) q[1];
    cx q[1],q[0];
772 u2(0,3.14159265358979) q[1];
    u2(0,3.14159265358979) q[0];
774 u2(0,3.14159265358979) q[3];
    u2(0,3.14159265358979) q[2];
776 cx q[2],q[3];
    u2(0,3.14159265358979) q[3];
778 u2(0,3.14159265358979) q[2];
    cx q[2],q[3];
780 u2(0,3.14159265358979) q[3];
    cx q[3],q[14];
782 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[3];
784 cx q[3],q[14];
    u1(0.785398163397448) q[3];
786 u2(0,3.14159265358979) q[14];
    cx q[13],q[14];
788 u2(-0.785398163397448,3.14159265358979) q[14];
    cx q[3],q[14];
790 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[13];
    u2(0,3.14159265358979) q[14];
792 u2(0,3.14159265358979) q[3];
    cx q[3],q[14];
794 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[3];
796 cx q[3],q[14];
    u2(0,3.14159265358979) q[3];
798 u2(0,3.14159265358979) q[14];
    cx q[13],q[14];
800 cx q[3],q[14];
    u2(0,3.14159265358979) q[14];
802 u2(0,3.14159265358979) q[3];
    cx q[3],q[14];
804 u2(0,3.14159265358979) q[13];
    cx q[13],q[14];
806 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[13];
808 cx q[3],q[14];
    u2(0,3.14159265358979) q[14];
810 u2(0,3.14159265358979) q[3];
    cx q[3],q[14];
```

```
812   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[3];
814   cx q[3],q[14];
      cx q[15],q[14];
816   u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[14];
818   u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
820   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
822   cx q[15],q[14];
      cx q[13],q[14];
824   u1(0.785398163397448) q[15];
      u1(0.785398163397448) q[14];
826   u1(0.785398163397448) q[13];
      cx q[15],q[14];
828   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
830   cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
832   u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
834   cx q[13],q[14];
      u2(0,3.14159265358979) q[15];
836   cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
838   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
840   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
842   cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
844   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
846   u1(-0.785398163397448) q[0];
      u2(0,3.14159265358979) q[15];
848   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
850   cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
852   u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
854   u2(0,3.14159265358979) q[13];
      u2(0,3.14159265358979) q[14];
856   cx q[15],q[14];
      u2(-0.785398163397448,3.14159265358979) q[15];
858   cx q[15],q[0];
```

```
     u2(0,3.14159265358979) q[15];
860  cx q[15],q[14];
     barrier q[14];
862  cx q[15],q[14];
     u2(0,3.14159265358979) q[15];
864  cx q[15],q[0];
     u2(0,2.35619449019234) q[15];
866  u1(0.785398163397448) q[0];
     cx q[15],q[14];
868  u2(-0.785398163397448,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
870  cx q[15],q[0];
     u1(0.785398163397448) q[0];
872  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
874  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
876  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
878  cx q[15],q[14];
     cx q[15],q[0];
880  u2(0.785398163397448,3.14159265358979) q[15];
     u2(0,3.14159265358979) q[2];
882  cx q[2],q[3];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
884  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[1],q[2];
886  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
888  cx q[1],q[2];
     u2(0,3.92699081698724) q[1];
890  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
892  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[1],q[2];
894  cx q[2],q[3];
     barrier q[3];
896  cx q[2],q[3];
     cx q[1],q[2];
898  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
     cx q[2],q[3];
900  u2(-0.785398163397448,3.14159265358979) q[3];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
902  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
904  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
```

```
906  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
908  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
910  cx q[2],q[3];
     u2(0,3.14159265358979) q[2];
912  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     cx q[1],q[2];
914  u1(0.785398163397448) q[2];
     cx q[15],q[2];
916  cx q[2],q[3];
     cx q[15],q[14];
918  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
920  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
922  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
924  cx q[15],q[0];
     u2(0,3.14159265358979) q[14];
926  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
928  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
930  u1(-0.785398163397448) q[13];
     u2(0,3.14159265358979) q[0];
932  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
934  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
936  cx q[15],q[0];
     u2(0,3.14159265358979) q[3];
938  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
940  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
942  cx q[2],q[3];
     cx q[3],q[14];
944  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
946  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
948  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
950  cx q[13],q[14];
     u2(0,3.92699081698724) q[13];
952  cx q[3],q[14];
```

```
     u2(0,2.35619449019234) q[14];
954  u2(0,2.35619449019234) q[3];
     cx q[13],q[14];
956  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
958  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
960  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
962  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
964  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
966  u2(0,3.14159265358979) q[13];
     cx q[3],q[14];
968  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
970  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
972  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
974  cx q[15],q[14];
     u2(0,3.14159265358979) q[3];
976  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
978  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
980  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
982  cx q[13],q[14];
     u1(0.785398163397448) q[15];
984  u2(0,2.35619449019234) q[14];
     u1(0.785398163397448) q[13];
986  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
988  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
990  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
992  cx q[15],q[0];
     u2(0,3.14159265358979) q[15];
994  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
996  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
998  u2(0,2.35619449019234) q[15];
     cx q[13],q[14];
```

```
1000  u2(0,3.92699081698724) q[14];
      cx q[15],q[14];
1002  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1004  u2(0,3.14159265358979) q[13];
      barrier q[13];
1006  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1008  u2(0,3.14159265358979) q[14];
      cx q[15],q[14];
1010  u2(-0.785398163397448,3.14159265358979) q[14];
      cx q[13],q[14];
1012  u1(-0.785398163397448) q[13];
      u2(0,3.14159265358979) q[14];
1014  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
      cx q[15],q[14];
1016  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1018  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
1020  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1022  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1024  u2(0,3.14159265358979) q[14];
      u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
1026  cx q[15],q[14];
      u2(0,3.92699081698724) q[14];
1028  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1030  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1032  cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
1034  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1036  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1038  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1040  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1042  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1044  u2(0,3.14159265358979) q[1];
      u2(0,3.14159265358979) q[2];
1046  cx q[2],q[3];
```

```
      u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
1048  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
      cx q[1],q[2];
1050  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1052  cx q[1],q[2];
      u2(0,2.35619449019234) q[1];
1054  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1056  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
      cx q[1],q[2];
1058  cx q[2],q[3];
      barrier q[3];
1060  cx q[2],q[3];
      cx q[1],q[2];
1062  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
      cx q[2],q[3];
1064  u2(-0.785398163397448,3.14159265358979) q[3];
      u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
1066  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1068  cx q[2],q[3];
      u2(0,3.14159265358979) q[3];
1070  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1072  u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
1074  cx q[2],q[3];
      u2(0,3.14159265358979) q[2];
1076  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
      cx q[1],q[2];
1078  u1(0.785398163397448) q[2];
      cx q[2],q[3];
1080  u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
1082  cx q[2],q[3];
      u2(0,3.14159265358979) q[3];
1084  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1086  u2(0,3.14159265358979) q[3];
      cx q[3],q[14];
1088  u2(0,3.14159265358979) q[14];
      cx q[15],q[14];
1090  u1(-0.785398163397448) q[14];
      u2(0,3.14159265358979) q[15];
1092  u2(0,3.14159265358979) q[3];
      cx q[2],q[3];
```

```
1094  u2(0 ,3.14159265358979) q[3];
      u2(0 ,3.14159265358979) q[2];
1096  cx  q[2],q[3];
      u2(0 ,3.14159265358979) q[3];
1098  u2(0 ,3.14159265358979) q[2];
      cx  q[2],q[3];
1100  cx  q[3],q[14];
      u2(0 ,3.14159265358979) q[2];
1102  cx  q[15],q[2];
      u2(0 ,3.14159265358979) q[2];
1104  u2(0.785398163397448 ,3.14159265358979) q[15];
      u2(0 ,3.14159265358979) q[14];
1106  u2(0 ,3.14159265358979) q[3];
      cx  q[3],q[14];
1108  u2(0 ,3.14159265358979) q[14];
      u2(0 ,3.14159265358979) q[3];
1110  cx  q[3],q[14];
      cx  q[2],q[3];
1112  u1(-0.785398163397448) q[3];
      u2(0 ,2.35619449019234) q[2];
1114  cx  q[3],q[14];
      u2(0 ,3.14159265358979) q[14];
1116  u2(0 ,3.14159265358979) q[3];
      cx  q[3],q[14];
1118  u2(0 ,3.14159265358979) q[14];
      u2(0 ,3.14159265358979) q[3];
1120  cx  q[3],q[14];
      cx  q[15],q[14];
1122  cx  q[3],q[14];
      u2(0 ,3.14159265358979) q[15];
1124  cx  q[15],q[2];
      u2(0 ,3.14159265358979) q[15];
1126  cx  q[15],q[0];
      u2(0 ,3.14159265358979) q[0];
1128  u2(0 ,3.14159265358979) q[15];
      cx  q[15],q[0];
1130  u2(0 ,3.14159265358979) q[0];
      u2(0 ,3.14159265358979) q[15];
1132  cx  q[15],q[0];
      u2(0 ,3.92699081698724) q[0];
1134  u2(0 ,3.14159265358979) q[15];
      u2(0 ,3.14159265358979) q[14];
1136  u2(0 ,3.14159265358979) q[3];
      cx  q[3],q[14];
1138  u2(0 ,3.14159265358979) q[14];
      u2(0 ,3.14159265358979) q[3];
1140  cx  q[3],q[14];
```

```
     u2(0,3.14159265358979) q[3];
1142 cx q[2],q[3];
     cx q[3],q[14];
1144 u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
1146 cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
1148 u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
1150 u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
1152 u2(0.785398163397448,3.14159265358979) q[14];
     cx q[13],q[14];
1154 u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
1156 cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
1158 u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
1160 u2(0,3.14159265358979) q[14];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[15];
1162 cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
1164 cx q[13],q[14];
     u2(0,3.92699081698724) q[14];
1166 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[15];
     cx q[15],q[14];
1168 u2(0,3.14159265358979) q[14];
     cx q[13],q[14];
1170 u2(0,3.14159265358979) q[13];
     barrier q[13];
1172 u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
1174 u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
1176 u2(-0.785398163397448,3.14159265358979) q[14];
     cx q[13],q[14];
1178 u1(-0.785398163397448) q[13];
     u2(0,3.14159265358979) q[14];
1180 u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
     cx q[15],q[14];
1182 u2(0,3.14159265358979) q[14];
     u2(0.785398163397448,3.14159265358979) q[15];
1184 cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
1186 u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
```

```
1188  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
1190  cx q[15],q[14];
      cx q[13],q[14];
1192  u2(0.785398163397448,3.14159265358979) q[13];
      cx q[13],q[14];
1194  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
1196  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1198  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1200  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[2];
1202  cx q[1],q[2];
      u2(0.785398163397448,3.14159265358979) q[2];
1204  cx q[2],q[3];
      u2(0,3.14159265358979) q[3];
1206  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1208  u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
1210  cx q[2],q[3];
      u2(0,3.14159265358979) q[3];
1212  u2(0,3.14159265358979) q[2];
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
1214  cx q[1],q[2];
      cx q[2],q[3];
1216  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[2];
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
1218  cx q[1],q[2];
      cx q[2],q[3];
1220  barrier q[3];
      cx q[2],q[3];
1222  cx q[1],q[2];
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
1224  cx q[2],q[3];
      u2(-0.785398163397448,3.14159265358979) q[3];
1226  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
      cx q[1],q[2];
1228  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1230  u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
1232  cx q[2],q[3];
      u2(0,3.14159265358979) q[3];
1234  u2(0,3.14159265358979) q[2];
```

```
     cx q[2],q[3];
1236 u2(0,3.14159265358979) q[2];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
1238 cx q[1],q[2];
     u1(0.785398163397448) q[2];
1240 cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
1242 u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
1244 u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
1246 cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
1248 cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
1250 cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
1252 u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
1254 u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
1256 cx q[15],q[14];
     cx q[13],q[14];
1258 u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
1260 u2(0,3.14159265358979) q[0];
     u3(0.785398163397448,1.57079632679490,4.71238898038469) q[15];
1262 u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
1264 cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
1266 u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
1268 u2(0,3.14159265358979) q[3];
     cx q[2],q[3];
1270 u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
1272 cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
1274 u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
1276 u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
1278 u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
1280 u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
```

```
1282  cx q[1],q[0];
      u2(0,3.14159265358979) q[0];
1284  u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
1286  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1288  cx q[15],q[2];
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
1290  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[15];
      u2(0.785398163397448,3.14159265358979) q[1];
1292  cx q[1],q[0];
      u2(0,3.14159265358979) q[0];
1294  u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
1296  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[1];
1298  cx q[1],q[0];
      u2(0,3.14159265358979) q[0];
1300  cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
1302  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1304  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1306  cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
1308  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1310  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1312  cx q[15],q[2];
      barrier q[15];
1314  u2(0,3.14159265358979) q[2];
      cx q[1],q[2];
1316  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1318  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1320  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1322  u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
1324  cx q[1],q[2];
      u2(0,3.92699081698724) q[2];
1326  u1(0.785398163397448) q[1];
      cx q[15],q[0];
1328  cx q[2],q[3];
```

```
      u2(0,3.14159265358979) q[3];
1330  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1332  u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
1334  cx q[2],q[3];
      u2(0,3.14159265358979) q[3];
1336  u2(0,3.14159265358979) q[2];
      cx q[2],q[3];
1338  u1(0.785398163397448) q[3];
      cx q[1],q[2];
1340  u1(0.785398163397448) q[2];
      cx q[2],q[3];
1342  barrier q[3];
      cx q[1],q[2];
1344  measure q[3] -> cr[3];
      barrier q[2];
1346  u2(0,3.14159265358979) q[1];
      measure q[2] -> cr[2];
1348  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1350  cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
1352  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1354  cx q[15],q[14];
      measure q[0] -> cr[4];
1356  u1(0.785398163397448) q[15];
      u1(-0.785398163397448) q[14];
1358  cx q[13],q[14];
      cx q[15],q[14];
1360  u2(0,3.14159265358979) q[13];
      u2(0,3.14159265358979) q[14];
1362  u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
1364  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
1366  cx q[15],q[14];
      u2(0,2.35619449019234) q[15];
1368  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1370  u2(0,3.14159265358979) q[14];
      u2(0.785398163397448,3.14159265358979) q[13];
1372  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1374  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
```

```
1376  u2(0 ,3.14159265358979)  q[14];
      u2(0 ,3.14159265358979)  q[13];
1378  cx  q[13] ,q[14];
      u2(0 ,3.14159265358979)  q[14];
1380  cx  q[15] ,q[14];
      u2(0 ,3.14159265358979)  q[14];
1382  cx  q[13] ,q[14];
      barrier  q[14];
1384  u2(0 ,3.14159265358979)  q[13];
      measure  q[14]  -> cr[0];
1386  u2(0 ,3.14159265358979)  q[15];
      barrier  q[15];
1388  measure  q[15]  -> cr[1];
```

Listing E.15: QASM of original circuit fold quantamorphism over XOR gate after running in quantum device.

```
      OPENQASM  2.0;
 2    include  "qelib1.inc";
      qreg  q[16];
 4    creg  cr[7];
      u2(0 ,3.14159265358979)  q[15];
 6    u2(0 ,3.14159265358979)  q[1];
      u2(0 ,3.14159265358979)  q[0];
 8    cx  q[13] ,q[14];
      cx  q[1] ,q[2];
10    cx  q[15] ,q[0];
      u2(0 ,3.14159265358979)  q[14];
12    u2(0 ,3.14159265358979)  q[13];
      cx  q[13] ,q[14];
14    u2(0 ,3.14159265358979)  q[14];
      u2(0 ,3.14159265358979)  q[13];
16    cx  q[13] ,q[14];
      u2(0 ,3.14159265358979)  q[13];
18    u2(0 ,3.14159265358979)  q[2];
      u2(0 ,3.14159265358979)  q[1];
20    cx  q[1] ,q[2];
      u2(0 ,3.14159265358979)  q[2];
22    u2(0 ,3.14159265358979)  q[1];
      cx  q[1] ,q[2];
24    u2(0 ,3.14159265358979)  q[2];
      u2(0 ,3.14159265358979)  q[0];
26    u2(0 ,3.14159265358979)  q[15];
      cx  q[15] ,q[0];
28    u2(0 ,3.14159265358979)  q[0];
      u2(0 ,3.14159265358979)  q[15];
30    cx  q[15] ,q[0];
```

```
   u1(0.785398163397448) q[0];
32 cx q[15],q[14];
   u2(0,3.14159265358979) q[14];
34 u2(0,3.14159265358979) q[15];
   cx q[15],q[14];
36 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[15];
38 cx q[15],q[14];
   u2(0,3.14159265358979) q[15];
40 cx q[15],q[2];
   u2(0,3.14159265358979) q[2];
42 cx q[1],q[2];
   u2(0,3.14159265358979) q[2];
44 u2(0,3.14159265358979) q[1];
   cx q[1],q[2];
46 u2(0,3.14159265358979) q[2];
   u2(0,3.14159265358979) q[1];
48 cx q[1],q[2];
   u2(0,3.14159265358979) q[2];
50 u2(0,3.14159265358979) q[15];
   u2(0,3.14159265358979) q[14];
52 cx q[13],q[14];
   u2(0,3.14159265358979) q[14];
54 u2(0,3.14159265358979) q[13];
   cx q[13],q[14];
56 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[13];
58 cx q[13],q[14];
   u2(0,3.14159265358979) q[14];
60 u2(0,3.14159265358979) q[13];
   cx q[13],q[14];
62 cx q[3],q[14];
   u2(0,3.14159265358979) q[14];
64 u2(0,3.14159265358979) q[3];
   cx q[3],q[14];
66 u2(0,3.14159265358979) q[14];
   u2(0,3.14159265358979) q[3];
68 cx q[3],q[14];
   cx q[15],q[14];
70 u1(0.785398163397448) q[15];
   u1(-0.785398163397448) q[14];
72 cx q[15],q[0];
   u2(0,3.14159265358979) q[3];
74 cx q[2],q[3];
   u2(0.785398163397448,3.14159265358979) q[3];
76 cx q[3],q[14];
   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
```

```
78  u2(0 ,3.14159265358979) q[14];
    u2(0 ,3.14159265358979) q[3];
80  cx q[3] ,q[14];
    u2(0 ,3.14159265358979) q[14];
82  u2(0 ,3.14159265358979) q[3];
    cx q[3] ,q[14];
84  cx q[13] ,q[14];
    u1( -0.785398163397448) q[13];
86  cx q[3] ,q[14];
    u2(0 ,3.14159265358979) q[14];
88  u2(0 ,3.14159265358979) q[3];
    cx q[3] ,q[14];
90  u2(0 ,3.14159265358979) q[14];
    u2(0 ,3.14159265358979) q[3];
92  cx q[3] ,q[14];
    u2(0 ,3.14159265358979) q[3];
94  cx q[2] ,q[3];
    u2(0 ,3.14159265358979) q[3];
96  u2(0.785398163397448 ,3.14159265358979) q[2];
    cx q[2] ,q[3];
98  u2(0 ,3.14159265358979) q[3];
    u2(0 ,3.14159265358979) q[2];
100 cx q[2] ,q[3];
    u2(0 ,3.14159265358979) q[3];
102 u2(0 ,3.14159265358979) q[2];
    cx q[2] ,q[3];
104 u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[15];
106 cx q[15] ,q[0];
    u2(0 ,3.14159265358979) q[0];
108 u2(0 ,3.14159265358979) q[15];
    cx q[15] ,q[0];
110 cx q[1] ,q[0];
    u1( -0.785398163397448) q[1];
112 cx q[15] ,q[0];
    u2(0 ,3.14159265358979) q[0];
114 u2(0 ,3.14159265358979) q[15];
    cx q[15] ,q[0];
116 u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[15];
118 cx q[15] ,q[0];
    cx q[15] ,q[14];
120 u1(0.785398163397448) q[14];
    cx q[15] ,q[14];
122 cx q[1] ,q[0];
    u2(0 ,3.14159265358979) q[14];
124 u2(0 ,3.14159265358979) q[15];
```

```
      cx q[15],q[14];
126   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
128   cx q[15],q[14];
      cx q[3],q[14];
130   u2(0,3.14159265358979) q[15];
      u2(0,3.14159265358979) q[14];
132   u2(0,3.14159265358979) q[3];
      cx q[3],q[14];
134   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[3];
136   cx q[3],q[14];
      cx q[13],q[14];
138   u2(0.785398163397448,3.14159265358979) q[13];
      cx q[13],q[14];
140   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
142   cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
144   u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
146   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[1];
148   cx q[1],q[0];
      u2(0,3.14159265358979) q[0];
150   u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
152   u2(0,3.14159265358979) q[1];
      u2(0,3.14159265358979) q[0];
154   cx q[15],q[0];
      u1(0.785398163397448) q[0];
156   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
158   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
160   cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
162   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
164   cx q[15],q[14];
      cx q[3],q[14];
166   cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
168   u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
170   u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[15];
```

```
172  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
174  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
176  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
178  u1(-0.785398163397448) q[1];
     u2(0,3.14159265358979) q[2];
180  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
182  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
184  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
186  cx q[13],q[14];
     u2(0,3.14159265358979) q[3];
188  cx q[2],q[3];
     u2(0,3.14159265358979) q[2];
190  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
192  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
194  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
196  cx q[1],q[2];
     u1(0.785398163397448) q[1];
198  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
200  u2(-0.785398163397448,3.14159265358979) q[3];
     u2(-0.785398163397448,3.14159265358979) q[2];
202  cx q[1],q[2];
     cx q[2],q[3];
204  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[3];
206  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
208  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
210  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
212  u2(0,3.14159265358979) q[2];
     cx q[1],q[2];
214  cx q[2],q[3];
     cx q[2],q[3];
216  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
218  cx q[2],q[3];
```

```
    u2(0 ,3.14159265358979) q[3];
220 u2(0 ,3.14159265358979) q[2];
    cx q[2],q[3];
222 u2(0 ,3.14159265358979) q[2];
    u2(0 ,3.14159265358979) q[1];
224 cx q[1],q[0];
    u2(0 ,3.14159265358979) q[0];
226 u2(0 ,3.14159265358979) q[1];
    cx q[1],q[0];
228 u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[1];
230 cx q[1],q[0];
    u1(0.785398163397448) q[0];
232 u2(0 ,3.14159265358979) q[1];
    cx q[1],q[2];
234 u3(-0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[2];
    u2(-0.785398163397448 ,3.14159265358979) q[1];
236 cx q[1],q[0];
    u2(0 ,3.14159265358979) q[0];
238 u2(0 ,3.14159265358979) q[1];
    cx q[1],q[0];
240 u2(0 ,3.14159265358979) q[0];
    u2(0 ,3.14159265358979) q[1];
242 cx q[1],q[0];
    u2(0 ,3.14159265358979) q[14];
244 u2(0 ,3.14159265358979) q[13];
    cx q[13],q[14];
246 u2(0 ,3.14159265358979) q[14];
    u2(0 ,3.14159265358979) q[13];
248 cx q[13],q[14];
    cx q[3],q[14];
250 u1(-0.785398163397448) q[14];
    u1(0.785398163397448) q[3];
252 cx q[13],q[14];
    u2(0 ,3.14159265358979) q[14];
254 u2(0 ,3.14159265358979) q[13];
    cx q[13],q[14];
256 u2(0 ,3.14159265358979) q[14];
    u2(0 ,3.14159265358979) q[13];
258 cx q[13],q[14];
    cx q[15],q[14];
260 u2(0 ,3.14159265358979) q[13];
    u2(0 ,3.14159265358979) q[14];
262 u2(0 ,3.14159265358979) q[15];
    cx q[15],q[14];
264 u2(0 ,3.14159265358979) q[14];
    u2(0 ,3.14159265358979) q[15];
```

```
266  cx q[15],q[14];
     cx q[15],q[0];
268  u1(0.785398163397448) q[0];
     u2(0,3.14159265358979) q[14];
270  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
272  cx q[3],q[14];
     u2(0,2.35619449019234) q[14];
274  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[13];
     cx q[13],q[14];
276  u2(0,3.14159265358979) q[14];
     cx q[3],q[14];
278  u2(0,3.14159265358979) q[3];
     barrier q[3];
280  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
282  u2(0,3.14159265358979) q[14];
     cx q[13],q[14];
284  u2(0.785398163397448,3.14159265358979) q[14];
     cx q[3],q[14];
286  u1(-0.785398163397448) q[3];
     u2(0,3.14159265358979) q[14];
288  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[13];
     cx q[13],q[14];
290  u2(0,3.14159265358979) q[14];
     u2(-0.785398163397448,3.14159265358979) q[13];
292  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
294  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
296  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
298  cx q[13],q[14];
     cx q[3],q[14];
300  u2(0.785398163397448,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[15];
302  cx q[15],q[2];
     u2(-0.785398163397448,3.14159265358979) q[15];
304  cx q[15],q[0];
     u2(0,3.14159265358979) q[15];
306  cx q[15],q[2];
     barrier q[2];
308  cx q[15],q[2];
     u2(0,3.14159265358979) q[15];
310  cx q[15],q[0];
     u2(0,3.92699081698724) q[15];
312  u1(-0.785398163397448) q[0];
```

```
     cx q[15],q[2];
314  u2(-0.785398163397448,3.14159265358979) q[2];
     cx q[1],q[2];
316  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
318  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
320  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
322  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
324  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
326  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
328  cx q[2],q[3];
     u2(0,3.14159265358979) q[15];
330  cx q[15],q[0];
     u1(0.785398163397448) q[0];
332  cx q[1],q[0];
     u2(0.785398163397448,3.14159265358979) q[1];
334  cx q[1],q[2];
     cx q[2],q[3];
336  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[3];
338  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
340  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
342  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
344  u2(0,3.14159265358979) q[2];
     cx q[1],q[2];
346  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
348  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
350  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
352  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
354  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
356  u1(0.785398163397448) q[3];
     u2(0,3.14159265358979) q[2];
358  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     cx q[1],q[2];
```

```
360  u2(-0.785398163397448,3.14159265358979) q[2];
     cx q[2],q[3];
362  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
364  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
366  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
368  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
370  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     cx q[1],q[2];
372  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
374  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
376  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
378  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
380  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
382  u2(0,3.92699081698724) q[3];
     u2(0,3.14159265358979) q[1];
384  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
386  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
388  u1(0.785398163397448) q[1];
     u1(-0.785398163397448) q[0];
390  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
392  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
394  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
396  cx q[15],q[2];
     cx q[15],q[14];
398  u1(0.785398163397448) q[15];
     u1(0.785398163397448) q[14];
400  cx q[13],q[14];
     cx q[1],q[2];
402  cx q[15],q[14];
     u2(0,3.14159265358979) q[2];
404  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
406  u2(0,3.14159265358979) q[2];
```

```
      u2(0,3.14159265358979) q[1];
408   cx q[1],q[2];
      cx q[1],q[0];
410   u1(0.785398163397448) q[0];
      u2(0,3.14159265358979) q[2];
412   u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
414   u2(-0.785398163397448,3.14159265358979) q[1];
      cx q[1],q[0];
416   u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
418   barrier q[2];
      cx q[1],q[2];
420   u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
422   u2(0,3.92699081698724) q[1];
      u1(0.785398163397448) q[0];
424   cx q[1],q[2];
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[2];
426   u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
428   u1(-0.785398163397448) q[0];
      cx q[1],q[0];
430   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[1];
432   cx q[1],q[0];
      u2(0,3.14159265358979) q[0];
434   u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
436   u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
438   u1(0.785398163397448) q[2];
      u2(0,3.14159265358979) q[1];
440   cx q[1],q[0];
      u2(0,3.14159265358979) q[0];
442   u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
444   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[1];
446   cx q[1],q[0];
      u2(0,3.14159265358979) q[13];
448   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
450   cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
452   u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
```

```
454  u2(0,2.35619449019234) q[15];
     u2(0,3.14159265358979) q[14];
456  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
458  u2(-0.785398163397448,3.14159265358979) q[13];
     cx q[13],q[14];
460  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
462  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
464  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
466  u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
468  u2(0,3.14159265358979) q[14];
     cx q[13],q[14];
470  u2(0,3.14159265358979) q[13];
     barrier q[13];
472  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
474  u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
476  u2(0.785398163397448,3.14159265358979) q[14];
     cx q[13],q[14];
478  u1(-0.785398163397448) q[13];
     u2(0,3.14159265358979) q[14];
480  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
     cx q[15],q[14];
482  u2(0,3.14159265358979) q[14];
     u2(-0.785398163397448,3.14159265358979) q[15];
484  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
486  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
488  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
490  cx q[15],q[14];
     cx q[13],q[14];
492  u2(0.785398163397448,3.14159265358979) q[13];
     cx q[15],q[2];
494  cx q[13],q[14];
     u2(0,3.14159265358979) q[2];
496  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
498  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
500  cx q[15],q[2];
```

```
     u2(0,3.14159265358979) q[15];
502  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
504  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
506  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
508  u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
510  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
512  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
514  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
516  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
518  cx q[15],q[14];
     u1(-0.785398163397448) q[3];
520  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
522  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
524  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
526  cx q[3],q[14];
     cx q[15],q[14];
528  u1(0.785398163397448) q[3];
     u1(-0.785398163397448) q[15];
530  u1(-0.785398163397448) q[14];
     cx q[3],q[14];
532  cx q[15],q[14];
     u2(0,3.14159265358979) q[3];
534  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
536  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
538  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
540  u2(0,3.14159265358979) q[14];
     cx q[3],q[14];
542  u2(0,3.14159265358979) q[14];
     cx q[15],q[14];
544  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
546  u1(0.785398163397448) q[15];
     u1(0.785398163397448) q[0];
```

```
548  cx q[1],q[0];
     cx q[15],q[0];
550  u2(0,3.14159265358979) q[1];
     u2(0,3.14159265358979) q[0];
552  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
554  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
556  cx q[15],q[0];
     u2(0,2.35619449019234) q[15];
558  u2(0,3.14159265358979) q[0];
     cx q[1],q[0];
560  u2(0,3.14159265358979) q[0];
     u2(-0.785398163397448,3.14159265358979) q[1];
562  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
564  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
566  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
568  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
570  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
572  cx q[1],q[0];
     u2(0,3.14159265358979) q[1];
574  barrier q[1];
     u2(0,3.14159265358979) q[1];
576  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
578  cx q[15],q[0];
     u2(0.785398163397448,3.14159265358979) q[0];
580  cx q[1],q[0];
     u1(-0.785398163397448) q[1];
582  u2(0,3.14159265358979) q[0];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
584  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
586  u2(-0.785398163397448,3.14159265358979) q[15];
     u2(0.785398163397448,3.14159265358979) q[3];
588  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
590  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
592  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
594  cx q[2],q[3];
```

```
    cx q[15],q[2];
596 u2(0,3.14159265358979) q[3];
    u2(0,3.14159265358979) q[2];
598 u2(0,3.14159265358979) q[15];
    cx q[15],q[2];
600 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[15];
602 cx q[15],q[2];
    cx q[1],q[2];
604 u2(0.785398163397448,3.14159265358979) q[1];
    cx q[1],q[0];
606 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[1];
608 cx q[1],q[0];
    u2(0,3.14159265358979) q[0];
610 u2(0,3.14159265358979) q[1];
    cx q[1],q[0];
612 cx q[1],q[2];
    cx q[15],q[0];
614 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
616 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
618 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
620 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[2];
622 u2(0,3.14159265358979) q[1];
    cx q[1],q[2];
624 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[1];
626 cx q[1],q[2];
    u2(0,3.14159265358979) q[2];
628 u2(0,3.14159265358979) q[1];
    u2(0,3.14159265358979) q[13];
630 cx q[13],q[14];
    u2(0.785398163397448,3.14159265358979) q[14];
632 u2(0.785398163397448,3.14159265358979) q[13];
    cx q[13],q[14];
634 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[13];
636 cx q[13],q[14];
    u2(0,3.14159265358979) q[14];
638 u2(0,3.14159265358979) q[13];
    cx q[13],q[14];
640 u2(0,3.14159265358979) q[14];
    cx q[3],q[14];
```

```
642  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
644  cx q[3],q[14];
     cx q[13],q[14];
646  u2(0,2.35619449019234) q[3];
     u2(0,2.35619449019234) q[14];
648  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
650  cx q[13],q[14];
     u2(0,3.14159265358979) q[13];
652  barrier q[13];
     u2(0,3.14159265358979) q[13];
654  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
656  cx q[3],q[14];
     u2(-0.785398163397448,3.14159265358979) q[14];
658  cx q[13],q[14];
     u1(-0.785398163397448) q[13];
660  u2(0,3.14159265358979) q[14];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[3];
662  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
664  u2(0.785398163397448,3.14159265358979) q[3];
     cx q[3],q[14];
666  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
668  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
670  u2(0,3.14159265358979) q[3];
     cx q[3],q[14];
672  cx q[13],q[14];
     u2(0.785398163397448,3.14159265358979) q[13];
674  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
676  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
678  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
680  cx q[13],q[14];
     cx q[15],q[14];
682  u2(0,3.14159265358979) q[15];
     cx q[15],q[0];
684  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[15];
686  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
688  u2(0,3.14159265358979) q[15];
```

```
      cx q[15],q[14];
690   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
692   cx q[15],q[14];
      u1(-0.785398163397448) q[14];
694   cx q[15],q[0];
      cx q[15],q[14];
696   u1(0.785398163397448) q[0];
      u1(-0.785398163397448) q[15];
698   u1(-0.785398163397448) q[14];
      cx q[15],q[0];
700   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
702   cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
704   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
706   cx q[15],q[14];
      u2(0,3.14159265358979) q[0];
708   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
710   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
712   cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
714   u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
716   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
718   cx q[15],q[14];
      cx q[15],q[0];
720   u2(0,3.14159265358979) q[15];
      u2(0,3.14159265358979) q[0];
722   cx q[15],q[0];
      cx q[13],q[14];
724   u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
726   cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
728   u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
730   u1(0.785398163397448) q[13];
      u2(0,3.14159265358979) q[0];
732   u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
734   u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
```

```
736  cx q[15],q[0];
     cx q[15],q[14];
738  u1(0.785398163397448) q[15];
     u1(-0.785398163397448) q[14];
740  cx q[3],q[14];
     u1(-0.785398163397448) q[14];
742  u2(0,3.14159265358979) q[0];
     cx q[1],q[0];
744  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[0];
     u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
746  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
748  cx q[15],q[2];
     u2(-0.785398163397448,3.14159265358979) q[1];
750  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
752  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
754  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
756  cx q[2],q[3];
     u1(0.785398163397448) q[3];
758  cx q[3],q[14];
     cx q[2],q[3];
760  u2(0,3.14159265358979) q[2];
     barrier q[2];
762  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
764  cx q[3],q[14];
     u1(-0.785398163397448) q[14];
766  u1(-0.785398163397448) q[3];
     cx q[2],q[3];
768  cx q[3],q[14];
     u1(-0.785398163397448) q[2];
770  u1(-0.785398163397448) q[14];
     u2(0,3.14159265358979) q[15];
772  cx q[15],q[0];
     u2(0,3.14159265358979) q[0];
774  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
776  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
778  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
780  cx q[1],q[0];
     u2(-0.785398163397448,3.14159265358979) q[15];
782  cx q[15],q[0];
```

```
    cx q[1],q[0];
784 u2(0,3.14159265358979) q[15];
    u2(0,3.14159265358979) q[0];
786 u2(0,3.14159265358979) q[1];
    cx q[1],q[0];
788 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[1];
790 cx q[1],q[0];
    u2(0,3.14159265358979) q[1];
792 u2(0,3.14159265358979) q[0];
    cx q[15],q[0];
794 barrier q[0];
    cx q[15],q[0];
796 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
798 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
800 u2(0,3.14159265358979) q[15];
    cx q[15],q[0];
802 u2(0,3.14159265358979) q[0];
    u2(0,3.14159265358979) q[15];
804 cx q[15],q[0];
    u2(0,3.14159265358979) q[0];
806 cx q[1],q[0];
    u2(0.785398163397448,3.14159265358979) q[0];
808 cx q[15],q[0];
    u1(-0.785398163397448) q[15];
810 cx q[15],q[2];
    u2(0,3.14159265358979) q[0];
812 u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
    cx q[1],q[0];
814 u3(0.785398163397448,1.57079632679490,4.71238898038469) q[1];
    u2(0,3.14159265358979) q[2];
816 u2(0,3.14159265358979) q[15];
    cx q[15],q[2];
818 u2(0,3.14159265358979) q[2];
    u2(0,3.14159265358979) q[15];
820 cx q[15],q[2];
    cx q[15],q[14];
822 u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
    cx q[13],q[14];
824 u2(0,3.14159265358979) q[2];
    cx q[1],q[2];
826 u2(0,3.92699081698724) q[2];
    cx q[15],q[2];
828 u2(0,3.14159265358979) q[14];
    u2(0,3.14159265358979) q[13];
```

```
830  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
832  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
834  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[2];
836  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
838  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
840  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
842  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
844  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
846  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
848  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
850  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
852  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
854  cx q[15],q[2];
     cx q[15],q[14];
856  u1(-0.785398163397448) q[2];
     u2(0,3.92699081698724) q[14];
858  cx q[15],q[2];
     u1(-0.785398163397448) q[15];
860  u1(-0.785398163397448) q[2];
     cx q[15],q[2];
862  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
864  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
866  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
868  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
870  u2(0,3.14159265358979) q[14];
     cx q[3],q[14];
872  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[3];
874  cx q[3],q[14];
     u2(0,3.14159265358979) q[14];
876  u2(0,3.14159265358979) q[3];
```

```
     cx q[3],q[14];
878  cx q[2],q[3];
     u1(0.785398163397448) q[3];
880  cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
882  u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
884  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
886  cx q[13],q[14];
     u2(0,3.14159265358979) q[15];
888  cx q[15],q[2];
     u2(0,3.14159265358979) q[15];
890  u2(0,3.14159265358979) q[2];
     cx q[15],q[2];
892  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[15];
894  cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
896  u2(0,3.14159265358979) q[15];
     cx q[15],q[2];
898  cx q[15],q[14];
     u1(0.785398163397448) q[15];
900  u1(0.785398163397448) q[14];
     cx q[13],q[14];
902  cx q[15],q[14];
     u2(0,3.14159265358979) q[2];
904  cx q[1],q[2];
     u2(0.785398163397448,3.14159265358979) q[2];
906  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[1];
     cx q[1],q[0];
908  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
910  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
912  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
914  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
916  cx q[1],q[2];
     u1(-0.785398163397448) q[2];
918  cx q[1],q[0];
     u1(-0.785398163397448) q[0];
920  cx q[1],q[0];
     u2(0,3.14159265358979) q[0];
922  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
```

```
924  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
926  cx q[1],q[0];
     cx q[1],q[2];
928  u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
930  cx q[1],q[0];
     barrier q[0];
932  cx q[1],q[0];
     u2(0,3.14159265358979) q[1];
934  cx q[1],q[2];
     u1(0.785398163397448) q[2];
936  u2(0,2.35619449019234) q[1];
     cx q[1],q[0];
938  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[0];
     u2(0,3.14159265358979) q[1];
940  cx q[1],q[2];
     u1(0.785398163397448) q[2];
942  cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
944  u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
946  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
948  cx q[1],q[2];
     cx q[2],q[3];
950  u2(0,3.14159265358979) q[1];
     cx q[1],q[0];
952  u2(0,3.92699081698724) q[0];
     u2(0,3.14159265358979) q[1];
954  u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
956  cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
958  u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
960  u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[13];
962  u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[15];
964  cx q[15],q[14];
     u2(0,3.14159265358979) q[14];
966  u2(0,3.14159265358979) q[15];
     cx q[15],q[14];
968  u1(0.785398163397448) q[15];
     u2(0,3.14159265358979) q[14];
970  cx q[13],q[14];
```

```
       u2(0,3.14159265358979) q[14];
972    cx q[15],q[14];
       u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[13];
974    u2(0,3.14159265358979) q[14];
       u2(0,3.14159265358979) q[15];
976    cx q[15],q[14];
       u2(0,3.14159265358979) q[14];
978    u2(0,3.14159265358979) q[15];
       cx q[15],q[14];
980    u2(0,3.14159265358979) q[14];
       cx q[13],q[14];
982    u2(0,3.14159265358979) q[14];
       u2(0,3.14159265358979) q[13];
984    cx q[13],q[14];
       cx q[15],q[14];
986    u2(0,3.14159265358979) q[15];
       barrier q[15];
988    u2(0,3.14159265358979) q[15];
       cx q[15],q[14];
990    u2(0,3.14159265358979) q[13];
       u2(0,3.14159265358979) q[14];
992    cx q[13],q[14];
       u2(0.785398163397448,3.14159265358979) q[14];
994    cx q[15],q[14];
       u1(-0.785398163397448) q[15];
996    u2(0,3.14159265358979) q[14];
       u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[13];
998    cx q[13],q[14];
       u2(0,3.14159265358979) q[14];
1000   cx q[15],q[14];
       u2(0,3.14159265358979) q[14];
1002   u2(0,3.14159265358979) q[15];
       cx q[15],q[14];
1004   u2(0,3.14159265358979) q[14];
       u2(0,3.14159265358979) q[15];
1006   cx q[15],q[14];
       u2(0,3.14159265358979) q[14];
1008   u3(0.785398163397448,1.57079632679490,4.71238898038469) q[13];
       cx q[13],q[14];
1010   u1(0.785398163397448) q[14];
       cx q[15],q[14];
1012   u2(0,3.14159265358979) q[14];
       u2(0,3.14159265358979) q[15];
1014   cx q[15],q[14];
       u2(0,3.14159265358979) q[14];
1016   u2(0,3.14159265358979) q[15];
       cx q[15],q[14];
```

```
1018  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1020  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1022  cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
1024  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1026  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1028  cx q[15],q[0];
      cx q[1],q[0];
1030  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1032  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[15];
      u2(0,3.14159265358979) q[0];
1034  u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
1036  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[1];
1038  cx q[1],q[0];
      u2(0,3.14159265358979) q[1];
1040  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1042  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1044  u2(0,3.92699081698724) q[1];
      u2(0,3.14159265358979) q[2];
1046  cx q[15],q[2];
      u2(-0.785398163397448,3.14159265358979) q[2];
1048  u2(-0.785398163397448,3.14159265358979) q[15];
      cx q[15],q[2];
1050  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[15];
1052  cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
1054  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1056  u2(0,3.14159265358979) q[2];
      cx q[1],q[2];
1058  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1060  cx q[1],q[2];
      cx q[15],q[2];
1062  cx q[1],q[2];
      u2(0,3.14159265358979) q[15];
1064  u2(0,3.14159265358979) q[2];
```

```
     u2(0,3.14159265358979) q[1];
1066 cx  q[1],q[2];
     u2(0,3.14159265358979) q[2];
1068 u2(0,3.14159265358979) q[1];
     cx  q[1],q[2];
1070 u2(0,3.14159265358979) q[2];
     cx  q[15],q[2];
1072 cx  q[1],q[2];
     u2(0,3.14159265358979) q[2];
1074 u2(0,3.14159265358979) q[1];
     cx  q[1],q[2];
1076 u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
1078 cx  q[1],q[2];
     u1(0.785398163397448) q[2];
1080 cx  q[1],q[0];
     u1(0.785398163397448) q[1];
1082 u1(-0.785398163397448) q[0];
     cx  q[1],q[0];
1084 cx  q[2],q[3];
     u2(0,3.14159265358979) q[3];
1086 u2(0,3.14159265358979) q[2];
     cx  q[2],q[3];
1088 u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
1090 cx  q[2],q[3];
     u2(0,3.14159265358979) q[2];
1092 u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
1094 cx  q[1],q[0];
     u2(0,3.14159265358979) q[0];
1096 u2(0,3.14159265358979) q[1];
     cx  q[1],q[0];
1098 u2(0,3.14159265358979) q[1];
     cx  q[1],q[2];
1100 u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
1102 cx  q[1],q[0];
     u2(0,3.14159265358979) q[0];
1104 u2(0,3.14159265358979) q[1];
     cx  q[1],q[0];
1106 u2(0,3.14159265358979) q[0];
     u2(0,3.14159265358979) q[1];
1108 cx  q[1],q[0];
     cx  q[1],q[2];
1110 u1(-0.785398163397448) q[0];
     u1(0.785398163397448) q[2];
```

```
1112  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1114  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1116  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1118  cx q[1],q[2];
      cx q[1],q[0];
1120  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1122  cx q[1],q[2];
      barrier q[2];
1124  cx q[1],q[2];
      u2(0,3.14159265358979) q[1];
1126  cx q[1],q[0];
      u2(0,2.35619449019234) q[1];
1128  u1(0.785398163397448) q[0];
      cx q[1],q[2];
1130  u2(-0.785398163397448,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1132  cx q[1],q[0];
      u1(0.785398163397448) q[0];
1134  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1136  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1138  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1140  cx q[1],q[2];
      cx q[1],q[0];
1142  u2(0.785398163397448,3.14159265358979) q[1];
      cx q[1],q[2];
1144  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[1];
1146  cx q[1],q[2];
      u2(0,3.14159265358979) q[2];
1148  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1150  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1152  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
1154  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1156  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1158  cx q[15],q[14];
```

```
      u1(0.785398163397448) q[15];
1160  u1(0.785398163397448) q[14];
      cx q[13],q[14];
1162  cx q[15],q[14];
      u2(0,3.14159265358979) q[13];
1164  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
1166  cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
1168  u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
1170  u2(0,2.35619449019234) q[15];
      u2(0,3.14159265358979) q[14];
1172  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1174  u2(-0.785398163397448,3.14159265358979) q[13];
      cx q[13],q[14];
1176  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
1178  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1180  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1182  u2(0,3.14159265358979) q[14];
      cx q[15],q[14];
1184  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1186  u2(0,3.14159265358979) q[13];
      barrier q[13];
1188  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1190  u2(0,3.14159265358979) q[14];
      cx q[15],q[14];
1192  u2(-0.785398163397448,3.14159265358979) q[14];
      cx q[13],q[14];
1194  u1(-0.785398163397448) q[13];
      u2(0,3.14159265358979) q[14];
1196  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
      cx q[15],q[14];
1198  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1200  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[13];
1202  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1204  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
```

```
1206   u2(0,3.14159265358979) q[14];
       u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[15];
1208   cx q[15],q[14];
       u1(0.785398163397448) q[14];
1210   cx q[3],q[14];
       u2(0,3.14159265358979) q[14];
1212   u2(0,3.14159265358979) q[3];
       cx q[3],q[14];
1214   u2(0,3.14159265358979) q[14];
       u2(0,3.14159265358979) q[3];
1216   cx q[3],q[14];
       cx q[2],q[3];
1218   u2(0,3.14159265358979) q[2];
       u2(0,3.14159265358979) q[3];
1220   u2(0,3.14159265358979) q[15];
       cx q[15],q[14];
1222   u2(0,3.14159265358979) q[14];
       u2(0,3.14159265358979) q[15];
1224   cx q[15],q[14];
       u2(0,3.14159265358979) q[14];
1226   u2(0,3.14159265358979) q[15];
       cx q[15],q[14];
1228   u2(0,3.14159265358979) q[15];
       cx q[15],q[2];
1230   u2(0,3.14159265358979) q[2];
       u2(0,3.14159265358979) q[15];
1232   cx q[15],q[2];
       u2(0,2.35619449019234) q[15];
1234   u2(0,3.14159265358979) q[2];
       cx q[2],q[3];
1236   u2(0,3.14159265358979) q[3];
       u2(0,3.14159265358979) q[2];
1238   cx q[2],q[3];
       u2(0,3.14159265358979) q[3];
1240   u2(0,3.14159265358979) q[2];
       cx q[2],q[3];
1242   u2(0,3.14159265358979) q[3];
       u2(0,3.14159265358979) q[2];
1244   cx q[2],q[3];
       u1(0.785398163397448) q[3];
1246   u2(0,3.14159265358979) q[2];
       cx q[15],q[2];
1248   u2(-0.785398163397448,3.14159265358979) q[2];
       cx q[2],q[3];
1250   u2(0,3.14159265358979) q[3];
       u2(0,3.14159265358979) q[2];
1252   cx q[2],q[3];
```

```
     u2 (0 ,3.14159265358979) q[3];
1254 u2 (0 ,3.14159265358979) q[2];
     cx q[2] ,q[3];
1256 u2 (0 ,3.14159265358979) q[3];
     u2 (0 ,3.14159265358979) q[2];
1258 u3 (0.785398163397448 ,1.57079632679490 ,4.71238898038469) q[15];
     cx q[15] ,q[2];
1260 cx q[2] ,q[3];
     u2 (0 ,3.14159265358979) q[3];
1262 cx q[3] ,q[14];
     u2 (0 ,3.14159265358979) q[14];
1264 u2 (0 ,3.14159265358979) q[3];
     cx q[3] ,q[14];
1266 u2 (0 ,3.14159265358979) q[14];
     u2 (0 ,3.14159265358979) q[3];
1268 cx q[3] ,q[14];
     u2 (0 ,3.14159265358979) q[3];
1270 u2 (0.785398163397448 ,3.14159265358979) q[2];
     u2 (0 ,3.14159265358979) q[15];
1272 cx q[15] ,q[14];
     u2 (0 ,3.14159265358979) q[15];
1274 cx q[15] ,q[0];
     u1 (0.785398163397448) q[15];
1276 u1 (-0.785398163397448) q[0];
     cx q[1] ,q[0];
1278 u1 (-0.785398163397448) q[0];
     cx q[15] ,q[2];
1280 cx q[3] ,q[14];
     u2 (0.785398163397448 ,3.14159265358979) q[14];
1282 u2 (-0.785398163397448 ,3.14159265358979) q[3];
     cx q[3] ,q[14];
1284 u2 (0 ,3.14159265358979) q[14];
     u2 (0 ,3.14159265358979) q[3];
1286 cx q[3] ,q[14];
     u2 (0 ,3.14159265358979) q[14];
1288 u2 (0 ,3.14159265358979) q[3];
     cx q[3] ,q[14];
1290 cx q[13] ,q[14];
     cx q[3] ,q[14];
1292 u2 (0 ,3.14159265358979) q[13];
     u2 (0 ,3.14159265358979) q[14];
1294 u2 (0 ,3.14159265358979) q[3];
     cx q[3] ,q[14];
1296 u2 (0 ,3.14159265358979) q[14];
     u2 (0 ,3.14159265358979) q[3];
1298 cx q[3] ,q[14];
     u1 (-0.785398163397448) q[3];
```

```
1300  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1302  u2(0,3.14159265358979) q[14];
      cx q[3],q[14];
1304  u3(-0.785398163397448,1.57079632679490,4.71238898038469) q[13];
      u2(0,3.14159265358979) q[14];
1306  u2(0,3.14159265358979) q[3];
      cx q[3],q[14];
1308  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[3];
1310  cx q[3],q[14];
      u2(0,3.14159265358979) q[14];
1312  cx q[13],q[14];
      u2(0,3.14159265358979) q[14];
1314  u2(0,3.14159265358979) q[13];
      cx q[13],q[14];
1316  cx q[3],q[14];
      u2(0,3.14159265358979) q[3];
1318  barrier q[3];
      u2(0,3.14159265358979) q[3];
1320  cx q[3],q[14];
      u2(0,3.14159265358979) q[13];
1322  u2(0,3.14159265358979) q[14];
      cx q[13],q[14];
1324  u2(-0.785398163397448,3.14159265358979) q[14];
      cx q[3],q[14];
1326  u1(-0.785398163397448) q[3];
      u2(0,3.14159265358979) q[14];
1328  u3(0.785398163397448,1.57079632679490,4.71238898038469) q[13];
      cx q[13],q[14];
1330  u2(0,3.14159265358979) q[14];
      cx q[3],q[14];
1332  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[3];
1334  cx q[3],q[14];
      u2(0,3.14159265358979) q[14];
1336  u2(0,3.14159265358979) q[3];
      cx q[3],q[14];
1338  u2(0,3.14159265358979) q[14];
      u3(0.785398163397448,1.57079632679490,4.71238898038469) q[13];
1340  cx q[13],q[14];
      u1(0.785398163397448) q[14];
1342  u2(0,3.14159265358979) q[13];
      u2(0,3.14159265358979) q[1];
1344  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[15];
1346  cx q[15],q[2];
```

```
      u2(0,3.14159265358979) q[2];
1348  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1350  u2(0,3.14159265358979) q[2];
      cx q[1],q[2];
1352  u2(0.785398163397448,3.14159265358979) q[1];
      cx q[1],q[0];
1354  u2(0,3.14159265358979) q[1];
      cx q[1],q[2];
1356  barrier q[2];
      cx q[1],q[2];
1358  u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
1360  u2(0,2.35619449019234) q[1];
      u1(0.785398163397448) q[0];
1362  cx q[1],q[2];
      u2(-0.785398163397448,3.14159265358979) q[2];
1364  cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
1366  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1368  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[15];
1370  cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
1372  u2(0,3.14159265358979) q[1];
      cx q[1],q[0];
1374  u1(0.785398163397448) q[0];
      cx q[15],q[0];
1376  u2(0.785398163397448,3.14159265358979) q[15];
      cx q[15],q[14];
1378  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
1380  cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
1382  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1384  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[15];
1386  cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
1388  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1390  u2(0,2.35619449019234) q[2];
      u2(0,3.14159265358979) q[15];
1392  cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
```

```
1394  cx q[3],q[14];
      u2(0,3.14159265358979) q[14];
1396  u2(0,3.14159265358979) q[3];
      cx q[3],q[14];
1398  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[3];
1400  cx q[3],q[14];
      u2(0,3.14159265358979) q[3];
1402  cx q[2],q[3];
      u2(0.785398163397448,3.14159265358979) q[15];
1404  u2(-0.785398163397448,3.14159265358979) q[3];
      u2(-0.785398163397448,3.14159265358979) q[2];
1406  cx q[15],q[2];
      cx q[15],q[14];
1408  u2(0,3.14159265358979) q[14];
      u2(0,3.14159265358979) q[15];
1410  cx q[15],q[14];
      u2(0,3.14159265358979) q[14];
1412  u2(0,3.14159265358979) q[15];
      cx q[15],q[14];
1414  cx q[3],q[14];
      u2(0,3.14159265358979) q[14];
1416  cx q[2],q[3];
      barrier q[14];
1418  u2(0,3.14159265358979) q[3];
      u2(0,3.14159265358979) q[2];
1420  cx q[15],q[2];
      cx q[13],q[14];
1422  u2(0,3.14159265358979) q[2];
      u2(0,3.14159265358979) q[15];
1424  cx q[15],q[2];
      u2(0,3.14159265358979) q[2];
1426  u2(0,3.14159265358979) q[15];
      cx q[15],q[2];
1428  cx q[15],q[0];
      u1(0.785398163397448) q[15];
1430  u1(-0.785398163397448) q[0];
      cx q[1],q[0];
1432  cx q[15],q[0];
      u2(0,3.14159265358979) q[1];
1434  u2(0,3.14159265358979) q[0];
      u2(0,3.14159265358979) q[15];
1436  cx q[15],q[0];
      u2(0,3.14159265358979) q[0];
1438  u2(0,3.14159265358979) q[15];
      cx q[15],q[0];
1440  u2(0,2.35619449019234) q[15];
```

```
     u2(0,3.14159265358979) q[0];
1442 cx q[1],q[0];
     u2(0.785398163397448,3.14159265358979) q[1];
1444 u2(0,3.14159265358979) q[14];
     u2(0,3.14159265358979) q[13];
1446 cx q[13],q[14];
     u2(0,3.14159265358979) q[14];
1448 u2(0,3.14159265358979) q[13];
     cx q[13],q[14];
1450 measure q[13] -> cr[2];
     cx q[3],q[14];
1452 u1(0.785398163397448) q[14];
     u1(0.785398163397448) q[3];
1454 cx q[2],q[3];
     u2(0,3.14159265358979) q[3];
1456 u2(0,3.14159265358979) q[2];
     cx q[2],q[3];
1458 u2(0,3.14159265358979) q[3];
     u2(0,3.14159265358979) q[2];
1460 cx q[2],q[3];
     cx q[3],q[14];
1462 u1(0.785398163397448) q[14];
     cx q[2],q[3];
1464 u1(0.785398163397448) q[3];
     cx q[3],q[14];
1466 barrier q[14];
     measure q[14] -> cr[3];
1468 cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
1470 u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
1472 u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
1474 cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
1476 cx q[15],q[2];
     u2(0,3.14159265358979) q[2];
1478 cx q[1],q[2];
     u2(0,3.14159265358979) q[2];
1480 u2(0,3.14159265358979) q[1];
     cx q[1],q[2];
1482 u2(0,3.14159265358979) q[2];
     u2(0,3.14159265358979) q[1];
1484 cx q[1],q[2];
     cx q[2],q[3];
1486 barrier q[3];
     u2(0,3.14159265358979) q[2];
```
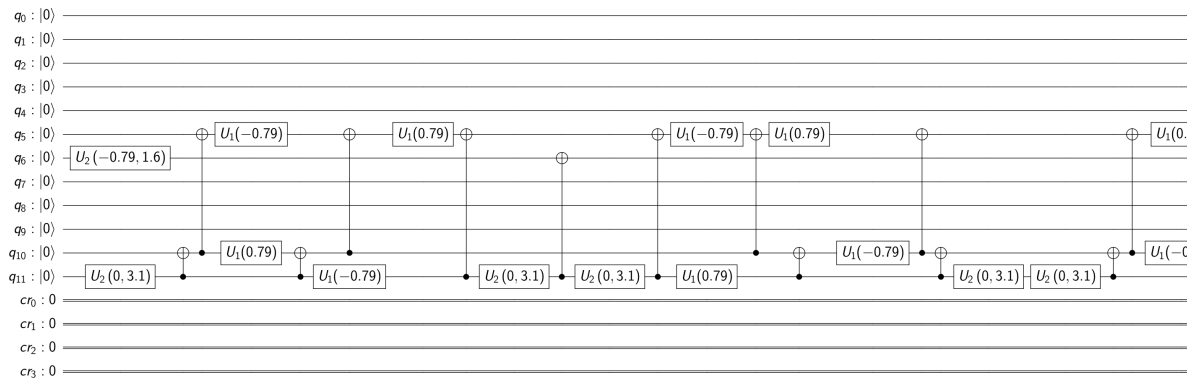
Figure E.1.: The circuit for-loop quantamorphism over Hadamard gate when running in backend 'ibmq_20_tokyo' (section 1).
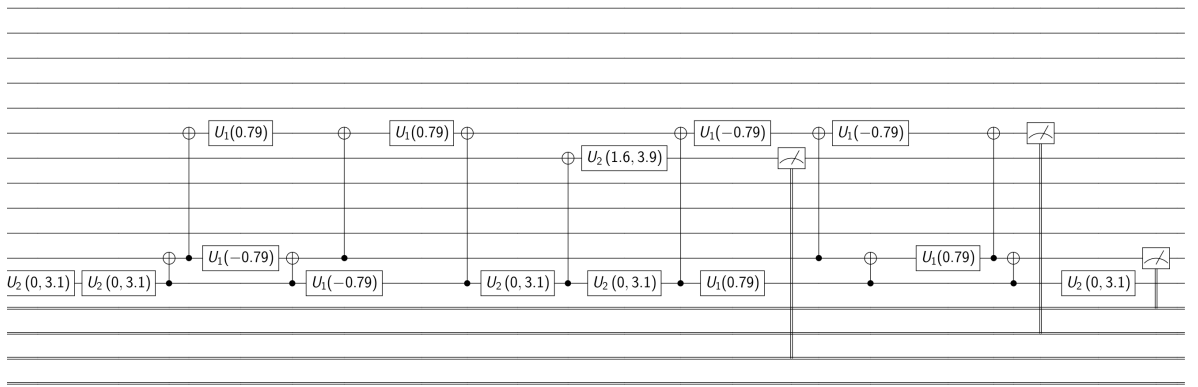

Figure E.2.: The circuit for-loop quantamorphism over Hadamard gate when running in backend 'ibmq_20_tokyo' (section 2).

```
1488  measure q[3] -> cr[4];
      u2(0,3.14159265358979) q[1];
1490  cx q[1],q[0];
      u2(0,3.14159265358979) q[1];
1492  barrier q[1];
      measure q[1] -> cr[6];
1494  u2(0,3.14159265358979) q[15];
      barrier q[15];
1496  measure q[15] -> cr[5];
```

Listing E.16: QASM of adapted circuit fold quantamorphism over XOR gate after running in quantum device.

RUNNING CIRCUITS IN `IBMQ_20_TOKYO`   The differences in this outputs are exclusively to running the program in a real device.

```
OPENQASM 2.0;
```

```
2   include "qelib1.inc";
    qreg q[12];
4   creg cr[4];
    u2(0,3.14159265358979) q[11];
6   u2(-0.785398163397448,1.57079632679490) q[6];
    cx q[11],q[10];
8   cx q[10],q[5];
    u1(0.785398163397448) q[10];
10  u1(-0.785398163397448) q[5];
    cx q[11],q[10];
12  u1(-0.785398163397448) q[11];
    cx q[10],q[5];
14  u1(0.785398163397448) q[5];
    cx q[11],q[5];
16  u2(0,3.14159265358979) q[11];
    cx q[11],q[6];
18  u2(0,3.14159265358979) q[11];
    cx q[11],q[5];
20  u1(0.785398163397448) q[11];
    u1(-0.785398163397448) q[5];
22  cx q[10],q[5];
    cx q[11],q[10];
24  u1(0.785398163397448) q[5];
    u1(-0.785398163397448) q[10];
26  cx q[10],q[5];
    cx q[11],q[10];
28  u2(0,3.14159265358979) q[11];
    barrier q[11];
30  u2(0,3.14159265358979) q[11];
    cx q[11],q[10];
32  cx q[10],q[5];
    u1(-0.785398163397448) q[10];
34  u1(0.785398163397448) q[5];
    cx q[11],q[10];
36  u1(-0.785398163397448) q[11];
    cx q[10],q[5];
38  u1(0.785398163397448) q[5];
    cx q[11],q[5];
40  u2(0,3.14159265358979) q[11];
    cx q[11],q[6];
42  u2(0,3.14159265358979) q[11];
    u2(1.57079632679490,3.92699081698724) q[6];
44  cx q[11],q[5];
    u1(0.785398163397448) q[11];
46  u1(-0.785398163397448) q[5];
    measure q[6] -> cr[2];
48  cx q[10],q[5];
```

```
   cx q[11],q[10];
50 u1(-0.785398163397448) q[5];
   u1(0.785398163397448) q[10];
52 cx q[10],q[5];
   cx q[11],q[10];
54 measure q[5] -> cr[1];
   u2(0,3.14159265358979) q[11];
56 measure q[10] -> cr[0];
```

Listing E.17: QASM of circuit for-loop quantamorphism over Hadamard gate after running in quantum device.

```
   {'counts': {'0000': 378,
2    '0001': 60,
     '0010': 78,
4    '0011': 52,
     '0100': 284,
6    '0101': 55,
     '0110': 66,
8    '0111': 51},
    'date': '2018-09-06T10:19:13.099Z',
10   'time': 15}
```

Listing E.18: Results of running circuit for-loop quantamorphism over Hadamard gate in `ibmq_20_tokyo` with no state preparation.

```
   {'counts': {'0000': 67,
2    '0001': 86,
     '0010': 64,
4    '0011': 206,
     '0100': 51,
6    '0101': 115,
     '0110': 121,
8    '0111': 314},
    'date': '2018-09-06T10:12:13.922Z',
10   'time': 15}
```
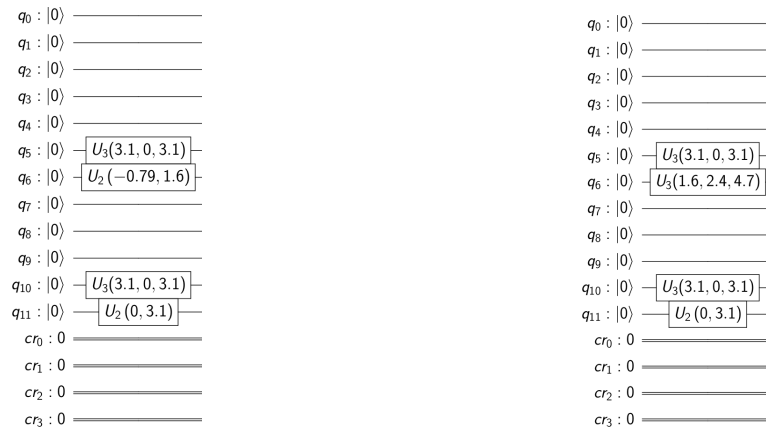
Listing E.19: Results of running circuit for-loop quantamorphism over Hadamard gate in `ibmq_20_tokyo` with the state preparation E.3a.
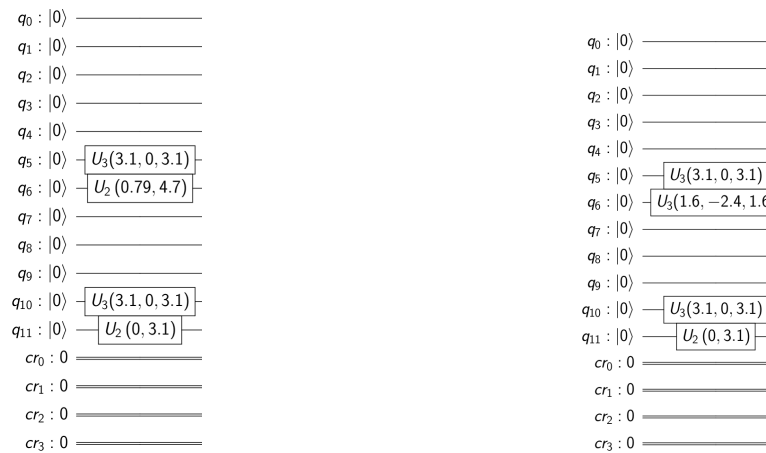
```
   {'counts': {'0000': 50,
2    '0001': 95,
     '0010': 78,
4    '0011': 328,
     '0100': 43,
6    '0101': 98,
```

(a) Gates to prepare the circuit with $q_0 = 1$, $q_1 = 1$ and $q_2 = 0$.

(b) Gates to prepare the circuit with $q_0 = 1$, $q_1 = 1$ and $q_2 = 1$.

(c) Gates to prepare the circuit with $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$.

(d) Gates to prepare the circuit with $q_0 = 1$, $q_1 = 1$ and $q_2 = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$.

Figure E.3.: State preparations.

```
     '0110': 73,
 8    '0111': 259},
     'date': '2018-08-30T18:01:51.973Z',
10   'time': 15}
```

Listing E.20: Results of running circuit for-loop quantamorphism over Hadamard gate in ibmq_20_tokyo with the state preparation E.3b.

```
   {'counts': {'0000': 46,
 2    '0001': 112,
      '0010': 86,
 4    '0011': 282,
      '0100': 58,
 6    '0101': 111,
      '0110': 105,
 8    '0111': 224},
     'date': '2018-09-06T10:07:37.865Z',
10    'time': 15}
```

Listing E.21: Results of running circuit for-loop quantamorphism over Hadamard gate in ibmq_20_tokyo with the state preparation E.3c.

```
   {'counts': {'0000': 79,
 2    '0010': 235,
      '0100': 53,
 4    '0110': 265,
      '1000': 66,
 6    '1010': 118,
      '1100': 69,
 8    '1110': 139},
     'date': '2018-09-06T10:14:42.666Z',
10    'time': 15}
```

Listing E.22: Results of running circuit for-loop quantamorphism over Hadamard gate in ibmq_20_tokyo with the state preparation E.3d.