Fakultät für Informatik und Mathematik
Universität Passau

Design Methods for Reliable Quantum Circuits

Author: Alexandru Paler
Thesis Advisor: Professor Dr. Ilia Polian

October, 2014

Thesis advisor: Professor Dr. Ilia Polian Alexandru Paler

Abstract

Quantum computing is an emerging technology that has the potential to change the perspectives and applications of computing in general. A wide range of applications are enabled: from faster algorithmic solutions of classically still difficult problems to theoretically more secure communication protocols. A quantum computer uses the quantum mechanical effects of particles or particle-like systems, and a major similarity between quantum and classical computers consists of both being abstracted as information processing machines. Whereas a classical computer operates on classical digital information, the quantum computer processes quantum information, which shares similarities with analog signals. One of the central differences between the two types of information is that classical information is more fault-tolerant when compared to its quantum counterpart.

Faults are the result of the quantum systems being interfered by external noise, but during the last decades quantum error correction codes (QECC) were proposed as methods to reduce the effect of noise. Reliable quantum circuits are the result of designing circuits that operate directly on encoded quantum information, but the circuit's reliability is also increased by supplemental redundancies, such as sub-circuit repetitions.

Reliable quantum circuits have not been widely used, and one of the major obstacles is their vast associated resource overhead, but recent quantum computing architectures show promising scalabilities. Consequently the number of particles used for computing can be more easily increased, and that the classical control hardware (inherent for quantum computation) is also more reliable. Reliable quantum circuits haev been investigated for almost as long as general quantum computing, but their limited adoption (until recently) has not generated enough interest into their systematic design.

The continuously increasing practical relevance of reliability motivates the present thesis to investigate some of the first answers to questions related to the background and the methods forming a reliable quantum circuit design stack.

The specifics of quantum circuits are analysed from two perspectives: their probabilistic behaviour and their topological properties when a particular class of QECCs are used. The quantum phenomena, such as entanglement and superposition, are the computational resources used for designing quantum circuits. The discrete nature of classical information is missing for quantum information. An arbitrary quantum system can be in an infinite number of states, which are linear combinations of an exponential number of basis states. Any nontrivial linear combination of more than one basis states is called a state superposition. The effect of superpositions becomes evident when the state of the system is inferred (measured), as measurements are probabilistic with respect to their output: a nontrivial state superposition will collapse to one of the component basis states, and the measurement result is known exactly only after the measurement. A quantum system is, in general, composed from identical subsystems, meaning that a quantum computer (the complete system) operates on multiple similar particles (subsystems). Entanglement expresses the impossibility of separating the state of the subsystems from the state of the complete system: the nontrivial interactions between the subsystems result into a single indivisible state. Entanglement is an additional source of probabilistic behaviour: by measuring the state of a subsystem, the states of the unmeasured subsystems will probabilistically collapse to states from a well defined set of possible states. Superposition and entanglement are the building blocks of quantum information teleportation protocols, which in turn are used in state-of-the-art fault-tolerant quantum computing architectures. Information teleportation implies that the state of a subsystem is moved to a second subsystem without copying any information during the process.

The probabilistic approach towards the design of quantum circuits is initiated by the extension of classical test and diagnosis methods. Quantum circuits are modelled similarly to classical circuits by defining gate-lists,

and missing quantum gates are modelled by the single missing gate fault. The probabilistic approaches towards quantum circuits are facilitated by comparing these to stochastic circuits, which are a particular type of classical digital circuits. Stochastic circuits can be considered an emulation of analogue computing using digital components. A first proposed design method, based on the direct comparison, is the simulation of quantum circuits using stochastic circuits by mapping each quantum gate to a stochastic computing sub-circuit. The resulting stochastic circuit is compiled and simulated on FPGAs. The obtained results are encouraging and illustrate the capabilities of the proposed simulation technique. However, the exponential number of possible quantum basis states was translated into an exponential number of stochastic computing elements.

A second contribution of the thesis is the proposal of test and diagnosis methods for both stochastic and quantum circuits. Existing verification (tomographic) methods of quantum circuits were targeting the reconstruction of the gate-lists. The repeated execution of the quantum circuit was followed by different but specific measurement at the circuit outputs. The similarities between stochastic and quantum circuits motivated the proposal of test and diagnosis methods that use a restricted set of measurement types, which minimise the number of circuit executions. The obtained simulation results show that the proposed validation methods improve the feasibility of quantum circuit tomography for small and medium size circuits.

A third contribution of the thesis is the algorithmic formalisation of a problem encountered in teleportation-based quantum computing architectures. The teleportation results are probabilistic and require corrections represented as quantum gates from a particular set. However, there are known commutation properties of these gates with the gates used in the circuit. The corrections are not applied as dynamic gate insertions (during the circuit's execution) into the gate-lists, but their effect is tracked through the circuit, and the corrections are applied only at circuit outputs. The simulation results show that the algorithmic solution is applicable for very large quantum circuits.

Topological quantum computing (TQC) is based on a class of fault-tolerant quantum circuits that use the surface code as the underlying QECC. Quantum information is encoded in lattice-like structures and error protection is enabled by the topological properties of the lattice. The 3D structure of the lattice allows TQC computations to be visualised similarly to knot diagrams. Logical information is abstracted as strands and strand interactions (braids) represent logical quantum gates. Therefore, TQC circuits are abstracted using a geometrical description, which allows circuit input-output transformations (correlations) to be represented as geometric sub-structures.

TQC design methods were not investigated prior to this work, and the thesis introduces the topological computational model by first analysing the necessary concepts. The proposed TQC design stack follows a top-down approach: an arbitrary quantum circuit is decomposed into the TQC supported gate set; the resulting circuit is mapped to a lattice of appropriate dimensions; relevant resulting topological properties are extracted and expressed using graphs and Boolean formulas. Both circuit representations are novel and applicable to TQC circuit synthesis and validation. Moreover, the Boolean formalism is broadened into a formal mechanism for proving circuit correctness.

The thesis introduces TQC circuit synthesis, which is based on a novel logical gate geometric description, whose formal correctness is demonstrated. Two synthesis methods are designed, and both use a general planar representation of the circuit. Initial simulation results demonstrate the practicality and performance of the methods.

An additional group of proposed design methods solves the problem of automatic correlation construction. The methods use validity criteria which were introduced and analysed beforehand in the thesis. Input-output correlations existing in the circuit are inferred using both the graph and the Boolean representation.

The thesis extends the TQC state-of-the-art by recognising the importance of correlations in the validation process: correlation construction is used as a sub-routine for TQC circuit validation. The presented cross-layer validation procedure is useful when investigating both the QECC and the circuit, while a second proposed method is QECC-independent. Both methods are scalable and applicable even to very large circuits.

The thesis completes with the analysis of TQC circuit identities, where the developed Boolean formalism is used. The proofs of former known circuit identities were either missing or complex, and the presented ap-

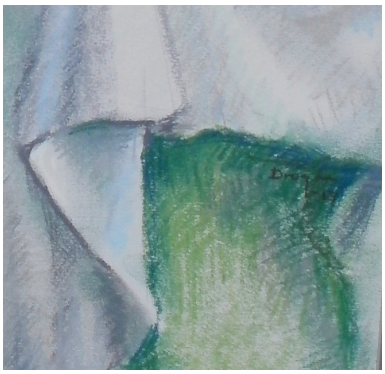
proach reduces the length of the proofs and represents a first step towards standardising them. A new identity is developed and detailed during the process of illustrating the known circuit identities.

Reliable quantum circuits are a necessity for quantum computing to become reality, and specialised design methods are required to support the quest for scalable quantum computers. This thesis used a twofold approach towards this target: firstly by focusing on the probabilistic behaviour of quantum circuits, and secondly by considering the requirements of a promising quantum computing architecture, namely TQC. Both approaches resulted in a set of design methods enabling the investigation of reliable quantum circuits.

The thesis contributes with the proposal of a new quantum simulation technique, novel and practical test and diagnosis methods for general quantum circuits, the proposal of the TQC design stack and the set of design methods that form the stack. The mapping, synthesis and validation of TQC circuits were developed and evaluated based on a novel and promising formalism that enabled checking circuit correctness.

Future work will focus on improving the understanding of TQC circuit identities as it is hoped that these are the key for circuit compaction and optimisation. Improvements to the stochastic circuit simulation technique have the potential of spawning new insights about quantum circuits in general.

Contents



1 Introduction

The discovery of quantum mechanics introduced new perspectives about reality, which at the beginning were difficult to accept. The major obstacles constituted the counter-intuitive effects possible to be described by the new theory, but the predictions of quantum mechanics were supported by numerous experiments. This led to new questions, and the most celebrated was formulated by Feynman [Fey82], asking if it would be possible to simulate quantum mechanical effects using a quantum computer.

The first quantum algorithm was formulated by Deutsch [Deu85,DJ92], and this development lead to a quick discovery of further algorithms and ways to use the quantum circuit formalism [BBC+95] for solving computational problems. One of the turning points was the discovery of Shor's algorithm, which showed that quantum computers enable fast integer factorisation. The efforts of building a quantum computer were thus increased leading to a new set of scientific questions being asked. What are feasible quantum computing architectures, and are there ways to compensate the environmental action (noise) on the computation by performing error checking and correction?

Decoherence is the loss of the quantum mechanical properties of a system due to the effect of quantum noise. An architecture describes the computing system by specifying its parts and relations, and a key concept in the architectural design is the decoherence time: the time for which the computer remains quantum-mechanically coherent. The feasibility of an architecture is dictated by the length of the longest possible quantum computation given by the ratio between the decoherence time and the time it takes to perform elementary computations (e.g. gates).

In a quantum computer, as in a classical analogue computer, small errors accumulate

over time and eventually add up to large errors, and it is difficult to find methods that can prevent or correct such small errors[Pre98b]. The solution was to *fight entanglement with entanglement*. In short, quantum entanglement means that multiple quantum systems are linked together in a way such their state is indivisible. The major source of errors during a computation is generated by the computing system being entangled with its environment, such that the system becomes susceptible to external influence and decoheres. Environmental interactions are reduced by including redundancies into the system, where homogeneous subsystems are entangled such that the resulting system has the same properties as its constituents. The first quantum error checking and correcting (QECC) protocols were extensions of classical error correction methods like the Hamming code[Got09,Pre98a,DMN13]. The initial developments sparked the interest for QECCs, and multiple classes of such codes were formulated. The most popular and practical classes were proposed by Kitaev[Pre98b] and use topological properties to encode and protect information.

The theoretical results gathered through the investigation of QECCs raised the attraction of quantum computing once more, and possible hardware architectures where proposed, for example in[JVMF+12]. The evolution of quantum architectures can be compared to the beginnings of the digital computer: various proof-of-concept experimental setups were presented, but few were shown to be sufficiently scalable to support quantum computing in the near future. A key aspect that will enable the adoption of any given architecture is its support for scalable quantum error correction methods[DFS+09].

Technological scalability and elasticity are two complementary notions. The search for scalable quantum systems stems from the aparent empirical utility witnessed this age by the increased computing and information storage capabilities (see Moore's Law[BC11]). Scalability is, in general, associated to the positive effect of increase, but *elasticity* would be a more fitting criterion for future quantum computers. Elasticity is one of the main concepts that allowed the definition of present cloud services, like distributed virtual networks[PFdM10]. An elastic technology has a utility margin that is not strongly fluctuating in the long term, and such a technology responds better when confronted with oscillating demand. When discussing scalable quantum computing the accent will fall on the capability of handling both small and large computational problems, whose solutions require either a small or large amount of error-correction.

Quantum computing attracted the attention of computer engineers that translated the concept of reversible computation into the field of classical Boolean circuits by using initial developments by Toffoli[FHA98]. Logical reversibility, first formulated by Landauer[FHA98], refers to the property of a computation being reversed, meaning that the input of an algorithm can be easily backtracked starting from the output. Classical reversible computation offered again a new set of questions to be answered. What circuit design methods are meaningful in the reversible context? How are reversible circuits tested and diagnosed? What cost measures are meaningful when optimising reversible circuits?

This work proposes efficient design methods that reduce the complexity of practical is-

sues in reliable quantum circuits. The methods are formulated in a period where the future of quantum computing is uncertain. The reasons range from the usefulness of quantum computing, intersect socio-economic aspects, and include the skeptisism towards disruptive technologies [Mor13]. The engineering feasibility of quantum architectures is also questioned. As a result, the design methods have to be architecture-agnostic to some extent, and the insights gained during their development should have a consequence outside their initial field.

1.1 Reliable quantum circuits

The quantum circuit formalism, developed after the publication of Deutsch's algorithm [Deu85], is a representation similar to classical circuit diagrams. The term quantum circuit is not unanimously accepted, and, for example, Deutsch disagrees with it and insists that it should be called quantum computational network because the word "circuit" implies a closed path*. However, this work will refer to quantum circuits, which is a term universally adopted in the computer engineering community. Quantum circuits are understood as a quantum analog of classical circuits, consisting of quantum gates and quantum bits (qubits). A quantum circuit is the abstraction of a quantum computation executed on a quantum system, and various implementations of a quantum computer are abstracting qubits using the spin of elementary particles (e.g. photons).

The reliability of a quantum circuit is generally analysed in an architecture independent manner, by assuming qubit error rates or other modeled parameters that negatively influence the circuit's execution. In general, system reliability was extensively investigated, and according to the taxonomy from [ALRL04] reliability is an attribute that refers to the continuity of correct service. For quantum computers reliability is a result of efficient fault-tolerance, where, according to the same taxonomy, a fault is the adjudged or hypothesised cause of an error (service deviation). For the purpose of this work, the term "service" will be understood as computation.

Reliable quantum circuits are the result of active and effective fault-tolerance (avoid service failures in the presence of faults [ALRL04]). Throughout this work Preskill's *laws of fault-tolerant computation* [Pre98b] are used: 1) do not use the same qubit twice; 2) copy the errors, not the data; 3) verify before encoding; 4) repeat operations; 5) use the right error correcting code. The first law introduces the redundancies necessary to reduce the propagation of errors, while the second law sets the focus on the errors and not on the encoded information. Ideally, quantum gates should directly operate on encoded data. The third and fourth law say that each computational step should be checked for correctness, and that repeating a step reduces the associated probability of failure. Finally, the fifth law requires the wise choice of the error-correcting method: some codes may be suited for a given application, while others may not. For example, codes with large resource overheads and good performance may not be useful for large non-critical portions of a computation.

*http://www.daviddeutsch.org.uk/2012/07/they-arent-quantum-circuits/

Reliable quantum circuits are the starting point for scalable quantum computing, and according to DiVincenzo[D+00], there are five requirements for the implementation of (scalable) quantum computation: 1) a scalable physical system with well characterised qubits; 2) the ability to initialise the state of the qubits to a simple fiducial state; 3) long relevant decoherence times, much longer than the gate operation time; 4) a universal set of quantum gates; 5) a qubit-specific measurement capability. Well-characterised qubits are represented by quantum systems whose physical parameters are accurately known. The second requirement arises from the straightforward computing requirement that registers should be initialized to a known value before the start of computation[D+00]. The second reason for this requirement is the first law of fault-tolerance, which requires a continuous supply of qubits. Decoherence times characterise the dynamics of a qubit in contact with its environment, and the third criterion ensures that computations are faster than the negative environmental influence. The fourth criterion asks for a quantum computer to support arbitrary quantum computations using a discrete set of gates, similarly to how NAND gates are used for expressing all Boolean functions. The final criterion states that it is possible to reliably measure the output of a computation.

This work is motivated by the reduced size of the apparatus regarding the automatic design of reliable quantum circuits. Design automation is being considered by the *reversible computing commmunity*[WD10,SM13] which investigates Boolean circuits composed of gates from the quantum context, but not supporting quantum specific phenomena. The methods are not capturing the complete complexity of quantum computations, and are not suited for general quantum circuits. On the other hand, in the physics community the pragmatic aspects of design automation have not been considered until recently.

Quantum circuit design methods were initially proposed as extensions of the methods used in classical circuit design. However, the properties of quantum mechanical systems are not intuitive and do not have classical counterparts, such that their exploitation is difficult. Hence, the extended classical methods do not always deliver the results hoped for. Nevertheless, first specialised methods were formulated from both a top-down and a bottom-up perspective. The top-down approach starts from the quantum algorithm analysis and tries to define appropriate quantum architectures, whereas the bottom-up approach considers technologically feasible quantum architectures and maps the algorithmic formulation to such architectures. The design of reliable quantum circuits is often architecture-specific or algorithm-specific, and a unified methodology similar to the one existing for classical circuits was not formulated.

1.2 Design flow

The explosive growth of VLSI (very large scale integration, the process of integrating large numbers of transistors on a single chip) was enabled by both the continuous shrinking of transistor sizes, but also by the refinement and better understanding of the necessary design

methods. The number of transistors that can be physically implemented on a chip grows faster than the ability to design the chips (a phenomenon known as the *design gap*) [WD10], and computer engineers and scientists are still not able to fully exploit the technical state of the art. The ability to verify the correctness of the designed circuits is also negatively affected (*verification gap*) [WD10]. The situation in the far more mature VLSI domain is not yet a reality for quantum circuits, but it is useful to conceive methods required in a probable future.

The frame of reference in this work is the circuit specification, which will be used to guide the decisions taken to propose the design methods. The relation between the specification of a circuit and a particular instance of the implemented specification can be noted by looking at the design flow of circuits.

Circuit design is a process that starts from a specification and ends at various concrete implementations [Lam08], and in between a series of steps is performed and reiterated when needed. Although different, the implementations are expected to be correct, and a major difference between implementations consists in the associated costs. For reversible and quantum circuits some examples are the number of qubits (wires), the number of gates and the nearest neighbour cost, which denotes the effort needed to make an arbitrary circuit consist only of gates operating on neighbouring qubits [WD10]. The choice of a particular implementation depends on the cost metric mix that fits the requirements.

The design process consists of two components [WG98]: the topology of the circuit (refers to the gate elements and the connections between them), and the gate set from which the circuit has to be constructed. Some gate sets supported by an architecture could be very difficult to implement in another, and the gate set is a factor influencing the choice of the architecture.

The different implementations need to be tested for correctness, and design verification is the reverse process of design: it starts with an implementation and confirms that the implementation meets its specification [Lam08]. The whole framework of testing the implementation correctness hypothesis requires that design verification makes a prediction that can be checked by observation (circuit is working indeed). For the observation to function some auxiliary assumptions are needed, but these introduce idealisations [Sob99].

Throughout this work the main idealisation is that the verification procedures are implemented correctly in software. Therefore, there are two types of errrors that could occur during verification: *implementation errors* and *software bugs*, but only the ones in the implementation are sought after.

There is a distinction to be made between *verification* and *validation*. According to [Ins08], validation is the technique of evaluating a product during or at the end of a project to ensure it complies with the specification [Ins08], while verification assures that the product satisfies certain imposed conditions. Considering circuit implementations as products, it is entirely possible that an implementation passes verification but fails validation.

Returning to the design flow, the first step is the synthesis of a circuit, where the specification (or a higher level description of the requirements) is transformed into an initial

circuit implementation. Afterwards, the circuit is optimised with regard to the targeted cost measures, but optimisation (manual or automatic) could introduce errors in the output implementation. A successful verification and debug phase is followed by an optional circuit optimisation phase, which requires a supplemental verification and debugging phase [WD10]. Finally, the circuit is either re-synthesised or simply declared as the final result of the design process.

There are two options for performing verification: by simulation or by formal methods. Formal verification investigates a design property which is part of the specification, and the checking is performed by abstracting the circuit behaviour using a canonical representation. This is a complex task, for there may exist unanticipated dependencies between design details and the implementation [Dil98]. However, a *canonical representation* will have the characteristic property that two functions are equivalent if their respective representations are isomorphic [Lam08].

Verification by simulation is often targeted at the hardware layer of the specification by inspecting design properties that can be inferred from the circuit execution. During simulation-based verification an *implementation under test* is checked for the existing relations between the inputs and the outputs. For classical Boolean circuits, it is similar to (partial) truth table inspection. Nevertheless, a complete verification by simulation is not computationally feasible for non-trivial instances: there is an exponential number of input stimuli (test vectors) required to exhaustively simulate the entire design [Lin08]. In spite of this fact, simulations can be a successful tool.

The increased design complexity (see design gap) reduces the applicability of simulation-based verification but also increases the difficulty of formal verification. As a result, a spectrum of intermediary semi-formal verification methods was proposed [Dil98]. A first option is *conventional simulation*. This can be extended to include the analysis of *coverage*, which is a measure that describes the degree to which the design has been verified. *Smart simulation* would generate test vectors (offline or online) based on a coverage metric test vector generation. Another option is *wide simulation* [Dil98], by symbolically representing large sets of states in relatively few simulations (e.g. binary decision diagrams). Wide simulation (also called symbolic simulation) would use logical expressions which have operators on higher-level data types in order to abstract away datapath elements. Finally, prioritised model checking [Dil98] explores state space partitions through heuristics that try to find errors early in search.

1.3 Problem statement

Design methods for error-corrected quantum computing architectures have not been until recently extensively investigated, and the problems of circuit validation, circuit equivalence and circuit synthesis have been proposed only in a formalism similar to the classical circuit (network of gates). The efforts of the research community have focused on circuit depth re-

duction algorithms or reduction of the required number of qubits (the quantum equivalent of classical bits) [WD10]. Moreover, in the quantum circuit formalism, error correcting methods are embedded into the computation as subcircuits, but another paradigm of reliable quantum computing was proposed by considering topological properties of particle interactions. The unreliability of quantum states and gates in the presence of environmental noise was shown to be reduced if particle-like structures called *anyons* [Col06] are used. A topological quantum computer would operate differently from a classical one. The model abstracts particle paths through strings, and the calculations are based on braided strings. The resulting braid pattern dictates the implemented computation. In this computational model the knot theory [Ada94] finds additional applications.

The solutions proposed in this thesis result from a twofold methodology. A first path of investigation states the problem in the context of probabilistic computing methods. The observed probabilistic behaviour of quantum circuits motivates the quest for alternate quantum circuit simulation techniques. Furthermore, existing reversible circuit design methods have been formulated from the computer engineering perspective, and it is interesting to transcend the scientific domains barrier and to ask which circuit fault models and verification techniques are applicable, and can be specialised for probabilistic circuits, in general, and for quantum circuits, in particular. The interdisciplinary approach results in a novel quantum circuit verification technique that significantly extends the one of classical circuits, as well as a new quantum circuit simulation method.

The second path of investigation in this thesis treats the design method problematic of topological quantum circuits. The proposed solutions were constructed starting from the topological error correcting codes, a simulated model inspired by anyonic interactions, which still have to be experimentally demonstrated. Moreover, this work is the first to introduce a formalisation of topological circuit equivalence from the perspective of computer engineering. The solution is based on a notation, which is simultaneously used to lay the foundations of topological circuit synthesis and verification.

1.4 Outline

The thesis is structured as follows. The second chapter introduces the necessary background. The quantum circuit representation is paralleled by the quantum graph-states, which lay the foundations of the briefly presented measurement based quantum computing model. The importance of error correction is acknowledged by describing its central notions, and finally the chapter presents an example of a topological error correcting code.

The third chapter is focused on placing quantum circuits into the probabilistic circuits frame. Furthermore, stochastic circuits operating with probabilities [Gai69] are presented as an alternative to classical circuits. The analogy is extended to include quantum circuits. The probabilistic perspective is used for verification purposes after presuming a practical and applicable quantum gate fault model. The first presented design method is a simulation

framework of quantum circuits using stochastic computing. Afterwards, the simulation-based validation of probabilistic circuits is introduced by starting from a set of simplified assumptions. The verification method is extended during the second step, and quantum specific phenomena are included into the fault model. The results indicate an improved verification that has a reduced computational complexity.

Chapter 4 introduces the subject of topological quantum computing, where a three-dimensional topological code is used in a measurement based computing model. The chapter solves the problem of mapping an abstract computational description on hardware elements that are still architecturaly agnostic. The findings are theoretical on their own and will be used as foundations of the constructive procedures described in the following chapter. The design methods in this chapter are examined through simulations, and their correctness is formally shown.

The fifth chapter investigates the properties of computations mapped to the topological code. The properties are extracted into the original notation, developed in the previous chapter, that enables a formal verification of the circuits. Equivalence checking, validation and synthesis of the topological circuits use the notation, too. The low computational complexity, mirrored in the simulation results, indicate the practicality of the approaches.

Chapter 6 concludes the present work and offers perspectives on future research.

2 Background

Quantum computation and quantum information are the study of the information processing tasks that can be accomplished using quantum mechanical systems[NC10]. Quantum computation is an application of quantum mechanics, where the evolution of a quantum system is described by a quantum algorithm. This observation is supported by the authors of[NC10], where they express their hope that quantum computing will be used as an introduction to quantum mechanics because "quantum computation and quantum information offer an excellent conceptual laboratory for understanding the basic concepts and unique aspects of quantum mechanics".

2.1 Quantum bits, gates and circuits

The quantum computing paradigm will be gradually presented and, where possible, analogies to classical computing will be made. For example the classical bit, the classical gates and the classical circuits have quantum analogon counterparts. Quantum circuits consist of quantum gates operating on quantum bits (qubits). Although physical systems able to represent quantum information are called qubits, in general the term will denote the mathematical object, as this is the case in the present chapter[NC10].

2.1.1 Postulates of quantum mechanics

The foundations of quantum mechanics are postulated, and the postulates are useful for introducing quantum computing using physical quantum systems.

State space

Associated to any *isolated physical system* is a complex vector space with inner product known as the *state space*. The system is completely described by its *state vector*, which is a unit vector in the system's state space[NC10].

The simplest quantum mechanical system is the qubit, having a two-dimensional state space, and the state vector describing a qubit is, in general, denoted by $|\psi\rangle$. The qubit is the quantum analog of the classical bit, but while a bit is either 0 or 1, the state of a qubit is as a linear combination of the $|0\rangle$ and $|1\rangle$ states. The $|0\rangle$ and $|1\rangle$ states are called *computational basis states*.

The *normalisation condition*, that $|\psi\rangle$ be a unit vector, $\langle\psi|\psi\rangle = 1$ is for a qubit equivalent to $|a|^2 + |b|^2 = 1$; as a result, the state is written as in Equation 2.2. The *bra-ket* notation is used as the standard quantum mechanical notation, and the properties of the notation are summarized in Table 2.1.

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle\,; \qquad |0\rangle = (1,0)^T; \qquad |1\rangle = (0,1)^T \tag{2.1}$$

$$|\psi\rangle = e^{i\gamma}(\cos\frac{\theta}{2}\,|0\rangle + e^{i\phi}\sin\frac{\theta}{2}\,|1\rangle), \text{ with } \theta,\phi,\gamma \in \mathbb{R} \tag{2.2}$$

The *phase* of a state is either global (γ) or relative (ϕ).

The vector space $\mathbb{C}^2$ can be spanned by the set of vectors $\{|0\rangle, |1\rangle\}$, but other spanning sets also exist. For example $\{|+\rangle, |-\rangle\}$ are used to express $|\psi\rangle$.

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}}; \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

$$|\psi\rangle = \frac{a+b}{\sqrt{2}}\,|+\rangle + \frac{a-b}{\sqrt{2}}\,|-\rangle\,; |+\rangle = (1,1)^T; |-\rangle = (1,-1)^T$$

In general, the parameters θ, ϕ, γ allow one to visualise the state of a qubit on the Bloch sphere (see Figure 2.1a), and the complex numbers a, b are called *amplitudes* of the states.

The main difference to classical bits is that qubits can be in a *superposition* of states. This is enabled by the states being linear combinations of the basis vectors, meaning that for a given basis vector set, more then one amplitude is different from zero. For example the state $|-\rangle$ is a superposition of $|0\rangle$ (amplitude $1/\sqrt{2}$) and $|1\rangle$ (amplitude $-1/\sqrt{2}$).

Evolution

The evolution of a closed quantum system is described by a *unitary* transformation. The state of the system at time t_1 is related to the state of the system t_2 by a unitary operation U which depends only on time t_1 and t_2[NC10]. Quantum evolution is reversible and this property is captured by the unitarity property. Thus, the system's state at time t_2 can be reversed to the one of time t_1 by the unitary operation $U^\dagger$.

$$|\psi'\rangle = U|\psi\rangle \qquad |\psi\rangle = U^\dagger\,|\psi'\rangle$$

Notation	Description
z^*	Complex conjugate of the complex number z
$\|\psi\rangle$	Column vector, also known as *ket*
$\langle\psi\|$	Row vector dual to $\|\psi\rangle$
$\langle\phi\|\psi\rangle$	Inner product between the vectors $\|\phi\rangle$ $\|\psi\rangle$
$\|\phi\rangle \otimes \|\psi\rangle$	Tensor product of $\|\phi\rangle$ $\|\psi\rangle$
$\|\phi\rangle \|\psi\rangle$	Abbreviated notation for tensor product of $\|\phi\rangle$ $\|\psi\rangle$
$\mathcal{A}^*$	Complex conjugate of the matrix $\mathcal{A}$
$\mathcal{A}^T$	Transpose of the $\mathcal{A}$ matrix
$\mathcal{A}^\dagger$	Hermitian conjugate of adjoint of the matrix $\mathcal{A}$, $\mathcal{A}^\dagger = (\mathcal{A}^T)^*$
$\langle\phi\| \mathcal{A} \|\psi\rangle$	Inner product between $\|\phi\rangle$ and $\mathcal{A}\|\psi\rangle$

Table 2.1: Summary of the bra-ket notation [NC10].

The linearity of quantum mechanics[NC10] implies that the state transformations are expressed as unitary square complex matrices, which results in $UU^\dagger = I$. Any unitary matrix is a valid quantum gate, but for practical reasons only some are used often, and the most frequent matrices are the Pauli matrices I, X, Y, Z, for which $X = X^\dagger$, $Y = Y^\dagger$, $Z = Z^\dagger$ and $Y = iXZ$.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

The X-gate is called *bit flip* gate and its action is similar to a classical bit flip, by transforming the $|0\rangle$ into $|1\rangle$, and vice versa. The Z-gate is called the *phase flip* gate. The effect of these gates is visualised as rotations on the Bloch sphere (see Figure 2.1b and 2.1c), and the consecutive application of two X/Z gates to a state will result in the output state being unchanged.

$$X(a|0\rangle + b|1\rangle) = a|1\rangle + b|0\rangle$$
$$Z(a|0\rangle + b|1\rangle) = a|0\rangle - b|1\rangle$$
$$XX|\psi\rangle = XX^\dagger|\psi\rangle = ZZ|\psi\rangle = ZZ^\dagger|\psi\rangle = |\psi\rangle$$

The exponentiation of the Pauli matrices results in the rotational gates R_x, R_y, R_z parameterised by the angle of the rotation[NC10]. Hence, the bit flip is a rotation by π around the X-axis, implying that $X = R_x(\pi)$, and the phase -flip is a rotation by π around the Z-axis,

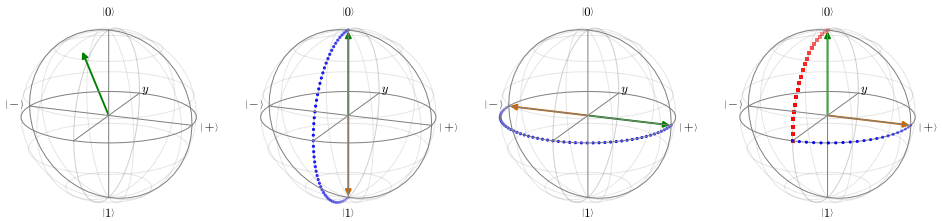


Figure 2.1: The Bloch sphere, in which the Z axis is the vertical axis with $|0\rangle$ at the north of the sphere and $|1\rangle$ at the south, while the X and Y axis form the equator: a) An arbitrary quantum state represented on the sphere; b) The bit flip operator implemented as a rotation around the X axis; c) The phase flip is a rotation around the Z axis; d) The effect of the Hadamard gate applied to the input $|+\rangle$ is a series of rotations that result in $|0\rangle$.

such that $Z = R_z(\pi)$.

$$
\begin{aligned}
R_x(\theta) \equiv e^{-i\theta X/2} &= \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} X = \begin{bmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \\
R_y(\theta) \equiv e^{-i\theta Y/2} &= \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} Y = \begin{bmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{bmatrix} \\
R_z(\theta) \equiv e^{-i\theta Z/2} &= \cos\frac{\theta}{2} I - i \sin\frac{\theta}{2} Z = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}
\end{aligned}
$$

In general, an arbitrary 2×2 unitary matrix U may be written as a product of rotations[NC10], where n and m are two orthogonal real unit vectors in three dimensions (see Equation 2.3).

$$
U = e^{i\alpha} R_n(\beta) R_m(\gamma) R_n(\delta) \tag{2.3}
$$

The Hadamard gate, the P and the T gate are commonly used in the context of quantum algorithms.

$$
\begin{aligned}
H &= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad P = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \\
T &= e^{i\frac{\pi}{8}} \begin{pmatrix} 1 & 0 \\ 0 & e^{-i\frac{\pi}{4}} \end{pmatrix} \quad R_x(\tfrac{\pi}{4}) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}
\end{aligned}
$$

The Hadamard gate is usually used to construct state superpositions by taking input states in a computational basis. Being a single-qubit gate, a Bloch sphere representation is possible, too (see Figure 2.1d). The Hadamard gate is also its own inverse ($HH = I$), and in

general, the following relations hold:

$$H|0\rangle = |+\rangle\,,\ H|1\rangle = |-\rangle$$
$$H = R_x(\pi/2)R_z(\pi/2)R_x(\pi/2)$$
$$T = e^{i\pi/8}R_z(\pi/4)$$
$$P = R_z(\pi/2),\ TT = P,\ PP = Z$$

The commutation properties between gates can be expressed by the commutator and the anti-commutator. If $[A, B] = AB - BA = 0$, then A and B commute, otherwise if $\{A, B\} = AB + BA = 0$, then A, B anti-commute. For example, the Pauli gates X, Z anti-commute, $\{X, Z\} = 0$, and their commutator is $[X, Z] = -2iY$. It should be noted that some gates neither commute nor anti-commute.

$$\text{commutator :} \quad [A, B] = AB - BA$$
$$\text{anti-commutator :} \quad \{A, B\} = AB + BA$$

Measurement

Quantum measurements are described by a collection $\{M_m\}$ of measurement operators (Hermitian complex matrices), for which $M_m^2 = M$. These act on the state space of the system being measured[NC10]: if the state of the quantum system is $|\psi\rangle$ immediatly before measurement, the probability of the result m is given by Equation 2.4 and the state $|\psi'\rangle$ of the system after the measurement is expressed by Equation 2.5. The output has *collapsed* to the result m, and the measured state is not the initial state before measurement.

$$p(m) = \langle\psi| M_m^\dagger M_m |\psi\rangle \tag{2.4}$$
$$|\psi'\rangle = \frac{M_m|\psi\rangle}{\sqrt{\langle\psi| M_m^\dagger M_m |\psi\rangle}} \tag{2.5}$$

For example, a measurement of a single qubit in the computational basis is described by the set M_0, M_1 of measurement operators, where $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$. For a state $|\psi\rangle = (a, b)^T$, the probabilities associated to outputs $|0\rangle$ and $|1\rangle$ are $p(|0\rangle)$ and $p(|1\rangle)$, while the state after the M_0 measurement will be $|0\rangle$ with an associated amplitude $a/|a|$. More specifically, by measuring $|+\rangle$ in the Z-basis, the measured state will be either $|0\rangle$ ($p(|0\rangle = 0.5)$ or $|1\rangle$ ($p(|0\rangle = 0.5)$.

$$p(0) = \langle\psi| M_0^\dagger M_0 |\psi\rangle = \langle\psi| M_0 |\psi\rangle = |a|^2$$
$$p(1) = \langle\psi| M_1^\dagger M_1 |\psi\rangle = \langle\psi| M_1 |\psi\rangle = |b|^2$$
$$\frac{M_0|\psi\rangle}{|a|} = \frac{a}{|a|}|0\rangle$$

A computational basis measurement, performed by the operators $\{|0\rangle\langle 0|, |1\rangle\langle 1|\}$, is also called a Z-measurement. An X-basis measurement is performed by using the operators $\{|+\rangle\langle +|, |-\rangle\langle -|\}$, such that the resulting state will be either $|+\rangle$ or $|-\rangle$.

A *rotated measurement* is performed in a basis different from the computational one, meaning that the Bloch sphere *axis*, along which the state is measured, results after rotating one of the main axis (X, Y, Z-axis). For example, X-basis measurement is achieved by first applying a Hadamard gate to the state, and then measuring in the Z-basis. In general, rotated measurements are either directly performed in the rotated basis, or the state to be measured is first rotated and then measured in the Z-basis.

The probability of a measurement result is not affected by the global phase $e^{i\gamma}$ of the state, and from an observational point of view two states $|\psi_1\rangle, |\psi_2\rangle$ that differ only by the global phase are equal[NC10].

$$|\psi_1\rangle^\dagger = \cos\frac{\theta}{2}\langle 0| + e^{i\phi}\sin\frac{\theta}{2}\langle 1|$$
$$\langle\psi_2| = |\psi_2\rangle^\dagger = e^{-i\gamma}(\cos\frac{\theta}{2}\langle 0| + e^{-i\phi}\sin\frac{\theta}{2}\langle 1|)$$
$$\langle\psi_2| e^{-i\gamma} M_m^\dagger M_m e^{i\gamma} |\psi_2\rangle = \langle\psi_1| M_m^\dagger M_m |\psi_1\rangle$$

Tensor product

The state space of a composite physical system is the tensor product of the state spaces of the component physical systems. Moreover, if the systems are numbered 1 through n, and system number i is prepared in the state $|\psi_i\rangle$, then the joint state of the total system is[NC10]:

$$|\psi_1\rangle \otimes |\psi_2\rangle \otimes \ldots \otimes |\psi_n\rangle$$

For example, the state of a two qubit system, where the first qubit is in the state $|1\rangle$ and the second qubit is $|0\rangle$, will be $|1\rangle|0\rangle = |10\rangle$. The tensor product postulate allows the definition of multiqubit gates. The simplest case extends the definition of a single-qubit gate U to be applied to a two-qubit system, but to affect only one of the qubits.

$$(I|\psi\rangle) \otimes (U|\phi\rangle) = (I \otimes U)|\psi\phi\rangle$$

The state of a multiqubit system is changed by applying a sequence of gates. For example, the sequence X, H applied to input state $|0\rangle$ will result in $|-\rangle$. Quantum gates can be applied in parallel on different qubits, if instead of applying a sequence of gates, the gates are applied simultaneously. This should not be confused with *quantum parallelism* which is the application of a single gate to a superposition of states.

$$\begin{aligned}
\text{Apply gate } U_1 &: (U_1|\psi_1\rangle)|\psi_2\rangle \ldots |\psi_n\rangle = |\psi_1'\rangle|\psi_2\rangle \ldots |\psi_n\rangle \\
\text{Apply gate } U_2 &: |\psi_1'\rangle(U_2|\psi_2\rangle) \ldots |\psi_n\rangle = |\psi_1'\rangle|\psi_2'\rangle \ldots |\psi_n\rangle \\
\text{Apply gate } U_n &: |\psi_1'\rangle|\psi_2'\rangle \ldots (U_n|\psi_n\rangle) = |\psi_1'\rangle|\psi_2'\rangle \ldots |\psi_n'\rangle \\
\text{Parallel application} &: (U_1 \otimes U_2 \otimes \ldots \otimes U_n)|\psi_1\psi_2 \ldots \psi_n\rangle
\end{aligned}$$

One of the most used multiqubit gates is CNOT (controlled-NOT). The gate has two inputs, known as *control* and *target* qubit, and, being a two-qubit gate, is defined as a 4×4 matrix. The CNOT conditions a bit flip of the target qubit on the control qubit being in the $|1\rangle$ state. For example, the $|10\rangle$ is transformed into $|11\rangle$ if the first qubit is the control and the second is the target. The CPHASE is similar to CNOT in the sense that it has a control and a target qubit, but it conditions a phase flip on the target if the control is $|1\rangle$. The *CPHASE* is control/target symmetric, as it transforms only the $|11\rangle$ state ($Z|0\rangle = |0\rangle$).

$$\mathit{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \qquad \mathit{CPHASE} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

$$\mathit{CNOT} : |00\rangle \to |00\rangle, |01\rangle \to |01\rangle, \quad |10\rangle \to |11\rangle, |11\rangle \to |10\rangle$$

$$\mathit{CPHASE} : |00\rangle \to |00\rangle, |01\rangle \to |01\rangle, \quad |10\rangle \to |10\rangle, |11\rangle \to -|11\rangle$$

The above state transformations illustrate that CNOT is a generalisation of classical XOR because $|\psi\rangle|\phi\rangle \to |\psi\rangle|\phi \oplus \psi\rangle$. Furthermore, CNOT and CHPASE are their own inverse, meaning that $\mathit{CNOT}^\dagger = \mathit{CNOT}$ and $\mathit{CPHASE} = \mathit{CPHASE}^\dagger$. A state is left unchanged if CNOT or CPHASE are applied twice.

Entanglement is a quantum specific phenomenon and it is modelled by the tensor product postulate. States that are a tensor product of other states are called *separable states*, but a state that cannot be expressed as a tensor product is an *entangled state*. The *Bell states* are often used as examples of such states.

$$B_{00} = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \qquad B_{10} = \frac{|00\rangle - |11\rangle}{\sqrt{2}}$$
$$B_{01} = \frac{|01\rangle + |10\rangle}{\sqrt{2}} \qquad B_{11} = \frac{|01\rangle - |10\rangle}{\sqrt{2}}$$

Entangled states are the result of applying entangling gates, like CNOT.

$$|00\rangle \xrightarrow{H_1} |+0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \xrightarrow{\mathit{CNOT}} B_{00}$$

Similarly to how classical Boolean circuits are entirely constructed by using logic gates from the set $\{\mathit{AND}, \mathit{OR}, \mathit{NOT}\}$, arbitrary quantum computations can be formulated from a reduced set of quantum gates. *Universality* can be achieved by using a *universal gate set*. One such set consists of CNOT (for its entangling action) and all the single qubit arbitrary gates[NC10]. However, there is a continuum of single qubit gates, while a reduced set of gates would be of a higher practical relevance.

The definition of a reduced gate set is based on the the Solovay-Kitaev (SK) theorem[DN06], which is a central result in quantum computation. The theorem shows that, roughly speaking, if a set of single-qubit quantum gates generates a dense subset of all quantum gates, that set is guaranteed to rapidly generate the set of unitary matrices of size 2 and determinant 1. It is possible to obtain good approximations to any desired gate using relatively short sequences of gates from the given generating set[DN06].

Very often the $\{H, Z, X, T, CNOT\}$ set is used to express arbitrary quantum computations by any desired accuracy of approximation. The Hadamard will construct superpositions, the T gate is used for Z-axis rotations like $P = TT$, and the $R_x(\pi/2)$ rotational gate is implemented by the sequential application of $HTTH = HPH$. This reduced gate set has also the advantage that there are known methods to implement it fault-tolerantly.

Another widely used set of gates is the *Clifford* group that consists of the CNOT gate, the Pauli gates X, Y, Z and the single qubit Hadamard gate and the P gate. The missing T gate from the set renders the Clifford group as not universal.

2.1.2 Quantum circuits

It is possible to visualise a quantum circuit as a series of gates acting on the qubits that are abstracted as *wires*. The wires do not have a direct physical representation, but are associated to a temporal axis. The inputs are on the left of the diagram, the outputs on the right, and in each time step a quantum gate is applied to a qubit's state abstracted by the wire. It is standard to assume the inputs being initialised in the computational basis $|0\rangle$, and, if otherwise, this will be indicated. Qubits, aiding during the construction of a circuit's functionality (are similar to temporary variables in programs), are called *ancilla*.

The control qubit is identified by a black dot for the controlled-gates, and a vertical line connects the control to the target. Due to being similar with the classical XOR gate, the target of CNOT will be marked with $\oplus$, but for CPHASE the target is also marked by a black dot. This notation results from the fact that the basis state transformation of the CPHASE is independent of which qubit is target or control. For a two qubit system, $|q_c\rangle$ (control qubit) and $|q_t\rangle$ (target qubit), $CPHASE\,|q_c q_t\rangle = CPHASE\,|q_t q_c\rangle$. Measurements in the computational basis are represented by  , while  indicates a qubit measurement in the basis B.

In contrast to classical circuits, no FANOUT is allowed in quantum circuits, as gates are reversible matrices. Furthermore, a FANOUT would allow information to be copied, which is impossible for quantum information, as stated by the *no-cloning theorem*[NC10]. If a unitary transformation COPY exists, and is able to copy at least two states $|\psi\rangle$ and $|\phi\rangle$ over a quantum register that is initialised into $|s\rangle$, the unitary gate COPY will preserve the inner product. Moreover, because the states are normalised, the inner product will equal its square. It follows that either the states are equal ($\langle\psi|\phi\rangle = 1$) or orthogonal ($\langle\psi|\phi\rangle = 0$) but not arbitrary.

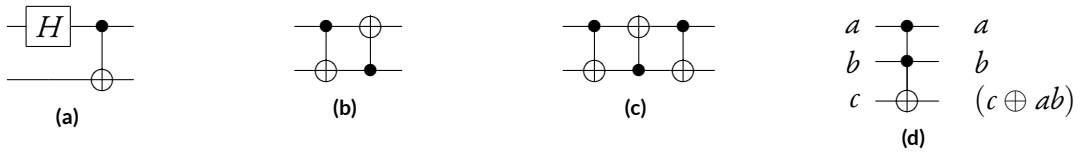


Figure 2.2: Quantum circuits: a) The Bell states are constructed by using the inputs $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$; b) A $|0\rangle, |1\rangle$ copying circuit; c) Swapping the states of two qubits using 3 CNOTS; d) The Toffoli gate.

$$\begin{aligned} COPY\,|\psi\rangle\,|s\rangle &= |\psi\rangle\,|\psi\rangle \\ COPY\,|\phi\rangle\,|s\rangle &= |\phi\rangle\,|\phi\rangle \\ \langle\psi|\phi\rangle &= (\langle\psi|\phi\rangle)^2 \\ \langle\psi|\phi\rangle &= 0 \rightarrow |\psi\rangle = |\phi\rangle \\ \langle\psi|\phi\rangle &= 1 \rightarrow |\psi\rangle\,, |\phi\rangle \text{ are orthogonal} \end{aligned}$$

The orthogonality constraint reduces heavily the generality of such a COPY gate. The direct implication is that, such a mechanism can only exist, for example, for $|0\rangle$ and $|1\rangle$ states (see Figure 2.2b), but the circuit will not copy the $|+\rangle$ state.

Although copying information is impossible, it is possible to swap quantum states. Compared to classical communication networks, where practically moving information between destination and source is performed by copying the information from the source and deleting it afterwards, in the quantum regime information is directly moved through swapping. Two completely arbitrary states can be swapped by the application of a CNOT sequence, as illustrated in Figure 2.2c.

$$\begin{aligned} |\psi\rangle\,|\phi\rangle &= \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle \\ &\xrightarrow{CNOT1} \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{11}\,|11\rangle + \alpha_{11}\,|10\rangle \\ &\xrightarrow{CNOT2} \alpha_{00}\,|00\rangle + \alpha_{01}\,|11\rangle + \alpha_{10}\,|01\rangle + \alpha_{11}\,|10\rangle \\ &\xrightarrow{CNOT3} \alpha_{00}\,|00\rangle + \alpha_{01}\,|10\rangle + \alpha_{10}\,|01\rangle + \alpha_{11}\,|11\rangle \end{aligned}$$

Reversible computing

The interest in classical reversible computing was initially motivated by Landauer's principle, which states that the erasure of information is dissipating energy[NC10]. Not erasing information equates to not allowing FANINs in the circuits, and this is the case for quantum circuits, which, due to the linearity of quantum computing, have an equal number of input and output qubits.

The practical impact of heat dissipation in classical circuits, which is a problem that hardware designers try to reduce, increased the relevance of the reversibility discussion. The initial hope was that computers might become more energy-efficient if classical computations would be reversible. Most gates are not reversible, and classical computation can only be reversed if these gates are reinterpreted as linear transformations. For example it is not possible to infer the inputs a and b after the application of an AND gate that resulted in the output c, $AND(a, b) = c$. The same observation holds for OR, XOR and, in general, for all the two input and one output gates. The NOT gate is however reversible as it's output is the negation of the input, and no information is erased.

A possible solution for reversing classical computations is to express the circuits using the *Toffoli* gate (see Figure 2.2d). The gate is a double controlled gate, has three qubits as inputs and two of them control the bit flip of the third qubit. Its action is expressed as the function *toffoli(a,b,c)=(a,b,c⊕ ab)*. By using Toffoli gates, arbitrary classical circuits can be completely constructed in a reversible manner. For example, the classical NAND gate is implemented as $toffoli(a, b, 1) = (a, b, 1 \oplus ab) = (a, b, \neg(ab))$, and the FANOUT is $toffoli(1, a, 0) = (1, a, 0 \oplus a) = (1, a, 0 \oplus a)$. As classical universality can be achieved by using exclusively NANDs and FANOUTs, it results that classical circuits can be transformed into reversible ones, as classical gates are functions of NANDs. The NAND constructions of AND, NOT and the OR gates are:

$$\begin{aligned} AND(a, b) &= NAND(NAND(a, b), NAND(a, b)) \\ OR(a, b) &= NAND(NAND(a, a), NAND(b, b)) \\ NOT(a) &= NAND(a, a) \end{aligned}$$

A different approach towards a formulation of classical reversibility starts from the quantum computing paradigm. In general, restricting a quantum copying gate only to orthogonal states is equivalent to reducing the representational power of quantum states. An arbitrary quantum state is described by 2^n complex amplitudes associated to computational basis states, whereas the orthogonal states, used in reversible computations, are representable by classical binary strings. Hence, classically simulating a copying gate does not introduce an exponential overhead. Such a reduced gate limits the computational power of a quantum circuit into one of a classical circuit. Maintaining the reversibility aspect of quantum computing, but considering only the copying of binary orthogonal states leads to reversible classical computing. Reversible circuits do not provide features of quantum circuits such as entanglement and state superposition.

Information teleportation

Quantum information (qubit states) cannot be copied, but there are ways to *move* information from one qubit to another through quantum state *teleportation*[BBC+93] (see Figure 2.3).

The teleportation circuit is defined over three qubits. The state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ (upper qubit) will be teleported using a Bell state (lower qubits) constructed using the circuit

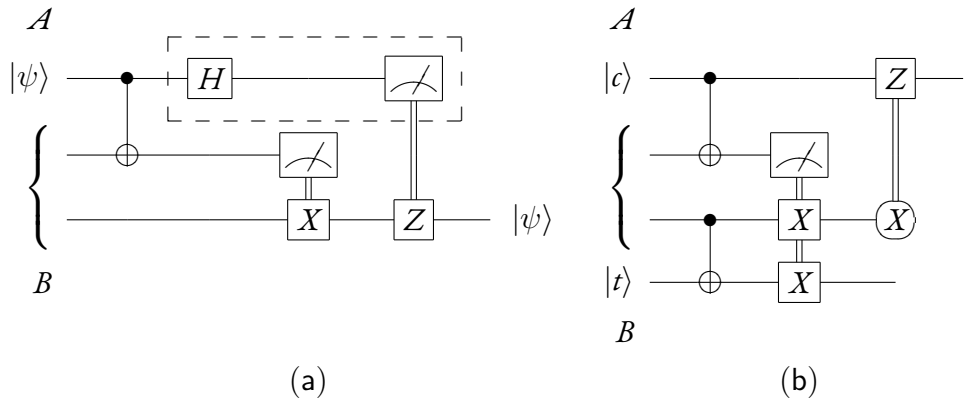


Figure 2.3: Teleportation circuits. The qubits indicated by the accolades are initialised into the B_{00} state: a) Circuit for teleporting a qubit [NC10]; b) Circuit for the remote CNOT [JP05].

X-measurement	Z-measurement	State	Correction
0	0	$\alpha\|0\rangle + \beta\|1\rangle$	I
0	1	$\alpha\|0\rangle - \beta\|1\rangle$	Z
1	0	$\alpha\|1\rangle + \beta\|0\rangle$	X
1	1	$\alpha\|1\rangle - \beta\|0\rangle$	XZ

Table 2.2: Teleportation of the state $\alpha\|0\rangle + \beta\|1\rangle$ is probabilistic and the measurement results dictate the necessary corrections of the output state

from Figure 2.2a. The entangled pair of qubits will be shared between two communication parties (A and B received each a qubit from the Bell pair), and A would like to send $|\psi\rangle$ to B. Measuring the qubit's state would destroy the state, such that another method for moving the state is necessary. It would be possible for A to entangle $|\psi\rangle$ with its Bell-pair-qubit (the CNOT in the circuit). The resulting three-qubit state is entangled, implying that $|\psi\rangle$ is in a way already shared between A and B. If A measures its two qubits in the X and in the Z-basis, the resulting state of B's qubit will be $X^{m1}Z^{m2}|\psi\rangle$, where $m1$ and $m2$ indicate the measurement results. Note that the X-basis measurement is an application of the Hadamard gate followed by a Z-measurement.

The $X^{m1}Z^{m2}$ *correction* is a direct result of the measurements being probabilistic. There is a 25% probability that $|\psi\rangle$ is correctly teleported, and Table 2.2 contains the measurement outcomes, the resulting output states and the necessary corrections. The final state of B is corrected depending on the X/Z-measurement results. The correction mechanism is illustrated in circuit diagrams by double vertical lines connecting the measurements to the X/Z gates.

After A performs the two measurements the corrections are communicated to B using classical means, as the measurement results are classical bits. This type of teleportation is consistent with the LOCC (local operations and classical communication) paradigm. While the state $|\psi\rangle$ is not communicated to B instantly, perfect quantum teleportation cannot

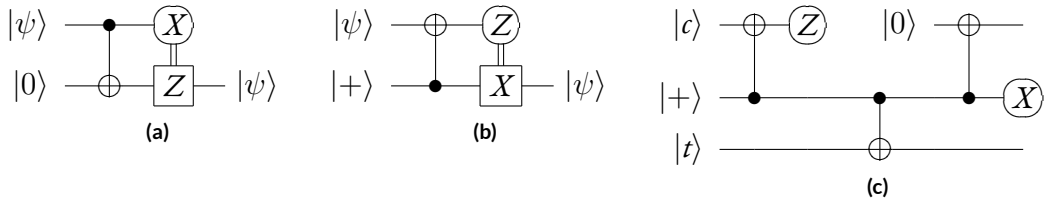


Figure 2.4: Circuits for teleporting the state of a source qubit to a neighbouring destination qubit, and a circuit for performing a CNOT between neighbouring qubits.

be faster than the speed of light. Furthermore, in a LOCC scenario the CNOT cannot be applied in a direct fashion between two qubits that are geographically separated, and for this reason A and B have to share a previously constructed Bell pair. The *remote CNOT* circuit presented in Figure 2.3 contains the control (A side) and target (B side) qubits including the Bell pair that enables the CNOT function.

Teleportation between two neighbouring qubits can be performed in the absence of the LOCC restriction (see Figure 2.4). In this situation, information is teleported after initialising the destination qubit to either $|0\rangle$ or $|+\rangle$, entangling the source with the destination qubit using a CNOT, and finally measuring the source qubit into the X- or Z-basis. Neither these circuits perform a perfect teleportation and the measurement results have to be interpreted such that corrections are applied on the circuit outputs. The action of these circuits is equivalent to the execution of a state swap (see Figure 2.2c), and the circuits are refered to in the literature as swapping circuits[NC10].

2.1.3 Gate teleportation

Information teleportation is a linear transformation of the destination qubit, such that its state is exactly the state of the source qubit. At the same time quantum gates are linear transformations, and information teleportation can be extended into *gate teleportation*. The direct result consists of teleported versions of the quantum gates from the discrete set $\{P, T, H, CNOT\}$, where the Pauli gates *could* be implemented using the teleported T and P gates, because $P = T^2, Z = P^2, X = HZH$. The Hadamard H gate is the application of three consecutive rotational gates: $R_z(\pi/2)$ followed by $R_x(\pi/2)$ and an $R_z(\pi/2)$ again, where $P = R_z(\pi/2)$.

The following derivation of rotational gates using teleportation circuits resembles the approach presented in[ZLC00]. Gate teleportation circuits, based on the similarities to the information teleportation circuits, consist of an entagling gate (CNOT), an ancilla qubit and the measurement of one the qubits. Constructing the gate teleportation requires to establish the following parameters: 1) the initial state of the ancilla; 2) which qubit will be control or target of the CNOT; 3) the basis of the final measurement; 4) the necessary corrections.

The initial state of the ancilla is inferred after taking into consideration that gate telepor-

tation acts as a black box that outputs an R_x or R_z rotated instance of an input state. It is known that R_x commutes with the target of CNOT, and R_z commutes with the CNOT control. Hence, in the R_z gate teleportation (see Figure 2.4b) the qubit corresponding to output (un-measured) will control the CNOT, and in the R_x gate teleportation (see Figure 2.4a) the output qubit will be targeted by the CNOT.

$$\begin{aligned} R_z(\theta)CNOT_c &= CNOT_c R_z(\theta) \\ R_x(\theta)CNOT_t &= CNOT_t R_x(\theta) \end{aligned}$$

The above observation enables the comparison of the gate teleporation circuits to the ones from Figure 2.4. The circuit in 2.4a is the structure of the teleported R_x, while 2.4b represents the structure of the teleported R_z. The only aspect that remains to be established is the states of the circuit ancillae. These states are inferred based on the above commutation properties, and it follows that the ancillae will be $R_x|0\rangle$ and $R_z|+\rangle$.

In particular, a special kind of states called *magic states* is used for gate teleportations[BK05], the major property of magic state being that there are know mechanisms for using them fault-tolerantly[FMMC12]. Magic states are used in the definition of a quantum computational model, in which the only required elementary operations are: 1) Clifford gates; 2) the initialisation into $|0\rangle$ and magic states; and 3) measurements in the computational basis. Two of the most common magic states are $|Y\rangle$ and $|\mathcal{A}\rangle$.

$$\begin{aligned} |Y\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle) = P|+\rangle = R_x^{\dagger}(\pi/2)|0\rangle \\ |\mathcal{A}\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}|1\rangle) = T|+\rangle \end{aligned}$$

The teleported $R_x(\pi/2)$ ($R_x^2 = R_x(\pi/2)$) uses $|Y\rangle$, the teleported P ($R_z^2 = R_z(\pi/2)$) uses the $|Y\rangle$ state, too, and for the T ($R_z^4 = R_z(\pi/4)$) the $|\mathcal{A}\rangle$ state is employed. Having derived the circuits for the gate teleportations, the corrections necessary at the circuit outputs of each of the three circuits have to be inferred.

Computing the corrections starts from the ones included in the information teleportation circuits; and the commutation properties between the Pauli corrections and the rotational gates are used. For example, in the teleported R_x case the commutation between Z (see Figure 2.4a) and R_x is necessary.

For R_x there are two possibilities to perform a $\pi/2$ rotation around the X-axis (depending on the direction of the rotation). Either the R_x^2 gate or its conjugate ($-\pi/2$ rotation) is implemented. The only difference between these options consists in the set of required corrections. The $R_x(-\pi/2)$ rotation commutes with the target of the CNOT, and transforms the $|0\rangle$ state into $|Y\rangle$. Therefore, if the X-measurement result is $|+\rangle$, no correction is

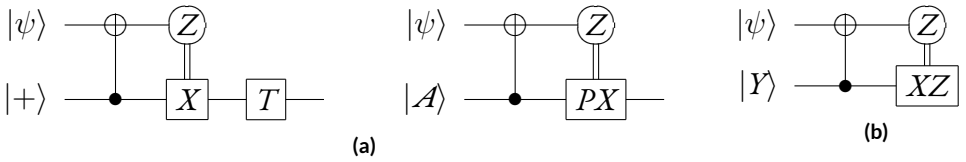


Figure 2.5: Achieving the effect of the rotational gates by using the magic states $|\mathcal{A}\rangle$, $|Y\rangle$ and corresponding teleportation circuits: a) The construction of the teleported T gate; b) The construction of the teleported P gate.

required. For the $|-\rangle$ measurement result, after commuting the initial Z-correction to the right of $R_x^{2\dagger}$, the final correction (up to global multiplicative factors) will be ZX.

$$\text{measurement result } |-\rangle \quad : \quad R_x^{2\dagger} Z = Z R_x^2 = (XZ) R_x^{2\dagger}; \rightsquigarrow XZ \text{ correction}$$

The implementation of the teleported R_x^2 resembles the one of the conjugate gate. For the previous implementation the $|+\rangle$ measurement result signaled no correction as the output state was rotated by $\pi/2$. For the current implementation the rotation should be $\pi/2$, and a correction by $R_x(\pi) = X$ is necessary. The $|-\rangle$ result indicates a Z correction, seen as the result of left-commuting the $R_x^{2\dagger}$ through the initial correction XZ.

$$\begin{aligned}
&\text{measurement result } |+\rangle \quad : \quad R_x^{2\dagger} X = R_x^2; \rightsquigarrow X \text{ correction} \\
&\text{measurement result } |-\rangle \quad : \quad R_x^{2\dagger} (XZ) = (X R_x^2)(XZ) = R_x^2 Z; \rightsquigarrow Z \text{ correction}
\end{aligned}$$

The teleported T gate is based on the circuit from Figure 2.4b. After measuring the $|0\rangle$ state the output of the circuit is correct and corresponds to $T|\psi\rangle$. Measuring $|1\rangle$ will indicate a necessary correction. The information teleportation circuit contained an X correction, such that the final correction is computed after left-commuting T with X.

$$\text{measurement result } |1\rangle \quad : \quad TXP = XT, \text{ because } PT^\dagger = T; \rightsquigarrow PX \text{ correction}$$

The PX correction from the T gate implies that there should be a teleported implementation of the P gate. This is indeed possible if instead of $|\mathcal{A}\rangle$ the $|Y\rangle$ state is used ($|Y\rangle = P|+\rangle$). The corrections of the teleported P gate are computed by noting that $XP = P^\dagger X$, and it follows that this implementation necessitates an XZ correction signaled by the measurement outcome $|1\rangle$.

$$TX = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ e^{i\frac{\pi}{4}} & 0 \end{pmatrix} = e^{-i\frac{\pi}{4}} \begin{pmatrix} 0 & e^{-i\frac{\pi}{4}} \\ 1 & 0 \end{pmatrix} = XT^\dagger$$

Gate teleportation circuits allow the application of remote gates, and CNOT can be also applied remotely. This version of CNOT is enabled by the single bit teleportations (see Figure 2.4c) and is useful in *nearest-neighbour* quantum computing architectures. In such architectures the remote application of quantum gates is restricted, due to technological factors, only on qubits that are physically neighbouring.

Gate	e_1	v_1	e_2	v_2
I	1	$\|0\rangle$	1	$\|1\rangle$
X	1	$\|+\rangle$	-1	$\|-\rangle$
Y	1	$(i,1)^T$	-1	$(-i,1)^T$
Z	1	$\|0\rangle$	-1	$\|1\rangle$

Table 2.3: The eigenvalues and the eigenvectors of the Pauli gates

2.1.4 Stabiliser formalism

The exponential difficulty of describing the evolution of a quantum system originates from the fact that, by incrementing the number of qubits operated on, the dimension of the state-space is doubled. Nevertheless, the eigenvalue properties of the Pauli gate set can be used to improve the representational overhead of a specific class of circuits. The Pauli gates have ± 1 eigenvalues, and the key idea is to represent quantum states as 1 eigenvectors. The eigenvalues and the eigenvectors of the Pauli gates are enumerated in Table 2.3.

The Pauli gates form a group under multiplication, and for a single qubit it consists of all the Pauli gates with multiplicative factors ± 1 and $\pm i$. The Pauli group for n qubits G_n consists of the n-fold tensor products of the Pauli matrices from G_1. The standard stabiliser notation is to drop the tensor product symbol when referring to the product of matrices: for example, XZ will be read $X \otimes Z$.

$$G_1 \quad = \quad \{\pm I, \pm iI, \pm X, \pm iX, \pm Y, \pm iY, \pm Z, \pm iZ\}$$

For a subgroup $ST \subset G_n$ ($-I \notin ST$) called the *stabiliser*, the states that are *stabilised* by ST are 1 eigenvectors of the stabilisers $s \in ST$. The state $|1\rangle$ is the 1-eigenvector of $-Z$, and $ST_1 = \{-Z\} \subset G_1$ stabilises $|1\rangle$. Another example is the Bell state B_{00} which is stabilised by $ST_2 = \{X_1X_2, Z_1Z_2\}$, where $X_1X_2 |B_{00}\rangle = |B_{00}\rangle$ and $Z_1Z_2 |B_{00}\rangle = |B_{00}\rangle$. The group ST is generated by a set of size that equals the number of qubits, and the order of the complete group is exponential in the size of the generating set. The generator set may contain also trivial identity stabilisers of the form $I_0 \ldots I_n$. A stabiliser generating set containing g non-trivial stabilisers will stabilise 2^{n-g} states[NC10].

For the previous examples, the ST sets presented were actually the generators of the ST-group. It can be noticed that $(X_1Z_1)(X_2Z_2)$ is a stabiliser of $|B_{00}\rangle$, too, and the neutral element of matrix multiplication is generated by $(X_1X_2)^2 = (I_1I_2)$. The generators of the stabiliser subgroup represent particular states in a very compact manner. Furthermore, the stabiliser formalism is the basis for introducing error-correction mechanisms for quantum information.

The *normaliser* $\mathcal{N}(G_n)$ of G_n is defined as the set of gates that transforms elements of G_n into elements of G_n[NC10]. The normaliser is usally called the *Clifford group*. The definition of this group is based on the observation that if the state $|\psi\rangle$ is stabilised by s, then the applica-

Gate	Input	Output	Gate	Input	Output
CNOT	X_1	X_1X_2	CPHASE	X_1	X_1Z_2
	X_2	X_2		X_2	X_2
	Z_1	Z_1		Z_1	Z_1
	Z_2	Z_1Z_2		Z_2	X_1Z_2
H	X	Z	X	X	X
	Z	X		Z	-Z
P	Z	Z	Z	X	-X
	X	Y		Z	Z

Table 2.4: Transformations of the elements of the Pauli group under conjugation by gates from its normaliser [NC10].

tion of the gate U is computed according to Equation 2.6.

$$\begin{aligned} s|\psi\rangle &= |\psi\rangle, s \in ST \\ U|\psi\rangle &= Us|\psi\rangle = (UsU^\dagger)U|\psi\rangle \end{aligned}$$

Under conjugation the gates from $\{CNOT, H, P\} \subset N(G_n)$ transform $s_i \in G_n$ into $s_o \in G_n$, and the gates generate the normaliser of G_n [NC10]. Table 2.4 enumerates the stabiliser transformation for the gates that are used throughout this work.

Not all the quantum gates are in the normaliser. For example, the T gate transforms the stabiliser X into a superposition of stabilisers. This shows that it is not a normaliser gate. Representing a circuit with T gates (or any other non-stabiliser gates) using the stabiliser formalism requires doubling the set of stabilisers each time a T is encountered (see Equation 2.6). Hence, the application of T gates results in an exponential increase of the observed state space. Due to this overhead, the stabiliser formalism cannot be used to represent universal quantum computations and their quantum states. The representable states still contain superposition and entanglement (due to H and CNOT).

$$TZT^\dagger = Z \quad ; \qquad TXT^\dagger = \frac{X+Y}{\sqrt{2}} \tag{2.6}$$

The stabiliser formalism can be used to efficiently simulate quantum circuits consisting of the $\{CNOT, H, P\}$ gates by representing the stabilisers as a table (*tableau* [AG04]). These gates are called *Clifford* gates. The result is known as the Gottesman-Knill theorem [AG04], and an application of the theorem is the CHP circuit simulator [AG04] that uses tables containing $(2n+1)$-bit long rows for representing the stabilisers (X and Z) together with the phase (positive or negative) of the associated state. During simulation the state stabilisers are constantly updated after each application of a gate by applying the transformation properties from Table 2.4.

State initialisation and the application of gates using the stabiliser formalism are direct applications of the stabilisers' transformations under conjugation by the gate matrix. The

formalism supports state measurements, too, and the commutation properties of the Pauli gates are used (see Figure 2.6). A stabiliser m to be measured either commutes with all the generators $s \in ST$, or it anti-commutes with one of the generators. If m anti-commutes with more then one generators $s_1, s_2 \ldots s_n$, then it can be shown that replacing s_2 by $s_2 s_1, \ldots, s_n$ by $s_n s_1$ leaves only s_1 anti-commuting with m[NC10].

If m anti-commutes with one generator, then the measurement output will be random, with the probabilities $p(m) = p(-m) = 0.5$[NC10]. For example, measuring the Z stabiliser of an X-stabilised qubit is equivalent, in the quantum circuit formalism, to performing a Z-measurement of the $|+\rangle$ state, which results in either $|0\rangle$ (stabilised by Z) or $|1\rangle$ (stabilised by $-Z$). In the table representing the stabilisers, the row containing s_i will be updated to reflect $\pm m$.

If m commutes with all the generators s, then it is a member of the stabiliser set ($m \in ST$) because $m|\psi\rangle = (sm)|\psi\rangle = m(s|\psi\rangle)$. The measurement result will be deterministic, and the phase of the result is indicated by the multiplication of the phases associated to the generators.

The computational complexity of stabiliser circuit simulation using the CHP is $\mathcal{O}(n^2)$. The application of a stabiliser gate is linear in the number of stabilisers that have to be updated, but measurement is more complex. After checking if m anti-commutes with at least one stabiliser, the table is updated using a method similar to Gaussian elimination. An improvement proposed by Aaronson[AG04] reduces the update complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

	1	2
0	Z	I
0	I	Z

(a)

	1	2
0	X	I
0	I	Z

(b)

	1	2
0	X	X
0	Z	Z

(c)

	1	2
0	Z	I
0	Z	Z

(d)

	1	2
1	Z	I
0	Z	Z

(e)

	1	2
1	Z	I
1	I	Z

(f)

Figure 2.6: The transformations of the stabiliser table for the circuit from Figure 2.2a. If the circuit contains an X-measurement on the first qubit and an Z-measurement on the second qubit, the last three tables illustrate the stabilisers after measurement. a) The initial stabilisers corresponding to the input $|00\rangle$ state; b) The Hadamard on the first qubit transforms Z into X; c) The CNOT having the control on the first qubit and the target on the second qubit transforms $XI \rightarrow XX$ and $IZ \rightarrow ZZ$; d) The Z measurement will result in Z and the phase is indicated by 0; e) The measurement result $-Z$ is indicated by the phase value 1. f) The stabiliser IZ is generated by $(ZI)(ZZ) = IZ$, and a Z-measurement result of the second qubit is stabilised by $-Z$.

2.1.5 Graph states

Graphs are useful for representations of stabiliser states[HEB04] and were used for introducing novel paradigms of quantum computing[RBB03]. Graphs were also used for an even more efficient simulation of stabiliser circuits[AB06]. The measurement operation in the CHP simulator had a complexity of $\mathcal{O}(n^2)$, whereas the graph representation of stabiliser states en-

ables the reduction of the measurement complexity to $\mathcal{O}(n\log(n))$, for stabiliser circuits of n qubits.

Graph states abstract qubits as vertices and the interactions between them as edges. Each vertex has an associated *VOP* (vertex operators), the key idea being that any stabiliser state is transformed into a graph state by applying a tensor product of local Clifford operations (as in LOCC, the CNOT gate is not included) that are saved in the *VOPs*.

According to [ABo06], an n-qubit graph state $|G\rangle$ is a quantum state associated with an undirected graph $G = (V, E)$, whose $|V| = n$ vertices correspond to the n qubits, while the edges E describe quantum correlations, in the sense that $|G\rangle$ is the unique state satisfying the n eigenvalue equations

$$K_G^{(a)} |G\rangle = |G\rangle \, , a \in V$$
$$\text{with } K_G^{(a)} = X_a \bigotimes_{b \in nbgh_a} Z_b$$

where the neighbourhood of a, $ngbh_a := \{b | \{a, b\} \in E\}$, is the set of vertices adjacent to a.

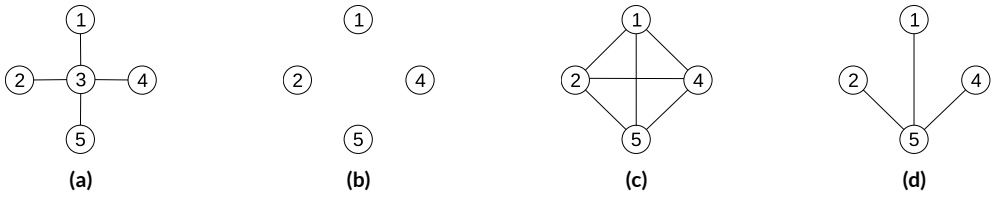


Figure 2.7: Graphs states (VOPs are not indicated): a) A graph state consisting of 5 qubits arranged as a star; b) The resulting graph after measuring qubit 3 in the Z-basis; c) The resulting graph after measuring qubit 3 in the Y-basis; d) The resulting graph after measuring qubit 3 in the X-basis.

A graph state $|G_n\rangle$ on n qubit is constructed by initialising it into the $|G_n\rangle = |+\ldots+\rangle$ state and applying CPHASE on all pairs of neighbouring qubits (see Figure 2.8). The stabiliser tables (phase bits are neglected, and empty cells stand for I) from Figure 2.8 illustrate the construction of the graph state from Figure 2.7a. The deletion of graph edges is performed by re-applying the CPHASE gate between two qubits already connected by an edge. For example, if the graph vertices were initialised into $|0\rangle$, then their *VOP*s would contain H for the beginning. Transforming $|0\rangle \rightarrow |+\rangle$ would result in the application of H to each *VOP*, leaving them $VOP = \{I\}$.

GraphSim is a stabiliser circuit simulator based on the graph representation of stabiliser states [ABo06], and its key advantage over the table simulation technique is the way measurements are performed. The implementation of the simulator will not be detailed, but it is important to mention the rules how the edges and the *VOP*s are updated. The discussion refers to the corrections that depend on the measurement result $r \in \{0, 1\}$. Measuring in one of the basis $m \in \{X, Y, Z\}$ will result in one of the eigenvalues of m being returned (see Table 2.3) which are of the form $(-1)^r$.

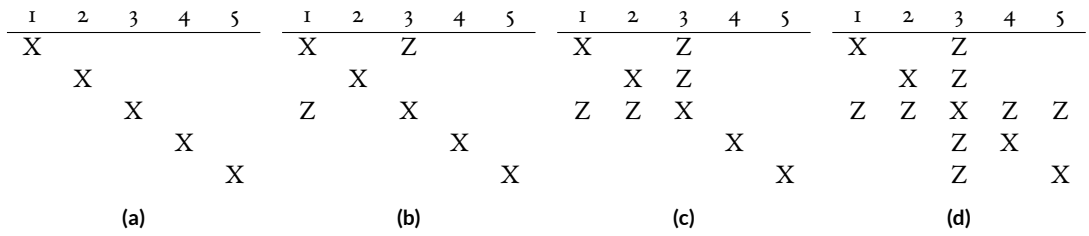


Figure 2.8: Construction of the graph-state from Figure 2.7a. a) The five qubits are initialiased into the $|+\rangle$ state; b) Qubit 3 is entangled to qubit 1 using the CPHASE gate; c) CPHASE between qubits 3 and 2; d) The final stabiliser table after entangling qubit 3 to the qubits $ngbh_3 = \{1, 2, 4, 5\}$.

Every element $C \in VOP$ is a Clifford gate (normaliser of G_n), and, in general, performing a computational basis measurement is equivalent to measuring in one of the $\{X, Y, Z\}$ basis. The element C transforms the measurement basis into the same or another measurement basis, due to the conjugation properties of the Pauli gates. Measurement transformations and how measurement results are complemented were derived in [HEB04].

The *deletion* [CLN05] of qubits and edges is performed by Z-measurements, and the measurement result is random (see for example the measurement of qubit 3 in Figure 2.8). In the following the *symmetric set difference* Δ will be used to update the set of edges, where $A \Delta B = (A \cup B) \setminus (A \cap B)$. A measurement of the Y observable requires the complementation of the graph edges and VOPs set according to:

$$\begin{aligned} E &= E\Delta\{(b,c)|b,c \in ngbh_a\} \\ C_b &= C_b R_z(\pi/2)^{(\dagger)}, \text{ for } b \in ngbh_a \cup \{a\} \\ C_b &\in VOP_b, \dagger \text{ in the paranthesis is applied for } r = 1 \end{aligned}$$

The X-measurement is more complex, as it requires more edge operations and VOP updates [AB06]. The measurement result r is random, too.

$$\begin{aligned} E &= E\Delta\{(c,d)|c \in nbgh_b, d \in nbgh_a\} \\ &\quad \Delta\{\{(c,d)|c,d \in nbgh_b \cap nbgh_a\}\} \\ &\quad \Delta\{(b,d)|d \in nbgh_a \setminus \{b\}\} \\ C_c &= \begin{cases} C_c Y^r, \text{ for } c = a \\ C_c R_y(\pi/2)^{(\dagger)}, \text{ for } c = b, \dagger \text{ in the paranthesis is applied for } r = 1 \\ C_c Z, \text{ for } c \in \begin{cases} ngbh_a \setminus ngbh_b \setminus \{b\}, \text{ for } r = 0 \\ ngbh_b \setminus ngbh_a \setminus \{a\}, \text{ for } r = 1 \end{cases} \\ C_c, \text{ otherwise} \end{cases} \end{aligned}$$

The following examples illustrate how measurements affect a graph-state. The result of measuring qubit 3 from Figure 2.7a in the Z-basis is illustrated in Figure 2.7b. Measuring

the central vertex in the Y-basis generates a new graph which is the complement of the initial one[HEB04], and the updated VOPs will contain the P Clifford gate (see Figure 2.7c). Should a second vertex of this graph be measured in the Y-basis, the measurement would results into an X-basis measurement as $PYP^{\dagger} = R_z(\pi/2)YR_z(\pi/2)^{\dagger} = X$. Finally, the X-basis measurement of qubit 3 is illustrated in Figure 2.7d, where qubit 5 (in the previous update rules this referred as qubit b) was considered the neighbour of qubit 3.

2.2 Measurement-based quantum computing

Teleportation based techniques for computing were introduced before a *measurement based quantum computing* (MBQC) was formulated. MBQC takes the concepts of teleportation and measurement further and the major characteristic of this paradigm is the constructive usage of measurements. This contrasts to the assumption that measurements are destructive and *collapse* the state of a quantum system into one of the two eigenstates of the measured observable (e.g., $|0\rangle$ or $|1\rangle$ for Z).

Restricting the possible operations only to qubit measurements is possible if the measured quantum state is prepared beforehand in a particular manner where the entanglement necessary for the computation is *embedded* into the input state. The temporal order and the type of the single qubit measurements applied to the composite input state defines the *measurement pattern*. For example, the measurement pattern $\{X_3, X_1, X_2\}$ applied on the three-qubit state $\left|q_1q_2q_3\right\rangle$ implies that first qubit q_3 is X-measured, followed by X-measurements of the qubits q_1 and q_2. The construction of the $\left|q_1q_2q_3\right\rangle$ state is explained after the applications of MBQC are enumerated.

The MBQC paradigm allows the existence of specialised entanglement-centres (similar to computing centres) that prepare large lattices of entangled qubits, and a client would receive a prepared cluster-state, which he only has to measure in order to perform computations. An MBQC computation would also support a computation scenario that is similar to cloud computing: a client sends to the entanglement-centre the measurement-patterns, and after the computation is finished the client receives its output. It is possible to perform this kind of cloud-like computation *blindly*, meaning that the entanglement-centre will not know what it computes[BFK09,BKB+12]. From a theoretical perspective the relevance of the property is mirrored in perfectly secure computations performed on untrusted remote platforms.

MBQC will be introduced as an application of graph states, by using the structure of *cluster states*, which are states represented as planar graphs having the vertices arranged into squares (see Figure 2.9a). The relation between MBQC and teleportation-based computational schemes[NC10] has been investigated in[CLN05,AL04,JP05]. The following introduction to MBQC starts from the construction of rotational gates using teleportation-based circuits operating on graph-states.

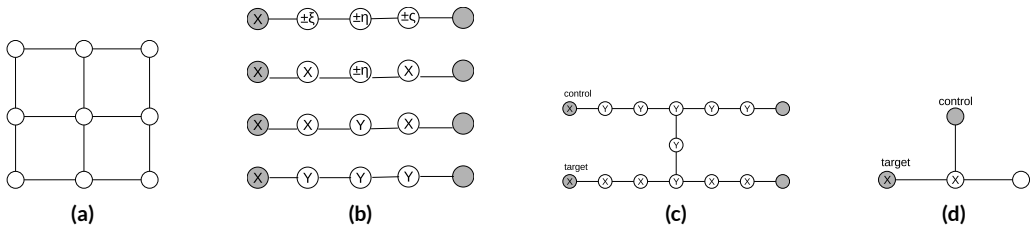


Figure 2.9: Elements of MBQC: a) A 3×3 graph state illustrative of the lattice structure used for computing; b) The most common single-qubit MBQC gates [RBB03] can be implemented using a linear graph consisting of 5 qubits, where the input state is at the left, and the output is at the right. The measurement order is from left to right, and the measurement angles in the equator plane of the Bloch sphere are indicated on the individual qubits. In their top-down order the presented implementations are an arbitrary rotation, an $R_z(\eta)$, a P gate and a Hadamard gate; c) The CNOT gate [RBB03]; d) A second possible implementation of the CNOT gate.

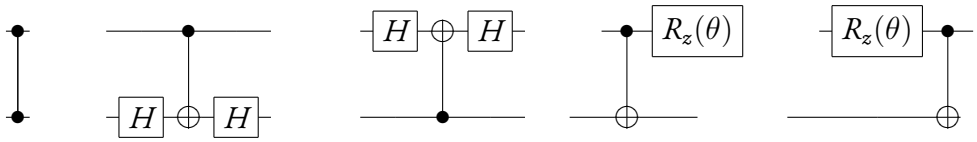


Figure 2.10: The CPHASE gate implemented by two Hadamard gates surrounding the target of a CNOT. $(I \otimes H_c)CNOT(I \otimes H_c) = CPHASE$, and the R_z rotational gates commute with the control of the CNOT.

MBQC GATES

Arbitrary rotational gates around the X and Z axis are building blocks of unitary gates (see Equation 2.3) enabling the universality of MBQC. The CNOT gate is not directly supported, as the measurements are performed on single qubits. However, the structure of the underlying cluster state can be used by noting that a CPHASE gate between two qubits is formed by a CNOT surrounded by two Hadamard gates (see Figure 2.10).

MBQC MEASUREMENT PATTERNS

The Z-measurement of a graph state vertex removes the vertex and its incident edges, and as a result, in the MBQC model, measurements are performed in the so-called *equator* of the Bloch sphere (the plane defined by the X and Y axis). The general measurement basis is $B(\phi) = \left\{ \frac{|0\rangle + e^{i\phi}|1\rangle}{\sqrt{2}}, \frac{|0\rangle - e^{i\phi}|1\rangle}{\sqrt{2}} \right\}$, where the angle ϕ is chosen such that arbitrary rotated measurements can be implemented.

A quantum computation expressed as a quantum circuit is directly mapped to an MBQC computation by translating each quantum gate into a pattern of measurements applied to the cluster states similar to the one from Figure 2.9a. Unnecessary cluster qubits (not required for the computation) are removed by Z-measurements [RBB03]. In Figures 2.9b and 2.9c the measurement patterns for some of the most common gates are presented. The measure-

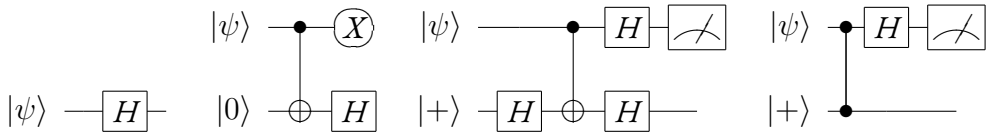


Figure 2.11: Derivation of the MBQC Hadamard gate construction using CPHASE circuit identities.

ment diagrams are based on the observation from Equation 2.7 where X-rotations are formulated using the Hadamard gate and Z-rotations. Equation 2.3 will be rewritten as Equation 2.8 for three measurement angles α, β, γ, and Equation 2.9 will express the unitary U.

$$R_x(\phi) = HR_z(\phi)H \tag{2.7}$$
$$U = R_z(\alpha)HR_z(\beta)HR_z(\gamma) \tag{2.8}$$
$$U = H(HR_z(\alpha))(HR_z(\beta))(HR_z(\gamma)) \tag{2.9}$$

There are two possible constructions for a Hadamard gate. The leftmost circuit from Figure 2.11 represents the state $|\psi\rangle$ which is transformed by a Hadamard gate, but by using a teleportation (ancilla state initialised into $|0\rangle$, CNOT and X-measurement) the circuit can be redrawn. Using the CPHASE identities from Figure 2.10, the third circuit will consist of the ancilla state initialised into the $|+\rangle$ state, and a CPHASE is used instead of the CNOT.

The X-measurement is probabilistic, and the output will be equivalent to the $Z^m H|\psi\rangle$, where m indicates the measurement result. This observation is also supported by the *VOP* update rules of measuring in the X-basis one qubit of a 2-vertex cluster state.

$$\begin{aligned}
|\psi\rangle|+\rangle &= a|00\rangle + a|01\rangle + b|10\rangle + b|11\rangle \\
\mathit{CPHASE} &\rightarrow a|00\rangle + a|01\rangle + b|10\rangle - b|11\rangle \\
\mathcal{M}_x^1 &\rightarrow a|+0\rangle + a|+1\rangle + b|+0\rangle - b|+1\rangle = |+\rangle(a|+\rangle + b|-\rangle) \\
\mathcal{M}_x^1 &\rightarrow a|-0\rangle + a|-1\rangle - b|-0\rangle + b|+1\rangle = |-\rangle(a|+\rangle - b|-\rangle)
\end{aligned}$$

The second Hadamard construction (see Figure 2.12) is performed by the rotations $H = R_x(-\pi/2)R_z(-\pi/2)R_x(-\pi/2)$, and is represented by the measurement pattern consisting of an X-measurement followed by three consecutive Y-measurements (see the bottom measurement pattern in Figure 2.9b). A Hadamard gate in front of a Z-measurement can be understood as an X-flipped Y-rotation $R_x(\pi)R_y(\pi/2)$, similarly to how a measurement in the Y-basis is the equivalent of $P^\dagger = R_z(-\pi/2)$-rotated X-measurement. The final state $|\psi'\rangle$ is (up to the required Pauli corrections) the transformation:

$$\begin{aligned}
|\psi'\rangle &= (HP^\dagger H)P^\dagger(HP^\dagger H) \\
&= R_x(-\pi/2)R_z(-\pi/2)R_x(-\pi/2) = H
\end{aligned}$$

The measurement pattern for the phase gate P is based on teleportation circuits, too, and it is easy to verify that the implemented state transformation is $(HH)P^\dagger(HH)$ which is correct up to a Pauli Z-correction. Without considering the Z-correction, the measurement

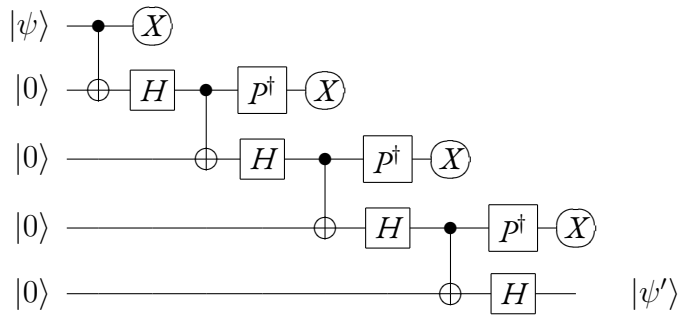


Figure 2.12: Derivation of the MBQC Hadamard gate construction using single-bit teleportation circuits.

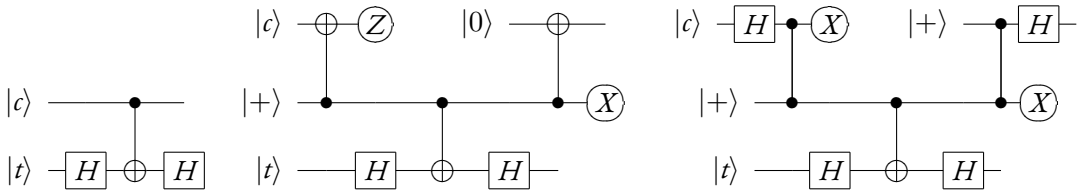


Figure 2.13: Construction of an MBQC CNOT gate starting from the entangling CPHASE gate by using circuit identities and single-bit teleportation circuits.

pattern $\{X, X, Y, X\}$ (see the third measurement pattern in Figure 2.9b) implements the $P = (HP^\dagger H)P^\dagger(H)$ state transformation[JP05]. After using Equation 2.8, the P gate construction is generalised into arbitrary Z-rotations (the second measurement pattern from Figure 2.9b). Furthermore, arbitrary rotations are performed by the first(top-most) measurement pattern from Figure 2.9b.

The constructions of the CNOT gate using various measurement patterns were presented, for example, in[RBB03] and[AL04], but in the following the gate derivation starts from the *CPHASE* gate by using teleportation circuit identities which, for simplicity, do not contain the correction gates (see Figures 2.13 and 2.9d). The CPHASE is rewritten as a CNOT and two Hadamard gates enclosing the target, implying that the simulated target qubit (lower wire) requires a Hadamard transformation before it is operated on (see the first Hadamard gate construction), and requires to be transformed back by a Hadamard gate after CNOT. The teleportation circuit from Figure 2.4b is used to extend the initial circuit, and the result after converting CNOTs to CPHASEs is the third circuit from Figure 2.13. The final diagram includes two H gates on the upper qubits, as these result from decomposing the CPHASEs. The simulated control qubit (top wire) should also be Hadamard transformed before and after the CNOT.

2.3 Fault-tolerant quantum computing

Quantum information processing using qubits, quantum circuits and the MBQC computational model were assumed to exist and function in an isolated environment (excepting initialisation and measurement). However, it is impossible to perfectly isolate a system in reality, and unwantend interactions with other systems can perturb its functionality. For example, the effect of cosmic rays as sources of soft-errors in classical computer memories was recognised in [ZL79], and it was predicted that even aggressive shielding of the memories would result in a non-zero probability of faults and errors. The existing interactions between two systems (e.g. universe and memory chip) is considered a source of *interaction faults*, which after activation result into errors, and can furthermore imply the failure [ALRL04] of one of the systems. Classical fault-tolerant computer design techniques include the use of error detection and correction methods combined with the construction of redundancy [Avi67], and these key aspects are also used for building fault-tolerant quantum computers.

2.3.1 Pure and mixed states

The inherent vulnerability to errors (expressed as probability of errors) is best captured in the context of fault-tolerance by the *quantum operator* formalism, which is motivated by the distinction between between *pure* and *mixed* states. The change of perspective from closed to open quantum systems is motivated by their evolution: open systems can start from a pure state and end into a mixed state. The result of a quantum computation is gradualy destroyed. A state is pure if its amplitudes (entries of the state vector) are exactly determined, whereas a state is mixed if it represents a set of possible (pure) states that have an associated probability. In order to express the mixture, the *density operator* is defined as the linear sum of the inner products of the pure states ψ_i and their associated probabilities p_i.

$$\rho \;=\; \sum_i p_i \left|\psi_i\right\rangle \left\langle\psi_i\right|$$

The criterion to decide if a state is mixed or pure is given by the trace of the squared density operator (matrix). For pure states $tr(\rho^2) = 1$, whereas for mixed states $tr(\rho^2) \leq 1$. The diagonal entries of the matrix correspond to the squared amplitudes of the orthonormal states (e.g. computational basis states) that span the state vector, and the probability of measuring the observable $\mathcal{M}_0$ is given by $p(m) = tr(\rho\mathcal{M}_0)$. Since the pure states are unit-vectors, it follows that $\sum_i tr(\rho\mathcal{M}_i) = 1$.

Mixed states cannot be represented by unit-vectors, and the density operator of a mixed state is the weighted sum ($\sum_j p_j = 1$) of the constituent pure states density operators ρ_j, implying that the probability of observable $\mathcal{M}_0$ is the weighted sum of individual probabilities. Furthermore, the exponentiation of the density operator representing a mixed state (ρ_{mix}) is the sum of the pure density matrices, weighted by the squares of probabilities p_j; because

$p^2 \leq p$, it follows that for mixed states $tr(\rho_{mix}^2) \leq 1$ [Bero8]:

$$\begin{aligned} \rho_{mix} &= \sum_j p_j \rho_j^{pure} \\ p(m) &= \sum_j p_j tr(\rho_j^{pure} M_0) \\ \rho_{mix}^2 &= \sum_j p_j^2 \rho_j^{pure} \end{aligned}$$

The most general quantum operation formulated using the operator-sum representation is presented in Equation 2.10, where the operator set $\{E_k\}$ contains the operation elements (called Kraus operators) of the operation [NC10], which satisfy the completeness relation (Equation 2.11).

$$\rho \rightarrow \sum_i E_i \rho E_i^\dagger \tag{2.10}$$

$$\sum_i E_i^\dagger E_k = I \tag{2.11}$$

Quantum information processing can be expressed using density operators instead of state vectors, and the postulates can be reformulated accordingly [NC10]. A quantum system is completely described by its density operator ρ, whereas the evolution of a closed system is an unitary transformation U, that takes ρ as input and outputs $\rho' = U\rho U^\dagger$. Quantum measurements are a collection of measurement operators M_m (previously observables). The probability of measurement outcome m is $p(m) = tr(M_m^\dagger M_m \rho)$, the state after the measurement is $\frac{M_m \rho M_m^\dagger}{tr(M_m^\dagger M_m \rho)}$, and the measurement operators satisfy the completeness equation $\sum_m M_m^\dagger M_m = I$.

A system could be in the state $\rho_I = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}I$, which is a mixture of $|0\rangle$ and $|1\rangle$ with probability one half. The same density operator would result if the same system was prepared with equal probability in the $|+\rangle$ and $|-\rangle$ states. As a result, it is wrong to conclude that two equal density operators correspond to equal states.

Mixed states have Bloch sphere visualisation, too; by extending the representation to a Bloch ball, pure states are points on the surface of the ball, and mixed states are points inside the ball. The difference between pure and mixed states is more obvious when the states are transformed by an unitary. For example, applying a Hadamard to $|+\rangle$, the amplitudes of the basis states interfere, resulting in $H|+\rangle = |0\rangle$. On the contrary, for ρ_I an incoherent mixture of two states , the Hadamard transformation $H(\rho_I)H = H(\frac{1}{2}I)H = \frac{1}{2}I$ leaves the state unchanged [Baco6]. On the Bloch ball, the evolution of a pure state into a mixed state is visualised as moving points from the surface towards the inner of the ball.

2.3.2 Quantum channels

The analog of classical communication channels are the quantum channels. The effect of quantum channels on input states is expressed by quantum operations, which are formulated as operator-sums. The quantum channels are models of quantum state transformations in time under quantum noise[Pre98a,NC10].

The analogy to classical communication channels can start from the symmetric bit flip channel. For a classical bit, which undergoes a bit flip with probability e, its state can be represented by a vector of probabilities $p = (p_0, p_1)$, for p_0 being the probability of 0, and p_1 the probability of 1. The probabilistic transition ($q = Ep$) of the bit is formulated using the linear transformation (E), where $q = (q_0, q_1)$ represents the output probabilities.

$$\begin{pmatrix} q_0 \\ q_1 \end{pmatrix} = \begin{pmatrix} 1-e & e \\ e & 1-e \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix}$$

Quantum channels are acting like unitaries, lineary transforming an input state into an output state, but the basis states amplitudes affected. For example, given the probability p of a bit flip on the state $|0\rangle$, the final state is $|f\rangle = ((1-p)I + pX)|0\rangle$, resulting in the mixed state $\rho_f = (1-p)|0\rangle\langle 0| + p|1\rangle\langle 1|$.

Bif flip and phase flip channels

either flip the computational basis states of a quantum state, or flip the sign of the relative phase.

$$\textit{Bit flip}: \sqrt{1-p}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \;;\; \sqrt{p}\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$
$$\textit{Phase flip}: \sqrt{1-p}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \;;\; \sqrt{p}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

Depolarising channels

represent a process in which the density matrix is replaced by a completely mixed state with probability $1/2$, and left unchanged with probability $1/2$. The Kraus operators of such a channel are the set D, where the probability of a bit flip, phase flip, or both are equal if an error occurs.

$$D = \{\sqrt{1-3p/4}I, \sqrt{p}X/2, \sqrt{p}Z/2, \sqrt{p}Y/2\}$$

Amplitude-damping channels

transform the $|1\rangle$ state into $|0\rangle$, corresponding to the physical process of losing a quantum of energy to environment, or leave $|0\rangle$ unchanged (if a quantum of energy was not lost),

but reduce the amplitude of the $|1\rangle$ state, as if the probability of $|0\rangle$ was higher than the probability of $|1\rangle$.

$$\begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix} \quad ; \quad \sqrt{p}\begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix}$$

Phase-damping channels

destroy the $|0\rangle$ and reduce the amplitude of the $|1\rangle$ state, or leave $|0\rangle$ unchanged but reduce the amplitude of $|1\rangle$ similarly to amplitude-damping channels. For this sort of channels the operators can be recombined into $\sqrt{q}I$ and $\sqrt{1-q}Z$ ($q = (1+\sqrt{1-p})/2$), relating them to the phase flips.

$$\begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix} \quad ; \quad \sqrt{p}\begin{pmatrix} 0 & 0 \\ 0 & \sqrt{p} \end{pmatrix}$$

2.3.3 Quantum error-correction

Quantum error-correcting codes (QECC) are one of the major factors enabling the discussion of large-scale quantum computers. The optimistic perspective, that such computing systems are possible to construct, is based on the observation that quantum state errors can be discretised, although these seem to form a continuum. It was shown, independent of the assumed quantum channel that error correction could be adapted to fit the chosen noise model[NC10]. If a QECC code corrected errors for a particular channel (e.g. bit flip), the same code could be adapted to correct errors for another single-qubit channel (e.g. amplitude damping).

Errors are not ultimately represented only by unitary operators; it is possible to express them using the operator-sum, where the error operations $\{E_i\}$ are normalised according to the completeness relation. For a single qubit, each E_i is a square complex matrix representing a linear combination of the Pauli matrices.

$$E_i \quad = \quad e_{i0}I + e_{i1}X + e_{i2}Z + e_{i2}Y$$

Performing an X, Z or Y measurement on the state affected by E_i will collapse the state to one of the four states $|\psi\rangle$, $X|\psi\rangle$, $Z|\psi\rangle$ or $Y|\psi\rangle$, and the recovery is performed by applying the inverse of the corresponding Pauli gates, resulting in the final state $|\psi\rangle$. Therefore, quantum errors can be discretised and quantum error correcting codes are required to protect only against bit (modelled as X gate applications) and phase errors (modelled as Z gate applications).

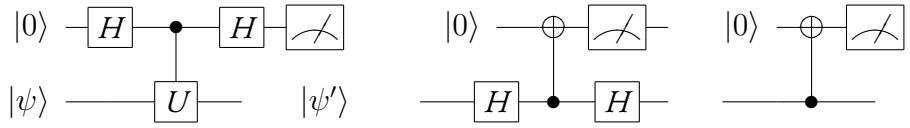


Figure 2.14: The three circuits are used for: a) Measuring an arbitrary unitary U; b) Measuring the X-operator; c) Measuring the Z-operator.

Parity checkers

In general, the *direct* measurement of a qubit leaves the output state changed, but it possible to perform the measurement of a unitary operator[NC10]. This measurement has the advantage of leaving the quantum state in a known eigenstate of the operator. The affected state is measured remotely, and the procedure is similar to the remote application of gates. The measurement result is stored on an ancilla, while the measured qubit will be available for future computations. The measurement result indicates the effect of the transformation on the output state, and this effect is used to determine if a state is erroneous: the ancilla state indicates the error syndrome of the output qubit. At the same time, this kind of remote measurement is used to initialise an unknown state into one with a known stabiliser.

Parity checkers are circuits which perform the same measurement on a set of qubits and output the measurement results parity. The *phase parity* of a single qubit is an illustrative example, and the computation is reduced to inferring the sign of the relative phase of the state $|\psi\rangle = |\pm\rangle$. The result is either $|+\rangle$ or $|-\rangle$. For this the following approach is available: 1) use an ancilla qubit initialised into $|0\rangle$; 2) apply an H gate to $|\psi\rangle$; 3) perform a CNOT gate controlled by $|\psi\rangle$ targetting the ancilla; 4) transform back $|\psi\rangle$ with another H; 5) perform Z-measurement on the ancilla. The measurement of the ancilla will be $|1\rangle$ if $|\psi\rangle = |-\rangle$, and $|0\rangle$ otherwise. But $|\pm\rangle$ are the eigenvectors of X, and, after the transformation rules from Figure 2.10 are used, the circuit from Figure 2.14 is obtained. The measurement of an arbitrary unitary Hermitian operator U is a generalisation of the previous example.

A first application of parity checkers is in the context of error detection. The concept is similar to classical bit parities, the only difference being that for quantum states two types of parities are possible: bit and *phase parities* (the sign of the relative phases is considered). The circuits act similarly to majority voters, and the existence of a possible bit or phase flip error is concluded based on the syndrome (measurement result). For example, the bit parity of $|010\rangle$ is odd ($|1\rangle$), and the phase parity of $|--+\rangle$ is even ($|0\rangle$).

Due to their functionality, the parity checker circuits force a state potentially affected by the error operator E_i to *decide* if the state is erroneous. This will be shown in the following section for a simple example, after the repetition code is introduced. A bit parity checking circuit will measure the Z operators of the investigated qubits, whereas a phase parity circuit will measure X operators.

Bit parity checking circuits can be represented as the measurement pattern of specific graph-states. Measuring the parity of n qubits is possible in a star-shaped graph with n sides

and a central qubit representing the ancilla that will store the measurement result. The central qubit in the graph structure will be X-measured (see the CPHASE transformation). For the graph from Figure 2.7a having the stabiliser table from Table 2.8, the measurement of qubit 3 will result in the Z-parity of the qubits 1,2,4,5 (qubit 3 is stabilised by $X_3Z_1Z_2Z_4Z_5$). This observation is also consistent with the *VOP* update rules of X-measurements.

The applications of parity circuits can be extended beyond the measurement of operators. It is possible to use them as circuits that apply transformations on an arbitrary state. In particular, the circuits are useful for the general stabilisation of states by the unitary U.

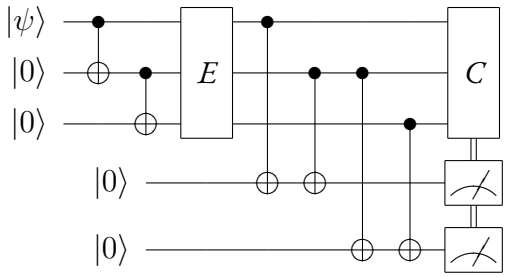
After the measurement of U, the state $|\psi\rangle$ is left into an eigenstate of U, and this is equivalent to projectively measuring the state $|\psi\rangle$ to one of the eigenstates of U. For example, the Bell pair B_{00} stabilised by the $\{X_1X_2, Z_1Z_2\}$ stabilisers can be obtained starting from the $|00\rangle$ state. The state $|00\rangle$ is already an eigenvector of the Z_1Z_2 operator, and the measurement of the operator will leave the state unchanged. Measuring the X_1X_2 operator will leave $|00\rangle$ stabilised by X_1X_2, and the final state is a simultaneous eigenstate of the two stabilisers (up to a single phase correction). Parity circuits will be used further in this work to set a multiqubit state into an eigenstate of the multiqubit unitary U.

Repetition code

The classical repetition code was used as the first QECC to protect against bit flips[NC10]. The repetition code is not a full QECC, due to its structure which cannot protect against both bit and phase flips. Encoding is performed by entangling the state $|\psi\rangle$ with two ancillae (see Section 2.1.2) initialised into $|0\rangle$. For $|0\rangle$ the resulting codeword is $|0_l\rangle = |000\rangle$ and for $|1\rangle$ the codeword is $|1_l\rangle = |111\rangle$. The repetition code can protect against a single bit flip, and detection of errors is performed by majority voting. However, error correction and detection works as long as the individual bit flip probability is $p < 1/2$.

Errors are detected by performing *syndrome diagnosis*, and the value of the syndrome indicates the required correction. The two bits of information required for the syndrome are obtained by performing two parity measurements: Z_1Z_2 and Z_2Z_3. During error detection the results of the measurements are compared. If Z_1Z_2 returns 0, it means that the Z-parity of the qubits 1 and 2 is even: they both are either 1-eigenstates of Z ($|0\rangle$), or both -1-eigenstates ($|1\rangle$), and otherwise the bits are opposite and a flip must have happened. The same applies when the qubits 2 and 3 are measured. The error syndrome is established after comparing the parities of the two measurements. After detecting the fliped qubit, its state is corrected by applying an X-gate. In Figure 2.15a, the codeword of the encoded state $|\psi\rangle$ is affected by an error E, and after the syndrome was computed using the two parity checkers, the output state is corrected by C (from Table 2.15b).

The repetition code is useful for the protection against phase flips, too. If a state is protected by the repetition code against bit flips, then the Hadamard linear transformation of the codeword will protect against phase flips. Applying a Hadamard gate to a computational basis state will encode the phase of the input state into the basis states of output, and



(a)

Z_1Z_2	Z_2Z_3	State	Correction
0	0	$a\|000\rangle + b\|111\rangle$	$I_1I_2I_3$
0	1	$a\|001\rangle + b\|110\rangle$	$I_1I_2X_3$
1	0	$a\|100\rangle + b\|011\rangle$	$X_1I_2I_3$
1	1	$a\|010\rangle + b\|101\rangle$	$I_1X_2I_3$

(b)

Figure 2.15: The repetition code: a) The encoder of the circuit, followed by the error E, the syndrome measurement and the controlled correction C; b) The four possible syndrome measurement bits, the corresponding projected output state and the necessary corrections.

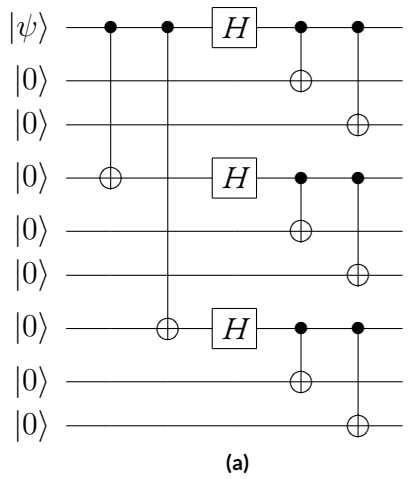
instead of protecting against bit flips, the code protects against phase flips. The logical computational basis states are transformed, too, into $|0_l\rangle = |+++\rangle$ and $|1_l\rangle = |---\rangle$, and the necessary syndrome measurements will be X_1X_2 and X_2X_3 (see Table 2.15b, where Z_1Z_2 and Z_2Z_3 are transformed according to Table 2.6).

A possible phase flip Z^f, for $f \in \{0, 1\}$, affecting the relative phase of the state $|\psi\rangle$ results into the multiplication of $(-1)^f$ to b. The Hadamard transformation of $|\psi\rangle$ transforms Z^f into X^f, thus the resulting state will be $(a - b)\,|0 \oplus f\rangle + (a + b)\,|1 \oplus f\rangle$, where $\oplus$ is the classical Boolean *XOR* operation. A bit flip affecting the Hadamard transformed state results, for $f = 1$, into the erroneous state $(a - b)\,|1\rangle + (a + b)\,|0\rangle$.

$$
\begin{aligned}
|\psi\rangle = a\,|0\rangle + b\,|1\rangle \quad &\rightarrow \quad \frac{1}{\sqrt{2}}((a+b)\,|0\rangle + (a-b)\,|1\rangle) \\
Z\,|\psi\rangle = a\,|0\rangle - b\,|1\rangle \quad &\rightarrow \quad \frac{1}{\sqrt{2}}((a-b)\,|0\rangle + (a+b)\,|1\rangle) \\
Z^f\,|\psi\rangle = a\,|0\rangle + (-1)^f b\,|1\rangle \quad &\rightarrow \quad X^f \frac{1}{\sqrt{2}}((a-b)\,|0\rangle + (a+b)\,|1\rangle)
\end{aligned}
$$

Threshold theorem and code concatenation

Practical error-corrected quantum computation is enabled by the *threshold theorem* which states that a quantum circuit containing $p(n)$ gates may be simulated with probability of error at most e using $O(poly(\log\, p(n)/e)p(n))$ gates on hardware whose components fail with probability at most p, provided p is below some constant threshold, and given reasonable assumptions about the noise in the underlying hardware[Got09]. This means that it is possible to make an error-corrected *logical quantum computation* to perform arbitrarily close to correct (error-free computation) with an overhead that is polylogarithmic in the length of the initial computation.



1	2	3	4	5	6	7	8	9
Z	Z	I	I	I	I	I	I	I
I	Z	Z	I	I	I	I	I	I
I	I	I	Z	Z	I	I	I	I
I	I	I	I	Z	Z	I	I	I
I	I	I	I	I	I	Z	Z	I
I	I	I	I	I	I	I	Z	Z
X	X	X	X	X	X	I	I	I
I	I	I	X	X	X	X	X	X

(b)

Figure 2.16: The 9-qubit Shor code: a) The encoding circuit; b) The stabilisers of the 9 qubits.

The effect of any QECC is that a supplemental abstraction layer is introduced on top of the non-QECC quantum circuits: *logical quantum circuits* consist of *logical gates* applied to *logical qubits*, which are defined over sets of physical (unencoded) qubits. The difference between a physical and a logical circuit element is that the physical one is not error-corrected, and the logical one is typically composed of multiple physical qubits (e.g. 3 in the case of the repetition code).

The threshold theorem is a direct result of *code concatenation*, where a QECC layer is encoded again into a second QECC layer, and the procedure is repetead for an arbitrary number of times. Re-encoding existing codewords with the same or different code will, in theory, reduce the error probability e. For example the Shor code is a concatenated code, where the repetition code is used in a first layer against phase errors, and in a second layer against bit errors. The construction of concatenated codes reduces the error rates associated with the topmost logical layer double exponentially, while the number of required qubits increases slower if the error-rate at the lowest layer is below a certain threshold (*threshold condition*)[DMN13], such that logical circuits do not generate an exponential resource consumption in terms of physical qubits (lowest layer, unencoded).

Shor code

The first QECC that could be used to detect and correct an arbitrary single error was presented by Shor, and is based on the repetition codes. During a first step, the $|\psi\rangle$ state is encoded against phase flips, and in a second step, each of the three resulting qubits is again encoded against bit flips.

For the Shor code, due to its structure, where $|\pm\rangle$ is encoded as $(|000\rangle \pm |111\rangle)2\sqrt{2}$, the

codewords are:

$$|0_l\rangle = \frac{(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)(|000\rangle + |111\rangle)}{2\sqrt{2}}$$
$$|1_l\rangle = \frac{(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)(|000\rangle - |111\rangle)}{2\sqrt{2}}$$

Errors are detected by checking for bit flips in each tuple of three qubits, and for phase flips in each pair of 3-qubit tuples. The procedure is very similar to how the repetition code was used for error checking and correction. For the encoding circuit from Figure 2.16a, considering the qubits numbered from the top, Table 2.16b contains the necessary error detection measurements. Checking for phase flips (in any of the qubits encoded against bit flips) is performed by measuring $(X_1X_2X_3)(X_4X_5X_6)$. Measuring any operator from the table, for example Z_1Z_2, will leave the encoded state into an eigenstate of the measured operator. It can be concluded that the operators used for syndrome computation are the codeword stabilisers of the Shor code.

In general, a QECC that uses n qubits to encode k qubits with Hamming distance d is written as $[[n, k, d]]$ (double square brackets denote quantum codes). The Hamming distance is the number of positions at which two codewords are different. The repetition code has distance 3, and there are three opposite phase signs between the codewords 0_l and 1_l of the Shor code; as a result, the Shor code is a $[[9, 1, 3]]$ code, and the number of errors that it can correct is $t = (d - 1)/2 = (3 - 1)/2 = 1$.

Stabiliser codes

After the discovery of the Shor code, it was observed that a special class of QECCs can be constructed from classical error correcting codes. The quantum extended classical $[7, 4, 3]$ Hamming code (7 bits used to encode 4 classical bits) is used to protect against both bit and phase flips, if the generator matrix of the code is transformed into a stabiliser table. The classical Hamming code is linear, meaning that its codewords are linear combinations of the generator matrix rows. In the quantum domain, if each 1 in the generator matrix is replaced by a Z and each 0 by an I, the obtained stabilisers can be used to check the bit parities of the codewords, similarly to how one would proceed to check if a classical codeword was in the generators of the Hamming code. For the same Hamming generator matrix, replacing each of the 1 entries with X and 0s with I will result in a QECC that corrects phase errors.

The previous two constructions use 7 qubits to encode 1 qubit with a distance-3 code, and the resulting quantum code is a $[[7, 1, 3]]$ *stabiliser code*.

In general, stabiliser codes are constructed as *CSS* codes (named after Calderbank, Shor and Steane), by using two different classical codes C_1, C_2 for the bit and phase protection. The previous 7-qubit code is the first CSS code presented, although it uses the same classical Hamming code ($C_1 = C_2$ are given by the generator matrix of Table 2.5). For general CSS

1 1 1 1 0 0 0	Z Z Z Z I I I	X X X X I I I
1 1 0 0 1 1 0	Z Z I I Z Z I	X X I I X X I
1 0 1 0 1 0 1	Z I Z I Z I Z	X I X I X I X
(a)	(b)	(c)

Table 2.5: a) The left part of the table represents the generator matrix of the classic $[7, 4, 3]$ code. b) Replacing the 1s and the 0s in the matrix results in the stabilisers of the bit flip protection code; c) The stabilisers of the phase flip protection code. Tables b) and c) are the generators of the $[[7, 1, 3]]$ code words.

codes, the construction condition is that $C_2^{\perp} \subseteq C_1$, where $C_2^{\perp}$ (dual code) denotes the codewords orthogonal to the ones from C_2. The Hamming code is self-dual. If C_1 is a classical $[n, k_1, d_1]$ code and C_2 is a $[n, k_2, d_2]$ code, then the corresponding quantum code will be a $[[n, k_1 + k_2 - n, \min(d_1, d_2)]]$ code[Goto9]. Error correction is performed by measuring the Z-generators of the C_1 code and the X-generators of the C_2 code. All the codewords have to satisfy the parity checks of the C_1 code, and at the same time the codewords will be superpositions. Simoultaneously, the X generators of the $C_2^{\perp}$ code add a word to the state, and the codewords of the CSS code are of the form:

$$\sum_{w \in C_2^{\perp}} |u + w\rangle \,, \text{ where } w \in C_1$$

The generators of a stabiliser code form a group $S \subset G_n$ (G_n is the Pauli group on n qubits), and all the n-qubit states (codewords) that are stabilised by a generator $g \in ST$ are 1-eigenstates of the unitary g ($g|\psi_{1,n}\rangle = |\psi_{1,n}\rangle$). A stabiliser QECC using n qubits to encode k qubits has $n - k$ generators, and the codespace is of size 2^k. Every stabiliser of a state splits the Hilbert space into two equal subspaces (1 and -1 eigenspaces). In general, for an n-qubit circuit the corresponding Hilbert space has a dimension of 2^n, but the $n - k$ generators reduce the number of possible states useful as codewords. Similarly to stabiliser circuits, the normaliser $N(ST)$ of the code ST is defined as the set containing all the $E \in G_n$ which commute with the generators $gE|\psi\rangle = Eg|\psi\rangle = E|\psi\rangle$.

Logical operators acting on codewords, $O \in N(ST) \setminus ST$, transform a codeword into another valid codeword (1-eigenstate). If one wishes to define two logical operators as X_l and Z_l, then these will be required to anticommute: $X_l Z_l = -Z_l X_l$. Defining X_l and Z_l automatically results in the definition of $Y_l = X_l Z_l$, while the logical identity operator I_l is the tensor of I on all qubits. The logical Pauli operators will span the logical space of the codeword. This observation will be used again in this chapter when logical operators based on other QECCs are defined. For the 7-qubit code, the logical operators are commuting anyway with the generators $g \in S$, and, for example, the logical X is $X_l = X_1 X_2 X_3 X_4 X_5 X_6 X_7$, and the logical Z is $Z_l = Z_1 Z_2 Z_3 Z_4 Z_5 Z_6 Z_7$.

For stabiliser codes, the number of correctable errors is given by their distance (similar to classical Hamming distance), which is the minimum weight of the elements from

$\mathcal{N}(ST) \setminus ST$. The weight of a tensor product is the number of terms not equal to identity (e.g. $weight(X_1 I_2 Z_3) = 2$)[NC10].

Furthermore, detectable errors are $E \in G_n \setminus \mathcal{N}(ST)$, and after detecting an error the code needs to correct it. Two errors E_a, E_b acting on two different codewords $|\psi_a\rangle, |\psi_b\rangle$ should result in orthogonal logical states (the theory of stabiliser codes allows I as a possible error): $\langle\psi_a| E_a^\dagger E_b |\psi_b\rangle = 0$. This observation implies that correctable errors need to be distinguished first. For an arbitrary stabiliser code, all the errors that are not in the normaliser of the code are distinguishable and correctable.

From a practical perspective, the errors E are detected by measuring the generators of the code ($gE|\psi\rangle = -Eg|\psi\rangle = -E|\psi\rangle$), and using the measurement results for computing error syndromes. Error E affects a state, if, after measuring the generator g (in Figure 2.14, replacing U with g) the result will be $|1\rangle$. A complete syndrome is computed by measuring all the generators of the code. For example, for the repetition code, considering the generators $\{Z_1Z_2, Z_2Z_3\}$ and a state affected by $E = X_2$, the measurement of the generators will indicate qubit 2 if the measurement results is $|1\rangle\,|1\rangle$.

2.3.4 Fault-tolerant design

Quantum computations can be protected against errors using QECCs, and the key idea behind fault-tolerant designs is to directly work on encoded information without decoding it (see Chapter 1). There are three elements in the execution of a circuit that need to be protected against faults: initialisations, application of gates and measurements. In the previous section the construction of logical states (logical qubits) was introduced, and in the following the fault-tolerant construction of the three other elements is presented.

The main issue to address in the design of fault-tolerant computations is the propagation of errors, which is an effect of gates acting on more than a single qubit. If the universal gate set $\{CNOT, H, P, T\}$ is implemented in a fault-tolerant manner, then the gate that can propagate errors is the CNOT. For example (see Table 2.4), a bit flip on the control qubit is transformed into two bit flips, one on the control and another on the target qubit. At the same time, a phase flip of the target is transformed into phase flips on both the control and the target.

Due to the propagation of both bit and phase flips, the CNOT will receive a special attention. The CNOT is an important gate as it is used in state initialisation circuits, where the remote measurement of an operator is equivalent to initialising the remote qubit into a logical state (see Figure 2.14). Moreover, CNOT is used for correcting erroneous encoded states (see Figure 2.3). Finally, CNOT is also used as an entangling gate in quantum circuits.

A fault-tolerant element is one where errors are not propagated *beyond control*. The main criterion in the design of fault-tolerant circuit elements is, according to[NC10,DMN13], for a single error in the input logical qubits to cause at most one error in each logical output qubit, if the used code has distance 3. The criteria indicates that errors are allowed to propagate, but

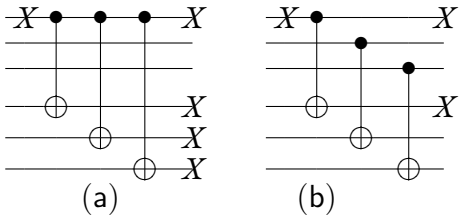


Figure 2.17: The effect of error cascading is illustrated using two logical qubits encoded on three qubits (e.g. repetition code): a) The X error on the control is cascaded to all three qubits of the logical target; b) The X error on the control propagates to a single qubit of the logical target qubit.

within limits. Using a distance-3 code, every logical qubit is error-protected and at most one error is corrected, and at most one error is tolerated.

Errors propagated beyond the capabilities of the QECC cannot be corrected anymore. For example, an error propagation that violates the criteria is *error cascading* as presented in Figure 2.17, where for two logical qubits (consisting of 3 physical qubits) it is assumed that the two circuits perform the same computation. The first circuit propagates three bit flips to the second codeword, while the other circuit propagates only one bit flip. Whereas, for the first circuit the three flips cannot be corrected, for the second circuit, it is possible to correct the codewords.

Fault-tolerant gates

are used in fault-tolerant circuits. Such gate constructions are QECC specific, and in the following the classical example is maintained by deriving fault-tolerant gates for the 7-qubit code (see Table 2.5). The code is a stabiliser code, and the logical operators corresponding to the Pauli gates are generated by the gate set $\{H, P, CNOT\}^{\text{NC10}}$, and the first gate constructions are those for the normaliser operations. From the perspective of a logical layer, it will be required to implement a logical Hadamard gate, a logical phase gate, and a logical CNOT gate.

Fault-tolerant quantum gates are easily constructed if *transversality* is used, which guarantees that a single failure anywhere in the encoded gate introduces at most one error per codeword. This will force the error probabilities not to grow out of the control of the QECC$^{\text{NC10}}$. The transversal logical gate G_l is the construction where G is applied on all the physical qubits of the codewords. For example, the bitwise application of the H gate will transform a logical qubit stabilised by X_l into a logical qubit stabilised by Z_l:

$$H_l X_l H_l^\dagger \quad = \quad (H_1 X_1 H_1^\dagger) \ldots (H_7 X_7 H_7^\dagger) = Z_1 \ldots Z_7$$

For the 7-qubit code, the Hadamard, the CNOT and the P gates are implementable in a transversal manner (see Figure 2.18), but for an arbitrary QECC this is not always possible. For the codes presented in this work the T gate will lack such a construction.

Error-corrected universal quantum computation is based on the fault-tolerant constructions of the Clifford gates and a fault-tolerant implementation of the T gate. The solution

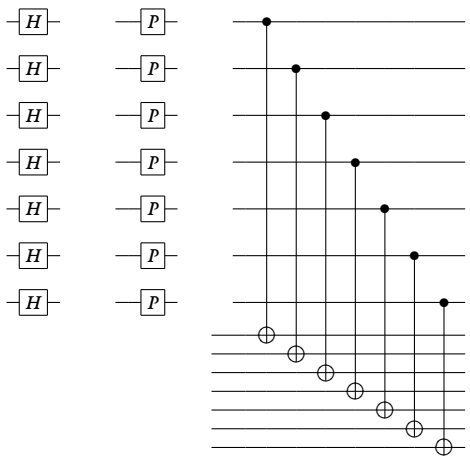


Figure 2.18: Transversal implementations of the H, P and CNOT gates.

for the fault-tolerant T gate is achieved using the teleportation circuits from Figure 2.5 (the contained CNOT is transversal).

The main issue with using magic states for the construction of fault-tolerant rotational gates is that the states themselves may be affected by errors. The initialisation of a qubit into a magic state needs separate attention for a fault-tolerant design of gates. It was discovered that these states can be *distilled* using encoding/decoding circuits[FMMC12,NC10]. Distillation is performed by taking multiple instances of magic states, applying a quantum circuit on the states, and outputing a single magic state that has a lower probability of error. For example, the distillation circuit of the $|Y\rangle$ state is the decoder of the $[[7, 4, 3]]$ QECC[FMMC12]. This procedure is somehow similar to code concatenation, where a supplemental code (the distillation procedure) is used to lower the error-probability associated to a given codeword. As with concatenation, the distillation procedure can be repeated infinitely, constanly reducing the error-probability.

Fault-tolerant initialisation and measurements

are based on the parity checker circuits. The previously considered circuits contained a single ancilla qubit initialised into $|0\rangle$, which controlled the application of the operators on the codewords, and, as illustrated in Figure 2.17, this is not a fault-tolerant construction.

Fault-tolerance is achieved by protecting the initialisation of the qubits against errors, reducing the propagation of errors, and finally, by reducing the error probability associated with the measurements. The circuit in Figure 2.19 depicts the following approach.

The ancilla qubit used for storing the eigenvalue of the measured operator U was prepared in the circuit from Figure 2.14 into the $|0\rangle$ state, which was immediately transformed by a Hadamard gate resulting into the superposition $|+\rangle$. In order to protect the ancilla initialisation, the repetition code will be used and the number of ancillae is increased. Ac-

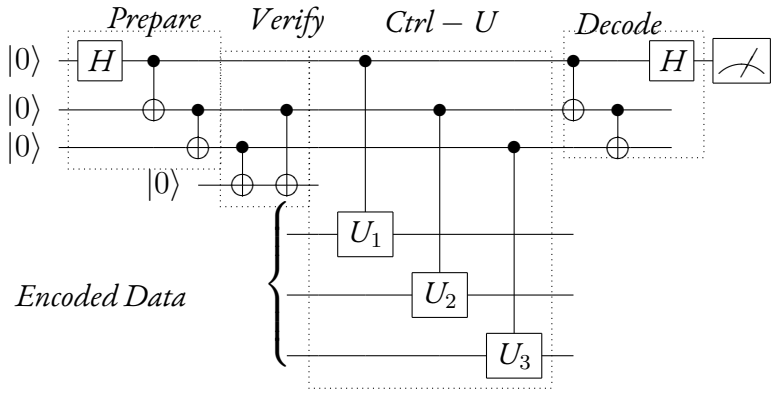


Figure 2.19: Schematic procedure for fault-tolerant measurement of $U = U_1 U_2 U_3$, which is performed on encoded data [NC10]. Although not depicted, the *Verify* step is repeated three times. The complete circuit is repeated three times, too. Three repetitions of a circuit element with error-probability p followed majority voting between the outputs reduce the probability of error to p^2.

cordingly, the encoded state will have to be the superposition $|0 \ldots 0\rangle + |1 \ldots 1\rangle$, and this is constructed by applying a Hadamard on the first ancilla, followed by subsequent CNOTs arranged such that errors will not cascade. At this stage the possible errors stem from the initialisation of the ancillae. After employing an additional parity checking circuit, the encoded state can be validated, and, if necessary, corrected. Even the parity checking could be faulty with probability p, and the step is repeated three times (majority voting) effectively reducing the error probability to p^2.

The fault-tolerant measurement of the operator can at this point be performed. Due to the structure of the encoded ancilla state, the application of U will be controlled by each of the ancillae. Again the choice is motivated in order to reduce error propagation.

After the measurement, the encoded ancilla will have to be decoded using the reversed encoding circuit. Errors in the encoded information could affect the last measurement, and the complete circuit will have to be executed three times. The error probability is exponentially reduced again, after choosing the correct result through majority voting.

2.3.5 The surface code

The class of stabiliser codes contains also the *surface code*. The code was first introduced by Kitaev as the *toric code* [Kito3], where the error-correction procedures were imagined as being defined over a lattice with periodic boundaries. For a lattice with open boundaries, the surface code is the planarised version of the toric code.

The high threshold of the surface code made it very atractive for large-scale quantum computation (see threshold theorem) and an ideal starting point for extending the MBQC computational model by error-correction. In the following descriptions the basics of the surface code are based on [FMMC12,Kito3].

The lattice used by the surface code is rectangular and represents qubits arranged ac-

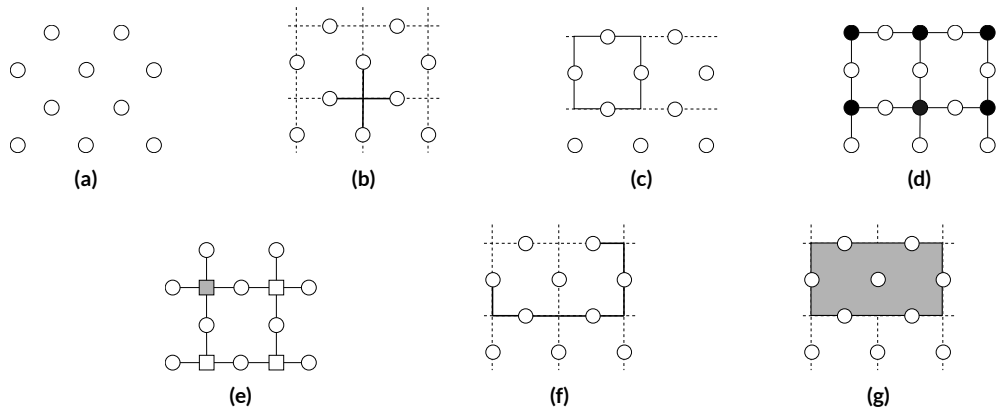


Figure 2.20: Elements of the surface code: a) Physical qubits organised as a two-dimensional lattice; b) A vertex stabiliser; c) A face stabiliser; d) The measure-X qubits are interspersed between the data qubits; e) The measure-Z qubits occuppy the remaining positions of the lattice; f) A stabiliser chain; g) A contractable loop constructed by multiplying two face stabilisers; or a non-contractable loop, if the corresponding measurement qubits from the grey area are not enforced.

cording to a strict pattern, which is indicative of nearest neighbour interactions (see Figure 2.20a). It is not assumed that the lattice is represented by a graph-state (at least not in the 2D version of the code).

The surface code is a stabiliser code, and the stabilisers of the physical qubits are of the form of vertex stars and face boundaries, where a *vertex* (also called site) is the cross arrangement of qubits, and a *face* (also called plaquette) is the rectangular arrangement of qubits. A visual representation of the stabilisers is offered in Figures 2.20b and 2.20c. Each pair of code stabilisers commutes since the number of common edges between a vertex star and a face boundary is either 0 or 2.

$$A_s^{\alpha} = \bigotimes_{j \in star(s)} X_j, \text{ for } s \text{ a vertex}$$

$$B_u^{\alpha} = \bigotimes_{j \in boundary(u)} Z_j, \text{ for } u \text{ a face}$$

The central property of the code is that it is a *local check code*[Kito3]. Every stabiliser involves a bounded number of qubits (at most 4), and each qubit is involved in a bounded number of stabilisers (at most 4). There is no limit on the number of single physical qubit correctable errors, and more errors are corrected by using more physical qubits for logical qubit encoding.

The qubits used for computing the paritites are called *measurement qubits*, and the measured qubits are called *data qubits*. Measurement qubits have a two-fold function. Firstly, measurement qubits are used to initialise the data qubits into given eigenstates of the code

stabilisers, effectively generating the underlying stabilised state. The circuits for measuring the A_s^x and B_u^z operators are presented in Figure 2.21. The circuits are parity checking circuits, and the measurement qubits are called measure-X and measure-Z qubits[FMMC12]. The data qubits are assumed to be initialised into $|0\rangle$ before being stabilised by the application of the parity checkers. An advantage of the parity circuits used in the surface code is that their depth is constant, which results in a reduction of the error checking overhead.

Data and measurement qubits

The Figures 2.20d and 2.20e show the structure of the parity checkers when included in the lattice of qubits from Figure 2.20a. The black vertices represent measure-X qubits, and the grey rectangles represent measure-Z qubits. The edges connecting the parity checking qubits and the data qubits should not be interpreted as CPHASE entanglement bonds, but only indicators of the neigbourhood that is checked.

Each data qubit is coupled to two measure-Z and two measure-X qubits, while each measurement qubit is coupled to four data qubits[FMMC12]. A phase flip on the physical qubit a will be detected by its two neighbouring measure-X qubits, and a bit flip will be detected by the two measure-Z qubits. The states protected by the surface code will be eigenstates of the complete set of face and vertex stabilisers.

The toric code, as initially proposed in[Kito3], supports the definition of only 2 logical qubits. In general, in a square lattice of $k \times k$ faces there are $n = 2k^2$ qubits arranged on the edges of the faces and $m = 2k^2 - 2$ independent stabiliser generators. Following the observation for $[[n, k, d]]$ stabiliser codes (see Section 2.3.3), there are $2^{n-k} = 2^m$ generators of the subgroup $S \subset G_n$, meaning that for the toric code $2^{n-m} = 2^k = 2^2$ stabilisers exist.

Contractable loops and chains

The structure of the local code stabilisers (vertex and face) implies that the normaliser of the surface (toric) code will contain stabilisers of the form X_n and Z_n, which commute with all the code stabilisers. Commuting stabilisers can be constructed by either multiplying face or edge stabilisers, and the results are so-called *loops* (see Figure 2.20g and 2.20f). The *chain* stabiliser from Figure 2.20f is the result of multiplying vertex stabilisers, and a loop could be constructed if the local stabilisers associated to the missing measure-X qubits (at the dotted crossings) were multiplied.

$$X_n = \bigotimes_j X_j \; Z_n = \bigotimes_j Z_j \text{ for } j \text{ physical qubit}$$

A *contractable loop* is a tensor product of local code stabilisers, and contraction is just the observation that the stabiliser is separable into multiple local stabilisers. Such *larger* lattice

stabilisers are in the stabilising set of the code, thus not being possible to use them for encoding information. As it results, for a torus (or a two-dimensional lattice of qubits), only two X_ns and Z_ns are non-contractable loops, meaning that these belong to the set difference $N(ST) \setminus ST$.

These are the stabilisers around the holes of the torus, or, in the case of the surface, the stabilisers connecting the edges of the two-dimensional lattice[Kit03,FMMC12]. The four stabilisers, X_n^1,X_n^2,Z_n^1 and Z_n^2, are the only members from the normaliser not in the stabiliser set. Each X_n anticommutes with the Z_ns, and these are used to encode the *two* logical qubits.

Four logical stabilisers would restrict the applicability of the code: arbitrary quantum computations require more than two logical qubits. Additional logical stabilisers represent additional degrees of freedom, and these are introduced by reducing the number of code stabilisers.

Using the surface code information is encoded and operated on by constantly *repeating* the application of selected sets of measurement qubits. Initially, when all all the measure-X and the measure-Z qubits are used, only the four logical operators mentioned before are existing. However, it is possible to dynamically modify (increase or reduce) the sets of used measurement qubits, and thus the number of applied lattice stabilisers.

Removing a lattice stabiliser is called in the literature *performing/creating a defect*[FMMC12]. The resulting loop-stabiliser around the defect is similar to the one from Figure 2.20g; the difference being that, in the figure, the corresponding measurement qubits were not used for the loop. After not enforcing the associated local qubit stabilisers, the 6 qubits on the boundary of the defect (the grey area) will form at this point a non-contractable loop: the loop is not *separable* into face or vertex stabilisers.

Removing a not neighbouring pair of vertex stabilisers introduces two X_n rings connected by a chain Z_n (see Figure 2.20f), and removing a not neighbouring pair of face stabilisers introduces two Z_n rings connected by a chain X_n. For the purpose of this section it suffices to introduce the logical qubits supported by the surface code. The qubits are of two types: *primal* and *dual*.

A primal qubit has its X_l^p operator defined as the pair of X_n rings, and its Z_l^p operator is a Z_n chain. A dual qubit will have its X_l^d operator defined as an X_n chain and the Z_l^d as the pair of Z_n rings. In contrast to previously discussed stabiliser codes, the surface code encode information by relaxing the stabiliser conditions on the physical qubits, whereas stabiliser codes employ classical encoding circuits.

The surface code supports an arbitrary number of logical qubits, given that enough physical qubits are available. The distance of the surface code is dictated by the minimal weight of the normalisers. Removing a single code stabiliser, the minimal weight will be $d_{\min} = 4$.

The construction of non-contractable loops represented the basis of the logical encoding. Non-contractability is also the key to error-correction. Correction is possible for error-operators E that are not members of the codespace. The detected and corrected errors, given a sufficient code distance, are the ones that have a similar structure to the logical operators:

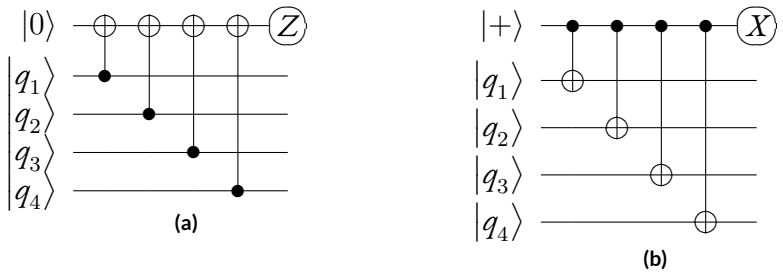


Figure 2.21: The parity checkers of the surface code: a) the $measure - Z$ circuit; b) the $measure - X$ circuit.

non-contractable rings or chains. The error-syndromes are computed using the measurement qubits, and the error correction methods is based on pairing the error locations (affected data qubits). The ends of a non-trivial (non-contractable) chain operators are constructed, and the corrections are applied on the *path* connecting the locations. For the Figure 2.20f it can be considered that an error occured at each ends of the depicted chain. After pairing the error locations, the path connecting the qubits is the non-trivial chain considered as the operator E. Correction is performed by applying single-qubit corrections on the data qubits along E. A much more detailed presentation of the error detection and correction technique is presented in [FMMC12].

The logical CNOT gate

The fault-tolerant gate constructions for the stabiliser codes started by the implementation of the normaliser operations, but due to the code structure, the surface code supports only the CNOT gate. The logical operators of same-type logical qubits (primal-primal, dual-dual) commute, and there is no possibility of interaction between them. However, the ring operators of primal qubits anticommute with the ring operators of the dual qubits ($X_l^p Z_l^d = -Z_l^d X_l^p$), and the same applies to the chains ($Z_l^p X_l^d = -X_l^d Z_l^p$). This aspect is used to construct a primal-dual CNOT gate.

The logical CNOT gate is performed by *braiding* the defect of one of the qubits around the defect of the other logical qubit. Surface *braiding* implies moving a defect (from a start position) around the other defect back to the initial start position. Moving, for example, a primal defect in any direction is possible by disabling a neighbouring measure-X qubit (enlarging the defect, see Figure 2.22b), and afterwards enforcing the previously disabled measurement qubit (reducing the defect, see Figure 2.22c) [FMMC12].

After the movement of a dual defect ($measure - Z$ defect), two possibilities can arise:

- its chain operator is X_l^d, and by braiding it around a primal defect, the resulting loop cannot be contracted (due to the primal defect); as a result, the logical X stabiliser of the dual qubit was *transferred* to the primal qubit (see Figure 2.22f).

- the chain operator of the primal qubit is Z_l^p, but, after the braiding, this logical stabiliser can neither be contracted this time due to the dual defect (see Figure 2.22i).

As a consequence of this phenomenon, the primal-dual logical CNOT is constructed by braiding a primal and a dual logical qubit: the dual qubit is interpreted as the logical control qubit ($XI \rightarrow XX$) and the primal qubit represents the logical target ($IZ \rightarrow ZZ$). Braiding two primal qubits will result in the logical identity: their logical stabilisers commute. The same applies for braiding dual logical qubits: a logical identity results.

It would seem, that it is not possible to perform a logical CNOT between logical qubits of the same type, but it is possible to extend the primal-dual CNOT, and to obtain a primal-primal logical CNOT gate by using the remote CNOT circuit from Figure 2.3. In that circuit the dual qubit is initialised into $|+_l\rangle$ (stabilised by X_l^d). Using other circuit indentities, it is possible to construct a dual-dual logical CNOT, too.

Short logical qubits

The surface code does not support the direct construction of the Hadamard, phase, and T gates. The gates are constructed at the logical layer by using the magic states and the circuits from Figure 2.5. Once a logical T_l gate and a logical $R_{xl}(\pi/2)$ gate are implemented, the other gates are composed from them.

The data qubits were initialised into $|0\rangle$, but in order to achieve computational universality at the logical layer, the data qubits are required to be initialised into at least two other states: $|A\rangle$ or $|Y\rangle$. The process of initialising data qubits into the two rotated states is called *injection*, and the introduction of the magic states into the code is done by the *short logical qubits*[FMMC12].

The short logical qubits are pairs of defects, where two vertex stabilisers are not enforced. The procedure is equivalent to not using two neighbouring measurement qubits. The data qubit at the boundary of the two defects will be Z-rotated, such that its resulting state is either $|A\rangle$ or $|Y\rangle$. Afterwards the defects will be separated, such that the chain and the ring operators are well defined, and the logical $|A_l\rangle$ and $|Y_l\rangle$ exist in the logical layer (see Figure 2.22kl).

Once the injected states are encoded the logical R_x and R_z rotational teleportation-based gates can be applied in order to construct all the other Clifford and non-Clifford gates. This enables universal quantum computation using the surface code.

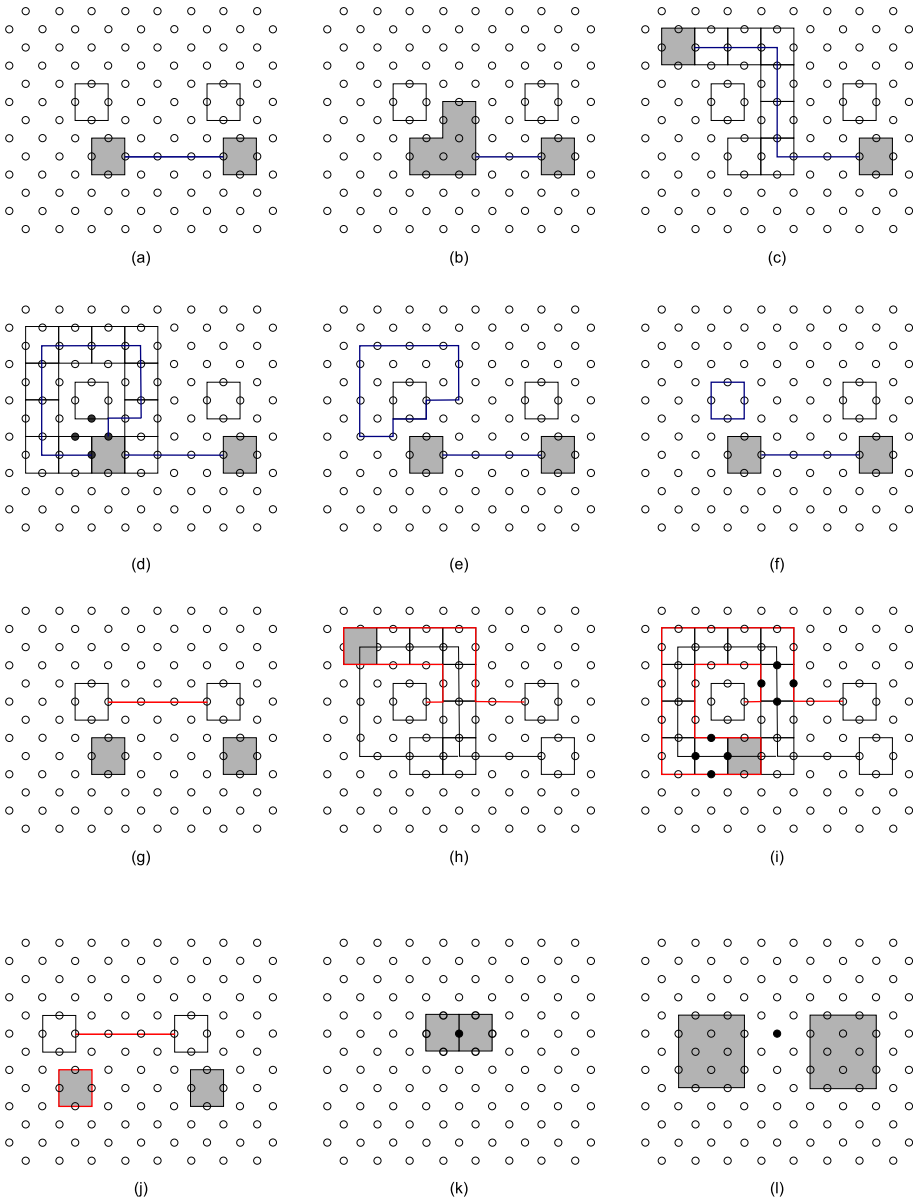


Figure 2.22: Braiding and short qubit construction: a) A primal defect pair (white) and a dual defect pair (grey), where the blue chain indicates the X_l^d; b) Extending one of the dual defects by disabling three corresponding measurement qubits; c) Moving the dual defect around one of the primal defects extends also the distance between the chain endpoints; d) After finishing the dual defect movement the logical X_l^d seems to be consisting of a ring and a chain that start on the same defect; e),f) After considering the effect (tensor multiplication) of the face stabiliser that includes the four black qubits, the previous structure is separated into a non-contractable loop (around the primal defect) and a non-contractable chain (between the dual defects), illustrating thus the logical stabiliser transfer; g)h)i) Performing the same movement of the dual defect the Z_l^p (red chain) of the primal logical qubit is transferred as a ring on the dual defect (the black data qubits correspond to vertex stabilisers); k) The black data qubit will be used for state injection, therefore two neighbouring defects are constructed, the lattice stabilisation procedure stops and the black qubit is rotated using R_z; l) The stabilisation of the lattice is continued, and the defects are enlarged and separated for improved error protection.

3 Probabilistic Circuits

Probabilistic effects existing in classical or quantum circuits can be exploited for performing computations by transforming randomness into a computational primitive similarly to how entanglement works as a primitive for MBQC. The initial work in [VN56] introduced the field of system reliability in the presence of faults, having its starting point at reliable neural networks. The threshold neurons were used to express Boolean logic functions, and through multiplexing and majority voting the influence of randomly faulty neurons was reduced. The aspects presented in that work opened new opportunities which initiated the study of *stochastic computing* [Gai69].

The construction of stochastic circuits starts from the key idea that bit flips should have a minor effect on the overall computation; thus, the effect of the error is mitigated by increasing the number of bits required for the representation of the numbers. The bit flip rate could be arbitrarily high, and an infinitely scalable architecture is required. Classical arithmetic logical units have a fixed maximum number of bits that can be operated on, and this reduces the scalability of the architectures. As a result, the stochastic computational paradigm based on classical gates is used.

Reliability, formulated based on the effects of random faults, was also used as the building block of fault-tolerant quantum circuits which include information teleportation subcircuits. The teleportations are based on the measurement of entangled states, and the output states require the application of correctional gates from the Pauli group G_1 (see Section 2.1.4). For example, the measurement result $|1\rangle$ inside the fault-tolerant P gate signals the XZ correction of the output, while $|0\rangle$ indicates a correct output state.

Furthermore, quantum circuits are inherently probabilistic due to the quantum specific

effects like state superposition and entanglement. For example, the result of measuring a state superposition is directly related to the probability amplitudes of the basis states. The probability of observing the output m is $p(m) = |a|^2 = aa^*$ (see Section 2.1), where a is the amplitude of state m. If the measurement is not performed in a basis equal to the output state, the results will depend on the angle between the output state and the measurement basis.

For the purpose of this chapter Definition 1 is used to express a class of circuits that includes diverse representatives like fault-tolerant quantum circuits implemented using teleportations, arbitrary quantum circuits whose outputs are measured in a basis different from the output state, and stochastic circuits as alternatives to classical circuits.

Definition 1. A probabilistic circuit C computes a mapping $F : I \rightarrow O$, where each input $i \in I$ is mapped to an output $j \in O$ with probability $p(i,j)$ and $\sum_j p(i,j) = 1$.

The analysis of the probabilistic circuits will initially consider the details of simulating quantum circuits on a stochastic computer. Quantum circuits are difficult to simulate due to exponential number of state amplitudes to keep track of and update during the computation, but the similarities between quantum and stochastic circuits will indicate possible improvements of the simulation.

Although not inherently resilient, quantum circuits can be constructed in a fault-tolerant manner by embedding QECCs, and the propagation of errors through the circuits is reduced by the fault-tolerant gates. Reducing the rate of possible qubit errors does not completely adress the effect of missing gate faults. Until recently this aspect has not been investigated, and quantum circuits will be analysed from the perspective of their probabilistic nature.

The final section of this chapter will present an algorithmic solution of the probabilistic Pauli correction mechanism necessary in the context of fault-tolerant quantum computing. Although the mechanisms were previously known, the effects of the corrections were not algorithmically captured before. The algorithm will complement the future tool kit required for the design of probabilistic/quantum circuits.

3.1 Simulation of quantum circuits using stochastic computing

Stochastic computing is a form of approximate computing[Gai69], and the main attraction is that complex arithmetic operations can be implemented by classical gates. *Stochastic circuits* (SC) encode a real number as the signal probability of a bit sequence (the probability of a 1 appearing in any clock cycle), and process such sequences (called *stochastic numbers*) (SN) on-the-fly by means of simple logic circuits.

There are multiple possible representations of probabilities on SNs[Gai69], and the two most used will be detailed. In their basic *unipolar* representation SNs approximate numbers from the real interval $[0, 1]$. For a unipolar SN, the probability associated to it is $p = m/n$,

where m is the number of 1s from an n-bit long bitstream. SNs using the *bipolar* representation are associated to real values from the interval $r \in [-1, 1]$, and the probability of a bit in the stream being one is $p = (r + 1)/2$. For example, the unipolar values 0, 0.5 and 1 correspond to the bipolar values -1, 0 and $+1$, respectively.

With a fixed SN length n, only $n + 1$ distinct numbers of the form m/n are represented exactly because the SN representation is highly redundant in that a given m/n can be represented in C_n^m different ways. For example, 0111, 1011, 1101 and 1110 are the four ways of representing in the unipolar value $p = 3/4$ when $n = 4$; the same value p is represented in 1820 different ways with $n = 16$.

Two important circuits are used to convert numbers between ordinary *binary* form N and stochastic form $p = N/2^k$. These are needed to interface binary and stochastic circuits. The (pseudo-) random number generator in Figure 3.1 is typically implemented by a linear feedback shift register (LFSR)[GWGH82].

Performing arithmetic operations using the SNs is reduced to the way probabilities of the ones in bitstreams are affected by classical gate operations. For the unipolar representation, the multiplication of p_1 and p_2 is equivalent to having a two-input gate that outputs a bit with value one only if two bits from the SNs p_1 and p_2 are one at the same time. This is the AND gate (see Figure 3.1). Squaring a value is not as easy as multiplying the same SNs with itself: due to the functionality of the AND gate, the multiplication result will be $AND(p_1, p_1) = p_1$. One possibility is to delay p_1 into the SN p_1' by placing a D-type flip-flop before one of the AND gate inputs. This uncorrelates the otherwise identical SNs and the multiplication result will be $AND(p_1, p_1') = p_1 p_1' \approx p_1^2$.

There are $n - m$ ($p' = (n - m)/n = 1 - p$) bits of one in the bitstream representing the application of a NOT gate to p, and the output is $p' = 1 - p$. After combining an AND and a NOT gate, the NAND of two unipolar SNs computes $1 - p_1 p_2$. Another often used gate is OR, which calculates an approximate sum of two unipolar SNs. The result of OR is seen either after expressing the gate by NANDs, or if one considers the truth table of the OR function.

$$\begin{aligned} OR(a, b) &= NAND(NAND(p_1, p_1), NAND(p_2, p_2)) \\ &= 1 - (1 - p_1)(1 - p_2) = p_1 + p_2 - p_1 p_2 \end{aligned}$$

A scaled addition between two unipolar SNs is implemented using a 2-input 1-output multiplexer (MUX, see Figure 3.3a), where the select bitstream s represents the 0.5 value and ensures that the sum is in the probability range $[0, 1]$. The scaling factor is supplied by an independent SN representing a purely random bitstream.

$$\begin{aligned} MUX(p_1, p_2) &= OR(AND(p_1, s), AND(p_2, NOT(s))) \\ MUX(p_1, p_2) &= sp_1 + (1 - s)p_2 - (1 - s)sp_1 p_2 \\ (1 - s)s = 0 &\mapsto MUX(p_1, p_2) = sp_1 + (1 - s)p_2 \end{aligned}$$

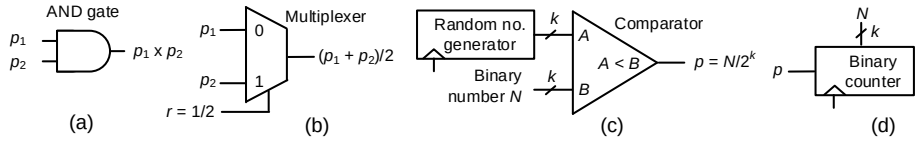


Figure 3.1: Unipolar SC computing elements [PKPH13]: (a) Multiplier; (b) Scaled adder; (c) Binary-to-stochastic converter; (d) Stochastic-to-binary converter.

For bipolar SNs the arithmetic operations are implemented using different gates. The NOT gate will be used to change the sign of the probability associated to the bipolar SN ($x \mapsto -x$), which is observed after considering the linear transformation between SN value and the probability of ones in the bitstream.

$$x = 2p - 1; x' = 2OR(p) - 1 = 2(1-p) - 1$$
$$x + x' = 0 \mapsto x' = -x$$

The multiplication between two bipolar SNs is performed using the XNOR gate. Its action can be easily observed, if for two probabilities p_1 and p_2, the gate initially operates on unipolar SNs, and considering that unipolar values are transformed into bipolar ones by the function $2p - 1$.

$$\begin{aligned} XNOR(p_1, p_2) &= OR(AND(NOT(p_1), NOT(p_2)), AND(p_1, p_2)) \\ &= (1-p_1)(1-p_2) + p_1 p_2 + ((1-p_1)p_1)((1-p_2)p_2) \\ &= (2p_1 - 1)(2p_2 - 1) \end{aligned}$$

Using similar considerations, it is shown that MUX still serves as a scaled adder for two bipolar SNs, and that there is no possibility to implement an approximate addition (like OR for unipolar SNs).

Having enumerated the most common mathematical operations, it should be noted that their results have a high precision as long as the SN lengths are sufficient. The SN length is chosen to control the trade-off between precision (longer sequences) and performance (shorter sequences). In general, stochastic circuits are suitable for resource-intensive applications, that process real numbers, and do not require perfect accuracy. Extremely resource-efficient stochastic implementations of primitive arithmetic operations and several important functions were introduced in early work [Gai69], but general synthesis flows for mapping a specification to a stochastic circuit have only appeared recently [QLR+11,AH12].

SC has several drawbacks, including long computing times, and inaccuracies due to bitstream correlation (e.g. the example of squaring SNs), the scaling of computations to the unit interval, lack of general design methodologies, and few successfully implemented applications. The central engineering aspect that requires special attention when using SC is achieving the necessary precision of computation.

A feature of SC that has not been extensively used is *progressive precision* (PP), which refers to the fact that (random) subsequences of a bitstream representing p have values that

(a) (b) (c) (d)

Figure 3.2: SC edge detection showing progressive precision: (a) Input image; (b-d) Output image after 4, 32 and 256 cycles[AH12].

approximate p. The randomness of SN representation means that some subsequences approximate p very poorly. As more clock pulses are applied to the stochastic number generator of Figure 3.1, the output bitstream's approximation to the target value p tends to get better. In general, each additional bit of probability approximation precision requires doubling the length of the SN, leading to an exponential overhead. To achieve the 8-bit precision, needed for many image-processing applications, the SNs requires length of $n = 256$. Nevertheless, PP implies that, for suitable applications, an SC computation step could be halted after $i < n$ cycles if the result at time i had *sufficient* precision. PP is in most situations difficult to manage (at least for the bipolar representation). The precision is negatively influenced by the scaling introduced after each MUX-addition, and an additional negative factor is the degree of correlation existing between SNs.

In spite of the disadvantages, some very promising new applications for SC have emerged recently, notably decoding LDPC codes[NMSG11] and real-time image processing[LL11]. Systematic approaches to SC design have also recently appeared[AH12]. There are applications of SC, as it can be fast in practice: (1) the basic clock cycle needed for key operations like add and multiply is very short; (2) SC tasks can often be parallelised efficiently using VLSI; (3) the PP property can sometimes be an advantage.

Simulating quantum circuits on a stochastic computer involves the mapping of the quantum circuit formalism to stochastic computing elements. An appropriate mapping will have to result in minimal hardware requirements associated to the mapped circuit, as well as suitable SN lengths for the necessary precision. The later aspect will be investigated from the PP viewpoint.

3.1.1 Mapping of quantum circuits to stochastic circuits

The probabilistic nature of SC coupled with their PP property will be used to analyse the simulation of quantum circuits using stochastic circuits. The simulation will be performed by implementing the multiplication of matrices (the quantum gates) with complex vectors (the quantum states). Generally, both the matrices and the vectors will be composed of complex numbers, and the easiest representation is to use a pair of SNs for each complex entry. The complex number $n = n_r + in_i$ is associated to (n_r, n_i), where the SN n_r represents the real part, and n_i is the SN of the imaginary part. The evolution of an arbitrary

quantum state, described by the multiplication with an unitary matrix, is computed using Equation 3.1, and each complex sum will be represented by the corresponding pair of SNs (Equation 3.2).

$$U|\psi\rangle = \begin{bmatrix} x & y \\ z & t \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax + by \\ az + bt \end{bmatrix} \tag{3.1}$$

$$s = ax + by = s_r + s_i i \tag{3.2}$$

For any quantum state, the square of the amplitudes will add to 1, and for the one qubit state $(a, b)^T$ the sum is written using real and imaginary parts:

$$aa^* + bb^* = (a_r^2 + a_i^2) + (b_r^2 + b_i^2) = 1$$

In a quantum circuit, after the application of gate U, the probabilities associated to each computational basis state will be less than 1; furthermore U is unitary, and the rows form an orthonormal basis:

$$(ax + by)(ax + by)^* \leq 1$$
$$xx^* + yy^* = (x_r^2 + x_i^2) + (y_r^2 + y_i^2) = 1$$

All pairs (n_r, n_i) representing the numbers $a, b, x, y \in \mathbb{C}$ have $-1 \leq n_r, n_i \leq 1$, such that multiplying any such two SNs of the pairs will result in values of the interval $[-1, 1]$. The multiplication result of two complex numbers (e.g. a and x) shows that the simulation of quantum circuits is possible by using bipolar SNs, and that there is no need for scaled additions. However, there are no known methods to implement a generalised non-scaled addition for bipolar SNs, and the multiplexer is still used as an adder.

$$\begin{aligned} ax &= (s_r, s_i) = r_a r_x e^{i\theta_a} e^{i\theta_x} = r_a r_x e^{i(\theta_a + \theta_x)} \\ s_r &= a_r x_r - a_i x_i = r_a r_x cos(\theta_a + \theta_x) \\ s_i &= a_i x_r + a_r x_i = r_a r_x sin(\theta_a + \theta_x) \end{aligned} \tag{3.3}$$

$$r_a^2 + r_b^2 = 1 \text{ and } r_x^2 + r_y^2 = 1 \mapsto -1 \leq s_r \leq 1 \text{ and } -1 \leq s_i \leq 1$$

3.1.2 Cost measures

The state of the quantum circuit needs to be explicitily represented, and simulating quantum circuits using SC exposes an exponential increase of the necessary hardware resources. For example, $|101\rangle$ will be the vector $(0, 0, 0, 0, 0, 1, 0, 0)^T$, where each amplitude is a pair of SNs. During the simulation the complete state vector is constantly updated. However, all the amplitudes (complex numbers) could be computed in parallel, transforming the simulation into a massively parallel computation. For this reason, the total number of gates

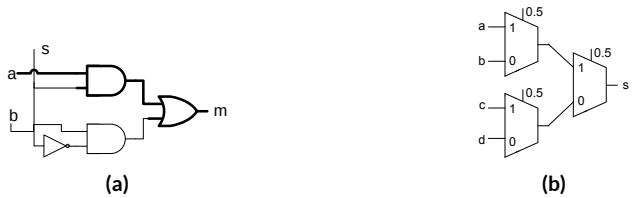


Figure 3.3: a) The multiplexer (MUX) implemented using three gates (two ANDs and one OR); b) A MUX-tree consisting of two layers used for adding four numbers.

N of a stochastic circuit is an expression of the overall hardware resources needed for the simulation (total number of gates N), and its longest path L is an indicator of the required execution time.

Updating the amplitudes of the state vector requires only multiplications (XNOR) and additions (MUX) as illustrated in Equation 3.2. With respect to the arithmetic of real numbers, the gate count and the path length parameters of MUX and XNOR will be used (see Figure 3.3a). For the required bipolar representation the realisation cost of the NOT gate is not considered. Stochastic additions through multiplexers introduce an unwanted scaling of the results, and the scaling factor S will have to be examined.

- $N_a = 3$ and $N_m = 1$ represent the total number of gates required by the MUX and the XNOR;
- $L_a = 2$ and $L_m = 1$ represent the contribution of a MUX and an XNOR to the longest path.

Due to the structure of the input state vectors and of the matrices there will always be an even number of SNs to be added for the computation of the output state. Furthermore, because of the scaled sums, the addition of more than two SNs requires multiple multiplexers to be arranged into a tree structure. Each layer of SN adders from the MUX-tree computes a set of partial sums, which are added into a shorter sequence of partial sums at the following layer (see Figure 3.3b). The partial sums after each tree-layer are scaled by 0.5, and after n levels of addition the scaling will be $S = 0.5^n$. For example, the addition of four numbers a, b, c, d computed by 3 multiplexers introduces a scaling of $S = 0.25$. Adding 2^n $(n \geq 1)$ real numbers using the tree structured addition requires $(2^{n+1} - 1)N_a$ gates, and the length of the path is increased by $\log(2^n)L_a$ gates.

$$\begin{aligned} s &= MUX(MUX(a, b, 0.5), MUX(c, d, 0.5), 0.5) \\ s &= 0.25(a + b + c + d) \end{aligned}$$

For the multiplication of more than two SNs, the multipliers are chained instead of being arranged as a tree. For example, the multiplication of a, b, c is performed by $XNOR(a, XNOR(b, c))$.

Multiplying n ($n \geq 1$) SNs introduces nN_m gates and increases the length of the longest path by nL_m gates.

From an arbitrary bra-ket representation the state vector has to be computed as a tensor product, and a supplemental scaling is introduced at the beginning of the simulation. The scaling factor depends on the number of qubits the circuit operates on.

3.1.3 Arbitrary quantum gates

In an arbitrary quantum circuit on q qubits, applying the general single-qubit gate U on the k-th qubit is equivalent to applying the gate $I_0 \otimes \ldots \otimes I_{k-1} \otimes U \otimes I_{k+1} \otimes \ldots \otimes I_{q-1}$ on all qubits. Similarly, applying the gates U and V simultaneously on the qubits k and t results in applying a tensor product containing I, U and V gates. In the following, the tensor product $I \otimes I \otimes \ldots \otimes I$ is considered a single q-qubit gate, although it is the trivial identity operator. An arbitrary q-qubit gate will be represented by $2(2^q \times 2^q)$ SNs.

The simulated state vector consists of 2^q complex numbers, thus 2^{q+1} SNs, and the vector is updated by computing matrix vector multiplications. A direct implementation of matrix multiplication is generic in that it allows a mapping of any quantum circuit to SC simply by defining enough SNs to represent the state, and then translating each quantum gate into stochastic multipliers and adders that perform matrix multiplication.

An input state vector is multiplied with each row of the matrix, where each vector entry is multiplied with the corresponding row entry. The multiplication of two complex numbers requires two multiplications and additions (see Equation 3.3) for each part of the result. A multiplication requires $2(2N_m + N_a)$ gates. After multiplying the state vector with a matrix row, there are two sets of partial sums, one for the real part and another for the imaginary part. Each partial sums addition is performed by a tree of depth $log(2^q) = q$, which introduces again $(2^{q+1} - 1)N_a$ gates into the simulation. As a result, the matrix multiplication of a an arbitrary gate will introduce N_G gates.

$$N_G(q) = 2^q \left[2^{q+1}(2N_m + N_a) + 2(2^{q+1} - 1)N_a\right]$$

The real and imaginary parts of the output state vector are computed in parallel, and for the tree-shaped adder of depth q the length of the longest path is:

$$L_G(q) = (L_m + L_a) + qL_a$$

Figures 3.4 and 3.5 illustrate the stochastic circuits used for the matrix multiplication and for computing an inner product.

3.1.4 Particular quantum gates

While a direct transformation of quantum gates is expensive, many quantum circuits of interest contain relatively simple quantum gates (Pauli or CNOT), for which efficient stochastic implementations exist.

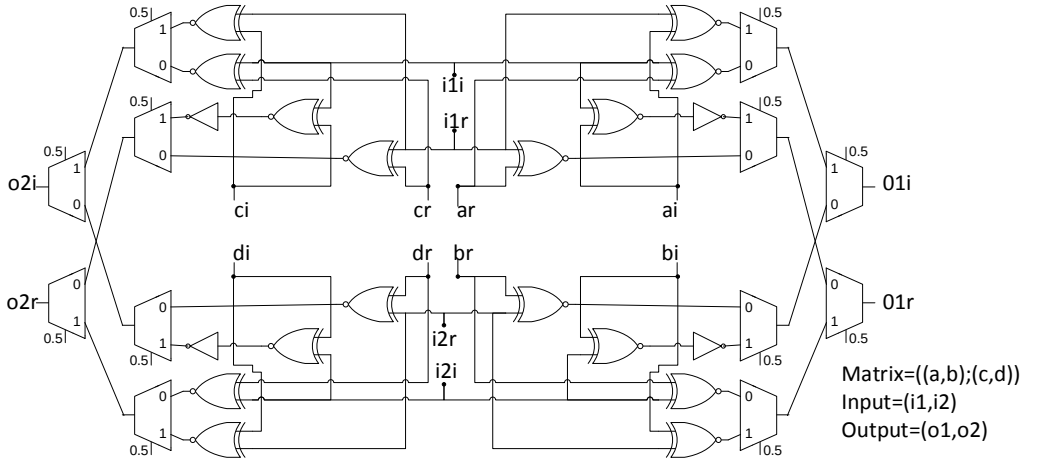


Figure 3.4: The stochastic circuit for matrix multiplication is the same circuit used for the application of a quantum gate to a state vector. In this figure the matrix represents a single-qubit gate.

Particular quantum gates, commonly used in quantum circuits, have the potential, when mapped to SC, to require less classical gates and not to influence the longest path. Pauli gates (X, Y, Z, I), CNOT gate and P gate, when applied to a general state vector perform a permutation of the state vector amplitudes. These gates are not required to be implemented by matrix-vector multiplication. The matrices of the I, X, Y, Z and CNOT gates contain on each line only a single non-zero entry, and an arbitrary tensor product of these gates will result in a matrix with a single non-zero element on each row.

Applying one of these gates in the context of QC simulation using SC requires only NOT gates (for changing the sign of the represented values) and permutations of SNs. For example, simulating the application of the Z gate on a one-qubit state vector $|\psi\rangle = (a, b)^T$ will transform the pair of SNs (b_r, b_i) into $(-b_r, -b_i)$, which is implemented by $(NOT(b_r), NOT(b_i))$.

The matrix of the CNOT gate represents a permutation, too, where the amplitude associated to the $|10\rangle$ (a_{10}) state is permuted with the amplitude of $|11\rangle$ (a_{11}). The CNOT gate is practically simulated by re-routing the affected SNs. For example, for a two-qubit input state vector with the SN pairs $a_{10} = (a_{10r}, a_{10i})$ and $a_{11} = (a_{11r}, a_{11i})$, the output state vector will contain the SN pairs $a'_{10} = (a_{11r}, a_{11i})$ and $a'_{11} = (a_{10r}, a_{10i})$.

The P gate is implemented similarly to the CNOT gate and the Pauli gates. Due to its matrix representation, the multiplication of an amplitude a by the complex number i is implemented $ai = (a_r, a_i)i = (NOT(a_i), a_r)$, where the first position in the SN pair is reserved for the real part, and the second position for the imaginary part.

As a conclusion, the total number of classic gates is not influenced by these particular quantum gates; the longest path and the scaling of the state vector entries remain unchanged.

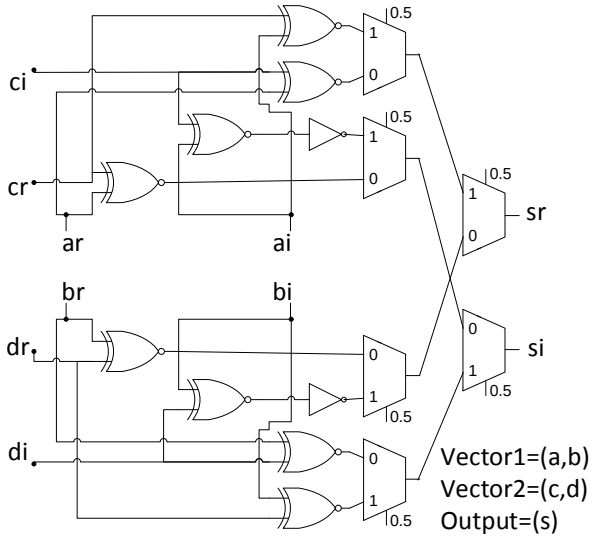


Figure 3.5: The stochastic circuit for computing an inner product between two complex vectors of length 2.

The Hadamard gate

The tensor product between a Hadamard gate and a Pauli gate or CNOT results in a square matrix with each line containing exactly two non-zero entries. In general, the tensor product of h Hadamard gates will contain 2^h non-zero entries.

The parallel application of h Hadamard gates in q-qubit circuits requires $2^q(2(2^h - 1))$ necessary additions for the SC simulation. The longest path is increased by $L_H(h) = hL_a$ gates, and the resulting scaling of the state vector SNs is $S_H(h) = \sqrt{2^{-h}}$. Although each addition introduces a scaling of $1/2$, by omiting the $1/\sqrt{2}$ factor of the Hadamard matrix the resulting scaling is reduced to $1/\sqrt{2}$.

Rotational gates

The last gate from the universal quantum gate set analysed throughout this work is the T gate, the $\pi/4$ Z-rotation. Rotational gates of this type are extensively used in quantum circuits and have the matrix from Equation 3.4, where $r \geq 2$. The matrix representation of the tensor product contains a single complex entry, and, using Euler's formula, the entry is expressed as: a pair of SNs.

$$R(r) = \begin{pmatrix} 1 & 0 \\ 0 & \exp\left(i\pi/2^r\right) \end{pmatrix} \tag{3.4}$$
$$e^{i\pi/2^r} \mapsto (\cos(\frac{\pi}{2^r}), \sin(\frac{\pi}{2^r}))$$

Both the cos and sin functions take values in the $[-1, 1]$ interval that is representable by bipolar SNs. Multiplying a state vector representing q qubits with rotational gates (between

1 and q in parallel) requires a maximum of 2^q complex number multiplications, which adds at most $N_{CR} = 2^q N_m$ gates to the total number of gates, and the longest path increases by $L_{CR} = L_G(q)$ gates. The introduced scaling does not depend on the number parallel gates, such that $S_{CR} = \frac{1}{2}$.

Simulation optimisations

The synthesis of stochastic circuits simulating quantum computations consists of multiple steps: 1) generation of the matrix-vector multiplication circuits; 2) reduction of the number of trivial SN additions and multiplications; 3) introduction of RNGs to generate the parameter SNs from inputs and matrices.

Ensuring that the computation is precise requires the assesment of the randomness requirements. In general, each input SNs will have its own associated random number generator (RNG): for the input-state SNs, and for the matrix entries. The RNGs will ensure that the SNs have desirable randomness characteristics (are uncorrelated). The high hardware cost of introducing a large set of RNGs is not negligible, and a first optimisation is to use a single n-bit LFSR to generate n independent SNs. This reduces the number of RNGs by a factor of n.

Implementing the simulation as a matrix multiplication requires an exponential amount of classical gates in general, although the Pauli gates and the CNOT are trivial to be simulated. The *difficult* gates are the arbitrary gates, and, when using the reduced universal gate set, the Hadamard and the rotational gates. In conclusion, the difficulty of simulating quantum computations is dictated by the number of Hadamard and T gates that appear in the Solovay-Kitaev decomposition of an arbitrary unitary gate[DN06].

The number of gates (N) and the length of the longest path (L) presented previously have to be understood as worst case costs, which can be reduced in certain situations. The results of parallel gate tensor products contain a high number of entries from the set $\{0, \pm 1, \pm i\}$. Multiplication by these values is trivial, and no RNGs, as well as no multipliers, are required to generate the SNs. As a result, matrix multiplications are performed similarly to how Pauli gates were simulated. Futhermore, the 0-entries in the tensor product matrices reduce the number of needed stochastic streams and multipliers.

3.1.5 Simulation results

The simulation of quantum circuits using SC is applied to two types of circuits: stabiliser (GHZn) and non-stabiliser (CRn). The Greenberger-Horne-Zeilinger (GHZn) circuits[NC10] are constructing highly-entangled states (similar to graph-states) of n qubits (see Figure 3.7c). The circuits consist of n gates: a single Hadamard gate on the first qubit and a sequence of $n-1$ CNOTs controlled by the first qubit and targeting each of the other qubits. The smallest GHZ circuits is for $n = 2$, an this is illustrated as a stochastic circuit in Figure 3.6.

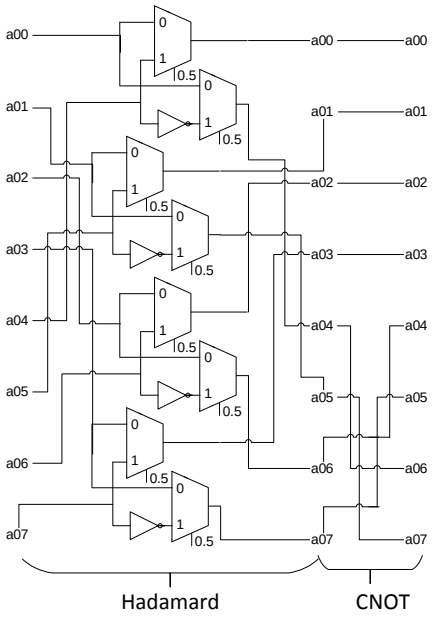


Figure 3.6: The GHZ2 circuit implemented as a stochastic circuit. The application of the Hadamard, after redundant multiplications were removed, is ilustrated in the first part of the circuit. The entangling subcircuit, shown in the second part, consists of CNOT gates implemented as SN reroutes.

The non-stabiliser circuits were synthetically constructed for the simulations, by applying the Hadamard gate in parallel on the n qubits, followed by a sequence of controlled-rotations of gate G; finally, a second round of parallel Hadamard gates is applied on all the qubits except the first (see Figure 3.7b).

The effect of using PP is illustrated for the software simulation of quantum circuits using SC. The results were obtained for the circuit from Figure 3.7a. The circuit operates on two qubits, and the state vector consists of 2^3 SNs (a pair for each amplitude). For the input $|10\rangle$ the diagrams from Figure 3.15 and 3.16 illustrate the evolution of the SN values with the continuous increase (32 bits per step) of the bitstream length. The first diagram shows the SN values for bistreams of length between $[32, 32000]$, while the last diagram contains the values for streams of length $[3.197 \times 10^7, 3.2 \times 10^7]$. The diagrams are discrete, as can be seen in Figure 3.14.

The expected values obtained after simulating the circuit using QuIDDPro[VMH09] are expressed as the SN pairs $((1/\sqrt{2}, 0), (-1/\sqrt{2}, 0), (0, 0), (0, 0))$. However, the stochastic circuit simulation introduces a scaling of 2^{-7} due to the construction of the state vector and the sequence of Hadamard applications. The SN values read at the output approximate 0.00552 (see Figure 3.16).

A second round of simulations was executed on an FPGA[PKPH13]. The target quantum circuit was automatically mapped to a stochastic version in VHDL (a hardware description language). The VHDL description was synthesised and transferred to an FPGA. Further

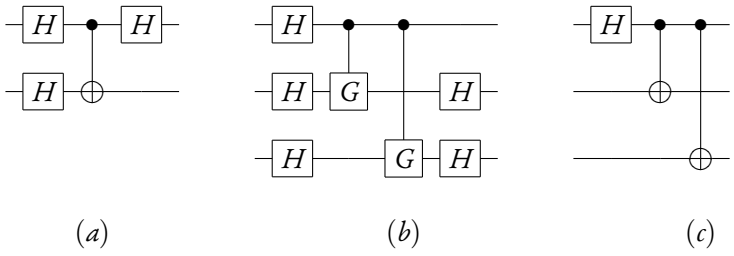


Figure 3.7: Circuits used for simulation: a) Example used for illustrating PP; b) The CRn circuit for $n = 3$ qubits for $G = ((1,0),(0,exp(i\pi/8)))$; c) The GHZ$n$ circuit for $n = 3$.

infrastructure, such as SN generators of stochastic numbers (32-bit LFSRs used as RNGs to generate 32 independent random bitstreams) and counters to convert SNs back to binary, were also synthesized. The Altera DE2-115 FPGA development board and its associated software were used as the development platform, and the FPGA-PC communication was controlled by the Nios II processor, which required several custom-designed communication blocks[PKPH13].

The experimental results are used to compare the resources necessary for both classical and stochastic computing to simulate a quantum circuit. Table 3.1 compares the equivalent number of transistors between classical and stochastic implementations of the benchmarked quantum circuits. The classical implementations were assumed to consist of 16-bit adders and multipliers (1,072 and 7360 transistors, respectively) as well as 32-bit adders and multipliers (2,442 and 29,440 transistors, respectively). The transistor counts for adders and multipliers were computed in[VZO05] and[ZL08], respectively.

The transistor counts of the full implementations of the quantum circuits GHZn and CRn are shown in the second and third columns of Table 3.1. The number of transistors for the stochastic implementations are reported in column 4, while the number of transistors required for the RNGs is given in column 5. Each RNG was estimated to have 408 transistors. The final two columns of the table show the ratio of the transistor counts for the 16-bit and 32-bit classical implementations, and those of the stochastic implementation. For example, the cost of the 16-bit classical implementation of GHZ3 is 1,246,240, whereas the cost of its stochastic realization is 1,600 + 408 = 2,008, implying an improvement factor of 621x. The resource requirements grow exponentially with the number of qubits both for the stabiliser (GHZ) and non-stabiliser (CR) circuits. Moreover, the stochastic implementation of a circuit is around four orders of magnitude more compact than its classical counterpart.

Table 3.2 shows the average error obtained during the simulations for different SN lengths. The trends observed earlier for GHZ3 apply to the other circuits, too. Very close to exact resolution is achieved for 2^{24} long SNs, and a very small deviation is observed for length 2^{16}. However, the circuits are small, and the major obstacle of a succesful simulation is the precision loss introduced by the MUX-addition.

	Classical		Stochastic		Improvement	
	16-bit	32-bit	Gates	RNGs	16-bit	32-bit
GHZ3	1,246,240	4,866,720	1,600	408	621	2,424
GHZ4	6,398,112	25,119,392	8,000	816	726	2,849
GHZ5	32,187,008	126,855,808	39,168	1,224	797	3,141
GHZ6	157,010,176	620,472,576	186,880	2,040	831	3,284
GHZ7	744,857,600	2,949,160,960	872,448	3,672	850	3,366
CR2	589,669	2,281,129	770	816	372	1,438
CR3	2,881,188	11,125,908	3,816	1,632	529	2,042
CR4	13,208,876	51,077,596	17,400	6,120	562	2,172
CR5	59,366,048	230,137,248	77,120	22,032	599	2,321

Table 3.1: Transistor counts of circuits used in the experiments

Circuit	Stochastic number length n					Optimal
	2^8	2^{12}	2^{16}	2^{18}	2^{24}	length
GHZ3	0.03662	0.013337	0.00359	0.0011	0.00019	2^8
GHZ4	0.05078	0.01550	0.00283	0.00155	0.00020	2^{12}
GHZ5	0.05029	0.01242	0.00366	0.00175	0.00014	2^{16}
GHZ6	0.04943	0.01309	0.00293	0.00159	0.00020	2^{24}
CR2	0.03925	0.00762	0.00328	0.00135	0.00015	2^{12}
CR3	0.05176	0.01076	0.00354	0.00188	0.00032	2^{14}
CR4	0.04774	0.01481	0.00384	0.00203	0.00020	2^{18}
CR5	0.04241	0.0126	0.00324	0.00152	0.00017	2^{23}
CR6	0.06257	0.01233	0.00325	0.00148	0.00017	2^{24}

Table 3.2: Errors for different SN lengths

3.1.6 Conclusion

Simulating quantum circuits is known to be challenging. Approaching this problem by mapping QC to SC is performed in a straightforward procedure, and the initial results show that the amount of hardware required to implement the simulation by SC is orders of magnitude less than a classical realisation. However, the main limitation of this approach is that very long run-times are needed to achieve adequate precision. Progressive precision has a great potential in raising the applicability of SC, but more ingenious models of simulating QC need investigation for maintaining a low precision degradation.

3.2 Verification of probabilistic circuits

Quantum circuits were introduced as probabilistic, and were linked to their stochastic counterparts which are resilient in the presence of faults affecting the SNs. The negative impact of qubit or SN errors was reduced by introducing redundancies in the circuits, either by incorporating QECCs or by extending the lenght of the SNs. To a high degree these protections are useful, but there are fault models that demand different approaches. The situation of missing gate faults is well investigated for classical Boolean circuits[BA00], and a common solution is circuit verification. Verifying a circuit implies detecting if a particular circuit instance is conforming to the modelled circuit specification.

The verification task can be translated into the area of probabilistic circuits, as the method for checking that a stochastic/quantum circuit adheres to its specification. The verification of stochastic circuits was introduced in[PAPH11], and for quantum circuits some previous work involved quantum states and quantum processes (computations) tomography, which were formulated starting from the density matrix representation[NC10]. Their major disadvantage is the high computational complexity. The complexity relies on the probabilistic measurements, but equally, on the high dimensionality of the Hilbert space associated to computation. In the following, some options for solving the verification of quantum circuits are presented. The testing and diagnosis of probabilistic circuits was jointly presented in[PAPH11], and extended for quantum circuits in[PPH12].

3.2.1 Quantum state tomography

One of the first problems encountered when assesing quantum circuits is the difficulty of knowing the output state given that the only observable output consists of measurement results. The method of deducing the state of quantum system is *state tomography*, whose goal is to reconstruct the density matrix of an ensemble of particles through a series of measurements[AJK05].

The density matrix is used to express general pure and mixed quantum states. One of its properties is that, for an arbitrary density matrix ρ, the probability of measuring the state $|\psi\rangle$ is given by the trace of multiplication between the outerproduct $|\psi\rangle\langle\psi|$ and ρ. This

procedure is the building-block of tomography approaches.

$$p(|\psi\rangle) \quad = \quad tr(|\psi\rangle \langle\psi| \rho)$$

Quantum process tomography is concerned with characterising the operation of quantum processes, and the procedure is exponentially difficult compared to state tomography [TŘHI13]. In this section only the tomography of states will be shortly introduced. In general, tomographic approaches are based on the concept of N-experiments. Tomographic approaches were initially applied to particles (e.g. photons), but when discussing about quantum circuits, the repeated execution of a circuit is similar to the repeated emission of photons from a photon source. All photons from the same source have identical quantum properties, and an ensemble of n photons should be understood as the output of an n-qubit circuit.

Definition 2. An N-experiment(C, i) consists of repeating for N times the execution of a probabilistic circuit C under the input $i \in I$ and measuring after each execution the outputs of the circuit. The number N is the *length* of the experiment.

Tomography is a probabilistic method, and in practice a quantum state cannot be approximated perfectly because the experiments would require an infinite length to yield the exact probabilities of observing the measured states. This is similar to stochastic computing (see Section 3.1) and its progressive precission property. However, in the following it will be assumed that after a sufficiently long experiment, the probabilities are inferred with a sufficiently high precision.

Single qubit tomography

The state of an arbitrary single qubit is spanned in the density matrix formalism by the four Pauli matrices and four real parameters S_i, where $\delta_0 = I$, $\delta_1 = X$, $\delta_2 = Y$ and $\delta_3 = Z$, and S_0 (corresponding to I) will always equal 1 [AJK05].

$$\rho \quad = \quad \frac{1}{2} \sum_{i=0}^{3} S_i \delta_i$$

Any single-qubit quantum state is specified by three independent parameters associated to three lineary independent matrices. The measurements necessary to determine the S_i parameters correspond to the states $|\phi_i\rangle$, where $|\phi_1\rangle = |+\rangle$, $|\phi_2\rangle = 1/\sqrt{2}|0\rangle + i\,|1\rangle$ and $|\phi_i\rangle = |0\rangle$. Thus, the measurements in the X, Y and Z basis will be used for inferring S_i. The order of measurements is not relevant: tomography works similarly to how a point is located in a 3D space, the only difference being that in a 3D space a point is located on the Bloch sphere (for pure states). The first measurement isolates the unknown state to a plane perpendicular to the measurement basis. Further measurements isolate the state to intersections of non-parallel planes, which, for the second and third measurements, correspond to a line and finally a point [AJK05].

Multi-qubit tomography

For a single qubit the number of parameters required to construct the density matrix is four. For a multi-qubit system the density matrix is expressed according to Equation 3.5, and there is an exponential number of parameters $(4^n - 1)$ to estimate, while $S_{0,0,\ldots 0}$ is always zero. For example, for two qubits, 15 independent parametes have to be estimated.

$$\rho = \frac{1}{2^n} \sum_{i_1, i_2, \ldots i_n = 0}^{3} S_{i_1, i_2, \ldots i_n} \sigma_{i_1} \otimes \sigma_{i_2} \otimes \ldots \otimes \sigma_{i_n} \tag{3.5}$$

In particular, there are some situations where a large number of parameters is zero (e.g. un-entangled/separable states). The density matrix of the state $|00\rangle$ is decomposed into a sum of only four S_i parameters, for instance.

$$\begin{aligned} \rho_{00} &= \frac{1}{2}(I + Z) \otimes \frac{1}{2}(I + Z) \\ &= \frac{1}{4}(I \otimes I + I \otimes Z + Z \otimes I + Z \otimes Z) \end{aligned}$$

3.2.2 Fault models

The reversibility of quantum circuits was used by [PHM04] for formulating the initial problem statement of testing reversible circuits. Quantum circuits were considered consisting of k-CNOT gates (CNOTs with k controls) and the employed fault models were the *single-stuck-at* and *multiple-stuck-at*, which are very similar to the ones used in classical circuit testing [BA00]. For this reason the approach did not capture the complete complexity of the quantum gate fault mechanism. A more recent overview considered fault model presented in [BAP10], and it includes faults of the following type: modelled by Pauli matrices, initialisation faults, lost phase faults, measurement faults and forced gate faults.

Probabilistic circuits containing n gates will be specified as a gatelist (a list of gates, and the connections existing between gates). More specifically, the gatelist of an arbitrary quantum circuit is the sequence $CL = \{G_0, G_1, \ldots, G_n\}$, where G_i represents a gate. In general, a fault is defined by a model that mimics physical defect mechanisms or typical designer errors, and the fault models that seem most adequate from an engineering perspective are those from [PFBH05] which include:

- Single Missing-Gate Fault (SMGF), where the gate G_i is missing from CL;
- Repeated-Gate Fault (RGF): the gate G_i is applied t times. Thus, if t is even, then RGFs are equivalent to SMGFs, and if t is odd, the fault is redundant;
- Multiple Missing-Gate Fault (MMGF), where multiple gates G_i are missing from CL.

Assuming the SMGF model, a probabilistic circuit C is transformed to a different probabilistic circuit C_f by a fault $f \in F$, where the fault list $F = \{f_1, f_2, \ldots, f_{|F|}\}$ is the set of possible faults. Each possible missing gate from the circuit C_0 is the source of a faulty circuit, and $|F| = n$.

The circuit C_0 is used to denote the fault-free circuit, while C_f indicates a faulty circuit where the fault f is present. The faulty circuit is modelled by removing the gate corresponding to $f \in F$ from the fault-free circuit C_0, and C_f has the same input and output space as the correct version C_0. In the case of quantum circuits, the faulty circuit C_f is the gate sequence $C_f = \{G_0, G_1, \ldots, G_{f-1}, G_{f+1}, \ldots, G_n\}$.

The fault-free C_0 and the faulty probabilistic circuits C_f, besides being modelled by gatelists, are represented by the probability distributions obtained at their outputs in the presence of a specified input. The probability distributions $C_j(i)$, for the input $i \in I$, can be computed by software simulation. For quantum circuits, a software circuit simulator is, for example, QuIDDPro [VMH09]. The obtained probability distributions represent the diagonal elements of the density matrix of the output state. A quantum circuit C on n qubits, considering only the $|I| = |O| = 2^n$ inputs in the computational basis, will generate 2^n long probability distributions $C(i) = (p(i,1), p(i,2), \ldots, p(i,|O|))$, and thus 2^{2n} probabilities that relate input states to output states. In testing terms $p(i,j)$ is the probability of measuring j at outputs of C when the test vector i is applied.

3.2.3 Distance measures

Two probability distributions obtained from two separate probabilistic circuits are compared by computing the distance between them. Some of the most used distances are the Euclidean, L_1 (trace distance) and fidelity, which are presented for the two probability distributions a and b:

$$\begin{aligned} \text{Euclidean:} \quad & dist(a,b)_e = \sqrt{\sum_i (a_i - b_i)^2} \\ \text{Trace:} \quad & dist(a,b)_t = \frac{1}{2} \sum_i |a_i - bi| \\ \text{Fidelity:} \quad & dist(a,b)_f = \sum_i \sqrt{a_i b_i} \end{aligned}$$

Distance measures are useful, for example, for comparing the probability distribution of faulty circuits C_f to the output of C_0. The approximation of a quantum state is compared to the *ideal* (expected) quantum state by using similar distance measures where the fidelity of two quantum states is a measure of state overlap [AJK05,NC10]. For two quantum states ρ_1 and ρ_2 the trace distance and the fidelity are defined as:

$$\text{Quantum trace:} \qquad dist(\rho_1, \rho_2)_t^q = \frac{1}{2} tr(\rho_1 - \rho_2)$$

$$\text{Quantum fidelity:} \qquad dist(\rho_1, \rho_2)_f^q = \left(tr(\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}) \right)^2$$

3.2.4 Tomographic testing and diagnosis

This section focuses on determining by a tomographic approach if a *circuit under test* (CUT) C operates correctly or not. *Tomographic testing* is the problem of detecting gate faults into circuits ("is the circuit correct?"), while *tomographic diagnosis* identifies with a high probability the existing fault ("if the circuit is faulty, which particular fault affected it?"). The assumed gate fault model is SMGF. In the following sections the investigated states are assumed to be pure, and the density matrix will be replaced by the state vector representation.

At first, a parallel analysis between stochastic circuits and quantum circuits is conducted based on their probabilistic behaviour (see Definition 1). Quantum mechanics can be thought of as extension of classical statistics dealing with entities called probability amplitudes that behave like probabilities, but are complex rather than real numbers [Aaro4]. Stochastic circuits do not support state superpositions, and, during the analysis, the input and the output states of the quantum circuits are considerd being in the computational basis (stabilised by Z) and measured in the same basis. This restriction does not affect the generality of the analysis, since measurements in an arbitrary basis are constructed by the application of rotational gates followed by Z measurements (see Section 2.1).

Classifying the CUT C as faulty or fault-free cannot be performed by investigating its gatelist, as the circuit is a black box, and the only obtainable information is the output $o \in O$ corresponding to the input $i \in I$. Testing and diagnosing the CUT C against a complete set of modelled fault-free and faulty circuits (represented by probability distributions) starts from the key assumption that only the tomograms (see Definition 3) of N-experiments (see Definition 2) are available to decide whether or not C is faulty.

For quantum circuits, the experimental approach is similar to quantum state tomography, but instead of choosing a large set of measurement-basis to perform the experiments, these are performed only in the computational basis. Arbitrary rotated measurements increase the complexity of the circuit and of the hardware experimental apparatus required for the circuit execution. Such measurements are difficult to perform, while restricting the discussion only to Z-measurements will penalise the methods in terms of precision, but will increase the feasibility of testing and diagnosis.

Definition 3. A *tomogram* $T(C, i, N) = (t_1, t_2, \ldots, t_{|O|})$ of the circuit C under input $i \in I$ is a probability distribution obtained after executing the N-experiment(C, i), such that $t_j = \frac{n_j}{N}$ where n_j is the number of times the output $j \in O$ was observed during the experiment.

A tomogram $T(C, i, N)$ is not necessarily deterministic; as a result repeating the N-experiment k times may yield up to k different probability distributions. In contrast, if the CUT C is non-probabilistic (and fault-free), each application of the test vector i will produce the same output value $C(i)$, and the tomogram $T(C, i, N)$ will have 1 in the i-th position and o's elsewhere.

In the following, the probability distribution $C_j(i)$ denotes the modelled (ideal) outputs of the circuits C_j $0 \leq j \leq n$ after using $i \in I$ as an input, while a tomogram indicates the results obtained for the CUT C after running an N-experiment having i as an input.

For example, for the probabilistic circuit CUT C with $O = \{o_1, o_2, o_3, o_4\}$ and the reference or *good* probability distribution $C_0(i) = (p_1, p_2, p_3, p_4) = (0, 0.5, 0.25, 0.25)$ for some input vector i, the $T(C, i, 5)$ of a 5-experiment may yield the four possible output values o_1, o_2, o_3 and o_4, for a total of 0, 3, 1 and 1 times, respectively. By Definition 3:

$$T(C, i, 5) = (t_1, t_2, t_3, t_4) = (0/5, 3/5, 1/5, 1/5) = (0, 0.6, 0.2, 0.2)$$

Definition 4. The true-positive probability TP is the probability that a tomogram produced by an instance of C_f is classified as produced by C_f. The false-positive probability FP is the probability that a tomogram produced by an instance of C_0 is classified as produced by C_f. The true-negative probability TN is the probability that a tomogram produced by C_0 is classified as produced by C_0. The false-negative probability FN is the probability that a tomogram produced by C_f is classified as produced by C_0.

Tomographic approaches are prone to both *false positives* (identifying a correct circuit as erroneous) and *false negatives* (identifying an erroneous circuit as correct). Intuitively, the probability of misclassifications is reduced if the length of the experiments is increased, but repeating experiments is costly, and it is of interest to know which experiment length is required in order to achieve a given confidence in a tomogram, i.e., to ensure that the probability of a misclassification does not exceed a certain pre-defined limit. The verification methods for probabilistic circuits are presented in the following.

Tomographic testing

uses a tomogram $T(C, i, N)$, and its $|F| + 1$ distance measures $D = \{d_0, d_1, d_2, \ldots, d_{|F|}\}$ with respect to the probability distributions of the correct circuit C_0 and the faulty circuits $C_1, C_2, \ldots, C_{|F|}$ are calculated. The minimal distance $\min(D)$ will indicate the closest probability distribution to the tomogram T. Detection is finalised by comparing if $\min(D) = d_0$ and, if this is the case, the circuit C is fault-free, otherwise C is faulty.

Tomographic diagnosis

has its equivalent for conventional circuits in the diagnosis of manufacturing defects[BA00], and (post-silicon) debugging where design errors are targeted[BA00]. Tomographic diagnosis

starts from the approaches used in the classic context, where a ranked list of candidate faults is constructed [BA00]. In general, a diagnostic method is considered efficient if the fault, which is present in the CUT C, is at the top of a ranked list.

Definition 5. The rank $rank(f)$ of a fault f is the position of f in the list of faults sorted by the distance between the tomogram of the faulty circuit C_f and the fault-free circuit C_0. For two errors f_1 and f_2 with $d(C_{f1}, C_0) \leq d(C_{f2}, C_0)$, then $rank(f_1) \leq rank(f_2)$. The minimal rank is 1.

Designers are interested in correcting the first reported error and the concept of erorr rank is taken into consideration. If eliminating the first ranked error fails, the designer will consider the second ranked error, and so on, until the actual error is repaired. The position of the error in the ranked list corresponds to the effort spent by the designer trying to correct false positives. The ranked diagnostic list is easy to construct for a given tomogram: the faults are sorted according to their distance to the tomogram, meaning that the set of distances D, computed during testing, will be ordered.

3.2.5 Binary tomographic tests

The detection of faulty circuits is a binary classification problem, where a set of test vectors (circuit inputs $i \in I$) is used to obtain the tomograms following the N-experiments. The use of binary tests allows the construction (generation) of *test sets*, (subsets of I) that allow a designer to prove with a given probability that an instance of a circuit is faulty or not. The use of test sets can be extended to the task of diagnosis, too, by using these to show that a circuit does not contain the specific fault $f \in F$ with a given probability.

The test and diagnosis approaches of the previous section are augmented by including the *binary decision tests* (BTT) (see Definition 6). For probabilistic circuits *all* the distances between a tomogram and the modelled probability distributions were computed, and no tractable proof was offered of the circuit being faulty or not. The use of BTTs, acting as pairwise comparisons between a tomogram and the modelled probability distributions, will be used as a mechanism of proof.

Definition 6. A binary tomographic test $BTT(C, f, i)$ for a circuit C against the error f is the procedure by which the tomogram T, obtained after an N-experiment(C, i), is compared to the modelled probability distributions $C_0(i)$: $C_f(t)$ and the circuit is classified as fault-free if $dist(C_0(t), T) < dist(C_f(t), T)$, otherwise C is faulty with fault f.

A BTT is based on an N-experiment and the obtained tomograms are specific for the used input i. Constructing a set of BTTs (see Algorithm 1) allows inferring a set of inputs that should be used for minimum-length N-experiments to correctly classify a CUT C as faulty or not. The probability threshold τ (see Definition 7) is used as the minimum probability of correct classification. The returned set of BTTs is computed based on the modelled probability distributions, and the BTTs will be used on actual instances of the CUT C. As a

result, after applying the resulting set of BTTs to C, a tomogram produced by C_f will be correctly identified as produced by C_f with at least probability τ. Intuitively, when any of the computed BTT is applied, a circuit having fault f will be identified as faulty with probability τ or higher.

Definition 7. The detection threshold τ is the minimum true-positive (TP) probability of classifying a circuit C by the $BTT(C, f, i)$ as an instance of C_f.

The task of tomographic diagnosis is to identify the fault responsible for a failure with high confidence. A major difference to testing is that, when at least one BTT classifies the circuit as faulty (i.e., having a fault f), the circuit is regarded faulty and testing stops. For diagnosis the fault-type is relevant, and additional tests have to be applied to identify whether the fault present is f or a different fault.

Adaptive diagnosis (Algorithm 2) attempts to prove for each modelled fault that it is not present. The diagnosis procedure operates directly on the circuit instance CUT C, and faults, for which their existence cannot be disproven, are considered candidates (suspects) and held in a set of suspects *SUSP*.

3.2.6 Slicing and modifiers of quantum circuits

The exclusive use of BTTs is not sufficient to improve the applicability of tomographic testing/diagnosis to quantum circuits, since specifics of quantum circuits are not included in the analysis. The methods have to be extended from the initial probabilistic circuit formulation, and to concentrate, for example, on faults affecting the relative phase of the output quantum state. Such faults could not have been detected by the Z-measurements at the outputs of the CUT C during an N-experiment. A solution similar to detecting phase-flip errors is required where circuits for measuring the Z-operator of the quantum state were used (see Section 2.3.3).

A more in-depth presentation of tomographic testing and adaptive diagnosis for quantum circuits, as illustrated by Algorithms 1 and 2, is based on the concepts of slicing and circuit modifiers.

Circuit slices

A first extension of tomographic testing is the use of quantum subcircuits (*slices*). These are sequences of gate applications. Slicing is used during the test and diagnosis as a means to implement the methods in a divide-and-conquer algorithmic manner. The subcircuit $C' = \{G_l, \ldots, G_m\}$ is a subsequence of gates from of C, and for

$$C = \{G_0, G_1, \ldots, G_{l-1}, C', G_{m+1}, \ldots, G_n\}$$

the gate sequences $B = \{G_0, G_1, \ldots, G_{l-1}\}$ and $E = \{G_{m+1}, \ldots, G_n\}$ are quantum gates, too (see the tensor product postulate of quantum mechanics in Section 2.1). Assuming the quantum circuits C and C' as unitary matrices, then $C' = B^\dagger C E^\dagger$.

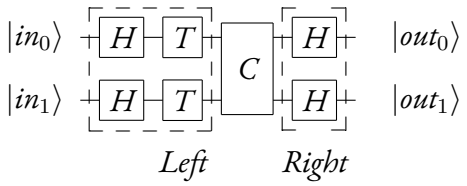


Figure 3.8: The *left*- and *right*-modifiers surrounding the circuit C.

The classical control compulsory for quantum systems motivates the slicing mechanism. For instance, trapped-ion quantum computers require a laser system to apply pulses of well-defined wavelength and duration to ions suspended in a specific EM field configuration [NC10]. Here, the classical computer controls such laser parameters as pulse frequency, direction and duration. Quantum gate operations are not represented by physical gates or wires like in the usual (classical) sense, and slicing is, as a result, feasible in a majority of quantum computing architectures. Slicing a circuit would be performed by not executing the commands corresponding to the sliced-out gates.

The detection of faults in a quantum circuit is performed by excluding the fault-free subcircuits (the unitaries B and E are sliced). This would indicate that the circuit black box assumption is violated: on the contrary, it is extended because the gates are now considered black boxes, and the time intervals between the gate applications by the classical control unit are presumed equal.

Left/right modifiers

The second extension of tomographic testing considers that quantum circuits (and slices) consist of gates from the universal gate set $\{CNOT, H, P, T\}$. The detection and diagnosis of missing gates is enhanced by using the *left/right modifiers* (LRM). The modifiers are similar to rotational gates, appended at the input (left) and output (right) of a quantum circuit. The *left*-modifier consists of the parallel application of Hadamard and T gates ($(TH)^{\otimes n}$) on all the n qubits before any gate of the circuit/slice is applied. The *right*-modifier consists of the parallel application of Hadamard gates ($H^{\otimes n}$) after the last gate of the circuit/slice (see Figure 3.8).

For tomographic testing/diagnosis the used inputs were in the computational basis, and while the Hadamard gate transforms the Z-stabilised input into an X-stabilised output, the P and the T gate keep the Z-stabilised states unchanged (see Table 2.4). The N-experiments are based on Z-measurements and a missing P or T gate will not be observed in the tomograms.

The illustration of slicing proceeds from slices of minimal size, containing a single gate from the set $\{P, T\}$. For the missing gate in the slice, Equation 3.6 is the output stabiliser of output, as the LRM circuit surrounding an empty slice is $HTIH$, where I is the identity gate. The output of an LRM circuit surrounding a minimal slice where the P gate is not missing is illustrated in Equation 3.7. The stabiliser is the result of bit flipping (applying X)

Algorithm 1 Algorithm BTTgen for fault detection

Require: Circuit under test C on q qubits; fault list $F = \{f_1, f_2, \ldots\}$
Require: Detection threshold τ; Test vector set $I = \{i_1, i_2, \ldots\}$, with $|I| = 2^q$
1: Select $i \in I$ which detects most faults by *BTT* with (or without) *LRM*.
2: Let F_t^{det} be the set of detected faults.
3: Determine the minimal length of the N-experiments of the BTTs such that $TP \geq \tau$ for each $f \in F_t^{det}$.
4: for all Undetected faults $f_u \in F \setminus F_t^{det}$ do
5: Let G be the gate associated with fault f_u
6: C_1', C_2' are slices obtained after slicing at the middle of the C gate sequence
7: Recursively apply BTTgen to C_1', C_2'
8: end for
9: return Set of BTTs such that if they all result in classification as correct, the circuit can be considered correct with at least probability τ.

the stabiliser of the empty slice. The LRM circuit of a slice containing a single T gate will be $HTTH = HPH$. The output stabiliser is presented in Equation 3.8, which is again different from the stabiliser of the empty slice.

The LRM technique differentiates between a faulty minimal and a correct slice. This is useful observation as SMGFs (from the universal gate set) are always detectable when this technique is used in conjuction with slicing. Comparing the LRMs to state tomography, the modifiers are a supplemental measurement basis, which will increase the detection and diagnosis rates of even larger slices.

$$Z \xrightarrow{H} X \xrightarrow{T} \frac{X+Y}{\sqrt{2}} \xrightarrow{\boxed{\mathrm{I}}} \frac{X+Y}{\sqrt{2}} \xrightarrow{H} \frac{Z+Y}{\sqrt{2}} \tag{3.6}$$

$$Z \xrightarrow{H} X \xrightarrow{T} \frac{X+Y}{\sqrt{2}} \xrightarrow{\boxed{\mathrm{P}}} \frac{Y+Z}{\sqrt{2}} \xrightarrow{H} \frac{Y+X}{\sqrt{2}} = \frac{X(Z+Y)}{\sqrt{2}} \tag{3.7}$$

$$Z \xrightarrow{H} X \xrightarrow{T} \frac{X+Y}{\sqrt{2}} \xrightarrow{\boxed{\mathrm{T}}} Y \xrightarrow{H} -Y \tag{3.8}$$

3.2.7 Simulation results

The results presented herein are based on simulations and will show the properties of tomographic testing/diagnosis. The tomograms for a given circuit (fault-free or faulty) are computed using a Monte Carlo simulation. The modelled probability distributions for a given input $i \in I$ (computed using [VMH09]) are used to generate the tomograms. For the probability

Algorithm 2 Algorithm BTTdiag for fault diagnosis

Require: Circuit under test C (fault-free or faulty);
Require: Circuit model C_0; fault candidates $FF = \{f_1, f_2, \ldots\}$;
Require: Allowed test vectors $II = \{i_1, i_2, \ldots\}$; $\mathcal{M}_{max}$
1: SUSP $\leftarrow$ FF // Set of candidate faults (suspects)
2: for all $i \in II$ and $f \in FF$ do
3: Determine $\mathcal{M}_{min}$ such that $BTT(C_0, f, t)$ has $TP \geq \tau$ with (or without) LRM
4: Record f with $\mathcal{M}_{min}\mathcal{M}_{max}$ in set F
5: end for
6: for all $f \in FF$ do
7: Perform BTT to obtain tomogram T of C
8: if $dist(C_0(t), T) < dist(C_f(t), T)$ then
9: $SUSP \leftarrow SUSP \setminus \{f\}$
10: end if
11: end for
12: if $|SUSP| \neq 0$// there are still suspect faults then
13: C_1', C_2' are slices obtained after slicing at the middle of the C gate sequence
14: Recursively apply BTTdiag to C_1', C_2' with $FF = SUSP$ and $C = C_1'$ or $C = C_2'$
15: end if
16: return Fault f such that C equals C_0 with fault f.

distribution $C(i) = (p_1, \ldots, p_n)$, where n is the length of the output of the probabilistic/quantum circuit, the tomogram T is computed by simulating an N-experiment:

- A (pseudo-)random integer between 1 and n is drawn according to the distribution $C(i)$;
- The entry t_j of T is computed as the number of times j is obtained and divided by N.

The distances between the tomograms and the probability distributions are computed using the Euclidean distance. For tomographic testing of probabilistic circuits the fraction of false positives and negatives among $\mathcal{M}$ N-experiments indicates the confidence placed in the accuracy of the tomogram. For instance, the 3qubit circuit of Figure 3.9b is affected by a single missing gate, and the fault list consists of a single missing-gate fault (SMGF). Applying the input vector $|000\rangle$ to both the correct and the faulty circuit results in the probabiliy distributions out_c and out_m respectively. The number of false positives and negatives among $\mathcal{M} = 100,000$ tomograms constructed using various lengths of the N-experiments, and different input vector are shown in Table 3.3.

$$\begin{aligned} out_c &= (0.286, 0.002, 0.210, 0.001, 0.210, 0.001, 0.287, 0.002) \\ out_m &= (0.167, 0.122, 0.122, 0.089, 0.122, 0.089, 0.167, 0.122) \end{aligned}$$

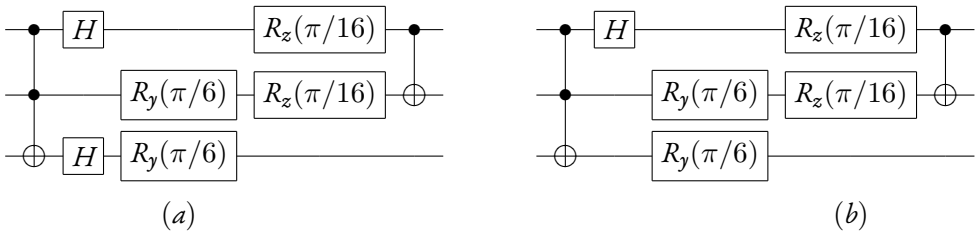


Figure 3.9: The 3qubit circuit: a) The fault-free circuit; b) A single gate (the lowest Hadamard gate) is missing from the initial circuit.

N	Nr. false positives				Nr. false negatives			
	$\|000\rangle$	$\|001\rangle$	$\|100\rangle$	$\|111\rangle$	$\|000\rangle$	$\|001\rangle$	$\|100\rangle$	$\|111\rangle$
5	1,316	1,219	1,280	1,211	20,224	20,307	20,099	20,085
10	42	43	48	50	10,697	10,729	10,747	10,587
15	1	3	1	2	5,653	5,560	5,748	5,622
20	1	0	0	1	2,777	2,743	2,827	2,722
100	0	0	0	0	1	0	1	2
1,000	0	0	0	0	0	0	0	0

Table 3.3: Misclassification results for the circuit from Figure 3.9 using various input vectors.

Several interesting conclusions arise from this first experiment. Firstly, increasing the length of the experiments improves the accuracy of classification: both false positive and negative rate fall significantly by rising N. Secondly, the choice of the used input vectors has no significant influence on the results, due the two Hadamard gates at beginning of the circuit that create equal superpositions. For inputs in the computational basis state, the superpositions are independent of the input vector (except for their relative phase).

Figure 3.10 shows, for several quantum circuits and various measurement counts N, the misclassification rate, i.e., the share of tomograms obtained from the correct circuits that have been classified as erroneous among 10,000 tomograms. The circuits used include 3qubit, three circuits provided with the QuIDDPro software package, and the 5- and 12-qubit quantum Fourier transform/addition (qftadd) subcircuits from the implementation of Shor's algorithm as presented in [FDH04]. The inputs used were $|0\ldots0\rangle$, except for the qftadd circuits, where $|0\ldots01\rangle$ was applied.

The probabilistic approach towards tomography of probabilistic circuits has some inherent limitations, and the concept of *undetectable* faults is introduced. A fault f with $dist(C, C_f) < 0.0001$ is hard to detect for a given input vector and excluded from the considerations of the true/false positive/negative rates. The numbers of qubits, single missing-gate faults, and detectable faults in the circuits are found in the headings of Table 3.4. The results in Figure 3.10 show that the classification accuracy is relatively poor for short experiments, while for N = 1,000 or more the classification is almost perfect for the small circuits.

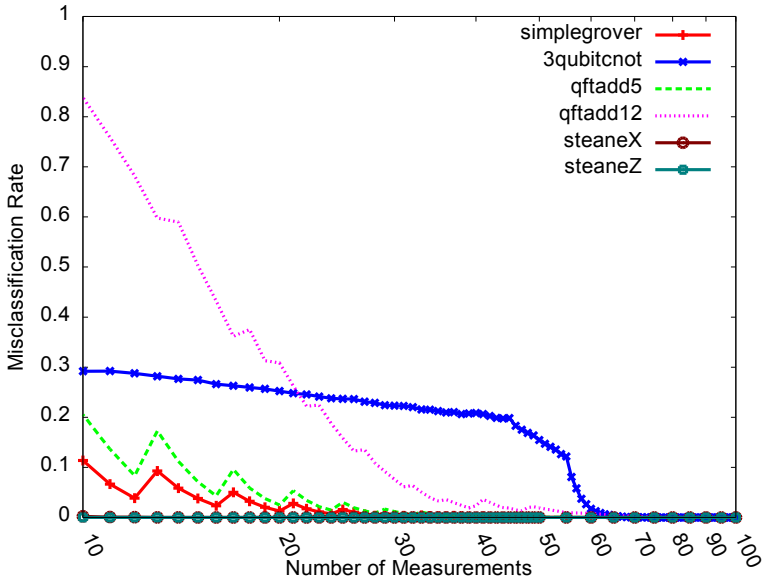


Figure 3.10: Correct quantum circuits misclassified as erroneous.

The evaluation of the tomographic diagnosis of probabilistic circuits is based on circuit simulations, too. Without the possibility of slicing and LRMs, the diagnostic rank of non-redundant faults was computed for the circuits from Table 3.4. The length of the N-experiments is selected from the set $\{5, 10, 20, 50, 100, 1000\}$. Starting from this set the minimum and maximum ranks are reported, along with the mean μ and the standard deviation δ. The rank improves with increasing accuracy of the tomograms, achieving the best resolution at $N = 100$ and $1,000$ for some of the circuits. The diagnosis of the 12-qubit qftadd circuit, where the state vector contains 2^{12} probabilities, requires lengthy experiments. Approximating the probabilities for a large circuit is exponentially more difficult than for small circuits like the 3qubit (2^3 probabilities).

The simulation results of the BTT set generation are presented in Table 3.5 and Table 3.6. The simulations were executed using binary decision tests, slicing and the left-right modifiers.

The circuits used during these simulations include: the qecc9 circuit, which is the implementation of the 9-qubit QECC, the simplegrover3 circuit, an implementation of Grover's algorithm[NC10], and chp10, which is a randomly generated stabiliser circuit containing 100 gates (CNOT, Hadamard or P, with equal probability).

The first three columns show the circuit's name, and the numbers of qubits and gates; the latter is also the number of SMGFs. Results are reported for three values of the detection threshold, quoted in column τ. The number of BTTs (which equals the number of slices), the total number of measurements, the test time (number of applied gate operations) and the number of undetected faults are given in the next columns. The maximum number of possible vectors is used for all circuits, except qecc9 and chp10 for which 32 random

Table 3.4: Diagnostic rank statistics.

N	Circuit 3qubit (3 qub., 6 faults, 4 non-red.)				Circuit simplegrover3 (3 qub., 9 faults, 4 non-red.)			
	Min	Max	Mean μ	SD δ	Min	Max	Mean μ	SD δ
5	1	2	1.0696	0.2544	1	3	1.0123	0.1147
10	1	2	1.0196	0.1386	1	3	1.0021	0.0469
20	1	2	1.0021	0.0458	1	2	1.0001	0.0071
50	1	1	1	0	1	1	1	0
100	1	1	1	0	1	1	1	0
1,000	1	1	1	0	1	1	1	0
N	Circuit qftadd5 (5 qub., 27 faults, 16 non-red.)				Circuit qftadd12 (12 qub., 143 faults, 112 non-red.)			
	Min	Max	Mean μ	SD δ	Min	Max	Mean μ	SD δ
5	1	9	1.3405	0.7573	1	108	14.2492	19.3405
10	1	7	1.1655	0.5130	1	103	13.2732	18.6559
20	1	5	1.1262	0.5104	1	103	12.1130	17.7040
50	1	5	1.1165	0.5102	1	103	11.2052	16.7547
100	1	5	1.1178	0.5187	1	103	10.5137	16.0932
1,000	1	5	1.1229	0.5362	1	103	9.9614	15.3526
N	Circuit steaneX (12 qub, 95 faults, 41 non-red.)				Circuit steaneZ (13 qub., 105 faults, 42 non-red.)			
	Min	Max	Mean μ	SD δ	Min	Max	Mean μ	SD δ
5	141	1.2327	0.5447	1	22	1.2263	0.6652	
10	1	5	1.1739	0.4454	1	7	1.1828	0.5307
15	1	3	1.1529	0.4271	1	5	1.1659	0.5015
20	1	3	1.1428	0.4243	1	4	1.1336	0.4579
100	1	3	1.138	0.4078	1	4	1.1215	0.4415
1,000	1	3	1.1141	0.3709	1	4	1.0318	0.2138

Table 3.5: BTT detection results for circuits without the LRM technique.

Circuit	Nr. qubits	Nr. gates	τ	BTTs	Nr. meas.	Test time	Undet. faults
3qubit	3	6	.99	2	50	260	1
simplegrover3	3	9	.99	2	40	280	0
qftadd5	5	24	.99	1	10	240	19
qecc9	9	60	.99	8	80	1,770	19
chp10	10	100	.95	9	120	2,870	11
			.999	150	4,200	11	
			.999	9	210	6,860	11

Table 3.6: BTT detection results for circuits with the LRM technique.

Circuit	Nr. qubits	Nr. gates	τ	BTTs	Nr. meas.	Test time	Undet. faults
3qubit	3	6	.99	2	50	260	1
simplegrover3	3	9	.99	2	40	280	0
qftadd5	5	24	.99	4	160	390	7
qecc9	9	60	.99	17	600	2,960	8
chp10	10	100	.99	15	390	6,610	3

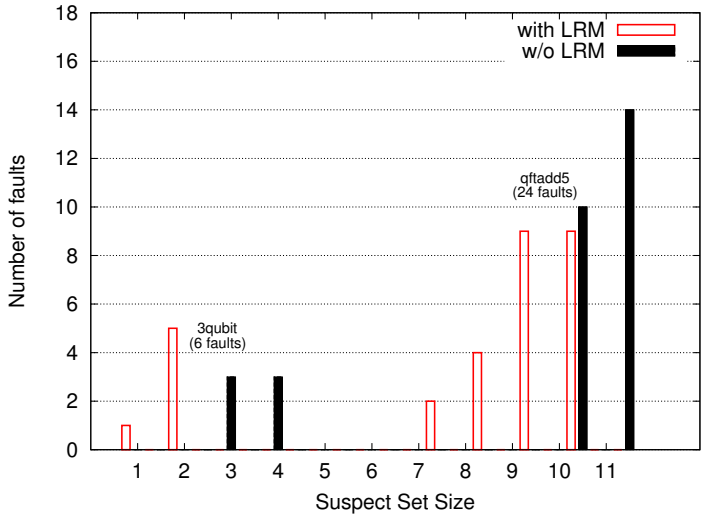


Figure 3.11: Diagnostic results for circuits 3qubit and qftadd5.

test vectors are employed. No metric of actual test cost is provided because the metric depends on the implementation technology. For example, trapped-ion quantum computers tend to require more effort to set up a measurement than to actually perform gate operations. For such technologies, the number of measurements is the appropriate metric. Some optical technologies have little overhead for measurements; here the sum of all the gate operations during tomographic testing (column *test time*) defines the test effort. Results for three values of τ are shown for circuit chp10. The impact is small, and even smaller for the other circuits, and therefore is not reported here. Table 3.6 contains the same information for the scenario when LRM is used. The number of undetected faults becomes considerably smaller, at the expense of additional test cost.

The main difference between adaptive diagnosis of quantum circuits and tomographic diagnosis of probabilistic circuits is that the adaptive approach tries to reduce the set of suspect faults, whereas the probabilistic method ranks the list of suspect faults. Figures 3.11 and 3.12 show the adaptive diagnosis results for non-randomly generated circuits. BTTdiag

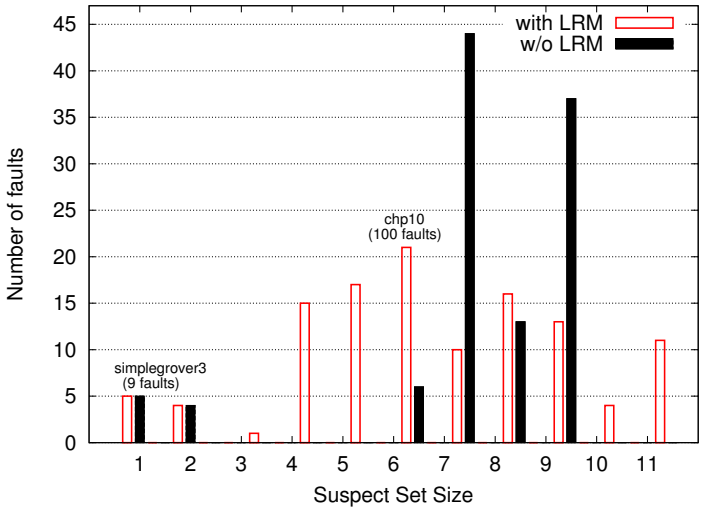


Figure 3.12: Diagnostic results for simplegrover3 and chp10.

was evaluated for g-gate circuits, by performing g experiments using all g possible circuits with an SMGF. Both figures are histograms, where for each circuit the frequency (the vertical axis) of the SUSP sizes (horizontal axis) is represented. The algorithm returned the set of suspect gates *SUSP*. In all cases, the actual missing gate was included in *SUSP*.

Two histograms are reported per circuit: the black columns show the distribution of the size of *SUSP* among the g experiments when the LRM technique is not employed, whereas the red-shaped columns represent the distribution when LRM is used. For instance, circuit 3qubit has 6 faults (the parallel application of R_z and R_y were considered a single gate); as a result, 6 experiments were performed. Without LRM, three of the experiments resulted in a suspect gate set size of 3, and another three yielded a *SUSP* size of 4. With LRM, one experiment resulted in a perfect resolution ($|SUSP| = 1$), and five experiments yielded SUSP size of 2. High-resolution diagnosis is indeed possible when the LRM technique is employed.

Random circuits with different numbers of qubits and gates (CNOT, Hadamard, P and T, with equal probability) are used to complement the results. Algorithms BTTgen and BTTdiag were applied to the circuits. Figure 3.13 summarizes the detection rate, the number of slices tested, and the test time required by BTTgen, and the suspect set sizes *SUSP* obtained by BTTdiag. The minimal, average and maximum *SUSP* sizes are represented by the error bar. The test and diagnosis quality (i.e., the detection rate and the *SUSP* size) are consistently good when LRM is used. The number of slices and the test time grow significantly with the number of gates, but scale up slowly with the number of qubits. Test time is more affected by this increase if the LRM technique is used.

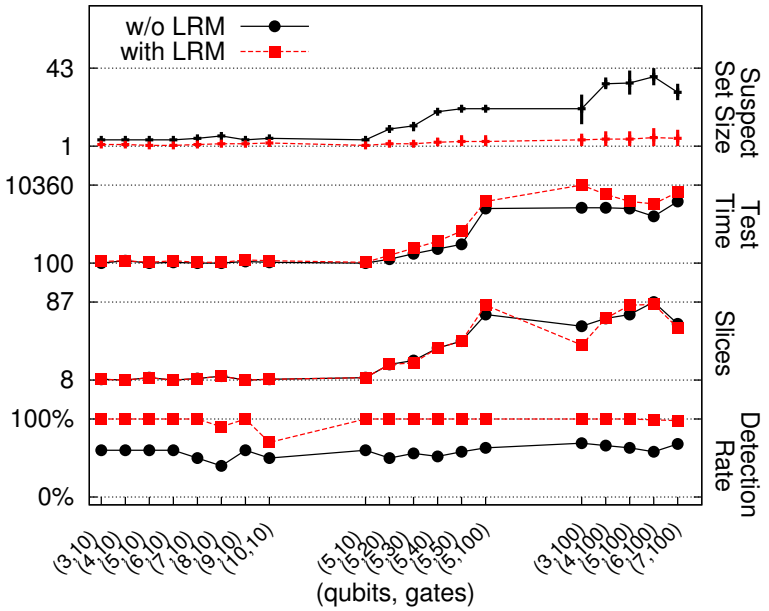


Figure 3.13: Detection and diagnosis results for randomly generated circuits.

3.2.8 Conclusion

Testing and diagnosis of probabilistic circuits was introduced starting from a tomographic approach similar to quantum state tomography. The main difference is that the presented methods reduced the types of available measurements at the output of the circuit, and used the Euclidean distance. For error correcting circuits the methods deliver good results, and treating quantum circuits as idealised probabilistic circuits offered also some insights about possible improvements like slicing and LRM. Albeit, the testing and diagnosis methods were augmented after considering the effects of relative phase and of correct subcircuits. The methods proved their efficiency when applied to circuits consisting from the standard universal gate set. The presented methods do not reconstruct a density matrix, and herein lies the difference compared to classical state tomography.

The exponential increase of the verified circuit's state space is still the major obstacle for verification approaches of this type. Whereas state-of-the-art circuits operate on a relatively small number of qubits, future circuits, which will also have to be fault-tolerant, will easily have orders of magnitude many more qubits. Future work will focus on reducing the negative impact of the exponential state space increase.

3.3 Pauli tracking

The fault-tolerant implementation of the universal quantum gate set presented in Section 2.3.3 consists entirely from CNOTs and measurements. The measurement of state su-

perpositions (internal to the circuit) renders these circuits as probabilistic: a correction is required or not depending on the measurement result.

Initially, the need for corrections would indicate that these need to be directly applied in the form of gates, i. e. the circuits are dynamically modified to include the correctional gates. However, for quantum circuits expressed entirely from the universal gate set, the commutation relations between Pauli and non-Pauli gates are well-known (see Table 3.7). The advantage is that the corrections are *tracked* through the circuit.

Definition 8. Pauli tracking is the operation by which a classical record of each teleportation result is constructed and later results of the computation are reinterpreted. [PDNP14a]

3.3.1 The quantum gate corrections

The software-based Pauli tracking replaces the need to perform active quantum corrections. The theoretical possibility is known in the quantum information community (referred to as working in the *Pauli frame*), but no details on a general algorithm necessary to perform the tracking have been previously presented.

In the following the tracking method will be detailed starting from the universal gate set of fault-tolerant gates. The fault-tolerant R_x and R_z gates presented in Section 2.3.3 were constructed similarly to teleportation circuits by using an ancilla initialised into one of the states $|A\rangle$ and $|Y\rangle$: the ancilla was entangled to the single-qubit input state that was to be rotated. CNOT gates do not employ teleportations and require no corrections, but their effect is to propagate the corrections from inputs to outputs according to the stabiliser transformations from Table 2.4. The corrections for each of the most commonly used single-qubit quantum gates are summarised in Table 3.7.

The $R_x(\pi/2)$ gate

The X-measurement in circuit of Figure 2.4a yields either $|+\rangle$ or $|-\rangle$. If the measurement result is $|+\rangle$, then the desired rotation $R_x(\pi/2)$ will be correct. If the measurement result is $|-\rangle$, the applied rotation will be $R_x(-\pi/2)$. The negative rotation is easily compensated by performing another rotation by angle π, namely applying $R_x(\pi) = X$. Consequently, quantum teleportation must be followed by executing the X-correction if the measurement result is $|-\rangle$.

The P gate

The two possible Z-measurement results from Figure 2.5b are $|0\rangle$ and $|1\rangle$. The state $|0\rangle$ indicates a correct teleportation, and a $|1\rangle$ is an indicator for the state $|\psi_f\rangle = \alpha_1 |0\rangle - i\alpha_0 |1\rangle$, where the input state was $|\phi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$. In order to obtain the correct state, a Z-correction followed by the X operation is applied, as it can be easily verified that $|\psi\rangle = XZ |\psi_f\rangle = \alpha_0 |0\rangle + i\alpha_1 |1\rangle = R_z(\pi/2) |\phi\rangle$.

g_i	s_k^{in}	b_i	s_k^{out}
$R_x(\pi/2)$	I	$\|+\rangle$	I
		$\|-\rangle$	X
	Z	$\|+\rangle$	X
		$\|-\rangle$	I
	X	$\|+\rangle$	X
		$\|-\rangle$	Z
	XZ	$\|+\rangle$	I
		$\|-\rangle$	Z

g_i	s_k^{in}	b_i	s_k^{out}
$R_z(\pi/2)$	I	$\|0\rangle$	I
		$\|1\rangle$	XZ
	Z	$\|0\rangle$	Z
		$\|1\rangle$	X
	X	$\|0\rangle$	XZ
		$\|1\rangle$	I
	XZ	$\|0\rangle$	X
		$\|1\rangle$	X

g_i	s_k^{in}	b_i	s_k^{out}
$R_z(\pi/4)$	I	$\|0*\rangle$	I
		$\|10\rangle$	XZ
		$\|11\rangle$	I
	Z	$\|0*\rangle$	Z
		$\|10\rangle$	X
		$\|11\rangle$	Z
	X	$\|00\rangle$	XZ
		$\|01\rangle$	I
		$\|1*\rangle$	I
	XZ	$\|00\rangle$	X
		$\|01\rangle$	Z
		$\|10\rangle$	X
		$\|11\rangle$	Z

Table 3.7: Correction status tracking λ for rotational gates.

The T gate

is implemented in two stages (see Figure 2.5a). The first teleporation maps state $|A\rangle$ to an intermediate state, which is afterwards given to the teleportation-based $R_z(\pi/2)$ gate from Figure 2.5b. The following three measurement outcomes have to be distinguished:

1. If the first measurement results in $|0\rangle$, no correction will be required, and the intermediate state is already the correct result.

2. If the first measurement results in $|1\rangle$, the output will be used as input for a $P = R_z(\pi/2)$ correctional rotation. If the second measurement returns $|1\rangle$, no further corrections will be necessary and $|\psi\rangle = \alpha_0 |0\rangle + e^{i\pi/4}\alpha_1 |1\rangle = R_z(\pi/4) |\phi\rangle$.

3. If the first measurement returns $|1\rangle$ and the second measurement yields $|0\rangle$, then an XZ correction will be required. The $P = R_z(\pi/2)$ correction will produce the state $|\psi_{f2}\rangle = i\alpha_0 |0\rangle + e^{-i\pi/4}\alpha_1 |0\rangle$. Then, as seen above, the XZ correction leads to $|\psi\rangle = XZ |\psi_{f2}\rangle = \alpha_0 |0\rangle + e^{i\pi/4}\alpha_1 |1\rangle = R_z(\pi/4) |\phi\rangle$.

3.3.2 Pauli tracking algorithm

This section offers a detailed description of the algorithm for performing tracking by considering circuits consisting of CNOT gates and the three types of rotational gates. The measurements inside the teleportation sub-circuits are still performed during the computation; however, their outcomes are stored in a variable rather than used for immediate correction. For each rotational gate g_i, the variable b_i holds the result of the measurement. Note that $b_i \in \{|+\rangle, |-\rangle\}$ if g_i is a $R_x(\pi/2)$ gate, $b_i \in \{|0\rangle, |1\rangle\}$ if g_i is a $R_z(\pi/2)$ gate, $b_i \in \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ if g_i is a $R_z(\pi/4)$ gate, where pairs of values refer to the outcomes of two consecutive measurements.

Algorithm 3 represents the tracking method that calculates, for a given combination of b_i values, the vector of *equivalent output correction statuses* $S = (s_1, \ldots, s_n)$. For qubit k, s_k assumes one of four values that indicate the required corrections: I (no correction), X (X-correction), Z (Z-correction), and XZ (both X- and Z-correction). The values in S are calculated such that running a teleportation-based quantum computation, *without applying corrections after the gates* but to the obtained output state, is equivalent to teleportation-based quantum computing with immediate correction.

S is calculated by propagating (*tracking*) the correction status $(s_1, \ldots, s_n)$ through the circuit. The calculation is applied *after* the teleportation-based quantum computation took place and the measurement results b_i associated with all rotational gates g_i are available. Each s_k is initialised to I (no correction). Consequently, the gates are considered in their regular order $g_1, \ldots, g_m$. If g_i is a rotational gate on qubit k, its b_i will be consulted to decide whether a correction is needed or not and the s_k is updated (the correction status is propagated). The propagated correction status appears at the inputs of subsequent gates and must be taken into account when calculating the correction status at the output of those gates. The *correction status tracking function* λ formalises the propagation.

There are two versions of λ: one for CNOT gates and for rotational gates (Table 3.7). The CNOT propagates corrections, and let c and t be the control and the target qubit of the CNOT gate, and s_c^{in} and s_t^{in} be the correction statuses at the inputs of these qubits, respectively. As a result, $\lambda(s_c^{\text{in}}, s_t^{\text{in}})$ produces a pair of correction statuses $(s_c^{\text{out}}, s_t^{\text{out}})$ at the outputs of the CNOT gates according to Equations 3.9 and 3.10. The $s \oplus Z$ and $s \oplus X$ flip the status of the respective correction in s, e.g. $XZ \oplus Z = X$, $X \oplus Z = XZ$.

$$s_c^{\text{out}} = \begin{cases} s_c^{\text{in}} & \text{if} \quad s_t^{\text{in}} \in \{I, X\} \\ s_c^{\text{in}} \oplus Z & \text{if} \quad s_t^{\text{in}} \in \{Z, XZ\} \end{cases} \tag{3.9}$$

$$s_t^{\text{out}} = \begin{cases} s_t^{\text{in}} & \text{if} \quad s_c^{\text{in}} \in \{I, Z\} \\ s_t^{\text{in}} \oplus X & \text{if} \quad s_c^{\text{in}} \in \{X, XZ\} \end{cases} \tag{3.10}$$

The correctness proof of the algorithm is based on the derivation of the R_x, R_z rotational gate corrections, which inluded the commutativity of gates from the Pauli group [PDNP14a].

3.3.3 Simulation results

The implementation of the Pauli tracking algorithm was simulated using quantum circuits that are contructing specific states used in the context of TQC (see Section 4). The circuits consist of $W \times H \times D$ unit-cells (see Figure 4.1c). As a result, the circuits operate on *Qub* qubits, and consist of *CZ* gates. The results in Table 3.8 are indicative of quantum circuit sizes necessary for practical quantum computing to become reality. However, Pauli tracking is fast and all calculations are performed within a few seconds for all cases.

The expected number of corrections without Pauli tracking is $0.5 \times m_2 + 0.75 \times m_4 \approx m$, where m_2 is the number of $R^x(\pi/2)$ and P gates (requiring a correction with probability

Algorithm 3 Pauli tracking

Require: n-qubit circuit with m gates
$g_1, \ldots, g_m \in \{CNOT, R_x(\pi/2), R_z(\pi/2), R_x(\pi/4)\}$
Require: measurement results b_i for every rotational gate g_i
Ensure: Equivalent output correction status $S = (s_1, \ldots, s_n)$
1: $s_1 := s_2 := \cdots := s_n := I$;
2: for $i := 1$ to m do
3: if g_i is a CNOT gate with control/target qubits c/t then
4: $(s_c, s_t) := \lambda(s_c, s_t)$;
5: else if g_i is a rotational gate on qubit k then
6: $s_k := \tau(s_k, b_i)$;
7: end if
8: end for
9: return $S = (s_1, \ldots, s_n)$;

W	H	D	Qub	CZ	RT
1000	100	10	6334110	18778110	0.221
1000	200	10	12637210	37485210	0.440
1000	300	10	18940310	56192310	0.655
1000	400	10	25243410	74899410	0.881
1000	500	10	31546510	93606510	1.104
1000	600	10	37849610	112313610	1.415
1000	700	10	44152710	131020710	1.754
1000	800	10	50455810	149727810	2.171
1000	900	10	56758910	168434910	2.520
1000	1000	10	63062010	187142010	2.884

Table 3.8: Run-times RT (in seconds) of the Pauli tracking algorithm for circuits with Qub qubits and CZ quantum gates.

0.5) and m_4 is the number of T gates which may require one or two corrections with the expected number of corrections equal to 0.75. Corrections have to be performed on each output k with $s_k \neq I$, and the expected number of corrections with Pauli tracking is bounded by n (XZ is a single correction). As most relevant circuits have far more gates than qubits, Pauli tracking substantially reduces the overall effort for corrections.

3.3.4 Conclusion

Fault-tolerant quantum gate constructions are probabilistic due to the used teleportation mechanisms; as a result, the class of fault-tolerant circuits is compatible with Definition 1. The probabilistic effects in these circuits reveal the need for Pauli tracking, and a first tracking algorithm was presented. This result fills an important gap in the classical control soft-

ware needed for large-scale quantum computation. Pauli tracking is instrumental for both error correction and teleportation based protocols; the algorithm is easily adjustable to incorporate the required tracking for a specific implementation of QECC. Future work will be focused on adapting this algorithm to popular error correction techniques such as topological codes[FMMC12,RHG07] which require more teleportation operations.

3.4 Summary

The analysis of probabilistic circuits started from a general definition that included quantum and stochastic circuits. The similarities between the circuits do not transcend the probabilistic behaviour, for quantum circuits are extremely precise but fragile and require fault-tolerant mechanisms. On the other hand, stochastic circuits are fault-tolerant by design, but reaching a high degree of precision requires exponentialy long bitstreams to operate on.

The progressive precision property of stochastic circuits was used to mitigate the complexity of simulating quantum circuits, and initial results were encouraging. The potential of PP was shown in conjunction with the direct mapping of the quantum circuit formalism on an FPGA. Although the runtime complexity of simulating quantum circuits was translated into the hardware resource requirements, the potential of this approach lies in the very flexible nature of the FPGAs and the fast clock times. Future work will be directed towards improving the mapping of quantum operations using stochastic computing elements, a more strategic optimisation of the hardware resources and the possibility of using stochastic number correlations to *partialy* emulate entanglement. Quantum correlations existing between entangled particles are different from classical correlations existing in stochastic circuits[NC10], and the last task could open new perspectives on the complexity of quantum circuits simulation.

The computer engineering aspect of testing and diagnosing missing gate faults in probabilistic circuits was the starting point of specialised quantum circuit diagnosis. Having introduced the tomography of probabilistic circuits, the initial method of applying inputs and measuring the outputs was extended by more advanced techniques including slicing and left/right modifiers. Existing quantum circuit tomography methods are computationally very expensive due to the exponential number of parameters that have to be inferred. Tomography is also more complex with respect to the required experimental apparatus as the hardware needs to support almost arbitrary measurement basis. The probabilistic method presented in this chapter offered adequate results when applied to arbitrary quantum circuits, and good results for stabiliser circuits which are commonly used in QECCs.

Using the definition of probabilistic circuits to include fault-tolerant circuits based on teleportation techniques completed the analysis from this chapter. Teleportation of information has become a central construct in the quantum computing architectures like MBQC. Although the output of such circuits is probabilistic, the correct result can easily be computed by tracking the required corrections though the circuit.

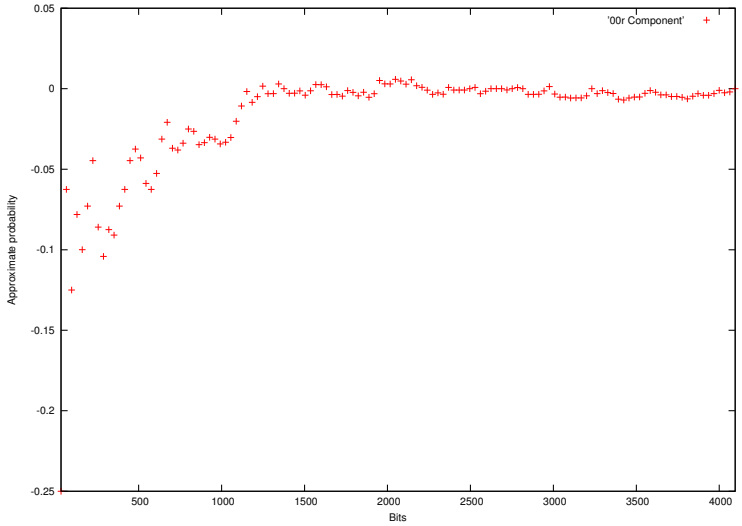


Figure 3.14: Progressive precision results for the circuit from Figure 3.7a for SN length between $[32, 4096]$ with increments of 32 bits.

On the whole, probabilistic circuits lay at the foundation of emerging technologies, and future work will consider a holistic perspective of such circuits with respect to their theoretical and practical utility.

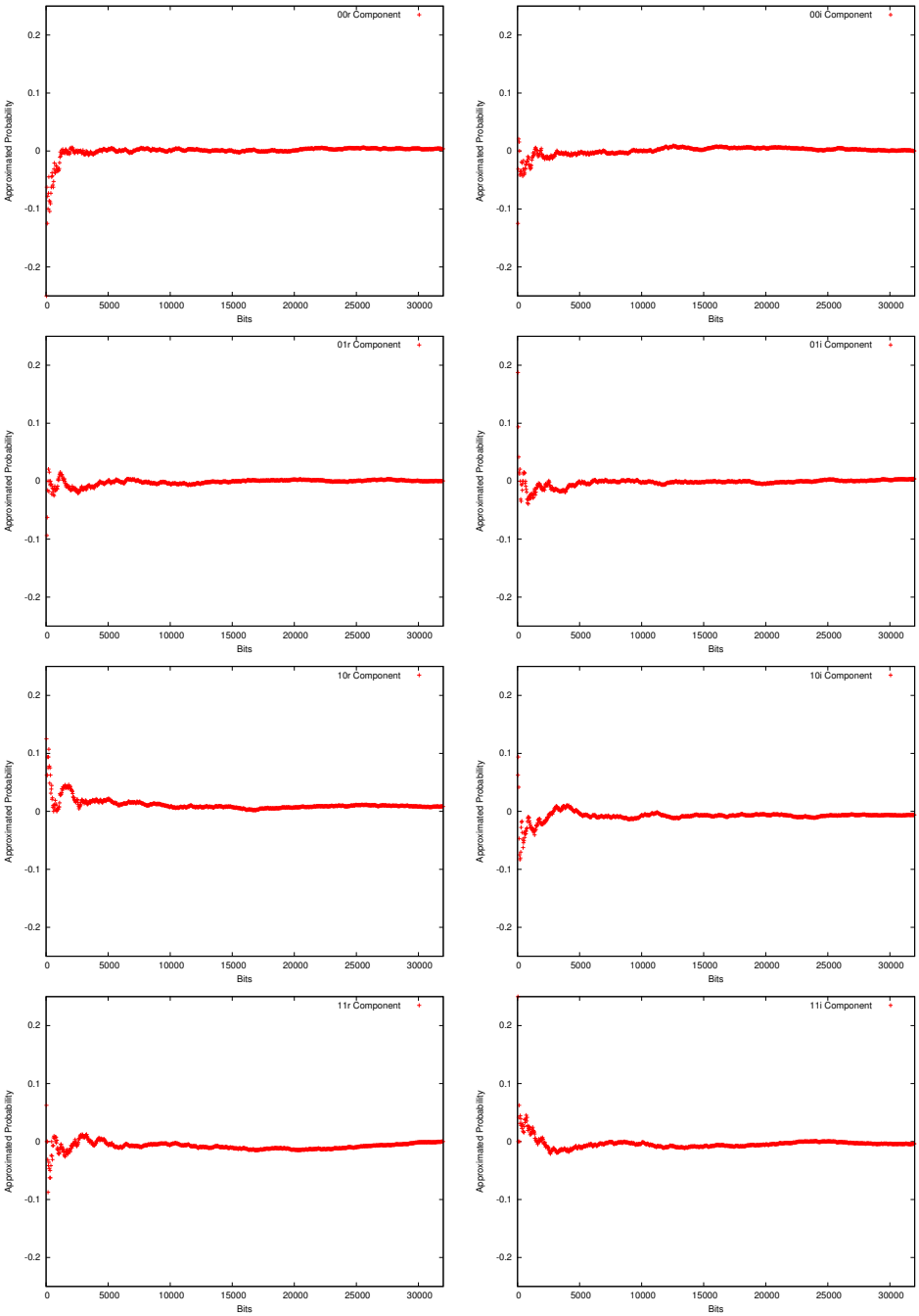


Figure 3.15: Progressive precision results for the circuit from Figure 3.7a for SN length between $[32, 32000]$.

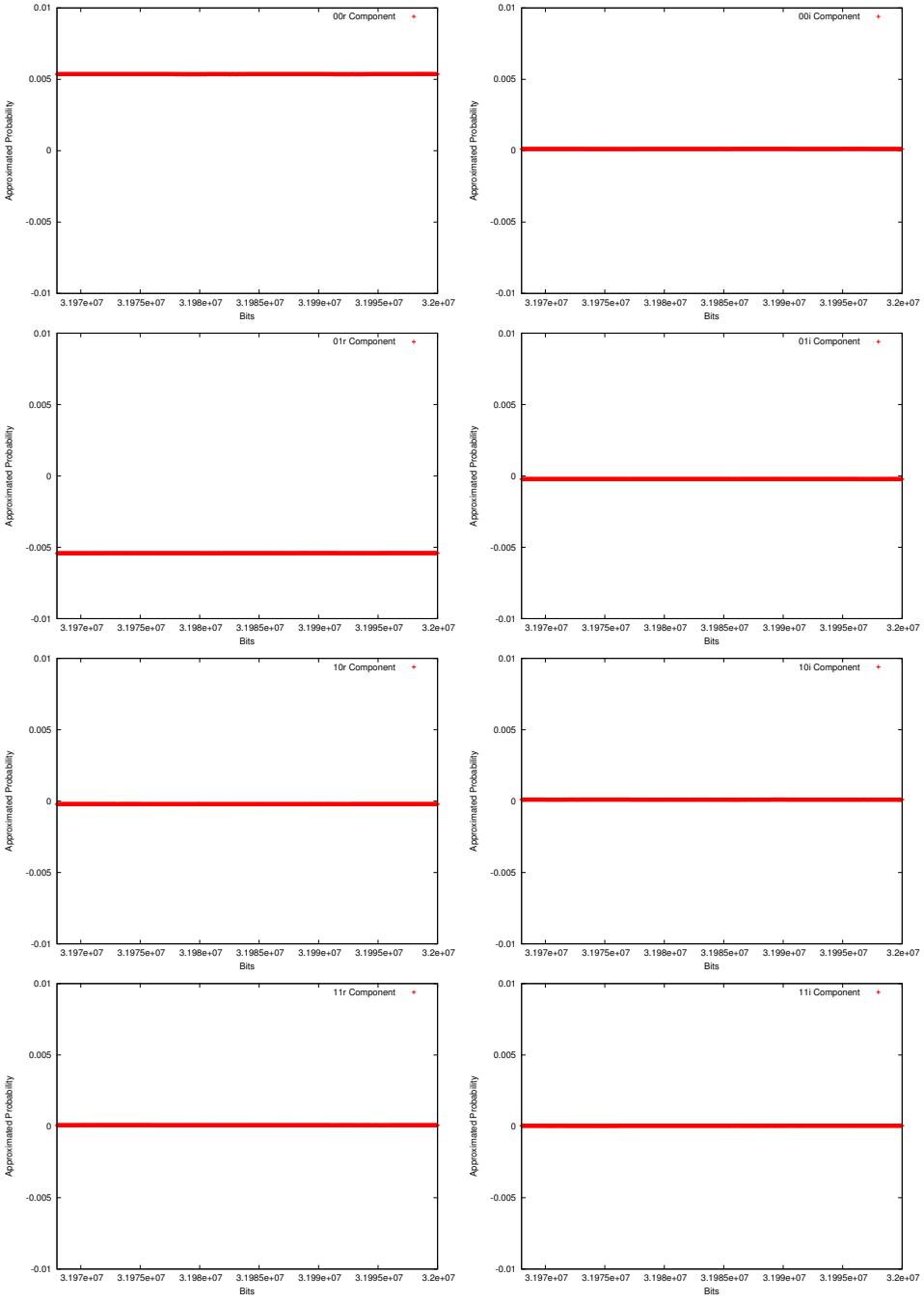
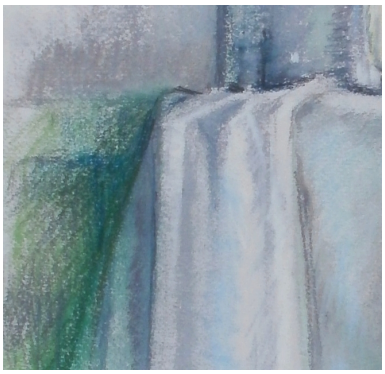


Figure 3.16: Progressive precision results for the circuit from Figure 3.7a for SN length between $[3.197 \times 10^7, 3.2 \times 10^7]$.



4

Topological Quantum Computing

Topological Quantum Computation (TQC)[FG09,RHG07] has emerged as a promising quantum computing model, being able to support large scale quantum information processing. The model incorporates the surface error correction code (see Section 2.3.5) and has been shown to be compatible with a large number of physical systems[DFS+09,JVMF+12]. While experimental technology is not yet of sufficient size to implement the full TQC model, there have been demonstrations of small scale systems and further expansion to a fully scalable quantum computer is envisioned.

Automatic design methods for TQC circuits were not investigated prior to this work as the field of topological quantum computation is still in its infancy. Although the details of the model were presented in self-contained works[FMMC12], there were no hints about how TQC specific problems could be solved in an algorithmic way. The TQC architectural considerations[DFS+09] were formulated more from a physics perspective, and not from a *Computer Engineering* one.

This work will present the first algorithms for the synthesis and validation of TQC circuits. After a short introduction to TQC, the concept of *correlation surface* is investigated, and starting from it the properties of TQC circuits are introduced. The results of the correlation surface analysis will yield the validation and synthesis procedures presented in the following chapter.

4.1 Preliminary analysis

The TQC computational model is an application of the surface code (Section 2.3.5), which relies on the measurement-based quantum computing paradigm (Section 2.2) and supports the minimum set of gates that achieve universality of quantum computation (Section 2.1). The introduction of this section will offer all the necessary details to motivate the decisions taken during the construction of the automatic design algorithms.

Similarly to the surface code, TQC is based on constructing *defects* by removing physical qubits from a lattice. The main difference is that the lattice structure is not two-dimensional but three-dimensional because the surface code array is mapped to a cluster-state (graph-state) visualised in 3D. The defect traces through the lattice (similar to the ones indicated in Figures 2.22c and 2.22h), generated by the encoding of logical qubits, can be abstracted by a geometric description. The error-correction capabilities (distance of the code) are directly related to the size of the defects, but, without affecting the generality of the discussion, defects of minimal size will be considered.

4.1.1 The surface code in MBQC

The transition from the two-dimensional surface code to its three-dimensional counterpart, which is applied in an MBQC setting, will be explained in this section. In order to use a measurement-based computational scheme which is error-corrected, the complete functionality of the surface code has to be compatible with the "initialise and entangle first, perform only measurements afterwards" method. Such an approach is not possible for the two-dimensional code due to the 2D lattice structure and the functionality of the measurement qubits, which is incompatible with MBQC: each measure-X or measure-Z qubit (see Section 2.3.5) would have to be repeatedly initialised, entangled to its neighbouring data qubits, and measured in order to enforce the local stabilisers and enable the error-correction.

A further problem is that MBQC is based on graph states (cluster-states), where each qubit is stabilised by $X \bigotimes Z_n$, n indicating the neighbouring qubits to which the qubit was entangled using *CPHASE* (see Section 2.1.5). The cluster-qubits are initialised before entanglement into the $|+\rangle$ state. For the measurement of the X- and Z-syndromes, the parity circuits used in the surface code contain *CNOT* gates, which is not a valid option when constructing the cluster-state.

The manipulation of the underlying cluster-state will have to be compatible to operating on graph-states. The surface code support for primal qubits (primals) and dual qubits (duals) will have to be adapted. Defects supporting the dual qubits were constructed in the surface code by not enforcing the action of measure-X qubits from the 2D-lattice, thus effectively removing these qubits. Dual qubits were constructed by removing measure-Z qubits. A measurement will effectively remove a qubit from the lattice without changing its remaining structure (see Section 2.1.5), and defects in graph-states (qubit deletion) are constructed by Z-measurements. In TQC, both types of defects will be constructed only by removing

X	Z		X		X	X		X	X		X
Z	X	Z	Z	X	Z	Z	X	Z	Z		Z
	Z	X		Z	X		X				
(a)			(b)			(c)			(d)		

Table 4.1: The effect of parity checkers revisited. The tables (a) and (b) represent the stabilisers before the X-basis measurement, and tables (c) and (d) illustrate the measurement result. The Is are not included in the tables. a) The initial stabiliser table; b) The transformed stabiliser table after multiplying the first stabiliser with the last one; c) The Z part of the last stabiliser is anti-commuting with the X measurement basis, and thus, after the measurement, the complete stabiliser is replaced by IXI; d) The stabiliser table after multiplying ZXZ with IXI.

cluster-qubits by such measurements, and not by stopping the function of measurement qubits.

The equivalent of the two-dimensional code is the 3D one investigated in [RHG06]. It was found that a three-dimensional cluster state is natively similar to the two-dimensional surface code, and the third dimension is just a readaptation of the surface code temporal operation. During the functioning of the surface code the third dimension is temporal in data storage, and is given by the fixed order of how the measure-qubits are applied (stabilising the neighbouring data qubits). For the 3D cluster-state, the previous temporal dimension is translated into a spatial dimension [RHG06].

The action of cluster-qubits is illustrated using the measurement qubits of the planar surface code. The measure-Z qubits are directly mapped to cluster-qubits, due to the stabiliser expression at each cluster-qubit. Measuring a lattice qubit in the X-basis returns the eigenvalue of the Z-stabilisers of the neighbouring qubits. For example, in a cluster of three qubits, the X-measurement of the middle qubit will have the identical effect to the application of a parity checking circuit (see Table 4.1 where the individual eigenvalues associated with each stabiliser were left out).

For the surface code, each measurement qubit was entangled to 4 data qubits, and an hypothetical cluster state that contains only data and measure-Z qubits will look similar to the one from Figure 4.1a.

The transformation of the measure-X mechanism starts from the following observation: by knowing the Z-parity of a qubit set, its Z-parity is related to an X-parity if the initial qubit states are transformed by Hadamard gates. In TQC this transformation is implemented by state-teleportations using cluster-states (see the implementation of the Hadamard gate in MBQC).

In the two-dimensional surface code the parities were computed for rings of data qubits surrounding the measure-X qubits. In TQC the eigenvalue of a ring of X-stabilisers is inferred by the measurement of a Z-stabiliser ring (first step) that is further transformed by Hadamards through teleportation given the underlying lattice structure (second step).

There is no structural difference between the mapped measure-X and measure-Z contructs, as both will be cluster qubits having 4 neighbouring qubits. Therefore, the spatial di-

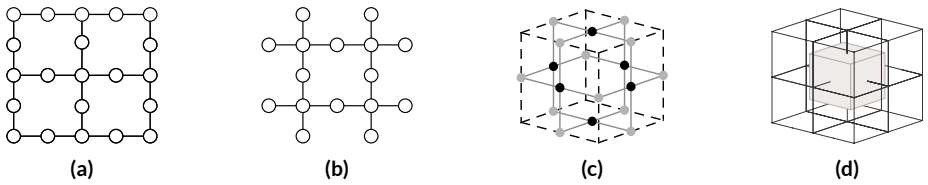


Figure 4.1: Elements of the three-dimensional surface code: a) Hypothetical lattice of data and measure-Z qubits; b) Shifting the previous lattice in both directions by one unit resembles the star arrangement of data qubits around measure-X qubits in the two-dimensional surface code; c) The TQC cluster state unit-cell; d) Stacking 8 primal unit-cells results in a dual unit-cell formed at the middle of the structure.

mension in the 3D cluster is factually interpreted locally as the mechanism of using cluster-qubits from the point of time t_0 to *simulate* measure-X qubits that are required at point of time t_1.

In a two-dimensional lattice supporting only "simulated measure-X" qubits, the structure from Figure 4.1b occurs. As a result, the 3D lattice supporting the cluster-state is constructed from two types of layers that are structurally similar two-dimensional lattices entangled according to the pattern from Figure 4.1c. The resulting cluster-state is a graph-state having each vertex-qubit v stabilised by $X_v \bigotimes_{i=1}^{4} Z_i$, where i indicates the 4 qubits entangled to v.

In order to maintain the notation from [FMMC12,RHG07], a layer containing simulated measure-X qubits will be called *primal*, and a *dual* layer will contain measure-Z qubits. At a later point it will be seen that the measurement qubits are in fact not of two types. However, the mapping of the measurement qubits discussed above shows that the mechanisms required by the 2D surface code are present by default in a 3D cluster. In the following, the cluster stabilisers will be used instead of referring to parity checkers.

The entanglement-relations between the layers introduce supplemental edges in the lattice, and by stacking three layers of consecutive different types (e.g. primal-dual-primal) the result will contain a highly regular element, the *unit-cell* (see Figure 4.1c) having 6 isomorphic faces. Each X-measurement will read the eigenvalues of the Z-stabilisers of the neighbouring qubits, and for a single unit-cell, the X-measurement of all the 6 qubits located in the middle of the faces will indicate even parity: the edge-qubits are referenced twice according to the cell structure.

The layered construction of the lattice results into two self-similar interlaced sub-lattices: the primal and the dual lattice. The construction is exemplified by the effect of stacking 8 primal unit-cells like in Figure 4.1d: a dual unit-cell is formed in the middle (marked grey in the figure). The two lattices are used for encoding two types of logical qubits.

Similarly to the planar surface code, logical qubits are encoded by constructing defects using Z-measurements of lattice qubits. In TQC logical qubits are defined using pairs of defects. Primal logical qubits are represented by pairs of primal defects constructed by Z-measuring primal face-qubits of primal unit-cells. Dual logical qubits are formed after re-

moving dual face-qubits of dual unit-cells.

Another similarity with the surface code is the supported universal gate set: the CNOT gate is natively constructed in a fault-tolerant manner, and the rotational gates R_x and R_z are constructed using the injection of the $|A\rangle$ and $|Y\rangle$ (see Section 2.1.2), where high fidelities of the injected states are achieved through state distillation (see Section 2.3.4). The MBQC rotational gates were implemented using rotated measurements, meaning that, besides the X/Z-measurements, rotated measurements of individual cluster qubits were supported, too. However, in order to reduce the requirements placed on TQC, only X/Z-measurements were chosen in the original model[FG09].

One of the differences to the surface code is that TQC error-correction will concentrate only on Z errors (phase flips), as these are detected by the X-basis measurements. A Z-error on a qubit is equivalent to an X-measurement error of the same qubit. A phase flip will change the sign of the eigenvalue returned by an X-measurement, because $ZXZ^{\dagger} = -X$. This is equivalent to an error-prone measurement that indicates the opposite eigenvalue instead of the correct one (e.g. -1 instead of 1).

The *CPHASE* gate transforms a two-qubit input stabilised by XI into XZ, and an input stabilised by IX into an output stabilised by ZX. Bit-flip (X) errors on individual cluster qubits are equivalent to single or multiple phase flips on the neighbouring qubits. The 3D structure of the TQC lattice renders the concepts of measure-Z and measure-X qubits from the surface code as superfluous. Consequently, the error-correction considers only Z-errors existing on the cluster qubits placed in the middle of the unit-cell faces (face-qubits). The correction is not investigated in this work and[FMMC12,RHG07,FG09] contain further details.

4.1.2 The lattice

The 3D-lattice resulting after mapping the surface code to MBQC is a graph-state: the qubits are initialised into the $|+\rangle$ state, and are entangled using the *CPHASE* gate, such that each qubit is stabilised by $X \bigotimes Z_n$ where n indicates its neighbouring qubits. The entanglement pattern is constructed in 3D for $n = 4$, and the resulting lattice is abstracted as the stacking of *unit-cells* along all the three dimensions (*w*idth, *h*eight and *d*epth). The 3D fashion of the lattice indicates that lattice qubits have an an associated coordinate.

A general unit-cell has 27 ($3 \times 3 \times 3$) vertex positions, 18 of which are occupied by physical qubits, denoted by grey and black circles in Figure 4.1c. Neighbouring cells share a face, and a lattice with $mc_w \times mc_h \times mc_t$ unit-cells has a total of $(2mc_w + 2) \times (2mc_h + 2) \times (2mc_t + 2)$ positions. The set of coordinates associated to the lattice physical qubits will be denoted by *TQCC*.

Each unit-cell from the lattice is adressed by the center coordinate of the cell. All unoccupied lattice-coordinates represent cell centres, and the set of cell centre coordinates will be denoted by *CEL*. The physical qubits at the faces of such a cell (six black circles in Figure 4.1c) are *face-qubits* denoted by $F_{cw,ch,ct}$, and the qubits at the sides (12 grey circles in Figure 4.1c) are *side-qubits* denoted by $S_{cw,ch,ct}$. The set F^p is the set of measure-X qubits used to

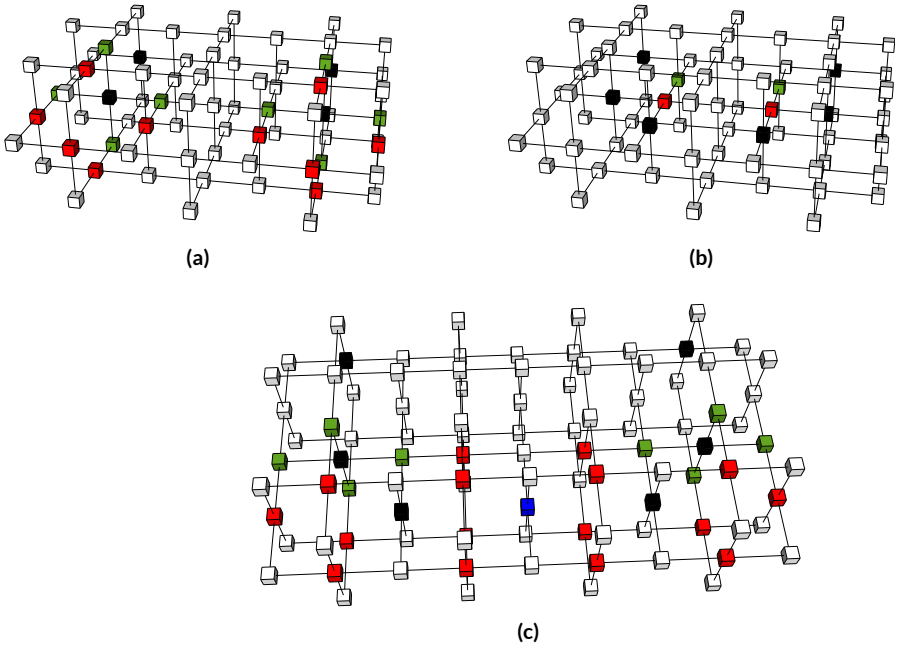


Figure 4.2: Initialisation patterns where the black qubits represent face-qubits used for the definition of defects and the blue qubit is an injection. The parities of the green qubit structures are inferred after measuring the red qubits in the X-basis: a) Two green rings; b) A green chain; c) The injected state is also measured in the X-basis.

define primal defects, while F^d is the set of measure-Z qubits used for dual defect construction.

The set of all qubits of a cell is $C_{cw,ch,ct} = F_{cw,ch,ct} \cup S_{cw,ch,ct}$. The primal lattice (with corresponding C^p, F^p, S^p) contains all the cells having odd coordinates, and the dual lattice (with corresponding C^d, F^d, S^d) contains all the cells having even coordinates. A coordinate is even or odd if all its components are even or odd. The set $TQCC = F^p \cup F^d$ is the complete set of lattice qubits.

4.1.3 Logical initialisation and measurement

Logical qubits (primal or dual) are defined by pairs of defects generated through the lattice by removing qubits from the corresponding set of face-qubits (F^p or F^d). The initialisation and measurement of logical qubits is performed by considering the X-measurement results of qubits arranged into specific measurement patterns. Only the initialisation and measurements of the primal qubits are presented, as the extension to dual qubits is straightforward.

Both initialisation and measurement procedures solve the problem of inferring the eigenvalue associated to a logical stabiliser, and the logical measurement of logical qubits is performed using measurement patterns identical to initialisation. The TQC logical measurement is implemented using exactly the same measurement patterns from logical initialisa-

tion, but attached in reversed temporal order to the defect configurations.

Initialising a logical qubit into a known X_l or Z_l eigenstate is equivalent to inferring the eigenvalue of a ring or chain of Z stabilised cluster qubits. Removing a cluster qubit (face-qubit) by a Z-measurement results in the parity of the 5 other face-qubits of the unit-cell indicating the eigenvalue of a Z-stabiliser ring. As illustrated in Figure 4.2a the ring surrrounding the created defect consists entirely of side-qubits, while the other side-qubits of the unit-cell cancel out when considering the lattice stabiliser generated by the 5 face-qubits.

A logical qubit, defined by a pair of defects, is initialised by computing eigenvalues d_1 and d_2 of the two corresponding defect rings (denoted by Z_1 and Z_2) containing, for example, 4 qubits. The X_l stabiliser of a primal qubit is $X_l = (-1)^{d_1 d_2} Z_1 Z_2$ (see Figure 4.2a).

The initialisation of a defect pair into a known eigenvalue of Z_l, represented as a chain of Z-stabilised cluster qubits, is performed by reducing the number of neighbouring cluster qubits between the two defects: construct a third defect between the two existing defects. It can be noted that s, the X-parity of the first correlation chain, indicates the eigenvalue of the Z-chain from the logical stabiliser (see Figure 4.2b). For a primal qubit, the logical Z_l stabiliser is defined as $(-1)^s Z_c$, where Z_c represents the set of qubits arranged as a chain.

State injection is used as a means to initalise logical qubits into one of the two rotated states $|A\rangle = T|+\rangle$ and $|Y\rangle = P|+\rangle$, which are needed for achieving universality. During the construction of the cluster state, the P and the T gates can be commuted with the CPHASE gate, representing a rotation about π around the Z-axis. The two possible injected states are after the commutation $P|+\rangle = |Y\rangle$ and $T|+\rangle = |A\rangle$, and there is no need for rotated measurements. Logical rotational gates are implemented by encoding *injection states* into the cluster as logical ancillae qubits, and then applying exactly the teleported rotational gate circuits from Section 2.1.3. An injected state is encoded by measuring the physical qubit existing in $|A\rangle$ or $|Y\rangle$ into the X-basis, and constructing from that qubit two defects of opposite direction [RHG07]. The construction ensures that the physical qubit state is encoded into $|A_l\rangle$ or $|Y_l\rangle$ (see Figure 4.2c).

The X_l and Z_l of encoded injected states is computated similarly to *normal* defect initialisation, except that the X-measurement result (previously the Z-measurement) of the injected qubit (previously a normal cluster-qubit) is used for calculating the eigenvalues of the loop and chain stabilisers.

4.1.4 Logical stabilisers

The surface code is defined starting from the observation that logical qubits are constructed correctly as long as their logical stabilisers, X_l and Z_l, are anticommuting. Similarly, the logical stabilisers used in TQC will have to be anticommuting, thus enabling the operations on the logical qubits. TQC primal qubits are defined by constructing two primal defects through the TQC lattice, whereas dual logical qubits are defined by a pair of dual defects.

As logical qubits are encoded into the TQC lattice, the X-measurement of each layer resembles the process of teleporting qubit states in the un-encoded MBQC scheme: each state

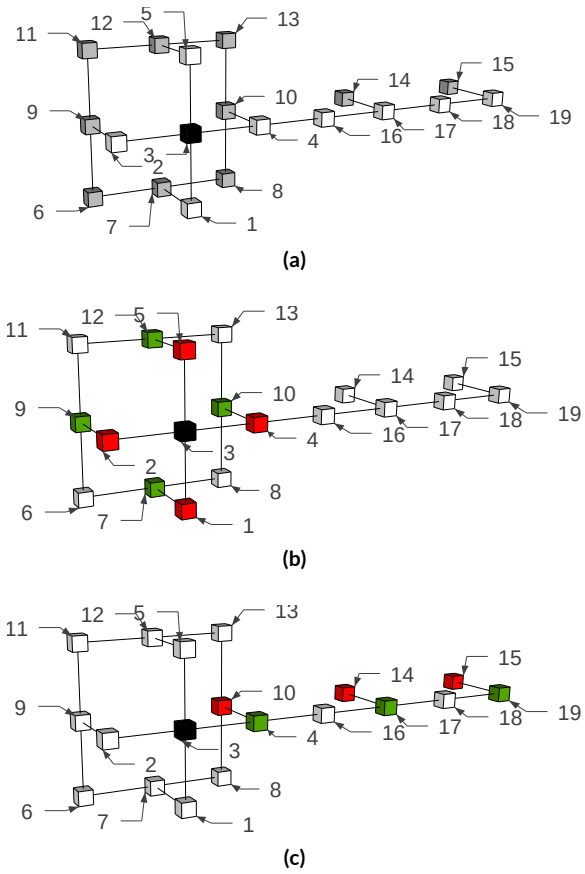


Figure 4.3: Elements of the logical stabilisers are exemplified using a reduced cluster-state. The red qubits mark the loops and chains of cluster qubits used for the definition of the logical stabilisers, while the green qubits mark the cluster qubits used for correlating the rings and loops throught the cluster: a) The grey qubits belong to a dual layer, the white ones to a primal layer, and the black qubit is marked for Z-measurement; b) After measuring the black qubit, the X-parity of the green qubits will indicate the eigenvalue of the ring of red qubits surrounding the black qubit; c) Assuming that a second defect is constructed, and a second black qubit would exist, the parity of the X-measurement of the green qubit chain will indicate the eigenvalue of the red qubit chain.

is teleported (including an inherent Hadamard transformation at each step) to its neighbouring entangled qubits, whereas in TQC information is teleported to the following layer along the temporal direction. For example, the data qubits of a primal layer are teleported during a first step to a dual layer, afterwards to the next primal layer, again to a primal layer etc.

For a primal defect (see Figure 4.3a), the eigenvalue of a *ring-stabiliser* $R = Z_1 Z_2 Z_4 Z_5$ surrounding the defect boundary (see Figure 4.3b) is *correlated* to the individual X-parity of the teleported data qubits in the dual layer, which is the ring R' of X-stabilisers. The follow-

ing relation occurs if the 3D lattice contains only a primal and a dual layer.

$$\begin{aligned} R' &= (X_7 Z_1 Z_6 Z_8)(X_9 Z_2 Z_{11} Z_6)(X_{12} Z_5 Z_{11} Z_{13})(X_{10} Z_4 Z_{13} Z_8) \\ &= (X_7 X_9 X_{10} X_{12}) R \end{aligned}$$

If the lattice consists of more than two layers, and the defect is constructed again in the following primal layers, then each two rings of Z-stabilisers are always correlated by rings of X-stabilisers existing in the dual layers.

The 2D surface code used chains of Z-stabilised qubits to construct logical stabilisers, and this definition is maintained in the case of TQC. The definition of a logical stabiliser, encoded by the Z-parity of a qubit chain, can start from $C = Z_{10} Z_{14} Z_{15}$ existing in dual layer (see Figure 4.3c). It can be noticed that C is expressed using a chain similar to $C' = X_4 X_{17} X_{19}$.

$$\begin{aligned} C' &= (X_4 Z_{10} Z_{16})(X_{17} Z_{16} Z_{18} Z_{14})(X_{19} Z_{18} Z_{15}) \\ &= (X_4 X_{17} X_{19}) C \end{aligned}$$

Again, if the lattice consists of more than two layers, and the defect is continued in the following primal layers, then each chain of Z-stabilisers from the dual layer is correlated to a chain of X-stabilisers existing in the primal layer.

For primal qubits, the logical X_l stabiliser is represented by a ring of Z-stabilised side-qubits, and Z_l is a Z-chain connecting the face-qubits of the unit-cells where the primal defects were defined (see Figure 4.3c). For dual logical qubits the interpretation is reversed, X_l is a Z-chain, while Z_l is related to the Z-rings.

The anticommutation property of X_l and Z_l is checked by noting that the multiplication of X_l by R' is still a valid stabiliser of the primal defect, and R' and Z_l share a common qubit (numbered 4). At the same time both stabiliser expressions refer to qubit 10: the first time X-stabilised, and the second time Z-stabilised. The anticommutation is illustrated by $X_{10} Z_{10} = -Z_{10} X_{10}$:

$$X_l Z_l = (X_l R') Z_l = -Z_l (R' X_l)$$

4.1.5 Correlation surfaces

The quantum-correlations resulting after qubit measurements can be used for tracking the functionality of a circuit[BBD+09]. This is also the case in TQC. The X_l stabiliser of a primal defect is a ring of Z-stabilised qubits, which is *correlated* through a ring of X-stabilised qubits (e.g. R') to the next primal layer. The Z_l stabiliser, defined between a pair of defects, is *correlated* between dual layers by chains of X-stabilised qubits (e.g. C').

Definition 9. A *correlation surface* is an X-stabiliser defined over the cluster qubits that connect the logical stabilisers of the circuit inputs to the logical stabilisers of the circuit outputs, such that information is propagated correctly during the circuit operation[FG09].

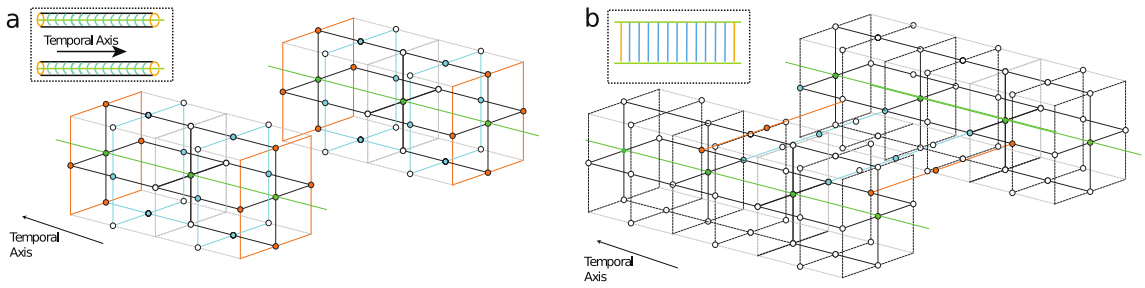


Figure 4.4: The two types of correlation surfaces: a) Tubes are formed by rings surrounding the defects; b) Sheets are formed by chains connecting the defects.

There are two types of correlation surfaces, named after the geometric shapes they resemble: *sheets* and *tubes* (see Figure 4.4). A tube is constructed from correlation-rings, and a sheet is constructed from correlation-chains. For a primal qubit the logical X_l will be correlated by two tubes, and the logical Z_l by a sheet, whereas for a dual qubit, tubes will correlate Z_l and the sheet will correlate X_l.

Throughout the chapter, when refering to correlation surfaces, the measurement and initialisation patterns of logical qubits, consisting of X-measurements of cluster-qubits, will be considered belonging of the surface. This choice is motivated by the possibility of using the cumulative X-parity of the surface for the following analysis.

For example, it is possible to show that the stabiliser of a defect in a layer is correlated to another defect in a layer of the same type. Let the example consider an input and an output cap.

The input cap used for X_l initialisation of primal qubits is a stabiliser of the form $IC = (\bigotimes Z^{(0)}_{r\in\mathcal{S}^p})(\bigotimes X_i)$, where r are side-qubits forming the ring-stabiliser, and i indicates the face-qubits next to the defect. The number (0) indicates that the Z of a qubit from the primal layer 0 is referred to. Each correlation ring existing in a dual layer has a stabiliser of the form $R^{(i)} = (\bigotimes Z^{(i-1)}_{r\in\mathcal{S}^p})(\bigotimes X^{(i)}_m)(\bigotimes Z^{(i+1)}_{r\in\mathcal{S}^p})$ for i odd numbers expressing the dual layers. The output cap is $OC = (\bigotimes Z^{(i+1)}_r)(\bigotimes X_o)$, where $i+1$ is an even number indicating the output cluster qubits belonging to the defect and o indexes the face-qubits next to the defect.

$$TUBE = \bigotimes_{i=0}^{n} R^{(i)} = (\bigotimes Z^{(0)}_{r\in\mathcal{S}^p})(\bigotimes Z^{(n)}_{r\in\mathcal{S}^p})(\bigotimes_{i}^{n} X^{(i=0)}_m)$$

The result of multiplying all the correlation-rings will be denoted $TUBE$, which consists of an X-stabiliser of the cluster and the Z-stabilisers rings surrounding the input and output defects. Multiplying $(IC)(TUBE)(OC)$ results in an X-stabiliser of the cluster.

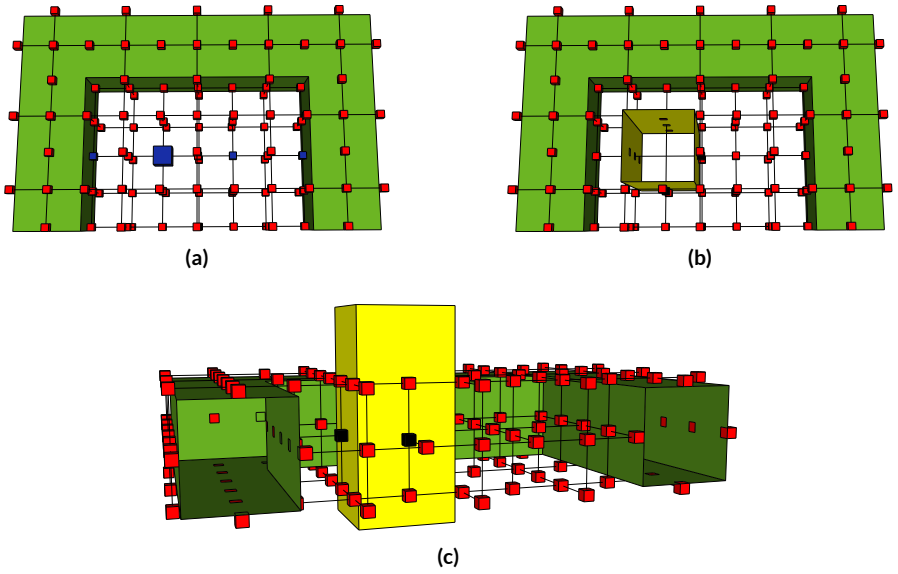


Figure 4.5: The effect of braids in the cluster: a) The green channel marks the path of a defect (e.g. primal) through the cluster, and the blue cluster qubits correspond to the chain of the sheet spanned between the two parallel sides of the defect; b) The oversized blue qubit is a face-qubit from the opposite space (e.g. dual), and, by removing it, a new (dual) defect is introduced; c) The blue qubit chain cannot be formed anymore because of the Z-measured qubit, but around the dual defect a ring results, indicated by black qubits, corresponding to the tube correlation surface.

4.1.6 The CNOT gate

The surface code supports the CNOT gate by braiding defects of opposite type: in TQC the logical CNOT between primal and dual qubits is implemented by braiding of defects, too. The construction of the CNOT starts from the MBQC observation that the relation between the quantum-correlations expresses the functionality of a circuit[BBD+09]. Translated into TQC, this implies showing correlation surface interactions: how tubes interact with sheets.

Figure 4.5 shows a lattice, where a primal defect (horizontal) is braided with a dual defect (vertical). In the same figure, the construction of the dual defect effectively removes a qubit required from a chain used in the definition of the primal qubit's sheet. One of the chains is interrupted, and, as a result, the sheet cannot be constructed. The construction of the primal defects will also remove a qubit required for the dual sheet. A correlation surface can still be constructed if only the tubes are *connected* to the sheets, thus relating ring-correlations to chain-correlations.

Proposition 1. Connecting a tube of a primal qubit to a sheet of a dual qubit is equivalent to a stabiliser transformation.

A primal qubit with input correlated by a tube is initialised into X_l, and a sheet corresponding to the dual qubit signals an X_l initialisation. Connecting the tube to the sheet

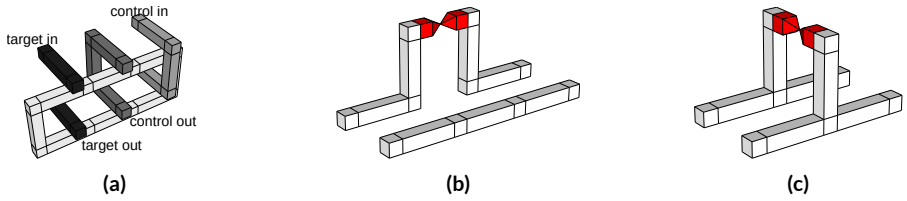


Figure 4.6: TQC logical gates: a) The CNOT between primal qubits. The white defect ring represents the dual qubit initialised into $|+\rangle$ and measured in the X-basis. The primal light grey qubit (control in) is measured in the Z-basis while the dark grey qubit (control out) is initialised into $|0\rangle$. The black pair of defects represents the primal target qubit. b) The logical R_x is based on the injection point marked red; c) The logical R_z gate.

implies that a larger correlation is constructed, where the inputs of the primal and of the dual qubit are stabilised by the two-qubit stabiliser XX. The sheet of the dual reaches its output, whereas the tube of the primal input *stops* on the sheet and the primal output is not assumed to be stabilised. Therefore, the outputs are stabilised by IX, and the stabiliser transformation is $XX \rightarrow IX$. There is a second possibility, where the inputs are stabilised by IX (the tube of the primal belongs to the output and not to the input) and the outputs are stabilised by XX, resulting in the $IX \rightarrow XX$ transformation.

At the same time, connecting a tube of a dual qubit (corresponding to Z_l) to a sheet of a primal qubit (corresponding to Z_l) equals the transformation $IZ \rightarrow ZZ$ (or $ZZ \rightarrow IZ$). Connections between tubes belonging to the same logical qubit will represent $IX \rightarrow IX$ (dual qubits) and $ZI \rightarrow ZI$ (primal qubits), where I indicates that the sheet could not be constructed due to the interrupted chain cluster stabilisers.

The sheet-tube connection scenario together with the stabiliser transformations of the CNOT gate (see Section 2.1.4) indicate that a logical CNOT between two logical qubits of opposite type is possible if the dual qubit is considered the control, and the primal qubit is the target (see Section 2.3.5). Furthermore, a CNOT between qubits of the same type is constructed by using three primal qubits and a single dual qubit and implementing in TQC the circuit identity from Figure 2.3. The resulting defect configuration is depicted in Figure 4.6a. Considering the teleportation circuits from Section 2.1.2, the same-type CNOT is formulated as: teleport the control (light grey primal) on the ancilla qubit (white dual), perform the dual-primal CNOT, teleport the control (this time on the dual) back to a new ancilla (dark grey primal).

Measurement byproducts

Quantum state teleportation is central to the TQC paradigm. The X-measurement of a cluster qubit is random, and thus the teleportation of its state has a 50% probability of success. When unsuccessful the end result requires to be corrected. In TQC the cluster qubits are measured only in the X-basis and the procedure is exactly the one from Figure 2.4a. After teleporting the state $|\psi\rangle$, the final state $|\psi'\rangle$ requires a correction if the measurement out-

put was $|-\rangle$. The measurement result $|+\rangle$ indicates a correct teleportation, and $|\psi\rangle = |\psi'\rangle$.

In the context of MBQC and TQC, it was recognised that, in order to *correct* the final state, the quatum-correlations[BBD+09] existing between the physical qubit measurements should be used. The procedure is conceptually similar to how Pauli tracking from Section 3.3 works.

Logical teleportations take place at the logical layer, and a separate kind of logical Pauli tracking is required for TQC when operating on logical qubits and stabilisers: logical stabilisers are interpreted based on the set of individual measurement byproducts of the correlation surface cluster qubits. TQC rotational gates use, for example, logical teleportations, and logical byproducts are generated by the logical measurements used in the gate constructions.

The relation between logical stabiliser at the input of a circuit and the output stabilisers is established by computing the X-parity of the cluster qubits forming correlation surfaces. Assuming the *IC*, *TUBE* and *OC* stabilisers from the previous section, let λ_i be the eigenvalue associated with the measurement of *IC*, and λ_o the eigenvalue returned after measuring *OC*. The parity of *TUBE* is computed as $\lambda_t = \lambda_i\lambda_o$. For this reason, by assuming that λ_i indicates a necessary correction and λ_i^c is the correct value, then $\lambda_o^c\lambda_i^c = \lambda_t$.

For example, when a logical qubit is initialised into $|0_l\rangle$, the used measurement pattern could indicate the $\lambda_i = -1$ eigenvalue instead of $\lambda_i^c = 1$, meaning that the logical qubit is actually $|1_l\rangle$ instead of $|0_l\rangle$. After executing the TQC circuit, the logical measurement the output could return the eigenvalue $\lambda_o = 1$ indicating that the output is $|0_l\rangle$. As the parity of the tube correlation surface (excluding the initialisation and measurement caps) connecting input and output is $\lambda_t = -1$, the correct eigenvalue at the output is $\lambda_o^c = -1$, while $\lambda_i^c\lambda_t = \lambda_o^c$. As a result, the output has to be interpreted as $|1_l\rangle$.

4.1.7 TQC circuit geometric description

Quantum computations expressed as quantum circuits (quantum networks) can be formulated as TQC circuits using the logical CNOT and the T and P single logical qubit gates. The rotational gates are teleportation-based implementations containing injected states (see Section 2.1.3). The geometry of a CNOT between qubits of the same type is presented in Figure 4.6a, and Figures 4.6b and 4.6c illustrate TQC sub-circuits of the T and P gates.

The 3D geometry of the lattice and its decomposition into unit-cells of two types indicates that a complete geometrical specification of a TQC circuit contains the coordinates of the defect endpoints (cell coordinates from *CEL*), the coordinates of the injection-points (physical qubit coordinates) and the coordinates of the input/output-points (physical qubit coordinates). The coordinate sets *IO*, for the inputs, and *IJ*, for injections, are associated to physical cluster qubit coordinates.

The major difference between injection- and input/ouput-points is that, whereas an injection state is fully specified (either $|A\rangle$ or $|Y\rangle$), the input/outputs are variable with regard to the logical stabiliser one wishes to initialise a logical qubit into. The set of injection states

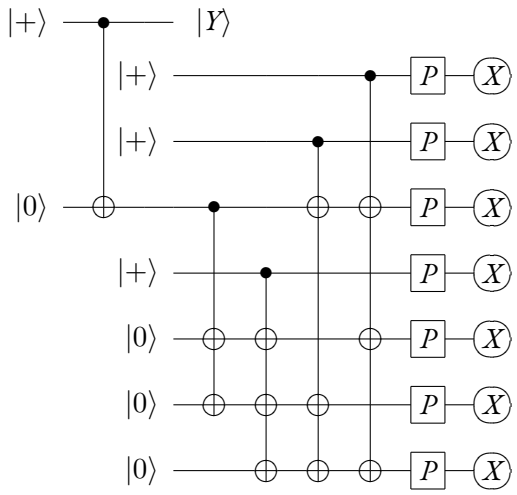


Figure 4.7: The Y-state distillation circuit requires seven P gates, each using a $|Y\rangle$, and outputs a $|Y\rangle$ with a lower error-probability.

is restricted to the previous two (TQC universality is achieved through them), but any state of the form $|0\rangle + r|1\rangle$ for $r \in \mathbb{C}$ can be used[FG09]. A TQC circuit processes the inputs specified by the user, whereas injection points are internal to the computation and are used for the construction of teleported gate applications.

The difference between the two possible defect input/output patterns is just a segment (see Figure 4.16a). The geometry is configurable at these points, as either a defect is constructed between the defect endpoints, or not. From a TQC circuit perspective, the input/output points are similar to the input/output pins of classical circuits.

The transformation of a quantum network to a TQC circuit is exemplified for the $|Y\rangle$-state distillation circuit[FD12,RHG07] from Figure 4.7, which is transformed into the circuit from Figure 4.8a. The primal qubits are colored white, and the dark grey qubits are the duals necessary for the CNOTs. The CNOT controls are recognisable by measurement geometry of the control-in and the initialisation geometry of the control-out qubits. The P gates are implemented as $R_z(\pi/2)$ and require an $|Y\rangle$ injection state ancilla, represented by the colored pyramid structures. The tip of the pyramids indicates the physical injected cluster qubit and the gradual size of the pyramid basis indicates the gradual construction of the defects (similar to how short logical qubits were constructed in the surface code). At the right of the circuit the primal ancillae qubits are measured in the X_l-basis.

The *volume* of a TQC circuit or its primitives is the number of required unit-cells in the lattice supporting the geometry. For example the logical CNOT gate has volume 12 [RHG07,FD12], and the circuit from Figure 4.8a has volume 272, while another version of this circuit presented in [FD12] has the volume of 192.

TQC computations are represented by topological properties of the geometries, and TQC circuits with equivalent topologies are computationally equivalent. The braiding re-

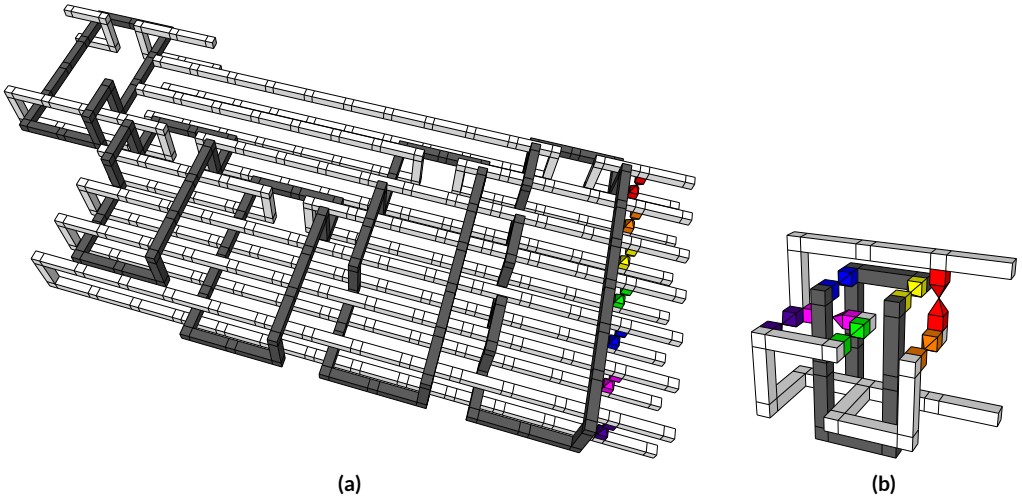


Figure 4.8: The TQC circuits for Y-state distillation: a) The straightforward translation of the quantum network from Figure 4.7; b) The compact version of the same circuit.

lation between the defects is one of the topological properties, as each braid is a CNOT representation. This is, for example, the case for the circuits in Figures 4.8a and 4.8b, where the same computation is expressed using topologically equivalent geometries. Being an MBQC derivation, the function of a TQC circuit is also determined by the type and the measurement order of the injection points.

TQC circuit *compaction*, as presented in [PFI3], is an option for reducing the volume of the required lattice. The method tries to reroute the defects, such that the *unused lattice* volume between them is minimised. An example of compactifying the $|Y\rangle$-state distillation circuit to an equivalent volume of 18 is shown in Figure 4.8b [FDI12]. Note that the CNOT gates were clearly visible in the initial circuit, but this is not the case in the compact circuit, although the computation is the same. Compactification changes the geometry but not the computation (topology).

4.1.8 Design of TQC circuits

The design of TQC circuits is formulated starting from a design stack as illustrated in Figure 4.9, which consists of several abstraction levels that differ from the ones used in classical circuit design. A given high level quantum algorithm is first decomposed into a quantum circuit, which does not include any QECC protocols. There are multiple possible QECCs to chose from, and each protection will lead to circuits having different number of qubits and/or depth. In TQC, each qubit identified in the circuit is logically encoded using the surface code. Operating with logical qubits requires referring to the *physical qubits* of the underlying lattice.

In general, QECCs restrict the types of operations supported on logical data, forcing an

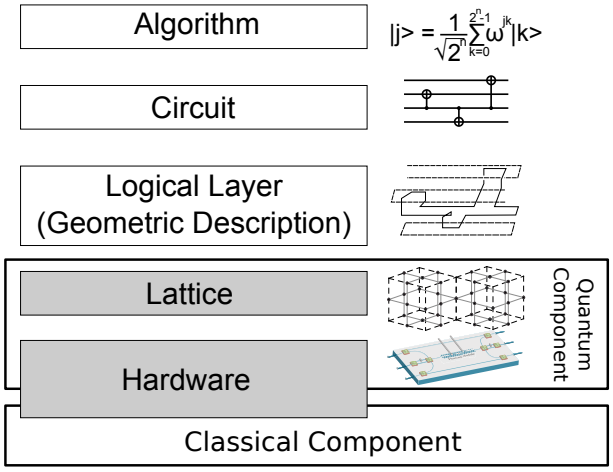


Figure 4.9: The TQC design stack.

arbitrary quantum circuit to be decomposed into gates from a discrete universal quantum gate set. Once these decompositions are complete, the resulting TQC circuit can be optimised with respect to the physical resources (e.g. equivalent volume). Afterwards the optimised circuit is translated to physical operations (e.g. measurement commands) sent to the hardware.

The lowest layer in the TQC design stack is the hardware layer, which is responsible only for producing a generic 3D lattice of qubits. The quantum hardware is controlled by a classical component (computer). As a result, programming in the TQC model is separated from the basic functionality of the quantum hardware.

In conclusion, designing TQC circuits is a layered approach, and the automated design methods presented in this work enable the transitions between the stack layers. Circuit synthesis transforms a quantum network into a geometric description, circuit mapping generates a 3D lattice based on the geometry, while circuit verification is used for checking that a TQC circuit specification is supported by the geometrical description.

4.1.9 Circuit specification

TQC circuits are specified starting from their geometric description and the IO/IJ sets (see Section 4.1.7). The circuit specification includes the measurements temporal order required for the correct execution of the computation.

The construction of a TQC circuit starts with the construction of the lattice, which is a graph-state resulting after using *CPHASE* to entangle qubits initialised into the $|+\rangle$ state. Hence, the lattice construction procedure is a stabiliser circuit: both the qubit states and the action of the entangling gates are representable in the stabiliser formalism. The measurements of the lattice qubits are either in the Z-basis (defects), X-basis (teleportation, parities) or in a rotated basis (injection points, rotational gates).

out	1(R)	2(O)	3(Y)	4(G)	5(B)	6(I)	7(V)
X			X		X	X	
	X		X		X		X
		X	X			X	X
Z	Z	Z	Z				
				X	X	X	X
		Z	Z	Z	Z		
	Z		Z	Z		Z	
	Z	Z		Z			Z

Table 4.2: The state stabilisers computed by the circuit from Figure 4.7 before the P gates are applied [FD12]. The qubits numbered from 1 to 7 correspond to the quantum wires counted downwards. The initials in the parantheses refer to the colors associated to the injection points.

The non-stabiliser behaviour results from the rotated measurements, as the order of the X/Z measurements does not dictate the performed computation: these transform stabiliser states (graphs) into stabiliser states (graphs with modified topology). The process of lattice initialisation followed by the non-rotated measurements is equivalent to the preparation of a cluster state to be used in an MBQC setting. The actual computation is performed by the rotated measurements, and their temporal ordering is comparable to the temporal ordering of the non-stabiliser gates from the initial quantum network.

At the error-corrected layer, the application of rotational gates (rotated measurements) is extended into fault-tolerant gate applications, where circuits from Figure 2.5 replace non-fault tolerant gate instances. During this construction, ancillae initialised into $|A\rangle$ or $|Y\rangle$ are introduced, and the number of circuit qubits is increased. However, two additional subcircuits are required to support the fault-tolerant T gate.

The T gate application may require a P gate correction (see Section 3.3), and the gate subcircuit should contain some mechanisms to support the dynamic application of it. Otherwise, during the circuit's execution the P gate will have to be dynamically inserted into the gatelist (see Section 3.2.2). The dynamic modification of the TQC computation's geometry shows that insertion is not feasible. In [Fow12] the solution is offered in the form of *selective source* and *selective destination* teleportations (see Figure 4.10).

Representing a TQC circuit as a fault-tolerant construct is a straightforward approach. For example, the T gate implementation is presented in Figure 4.11. A first conclusion of this construct is that error-corrected (logical) circuits can be implemented in three steps: firstly, the qubits are initialised into one of the states $\{|0\rangle, |1\rangle, |A\rangle, |Y\rangle\}$; secondly, a network of CNOTs is applied which includes the computational part and the correction mechanisms (selective teleportations); and thirdly, the logical qubits are measured. As logic qubits could be measured in a rotated basis, the measurement order of the logical qubits includes the measurement order of lattice injection points (e.g. Figure 4.7).

The specification of a circuit follows from the above observation: the logical stabilisers that exist right before logical measurement of injection points can be used to fully specify a

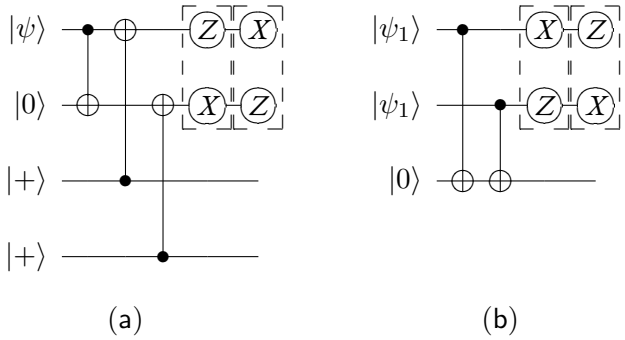


Figure 4.10: a) Selective destination teleportation: the first group of measurements will teleport $|\psi\rangle$ on the third qubit, while the second group of measurement will teleport the state to the fourth qubit; b) Selective source teleportation: the first group of measurements will select $|\psi_1\rangle$ for teleportation on the third qubit, while the second measurement group will teleport $|\psi_2\rangle$ [Fow12].

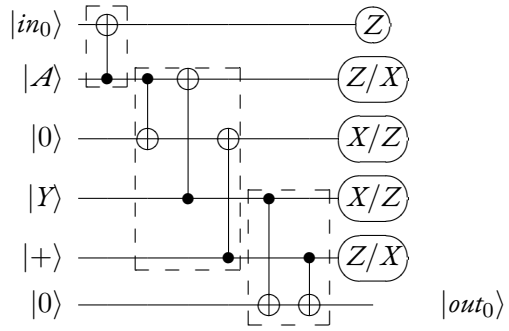


Figure 4.11: The fault-tolerant implementation of the T gate including the probable P correction. The circuit contains selective destination and selective destination teleportations, and the fault-tolerant P gate implementation.

circuit. These stabilisers result from the input stabilisers being transformed by the network of CNOT gates. For the circuit from Figure 4.7, the stabilisers before the application of the logical rotated measurements (application of the P gate followed by X-measurement) is presented in Table 4.2. In order to include also the injected qubits (if the P gate had been applied fault-tolerantly), the stabiliser table of the specification would have been larger.

The specification of a TQC circuit includes a logical stabiliser table that represents the circuit state after the CNOT part was executed and before any of the measurements are performed. The specification is formulated as the tuple $QCS = \{ST, IO, IJ, \mathcal{M}\}$, where ST is the stabiliser table, and $\mathcal{M}$ is the ordered set of measurements of the injection points.

4.2 Validity of correlation surfaces

The design of TQC circuits requires a better understanding of correlation surfaces and their properties. The key property to be investigated in this section is the validity of correlation surfaces. The following analysis is based on the introduction provided in [FGo9], and is further

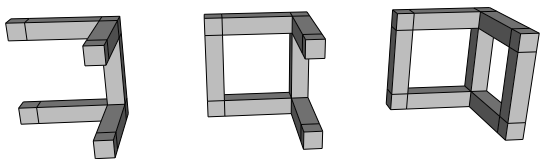


Figure 4.12: Possible junction configurations: a) An open defect structure, where both pairs of defects are not connected; b) An open defect structure, where one pair of defects is not connected; c) A closed defect structure.

augmented by case discussions, definitions and illustrations of the lattice.

There are two TQC-specific constructs that can influence the validity of correlation surfaces: the braids between defects of opposite type and defect *junctions* (e.g. see Figure 4.12), which are an example of non-linear topology. Thus, braids and junctions will be the key elements used when analysing the validity of surfaces. Defects containing junctions will be refered to as *defect structures*, and the distinction between *open* and *closed* defects (structures) will also play a role: an open defect (structure) has endpoints, whereas a closed defect (structure) contains no endpoints.

The existence of two self-similar structures in the lattice (the primal and the dual lattice) is the support for defining tubes and sheets in both spaces (primal/dual tubes and primal/dual sheets).

Definition 10. A surface is primal (or dual) if it resides in the primal (or dual) space.

This distinction made by Definition 10 avoids ambiguities when discussing primal/dual logical qubits and their associated correlation surfaces. When discussing the validity of individual tubes, sheets or combined constructs consisting of both, it will be assumed that the surfaces belong to the same space. As mentioned previously, each physical cluster qubit is stabilised by $X \bigotimes Z_n$, where n indicates the neighbouring entangled qubits, and each correlation surface S is a stabiliser defined over a set of cluster qubits $Q(S) \subset TQCC$.

Definition 11. A valid correlation surface is an X-stabiliser of the underlying physical cluster qubits[FG09].

The analysis of valid correlation surfaces will proceed from Definition 11, which implies that all valid geometrical constructs representing correlation surfaces can be enumerated, due to the restricted 3D-geometry supported by the cluster. For the construction of a surface, the set of cluster qubits has to be chosen such that the stabiliser $S = \bigotimes X_i$, for $i \in Q(S) \subset TQCC$ is valid according to the previous definition.

Surfaces were defined using rings or chains of X-stabilisers, and surface validity will be analysed also from their X-measurement parities perspective. Measuring all the non-defect qubits in the X-basis implies that the results of surface-qubit measurements will be one of the two eigenvalues (1 or -1) corresponding to one of the cluster qubit states $|+\rangle$ or $|-\rangle$.

Proposition 2. The parity of a correlation surface is the parity of the X-measurement results of the qubits forming the surface.

For a surface S defined over the physical cluster qubit set $Q(S)$, the function $mpar : Q(S) \rightarrow \{-1, 1\}$ is computed as the product of the resulting measurement eigenvalues $mpar(S) = \prod_{q \in Q(S)} \mathcal{M}_X(q)$. If $mpar(S) = -1$, then the surface has odd parity, and $mpar(S) = 1$ indicates an even parity.

Proposition 3. A valid correlation surface has even parity in the absence of cluster qubit errors.

The measurement of a surface-qubit computes the Z parity of its neighbouring entangled cluster-qubits, and each cluster unit-cell will have even parity when completely X-measured. For a cluster cell having 6 face-qubits (each face-qubit with 4 neighbouring edge-qubits), there are 12 side-qubits, and the cell-parity contains twice the eigenvalues $z_e \in S^{p/d}$. For simplicity, consider a single unit-cell cluster where each edge-qubit has two neighboring face-qubits. In an arbitrary cluster, an edge-qubit has either 2 or 4 neighbouring face-qubits.

$$mpar(cell) = \prod_{q \in Q(cell)} \mathcal{M}_X(q) = \prod_{f=1}^{6} (\prod_{e=1}^{4} z_{ef}) = \prod_{e'=1}^{12} (z_e'^2) = 1 \tag{4.1}$$

An informal visual representation of correlation surfaces (including the initialisations and measurements) is possible by considering these deformations of a unit-cell: tubes are visualised as extended cells, and sheets are planarised cells (having their boundary on defects). Generally, the implication of Proposition 3 is that, although a surface S can have either even or odd parity, only an even parity surface is valid.

A short reinspection of Definition 11 from the perspective of even parity indicates that, if the function *par* for a surface S is not always even, then the surface stabiliser will be of the form $(\bigotimes_{q \in Q(S)} X_q)(\bigotimes_e Z_e)$. The Z_e stabilisers belong to cluster qubits that are left *uncancelled* (e.g. Figure 4.5a), while the cluster qubits indicated by the Z_e stabilisers do not belong to $Q(S)$.

$$par : Q(S) \rightarrow TQCC \setminus Q(S), \text{ for } Q(S) \subset TQCC \text{ and } S \text{ correlation surface} \tag{4.2}$$

The function *par* is introduced (Equation 4.2), and it returns the set of qubits from $TQCC$ for which the Z_e do not cancel. These cluster qubits are indicated by the multiplication result of the stabilisers associated to qubits from $Q(S)$. For even parity (when $mpar = 1$) the function will return the empty set $\emptyset$. When referring to two surfaces (S, T) having the same parity, the equality between the outputs of the *par*-function will be referred to $(par(S) = par(T))$.

Junctions

The analysis of geometric structures is initiated for the validity criteria of unbraided defects. An arbitrary defect structure (unless it is a closed defect) has at least two endpoints, and each

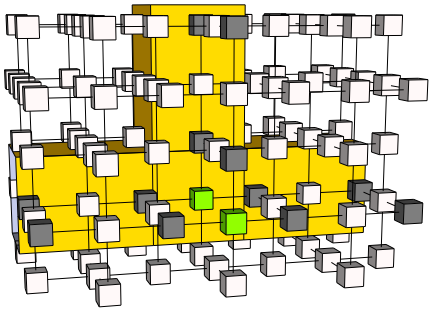


Figure 4.13: Forming sheets at junctions. The yellow channel marks the defects constructed through a cluster. The grey qubits indicate parts of sheet chains starting from the defects, and the green qubits indicate that if a sheet splits the junction, then the resulting cluster stabiliser would be invalid. Physical qubits are locally stabilised by XZ_{ngh}, and the multiplication of three grey qubit stabilisers will result in the Z of the green qubits being uncancelled. According to Definition 11 this is not allowed.

defect-junction introduces a supplemental endpoint. In conclusion, for any defect containing j junctions along its path there are *at most* $e = 2 + 2 \times j$ endpoints, where every junction introduces three supplemental endpoints. In order to comply with its definition (see Definition 11 and Equation 4.2), a tube *must* reach all the e endpoints of the defect. Thus, when starting to construct a tube from any endpoint, the tube is always being *split* at a junction and the construction proceeds towards the remaining endpoints.

Valid sheets expose a different behaviour around junctions: a sheet must have all its boundaries on defects (see Figure 4.4 and Definition 11). As a result, valid sheets are defined only for closed defect structures. In contrast to tubes, a single sheet cannot be split along a junction (see Figure 4.13), because the parity of the sheet will not be even (*par*-function will return $\neq \emptyset$). As long as closed defect structures are joined at junctions (maximum number of junction endpoints is not reached), junctions are the support of multiple sheets.

Braids

Braids introduce the possibility to *connect* tubes and sheets, and the four valid constructions are enumerated in Figure 4.14. A sheet (e.g. S) will interact with two separate tubes at a braiding-point (e.g. a, b), and, considering that there are no other braids defined at this sheet, let $par(S) = par(a) = par(b)$. The four even parity surface constructions, for example $par(Sa) = \emptyset$, represent all possible valid surfaces.

The construction of an even-parity sheet involved in more than two braids requires separate attention. Let S be a sheet involved into n braids, and $par(S) = \bigcup_{i=0}^{n} par(s_i)$ be the set of all the physical qubits involved in the parity of S. For each braid $par(s_i)$ indicates the set of physical qubits resulting at intersection between a sheet and the defect (e.g. the black qubits in Figure 4.5c). Around each braid there are defined two tubes a_i and b_i, and it follows that $par(b_i) = par(a_i) = par(s_i)$. For constructing an even parity sheet S, each s_i will have to be cancelled by either a_i or b_i.

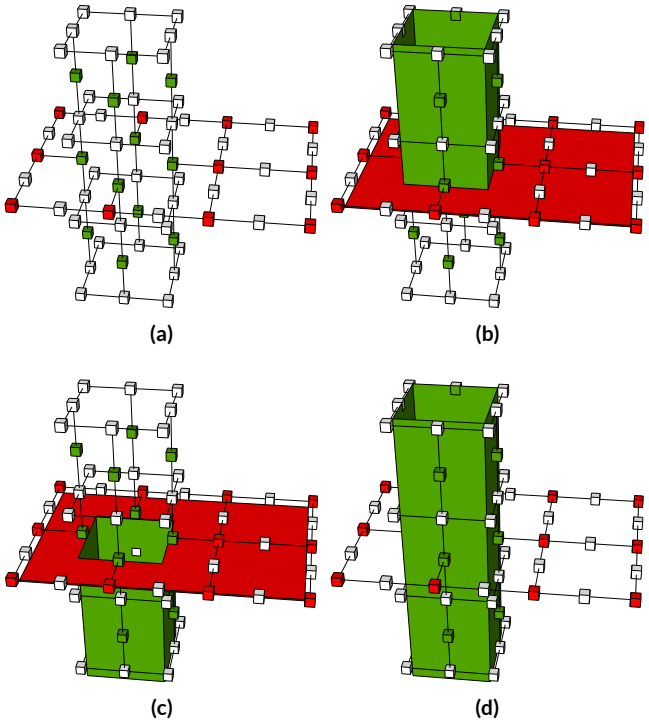


Figure 4.14: Fragment of a cluster illustrating a sheet (red qubits) and a tube (green qubits) after an apparent braid. Following the discussion from Figure 4.13, there are four possibilities to construct valid correlation surfaces: a) Both the sheet and the tube are not constructed; b) The upper part of the tube together with the sheet is connected; c) The lower tube is considered together with the sheet; d) The complete tube is considered.

It is impossible to construct an even-parity sheet that is connected to both tubes around a braid. The corollary follows: if two tubes are connected at a braiding-point, the involved sheet cannot be constructed.

Further surface operations

the construction of valid surfaces in the presence of junctions of braids was presented in the previous section. At the same time, surfaces are cluster stabilisers which are modified through *surface addition* even in the absence of junctions or braids.

Definition 12. The addition of two surfaces S_1 and S_2 is the result of multiplying the two corresponding stabilisers.

$$S_r \quad = \quad S_1 + S_2 = (\bigotimes X_i)(\bigotimes X_j) = \bigotimes X_{i \neq j}, \text{for } i \in Q(S_1), j \in Q(S_2) \qquad (4.3)$$

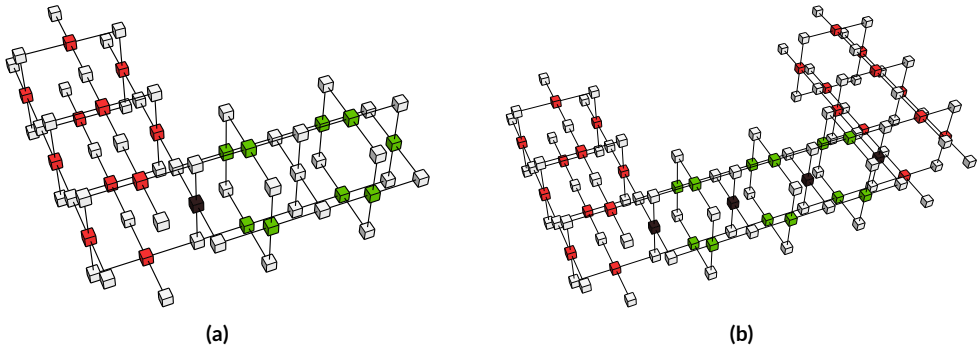


Figure 4.15: Tube extrusion examples. The red qubits indicate the tube surrounding a defect. The black qubits indicate cluster qubits that would be measured if the defects were extended. The green qubits correspond to the extruded surface without measuring the black qubits. a) A single tube is extended; b) Two tubes are added without constructing a defect. If the black qubits were Z-measured, the previous two defects would be bridged. No assumption is made if bridging is correctly applied or not.

The correlation surface sum S_r will be defined over the set of qubits $Q(S_r)$, which is the symmetric set difference Δ of $Q(S_1)$ and $Q(S_2)$, as the individual X stabilisers of qubits from the set union will cancel out ($XX = I$).

$$Q(S_r) = Q(S_1)\Delta Q(S_2) = Q(S_1) \cup Q(S_2) \setminus (Q(S_1) \cap Q(S_2))$$

Defects *enforce* the existence of surfaces, meaning that for a given defect a valid surface has to exist. Any X-stabiliser of the lattice is a valid tube, and *trivial surfaces* can be constructed in the absence of defects. For example, for a single unit cell cluster, the trivial surface contains all the 6 face-qubits. Furthermore, a two unit cell cluster contains 11 cell faces, where 10 of these are not shared between the cells, and the largest trivial surface contains the corresponding 10 face-qubits.

Surfaces can be arbitrarily deformed without changing their topology making tubes and the sheets not unique in an arbitrary defect structure. The deformations are the result of using surface addition, where a *trivial surface* is added to the initial surface. Considering trivial tubes, the basic surface deformation operation is tube extrusion in the absence of defects (see Figure 4.15a).

Disjoint surfaces can be connected through surface addition, and of interest are the sheet-to-sheet and the tube-to-tube cases. Connecting disjoint surfaces is performed by arbitrary tubes that do not have any defect support (see Figure 4.15b). This situation is orthogonal to when sheets are connected by braiding. While in the pure-addition case the sheets are considered separately, in the braiding-case this is impossible. Similarly, disjoint tubes are connected by adding a third tube-surface without a defect support. Again, the tubes are independent and there is no junction between them.

A particular case of the tube-to-tube connection is bridging, where two disjoint tubes are connected through a third tube *with* defect support. As a result, a junction is introduced.

This surface operation was presented in [FD12], and can be performed between any two defect structures, where at least one of them supports a sheet (closed defect structure). After bridging a closed with an open defect structure, the resulting defect structure will be open. Although not obvious, the resulting defect structure includes the support for the same sheets as the initial two structures [FD12].

Surface addition in the absence of defects is the equivalent operation (at the logical layer) to multiplying the generators from a stabiliser table (see the example in Table 4.1c,d, where $(ZXZ)(IXI) = ZIZ$). Without applying any gate, the stabilisers are left unchanged, and by multiplication only another set of generators of the stabiliser group is computed. Therefore, in the search of valid surfaces, only those with a defect support are relevant, as junctions and braids between defects imply the application of logical quantum gates. The following sections of this chapter will concentrate only on this situation.

4.3 Graph representation of TQC circuits

The geometrical description of a TQC circuit can be represented by graphs, which serve as a foundation for automatic TQC design methods. The properties of arbitrary defect structures containing braids, junctions and *bridged geometries* (resulting after bridging) will be more easily investigated from a graph perspective.

A TQC circuit will be represented by the set of two graphs $TQC^g = \{G_p, G_d\}$, where G_p (G_d) describes the geometry existing in the primal (dual) space. The dual graph will be denoted as the graph of *opposite* type to the primal graph, and vice versa (e.g. G_p is the opposite of graph G_d).

For a given circuit the graph $G_t \in TQC^g, t \in \{p, d\}$ is defined as the tuple $G_t = (V_t, E_t)$, where V_t is the set of nodes (vertices) and E_t is the set of undirected edges defined as pairs $(n_1, n_2) \times (n_2, n_1) \in V_t \times V_t$. The set of nodes V_t abstracts the cluster coordinates of defect endpoints, junction points, injection points, input/output points and braiding points.

The set of nodes $IO_t \subset V_t, IO_t \subset IO$ abstracts the injection and input/output points from the original circuit, where IO is part of the circuit specification and is the set of both injection and input/output points. The set corresponding to the injection-points is $IJ_t \subset IO_t$. The set of edges E_t abstracts the defect-segments connecting all the points mentioned before.

A graphical example for a graph abstracting defects is offered in Figure 4.16a, where the defect geometries for initialisation and measurement are illustrated as subgraphs. Figure 4.16b illustrates an example of a primal-primal CNOT graph. The blue nodes are the input/output nodes from the set IO_t, the white nodes correspond to geometric junction points, and the red and green nodes to braiding nodes. The figure contains both graphs existing in TQC geometry (primal and dual) and grey edges indicate the dual graph, while red nodes correspond to braids of the primal edges with sheets spanned by dual geometric rings.

The graphs are initially constructed without the braiding-nodes, which require a separate

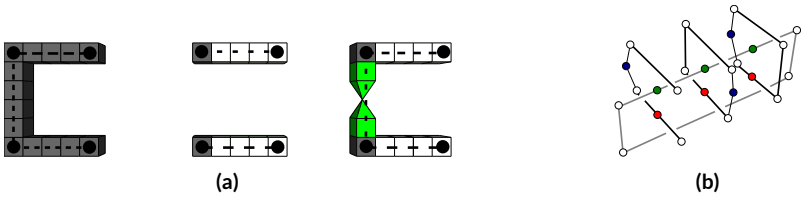


Figure 4.16: Elements of the graph representation: a) The initialisation/measurement geometries are abstracted through graph nodes and edges; b) The primal-primal CNOT circuit represented as a graph.

calculation step explained in Section 4.5. The next algorithms assume that this step has been performed and correct braiding-nodes are part of the graphs.

The lattice coordinates of the nodes $V_t \setminus IO_t$ are related to cells from the set CEL (see Section 4.1.2). There is a function $coord$ between these nodes and CEL: the function $coord(k)$ yields the three $(w/h/t)$ lattice coordinates of the cell corresponding to k. The inverse function $coord^{-1}$ returns the node at the specified lattice coordinates.

4.3.1 Properties

Surfaces play a central role in the functionality of a circuit, and graph-representations of TQC circuit geometries are used for the analysis of correlation surface support. Surfaces of both types (tubes and sheets) are enforced by the presence of defects, which are abstracted through graph-edges. The construction of valid tubes results in the defects being abstracted as tree-shaped subgraphs. On the other hand, valid sheets are bounded by closed defect structures, and this implies that the analysis of sheets will be based on the investigation of *subgraph-cycles*.

After traversing G_t, the set $CMP(G_t)$ of all connected components is computed. A connected component is a subgraph for which all the nodes are connected through edges. In addition, for each component $m \in CMP(G_t)$, the set $CY(m)$ of graph-cycles is the possible support for logical qubits by m.

Let the set of graph-cycles be $CY(G_t) = \bigcup_{m \in CMP(G_t)} CY(m)$. Each cycle is a subgraph of G_t and is described by $c = (V_t^c, E_t^c), V_t^c \subset V_t, E_t^c \subset E_t$.

Braiding-nodes

Braids are particularly important for the discussion as the braiding-nodes of a primal graph are a reference to the sheets supported by the graph-cycles from the dual graph, and vice versa. For example, the cycle $cd \in CY(G_d)$ in the primal graph G_p is used for spanning the sheet $sheet(cd)$, and is abstracted through braiding-nodes in the dual graph G_p.

The computation of the braiding-nodes from G_p starts with the sheet-finding procedure from Section 4.5.3: for the cycles $cd \in CY(G_d)$, the correlation-surfaces $sheet(cd)$ are computed. The braiding-nodes abstract the intersections between the defects from the geometri-

cal description of G_p with the computed sheets for G_d. The intersections can be computed, for both the defects and the sheets are sets of lattice qubits.

The set of braiding-nodes is a subset of the graph-nodes, $B_t \subset V_t$ where $B_t \cap IO_t = \emptyset$. The relation between braiding-nodes from $G_{t'}$ and sheets from the graph G_t is the following:

$$\begin{aligned} \forall \text{ cycle } c \subset G_t, G_t \in TQC^g, t \in \{p, d\} \\ b \text{ braiding point on } \mathit{sheet}(c) \\ b \in B^{t'}, B^{t'} \subset V^{t'} \text{ where } G_{t'} \in TQC^g,\ t \neq t' \end{aligned}$$

A graph G_t will contain input-nodes with exactly two incident edges, junction-nodes with at least three incident edges, and braiding-nodes. The braiding-nodes will influence the construction of surfaces. Initially, for any braiding-node there are two incident edges, one abstracting each tube interacting with the sheet. However, connecting two tubes at a braiding-point renders the involved sheet as impossible to construct, but multiple braiding-nodes can point to a common sheet. As a result, connecting both tubes at a braiding-node implies, for consistency reasons, that the same construction has to be made at each related braiding-node. It is reasonable to assume that related braiding-nodes are connected in the graph, thus increasing the number of incident edges. Edges connecting related braiding-nodes are called *relation-edges*.

Number of tubes

For a given graph component m, there are at most $(b - 1)$ valid tubes, where b is the number of braiding-nodes in the component. When $b = 1$, it is not possible to construct two valid tubes at the same time. This would conflict with the validity criteria enumerated in Section 4.2.

Number of sheets

The number of sheets supported by a geometrical description is a function of the number of connected components $|CMP(G_t)|$ in the graph, and the number of graph-cycles each component $m \in CMP(G_t)$ contains.

It is challenging to determine the number of sheets supported by defect structures constructed by bridging. Finding out the number of supported sheets is performed by searching all the cycles in the component. However, surface addition between two supported sheets can result in another supported sheet, as it was the case when geometric defect junctions were discussed. The minimum and maximum number of sheets in a component will be investigated in the following.

The maximum number is computed by considering n separate rings (one cycle): there are 2^n possible combinations of the sheets being constructed or not, as each cycle will support

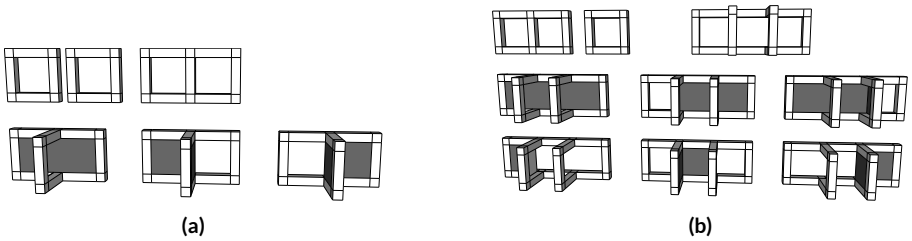


Figure 4.17: The number of cycles in a closed defect structure is equivalent to the number of supported sheets. The figure enumerates all the possible valid sheets for defect structures resulting after: a) Bridging 2 rings; b) Bridging 3 single rings.

a sheet. After bridging the n rings together, the resulting defect structure will still have to support the 2^n separate sheet constructions. However, the number of cycles in the resulting structure will be greater than n.

The computation of the minimum number of cycles starts from the same n rings, but this time visualised as aligned on the horizontal. After bridging the first two rings $m_1, m_2 \in CMP(G_t)$ (where $|CY(m_1)| = |CY(m_2)| = 1$ cycles), the result will be a single component m_r^2 for which $CY(m_r^2) \geq 3$. Bridging m_r^2 again with the third ring m_3, containing $|CY(m_3)| = 1$ cycles, will result in m_r^3 having $CY(m_r^3)| \geq 6$ (see Figure 4.17). By induction, it follows that after bridging n rings, the resulting component m_r^n will have $|CY(m_r^n)| \geq n(n+1)/2$.

The set of sheets supported by $CY(m_r^n)$ together with the sheet-addition operation forms a group having a generating set of size at least $\sqrt{|CY(m_r^n)|}$. The sheets generator set is further used instead of the complete set. The reduced number of defect-sheet intersections to be computed leads to less braiding-nodes in the opposite graph $G_{t'}$ ($t \neq t'$). All the supported sheets can still be determined, and the generality of the graph-representation is not affected.

4.3.2 Relation to stabilisers

Correlation surfaces, according to Definition 9, connect inputs with outputs, and this property is translated to the graph-representation of circuits. The functionality of a TQC circuit is expressed as transformations of input stabilisers into output stabilisers (the stabiliser table ST from Section 4.1.9). The inputs and outputs of a circuit are abstracted as the IO_t set of graph-nodes.

The stabilisers are mappings of correlation surfaces derived from the circuit graph, but stabiliser transformations from the table ST can be represented as graphs, too. The derivation of correlation surfaces started from the relation between defects (graph-edges) and tubes, and continued with the relation between closed defect structures (graph-cycles) and sheets.

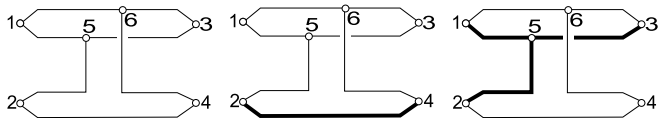


Figure 4.18: A graph representing a possible (but incorrect) CNOT implementation. The nodes $\{1, 2\}$ are for the inputs, and $\{3, 4\}$ for the outputs.

For every node $b \in IO_t$ there will be two associated edges that support the possible tubes, and the interpretation of tubes depends on the type of the geometry (and graph): X-correlation for primal defects, and Z-correlation for the duals. For associating stabilisers with correlations the *spec* function will indicate, based on the type of graph, what kind of correlation is required for a given logical stabiliser: a tube in the primal or dual space, or a sheet in the primal or dual space.

$$spec : IO_t \times \{X, Z\} \rightarrow \{tube_p, tube_d, sheet_p, sheet_d\}$$

The stabiliser transformations described in the ST have to be first interpreted in terms of correlation surfaces. For the classical $CNOT$ example, a TQC circuit implementing it will have to support the $IX \rightarrow IX$ transformation (control is the first qubit, target the second one), which equates to transforming $spec(X) = tube_p$ at target input into $spec(X) = tube_p$ at target output, if the target qubit is in the primal space.

The graph contains two edges starting from an input-node associated to the circuit's input, and care has to be taken such that the graph supports the tube-transformation along both paths that start from the input. This is required due to the manner how logical qubits are initialised to stabilisers whose correlations are expressed as pairs of tubes: the measurement/initialisation pattern is a pair of *caps*.

The discussion is illustrated by Figure 4.18 in which the graph is assumed to represent a $CNOT$. The $IX_2 \rightarrow IX_4$ transformation is interpreted as the correlation surface connecting *only* the input-nodes $2, 4$. Only two possible paths can be constructed from node 2: $\{2, 4\}$ (the lower path in the graph) and $\{2, 5, 1, 3\}$, and there is no other way to connect 2 and 4. The input-tubes cannot be completely transformed into output-tubes, and, thus, the assumption that this circuit implements a $CNOT$ is wrong. The $\{2, 5, 1, 3\}$ path represents the $X_1X_2 \rightarrow X_3I_4$ stabiliser transformation. The latter transformation is not specific for a CNOT.

Proposition 4. In a valid geometric description graph, a tube-correlation surface is supported iff for any input node, which is required to support tubes according to ST, there are at least two paths leading to the ouput nodes that have to support tubes according to ST.

4.4 Boolean representation of TQC circuits

Valid correlation surfaces will be represented based on the analysis from the previous section. A first approach uses Boolean expressions formulated in conjunctive normal form

(CNF), which is a conjunction ($\wedge$) of clauses, where each clause is disjunction ($\vee$) of literals. A literal is either a positive variable (e.g. x), or a negated variable (e.g. $\neg x$). A *unit* clause contains a single literal.

The CNF Boolean expression $f = (x \vee y) \wedge (\neg x \vee \neg y)$ contains two clauses (the two expressions in the parantheses), where x and y are the variables. For the previous expression f is *true*, for example, after assigning $x = true$ and $y = false$. In general, Boolean expressions formed in CNF are used for determining a variable assignment such that the expression evaluates to *true*. This problem is known as SAT (*Boolean satisfiability*) and is an NP-complete problem. There are, however, situations in which the problem is easily solved for particular types of CNFs. Some examples will be offered in this work.

Boolean expressions are constructed by choosing the necessary variables for the literals. For each surface type there will be a corresponding variable type: *tube-variables* and *sheet-variables*. The variables used for the literals are inferred starting from the TQC^s set of graphs, after having computed the set of components $CMP(G_t)$, $G_t \in TQG^s$.

The result of mapping the TQC^s circuit-graph to Boolean expressions will be the set $TQC^b = \{E_p, E_d\}$, where E_p is the Boolean expression for the geometries in the primal space, and E_d for the dual space.

4.4.1 Types of variables

The Boolean variables are computed using graph-based searches on each component $m \in CMP(G_t)$. Both variable types will represent subgraphs: the tube-variables abstracting trees of the graphs, and the sheet-variables abstracting cycles of the graphs. The naming of the variables will be standardised: capital letters for sheet-variables and small letters for tubes.

Tube-variables are selected by traversing m and building trees having their braiding-nodes as leaves. Tubes have to reach all the endpoints of an associated defect structure, and the tree structure arises as an effect of the junction-nodes in the graph (see Section 4.2). Braids introduce a supplemental relation between tubes and sheets, and the traversal ends when a braiding-node is found. In a component containing b braiding-nodes there are b possible tubes (see Section 4.3.1) and each one has a tube-variable assigned.

The introduction of sheet-variables is supported by the braiding-nodes existing in m, as each braiding-node from $m \subset G_t$ is a direct reference to a cycle from the opposite graph $CY(G_{t'})$ (see Section 4.2). For a component m with b braiding-nodes, there will be at most b distinct sheets, as some of the braiding-nodes could represent the same sheet. Thus, after $CY(G_{t'})$ was computed, each cycle has a corresponding sheet-variable. To the braiding-nodes, that result on the sheet, the variable of the sheet is assigned.

Following the distinction made between the braiding- and the input-nodes of a graph G_t, a similar distinction is made between the variables: input and non-input variables.

Definition 13. An input variable (tube or sheet) represents a subgraph of G_t that contains at least one node from IO_t.

4.4.2 Forming the clauses

The mapping of TQC^g graphs to CNF Boolean expressions continues with the introduction of the clauses which describe the valid combinations of tubes and sheets. These are *generated* starting from the braiding-nodes of $G_t \in TQC^g$.

A variable of the Boolean formula will be *true* if its associated tube or sheet is present (enabled) and *false* if it is absent (disabled). For example, if the sheet S is enabled, either the tube a or the tube b will be connected to S. Otherwise (S is disabled, represented using the literal $\neg S$), either both a and b are connected resulting in the tube $a \wedge b$, or both a and b are disabled, which is expressed as $\neg S \wedge \neg a \wedge \neg b$. The Boolean expression $B(S, a, b)$ abstracting a braiding-node models the four valid possible surface constructions:

$$B(S,a,b) = (S \wedge a\neg \wedge b) \vee (S \wedge \neg a \wedge b) \vee (\neg S \wedge a \wedge b) \vee (\neg S \wedge \neg a \wedge \neg b) \tag{4.4}$$

$$= (S \vee a \vee \neg b) \wedge (S \vee \neg a \vee b) \wedge (\neg S \vee \neg a \vee \neg b) \wedge (\neg S \vee a \vee b) \tag{4.5}$$

$$= S \leftrightarrow (a \oplus b) \tag{4.6}$$

4.4.3 Complete Boolean expressions

The expression E_t is constructed for a graph G_t, containing the set B_t of braiding-nodes, by forming the conjunction of all the braiding-node-clauses for $n \in B_t$:

$$E_t = \bigwedge_{n \in B_t} B(S,a,b), \text{ where } S \text{ sheet-literal and } a, b \text{ tube-literal} \tag{4.7}$$

If G_t contains no braids, then $E_t = \mathit{true}$. The Boolean expression E_p (for the primal graph G_p) and the Boolean expression (E_d for the dual graph G_d) could be combined to a single Boolean expression. A correlation surface cannot be constructed such that it is spanned in both the primal and the dual space and the two expressions are defined over distinct sets of variables.

4.5 Construction of correlation surfaces

Constructing a valid correlation surface is equivalent to searching for a valid (always-even-parity) surface, and both the graph-representation TQC^g and the Boolean representation TQC^b can be used in the process. The correlation surface construction problem is formulated as the question: "Given a subset IO^c of the input/output nodes IO, and a stabiliser transformation between the circuit's inputs and outputs, is there a valid a correlation surface accordingly?".

The question is answered by having a second look at the definitions of a correlation surface (connects inputs to ouputs), of their validity (even-parity) and the relation between stabiliser transformations and the graph-representation (when starting with tubes from an input-node, two paths should lead to the output-node, Proposition 4). A valid correlation

surface can be visualised, using the graph-representation of a TQC circuit, as a subgraph of one of the graphs from TQC^g. The subgraph has the shape of a tree: an arbitrary tube surface is a tree, while sheets are abstracted as supplemental nodes introducing tube-branches. The set of all the leaves equals the set of the nodes IO^c. For example, in the right panel of Figure 4.18, the constructed correlation surface has $IO^c = \{1, 2, 3\}$ as leaves.

A correlation surface cannot be determined at the same time by tubes or sheets of opposite types (see Definition 10). The stucture of the underlying cluster restricts a valid surface to contain primal and dual tubes (or primal and dual sheets) and the right type of the surface is computed by the *spec*-function (see Section 4.3.2), and the corresponding G_t or E_t is chosen.

A primal correlation surface, implying the use of G_p or E_p, contains primal tubes determined by primal defects and primal sheets spanned by dual defects. At the same time, a dual surface (G_d, E_d) implies the search of a correlation surface consisting of dual tubes spanned by dual defects and dual sheets spanned between primal defects.

The goal of TQC circuit mapping is to extract from a geometry the necessary information for the (classical) control software of a quantum computer (see Figure 4.9). The information includes the measurement basis of the individual lattice qubits, and the set of lattice qubits that form correlation surfaces of logical qubits. The measurement outcomes of the correlation surface-qubits are necessary for calculating the corrections to the encoded data (Pauli tracking at the logical layer), as information is propagated through more complicated topological structures. The surfaces are not specified within the TQC circuit, and the mapping procedure must derive them from the geometric structure, and map them to actual sets of lattice qubits.

Graphs enable an algorithmic formalisation of the solution, and the mapping approach will be detailed from the perspective of the graph-representation. The presented mapping algorithms will operate on TQC^g, but an arbitrary component m is further considered as input, as each graph G_t is the union of its components $CMP(G_t)$.

4.5.1 Mapping of TQC circuits

The mapping starts by taking each component $m \in CMP(G_t)$ and constructing a set $\mathcal{M}_m$.

$$\mathcal{M}_m = (t, D^t_m, I^t_m, O^t_m, J^t_m, X^t_m, Z^t_m)$$

Within $\mathcal{M}_m$ (the mapped m), t stands for the type of the component (primal or dual), and D^t_m includes all defect-internal physical qubits (i.e., those to be measured in the Z basis). I^t_m and O^t_m are physical qubits that define inputs and outputs, respectively. The sets X^t_m and Z^t_m include all physical qubits that are part of the X and Z correlation surface, respectively. Finally, J^t_m is the set of injection points.

The ultimate outcome of the mapping is the set $\mathcal{M}$, which contains tuples $\mathcal{M}_m$ for all the components m from TQC^g. The physical qubits from the $\mathcal{M}_m$ sets are from the $TQCC$ set

(the set of all lattice qubits, see Section 4.1.2). Therefore, qubits are addressed using their 3D-coordinates

The $\mathcal{M}_m$ tuple refers to the correlation surfaces as X and Z, although, until now, the surfaces were referred to as tubes and sheets. However, as mentioned in Section 4.1.4, the Z correlation surface is a sheet for a primal logical qubit and a tube for a dual logical qubit. The X correlation surface is a tube for a primal logical qubit and a sheet for a dual logical qubit. Following the discussion from Section 4.1.2, where the tubes and the sheets were defined using face and side-qubits, the mapping procedure identifies the physical qubits for each correlation surface ($X^{p,d} \subset F^p$ and $Z^{p,d} \subset S^p$). For convenience, two functions are introduced. The function $sheet : \mathcal{M} \rightarrow TQCC$ returns Z^p_m for primal logical qubits and X^d_m for dual logical qubits. The function $tube : \mathcal{M} \rightarrow TQCC$ returns the corresponding tube similarly to *sheet*.

Cycles were recognised as the support of sheets and the construction of the sets from $\mathcal{M}_m$ proceeds by traversing the cycles from the set $CY(m)$. The cycles are used for the construction of the tubes, too, although this is not necessary and the graph-edges could have been used. This decision was taken in order to keep the construction mechanisms similarly. It is assumed that a cycle $c \in CY(m)$ is defined by the set of nodes V^c_t and the edges E^c_t. Furthermore, the cycles will be considered directed, in order to ease their processing, and the function $ngh^n(k), k \in V^c_t, n \in \mathbb{Z}$ indicates the n-th neighbour of vertex k in the direction of the traversal. The direction of the cycles determines the direction in which the geometrical segments are traversed, and the function $dir : E^c_t \rightarrow \{\pm w, \pm h, \pm t\}$ yields the lattice-direction of the segment represented by the given edge.

Two more functions will be used during the mapping. The function

$$type : V \rightarrow \{input, output, inject, junction\}$$

returns the type of a geometry point that was translated into a graph-node. The type is used after physical qubits associated with the segment have been calculated in order to decide which set from $\mathcal{M}_m$ they belong to. Based on the node type, the function *set* takes $\mathcal{M}_m$ as input and returns the corresponding coordinate set of physical qubits. For example, physical qubits for the injection-points *inject* will be added to the set J^t_m, while the qubits associated to edges containing input-nodes will be added to I^t_m.

4.5.2 Computing tubes

For computing all the qubit sets except the ones for the sheets, the component m is used as input to Algorithm 4. After selecting a random starting node from a cycle, the lattice-direction d associated with each edge of the graph is computed. For each cell cc of a segment, F_{cc} is its complete set of physical face-qubits (see Section 4.1.2), and two face-qubits along the segment are defect-internal and added to D_{cc} (Line 8). The remaining four unit cell qubits form a ring, and are part of the tube correlation surface. The qubits form the set T_{cc} that is then added to $tube(q_\sigma)$. Coordinates of input/outputs/injection points are

Algorithm 4 Finding Defects and Tube Surfaces

Require: $\mathcal{M}_m = (l, D, I, O, J, X, Z)$
Require: the component $m \in CMP(G_t)$
1: for all $c \in CY(m)$ do
2: $start \leftarrow$ random $k \in V_t^v$
3: $ck \leftarrow start$
4: repeat
5: $d \leftarrow dir((ck, ngh(ck)))$
6: $b \leftarrow coord(ck)$; $e \leftarrow coord(ngh(ck))$;
7: for all $cc \in (b, e)$ along d do
8: $D_{cc} \leftarrow \{p | p = cc \pm 1 \text{ along } d, p \in F_{cc}\}$
9: $T_{cc} \leftarrow F_{cc} \setminus D_{cc}$
10: if $type(ck) = type(ngh(ck)) = junction$ then
11: $tube(\mathcal{M}_m) \leftarrow tube(\mathcal{M}_m) \cup T_{cc}$
12: end if
13: $set(q_\sigma, defect) \leftarrow set(q_\sigma, defect) \cup D_{cc}$
14: end for
15: if $type(ck) \neq junction$ then
16: $set(\mathcal{M}_m, type(ck)) \leftarrow set(\mathcal{M}_m, type(ck)) \cup \{b\}$
17: end if
18: $ck \leftarrow ngh(ck)$
19: until $start = ck$
20: end for
21: return q_σ

added afterwards to the corresponding sets. The coordinates of the defect qubits are added at Line 13.

4.5.3 Computing sheets

The sheet surfaces for a TQC graph-component are found by a procedure that iteratively reduces the cycles of the component until it consists of just two vertices. Although a component will generally consist of more than one cycle, the following discussion assumes that $\mathcal{M}_m$ will contain a single sheet, generated by a single cycle. The cycles (supporting sheets) of a component form a group under surface addition (see Section 4.3.1), and it is correct to assume that only one of the sheets is required for a certain construction. The procedure is repeated for each cycle existing in the component.

Sheet-finding will start from the observation that a component-cycle will have an even number of nodes corresponding to *corners* in the geometry, because only $90°$ angles between the segments are possible.

Algorithm 5 Finding Complete Sheet Surfaces

Require: $SUBS$
Require: $\mathcal{M}_m = (t, D, I, O, X, Z); m \in CMPG_t$
1: $sheet(\mathcal{M}_m) \leftarrow \emptyset$
2: for all $ss \in SUBS$ do
3: $\quad SSQ \leftarrow \{q | coord(q) \in bbox(ss) \cap S^t\}$
4: $\quad sheet(\mathcal{M}_m) \leftarrow sheet(\mathcal{M}_m) \Delta SSQ$
5: end for
6: return σ

Mapping a component-cycle to a sheet is a constructive approach, where the complete sheet is found piecewiese: one sub-sheet in each iteration. A sub-sheet contains physical qubits bounded by a rectangle in either the *wh*, *wt* or *ht* plane of the lattice (expressed in 3D coordinates). Two points are necessary to specify a sub-sheet: $(ss^1, ss^2); ss^i \in CEL$, where ss^i define the diagonal coordinates of the rectangle and have a single equal coordinate index (e.g $((0,0,0),(2,2,0))$).

Sub-sheets

computed by Algorithm 6 are disjoint sets of lattice qubits, and the union of all the found sub-sheets is the complete sheet. The algorithm takes a cycle of a component as an input, and transforms the cycle by eliminating or moving nodes, while sub-sheets are calculated. The transformations do not modify the geometry of the computation, and are simply used during the calculation. The cycle-operations *reduce*, *reshape*, *insert* are of a hybrid nature, operating on graphs but requiring the geometric description for their functionality. Node-insertion using *insert* is not directly applied by the algorithm, and the number of nodes is modified only during the *reduce* operation, or after the *reshape* operations was applied. The $insert(a, x, b)$ function will insert the node x between a and b.

Algorithm 6 is used to compute the $SUBS$ set of sub-sheets for each cycle, and will remove vertices by continuously traversing it (Lines 2–26), until only 2 vertices are left (Line 2). Algorithm 5 takes each $SUBS$ set and constructs the $sheet(\mathcal{M}_m) \subset TQCC$. For the sub-sheet (ss^1, ss^2) the set SSQ contains the coordinates of the physical qubits returned by the function $bbox_{ss} = \times_{i \in \{w,h,t\}} [min(ss_i^1), max(ss_i^2)] \cap TQCC$. For geometries having a defect cross-section larger than a cluster-cell the set union operations on Lines 11 and 23 are to be interpreted as $\mathcal{A}\Delta\{a\} = (\mathcal{A} \setminus \{a\}) \cup ((\{a\} \setminus \mathcal{A}) \cap \{a\})$, meaning that if element a existed in the set $\mathcal{A}$ it would be removed, otherwise it would be included.

Similarly to the previous algorithm, some utility functions are necessary. The function $mirr : K \rightarrow K; mirr(a) = coord^{-1}(coord(ngh^{-1}(a)) + coord(ngh(a)) - coord(a))$ returns the node a mirrored at the line through its predecessor and successor in the cycle, while function $clst : K^3 \rightarrow K; clst(a, b, c)$ returns either b or c depending which is closer to

Algorithm 6 Finding Sub-Sheet Surfaces

Require: $c = (V_t^c, E_t^c) \in CY(m), m \in CMP(G_t)$

1: $SUBS \leftarrow \emptyset$
2: while $|V_t^c| \geq 2$ do
3: start $\leftarrow$ random $k \in V_t^c$
4: $ck \leftarrow start$
5: $compact \leftarrow false$
6: $a \leftarrow ngh^1(ck)$; $b \leftarrow ngh^2(ck))$
7: repeat
8: if $dir((ck, a)) = -dir((b, ngh(b)))$ then
9: $compact \leftarrow true$
10: $reduce(a, b)$
11: $SUBS \leftarrow SUBS \cup (ngh(ck), a) \cup (ngh(ck), b)$
12: else
13: if $dir((ck, a)) = \pm dir((a, b))$ then
14: $remove(a)$
15: $compact \leftarrow true$
16: else
17: $ck \leftarrow a$
18: end if
19: end if
20: until $start = ck$
21: if $compact = false$ then
22: $reshape(start, ngh(start), ngh^2(start))$
23: $SUBS \leftarrow SUBS \cup (start, ngh^2(start))$
24: $start \leftarrow ngh(start)$
25: end if
26: end while
27: return $SUBS$

vertex a.

After implementing the sheet-finding algorithm, it is possible to illustrate its output. Figure 4.24 shows a complete sheet composed of sub-sheets, and Figure 4.19 depicts the progressive calculation of sub-sheets.

The reduce operation

Algorithm 6 modifies the graph by applying the *reduce* operation. The path is reduced if, for 3 consecutive edges, the first and the second edge represent opposite associated directions into the lattice (Line 10). For example, this is the case for the edges (B, C), (C, D), (D, E),

where (B, C) and (D, E) have opposite directions in Figure 4.20).

The $reduce(a, b); a, b \in V_t^c$ operation is defined as the sequential application of:

$$\begin{aligned} Rm &= \{a, b\}; \\ n_a = ngh^{-1}(a); n_b &= ngh^1(b); \mathcal{N}g = \{n_a, n_b\} \\ V^{red} &= \{clst(a, mirr(b), n_a), clst(b, mirr(a), n_b)\} \\ V^{red}_{ins} &= V^{red} \setminus \mathcal{N}g; \\ V^{red}_{del} &= \mathcal{N}g \cap V^{red}; \\ remove(v); &\text{ for all } v \in Rm \\ insert(n_a, v_i, n_b); &\text{ for all } v_i \in V^{red}_{ins}; \\ remove(v_d); &\text{ for all } v_d \in V^{red}_{del}; \end{aligned}$$

The operation is illustrated by applying it to the nodes C and D in Figure 4.20. The sets R, $\mathcal{N}g$, V^{red} are constructed.

$$\begin{aligned} Rm &= \{C, D\}, \mathcal{N}g = \{B, E\} \\ V^{red} &= \{clst(C, mirr(D), B), clst(D, mirr(C), E)\} \end{aligned}$$

Because $mirr(C) = E$ and $mirr(D) = B$, the set $V^{red} = \{B, E\}$ is equal to $\mathcal{N}g$ and $V^{red}_{del} = V^{red}_{ins} = \emptyset$. After the nodes from Rm are removed, no further nodes are inserted or removed, for the corresponding sets are empty. However, for the example in Figure 4.20, this is not the case as the number of deleted and inserted nodes is one ($|V^{red}_{del}| = |V^{red}_{ins}| = 1$), thus effectively removing one node (B) and inserting another one (C').

The reshape operation

Reducing the graph may require to use an equivalent geometrical description. Thus, nodes are not removed or deleted, but moved ($|V_t^c|$ remains constant). The $reshape(a, b, c); a, b, c \in V_t^c$ operation is the sequential application of:

$$remove(b); insert(a, mirr(b), c);$$

In the context of the Algorithm 6, the function is called if, during a complete cycle traversal, the *reduce* operations cannot be applied. For the example of Figure 4.20, where the operation $reshape(B, C, D)$ is called, the resulting cycle will be obtained by removing C and inserting $C' = mirr(C)$. Applying *reshape* for a second time at the same position would undo the initial application. As a result, $reduce(B, C', D)$ is the inverse of $reduce(B, C, D)$ and the *start* pivot (Line 3), used for checking if a traversal completed (Line 20), needs to be updated (Line 24).

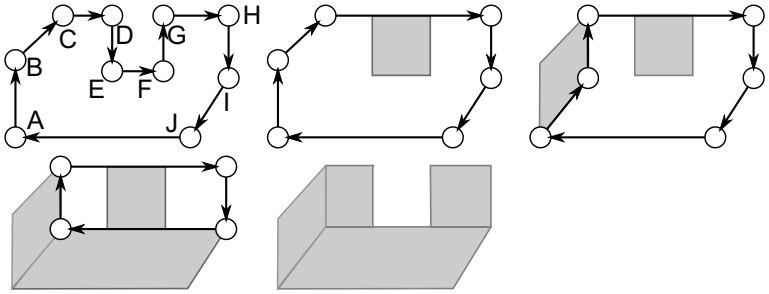


Figure 4.19: The example illustrates the application of Algorithm 6 for the graph $(\{A, \ldots, J\}, \{(A, B), (B, C), \ldots (J, A)\})$. For example, starting from the vertex $start = A$, the first possible operation is $reduce(E, F)$ and a first sub-sheet is found $SUBS = \{(E, G)\}$. In addition *co-linear* nodes C, D, G, H will result in $remove(D)$ and $remove(G)$ being applied. The cycle is traversed until the *start* is reached again, and the first traversal completes. After the second traversal, neither *reduce* nor *remove* were applied. The $reshape(A, B, C)$ is applied, and a second sub-sheet is found $SUBS = \{(A, C), (E, G)\}$. Finally, the last two sub-sheets are inferred leading to $SUBS = \{(A, I), (C, I), (A, C), (E, G)\}$. The complete sheet is found by combining all the sub-sheets according to Algorithm 5.

Number of nodes during processing

In a geometry where only $90°$ angles are allowed, any closed geometric contour will contain an even number of segments, and the number of nodes of a cycle-graph will be initially even. Moreover, the number of nodes remains even during the execution of the algorithm. Operation *reduce* eliminates exactly two vertices from the graph (set Rm), while the sets of further added and deleted vertices (V_{ins}^{red} and V_{del}^{red}, respectively) are always of the same size. Operation *reshape* does not add or delete vertices. However, three consecutive vertices may represent a straight line after a *reshape* operation, in which case they are replaced by two vertices (Line 14). A further vertex elimination will follow, keeping the overall number of vertices even.

4.5.4 Complexity

The complexity of the mapping algorithms is analysed in the following. The tube-mapping algorithm requires a single cycle traversal, while the computation of the coordinates is straightforward (see the illustration presented in Figure 4.4). For an arbitrary graph component $m \in CMP(G_t)$, the runtime complexity is linear in the number of nodes that represent the geometry.

For the runtime complexity analysis of the sheet mapping procedure, a *worst-case geometry* can be defined. Such a geometry, when mapped to the lattice, will have to necesitate a maximum number of *reshape* applications in order to be able to compute the corresponding sub-sheets. When searching for a worst-case geometry it should be considered that Algorithm 6 randomly selects a node from the cycle (Line 1), and that the node is used as a *pivot* for the *reshape*. These assumptions imply that, even after a sequence of *reshape* operations,

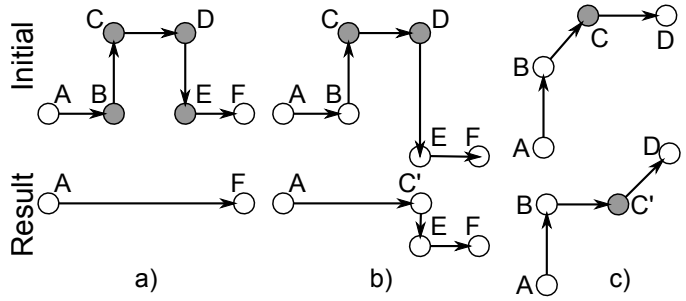


Figure 4.20: Graph-operations: a) Applying $reduce(C, D)$ results in the number of nodes being decreased by 2, because C, D are removed. After the operation, because A, B, E, F correspond to co-linear lattice coordinates, $remove(B)$; $remove(E)$ can be further applied, and the number of nodes again decreased by 2. b) The $reduce(C, D)$ is applied. The mirrored vertices $V^r = \{C', D'\}$ are computed, with $coord(D') = coord(B)$, thus $V^r = \{C', B\}$. The vertices from $V_i^r = \{C', B\} \setminus \{B, E\} = \{C'\}$ will be inserted, and the vertices from $V_d^r = \{B\}$ will be deleted. c) The effect of the $reshape(B, C, D)$ operation is that vertex C is replaced by vertex C'.

there is no possibility to find a sub-sheet and to reduce the number of vertices. Hence, a further *reshape* is necessary.

For a cycle with $|V_t^c|$ nodes, the maximum number of cycle traversals is $O(|V_t^c|^2)$ for the case that after each traversal a *reshape* operation is required. The number of consecutive *reshape* operations is bounded by $|V_t^c| - 3$. The worst-case situation arises when $|V_t^c| - 3$ vertices are arranged in a pattern similar to the one in Figure 4.21a, where the red node indicates the pivot and the cycle is traversed clockwise. All other nodes are not represented. After a first traversal of the cycle, the *reduce* operation was not applied, and a *reshape* followed by two corresponding *remove* operations will transform the cycle similarly to the one from Figure 4.21b. Until the steps-like geometry is not fully reshaped, the *reduce* operation cannot be applied. However, after each *reshape*, co-linear vertices are *removed*, thus reducing $|V_t^c|$. For each *reshape* the complete cycle has to be traversed, and, overall, the complexity of the sheet-finding procedure is bounded by $O(|V_t^c|^2)$.

While the mapping algorithm scales polynomially with the number of the cycles, no claims can be made regarding the length of the cycles in a TQC circuit, as the size of the cycle is essentially related to the number of 90° angles. It is still unknown how large the set of cycles is for a complete and practical TQC quantum computation (e.g. Shor's algorithm for a large number of bits).

4.5.5 Correctness

The correctness of the mapping algorithms implies verifying their termination, and the fact that the correct physical qubit coordinates are computed.

Algorithm 4 terminates after a single traversal of the cycle. The correctness of the coordinates is shown by comparing the output of the algorithm with the definition of defect qubit coordinates (per unit-cell, two face-qubits) and tube qubit coordinates (per unit-cell, four

Figure 4.21: The worst-case situation when applying the reshape operation.

face-qubits). The direction d (Line 5) is associated with the green line in Figure 4.4, where b and e are the *CEL* coordinates of the edge vertices (for example, considering that the two cells from the lower defect are b and e). This implies that by selecting the two neighbouring coordinates (Line 8), the coordinates of the green marked qubits (the defect set D) are computed. The remaining 4 qubits (Line 9) that do not belong to D are the light blue marked qubits, which are associated with the *tube* correlation surface that surrounds the defect region.

The termination of the sheet-finding algorithm (Algorithm 5) is shown by starting from the fact that the geometric description is mapped into a 3D representation, where only 6 segment directions are possible (see the discussion of the *reshape* and *reduce* operations). A geometrically described defect configuration of a logical qubit (in the absence of any possibility to apply *reduce* or *remove*) will have an even number of edges in its associated cycle. Considering the worst-case geometries (see Figure 4.21a), it follows that after each traversal, either *reshape* or *reduce* (followed by *remove*) can be applied. The number of maximum consecutive *reshapes* is bounded by $|V_t^c|$, but the number of nodes is continuously reduced, and the termination of the algorithm is guaranteed.

The correctness of the coordinates computed from the sub-sheets is shown by comparing Lines 3 and 4 of Algorithm 5 with Figure 4.4. For a cycle c of type t, the set of all possible *side-qubits* (see Figure 4.1c for a single cell) is S^t. For the two defects from Figure 4.4, the instruction on Line 3 will return the coordinates of both the orange and the blue marked qubits from Figure 4.4, which intersected with S^t will return only the blue qubits. These blue qubits are the ones necessary for the computation of a sub-sheet. If two sub-sheets computed by Algorithm 6 overlap, then the intersection set is not part of the set of qubits defining the complete sheet (Line 4).

A valid correlation surface is an even-parity surface (see Proposition 3), and sheets are defined using side-qubits from a unit-cell (the set S^l in Section 4.1.2). In the following it will be shown that the *reduce* and *reshape* operations are not affecting the validity of a correlation surface that is constructed. Both operations are based on the geometric description and are used to compute a complete correlation surface by deforming the sheet boundaries. The *reshape* operation changes the directions of the affected geometric segments, while the *reduce* operation removes segments.

A formal proof of the operation correctness is based entirely on surface additions. The

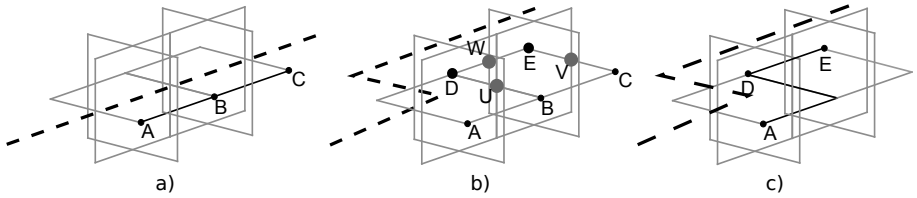


Figure 4.22: A cluster fragment to illustrate the correctness of the sub-sheet finding routine.

stabilisers are Hermitian matrices, and multiplying a stabiliser by itself results in the identity. The deformation of a correlation surface consists, in general, of two consequtive stabiliser multiplications $S_1 = ((SHEET)R)\mathcal{M}$, where R is the cluster stabiliser to be removed, and $\mathcal{M}$ is the cluster stabiliser to be added: $Q(SHEET) \cap Q(\mathcal{M}) = \emptyset$, and $Q(R) \subset Q(SHEET)$.

The operations used in Algorithm 6 are particular types of deformations. For a given cluster stabiliser $SHEET$ where $Q(SHEET) \subset TQCC$, the *reduce* operation is the stabiliser multiplication $S_2 = (SHEET)R$. As a result, $Q(S_2) \cap Q(R) = \emptyset$ and $Q(S_2) \subset Q(SHEET)$. The *reshape* operation, considering a sub-sheet T, is either the multiplication $S_3 = (SHEET)R$ when $R = T, Q(T) \subset Q(SHEET)$, or $S_4 = (SHEET)\mathcal{M}$ when $R = \mathcal{M}, Q(T) \subset Q(SHEET)$.

The following example illustrates the previous discussion. The modification of a sheet boundary is illustrated in Figure 4.22. After changing the direction of the segment, the boundary $A, B, C \in SHEET$ is transformed into $A, D, E \in SHEET'$; $SHEET' = SHEET \cup \{D, E\}$, where the previous $\mathcal{M}$ stabiliser is defined over the qubits D, E. In terms of sub-sheets, the sub-sheet (B, E) is added after the change of defect direction. The dashed line in the figure indicates the direction of one defect involved in generating the sheet surface $SHEET$.

Without affecting the generality of the example, it can be assumed that the nodes A, B, C mark the only qubits of $SHEET$. The lattice neighbourhood of these qubits (nodes) will contain only 3 qubits (one was removed for the defect bounding the sheet): the qubit A will be entangled to the same qubit to which B is entangled, and C will be entangled to another qubit to which B is entangled. The neighbouring lattice qubits of the nodes are assumed to exist on the edges (A, B) and (B, C), and the parity of the sheet is $par(SHEET) = \emptyset$.

After deforming the defect structure, on the new boundary of the defect the qubits marked by the nodes $\{D, E\}$ are interpreted as a sub-sheet $SSHEET$. Not being rectangular it does not exactly correspond to the definition of one. However, the parity $par(SSHEET) = \{U, V\}$ is computed, because the face-qubit W is entangled to both E, D, and its Z-stabiliser is cancelled during the parity set computation. After moving the defect boundary, the qubits $B, C \in SHEET$ are entangled to U respectively V, and the sheet parity is $par(SHEET) = \{U, V\}$. As a result, the fact that the set $par(SHEET + SSHEET) = \emptyset$ is empty indicates that the possibility of an even-parity surface construction. Hence, the resulting sheet $SHEET'$ is correct.

According to the definition of sub-sheets even-parity is maintained during their construc-

tion, and the direct conclusion is that changing the direction of defects does not affect the correctness of the infered sheets.

4.5.6 Graph-based construction

Correlation surfaces can be constructed starting from TQC^g if the graph-edges are considered abstractions of possible tube-surfaces and the graph-braiding-nodes as abstraction of possible sheet-surfaces. For the given IO^c, the sought correlation surface will be a tree having the leaves as the set of nodes $V^{src} \subset V_t$ of $G_t, t \in \{p, d\}$.

Mapping the IO^c set to V^{src} is directly accomplished for the IO^c points where *spec* indicates $tubes_t$: the associated node from IO_t is added to V^{src}. For the points at which *spec* returns $sheet_t$, appropiate braiding-nodes from B_t have to be selected, as each brading-node is related to a sheet (cycle) from the opposite space (graph). Let $IO^s \subset IO^c$ be the set of inputs that are required to exist on sheets.

When the opposite graph $G_{t'}$ was constructed, for determining braiding-nodes, at each graph-component only the generator-sheets were employed, in order to reduce the number of intersections with defects. The selection of the correct braiding-nodes to be included into V^{src} is detailed starting from the properties of the cycles. Initially, the graphs are considered without braiding-nodes.

In an arbitrary graph, there will be cycles containing nodes from IO and cycles that run entirely over junction-nodes. For nodes from IO_t, the edges of the cycles represent tube correlations between pairs of nodes, and for nodes from IO^s, the spanned sheet could be part of the final correlation surface. The selection of the appropiate braiding-nodes from B_t to include into V^{src} is a combinatorial problem: a reduced set of generator-sheets represents all the possible sheet-constructions supported by a graph-component.

The IO^s set of inputs has to be firstly partioned by finding which graph-components contain the associated input-nodes, as it is already known which nodes exist in each component. For a component $m \in CMP(G_t)$ containing the set of input-nodes $CIO(m) \subset IO_t$, each cycle $c \in CY(m)$ contains the inputs from the set $CIO(c)$, such that $CIO(m) = \bigcup CIO(c)$. A cycle not running over input-nodes will have $CIO(c) = \emptyset$.

For IO^s the set of components $\{m_1, m_2, \ldots\}$ is computed such that $IO^s = \bigcup CIO(m_i)$, and the next step is to find out if the m_i component supports a cycle cs such that $CIO(cs) = CIO(m_i)$. For each component m_i, the solution consists of the cycle combination resulting in the inputs $CIO(cs)$. Combining cycles is a constructive approach, and the same approach was used to compute the minimum number of cycles in a component (Section 4.2). After combining two cycles c_1, c_2, the result c_3 is defined over $CIO(c_3) = CIO(c_1) \Delta CIO(c_2)$ (inputs existing on common edges will cancel out, as the common edges disappear). The construction of a correlation surface is unsuccessful, if for any of the components m_i, the search for a specified cycle returned no result. Otherwise, the set of generator-cycles is returned and is further used.

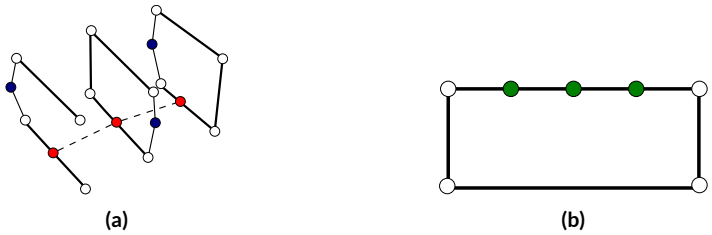


Figure 4.23: The primal and the dual graph corresponding to the primal-primal CNOT circuit. The edges abstracting the primal defects are braided with the same dual sheet, requiring the red braiding-nodes to be connected by dashed edges corresponding to relation-edges. In the dual graph, green braiding-nodes correspond to different dual sheets and are not related.

Having computed all the cycles supporting the sought-after sheets, the corresponding braiding-nodes can finally be appended to V^{src}, and the search of a correlation surface can proceed. A straightforward method of searching for the surface subgraph is to use a depth-first-search algorithm, but the depth-search will have to differentiate between junction-nodes and braiding-nodes. Whereas at junction-nodes all the incident edges have to be traversed, at *related braiding-nodes* either both "normal" edges and none of the relation-edges are traversed, or one normal edge and all the relation-edges. This is necessary for complying with the validity criteria from Section 4.2.

The construction algorithm can either output a tree according to V^{src}, which implies that a valid correlation surface was constructed, or signal the construction failure by returning an empty output. The resulted correlation surface will contain tubes for each normal edge in the tree, junctions at each non-leaf-node, and sheets indicated by the braiding-nodes that were traversed along the relation-edges.

4.5.7 Boolean-expression-based construction

The Boolean expressions of the circuits were formed in CNF, and usually a valid assignment of such formulas can be computed by SAT solvers. Each B-clause (see Section 4.4) contains only XORs of three variables, where one of the variables is negated and the other two are not. Thus, the $B(S, a, b)$ can be reformulated as $\neg S \oplus a \oplus b \oplus 1 = 0$, and a valid variable assignment of the Boolean expressions will have to result into *false*, and not *true*. Each B-clause represents a linear equation, and the $\wedge$ conjunctions are the means by which the equations are bundled into a linear equation system over GF(2). Hence, computing a valid variable assignment is equivalent to solving a linear equations system by Gaussian elimination or other polynomial-time algorithms.

As a conclusion, the construction problem is *XORSAT* and belongs to the complexity class *P*. Solving this problem is not difficult, and from a practical perspective, a linear systems solver can be employed to solve the system. The solution space of the system is an affine subspace of the Boolean field of size 2^n (n variables), making the problem a member

of the complexity class $\#P$ (the class of counting problems associated to decision problems[MM09]), implying that the number of solutions can be computed before trying to solve the equations.

The graph-based approach was equivalent to a SAT based solution and was not optimal. The construction algorithm performed the subgraph-search by computing all the combinations and selecting the one offering the required support for a valid correlation surface.

From a Boolean perspective, searching for a subgraph is equivalent to asking which edges from a graph should be *enabled* (*true*), and which should be *disabled* (*false*). Graph-nodes having exclusively disabled incident edges will be discarded. The relation-edges are not required for this construction method, as the conjunction of braiding-clauses is an equivalent approach: related braiding-clauses refer to the same sheet-literals.

The construction of a correlation surface according to IO^c and the specified stabiliser transformation requires the same initial steps as the graph-based method. The major difference is that a solver will be used to replace the subgraph-search algorithm.

After determining if the surface should be in the primal or in the dual space, the matching expression E_t from TQC^b is selected. The truth-value of the expressions is evaluated by searching for variable assignments for which $E_t = true$, and, if a solution exists, the solver will indicate the valid assignment.

Although two different Boolean variables generally represent two different correlations supported by the geometrical description, there are situations when the surfaces (irrespective of their type, tube or sheet) connect the same cluster-locations (in Section 4.2 referred to as sets of qubits stabilised by Z).

Definition 14. Two non-input variables x, y are *correlation equivalent* if the associated surfaces are incomplete (not valid) and have the same parity, $par(x) = par(y) \neq \emptyset$ (see Section 4.2).

The correlation equivalence (different from logical equivalence) was restricted to non-input variables, as the parity at the TQC circuit inputs/outputs depends on the used logical qubit initialisation/measurement. Of course, any input-variable is equivalent to itself, but the same is true for the non-input variables. Two surfaces S and T having equal parity sets ($par(S) = par(T)$) are by Definition 14 correlation equivalent ($S \equiv T$) and any of the two can be considered during the process of constructing a valid surface.

The expression E_t represents all the supported surface constructions, where each solution is a surface having always-even-parity. A specific solution can be computed by iterating over the Boolean variable assignments of E_t, and selecting the one containging only the variables from V^{src} being *true*.

Another possibility of constructing a specific correlation surface is to iterate over the solutions of a constrained expression E_t. This is achieved by appending unit clauses of the form (l) to E_t, and each clause corresponds to a variable associated to the nodes from V^{src}. For example, for an arbitrary circuit containing the input-variables S and a, the constraint expression will be $E'_t = E_t \wedge (S) \wedge (a)$. The supplemental clauses reduce the search space for

the possible assignments satisfying the existing braiding-clauses from E_t. The unit clauses are *propagated* to all the other clauses from E_t [HJB10]. The clauses that contain the literal l are removed from E_t ($x \vee \mathit{true} = \mathit{true}$), and from the remaining clauses the literal $\neg l$ is removed ($x \vee \mathit{false} = x$). By solving the Boolean linear equations from E'_t only a reduced set of surfaces will be inferred, e.g. where the sheet S and the tube a exist.

4.6 Summary

Topological quantum computation is one of the most probable technologies to be used in future quantum computers, and this chapter developed, presented and analysed some design methods of TQC circuits. This chapter shortly introduced the topological quantum computing concepts required for a more extended analysis of the correlation surfaces existing in the circuits. Being a measurement-based computing paradigm, the surfaces represent in TQC the quantum correlations formed by using teleportation as a means to process information. The visual aspect of TQC circuits, which was captured by the geometric description, generated a new approach of representing the circuits as graphs. The graphs were extended to the Boolean representation of the circuits.

It can be concluded that the geometric, graph and Boolean representation of TQC circuits fulfil the same task, but each of these has both advantages and disadvantages. Excepting the visualisation purposes, the graph representation was instrumental in allowing the computation of correlation surfaces as qubit subsets from the underlying cluster state. The Boolean representation of the circuits allowed the formalisation of the correlation surface construction problem, which had an unknown difficulty. Its mapping to XORSAT computational problems enables the design and the construction of efficient algorithms that fit in the TQC design stack.

The circuit representations used in this chapter were motivated by the correlation surface construction problem. The same representations will be translated in the next chapter as the mechanisms underlying automatic synthesis and validation of TQC circuits.

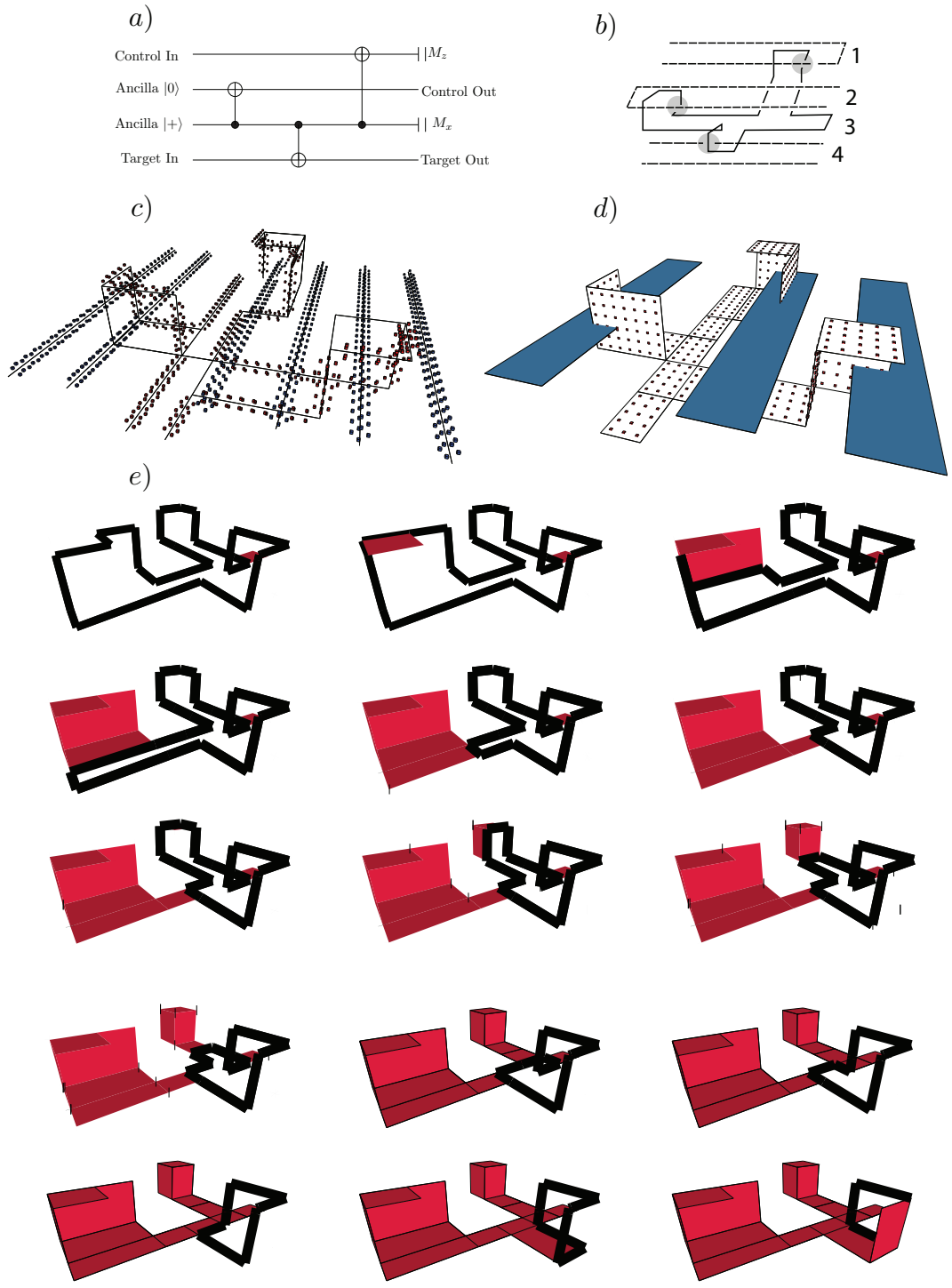


Figure 4.24: Example of mapping the primal-primal CNOT circuit: a) The network of gates; b) The geometrical description where the numbered defect structures correspond to the qubits from the gate network; c) All the possible tubes; d) All the possible sheets; e) The iterative construction the sheet corresponding to qubit 3 (the ancilla initialised in $|+\rangle$ and measured in X) using Algorithm 6.

5 Topological Circuit Equivalence

The braiding relation is central in the construction of valid correlation surfaces, and leads to a definition of *canonical* Boolean representation of TQC circuits. Similarly to the construction of surfaces, the canonical representation consists of two expressions: one for the primal and another one for the dual circuit elements. The approaches presented in this chapter are motivated by the Boolean expressions representing the geometrical descriptions, and circuit equivalence checking methods incorporate operations on the represented correlation surfaces. A formalised representation of surface operations is useful for illustrating circuit identities and, in an extended form, for algorithms performing correct circuit compaction.

This chapter will focus on TQC circuit identities from the perspective of the specific canonical representation. The canonical TQC circuit representation using the *B*-notation is defined as the expressions containing only conjuctions of braiding-relations called *B*-clauses (see Equation 4.7). For brevity, the $\wedge$ operand will be ommited from the expressions: thus, ab should be interpreted as $a \wedge b$.

The direct mapping between the braiding relation and the geometric representation allows one to easily draw a TQC circuit starting from a canonical representation and, vice versa, to construct an expression using the computation's geometry. Transformations of the representation will keep this advantage in place. In the following, the *B*-notation and its properties are introduced starting from the braiding-relation and the properties of the underlying Boolean expressions. The developed *B*-notation will be also used in this chapter for introducing TQC circuit synthesis. In addition, the chapter will use insights gained after inspecting the notation to formulate TQC circuit validation methods.

The initial form of the braiding-relation stemmed from four possible constructions between the tubes and the sheet at the braiding-point (see Equation 4.4), which was rewritten as an equivalence relation between the existence of the sheet S and the exclusive existence of the tubes a and b: $B(S, a, b) = S \leftrightarrow (a \oplus b)$. In other words, if both tubes are enabled or disabled at the same time, then S is disabled $B(S, a, b) = \neg S \oplus a \oplus b$.

The Boolean expression of a TQC circuit, due to its CNF form, is similar to a *chain* of Boolean equivalences between variables: the solution of a braiding-relation will imply the solution of the next braiding-relation, and so on. By manipulating the form of a canonical TQC circuit representation, the underlying represented geometry is (indirectly) manipulated: the same quantum computation can be described by different but equivalent geometries. The properties of the *B*-notation used for transforming the Boolean expressions of a circuit will lead to TQC circuit automated task solutions like verification and compactification.

The direct mapping between geometry and Boolean formula has to be easily traced and it is useful to present properties that are intuitive from the geometric perspective. However, the following properties hold for arbitrary Boolean variables, meaning that, although the notation refers to sheet- and tube-variables, the variable-types are not enforced for the properties to hold.

The construction of a correlation surface was formulated in the previous chapter by a system of linear equations, where each equation was a *B*-clause. The simplest algorithm for solving such systems is the Gaussian elimination that relies on the addition of equations. The following presented properties can be understood as *local steps* of the Gaussian elimination, where *locality* refers to an associated place from the geometry.

First of all, as the logical *XOR* operation ($\oplus$) is commutative, it is possible to permute the parameters of the *B*-notation. Moreover, for any two Boolean variables a, b, $a \leftrightarrow b = \neg a \oplus b = a \oplus \neg b$, and as a result:

$$B(S, a, b) = B(a, S, b) = B(b, a, S) = \dots \tag{5.1}$$

In general, for any three equivalent Boolean variables the syllogism 5.3 holds:

$$(a \leftrightarrow b)(b \leftrightarrow c) \quad = \quad (a \leftrightarrow c)(b \leftrightarrow c) \tag{5.2}$$

$$(a \leftrightarrow b)(b \leftrightarrow c) \quad \rightarrow \quad (a \leftrightarrow c) \tag{5.3}$$

The transitivity of the Boolean equivalence will be used to compactify TQC circuits or validate the correctness of surface construction, and replacing a variable, as in Equation 5.2, is the direct result of applying the transitivity property of the logical equivalence. Removing a variable, like in Equation 5.3, is the logical implication of using the *transitive role* of variable b. It is possible to remove variables that occur in two brading-relations to obtain a shorter and satisfiability-equivalent formula, provided that such variables do not occur in

other clauses. In the following sections the logical implication $\rightarrow$ is used whenever a variable is removed due to its transitive role.

For general Boolean formulas, any variable can be removed, but for TQC circuits expressed using the *B*-notation attention has to be paid: input variables should not be removed, because these are required for the construction of correlation surfaces according to the circuit specification.

The *B*-notation is extended using Equations 5.2 and 5.3. Two braids that differ in a single variable (e.g. a and b) are logically equivalent ($a \leftrightarrow b$), and replaced (e.g. by y):

$$\begin{aligned} B(S,a,x)B(S,b,x) &= (\neg S \oplus a \oplus x)(\neg S \oplus b \oplus x) \\ &= (\neg a \oplus (S \oplus x))(\neg b \oplus (S \oplus x)) \\ &= (\neg a \oplus (S \oplus x))(\neg b \oplus (S \oplus x))(a \leftrightarrow b) \\ &= B(S,a,x)B(S,b,x)(a \leftrightarrow b) \\ &= B(S,ab,x)B(S,ab,x) = B(S,ab,x) \\ &\rightarrow B(S,y,x) \end{aligned} \tag{5.4}$$

The *XOR* of two of the *B*-notation parameters through an arbitrary variable results in:

$$\begin{aligned} B(S,a,b) &= (\neg S \oplus a \oplus b \oplus x \oplus x) \\ &= (\neg S \oplus (a \oplus x) \oplus (b \oplus x)) \\ &= B(S, a \oplus x, b \oplus x) \end{aligned} \tag{5.5}$$

Furthermore, the notation supports the situation when two braids differ in two variables. Eeach braiding-relation is a Boolean equivalence, and it follows that $x \leftrightarrow (T \oplus b)$, such that $x = x(T \oplus b)$,. The variable x *could* be removed without changing the validity of the formula (Equation 5.6). The final form (Equation 5.7) of the property is achieved after removing x. The removal is performed using the property from Equation 5.5 if the canonic expression contains $x \leftrightarrow (T \oplus b)$ as the only condition regarding the existence of x.

$$\begin{aligned} B(S,a,x)B(T,b,x) &= B(S,a,x)(x \leftrightarrow (T \oplus b)) \\ &= B(S,a,x(T \oplus b)) \qquad (5.6) \\ &\rightarrow B(S,a,T \oplus b) = B(S \oplus T, a, b) \qquad (5.7) \end{aligned}$$

Logical and correlation equivalence

Two Boolean expressions are *logically equivalent* if they have the same set of satisfying assignments over the common variables[HJB10]. For TQC circuits mapped to Boolean expressions, the logical equivalence is compared to the previously introduced correlation equivalence.

The variables do not need to be different and it is possible to consider formulas with the same variable present multiple times in a *B*-relation (for $B(a,b,c)$ $a \neq b \neq c$). However,

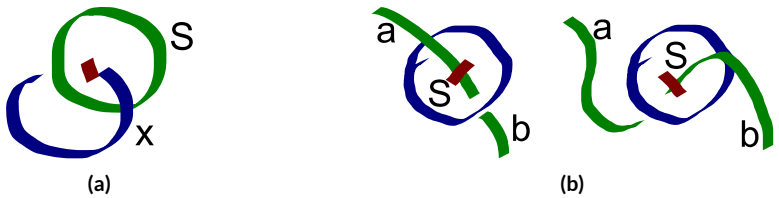


Figure 5.1: The diagrams of this kind are sketched without making any assumptions about the types of the defects. The blue and green defects are of opposite types, and the colors are not associated to a specific type (primal or dual). The red markings in the diagrams are used only to illustrate the existence of braids. a) Example of a invalid braid; b) The tubes at a braiding-point can be switched, and the corresponding B-clause remains unchanged.

this will detach the meaning of the B-relation from the braiding interpretation of sheets and tubes:

$$B(a,a,b) = (\neg a \oplus a \oplus b) \rightarrow \neg b \tag{5.8}$$

$$B(a,b,c \oplus a) = B(\emptyset,b,c) = b \leftrightarrow c \tag{5.9}$$

Equation 5.8 represents, for example, the situation from Figure 5.1a, where the braid on sheet S involving the tube x is an "invalid" braid, for the same tube variable would be determined by a single braiding-node. This implies that, independent of the truth-value of x, the sheet S cannot be constructed, as $B(S,x,x) = \neg S$, thus $S = \mathit{false}$. The geometric construction is valid in the context of TQC circuits; however, it cannot be considered a braid.

The logical equivalence between two variables (Equation 5.9) can be used, from a practical point of view, together with Equation 5.5 as a second proof of Equation 5.4.

$$\begin{aligned} B(a,b,c)B(a,b,d) &= B(a,b,d)B(b,c,b \oplus d); \text{(replace } a \text{ with } b \oplus d) \\ &= B(a,b,d)B(\emptyset,c,d) \overset{5.9}{=} B(a,b,d)(c \leftrightarrow d) = B(a,b,cd) \end{aligned}$$

The braiding-relation was defined to mimic the way valid correlation surfaces are locally constructed, and in that context the concept of correlation equivalence (the $\equiv$ operation) was mentioned. The equivalence of correlation surfaces will be transformed in this section into logical equivalence (the $\leftrightarrow$ Boolean operation). For this, let a, b, c be three variables of an arbitrary type. Assuming that $par(a) = par(b)par(c)$, it follows that $a \equiv bc$, but $b \equiv ac$, $c \equiv ac$ and $abc \equiv \emptyset$. These observations are based on the fact that each surface (represented by the corresponding variable) is constructed using the two other surfaces (e.g. instead of a the two surfaces b and c can be used), and the surface resulting after combining the three individual surfaces has always even parity (the set $par(abc) = \emptyset$). After combining the three individual surfaces, the result is expressed as $ab\neg c \oplus a\neg bc \oplus \neg abc \oplus \neg a\neg b\neg c = B(a,b,c)$. Moreover, if c is always set to false, then $a \equiv b \rightarrow a \leftrightarrow b$, which is expressed as $B(a,b,0)$.

For the above reasons, in general, the $x \equiv \bigcup_0^n V_i$ relation (the union $par(x) = par(\bigcup_0^n V_i)$ of multiple surfaces V_i) will be formulated using the B-notation by introducing the *temporary variables* G_i. These variables represent the linear combination of individual correlation

surfaces from the TQC circuit (e.g. sheets formed at junctions).

$$B(x, V_0, G_0)B(G_0, V_1, G_2)\ldots B(G_{i-2}, V_{n-1}, V_n) \text{ for } G_i = V_{i+1}V_{i+2} \tag{5.10}$$

This section was showed that correlation equivalence between variables of arbitrary types can be expressed using the *B*-notation. *B*-clauses will not always be interpreted as representing braids where two tubes interact with a sheet.

5.1.1 Junctions

The relations existing between sheets formed at junctions is supported by the *B*-notation, too. In a bridged geometry the number of cycles existing in the associated graph equals the number of sheets supported by the geometry. The geometry in Figure 4.12 is the support of three sheets (S, R, T), where $par(S) = par(R)par(T)$, and $S \equiv RT$. Hence, any sheet is expressed using the two other sheets. The correlation equivalence is formulated as logical equivalence by $B(S, R, T) = (\neg S \oplus R \oplus T) = S \leftrightarrow (R \oplus T)$. This property of three sheets at a junction is helpful for showing that in a geometric description the sheets belong to the same bridged geometry.

5.1.2 *B*-notation as a graph

The CNF form of the Boolean expression representing a geometric description implied that the *B*-notation is a canonical representation for TQC circuits. Constructing a valid correlation surface (solving the CNF formed expression) requires *searching* for a variable assignment using each *B*-clause as a condition that drives the search towards the final variable values. From the perspective of Boolean implications, the way *b* and *c* are related depends on the value of *a*, and these relations can be visualised using structures similar to implication graphs [Tar72]. In an implicaton graph, the literals are represented as nodes and directed edges indicate the Boolean implications between the variables. A variable assignment is found by tracking through the paths existing in the implication graphs.

A *B*-clause is a graph-cycle containig three nodes (a *3-cycle*), and a graph representing the complete canonical representation is constructed by appending further 3-cycles at the nodes that are shared between the clauses. A simple 3-cycle is presented in Figure 5.2a. The graph-nodes can have one of three possible values $\{always - true, always - false, true - false\}$. Searching for a variable assignment requires to associate values to the graph-nodes such that the conditions existing in the 3-cycles are fulfilled. The $true - false$ case is when a variable can have any Boolean value, and this is the usual situation for a variable. The $always - false$ is used when the relation $B(a, b, 0)$ exists, and, expressed as $B(a, b, c)B(c, 0, 0)$, results into $c = false$ and the node representing the variable *c* will be $always - false$. The $always - true$ value of a node models exactly the linear equations construction of correlation surfaces, where the search-constraints were appended as supplemental unit clauses containing the corresponding input variable as a positive literal (see Section 4.5.7).

The graph-edges represent Boolean equivalences between the variables associated to the nodes, and for this reason the condition imposed by a 3-cycle on the values of the nodes can be compared to a parity checker ($a \leftrightarrow bc$ results in $B(a, b, c)$). If one assigns Boolean values to all the nodes, in each 3-cycle only two or zero nodes will be allowed to be *true*. Another analogy for this behaviour is that it emulates the valid assignments of a the *B*-clause, where either one variable is *false* or all three are *false*.

For visualisation purposes, and to simplify the way identities are presented, in the following sections the graph-representation and the *B*-notation will be used alternately.

5.1.3 Variables

The previous discussion focused on the properties of the notation, showing that the notation works like a layer on the vast number of logical implications existing due to the braiding relation. The graph representation was directly based on the underlying Boolean implications existing between the variables. Although variables were extensively used throughout the previous sections, it was not entirely mentioned how these are introduced or removed. From a Boolean perspective, the following observations will be straightforward, but in order to maintain the analogies to circuit geometries, the effects of the transformations will be detailed.

Introducing variables

The braiding-relations were introduced to represent correlation surfaces and the variables were defined based on the graph-representation of the TQC geometric description. Variables will be introduced in the *B*-notation for performing operations on the abstracted TQC circuit, and in general the variables express a transitivity of the surface construction. A first option for introducing a variable is to use the logical equivalence between an existing variable a and the new variable x, resulting in the *B*-clause $B(a, x, 0)$. When expressing the correlation equivalence between more than 3 variables, the temporary variables used (see G_i in Equation 5.10) were also introduced for achieving transitivity between the clauses.

Introducing variables can result in the construction of junctions, too. The $S \oplus T$ in Equation 5.7 corresponds to a third variable, e.g. R, thus $B(R, S, T)$ is formed. If the introduced variable will represent a sheet, then a junction is formed having the two previous sheets as components. The variable R is the linear combination of S and T, and its introduction expresses the possible correlation equivalence existing between S and T (see Figure 5.2b).

Removing clauses and variables

In the preceding example, variables were removed starting from their transitive role. The notation is based on the conjuction of *B*-clauses and by removing variables the corresponding clauses could be disregarded, too. In terms of the graph representation, the removal is

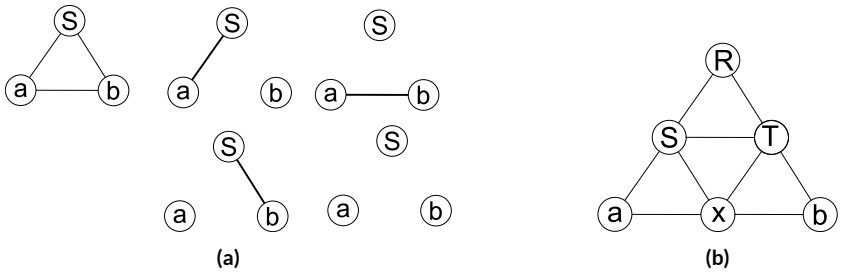


Figure 5.2: The graph representation of the B-notation: a) A B-clause is represented as three nodes connected by three edges (top left), and there are four possibilities to represent the Boolean equivalences, where a single edge (dashed) exists between the three nodes; b) Introducing the variable R using $B(R, S, T)$ generates another 3-cycle in the graph already representing $B(S, a, x)$ and $B(T, x, b)$.

the inverse of operation of the one from Figure 5.2b, where variable R was introduced to express $S \oplus T$. The removal of transitive variables will be analysed through two scenarios of tubes and sheet interactions around braiding-points.

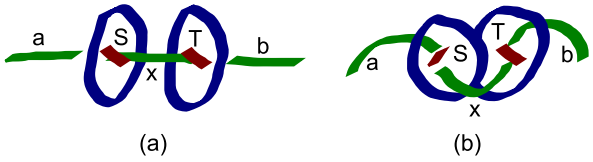


Figure 5.3: Correlation surface transitivities: a) Tube transitivity, where the tube x is existing between the sheets S and T, and a correlation surface, that includes a but not b, is valid either if it is connected directly to S or to T by x; b) Sheet transitivity, where the linear combination of the sheets S and T (ST) is correlation equivalent to the tube x, and a correlation surface, that includes both a and b, is valid if it is connected to either ST or to x.

The *tube transitivity* (see Figure 5.3a) scenario takes place when a tube-variable is transitive between two sheets, similar to when in a circuit's geometry two braids share the same tube (see Equation 5.6). The scenario is accompanied by the strong assumption $x \equiv ST$, stating that the tube x is correlation equivalent with the sheets S and T, which implies the existence of $B(S, T, x)$. It is also possible not to assume the $x \equiv ST$ equivalence, and this will be later used for illustrating TQC circuit identities.

The *sheet transitivity* (see Figure 5.3b) scenario occurs when a sheet has a transitive role between two other sheets, similar to the sheet constructions at junctions. At a junction, only two out of three sheets need to be considered, because the third one is the linear combination of the other two. The third sheet-variable could be removed from the expression, while the same conclusion was drawn to reduce the number of cycles considered from a bridged-geometry (see Section 4.3.1).

A reconsideration of Equation 5.6 in conjunction with the two previous scenarios indicates that Equation 5.7 can be rewritten as a single B-clause. In Equation 5.6 it was considered that x could be removed if only a single braiding-relation contains x. The circumstances

will be detailed in the following paragraphs.

The removal of x can be executed in two ways. The first option, according to tube-transitivity, is to introduce the clause $B(S, T, x)$, which will result into $B(b, S, 0)$ and $B(a, T, 0)$: the outer tubes are logically equivalent to the neighbouring sheet. From the perspective of TQC circuits, the tube x acts like a sheet, for it conditions the separate existence of a and b by $B(x, a, b)$. As a result, S and T are disregarded due to their equivalence to the a respectively b tube, and the tube x is *replaced* with a new sheet-variable*.

$$\begin{aligned} & B(S, a, \underline{x})B(T, \underline{x}, b)B(S, T, \underline{x}) \\ = \; & B(S, a, x)B(T, x, b)B(S, T, x)\underline{B(b, S, 0)} \\ = \; & B(\underline{Sb}, a, x)B(T, x, \underline{Sb})\underline{B(a, T, 0)} \\ = \; & B(Sb, \underline{Ta}, x)B(\underline{Ta}, x, Sb) \rightarrow B(x, a, b) \end{aligned}$$

The second option uses a direct approach by introducing the sheet-variable R that joins S, T into a junction (sheet-transitivity): the clause $B(R, S, T)$ is appended. The tube-transitivity used another supplemental clause which was interpreted as "the variable x represents the option of choosing between S and T", and the final step was to replace x with a sheet-variable. But this time, referring to Equation 5.10, the variables x, S and T have a transitive role, and act as temporary variables being not present into other B-clauses. The variables fulfill a purely transitive role and the final form of Equation 5.6 is $B(R, a, b)$.

Introducing and removing variables is visualised using 3-cycles for the B-notation. For this particular example, Figure 5.2b illustrates how three 3-cycles (the initial three clauses) can be abstracted to a single 3-cycle representing $B(R, a, b)$. If one *splits* a B-clause into three sub-clauses, the inverse operation will be performed, and three variables automatically introduced. For the previous example these would be S, T and x.

Removing variables is comparable to the transitive reduction of a directed graph [AGU72], which is a often used procedure during the pre-processing of CNF formulas that are submitted to a SAT solver. Although the graphs of the B-notation are undirected, for this comparison the edges can be decomposed into directed ones (see Figure 5.4a). The *transitive closure* of a graph G is defined as the smallest subgraph G' of G, such that there is a path from vertex u to vertex v in G', whenever there is a path from u to v in G [AGU72]. Through transitive reduction of a graph, its closure can be computed, and for DAGs (directed acyclic graphs) the transitive closure is unique, while graphs containing cycles do not have a unique closure [AGU72]. The graph from Figure 5.4 is cyclic, and the two possible closures are illustrated in Figure 5.4b and Figure 5.4c.

A graph where no transitive reductions are possible is its own transitive closure, and such graphs are sought during the correlation surface construction process (e.g. when variables are removed). A correlation surface, defined as the path connecting nodes from the B-graph, is based on the transitive relations from graphs abstracting a circuit's geometry. For exam-

*In the following, underlined parts of the expressions are used to indicate transformations.

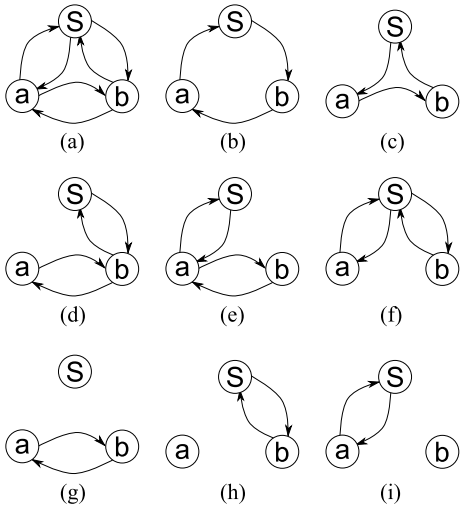


Figure 5.4: The transitive closure of the 3-cycle: a) An undirected 3-cycle is transformed into a directed graph; b) A first possible transitive reduction of the directed 3-cycle; c) The second possible transitive reduction; d,e,f) A 3-cycle where two undirected edges were transformed into pairs of directed edges, and the transitive closures are equivalent to the graphs from b) and c); g,h,i) The transitive closure of the graphs equals the graphs themselves.

ple, the directed 3-cycle from Figure 5.4f results if the construction of the tube *a* implies the construction of sheet *S*, which in turn implies the construction of tube *b*.

The transitive closure of a 3-cycle having two undirected edges that were decomposed into four directed edges (see Figures 5.4d,e,f) is equivalent to the transitive closures of the graphs from Figure 5.4b,c. It results, that 3-cycles with two undirected edges are not valid surface constructions: a single undirected edge is required. Nevertheless, the graphs from Figures 5.4g,h,i are their own transitive closures, meaning that no further transitive reductions are possible, and no additional node transitivities exist.

Operations in the *B*-notation are visualised using graphs formed of 3-cycles, where in each 3-cycle only one undirected edge exists, and that edge corresponds to the Boolean implication between the variables represented as nodes. The properties of the *B*-notation, like Equation 5.7, are thus expressing the operations such that the resulting graphs are not reducible. Introducing variables increases the transitivity (number of paths) of the graph-representation, whereas variable removal reduces the transitivity.

For the *B*-notation graphs there will be more than one possible transitive reduction, meaning that a certain correlation surface could be constructed in different ways. This observation is supported by the unrestricted construction of correlation surfaces. The method presented in the previous chapter did not focus on the number of graph-nodes (for the graph-representation of a TQC circuit) or on the number of variables (for the Boolean representation).

5.2 TQC circuit identities

The TQC computational paradigm was introduced as a model of computing in an error-corrected manner, where the cluster state is used as a computational resource, which is expected to be efficiently used. As a result, circuit identities that reduced the required dimensions of the cluster state were investigated in the literature. The identities allowed the transformation of the geometric descriptions without affecting the implemented computation. The initial identities were introduced by Raussendorf in [RHG07] and had been the only ones known until the bridging-rule (see Section 4.2) was enunciated. None of these identities were proven in a compact formal manner. The Raussendorf identities were mentioned as being correct after listing all the correlation surfaces supported by the geometries supposed to be equivalent, while the proof of the bridging-rule was textually presented.

This section uses the *B*-notation, its graph representation and the geometric descriptions to present proofs of circuit identities in a compact way. A constructive approach is employed, by starting from basic identities which gradually serve to illustrate more complex ones. In order to reduce the length of some of the proofs at certain times, only the geometric description will be referred to.

5.2.1 The no-braid

The first circuit identity, the *no-braid* illustrated in Figure 5.5a, is similar to an axiom, and showing its correctness is not possible through the *B*-notation. However, this identity can be understood as a practical example for removing variables and clauses.

Some of the *B*-notation properties were deduced for arbitrary types of variables that appeared in the *B*-clauses, and for those properties no assumptions were made about how many times a variable appeared in a circuit's Boolean expression. The target of the circuit identity is to show that there are situations when the simplification of geometries is done by discarding *trivial* braids. A pattern for recognising such braids is the following: if a tube-variable appears in a single *B*-clause, locally the geometry is similar to the one depicted in Figure 5.5a where the tube x is involved in two braids on the same surface. The corresponding clauses can be removed.

The proof proceeds without recursing to the introduction of variables, but in order to show that the corresponding $B(S, a, x)$ can be removed, the definition of surface parities (see Definition 2 and Proposition 3) are necessary.

Let $P(x) \subset TQCC$ be the subset of cluster qubits involved in the parity $par(x)$. But $P(x)$ is also a subset of $P(S)$, thus let $P(S') = P(S) \setminus P(x)$ resulting in $par(S') = par(Sx)$. Furthermore, $P(x) = P(a) \cap P(S)$, thus $P(S) \cup P(a) = P(S') \cup P(a)$ results in $par(Sa) = par(S'a)$. The reason for introducing S' is to decouple the definition of sheet S from from the tubes a and x. The sheet S is a *deformation* of sheet S by the tube x (see Section 4.2), and it follows that $par(\neg Sax) = par(ax)$ and $par(\neg S\neg a\neg x) = par(\neg a\neg x)$. As mentioned in Section 4.2 by Definition 14, the equality of the parities leads to the equivalence of surfaces,

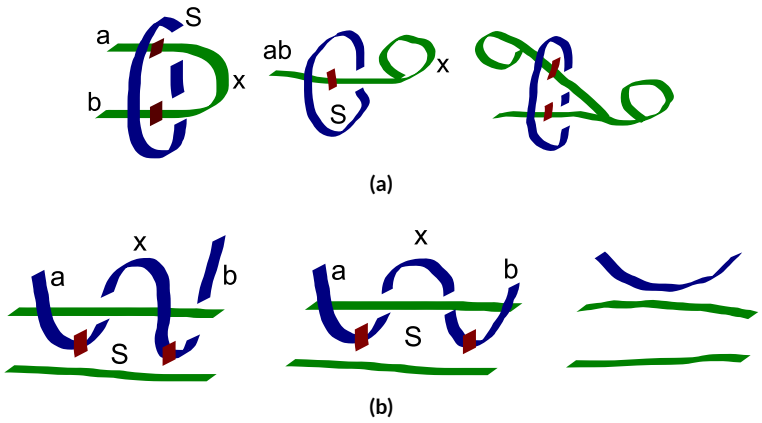


Figure 5.5: a) The no-braid circuit identity can be redrawn as a single tube that intersects a ring-sheet, and this construction can be repeated; b) The double-braiding identity, where the tube x can redrawn behind the green defect, and it does not affect the existing braiding relations.

and, as a result:

$$\begin{aligned} B(S,x,a) &= (S \wedge a \wedge \neg x) \vee (S \wedge \neg a \wedge x) \vee (\neg S \wedge \neg a \wedge \neg x) \vee (\neg S \wedge a \wedge x) \\ &= (S' \wedge a) \vee (S' \wedge \neg a) \vee (\neg a \wedge \neg x) \vee (a \wedge x) \\ &= S' \vee (a \leftrightarrow x) \end{aligned} \tag{5.11}$$

The logical disjunction $\vee$ from Equation 5.11 indicates that the existence of sheet S' is independent of the two tubes a and x. This statement is equivalent to the braid being trivial. Although a defect interacts with a sheet, the correlation surfaces at the braiding-point, due to their equivalence, are constructed as if the braid did not exist.

The logical equivalence $a \leftrightarrow b$ shows in the no-braid scenario the equivalence between the expressions $B(S,a,x)B(S,b,x)$ and $B(S,ab,x)$. The last expression corresponds to the geometric description from the middle of Figure 5.5a. As no-braids are trivial circuit constructions, it is possible to introduce an arbitrary number of no-braids into a geometric description without changing the performed computation, and a potential construction is presented on the right side of Figure 5.5a.

5.2.2 Double-braiding

One of the first circuit identities presented in [RHG07] (see Figure 5.5b) is the *double monodromy* rule, which states that a defect braided twice around an opposite-type defect is equivalent to the *no-braid* case. Each braid represents a *CNOT* between the qubits c and t, and by using the quantum circuit formalism, this is easily proven by showing that $CNOT(c,t)$ $CNOT(c,t) = I \otimes I (CNOT = CNOT^{\dagger})$. However, the identity will be proven using the B-notation.

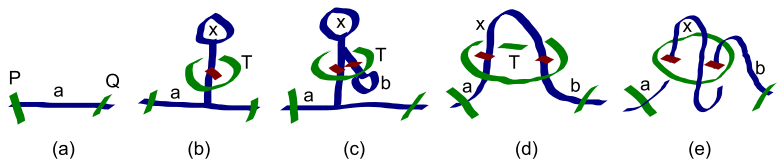


Figure 5.6: Introducing double braids starts with the introduction of no-braids, and continues with the applicaiton of the operation from Figure 5.1b.

It is possible to draw the initial geometry similarly to the no-braid from Figure 5.5a after using the property from Figure 5.1b and the double-braiding expression is

$$E_y = B(S,a,x)B(S,x,b) = B(S,x,a)B(S,x,b) \text{ (similar to Equation 5.4)}$$

The expression E_y implies $B(a,b,0)$, meaning that $a \leftrightarrow b$, and after replacing $ab = y$ in Equation 5.4, the final expression, $E_y = B(S,x,y)$, has an identical form to Equation 5.11. The tube-variable x is not used in any other braid-expressions, and, as a result, the initial two braids can be disregarded.

Double braids can be removed from the geometry, but these can also be introduced using the B-notation. For this example, the tube a will have to be double-braided with the sheet S, such that S will be involved in two B-clauses. For this, assume that a is defined between the sheets P and Q such that the local canonical expression is $B(P,\bullet,a)B(Q,a,\bullet)$ ($\bullet$ is used for variables that are not of interest during this example). The supplemental variables x and b are introduced using the no-braid relation, and the variable equivalence between a and b is used in order to construct the final geometry. The derivation of the geometric description accompanying the Boolean expressions is presented in Figure 5.6.

$$\begin{aligned}
B(P,\bullet,a)B(Q,a,\bullet) &\rightarrow B(P,\bullet,a)B(Q,a,\bullet)B(T,a,x) \text{ (Figure 5.6b)} \\
&\rightarrow B(P,\bullet,a)B(Q,a,\bullet)B(T,a,x)B(T,x,b) \text{ (Figure 5.6c)} \\
&= B(P,\bullet,a)B(Q,a,\bullet)B(T,a,x)B(T,x,b)B(a,b,0) \text{ (Figure 5.6d)} \\
&= B(P,\bullet,a)B(Q,b,\bullet)B(T,a,x)B(T,x,b) \text{ (Figure 5.6e)}
\end{aligned}$$

5.2.3 Ring rotation

In order to show the no-braid relation, the Boolean formalism had to be supplemented with the concept of the surface parities. The proof of circuit identities uses another TQC specific behaviour: the construction of tube pairs in the presence of sheets. The outcome wil be that, as long as a sheet is involved in only two braids and the tubes corresponding to the ring are not involved in other braiding relations, the ring can be *rotated* (see Figure 5.7a).

The key idea is to show that the sheet acts like an intermediary between the tubes and has a transitive role (sheet transitivity). For the left-hand circuit, E_p^1 is the canonical representation of the two braids, while the rotated ring generates E_p^2, and both expressions imply the

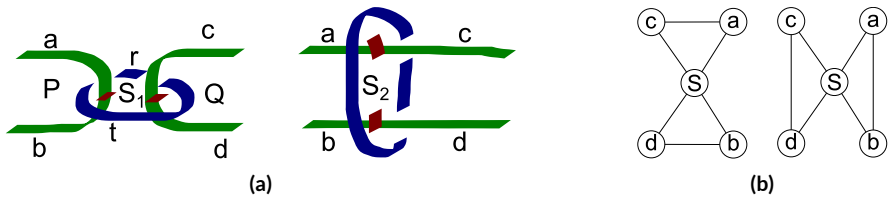


Figure 5.7: a) The ring rotation circuit identity; b) The graph representation of the rule.

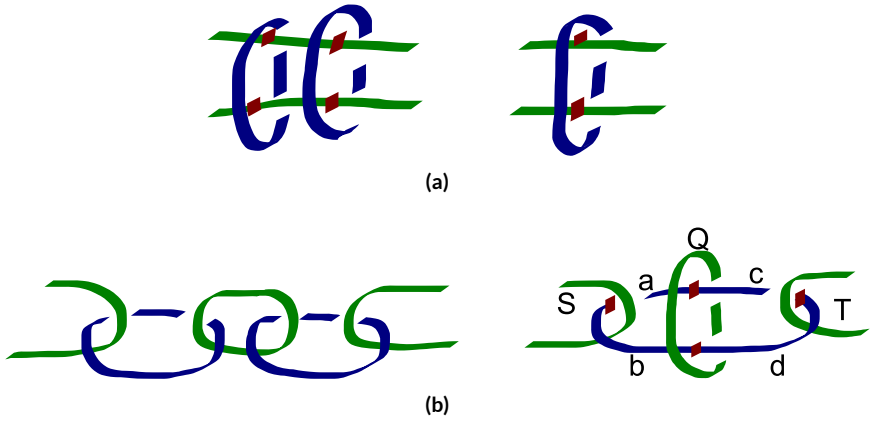


Figure 5.8: a) The rotate ring circuit identity; b) The two rings circuit identity.

same relation between the tube-variables. The rotated ring introduces two supplemental braids in the dual space (expression E_d^2), but the affected sheets (P, Q) are logically equivalent.

$$\begin{aligned} E_p^1 &= B(S_1, a, b) \wedge B(S_1, c, d) \rightarrow B(a, b, c \oplus d) \\ E_p^2 &= B(S_2, a, c) \wedge B(S_2, b, d) \rightarrow B(a, b, c \oplus d) \\ E_d^2 &= B(P, r, t) \wedge B(Q, r, t) = B(PQ, r, t) \rightarrow (P \leftrightarrow Q) \end{aligned}$$

In the previous expressions, the sheets S_1 and S_2 are not logically or correlation equivalent as they were previously described. From the tubes point of view the sheets result in the same tube pairs being constructed. Figure 5.7b illustrates the ring rotation using the graph-representation of the B-notation. The central graph-node S represents the function $a \oplus b \oplus c \oplus d$, while the same function is supported by the central node if the 3-cycles were modified according to the *rotated* braiding-pattern.

5.2.4 Two rings

A further circuit identity involves the presence of two consecutive rings surrounding the same pair of defects. There are two possibilities to show that only one of the rings is in fact

non-trivial, while the other one can be removed. Applying the ring rotation identity on both rings, the geometry from Figure 5.8a is redrawn similar to Figure 5.8b , and, after a second rotation of the opposite type ring, the geometry will be identical to the right side in Figure 5.8b. The proof will show that the sheet Q can be disregarded, thus decoupling the ring from the geometry and highlighting the braids related to Q as trivial.

The $B(Q, a, b)$-relation is appended to the initial canonical expression E_p that involves Q. This is possible because of ab is correlation equivalent to S ($ab \equiv Q$). Using Equation 5.4 it follows that $E_p \leftrightarrow B(SQ, a, b)$ and $S \leftrightarrow Q$. The sheet Q is logically equivalent to S and can be disregarded. A further implication of E_p is that $S \leftrightarrow T$, which is consistent with the rotation of a single ring. Q is *absorbed* by S, the associated ring *disappears*, and the geometry (after applying a ring rotation again) will contain a single ring.

$$\begin{aligned} E_p &= \underline{B(S,a,b)}B(Q,a,c)B(Q,b,d)B(T,c,d)\underline{B(Q,a,b)} \\ &= B(SQ,\underline{a},b)B(SQ,a,c)B(SQ,\underline{b},d)B(T,\underline{c},\underline{d}) \\ &= B(SQ,a,b)B(T,c,d)\underline{(a \leftrightarrow d)(b \leftrightarrow c)} \\ &\rightarrow B(SQ,ad,bc)B(T,ad,bc) \end{aligned}$$

An equivalent approach to showing the identity is to start from the simplified assumption that both S and T are involved in only one braid (for multiple braids the procedure is the same). The geometry will resemble the one in Figure 5.3a. After using Equation 5.5 and introducing the variable R to represent ST (R is the linear combination of the sheets S and T, $R \equiv ST$), the final form of the expression is shortened. Because $\neg R \leftrightarrow (S \leftrightarrow T)$, the three B-clauses in the parantheses will indicate the no-braid situation, such that if one is interested in a circuit that connects the tubes a and b, then the final form of the expression will be correct. The graph-representation of these equations is presented in Figure 5.2b, where the 3-cycle at the top represents the new variable R.

$$\begin{aligned} E_p &= B(S,a,x)B(T,x,b) = B(S,a,x)B(T,x,b)B(S \oplus T,a,b) \\ &= B(R,a,b)(\underline{B(R,S,T)B(S,a,x)B(T,x,b)}) \rightarrow B(R,a,b) \end{aligned}$$

5.2.5 The Raussendorf-ring

This circuit identity[RHG07] shows that a ring, representing a logical qubit measurement, can be removed from the geometric description, as long as the ring is *surrounding* a single pair of opposite-type-defects, and is not braided with other defects. Assuming that the ring is dual, it will define a primal surface S, and a dual tube r. The tube r is not braided with any other surfaces and it will not appear in the canonical expression of the dual space. Hence, only the removal of the sheet S will have to be motivated.

The behaviour of the sheet has to mimic the two geometric caps appended for measuring the pair of tubes in Figure 4.16a. The sheet is associated to a logical measurement, which implies that either the tubes a, c are connected to S (i.e. $a \leftrightarrow c$) or the tubes b, d are connected

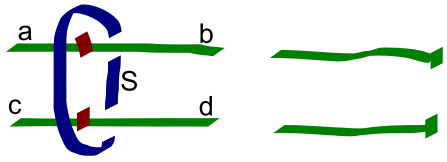


Figure 5.9: The Raussendorf-ring rule. The horizontal segments at the end of the green defects imply that the logical qubit is logically measured similarly to the measurement pattern in Figure 4.16a.

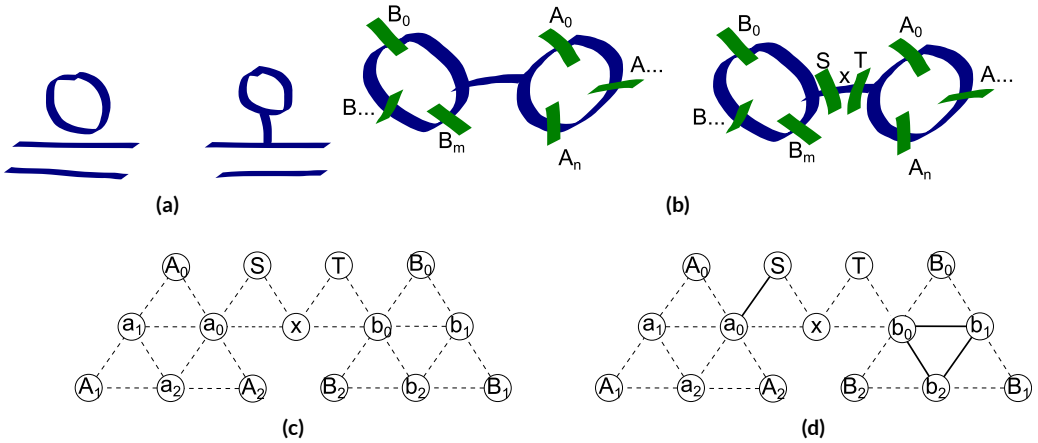


Figure 5.10: The circuit bridge identity: a) A sketch of the identity; b) For two separate rings that are braided with the sheets $\mathcal{A}_0 \ldots \mathcal{A}_n$ and $\mathcal{B}_0 \ldots \mathcal{B}_n$, respectively, two separator sheets S and T are introduced along the bridge; c) The B-notation graph before searching for a correlation surface construction; d) The construction of a valid tube a_0 can either stop in the separator S, or be connected with a ring of all the b_j tubes, if none of the B_j sheets are considered.

(i.e. $b \leftrightarrow d$). In both cases, $B(S,a,c)S(S,b,d)(a,c,0)$ leads to $(S,0,0)$. This is a situation identical to the no-braid case.

As a result, having $S = \mathit{false}$ and r not considered, the initial ring from the geometric description is removed (if the circuit specification considered it a logical measurement). The conclusion is that, in general, single rings *cannot be removed* if their interpretation as circuit elements is not known. The logical equivalence between tubes is enforced *iff* the rings are *always* interpreted as logical measurements.

5.2.6 Bridging

The structure of closed defects arises in bridged-geometries (see Figure 5.10a), and it was mentioned in Section 4.3.1 that only a reduced set of cycles in the structure is required for correlation surface construction. The relationships existing in bridged-geometries between their tubes and sheets is formally introduced in this section.

For the beginning, let *two* completely unbraided (disjoint) closed geometric structures exist in the geometric description of a TQC circuit. Each of the two geometries may be

braided with another geometries of the circuit, and not even by transitivity are the disjoint structures interacting. For a circuit containing g geometric structures, let the birdging and braiding interactions be signaled by the function $inter : \{1 \ldots n\}^2 \rightarrow \{true, false\}, i, j < n$. Two interacting components r, t are signaled by $inter(r, t) = true$; otherwise $inter(r, t) = false$. The *inter* function is transitive $inter(i, j) = inter(i, k)inter(k, j)$. In an arbitrary geometry, for two un-bridged un-braided components r, t: $interact(r, t) = false$.

From the computational point of view, the underlying cluster, where the geometries were mapped, supports two separate computations (one by each geometric structure). For simplification purposes, each of the previous two geometries is considered a ring. After bridging these two geometries, the result will be a bridged-geometry: $inter(r, t) = true$, where $r = t$ (bridging results in a single structure) and the circuit will consist of $g - 1$ geometric structures.

The effect of bridging on the two independent sheets is investigated. Each of the two rings supported a sheet, and the resulting bridged-geometry supports three individual sheets: one for each ring, and a sheet as the linear combination of the other two. This is the exact situation to when sheets were constructed at junctions (see Section 4.3.1). From the sheet-perspective, the resulting computation still supports the previous two disjoint computations, but also a combined computation (a maybe unwanted effect). Accordingly, initially there were two separate computations being executed, but, even though the bridged geometry creates an impression of a single supported computation, the two computations are actually performed, too.

The effect on the construction of tubes remains to be investigated. In order to ensure that the two geometries are still disjoint, two separator sheets (S and T) are introduced, as in Figure 5.10b. The separators will ensure that, if the hypothetical tube a has to be constructed, then the separator $S = true$, and if b has to constructed, then $T = true$. Even if a and b have to be constructed simultaneously (although the computations are disjoint); the separator sheets will keep the computations disjoint (the tube x connecting the two sheets is not of interest). This scenario is different from the two rings circuit identity: in that one the tubes a and b were considered belonging to the same computation, and variable replacement simplified the circuit.

The triviality of the separators is shown by placing them into no-braid scenarios. For the geometries indiced by r and t, let a_i denote the tubes constructed using r, and b_j the tubes from t. The parity of a closed defect structure is always even, and, as a result, all the tubes associated with the given defect structure need to be connected. For example, for the r-th structure with n tubes a_i, the relation $a_i \leftrightarrow a_{i+1}$ for $i \in [0, n)$ with $a_0 = true$ has to hold. The E_t and E_r expressions express the relation between the tubes of the r, t-geometries and the complete set of sheets that are braided with these geometries. The Boolean expressions represented as graphs will additionally contain the separator sheets S, T and the tube x be-

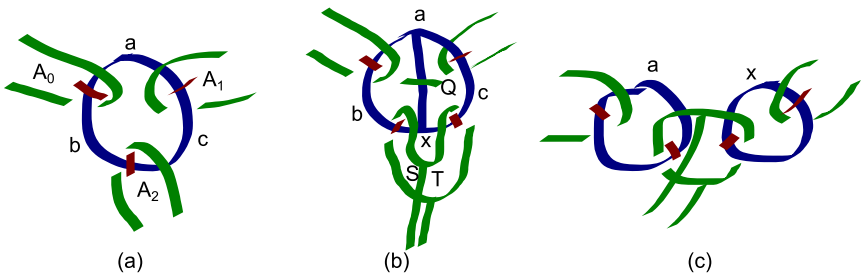


Figure 5.11: The splitting rule takes a sheet and splits it into multiple sheets connected by tubes forming junctions.

tween the separators (see Figure 5.10d).

$$E_r = \bigwedge_{i=0}^{n} B(\mathcal{A}_i, a_i, a_{i+1}); \quad E_t = \bigwedge_{j=0}^{m} B(B_j, b_j, b_{j+1})$$

Let $Sa\neg x$ denote the construction that guards against a being *wrongly* connected to the tubes from the second geometry, thus making sure that is possible to consider the computations separately. If $\neg Sax$ is considered, the no-braid relation arises if $x \leftrightarrow b_0$, and, as a result, all the sheets will not be enabled: $B_{j-1} \leftrightarrow B_j, B_0 = \mathit{false}$. The direct result is that the separator S is not needed. A similar approach shows that the separator T becomes trivial, too.

5.2.7 Splitting

The circuit identity resulting through bridging tube-variables was described in [FD12], and its inverse operation, *unbridging*, is a direct result of separating geometries. There exists the possibility to *split* closed geometric structures, where it is unknown if that structure was previously bridged or intended for performing two or a single computation. Unbridging assumes that bridging was performed in a previous step, whereas splitting does not require this assumption.

In the following it will be shown how a geometric structure is split, as long as the resulting disjunct closed defect structures (e.g. r, t) are still interacting with each other (in the terminology of last section, $\mathit{inter}(r, t) = \mathit{true}$). If the interaction is not performed through tubes (the geometries are not bridged), it can be achieved through braids. An example of such a circuit identity is presented in Figure 5.11.

Let the geometric description contain a single ring that is braided with n other sheets $\mathcal{A}_i$ that could be enabled in any possible configuration. The target of the discussed circuit identity is to show that a ring can be split into two separate rings, which interact through an independent junction. Without affecting the generality, let $n = 3$ (see expression E_1). In order to construct the junction, the variables x, S, T are introduced, accompanied by their

B-clauses:

$$\begin{aligned} E_1 &= B(\mathcal{A}_0, a, b)B(\mathcal{A}_1, a, c)B(\mathcal{A}_2, b, c) \text{ (Figure 5.11a)} \\ E_2 &= E_1 B(S, b, x)B(T, c, x)\underline{B(\mathcal{A}_2, S, T)} \end{aligned}$$

A correlation surface involving, for example, only the tube a could be required to be supported, and it is unsure if a may be allowed to be directly connected to x, and a separator sheet Q will be introduced. The sheet Q ensures that the tube-surface a is still supported by the new geometry. A no-braid situation appears, due to surface equivalence, $Q \equiv xST$ ($par(xQST)$ is always even), where xST acts similar to the trivial tube from Figure 5.5a. The situation is eliminated by disabling the sheet Q. Hence, as $Q = false$ and $B(Q, 0, 0)$, it is implied that $(a, x, 0)$ and $x \equiv ST$, thus E_4 is formed.

$$\begin{aligned} E_3 &= E_2\underline{B(Q, a, x)} \text{ (Figure 5.11b)} \\ E_4 &= E_3\underline{B(Q, 0, 0)} = E_3\underline{B(a, x, 0)}B(\mathcal{A}_0, a, b)B(\mathcal{A}_1, a, c) \\ E_5 &= B(\mathcal{A}_0, a, b)B(\mathcal{A}_1, x, c)B(S, a, b)B(T, x, c)B(\mathcal{A}_2, S, T) \text{ (Figure 5.11c)} \end{aligned}$$

The result from Equation E_5 can be interpreted as the way of constructing, for example, the correlation surface $\mathcal{A}_0 a \mathcal{A}_1$ by using the transitivity of the sheet ST, where $\mathcal{A}_2 = ST$ represents the third sheet supported at the junction $B(\mathcal{A}_2, S, T)$ (see Figure 5.3b. From the perspective of x from E_4, whenever it is *true*, the sheets S and T are either both enabled or not at the same time, which leads to $x \leftrightarrow \neg\mathcal{A}_2$.

$$\begin{aligned} \mathcal{A}_0 a \mathcal{A}_1 &= \mathcal{A}_0 a(SxT)\mathcal{A}_1 \text{(Equation } E_4\text{, Figure 5.11b)} \\ &\rightsquigarrow \mathcal{A}_0 a(ST)x\mathcal{A}_1 \text{ (Equation } E_5\text{, Figure 5.11c)} \end{aligned}$$

As a conclusion, the two bridged rings can be unbridged, and the initial sheet is thus split into separate sheets. Nonetheless, the sheets are not computationally disjunct: the newly introduced junction offers the required transitivity to construct all the previously supported surfaces.

5.2.8 The Raussendorf-cage

The *cage* rule, named after its effect on the geometric description, was the central TQC circuit identity defined in [RHG07]. The rule transforms a set of braids into a set of bridged-rings around defects. Before presenting the proof of the identity, the example from Figure 5.12 is useful to sketch how equivalent circuits are constructed. The initial circuit contains tubes that are extended outside the ring, in [RHG07] called *legs*. The circuit identity will be shown in two parts: initially without the *legs*, and afterwards the legs will be connected to a single geometric structure. Without affecting the generality, the same geometric descriptions from Figure 5.12 is used: three braids on a sheet that are generated by a ring with legs.

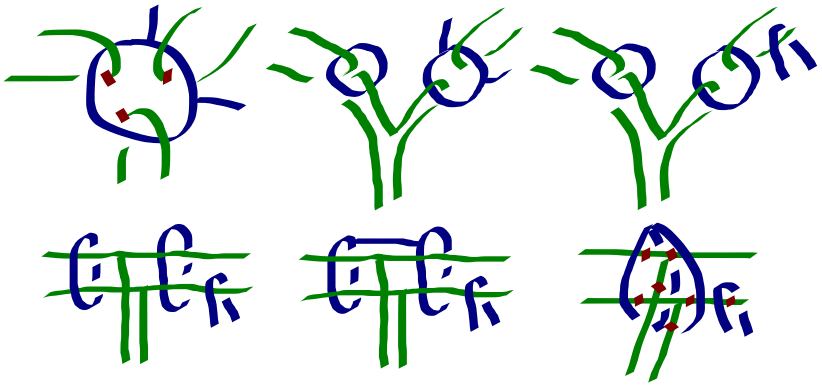


Figure 5.12: The Raussendorf-cage rule.

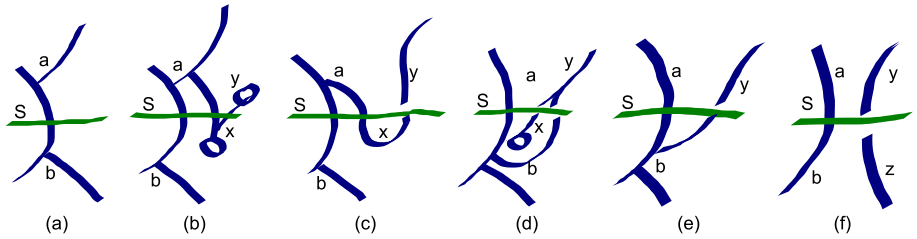


Figure 5.13: The decoupling of the legs from two tubes is illustrated using the geometric description. After inserting two no-braids with the variables x, y, the geometry is rearranged such that the leg from a is moved to b, and then the resulting pair of legs is unbridged by introducing the variable z.

The junction- and cage-construction part are shown using circuit-identities introduced in the previous sections. Figure 5.12 illustrates the steps of the proof using the geometric description. The ring is split into multiple sub-rings that interact through junctions, each resulting ring is rotated, and then the rings are bridged.

For the second part of the circuit identity, the legs will be moved around the ring. This is possible by introducing supplemental tube-variables. Connecting the legs a and b is expressed using the B-notation, by introducing twice a no-braid situation: $B(S,a,x)B(S,x,y)$. The equivalence $a \leftrightarrow y$ implies that the geometric representation of the final expression resembles the situation where the leg a has moved to b.

$$\begin{aligned} B(S,a,b) &= B(S,a,b)B(S,a,x)B(S,x,y) \text{ (Figure 5.13b)} \\ &= B(S,a,b)B(S,a,x)(x,b,0)B(S,x,y) \text{ (Figure 5.13c)} \\ &= B(S,a,b)B(S,b,y)B(S,x,y) \text{ (Figure 5.13d)} \\ &= B(S,a,b)B(S,b,y) \text{ after removing the no-braid (Figure 5.13e)} \end{aligned}$$

Having moved all the legs onto a common one, e.g. b, the resulting geometric structure can be separated from the ring, by introducing the variable $z \leftrightarrow b$ (see Figure 5.13f). Unbridging is enabled for the new defect (yz) consists of tube-variables equivalent to the initial

tubes a, b with legs. The procedure presented in Figure 5.13 performs the decoupling of the legs z and y from the initial tubes a and b.

5.2.9 Conclusion

The existing TQC circuit identities could be proven using the B-notation. The properties of the notation were used to reformulate the transitive relations existing between the clauses of the Boolean expressions, and this approach was similar to the transitive reduction of graphs.

The initial circuit identities existing in the TQC literature have not been derived in a constructive manner, and the previous attempts at showing their correctness have been based on exhaustive enumerations of the supporting correlation surfaces. Using the B-notation, some simple identities were used to construct more complex ones, and a proof of the Raussendorf-cage rule was offered. In the process of illustrating the identities a new circuit identity was discovered, namely the splitting of surfaces. It is not known if splitting, bridging and the Raussendorf identity are the only possible circuit identities, but for the time being they represent the basis of the following TQC design algorithms.

5.3 Synthesis of TQC circuits

Circuit synthesis is the process of expressing a high-level description in form of a circuit that consists of basic elements (logic gates). Automating this process by means of algorithms is necessary in order to streamline the circuit design phase. In the context of TQC, circuit synthesis refers to the automatic generation of geometric descriptions using the quantum circuit formalism as an input. The work presented in this section was partialy described in [PDNP12] and consists of investigations of TQC circuit synthesis.

Universal quantum computations are in general performed using a reduced and discrete set of quantum gates, whose direct application is compatible with the underlying encoding. The TQC computational model is defined using *CNOT* as the two-qubit entangling gate, and single-qubit gates are applied via teleportations to a data qubit by utilising an ancilla prepared in either the $|A\rangle$ or $|Y\rangle$ state. The geometric description abstracts qubits initialised into these states as injection points, and these belong to one of the sets IO_t of the graph-representation (see Section 4.1.7).

The problem statement is different from the synthesis problem in the field of reversible circuits (Boolean circuits consisting of *CNOT* gates and other bijective gates). Reversible circuit synthesis [SPMH03,PHM04,SM13,GAJ06,WD09] takes either a high-level description or a gatelist of a circuit as input, and calculates an (optimised) gatelist with an equivalent functionality. In contrast, the presented algorithms generate a geometrical description, and no agressive optimisations [PF13,FD12] of the geometry are performed, although such operations could improve the physical hardware requirements necessary for TQC.

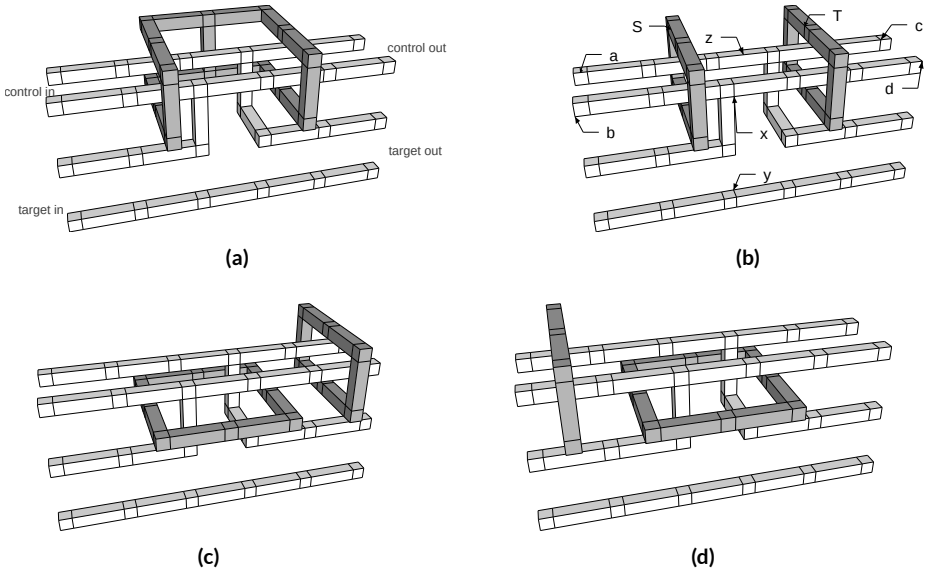


Figure 5.14: Equivalent representations of a CNOT in TQC.

5.3.1 The *CNOT* gate in TQC

CNOT gates are built in the TQC model by braiding defects of opposite type, and by definition the dual defect is the control and the primal defect is the target. However, it is possible to construct a CNOT between defects of the same type by using four braids and three logical qubits (pairs of defects), and this mechanism is illustrated in Figure 4.6a. Synthesising TQC circuits, where the *CNOT* is an operation between qubits of the same type, is a more straightforward approach, and equivalent representations of the gate will be investigated. TQC circuits are large networks of CNOT gates applied on a large set of qubits initialised into a discrete set of quantum states (see Section 4.1.9). As a consequence, the realisation of CNOTs is a key aspect during the synthesis.

There are four equivalent geometrical descriptions of the CNOT gate. By using the Raussendorf-cage circuit identity, the initial description (Figure 4.6a) can be transformed into the second one (see Figure 5.14a). The third representation is illustrated in Figure 5.15. The validity of the fourth representation, not completely demonstrated in [PDNP12], is formally proven in this section using the properties of the *B*-notation from Section 5.1. This is shown by starting from the braid-*CNOT*, where the dual ring is split (a junction is created), and the two resulting dual rings are rotated.

There are three possible combinations of how the two dual rings encircle the defect pairs joined at the junction (see Figures 5.14b,c,d). The configurations are equivalent if implementing a CNOT, and, if any configuration of the three is bridged, the result will be the cage-*CNOT*. However, for the proof, the two rings will not be joined into a cage.

The validity of the braid CNOT was already shown in Section 4.1.6. In the following, it

is shown that a CNOT can also be constructed using a junction and a single ring. In Figure 5.14b, the upper pair of defects represents the control qubit, whereas the lower pair the target qubit. This is expressed by the B-notation:

$$B(S,a,z)B(S,b,x)B(T,c,z)B(T,d,x)$$

The number of rings will be reduced after investigating the correlations supported by the geometry. An implementation of the CNOT stabiliser table requires constructing the $X_cI_t \rightarrow X_cX_t$ and $I_cX_t \rightarrow I_cX_t$ correlation surfaces. Surfaces associated to the entries $Z_cI_t \rightarrow Z_cI_t\, I_cZ_t \rightarrow Z_cZ_t$ are a direct result of sheet constructions at junction (sheets represent Z-correlations in the primal space).

The correlation transfer from both tubes associated to qubit defects has to be assured (see Proposition 4), and the $XI \rightarrow XX$ surface construction (tubes represent X-correlations in the primal space) requires that $(a \oplus b)(c \oplus d)(z \oplus y)$. The reformulated canonical expression is:

$$B(S \oplus T, a \oplus c, 0)B(S \oplus T, b \oplus d, 0)B(S,a,z)B(T,d,x)$$

The relation $(a \oplus c) \leftrightarrow (S \oplus T)$ includes a contradiction of the $a \leftrightarrow c$ relation that was conditioned on the construction of the tubes: therefore $(a \leftrightarrow c) \leftrightarrow (S \leftrightarrow T)$. This result indicates that in order to construct $XI \rightarrow XX$, the sheets S and T have to be logically equivalent (either both are enabled, or both are disabled). This behaviour is consistent with the valid surfaces constructed after visually inspecting the geometry: *azc* and *bSzTd*.

The construction of $IX \rightarrow IX$ is similar, as it is conditioned by $(x \oplus y)(z \oplus y) = y \oplus xz$. The $B(a \oplus b, x \oplus z, 0)$ will result to $a \leftrightarrow b$ because of the xz condition, and $B(c \oplus d, x \oplus z, 0)$ will result to $c \leftrightarrow d$; thus $B(S, ab, xz)B(T, cd, xz)$ results in $B(S \oplus T, ab \oplus cd, 0) = B(S \oplus T, 0, 0) = S \leftrightarrow T$. Once more, the logical equivalence between the sheets is consistent with the surface constructions: y and *xSTz*.

The $S \leftrightarrow T$ relation indicates that the two rings can be reduced to a single one placed according to the geometry of the ring-*CNOT*. That particular ring is the linear combination of the previous two, and for implementing the *CNOT* gate in TQC no other relations between rings surrounding the junction are required.

The geometric properties of the ring-*CNOT* recommend it for being used in the TQC circuit synthesis method presented in this section. The procedure is applicable to other CNOT versions with minimal modifications.

5.3.2 Synthesis algorithms

The synthesis of TQC circuits will be automated, and an important criteria when evaluating the output is the area of the synthesised field. The two algorithms presented and evaluated in this section receive the set of gates G as input, and output a *field* of *CNOT*s. The algorithms were designed starting from the assumption that the geometrical description of the

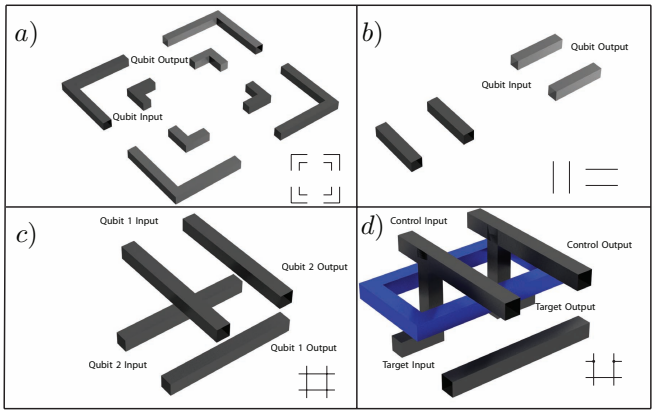


Figure 5.15: Compact topological representations of the identity gates and of the CNOT. Each encoded qubit is a pair of defects which always maintain a fixed separation from all other defects (except at junction points). This maintains the error correction strength of the code. The defect ring that encircles the CNOT is implicit in the 2D representation.

output is planar and will be *consumed* from the left to the right by the TQC hardware. The first gate will be the left-most in the field, which is generated on a per-gate basis. The computational complexity of the algorithms is linear in the number of gates.

The field of CNOTs

The algorithms use eight primitives (see Figure 5.15) to implement the *CNOT* and identity gates. Both the 3D realisation and the two-dimensional representation of the primitives are shown in the figure. The six primitives from Figure 5.15a,b perform the identity operation on one qubit, and the primitive in Figure 5.15c keeps two qubits unchanged (represented by horizontal and vertical pairs of defects). Although the two pairs of defects appear to touch each other in the two-dimensional representation, in 3D these pass underneath each other. From a 3D perspective, the CNOT (Figure 5.15d) has a discontinuous qubit represented as the target which is physically connected to the control. In two dimensions, the control qubit is vertical and the target qubit is horizontal. The ring that encircles the junction within the CNOT is implicit within the 2D representation.

There is a direct mapping from the two-dimensional representation to a 3D geometrical description, and assuming that for a computation the gates are placed on the same *layer*, then the compact two-dimensional representation is adequate for representing arbitrary circuits.

In the following *multi-target* CNOTs are used, and such gates defined as the gate $(c, t_0, \ldots t_k)$ having control c and targeting the qubits $t_0, \ldots, t_k$ can be decomposed into a series of CNOTs: $(c, t_0) \ldots (c, t_k)$. The field of *CNOT*s is a matrix-like regular structure having the following properties:

- the target qubits of each gate are located on the rows;

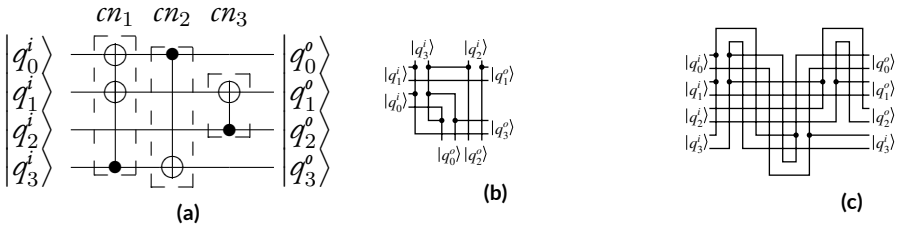


Figure 5.16: A circuit and two possibilities to construct the corresponding fields

- the control qubit of each gate is located on the columns.

In 3D the planes supporting the qubits are parallel (see Figure 5.15) and this geometrical property motivates the qubit arrangement.

The area of the field depends on the width (number of columns) and the height (number of rows), which in turn are influenced by characteristics of the quantum computation, such as number of gate and qubits. Throughout the following sections a distinction between cluster (a 3D structure with fixed depth) and two-dimensional field will be made. The cluster volume is linearly related to the field area, and for analysing the efficiency of the synthesis the field area will be referred.

Two alternative two-dimensional fields implementing the 4-qubit, 3-CNOT circuit from Figure 5.16a are shown in Figure 5.16b and 5.16c. Note that cn_1 is a multi-target CNOT (it has two targets) and is represented by two single-target CNOTs. The cn_2 operation is implemented on the second column of the field: qubit q_0 is transformed into a control for qubit q_3. Finally, cn_3 is implemented on the third column. Positions not occupied by CNOTs implement identities. The area is $3 \times 3 = 9$. The field in Figure 5.16c implements the same computation in a less efficient way using 12 identity gates, and has an area of $8 \times 6 = 48$.

The equivalent volume (see Section 4.1.7) of the ring-CNOT is 16. Hence, each two-dimensional cell in the CNOT field (the intersection of a row and a column) requires in 3D the volume of a ring-CNOT. For example the field of area 48 occupies an equivalent volume of 48×16.

In the following algorithm the *control* and *targets* functions are used for inferring the qubits that act as controls or targets of the CNOT synthesised at a given timestep.

Unbounded synthesis

Algorithm 7 constructs the field on a per-gate basis, and inserts qubits involved in the computation as they are needed. If a target qubit does not exist in the circuit created so far, it will be inserted on a row, and if a control qubit is needed, it will be inserted directly on a column. Also, if for a given gate that is currently synthesised, an existing target qubit is used as a control, then the qubit will be moved to the next free column. An existing control qubit needed as a target will be first moved to the next free row, and then to the next free column.

Algorithm 7 Unbounded: Moving the control downwards

1: Initialise an empty field
2: Start with no qubits inserted in the field
3: for all CNOT $cn \in G$ do
4: for all $q \in \mathit{targets}(cn) \cup \{\mathit{control}(cn)\}$ do
5: if q does not exist in the field then
6: insert a row/column for q
7: else
8: move qubit q to the next free row
9: end if
10: end for
11: move all $q \in \mathit{targets}(cn) \cup \{\mathit{control}(cn)\}$ to the next free column
12: move $\mathit{control}(cn)$ to the next free row using it as control for $\mathit{targets}(cn)$
13: end for

Algorithm 8 Bounded: Weaving of qubits

1: Compute $|Q|$ from G
2: Initialise a field with $|Q| + 2$ rows
3: Place each qubit $q_i \in Q$ on the row $i + 1$
4: for all CNOT $cn \in G$ do
5: Select the target qubit $\mathit{control}(cn)$
6: Transform $\mathit{control}(cn)$ into a control qubit
7: Continue $\mathit{control}(cn)$ while using it as a control for $\mathit{targets}(cn)$: a) downwards to row $|Q| + 1$; b) to the right one column; c) upwards to row 0; d) to the right one column; e) downwards to row $\mathit{control}(cn) + 1$.
8: Transform $\mathit{control}(cn)$ back into a target qubit
9: end for

The fields generated by Algorithm 7 typically have the main diagonal occupied, leading to a suboptimal area utilisation. With regard to the field area, the worst-case scenario occurs when the control of each gate was the control of the previous gate, and the number of targets equals the maximum number of qubits: $A_1^w = |G|^2 \times (|Q| - 1) \times |Q|$.

Bounded synthesis

A second algorithm bounds the number of rows, and leads to a more efficient and predictable placement of gates. The set of gates G is analysed before the synthesis and the number of qubits $|Q|$ involved in the quantum computation is extracted.

When $\mathit{control}(cn)$ is transformed into a control and back into a target, the control qubit is moved from a row to a column and back, and, as a result, the output of Algorithm 8 re-

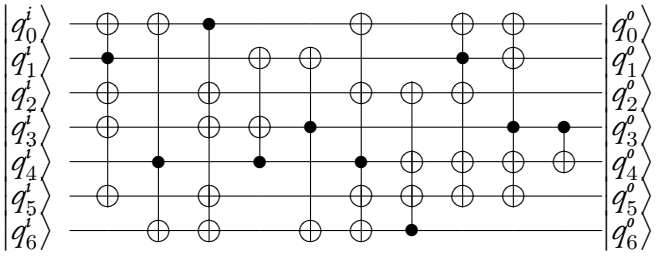


Figure 5.17: A sample circuit used for the bounded and the unbounded synthesis.

sembles a *fabric of qubits*. The transformations are equivalent to the movements of a qubit inside a TQC cluster (see Figure 5.15), and all the possible targets are affected due to the control qubit movement pattern.

The algorithm requires two *buffer* rows (the highest and the lowest) to allow the control qubit to change its direction from upwards to downwards, and/or the other way around. While a complete movement of the control qubit *occupies* three columns of the field, the worst-case area of the output field will be $\mathcal{A}_2^w = 3 \times (|Q| + 2) \times |G|$. However, the area of the output field is improved by incorporating additional heuristics into the algorithm. For example, information extracted from the set of targeted qubits could be used for the control qubit movement. If, for a given *CNOT cn*, all the target qubits are placed above the control qubit ($\forall t \in G, \mathit{control}(cn) > t$), then the control will be only moved upwards in the field. The downwards movement is triggered when all the targets are placed below the control ($\forall t \in G, \mathit{control}(cn) < t$).

5.3.3 Results

The presented algorithms were implemented, and using Figure 5.18 and Figure 5.19 the typical output fields generated by each algorithm can be visually compared. The quantum computation for which the fields were synthesised is depicted in Figure 5.17: it is applied on 7 qubits, and consists of 10 CNOTs targeting maximum 4 qubits.

For the evaluation of the algorithms, randomly generated circuits consisting entirely of CNOTs were used. The simulation results are presented in Table 5.1. The simulations were parametrised for 10, 100, 1,000 and 10,000 gates operating on 10, 100 and 1,000 qubits. The column $\mathcal{MT}$ contains the value of the maximum size of $\mathit{targets}(cn)$ per randomly generated CNOT, and the columns *Columns* and *Rows* contain the average number of rows and columns obtained by the algorithms after executing them 100 times for each combination of simulation parameters. Algorithm 8 always generates a field with $|G| + 2$ rows, therefore column *Rows* has been omitted from the table. It can be observed that the $\mathcal{MT}$ parameter is directly influencing the average number of columns and rows obtained through Algorithm 7, but the results of Algorithm 8 are not as strongly affected.

The *Red* column contains the reduction obtained when utilising the bounded synthesis instead of the unbounded algorithms. The unbounded algorithm can also perform better

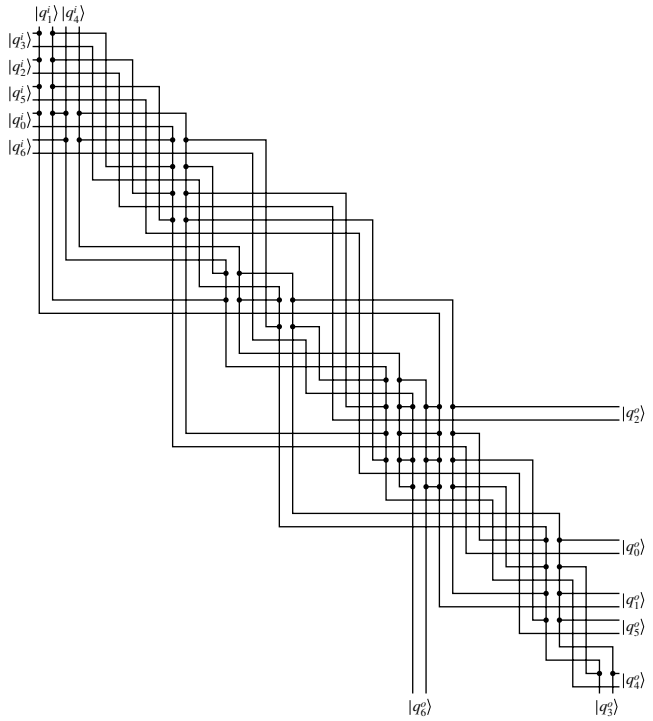


Figure 5.18: Output field of the unbounded synthesis.

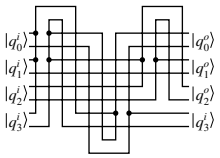


Figure 5.19: Output field of the bounded synthesis.

than the bounded algorithm (an example is shown in Figs. 5.16b and 5.16c). The simulation results contain such a situation: circuits with 100 qubits and 10 CNOTs having a maximum of 20 targets (see the highlighted entry in Table 5.1). In general, unbounded synthesis tends to perform better for circuits with less gates and less targets per CNOT.

Overall, the average area of the fields synthesised by the unbounded algorithm is larger compared to the average field area obtained with the bounded synthesis. This effect is particularly significant for large circuits like the ones consisting of $10,000$ CNOTs.

The layout of the output fields is used as a criteria for comparing the algorithms. There may be situations when a synthesised computation will be repeatedly used as a subroutine of a larger computation. All input and output qubits are placed on the same row, and the bounded algorithm will deliver better results, thus enabling a more convenient way of connecting subroutines.

There are situations, too, when the unbounded algorithm performs better then the bounded:

for small circuits with a reduced maximum number of targets per CNOT. An example is offered in Figure 5.16. Nevertheless, the deterministic functionality of the algorithms is an advantage, and the area of the output field can be computed beforehand.

5.3.4 Conclusion

A first automatic synthesis method of TQC circuits was proposed and its implementation was evaluated. Two deterministic synthesis algorithms were presented, and their performance was analysed by comparing the areas of the output circuits. Both algorithms used a novel geometric reprentation of the topological CNOT gate, and the correctness of the gate has been proven using the original formalism developed in the previous sections of this work.

The synthesis methods were based on the observation that TQC circuits consist of three parts: an initialisation part where both circuit inputs and injection points are set up, a second part consisting of a CNOT gate network, and a part where all the logical qubits are measured. The representation allowed the abstraction of the CNOT subcircuit using a two-dimensional field of topological CNOT gate representations. Future work will focus on investigating the feasibility of stacking the two-dimensional field in 3D.

5.4 Validation of TQC Circuits

The primary goal of TQC circuit synthesis was to enable an automated procedure that not only performs the required translation from a quantum circuit to a TQC circuit, but also optimises the volume of these structures to ultimately reduce the physical resources needed by the hardware. An example of an optimised circuit was presented in Figures 4.8a and 4.8b.

Validation of synthesised circuits is a necessity, because the synthesis software may be buggy or use incorrect compaction rules. Optimised circuits often bare little resemblance to their original geometric description, and need to be validated, too. Moreover, manually performed circuit compaction destroys any validation results and a new validation round is needed.

An automated validation method is required, as large topological circuits are complex objects, where the gatelist is difficult to be extracted and unfeasible to verify manually. In this section arbitrary geometric descriptions representing TQC circuits are considered for validation. The structures are not presumed to be synthesised by the algorithm from Section 5.3.1 and, hence, the geometries are not only 2D arrangements of CNOT gates. The methods described in this section were partialy presented by the author in [PDNP14b].

TQC circuits are specified using their geometric description and the stabiliser table that should be supported. The geometric description references the set IO (see Section 4.1.9), and the injection points set IJ. The validation of a TQC circuit will start from a circuit specification $QCS = \{ST, IO, IJ, \mathcal{M}\}$, where ST is a stabiliser table and $\mathcal{M}$ is the ordered set of

logical qubit measurements. Given an implementation QC, which is also mapped to a tuple $\{ST', IO', IJ', \mathcal{M}'\}$, the validation procedure establishes the circuit equivalence of both descriptions ($QC \equiv QCS$).

Logical qubits can contain injection points anywhere along the geometric structure, and the target of validation is to check that, before the injection point will be measured, every logical qubit is correctly stabilised. Otherwise the result of the rotated measurement will be faulty, and the whole quantum computation is compromised. Assuming that the number of injection points and their order of measurement is not changed, as it will directly affect the computation being performed, this question is reduced to the equivalence checking of the stabiliser circuit parts ($ST \equiv ST'$), which has been previously investigated in [WGMD09,AG04].

However, checking the equivalence of a TQC geometric description against the specification QCS is more challenging because the geometrical description of a TQC circuit would have to be translated to a stabiliser table. However, no such method is currently known. In the following, different validation approaches are outlined, which check if a given stabiliser table (specification) is supported by a circuit's geometry (implementation).

There are at least two possible validation scenarios. The *symbolic validation* is a method where each stabiliser entry from ST is constructed using the Boolean-expression-based approach (see Section 4.5.7). The *cross-layer* validation is a simulation based procedure of a cluster where the geometric description of the TQC circuit was mapped. The cross-layer scenario necessitates the construction of correlation surfaces, which also yields information used for inspecting the measurement-correlations existing between the correlation surface qubits.

5.4.1 Symbolic validation

The geometry of a TQC circuit is the support of various TQC computations. Recognising a computation based on the geometry requires supplemental information, like the IO set of inputs, or the location of the injection points. For example, the geometry of the braid-$CNOT$ (Figure 4.6a) is the same to the one used to encode using the repetition code. However, the difference between the braid-CNOT and the encoding circuit is the set of inputs and outputs. The CNOT has two inputs and two outputs, while the encoder assumes three inputs (the unencoded qubit and two ancillae) and three outputs (the encoded qubit).

For validation, the first criteria is to check that the geometry contains an equivalent set of injection points as specified by QCS. The second validation step is to check the construction of correlation surfaces representing the stabiliser transformations as enumerated in ST. A geometry is valid according to ST if it supports all the entries from the stabiliser table. The symbolic validation method (Algorithm 9) is the algorithmic description of the approach sketched previously.

The algorithm receives a geometric description and a specification as inputs, and tries to construct a valid correlation surface for each stabiliser entry from ST. The validation is

Algorithm 9 Symbolic validation

Require: Circuit *TQC* as a geometrical description and the specification *QCS*
1: for all Stabiliser s from *ST* of *QCS* do
2: Compute for s the correlation surface *CORS*
3: if $s = \emptyset$ then
4: return *TQC* is NOT valid according to *QCS*
5: end if
6: end for
7: return *TQC* is valid according to *QCS*

successful (Line 7) if all the entries were constructed, otherwise the geometry is not valid (Line 4).

5.4.2 Cross-layer validation

The equivalence of a geometric description against a specification *QCS* is checked by mapping the logical qubits to a cluster state and simulating the cluster. This a two step procedure. First, the geometrical description is mapped to an unmeasured cluster (see Section 4.5). In the second step, for every entry of the stabiliser table *ST* from the specification, the topological computation in the cluster is simulated using a (stabiliser) quantum circuit simulator. Note that the simulated geometry is largely given by the shapes of the logical qubits which are independent from the processed *ST* entry. The *ST* entries determine only the initialisation and measurement parts of the logical qubits (see Figure 4.16a).

The cross-validation, expressed as Algorithm 10, starts by mapping the geometry to a cluster (Lines 1, 2). The set *TQCC* is specified as a finite set of associated coordinates of physical qubits, that are marked for measurement in the X- or Z-basis and initialisation into $|+\rangle$, $|A\rangle$ or $|Y\rangle$. The 3D-coordinates correspond to the lattice geometry presented in Figure 4.4.

$$\begin{aligned} TQCC &= \{(x,y,z) |\ x,y,z \in \mathbb{N}, meas(x,y,z) \in \{X,Z\}, init(x,y,z) \in STATES\} \\ STATES &= \{|+\rangle, |A\rangle, |Y\rangle\} \end{aligned}$$

The mapping of defect geometries to the 3D lattice takes an initial cluster *TQCC*, where no measurements were marked, and updates it: Z-basis measurements for defect-internal physical qubits, and X-basis measurements for all others. Injection points (physical qubits initialised into $|A\rangle$ or $|Y\rangle$) are measured in the X-basis.

In the absence of errors (which is assumed during the validation of circuit functionality)[†] the topology of the 3D-cluster guarantees that the measurement parity of all the unit-cell face-qubits is even (see Section 4.2). The existence of the logical stabiliser support is proven

[†]The cluster measurement outputs of this method can be used to investigate the *effects* of physical qubit errors and their effect on the correlation surfaces.

by computing the parity of a correlation surface, as even parity indicates that the stabiliser is correctly constructed using physical-cluster qubits.

During the validation injection points do not need to be explicitly considered, and without affecting the correctness of the method, these are initialised into the $|+\rangle$ state. By *re-interpreting* the injection points, the $TQCC$ set is transformed into the $TQCC^{+}$ set from Line 2:

$$TQCC^{+} = \{(x, y, z) | x, y, z \in \mathbb{N}, meas(x, y, z) \in \{X, Z\}, init(x, y, z) \in \{|+\rangle\}\}$$

Similarly to classical circuits where input and output pins are used for the inputs and outputs of the circuit, $P_{in} \subset TQCC$ is a set of lattice coordinates of the physical qubits used for initialising the logical qubits. The same applies for physical qubits used for logical measurement. These are used to read the information from the TQC circuit; their coordinates are contained in the set $P_{out} \subset TQCC$ representing the *output pins*. The physical qubits from both sets are marked for either X- or Z-basis measurement, in order to respect the defect geometries from Figure 4.16a (Line 3). Cluster injection points are elements of P_{out}. Circuit simulation is performed for each logical stabiliser specified in ST, and the P_{in} and P_{out} sets will be constructed accordingly. A mapped cluster supports a logical stabiliser if the correlation surface that connects the corresponding input and output pins has even parity (Lines 19, 23).

In order to check the existence of all the logical stabilisers specified in ST, the validation method checks each entry in the table sequentially (Lines 4 – 23), similarly to the validation by construction. Depending on the logical stabiliser to be checked, the injection point coordinate will be marked in $TQCC^{+}$ (Line 7) with either an X-basis measurement (if the logical qubit should be stabilised by logical-X) or with a Z-measurement (if the logical qubit should be stabilised by logical-Z).

The support of a logical stabiliser is checked twofold: first by construction and then by simulations of the cluster. Each simulation involves multiple steps. The first step is to compute the correlation surface of the investigated stabiliser using the approach described in Section 4.5 (Line 8). A correlation surface connects the input to the output pins only for the logical qubits referenced by the stabiliser. In the second step all the physical qubits are measured according to their information from $TQCC^{+}$. In a third step, the existence of the stabiliser is determined based on the parity of the correlation surface (Lines 10 – 18). The error-correction is neglected, as it does not manifest itself in the validation of the specification $QC = \{ST, IJ, \mathcal{M}\}$.

The measurements performed during the second step are done in an arbitrary order, and a layered approach is adopted: one of the three dimensions of the cluster is defined to be the temporal axis. In a cluster of size $m \times n \times t$, the memory requirements are reduced by instead simulating a physical lattice of $t - 1$ *layer pairs* with $m \times n \times 2$ qubits of the cluster dynamically. Each layer pair (Line 13) consists of two cross-sections of the cluster (e.g. Figure 4.11a). Layer i contains all physical qubits with t-coordinate equal to i.

Algorithm 10 Cross-level Validation

Require: Circuit TQC as a geometrical description and the specification QCS
1: Compute $TQCC$ starting from the geometry of TQC
2: Compute $TQCC^{+}$ from $TQCC$ by marking injection points as $|+\rangle$ initialised
3: Compute $P^{q}_{in}, P^{q}_{out} \subset TQCC^{+}$
4: for all stabiliser s from ST of QCS do
5: $SIMTQC \leftarrow TQCC^{+}$
6: for all Logical qubit q stabilised by s do
7: Mark in $SIMTQC$ at $coord \in P^{q}_{in}, P^{q}_{out}$ the geometric patterns for initialisation and measurement of q according to s
8: Compute for s the correlation surface $CORS$
9: end for
10: $parity \leftarrow 1$
11: Construct layer l_0 of $SIMTQC$
12: for all Layer l_i of $SIMTQC$, $i > 0$ do
13: Construct l_i and Entangle with l_{i-1}
14: for all Cluster qubits cq in l_{i-1}, $cq \in CORS$ do
15: $ev \leftarrow$ measure cq in X-basis
16: $parity = parity \times ev$
17: end for
18: end for
19: if $parity = -1$ then
20: return TQC is NOT valid according to QCS
21: end if
22: end for
23: return TQC is valid according to QCS

In the i-th simulation run ($i = 0, \ldots, t-1$), layers i and $i+1$ are considered. The first simulation run considers only qubits from layers 1 and 2 with all connections between these layers. However, X and Z measurements are only performed on qubits with the t-coordinate 1. In the second simulation run, the qubits from the second layer, which retain their states from the first simulation run, are entangled with (hitherto unconsidered) qubits from layer 3 (initialised to $|+\rangle$) according to the unit-cell structure that is used throughout the complete $m \times n \times t$ cluster. Only second-layer qubits are measured, which influences the entangled third-layer qubits. This process is continued until the qubits of the final layer t are measured.

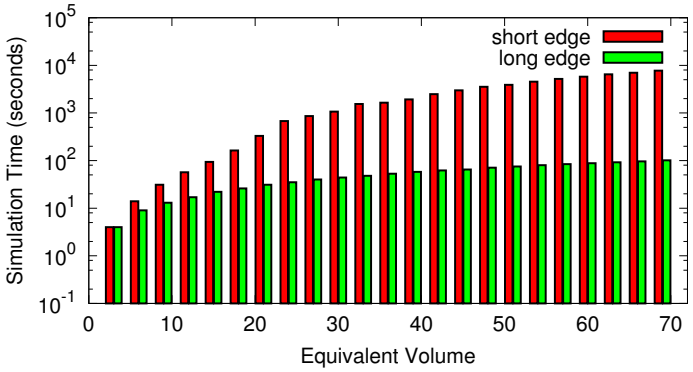


Figure 5.20: Simulation times after choosing different temporal axes.

5.4.3 Results

The practicality of the validation procedures is dictated by the size of the stabiliser table ST and the construction of each correlation surface. Checking a complete stabiliser truth table ST will require between 1 and $|ST|$ surfaces to be constructed.

Cross-layer validation

To evaluate the performance and the scalability of the cross-layer validation procedure the quantum circuit simulator CHP[AG04] was integrated for the cluster simulation step. TQC circuits consisting of logical *CNOT* gates acting on logical qubits were considered. Their sizes are expressed as an *equivalent volume*[FD12], a quantity that measures the volume of a topological structure compared to a set of independent regularly stacked logical *CNOT* gates. The results indicate that TQC circuits are feasible to simulate, and thus to validate. Average simulation times for one pair of layers in such circuits are reported in Figure 5.21. For example, the number of physical qubits required to be simultaneously simulated for the circuit having the equivalent volume of three *CNOT* gates was 1,462, and this number was 84,052 for the equivalent volume of 243 *CNOT* gates. The results suggest that even large and complex topological quantum circuits can be cross-validated in reasonable time.

The selection of one of the three axes in the cluster as the temporal axis is arbitrary, which provides an additional degree of freedom for the validation. The complete computation is confined to a 3D volume where the three edges may have different lengths. Selecting a short edge as the temporal axis will result in relatively small number of relatively large simulation instances, while selecting a long edge will require more simulations with less qubits per simulation. Note that the simulated functionality is identical for both options. Figure 5.20 compares the run times for these possibilities. The simulation is orders of magnitude faster when the longest edge is selected, and this is not surprising as the measurement of stabilisers is of quadratic complexity in the number of qubits. Considering less qubits per simulation instance outweighs the higher number of simulation runs.

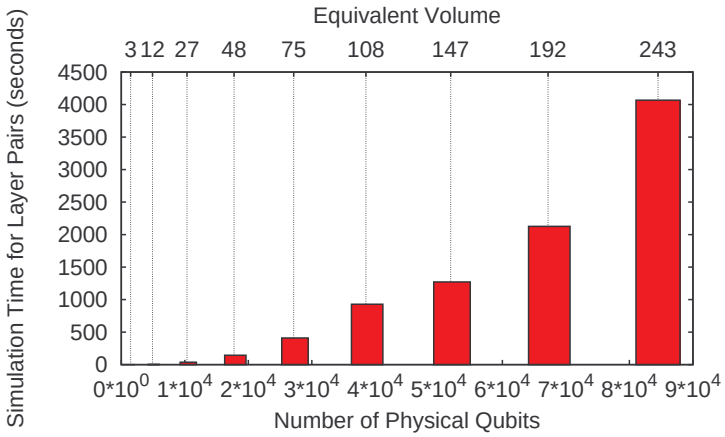


Figure 5.21: Average simulation times for pairs of layers.

Symbolic validation

This validation method was prototypically implemented using Java and sat4j[LBP+10] as a backend SAT-solver, and for visualisation purposes jreality was chosen[WGB+09]. A SAT-solver was preferred (although the problem is XORSAT) because some solvers include optimised mechanisms for reducing the CNF formula and for solving XORSAT. A further reason is that the solvers allow extensions of the *B*-notation that transform the validation problem into a pure SAT instance. The target of the simulations was to verify the correctness of the approach and not its performance. The scalability and speed of validation were secondary as sat4j is a Java implementation.

For example, the results of validating the Y-state distillation circuit are presented in Figure 5.22. The successful construction of each entry from the stabiliser Table 4.2 indicates the correctness of the circuit (see Line 7 in Algorithm 9). Following the discussion from Section 4.3.2 concluded in Proposition 4, there may be multiple valid surface constructions. Therefore, the result figures contain, when possible, two equivalent correlation surfaces (e.g. Figure 5.23).

In Figure 5.22 the yellow segments represent the primal defects, and the blue segments the dual defects. The grey dots on the segments illustrate the injection points named $\{R, O, Y, G, B, V, I\}$ together with the output point *out*. The thick green segments represent tubes and the red rectangles sheets resulting after constructing a correlation surface. The text in the parantheses after the name of the injection points (e.g. $Y(S)$ or $out(T)$) indicate that a particular point should be included on a *T*ube or *S*heet. For each entry in the stabiliser table of the circuit the figure contains two possible constructions of the correlation surfaces, except for the top right panel which represents the sheet containing the points $\{R, O, V, G\}$.

The failed validation of an incorrect circuit is signaled by the impossibility of constructing at least one of the correlation surfaces from the stabiliser specification. For example, a faulty ring-CNOT could have the dual ring missing and the circuit will be similar to the one

from Figure 4.18. The correlation surface connecting the input target to the output target, corresponding to the stabiliser transformation $IX \rightarrow IX$, cannot be constructed, and the circuit is not a valid CNOT (see Line 4 in Algorithm 9).

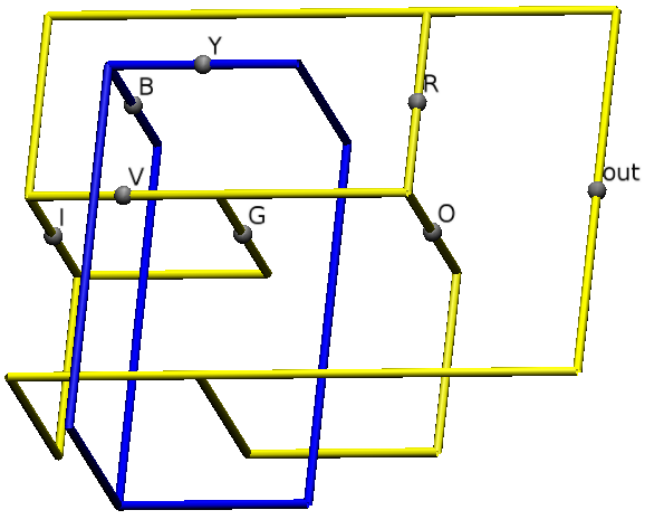


Figure	out	R	O	Y	G	B	I	V
Figure 5.23	X			X		X	X	
Figure 5.24		X		X		X		X
Figure 5.25			X	X			X	X
Figure 5.26	Z	Z	Z	Z				
Figure 5.27					X	X	X	X
Figure 5.28			Z	Z	Z	Z		
Figure 5.29		Z		Z	Z		Z	
Figure 5.30		Z	Z		Z			Z

Figure 5.22: The geometry and stabilizer table for the Y-state distillation circuit.

5.4.4 Conclusion

This section focused on both the simulation-based (cross-layer) and the symbolic verification of TQC circuits. The validation procedures compared an instance of TQC circuit with the circuit specification. The same observation from Section 5.3.1 was used: an arbitrary TQC circuit can be decomposed into fault-tolerant primitives where the resulting circuit contains a layer of injection points, a large network of CNOT gates, and an adaptive measurement layer. This observation led to the circuit specification to reference the stabiliser table of the CNOT subcircuit. Moreover, due to the difficulty of translating a circuit's geometric description into a stabiliser circuit, the verification problem was mapped to the problem of constructing correlation surfaces.

Table 5.1: Unbounded and Bounded Synthesis Results

|Q|	|G|	MT	Unbounded Synthesis			Bounded Synthesis		Red
			Rows	Cols	Area	Cols	Area	
10	10	2	18	13	2.7e+02	17	1.9e+02	1.42e+00
10	10	5	33	19	6.2e+02	18	2.1e+02	2.95e+00
10	10	9	57	30	1.7e+03	19	2.2e+02	7.73e+00
10	100	2	222	160	3.6e+04	159	1.9e+03	1.89e+01
10	100	5	381	238	9.0e+04	171	2.1e+03	4.29e+01
10	100	9	589	341	2.0e+05	179	2.2e+03	9.09e+01
10	1,000	2	2,292	1,645	3.8e+06	1,587	1.9e+04	2.00e+02
10	1,000	5	3,832	2,413	9.3e+06	1,710	2.1e+04	4.43e+02
10	1,000	9	5,887	3,440	2.0e+07	1,786	2.1e+04	9.52e+02
10	10,000	2	22,780	16,389	3.7e+08	15,832	1.9e+05	1.95e+03
10	10,000	5	38,512	24,254	9.3e+08	17,102	2.1e+05	4.43e+03
10	10,000	9	59,140	34,567	2.0e+09	17,855	2.1e+05	9.52e+03
100	10	20	88	20	1.8e+03	20	2.0e+03	9.00e-01
100	10	50	234	77	1.8e+04	20	2.1e+03	8.57e+00
100	10	99	473	194	9.2e+04	20	2.1e+03	4.38e+01
100	100	20	1,104	561	6.2e+05	188	2.0e+04	3.10e+01
100	100	50	2,559	1,284	3.3e+06	194	2.0e+04	1.65e+02
100	100	99	5,067	2,535	1.3e+07	196	2.0e+04	6.50e+02
100	1,000	20	11,391	6,154	7.0e+07	1,868	1.9e+05	3.68e+02
100	1,000	50	26,425	13,666	3.6e+08	1,930	1.9e+05	1.89e+03
100	1,000	99	50,928	25,916	1.3e+09	1,959	2.0e+05	6.50e+03
100	10,000	20	114,003	61,960	7.1e+09	18,676	1.9e+06	3.74e+03
100	10,000	50	264,204	137,056	3.6e+10	19,294	2.0e+06	1.80e+04
100	10,000	99	509,695	259,799	1.3e+11	19,577	2.0e+06	6.50e+04
1,000	10	200	806	88	7.1e+04	21	2.1e+04	3.38e+00
1,000	10	500	2,183	629	1.4e+06	21	2.1e+04	6.67e+01
1,000	10	999	4,541	1,778	8.1e+06	21	2.1e+04	3.86e+02
1,000	100	200	9,781	4,450	4.4e+07	198	2.0e+05	2.20e+02
1,000	100	500	24,787	11,947	3.0e+08	200	2.0e+05	1.50e+03
1,000	100	999	50,039	24,572	1.2e+09	200	2.0e+05	6.00e+03
1,000	1,000	200	100,963	50,491	5.1e+09	1,976	2.0e+06	2.55e+03
1,000	1,000	500	249,909	124,959	3.1e+10	1,990	2.0e+06	1.55e+04
1,000	1,000	999	499,209	249,607	1.3e+11	1,994	2.0e+06	6.50e+04
1,000	10,000	200	1,015,910	512,465	5.2e+11	19,756	2.0e+07	2.60e+04
1,000	10,000	500	2,514,665	1,261,836	3.2e+12	19,886	2.0e+07	1.60e+05
1,000	10,000	999	5,001,083	2,505,044	1.3e+13	19,936	2.0e+07	6.50e+05

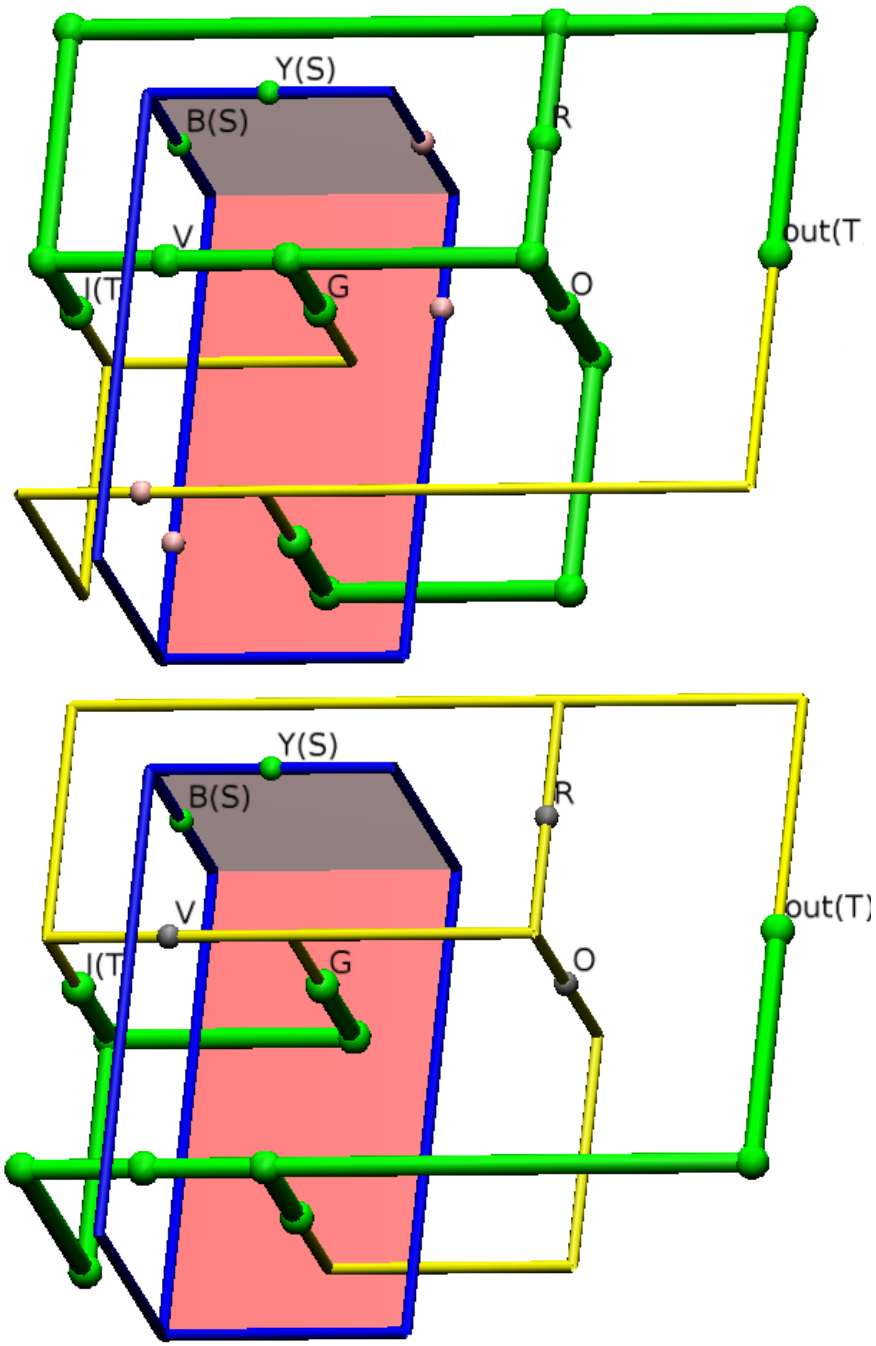


Figure 5.23: The correlation surface $X_{out}X_YX_BX_I$.

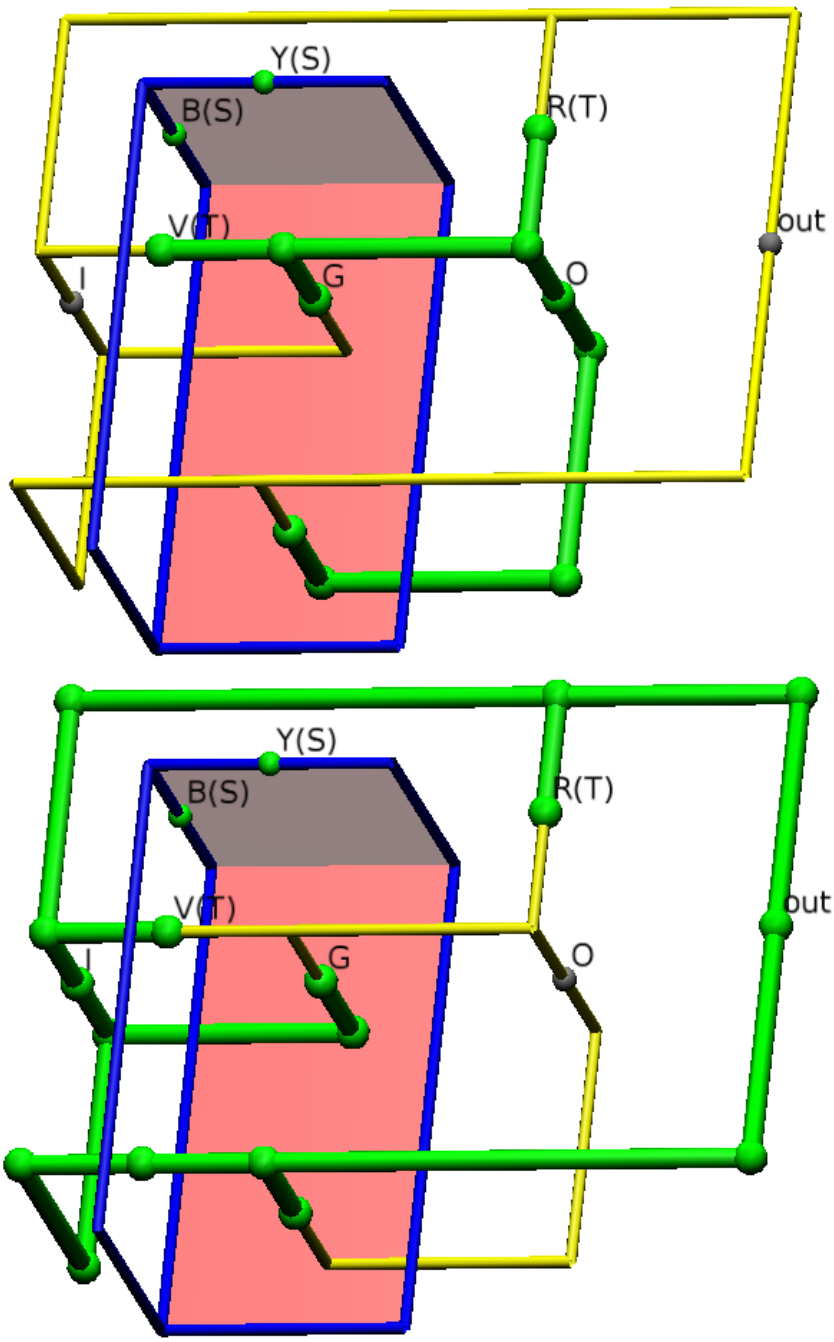


Figure 5.24: The correlation surface $X_R X_Y X_B X_V$.

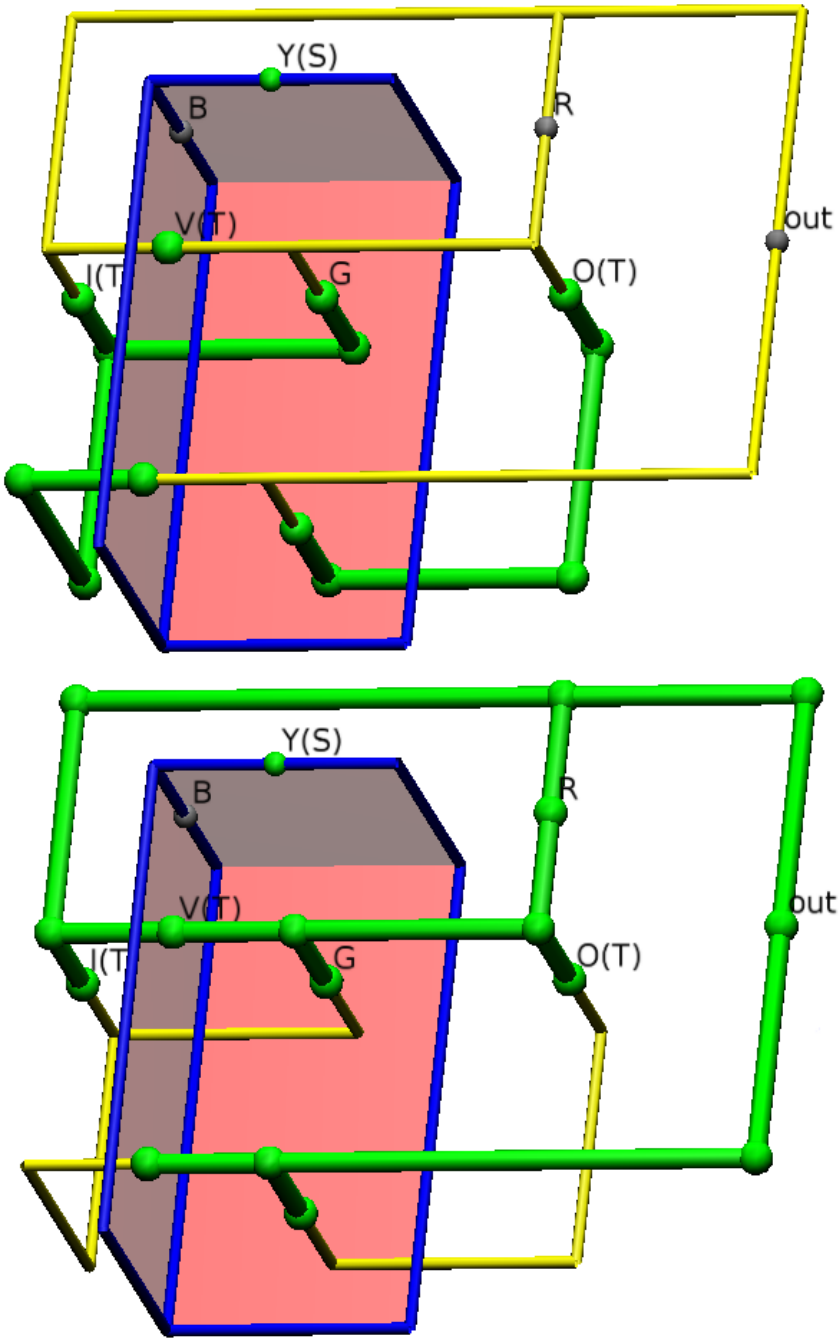


Figure 5.25: The correlation surface $X_O X_Y X_I X_V$.

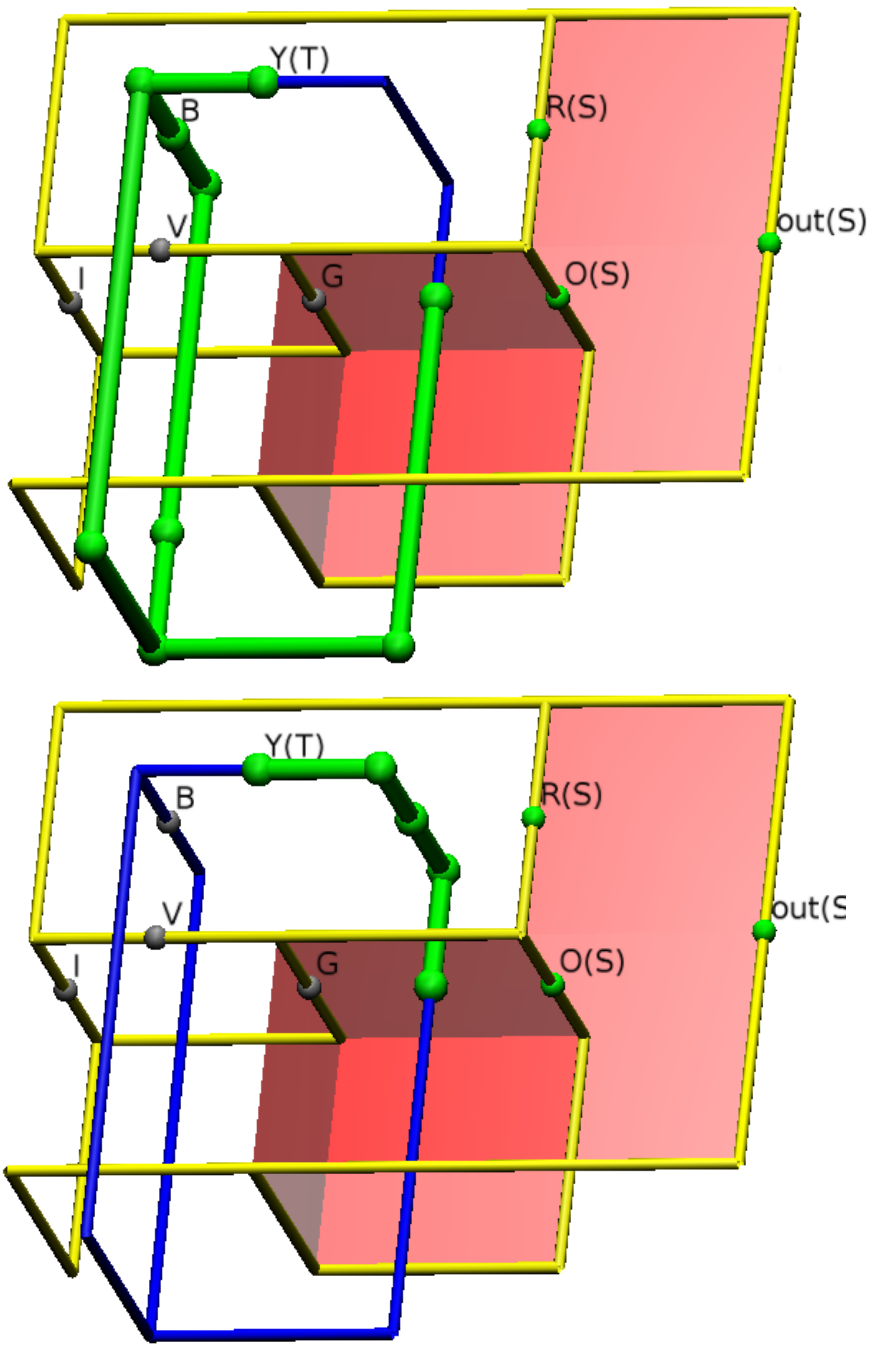


Figure 5.26: The correlation surface $Z_{out}Z_RZ_OZ_Y$.

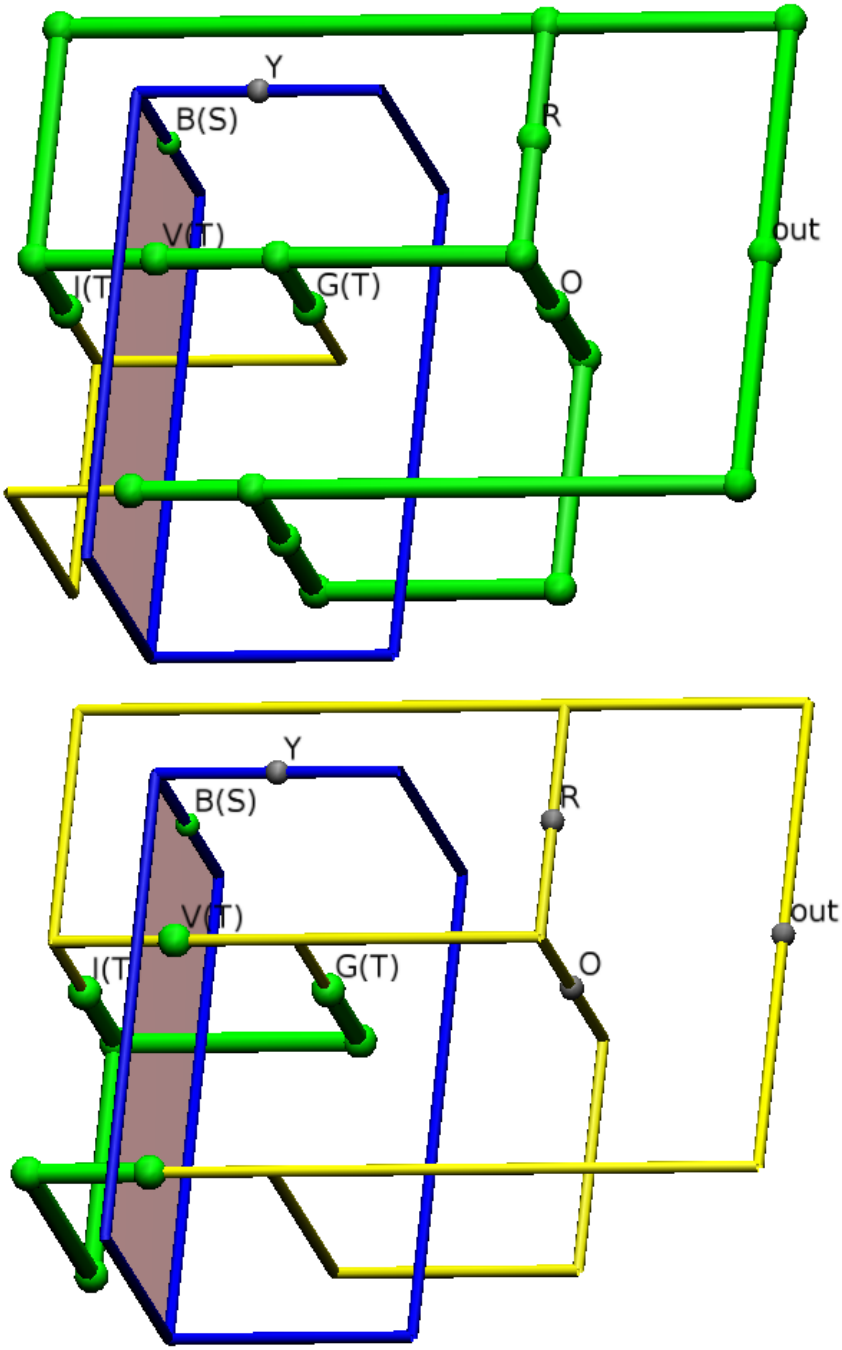


Figure 5.27: The correlation surface $X_G X_B X_I X_V$.

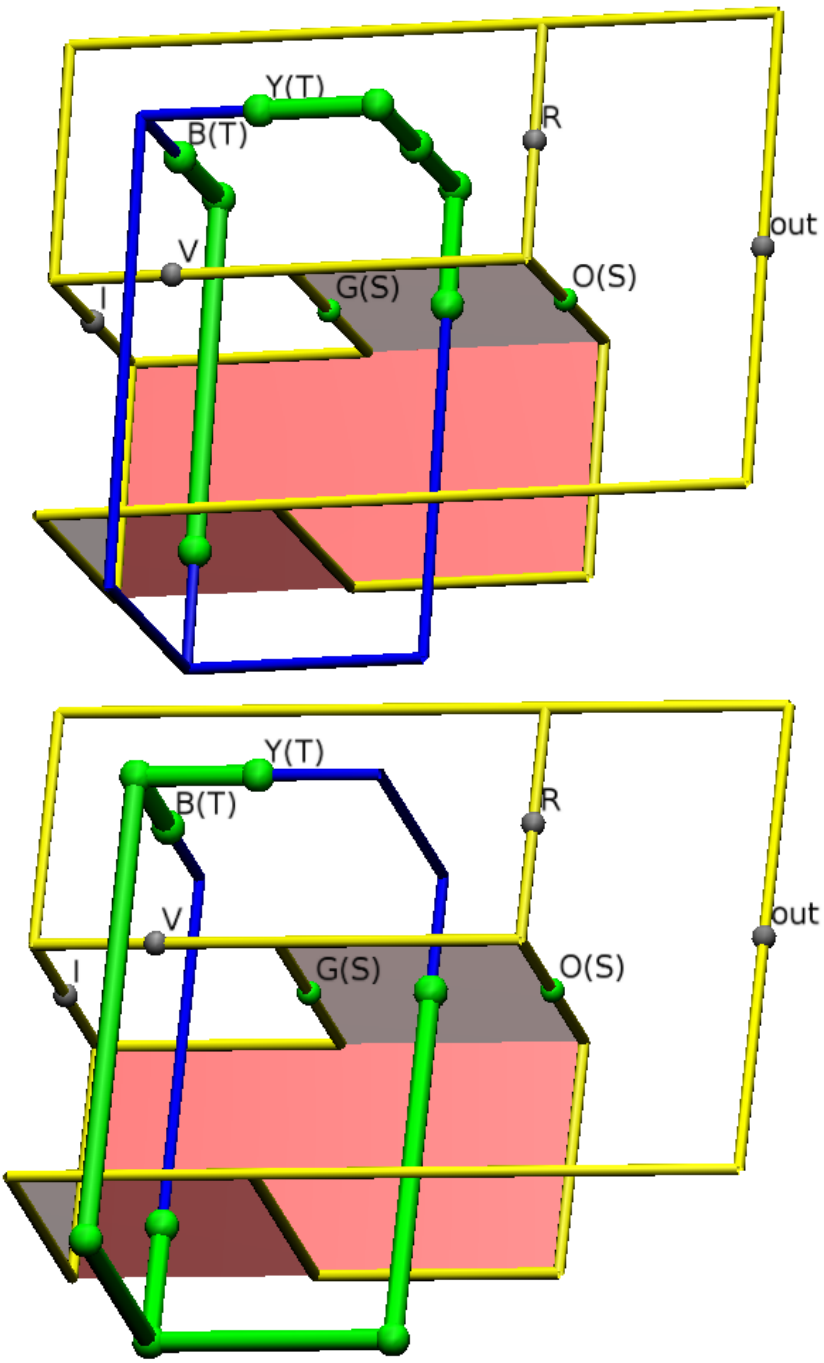


Figure 5.28: The correlation surface $Z_O Z_Y Z_G Z_B$.

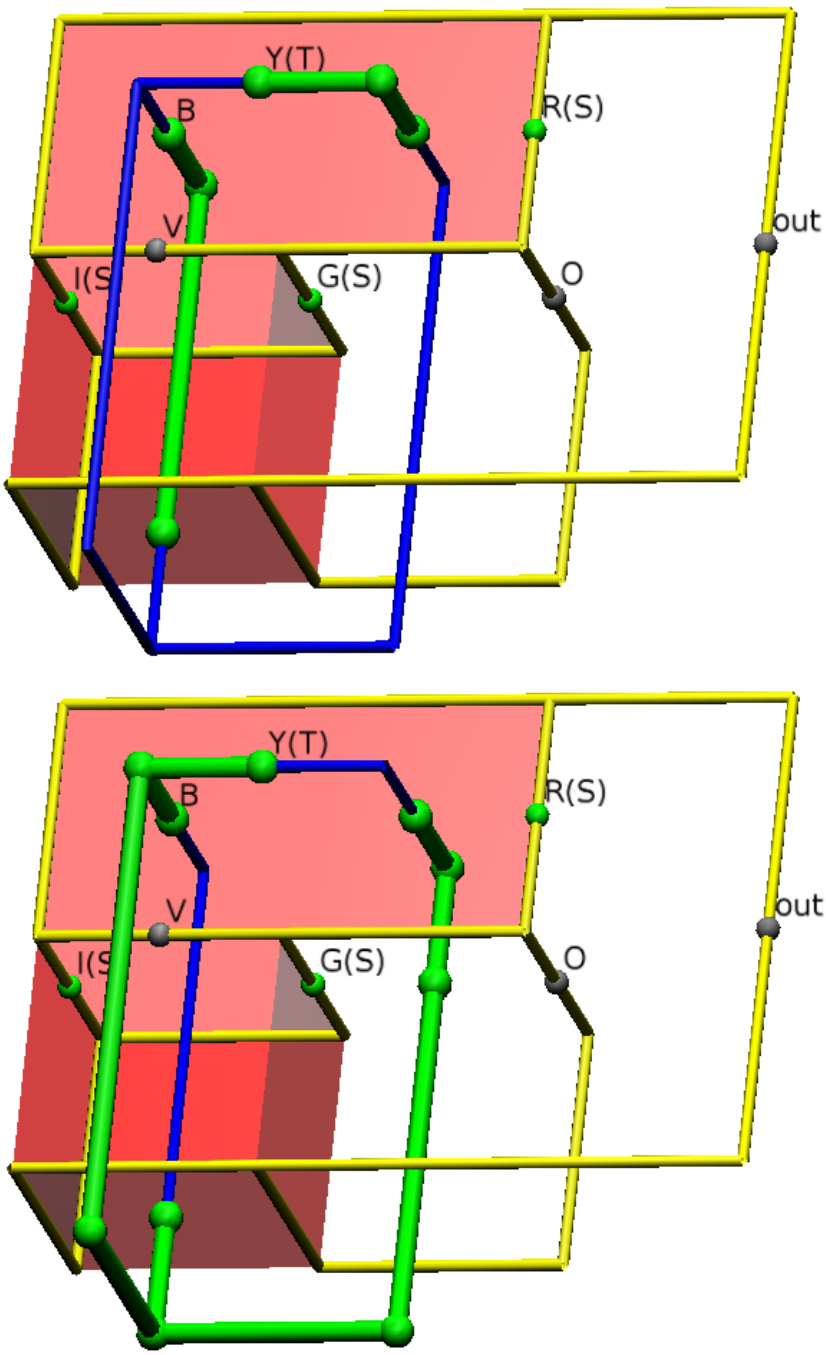


Figure 5.29: The correlation surface $Z_R Z_Y Z_G Z_I$.

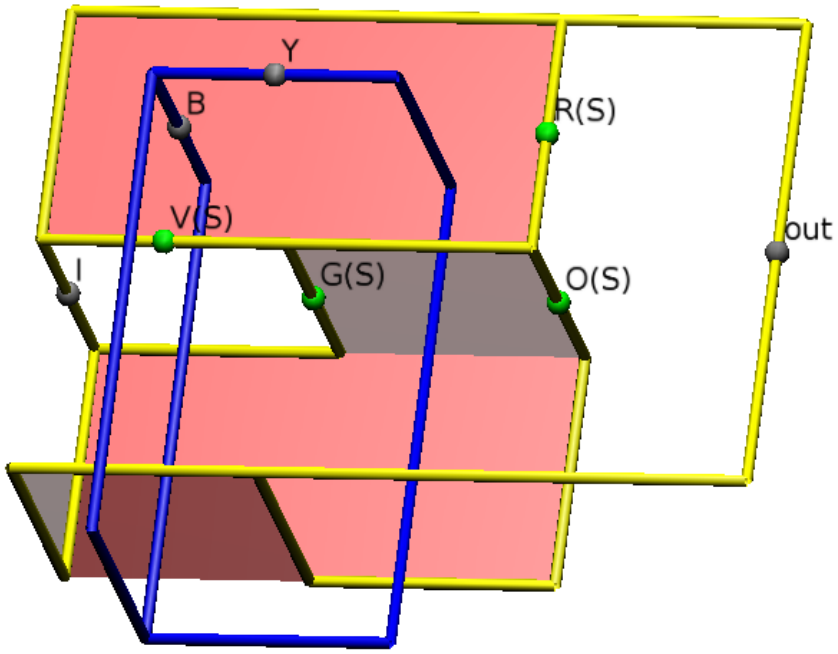


Figure 5.30: The correlation surface $Z_R Z_O Z_G Z_V$.

This section showed that the complexity of the validation methods is linear in the size of the stabiliser table, and the complexity of each correlation surface construction is polynomial. Note that only the stabiliser part of the circuit is considered, and the procedures do not imply that a quantum circuit is simulated in polynomial time, but that some of its properties are easily checked. Furthermore, the decomposition of a TQC circuit shows that there is an exponential number of measurement configurations.

Future work will investigate the automatic translation of arbitrary geometric descriptions into the quantum circuit formalism. The first steps towards this goal will focus on the automatic extraction of circuit properties from geometric descriptions.

5.5 Summary

The previous chapter introduced TQC and an extensive analysis of correlation surfaces allowed the Boolean representation to be formulated. This chapter augmented those concepts and introduced the *B*-notation to formalise the relationships between the elements of correlation surfaces. Its properties were investigated and illustrated from both a Boolean and a graph perspective, and this allowed the introduction of a systematic approach towards the validation of circuit identities.

Some simple circuit identities used as the basis of more complex identities supported, for the first time, a better formulation of inherent TQC circuit aspects. Existing circuit iden-

tities were proven using the *B*-notation, and the splitting of sheets was also introduced for the first time. This was subsequently employed to show that the Raussendorf cage circuit identity is structurally equivalent to splitting and tube rotations.

The chapter contributed to the TQC state of the art by introducing TQC circuit synthesis and formalising the validation procedure using correlation surfaces. The correctness of the synthesis method was shown using the introduced formalism. The presented results consist of simulation results and their visualisation, which allow an intuitive inspection.

Future work will concentrate on building an algebra of circuit identities to be used for complex automatic compaction and verification. This will enable large scale integration of TQC circuits and could illustrate the existence of practical quantum circuit design methods.

6 Conclusion

Reliable quantum circuits are building blocks of scalable quantum computers. The advent of quantum computing and the refinement of both quantum computing architectures and quantum error correcting codes motivated the investigation of novel automatic design methods devised specifically for reliable circuits. The presented results aggregate the findings of analysing quantum circuits from two perspectives: their probabilistic behaviour and their topological properties.

The applied design flow is similar to the one of classical circuits, and the main difference lies in the challenges posed by quantum circuits. Starting from circuit specification and ending at implementation, the synthesis, optimisation, verification and validation steps can be executed in a loop that feedbacks information from one step to a previous one. The target of the complete process is to have circuits that are cost efficient and correct. Thus, the central concept in the thesis' methodology was the circuit specification, which was defined, for the probabilistic approach, as a list of gates from an universal gate set, while for the topological approach the specification was similar to a truth table. The circuit specification was the point of reference by which a circuit under test was considered correct or not. Faults existing in the circuits were either missing gates from the gatelist specifications or topological operations that could have resulted in the truth table being negatively affected.

The first part of the thesis defined probabilistic circuits as a computational model where inputs are associated to outputs by some probability. The probabilities are either equal among all the outputs, as in the case of teleportation based fault-tolerant quantum circuit implementations, or distributed according to some properties of the circuit's state before measurement. The later situation is the case for stochastic and generic quantum circuits.

The similarity between stochastic and quantum circuits relies in the output probabilities being approximated by repeated executions of the same circuit. For the first time, this similarity was used for formulating the simulation of quantum circuits using stochastic computing and for introducing new testing and diagnosis methods of quantum circuits.

The approximate simulation of quantum circuits increased the applicability of stochastic computing, and showed the practicality of the proof-of-principle mechanism. Compared to classical circuits used for the simulation, it was possible to reduce the stochastic circuit's gate count by a constant factor. However, the gate count is dominated by the exponential overhead originating from the high dimensionality of a quantum computation's Hilbert space. The results indicated that future work will focus on more compact state representations that can be easily mapped to stochastic computing elements.

Quantum state and process tomography were replaced in this work by tomographic testing and diagnosis of quantum circuits. Tomography attempts to reconstruct a quantum state starting from the approximated output probabilities. The measurement of an unknown quantum state determines the observable output, and quantum computations are reversible until the outputs are read out. Due to the resemblences between a quantum circuit and a black box, tomography is the process of inferring the contents of the box. Once more, the high dimensionality of the Hilbert space is the source of the exponential computational complexity required by the tomographic approaches. The proposed validation methods circumvent this issue after assuming that the quantum circuit is expressed from a discrete set of quantum gates (the left-right modifier technique), and that the circuit expressed as a black box can be broken into multiple sub-boxes (the slicing mechanism). The practicality of the methods is illustrated by the simulation results, but the *curse of dimensionality* is still limiting the applicability of the methods for very large scale quantum computations.

The second part of the work presented the first systematic approaches towards the validation of topological quantum circuits. The topological properties of a circuit refer to invariants of a circuit's geometric description when mapped to a particular type of quantum error correcting codes. At first sight, the underlying code enables a straightforward representation of the computations, but the geometry becomes difficult to be interpreted when non-trivially deformed. Previous work recognised the need to reduce the implementation cost of a topological circuit by minimising the geometric description volume. However, no solutions have been known for an automatically correct circuit compactification, and no algorithmic formalism has existed for the validation of already compactified circuits.

The proposed methodology was introduced stepwise. First the geometric description was mapped to a set of architecture agnostic hardware commands. During this step, specific properties of the geometry were computed, being used in a second step to represent the circuit's geometry by an original notation. The notation enables a semi-formal verification method (see Section 1.2) based on a canonical representation abstracting the circuit layout. As the notation captures the state transformations performed by the circuit, this allows solv-

ing circuit validation by enumerating supported transformations. The direct effect was a method with a reduced computational complexity. A further use of the introduced notation was to prove the correctness of geometric description derivations. Finally, a cross-layer validation procedure was developed. The method has the advantage that it supports checking the properties of the underlying QECC.

6.1 Critique

Large scale quantum computing will be of industrial practicality when the number and relevance of applications outweigh the long term engineering challenges. There are almost no borders between the domains that generate the engineering challenges fo quantum computing, and a comprehensive understanding of a majority of the aspects connected to this field of research is necessary. This work resulted in an interdisciplinary approach.

The topic of reliable quantum circuits has still not been exhaustively explored, and there may be future developments that will change the directions of current research. There are opinions that question the correctness of the theoretical assumptions underlying the fault-tolerance in quantum computing[LB13,Ali13], and indicate that it could well happen that a large quantum computer will have practical capabilities equivalent to a classical analog machine[Ali13]. Moreover, there has also been argued that not all the properties of the topological codes can be achieved. The opposite position towards reliable quantum computing is more optimistic, and includes the view that, if a quantum computer were ever built, there would be a high probability to be a topological one[FG09].

This thesis tries to stay equidistant between the optimistic and pessimistic viewpoints, and the presented methods show that there are interesting research problems to be answered, even if quantum computing would be seriously flawed. The perspective that computations can be performed by braids is intriguing as it is a more visual abstraction compared to Boolean functions. It has the potential to also generate completely new computing architectures. There are therefore chances of extending the verification of topological quantum circuits to the abstract computations with knots.

The initial enthusiasm for quantum computing was reduced for a period because of the technical difficulties encountered, and the community started considering the problem of *quantum simulation*[Tra12,CZ12]. A quantum simulator (not to be confused with quantum circuit simulation software) would be a machine (hardware) that could imitate any quantum system[Fey82], and such a device would be as difficult to build as a quantum computer[CZ12]. However, a simulator would be a more specialised device, used for investigating current problems that are difficult to investigate using a classical computer. Instead of focusing on the complete description of the simulated system (computation), only some properties are analysed, and a quantum simulator would not be as exact as a quantum computer[CZ12]. A quantum simulator is expected to be more robust against imperfections than a quantum computer[CZ12]. The use of QECC would not be necessary, and if the arguments against error

correction were valid, a simulator would be an uncorrected quantum computer. The proposed design methods, especially the ones related to probabilistic circuits, would therefore be valid for the situation of simulators.

6.2 Future work

A number of open problems has to be solved for quantum computing to become a reality, and many of the problems are related to the design of reliable quantum circuits. For computations described using the quantum circuit formalism, it would be advantageous to have a canonical representation of fault-tolerant elements. Such a representation will have to be flexible in the sense that new insights regarding fault-tolerance should be easy to introduce. The representation would be similar to the canonical form of stabiliser circuits established in [AG04], where the circuit consists of and 11-round sequence of Hadamard, Phase and CNOT gates (H-C-P-C-P-C-H-P-C-P-C). Furthermore, the representation has to be architecture-agnostic in order to be relevant on the long-term, and must support only the CNOT gate and a reduced set of input/output initialisations/measurements in a comparable manner to how TQC is defined. The practicality of techniques operating on a canonical representation would consist in the standardisation of verification and possibly in the better understanding of circuit optimality. Future work will look into a more complete understanding of the TQC circuit geometry and possible circuit identities.

A more extensive comparison between analog computing (with its alike digital stochastic counterpart) and quantum computing is a further topic of future work. The resemblances as well as the differences between analog and quantum computation are to indicate some of the limits of quantum circuit simulation and verification. It is not expected that some kind of exponential reduction of complexity can be achieved by novel data structures or methods, but even linear speed-ups may make certain algorithms practical for relevant circuit instances. Also, comparing the stochastic simulation of quantum circuits with the proposed techniques of quantum simulation could spark a new interdisciplinary effort between computer engineers and physicists.

6.3 Summary

The present thesis focused on offering solutions to existing problems of quantum circuit design, and it is hoped that the proposed methods are useful as a future work basis. Synthesis, optimisation and verification of quantum circuits, in general, and of reliable circuits, in particular, is a vast subject, but the thesis offered a systematic perspective of these topics. The work demonstrated that verification is feasible for specific classes of general circuits, while for topological circuits efficient synthesis and verification were facilitated by an original notation abstracting circuit properties.

References

[Aar04] Scott Aaronson. Is quantum mechanics an island in theoryspace? *arXiv preprint quant-ph/0401062*, 2004.

[AB06] Simon Anders and Hans J Briegel. Fast simulation of stabilizer circuits using a graph-state representation. *Physical Review A*, 73(2):022334, 2006.

[Ada94] Colin C Adams. *The knot book*. American Mathematical Soc., 1994.

[AG04] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[AGU72] Alfred V Aho, Michael R Garey, and Jeffrey D Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[AH12] Armin Alaghi and John P Hayes. A spectral transform approach to stochastic circuits. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 315–321. IEEE, 2012.

[AJK05] Joseph B Altepeter, Evan R Jeffrey, and Paul G Kwiat. Photonic state tomography. *Advances in Atomic, Molecular, and Optical Physics*, 52:105–159, 2005.

[AL04] Panos Aliferis and Debbie W Leung. Computation by measurements: a unifying picture. *Physical Review A*, 70(6):062314, 2004.

[Ali13] Robert Alicki. Critique of fault-tolerant quantum information processing. *arXiv preprint arXiv:1310.8457*, 2013.

[ALRL04] Algirdas Avižienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

[Avi67] Algirdas Avižienis. Design of fault-tolerant computers. In *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, pages 733–743. ACM, 1967.

[BA00] Michael Bushnell and Vishwani D Agrawal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*, volume 17. Springer, 2000.

[Bac06] Dave Bacon. *CSE 599d Quantum Computing, Lecture Notes*. http://courses.cs.washington.edu/courses/cse599d/06wi/, 2006.

[BAP10] Jacob D Biamonte, Jeff S Allen, and Marek A Perkowski. Fault models for quantum mechanical switching networks. *Journal of Electronic Testing*, 26(5):499–511, 2010.

[BBC+93] Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical Review Letters*, 70(13):1895, 1993.

[BBC+95] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[BBD+09] Hans J Briegel, David E Browne, W Dür, Robert Raussendorf, and Martin Van den Nest. Measurement-based quantum computation. *Nature Physics*, 5(1):19–26, 2009.

[BC11] Shekhar Borkar and Andrew A Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[Ber08] Reinhold A Bertlmann. *Theoretical Physics T2 Quantum Mechanics, Course of Lectures*. http://homepage.univie.ac.at/Reinhold.Bertlmann/, 2008.

[BFK09] Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. Universal blind quantum computation. In *Foundations of Computer Science, 2009. FOCS'09. 50th Annual IEEE Symposium on*, pages 517–526. IEEE, 2009.

[BK05] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

[BKB+12] Stefanie Barz, Elham Kashefi, Anne Broadbent, Joseph F Fitzsimons, Anton Zeilinger, and Philip Walther. Demonstration of blind quantum computing. *Science*, 335(6066):303–308, 2012.

[CLN05] Andrew M Childs, Debbie W Leung, and Michael A Nielsen. Unified derivations of measurement-based schemes for quantum computation. *Physical Review A*, 71(3):032318, 2005.

[Col06] Graham P Collins. Computing with quantum knots. *Scientific American*, 294(4):56–63, 2006.

[CZ12] J Ignacio Cirac and Peter Zoller. Goals and opportunities in quantum simulation. *Nature Physics*, 8(4):264–266, 2012.

[D+00] David P DiVincenzo et al. The physical implementation of quantum computation. *arXiv preprint quant-ph/0002077*, 2000.

[Deu85] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[DFS+09] Simon J Devitt, Austin G Fowler, Ashley M Stephens, Andrew D Greentree, Lloyd CL Hollenberg, William J Munro, and Kae Nemoto. Architectural design for a topological cluster state quantum computer. *New Journal of Physics*, 11(8):083032, 2009.

[Dil98] David L Dill. What's between simulation and formal verification? In *Design Automation Conference*, pages 328–329. ACM, 1998.

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, 1992.

[DMN13] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[DN06] Christopher M Dawson and Michael A Nielsen. The Solovay-Kitaev algorithm. *Quantum Information & Computation*, 6(1):81–95, 2006.

[FD12] Austin G Fowler and Simon J Devitt. A bridge to lower overhead quantum computation. *arXiv preprint arXiv:1209.0510*, 2012.

[FDH04] Austin G Fowler, Simon J Devitt, and LLloyd CL Hollenberg. Implementation of Shor's algorithm on a linear nearest neighbour qubit array. *Quantum Inf. Comput.*, 4(quant-ph/0402196):237–251, 2004.

[Fey82] Richard P Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[FG09] Austin G Fowler and Kovid Goyal. Topological cluster state quantum computing. *Quantum Information & Computation*, 9(9):721–738, 2009.

[FHA98] Richard Phillips Feynman, JG Hey, and Robin W Allen. *Feynman lectures on computation*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[FMMC12] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[Fow12] Austin G Fowler. Time-optimal quantum computation. *arXiv preprint arXiv:1210.4626*, 2012.

[Gai69] Brian R Gaines. Stochastic computing systems. *Advances in Information Systems Science*, 2(2):37–172, 1969.

[GAJ06] Pallav Gupta, Abhinav Agrawal, and Niraj K Jha. An algorithm for synthesis of reversible logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 25(11):2317–2330, 2006.

[Got09] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum Information Science and Its Contributions to Mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2009.

[GWGH82] Solomon W Golomb, Lloyd R Welch, Richard M Goldstein, and Alfred W Hales. *Shift register sequences*, volume 78. Aegean Park Press Laguna Hills, CA, 1982.

[HEB04] Marc Hein, Jens Eisert, and Hans J Briegel. Multiparty entanglement in graph states. *Physical Review A*, 69(6):062311, 2004.

[HJB10] Marijn Heule, Matti Järvisalo, and Armin Biere. Clause elimination procedures for CNF formulas. In *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 357–371. Springer, 2010.

[Ins08] Project Management Institute, editor. *A Guide to the Project Management Body of Knowledge (PMBOK Guide): An American National Standard ANSI/PMI 99-001-2008*. Project Management Institute, Newtown Square, PA, 4 edition, 2008.

[JP05] Philippe Jorranda and Simon Perdrix. Unifying quantum computation with projective measurements only and one-way quantum computation. In *Proc. of SPIE Vol*, volume 5833, pages 44–51, 2005.

[JVMF+12] Cody N Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical Review X*, 2(3):031007, 2012.

[Kit03] Alexei Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[Lam08] William K Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall PTR, 2008.

[LB13] Daniel A Lidar and Todd A Brun. *Quantum error correction*. Cambridge University Press, 2013.

[LBP+10] Daniel Le Berre, Anne Parrain, et al. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[Lin08] Ming-Bo Lin. *Digital System Designs and Practices: Using Verilog HDL and FPGAs*. Wiley Publishing, 2008.

[LL11] Peng Li and David J Lilja. Using stochastic computing to implement digital image processing algorithms. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 154–161. IEEE, 2011.

[MM09] Marc Mezard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.

[Mor13] Evgeny Morozov. *To save everything, click here: Technology, solutionism, and the urge to fix problems that don't exist*. Penguin UK, 2013.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[NMSG11] Ali Naderi, Shie Mannor, Mohamad Sawan, and Warren J Gross. Delayed stochastic decoding of LDPC codes. *Signal Processing, IEEE Transactions on*, 59(11):5617–5626, 2011.

[PAPH11] Alexandru Paler, Armin Alaghi, Ilia Polian, and John P Hayes. Tomographic testing and validation of probabilistic circuits. In *European Test Symposium (ETS), 2011 16th IEEE*, pages 63–68. IEEE, 2011.

[PDNP12] Alexandru Paler, Simon Devitt, Kae Nemoto, and Ilia Polian. Synthesis of topological quantum circuits. In *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on*, pages 181–187. IEEE, 2012.

[PDNP14a] Alexandru Paler, Simon Devitt, Kae Nemoto, and Ilia Polian. Software-based Pauli tracking in fault-tolerant quantum circuits. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–4. IEEE, 2014.

[PDNP14b] Alexandru Paler, Simon J Devitt, Kae Nemoto, and Ilia Polian. Mapping of topological quantum circuits to physical hardware. *Scientific reports*, 4, 2014.

[PF13] Adam Paetznick and Austin G Fowler. Quantum circuit optimization by topological compaction in the surface code. *arXiv preprint arXiv:1304.2807*, 2013.

[PFBH05] Ilia Polian, Thomas Fiehn, Bernd Becker, and John P Hayes. A family of logical fault models for reversible circuits. In *Test Symposium, 2005. Proceedings. 14th Asian*, pages 422–427. IEEE, 2005.

[PFdM10] Alexandru Paler, Andreas Fischer, and Hermann de Meer. Demonstrating distributed virtual networks. In *Towards a Service-Based Internet*, pages 229–230. Springer, 2010.

[PHM04] Ketan N Patel, John P Hayes, and Igor L Markov. Fault testing for reversible circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 23(8):1220–1230, 2004.

[PKPH13] Alexandru Paler, Josef Kinseher, Ilia Polian, and John P Hayes. Approximate simulation of circuits with probabilistic behavior. In *Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on*, pages 95–100. IEEE, 2013.

[PPH12] Alexandru Paler, Ilia Polian, and John P Hayes. Detection and diagnosis of faulty quantum circuits. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 181–186. IEEE, 2012.

[Pre98a] John Preskill. *Lecture Notes For Physics: Quantum Information and Quantum Computation*. http://www.theory.caltech.edu/ preskill/ph219, 1998.

[Pre98b] John Preskill. Reliable quantum computers. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):385–410, 1998.

[QLR+11] Weikang Qian, Xin Li, Marc D Riedel, Kia Bazargan, and David J Lilja. An architecture for fault-tolerant computation with stochastic logic. *Computers, IEEE Transactions on*, 60(1):93–105, 2011.

[RBB03] Robert Raussendorf, Daniel E Browne, and Hans J Briegel. Measurement-based quantum computation on cluster states. *Physical review A*, 68(2):022312, 2003.

[RHG06] Robert Raussendorf, Jim Harrington, and Kovid Goyal. A fault-tolerant one-way quantum computer. *Annals of physics*, 321(9):2242–2270, 2006.

[RHG07] Robert Raussendorf, Jim Harrington, and Kovid Goyal. Topological fault-tolerance in cluster state quantum computation. *New Journal of Physics*, 9(6):199, 2007.

[SM13] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.

[Sob99] Elliott Sober. Testability. In *Proceedings and Addresses of the American Philosophical Association*, pages 47–76. JSTOR, 1999.

[SPMH03] Vivek V Shende, Aditya K Prasad, Igor L Markov, and John P Hayes. Synthesis of reversible logic circuits. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 22(6):710–722, 2003.

[Tar72] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

[Tra12] Andreas Trabesinger. Quantum simulation. *Nature Physics*, 8(4):263–263, 2012.

[TŘH13] Yong Siah Teo, Jaroslav Řeháček, and Zdeněk Hradil. Informationally incomplete quantum tomography. *Quantum Measurements and Quantum Metrology*, 1:57–83, 2013.

[VMH09] George F Viamontes, Igor L Markov, and John P Hayes. *Quantum circuit simulation*. Springer, 2009.

[VN56] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[VZO05] Milena Vratonjic, Bart R Zeydel, and Vojin G Oklobdzija. Low-and ultra low-power arithmetic units: design and comparison. In *Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on*, pages 249–252. IEEE, 2005.

[WD09] Robert Wille and Rolf Drechsler. BDD-based synthesis of reversible logic for large functions. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, pages 270–275. IEEE, 2009.

[WD10] Robert Wille and Rolf Drechsler. *Towards a design flow for reversible logic*. Springer, 2010.

[WG98] Colin P Williams and Alexander G Gray. Automated design of quantum circuits. In *Selected papers from the First NASA International Conference on Quantum Computing and Quantum Communications*, pages 113–125. Springer-Verlag, 1998.

[WGB+09] Steffen Weißmann, Charles Gunn, Peter Brinkmann, Tim Hoffmann, and Ulrich Pinkall. jReality: a java library for real-time interactive 3D graphics and audio. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 927–928. ACM, 2009.

[WGMD09] Robert Wille, Daniel Große, D Michael Miller, and Rolf Drechsler. Equivalence checking of reversible circuits. In *Multiple-Valued Logic, 2009. ISMVL'09. 39th International Symposium on*, pages 324–330. IEEE, 2009.

[ZL79] James F Ziegler and WA Lanford. Effect of cosmic rays on computer memories. *Science*, 206(4420):776–788, 1979.

[ZL08] Da Zhang and Hui Li. A stochastic-based FPGA controller for an induction motor drive with integrated neural network algorithms. *Industrial Electronics, IEEE Transactions on*, 55(2):551–561, 2008.

[ZLC00] Xinlan Zhou, Debbie W Leung, and Isaac L Chuang. Methodology for quantum logic gate construction. *Physical Review A*, 62(5):052316, 2000.

The above illustration is by Draga Paler. This thesis was typeset using LaTeX. The body text is set in 12 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. A template that can be used to format a PhD thesis with this look and feel has been released under the permissive MIT (XII) license at github.com/suchow/Dissertate.