SPRINGER BRIEFS IN PHYSICS

Philipp Niemann
Robert Wille

Compact Representations for the Design of Quantum Logic

Springer

SpringerBriefs in Physics

More information about this series at http://www.springer.com/series/8902

Philipp Niemann · Robert Wille

Compact Representations for the Design of Quantum Logic

Springer

Philipp Niemann
Department for Cyber-Physical Systems
German Research Center for Artificial
Intelligence (DFKI)
Bremen
Germany

Robert Wille
Institute for Integrated Circuits
Johannes Kepler University Linz
Linz
Austria

ISSN 2191-5423 ISSN 2191-5431 (electronic)
SpringerBriefs in Physics
ISBN 978-3-319-63723-5 ISBN 978-3-319-63724-2 (eBook)
DOI 10.1007/978-3-319-63724-2

Library of Congress Control Number: 2017947865

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Quantum computation provides a new way of computation based on so-called quantum bits (qubits). In contrast to the conventional bits used in Boolean logic, qubits do not only allow to represent the basis states 0 and 1, but also superpositions of both. By this, qubits can represent multiple states at the same time which enables massive parallelism. Additionally, exploiting further quantum-mechanical phenomena such as phase shifts or entanglement allows for the design of algorithms with an asymptotic speed-up for many relevant computational problems such database search, integer factorization, or graph and algebraic problems.

Motivated by these prospects, researchers from various domains investigated this emerging technology. While, originally, the exploitation of quantum-mechanical phenomena has been discussed in a purely theoretical fashion, in the past decade also the consideration of physical realizations has gained significant interest. In fact, building a quantum computer is a very challenging task. Nonetheless and despite the severe physical challenges, suitable technologies for the modelling and design of powerful, large-scale quantum computers composed of dozens of usable qubits are in the range of vision.

However, for quantum systems of larger size, the manual design of quantum circuits which realize a given (quantum) functionality in technology is no longer an option. In order to keep up with the technological progress, methods need to be provided which, similar to the design and synthesis of conventional circuits, automatically generate a circuit description of the desired (quantum) functionality—finally leading to *Computer-Aided Design* (CAD) of quantum circuits.

But in order to formalize the quantum-mechanical phenomena, states of qubits are modelled as vectors in high-dimensional Hilbert spaces and are manipulated by quantum operations which are described by unitary matrices—possibly including complex numbers. This poses serious challenges to the representation, but even more to the development of proper and efficient methods for quantum circuit synthesis that would scale to quantum systems of considerable size. Hence, an efficient representation of the desired quantum functionality is of the essence.

In this book, we provide a comprehensive analysis of the challenges for a compact representation of quantum functionality, extensively review the state-of-the-art in this area as well as describe in detail a proper solution in terms of *Quantum Multiple-Valued Decision Diagrams* (QMDDs). Based on QMDDs, we demonstrate applications where this compact representation can be exploited in order to develop scalable and automated methods for quantum circuit synthesis and verification. The book is written in order to be comprehensible for computer scientists as well as computer architects with little background in quantum logic and, thus, does not require previous knowledge about the underlying physics. The respective concepts as well as the used models are briefly introduced. All approaches are described in a self-contained manner. The content of the book does not only convey a coherent overview about current research results but also build the basis for future work on quantum logic CAD.

This book is the result of several years of intensive research in the area of quantum logic at the University of Bremen, Germany, and the Johannes Kepler University Linz, Austria. During this time, we experienced broad support from many people for which we would like to thank them very much.

Most importantly, we would like to thank Prof. Dr. *Rolf Drechsler* for his continuous support in all these years and for providing a comfortable and inspirational environment from which both authors benefit until today. We are also indebted to thank Prof. *D. Michael Miller* from the University of Victoria, Canada —one of the "fathers" of QMDDs—for many fruitful collaborations which laid the foundation of his book. The numerous inspiring discussions, the helpful comments and suggestions, and the thorough reading and review of this manuscript made this book possible. In this context, the German Academic Exchange Service (DAAD) earns our thanks for funding the close contact with Prof. Miller's group. Beyond that, we are very thankful for the support by the European Union through the COST Action IC1405.

Many thanks also go to Prof. *Mitch Thornton* (Southern Methodist University, Texas, USA) and *Alwin Zulehner* (Johannes Kepler University Linz, Austria) for the very productive collaboration that resulted in some research papers which form the basis of this book.

Finally, we would like to thank the many people who contributed in further, more or less indirect ways to the completion of this book: In particular, the whole Group for Computer Architecture at the University of Bremen as well as the Institute for Integrated Circuits at the Johannes Kepler University earn many thanks for providing a comfortable and inspirational environment. Particularly, very special thanks go to *Oliver Keszöcze* with whom the first author of this book does not only share the same wavelength, but also a rather green office. Finally, we would like to thank *Arne Grenzebach* for proofreading the manuscript as well as for establishing the contact to *Sabine Lehr* (Associate Editor at Springer).

Bremen, Germany Philipp Niemann
Linz, Austria Robert Wille
June 2017

Contents

Part I Introduction and Background

Part II Representation of Quantum Functionality

Part III Design of Quantum Logic

Part I
Introduction and Background

Chapter 1
Introduction

Quantum computation [NC00] provides a new way of computation based on so-called *quantum bits* (qubits). In contrast to the conventional bits used in Boolean logic, qubits do not only allow to represent the basis states 0 and 1, but also superpositions of both. By this, qubits can represent multiple states at the same time which enables massive parallelism. Additionally exploiting further quantum-mechanical phenomena such as phase shifts or entanglement allows for the design of algorithms with an asymptotic speed-up for many relevant problems (e.g. database search [Gro96] or integer factorization [Sho94]). In other words: quantum computers, i.e. physical devices that operate at the atomic level and are able to exploit the above phenomena, promise to solve important computational problems significantly faster than will ever be possible on conventional computers. In addition, quantum logic offers new methods for secure communication (e.g. quantum key distribution) and has several other appealing applications.

Motivated by these prospects and by the fact that recent developments in conventional computing technologies are approaching the level of atoms anyway, researchers from various domains have investigated this emerging technology. While, originally, the exploitation of quantum-mechanical phenomena has been discussed in a purely theoretical fashion (see e.g. [Deu85, Sho94, Gro96] for three well-known quantum algorithms), in the past decade also the consideration of physical realizations (see e.g. [Gal07, OFV09, MSB+11]) has gained significant interest.

In fact, building a quantum computer is a very challenging task. As severe physical obstacles have to be overcome, the devices proposed and used for quantum computing so far are very prototypical in nature, operate on quantum systems consisting of few particles, and have a low computational power. Consequently, until now it was sufficient and easily possible to manually design the "programs" that realize the desired (quantum) functionality on these devices. However, suitable technologies for the modelling and design of powerful, large-scale quantum computers and other information processing techniques that exploit quantum-mechanical principles are

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_1

in the range of vision. As a consequence, the manual design of quantum logic will no longer be a feasible option in the future.

In order to keep up with this technological progress, there is a need to develop methods that allow for *Computer-Aided Design* (CAD), i.e. methods that (1) *automatically* generate an executable description of the desired quantum functionality, (2) take into account the physical constraints of the target technology, and (3) scale to quantum systems of considerable size.

In this book, we provide substantial improvements to the state-of-the-art in quantum logic design, especially with respect to and based on the efficient representation of quantum functionality. The presented approaches overcome several drawbacks and limitations of previously proposed approaches and provide scalable, technology-aware, and automated solutions for important sub-tasks of quantum logic design. To this end, they constitute a major step towards CAD for quantum logic. A more detailed overview of the topics that are covered in this book is provided in Sect. 1.3.

Before that, we provide a comprehensive introduction to the field. More precisely, Sect. 1.1 first reviews the historic developments from the "birth" of quantum mechanics to the evolution of a quantum logic. Based on that, the exploitation of quantum-mechanical effects for quantum computing as well as the corresponding design challenges are discussed in Sect. 1.2.

1.1 Quantum Mechanics and Quantum Logic

When the seminal works of Max Planck [Pla00] and Albert Einstein [Ein05] established *quantum mechanics* as a new, fundamental branch of physics at the beginning of the last century, no one was actually thinking of exploiting the corresponding phenomena and effects for something like quantum computing, i.e. in order to obtain solutions for classical computation problems. In fact, not even the foundations of classical computation were laid at that time. Initially, the discovered phenomena and implied predictions were subject to highly controversial discussions. To name a few of them:

- **Uncertainty Principle**: Particles have particular properties (such as position, momentum, spin, or polarization) that are not determined exactly. When these are measured, the result is rather drawn randomly from a probability distribution. Moreover, complementary properties like position and momentum can not be known precisely at the same time, i.e. the more precisely a particle's position is determined, the less precisely its momentum can be known.
- **Quantum Entanglement**: A group of particles can be connected to each other in such a way that manipulations (measurements) of one of them has an effect on the whole system. More precisely, any other of these particles might be affected—even when separated by arbitrarily large distances. Einstein called this phenomenon "spooky action at a distance".

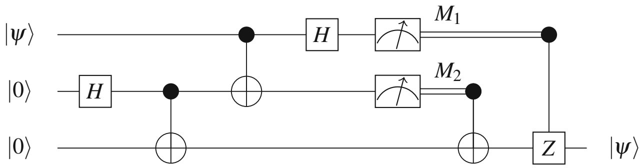


Fig. 1.1 Visualization of a quantum logic "formula" (cf. [NC00, p. 27]).

- **Quantum Superposition**: Particles can be in different (energy) states at the same time. In other words, any two quantum states can be added together ("superposed") and the result will be another valid state.

While these counter-intuitive predictions of quantum mechanics could be experimentally verified after all, for the time being numerous mathematically equivalent formulations of quantum mechanics were developed e.g. by Heisenberg, Schrödinger, Dirac, von Neumann, and Hilbert. The first complete formulation, known as the *Dirac—von Neumann* axioms, was presented in 1932 [Neu32]. It is still widely used today and also employed in this book. In this formulation, the state of a quantum system—usually denoted as $|\psi\rangle$ using Dirac's *ket*-notation [Dir39]—is represented by a unit vector in a *high-dimensional Hilbert space*, i.e. a special kind of vector-space over the complex numbers. This state is manipulated by quantum operations that can be identified with generalized rotations of this space which preserve lengths and angles—so-called *unitary operators*. The latter have a representation as complex-valued unitary matrices which allow the manipulation of a quantum state to be interpreted as a conventional matrix-vector multiplication.

From this abstract picture, however, it is still far from obvious to use quantum mechanics for computation purposes. An important step on this way is to consider quantum mechanics from a logic perspective, i.e. constructing and evaluating formulas consisting of variables, logical connectives, predicates, and/or quantifiers. In fact, it should take some decades to develop a logic model for quantum mechanics.[1] In this model, quantum systems are interpreted as ensembles of qubits (or multiple-valued quantum digits, so-called *qudits*) and complex quantum operations as a cascade of elementary operations—so-called *quantum gates*—that are applied to a small subset of qubits only rather than the entire system.[2]

Example 1.1 Consider the quantum logic "formula" shown in Fig. 1.1. This kind of a visualization has been termed a *quantum gate array* by Deutsch [Deu89] and later became popular under the name *quantum circuit*—in analogy to the circuit representation of Boolean logic. However, the horizontal lines do not correspond

[1] Actually, the term *quantum logic* was initially used by Birkhoff and von Neumann for investigations about logical anomalies of propositional logic that occur when reasoning about measurements of quantum systems—mostly related with the uncertainty principle [BN36].

[2] Indeed, the term *qubit* first occurred in the 1990s and is credited to Ben Schumacher and William Wootters [Sch95].

to actual circuit wires, but rather represent the qubits of the considered quantum system to which the quantum gates $\boxed{H}$, ●–⊕, etc. as well as measurements $\boxed{\nearrow}$ are applied in a chronological order. Many quantum gates are conditional gates (like the controlled NOT gate ●–⊕) that are only applied to the target qubit if all controls (denoted by solid dots ●) are in the appropriate state.

For the purpose of this introductory chapter, it is completely sufficient to understand the general semantics of quantum logic "formulas" or circuits, namely that they describe a time-dependent process rather than some physical layout. However, note that the circuit in Fig. 1.1 is not just an arbitrary sample circuit. It rather exploits the interesting quantum-mechanical phenomenon of quantum entanglement and, thus, allows for *teleporting* qubit states over potentially large distances. For this reason, we will return to it later in Example 2.9 on page 17 when we have introduced sufficient background to understand the details of this amazing functionality.

For the time being, we continue by observing that the measurements used in quantum logic are non-deterministic. In fact, recall that it is a general restriction of quantum mechanics that a precise read-out of the state of a quantum system is not possible anyway and the result of measurements is rather drawn randomly from a probability distribution. These non-deterministic measurements play an important role in quantum logic. On the one hand, they allow one to build non-deterministic formulas by measuring individual qubits and using the result to control subsequent gates (as in Example 1.1). On the other hand, they offer the opportunity to evaluate quantum logic "formulas" in terms of traditional logic (resulting in tuples of 0 and 1). These potentially non-deterministic results are essentially the basis for quantum computation.

1.2 Quantum Computation and Circuit Design

With the quantum logic model sketched above, it becomes possible to actually perform quantum computing, i.e. to employ quantum-mechanical effects in order to obtain solutions for classical computation problems. However, the model is fundamentally different from that used in classical computing and, for a long time, there has been only little interest in this area. Researchers were heavily busy with the impressive progress in building more and more powerful classical computers and a limit of this development was not in sight. This changed in the last decades of the 20th century, when first theoretical results regarding the power of quantum computation were published in terms of concrete algorithms. These showed a significant asymptotic speed-up for many relevant problems. Most prominent examples include Grover's algorithm [Gro96] for the search in unstructured databases or Shor's algorithm [Sho94] for the factorization of integers with significant consequences for state-of-the-art cryptography algorithms. Nowadays, there is a "quantum algorithm zoo" [Jor16] with diverse quantum algorithms for diverse problems.

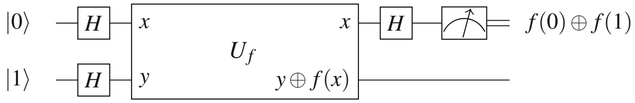


Fig. 1.2 Deutsch-Algorithm (improved version, [NC00, p. 33])

A widely used pattern that many of these algorithms share is called *quantum parallelism* and exploits that quantum superposition enables a quantum system to be in multiple basis states at the same time. This in principle allows a (Boolean) function to be evaluated in parallel for all possible inputs at once. However, this alone does not help much, as the results can not be read out directly. In fact, a clever post-processing has to be applied, such that the final measurement yields a meaningful result.

Example 1.2 Using classical computation, a Boolean function $f\colon \{0, 1\} \to \{0, 1\}$ needs to be evaluated twice in order to check whether it is constant or not. More precisely, one has to compute $f(0)$ and $f(1)$ separately. With the quantum circuit shown in Fig. 1.2, an improved version of the so-called *Deutsch-Algorithm*, this task can be performed by a single evaluation of f. More precisely, the quantum system is first brought into a superposition of the four basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$ using $\boxed{H}$ gates. Then, the block in the center computes the mapping $(x, y) \mapsto (x, y \oplus f(x))$ for all these basis states simultaneously, i.e. four evaluations of f at once. After some post-processing in form of another $\boxed{H}$ gate applied to the first qubit, this qubit is in the state $|f(0) \oplus f(1)\rangle$. Consequently, a measurement yields 0 if, and only if, $f(0) = f(1)$, i.e. f is constant.

The Deutsch-Algorithm was historically the first well-defined quantum algorithm achieving a speed-up over classical computation. As a special characteristic of this algorithm, the measurement yields the desired result in a deterministic fashion. However, in general this is not the case in quantum logic. Hence, quantum algorithms take this uncertainty into account by conducting the same quantum computation several times and deducing the final result from the measurements recorded after each execution.

Besides these difficulties concerning the transfer of results from the quantum to a discrete, conventional logic level, there are various other obstacles for the design of quantum algorithms and circuits. Actually, the term quantum “circuit” is a bit misleading since what it describes has little in common with conventional, electronic circuits. In fact, there are no circuit wires carrying information from the inputs to the outputs, such that the input and output “values” are present at different places at the same time. A quantum circuit is more like a register on which computations are executed in-place, i.e. immediately replacing the original values. However, according to quantum-mechanical principles, the operations are performed in a reversible manner (except measurements), such that always the preceding state could possibly be restored. This *inherent reversibility* poses severe challenges, especially since the (Boolean) functions that have to be implemented in a reversible fashion are usually far from being reversible. Moreover, in conventional circuit realizations of these functions, bits correspond to signals, i.e. electrical currents, that can easily be "cloned" physically by splitting up (*fan-out*). In contrast, the state of a qubit can not be cloned. More precisely, the well-known *no-cloning theorem* states that there is no quantum circuit that can clone an arbitrary state of a qubit [NC00, p. 532].

While rather satisfactory solutions to these structural problems could be obtained at the theoretical level, the practical realization of quantum circuits is still subject to fundamental physical problems. For instance, since the implementations have to be conducted at an atomic scale, they are very sensitive to environmental noise and dedicated error-correcting schemes have to be applied. Despite the severe challenges, several promising technologies that support quantum computation in principle have been proposed in the past decade. A broad survey of these quantum technologies has been conducted in the ARDA quantum computing roadmap [ARD]. Although qubits are a heavily limited resource in each of these technologies, quantum systems consisting of dozens of usable qubits are in the range of vision [MSB+11].

For quantum systems of this size, the manual design of quantum circuits realizing a given quantum functionality is no longer an option. In fact, the underlying complex-valued matrices that are used to precisely describe quantum functionality grow exponentially with the size of the quantum system. Beyond that, the set of quantum gates that correspond to physical operations (e.g. laser beams) and can, thus, be executed directly is in general rather small and largely depends on the targeted technology. As a consequence, most quantum gates can not be realized exactly, but rather have to be approximated—accepting a small amount of error.[3]

Overall, in order to keep up with the technological progress, there is an essential need for significant improvements at the design level that address these challenges—finally aiming at CAD for quantum circuits. More precisely, the underlying problems have to be considered from a logic design and logic synthesis perspective with the overall goal to provide methods which, similar to the design and synthesis of conventional circuits, (1) *automatically* generate a circuit description of the desired (quantum) functionality, (2) take into account the physical constraints of the target technology, and (3) scale to (quantum) systems of considerable size.

While this general scheme is perfectly in line with established design flows for conventional circuits (where electro-technical devices are abstracted and eventually designed on a Boolean logic level), how to design quantum circuits significantly differs from established CAD methods and flows:

- First of all, approaches for the efficient representation of the desired functionality—which are an essential tool in conventional CAD—are not applicable to the unitary operators considered in the quantum domain.
- Besides these serious challenges to the representation, the structural discrepancy between the quantum and conventional circuit model has fundamental implications that are not compatible with conventional design methods—first and foremost the inherent reversibility of quantum logic realizations.

[3] Fortunately, this approximation is to some extent compensated by the robust design of quantum algorithms due to the non-deterministic measurements, anyway.

- Finally, the actual physical realization of the circuits enforce the consideration of further constraints that are not present or of significantly less importance for conventional circuits. For instance, in certain possible technologies, only physically adjacent qubits may interact (this leads to so-called *nearest neighbour constraints*). Moreover, the quantum gates have to be implemented in a fault-tolerant fashion in order to cope with environmental perturbations. Above that, the available gates often only allow for an approximation rather than an exact realization of the desired functionality—a problem that does not have to be covered in conventional CAD at all.[4]

Overall, these constraints require the development of completely new methods for two central tasks of quantum logic design, namely synthesis and verification. By this, we mean methods that generate a circuit description of the desired quantum functionality (synthesis) and ensure the correctness of the generated circuits that result from the transformations and optimizations which are applied in this context (verification). However, without having an efficient representation of the targeted functionality in the first place, there is hardly any chance of success for the development of a quantum logic CAD.

1.3 Topics Covered in This Book

With the prospect of manageable quantum systems of considerable size, the interest in applications like quantum computation increased and preliminary investigations towards the development of methods for quantum logic CAD were conducted. Due to severe obstacles resulting from the complex theoretical foundations and extremely difficult physical realizations, most of these methods consider quantum logic design from a rather theoretical point of view. More precisely, the majority of approaches takes into account physical limitations and constraints as well as special characteristics of the targeted technology only to a minor degree. Moreover, in absence of efficient representations, many synthesis approaches provide theoretical upper bounds rather than algorithms that would scale to quantum systems consisting of dozens of usable qubits.

Addressing this central role of efficient representations for the development of a quantum logic CAD in general and for synthesis in particular, we extensively attend to the domain of representing quantum functionality in Part II of this book. Most previous approaches aiming at an efficient representation of quantum functionality tried to apply the same concepts that were beneficial in the conventional logic setting to the quantum logic domain in a rather straight-forward way—despite the fundamental differences between the two logic worlds. In contrast, a complementary approach is given by *Quantum Multiple-Valued Decision Diagrams* (QMDDs) which

[4] Nonetheless, approximation has recently also become a trend in classical computing, since approximate results are sometimes sufficient and realizable in shorter time or using fewer resources [GMP+11].

explicitly take into account the challenges and characteristics of quantum logic. However, the initial proposal [MT06] had some shortcomings that significantly restricted its applicability to general quantum logic. To this end, we provide a revised version of QMDDs that overcomes these limitations and, thus, may serve as a very promising basis for quantum logic CAD.

Having this powerful tool at hand, we demonstrate the application of these representations in the design of quantum circuits in Part III. To this end, we present dedicated synthesis approaches for two important classes of quantum functionality, namely Boolean components and Clifford group operations. In either case, we put a strong focus on a good scalability of the approaches. Additionally, we consider one of the most obvious applications of decision diagrams for logic design, namely design verification. More precisely, one frequently needs to verify whether different functional descriptions that are generated during the design process (e.g. before and after performing an optimization step or moving to a lower level of abstraction) indeed describe an equivalent functionality. To this end, we develop a scheme for checking whether different (circuit) descriptions of the same functionality are indeed equivalent.

Before that, however, in the next chapter we provide the mathematical foundations that are needed to understand the following investigations and make this book self-contained.

Chapter 2
Background

In this chapter, we present the mathematical foundations of quantum logic and computation which are required to make this book self-contained.[1] The respective descriptions are kept brief; readers wishing an in-depth introduction are referred to the respective literature containing more comprehensive descriptions such as e.g. [NC00, Mer07].

We begin in Sect. 2.1 by providing an introduction to Boolean logic with a particular focus on reversible logic, i.e. reversible Boolean functions. The latter play an important role for quantum computation. Afterwards, the basics of quantum logic and circuits are provided in Sects. 2.2 and 2.3, respectively.

2.1 Boolean Logic

Logic computations can be defined as functions over Boolean variables. More precisely:

Definition 2.1 A *Boolean function* over the standard Boolean algebra $\mathbb{B} = \{0, 1\}$ is a mapping $f : \mathbb{B}^n \rightarrow \mathbb{B}$ with $n \in \mathbb{N}$, where $\mathbb{B}^n$ is the n-fold Cartesian product of $\mathbb{B}$, i.e. $\mathbb{B}^n = \mathbb{B} \times \ldots \times \mathbb{B}$. A function f is defined over its *primary input* variables $X = \{x_1, x_2, \ldots, x_n\}$ and, hence, is also denoted by $f(x_1, x_2, \ldots, x_n)$.

Definition 2.2 A *multi-output Boolean function* is a mapping $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ with $n, m \in \mathbb{N}$. More precisely, it is a system of Boolean functions $f_i(x_1, x_2, \ldots, x_n)$.

[1] The reader is assumed to be familiar with basic concepts of set theory and linear algebra. Additionally, some basic knowledge about group theory is required to completely understand the proofs in Sect. 7.2.3.

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_2

The respective functions f_i $(1 \leq i \leq m)$ are also denoted as *component functions* or *primary outputs*. The set of *all* Boolean functions with n inputs and m outputs is denoted by $\mathscr{B}_{n,m} = \{f \mid f : \mathbb{B}^n \to \mathbb{B}^m\}$.

Multi-output functions can also be denoted as n-input, m-output functions or $n \times m$ functions. The precise mapping defining a Boolean function or multi-output Boolean function can be described by a truth table or in terms of Boolean algebra with expressions formed over the variables from X and operations like $\wedge$ (*AND*), $\vee$ (*OR*), $\oplus$ (*XOR*), and $\bar{\cdot}$ (*NOT*).

An important subset of multi-output Boolean functions highly relevant to quantum computing is constituted by reversible functions which are defined as follows:

Definition 2.3 A multi-output function $f \in \mathscr{B}_{n,m}$ is termed *reversible* if it is a bijection, i.e.

- its number of inputs is equal to the number of outputs (i.e. $n = m$) and
- it maps each input pattern to a unique output pattern (i.e. performs a permutation of the set of input patterns).

A Boolean function that is not reversible is termed *irreversible*.

Example 2.1 Table 2.1(a) shows the truth table of a 3-input, 2-output function representing a 1-bit adder. This function is irreversible, since $n \neq m$. The function in Table 2.1(b) is also irreversible since, although the number n of inputs is equal to the number m of outputs, the mapping is not a permutation; e.g. both inputs 000 and 001 map to the output 000. In contrast, the 3×3 function shown in Table 2.1(c) is reversible, since each input pattern maps to a unique output pattern.

The input/output mapping of a (reversible) Boolean function can also be represented in terms of a characteristic function.

Definition 2.4 The characteristic function for a Boolean function $f \in \mathscr{B}_{n,m}$ is defined as $\chi_f : \mathbb{B}^n \times \mathbb{B}^m \to \mathbb{B}$ where $\chi_f(x, y) = 1$ if, and only if, f maps the input pattern determined by the first n inputs of χ_f to the output pattern determined by the remaining m inputs, i.e. $f(x) = y$.

Table 2.1 Sample Boolean functions

(a) Irrev. (Adder)

x_1	x_2	x_3	f_1	f_2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b) Irreversible

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

(c) Reversible

x_1	x_2	x_3	f_1	f_2	f_3
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	0	0	1
1	0	1	0	0	0
1	1	0	1	1	0
1	1	1	1	1	1

$$\begin{array}{cc|cccccccc|}
 & & \multicolumn{8}{c}{\text{Inputs}} \\
 & {}^{x_3}_{x_1 x_2} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
\multirow{8}{*}{\rotatebox{90}{Outputs}} & 000 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
 & 001 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 & 010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & 011 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 & 100 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 101 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 110 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
 & 111 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
\end{array}$$

Fig. 2.1 Matrix representation of the reversible function from Table 2.1(c)

Finally, permutation matrices are a common representation for reversible Boolean functions, especially in the context of quantum computation.

Definition 2.5 A reversible Boolean function with n variables describes a permutation σ of the set $\{0, \dots, 2^n - 1\}$. This permutation can also be described using a *permutation matrix*, i.e. a $2^n \times 2^n$ matrix $\mathbf{P} = [p_{i,j}]_{2^n \times 2^n}$ with $p_{i,j} = 1$ if $i = \sigma(j)$ and 0 otherwise, for all $i, j = 0, \dots, 2^n - 1$. Each column (row) of the matrix represents one possible input pattern (output pattern) of the function. If $p_{i,j} = 1$, then the input pattern corresponding to column j maps to the output pattern corresponding to row i.

Example 2.2 The reversible Boolean function defined by the truth table from Table 2.1(c) is also represented by the permutation matrix shown in Fig. 2.1.

The above concepts can readily be extended from the Boolean domain to the multiple-valued case. Here, variables can take values in $\{0, \dots, r - 1\}$, where r is called the radix. Then, a truth table for a multiple-valued function over n variables has r^n rows. The concept of reversibility remains the same, i.e. a multiple-output multiple-valued function is reversible if it has the same number of inputs and outputs and each input pattern maps to a unique output pattern. Such a function can be represented by an $r^n \times r^n$ permutation matrix. The entries of the matrix are 0's and 1's since they denote correspondence in the mapping and not logical values.

2.2 Quantum Logic

Next we consider the preliminaries on quantum logic. We begin by introducing the means for describing and measuring quantum systems in Sect. 2.2.1. This is followed by an introduction of the mathematical description of quantum operations and their effect on quantum systems in Sect. 2.2.2. Finally, we consider the realization of complex quantum operations in terms of quantum circuits and review gate libraries proposed for this purpose in Sect. 2.3.

2.2.1 Qubits and Measurement

Quantum systems are composed of *qubits*. Analogously to conventional bits, a qubit can represent a (Boolean) 0 or 1, but also superpositions of the two. More formally:

Definition 2.6 A *qubit* is a two-level quantum system, described by a two-dimen-sional Hilbert space. A basis of this *state space* is given by the two orthogonal quantum states $|0\rangle \equiv \binom{1}{0}$ and $|1\rangle \equiv \binom{0}{1}$ which are used to represent the Boolean values 0 and 1. Thus, the state of a qubit may always be written as

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle,$$

where α_0 and α_1 are complex numbers with $|\alpha_0|^2 + |\alpha_1|^2 = 1$, i.e. as a so-called *superposition* of the (computational) *basis states* $|0\rangle$ and $|1\rangle$.

The quantum state of a single qubit is denoted by the vector $\binom{\alpha_0}{\alpha_1}$. The state space of larger quantum systems composed of $n > 1$ qubits is given by the tensor product of the state spaces of the individual qubits. Accordingly, there are 2^n basis states $(|0\ldots00\rangle, |0\ldots01\rangle, \ldots, |1\ldots11\rangle)$, and the system can be in an arbitrary superposition[2] of these states

$$|\psi\rangle = \sum_{k=0}^{2^n-1} \alpha_k|k\rangle$$

for complex numbers $\alpha_0, \ldots, \alpha_{2^n-1}$ with $\sum_{k=0}^{2^n-1} |\alpha_k|^2 = 1$. The corresponding state vector $(\alpha_k)_{0\leq k\leq 2^n-1}$ has dimension 2^n. Note, that two states $|\psi_1\rangle$ and $|\psi_2\rangle$ actually represent the same physical state of a quantum system if, and only if, there is a complex number $\phi \neq 0$, such that $|\psi_1\rangle = e^{i\phi} \cdot |\psi_2\rangle$. In this case, we say that $|\psi_1\rangle$ and $|\psi_2\rangle$ differ by a *global phase* shift.[3]

Due to physical limitations there is no possibility to precisely read-out the state of a quantum system, i.e. to obtain the so-called *amplitudes* α_k. In fact, it is only possible to perform a *measurement* which destroys the quantum state of the system and causes it to collapse to some basis state where the probability for measuring basis state $|k\rangle$ is given by $|\alpha_k|^2$.

[2] More precisely, we say that a system is in *superposition* if more than one α_k is non-zero.

[3] Strictly speaking, the state spaces are *projective* Hilbert spaces where each state $|\psi\rangle$ is represented by a *ray* of vectors, i.e. a set $\{c \cdot |\psi\rangle \mid c \in \mathbb{C}, |c| = 1\}$.

Example 2.3 Consider the following three different states of a qubit: $|\psi_1\rangle = \frac{1}{\sqrt{2}}\binom{1}{1}$, $|\psi_2\rangle = \frac{1}{\sqrt{2}}\binom{1}{-1}$, $|\psi_3\rangle = \frac{1}{\sqrt{2}}\binom{i}{i}$. For all three states, the probability of measuring $|0\rangle$ is the same as measuring $|1\rangle$: $|\alpha_0|^2 = |\alpha_1|^2 = \frac{1}{2}$.

In this regard, the amplitudes are often represented in *polar coordinates* by $\alpha_k = p_k \cdot e^{i\theta_k}$, i.e. as a decomposition into the *modulus* $p_k = |\alpha_k| \in [0, 1]$ (determining the probability of measuring the corresponding basis state $|k\rangle$) and the so-called *phase (factor)* $\theta_k \in [0, 2\pi)$.

Example 2.4 Consider again the three qubit states from Example 2.3. While the modulus is always the same ($\frac{1}{\sqrt{2}}$), we observe the three different phases 0 ($e^{i\cdot 0} = 1$), $\frac{\pi}{2}$ ($e^{i\cdot\pi/2} = i$), and π ($e^{i\cdot\pi} = -1$). Moreover, $|\psi_3\rangle = i \cdot |\psi_1\rangle$ which means that $|\psi_1\rangle$ and $|\psi_3\rangle$ are equal up to global phase and, thus, there is no way to find out which of them is actually present. In contrast, $|\psi_2\rangle$—though having the same moduli—can in principle be distinguished from these states as shown later on in Example 2.6.

2.2.2 Quantum Operations

According to the postulates of quantum mechanics, the time-dependent evolution of a quantum system (so-called *Hamiltonians*) can be described by linear transformation operations on the state space satisfying the following

Definition 2.7 A *quantum operation* over n qubits can be represented by a unitary *transformation matrix*, i.e. a $2^n \times 2^n$ matrix $\mathbf{U} = [u_{i,j}]_{2^n \times 2^n}$ with

- each entry $u_{i,j}$ assuming a complex value and
- the inverse $\mathbf{U}^{-1}$ of $\mathbf{U}$ being the *adjoint* $\mathbf{U}^\dagger$, i.e. the conjugate transpose matrix of $\mathbf{U}$, such that $\mathbf{U} \cdot \mathbf{U}^\dagger = \mathbf{U}^\dagger \cdot \mathbf{U}$ yields the identity matrix.

According to this definition, every quantum operation is reversible since the matrix $\mathbf{U}$ defining any quantum operation is invertible and $\mathbf{U}^{-1} = \mathbf{U}^\dagger$ describes the inverse quantum operation.

Commonly used quantum operations include the *Hadamard* operation H (setting a qubit into a balanced superposition) as well as the *rotation* operations R_x, R_y, and R_z (parametrized by a rotation angle θ). Important special cases of the latter are the *NOT* operation $X = R_x(\pi)$ which flips the basis states $|0\rangle$ and $|1\rangle$, and the *phase shift* operations T, $S = T^2$, and $Z = S^2 = T^4$ which correspond to $R_z(\theta)$ rotations with the rotation angles $\theta = \frac{\pi}{4}, \frac{\pi}{2}$, and π, respectively.[4]

[4] More precisely, the equalities only hold up to *global phase*, i.e. a multiplicative scalar factor.

The corresponding transformation matrices are defined as

$$\mathbf{H} = \tfrac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \ \mathbf{R_x}(\theta) = \begin{pmatrix}\cos\frac{\theta}{2} & -i\sin\frac{\theta}{2}\\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix},$$
$$\mathbf{R_y}(\theta) = \begin{pmatrix}\cos\frac{\theta}{2} & -\sin\frac{\theta}{2}\\ +\sin\frac{\theta}{2} & \cos\frac{\theta}{2}\end{pmatrix}, \ \mathbf{R_z}(\theta) = \begin{pmatrix}e^{-i\theta/2} & 0\\ 0 & e^{i\theta/2}\end{pmatrix}, \ \textit{and}$$
$$\mathbf{X} = \begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}, \ \mathbf{T} = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{\pi i}{4}}\end{pmatrix}, \ \mathbf{S} = \begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}, \ \mathbf{Z} = \begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}.$$

In any case, the columns of the transformation matrix correspond to the output state vectors that result when applying the respective operation to basis states as inputs. Thus, the entry $u_{i,j}$ in the ith row and jth column of the matrix describes the amplitude mapping from the input basis state $|j\rangle$ to the output basis state $|i\rangle$.

Example 2.5 Applying the Hadamard operation H to the input basis state $|1\rangle = \binom{0}{1}$, i.e. computing $\mathbf{H} \cdot |1\rangle$ yields a new quantum state

$$|\psi'\rangle = \mathbf{H} \cdot |1\rangle = \tfrac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\begin{pmatrix}0\\ 1\end{pmatrix} = \tfrac{1}{\sqrt{2}}\begin{pmatrix}1\\ -1\end{pmatrix} = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

for which $\alpha_0 = -\alpha_1 = \frac{1}{\sqrt{2}}$, i.e. the input amplitude of $|1\rangle$ is split up equally to the output amplitude of $|0\rangle$ and $|1\rangle$. Measuring the qubit would either lead to a Boolean 0 or a Boolean 1 with a probability of $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$ each. This computation represents one of the simplest quantum computers—a single-qubit random number generator.

Example 2.6 Applying H to the basis state $|0\rangle = \binom{1}{0}$, i.e. computing $\mathbf{H} \cdot |0\rangle$ yields the state $|\psi_1\rangle = \frac{1}{\sqrt{2}}\binom{1}{1}$ from Example 2.3. Similarly, we obtain that $\mathbf{H} \cdot |\psi_1\rangle = |0\rangle$ and $\mathbf{H} \cdot |\psi_2\rangle = |1\rangle$. This means that we can apply the Hadamard operation in order to distinguish between $|\psi_1\rangle$ and $|\psi_2\rangle$ which is not possible with a direct measurement (cf. Example 2.4).

Besides these operations that work on a single *target qubit*, there are also controlled operations on multiple qubits. The state of the additional *control qubits* determines which operation is performed on the target qubit. More precisely, the operation on the target qubit is executed if, and only if, the control qubits with a *positive* control are in the $|1\rangle$-state and the ones with a *negative* control are in the $|0\rangle$-state.

Example 2.7 A very prominent example for multiple-qubit operations is the *controlled NOT* (CNOT) operation on two qubits which applies the NOT operation to the target if, and only if, the control qubit is in the $|1\rangle$-state (positive control). On the matrix level, it is defined by

$$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix},$$

which can be expressed as $\mathbf{I} \oplus \mathbf{X}$ using the direct matrix product notation “$\oplus$”.

The above concepts can readily be extended to the multiple-valued case of so-called *qudits* where there are r basis states $|0\rangle, \ldots, |r-1\rangle$ and the state of a qudit can be written as $|x\rangle = \Sigma_{i=0}^{r-1}\alpha_i|i\rangle$ with $\Sigma_{i=0}^{r-1}|\alpha_i|^2 = 1$. The transformations in this case are described by $r^n \times r^n$ unitary matrices.

2.3 Quantum Circuits and Gate Libraries

Complex quantum operations (e.g. described by a quantum algorithm) are usually realized by a set of elementary, low-level quantum operations or quantum computational instructions (e.g. represented in terms of *quantum gates* g_i) that are performed in a predetermined serial fashion as a *quantum circuit* $G = g_1 \ldots g_l$ with $1 \leq i \leq l$. On the matrix level, such a composition of quantum gates can be expressed by a direct matrix multiplication of the corresponding gate matrices.

Example 2.8 Consider the 3-qubit quantum circuit shown in Fig. 2.2. It realizes a 2-controlled NOT operation known as the *Toffoli gate* (depicted in Fig. 2.3). More precisely, the basis states of the third qubit are swapped if, and only if, the first and second qubits are in the $|1\rangle$-state. Following the established conventions, horizontal lines represent qubits. Quantum operations $\boxed{H}$ (Hadamard operation, cf. Example 2.5), $\boxed{T}$ (phase shift by $\frac{\pi}{4}$), $\bullet\!\!-\!\!\oplus$ ($CNOT$ with a positive control), etc. are applied successively from left to right.

Example 2.9 With the concepts introduced above, we are now able to understand the details of the circuit from Fig. 1.1 (cf. Example 1.1 on page 5). Recall, that it

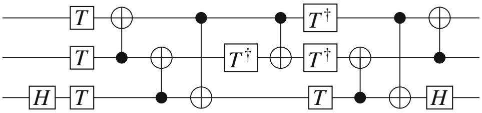


Fig. 2.2 A quantum circuit

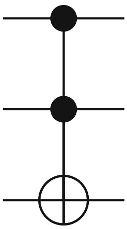


Fig. 2.3 The Toffoli gate

allows for teleporting qubit states over potentially large distances. Let us follow the precise evolution of the quantum state.

The first two gates transform the lower pair of qubits from $|00\rangle$ to $\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$ and finally to $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. Note that this state has an important property: measuring a $|0\rangle$ on one qubit enforces a $|0\rangle$ on the other one (and analogously for $|1\rangle$). Thus, the two gates create *entanglement* between the two qubits, since measuring one of it completely determines the state of the other. The third gate of the circuit encodes the information of the first qubit on the entangled pair (thereby entangling all qubits of the system). More precisely, if $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, we obtain for the state of the entire system:

$$\alpha_0|0\rangle \cdot \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) + \alpha_1|1\rangle \cdot \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)$$

Applying the final Hadamard gate to this state yields

$$\frac{\alpha_0}{2}(|0\rangle + |1\rangle) \cdot (|00\rangle + |11\rangle) + \frac{\alpha_1}{2}(|0\rangle - |1\rangle) \cdot (|10\rangle + |01\rangle).$$

Now, in order to establish ψ on the third qubit, we have to destroy the entanglement by measuring the other qubits. More precisely, measuring the first qubit results in the quantum state

$$\frac{\alpha_0}{\sqrt{2}} \cdot (|00\rangle + |11\rangle) \pm \frac{\alpha_1}{\sqrt{2}} \cdot (|10\rangle + |01\rangle),$$

where the minus occurs only for the case that $M_1 = |1\rangle$ is measured. This possible phase shift is taken care of with the final controlled Z gate. Finally, we observe that measuring a $|0\rangle$ on the second qubit yields the desired state on the third qubit (modulo the phase shift), while measuring a $|1\rangle$ leads to $\alpha_0|1\rangle \pm \alpha_1|0\rangle$. In the latter case, the basis states $|0\rangle$ and $|1\rangle$ have to be swapped. This is achieved by the controlled NOT that is applied depending on the result of M_2. Overall, the circuit demonstrates how to transfer the state of one qubit to another qubit without a direct interaction between them. Even more, the third qubit could have possibly been separated from the others after applying the first two gates and only the measurement results M_1 and M_2, i.e. two classical bits (indicated by double lines), are required to establish the desired state on the distant qubit. In fact, the two controlled quantum gates that are applied for this purpose do not depend on $|\psi\rangle$, but only (!) on the measurement results.

Note that inverting a given quantum circuit is a relatively easy task in most cases. In fact, most quantum gates are self-inverse (like the Hadamard, NOT, and CNOT gate) or determining the inverse is straight-forward (e.g. for rotations by taking the negated rotation angle). Consequently, the inverse quantum circuit is obtained by (1) reversing the gate order and (2) replacing each gate by its inverse.

An important sub-class of quantum circuits is constituted by *reversible circuits*. These circuits are composed of (classical) reversible gates, such that the

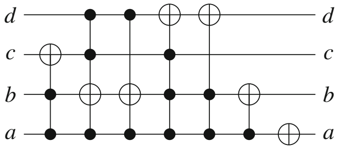


Fig. 2.4 A reversible circuit

transformation matrix for each gate and the entire circuit is a permutation matrix. The most established type of reversible gate is the *multiple-controlled Toffoli* (MCT) gate. MCT gates consist of a possibly empty set of control qubits and a single target qubit on which a NOT gate is performed if, and only if, all control qubits carry the appropriate value. Note that the MCT gate library also includes the special cases of NOT (empty set of controls) and *controlled NOT* (CNOT) gates (singleton set of controls). For historical reasons and for brevity, we simply use the term *Toffoli gate* to refer to the 2-controlled Toffoli gate.

Example 2.10 Consider the reversible circuit shown in Fig. 2.4 that realizes a modulo 10 counter. More precisely, if the input (taken as a binary number $dcba_2$) is less than the (decimal) number 10, then the output is incremented and taken modulo 10, i.e. the output is $((dcba + 1)\%10)_2$. For binary numbers greater than or equal to 10, the circuit does not behave according to this formula. However, it is clear that—due to reversibility—the output also has to be greater than or equal to 10.

It is a common phenomenon that, as in the previous example, reversible circuits have a meaningful output only for a subset of the input patterns. This is because many reversible functions are obtained from irreversible functions by adding extra inputs/outputs in order to ensure a bijective mapping, as already seen in Example 2.1. This process is called *embedding* and is discussed in more detail later in Sect. 7.1.1.

Several libraries of elementary quantum operations have been proposed in the literature. From a theoretical point of view, the set of arbitrary one-qubit gates (unitary 2×2 matrices) and a single 2-qubit gate, namely the controlled NOT (CNOT) gate, is sufficient to approximate any quantum operation to an arbitrary precision [BBC+95]. Libraries with this property are called *universal*. To this end, it has been shown that a much smaller set of operations, comprising only CNOT, H, and T operations (forming the so-called *Clifford+T library*), is also universal [BMP+00]. Moreover, these quantum operations can in principle be implemented in a fault-tolerant fashion [BMP+00]—a crucial property since quantum computing is inherently very sensitive to environmental factors such as radiation and, hence, fault-tolerance is even more important than for conventional systems. However, the technologies that are actually used for the physical realization of quantum circuits support a small subset of quantum operations only. This is discussed in more detail in Sect. 6.1.

Part II
Representation of Quantum Functionality

Chapter 3
Challenges and Initial Approaches

Efficient representations of the desired functionality were crucial for the development of conventional logic CAD, and they are likely to play an equivalent role for quantum logic CAD as well. In this part, we thus consider the state-of-the-art in representing quantum functionality. In the present chapter, we first argue in Sect. 3.1 why approaches used in conventional logic are insufficient and what major challenges have to be overcome in order to develop efficient representations of quantum logic. In Sect. 3.2, we then discuss initial approaches to address these challenges in terms of decision diagrams and illustrate their particular shortcomings. Afterwards, in Chap. 4, we present a promising solution to these shortcomings, namely *Quantum Multiple-Valued Decision Diagrams* (QMDDs), a decision diagram kind that explicitly supports many quantum-mechanical properties and is a dedicated data-structure for the efficient representation and manipulation of quantum functionality as confirmed by an experimental evaluation. We finally conclude this part in Chap. 5 with a comparison of QMDDs to previously proposed approaches and illustrate the superiority of this dedicated data-structure also from a theoretical perspective. Moreover, we discuss why graphical representations and especially QMDDs are rarely employed in state-of-the-art quantum logic design and which potential benefits have therefore remained unused so far.

3.1 From Conventional to Quantum Logic

In conventional logic, a large body of research has focused on the unique and efficient representation of Boolean functions $f : \mathbb{B}^n \to \mathbb{B}$. More precisely:

- The naive way to represent a Boolean function in a unique fashion is in terms of a *truth table*, i.e. a complete, enumerative list of all input/output mappings.

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_3

However, this representation is only feasible for functions with a small number of primary inputs n as truth tables always have an exponential complexity.

- Many Boolean functions can be represented more efficiently by using *algebraic representations* to characterize their *ON-set*, i.e. the set of inputs for which f evaluates to true: $\mathrm{ON}_f = \{(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n) = 1\}$. The most established representatives of this class are the *Disjunctive/Conjunctive Normal Form* (DNF/CNF) which are also called *Sum of Products* and *Product of Sums* representations as they represent f as a disjunction of conjunctions and conjunction of disjunctions, respectively. While these representations can be rather compact, they share the drawback of not being canonic, i.e. the same Boolean function may be represented by different DNFs or CNFs. In order to achieve uniqueness and canonicity, all conjunctions (disjunctions) in fact have to be *minterms* (*maxterms*), thus, comprising all n variables either as a positive or negative literal. By this, however, the advantage over truth tables reduces dramatically. Overall, algebraic representations can either be unique—at the cost of being rather inefficient—or compact—at the cost of not being unique.
- To this end, a compact representation which is—at the same time—also unique can be achieved using graphical representations like *decision diagrams*. The most prominent representative of this class is the *Binary Decision Diagram* (BDD) which essentially is a directed acyclic graph whose vertices represent the Shannon decomposition of f (a more precise definition is given in Sect. 3.2.1). Moreover, BDDs do not only tend to be even more compact than CNFs or DNFs (and, of course, their canonic versions), they can also be evaluated and manipulated very efficiently. For instance, the value of f for a given assignment of the n input variables can always be computed in at most n steps, while for CNFs/DNFs all disjunctions/conjunctions have to be considered in the worst case.

The three approaches outlined above are illustrated by means of the following

Example 3.1 Consider the Boolean function $f : \mathbb{B}^3 \to \mathbb{B}$ with $n = 3$ primary inputs given by the Boolean formula $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \oplus x_3$.

- The truth table of f is shown in Fig. 3.1a. Here, the first three columns list all possible assignments to the input variables x_1 to x_3, while the last column shows the corresponding values of f.
- In Fig. 3.1b, algebraic representations characterizing the ON-set of f are shown. For both CNF and DNF, the canonic representation consisting of min-/maxterms is given. Note that f could also be described more compactly in DNF as $(\overline{x_1} \wedge x_3) \vee (\overline{x_2} \wedge x_3) \vee (x_1 \wedge x_2 \wedge \overline{x_3})$ or in CNF as $(x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$ which can be more adequate if uniqueness is not of primary importance. This is e.g. the case when the value of f shall be determined for a particular assignment of the input variables and one, thus, has to check whether at least one conjunction (DNF) or all disjunctions (CNF) evaluate to true.
- Finally, Fig. 3.1c shows the graphical BDD representation of f. There are two terminal vertices (labelled 0 and 1) which represent the possible outcomes of f. The labels of the remaining vertices represent the input variables of f. Each path

x_1	x_2	x_3	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

(a) Truth table

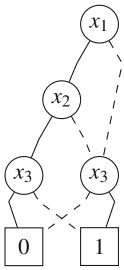
DNF / Sum of Products:

$(\overline{x_1}\wedge\overline{x_2}\wedge x_3)\vee(\overline{x_1}\wedge x_2\wedge x_3)\vee$
$(x_1\wedge\overline{x_2}\wedge x_3)\vee(x_1\wedge x_2\wedge\overline{x_3})$

CNF / Product of Sums:

$(x_1\vee x_2\vee x_3)\wedge(x_1\vee\overline{x_2}\vee x_3)\wedge$
$(\overline{x_1}\vee x_2\vee x_3)\wedge(\overline{x_1}\vee\overline{x_2}\vee\overline{x_3})$

(b) Canonic CNF/DNF



(c) BDD

Fig. 3.1 Unique representations of a Boolean function

from the root vertex (labelled by x_1) down to a terminal vertex represents a set of one or more assignments to the input variables x_1, x_2, x_3 leading to the same outcome of f. More precisely, solid edges indicate an assignment $x_i = 1$, while dashed edges represent the case $x_i = 0$. For instance, the path on the very left following the solid lines to the 0-terminal represents the assignment $x_1 = x_2 = x_3 = 1$ for which f evaluates to 0. In contrast, the path on the very right represents the two assignments with $x_1 = 0, x_3 = 1$ (x_2 can be chosen arbitrarily).

The availability of unique, efficient and easily manipulable representations of Boolean functions played a vital role for the rapid progress in conventional logic design in the past decades and it would, thus, be highly desirable to exploit such representations in the quantum domain as well. However, although many quantum algorithms contain a substantial Boolean component, i.e. a part or module that realizes a Boolean function (e.g. the modular exponentiation in Shor's algorithm or the oracle in Grover's algorithm), none of the so far mentioned representations is suitable to represent quantum functionality in general—including the whole range of quantum-mechanical effects which makes quantum computation so powerful like superposition, entanglement, or phase shifts.

More precisely, the objects of interest which have to be represented in quantum logic are Hamiltonians of a quantum system. These are linear, unitary mappings $\mathbb{C}^{2^n} \to \mathbb{C}^{2^n}$ that describe the system's evolution over time (where n is the number of qubits). The most natural way to represent such an Hamiltonian is to choose a basis of the Hilbert space $\mathbb{C}^{2^n}$ and then consider the corresponding transformation matrix—the counterpart to truth tables in conventional logic—, which is a $2^n \times 2^n$ complex-valued unitary matrix. As for truth tables, these matrices grow exponentially with the size of the quantum systems. For instance, a transformation matrix for a 20-qubit system has $2^{20} \times 2^{20} \approx 1.0995116 \cdot 10^{12}$ (around a trillion!) entries.

State-of-the-art approaches to handle these matrices, i.e. to perform required matrix operations and cope with the complex values are often based on computer algebra software like MatLab/Octave or other generic math libraries. Consequently, they scale very poorly when it comes to unitary matrices and are, thus, only applicable to rather small quantum systems. In order to deal with larger quantum systems, there is a need to derive more compact representations of the transformation matrices. In fact, they should exploit structural similarities of the matrices which result from quantum-mechanical properties like superposition or phase shifts, and, at the same time, should be able to handle complex values. Initial attempts to address these challenges in terms of decision diagrams are reviewed in the following section.

3.2 Decision Diagrams for Quantum Logic

Graphical representations were a break-through in the domain of conventional circuit design in the 1990s—first and foremost *Binary Decision Diagrams* (BDDs, [Ake78, Bry86]) as the most prominent example. As a consequence, with the prospect of large quantum computers becoming a more and more realistic vision, researchers have tried to transfer this approach to quantum logic in order to enable a similar improvement for quantum logic design. In this section, we describe the most established approaches that have been proposed in this context and also discuss their particular shortcomings.

3.2.1 Basic Concepts: Binary Decision Diagrams (BDDs)

To begin with and to review basic concepts of decision diagrams, we consider the conventional logic setting in which BDDs are employed to describe the desired functionality in a compact and easily manipulable fashion.

Definition 3.1 A *Binary Decision Diagram (BDD)* is a directed, acyclic graph (V, E) with the following properties:

- There are two *terminal* vertices $t_0, t_1 \subset V$, $t_0 \neq t_1$ representing the Boolean values 0 and 1.
- Each non-terminal vertex $v \in V \setminus \{t_0, t_1\}$ is labelled by a decision variable x_i with $1 \leq i \leq n$ and has exactly two outgoing edges. These edges are called *high* and *low edge*. Accordingly, the vertices to which these edges point are called *high/low child* and are denoted by $high(v)$ and $low(v)$, respectively.
- There is a unique *root vertex* v_0 that has no incoming edges.
- A BDD is called *reduced* when there are no *redundant* vertices for which both the high edge and the low edge point to the same child vertex $high(v) = low(v)$ and when there is no pair of vertices with the same variable label, the same high child, and the same low child. It is called *ordered* if the decision variables appear in the same, fixed order on each path from the root to the terminals.

Interpretation and Construction

Each non-terminal vertex v of a BDD represents a Boolean function. This function is constructed recursively using the functions represented by the child vertices of v. More precisely, the following composition formula is applied for this purpose:

$$f = \left(x_i \wedge f_{high(v)}\right) \vee \left(\overline{x_i^a} \cdot f_{low(v)}\right), \tag{3.1}$$

where "$+/\cdot$" denote the logical OR/AND operation, x_i denotes the decision variable by which v is labelled, and $f_{high(v)}$ and $f_{low(v)}$ denote the functions given by the high and low child, respectively. This formula corresponds to the well-known *Shannon decomposition*

$$f_v = x_i \cdot f_{x_i=1} + \overline{x_i^a} \cdot f_{x_i=0} \tag{3.2}$$

by setting $f_{high(v)} = f_{x_i=1}$ and $f_{low(v)} = f_{x_i=0}$, i.e. by using the positive and negative co-factors of f. Thus, using the above (de-)composition formulas, a corresponding BDD can be constructed for any given Boolean function and vice versa.

Example 3.2 Consider again the Boolean function $f : \mathbb{B}^3 \to \mathbb{B}$ from Example 3.1 on page 24 given by $f(x_1, x_2, x_3) = (x_1 \wedge x_2) \oplus x_3$. Assuming the variable order $x_1 \succ x_2 \succ x_3$, we first apply Eq. (3.2) for $i = 1$, i.e. we perform the Shannon decomposition for x_1. This yields

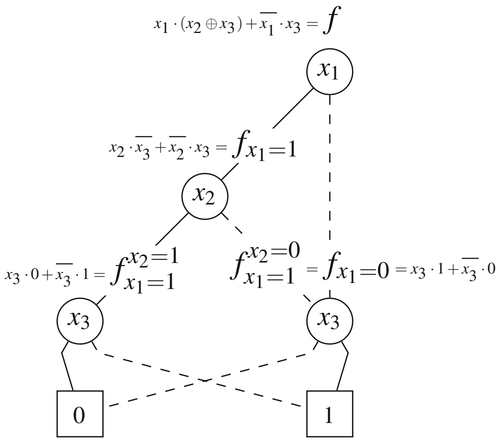
$$f = x_1 \cdot (x_2 \oplus x_3) + \overline{x_1} \cdot x_3,$$

as the positive/negative co-factors of f are $f_{x_1=1} = x_2 \oplus x_3$ and $f_{x_1=0} = x_3$. Now, decomposing $f_{x_1=1}$ likewise w.r.t. x_2 yields

$$f_{x_1=1} = x_2 \cdot \overline{x_3} + \overline{x_2} \cdot x_3$$

with the co-factors being $f_{x_1=1}^{x_2=1} = \overline{x_3}$ and $f_{x_1=1}^{x_2=0} = x_3$. Using this set of co-factors, the BDD representation of f can be constructed as shown in Fig. 3.2 where the co-factors are annotated to the corresponding edges. As in Fig. 3.1c, we use the established notation for distinguishing between high and low edges by representing them as solid and dashed lines, respectively.

Fig. 3.2 BDD construction using Shannon decomposition


$x_1 \cdot (x_2 \oplus x_3) + \overline{x_1} \cdot x_3 = f$
x_1
$x_2 \cdot \overline{x_3} + \overline{x_2} \cdot x_3 = f_{x_1=1}$
x_2
$x_3 \cdot 0 + \overline{x_3} \cdot 1 = f_{x_1=1}^{x_2=1}$
$f_{x_1=1}^{x_2=0} = f_{x_1=0} = x_3 \cdot 1 + \overline{x_3} \cdot 0$
x_3
x_3
0
1

Manipulation and Skipped Variables

Moreover, the applied composition formula also allows one to apply operations of Boolean logic like AND, OR, etc. (denoted by OP in the following) as recursive graph operations. More precisely, we make use of the fact that constructing co-factors is compatible with applying logic operations. This means that it yields the same result to (a) first apply a Boolean operation to the operand functions and then consider the co-factor of the result or to (b) first restrict attention to the respective co-factors and then apply the Boolean operation directly on them. In terms of a formula, this means $(f \text{ OP } g)_{x_i=k} = \left(f_{x_i=k} \text{ OP } g_{x_i=k}\right)$ for $k = 0, 1$. To intuitively verify this formula, observe that it obviously does not make a difference for the result whether OP is first calculated on the whole part of f/g before one half of the results is dropped or whether at first the same half of the original functions is dropped and OP is only computed on the remaining part. Consequently, the Shannon decomposition for f OP g can be rewritten as

$$f \text{ OP } g = x_i \cdot \left(f_{x_i=1} \text{ OP } g_{x_i=1}\right) + \overline{x_i} \cdot \left(f_{x_i=0} \text{ OP } g_{x_i=0}\right). \tag{3.3}$$

Transferring this to BDD vertices v, w representing the functions f_v and f_w, respectively, the formula becomes

$$f_v \text{ OP } f_w = x_i \cdot \left(f_{high(v)} \text{ OP } f_{high(w)}\right) + \overline{x_i} \cdot \left(f_{low(v)} \text{ OP } f_{low(w)}\right), \tag{3.4}$$

which means that OP can be computed for two BDD vertices recursively by computing it on the high and low children separately.

However, a close look reveals that Eq. (3.4) is only correct if both vertices v and w are labelled by the same variable x_i. More precisely, in the case that w is not labelled by x_i it is not necessarily true that $f_{high(w)} = (f_w)_{x_i=1}$ and $f_{low(w)} = (f_w)_{x_i=0}$. Special care has to be taken for the case that the vertices are labelled by different variables. This case can happen when a BDD is not ordered, i.e. the variables may occur in different orders on different paths. In addition, even if the BDD is ordered, some variables may not occur at all on some paths, e.g. x_2 does not occur on the rightmost path in Fig. 3.2. This phenomenon of so-called *skipped variables* occurs when the (sub-)function does not depend on these variables, i.e. the positive and negative co-factors are equivalent $f_{x_i=0} = f_{x_i=1}$. In this case, a vertex w' labelled by x_i would point with both edges to the same vertex $high(w') = low(w') = w$.[1]

As this implies $f_w' = f_w$, the vertex w' is redundant and can therefore be removed. However, by assuming w' to be present and using the above identities $f_w' = f_w$ as well as $high(w') = low(w') = w$, we can rewrite Eq. (3.4) as follows:

[1] Without loss of generality, we assume that x_i (the decision variable of v) precedes the decision variable of w in the variable order. Otherwise, v and w can simply be swapped.

$$\begin{aligned} f_v \text{ OP } f_w &= f_v \text{ OP } f'_w \\ &= x_i \cdot \left(f_{high(v)} \text{ OP } f_{high(w')}\right) + \overline{x_i} \cdot \left(f_{low(v)} \text{ OP } f_{low(w')}\right) \\ &= x_i \cdot \left(f_{high(v)} \text{ OP } f_w\right) + \overline{x_i} \cdot \left(f_{low(v)} \text{ OP } f_w\right) \end{aligned} \tag{3.5}$$

Thus, by using w itself instead of w's children (as in the original Eq. (3.4)), f_v OP f_w can be computed also in case of skipped variables.

Overall, logic operations on Boolean functions can be computed efficiently on the corresponding BDDs if they are ordered w.r.t. the same variable order.

Canonicity and Variable Orders

In this book, BDDs are always assumed to be reduced and ordered (cf. Definition 3.1). These *Reduced Ordered BDDs* (ROBDDs, [Bry86]) have been shown to be unique representations of Boolean functions, i.e. different BDDs represent different Boolean functions such that there is exactly one BDD representing a particular Boolean function. More precisely, the representation is unique up to the order of decision variables that is applied. This property which is also called *canonicity* is highly desirable as it can be exploited for many purposes, e.g. for efficiently checking the equivalence of two Boolean functions by comparing their corresponding BDD representation.

For different variable orders, however, the size of the BDDs representing the same Boolean function, i.e. their number of vertices, can vary significantly (from linear to exponential w.r.t. the number of variables) and a large body of research has focused on methodologies for finding a "good" variable order [Rud93, EFD05].

Overall, BDDs are an important representative for a compact and at the same time unique representation of Boolean functions which can easily be manipulated as well and, thus, have become an indispensable tool in conventional logic design. However, BDDs are not directly applicable for the representation of the complex-valued transformation matrices that occur in the quantum logic setting. Before we see how the above concepts can nevertheless be utilized to represent those matrices, we shortly visit another interesting decision diagram construction that—as BDDs—is also only applicable to Boolean functions, but is still worth to be reviewed as it explicitly aims for a realization of the respective functionality in the quantum logic domain.

3.2.2 Still Boolean: Quantum Decision Diagrams (QDDs)

Though their name may raise higher expectations, *Quantum Decision Diagrams* (QDD, [AP06]) were proposed as a means to represent Boolean functions, even if their main purpose is to represent a given Boolean function in a fashion that allows for an efficient quantum logic synthesis using controlled $R_x(\theta)$ rotations.

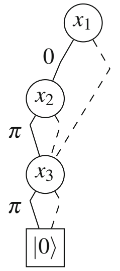


Fig. 3.3 QDD representation

Definition 3.2 A *Quantum Decision Diagram (QDD)* is a BDD with the following enhancements:

- The two non-terminal vertices represent the qubit basis states $|0\rangle$ and $|1\rangle$.
- Each high edge has an annotated weight $\theta \in (-\pi, \pi]$.

Instead of the Shannon decomposition, a so-called *functional bi-decomposition*

$$f = x_i \cdot R_x(\theta) \cdot f_{high(v)} + \overline{x_i^a} \cdot f_{low(v)} \tag{3.6}$$

is applied where the rotation angle $\theta \in (-\pi, \pi]$ corresponds to the high edge weight and $f_{low(v)}$, $f_{high(v)}$ again denote the (Boolean) functions that are given by the low and high child, respectively. More precisely, the co-domain of all these functions is given by the range of qubit states that can be reached from $|0\rangle$ by arbitrary R_x rotations: $\{R_x(\theta) \cdot |0\rangle \mid \theta \in (-\pi, \pi]\}$. Consequently, in Eq. (3.6) the "+" denotes component-wise addition and "·" denotes scalar and matrix-vector multiplication, respectively. Moreover, the Boolean variables $x_i \in \{0, 1\} \subset \mathbb{C}$ are interpreted as scalar values (with $\overline{x_i} = 1 - x_i$).

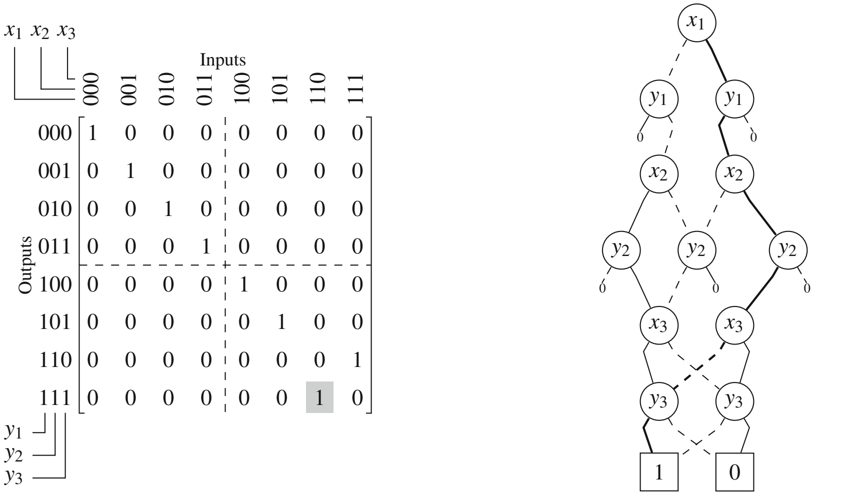
Example 3.3 The QDD representation of the Boolean function from Example 3.2 is shown in Fig. 3.3. Note that there is only one terminal vertex representing $|0\rangle$. In fact, as $R_x(\pi)$ swaps the basis states $|0\rangle$ and $|1\rangle$, the latter is achieved here using rotation angles $\theta = \pi$ on the high edges. From a Boolean logic perspective, a logical NOT is performed on all paths going through a high edge with this rotation angle. This essentially corresponds to the concept of complemented edges for BDDs [Ake78].

Overall, QDDs are a very close adaption of BDDs for the quantum domain. However, QDDs are only applicable for a small sub-class of quantum functionality—especially for (reversible) Boolean functions. While Boolean components constitute important parts of many quantum algorithms, the whole range of quantum-mechanical effects like superposition, entanglement, or phase shifts can not be covered with decision diagrams like BDDs or QDDs. Indeed, as we can see in the following, a quite different approach is necessary.

3.2.3 Characteristic Functions: QuIDDs and XQDDs

Instead of Boolean functions, unitary mappings within a 2^n-dimensional Hilbert space have to be considered in the general quantum logic setting. Thus, complex-valued transformation matrices need to be represented which may contain arbitrary complex values α with $|\alpha| \leq 1$. So, which (Boolean) variables can be taken as decision variables in order to represent these matrices as decision diagrams? While for classical BDDs, the input variables of the Boolean function can be used in a natural way for this purpose, the input of the unitary mapping is a state vector, i.e. a complex-valued column vector with 2^n entries, which is not suitable at all. Instead, it has been proposed to introduce decision variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ according to the input-output mapping that is induced by transformation matrices (cf. Sect. 2.2.2). More precisely: each possible assignment to the x_i or y_i variables represents a particular input basis state $|x_1 \ldots x_n\rangle$ or output basis state $|y_1 \ldots y_n\rangle$ and, thus, determines a column or row of the matrix, respectively. As a consequence, each single cell/entry of the matrix can be addressed by a particular combination of the x_i and y_i variables.

Example 3.4 Consider the transformation matrix in Fig. 3.4 which represents the reversible Boolean function $f : \mathbb{B}^3 \rightarrow \mathbb{B}^3$ whose components are given by $f_1 \equiv x_1$, $f_2 \equiv x_2$, and $f_3 \equiv (x_1 \wedge x_2) \oplus x_3$. Each column and row is labelled by the corresponding assignment to the index variables x_i, y_i $(i = 1, 2, 3)$. For instance, the highlighted cell in the bottommost row can be addressed with the assignment $(x_1, x_2, x_3) = (1, 1, 0)$ and $(y_1, y_2, y_3) = (1, 1, 1)$.



$$\begin{array}{c|cccc|cccc|}
 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
001 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
011 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\hline
100 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
101 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
111 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}$$

(a) Transformation matrix **M**

(b) BDD for χ_M (QuIDD for **M**)

Fig. 3.4 Transformation matrix and its characteristic function

In order to use these Boolean index variables for representing a complex-valued transformation matrix $\mathbf{M} = [m_{i,j}]$ in terms of a decision diagram, the matrix needs to be represented as a function over these variables. Such a function that represents $\mathbf{M}$ can be constructed by defining $\chi_M(x, y) = m_{x,y}$. This definition induces a function χ_M that returns the particular matrix entry which is determined by the assignment of the x_i and y_i variables.

Example 3.5 For the permutation matrix $\mathbf{M}$ shown in Fig. 3.4, χ_M is a conventional Boolean function $\mathbb{B}^6 \to \mathbb{B}$ as all matrix entries are from the set $\{0, 1\}$. More precisely, we have

$$(\chi_M(x, y) = 1) \Leftrightarrow \forall i \in \{1, 2, 3\}\colon\ y_i = f_i(x_1, x_2, x_3)$$

where the f_i are the component functions of f from Example 3.4. Consequently, in this particular case χ_M can be represented as a conventional BDD shown in Fig. 3.4b. For the sake of a better readability, several edges pointing to the 0-terminal are indicated by stubs.

However, a general unitary transformation matrix may contain an arbitrary number of different complex values such that χ_M is a genuine complex-valued function $\chi_M : \mathbb{B}^{2n} \to \mathbb{C}$. From a quantum logic perspective, χ_M returns the output amplitude of $|y_1 \dots y_n\rangle$ given that the input was the basis state $|x_1 \dots x_n\rangle$ and, thus, describes the corresponding quantum operation in terms of a (generalized) characteristic function of M with multiple possible outcomes. In order to represent this function—and, thus, a complex-valued transformation matrix—in terms of a BDD, the main idea is to use multiple terminal vertices, one for each different outcome of χ_M, i.e. each different value that occurs in the matrix. More precisely [VMH04, VMH07]:

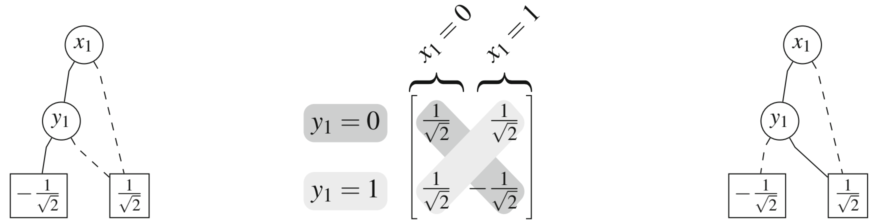
Definition 3.3 A *Quantum Information Decision Diagram (QuIDD)* is a BDD with the following enhancement:

- There is an arbitrary number of terminal vertices—each representing a complex value α with $|\alpha| \leq 1$. There are no two redundant terminal vertices that represent the same complex value.
- The range of decision variables is extended to $\{x_i, y_i \mid 1 \leq i \leq n\}$ such that each x_i is succeeded by y_i in the variable order (interleaved structure).

In order to represent a transformation matrix, each path in the diagram has to lead to the appropriate terminal vertex, i.e. to the terminal vertex representing the value of the particular matrix entry which is determined by the assignment of the x_i and y_i variables on that path.

As QuIDDs are essentially BDDs, also the composition formula for QuIDDs is basically the same as reviewed before for BDDs in Eq. (3.1). However, as we are representing complex-valued (characteristic) functions, note that the "+" has to be interpreted as scalar addition and the "·" as scalar multiplication while the decision variables $x_i, y_i \in \{0, 1\} \subset \mathbb{C}$ are interpreted as scalar values like in Eq. (3.6).

Example 3.6 The QuIDD for the transformation matrix shown in Fig. 3.4a (cf. Example 3.4 on page 31) is essentially isomorphic to the BDD in Fig. 3.4b.



(a) QuIDD (b) Variable assignment for XQDD (c) XQDD

Fig. 3.5 Decision diagram representations of the 2 × 2 Hadamard matrix

A particular matrix entry is represented by the path determined by the corresponding assignment of the index variables and its complex value is represented by the terminal vertex at the end of the path. For instance, the matrix entry highlighted in grey in Fig. 3.4a is represented by the path highlighted in bold in Fig. 3.4b.

While the BDD/QuIDD in Fig. 3.4a represents a permutation matrix, i.e. a reversible Boolean function, the QuIDD in Fig. 3.5a represents the true quantum Hadamard matrix $\mathbf{H} = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$ and, thus, uses multiple terminals to represent the real-valued matrix entries $\frac{1}{\sqrt{2}}$ and $-\frac{1}{\sqrt{2}}$.

A very similar approach is given by *X-decomposition Quantum Decision Diagrams* (XQDDs, [WLTK08]), which basically differ from QuIDDs only in the semantics of the x_i and y_i variables as shown in Table 3.1 on page 33. More precisely, the x_i variables still specify columns, while the y_i variables are used to specify diagonal blocks. The underlying working hypothesis for XQDDs is that it is more likely to have identical blocks, especially non-zero blocks, on the diagonal (and on the off-diagonal) of a matrix than in the same rows or columns.

Example 3.7 The variable assignment of the XQDD decision variables is illustrated by means of Fig. 3.5b for the 2 × 2 Hadamard matrix **H**. The resulting XQDD representation is shown in Fig. 3.5c. The only difference in comparison to the corresponding QuIDD representation in Fig. 3.5a is that the high and low edge of the y_1-vertex swap their targets.

Table 3.1 Assignment to decision variables for QuIDDs and XQDDs

QuIDD		**XQDD**	
x_i	y_i	x_i	y_i
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0
column	row	column	diagonal

Overall, QuIDDs and XQDDs apply a BDD-based approach to quantum logic and, therefore, benefit from existing methodologies for BDDs—especially algorithms for efficient function manipulation and optimization in terms of variable reordering. However, this approach to quantum logic is lacking a dedicated consideration of quantum-mechanical properties like phase shifts and is also ignoring the fact that quantum logic is potentially multiple-valued and employs qudits, i.e. multi-level quantum systems with more than two basis states. While the latter can in principle be overcome by moving from Binary to *Multiple-Valued Decision Diagrams* (MDDs) as the underlying data structure (as demonstrated in [LWK11]), a complementary approach is needed to take into account the special characteristics of quantum logic and overcome the shortcomings of the previous approaches. Such a complementary approach is presented in the next chapter.

Chapter 4
Quantum Multiple-Valued Decision Diagrams

In this chapter, we introduce a data-structure termed *Quantum Multiple-valued Decision Diagram* (QMDD, [NWM+16]) which can represent both binary and multiple-valued quantum functionality, i.e. quantum systems composed of qubits and/or qudits with more than two basis states, in a compact and efficient manner.[1] We first intuitively motivate basic concepts of the QMDD structure in Sect. 4.1. Afterwards, we provide a formal definition of QMDDs in Sect. 4.2 and argue why QMDDs are not only a compact, but also a canonic representation of quantum functionality in Sect. 4.3. As QMDDs are complementary to the BDD-based approaches reviewed in the previous chapter, dedicated algorithms have to be employed for their construction and manipulation. In addition, the established scheme for variable ordering of BDDs has also to be modified for the use with QMDDs. We present such algorithms for efficiently performing several matrix operations essential to quantum logic directly on QMDDs in Sect. 4.4. These allow for computing the QMDD representations of complex quantum functionality—quantum circuits—from the QMDDs for elementary quantum operations, which in turn can be derived rather easily. Moreover, in Sect. 4.5 we present a dedicated scheme for efficiently changing the variable order in QMDDs which can effectively reduce the size of the representation in terms of the number of vertices. An evaluation of the overall efficiency of QMDDs is afterwards presented in Sect. 4.6.

[1] In this book, we describe the revised version of QMDDs according to [NWM+16]. In comparison to the first proposal of QMDDs [MT06], the most significant improvements are the increased focus on a formal description from a quantum logic perspective, especially regarding the decomposition scheme, and the possibility for optimization through variable re-ordering.

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_4

4.1 Basic Concepts

As argued above, quantum operations are commonly represented by unitary transformation matrices, i.e. complex-valued $r^n \times r^n$ matrices where r is the radix and n is the number of variables (inputs and outputs). These matrices exponentially grow in size which is why conventional matrix representation and manipulation techniques are applicable for rather small instances only. However, two observations can be made which build the basis for a compact representation:

Observation 4.1 *Elementary quantum operations typically affect only a small number of qudits of a quantum system. The transformation matrix for the whole system, which is the Kronecker/tensor product of the respective (smaller) operation matrix and identity matrices, often contains the same pattern repeatedly throughout the matrix. These similar structures which may be identical or equal up to a constant multiplier can be exploited in reducing the representation of a matrix.*

Observation 4.2 *Transformation matrices are often sparse with many zero entries frequently appearing in blocks. Blocks of zeros can be represented very compactly leading to operation efficiencies particularly in matrix multiplication which is central to dealing with quantum logic where this operation is required very often for concatenating multiple quantum operations.*

To make use of these possible reductions, the fundamental idea of QMDDs is to use a partitioning of the original matrix into sub-matrices which in turn are partitioned in the same manner. These decomposition steps are represented by vertices eventually forming a decision diagram. Then, the redundancies following from the observations above can be avoided by using shared structures. More precisely, we observe—starting with $r = 2$—that a $2^n \times 2^n$ matrix can be partitioned into 4 sub-matrices of dimension $2^{n-1} \times 2^{n-1}$ as follows:

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_{00} & \mathbf{U}_{01} \\ \mathbf{U}_{10} & \mathbf{U}_{11} \end{bmatrix} \tag{4.1}$$

This partitioning is relative to the most significant row and column variable.

Example 4.1 Consider again the matrix shown in Fig. 3.4a on page 31. This matrix is partitioned with respect to variable x_1. Note that, in contrast to QuIDDs, the same variable is used for labelling both rows and columns. The sub-matrices are identified by subscripts giving the row (output) and column (input) value for that variable identifying the position of the sub-matrix within the matrix. For instance, $\mathbf{U}_{01}$ corresponds to the top-right sub-matrix which describes the mapping of the remaining variables when x_1 is mapped from input value 1 to output value 0. Using this partition, a matrix can be represented as a graph with vertices as shown in Fig. 4.1. The vertex is labelled by the variable associated with the partition and has directional edges pointing to vertices corresponding to the sub-matrices.

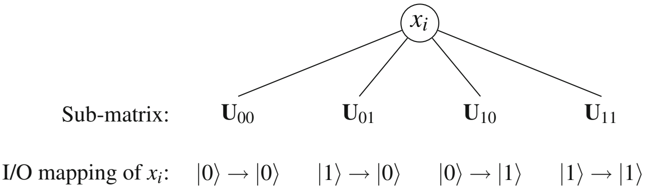

x_i
Sub-matrix:
$\mathbf{U}_{00}$
$\mathbf{U}_{01}$
$\mathbf{U}_{10}$
$\mathbf{U}_{11}$
I/O mapping of x_i:
$|0\rangle \rightarrow |0\rangle$
$|1\rangle \rightarrow |0\rangle$
$|0\rangle \rightarrow |1\rangle$
$|1\rangle \rightarrow |1\rangle$

Fig. 4.1 Vertex representing the matrix partitioning (for $r = 2$)

The partitioning process can be applied recursively to each of the sub-matrices and to each of the subsequent levels of sub-matrices until one reaches the terminal case where each sub-matrix is a single value. The result is that the initial matrix is represented by a tree. By traversing the tree, one can access the successively partitioned sub-matrices of the original matrix down to the individual elements.

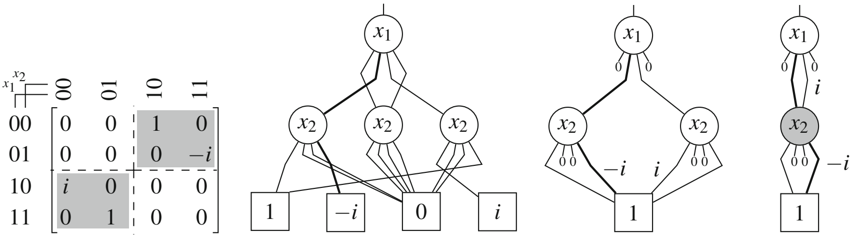
The partitioning in Eq. (4.1) can readily be extended to the multiple-valued case as

$$\mathbf{U} = \begin{bmatrix} \mathbf{U}_{00} & \mathbf{U}_{01} & \cdots & \mathbf{U}_{0,r-1} \\ \mathbf{U}_{10} & \mathbf{U}_{11} & \cdots & \mathbf{U}_{1,r-1} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{U}_{r-1,0} & \mathbf{U}_{r-1,1} & \cdots & \mathbf{U}_{r-1,r-1} \end{bmatrix} \tag{4.2}$$

where matrix $\mathbf{U}$ has dimension $r^n \times r^n$ and the r^2 sub-matrices each have dimension $r^{n-1} \times r^{n-1}$.

This structure allows for representing equal sub-matrices by shared parts of the diagram. However, further vertex sharing and, thus, reduction is possible by extracting common multipliers as illustrated in the following

Example 4.2 Consider the matrix in Fig. 4.2a. Applying the recursive partitioning above would yield a tree as depicted in Fig. 4.2b: with a root labelled x_1, three internal vertices labelled x_2 (the two zero blocks sharing the same vertex), and four terminal vertices (one for each value 0, 1, i, $-i$). By extracting common multipliers



(a) Transformation matrix (b) DD, multiple terminals (c) DD, edge weights (d) QMDD

Fig. 4.2 Representations of a 2-qubit quantum operation

and using them as edge weights, we can reduce the tree to the graph in Fig. 4.2c. Then, the four terminal vertices in the tree can be collapsed to a single terminal vertex with value 1. The actual value of a matrix entry is the product of the weights on the path corresponding to the sub-matrix partitioning through the matrix that leads to the entry. For example, the highlighted matrix entry $-i$ in Fig. 4.2a corresponds to the path highlighted in bold in Fig. 4.2c. Since we are taking the product of weights, edges with 0 weight can point directly to the terminal vertex. For the sake of better readability, we show 0 weight edges as stubs and do not extend them to the terminal. For the same reason, we do not explicitly indicate the edge weight if it is equal to 1.

However, even more reduction is possible: The matrix in Fig. 4.2a has two structurally equivalent sub-matrices (highlighted in grey) which differ only by a common multiplier. These correspond to the two vertices labelled x_2 in Fig. 4.2c. By factoring out i from the lower left sub-matrix as a weight, we can represent the two sub-matrices by one shared structure as shown in Fig. 4.2d.

Clearly, the choice of the edge weights is not unique, though it is central for our objective to share common matrix structures and to efficiently deal with sub-matrices entirely composed of 0's. To this end, we introduce the concept of normalizing a (sub-)matrix:

- To achieve an optimal structure sharing, we want to store only normalized forms of the sub-matrices. For this purpose, the normalized form $\widehat{\mathbf{M}}$ of a matrix $\mathbf{M}$ shall be the same for any (non-zero) multiple of $\mathbf{M}$, i.e.

$$\widehat{\mathbf{M}} = \widehat{\alpha\mathbf{M}} \text{ for all } \alpha \neq 0. \tag{4.3}$$

- Moreover, as the terminal vertex represents the matrix $[1]_{1\times 1}$, this matrix shall be the normalized form of any 1×1 matrix, i.e.

$$\widehat{[\alpha]}_{1\times 1} = [1]_{1\times 1} \text{ for all } \alpha \in \mathbb{C}. \tag{4.4}$$

To obtain normalized forms we need a normalization scheme to determine which common multiplier $N(\mathbf{M})$ (*normalization factor*) is extracted from a matrix $\mathbf{M}$, such that

$$\mathbf{M} = N(\mathbf{M}) \cdot \widehat{\mathbf{M}}. \tag{4.5}$$

Formally, interpreting the normalization scheme as a mapping N from the set of matrices to the set of normalization factors (complex numbers), we require the following properties:

1. $N(\alpha\mathbf{M}) = \alpha N(\mathbf{M})$ for any complex-valued matrix $\mathbf{M}$ and any complex-valued number α.
2. $N([\alpha]_{1\times 1}) = \alpha$ for any complex number α.
3. $N(\mathbf{M}) = 0 \Leftrightarrow$ all entries in $\mathbf{M}$ are zero.

Example 4.3 A very simple normalization scheme is obtained by defining the normalization factor of a matrix to be (1) the first non-zero entry of the matrix that is found when scanning the matrix row by row and entry by entry or (2) zero if no non-zero entry is found. It is readily observed that this scheme indeed satisfies the required properties.

Note that property (3) allows us to directly compute the normalized form of non-zero matrices from Eq. (4.5) as

$$\widehat{\mathbf{M}} = \frac{1}{N(\mathbf{M})} \cdot \mathbf{M} \quad (\mathbf{M} \neq 0) \tag{4.6}$$

and one can easily check that this definition satisfies the desired properties of normalized forms given in Eqs. (4.3) and (4.4). For $\mathbf{M} = [0]_{k\times k}$ we could choose the normalized form arbitrarily according to Eq. (4.5), but for consistency with the case $k = 1$ we set

$$\widehat{[0]}_{k\times k} = [1]_{k\times k} \text{ for any } k. \tag{4.7}$$

Recall that our aim was to identify structurally equivalent sub-matrices and extract common multipliers in order to obtain as much structure sharing as possible by using normalized forms of the sub-matrices. For a formal description of how the above normalization is integrated into the decomposition process, we make use of the *Khatri-Rao* product (introduced in [KR68, ZYC02]). The Khatri-Rao product provides for a mathematical description of the QMDD vertex decomposition in analogy to the Shannon decomposition for BDDs (cf. Eq. 3.2) and other BDD-like structures such as the QDD, XQDD or QuIDD. It is defined as follows:

Definition 4.1 The Khatri-Rao (KR) product $*$ is a particular type of matrix multiplication that operates over matrix partitions and can be described in terms of the tensor product, denoted by $\otimes$, as $\mathbf{A} * \mathbf{B} = \left[\mathbf{A}_{ij} \otimes \mathbf{B}_{ij}\right]_{ij}$.

Example 4.4 Assume two matrices **A** and **B** are of the form

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \hline \mathbf{A}_{10} & \mathbf{A}_{11} \end{array}\right]_{2k\times 2k} \quad \mathbf{B} = \left[\begin{array}{c|c} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \hline \mathbf{B}_{10} & \mathbf{B}_{11} \end{array}\right]_{2l\times 2l}.$$

Then, the KR product $\mathbf{A} * \mathbf{B}$ becomes

$$\mathbf{A} * \mathbf{B} = \left[\begin{array}{c|c} \mathbf{A}_{00} \otimes \mathbf{B}_{00} & \mathbf{A}_{01} \otimes \mathbf{B}_{01} \\ \hline \mathbf{A}_{10} \otimes \mathbf{B}_{10} & \mathbf{A}_{11} \otimes \mathbf{B}_{11} \end{array}\right]_{2(k\cdot l)\times 2(k\cdot l)}.$$

Note that, according to the definition of the tensor product, the block $\mathbf{A}_{ij} \otimes \mathbf{B}_{ij}$ has dimension $k \cdot l \times k \cdot l$ assuming that $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ have dimensions $k \times k$ and $l \times l$, respectively.

Conceptually, the proposed normalization from Eq. (4.5) is to be applied to each sub-matrix separately in the course of the partitioning and the extracted normalization factors (edge weights) are to be applied only to the particular sub-matrix.

This can be formally described using the KR matrix product by the following decomposition relationship:

$$\mathbf{M} = \mathbf{W}(\mathbf{M}) * \widehat{\mathbf{M}} \tag{4.8}$$

In this central equation,

- $\mathbf{M}$ is the original $r^n \times r^n$ matrix to be decomposed,
- $\mathbf{W}(\mathbf{M}) = [N(\mathbf{M}_{ij})]_{0 \leq i,j < r}$ is the $r \times r$ matrix containing all scalar factors (weights) which are extracted from the sub-matrices $\mathbf{M}_{ij}$, and
- $\widehat{\mathbf{M}}$ has dimension $r^n \times r^n$, partitioned into the r^2 normalized sub-matrices $\widehat{\mathbf{M}}_{ij}$ of dimension $r^{n-1} \times r^{n-1}$.

Here, the KR product essentially ensures that the extracted weights will be associated with the appropriate sub-matrix.

Example 4.5 For $r = 2$, the decomposition is given by the equation

$$\mathbf{U} = \left[\begin{array}{c|c} N(\mathbf{U}_{00}) & N(\mathbf{U}_{01}) \\ \hline N(\mathbf{U}_{10}) & N(\mathbf{U}_{11}) \end{array}\right] * \left[\begin{array}{c|c} \widehat{\mathbf{U}}_{00} & \widehat{\mathbf{U}}_{01} \\ \hline \widehat{\mathbf{U}}_{10} & \widehat{\mathbf{U}}_{11} \end{array}\right].$$

The decomposition is represented by a vertex labelled by the corresponding partition variable and, as shown in Fig. 4.2c on page 37, the extracted normalization factors are attached to the edges that point to the vertices representing the decomposition of the corresponding sub-matrices. Note, however, that the diagram in Fig. 4.2c does not correspond to a proper decomposition since the same weight is extracted for the two non-zero sub-matrices though they differ by a constant multiplier only. Conversely, the diagram in Fig. 4.2d results from a proper decomposition using normalized forms as they would result from using the normalization scheme outlined in Example 4.3.

4.2 Formal Definition

Based on the concepts described above, we now present a formal definition.

Definition 4.2 A *Quantum Multiple-Valued Decision Diagram* (QMDD) is a representation of an $r^n \times r^n$ complex matrix as a rooted directed acyclic graph with a set V containing two types of vertices: a single terminal vertex and zero or more non-terminal vertices. The terminal vertex, labelled 1, represents the matrix $[1]_{1\times1}$. It has no outgoing edges. Each non-terminal vertex is labelled by an r-valued variable and has r^2 outgoing edges, each pointing to a vertex in V. Each non-terminal vertex denotes the partitioning of a matrix by the application of Eq. (4.8) and, thus, has r^2 outgoing edges e_p, $0 \leq p < r^2$, which are labelled $00, 01, \cdots, r-1r-1$ and have associated complex weights $w(e_{00})$, $w(e_{01}), \ldots, w(e_{r-1,r-1})$. There is an initial edge with no source vertex which points to the root vertex and has an associated complex weight representing the normalization factor of the represented matrix according to Eq. (4.5). We term this the *root edge*.

A QMDD has the following properties:

1. The non-terminal vertex labels (variables) are *ordered* which means that if a non-terminal vertex with label x_i has an edge pointing to a non-terminal vertex labelled x_j, x_i is the more significant variable in the row and column labelling of the matrix represented by the QMDD.
2. A QMDD is *reduced* which means (a) there is no non-terminal vertex where all outgoing edges point to the same vertex and have the same weight, and (b) all vertices are unique and no two non-terminal vertices represent the same sub-matrix, i.e. are labelled by the same variable and have all corresponding edges pointing to the same vertex with the same weight.
3. Any constant (sub-)matrix, regardless of its size, is represented as an edge pointing directly to the terminal vertex.

It is important to note that a QMDD is a recursive representation as every edge in the QMDD can be seen as the root edge of the QMDD representation of a sub-matrix. This is a key observation used to describe how matrix operations are implemented with QMDDs.

Another key factor in interpreting QMDDs concerns the variables labelling the non-terminal vertices. Each non-terminal vertex is labelled by an r-valued variable and that variable labels both the rows and columns of the matrix. The corresponding matrix partitioning divides the rows into r sections and the columns into r sections for a total of r^2 sub-matrices, see Eq. (4.8).

The notion of variable ordering introduced in Definition 4.2 means that if the matrix row and column variables are ordered by a function $index()$ such that $index(x_i) < index(x_j)$ if, and only if, x_i precedes x_j, then the QMDD satisfies the following two properties:

- Each variable appears at most once on each path from the root vertex to the terminal vertex.
- An edge from a non-terminal vertex labelled x_i points to a non-terminal vertex labelled x_j, $index(x_j) > index(x_i)$ or to the terminal vertex. Hence, the variable indices along any path from the root to the terminal satisfy the order imposed by $index()$ and that order corresponds to the variables order for the matrix and column labelling from most to least significant.

Another important observation is that all edges with weight 0 (also called *0-edges* in the following) point directly to the terminal vertex. This is because they represent the normalization factor 0 which, by definition, may only occur if the edge represents a sub-matrix $[0]_{k\times k}$. Consequently, the edge points to a vertex representing the normalized form, which is $[1]_{k\times k}$ by Eq. (4.7). However, since the QMDD is reduced, there may not be a non-terminal vertex representing this matrix, because all its outgoing edges would point to the same vertex with the same weight. Similarly, any constant sub-matrix is represented by an edge directly pointing to the terminal vertex with the appropriate weight, as stated in property 3 of Definition 4.2. In summary, these structural sharings allow for a compact representation of the corresponding quantum functionality.

4.3 Canonicity

In addition to the desired feature of allowing for a compact representation of functionality, the uniqueness of a function representation is also of interest. In particular, this is of significant importance for many application areas such as equivalence checking. QMDDs satisfy this criteria, i.e. any normalized QMDD is *canonic* with respect to a given variable order and normalization scheme as is proven next.

Theorem 4.1 *Given a normalization scheme N, the corresponding QMDD representation of any complex-valued $r^n \times r^n$ matrix is unique up to variable order.* □

Proof The proof is by contradiction, i.e. we assume there exists a matrix M that has two different QMDD representations G_1 and G_2 which adhere to the same normalization scheme and variable order, and show that such a matrix may not exist. Recall that, according to the decomposition in Eq. (4.8), any non-terminal vertex of a QMDD represents the normalized form of the corresponding sub-matrix. Consequently, each vertex in G_1 has an equivalent in G_2 and vice versa. Roughly speaking, both representations employ the same set of vertices. Since G_1 and G_2 are different, one representation must include an edge e_1 that is not present in the other representation. This edge must have a different weight or a different target compared to the corresponding edge e_2 in the other representation. However, both cases may not happen as e_1 and e_2 have the same source and, thus, represent the same sub-matrix for which the normalization factor (edge weight) and normalized form (target vertex) are uniquely defined. This contradicts the assumption that there are two representations for M.

Clearly, QMDD representations may be different for different variable orders. Moreover, even for a fixed order, the resulting QMDDs may differ for different normalization schemes as well. However, the following theorem shows that the resulting diagram structure is always the same.

Theorem 4.2 *Given a complex-valued $r^n \times r^n$ matrix and a fixed variable order, the corresponding QMDD representations are isomorphic for any two normalization schemes.* □

Proof Consider two QMDD representations of the same matrix corresponding to different normalization schemes. Again, the non-terminal vertices in both representations represent normalized forms of the corresponding sub-matrices. Since QMDDs are reduced, a single vertex exists for each occurrence of a sub-matrix and its multiples. Consequently, there is a bijection between the vertex sets of both representations that identifies vertices representing the same class of multiples of a sub-matrix. We want to show that this map is indeed a graph isomorphism, i.e. it also preserves edges. To do that, we have to show that (each pair of) corresponding edges indeed point to corresponding vertices. For this purpose, consider an arbitrary pair of corresponding edges, i.e. two outgoing edges e_{ij} and $e_{ij}^{'}$ (both labelled ij) from corresponding vertices from both representations. Since the source vertices correspond to the same class of multiples of a sub-matrix M, both vertices represent a multiple of M: its respective normalized form. Consequently, both edges represent multiples of the same sub-matrix M_{ij} of M and, thus, point to corresponding vertices (representing the respective normalized forms of M_{ij}). Overall, the bijection between the vertex sets preserves edges and is, therefore, a graph isomorphism.

The two theorems proven in this section indeed show that QMDDs provide unique representations (up to variable order) while the resulting structure does not depend on which normalization scheme is actually used. This is beneficial since it ensures the highest possible structure sharing and, hence, the most compact representation regardless of how exactly the weights are determined. Thus, even simple normalization schemes as the one discussed in Example 4.3 are sufficient. This is even more important as it is infeasible for large matrices to compute sophisticated normalization factors in a top-down fashion and, as a consequence, normalization of QMDDs is practically performed in a bottom-up fashion as we can see in the following section.

4.4 Construction and Manipulation

Thus far, we showed that the proposed QMDDs provide a compact and canonic representation of arbitrary quantum functionality. However, to be of practical use, QMDDs must additionally allow for an efficient construction and manipulation. These issues are covered in this section. More precisely, we discuss how essential matrix operations can efficiently be performed on QMDDs and show that QMDD representations for elementary quantum operations can easily be derived. These issues are exemplarily illustrated by means of constructing a QMDD for a given quantum circuit.

4.4.1 Normalization

Before we describe how essential matrix operations can efficiently be performed directly on the QMDD structure, we first need to consider how normalization, the key for QMDDs being canonic representations, is achieved in practice. For larger matrices it is infeasible to determine the edge weights, i.e. the normalization factors as they arise from the partitioning of a matrix as given in Eq. (4.8), in a top-down fashion. As the QMDD is rather built bottom-up, also the edge weights have to be determined this way. More precisely, by performing *vertex normalization* of each non-terminal vertex when it is constructed, edge weights are determined subsequently, finally making the QMDD a canonic representation.

Consider a QMDD non-terminal vertex v with outgoing edges e_p (where $p \geq 0$ and $p \leq r^n - 1$) that are pointing to sub-matrices $\mathbf{U}_{00}, \mathbf{U}_{01}, \ldots, \mathbf{U}_{r-1,r-1}$, respectively, and let $w(e_p)$ denote the weight on edge e_p.

Definition 4.3 The non-terminal vertex v is *normalized* if $w(e_j) = 1$ for the lowest j for which $w(e_j) \neq 0$.

It is straightforward to normalize a given QMDD non-terminal vertex v according to the above definition: a single normalization factor equal to the $w(e_j)$ for the lowest j for which $w(e_j) \neq 0$ is identified and the weights on all edges leading from the vertex are divided by that factor. Note that the existence of at least one edge with non-zero weight is for sure as we have shown earlier that edges with weight 0 point directly to the terminal vertex and, since the QMDD is reduced, there may not be a vertex with all edges pointing to the same vertex with the same weight.

The vertices that v points to are not affected, but the edges pointing to v have to be adjusted by multiplying their weights by the normalization factor. An example for the binary case is shown in Fig. 4.3: normalizing the vertex on the left leads to the vertex shown on the right. In this case the normalization factor is $-i$.

Note that, as QMDDs are built and normalized bottom-up, this propagation of normalization factors to the top can easily be performed without possibly destroying the normalization of existing vertices. Finally, note that this procedure indeed establishes a normalization scheme similar to the one defined in Example 4.3, though now the matrix is scanned for non-zero entries in a more elaborate fashion.

Example 4.6 For vertex normalization, the order in which a 4 × 4 matrix is being scanned for non-zero entries is given by

$$\text{(a)} \left[\begin{array}{cc|cc} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \\ \hline 9 & 10 & 13 & 14 \\ 11 & 12 & 15 & 16 \end{array}\right] \text{ and (b) } \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right]$$

for (a) vertex normalization and (b) the normalization scheme from Example 4.3 on page 48, respectively.

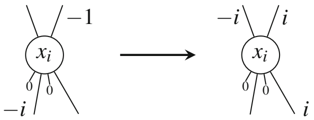


Fig. 4.3 Normalizing a vertex

4.4.2 Matrix Operations

Knowing how a normalized representation is actually achieved, we can now describe how matrix operations can be computed directly on a QMDD. We do this for the three common operations addition, direct multiplication, and tensor or Kronecker multiplication. It is noted that the calculation of the tensor product and the Kronecker product are identical multiplicative operations over the class of unitary matrices used to represent quantum system evolutions. Other operations are readily implemented using the methods described in the following.

Implementing these operations uses the fact that a QMDD is a recursive representation that represents a matrix as a composition of sub-matrices which are in turn represented by smaller sub-matrices. This allows the operations to be expressed in terms of operations on sub-matrices. For example, matrix addition for the binary case can be expressed as the addition of sub-matrices:

$$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} + \begin{bmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{B}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{00} + \mathbf{B}_{00} & \mathbf{A}_{01} + \mathbf{B}_{01} \\ \mathbf{A}_{10} + \mathbf{B}_{10} & \mathbf{A}_{11} + \mathbf{B}_{11} \end{bmatrix}.$$

Likewise, matrix multiplication for the binary case is expressible as

$$\begin{bmatrix} \mathbf{A}_{00} & \mathbf{A}_{01} \\ \mathbf{A}_{10} & \mathbf{A}_{11} \end{bmatrix} \begin{bmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{B}_{11} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{00}\mathbf{B}_{00} + \mathbf{A}_{01}\mathbf{B}_{10} & \mathbf{A}_{00}\mathbf{B}_{01} + \mathbf{A}_{01}\mathbf{B}_{11} \\ \mathbf{A}_{10}\mathbf{B}_{00} + \mathbf{A}_{11}\mathbf{B}_{10} & \mathbf{A}_{10}\mathbf{B}_{01} + \mathbf{A}_{11}\mathbf{B}_{11} \end{bmatrix}.$$

In order to formalize these operations on the QMDD, the following definitions are applied.

Definition 4.4 Given an edge e, we use $w(e)$ to denote the weight on e, $v(e)$ to denote the vertex e points to, $x(e)$ to denote the variable that labels the vertex e points to (not defined for the terminal vertex), $E_i(e)$ to denote the ith edge out of the vertex that e points to, and $T(e)$ to denote a Boolean test that is true if e points to the terminal vertex.

Furthermore, we assume that the variables adhere to the same order in all considered QMDDs and we shall use $\prec$ to denote the fact that one variable precedes another and, hence, appears closer to the terminal vertex in the QMDD. For generality, we consider the multiple-valued case where r is the radix, i.e. every non-terminal vertex has r^2 outgoing edges (for the Boolean case, r can simply be set to $r = 2$).

Having that, matrix operations can be conducted on QMDDs as follows noting that a matrix is uniquely identified by the root edge for the QMDD.
Matrix Addition: Let e_0 and e_1 be the root edges of two QMDD (matrices) to be added. The procedure is recursive and involves the following steps:

1. If $T(e_1)$, swap e_0 and e_1.
2. If $T(e_0)$,
    a. if $w(e_0) = 0$, the result is e_1;
    b. if $T(e_1)$, the result is an edge pointing to the terminal vertex with weight $w(e_0) + w(e_1)$.
3. If $x(e_0) \prec x(e_1)$, swap e_0 and e_1.
4. For $i = 0, 1, \ldots, r^2 - 1$
    a. Create an edge p pointing to $v(E_i(e_0))$ with weight $w(e_0) \cdot w(E_i(e_0))$.
    b. If $x(e_0) = x(e_1)$, create an edge q pointing to $v(E_i(e_1))$ with weight $w(e_1) \cdot w(E_i(e_1))$, else set $q = e_1$.
    c. Recursively invoke this procedure to add p and q giving z_i.
5. The result is an edge pointing to a vertex labelled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, \ldots, r^2 - 1$. This vertex and the edge pointing to it are normalized.

Matrix Multiplication: Let e_0 and e_1 be the root edges of two QMDD (matrices) to be multiplied. The procedure is recursive and involves the following steps:

1. If $T(e_1)$, swap e_0 and e_1.
2. If $T(e_0)$, then
    a. if $w(e_0) = 0$, the result is e_0;
    b. if $w(e_0) = 1$, the result is e_1;
    c. otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \cdot w(e_1)$.
3. If $x(e_0) \prec x(e_1)$, swap e_0 and e_1.
4. For $i = 0, r, 2r, \ldots, (r - 1)r$

```
    For $j = 0, 1, \ldots, r - 1$
        Set $z_{i+j}$ to be a 0-edge.
        For $k = 0, 1, \ldots, r - 1$
            (i) Create an edge $p$ pointing to $v(E_{i+k}(e_0))$
                with weight $w(e_0) \cdot w(E_{i+k}(e_0))$.
            (ii) If $x(e_0) = x(e_1)$, create an edge $q$
                pointing to $v(E_{j+r \cdot k}(e_1))$ with weight $w(e_1) \cdot w(E_{j+r \cdot k}(e_1))$,
                else set $q = e_1$.
            (iii) Recursively invoke this procedure to multiply the QMDD pointed to
                by $p$ and $q$ and then use the procedure above to add
                the result to the QMDD pointed to by $z_{i+j}$.
                The result of the addition replaces $z_{i+j}$.
```

5. The result is an edge pointing to a vertex labelled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, \ldots, r^2 - 1$. This vertex and the edge pointing to it are normalized.

Kronecker Product: Let e_0 and e_1 be the root edges of two QMDD (matrices) for which we want to compute the Kronecker product $\mathbf{A} \otimes \mathbf{B}$ (note that this operation is not commutative). For the application considered here, the selection variables for

B precede the selection variables for **A**. This greatly reduces the complexity of the algorithm for computing the Kronecker product of two QMDD.

The procedure is recursive and involves the following steps:

1. If $T(e_0)$ then
   a. if $w(e_0) = 0$, the result is e_0;
   b. if $w(e_0) = 1$, the result is e_1;
   c. otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \cdot w(e_1)$.
2. For $i = 0, 1, \ldots, r^2 - 1$
   Recursively invoke this procedure to find the Kronecker product of $E_i(e_0)$ and e_1 setting z_i to the result.
3. The result is an edge pointing to a vertex labelled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, \ldots, r^2 - 1$. This vertex and the edge pointing to it are normalized.

Performing these matrix operations directly on the QMDD structure makes them very effective which we exemplarily illustrate in the next section by means of constructing a QMDD for a given quantum circuit.

4.4.3 Construction

For actually using QMDDs in quantum logic design it is important to be able to efficiently construct a QMDD representing the desired quantum functionality. General quantum functionality is usually either given (a) in terms of an abstract *quantum algorithm* which describes a series of computational steps or complex quantum operations (modules) to be conducted or (b) in terms of a *quantum circuit* consisting of a cascade of elementary quantum operations (so-called *quantum gates*) that form a more complex operation.

For quantum algorithms as well as circuits, the representation/description of the overall functionality is successively built from functional descriptions/representations of the individual parts (modules or gates). More precisely, for a cascade of modules/gates $g_1 g_2 \ldots g_l$ where the transformation for module/gate g_i is defined by matrix $\mathbf{M}_i$, the transformation for the complete algorithm/circuit is given by the direct matrix product $\mathbf{M}_l \cdot \mathbf{M}_{l-1} \cdot \ldots \cdot \mathbf{M}_1$. Note that the order of the matrices has to be reversed to achieve the correct order of applying the modules/gates (first g_1, then g_2, etc.). To construct this matrix product, the QMDDs for the single modules/gates simply have to be multiplied using the QMDD-based algorithm for matrix multiplication presented above. Consequently, for the remainder of this section we focus on how the QMDD representations for elementary quantum gates can be constructed efficiently.

Again for generality, we consider the multiple-valued case. Assume, as above, the variable order $x_1 \succ x_2 \succ \ldots \succ x_n$ from the root vertex towards the terminal vertex. A gate g is specified by the $r \times r$ base transition matrix **B**, the target qudit x_t and a possible empty set of control qudits $C \subset \{x_1, \ldots, x_n\}$ (with $x_t \notin C$) together with a map $\alpha \colon C \to \{|0\rangle, \ldots, |r-1\rangle\}$ which describes the activating values, i.e. basis states, of each control qudit. The QMDD for a quantum gate is built variable by variable (qudit by qudit) in a bottom-up fashion from the terminal to the root vertex. In order to indicate which set of variables has been processed so far, we use the notation $\mathbf{M}_{\{x_k,\ldots,x_n\}}$. Moreover, for the sake of an easier reference, we term those edges of a QMDD vertex *diagonal* that correspond to a $|i\rangle \to |i\rangle$ mapping ($i = 0, \ldots, r-1$) and the remaining edges *off-diagonal*.

Although it is possible to construct the QMDD for the gate in a single run (as roughly sketched in [MT06]), for a better understanding we construct two QMDDs representing the cases that the gate is active (all control qudits are in their activating state) or inactive (at least one control qudit is not).[2] By adding these QMDDs, the actual QMDD for the gate results.

Case "gate is active", i.e. the base transition $\mathbf{B}$ is performed on qudit x_t if, and only if, all controls are in their activating state. All other qudits preserve their original state.

Consequently, the QMDD for the active case contains all (non-zero) paths of the final QMDD for which all decision variables (qudits) except for the target have an activating assignment.

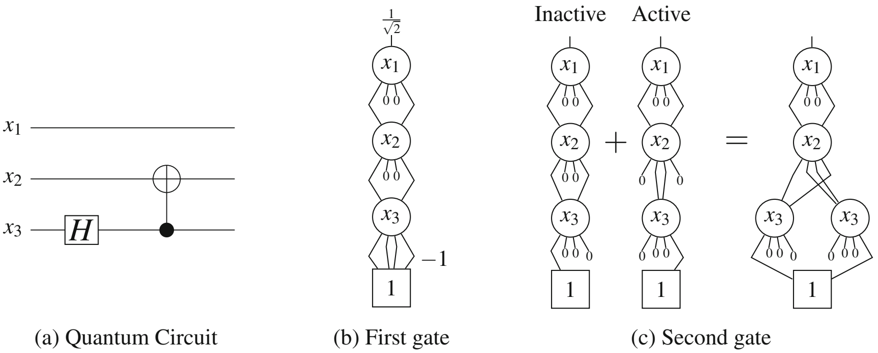
In order to have a valid starting point, we begin at the terminal level with an edge pointing to the terminal vertex with weight 1, i.e. $\mathbf{M}_{\emptyset} = [1]_{1\times 1}$.[3] Afterwards, the qudits are processed bottom-up. If the current qudit x_c

- is neither a control nor the target, i.e. $x_c \neq x_t, x_c \notin C$, the gate is active regardless of the qudit's state. Consequently, at the matrix level the result is $\mathrm{id}_{\mathbf{r}\times\mathbf{r}} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$ which corresponds to a QMDD vertex labelled x_c where all diagonal edges point to the existing QMDD and all remaining edges are 0-edges.
- is a control, i.e. $x_c \in C$, the gate is only active for one control value $|i\rangle = \alpha(x_c)$. Consequently, the result is a vertex labelled x_c with only 0-edges except for the edge $|i\rangle \to |i\rangle$ which is pointing to the existing QMDD.
- is the target, i.e. $x_c = x_t$, the base transition is performed. Consequently, the result is $\mathbf{B} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$, i.e. a vertex labelled x_t with all edges pointing to the existing QMDD with the corresponding edge weight taken from the base transition matrix $\mathbf{B}$ (if a weight is zero, the corresponding edge is a 0-edge directly pointing to the terminal).

During this construction, the QMDD is normalized as described in Sect. 4.4.1.

[2] Without loss of generality, we consider only basis states of the underlying quantum system, i.e. each qudit is assumed to be in one of its basis states. Due to the linearity of quantum operations, these are sufficient to construct the corresponding transformation matrix which yields the correct behaviour also for the case of superposed input states.

[3] The appropriate weights of the base transition will be incorporated later.



(a) Quantum Circuit (b) First gate (c) Second gate

Fig. 4.4 A quantum circuit built from elementary quantum gates

Example 4.7 Consider the QMDD in Fig. 4.4b which represents the first gate of the quantum circuit ($r = 2$) shown in Fig. 4.4a. As this gate does not have any controls, it is always active and, thus, it suffices to build the QMDD representing the active part. We start with an edge to the terminal vertex with weight 1. As the bottommost qubit is already the target qubit, all edges of the x_3-vertex point directly to this terminal with the appropriate weight of the Hadamard transformation matrix $\mathbf{H} = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right)$. Note that normalization will propagate the common multiplier $\frac{1}{\sqrt{2}}$ of this matrix to the root edge. The remaining qubits are neither control nor target. Thus, vertices representing an identity mapping of these qubits are inserted.

The QMDD for the inactive case is constructed similarly.

Case "gate is inactive", i.e. the identity transition is performed on qudit x_t since at least one control is not in its activating state. All qudits preserve their original state, i.e. none but diagonal edges are populated at all.
Consequently, the QMDD for the inactive case contains all (non-zero) paths of the final QMDD for which at least one decision variable (qudit) does not have an activating assignment.
However, when constructing the QMDD in a bottom-up fashion, we always use the hypothesis that all controls above the current qudit are in their activating states and at least one control below is not.
To make sure that this hypothesis gives the correct result even for the bottommost control (for which no inactive control may exist below), we start at the terminal level with an edge pointing to the terminal vertex with weight 0, i.e. $\mathbf{M}_{\emptyset} = [0]_{1\times 1}$. This ensures that all edges corresponding to the activating value of this bottommost control are 0-edges.

The remaining qudits are processed as follows. If the current qudit x_c

- is neither a control nor the target, i.e. $x_c \neq x_t, x_c \notin C$, the gate is inactive regardless of the qudit's state. Consequently, the result at the matrix level is $\mathrm{id}_{\mathbf{r}\times\mathbf{r}} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$ which corresponds to a QMDD vertex labelled x_c where all diagonal edges point to the existing QMDD and all remaining edges are 0-edges.
- is a control, i.e. $x_c \in C$, the gate is definitely inactive for all but one control value $|i\rangle = \alpha(x_c)$. For the latter, the activity of the gate depends on the remaining qudits. Consequently, the result is a vertex with all diagonal edges pointing to the k-fold tensor product $\mathrm{id}_{\mathbf{r}\times\mathbf{r}}{}^{\otimes k}$ (nothing happens to all k qudits below the current one) except for the edge $|i\rangle \rightarrow |i\rangle$. The latter handles the case that the qudit is in its activating state and is pointing to the existing QMDD $\mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$.[4] All off-diagonal edges are 0-edges.
- is the target, i.e. $x_c = x_t$, the identity transformation is performed on the target. Consequently, the result is $\mathrm{id}_{\mathbf{r}\times\mathbf{r}} \otimes \mathbf{M}_{\{x_{c+1},\ldots,x_n\}}$ like in the unconnected case.

Example 4.8 The QMDDs for the circuit's second gate is shown in Fig. 4.4c.

For the inactive part, we start with a 0-edge. For the control on x_3, we construct a vertex which uses this 0-edge as e_{11} and for which the other diagonal edge e_{00} represents the identity $\mathrm{id}_{\mathbf{2}\times\mathbf{2}}{}^{\otimes 0} = [1]_{1\times 1}$, i.e. it points to the terminal vertex with weight 1. As x_3 is the only control, we simply add vertices representing an identity mapping for the remaining qubits.

For the active part, we start with an edge to the terminal vertex which becomes the e_{11} edge of the x_3-vertex, as the activating state of x_3 is $|1\rangle$. For the target qubit x_2 with the base transition matrix $\mathbf{X} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, an x_2-vertex is added. For this vertex, both off-diagonal edges point to the x_3-vertex constructed before (with weight 1 as the corresponding entry in $\mathbf{X}$ is 1) and both diagonal edges are 0-edges (as the corresponding entry in $\mathbf{X}$ is 0). Last, but not least, for the unconnected qubit x_1 a vertex representing its identity mapping is added. Finally, by adding the QMDDs for the inactive and active part, we obtain the actual QMDD for the CNOT gate.

Overall, the resulting QMDDs for the active as well as the inactive part of the gate are linear in the number of variables—regardless of the complexity of the gate under consideration. Both QMDDs can be constructed in parallel while iterating through the variables in a bottom-up fashion. In addition, they describe disjoint parts of the gate matrix, while they are padded with zeros outside of that particular part. Consequently, their sum can be computed in linear time and will also be linear in size. In fact, there are only trivial additions where at least one of the summands is a 0-matrix and, as already recognized in [MT06], the addition could be saved entirely, such that the whole construction could be performed in a single pass from the terminal to the root vertex with no backtracking or recursion. Either way, QMDD representations for single gates can be computed very efficiently and the potentially rather expensive part of constructing a QMDD representation for quantum algorithms or quantum

[4] If there is no further control below the current qudit, the gate inactivity is ensured by choosing a 0-edge as the initial QMDD.

circuits (as well as any other quantum logic representation) is given by the (QMDD-based) matrix multiplication that is required to concatenate the representations of single modules/gates.

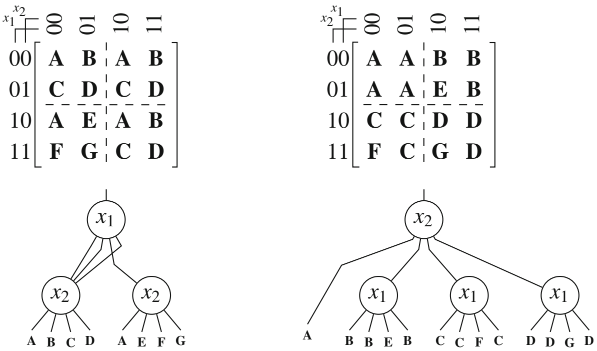
4.5 Changing the Variable Order

As we have seen so far, QMDDs offer a compact and canonic representation of quantum logic and promise an efficient handling and manipulation of quantum functionality. However, it is a common observation for decision diagrams that the variable order may have a large impact on the size of the representation and, hence, is crucial for the overall efficiency. In this section, we will first analyze in which way variable re-ordering can possibly reduce the vertex count of QMDDs. After that, we illustrate which obstacles arise when performing local modifications (like variable interchanges) on QMDDs, provide a solution to this problem and, finally, present the resulting interchange scheme for adjacent variables in QMDDs. This scheme enables to use many established re-ordering techniques for decision diagrams that rely on interchanges of adjacent variables like, e.g. sifting or window permutation [Rud93]. An experimental evaluation how variable re-ordering affects the diagram size is presented later in Sect. 4.6.

4.5.1 Shared Vertices and Skipped Variables

Changing the variable order of a QMDD can be interpreted as permuting rows and columns of the corresponding matrix. In fact, this change can have a significant impact on structural equivalence and, hence, on shared vertices. In some cases it allows one to join identical blocks which leads to redundant vertices for which all outgoing edges would point to the same vertex with the same weight. However, redundant vertices are, by definition, not allowed in QMDDs and will be represented by an edge that *skips* the particular variable (say x_j) and points to a vertex which is labelled by a variable that succeeds x_j in the variable order. Both are illustrated by means of the following:

Example 4.9 Consider the matrices in Fig. 4.5. Suppose $\mathbf{A}, \ldots, \mathbf{G}$ are mutually different sub-matrices of the same size $2^k \times 2^k$. Both matrices represent the same functionality (though employing a different variable order where x_1 and x_2 change places) and each matrix can be obtained from the other by variable interchange, i.e. by swapping the 01 and 10 rows and columns. The matrix on the left has three identical blocks which can be represented by a shared x_2-vertex. The matrix on the right does not offer shared vertex compression, but the top-left sub-matrix consists of four identical blocks which gives rise to a skipped variable.



$$\begin{array}{c|cccc} x_1 \backslash x_2 & 00 & 01 & 10 & 11 \\ \hline 00 & \mathbf{A} & \mathbf{B} & \mathbf{A} & \mathbf{B} \\ 01 & \mathbf{C} & \mathbf{D} & \mathbf{C} & \mathbf{D} \\ 10 & \mathbf{A} & \mathbf{E} & \mathbf{A} & \mathbf{B} \\ 11 & \mathbf{F} & \mathbf{G} & \mathbf{C} & \mathbf{D} \end{array} \qquad \begin{array}{c|cccc} x_2 \backslash x_1 & 00 & 01 & 10 & 11 \\ \hline 00 & \mathbf{A} & \mathbf{A} & \mathbf{B} & \mathbf{B} \\ 01 & \mathbf{A} & \mathbf{A} & \mathbf{E} & \mathbf{B} \\ 10 & \mathbf{C} & \mathbf{C} & \mathbf{D} & \mathbf{D} \\ 11 & \mathbf{F} & \mathbf{C} & \mathbf{G} & \mathbf{D} \end{array}$$

Fig. 4.5 Variable interchange: structural equivalence and skipped variables

A special case of skipped variables are blocks of zero which result in a 0-edge that directly points to the terminal vertex and skips all succeeding variables. It can be shown that 0-edges are the only type of skipped variables that can occur for QMDDs which represent a reversible Boolean function (permutation matrix) [MFT07]. This is because skipped variables always indicate identical sub-matrices and since in permutation matrices there is a single non-zero entry in each row and column, the identical sub-matrices can only be blocks of zeros. From this perspective, the aim of QMDD minimization for reversible operations can hence be described as changing the position of the non-terminal vertices such that those vertices that have more outgoing 0-edges are closer to the root vertex. Corresponding metrics for guiding the re-ordering process, based upon the ratio of the number of non-zero weight edges versus the total number of vertices, are reported in [FTM09].

However, in general quantum logic there are also functions whose transformation matrices are completely populated, i.e. which do not contain a single zero entry. For instance, this is the case for *Quantum Fourier Transforms* (QFT) which occur as part of Shor's factorization algorithm [Sho94, VSB+01]. In fact, these illustrate nicely how skipped variables and a high rate of shared vertices can reduce the QMDD size. More precisely, the corresponding QMDD representations do not show any shared vertices in standard variable order as there are no structurally equivalent sub-matrices. Hence, they have the maximum QMDD size with respect to the matrix size, i.e. $\frac{4^n-1}{3}$ non-terminal vertices in the binary case $r = 2$. However, when applying the inverse variable order as shown in Fig. 4.6, the QMDD size is reduced significantly, e.g. from 21 to 8 non-terminal vertices for $n = 3$.

The existence of skipped variables, excluding the special case of 0-edges, seems to be a rare phenomenon for unitary matrices representing arbitrary quantum operations. However, it is possible to construct such matrices with edges that skip an arbitrary number of variable levels [FT11]. Therefore, skipped variables have to be taken into account carefully in the design of algorithms for QMDDs.

Fig. 4.6 QFT for $n = 3$ qubits and inverse variable orders

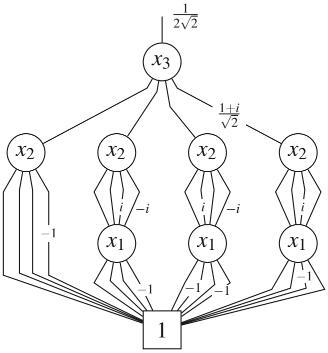
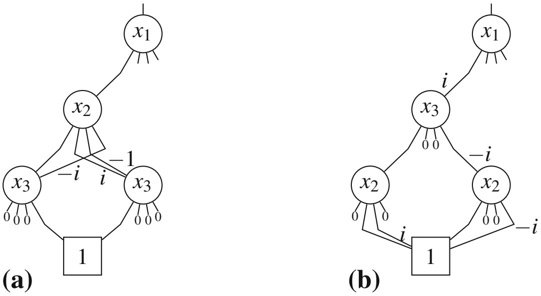


Fig. 4.7 Variable interchange in a QMDD



4.5.2 Local Modifications and Vertex Weights

We have already seen that large effort is put on normalization (of edge weights) in order to ensure canonical representations. Local modifications within a QMDD, e.g. due to a variable interchange, may lead to changes of edge weights which destroy normalization and, hence, require a rework of a large part of the QMDD in order to restore the normalization.

Example 4.10 Consider the QMDD shown in Fig. 4.7a which was built using vertex normalization. Assume that, as part of a re-ordering process, we interchange variables x_2 and x_3. This leads to a QMDD structure as shown in Fig. 4.7b, i.e. the weight of the leftmost edge of the x_1-vertex changes from 1 to i. Thus, this vertex is not normalized anymore according to Definition 4.3. In the worst case, changes like this propagate through the entire QMDD structure. As a result, variable interchanges (and local modifications in general) are no longer local operations which can have a significant effect on the overall efficiency.

The basic idea to overcome this problem is to store weight changes (as they result from local modifications) within the vertices as *vertex weights* instead of propagating them to incoming edges. The advantage of this approach is that we easily maintain a normalized structure. More precisely, vertex weights can be interpreted as the normalization factors of the respective (sub-)matrices. So far, these were aimed to be equal to 1, i.e. all vertices were supposed to represent a normalized matrix with a normalization factor of 1. In this case, vertex weights do not have any effect. Otherwise, they can be removed by simply applying them to the weights of all outgoing edges and then performing vertex normalization. However, we are not actually performing these multiplications. This is because they would only affect the weights of certain edges, but the whole QMDD would still be normalized—w.r.t. a slightly different normalization scheme, for which the normalization factor of that particular sub-matrix is adjusted. In short, the change or introduction of vertex weights is just a small change to the applied normalization scheme, but maintains the normalized structure. Consequently, the use of vertex weights preserves the (optimal) structure sharing according to Theorem 4.2 and, hence, enables local operations to be performed efficiently. This is outlined in detail in the following section for the purpose of variable interchange.

4.5.3 Variable Interchange Scheme for QMDDs

As discussed above, achieving normalization can be a severe obstacle when performing modifications on QMDDs such as adjacent variable interchanges. However, using the concept of vertex weights, this problem is solved, i.e. a local modification such as a variable interchange can be performed without ramifications to other parts of the QMDD structure. The particular way of employing vertex weights is demonstrated in this section.

We use an interchange scheme which is similarly applied in other decision diagram types, e.g. BDDs:

Example 4.11 Consider a BDD where two adjacent variables x_1 and x_2 shall be interchanged. Then, each x_1-vertex is replaced by an x_2-vertex which shall represent the *same* Boolean function in order to make the swap a local operation. This is done by interchanging the labels of the vertices and permuting the sub-trees representing the respective co-factors [Bry86] as sketched in Fig. 4.8.

Analogously, for QMDDs each x_1-vertex is replaced by an x_2-vertex which shall represent the same functionality. By doing so, an interchange of variables x_1 and x_2 for a given matrix leads to a permutation of sub-matrices as illustrated in Fig. 4.9a, i.e. the swapping of certain rows and columns. This accordingly needs to be conducted

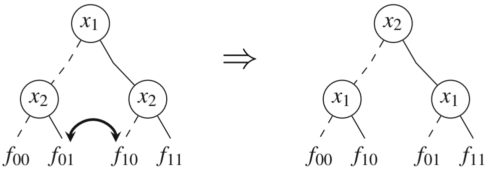
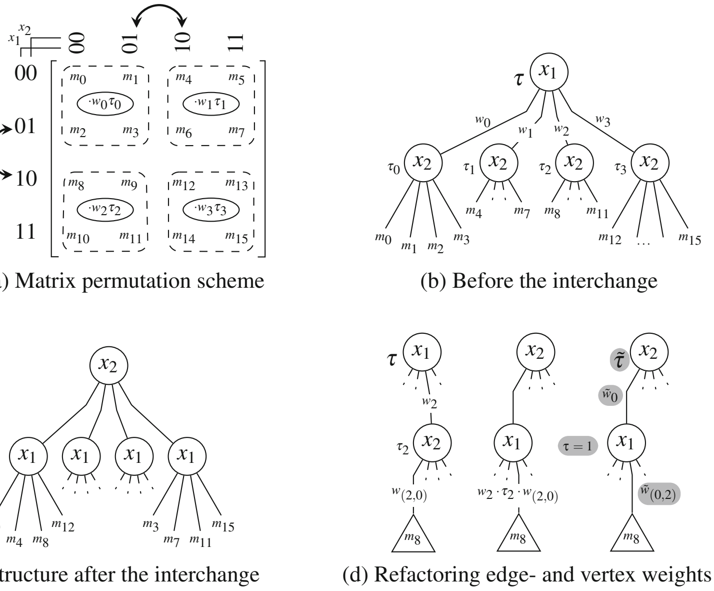


Fig. 4.8 Variable interchange in a BDD



(a) Matrix permutation scheme

(b) Before the interchange

(c) Structure after the interchange

(d) Refactoring edge- and vertex weights

Fig. 4.9 Sketch of the variable interchange procedure for QMDDs ($r = 2$)

in the QMDD structure in which each of the affected sub-matrices is represented by a vertex as well as weighted edges.

That is, to interchange two adjacent variables x_1 and x_2 in a QMDD (where variable x_1 precedes x_2 in the variable order), we process all vertices that are labelled by x_1. We skip all such vertices that do not point to any x_2-vertex. For each of the remaining x_1-vertices v with outgoing edges $e_i^v (i = 0, \ldots, r^2 - 1)$, from which at least one edge points to an x_2-vertex, we perform the following three steps:

1. Create an $r^2 \times r^2$ square matrix $\mathbf{T} = (t_{ij})$ and set t_{ij} to be the jth outgoing edge of the x_2-vertex pointed to by e_i^v and multiply the weight of t_{ij} with the weight of e_i^v and the (vertex) weight of the x_2-vertex. If the destination of e_i^v is not labelled with x_2, set $t_{ij} = e_i^V$ instead.
2. From each column j of $\mathbf{T}$ create a vertex labelled x_1 with outgoing edges $e_i = t_{ij}$ and let e_j^V point to this vertex. Relabel v to x_2.
3. Apply the normalization scheme and store the normalization factor of v by multiplying it to the current vertex weight τ_v.

This procedure is illustrated by the following

Example 4.12 Consider the case of a binary QMDD ($r = 2$) in which two adjacent variables x_1 and x_2 are to be interchanged. At the matrix level, this corresponds to a permutation of rows and columns as illustrated in Fig. 4.9a. According to Step 1, a matrix containing all sub-trees representing the sub-matrices m_0 until m_{15} is created first (see Fig. 4.9b). Then, these sub-trees are re-arranged in Step 2 eventually leading to the structure shown in Fig. 4.9c. Finally, the respective vertices are normalized in Step 3. This is illustrated in Fig. 4.9d for the sub-tree m_8. First, this sub-tree is relocated (according to the previous steps). Then, the product of the corresponding edge and vertex weights is concentrated at the bottom level. The final factorization of this product (highlighted in grey) is achieved by applying vertex normalization to the new structure.

The interchange procedure operates in the same fashion on each sub-matrix of the particular partitioning level that corresponds to the interchanged variables. Thus, it preserves structural equivalence. This guarantees that (an optimal) vertex sharing is maintained and we will not create vertices that only differ by their vertex weight. By using vertex weights, a normalized structure can be achieved without the need to correct ramifications in possibly large parts of the QMDD. The potentially expensive transformation to a QMDD with trivial vertex weights (canonical representation) has to be performed at most once, after we have arrived at the final variable order. However, most effective vertex weights ($\neq 1$) can be expected to vanish through further variable interchanges.

Overall, this enables us to perform variable interchanges efficiently as local operations and to use these interchanges as building blocks for variable re-ordering techniques. The effectiveness of using variable re-ordering for reducing the QMDD size is evaluated in the following.

4.6 Efficiency of QMDDs

In order to demonstrate the overall efficiency of QMDDs in terms of (a) the compactness that is achieved by QMDD representations as well as (b) the efficiency of constructing and manipulating these representations, we performed an experimental evaluation. To this end, we built QMDD representations for a set of important quantum algorithms and, afterwards, performed two complementary approaches for optimizing the QMDDs by applying different variable orders, one heuristic and one exact approach. The results are summarized in Table 4.1. Here, the respective QMDD sizes (i.e. the number of non-terminal vertices; denoted by *Size*) are presented for a selection of benchmark functions. As benchmarks we applied circuits realizing Grover algorithms (*Grover-N*), error correction functionality (*k-qubit-code*, taken from [Mer07]), and Quantum Fourier Transforms (*QFT-N*) where N denotes the number of qubits. We distinguish between (1) the original approach which uses the variable order given from the initial description of the quantum functionality, (2) an improved approach which applies changes to the variable order following a sifting scheme [Rud93], and (3) an exact method which establishes the *optimal* variable order with respect to the size of the resulting QMDD. In addition to the absolute size values, we also provide the percentaged improvements w.r.t. the number of vertices (denoted by *Improvement*) as well as the run-time (in CPU seconds) required to

Table 4.1 Evaluation

Benchmark		Original		Sifting			Exact			
					Improvement			Improvement		
Name	#Qubits	Size	Time	Size	over Original	Time	Size	over Original	over Sifting	Time
Grover-7	7	187	0.01	36	−81%	<0.01	35	−81%	−3%	0.37
Grover-9	9	722	0.02	52	−93%	0.01	51	−93%	−2%	29.14
Grover-11	11	2817	0.15	67	−98%	0.02	66	−97%	−1%	3709
5-qubit-code	9	90	0.01	57	−37%	0.01	43	−52%	−25%	24.73
7-qubit-code	7	44	<0.01	26	−41%	<0.01	26	−41%	–	0.35
9-qubit-code	9	40	<0.01	22	−45%	0.01	22	−45%	–	24.47
9-qubit-code	17	1172	0.01	60	−95%	0.04	(84)[a]	(−93%)	(+40%)	>7200
QFT-3	3	22	<0.01	9	−59%	<0.01	9	−59%	–	<0.01
QFT-4	4	86	<0.01	24	−72%	<0.01	24	−72%	–	0.01
QFT-5	5	342	<0.01	40	−88%	<0.01	40	−88%	–	0.01
QFT-6	6	1366	<0.01	103	−92%	<0.01	103	−92%	–	0.1
QFT-7	7	5462	0.02	167	−97%	0.02	167	−97%	–	1.2

[a] The exact approach did not terminate within the limit of 7200 CPU seconds. At that time, the best result achieved so far was a QMDD size of 84 vertices

generate the QMDDs (denoted by *Time*). All experiments have been conducted on a 2.8 GHz Intel Core i7 machine with 8 GB of main memory running Linux.

It can be seen that the initial representations which were built from the original description are already rather compact—given the fact that the corresponding transformation matrices are of dimension $2^N \times 2^N$. Moreover, these representations can be established in negligible run-time. Much better results in terms of QMDD sizes, however, can be achieved when changes in the variable order are allowed. These changes are enabled by the scheme for interchanging adjacent variables presented in Sect. 4.5.3 and were not possible with the initial proposal of QMDDs (as introduced in [MT06, MT08]). By employing this interchange scheme, reductions of up to two orders of magnitude can be observed. A comparison of the sifting and the exact approach shows that near-to-optimal results can already be achieved in almost no run-time by the heuristic approach. Overall, these evaluations show that QMDDs—which rely on a decomposition scheme that models quantum systems more naturally than the (Shannon) decomposition used in QuIDDs or XQDDs [AP06]—offer a very compact representation of quantum functionality. The proposed concepts allow for an efficient construction and manipulation of practically relevant quantum circuits. Due to fundamental improvements, different variable orders can efficiently be applied which has a significant impact on the resulting QMDD size and, thus, on the overall efficiency of the representation.

In order to allow for a reproduction of the above results in particular and to promote the use of QMDDs in academia and beyond in general, the implementation of QMDDs used for the above evaluation—based on the original QMDD package by Miller et al. presented in [MTG06]—has been made available to the public at http://www.informatik.uni-bremen.de/agra/eng/qmdd.php.

Chapter 5
Discussion and Outlook

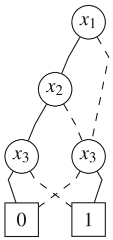
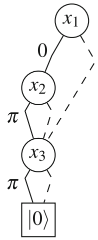
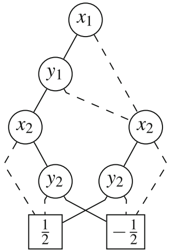
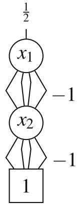
In quantum logic, one considers unitary mappings of quantum systems (so-called Hamiltonians). These can be identified with complex-valued transformation matrices describing the evolution of the system in terms of its r^n basis states. In this part of the book, we discussed that—due to the exponential growth of those matrices—there is an evident need for dedicated, compact representations that exploit structural similarities within these matrices. As argued in Sect. 3.1, techniques for the compact representation of conventional (Boolean) logic, e.g. BDDs, are not applicable to quantum logic at all. Nonetheless, as reviewed in Sect. 3.2, several initial proposals for the representation of quantum functionality are based on BDDs, such as QDDs, XQDDs, or QuIDDs. However, as summarized in Table 5.1, these initial approaches have substantial shortcomings:

- QDDs, a very direct adaption of BDDs for the quantum domain, are only applicable for a small sub-class of quantum functionality—essentially they are limited to (reversible) Boolean functions. While Boolean components constitute important parts of many quantum algorithms, the whole range of quantum-mechanical effects like superposition, entanglement, or phase shifts can not be covered with decision diagrams like BDDs or QDDs.
- QuIDDs and XQDDs are in principle able to represent arbitrary quantum functionality, but they rely on a decomposition scheme which is not natural for quantum logic and are essentially BDDs with multiple terminals. While they benefit from existing BDD algorithms for efficient function manipulation and optimization, they are not able to exploit quantum-mechanical properties like phase shifts and do not natively support multiple-valued quantum logic.

A complementary approach is needed to take into account the special characteristics of quantum logic and overcome the shortcomings of the previous approaches. To this end, we presented *Quantum Multiple-Valued Decision Diagrams* (QMDDs, [NWM+16]). QMDDs employ a decomposition scheme that more naturally models quantum systems and explicitly supports quantum-mechanical effects like phase shifts.

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_5

Table 5.1 Overview on decision diagrams for (Quantum) logic

	BDD	QDD	QuIDD/XQDD	QMDD
Represents:	Boolean function $f : \mathbb{B}^n \to \mathbb{B}$	Function $f : \mathbb{B}^n \to \{R_x(\theta) \cdot \vert 0\rangle\}$	(Unitary) matrix $\mathbf{M} \in \mathbb{C}^{2^n \times 2^n}$	(Unitary) matrix $\mathbf{M} \in \mathbb{C}^{r^n \times r^n}$
		 x_1 0 x_2 π x_3 π $\vert 0\rangle$	 x_1 y_1 x_2 x_2 y_2 y_2 $\frac{1}{2}$ $-\frac{1}{2}$	
Variables	n (input)	n (input)	$2n$ (input/output)	n (qudits)
Decomposition	$f_v = x_i \cdot f_{x_i=1} + \overline{x_i} \cdot f_{x_i=0}$ (Shannon, cf. Eq. 3.2)	$f = x_i \cdot R_x(\theta) \cdot f_{high(v)} + \overline{x_i} \cdot f_{low(v)}$ (Bi-decomp., cf. Eq. 3.6)	$f_v = x_i \cdot f_{high(v)} + \overline{x_i} \cdot f_{low(v)}$ (Shannon with arithmetic interpretation, cf. Eq. 3.1)	$\mathbf{M} = \mathbf{W}(\mathbf{M}) * \widehat{\mathbf{M}}$ (Khatri-Rao, cf. Eq. 4.8)
#Terminals	1 or 2 (Boolean 0 and/or 1)	1 or 2 (qubit states $\vert 0\rangle$ and/or $\vert 1\rangle$)	Multiple (one for each different value in **M**)	Exactly 1 (matrix $[1]_{1\times 1}$)
Edge weights	✗	✓ (rotation angles, high edges only)	✗	✓ (normalization factors)
Supports:				
Boolean functions	✓	✓	✓	✗
Reversible Boolean func.	✓	✓	✓	✓
Unitary matrices	✗	✗	✓	✓
Matrix manipulation	✗	✗	✓ (based on BDD algorithms)	✓ (dedicated algorithms)

(continued)

Table 5.1 (continued)

	BDD	QDD	QuIDD/XQDD	QMDD
Represents:	Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}$	Function $f : \mathbb{B}^n \rightarrow \{R_x(\theta) \cdot \lvert 0\rangle\}$	(Unitary) matrix $\mathbf{M} \in \mathbb{C}^{2^n \times 2^n}$	(Unitary) matrix $\mathbf{M} \in \mathbb{C}^{r^n \times r^n}$
Supports: (continued)				
Exploitation of quantum-mechanical phenomena	✗	✗	✗	✓
Multiple-valued logic	✗	✗	✗ (extension proposed)	✓ (natively supported)
Summary:	Origins in conventional logic, not applicable to quantum logic at all	Very restricted applicability to quantum logic (small subset only)	BDD-based approach, applicable to general quantum logic, but use of (Shannon) decomposition scheme which is not natural for quantum logic	Complementary approach with more natural decomposition scheme and dedicated algorithms, native support for qudits

This already becomes apparent when considering the representation of generalized Hadamard operations $H^{\otimes n}$ which are commonly used in quantum logic. The QuIDD and QMDD representations for the case $n = 2$, i.e. the 4×4 matrix $\mathbf{H} \otimes \mathbf{H} = \frac{1}{\sqrt{2}}\begin{pmatrix} \mathbf{H} & \mathbf{H} \\ \mathbf{H} & -\mathbf{H} \end{pmatrix}$, are shown in Table 5.1. While QuIDDs can not take advantage from the fact that the lower right block is a phase-shifted copy of the other blocks, this property allows for a QMDD representation with a single vertex at each level. This effect can in the worst case lead to situations where QMDD representations are linear in size while the corresponding QuIDD/XQDD is always exponential in size (for arbitrary variable orders).

Example 5.1 An example for this case is given by combined R_z rotations on multiple qubits such that the elementary rotation $R_z(\pi/2^k)$ is applied to the kth qubit. As in the case of generalized Hadamard operations, the corresponding QMDD representations require a single vertex per qubit and are, thus, linear in size. In contrast, any QuIDD/XQDD representation requires an exponential number of terminal vertices, since it can be shown that all 2^n powers of $e^{\pi/2^n}$ occur in the corresponding matrix. Consequently, the QuIDD/XQDD is exponential in size regardless which variable order is applied.

On the contrary, under no circumstances can QMDD representations be significantly larger than the corresponding QuIDD/XQDD representations. In fact, due to the interleavedness of x and y variables in QuIDDs/XQDDs it is always possible to construct a QMDD with a corresponding variable order, such that each x_k-vertex of the QuIDD/XQDD (together with its y_k-children) can be represented by a single QMDD x_k-vertex. In addition, QMDDs have another notable advantage: matrices that are equivalent up to global phase are represented by QMDDs that only differ in the root edge weight, while the corresponding QuIDDs/XQDDs may exhibit completely disjoint sets of terminal vertices. This property can be become very advantageous when it comes to the verification of quantum logic designs (cf. Chap. 8).

Because of that and as also demonstrated by an experimental evaluation showing their overall efficiency and practical usefulness (cf. Sect. 4.6), QMDDs are a very promising representation to be used for the design of quantum logic. In fact, unitary transformation matrices that describe quantum operations are vital for the design and analysis of quantum systems, e.g. for synthesis, simulation, or verification of quantum circuits. The availability of an efficient representation of these matrices is essential, as the absence of such representations significantly limits the development and applicability of automated methods for the design of quantum logic. With large-scale quantum computers coming to the range of vision, such solutions become more and more essential for quantum logic design in order to keep up with the technological progress.

With QDDs, QuIDDs, and XQDDs, as well as the initial QMDD proposal, several approaches for employing decision diagrams as an efficient representation of quantum functionality have already been around for a while. In fact, they have shown their benefits in various applications such as synthesis (see e.g. [AP06, SWH+12]), simulation (see e.g. [VRMH03, GTFM07, VMH09]), and verification (see e.g. [VMH07, WLTK08, WGMD09]) of quantum circuits. In these scenarios, they have proven to be very effective for speeding up matrix-based approaches that operate on transformation matrices or exploit properties of those.

However, significant shortcomings have limited their applicability for a long time. As a consequence, the full potential of QMDDs or decision diagrams for quantum functionality in general has hardly been exploited yet and they have not found adoption in state-of-the-art approaches for the design of arbitrary quantum functionality so far. We believe that the revised version of QMDDs presented in this book provides the basis for a more sophisticated application of decision diagrams in the domain of quantum computation including solutions for synthesis, simulation and verification. To this end, several methods aiming at this goal and relying on efficient representations of the desired quantum functionality, especially in terms of QMDDs, are presented in the following.

Part III
Design of Quantum Logic

Chapter 6
Challenges and Initial Approaches

Motivated by its superiority compared to conventional solutions for many computational problems, quantum computation has been intensely investigated from a theoretical perspective in the past decades. Rapid advancements have been made in the area of quantum algorithms. In fact, algorithms exploiting quantum-mechanical effects and promising significant improvements over any known classical solution have been developed for many practically relevant problems such as factorization (and its application in cryptography), database search, graph/algebraic problems, and many more [Jor16]. Due to these prospects, the—very challenging—physical realization of quantum functionality, i.e. the development of physical devices which can be employed to actually conduct quantum computation, has also received significant attention in recent years. In fact, despite severe physical challenges, a variety of technologies has been found to be promising for this purpose and the implementation of important quantum operations has been successfully demonstrated.

The prospects of actually working quantum computers give rise to investigations of the design of *quantum circuits*, i.e. the construction of circuits realizing a given quantum functionality in terms of a series of elementary quantum operations that are executed sequentially or in parallel. In fact, this domain has become a prospering field of study and several approaches for the realization of arbitrary quantum functionality have been proposed. However, most of these methods tackle the problem from a rather theoretical point of view, e.g. make very simplified assumptions about the set of available quantum operations, and often provide theoretical upper bounds rather than algorithms that would scale to quantum systems consisting of dozens of usable qubits. As the latter are expected to be available sooner or later, there is a need for developing methods for *Computer-Aided Design* (CAD) of quantum circuits, i.e. methods that (1) *automatically* generate a circuit description of the desired quantum functionality, (2) take into account the physical constraints of the target technology, and (3) scale to quantum systems of considerable size.

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_6

In this part, we present several approaches that contribute to achieving this objective especially by exploiting efficient representations of quantum functionality as they were discussed above.

More precisely, in the present chapter we first review the major design challenges to be addressed and which technologies have been proposed for the actual physical realization of quantum functionality in Sect. 6.1. In Sect. 6.2, we then review the state-of-the-art in the design of quantum logic. Based on these discussions, we show how compact representations of quantum functionality (as reviewed in the previous part) can be employed to develop more scalable solutions for important design tasks, namely synthesis and verification, in Chaps. 7 and 8, respectively. Last, but not least, the book is concluded in Chap. 9 with a discussion of the significance of compact representations and adequate design algorithms as an important step towards CAD for quantum logic.

6.1 Design Challenges

The implementation of quantum functionality on a physical machine, i.e. building a quantum computer, is a very challenging task. It requires a physical system that allows for well-formed qubit states as well as their controlled transformation according to the time-dependent Hamiltonians of quantum operations. Regardless of which particular technology is chosen, various physical limitations have to be taken into account:

- **Quantum Decoherence:** The *computation time*, i.e. the time available between the initial preparation of a quantum system which brings all qubits into a well-defined state and the final read-out (measurement), is rather short due to the fact that quantum systems in an "excited state" may spontaneously decay to a ground/basis state such that the computation result is lost and destroyed irretrievably.
- **Environmental factors**: As quantum systems are operated at an atomic scale, they are prone to environmental perturbation caused e.g. by radiation—much more than classical computation technologies like CMOS or alike. Large effort has to be made to cope with these effects and enable computations to be performed in a *fault-tolerant* fashion. As a consequence, individual logical particles (qubits) have to be realized by an ensemble of multiple physical particles and error-correcting codes have to be applied which require additional computations [NC00, p. 425].
- **Topological/functional restrictions:** The capability of modifying individual particles is restricted to applying a few physical operations (e.g. a laser beam or alike) with a limited precision. In addition, even if an operation that involves multiple particles is realizable in principle, in most technologies considered so far the particles have to be adjacent in order to actually conduct the operation (this induces so-called *nearest neighbour* constraints). Consequently, additional efforts are necessary to create adjacency of the respective particles, e.g. by pairwise swapping of particles.

Despite these challenges, several technologies that support quantum computation in principle have been successfully explored in the past. A *quantum technology* describes a physical system which allows for the realization of qubits together with a set of supported quantum operations for realizing Hamiltonians and is summarized in form of a *Physical Machine Description* (PMD) [LCJ13]. A broad survey of these quantum technologies has been conducted in the ARDA quantum computing roadmap [ARD]. Each PMD is different in terms of its quantum-mechanical properties. This leads to different Hamiltonians and, hence, a different set of supported operations (often also denoted as *primitive* or *elementary* operations). In the following, we provide a brief review of six popular quantum technologies, namely:

- *Quantum Dots (QD)*
  In this system, a qubit is defined by the spin state of a single-electron quantum dot, which is confined by electrostatic potential. The desired quantum operations are implemented by gating of the tunneling barrier between neighbouring dots [TPJ+07].
- *Superconducting Qubits (SC)*
  In a superconducting system, a qubit is simply represented by the two rotation directions of the persistent super-current of Cooper pairs in a superconducting ring containing Josephson tunnel junctions [SJD+03]. The state of a qubit is defined by a distribution of voltages or currents, each characterized by an amplitude and phase, which are functions of time.
- *Ion Traps (IT)*
  Ion-trap quantum computation can be implemented by confining a string of ions in a single trap, exploiting their electronic states as qubit logic levels, and using mutual Coulomb interaction for transferring quantum information between ions [CZ95].
- *Neutral Atoms (NA)*
  A system of trapped neutral atoms is a good candidate for implementing scalable quantum computing [DBJ00, BCJ+00]. That the atoms are neutral means that they are feebly coupled to the environment. Hence, decoherence is minimized. Trapped atoms can be cooled to the motional ground state of the quantized potential wells, and the initialization of the internal atomic states can be performed using standard techniques of laser spectroscopy. The different qubit levels can be described by various motional and internal states of the neutral atoms.
- *Linear Photonics (LP)*
  In linear photonics, the qubits are represented by the quantum state of single photons. Quantum logic gates can be constructed using only linear optical elements, such as mirrors and beamsplitters, additional resource photons, and triggering signals from a single-photon detector [KLM01].
- *Non-Linear Photonics (NP)*
  In non-linear photonics, quantum logic gates are implemented using interactions of photons with non-linear photonic crystals. The photonic crystals include layers of a Kerr medium [IA05] and, thus, perform a non-linear shift of the photonic wave function.

Table 6.1 Primitive quantum operations supported by different PMDs

PMD	One-qubit operations	Two-qubit operations
QD	R_x, R_z, σ_x, σ_z, S, T	Controlled Z
SC	R_x, R_y, R_z	$iSWAP$, Controlled Z
IT	R_{xy}, R_z	G
NA	R_{xy}	Controlled Z
LP	R_x, R_y, R_z, σ_x, σ_y, σ_z, S, T, H	$CNOT$, Controlled Z, $SWAP$, $ZENO$
NP	A_{squ}, R_x, R_y, R_z, H	$CNOT$

Each of the quantum systems described above relies on a different quantum-mechanical property and, hence, a different set of supported (primitive) quantum operations. These are listed in Table 6.1 [LCJ13]. While many of these operations have already been considered in Sect. 2.2, it is not necessary for the purpose of this book to understand all operations in detail. However, it is important to note that the set of primitive operations which indeed have a direct physical counterpart, e.g. a laser beam, largely depends on the considered technology. Moreover, this set is in general rather small and only contains one- and two-qubit gates. This poses severe obstacles to the design of quantum circuits that realize a given quantum functionality by means of a series of primitive quantum operations that are available on a physical machine.

Probably, the most severe implication is that exact quantum computation, i.e. the precise implementation of a Hamiltonian, is not possible in general and the desired quantum functionality rather needs to be approximated using the small set of physical operations that is available. To this end, the approximation itself is not the crucial point. The state of quantum systems can not be precisely read-out anyway and, thus, approximations up to some small amount of error are usually acceptable and taken into account for the robust design of quantum algorithms. Much more severe is the fact that the desired quantum functionality is initially given in terms of e.g. a quantum algorithm, a transformation matrix describing the unitary operator of the Hamiltonian or a quantum circuit consisting of high-level quantum gates or modules that can not be mapped to some atomic physical operation. Nonetheless, this functionality has to be realized in terms of a cascade of primitive quantum operations that do have a direct physical counterpart. This set of primitive operations differs significantly for different quantum technologies and—in each case—only contains a small set of one- and two-qubit operations.

Moreover, in order to incorporate the requirements of physical implementations regarding fault-tolerance, the use of dedicated gate libraries was suggested. More precisely, each (non fault-tolerant) gate is approximated by a cascade of gates that can be implemented in a fault-tolerant fashion. This process is called quantum *compilation*. Initially, the universal (and discrete) *Clifford + T library* consisting of CNOT, H, and T operations (where $T = R_z(\pi/4)$) has been proposed for this purpose as fault-tolerant constructions of these gates could be provided—at least at a theoretical level [BMP+00]. Indeed, the synthesis of arbitrary one-qubit operations in this library has become rather well understood [KMM12, AMMR13].

More recently, increasing knowledge about promising technologies for quantum computation led to another approach named *Fault-Tolerant Quantum Logic Synthesis* (FTQLS, [LCJ14a]) that performs the compilation in a technology-aware fashion, i.e. taking into account information about primitive gates as well as fault-tolerant implementations. For instance, if $R_x(\theta)$ rotations are supported in principle in some technology, they are only employed for a small set of rotation angles, i.e. $\theta = k \cdot \frac{\pi}{4}$ (for $k = 0, 1, \ldots, 7$). Moreover, multiple optimization steps are conducted using an elaborated ruleset [LCJ14b].

Overall, while several approaches exist that aim at the actual physical realization of a quantum circuit, in most cases a corresponding circuit for the desired functionality is not given initially, but rather needs to be generated from the original description first. This design task is called *synthesis* and is considered in more detail in the following.

6.2 Initial Synthesis Approaches

It was recognized very early that arbitrary (multi-qubit) operations would be hard to implement directly due to the above mentioned physical constraints and due to the fact that, as in conventional logic, the quantum functionality needs to be decomposed into a cascade of primitive/elementary operations. In contrast to conventional logic, however, where there is only one unary (NOT) and several binary operations (AND, OR, etc.), the focus here was—initially—put on a realization in terms of a rather large set of one-qubit operations together with a rather small set of two-qubit operations. This working hypothesis was supported by theoretical investigations on so-called *universal* quantum gate libraries. In this context, the attribute universal means that an arbitrary quantum operation can be approximated to an arbitrary precision using only operations (gates) from the library.[1] In fact, it was shown that the precise shape of the used gate library can be considered secondary, as *any* universal library would allow for efficient approximations. More precisely, the well-known *Solovay-Kitaev theorem* [DN06] states that a precision up to an error of ε can be achieved with $O(\log^c(1/\varepsilon))$ gates for unitary one-qubit operations. Here the constant $c < 4$ depends on which particular library is used.

As a consequence, a lot of research has focused on the decomposition of (arbitrary) quantum functionality and synthesis of corresponding quantum circuits. Traditionally, the most popular gate library used for this purpose consists of arbitrary one-qubit operations together with a single, distinguished two-qubit operation, namely the controlled NOT (CNOT) which was considered most promising for a physical realization. This library, however, hardly takes into account the requirements of physical

[1] From a mathematical perspective, a universal quantum gate library is a dense subset of the set of all quantum operations.

realizations, namely the need for fault-tolerant implementations and a realization in terms of primitive operations supported by the respective technology. Consequently, in order to comply with these requirements the research focus has recently been put on the *compilation* of single qubit operations by means of fault-tolerant and/or technology-specific gate libraries (as reviewed above).

As we will see, the proposed approaches for synthesizing arbitrary quantum functionality apply matrix decompositions that require a very high computational effort. In fact, their complexity is only polynomial in terms of the matrix' size, but the latter grows exponentially with respect to the size of the considered quantum system. As a consequence, dedicated approaches have been developed for the synthesis of Boolean components. These constitute important parts of many quantum algorithms and applying a specialized scheme can significantly reduce the synthesis complexity. This scheme is reviewed in Sect. 6.2.1. After this, we consider in detail the state-of-the-art approaches for synthesizing arbitrary quantum functionality in Sect. 6.2.2.

6.2.1 Synthesis of Boolean Components

Many quantum algorithms contain a considerable Boolean component, e.g. the oracle in Grover's database search algorithm [Gro96] or the modular exponentiation used in Shor's factorization algorithm [Sho94]. Applying synthesis approaches dedicated to realizing Boolean components has shown to be promising in order to significantly reduce the complexity compared to generic approaches. But still, the synthesis of Boolean components is a complex task due to the following major challenges:

- **Non-reversibility**: The underlying Boolean functions are often not reversible and, thus, can not be implemented directly as a quantum operation since the latter are inherently reversible.
- **Adequate gate library**: In addition, the logic operations commonly used to describe and realize Boolean functionality in conventional logic are not available in quantum logic (again due to their non-reversibility). In fact, the functionality has to be realized in terms of quantum gates which can finally be executed on the targeted quantum device.

As a consequence, most state-of-the-art methods follow a synthesis flow that *does not directly* realize the given quantum functionality in one step. Instead, they employ a *multiple-step* approach that addresses the two major challenges, i.e. non-reversibility of the original functionality and the use of a dedicated quantum gate library, separately. More precisely, as shown in Fig. 6.1, the desired quantum functionality of Boolean components is

- first realized as a *reversible circuit* (cf. Sect. 2.3) which provides a reversible description of the original functionality. To this end, two complementary approaches have been introduced in the past.

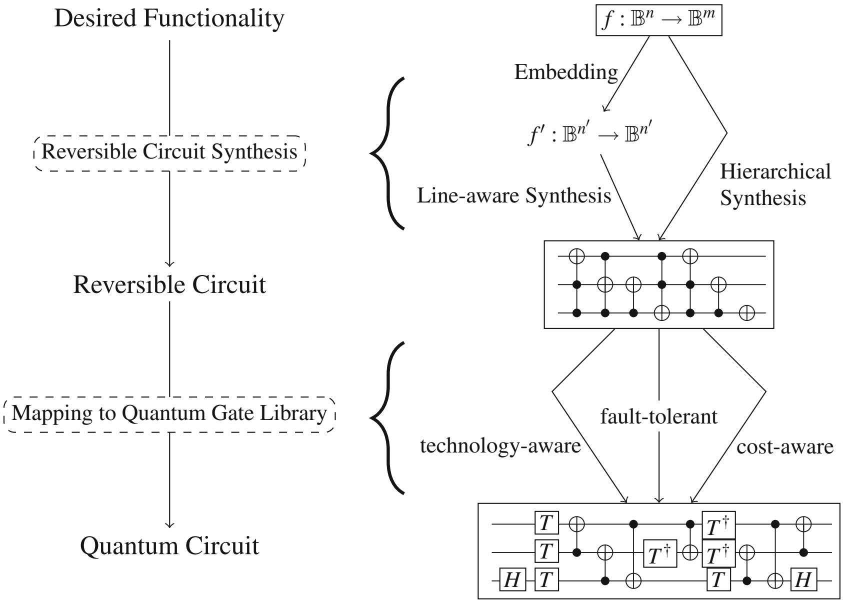


Fig. 6.1 Synthesis flow for Boolean components

- One set of *line-aware* solutions requires a fully reversible function as input (see e.g. [SPMH03, MMD03, GAJ06, GWDD09, SZSS10] or [SWH+12]). In order to enable support for non-reversible functionality, a pre-synthesis process called *embedding* is conducted before (see e.g. [WKD11, SWK+15]).
- As an alternative, *hierarchical* solutions e.g. based on decision diagrams or two-level representations have been proposed in [WD09, LJ14] or [FTR07], respectively. Here, large functionality—regardless of its (non-)reversibility—is decomposed into smaller sub-functions for which corresponding sub-circuits can be derived.

However, the circuits resulting from any of these approaches consist of *multiple-control Toffoli (MCT) gates* which are not directly realizable on any so far known quantum device.

- Consequently, to comply with the gate library restrictions of the targeted technology, the reversible circuit is afterwards *mapped* to an equivalent cascade of primitive quantum operations (see e.g. [BBC+95, MYDM05, MWS11, WSOD13]) which can be executed directly on the quantum device. This mapping of the MCT gates to primitive quantum operations can be conducted with respect to various design paradigms such as cost-aware, fault-tolerant, and technology-aware.

The numerous, promising opportunities to exploit compact representation of the corresponding functionality for synthesis are discussed in detail in Sect. 7.1.

6.2.2 Synthesis of Arbitrary Quantum Functionality

As stated above, the most prominent universal gate library used for the synthesis of arbitrary quantum functionality is constituted by the CNOT gate together with the set of arbitrary one qubit gates (denoted by *one-qubit + CNOT* in the following). The success of this library is actually not due to the fact that it was (one of) the first universal gate libraries that have ever been proposed. More precisely, already in 1989—6 years before the 1995 seminal paper by Barenco et al. [BBC+95]—David Deutsch proposed a single, parameterized 3-qubit gate which he could prove to be universal [Deu89] and following investigations even showed that in fact almost any multiple-qubit gate is universal [DBE95]. The success of the one-qubit+CNOT library is rather founded on the fact that the Barenco et al. paper [BBC+95] contains explicit constructions for the realization of arbitrary quantum functionality in terms of the library.

This constructiveness already becomes apparent when considering the proof of universality which is conducted in three steps:

1. It is shown that any one-controlled gate is realizable by a cascade of four one-qubit and two CNOT gates (cf. [BBC+95, Corollary 5.3]). More precisely, for any one-qubit gate W there exist four one-qubits gates A, B, C, D with $ABC = I$ such that the equality in Fig. 6.2 holds.
2. It is shown that any two-controlled gate is realizable by a cascade of three one-controlled gates (from the first step) and two CNOT gates (cf. [BBC+95, Lemma 6.1]). More precisely, for any one-qubit gate U there exists a gate W applying a rotation along the same axis as U, but with half of the original rotation angle, such that $W^2 = U$ and the equality in Fig. 6.3 holds.
3. From this, the universality follows from the fact that the above constructions allow for a realization of the 2-controlled $iR_x(\theta)$ gate (the so-called *Deutsch* gate) which is known to be universal [Deu89].

These constructions are generalized to controlled gates with arbitrarily many controls and, finally, to arbitrary quantum functionality—thereby providing

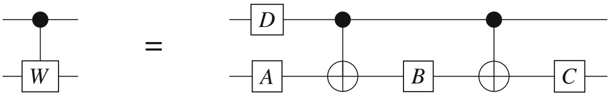


Fig. 6.2 Construction of 1-controlled unitary operators ($ABC = I$)

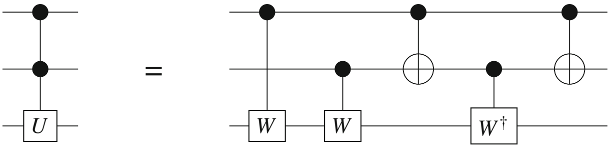


Fig. 6.3 Construction of 2-controlled unitary operators ($W^2 = U$)

theoretical upper bounds for their realization in terms of the one-qubit+CNOT library. These bounds essentially motivated a research competition that aimed at improving the bounds further and further (see e.g. [VMS04, BVMS05, NKS06, SBM06, SAZS11]).

The most notable constructions in this context are given by the *Cosine-Sine-Decomposition* (CSD, [PW94]) as well as the *Quantum Shannon Decomposition* (QSD, [SBM06]). These two approaches constitute state-of-the-art solutions for the synthesis of arbitrary quantum functionality. In the following, they are reviewed in more detail in order to illustrate the complexity of the underlying decompositions which lead to poor scalability as well as the drawbacks of the resulting circuits.

6.2.2.1 Cosine-Sine-Decomposition

Using the CS-Decomposition (CSD), a unitary matrix $\mathbf{U} \in U(2^n)$ of dimension $2^n \times 2^n$ can be decomposed into four unitary matrices $\mathbf{L}_1, \mathbf{L}_2, \mathbf{R}_1, \mathbf{R}_2 \in U(2^{n-1})$ of dimension $2^{n-1} \times 2^{n-1}$ and two real-valued diagonal matrices $\mathbf{C}, \mathbf{S}$ of the same dimension with $\mathbf{C}^2 + \mathbf{S}^2 = \mathbf{I}$ (where $\mathbf{I}$ is the identity matrix) as follows:

$$\mathbf{U} = \begin{pmatrix} \mathbf{L}_1 & 0 \\ 0 & \mathbf{L}_2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{C} & \mathbf{S} \\ -\mathbf{S} & \mathbf{C} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{R}_1 & 0 \\ 0 & \mathbf{R}_2 \end{pmatrix} \tag{6.1}$$

For the purpose of quantum circuit synthesis, the CS-Decomposition has the advantage that the central matrix in Eq. (6.1) can be realized by a special structure of *uniformly controlled* R_y rotations which can be realized in terms of 2^{n-1} CNOT and uncontrolled R_y gates (more details on this construction can be found in [BVMS05]). Moreover, by applying it recursively to $\mathbf{L}_1 \oplus \mathbf{I}, \mathbf{I} \oplus \mathbf{L}_2, \mathbf{R}_1 \oplus \mathbf{I}, \mathbf{I} \oplus \mathbf{R}_2$, the entire matrix can be decomposed. While CSD is a standard decomposition technique that can in principle be computed using computer algebra software like MatLab, the problem with this approach is, however, that the matrices used in the recursion essentially have the same dimension as the initial matrix (see the circuit representation in Fig. 6.4 where a backslash denotes a circuit "wire" carrying an arbitrary number of qubits an d an empty square □ represents a uniform control).

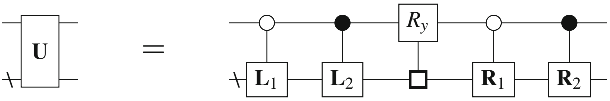


Fig. 6.4 Cosine-Sine-decomposition (adapted from [SBM06])

6.2.2.2 Quantum Shannon Decomposition

A decomposition that overcomes this problem is given by the Quantum Shannon Decomposition (QSD). To this end, note that a matrix $\mathbf{U}_1 \oplus \mathbf{U}_2 \in U(2^n)$ can be decomposed as

$$\begin{pmatrix} \mathbf{U}_1 & 0 \\ 0 & \mathbf{U}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{V} & 0 \\ 0 & \mathbf{V} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D} & 0 \\ 0 & \mathbf{D}^\dagger \end{pmatrix} \cdot \begin{pmatrix} \mathbf{W} & 0 \\ 0 & \mathbf{W} \end{pmatrix} \tag{6.2}$$

for unitary $\mathbf{V}, \mathbf{W} \in U(2^{n-1})$ and a complex-valued diagonal matrix $\mathbf{D}$. The particular matrices can be obtained by diagonalizing $\mathbf{U}_1\mathbf{U}_2^\dagger$. More precisely, by using the subequations $\mathbf{U}_1 = \mathbf{VDW}$ and $\mathbf{U}_2 = \mathbf{VD}^\dagger\mathbf{W}$ that can be computed from Eq. (6.2), we obtain $\mathbf{U}_1\mathbf{U}_2^\dagger = \mathbf{VDWW}^\dagger\mathbf{DV}^\dagger = \mathbf{VD}^2\mathbf{V}^{-1}$. Thus, $\mathbf{D}$ and $\mathbf{V}$ can be computed directly from the diagonalization of $\mathbf{U}_1\mathbf{U}_2^\dagger$, and $\mathbf{W}$ as $\mathbf{W} = \mathbf{DV}^\dagger\mathbf{U}_2$. By applying this to the left- and rightmost matrix in Eq. (6.1), we obtain the QSD. More precisely,

$$\mathbf{U} = \begin{pmatrix} \mathbf{V}_l & 0 \\ 0 & \mathbf{V}_l \end{pmatrix} \begin{pmatrix} \mathbf{D}_l & 0 \\ 0 & \mathbf{D}_l^\dagger \end{pmatrix} \begin{pmatrix} \mathbf{W}_l & 0 \\ 0 & \mathbf{W}_l \end{pmatrix} \cdot \begin{pmatrix} \mathbf{C} & \mathbf{S} \\ -\mathbf{S} & \mathbf{C} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{V}_r & 0 \\ 0 & \mathbf{V}_r \end{pmatrix} \begin{pmatrix} \mathbf{D}_r & 0 \\ 0 & \mathbf{D}_r^\dagger \end{pmatrix} \begin{pmatrix} \mathbf{W}_r & 0 \\ 0 & \mathbf{W}_r \end{pmatrix} \tag{6.3}$$

where the diagonal matrices $\mathbf{D}_l \oplus \mathbf{D}_l^\dagger$ and $\mathbf{D}_r \oplus \mathbf{D}_r^\dagger$ can be implemented using the construction of uniformly controlled R_z rotations as above (see the circuit representation in Fig. 6.5). With this construction, the matrices occurring in the recursive decomposition actually become smaller with each recursive step. This allows for stopping the recursion at some level where the matrices are small enough such that dedicated approaches can be applied.

The overall complexity of computing these decompositions is polynomial in the size of the matrices (diagonalization, matrix multiplication), but as the latter grow exponentially in the number of qubits, the computation becomes infeasible in practice for rather small quantum systems. Moreover, they lead to a significant number of gates and additionally rely on (at least) arbitrary R_y, R_z rotations. However, in general these can not be realized in a fault-tolerant fashion and, beyond that, may not even be supported by the targeted physical device (i.e. they are not contained in the set of primitive operations as listed before in Table 6.1) such that a computationally expensive compilation to an adequate gate library has to be conducted.

Overall, the drawback of these generic synthesis approaches that aim at the synthesis of arbitrary quantum functionality is that they (a) lead to a significant number of gates even for a small number of qubits and (b) use decomposition techniques that do not scale to larger quantum systems. However, many design objectives can

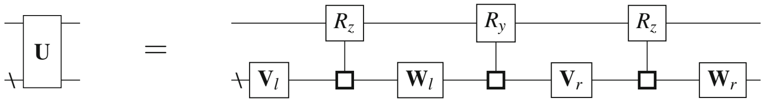


Fig. 6.5 Quantum shannon decomposition (adapted from [SBM06])

be realized without employing the full power of arbitrary quantum operations. To this end, it has shown promise to consider essential sub-problems of quantum logic design separately. One of these is the synthesis of Boolean components which has already been outlined above and is discussed in more detail in Sect. 7.1. In addition to that, in Sect. 7.2 we focus on the synthesis of quantum circuits that implement Clifford Group operations (based on the initial description in [NWD14a]). These circuits are essential for many quantum applications and can be realized in terms of a dedicated, fault-tolerant gate library. The proposed approach exploits specific properties of the associated transformation matrices that have a direct counterpart in the corresponding QMDDs (cf. Chap. 4) which are employed for their efficient representation. Therefore, it strikingly shows the potential of using compact representations for quantum circuit synthesis.

Regardless of which particular approach is taken to generate the quantum circuits or to compile individual gates to the targeted gate library, it is vital to ensure the correctness of the transformations, optimizations and technology mappings that are conducted throughout this process. More precisely, different descriptions of quantum functionality need to be checked for equivalence, i.e. it has to be verified that they indeed describe the same or an equivalent functionality. In Chap. 8, we present a scheme that addresses this need and explicitly relies on efficient representations of the corresponding transformation matrices. As a further benefit, it also covers implementations in multi-level quantum systems (with $r > 2$ basis states).

Chapter 7
Synthesis of Quantum Circuits

In this chapter, we demonstrate the importance of compact representations and their applicability for the synthesis of quantum circuits. To this end, we consider the synthesis problem for two important classes of quantum functionality, namely Boolean components as well as Clifford group operations, and present corresponding algorithms in Sects. 7.1 and 7.2, respectively. While the algorithms are conceptually presented by means of transformation matrices, they natively allow for the use with compact representations (especially QMDDs) and, by actually doing so, they become very scalable state-of-the-art solutions for the respective synthesis task. To conclude the chapter, we provide an outlook on the generalization of the presented techniques to arbitrary quantum functionality in Sect. 7.3.

7.1 Synthesis of Boolean Components

As discussed in the previous chapter, the synthesis of Boolean components is an important sub-task of quantum logic synthesis and is commonly conducted in two steps (cf. Sect. 6.2.1). More precisely, the original functionality is first realized in terms of a *reversible circuit* which is afterwards mapped to an equivalent cascade of quantum gates. The latter of these steps usually employs a gate-wise mapping scheme in combination with optimizations at the circuit level like gate reordering and rewriting and other simplifications [Sas12]. Especially, this step does not rely on an efficient functional representation (e.g. by means of QMDDs). Hence, we focus on the first step, i.e. reversible circuit synthesis, in the following.

There are two complementary approaches to this task: On the one hand, there are *hierarchical* solutions like [FTR07, WD09, LJ14] that decompose large functionality into smaller sub-functions for which corresponding sub-circuits can be derived. While these solutions can be applied regardless of the (non-)reversibility of the objective function, they usually lead to a high number of additional (unnecessary)

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_7

qubits/circuit lines.[1] As these are a very rare resource in actual physical implementations, the corresponding approaches are not considered in more detail here.

On the other hand, there are many design methods (termed *line-aware* solutions) which are able to realize the desired functionality without adding further qubits/circuit lines, e.g. those proposed in [MMD03, GWDD09, SZSS10, SWH+12]. However, the drawback of all these approaches is that they require a fully reversible function as input, while frequently also irreversible functionality is to be realized.

Consequently, in Sect. 7.1.1 we first discuss how irreversible function descriptions have to be pre-processed in order to be applicable to such methods. Then, in Sect. 7.1.2 we describe how corresponding QMDD representations can be constructed in order to use them for reversible circuit synthesis (the description is based on [NZWD17]), before the actual synthesis algorithm that is able to take advantage of these representations is presented in Sect. 7.1.3.

7.1.1 Embedding: Handling Irreversible Function Descriptions

In order to realize irreversible Boolean functions in a reversible fashion (as it is inherently required in quantum logic), a pre-synthesis process called *embedding* needs to be conducted (see e.g. [WKD11, SWK+15]).

To this end, additional outputs (so-called *garbage outputs*) are added to the considered function $f \in \mathscr{B}_{n,m}$. More precisely, $\lceil \log_2(\mu(f)) \rceil$ additional outputs are required, whereby $\mu(f)$ is the maximal number of times an output pattern is generated by f, i.e.

$$\mu(f) = \max_{y \in \mathbb{B}^m}(|\{x \mid y = f(x)\}|).$$

In order to keep the number of inputs and outputs equal, this may also result in the addition of further inputs. That is, an irreversible function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ is embedded into a function $f' : \mathbb{B}^{m+\lceil \log_2(\mu(f)) \rceil} \rightarrow \mathbb{B}^{m+\lceil \log_2(\mu(f)) \rceil}$. While f' is to be specified in a fully reversible fashion, i.e. f' is a bijection, the desired target functionality can be employed by setting the additionally added inputs to a constant value and recognizing only the non-garbage outputs. An example illustrates the idea.

Example 7.1 Consider the Boolean function $f : \mathbb{B}^2 \rightarrow \mathbb{B}^1$ with $f(x_1, x_2) = x_1 \wedge x_2$ to be synthesized as a reversible circuit. Obviously, f is irreversible. The maximal number of times an output pattern is generated by f is $\mu(f) = 3$ (namely 0 for the input patterns 00, 01, and 10). Hence, in order to realize f in quantum logic, the function has to be embedded into a function $f' : \mathbb{B}^{2+1} \rightarrow \mathbb{B}^{1+2}$ with $\lceil \log_2(3) \rceil = 2$ additional outputs and $1 + \lceil \log_2(3) \rceil - 2 = 1$ additional input. The resulting function $f' : \mathbb{B}^3 \rightarrow \mathbb{B}^3$ can be specified as $f'_1(x_1, x_2, x_3) = x_1$, $f'_2(x_1, x_2, x_3) = x_2$,

[1] As the design of reversible circuits does not exclusively aim for a realization on quantum devices, the more generic notion of *(circuit) line* is used for what is represented by a qubit.

$f_3'(x_1, x_2, x_3) = (x_1 \wedge x_2) \oplus x_3$. This function is reversible (as can be checked by applying all $2^3 = 8$ possible input assignments) and realizes the target functionality f by setting x_3 to a constant zero value (as the calculation $f = x_1 \wedge x_2 = (x_1 \wedge x_2) \oplus 0 = f_3'(x_1, x_2, 0)$ shows).

However, generating an embedding as sketched above is an exponentially complex task: In order to determine $\mu(f)$, all 2^n output patterns generated by the inputs have to be inspected. Hence, it has been tried to avoid this complexity by not aiming for a minimal result with respect to the number of additionally required outputs. In fact, since $\mu(f)$ can never exceed 2^n, at most $\lceil \log_2(2^n) \rceil = n$ additional garbage outputs are required [WKD11], i.e. *any* irreversible function can be embedded into a function $f' \colon \mathbb{B}^{n+m} \to \mathbb{B}^{m+n}$. But, the question remains how to specify the functionality of the newly added garbage outputs. Although heuristics assigning the additional outputs with a dedicated functionality as e.g. done in Example 7.1 are very promising, for a long time no solutions have been available which would guarantee that the resulting function f' is indeed reversible. Consequently, synthesis approaches relying on a reversible function description were applicable to small functions only.

Recently, however, a method has been proposed that employs QMDDs to a) determine $\mu(f)$ and b) compute an actual embedding of f, i.e. a reversible function f' with the minimal number of in-/outputs $m + \lceil \log_2(\mu) \rceil$ that embeds f [ZW17].

The basic idea of this approach is to construct (the QMDD of) the corresponding function matrix $\mathbf{M_f}$ where $m_{i,j} = 1$ if, and only if, f maps the input pattern corresponding to column j to the output pattern corresponding to row i. Otherwise $m_{i,j} = 0$. Having the function matrix $\mathbf{M_f}$, the number of input patterns that are mapped to a particular output pattern can be determined as the row sum of the corresponding column in $\mathbf{M_f}$.

Example 7.2 A *half adder* can be described by the multi-output Boolean function $f \colon \mathbb{B}^2 \to \mathbb{B}^2$ with component functions $f_1(x_1, x_2) = x_1 \wedge x_2$ (usually denoted as *carry*) and $f_2(x_1, x_2) = x_1 \oplus x_2 = x_1\overline{x_2} \vee \overline{x_1}x_2$ (usually denoted as *sum*). The corresponding truth table and function matrix representations are shown in Figs. 7.1a, b, respectively. Each line of the truth table is represented by a single 1 entry in the

x_1	x_2	f_1	f_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(a) Truth table

x_1 x_2

Inputs

Outputs

$$\begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 0 & 1 & 1 & 0 \\ 10 & 0 & 0 & 0 & 1 \\ 11 & 0 & 0 & 0 & 0 \end{array}$$

f_1/y_1

f_2/y_2

(b) Function matrix

x_1	x_2	y_1	y_2	χ_f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
⋮	⋮	⋮	⋮	⋮
1	1	1	1	0

(c) Char. function

Fig. 7.1 Representations of a half adder

function matrix. For instance, the third line stating that (1, 0) is mapped to (0, 1) is represented by the 1 in the third column (10), second row (01). By considering the row sums of the matrix, it can be seen that $\mu(f) = 2$.

In order to compute the row sums of $\mathbf{M_f}$, the matrix is multiplied with its transpose (i.e. $\mathbf{M_f} \cdot \mathbf{M_f}^T$ is computed). Then, the row sums occur on the diagonal of the resulting matrix and $\mu(f)$ can easily be determined as the maximum of these. Moreover, the above computation (which can be performed very efficiently using QMDDs) also serves as a necessary and sufficient check for reversibility: if, and only if, the result is the identity matrix, i.e. $\mu(f) = 1$, each input pattern is mapped to a unique output pattern and the original function is proven to be reversible.

It can be observed that the proposed algorithm for determining an actual (minimal) embedding has a strong similarity to the actual synthesis approach presented later in this section and can simply be modified in order to determine a corresponding reversible circuit as a "by-product", i.e. without significant overhead. Thus, in order to determine $\mu(f)$ and actually conduct the embedding, it only remains open how to construct a compact representation of the function matrix of f. This is covered in the following.

7.1.2 Construction of QMDDs for Boolean Functions

In order to exploit compact QMDD representations for the synthesis of Boolean components, the desired functionality (regardless of whether it is reversible or not) first of all needs to be represented in terms of a QMDD. A multi-output Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$ is commonly given in terms of descriptions of its *primary outputs* $f_1, \ldots, f_m$ (also termed *component functions*). These single-output Boolean functions $\mathbb{B}^n \rightarrow \mathbb{B}$ are commonly described in terms of Boolean Algebra, i.e. as *Sums of Products* (SOP), *Products of Sums* (POS), or alike.

While all these representations are essentially more compact representation of the truth table of f, we aim for a QMDD representation that is essentially a more compact representation of the function matrix of f. In order to bridge this gap, the main idea is to employ the *characteristic function* χ_f of f, i.e. a Boolean function $\mathbb{B}^n \times \mathbb{B}^m \rightarrow \mathbb{B}$ with n inputs labelled $x = x_1, \ldots, x_n$ and m inputs labelled $y = y_1, \ldots, y_m$, where $\chi_f(x, y) = 1$ if, and only if, $f(x) = y$. In other words, χ_f evaluates to true if, and only if, the backmost m inputs represent the correct output pattern that is generated when applying f to the input pattern specified by the first n inputs. Thus, the entries of the function matrix can be interpreted as the outcomes of χ_f.

Example 7.3 The characteristic function of the half adder from Example 7.2 is shown in Fig. 7.1c in terms of its truth table. Each line corresponds to one entry of the function matrix. More precisely, writing all columns of the function matrix on top of each other would yield the χ_f column of the truth table.

As it is infeasible to construct and store the whole function matrix at once due to its exponential complexity, we rather employ compact, graphical representations

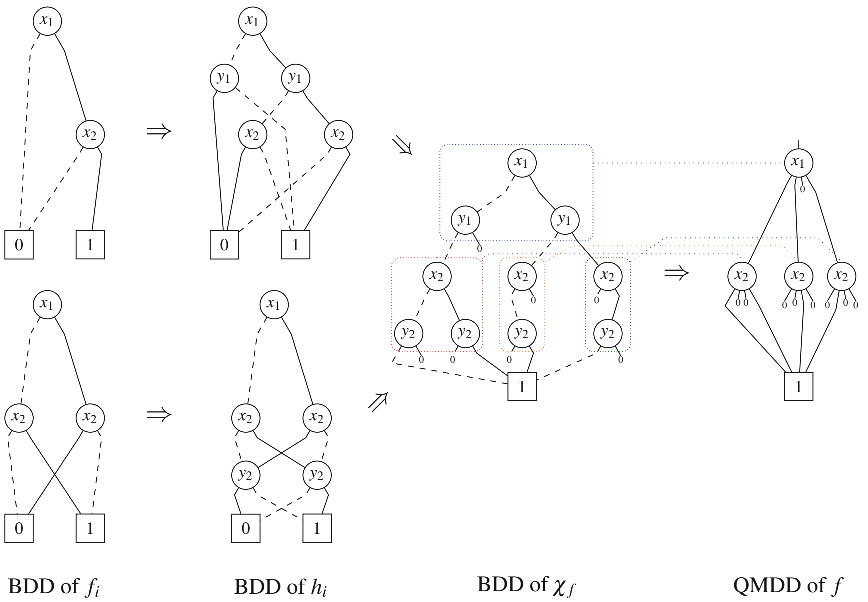


Fig. 7.2 Construction of the QMDD for the half adder

of Boolean functions (especially of the characteristic functions) in terms of BDDs from which the desired QMDD representation can then be derived directly without explicitly considering the function matrix.

Actually, there is a large body of research on how to derive BDD representations from various other, algebraic or net-list based, representations of Boolean functions [Som01].

Example 7.4 The BDDs for the component functions of the half adder reviewed in Examples 7.2 and 7.3 are shown on the left-hand side of Fig. 7.2.

Overall, there is a well-developed methodology for constructing the BDD representation of the component functions of f. These BDDs have then to be composed in a second step to obtain the BDD of the characteristic function χ_f. Since the outcomes of χ_f essentially describe the entries of the desired function matrix, the resulting BDD can eventually be transformed to a QMDD. In the following, these steps are described in more detail.

7.1.2.1 Generating the BDD of the Characteristic Function

In order to derive the BDD representing the characteristic function χ_f of a multi-output function $f\colon \mathbb{B}^n \rightarrow \mathbb{B}^m$, we first introduce new variables y_i for the primary outputs of f (referred to as *output variables* in the following). While the original (input) variables are used to encode the column index of the function matrix, the output variables encode rows. Then, we construct the characteristic function for each output. More precisely, we construct the helper functions h_i given by

$$h_i(x_1, \dots, x_n, y_i) = f_i(x_1, \dots, x_n) \odot y_i,$$

where $\odot$ denotes the XNOR-operation. This logical operation—and, thus, the entire function h_i—evaluates to true if, and only if, both operands are equal, i.e. $f_i(x_1, \dots, x_n) = y_i$. Consequently, the h_i-function can be interpreted as characteristic functions of the primary outputs of f.

Afterwards, the BDD of χ_f can be constructed by AND-ing the BDDs representing the h_i-functions as the following calculation shows (for an arbitrary $(x_1, \dots, x_n, y_1, \dots, y_m) \in \mathbb{B}^{n+m}$):

$$\begin{aligned}
&\chi_f(x_1, \dots, x_n, y_1, \dots, y_m) = 1 \\
\Leftrightarrow\, & f(x_1, \dots, x_n) = (y_1, \dots, y_m) \\
\Leftrightarrow\, & \forall i \in \{1, \dots, m\} : f_i(x_1, \dots, x_n) = y_i \\
\Leftrightarrow\, & \forall i \in \{1, \dots, m\} : h_i(x_1, \dots, x_n, y_i) = 1 \\
\Leftrightarrow\, & h_1(x_1, \dots, x_n, y_1) \wedge h_2(x_1, \dots, x_n, y_2) \wedge \dots \wedge h_m(x_1, \dots, x_n, y_m) = 1
\end{aligned}$$

Remark 7.1 If $n > m$, i.e. if f has more primary inputs than outputs, we pad the function with zeros in order to obtain a Boolean function with the same number of inputs and outputs, such that the resulting function matrix is square. More precisely, we add $n - m$ additional constant outputs/component functions $f_j \equiv 0$. While these can, in principle, be added at any position, we add them in front of the original outputs/component functions. If, in contrast, $m > n$, we add $m - n$ additional inputs that have no impact on the functionality of f. Again, these inputs can, in principle, be added at any position, but we add them in front of the original inputs. Overall, this ensures that the original functionality is represented by the sub-matrix of dimension $2^m \times 2^n$ in the top-left corner of the square function matrix. Moreover, this allows us to assume in the following that $n = m$ without restriction.

As the BDD representing χ_f is guaranteed to be exponential in size for the variable order $x_1 \succ \dots \succ x_n \succ y_1 \succ \dots \succ y_m$ (at least for reversible functions), we enforce an interleaved variable order $x_1 \succ y_1 \succ x_2 \succ y_2 \succ \dots \succ x_n \succ y_n$ when constructing the BDD for χ_f.

Example 7.5 Consider again the half adder example. The BDDs representing the helper functions $h_1 = f_1 \odot y_1$ and $h_2 = f_2 \odot y_2$ are computed using the BDD equivalent of the logical XNOR operation and are shown in Fig. 7.2 on page 83 (next to the BDDs representing f_1 and f_2). By AND-ing these BDDs, we obtain the BDD representing χ_f which is shown in the center of Fig. 7.2. In this BDD, all edges pointing to the zero-terminal are indicated by stubs for the sake of a better readability and to emphasize the similarity to the targeted QMDD.

7.1.2.2 Transforming the BDD into a QMDD

With a BDD in interleaved variable order representing χ_f, the matrix partitioning employed by QMDDs is already laid out implicitly. In fact, corresponding bits of the column and row indices are represented by different, but adjacent variables (x_i and y_i), while QMDDs combine these in a single variable. Consequently, the BDD of χ_f can be transformed into the QMDD for f using the general transformation rule shown in Fig. 7.3. However, there are two special cases that have to be treated separately:

- If an input variable x_i is skipped (more precisely: a vertex labelled by y_i is the child of a vertex not labelled by x_i), this implies the x_i vertex would be redundant, i.e. high and low edge point to the same vertex. This case can easily be handled by setting $f_{00} = f_{10} = f_0$ or $f_{01} = f_{11} = f_1$, respectively, as illustrated on the left-hand side of Fig. 7.4a. If, however, x_i is not an original input of the function, but has been introduced later in order obtain the same number of in- and outputs, we set $f_{10} = f_{11} = 0$ instead to ensure that the original functionality occurs only once in the final function matrix (as illustrated on the right-hand side of Fig. 7.4b).
- If an output variable level y_i is skipped (more precisely: the high or low edge of a vertex labelled by x_i point to a vertex labelled by $l \neq y_i$), this implies the skipped y_i vertex would be redundant (both children would be the same). This case can easily be handled by setting $f_{00} = f_{01} = f_0$ or $f_{10} = f_{11} = f_1$, respectively, before applying the general transformation rule. For instance, the case of a skipped variable on the low edge is illustrated in Fig. 7.4b.

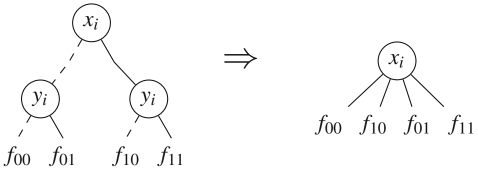

x_i
y_i
y_i
f_{00} f_{01} f_{10} f_{11}
$\Rightarrow$
x_i
f_{00} f_{10} f_{01} f_{11}

Fig. 7.3 General transformation rule from characteristic BDDs to QMDDs

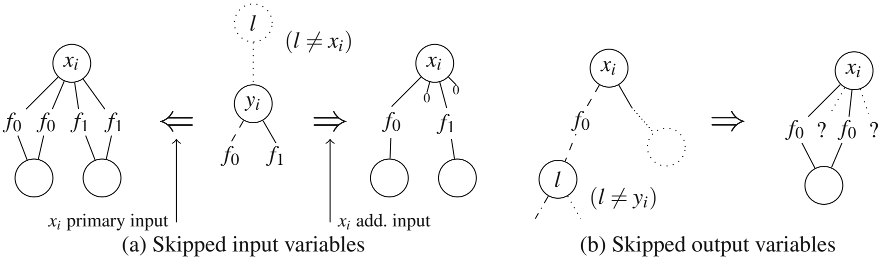


Fig. 7.4 Handling skipped variables

Example 7.6 Consider again the characteristic BDD shown in the center of Fig. 7.2 on page 83. Here, the single x_1 vertex and the leftmost x_2 vertex can be transformed to their QMDD equivalent by applying the general transformation rule. For the remaining x_2 vertices, the methodology for skipped y_2 output variables is to be applied. Overall, this yields the QMDD shown on the right-hand side of Fig. 7.2.

Overall, the procedure yields a QMDD representing the function matrix of any Boolean function f (which, in case of a reversible function, is a permutation matrix). In fact, as demonstrated in [NZWD17], corresponding representations for functions with around 100 in- and outputs can be determined within few CPU seconds on an ordinary machine.

7.1.3 QMDD-Based Synthesis of Reversible Circuits

Most line-aware synthesis approaches are based on an inefficient representation of the objective function (e.g. the transformation-based synthesis approach presented in [MMD03] is based on truth tables) and do not scale well. However, it turned out that the use of dedicated data-structures like QMDDs can significantly improve the scalability and one of the most scalable line-aware synthesis approaches is based on QMDDs [SWH+12].

Regardless of the underlying data-structure, there are two main ideas that most line-aware synthesis approaches have in common:

(1) they successively synthesize the component functions of f and
(2) they perform the synthesis in a reverse fashion.

More precisely, a cascade of gates is determined that transforms the original function f to the identity, i.e. the component functions are projections onto the input variables ($f_1 = x_1$, $f_2 = x_2$ and so on). As all gates are self-inverse, the desired circuit can afterwards be obtained by simply reversing the order of the gates. In the following, we outline the general scheme of QMDD-based synthesis by means of a matrix representation of the objective function. For this purpose, we use the running example shown in Fig. 7.5. For the sake of brevity and in order to be comparable with other line-aware approaches, we assume a reversible Boolean objective function for which the (transformation/function) matrix is in fact a permutation matrix (cf. Sect. 2.1). However, as discussed earlier, only small adjustments are required to make the presented technique applicable to irreversible functions, too [ZW17].

7.1.3.1 General Scheme

For a given Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^n$, we begin with the first component function f_1. In order to achieve an identity mapping, i.e. a projection on x_1, the approach aims to turn all $0 \mapsto 1$ into $0 \mapsto 0$ mappings and all $1 \mapsto 0$ into $1 \mapsto 1$

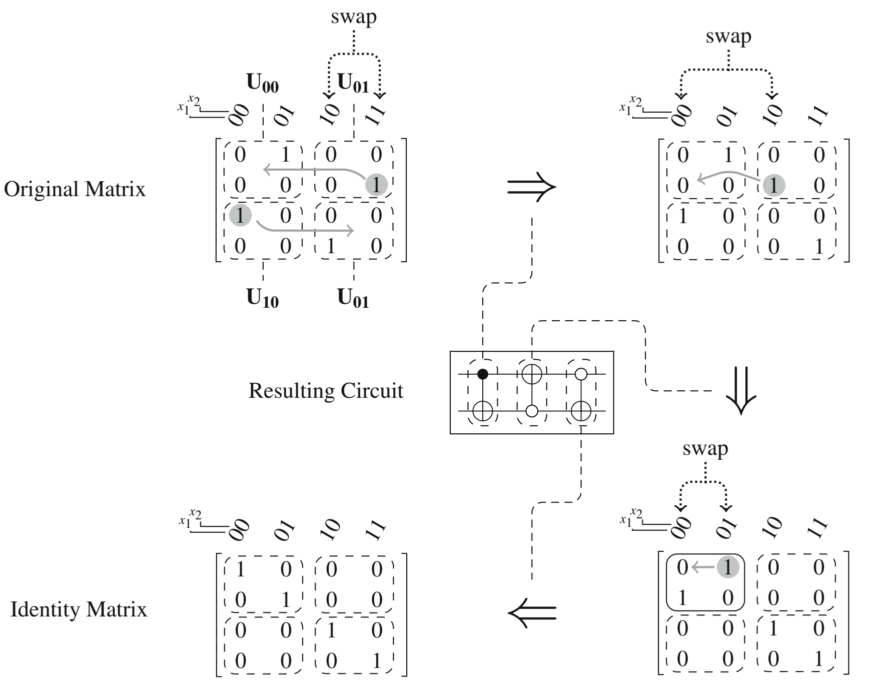


Fig. 7.5 Line-aware synthesis at the matrix level

mappings. At the level of transformation matrices, this corresponds to moving all non-zero entries from the off-diagonal blocks $\mathbf{U_{01}}$ and $\mathbf{U_{10}}$ to the $\mathbf{U_{00}}$ and $\mathbf{U_{11}}$ blocks on the diagonal, respectively, as shown in the top-left part of Fig. 7.5. The following Lemma tells us that we may even concentrate on only one of these blocks, as the other one is cleared automatically.

Lemma 7.1 *Let* $\mathbf{M}$ *be a* $2^n \times 2^n$ *permutation matrix, i.e.* $\mathbf{M} \in \{0, 1\}^{2^n \times 2^n}$*, partitioned into four sub-matrices* $\mathbf{U}_{00}, \mathbf{U}_{01}, \mathbf{U}_{10}, \mathbf{U}_{11}$ *of dimension* $2^{n-1} \times 2^{n-1}$ *as follows:*

$$\mathbf{M} = \begin{bmatrix} \mathbf{U}_{00} & \mathbf{U}_{01} \\ \mathbf{U}_{10} & \mathbf{U}_{11} \end{bmatrix}$$

Then $\mathbf{U}_{01} = [0]_{2^{n-1} \times 2^{n-1}}$ *if, and only if,* $\mathbf{U}_{10} = [0]_{2^{n-1} \times 2^{n-1}}$.

Proof Since $\mathbf{M}$ is a permutation matrix with a single 1 in each row and column, there must be exactly k non-zero entries in the leftmost k columns as well as in the topmost k rows (for $1 \leq k \leq 2^n$). If $\mathbf{U}_{01}$ is "empty", all non-zero entries of the topmost 2^{n-1} rows (blocks $\mathbf{U}_{00}$ and $\mathbf{U}_{01}$) are gathered in the leftmost 2^{n-1} columns, i.e. block $\mathbf{U}_{00}$. Consequently there can not be any single non-zero entry in the lower halves of these columns, i.e. block $\mathbf{U}_{10}$ is also "empty". The converse is proven in an analogous fashion.

So, how do we clear block $\mathbf{U}_{01}$ and turn all $0 \mapsto 1$ mappings into $0 \mapsto 0$ mappings? The main idea for this purpose is to swap entire columns which corresponds to applying appropriate multiple-controlled Toffoli (MCT) gates. To this end, recall that each column corresponds to a particular input pattern $x_1 \dots x_n$ (called its *signature* in the following). Now, in order to swap exactly two columns whose signatures differ in one position x_k only (called x_k*-partner*), we apply an MCT gate with the target on x_k and positive/negative controls on the remaining lines corresponding to the columns' signature. If some of these controls are dropped, a larger set of columns will be swapped.

Example 7.7 Consider again the matrix in the upper left part of Fig. 7.5. Applying an MCT gate with the target on x_2 and a positive control on x_1 (i.e. the controlled NOT gate which is the first gate of the circuit shown in the center of Fig. 7.5) will swap the columns 10 and 11. The resulting matrix is shown in the upper right part.

This relation between applying MCT gates and swapping columns can be employed for clearing block $\mathbf{U_{01}}$. More precisely, the following two phases are carried out in rotation unless all non-zero entries from block $\mathbf{U_{01}}$ have been moved to block $\mathbf{U_{00}}$.

Swapping Phase Determine all columns in block $\mathbf{U_{01}}$ which contain a 1 while their x_1-partner in block $\mathbf{U_{00}}$ is empty. These columns can be swapped directly by applying an MCT gate with the target on x_1 and controls corresponding to the columns' signatures.

Moving Phase Determine those x_1-partners which are empty in both blocks and move a non-zero entry within block $\mathbf{U_{01}}$ to such an empty column. This is done by applying MCT gates with a positive control on x_1 (which ensures that the column movement is only performed in block $\mathbf{U_{01}}$ and not also in block $\mathbf{U_{00}}$) and targets on all lines where the signatures of source and target column differ.

After performing the swapping phase, i.e. after moving all currently movable entries to block $\mathbf{U_{00}}$, or if no directly movable entry can be found in the first place, new movable entries in block $\mathbf{U_{01}}$ are created in the moving phase. This guarantees the creation of at least one new movable column which can be moved in the next switching phase.

Example 7.8 Consider again the matrix in the upper left part of Fig. 7.5. In the first swapping phase nothing can be done as there is no directly movable entry in $\mathbf{U}_{01}$ (the x_1-partner of column 11 is already occupied in $\mathbf{U}_{00}$). However, the x_1-partners 00 and 10 are both empty, such that by applying an MCT gate with a target on x_2 and positive control on x_1 in the moving phase (cf. Example 7.7), we obtain a directly movable column 10. The corresponding matrix is shown in the upper right part. In the following swapping phase, this column 10 can be swapped with column 00 using an MCT gate with a target on x_1 and a negative control on x_2. The corresponding gate is shown as the second gate of the circuit in the center of the figure; the negative control is indicated by a white dot ○. By applying this gate, block $\mathbf{U}_{01}$ is cleared and the synthesis for f_1 is complete. The resulting matrix is shown in the lower right part of Fig. 7.5.

After this process is completed for f_1, it is iteratively performed for all remaining component functions. On the matrix level, this means that we have to consider the two blocks $\mathbf{U}_{00}$ and $\mathbf{U}_{11}$ separately; more general, the number of blocks doubles with every step. In order to address a particular block within the matrix, the MCT gates need to be equipped with additional controls that limit the column movement to the respective block.

Example 7.9 Consider again the matrix in the lower right part of Fig. 7.5. As blocks $\mathbf{U}_{01}$ and $\mathbf{U}_{10}$ are empty, we can continue with the diagonal blocks, i.e. we perform the synthesis for f_2. Since $\mathbf{U}_{11}$ is already an identity matrix, we only need to consider $\mathbf{U}_{00}$. Here, both columns 00 and 01 need to be swapped. This is done by the MCT gate with a target on x_2 shown as the rightmost gate of the circuit in the center. Here, a negative control on x_1 limits the swapping to $\mathbf{U}_{00}$. Finally, we obtain an identity matrix (shown in the lower left part of Fig. 7.5) and the synthesis is complete.

Overall, the entire approach could be performed directly on the matrices, but then we would quickly face severe scalability issues as permutation matrices scale even worse than truth tables. It is QMDDs what makes the approach actually scalable: QMDDs offer an efficient way to represent permutation matrices, to find the signatures of empty or occupied columns, and to apply Toffoli gates—used for moving columns within $\mathbf{U}_{01}$ or from $\mathbf{U}_{01}$ to $\mathbf{U}_{00}$—to the *entire* matrix. Indeed, as demonstrated in [SWH+12], the approach could be applied to functions with up to 100 inputs and additionally improved the quantum cost significantly in comparison to previous approaches.

7.2 Synthesis of Clifford Group Operations

In the previous section, we considered the synthesis of Boolean components which is an important sub-task in the design of quantum logic. However, the corresponding approaches do not allow for the realization of more general quantum functionality. For this matter, several approaches have been proposed, all of which rely on the fact that one-qubit gates together with the *controlled NOT* (CNOT) gate form a *universal* gate library that is sufficient to realize any given unitary matrix. In fact, determining corresponding quantum circuits has a rich history (cf. Sect. 6.2.2). The drawback of these generic approaches is, however, that they lead to a significant number of gates (even for a small number of qubits) and that they rely on a set of arbitrarily parameterized one-qubit gates. The latter poses a severe obstacle since, in physical realizations, these must be approximated by a restricted set of gates, in particular when fault-tolerant methods are applied.

In this section, we provide an alternative synthesis approach (based on the description in [NWD14a]) that considers synthesis of quantum circuits implementing Clifford Group operations.[2] It has been shown in [Got97] that each Clifford Group

[2] The term *Clifford group* for this set of operations has been introduced by Bolt et al. [BRW61].

operation can be realized by a cascade composed of just Hadamard, Phase, and CNOT gates, i.e. from a group-theoretical viewpoint these gates are the *generators* of the Clifford Group. Thus, we avoid relying on a generic gate library and, instead, can realize quantum functionality with a precise and established set of gates which can be implemented in a fault-tolerant fashion [BMP+00]. At the same time, this restricts the applicability of the approach (arbitrary unitary matrices are not supported). However, Clifford group circuits are essential for many quantum applications and cover core aspects of quantum functionality such as superposition, entanglement, and phase shifts [KMM12]. Moreover, their functionality is sufficient for various quantum applications, particularly as stabilizer circuits for error-correcting codes [Mer07], but also for the realization of the Greenberger-Horne-Zeilinger experiment [GHZ89], for quantum teleportation [BBC+93], or dense quantum coding [BW92].

This compromise on applicability enables a synthesis methodology allowing for the realization of considerably more compact quantum circuits. We explicitly exploit the effects of the clearly defined gate library in order to directly modify the given unitary matrices to be synthesized. These effects are discussed in Sect. 7.2.1.2. In contrast to the previously proposed (generic) approaches and as confirmed by an experimental evaluation, this enables a reduction of the circuit sizes by several orders of magnitude. By additionally using a compact data-structure, namely QMDDs (cf. Chap. 4), our approach enables an efficient processing of the respective unitary matrices.

The remainder of the section is structured as follows. Section 7.2.1 introduces the main concepts of the proposed synthesis scheme, before the resulting synthesis algorithm is described in detail in Sect. 7.2.2. A theoretical analysis of the algorithm is carried out in Sect. 7.2.3 and experimental results are presented in Sect. 7.2.4.

7.2.1 Main Concepts of the Synthesis Approach

In this section, we introduce the general idea of the proposed synthesis approach. Furthermore, we describe the effect of the Clifford Group generators (Hadamard, Phase, CNOT) to a given transformation matrix. Based on that, the actual synthesis algorithm is afterwards described in Sect. 7.2.2.

7.2.1.1 General Idea

The task of synthesis is to determine a quantum circuit representing the desired quantum functionality **F** given in terms of a transformation matrix. We already know that all circuits considered here can be realized by a cascade composed of Hadamard, Phase, and CNOT gates. Therefore, applying transformation matrices representing these gates modifies **F** and for a distinct choice of these matrices we will eventually reach the identity matrix. Hence, the main goal of the proposed synthesis approach is to determine a sequence of quantum gates $g_1 \dots g_l$ with this property. For this purpose, we identify the following three steps that are also illustrated in Fig. 7.6.

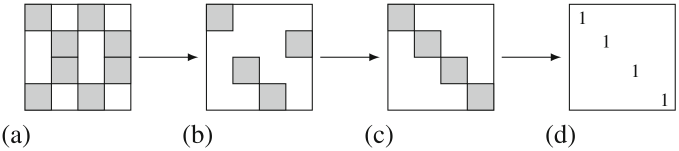


Fig. 7.6 General scheme of the synthesis procedure

1. *Eliminate superposition*, i.e. apply quantum gates so that all multiple non-zero matrix entries in rows/columns are removed. This leads to a matrix that has a single non-zero entry per row/column (as illustrated in Fig. 7.6b), each of which has magnitude 1. Thus, the matrix is structurally equivalent to a permutation matrix. The corresponding circuit maps basis states to (possibly different) basis states and potentially applies phase shifts.
2. *Diagonalize*, i.e. establish the structure of a diagonal matrix as shown in Fig. 7.6c. By this, the structure of the transformation matrix is already equal to the structure of the identity matrix, i.e. each basis state is mapped onto itself. However, phase shifts still might be applied.
2. *Eliminate phase shifts* to eventually reach the identity matrix (Fig. 7.6d).

All gates applied in order to perform these steps lead to a circuit realizing the inverse of the given transformation matrix **F**. Since Hadamard and CNOT gates are self-inverse and the inverse of the Phase gate is $\mathbf{S}^3 = \mathbf{S} \cdot \mathbf{Z}$, the actually desired circuit can then be derived by simply reversing the order of all gates and replacing Phase gates by their inverses. Alternatively, we can convert **F** to its inverse in advance which can be performed efficiently using a dedicated data-structure like QMDDs.

7.2.1.2 Effect of the Clifford Group Generators

All synthesis steps sketched above can be accomplished by a clever sequential application of Hadamard, Phase, and CNOT gates, the generators of the Clifford Group. But before the respective algorithm is described in detail, we briefly investigate the effect of these gates to a transformation matrix **F**.

For simplicity, we illustrate all operations on a 4 × 4 transformation matrix **F** over two qubits x_1 and x_2. The generalization to larger matrices with more qubits is straightforward. In the following, we write $\mathbf{U}_{\text{target}}$ for the transformation matrix that represents the (uncontrolled) U operation applied to a target qubit. Similarly, we use $\mathbf{U}_{\text{target}}^{\text{control}}$ for a controlled **U**-gate, i.e. $\mathbf{X}_{x_2}^{x_1}$ denotes the CNOT gate with control qubit x_1 and target x_2.

Figure 7.7 summarizes how the application of important Clifford operations affects a given 4 × 4 transformation matrix **F** with columns 00, 01, 10, and 11, i.e. $\mathbf{F} = (00, 01, 10, 11)$. For our purposes, three main effects are important:

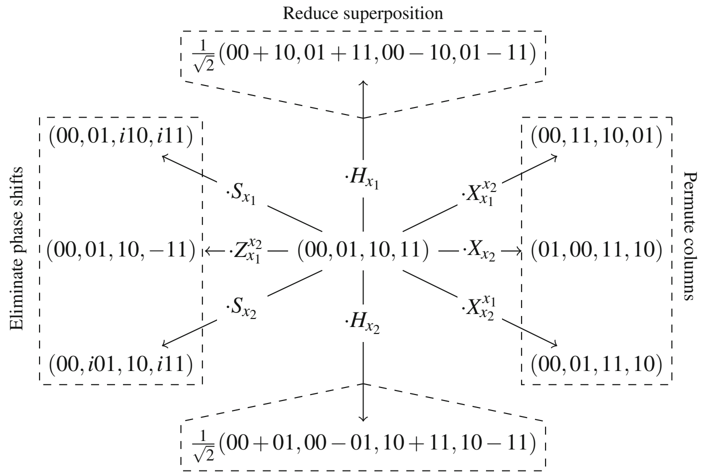


Fig. 7.7 Operation scheme of gate matrices

7.2.1.3 Permutation of Columns

The CNOT gates $\mathbf{X}_{x_2}^{x_1}$ and $\mathbf{X}_{x_1}^{x_2}$ lead to matrices with columns 10 and 11 or 01 and 11 being interchanged, respectively. To additionally interchange column 00 with another one, an (uncontrolled) NOT gate ($\mathbf{X}$) can be applied. This gate can be generated with Hadamard and Phase gates, more precisely $\mathbf{X} = \mathbf{H} \cdot \mathbf{S} \cdot \mathbf{S} \cdot \mathbf{H}$. Hence, by composing gates $\mathbf{X}_{x_2}^{x_1}$, $\mathbf{X}_{x_1}^{x_2}$, and $\mathbf{X}_{x_2}$, the columns of $\mathbf{F}$ can be re-arranged until any desired permutation is achieved. Note that for larger matrices we are no longer able to achieve any desired permutation of the columns, but we can still move a (single) column to any desired place.

7.2.1.4 Reducing Superposition

Hadamard gates $\mathbf{H}_{x_1}$ and $\mathbf{H}_{x_2}$ link together pairs of columns as illustrated in Fig. 7.7, thereby allowing to create or reduce superposition. Hence, this operation is important for the first step of the synthesis approach sketched above. However, reduction of superposition is only possible for suitable pairs of columns (i.e. for 00 and 10 or 01 and 01) and only if the columns differ by no more than signs of the entries. If other pairs need to be linked together (e.g. 00 and 11), the corresponding columns have to be re-arranged before. For this purpose, permutation of columns as described above can be applied.

7.2.1.5 Eliminating Phase Shifts

Finally, Fig. 7.7 shows that Phase gates $\mathbf{S}_{x_1}$ and $\mathbf{S}_{x_2}$ apply a phase shift of i to a particular subset of columns of $\mathbf{F}$. Beyond that, they do not alter the order of the columns. Applied again, they change this phase shift to -1 and $-i$ successively and, finally, remove the phase shift completely. Obviously, this is relevant for the third step of the synthesis approach sketched above. As Phase gates also only work on pairs of columns, we additionally might need controlled Z gates defined by

$$\mathbf{Z}_{x_2}^{x_1} = \mathbf{Z}_{x_1}^{x_2} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

These apply a phase shift by -1 to a more restricted set of columns than Phase gates (here: a single column). This gate type can readily be constructed using Hadamard and CNOT gates, more precisely $\mathbf{Z}_{x_2}^{x_1} = \mathbf{H}_{x_1} \cdot \mathbf{X}_{x_1}^{x_2} \cdot \mathbf{H}_{x_1}$.

Exploiting these effects of the Clifford group generators, the sketched synthesis approach can be realized as described next.

7.2.2 *Algorithm*

Based on the main concepts introduced in the previous section, we now describe the resulting synthesis approach in detail. The approach follows the three steps from Fig. 7.6. Furthermore, we employ QMDDs as reviewed in Chap. 4 as an efficient representation of transformation matrices on which all steps are conducted. All steps are illustrated by a running example, namely the transformation matrix depicted in Fig. 7.8a and the corresponding QMDD depicted in Fig. 7.8b. We will exploit the specific property of Clifford group transformation matrices that all non-zero entries are multiples of a *basis weight*. More precisely, there is a complex number ω such that each non-zero entry is of the form $u \cdot \omega$ for $u \in \{\pm 1, \pm i\}$. This property follows from the theory of stabilizer circuits as discussed in [Got98] and will be proved in Sect. 7.2.3.

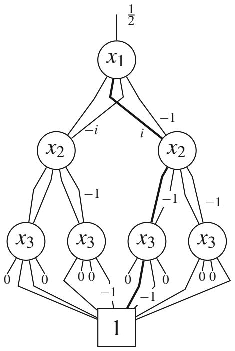
7.2.2.1 Eliminating Superposition

As discussed in the previous section, superposition can be eliminated using Hadamard gates. More precisely, superposition involving a suitable pair of columns with no phase shifts can straightforwardly be tackled using a single Hadamard gate applied to the corresponding qubit.

$$
\begin{array}{c|cccc|cccc}
 & \multicolumn{8}{c}{\text{Inputs}} \\
x_1 x_2 x_3 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & \frac{\mathbf{i}}{\mathbf{2}} & 0 & \frac{-i}{2} \\
001 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & \frac{-i}{2} & 0 & \frac{i}{2} & 0 \\
010 & \frac{1}{2} & 0 & \frac{-1}{2} & 0 & \frac{-i}{2} & 0 & \frac{-i}{2} & 0 \\
011 & 0 & \frac{-1}{2} & 0 & \frac{1}{2} & 0 & \frac{-i}{2} & 0 & \frac{-i}{2} \\
\hline
100 & 0 & \frac{-i}{2} & 0 & \frac{-i}{2} & 0 & \frac{-1}{2} & 0 & \frac{1}{2} \\
101 & \frac{-i}{2} & 0 & \frac{-i}{2} & 0 & \frac{1}{2} & 0 & \frac{-1}{2} & 0 \\
110 & \frac{-i}{2} & 0 & \frac{i}{2} & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\
111 & 0 & \frac{i}{2} & 0 & \frac{-i}{2} & 0 & \frac{1}{2} & 0 & \frac{1}{2}
\end{array}
$$

Outputs

(a)

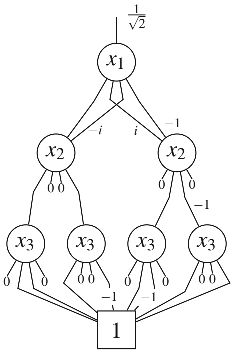


(b)

Fig. 7.8 Matrix and QMDD representation of a 3-qubit quantum circuit

$$
\begin{array}{c|cccc|cccc}
x_1 x_2 x_3 & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{i}{\sqrt{2}} \\
001 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{-i}{\sqrt{2}} & 0 \\
010 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{-i}{\sqrt{2}} & 0 & 0 & 0 \\
011 & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} & 0 & \frac{-i}{\sqrt{2}} & 0 & 0 \\
\hline
100 & 0 & \frac{-i}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{-1}{\sqrt{2}} \\
101 & \frac{-i}{\sqrt{2}} & 0 & 0 & 0 & 0 & 0 & \frac{1}{\sqrt{2}} & 0 \\
110 & 0 & 0 & \frac{-i}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0 & 0 \\
111 & 0 & 0 & 0 & \frac{i}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 & 0
\end{array}
$$

(a)



(b)

Fig. 7.9 Running example after applying H_{x_2}

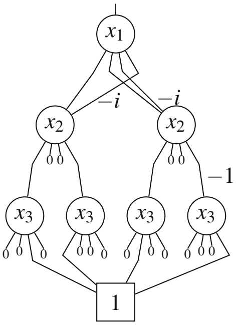
Example 7.10 In the running example from Fig. 7.8a, it can be seen that column 001 matches pairwise to column 011 (i.e. the entries differ only by sign). Hence, applying a Hadamard gate on qubit x_2 reduces the superposition of the matrix and leads to a matrix as shown in Fig. 7.9a.

However, cases might be encountered where no Hadamard gates can be applied directly. Then, the columns have to be re-arranged using CNOT gates and possible phase shifts have to be removed using Phase gates. This is illustrated by the following example before the actually applied elimination scheme is described next.

Example 7.11 The resulting matrix from Fig. 7.9a still includes superposition. In order to eliminate this, we need to derive a "valid" pair of columns for which a Hadamard gate can be applied. We choose columns 001 and 111 and first use a CNOT gate $\mathbf{X}_{x_2}^{x_1}$ to move column 111 to 101. Then, we perform a Phase gate on qubit

$$\begin{array}{c|cccc:cccc|}
{}_{x_1 x_2 x_3} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
001 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
011 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
\hdashline
100 & 0 & -i & 0 & 0 & 0 & 0 & 0 & 0 \\
101 & 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & -i & 0 \\
111 & 0 & 0 & 0 & i & 0 & 0 & 0 & 0 \\
\hline
\end{array}$$

(a)



(b)

Fig. 7.10 Running example after eliminating superposition

x_1 to remove the phase shift to column 001. Finally, the application of a Hadamard gate at qubit x_1 eventually eliminates all superposition in this matrix. The resulting matrix is shown in Fig. 7.10a and the corresponding QMDD in Fig. 7.10b.

Overall, superposition can gradually be eliminated by repeating the following steps.

1. Determine the first two non-zero entries α_1, α_2 in the first row of the matrix and store their quotient $q = \frac{\alpha_1}{\alpha_2}$.
2. Rearrange the columns in such a way that their column index only differs in one place x_d. This can be done by choosing a place x_d where the indices initially differ and applying controlled NOT gates (controlled by x_d) on any other place where the indices differ.
3. If $q = \pm i$, perform a Phase gate on x_d.
4. Afterwards, perform a Hadamard gate on x_d.

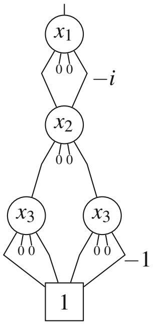
Using QMDDs, these steps can be conducted efficiently. Since the basis weight ω of a matrix can be obtained through the weight of the root edge of its QMDD, it is easy to check whether there is superposition in the matrix or not. Moreover, for each QMDD vertex, it can be stored whether there is a non-zero entry in the first row in the left/right half of the represented sub-matrix. This information can be computed by a single depth-first-traversal of the QMDD and needs a very small amount of additional memory, but can accelerate the search for non-zero entries dramatically.

7.2.2.2 Diagonalization

Once superposition has been removed from the matrix, there is a single non-zero entry per row and column (as in Fig. 7.10a). By unitarity, all of these entries and, hence, also the basis weight must have magnitude 1, i.e. the matrix is structurally equivalent to a permutation matrix and the corresponding circuit maps basis states to

$$\begin{array}{c|cccc:cccc|}
{}_{x_1 x_2 x_3} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\
\hline
000 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
001 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
010 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
011 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
\hdashline
100 & 0 & 0 & 0 & 0 & -i & 0 & 0 & 0 \\
101 & 0 & 0 & 0 & 0 & 0 & -i & 0 & 0 \\
110 & 0 & 0 & 0 & 0 & 0 & 0 & -i & 0 \\
111 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & i
\end{array}$$

(a)



(b)

Fig. 7.11 Running example after diagonalization

other basis states—possibly with phase shifts. Hence, in order to derive the structure of a diagonal matrix, the columns have to be permuted only. This can be done by applying suitable NOT and CNOT gates. In order to determine which gates to apply, we compute *line functions* f_{x_i} for each qubit x_i. These denote the (logic) formula that expresses for which inputs (columns) we get an output $|1\rangle$ on the respective qubit.

Example 7.12 We read from the current matrix in Fig. 7.10a that

$$\begin{aligned}
f_{x_1} &= 001 \vee 011 \vee 100 \vee 110 = x_1 \oplus x_3 \\
f_{x_2} &= 010 \vee 011 \vee 110 \vee 111 = x_2 \\
f_{x_3} &= 000 \vee 011 \vee 100 \vee 111 = \overline{x_2 \oplus x_3}
\end{aligned}$$

In order to achieve a diagonal structure, we need $f_{x_i} = x_i$. To establish this for x_1, we need to XOR x_3 by applying $\mathbf{X}_{x_1}^{x_3}$. For x_3, this can similarly be accomplished by applying $\mathbf{X}_{x_3}^{x_2}$ and $\mathbf{X}_{x_3}$. This leads to a matrix as shown in Fig. 7.11a.

It can be shown, that the line functions are always such XOR products of the qubits (in fact, the proof will be provided in Sect. 7.2.3.2). However it may happen that x_i does not appear in f_{x_i} (but $x_j \neq x_i$ does). In this case, we firstly need to swap the qubits by applying $\mathbf{X}_{x_j}^{x_i}$, $\mathbf{X}_{x_i}^{x_j}$, and $\mathbf{X}_{x_j}^{x_i}$.

Thus, the respective gates to be applied are derived by computing the line function for each qubit and, based on it, successively applying NOT/CNOT gates until the desired diagonal structure results. Using QMDDs, the line functions can easily be computed in a compressed form (as a Binary Decision Diagram) if the respective qubit is on the top level of the diagram.

More precisely, the following algorithm is applied (all variables are initially marked unvisited):

1. Pick an unvisited variable x and move it to the top of the QMDD by variable interchange.

2. Compute the output function f_x of x.
3. If f_x does not depend on x, i.e. if the top vertex of its compressed representation is not labelled by x (but by y), perform the CNOT gates $\mathbf{X}_y^x$, $\mathbf{X}_x^y$, and $\mathbf{X}_y^x$.
4. For each variable $y \neq x$ on which f_x depends, perform a CNOT gate with target x and controlled by y, i.e. $\mathbf{X}_x^y$.
5. Mark x as visited.

After all variables have been visited, each basis state is mapped onto itself and, thus, a diagonal matrix structure has been achieved. For the running example, this leads to the QMDD as shown in Fig. 7.11b.

7.2.2.3 Eliminating Phase Shifts

Finally, possible phase shifts are eliminated, i.e. all diagonal entries of the matrix are equalized. This is conducted by Phase gates (performing shifts by $\pm i$) and controlled Z gates (performing partial shifts by -1).

Example 7.13 The phase shifts in the current matrix shown in Fig. 7.11a can be eliminated by applying a Phase gate at qubit x_1 ($\mathbf{S}_{x_1}$). This transforms the values $\pm i$ to ± 1. To remove the phase shift -1 of $|011\rangle$, a controlled Z gate with target x_3 and controlled by x_2 ($\mathbf{Z}_{x_3}^{x_2}$) is applied. This eventually leads to the identity matrix and, hence, terminates the synthesis.

Note that we do not require the first entry of the diagonal (first row, first column) to have value $+1$. Any complex value of magnitude 1 is acceptable and all other diagonal entries are transformed to the same value. This leads to a matrix that might not be the identity matrix itself, but that is equivalent up to *global phase* and, hence, physically indistinguishable (cf. Sect. 2.2.1).

Since phase shifts are indicated by edge weights $\pm i$ and -1 in the QMDD, they can easily be eliminated by applying the corresponding gate to qubits as illustrated in Fig. 7.12. In order to address edges on lower levels, re-ordering of the QMDD

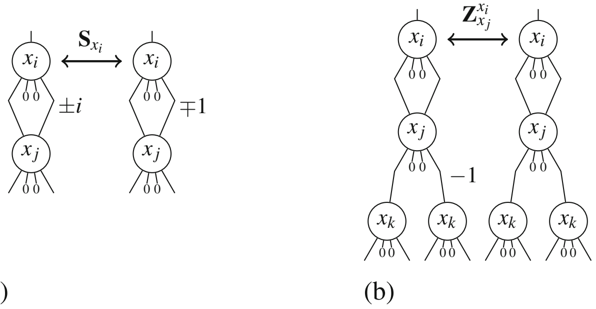


Fig. 7.12 Removing non-trivial edge weights from a QMDD

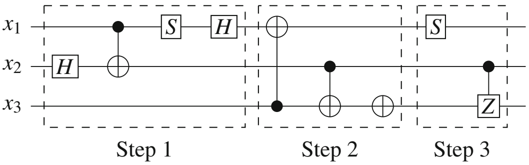


Fig. 7.13 Resulting quantum circuit

structure is applied. Eventually, all phase shifts can be eliminated by "moving" the respective qubit variable to the root level and applying the gates as shown in Fig. 7.12.

Overall, performing the respective steps as described above eventually leads to the quantum gate cascade depicted in Fig. 7.13. As explained in Sect. 7.2.1.1, this realizes the inverse of the given unitary matrix of our running example (see Fig. 7.8). Before an experimental evaluation of the algorithm is presented in Sect. 7.2.4, in the following section we provide a theoretical analysis and detailed proof for the algorithm's convergence.

7.2.3 Theoretical Analysis

In this section, we provide a theoretical analysis of the proposed synthesis scheme. More precisely, it is shown that, given a Clifford group operation, the approach always terminates and leads to the desired circuit. In addition, we provide an upper bound for the size of the resulting circuits, i.e. the maximum number of quantum gates that may result from applying the proposed approach.

For this purpose it is helpful to consider a formal, mathematical definition of the Clifford group. The matrices $\mathbf{X}$, $\mathbf{Z}$, $\mathbf{Y} = -i\mathbf{XZ}$, and the identity matrix $\mathbf{I}$ generate the so-called Pauli group P_1. Higher-dimensional Pauli groups P_n consists of all n-qubit tensor products of these matrices. These correspond to circuits where there is either none or a single $\mathbf{X}$, $\mathbf{Y}$, or $\mathbf{Z}$ gate applied to each qubit. Additionally, each tensor product may appear with an overall global phase of ± 1 and $\pm i$ and, hence, $|P_n| = 4 \cdot 4^n$. In this context, the *Clifford group* C_n is defined to be the (group-theoretical) stabilizer of the Pauli group P_n, i.e.

$$C_n := \{\mathbf{M} \in U(2^n) \mid \mathbf{MPM}^\dagger \in P_n \text{ for all } \mathbf{P} \in P_n\}$$

where $U(2^n)$ denotes the set of unitary $2^n \times 2^n$ matrices and $\mathbf{M}^\dagger$ denotes the adjoint (complex conjugate of the transpose) of $\mathbf{M}$. In other words, the Clifford group consists of all circuits that map operators from the Pauli group onto each other. For this reason these circuits are also called *stabilizer circuits*. Thus, applying an n-qubit Clifford group circuit $\mathbf{M}$, the Pauli group operators $\mathbf{X}_{x_i}$ and $\mathbf{Z}_{x_i}$ evolve to $\overline{\mathbf{X}}_i = \mathbf{MX}_{x_i}\mathbf{M}^\dagger$ and $\overline{\mathbf{Z}}_i = \mathbf{MZ}_{x_i}\mathbf{M}^\dagger$ $(i = 1, \ldots, n)$ which are also Pauli group operators by the above definition of C_n. Moreover, since the basis state $|0\ldots 0\rangle$ has the unique property that it is a simultaneous ($\lambda = 1$)-eigenvector of all operators $\mathbf{Z}_{x_1}, \ldots, \mathbf{Z}_{x_n}$, the circuit will map it to the unique ($\lambda = 1$)-eigenvector of the operators $\overline{\mathbf{Z}}_1, \ldots, \overline{\mathbf{Z}}_n$. This results from the fact that $\mathbf{M}$ (and $\mathbf{M}^\dagger$) map eigenvectors of any Pauli group operator $\mathbf{P}$ to eigenvectors of $\overline{\mathbf{P}} = \mathbf{M}\mathbf{P}\mathbf{M}^\dagger$ with the same eigenvalue (and vice versa). More precisely, if v is an eigenvector of $\mathbf{P}$ with eigenvalue α (i.e. $\mathbf{P}v = \alpha v$), then

$$\overline{\mathbf{P}}(\mathbf{M}v) = (\mathbf{M}\mathbf{P}\mathbf{M}^\dagger)(\mathbf{M}v) = \mathbf{M}\mathbf{P}(\mathbf{M}^\dagger\mathbf{M})v = \mathbf{M}\mathbf{P}v = \alpha(\mathbf{M}v),$$

i.e. $\mathbf{M}v$ is an eigenvector of $\overline{\mathbf{P}}$ with the same eigenvalue α. Now, from the image of $|0\ldots0\rangle$ under $\mathbf{M}$ (which corresponds to the first column of the transformation matrix), the image of any other basis state, i.e. any other column of the matrix, can easily be computed by using $\overline{\mathbf{X}}_1, \ldots, \overline{\mathbf{X}}_n$ as the following example shows.

Example 7.14 Let ψ be the image of $|0\ldots0\rangle$ under $\mathbf{M}$, i.e. $\psi := \mathbf{M}(|0\ldots0\rangle)$. From this, the image of $|10\ldots0\rangle = \mathbf{X}_{x_1}(|0\ldots0\rangle)$ can be obtained by computing

$$\begin{aligned}\mathbf{M}(|10\ldots0\rangle) &= \mathbf{M}\mathbf{X}_{x_1}(|0\ldots0\rangle)\\ &= \mathbf{M}\mathbf{X}_{x_1}(\mathbf{M}^\dagger\mathbf{M})(|0\ldots0\rangle) \; |\mathbf{M}^\dagger\mathbf{M} = \mathbf{I}\\ &= (\mathbf{M}\mathbf{X}_{x_1}\mathbf{M}^\dagger)\mathbf{M}(|0\ldots0\rangle)\\ &= \overline{\mathbf{X}}_1(\psi).\end{aligned}$$

Overall, any column of the transformation matrix differs from the first column (representing the image of $|0\ldots0\rangle$) only by the application of a particular Pauli matrix, i.e. by permutation and phase shifting of the entries.

We refer to [Got98] for more information about the theory of stabilizer circuits and proceed by applying this theory to each step of the synthesis algorithm—thereby obtaining worst case estimations about the number of required gates.

7.2.3.1 Eliminating Superposition

The first step of the synthesis algorithm heavily relies on the fact that Clifford group matrices have a basis weight. More precisely, there is a complex number ω such that each non-zero entry is of the form $u \cdot \omega$ for $u \in \{\pm1, \pm i\}$.

We first proof the existence of such a number for the first column of the matrix, i.e. the image of $|0\ldots0\rangle$. More precisely, we proof the following

Theorem 7.1 *Let* $\mathbf{M}$ *be a Clifford group operation on n qubits and*

$$|\psi\rangle := \mathbf{M}|0\ldots0\rangle = \sum_{k=0}^{2^n-1} \alpha_k |k\rangle.$$

Then, the set $N = \{|k\rangle \mid \alpha_k \neq 0\}$ *contains* 2^l *basis states for some* $l \in \{0, \ldots, n\}$. *Moreover, for any two such basis states* $|k_1\rangle, |k_2\rangle \in N$ *the corresponding amplitudes satisfy* $\alpha_{k_1} = u \cdot \alpha_{k_2}$ *for* $u \in \{\pm1, \pm i\}$.

Proof The operators $\overline{\mathbf{Z}}_1, \ldots, \overline{\mathbf{Z}}_n$ generate a subgroup $S \leq P_n$. As all generators commute pairwise, S is abelian. Moreover, as $|\psi\rangle$ is a simultaneous $(\lambda = 1)$-eigenvector for all generators, it is a $(\lambda = 1)$-eigenvector for all operators $\mathbf{T} \in S$. Consequently, any operator $\mathbf{T} \in S$ maps basis states from N onto each other (up to a phase shift of $\pm i$ or -1) and, thus, acts on N as a permutation (ignoring possible phase shifts). More precisely, as $\mathbf{T}$ is a Pauli matrix, it performs a bit flip on a specific set of qubits, i.e. its action on basis states can be described as $|x\rangle \mapsto |x \oplus t\rangle$ for some $t \in \mathbb{B}^n$. These vectors span a linear sub-space L of $\mathbb{B}^n$. Any such sub-space has dimension 2^l for some $l \in \{0, \ldots, n\}$. This can be seen as follows: given a vector $v \in L$, the sum $v + t$ is also L if, and only if, $t = v + (v + t)$ is in L. Thus, adding a vector to the span either does nothing to the spanned sub-space (if the vector is already contained) or doubles its size.

Consequently, the S-orbit of any $|k\rangle \in N$, i.e. the set of elements in N that can be reached by applying an arbitrary operator from S, contains 2^l elements for some $0 \leq l \leq n$. Now, the restriction of $|\psi\rangle$ on those basis states contained in an S-orbit yields a vector $|\psi'\rangle$. This vector is a simultaneous $(\lambda = 1)$-eigenvector of all operators in S, as with each basis state $|x\rangle$ also $|x \oplus t\rangle$ is contained in $|\psi'\rangle$ (with the correct amplitude). As the simultaneous $(\lambda = 1)$-eigenvector of $\overline{\mathbf{Z}}_1, \ldots, \overline{\mathbf{Z}}_n$ is uniquely defined, we must have $|\psi\rangle = |\psi'\rangle$. This implies that there is only one S-orbit in N (S is a transitive group operation on N) and finally shows that $|N| = 2^l$.

Moreover, for arbitrary $|k_1\rangle, |k_2\rangle \in N$ there is a Pauli matrix $\mathbf{T}_{k_1,k_2} \in S$ which maps these basis state onto each other. As Pauli matrices only apply bit flips and/or phase shifts by ± 1 or $\pm i$, the corresponding amplitudes α_{k_1} and α_{k_2} only differ by a factor $u \in \{\pm 1, \pm i\}$.

From this, the existence of a basis weight immediately follows using the fact that the remaining columns differ from the first column only by permutation and phase-shifting of the entries (cf. Example 7.14). Thus, it is sufficient to consider only the first two non-zero entries in the first row of the matrix, since, if we manage to link these together with a Hadamard gate, the basis weight is increased by a factor of $\sqrt{2}$. Moreover, if there are initially 2^l non-zero entries in each column (by unitarity the basis weight has magnitude $2^{-l/2}$) we need to apply l Hadamard gates and for each of these at most one Phase and n CNOT gates. Since $l \leq n$, we overall need at most n Hadamard and Phase gates, and n^2 CNOT gates for the first synthesis step.

7.2.3.2 Diagonalization

In the second step of the synthesis algorithm, we employ the fact that Clifford group matrices with a permutation matrix structure have line functions that are XOR products of the qubits. An equivalent formulation is that flipping a fixed bit of any input basis state always flips the same set of output bits. We have seen that flipping a fixed input bit corresponds to applying a particular Pauli matrix to the image (in Example 7.14: $\overline{\mathbf{X}}_1$). This matrix actually does not depend on the assignment of the other bits, and, since Pauli matrices correspond to circuits with either none or a single

$\mathbf{X}$, $\mathbf{Y}$, or $\mathbf{Z}$ gate on each qubit, it actually flips an output bit if, and only if, there is an $\mathbf{X}$ or $\mathbf{Y}$ on the corresponding line. This shows that flipping a fixed input bit always flips the same set of output bits. Thus, the line functions are XOR products of the inputs and diagonalization can be performed as outlined in Sect. 7.2.2.2. In doing so, we need at most one NOT gate (equivalently two Hadamard and Phase gates each) and n CNOT gates for each qubit.

7.2.3.3 Removing Phase Shifts

In the last step of the synthesis algorithm, phase shifts have to be removed. Since we have already reached a diagonal form, when flipping an input bit only the same output bit is flipped. Thus, the Pauli matrices $\overline{\mathbf{X}}_1, \ldots, \overline{\mathbf{X}}_n$ (to which the bit-flipping Pauli operators $\mathbf{X}_{x_1}, \ldots, \mathbf{X}_{x_n}$ evolve) ensure to have a bit-flipping gate ($\mathbf{X}$ or $\mathbf{Y}$) on the designated qubit and none or $\mathbf{Z}$ gates on the other qubits.

For the identity circuit we need $\mathbf{X}_{x_i} = \overline{\mathbf{X}}_i$ for all $i = 1, \ldots, n$. In other words, the $\overline{\mathbf{X}}_i$ need to be composed of a single $\mathbf{X}$ (on the respective line) and no other gates, with an overall positive sign.

To ensure $\mathbf{X}$s (instead of $\mathbf{Y}$s) and positive signs, it is sufficient to remove possible phase shifts between $|0\ldots0\rangle$ and basis states where a single bit is set to 1 (first level phase shifts). Similarly, to ensure that there is no $\mathbf{Z}$ gate remaining, we must remove possible phase shifts for basis states with exactly two bits set to 1 (second level phase shifts), as demonstrated in the following

Example 7.15 If there is a $\mathbf{Z}$ gate remaining on the qubit x_j in $\overline{\mathbf{X}}_i$, then this shows in a phase shift between the images of $|0\ldots0\rangle$ and the basis state where exactly bits i and j are set to 1. This can be seen as follows: we can reach the image of the latter state from the image of $|0\ldots0\rangle$ by applying $\overline{\mathbf{X}}_j$ and $\overline{\mathbf{X}}_i$ (cf. Example 7.14). By applying $\overline{\mathbf{X}}_j$ (which has a positive sign and $\mathbf{X}$ on line x_j) bit j is flipped, but the phase is not changed. Now, by applying $\mathbf{X}_i$ not only bit i flips, but since bit j is set we obtain a phase shift of -1 (from the $\mathbf{Z}$ gate on qubit x_j).

Overall, it is sufficient to ensure that there is no phase shift between any two basis states with at most two bits set to 1. Clearly, Phase and controlled Z gates can be used to eliminate phase shifts between $|0\ldots0\rangle$ and basis states which have at most two bits set to 1. To eliminate all first level phase shifts we need at most 3 Phase gates per qubit ($\mathbf{S}^4 = I$). Second level phase shifts are eliminated by at most n^2 controlled Z gates which translate to $2n^2$ Hadamard and n^2 CNOT gates.

In total, our algorithm is capable of realizing each Clifford group functionality as a quantum circuit with at most $3n^2$ CNOT gates and $9n + 2n^2$ single qubit gates (i.e. Hadamard and Phase gates). This is only slightly larger than the (theoretical) upper bound derived in [AG04].

7.2.4 Experimental Results

The synthesis approach discussed above has been implemented on top of the QMDD package. In this section, we evaluate the results obtained by the approach and compare them to synthesis schemes previously proposed in [SBM06, SAZS11]. To this end, arbitrary transformation matrices with up to 20 qubits (denoted *arbitrary*) as well as established quantum functionality taken from [Mer07] and realizing Shor's 9-qubit error-correcting code (denoted by *9qubitN1* and *9qubitN2*), a 7-qubit encoding (denoted by *7qubitcode*), as well as an error syndrome measurement circuit for a 5-qubit code (denoted by *5qubitcode*) have been used.

The results are summarized in Table 7.1. The first column provides the identifiers of the respective benchmarks followed by its number of qubits. In the remaining columns, the costs, i.e. the number of gates, of the resulting circuits are provided. It is a common understanding that (physical) implementations of CNOT gates are more error-prone and have greater delays than one-qubit gates. Therefore and in accordance with the evaluation of [SBM06, SAZS11], we distinguish between the number of one-qubit gates and the number of CNOTs in Table 7.1. Furthermore, the best available results from [SBM06, SAZS11] are provided for comparison. Finally, the run-time of the proposed synthesis approach is provided in the last column. All experiments have been conducted on a 2.8 GHz Intel Core i7 machine with 8 GB of main memory running Linux.

First of all, we emphasize again that the approaches proposed in [SBM06, SAZS11] rely on a gate library composed of arbitrarily parameterized one-qubit gates together with the CNOT gate. This poses a severe obstacle since physical realizations and particularly fault-tolerant methods impose limitations on the set of gates that may be used. In contrast, the approach presented here can realize the considered quantum functionality with a precise and established set of gates including only Hadamard, Phase, and CNOT gates.[3]

Table 7.1 clearly shows that, by using the proposed method, much more efficient quantum circuits can be realized for Clifford Group functionality compared to the more generic approaches presented before. In fact, reductions of several orders of magnitudes of CNOT gates can easily be obtained.

[3] Note again that the controlled Z gates applied e.g. for eliminating phase shifts are eventually realized by a Hadamard–CNOT–Hadamard cascade.

Table 7.1 Experimental evaluation of the Clifford group synthesis approach

Benchmark	#Qubits	Prev. approach [SBM06]	Prev. approach [SAZS11]		Proposed approach		
		#CNOTs	#CNOTs	#one-qubit gates	#CNOTs	#one-qubit gates	Time (s)
Arbitrary transformation matrices							
arbitrary4	4	100	112	138	15	33	<0.01
arbitrary5	5	444	480	537	26	43	<0.01
arbitrary6	6	1,868	1,976	2,209	36	46	<0.01
arbitrary7	7	7,660	8,040	8,528	45	55	0.02
arbitrary8	8	31,020	32,456	33,455	61	72	<0.01
arbitrary9	9	124,844	130,408	134,415	68	61	0.03
arbitrary10	10	500,908	522,920	531,022	87	89	0.05
arbitrary11	11	2,006,700	2,094,376	2,110,669	102	98	0.10
arbitrary12	12	8,032,940	8,382,888	8,448,077	144	135	0.28
arbitrary15	15	$\approx 5.14 \cdot 10^8$	–	–	203	173	2.37
arbitrary20	20	$\approx 5.27 \cdot 10^{11}$	–	–	217	222	26.40
Quantum functionality taken from [Mer07]							
5qubitcode	9	124,844	–	–	28	28	<0.01
7qubitcode	7	7,660	–	–	11	19	<0.01
9qubitN1	9	124,844	–	–	8	3	<0.01
9qubitN2	17	$\approx 8.23 \cdot 10^9$	–	–	34	4	<0.01

Benchmark: Name of benchmark – #Qubits: Number of qubits – #CNOTs: Number of CNOT gates – #one-qubit gates: Number of one-qubit gates

7.3 Conclusions

In this chapter, we considered the synthesis of quantum circuits which is a major task in the design of quantum logic. More precisely, we considered two important classes of quantum functionality, namely Boolean components as well as Clifford group circuits. While Boolean components are a vital part of many quantum algorithms, Clifford group circuits are essential for many quantum applications such as stabilizer circuits, quantum teleportation, and more. In contrast to previous approaches for the synthesis of arbitrary quantum functionality, we avoid relying on a generic gate library and, instead, exploit the specific effects of the Clifford Group generators to the unitary matrix to be synthesized. Experiments confirmed that, compared to the generic approaches presented before, quantum circuits with several orders of magnitude less gates can be realized using the proposed approach. Both presented synthesis algorithms natively allow for the use with compact representations (especially QMDDs). In fact, experimental evaluations confirm that, by actually doing so, these algorithms become the most scalable state-of-the-art solutions for the respective synthesis tasks. Overall, this demonstrates the importance of compact representations for the synthesis of quantum circuits as well as their practical applicability.

Probably the most challenging question is whether the proposed synthesis approach for Clifford group operations can somehow be extended to more general functionality. To this end, it can be shown that adding a single, arbitrary external gate to the Clifford group library allows for universal quantum computation [NRS01, Theorem 6.5]. In most cases, the T gate ($R_z(\pi/4)$) is employed for this purpose.

In order to extend the proposed synthesis scheme, it seems, however, more appropriate to use the Toffoli gate instead. In fact, this would allow for arbitrary column permutations in the first phase of the algorithm (removing superposition) and the use of the QMDD-based synthesis approach [SWH+12] in the diagonalization phase. However, in the general case, unitary transformation matrices do not exhibit a basis weight. Nonetheless, one may try to adapt the general idea of step-wise increasing the minimal moduli of all matrix entries. By using a dedicated normalization scheme that propagates smallest moduli to the top, the matrix entries with the minimal modulus could easily be located and selected. For this purpose, the usually applied normalization—which can illustratively be described as a "leftmost non-zero value" normalization—may be replaced with the "leftmost smallest magnitude" normalization described in [NWD13]. Then, all edge weights in the resulting QMDD (except the root edge) have a magnitude ≥ 1 and after removing all edges with a weight larger than 1 only the desired entries with the minimal modulus remain. Preliminary investigations in this direction have yielded promising results. In fact, the modified algorithm was able to successfully synthesize several instances of Grover's algorithm. However, in many other cases it got stuck and was not able to establish a permutation matrix structure. Overall, this approach deserves a more thorough investigation given that the synthesis of arbitrary quantum functionality is a considerably ambitious task and no really satisfying approaches exist so far. Nonetheless, the sketched extension still relies to a quite large extent on properties of the corresponding transformation matrices.

Clearly, it would be more desirable to have an approach that predominantly operates on more efficient representations like decision diagrams, especially QMDDs. Ideally, one would like to have a direct mapping that relates (the size of) a decision diagram with a corresponding quantum circuit as it is e.g. possible in conventional logic for the multiplexer circuits which can be derived directly from BDDs. Unfortunately, sub-structures (sub-matrices) of unitary matrices do not have to be unitary and it is completely open how to deal with this problem in order to derive a circuit mapping for QMDDs or a similar structure.

Chapter 8
Correctness of Multiple-Valued Implementations

In conventional logic CAD and in equal measure also in quantum logic CAD, it is vital to ensure the correctness of the transformations and mappings throughout the design flow. More precisely, the desired functionality is realized at different levels of abstraction and several optimization steps are conducted at each level. Consequently, in order to ensure a consistent design, different (circuit) descriptions of the same functionality need to be checked for equivalence.

The major problem to this is that equivalence has to be verified for all possible inputs. As quantum operations are linear, it actually suffices to check whether both descriptions yield the same output for all basis states only. However, as there are still exponentially many of those (with respect to the size of the considered quantum system), it quickly becomes infeasible to perform all these checks e.g. by simulating the (circuit) behavior for each of the basis states. In contrast, decision diagrams are very promising candidates for this task and have been employed for this purpose in conventional logic design, as they describe the entire functionality in a compact and unique manner. In fact, once the corresponding representations are built, the comparison can be conducted for all possible inputs (basis states) at once. Moreover, if standard techniques like *unique tables* are used in the implementation of the decision diagram, the final comparison can be performed in constant time by simply checking whether the root edges are identical.[1]

In fact, several methods for equivalence checking of quantum functionality have been proposed in the past (e.g. based on simulation with QuIDDs [VMH07], combined with a pre-processing using Boolean satisfiability [YM10], based on XQDDs [WLTK08] or QMDDs [WGMD09]). However, all of these only support

[1] Note that only for QMDDs this is even true if the descriptions are not perfectly equal, i.e. they lead to identical transformation matrices, but are equal up to a *global phase*, i.e. the matrices are equal up to a multiplicative factor $e^{i\theta}$. All other decision diagrams reviewed in Chap. 3 require much more effort to determine equivalence up to global phase—a phenomenon which frequently occurs in quantum logic design.

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_8

two-level quantum systems composed of qubits. But, the considered quantum systems offer multiple levels to be exploited. These levels are readily accessible and using them for state preparation and read-out has been demonstrated [NAB+09]. As a result, computations can be performed on so-called *qudits* rather than qubits. Researchers have investigated possible exploitation of these additional levels e.g. for matters of simplified implementation or improved design of quantum operations. They were able to show that multi-level systems are useful for many promising applications and provide several practical advantages in the design of particular operations (see e.g. [CDNS11, LBA+08]). This is discussed in detail later in Sect. 8.1. As a consequence, several approaches for representing and realizing quantum functionality in various quantum systems exist. This raises the question of how to ensure the correctness of such multiple-valued implementations, which essentially comes down to verifying whether or not two quantum operations given in different quantum systems indeed realize the same functionality.

In this chapter, we address the problem of checking functional equivalence between operations that are realized in multi-level quantum systems. This explicitly includes comparisons between realizations in different dimensions, i.e. quantum systems with a different number of levels. For this purpose, we first discuss and define functional equivalence in this context in Sect. 8.2.1. Afterwards, in Sect. 8.2.2, a verification scheme based on the formal representation of quantum functionality by unitary matrices is proposed. Since these matrices grow exponentially with the number of considered qubits, we additionally demonstrate how the proposed scheme can be incorporated into data-structures such as QMDDs (cf. Chap. 4) which are explicitly suited for the compact representation of quantum functionality. By this, an equivalence checker for multi-level quantum systems results. The efficiency of the proposed scheme is confirmed by an experimental evaluation in Sect. 8.3—considering a wide range of operations realized in different quantum systems. The following description is based on [NWD14b].

8.1 Multi-level Quantum Systems

Research on quantum computation is considered in numerous facets. Originally, the exploitation of quantum.mechanical phenomena e.g. for database search [Gro96], factorization [Sho94], and other applications has been discussed in a purely theoretical fashion. But in the past decades several physical realizations have been proposed—including prototypical implementations based on trapped ions [CZ95], superconducting qubits [Gal07], and photons [OFV09]. However, most of these considerations and implementations focused on two-level quantum systems, i.e. systems based on qubits with the two basis states $|0\rangle$ and $|1\rangle$.

But, as a matter of fact, quantum computation allows for multiple basis states. Instead of qubits, *d-levelled qudits* are then used as basic building blocks. These do not rely on only two orthogonal basis states (cf. Definition 2.6 in Sect2.2.1) but a total of d basis states $|0\rangle, |1\rangle, \ldots, |d-1\rangle$. More precisely, a qudit is described by a

d-dimensional Hilbert space, where the *state space* is formed by all superpositions $|\psi\rangle = \sum_{i=0}^{d-1} \alpha_i |i\rangle$ for complex-valued α_i with $\sum_{i=0}^{d-1} |\alpha_i|^2 = 1$. Prominent examples of qudits are *qutrits* ($d = 3$) and *ququarts* ($d = 4$) which received most attention so far [KGRS03, MHT05, GSG+04, MMSK06, CDNS11].

Multiple qudits with levels $d_0, \ldots, d_{n-1}$ form a $\hat{d}$*-level quantum system* where $\hat{d}$ is the maximum of the d_i. The underlying Hilbert space is the tensor product of the respective spaces of the single qudits. Accordingly, the state of such systems can be expressed by a state vector of length $\prod_{i=0}^{n-1} d_i$ and is given by a linear combination of the *tensor states* $|x_1 \ldots x_n\rangle$ where $0 \leq x_i < d_i$ for $1 \leq i \leq n$. Note that tensor states are the tensor products of basis/ground states of the individual qudits. As they also form a basis of the entire state space, in the previous chapters we simply used the term *basis states* when referring to them. In this chapter, however, we often require to refer to one of the d_i basis/ground states of individual qudits and, thus, use the term basis state solely in this context.

Operations over qudits are described by extended transformation matrices.

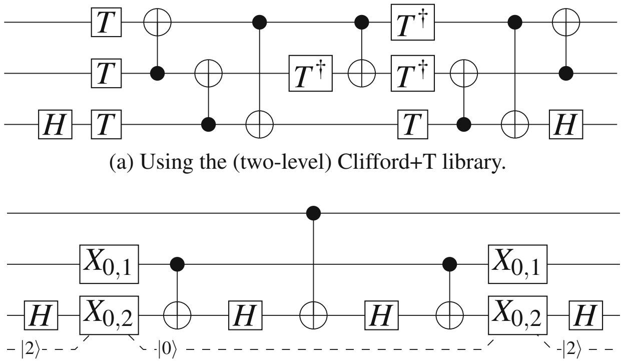
Example 8.1 The qutrit operation X which exchanges the basis states $|0\rangle$ and $|2\rangle$ can be described by the matrix

$$X_{0,2} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \text{ while } H_{0,1} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & \sqrt{2} & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix}$$

represents the ququart operation that performs the Hadamard operation on basis states $|0\rangle$ and $|1\rangle$, leaving the remaining basis states untouched.

Multi-level systems are not only of theoretical interest [GSG+04], but also prove to be useful for promising applications of quantum computation (see e.g. [CDNS11, LBA+08]). Moreover, the use of multi-level quantum systems offers several practical advantages compared to qubit systems. More precisely:

- Multi-level quantum systems allow for much more efficient realizations of multi-qubit operations [LBA+08]. For example, Fig. 8.1a shows a minimal implementation (in terms of T-depth, i.e. the number of sequential T operations) of a Toffoli gate within the Clifford+T library, i.e. based on a two-level system (taken from [AMMR13]). The same functionality can be realized with significantly less operations in a multi-level system (qutrit) as shown in Fig. 8.1b (taken from [LBA+08]).
- A theoretical analysis showed that ququart operations may have a general advantage over qubit operations when it comes to the realization of generalized Toffoli gates. In fact, mapping these Toffoli gates to quantum operations using qubit-based techniques (e.g. [BBC+95]) requires an exponential effort. In contrast, a recently proposed four-valued approach can realize each Toffoli operation with linear complexity [SWM12].



(a) Using the (two-level) Clifford+T library.

(b) Using a multi-level system.

Fig. 8.1 Realizations of the Toffoli gate

These advantages lead to an increased interest in multi-level quantum systems and the implementation of quantum operations in various dimensions. Consequently, as for qubit systems, the synthesis of general quantum functionality has also been studied for multi-level systems [MS00, BOB05, DW13]. In [DW13], a generalized CNOT operation is suggested that reacts on an arbitrary control state and swaps an arbitrary pair of states on the target qudit. The advantage of this approach is that it is physically realizable by using standard CNOT operations and certain laser beams (Rabi oscillations) to swap basis states. By this, synthesis of many important multi-level circuits becomes possible with established technology.

Overall, various representations and realizations of quantum functionality for different quantum systems exist. But whether or not two given quantum operations in different dimensions indeed realize the same functionality has hardly been considered yet. This issue is addressed in the following, i.e. we present a scheme which automatically checks for the equivalence of operations in multi-level systems.

8.2 Equivalence Checking in Multi-level Quantum Systems

While, thus far, equivalence checking for quantum functionality has intensely been considered in the past (e.g. based on simulation with QuIDDs [VMH07], combined with a pre-processing using Boolean satisfiability [YM10], or based on XQDDs [WLTK08]), usually only operations in the same dimension have been compared. In contrast, we propose a verification scheme which is capable of proving the functional equivalence between quantum operations even if they are realized in different dimensions. For this purpose, this section first discusses fundamental preconditions and provides a precise definition of the functional equivalence that

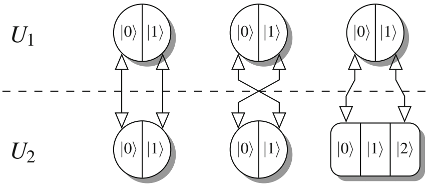


Fig. 8.2 Possible mapping of basis states between quantum systems

we are going to address. Afterwards, the proposed equivalence checking scheme is introduced. Based on these concepts, we illustrate an efficient implementation of the proposed scheme.

8.2.1 Functional Equivalence for Quantum Operations

The purpose of equivalence checking is to verify whether two quantum operations realize the same functionality. In the following, we denote the two quantum operations to be compared by U_1 and U_2. The underlying quantum systems may have different dimensions d_1 and d_2 (for U_1 and U_2, respectively), where we assume $d_2 \geq d_1$ (without loss of generality). In order to check for equivalence between U_1 and U_2, it is important to have a precise definition of which basis states of the quantum systems actually correspond to each other. Basis states can either be *shared states*, if there is a corresponding basis state in the other system, or *don't-care states*, if there is no counterpart.

Example 8.2 Consider two quantum operations U_1 and U_2, which are realized in a 2-level and 3-level quantum system, respectively. More precisely, the 2-level system consists of three qubits whereas the 3-level system is a hybrid system composed of two qubits and a single qutrit. A possible mapping between basis states is shown in Fig. 8.2. Here, all basis states are shared states except the $|1\rangle$ state of the qutrit in U_2, which has no counterpart in U_1 and, thus, is a don't-care state.

In the following, the correspondence of basis states is represented by a function ψ. It is assumed that ψ is either derived from the specification of the respective technology mapping or directly provided by the designer.

In addition, we require that both quantum systems are composed of the same number of qudits and do not consider corner cases in which e.g. a ququart is realized by two qubits or even more scattered mappings. Although the proposed approach could be extended in order to support also these cases, our simplification is strongly motivated by the following facts:

- It is a natural requirement to enable the same set of measurements for U_1 and U_2. Since only entire qudits can be measured, this is only possible if there is a one-to-one relation between qudits in both systems.

- In order to interpret a measurement result correctly, there may not be cross-mappings between basis states that do not belong to corresponding qudits.

Don't-care states may be employed during the operation, like e.g. in the multi-level realization of the Toffoli operation shown in Fig. 8.1b. But, we assume that neither input nor corresponding output states carry a don't-care component.

Having these definitions and assumptions, two quantum operations U_1 and U_2 are *functionally equivalent* ($U_1 \equiv U_2$) if they perform an equivalent transformation on shared states. The behaviour on don't-care states, however, may be arbitrary.

Example 8.3 Consider the matrix $H_{0,1}$ from Example 8.1 describing a Hadamard operation on a ququart. Assuming the trivial mapping of shared states $\psi(|i\rangle) = |i\rangle$ (for $i = 0, 1$), $H_{0,1}$ is equivalent to the standard Hadamard operation H on qubits. However, with the same mapping, this is not the case for

$$H_{0,2} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & \sqrt{2} & 0 & 0 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & \sqrt{2} \end{pmatrix},$$

which also performs a Hadamard operation on a ququart, but on different basis states.

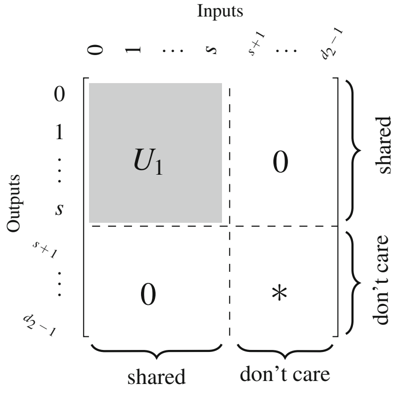
8.2.2 Proposed Equivalence Checking Scheme

Assume two quantum operations U_1 and U_2 (realized in quantum systems with dimensions $d_2 \geq d_1$) together with a mapping ψ and the corresponding distinction in shared states and don't-care states. Then, functional equivalence of these operations can be verified in two steps:

1. Check whether the sub-matrices of U_1 and U_2 representing the mapping of shared input states to shared output states are equivalent.
2. Check whether the sub-matrices of U_1 and U_2 representing the mapping of don't-care input states to shared output states (and vice versa) are zero matrices.

If both checks evaluate to true, then U_1 and U_2 are equivalent. This scheme is illustrated by means of Fig. 8.3 on the basis of single qudit systems. More precisely, Fig. 8.3 shows the matrix representing the quantum operation U_2, i.e. within the higher level system. Without loss of generality, assume that the basis states $|0\rangle, \ldots, |s\rangle$ of the U_1-system are shared states ($s < d_1$) and that ψ maps them to the basis states $|0\rangle, \ldots, |s\rangle$ of the U_2-system. The remaining states are assumed to be don't-cares. Then, the top-left $(s + 1) \times (s + 1)$ sub-matrix of U_2 in Fig. 8.3 represents the mapping of shared input to shared output states. If $U_1 \equiv U_2$, this mapping obviously has to be equivalent to the corresponding mapping described in U_1. This is checked in Step 1.

Fig. 8.3 Matrix of U_2 to be compared against U_1

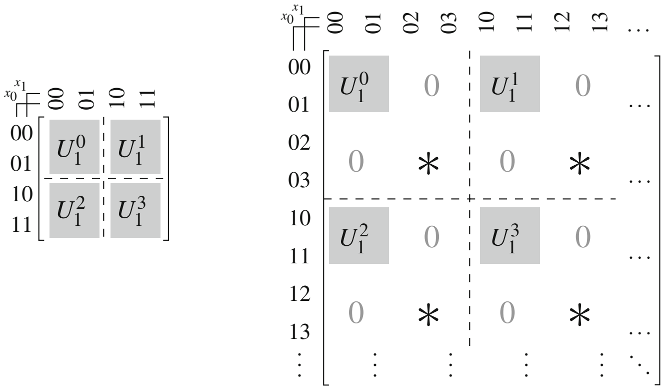


Next, we exploit the fact that, as discussed in Sect. 8.2.1, only superpositions of shared basis states are applied to U_2, i.e. the basis states $|s+1\rangle$, ..., $|d_i - 1\rangle$ are always prepared (expected) with zero amplitude for input (output) states. As a result and in order to keep the overall matrix unitary, no further mappings from don't-care states to shared states (represented in the top-right sub-matrix) and from shared states to don't-care states (represented in the bottom-left sub-matrix) can exist. That is, the corresponding matrices have to be zero matrices. This is checked in Step 2. Note that we do not need to consider the bottom-right sub-matrix representing the mapping from don't-care input to don't-care output states, since arbitrary behaviour is allowed for those mappings.

Example 8.4 Once again, consider the usual Hadamard operation H as well as the operation $H_{0,1}$ from Example 8.1. together with the trivial mapping of shared states between the underlying 2- and 4-level quantum systems, i.e. $\psi(|i\rangle) = |i\rangle$ for $i = 0, 1$.

The 4-level operation $H_{0,1}$ is equivalent to the 2-level operation H, because (1) the mappings of shared states are equivalent and (2) no mappings from don't-care states to shared states and vice versa exist. In contrast, these properties do not hold for the operation $H_{0,2}$ (from Example 8.3), showing its non-equivalence to the other two operations.

This scheme can be extended to quantum systems composed of an arbitrary number of qudits. However, the checks have to consider the more scattered distribution of the respective sub-matrices. This is sketched in Fig. 8.4, where U_1 (realized in a 2-level quantum system) is to be compared to U_2 (realized in a 4-level quantum system composed of two ququarts). Here we assume that there are no don't-care states in the U_1-system and again, without loss of generality, that ψ maps the basis states $|0\rangle$ and $|1\rangle$ of the U_1-system to the shared basis states $|0\rangle$ and $|1\rangle$ of the U_2-system. As can be seen, all (shared and don't-care) basis states are considered separately for each qudit. Accordingly, the sub-matrices to be checked against U_1, the zero matrices, and don't-care matrices ($*$) are scattered throughout the whole transformation matrix. This, however, does not restrict the applicability of the proposed equivalence



(a) U_1 (2-level system).

(b) U_2 (4-level system).

Fig. 8.4 Equivalence of operations in multi-qudit systems

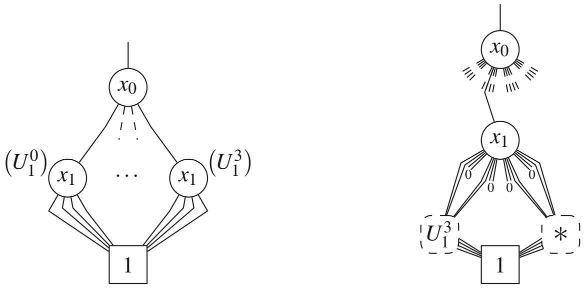
checking scheme, but of course harms the efficiency of the checks. Note that this is even more the case for more complex mappings of shared states. Then, the matrices under consideration can be in a more dispersed shape and the scheme might result in checking the equivalence of many small non-adjacent sub-matrices. An efficient implementation of this scheme even in these cases is essential and described next.

8.2.3 Implementation Using QMDDs

While the concepts introduced above are sufficient to check equivalence between arbitrary quantum operations, the matrix representations used above for illustration constitute a serious hurdle to the applicability of the proposed scheme. In fact, matrix descriptions grow exponentially with the number of qudits in a system. Hence, a naive implementation based on matrices is infeasible for quantum systems of all but small size.

In order to address this issue, we implemented the proposed scheme by means of the QMDD data-structure (cf. Chap. 4). In this data-structure, each vertex represents a matrix which is partitioned into four sub-matrices (for qubit systems). Each sub-matrix is then represented by a successor of the current vertex. In case of multi-level quantum systems, the number of successors grows accordingly with the number of basis states.

Example 8.5 Figure 8.5 sketches the QMDD representations of the quantum operations already discussed in Fig. 8.4. As U_1 assumes a two-level quantum system, the overall matrix is partitioned into four sub-matrices. In contrast, the four-level system of U_2 is composed of $4 \cdot 4 = 16$ sub-matrices. Hence, the respective vertices have four and 16 successors, respectively. The x_1-vertices in Fig. 8.5a represent the



(a) QMDD for U_1

(b) QMDD for U_2

Fig. 8.5 QMDDs for the quantum operations sketched in Fig. 8.4

sub-matrices U_1^0 and U_1^3, respectively (as indicated in brackets). The x_1-vertex in Fig. 8.5b sketches the second-top-right sub-matrix. Its sub-blocks U_1^3 and $*$ are represented by distinct sets of edges (which are indicated by a correspondingly labelled $\boxed{M}$, but are not part of the original QMDD).

Due to efficient techniques like *shared vertices* and *unique tables*, QMDDs are capable of representing quantum functionality for several dozens of qubits and/or qudits. Moreover, computed tables enable a very efficient implementation of the equivalence checking scheme outlined above.

For the purpose of equivalence checking, the QMDD representations of the operations have to be aligned. More precisely, we

- align the number of don't-care states for corresponding qudits by "blowing up" vertices with additional successors (e.g. to introduce two additional don't-care states for each qubit, all vertices in Fig. 8.5a are equipped with 12 additional 0-edges),
- align basis states (if the mapping of shared states is non-trivial) by re-arranging edges appropriately, and
- align possibly different don't-care to don't-care mappings ($*$) by setting the corresponding edges to zero.

This transformation can be done in a single traversal of each QMDD and leads to representations of two matrices (of equal size), which are identical if, and only if, the operations are functionally equivalent. The latter can be verified in constant time by a single unique table look-up, since QMDDs provide canonical representations (cf. Sect. 4.3). By this, equivalence checking can be conducted efficiently even for large quantum systems. This has been confirmed by an experimental evaluation whose results are summarized and discussed in the next section.

8.3 Experimental Results

The equivalence checking scheme described above has been implemented on top of the QMDD package and evaluated on a wide range of operations realized in different quantum systems. More precisely, we considered

- 2-level and 4-level representations of various quantum operations including Shor's 9-qubit error-correcting code (denoted by *9qubitN1* and *9qubitN2*), as well as a 7-qubit encoding (denoted by *7qubitcode*) taken from [Mer07] and instances of Grover's algorithm (denoted by *Grover-k*) and Quantum Fourier Transforms (denoted by *QFT-k*) taken from [NC00] (*k* is the number of qubits),
- multi-qubit operations taken from RevLib [WGT+08], mainly realizing Boolean functionality for 2-level systems (denoted by their respective RevLib identifier), that additionally have been mapped to 4-level representations based on the methods described in [SWM12], and
- randomly generated quantum operations with up to 25 qubits (denoted *arbitrary*).

In total, 296 benchmarks have been considered. For each of them, the 2-level representation has been compared against the respective 4-level representation. In order to additionally evaluate the performance of the proposed approach for non-equivalent operations, for each pair of representations we introduced an error through random changes (to one of them) and compared this to the original operation. All experiments have been conducted on a 2.8 GHz Intel Core i7 machine with 8 GB of main memory running Linux. The timeout was set to 500 CPU seconds.

The results are summarized in Table 8.1 for a representative selection of the conducted experiments. The first two columns provide the identifiers of the respective benchmarks followed by its number of qudits. Afterwards, the run-time (in CPU seconds) for building up the data-structure (QMDD) as well as performing the actual equivalence check (EC) is provided for both cases, i.e. when both operations are equivalent and when they are not equivalent. As can be seen, the proposed scheme is able to efficiently check the equivalence of two quantum operations for the majority of all benchmarks. In fact, for 224 out of the 296 benchmarks, we were able to check their equivalence in less than a minute. While the actual equivalence check can always be conducted in almost no time, the limiting factor is the time needed for the construction of the representation of the respective quantum functionality, i.e. the QMDD in this case. Hence, the efficiency of the proposed scheme only relies on the chosen description. As improving those is an active research area and the proposed scheme can easily be adapted to other representations, further benefits can be expected in the future.

Table 8.1 Experimental evaluation

Benchmark	#Qudits	Run-times (s)			
		Equivalence		Non-equivalence	
		QMDD	EC	QMDD	EC
7qbitcode	7	<0.01	<0.01	<0.01	<0.01
9qubitN1	9	<0.01	<0.01	<0.01	<0.01
9qubitN2	17	0.04	<0.01	0.04	<0.01
Grover-5	11	0.41	<0.01	0.38	<0.01
Grover-6	13	0.04	<0.01	0.05	<0.01
QFT-5	5	<0.01	<0.01	0.01	<0.01
QFT-7	7	0.01	0.01	0.02	<0.01
add16_174	49	0.03	<0.01	0.02	<0.01
add32_183	97	0.08	<0.01	0.08	<0.01
alu2_199	16	117.84	0.01	115.94	0.02
alu3_200	18	224.42	0.04	217.3	0.04
apla_203	22	14.77	0.02	15.3	0.02
bw_291	87	>500	–	>500	–
cm163a_213	29	1.63	<0.01	1.74	0.03
cu_219	25	4.36	<0.01	4.59	0.02
cycle10_293	39	22.91	<0.01	25.29	<0.01
ham15_107	15	103.77	0.31	88.6	0.25
hwb7_61	7	3.24	<0.01	2.94	<0.01
lu_326	299	>500	–	>500	–
mod5add_306	32	326.98	0.4	307.95	0.36
arbitrary10	10	0.7	<0.01	0.73	<0.01
arbitrary15	15	15.04	0.2	25.41	0.55
arbitrary20	20	26.76	0.15	41.34	0.35
arbitrary25	25	>500	–	>500	–

8.4 Conclusions

In this chapter, we presented a scheme for checking the equivalence between two quantum operations working in quantum systems with potentially different numbers of levels. By this, the recent developments showing the advantages and benefits of multi-level quantum systems are taken into account and the correctness of the design can be verified even in these settings. The proposed scheme can be incorporated into data-structures particularly suited for the representation of quantum functionality. An experimental evaluation confirmed that this enabled an efficient and fast equivalence checking which is mainly limited by the representation of the considered quantum functionality.

Chapter 9
Discussion and Outlook

Quantum computations are changing the way how certain problems will be tackled in the near future. By exploiting quantum-mechanical phenomena such as superposition, phase shifts or entanglement, they allow for algorithms with asymptotic speed-ups for many relevant problems such as database search, integer factorization, and more. While only considered theoretically for a long time, the past decade also showed first physical realizations—with further to come. Hence, quantum computers with dozens of qubits seem to be a realistic vision for the future [MSB+11].

This more and more raises the question how the respective quantum circuits can be designed efficiently. In order to keep up with the technological progress, there is a need for significant improvements—particularly with respect to Computer-Aided Design of quantum circuits. More precisely, methods have to be developed that (1) automatically generate a circuit description of the desired quantum functionality (2) take into account the physical constraints of the target technology, and (3) scale to quantum systems of considerable size.

Generic approaches that aim for the synthesis of arbitrary quantum functionality are hardly applicable in practice as they are neither efficient nor effective. More precisely, they employ decomposition schemes that do not scale to large quantum systems and, even worse, lead to considerably large circuits and rely on a huge gate library that is not compatible with the needs of actual physical implementations. Consequently, in order to develop methods that meet the above requirements, the design of quantum circuits is not to be considered as a single design step, but as a separation of concerns. To this end, we presented extensions to the state-of-the-art in quantum logic design in various areas:

- We discussed the application of QMDDs in the synthesis of Boolean components. The latter constitute important parts in many quantum algorithms and applying a dedicated synthesis scheme promises to reduce the overall synthesis complexity. Often, the function to be realized is not reversible and needs to be embedded prior to synthesis. For both steps, determining a (minimal) embedding as well

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2_9

as performing the actual synthesis, QMDDs enable very powerful and scalable solutions.

- We presented an automatic synthesis scheme for Clifford group operations, an important sub-class of quantum operations. In contrast to generic synthesis approaches, the approach employs a dedicated, fault-tolerant gate library and leads to significantly smaller circuits.
- We presented a scheme for automatically checking whether two different realizations of (the same) quantum functionality are equivalent—an important tool for ensuring the correctness of the transformations and optimizations applied in the design flow. Taking into account recent developments in multi-level quantum systems, the scheme especially covers multiple-valued implementations.

These applications of QMDDs, especially the synthesis approach for Clifford group operations, illustrate nicely that compact and efficient representations of quantum functionality like QMDDs are the key to significantly improve the scalability of design solutions and additionally also offer inherent characteristics that are ready to be exploited for sophisticated applications in quantum logic CAD like e.g. specialized synthesis approaches.

Certainly, the full potential of QMDDs has not completely been utilized yet. The presented applications for synthesis and verification are just the beginning. Decision diagrams in general became a vital and indispensable tool for the design of conventional circuits and systems—with applications in synthesis, optimization, verification, test, and many more. In a similar fashion, QMDDs may be utilized in the near future. In this sense, this book may provide one basis towards this.

References

[AG04] S. Aaronson, D. Gottesman, Improved simulation of stabilizer circuits. Phys. Rev. A **70**, 052328 (2004)

[Ake78] S.B. Akers Jr., Binary decision diagrams. IEEE Trans. Comput. **27**(6), 509–516 (1978)

[AMMR13] M. Amy, D. Maslov, M. Mosca, M. Roetteler, A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. IEEE Trans. CAD **32**(6), 818–830 (2013)

[AP06] A. Abdollahi, M. Pedram, Analysis and synthesis of quantum circuits by using quantum decision diagrams, in *Design, Automation and Test in Europe* (2006), pp. 317–322

[ARD] ARDA. Quantum computation roadmap, http://qist.lanl.gov/qcomp_map.shtml

[BBC+93] C. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, W. Wootters, Teleporting an unknown quantum state by dual classical and EPR channels. Phys. Rev. Lett. **70**, 1895–1898 (1993)

[BBC+95] A. Barenco, C.H. Bennett, R. Cleve, D.P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J.A. Smolin, H. Weinfurter, Elementary gates for quantum computation. Phys. Rev. A **52**, 3457–3467 (1995)

[BCJ+00] H.J. Briegel, T. Calarco, D. Jaksch, J.I. Cirac, P. Zoller, Quantum computing with neutral atoms. J. Mod. Opt. **47**, 415–451 (2000)

[BMP+00] P.O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, F. Vatan, A new universal and fault-tolerant quantum basis. Inf. Process. Lett. **75**(3), 101–107 (2000)

[BN36] G. Birkhoff, J. Von Neumann, The logic of quantum mechanics. Ann. Math. **37**(4), 823–843 (1936)

[BOB05] S.S. Bullock, D.P. O'Leary, G.K. Brennen, Asymptotically optimal quantum circuits for d-level systems. Phys. Rev. Lett. **94**(23), 230502 (2005)

[BRW61] B. Bolt, T.G. Room, G.E. Wall, On the Clifford collineation, transform and similarity groups I. J. Aust. Math. Soc. **2**, 60–79 (1961)

[Bry86] R.E. Bryant, Graph-based algorithms for Boolean function manipulation. IEEE Trans. Comp. **35**(8), 677–691 (1986)

[BVMS05] V. Bergholm, J.J. Vartiainen, M. Mottonen, M.M. Salomaa, Quantum circuits with uniformly controlled one-qubit gates. Phys. Rev. A **71**, 052330 (2005)

[BW92] C. Bennett, S.J. Wiesner, Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states. Phys. Rev. Lett. **69**, 1992 (2881)

[CDNS11] A. Cabello, V. D'Ambrosio, E. Nagali, F. Sciarrino, Hybrid ququart-encoded quantum cryptography protected by Kochen-Specker contextuality. Phys. Rev. A **84**(3), 030302 (2011)

P. Niemann and R. Wille, *Compact Representations for the Design of Quantum Logic*, SpringerBriefs in Physics, DOI 10.1007/978-3-319-63724-2

[CZ95] J.I. Cirac, P. Zoller, Quantum computations with cold trapped ions. Phys. Rev. Lett. **74**(20), 4091–4094 (1995)

[DBE95] D. Deutsch, A. Barenco, A. Ekert, Universality in quantum computation. Proc. R. Soc. Lond. **449**(1937), 669–677 (1995)

[DBJ00] I. Deutsch, G. Brennen, P. Jessen, Quantum computing with neutral atoms in an optical lattice. Fortschritte der Physik [Progress of Physics] **48**, 925–943 (2000)

[Deu85] D. Deutsch, Quantum theory, the church-turing principle and the universal quantum computer. Proc. R. Soc. Lond. **400**(1818), 97–117 (1985)

[Deu89] D. Deutsch, Quantum computational networks. Proc. R. Soc. Lond. **425**(1868), 73–90 (1989)

[Dir39] P.A.M. Dirac, A new notation for quantum mechanics. Math. Proc. Cambridge Philos. Soc. **35**(3), 416–418 (1939)

[DN06] C.M. Dawson, M.A. Nielsen, The Solovay-Kitaev algorithm. Quantum Inf. Comput. **6**(1), 81–95 (2006)

[DW13] Y.-M. Di, H.-R. Wei, Synthesis of multivalued quantum logic circuits by elementary gates. Phys. Rev. A **87**, 012325 (2013)

[EFD05] R. Ebendt, G. Fey, R. Drechsler, *Advanced BDD optimization* (Springer, 2005)

[Ein05] A. Einstein, Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt. Annalen der Physik **322**, 132–148 (1905)

[FT11] D.Y. Feinstein, M.A. Thornton, On the skipped variables of Quantum multiple-valued decision diagrams, in *International Symposium on Multiple-Valued Logic*, 164–169, 2011

[FTM09] D.Y. Feinstein, M.A. Thornton, D.M. Miller, Minimization of quantum multiple-valued decision diagrams using data structure metrics. Multiple-Valued Log. Soft Comput. **15**(4), 361–377 (2009)

[FTR07] K. Fazel, M. Thornton, J. Rice, ESOP-based Toffoli gate cascade generation, in *Pacific Rim Conference on Communications, Computers and Signal Processing*, 2007, pp. 206–209

[GAJ06] P. Gupta, A. Agrawal, N.K. Jha, An algorithm for synthesis of reversible logic circuits. IEEE Trans. CAD **25**(11), 2317–2330 (2006)

[Gal07] A. Galiautdinov, Generation of high-fidelity controlled-NOT logic gates by coupled superconducting qubits. Phys. Rev. A **75**(5), 052303 (2007)

[GHZ89] D.M. Greenberger, M.A. Horne, A. Zeilinger, *Bell's Theorem, Quantum Theory, and Conceptions of the Universe* (Kluwer Academic Press, 1989)

[GMP+11] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan, K. Roy, Impact: imprecise adders for low-power approximate computing, in *International Symposium on Low Power Electronics and Design*, 409–414, 2011

[Got97] D. Gottesman, Stabilizer codes and quantum error correction. arXiv:9705052 (1997)

[Got98] D. Gottesman, The Heisenberg representation of quantum computers. arXiv:9807006 (1998)

[Gro96] L. K. Grover, A fast quantum mechanical algorithm for database search, in *Theory of computing*, pp. 212–219 (1996)

[GSG+04] A.D. Greentree, S. Schirmer, F. Green, L.C. Hollenberg, A. Hamilton, R. Clark, Maximizing the Hilbert space for a finite number of distinguishable quantum states. Phys. Rev. Lett. **92**(9), 097901 (2004)

[GTFM07] D. Goodman, M.A. Thornton, D.Y. Feinstein, D.M. Miller, Quantum logic circuit simulation based on the QMDD data structure. in *International Reed-Muller Workshop* (2007)

[GWDD09] D. Große, R. Wille, G.W. Dueck, R. Drechsler, Exact multiple control Toffoli network synthesis with SAT techniques. IEEE Trans. CAD **28**(5), 703–715 (2009)

[IA05] S.I. Inoue, Y. Aoyagi, Design and fabrication of two-dimensional photonic crystals with predetermined nonlinear optical properties. Phys. Rev. Lett. **94**, 103904 (2005)

[Jor16] S. Jordan, Quantum Algorithm Zoo (October 2016), http://math.nist.gov/quantum/zoo/

[KGRS03] A. Klimov, R. Guzman, J. Retamal, C. Saavedra, Qutrit quantum computer with trapped ions. Phys. Rev. A **67**(6), 062313 (2003)

[KLM01] E. Knil, R. LaFlamme, G.J. Milburn, A scheme for efficient quantum computation with linear optics. Nature **409**, 46–52 (2001)

[KMM12] V. Kliuchnikov, D. Maslov, M. Mosca, Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits, CoRR, abs/1212.0822 (2012)

[KR68] C.G. Khatri, C.R. Rao, Solutions to some functional equations and their applications to characterization of probability distributions. Sankhya **30**, 167–180 (1968)

[LBA+08] B.P. Lanyon, M. Barbieri, M.P. Almeida, T. Jennewein, T.C. Ralph, K.J. Resch, G.J. Pryde, J.L. O'Brien, A. Gilchrist, A.G. White, Simplifying quantum logic using higher-dimensional Hilbert spaces. Nat. Phys. **5**(2), 134–140 (2008)

[LCJ13] C. Lin, A. Chakrabarti, N.K. Jha, Optimized quantum gate library for various physical machine descriptions. IEEE Trans. Very Large Scale Integration (VLSI) Syst. **21**(11), 2055–2068 (2013)

[LCJ14a] C. Lin, A. Chakrabarti, N.K. Jha, FTQLS: Fault-tolerant quantum logic synthesis. IEEE Trans. Very Large Scale Integration (VLSI) Syst. **22**(6), 1350–1363 (2014)

[LCJ14b] C. Lin, A. Chakrabarti, N.K. Jha, QLib: Quantum module library. ACM J. Emerg. Technol. Comput. Syst.**11**(1), (2014)

[LJ14] C. Lin, N.K. Jha. RMDDS: Reed-Muller decision diagram synthesis of reversible logic circuits. ACM J. Emerg. Technol. Comput. Syst. **10**(2), (2014)

[LWK11] C. Lu, S. Wang, S. Kuo, An extended XQDD representation for multiple-valued quantum logic. IEEE Trans. Comput. **60**(10), 1377–1389 (2011)

[Mer07] N.D. Mermin, *Quantum Computer Science: An Introduction* (Cambridge University Press, 2007)

[MFT07] D.M. Miller, D.Y. Feinstein, M.A. Thornton, QMDD minimization using sifting for variable reordering. Multiple-Valued Log. Soft Comput. **13**(4–6), 537–552 (2007)

[MHT05] DMc Hugh, J. Twamley, Trapped-ion qutrit spin molecule quantum computer. N. J. Phys. **7**(1), 174 (2005)

[MMD03] D.M. Miller, D. Maslov, G.W. Dueck, A transformation based algorithm for reversible logic synthesis, in *Design Automation Conference*, 2003, pp. 318–323

[MMSK06] E. Moreva, G. Maslennikov, S. Straupe, S. Kulik, Realization of four-level qudits using biphotons. Phys. Rev. Lett. **97**(2), 023602 (2006)

[MS00] A. Muthukrishnan, C.R. Stroud Jr., Multi-valued logic gates for quantum computation. Phys. Rev. A **62**(5), 052309 (2000)

[MSB+11] T. Monz, P. Schindler, J.T. Barreiro, M. Chwalla, D. Nigg, W.A. Coish, M. Harlander, W. Hänsel, M. Hennrich, R. Blatt, 14-qubit entanglement: creation and coherence. Phys. Rev. Lett. **106**, 130506 (2011)

[MT06] D.M. Miller, M.A. Thornton, QMDD: A decision diagram structure for reversible and quantum circuits, in *International Symposium on Multiple-Valued Logic*, 6, 2006

[MT08] D.M. Miller, M.A. Thornton, *Multiple-Valued Logic: Concepts and Representations* (Morgan and Claypool, 2008)

[MTG06] D.M. Miller, M.A. Thornton, D. Goodman, A decision diagram package for reversible and quantum circuit simulation, in *IEEE World Congress on Computational Intelligence*, 2006, pp. 8597–8604

[MWS11] D.M. Miller, R. Wille, Z. Sasanian, Elementary quantum gate realizations for multiple-control Toffoli gates, in *International Symposium on Multiple-Valued Logic*, 288–293, 2011

[MYDM05] D. Maslov, C. Young, G.W. Dueck, D.M. Miller, Quantum circuit simplification using templates, in *Design, Automation and Test in Europe* (2005), pp. 1208–1213

[NAB+09] M. Neeley, M. Ansmann, R.C. Bialczak, M. Hofheinz, E. Lucero, A.D. O'Connell, D. Sank, H. Wang, J. Wenner, A.N. Cleland et al., Emulation of a quantum spin with a superconducting phase qudit. Science **325**(5941), 722–725 (2009)

[NC00] M. Nielsen, I. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, 2000)

[Neu32] J. Von Neumann, *Mathematische Grundlagen der Quantenmechanik* (Springer, 1932)

[NKS06] Y. Nakajima, Y. Kawano, H. Sekigawa, A new algorithm for producing quantum circuits using KAK decompositions. Quantum Inf. Comput. **6**(1), 67–80 (2006)

[NRS01] G. Nebe, E.M. Rains, N.J.A. Sloane, The invariants of the Clifford groups. Des. Codes Crypt. **24**(1), 99–122 (2001)

[NWD13] P. Niemann, R. Wille, R. Drechsler, On the "Q" in QMDDs: Efficient representation of quantum functionality in the QMDD data-structure, in *International Conference on Reversible Computation*, 2013, pp. 125–140

[NWD14a] P. Niemann, R. Wille, R. Drechsler, Efficient synthesis of quantum circuits implementing Clifford group operations, in *ASP Design Automation Conference*, 2014, pp. 483–488

[NWD14b] P. Niemann, R. Wille, R. Drechsler, Equivalence checking in multi-level quantum systems, in *International Conference on Reversible Computation*, 2014, pp. 201–215

[NWM+16] P. Niemann, R. Wille, D.M. Miller, M.A. Thornton, R. Drechsler, QMDDs: Efficient quantum function representation and manipulation. IEEE Trans. CAD **35**(1), 86–99 (2016)

[NZWD17] P. Niemann, A. Zulehner, R. Wille, R. Drechsler, Efficient construction of QMDDs for irreversible, reversible, and quantum functions, in *International Conference on Reversible Computation*, 2017, page in publication

[OFV09] J.L. O'Brien, A. Furusawa, J. Vučković, Photonic quantum technologies. Nat. Photonics **3**(12), 687–695 (2009)

[Pla00] M. Planck, Über eine Verbesserung der Wienschen Spektralgleichung. Verhandlungen der Deutschen Physikalischen Gesellschaft **2**, 202–204 (1900)

[PW94] C. Paige, M. Wei, History and generality of the CS decomposition. Linear Algebra Appl. **208**, 303–326 (1994)

[Rud93] R. Rudell, Dynamic variable ordering for ordered binary decision diagrams, in *International Conference on CAD*, 1993, pp. 42–47

[Sas12] Z. Sasanian, Technology mapping and optimization for reversible and quantum circuits. PhD thesis, University of Victoria, Canada (2012)

[SAZS11] M. Saeedi, M. Arabzadeh, M.S. Zamani, M. Sedighi, Block-based quantum-logic synthesis. Quantum Inf. Comput. **11**(3&4), 262–277 (2011)

[SBM06] V.V. Shende, S.S. Bullock, I.L. Markov, Synthesis of quantum-logic circuits. IEEE Trans. CAD **25**(6), 1000–1010 (2006)

[Sch95] B. Schumacher, Quantum coding. Phys. Rev. A **51**, 2738–2747 (1995)

[Sho94] P.W. Shor, Algorithms for quantum computation: discrete logarithms and factoring, in *Foundations of Computer Science*, 1994, pp. 124–134

[SJD+03] F.W. Strauch, P.R. Johnson, A.J. Dragt, C.J. Lobb, J.R. Anderson, F.C. Wellstood, Quantum logic gates for coupled superconducting phase qubits. Phys. Rev. Lett. **91**, 167005 (2003)

[Som01] F. Somenzi, Efficient manipulation of decision diagrams. Softw. Tools Technol. Transfer **3**(2), 171–181 (2001)

[SPMH03] V. Shende, A. Prasad, I. Markov, J. Hayes, Synthesis of reversible logic circuits. IEEE Trans. CAD **22**(6), 710–722 (2003)

[SWH+12] M. Soeken, R. Wille, C. Hilken, N. Przigoda, R. Drechsler, Synthesis of reversible circuits with minimal lines for large functions, in *ASP Design Automation Conference*, 2012, pp. 85–92

[SWK+15] M. Soeken, R. Wille, O. Keszocze, D.M. Miller, R. Drechsler, Embedding of large Boolean functions for reversible logic. J. Emerg. Technol. Comput. Syst. **12**(4), 41:1–41:26 (2015)

[SWM12] Z. Sasanian, R. Wille, D.M. Miller, Realizing reversible circuits using a new class of quantum gates, in *Design Automation Conference*, 2012, pp. 36–41

[SZSS10] M. Saeedi, M.S. Zamani, M. Sedighi, Z. Sasanian, Synthesis of reversible circuit using cycle-based approach. J. Emerg. Technol. Comput. Syst. **6**(4), (2010)

[TPJ+07] J.M. Taylor, J.R. Petta, A.C. Johnson, A. Yacoby, C.M. Marcus, M.D. Lukin, Relaxation, dephasing, and quantum control of electron spins in double quantum dots. Phys. Rev. B **76**, 035315 (2007)

[VMH04] G. Viamontes, I. Markov, J. Hayes, High-performance QuIDD-based simulation of quantum circuits, in *Design, Automation and Test in Europe*, 2004, pp. 1354–1355

[VMH07] G.F. Viamontes, I.L. Markov, J.P. Hayes, Checking equivalence of quantum circuits and states, in *International Conference on CAD*, 2007, pp. 69–74

[VMH09] G.F. Viamontes, I.L. Markov, J.P. Hayes, *Quantum Circuit Simulation* (Springer, 2009)

[VMS04] J.J. Vartiainen, M. Mottonen, M.M. Salomaa, Efficient decomposition of quantum gates. Phys. Rev. Lett. **92**(177902) (2004)

[VRMH03] G.F. Viamontes, M. Rajagopalan, I.L. Markov, J.P. Hayes, Gate-level simulation of quantum circuits, in *ASP Design Automation Conference*, 2003, pp. 295–301

[VSB+01] L.M.K. Vandersypen, M. Steffen, G. Breyta, C.S. Yannoni, M.H. Sherwood, I.L. Chuang, Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature **414**, 883–887 (2001)

[WD09] R. Wille, R. Drechsler, BDD-based synthesis of reversible logic for large functions. in *Design Automation Conference*, 2009, pp. 270–275

[WGMD09] R. Wille, D. Große, D.M. Miller, R. Drechsler, Equivalence checking of reversible circuits. in *International Symposium on Multiple-Valued Logic*, 324–330 2009

[WGT+08] R. Wille, D. Große, L. Teuber, G.W. Dueck, R. Drechsler. RevLib: an online resource for reversible functions and reversible circuits. in *International Symposium on Multiple-Valued Logic*, 220–225 2008. RevLib is available at http://www.revlib.org

[WKD11] R. Wille, O. Keszöcze, R. Drechsler, Determining the minimal number of lines for large reversible circuits, in *Design, Automation and Test in Europe*, 2011, pp. 1204–1207

[WLTK08] S.-A. Wang, C.-Y. Lu, I.-M. Tsai, S.-Y. Kuo, An XQDD-based verification method for quantum circuits. IEICE Trans. **91–A**(2), 584–594 (2008)

[WSOD13] R. Wille, M. Soeken, C. Otterstedt, R. Drechsler. Improving the mapping of reversible circuits to quantum circuits using multiple target lines, in *ASP Design Automation Conference*, 2013, pp. 145–150

[YM10] S. Yamashita, I.L. Markov, Fast equivalence-checking for quantum circuits. Quantum Inf. Comput. **10**(9&10), 721–734 (2010)

[ZW17] A. Zulehner, R. Wille, Make it reversible: efficient embedding of non-reversible functions, in *Design, Automation and Test in Europe*, 2017, pp. 458–463

[ZYC02] X. Zhang, Z. Yang, C. Cao, Inequalities involving Khatri-Rao products of positive semi-definite matrices. Appl. Math. E-notes **2**, 117–124 (2002)