Universiteit Leiden

A Model Counting Approach to Depth-Optimal Quantum Circuit Synthesis

Master Computer Science

Name:	Dekel Zak
Student ID:	s4016599
Date:	18/07/2025
Specialisation:	Foundations of Computing
1st supervisor:	Dr. Alfons Laarman
2nd supervisor:	Dr. Jean-Marie Lagniez
daily supervisor:	Jingyi Mei, MSc

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Quantum circuit synthesis is fundamental in compiling quantum algorithms into executable programs on real-world quantum hardware. It involves translating a high-level specification into a sequence of quantum gates while respecting hardware constraints such as native gate sets and qubit connectivity. The resulting circuit is optimized for specific metrics (e.g., depth or gate count) and targets either exact synthesis (functional equivalence) or approximate synthesis within a defined error threshold.

In this thesis, we introduce a novel application of Maximum Weighted Model Counting (MWMC) for quantum circuit synthesis. While MWMC is a well-established technique in symbolic reasoning and combinatorial optimization, its potential in quantum compilation has remained largely untapped. Prior work has shown the success of using Weighted Model Counting (WMC) for quantum circuit simulation and equivalence checking. We build on that and propose to leverage MWMC for constructive synthesis. Our central idea is to encode quantum synthesis tasks — both exact and approximate — as MWMC problems, where the goal is to find the optimal circuit configuration of a given depth that maximizes equivalence with the input specification. By incrementally increasing the allowed circuit depth, our approach also yields depth-optimal synthesis results.

We present an open-source implementation of our approach, `Quokka#`, a unified encoding framework that represents quantum computations using weighted Boolean formulas, supporting simulation, equivalence checking, and synthesis. We have developed two complementary encoding bases: one based on the Pauli basis, leveraging algebraic properties of the Clifford group, allowing for compact encoding variations; and another in the computational basis, which requires fewer variables at the price of needing complex-valued weights. Lastly, to support `Quokka#`, we extended existing model counters (`GPMC`, `d4Max`) to handle complex weights.

Extensive experiments compare the proposed approach against other state-of-the-art tools. Results demonstrate that our method achieves competitive performance and offers formal guarantees on circuit equivalence and approximation fidelity. This work establishes a novel reduction of the quantum circuit synthesis problem to model counting, offering a foundation for future research to enhance both the scalability and generality of the method. Moreover, it demonstrates a new application of MWMC with complex weights, motivating further development of solvers to better handle such expressive symbolic encodings.

Acknowledgements

I want to thank everyone who supported me throughout the course of this thesis.

First, I am especially grateful to Jingyi Mei, my daily supervisor. Her patience, encouragement, and thoughtful guidance were essential at every stage of this project. She was always willing to take the time to answer questions, discuss ideas, and help me move forward when I encountered challenges. I truly appreciate the clarity and calm she brought to the process, and the genuine support she offered throughout.

I would also like to thank Dr. Alfons Laarman, whose guidance helped shape the direction of the research. His insights, feedback, and timely interventions were invaluable, particularly when key decisions needed to be made or when we found ourselves at difficult points in the work.

Towards the end of the project, Dr. Jean-Marie Lagniez joined as my second supervisor. I appreciate his involvement and look forward to his role in the defence.

Finally, I am grateful to my friends and family for their support and encouragement during this journey. Your belief in me made all the difference.

Disclaimer on the Use of AI Tools

Writing has always been the most difficult part of academic work for me. While all the ideas, reasoning, and conclusions in this thesis are entirely my own, I used AI tools — specifically ChatGPT and Grammarly — to help me communicate those ideas more clearly and professionally.

This support was mostly in text-centered sections, such as the introduction and conclusion, where expressing ideas in natural language is key. I did not use AI tools for technical content like the preliminaries, formal definitions, proofs, or mathematical reasoning.

In practice, I often begin by writing down core ideas in a rough or list format (much like how I'm writing this disclaimer) and then use ChatGPT to help turn those into well-structured paragraphs. This was usually an iterative process, with multiple rounds of editing and refinement to ensure the result matched what I intended to say. I also used Grammarly and, at times, ChatGPT, to review and improve the grammar and style of the text I had written myself. Crucially, I always carefully checked the correctness of ChatGPT's suggestions before including them. I reviewed, fact-checked, and edited every suggestion from these tools before inclusion.

As a rule of thumb, I only use AI for tasks I consider NP-complete: problems where finding a good solution can be hard, but verifying one is easy. I see large language models as decent heuristics — they're good at suggesting plausible candidates, even if they can't guarantee correctness. Since LLMs tend to hallucinate with confidence, I limited their use to areas where I could easily check the output myself. In short: no AI-generated theorems, just AI-assisted phrasing.

Contents

Chapter 1

Introduction

1.1 Context and Motivation

The primary goal in quantum computing today is to demonstrate *quantum supremacy*—that is, to solve a computational problem that is infeasible for any classical computer to solve within a reasonable amount of time. [93, 94]. Achieving this milestone is widely seen as the first step toward realizing quantum advantage for meaningful, real-world applications of societal importance. These include secure quantum communication and cryptography [15], efficient simulation of quantum systems for drug discovery and materials science [29, 80], financial modeling [88], and large-scale combinatorial optimization [42, 53].

Encouraging progress has been made. For instance, Arute et al. [10] demonstrated that a noisy quantum processor could solve a sampling problem considered intractable for classical supercomputers. IBM responded with alternative results and interpretations [90]. Still, many researchers believe that true quantum advantage requires *quantum error correction (QEC)* [105], which enables the realization of ideal quantum computation as formalized in quantum Turing machines [17] or families of quantum circuits [2].

QEC alone, however, is only one component of a larger vision: full-scale *fault-tolerant quantum computing*. Fault tolerance builds on QEC to ensure that quantum operations — gates, measurements, and state preparations — can be performed reliably even in the presence of errors, enabling robust execution of large and complex quantum algorithms. With fault-tolerant quantum hardware, the full arsenal of quantum algorithms becomes practically accessible.

Nonetheless, significant software challenges remain. One of the challenges is compiling high-level quantum algorithms into circuits that are compatible with the constraints of real hardware. Different quantum architectures — such as photonic systems [44], superconducting qubits [67], and neutral atoms [55] — introduce platform-specific constraints including qubit connectivity, native gate sets, limited gate fidelities, and limited coherence times [75, 113]. Effective circuit compilation must therefore bridge the gap between abstract algorithm design and the realities of physical hardware.

Given these constraints, it is essential to optimize not only for correctness, but also for circuit *width* (number of qubits used) and especially *depth* (sequential gate layers), which directly affects the probability of decoherence during execution [6, 37]. Those considerations become particularly critical on early quantum hardware, where resources are scarce and noise levels are

high. As a result, depth-optimal synthesis — synthesis that guarantees a minimal-depth circuit — can play an important role in improving the practicality of quantum computations on early devices.

To meet these optimization requirements, we can draw inspiration from classical formal methods that have long been applied to synthesis and verification tasks. Classical program synthesis methods exhibit doubly exponential worst-case complexity in particular cases [71], motivating the need for good heuristics. For quantum circuit synthesis, some recent tools use heuristics [79] and do not guarantee optimality. Others, that do guarantee optimality, use algebraic decomposition [89] or meet-in-the-middle strategies [7], but result in limited scalability: 4 qubits with T-depth 3 (i.e., three layers containing T gates) and depth 4 respectively. In parallel, symbolic methods from AI — successful across many domains — have also shown strong performance in quantum circuit analysis. For example, techniques based on decision diagrams [116, 25, 107], tree automata [34], and Boolean satisfiability (SAT) [103].

SAT solving is one of the central problems in computer science, and although NP-complete in general, modern SAT solvers have seen remarkable advances over the past two decades [20]. These advances are not only theoretical but also practical: SAT solvers now routinely handle real-world instances with millions of variables, thanks to innovations in conflict-driven clause learning [77], heuristics [74], preprocessing [21], and parallelization [52]. One striking demonstration of this progress is the so-called *Time Leap Challenge* [43], in which benchmark instances from SAT competitions in the early 2000s can now be solved orders of magnitude faster, sometimes within milliseconds, using modern solvers. This kind of exponential improvement underlines the value of SAT-based methods in synthesis workflows, especially when paired with domain-specific encodings for quantum circuits.

Building on this foundation, recent research has highlighted the utility of *weighted model counting* (WMC) as a powerful extension of SAT for quantum circuit analysis. Unlike SAT, which asks whether a satisfying assignment exists, model counting considers the number of such assignments, potentially weighted, making it applicable to probabilistic and quantum domains. Recent applications of WMC have demonstrated promising results in quantum circuit simulation [81] and exact equivalence checking [82], tackling problems that are #P-complete [41]. These approaches often leverage off-the-shelf solvers and carefully engineered encodings to achieve scalable performance.

Encouraged by this progress, we explore whether WMC can also be leveraged for the quantum circuit *synthesis* problem. While exact synthesis is already hard, universal quantum circuit synthesis is even more challenging, as it inherently involves approximation: no finite gate set can exactly generate all unitary operators. This brings the problem into the realm of QMA, the quantum analogue of NP, since approximate equivalence checking is known to be QMA-complete [58]. Fortunately, the Solovay-Kitaev theorem [39] guarantees that any target unitary can be approximated to arbitrary precision using circuits of polylogarithmic depth in $1/\epsilon$, providing a theoretical foundation for practical synthesis via approximation.

1.2 Problem Statement

Quantum circuit synthesis translates a desired quantum specification — defined by a unitary matrix, high-level algorithm, or behavioral description — into a low-level quantum circuit that

can be executed on a physical device. The synthesis process can take different forms depending on the task and the constraints involved.

Exact synthesis aims to generate a circuit that implements a given unitary exactly (often over a finite gate set). Approximate synthesis produces circuits that approximate the target operation within a specified error tolerance ϵ. Approximate synthesis is necessary when the target operation cannot be exactly (or within reasonable depth) expressed over the discrete gate set, for instance, when implementing arbitrary single-qubit rotations [69] or irrational angles. Foundational results like the Solovay-Kitaev theorem [39] ensure that approximate synthesis is always possible with polylogarithmic overhead in gate depth, assuming access to a universal gate set. However, this guarantee is largely asymptotic and does not ensure efficiency in practice. In real-world settings, synthesis must balance accuracy, resource usage, and hardware compatibility, especially given the severe limitations of current quantum hardware.

Optimization-aware synthesis focuses on producing circuits that minimize certain cost metrics, such as depth, gate count, or T-count (the number of T gates, which is especially important for fault-tolerant quantum computing). These tasks are often further constrained by the available gate set and whether ancilla qubits are allowed. Depending on the setting, synthesis may also involve mapping to hardware topologies, decomposing unitaries into primitive gates, or restructuring circuits to meet platform-specific constraints.

Quantum circuit synthesis tools that provide optimality guarantees remain limited in scope, with current methods typically unable to scale beyond 4 qubits and circuits of 6 layers or T-depth 3. While WMC has succeeded in circuit simulation and exact equivalence checking, it has yet to be leveraged for synthesis, despite being a general-purpose tool designed for propositional reasoning and combinatorial optimization.

This thesis addresses this gap: Can weighted model counting techniques be extended beyond simulation and equivalence checking to support synthesis, particularly depth-optimal and approximate synthesis for universal gate sets?

1.3 Scope and Objectives

The overarching goal of this work is to explore the feasibility and utility of model counting for quantum circuit synthesis. We phrase the guiding question as follows:

Main Research Question: *Can quantum circuit synthesis be effectively achieved using SAT-based techniques?*

To address this question, we first examine how model counting can be leveraged to perform quantum circuit synthesis. Since synthesis inherently depends on the ability to check circuit equivalence, we build on recent work using weighted model counting for equivalence checking [82], assessing how these techniques must be adapted or extended to support synthesis. In particular, we consider how to support not only exact, but also approximate synthesis, where the goal is to generate circuits that implement a target operation within a specified error tolerance. This introduces new challenges in defining suitable notions of approximate equivalence that are amenable to SAT-based encoding. In addition to ensuring functional correctness, we also aim to optimize the circuit representation, with a specific focus on achieving depth-optimal

synthesis. This allows us to generate circuits that are not only correct (or approximately correct), but also as shallow as possible under given constraints. Finally, we seek to develop a synthesis method that is efficient and scalable, enabling the solution of increasingly complex circuits and more challenging synthesis instances as the problem size grows.

Another key component of this effort is selecting an appropriate encoding basis. Each basis introduces trade-offs between expressiveness, encoding complexity, and solver performance. We investigate two distinct bases for encoding quantum computations: the Pauli basis, which represents quantum operations through the Pauli decomposition of density matrices and has a particularly efficient encoding variant for exact equivalence checking; and the computational basis, which encodes quantum states as vectors and is more intuitive and concise, allowing for better scalability. By evaluating both approaches, we aim to clarify their respective strengths and determine their suitability for exact and approximate synthesis scenarios.

1.4 Contributions

This thesis makes the following contributions to the field of quantum circuit synthesis and SAT-based reasoning:

- **One-Shot Equivalence Checking:** We build on the encoding of [82], modifying it to enable equivalence checking with only a single call to a WMC solver. This one-shot variation is essential for integration with synthesis workflows, where repeated solver calls can not be supported. The various Equivalence Checking encodings are presented in Theorem 4.1 in Chapter 4.

- **Fidelity-Based Approximate Equivalence Checking:** We further extend the equivalence checking encoding to support approximate reasoning by computing the fidelity between quantum circuits — a quantitative measure of how similar their outputs are. This allows us to go beyond exact synthesis and reason about approximate synthesis within a controlled error tolerance. Details are provided in Chapter 5.

- **SAT-Based Synthesis Encoding:** Building on the equivalence checking framework described above, we propose a novel method for reducing quantum circuit synthesis to a model counting problem. The key idea is to generalize the equivalence checking encoding to allow not just fixed circuits, but entire families of circuits as candidates. As shown in Figure 1.1, we encode a parameterized synthesis circuit of depth d, consisting of layers, each capable of representing all possible gate applications at that step. Each layer is controlled by a set of gate selection variables, which determine which gates are applied. This allows us to encode all candidate circuits within a single Boolean formula. We prepend the inverse of the specification, and impose equivalence constraints on the resulting composite transformation to evaluate correctness. These constraints are designed so that the weight reflects the similarity between the synthesized circuit and the specification. The full encoding is then solved using Maximum Weighted Model Counting (MWMC), which seeks an assignment to the gate selection variables that satisfies the constraints while maximizing the total assigned weight. This formulation

supports both exact and approximate synthesis using SAT-based solvers. Details of the encoding and its variations are presented in Chapter 4 and Chapter 5.

- **Depth-Optimal Synthesis:** Our synthesis approach achieves depth optimality by querying the SAT-based solver for increasing circuit depths. This strategy guarantees minimal-depth circuits by systematically exploring and ruling out shallower candidates. We cover this in Chapter 4.

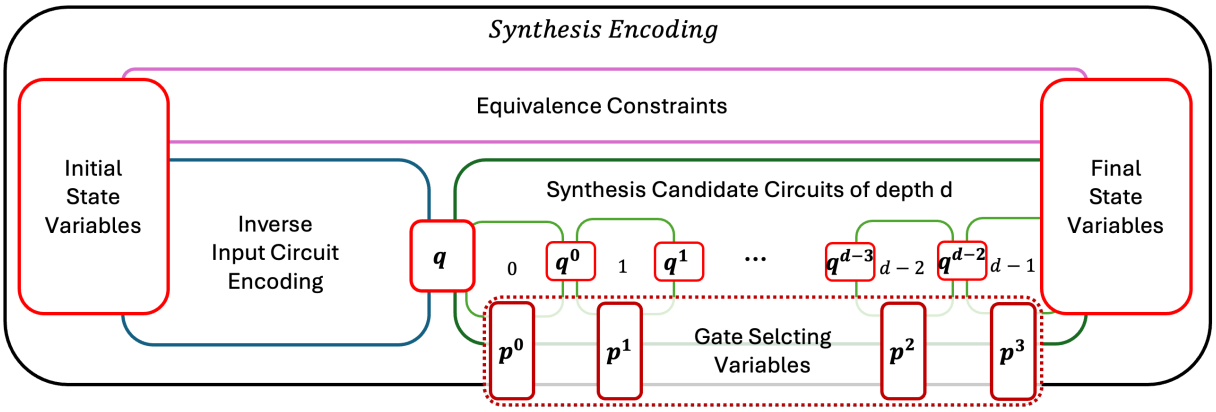


Figure 1.1: A high-level illustration of the structure of the synthesis encoding. A parameterized synthesis circuit of depth d is encoded using gate selection variables, allowing all candidate circuits of that depth to be represented within a single Boolean formula. The inverse of the specification circuit is prepended to enable equivalence checking. Constraints are imposed on the combined transformation to ensure functional correctness (either exact or approximate), and the encoding is solved via Maximum Weighted Model Counting (MWMC), which searches for the highest-weight satisfying assignment corresponding to a valid synthesized circuit.

- **Alternative Encoding Bases for Quantum Computation:** We explore two distinct encoding bases for representing quantum computation: the Pauli basis and the computational basis. The Pauli basis, used in prior work [81, 82], enables efficient encoding for exact equivalence checking via density matrix decomposition. However, it is not as well-suited to approximate reasoning. To address this, we introduce a new encoding scheme based on the computational basis, which directly represents quantum states as vectors in Hilbert space. This basis enables a more intuitive and compact encoding that reduces the number of variables and constraints, leading to improved scalability and solver performance on more complex benchmarks, and naturally supports approximate reasoning. This is explained in detail in Chapter 6, and the different encodings are compared at length in Chapter 7. We display a high-level comparison in Table 1.2.

- **Implementation of `Quokka#`, a SAT-Based Quantum Computing Tool:** We realize the proposed synthesis method in a functional prototype, `Quokka#`, an open-source tool integrating a full pipeline for depth-optimal exact and approximate synthesis. In addition to synthesis, `Quokka#` supports quantum circuit simulation as well as both exact and approximate equivalence checking. These functionalities share much of their underlying structure, as they all leverage the same encoding bases and implement the corresponding

Basis	Pauli	computational
State Encoding	$2n$ variables	n variables
Weights	real values	complex values
Approx. Equivalence Checking	$2n$ initial states	2^n initial states
Exact Equivalence Checking	4^n initial states	2^n initial states

Table 1.2: A high-level comparison of the encoding bases.

functionalities, which build upon each other (for instance, synthesis incorporates equivalence checking within its pipeline). By unifying these capabilities within a single tool and shared codebase, `Quokka#` provides a coherent framework for quantum circuit reasoning. To support this, we extend existing SAT-based solvers to handle complex weights required by parts of the encoding. Chapter 8 outlines the tool's functionalities and how to access and use it.

Based on these contributions, a peer-reviewed paper has been accepted to the 31st International Conference on Principles and Practice of Constraint Programming (CP 2025) [124]. This paper primarily presents the theoretical synthesis approach developed in this thesis, along with the corresponding model encodings and evaluation results. A second paper, submitted to the 37th International Conference on Computer Aided Verification (CAV 2025) as a tool paper, showcases `Quokka#`. This paper focused on the implementation of the complete synthesis workflow and, for the first time, integrated several novel SAT-based encodings for quantum reasoning into a working tool [81, 82]. However, despite acceptance of the written paper, the submission was ultimately rejected during the artifact evaluation phase due to missing components in the supplementary materials.

1.5 Overview of the Thesis

The structure of this thesis follows a systematic development from fundamental concepts to novel contributions and empirical evaluation.

The thesis begins in Chapter 2 by providing the necessary mathematical background on quantum computing, including formal definitions of circuit simulation and fidelity as a measure of circuit closeness. It then introduces SAT-based tools such as Model Counting (MC), Weighted Model Counting (WMC), and Maximum Weighted Model Counting (MWMC). The chapter concludes with an overview of the encodings presented in prior quantum circuit encoding methods from [81, 82].

In Chapter 3, we formally define the central problems addressed in this thesis: quantum circuit equivalence checking and synthesis, both in exact and approximate forms. These definitions lay the groundwork for the development of new encoding techniques in the following chapters.

Chapter 4 presents our first major technical contribution: a detailed encoding method for

exact synthesis using MWMC. This chapter includes rigorous correctness proofs and explains how circuit structure and gate selection can be encoded within a SAT-based framework for synthesis. Building on that, Chapter 5 extends these results to show how one of the encodings can be adapted to perform approximate synthesis, leveraging the fidelity metric introduced earlier.

In Chapter 6, we present an alternative encoding based on the computational basis representation of quantum computation. This approach reduces encoding size in several cases and unifies exact synthesis, approximate synthesis, and equivalence checking within a single framework.

Chapter 7 provides a detailed comparative analysis of the two quantum encoding bases — Pauli basis and the computational basis — as defined throughout the preceding chapters. It systematically examines their structural and representational differences and evaluates their asymptotic encoding size across all core functionalities: simulation, equivalence checking, and synthesis. The chapter offers concrete guidance on choosing the most suitable encoding strategy based on the problem.

In Chapter 8 we introduce `Quokka#`, the implementation of our methods as an open-source tool that integrates all the different functionalities for quantum circuits: simulation, as well as both exact and approximate equivalence checking and synthesis. This chapter describes how to obtain and use the tool, offers usage examples, and provides an overview of the codebase.

In Chapter 9, we situate our approach within the broader research landscape, reviewing related work in quantum compilation, SAT-based reasoning, and synthesis, and positioning our contributions in relation to the current state of the art.

Chapter 10 evaluates our techniques empirically. Using `Quokka#`, we benchmark the various encodings across multiple tasks and compare their performance against existing state-of-the-art tools for simulation, equivalence checking, and synthesis.

Finally, Chapter 11 summarizes the contributions of the thesis, reflects on the implications of our results, and outlines promising directions for future work.

Together, these chapters provide a comprehensive journey from fundamental concepts to novel synthesis methods, their implementation, evaluation, and positioning within the broader quantum computing landscape.

Chapter 2

Preliminaries

2.1 Quantum Computing

Qubits. Similar to bits in classical computing, a quantum computer operates on quantum bits, namely *qubits*. A bit is either 0 or 1, while a qubit can be in states $|0\rangle$, $|1\rangle$ or a *superposition* of both, $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ such that $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Here '$|\rangle$' is the *Dirac notation*, representing a unit column vector, i.e., $|\psi\rangle = [\alpha, \beta]^{\mathrm{T}}$, while $\langle\psi|$ denotes the complex conjugate and transpose of $|\psi\rangle$, that is a row vector: $\langle\psi| = |\psi\rangle^{\dagger} = [\alpha^*, \beta^*]$.

Quantum States. The quantum states describe the state of the qubits in the system. For an n-qubit system, the *computational basis states* are $|\boldsymbol{b}\rangle$ for each bitstring $\boldsymbol{b} \in \{0,1\}^n$, a vector of dimension 2^n with all entries set to 0 except at index $\boldsymbol{b}$, which is set to 1. A general quantum state is a superposition of the computational basis states, defined as a normalized vector $|\psi\rangle = \sum_{b\in\{0,1\}^n} \alpha_b\,|b\rangle = [\alpha_{0..0}, ..., \alpha_{1...1}]^{\mathrm{T}}$ ($\forall b \in \{0,1\}^n : \alpha_b \in \mathbb{C}$ and $\sum_{b\in\{0,1\}^n} |\alpha_b|^2 = 1$). Alternatively, a quantum state $|\psi\rangle$ can be represented by its *density operator* $|\psi\rangle\langle\psi|$. A density operator is a Hermitian positive semi-definite operator with unit trace.

Inner Product. The inner product between two states $|\psi\rangle = [\alpha_{0...0}, ..., \alpha_{1...1}]^{\mathrm{T}}$ and $|\psi'\rangle = [\beta_{0...0}, ..., \beta_{1...1}]^{\mathrm{T}}$ is denoted as $\langle\psi'|\psi\rangle$ and is defined as the vector product:

$$\langle\psi'|\psi\rangle \triangleq [\beta^*_{0...0}, ..., \beta^*_{1...1}] \cdot \begin{bmatrix} \alpha_{0...0} \\ \vdots \\ \alpha_{1...1} \end{bmatrix} = \sum_{b\in\{0,1\}^2} \beta^*_b \cdot \alpha_b$$

Note, that for any state $|\psi\rangle = [\alpha_{0...0}, ..., \alpha_{1...1}]^{\mathrm{T}}$, it holds that $\langle\psi|\psi\rangle = 1$ since state vectors are normalized. Additionally, we have $\langle b|\psi\rangle = \alpha_b$ for all $b \in \{0,1\}^2$.

In this work, we fix n to be the number of qubits, and write $[m]$ for the set $\{0, \ldots, m-1\}$.

Example 2.1. The two-qubit state $|01\rangle = [0,1,0,0]^{\mathrm{T}}$ has the density operator:

$$|01\rangle\langle 01| = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \cdot [0,1,0,0] = \begin{bmatrix} 0&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix}. \qquad \triangle$$

Kronecker Product. To combine states or operators of different subsets of qubits, we use the tensor (or Kronecker) product, which is defined as follows. Given $r_A \times c_A$ matrix A and $r_B \times c_B$ matrix B, the $r_A r_B \times c_A c_B$ matrix $A \otimes B$ is

$$A \otimes B = \begin{bmatrix} A_{00}B & A_{01}B & \dots & A_{0c_A}B \\ A_{10}B & A_{11}B & \dots & A_{1c_A}B \\ \vdots & \vdots & \ddots & \vdots \\ A_{r_A0}B & A_{r_A1}B & \dots & A_{r_Ac_A}B \end{bmatrix}.$$

For example, a two-qubit computational basis state can be expressed as the Kronecker product of the individual qubit states, e.g., $|01\rangle = |0\rangle \otimes |1\rangle$.

Matrix Trace. The *trace* of a square matrix A, denoted $\mathrm{Tr}(A)$, is defined as the sum of its diagonal entries, i.e., $\mathrm{Tr}(A) = \sum_i A_{ii}$. It follows from the definition of the trace that for two matrices A, B we have $\mathrm{Tr}(A \otimes B) = \mathrm{Tr}(A) \cdot \mathrm{Tr}(B)$.

Quantum Gates. Operations on quantum states are given by *quantum gates*. For an n-qubit quantum system, a quantum gate G is a function described by an $2^n \times 2^n$ *unitary matrix* G, i.e., with the property that $GG^\dagger = G^\dagger G = I^{\otimes n}$, where $I^{\otimes n}$ is the identity matrix of dimensions $2^n \times 2^n$. Applying a quantum gate G to a state $|\psi\rangle$ is computed as $G \cdot |\psi\rangle$, and if the quantum state is represented by a density matrix $|\psi\rangle\langle\psi|$, then the density matrix after applying G is given by *conjugation* of $|\psi\rangle\langle\psi|$, i.e., $G\,|\psi\rangle\langle\psi|\,G^\dagger$. Gates can be combined via the Kronecker product to be applied in parallel; for instance, $X \otimes Z$ applies X to the first qubit and Z to the second. In most cases, we limit quantum operations to a specific set of gates, which we call a *gate set*, denoted by $\mathcal{G}$.

Quantum Gate Layers. A *quantum gate layer* (or simply a *layer*) is a set of quantum gates applied simultaneously to specific qubits, such that each qubit is acted on by at most one non-identity gate. If no gate is applied to a qubit in the layer, the identity operator I is implicitly used. As a result, all gates within a layer are mutually parallel and can be executed in the same time step. We will denote layers by D. To indicate that a single-qubit gate G is applied to qubit $i \in [n]$, we write G_i. Similarly, $G_{i,j}$ denotes a two-qubit gate applied to qubits i and j, and so on. The unitary form of G_i can be constructed by $G_i = I^{\otimes i} \otimes G \otimes I^{\otimes n-i-1}$. For two-qubit gates, we do not explicitly construct the full matrix form in most cases. Instead, we assume that $G_{i,j}$ denotes a unitary that acts as G on qubits i and j, and as the identity on all other qubits.

Since the gates in a layer act on disjoint sets of qubits, their unitaries commute and can be safely composed via a product (in any order). Thus, the overall unitary U_D of a layer D is given by $U_D = \prod_{U \in D} U$ where the product denotes a composition of parallel, non-overlapping gate operations.

Example 2.2. Let us apply the Hadamard gate $H = \frac{1}{\sqrt{2}} \cdot \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$ on the first qubit in the

state $|01\rangle$, i.e., apply the layer $D = \{H_0\}$ represented by the unitary

$$U_D = (H \otimes I)$$
$$= \frac{1}{\sqrt{2}} \cdot \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \cdot \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}.$$

We compute the new state by doing the vector-matrix multiplication as follows:

$$U_D |01\rangle = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot (|01\rangle + |11\rangle).$$

If we do the computation on the density operator of the same state, we will get the corresponding density operator of the new state:

$$U_D |01\rangle \langle 01| U_D^\dagger = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$
$$= \frac{1}{2} \cdot \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix},$$

which is exactly $\frac{1}{\sqrt{2}} \cdot (|01\rangle + |11\rangle) \cdot \frac{1}{\sqrt{2}} \cdot (\langle 01| + \langle 11|)$. △

Pauli Basis. An important set of quantum gates is *Pauli gates*:

$$X \triangleq \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad , \quad Y \triangleq \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad , \quad \text{and} \quad Z \triangleq \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

Let $\mathcal{P}_n$ denote the set of all n-fold tensor products of single-qubit Pauli gates with the identity gate, commonly referred to as *Pauli strings*, i.e, $\mathcal{P}_n = \{I, X, Y, Z\}^{\otimes n}$. The Pauli strings form an orthonormal basis, referred to as the Pauli basis, for the full complex matrix space [61]. Hence, we can decompose any $2^n \times 2^n$ complex matrix M as $M = \sum_{P \in \mathcal{P}_n} \gamma_P \cdot P$. Since the Pauli basis is orthonormal, we have for $P, P' \in \mathcal{P}_n$ that $\mathrm{Tr}\Big(P'^\dagger \cdot P\Big) = 0$ if and only if $P' \neq P$. In addition, since P is unitary, we have $\mathrm{Tr}\Big(P^\dagger \cdot P\Big) = \mathrm{Tr}(I) = 2^n$. Thus, the *Pauli coefficients* γ_P can be computed by $\gamma_P = \frac{1}{2^n} \mathrm{Tr}\Big(P^\dagger \cdot M\Big)$, and further as $\gamma_P = \frac{1}{2^n} \mathrm{Tr}(P \cdot M)$ as P is also Hermitian ($P^\dagger = P$). In general, the coefficients γ_P are complex numbers, but for Hermitian matrices, such as density operators and quantum gates, they are real [82, 47].

Let us define the *multiplicative Pauli basis* as the local Pauli matrices X, Z, i.e., $\mathbb{P}_n = \{X_0, Z_0 \ldots, X_{n-1}, Z_{n-1}\}$. The Pauli strings can be described as the multiplicatively closed set of the multiplicative Pauli basis, i.e., $\mathcal{P}_n = \{\prod_{P \in \mathbb{P}_n} M_P \mid M_P \in \{I, P\}\}$. This definition reflects the fact that $I = X \cdot X = Z \cdot Z$ and $Y = i(X \cdot Z)$ so all Pauli strings can be constructed using only X, Z, and their products, up to a global phase.

Example 2.3. Given a density operator $\rho = \frac{1}{2} \cdot (|00\rangle + |11\rangle)(\langle 00| + \langle 11|)$, we can calculate its coefficients in Pauli basis by going over all Pauli matrices in $\mathcal{P}_2$. We get that the non-zero coefficients are:

$$\frac{1}{2^1}\operatorname{Tr}((I \otimes I)\rho) = \frac{1}{2} \quad , \quad \frac{1}{2^1}\operatorname{Tr}((Z \otimes Z)\rho) = \frac{1}{2},$$
$$\frac{1}{2^1}\operatorname{Tr}((X \otimes X)\rho) = \frac{1}{2} \quad , \quad \frac{1}{2^1}\operatorname{Tr}((Y \otimes Y)\rho) = -\frac{1}{2}$$

It is straightforward to verify that these are ρ's Pauli *real* coefficients, since

$$\frac{1}{2}(I \otimes I) + \frac{1}{2}(Z \otimes Z) + \frac{1}{2}(X \otimes X) - \frac{1}{2}(Y \otimes Y)$$
$$= \frac{1}{2}\left(\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} - \begin{bmatrix}1&0&0&0\\0&-1&0&0\\0&0&-1&0\\0&0&0&1\end{bmatrix} + \begin{bmatrix}0&0&0&1\\0&0&1&0\\0&1&0&0\\1&0&0&0\end{bmatrix} - \begin{bmatrix}0&0&0&-1\\0&0&1&0\\0&1&0&0\\-1&0&0&0\end{bmatrix}\right) = \frac{1}{2} \cdot \begin{bmatrix}1&0&0&1\\0&0&0&0\\0&0&0&0\\1&0&0&1\end{bmatrix},$$

which is the matrix representation of ρ. Additional examples are:

$$|0\rangle^{\otimes n}\langle 0|^{\otimes n} = (\frac{I+Z}{2})^{\otimes n} \quad , \quad |1\rangle^{\otimes n}\langle 1|^{\otimes n} = (\frac{I-Z}{2})^{\otimes n},$$
$$|+\rangle^{\otimes n}\langle +|^{\otimes n} = (\frac{I+X}{2})^{\otimes n} \quad , \quad |-\rangle^{\otimes n}\langle -|^{\otimes n} = (\frac{I-X}{2})^{\otimes n}. \qquad \triangle$$

Clifford Group. An important set of quantum gates is the *Clifford group*, as it can describe interesting quantum mechanical phenomena such as entanglement, teleportation, and superdense encoding. More importantly, it is widely used in quantum error-correcting codes [28, 108] and measurement-based quantum computation [98]. The Clifford group is the set of unitary operators that map the Pauli group to itself through conjugation, i.e., all the $2^n \times 2^n$ unitary matrices U such that $UPU^\dagger \in \{\lambda P' \mid P' \in \mathcal{P}_n, \lambda \in \{\pm 1, \pm i\}\}$ for all $P \in \mathcal{P}_n$. It can be generated by the Hadamard gate H, the phase gate S, and the two-qubit control-not gate CX:

$$H \triangleq \frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix} \quad , \quad S \triangleq \begin{bmatrix}1&0\\0&i\end{bmatrix} \quad , \quad \text{and} \quad CX \triangleq \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}.$$

By extending the Clifford gate set with any non-Clifford gate, we obtain a *universal gate set*, in the sense that any unitary operator can be approximated to arbitrary accuracy using only gates from this set [39, 65, 66]. Examples of such gates are the $T = \sqrt{S}$ or $\mathit{Toffoli} = CCX$ gates defined as:

$$T \triangleq \begin{bmatrix}1&0\\0&e^{i\pi/4}\end{bmatrix} \quad , \quad \mathit{Toffoli} \triangleq \begin{bmatrix}I^{\otimes 2}&\mathbf{0}\\\mathbf{0}&CX\end{bmatrix},$$

where $\mathbf{0}$ represents the zero matrix of suitable dimensions. Other examples are arbitrary rotation gates such as:

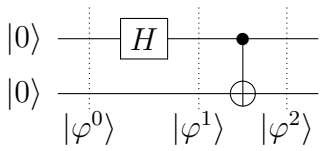
$$R_X(\theta) \triangleq \begin{bmatrix}\cos\frac{\theta}{2}&-i\sin\frac{\theta}{2}\\-i\sin\frac{\theta}{2}&\cos\frac{\theta}{2}\end{bmatrix}, \quad R_Y(\theta) \triangleq \begin{bmatrix}\cos\frac{\theta}{2}&-\sin\frac{\theta}{2}\\\sin\frac{\theta}{2}&\cos\frac{\theta}{2}\end{bmatrix}, \quad \text{or} \quad R_Z(\theta) \triangleq \begin{bmatrix}e^{-i\theta/2}&0\\0&e^{i\theta/2}\end{bmatrix}.$$

Quantum Circuit. The evolution of a quantum system is modeled by a *quantum circuit*, a sequence $\mathcal{C} \equiv (D^0, \ldots, D^{m-1})$ of quantum gate *layers* D^t, applied to all qubits at time step $t \in [m]$. We can construct the unitary $U_\mathcal{C}$ of a circuit $\mathcal{C}$ from the uniteries U_{D^t} of the layers as $U_\mathcal{C} = U_{D^{m-1}} \cdots U_{D^0}$. We define the circuit's *depth* as its number of layers.

For two circuits $\mathcal{C}_1 \equiv (D_1^0, \ldots, D_1^{m_1-1})$ and $\mathcal{C}_2 \equiv (D_2^0, \ldots, D_2^{m_2-1})$ we define there composition (applying $\mathcal{C}_1$ and then $\mathcal{C}_2$) to be $\mathcal{C} = \mathcal{C}_2 \cdot \mathcal{C}_1 = (D_1^0, \ldots, D_1^{m_1-1}, D_2^0, \ldots, D_2^{m_2-1})$. It is thus represented by the unitary $U_{\mathcal{C}} = U_{\mathcal{C}_2} \cdot U_{\mathcal{C}_1}$.

A circuit can be drawn with a line to represent each qubit (—) and a labeled square to represent gates ($\boxed{G}$). Control gates are represented by a dot (•) for the control qubit with a line to the target gate. For CX (controlled-X), the X is represented by an $\oplus$. We use vertical dashed lines (┆) to denote intermediate states between layers.

Example 2.4. Let us consider the circuit $\mathcal{C} = (\{H_0\}, \{CX_{0,1}\})$. It is drawn as follows:



The unitary of the circuit $\mathcal{C}$ is given by multiplying the matrices of $CX_{0,1}$ and H_0:

$$U_{\mathcal{C}} = CX_{0,1} \cdot H_0 = \begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&0&1 \\ 0&0&1&0 \end{bmatrix} \cdot \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 1&0&-1&0 \\ 0&1&0&-1 \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 0&1&0&-1 \\ 1&0&-1&0 \end{bmatrix}.$$

And the output state $|\varphi_2\rangle$ is obtained by computing:

$$|\varphi_2\rangle = U_{\mathcal{C}} \cdot |00\rangle = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1&0&1&0 \\ 0&1&0&1 \\ 0&1&0&-1 \\ 1&0&-1&0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \cdot (|00\rangle + |11\rangle). \qquad \triangle$$

Measurements. A linear operator $\mathcal{M}$ is a *projector* if and only if $\mathcal{M}\mathcal{M} = \mathcal{M}$. A *projective measurement* is defined by a set of projectors $\{\mathcal{M}_0, \ldots, \mathcal{M}_{k-1}\}$ — one for each measurement outcome in $[k]$ — satisfying $\sum_{j\in[k]} \mathcal{M}_j = I^{\otimes n}$. The probability of getting the outcome j when measuring a state $|\psi\rangle$ is $\langle\psi| \mathcal{M}_j |\psi\rangle$. If a projector satisfies $\mathcal{M}^\dagger = \mathcal{M}$, we say the projector is orthogonal. For a two-qubit system, measuring under the two-qubit computational basis is given by the measurement $\{\ |00\rangle\langle 00|\,,\ |10\rangle\langle 10|\,,\ |10\rangle\langle 10|\,,\ |11\rangle\langle 11|\ \}$. Note that for a projector $\mathcal{M} = |\psi'\rangle\langle\psi'|$ we have $\langle\psi| \mathcal{M} |\psi\rangle = |\langle\psi'|\psi\rangle|^2$.

Simulation. There exist multiple definitions of how a quantum computation can be simulated. In this work, we use the following definition, referred to as *strong simulation* in the literature:

Definition 2.5 (Strong Simulation [33])**.** Given an n-qubit quantum circuit $\mathcal{C}$ applied to an initial state $|\psi\rangle$ and a projector $\mathcal{M}$, a strong simulation computes the value $\langle\psi| \mathcal{C}^\dagger \mathcal{M} \mathcal{C} |\psi\rangle$.

Example 2.6. Continuing with example Example 2.4, a strong simulation of the projector $\mathcal{M} = |11\rangle\langle 11|$ for the circuit $\mathcal{C} = (\{H_0\}, \{CX_{0,1}\})$ applied to the initial state $|00\rangle$ will give us the probability:

$$\begin{aligned} p &= \langle 00| \mathcal{C}^\dagger |11\rangle\langle 11| \mathcal{C} |00\rangle = |\langle 11| \cdot CX_{0,1} \cdot H_0 \cdot |00\rangle|^2 \\ &= |\langle 11|\varphi_2\rangle|^2 = |\langle 11| \cdot \frac{1}{\sqrt{2}} \cdot (|00\rangle + |11\rangle)|^2 \\ &= |\frac{1}{\sqrt{2}}(0+1)|^2 = \frac{1}{2} \end{aligned} \qquad \triangle$$

Jamiołkowski Fidelity. The *fidelity* between two quantum states $|\psi\rangle$ and $|\phi\rangle$ is defined as

$$\mathrm{Fid}(|\phi\rangle, |\psi\rangle) \triangleq \mathrm{Tr}(|\psi\rangle\langle\psi| \cdot |\phi\rangle\langle\phi|) = \mathrm{Tr}(\langle\phi|\psi\rangle \cdot \langle\psi|\phi\rangle) = |\langle\psi|\phi\rangle|^2.$$

In the equation above, we use the fact that the trace is invariant under cyclic permutations. That is, for any matrices A, B, C (of compatible dimensions) it holds that: $\mathrm{Tr}(ABC) = \mathrm{Tr}(CAB) = \mathrm{Tr}(BCA)$. The fidelity between states can be extended to measure the distance between unitary operators with the help of *Choi-Jamiołkowski isomorphism*, that maps a n-qubit operator U to a $2n$-qubit state $|\Psi_U\rangle = (U \otimes I^{\otimes n})|\Psi_n\rangle$, where $|\Psi_n\rangle$ is the $2n$-qubit maximally entangled state $\frac{1}{\sqrt{2^n}} \sum_{i\in\{0,1\}^n} |ii\rangle$ and $|ii\rangle$ is short for $|i\rangle \otimes |i\rangle$. Then, the *Jamiołkowski fidelity* [97] between two n-qubit unitary operators U and V can be formally defined as:

$$\mathrm{Fid}_J(U, V) \triangleq \mathrm{Fid}(|\Psi_U\rangle, |\Psi_V\rangle). \tag{2.1}$$

and in Chapter 6, we further show that:

$$\mathrm{Fid}_J(U, V) = \frac{1}{4^n} \left|\mathrm{Tr}\left(V^\dagger U\right)\right|^2$$

In particular, when $\mathrm{Fid}_J(U, V) = 1$, it follows that the uniterities are equal up to a global phase, i.e., $U = \lambda V$ for some λ such that $|\lambda|^2 = 1$. For circuits $\mathcal{C}_1$, $\mathcal{C}_2$, we will define the Jamiołkowski fidelity between them to be the Jamiołkowski fidelity of their unitaries, i.e., $\mathrm{Fid}_J(\mathcal{C}_1, \mathcal{C}_2) \triangleq \mathrm{Fid}_J(U_{\mathcal{C}_1}, U_{\mathcal{C}_2})$, and denote $|\Psi_{\mathcal{C}}\rangle \triangleq |\Psi_{U_{\mathcal{C}}}\rangle$.

Example 2.7. Let us compute the Jamiołkowski fidelity between the single qubit circuits $\mathcal{C}_2 = (T)$ and $\mathcal{C}_1 = (R_Z(\frac{\pi}{8}))$:

$$\begin{aligned}
|\Psi_{\mathcal{C}_1}\rangle &= (U_{\mathcal{C}_1} \otimes I)|\Psi\rangle \\
&= (T \otimes I)\left(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\right) \\
&= \frac{1}{\sqrt{2}}\left(|00\rangle + e^{i\pi/4}|11\rangle\right) \\
|\Psi_{\mathcal{C}_2}\rangle &= (U_{\mathcal{C}_2} \otimes I)|\Psi_1\rangle \\
&= \left(R_Z\left(\frac{\pi}{8}\right) \otimes I\right)\left(\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)\right) \\
&= \frac{1}{\sqrt{2}}\left(e^{-i\pi/16}|00\rangle + e^{i\pi/16}|11\rangle\right)
\end{aligned}$$

$$\begin{aligned}
\mathrm{Fid}_J(\mathcal{C}_1, \mathcal{C}_2) &= |\langle\Psi_{\mathcal{C}_2}|\Psi_{\mathcal{C}_1}\rangle|^2 \\
&= \left|\frac{1}{\sqrt{2}}\left(e^{i\pi/16}\langle 00| + e^{-i\pi/16}\langle 11|\right) \cdot \frac{1}{\sqrt{2}}\left(|00\rangle + e^{i\pi/4}|11\rangle\right)\right|^2 \\
&= \frac{1}{4}\left|e^{i\pi/16} + e^{-i\pi/16} \cdot e^{i\pi/4}\right|^2 = \frac{1}{4}\left|e^{-i1\pi/16} + e^{i1\pi/16}\right|^2 = \frac{2}{4}\sin^2\left(1\pi/16\right) = \\
&= 0.962
\end{aligned}$$

△

2.2 Maximum Weighted Model Counting

Let $\mathbb{B} = \{0, 1\}$ denote the Boolean domain. A *Boolean formula* over a finite set of variables A is an expression built by combining *literals* using logical operators such as $\wedge$, $\vee$, and $\neg$. Each variable $a \in A$ gives rise to two *literals*: the *positive literal* a, which is satisfied when $a = 1$, and the *negative literal* $\overline{a}$, which is satisfied when $a = 0$.

Syntactic Boolean formulas define *Boolean functions* $F \colon \mathbb{B}^A \to \mathbb{B}$, which map variable assignments to output values. An *assignment* $\tau \in \mathbb{B}^A$ gives a value to each variable in A. We often write such assignments using *cube notation*, i.e., as a conjunction of literals — for example, $a \wedge b$, or more compactly, ab.

A Boolean formula F is said to be *satisfiable* if there exists an assignment $\tau \in \mathbb{B}^A$ such that $F(\tau) = 1$. The set of all satisfying assignments is defined as

$$SAT(F) \triangleq \{\tau \in \mathbb{B}^A \mid F(\tau) = 1\}.$$

Example 2.8. Given the formula $F = b\overline{c} \wedge \overline{b}c$ over $A = \{a, b, c\}$, there are four satisfying assignments: $SAT(F) = \{abc,\ a\overline{b}\overline{c},\ \overline{a}bc,\ \overline{a}\overline{b}\overline{c}\}$. △

A *model counting* problem (MC) counts the number of satisfying assignments, i.e.,

$$\#SAT(F) \triangleq |SAT(F)|.$$

Example 2.9. For the formula F, from Example 2.8, we have $\#SAT(F) = 4$. △

A *weighted model counting* problem (WMC) extends model counting by assigning each solution a weight, which has applications in areas like Bayesian inference and network analysis [101, 30]. Allowing the user to specify an arbitrary weight function would hinder most heuristics typically used in WMC. Therefore, we define the weight function on literals, following [83]. We use a function $W : A \times \mathbb{B} \to \mathbb{R}$ which maps each variable and its Boolean assignment to a real-valued weight. Given an assignment τ, the weight of this assignment, written $W(\tau)$, is the product of the weight of each variable with its assignment $W(\tau) = \prod_{a \in A} W(a, \tau(a))$. For notational convenience, we write the weights of literals, such that for a variable $a \in A$, we denote $W(a) = W(a, 1)$ and $W(\overline{a}) = W(a, 0)$. We say a variable a is *unbiased* if and only if $W(a) = W(\overline{a}) = 1$. If the weight of a literal is not stated, it is assumed to be 1. For a propositional formula F over variables in A and weight function W, *weighted model counting* sums the weights of the satisfying assignments, denoted as:

$$\#SAT_W(F) \triangleq \sum_{\tau \in SAT(F)} W(\tau).$$

Example 2.10. For the formula F, from Example 2.8, we define the weight function W as $W(a) = \frac{1}{4}$, $W(\overline{a}) = -\frac{3}{4}$, $W(b) = -\frac{1}{2}$, $W(\overline{b}) = \frac{1}{3}$, and c remains unbiased ($W(c) = W(\overline{c}) = 1$).

Then the weight of the assignment $abc \in SAT(F)$ can be computed as:

$$W(a)W(b)W(c) = \frac{1}{4} \cdot (-\frac{1}{2}) \cdot 1 = -\frac{1}{8}$$

By summing up the weights of all satisfying assignments, we get $\#SAT_W(F) = \frac{1}{12}$. △

A maximum weighted model counting problem (MWMC) extends WMC by giving an assignment to a subset of the variables to maximize the WMC over the given formula. The problem can be formally defined as follows:

Definition 2.11 (MWMC[11])**.** Given a formula $F(A, B)$ over disjoint sets of variables A and B with a weight function W over $\mathbb{B}^{(A\cup B)}$, the MWMC problem is to determine an assignment τ to A that maximizes $\#SAT_W(F(\tau, B))$.

For notation, we define an oracle $Max\#SAT_W$, which takes the Boolean formula $F(A, B)$ with the weight function W and returns an assignment τ to A that maximizes the WMC of F and the maximum weight w_{max} it achieves, i.e., $Max\#SAT_W(F(A, B)) = (\tau, w_{max})$.

Example 2.12. For the formula F with its weighted function W from Example 2.10, we consider the MWMC of the formula $F(\{a, c\}, \{b\})$. We assign the variables a to be false and c to be true to achieve the maximum weight $w_{max} = W(\overline{a})W(b)W(c) = \frac{3}{8}$. Thus, we have:

$$Max\#SAT_W(F(\{a, c\}, \{b\})) = (\overline{a}c, \frac{3}{8}). \qquad \triangle$$

2.3 Encoding Quantum Computing in Pauli basis

In this section, we will describe the encoding of quantum computing in the Pauli basis, based on the work in [81], denoted by **PB**. In Chapter 6, we will present an alternative encoding in the computational basis, denoted by **CB**.

Encoding Pauli Strings. To encode one Pauli matrix, we reserve two Boolean variables x and z, and define Boolean formulas such that the satisfying assignments represent the matrix. Let us denote a formula encoding a matrix M as F_M. Then we encode the Pauli matrix as $F_X(x, z) = x\overline{z}$, $F_Z(x, z) = \overline{x}z$, $F_Y(x, z) = xz$ and $F_I(x, z) = \overline{x}\overline{z}$. A Pauli string can thus be encoded by $2n$ Boolean variables $\{x_1, z_1, \ldots, x_n, z_n\}$.

Example 2.13. $P = X \otimes Y \otimes Z$ is encoded by:

$$F_P(x_0, z_0, x_1, z_1, x_2, z_2) = x_0\overline{z}_0x_1z_1\overline{x}_2z_2. \qquad \triangle$$

Encoding Quantum States. In the **PB** encoding, we encode the density matrix of the state. Since density matrices of quantum states are Hermitian, they can be decomposed into a linear combination of Pauli strings $P \in \mathcal{P}_n$ with real coefficients. The density matrices are thus represented by a Boolean formula, such that each satisfying assignment represents a Pauli string, and the weight of the assignment corresponds to its coefficient. To that end, for each qubit $i \in [n]$ we reserve two variables $\boldsymbol{q}_i = (x_i, z_i)$ to encode states, and add weighted auxiliary variables where needed. We will omit the auxiliary variables in the notation, i.e., write $F(\boldsymbol{q})$, for $\boldsymbol{q} = (\boldsymbol{q}_0, \ldots, \boldsymbol{q}_{n-1})$, since all the auxiliary variables are fully determined by $\boldsymbol{q}$.

Example 2.14. Let us consider the all-zero quantum state $|0\rangle^{\otimes n}\langle 0|^{\otimes n} = (\frac{Z+I}{2})^{\otimes n}$. The Pauli string decomposition of this state consists of all tensor products in $\{I, Z\}^{\otimes n}$. This is encoded by $\overline{x}_i$ for $i \in [n]$ (z_i is left free), allowing for $\overline{x}_i\overline{z}_i$ and $\overline{x}_iz_i$, corresponding to

I and Z. To add the coefficients, which are all $\frac{1}{2^n}$ in this case, we introduce an auxiliary variable, w, with the weight $W(w) = \frac{1}{2^n}$ ($W(\overline{w}) = 1$), and add w to the encoding, to ensure the weight is applied to all assignments. Thus, we have:

$$F_{|0\rangle^{\otimes n}}(\boldsymbol{q}) = \overline{x}_0\overline{x}_1 \dots \overline{x}_{n-1}w. \qquad \triangle$$

Encoding Quantum Gates. Quantum gates map each Pauli string to one or more Pauli strings, with possible coefficients. The encoding of a quantum gate G is written as $F_G(\boldsymbol{q}, \boldsymbol{q}')$, where $\boldsymbol{q}$ and $\boldsymbol{q}'$ respectively represent the Pauli strings before and after applying the gate. Thus, $F_G(\boldsymbol{q}, \boldsymbol{q}')$ should limit the before and after Pauli string combination to correspond to the operator, and when needed, an auxiliary variable with a weight will be introduced to encode the coefficients. We omit auxiliary variables from the signature, as they are uniquely generated per instance. In this manner, we can represent the state $|\psi'\rangle = G|\psi\rangle$ by the encoding $F_{|\psi'\rangle}(\boldsymbol{q}') \equiv F_{|\psi\rangle}(\boldsymbol{q}) \wedge F_G(\boldsymbol{q}, \boldsymbol{q}')$. The encoding for the H, T, and CX gates is displayed in Table 2.1.

	Gate Action	**Encoding**	**Weights**
$I = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$	$IXI^\dagger = X$, $IZI^\dagger = Z$	$(x' \Leftrightarrow x) \wedge (z' \Leftrightarrow z)$	
$H = 1/\sqrt{2}\left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$	$HXH^\dagger = Z$, $HZH^\dagger = X$	$(x' \Leftrightarrow z) \wedge (z' \Leftrightarrow x) \wedge (r \Leftrightarrow xz)$	$W(r) = -1$
$T = \left[\begin{smallmatrix} 1 & 0 \\ 0 & \frac{i+1}{\sqrt{2}} \end{smallmatrix}\right]$	$TXT^\dagger = \frac{1}{\sqrt{2}}(X+Y)$, $TZT^\dagger = Z$	$(x' \Leftrightarrow x) \wedge (x \vee (z' \Leftrightarrow z)) \wedge$ $(r \Leftrightarrow xz\overline{z}') \wedge (u \Leftrightarrow x)$	$W(r) = -1$ $W(u) = \frac{1}{\sqrt{2}}$
$CX = \left[\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right]$	$CX(I \otimes X)CX^\dagger = (I \otimes X)$, $CX(I \otimes Z)CX^\dagger = (Z \otimes Z)$, $CX(X \otimes I)CX^\dagger = (X \otimes X)$, $CX(Z \otimes I)CX^\dagger = (Z \otimes I)$	$(x'_c \Leftrightarrow x_c) \wedge (x'_t \Leftrightarrow x_c \oplus x_t) \wedge$ $(z'_c \Leftrightarrow z_c \oplus z_t) \wedge (z'_t \Leftrightarrow z_t) \wedge$ $(r \Leftrightarrow x_c z_t (x_t \oplus \overline{z}_c))$	$W(r) = -1$

Table 2.1: Overview of the **PB** encoding for the gates I, H, T, and CX. The **Gate Action** column shows how each gate transforms the multiplicative Pauli basis states under conjugation. The **Encoding** column describes the **PB** encoding, with **Weights** describing the weight on the relevant variables. For a gate G we show the encoding of $F_G(\boldsymbol{q} = (x, z), \boldsymbol{q}' = (x', z'))$ if it is single-qubit and of $F_G(\boldsymbol{q} = (x_c, z_c, x_t, z_t), \boldsymbol{q}' = (x'_c, z'_c, x'_t, z'_t))$ if it is two-qubit. Unless otherwise stated, literals are assigned a weight of 1.

Encoding Quantum Layers. A layer is encoded by concatenating all the gate encodings, such that each is applied to the relevant state variables. Recall that each qubit has exactly one gate applied to it in a layer, including I gates. So for a layer D, if we denote for a gate $G_{indxes} \in D$ with $var(indxes)$ the state variables corresponding to the qubits G is applied to, the encoding for D is $F_D(\boldsymbol{q}) = \bigwedge_{G_{indxes} \in D} F_G(var(indxes))$.

Encoding Quantum Circuits. A quantum circuit $\mathcal{C} = (D^0, \dots, D^{m-1})$ is encoded by conjoining the layer encodings, such that each is applied to consecutive state variables, i.e., $F_{\mathcal{C}}(\boldsymbol{q}^0, \dots, \boldsymbol{q}^m) = \bigwedge_{j \in \{0,\dots,m-1\}} F_{G^j}(\boldsymbol{q}^j, \boldsymbol{q}^{j+1})$. Each circuit instance introduces new intermediate state variables, which are omitted from the signature, i.e., write $F_{\mathcal{C}}(\boldsymbol{q}^0, \boldsymbol{q}^m)$ or even $F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q}')$.

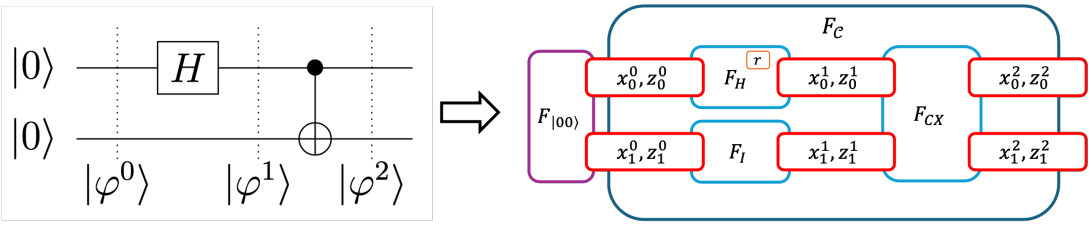


Figure 2.2: An illustration for encoding the circuit from Example 2.15. Formulas are defined over the variables they overlap with. The rectangles represent: Red - state variables; Orange - weighted auxiliary variables; Blue - the circuit formula; Cyan - gate formulas; Purple - state formulas.

Example 2.15. Let us show the formula for the circuit $\mathcal{C} = (\{H_0\}, \{CX_{0,1}\})$ from Example 2.2, applied on the state $|00\rangle$, and compute the satisfying assignments representing the output state. Figure 2.2 illustrates the encoding.

The formula for this problem is

$$F_{\mathcal{C}|00\rangle}(\boldsymbol{q}^2) = F_{|00\rangle}(\boldsymbol{q}^0) \wedge \overbrace{F_H(\boldsymbol{q}_0^0, \boldsymbol{q}_0^1) \wedge F_I(\boldsymbol{q}_1^0, \boldsymbol{q}_1^1) \wedge F_{CX}((\boldsymbol{q}_0^1, \boldsymbol{q}_1^1), (\boldsymbol{q}_0^2, \boldsymbol{q}_1^2))}^{F_{\mathcal{C}}(\boldsymbol{q}^0, \boldsymbol{q}^1, \boldsymbol{q}^2)}$$

where $\boldsymbol{q}_i^t = (x_i^t, z_i^t)$ represents qubit i in the density operator $|\varphi_t\rangle\langle\varphi_t|$ and:

$$\begin{aligned}
F_{|00\rangle}(\boldsymbol{q}^0) &= \overline{x}_0^0 \overline{x}_1^0 w^0 \\
F_H(\boldsymbol{q}_0^0, \boldsymbol{q}_0^1) &= (x_0^1 \Leftrightarrow z_0^0) \wedge (z_0^1 \Leftrightarrow x_0^0) \wedge (r_0^1 \Leftrightarrow x_0^0 z_0^0) \\
F_I(\boldsymbol{q}_1^0, \boldsymbol{q}_1^1) &= (x_1^1 \Leftrightarrow x_1^0) \wedge (z_1^1 \Leftrightarrow z_1^0) \\
F_{CX}((\boldsymbol{q}_0^1, \boldsymbol{q}_1^1), (\boldsymbol{q}_0^2, \boldsymbol{q}_1^2)) &= (x_0^2 \Leftrightarrow x_0^1) \wedge (x_1^2 \Leftrightarrow x_0^1 \oplus x_1^1) \\
&\quad \wedge (z_0^2 \Leftrightarrow z_0^1 \oplus z_1^1) \wedge (z_1^2 \Leftrightarrow z_1^1) \\
&\quad \wedge (r_0^2 \Leftrightarrow x_0^1 z_1^1 (x_1^1 \oplus \overline{z}_0^1)) \\
W(v) &= \begin{cases} \frac{1}{2^2}, & \text{if } v = w^0 \\ -1, & \text{if } v \in \{r_0^1, r_0^2\} \\ 1, & \text{otherwise} \end{cases}
\end{aligned}$$

For this formula, we have:

$$SAT(F(\boldsymbol{q}^2)) = \{\overline{x}_0^2 \overline{z}_0^2 \overline{x}_1^2 \overline{z}_1^2,\ \overline{x}_0^2 z_0^2 \overline{x}_1^2 z_1^2,\ x_0^2 \overline{z}_0^2 x_1^2 \overline{z}_1^2,\ x_0^2 z_0^2 x_1^2 z_1^2\}$$

such that for $\tau \in SAT(F(\boldsymbol{q}^2))$ we have:

$$W(\tau) = \begin{cases} -\frac{1}{4}, & \text{if } \tau = x_0^2 x_1^2 z_0^2 z_1^2 \\ \frac{1}{4}, & \text{otherwise} \end{cases}$$

Note that we are only interested in the assignment to the variables in $\boldsymbol{q}^2$. If multiple satisfying extensions of this assignment exist, we sum the weights of all such complete assignments.

The weighted assignments represent to the density operator:

$$\rho = \frac{1}{4}\left((I \otimes I) + (Z \otimes Z) + (X \otimes X) - (Y \otimes Y)\right)$$

Based on Example 2.2 the expected output is

$$|\varphi_2\rangle = \frac{1}{\sqrt{2}} \cdot (|00\rangle + |11\rangle)$$

for which we have $\rho = |\varphi_2\rangle\langle\varphi_2|$ based on Example 2.3. △

Encoding Strong Simulation. To encode a simulation, we begin by encoding projectors. Since a projector is a matrix, it can be represented using the same encoding method as the density matrix of states. Note that since we use only real weights in the encoding, we can only encode orthogonal projectors ($\mathcal{M}^\dagger = \mathcal{M}$). Given encodings for states, circuits, and projectors, we can then compute the probability of measuring the outcome corresponding to an orthogonal projector $\mathcal{M}$, for a given circuit $\mathcal{C}$ applied to an initial state $|\psi\rangle$, as follows:

Proposition 2.16 (Strong Simulation in **PB** [81])**.** Let $\mathcal{C}$ be an n-qubit quantum circuit, $|\psi\rangle$ a state over n-qubit, and $\mathcal{M}$ an orthogonal projector. We will denote their **PB** encodings $F_{|\psi\rangle}, F_{\mathcal{C}}$, and $F_{\mathcal{M}}$ respectively, with an according weight function W. It holds that:

$$\langle\psi|\,\mathcal{C}^\dagger\mathcal{M}\mathcal{C}\,|\psi\rangle = \#SAT_W(F_{|\psi\rangle}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q}') \wedge F_{\mathcal{M}}(\boldsymbol{q}')).$$

In this formulation, the conjunction $F_{|\psi\rangle}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q}')$ encodes the entire output distribution of the circuit applied to the input state. The structure of this encoding compactly represents all possible measurement outcomes and their associated amplitudes. To extract a specific probability or measurement statistic, we simply conjoin the corresponding projector encoding $F_{\mathcal{M}}(\boldsymbol{q}'))$, and evaluate the result using WMC. Moreover, note that a wide variety of measurements can be simulated in this fashion, including measurements on subsets of the qubits.

Chapter 3

Problem Statement

The quantum circuit synthesis problem aims to find a circuit that implements a desired specification. The desired specification is defined by either a circuit in a different gate set or a unitary operator. A key component is to determine if the desired specification is *equivalent* to the synthesized circuit, as formalized in Definition 3.1.

Definition 3.1 (Equivalence checking)**.** Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be n-qubit circuits. Then the circuits are ϵ-equivalent for an accuracy parameter $\epsilon \in [0, 1]$, denoted by $\mathcal{C}_1 \simeq_\epsilon \mathcal{C}_2$, if and only if $\mathrm{Fid}_J(\mathcal{C}_1, \mathcal{C}_2) \geq 1 - \epsilon$. The circuits are equivalent, denoted by $\mathcal{C}_1 \equiv \mathcal{C}_2$, if and only if $\mathrm{Fid}_J(\mathcal{C}_1, \mathcal{C}_2) = 1$.

In the above definition, we use the Jamiołkowski fidelity, defined in Equation 2.1, to measure the distance between two quantum functionalities. Note that in this work, we define equivalence up to a global phase [86], i.e., $\mathcal{C}_1 \equiv \mathcal{C}_2$ if and only if $U_{\mathcal{C}_1} = \lambda U_{\mathcal{C}_2}$ for some λ such that $|\lambda|^2 = 1$.

Using the definition of equivalence checking, Definition 3.2 now defines synthesis.

Definition 3.2 (Synthesis)**.** Let $\mathcal{C}_{in}$ be a quantum circuit implementing a specification in a gate set $\mathcal{G}_{in}$, and let $\mathcal{G}_{out}$ be a target gate set. Given an accuracy parameter $\epsilon \in [0, 1]$, the *synthesis problem* asks for a circuit $\mathcal{C}_{out}$ over $\mathcal{G}_{out}$ such that $U_{\mathcal{C}_{in}} \simeq_\epsilon \mathcal{C}_{out}$. When $\epsilon = 0$, this is referred to as *exact synthesis*; for $\epsilon \in (0, 1]$, it is an instance of *approximate synthesis*.

We consider Clifford+T as the elementary target gate set for synthesis due to its universality and its central role in error-corrected quantum computing [46, 4]. A unitary operator can be exactly synthesized using Clifford+T gates if and only if its matrix entries lie in the ring $\mathbb{Z}[\frac{1}{\sqrt{2}}, i]$, possibly requiring the use of ancilla qubits in the resulting circuit [49]. More generally, however, any unitary can be approximately synthesized to arbitrary precision using only Clifford+T gates and without the need for ancillae [39].

In this work, we consider both exact and approximate synthesis and, for the first time, show that both problems can be reduced to MWMC. Moreover, we achieve depth-optimal synthesis.

Chapter 4

Exact Synthesis

This chapter presents our reduction of the quantum circuit synthesis problem to the MWMC problem. From Definition 3.2, the exact synthesis problem has the following components:

- **Input:** A quantum circuit $\mathcal{C}_{in}$ in a gate set $\mathcal{G}_{in}$ and a target gate set $\mathcal{G}_{out}$.
- **Output:** If possible, a depth-optimal circuit $\mathcal{C}_{out}$ in the $\mathcal{G}_{out}$ gate set such that $\mathcal{C}_{out} \equiv \mathcal{C}_{in}$.

Given the output gate set $\mathcal{G}_{out}$, we define $\mathcal{D}_n$ to be the set of all possible layers for n qubits, such that on each qubit exactly one gate is applied, including I gates. The problem can be expressed as exhaustively searching over all possible layers $\mathcal{D}_n$ for each layer in $\mathcal{C}_{out}$. To achieve depth-optimality, we increment the depth d until the following holds:

$$\exists D^0, \ldots, D^{d-1} \in \mathcal{D}_n \ : \ \mathcal{C}_{in} \equiv \mathcal{C}_{out} \text{ where } \mathcal{C}_{out} = (D^0, \ldots, D^{d-1}). \tag{4.1}$$

In the following sections, we begin by encoding exact equivalence checking in WMC, which serves as a basis for exact synthesis. Then, to explore the space of possible gate layers, we encode the complete set $\mathcal{D}_n$ by introducing gate-selecting variables — once these variables are fixed, they uniquely determine a specific gate layer within the set. Lastly, we merge those and, by searching for solutions of increasing depths, synthesize a circuit equivalent to the given one.

4.1 Encoding Gate Layers for Synthesis

To encode a general gate layer $\mathcal{D}_n$, we introduce gate-selection variables representing all possible gate choices within the layer. Note that we always include the identity gate I_i, even if it is not explicitly listed in the target gate set, to allow for qubits left unaffected in a given layer. We will describe the encoding for a general gate set $\mathcal{G}$. For example, for the gate set $\mathcal{G}_{CT} = \{CX, H, T, T^\dagger\}$, a layer $D \in \mathcal{D}_n$ is a maximal set of parallel gates such that $D \subseteq \{I_i, H_i, T_i, T_i^\dagger, CX_{i,j} \mid i, j \in [n], i \neq j\}$.

Let $\mathcal{G}(k) \subseteq \mathcal{G}$ be a subset of gates applied to $k \in [n]$ qubits, such that $\mathcal{G}(1)$ is the set of single-qubit gates, $\mathcal{G}(2)$ is the set of two-qubit gates, and so on. For example, we have $\mathcal{G}_{CT}(1) = \{I, H, T, T^\dagger\}$ and $\mathcal{G}_{CT}(2) = \{CX\}$. In this encoding, we assume the gate set

has no gates applied to more than two qubits, but if there are, the definition is expanded accordingly.

We first consider single-qubit gates $\mathcal{G}(1)$. For each gate $G \in \mathcal{G}(1)$ and each qubit $i \in [n]$, we define the Boolean variable $p_{G,i}$. The variable is true if and only if gate G is applied to qubit i, i.e., G_i is in the layer if and only if $p_{G,i}$ is assigned 1. Thus, we can encode the single-qubit gates of the layer as:

$$F_{\mathcal{G}(1)}(\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{p}(1)) = \bigwedge_{i \in [n]} \bigwedge_{G \in \mathcal{G}(1)} (p_{G,i} \Rightarrow F_G(\boldsymbol{q}_i, \boldsymbol{q}'_i)), \tag{4.2}$$

where the variables $\boldsymbol{q}$ and $\boldsymbol{q}'$ encode the states before and after the layer respectively, and $\boldsymbol{p}(1) = \{p_{G,i} \mid G \in \mathcal{G}(1), i \in [n]\}$ are the single-qubit gate-selecting variables.

Similarly, for the two-qubit gates $\mathcal{G}(2) \subseteq \mathcal{G}$, we introduce a variable for each gate for each pair of qubits: $\boldsymbol{p}(2) = \{p_{G,i,j} \mid G \in \mathcal{G}(2), i, j \in [n], j \neq i\}$. The encoding is then defined as follows:

$$F_{\mathcal{G}(2)}(\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{p}(2)) = \bigwedge_{i,j \in [n], j \neq i} \bigwedge_{G \in \mathcal{G}(2)} (p_{G,i,j} \Rightarrow F_G(\boldsymbol{q}_i, \boldsymbol{q}_j, \boldsymbol{q}'_i, \boldsymbol{q}'_j)), \tag{4.3}$$

This gives us the set of gate-selecting variables $\boldsymbol{p} = \boldsymbol{p}(1) \cup \boldsymbol{p}(2)$ of the synthesis layer.

Lastly, let us remember the requirement to have exactly one gate applied to each qubit within a layer. To that end, we define :

$$\texttt{EXO}(V) = \left(\bigvee_{v \in V} v \right) \wedge \left(\bigwedge_{u,v \in V, u \neq v} (\overline{v} \vee \overline{u}) \right), \tag{4.4}$$

a constraint ensuring that exactly one variable in the set V is assigned 1, and apply it for all qubits:

$$F_{\texttt{EXO}}(\boldsymbol{p}) = \bigwedge_{i \in [n]} \texttt{EXO}(\boldsymbol{p}_i), \tag{4.5}$$

where $\boldsymbol{p}_i = \{p_{G,i} \mid G \in \mathcal{G}_{out}(1)\} \cup \{p_{G,i,j}, p_{G,j,i} \mid G \in \mathcal{G}_{out}(2), j \in [n], j \neq i\}$.

Combining the three constraints in Equations 4.2, 4.3 and 4.5 gives us the layer encoding:

$$F_{\mathcal{D},\mathcal{G}}(\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{p}) = F_{\mathcal{G}(1)}(\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{p}(1)) \wedge F_{\mathcal{G}(2)}(\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{p}(2)) \wedge F_{\texttt{EXO}}(\boldsymbol{p}) \tag{4.6}$$

For the universal gate set $\mathcal{G}_{CT}$ the encoding of a layer is as follows:

$$\begin{aligned} F_{\mathcal{D},\mathcal{G}_{CT}}(\boldsymbol{q}, \boldsymbol{q}', \boldsymbol{p}) = \bigwedge_{i \in [n]} \Big(& (p_{I,i} \Rightarrow F_I(q_i, q'_i)) \wedge (p_{H,i} \Rightarrow F_H(q_i, q'_i)) \wedge \\ & (p_{T,i} \Rightarrow F_T(q_i, q'_i)) \wedge (p_{T^\dagger,i} \Rightarrow F_{T^\dagger}(q_i, q'_i)) \Big) \\ \wedge \bigwedge_{i,j \in [n], j \neq i} & \Big(p_{CX,i,j} \Rightarrow F_{CX}((q_i, q_j), (q'_i, q'_j)) \Big) \quad \wedge \bigwedge_{i \in [n]} \texttt{EXO}(\boldsymbol{p}_i), \end{aligned} \tag{4.7}$$

where $\boldsymbol{p}_i = \{p_{I,i}, p_{H,i}, p_{T,i}, p_{T^\dagger,i}\} \cup \bigcup_{j \in [n], j \neq i} \{p_{CX,i,j}, p_{CX,j,i}\}$ and $\boldsymbol{p} = \bigcup_{i \in [n]} \boldsymbol{p}_i$.

Note that, by construction, each satisfying assignment to the gate-selection variables $\boldsymbol{p}$ corresponds to exactly one valid gate configuration in the layer. That is, there is a one-to-one correspondence between such assignments and valid gate layers.

Extending this to full circuits, we can encode a generic circuit of depth d as follows:

$$F_{\mathcal{C}_{syn},\mathcal{G},d}(\boldsymbol{q}^0, \boldsymbol{q}^d, \boldsymbol{p}) = \bigwedge_{t\in[d]} F_{\mathcal{D},\mathcal{G}}(\boldsymbol{q}^t, \boldsymbol{q}^{t+1}, \boldsymbol{p}^t). \tag{4.8}$$

where $\boldsymbol{p} = \bigcup_{t\in[d]} \boldsymbol{p}^t$. In this formulation, each satisfying assignment to $\boldsymbol{p}$ encodes exactly one valid quantum circuit composed of d layers. Figure 4.1 illustrates this encoding for 3 layers. Thus, the encoding induces a bijection between satisfying assignments to the gate-selection variables and the space of d-depth circuits over the gate set $\mathcal{G}$.

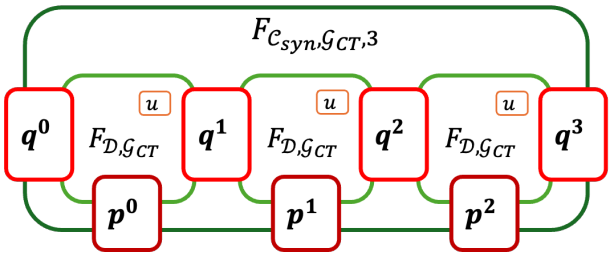


Figure 4.1: An illustration for encoding a synthesis circuit with 3 layers, as described in Equation 4.8. Formulas are defined over the variables they overlap with. The rectangles represent: Maroon - gate-selecting variables; Red - state variables; Orange - weighted auxiliary variables; Dark Green - the synthesis circuit formula; Light Green - synthesis layers formulas (Equation 4.7).

4.2 Exact Equivalence Checking

Due to the unitarity of quantum circuits, verifying the equivalence of the circuits $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$ is reducible to checking if the circuit $\mathcal{C} = \mathcal{C}_{out} \cdot \mathcal{C}_{in}^{\dagger}$ is equivalent to the identity. The exact equivalence checking problem is efficiently tackled with WMC in [82], using a so-called *linear encoding*. It depends on the **PB** encoding, and consists of $2n$ WMC calls. In Theorem 4.1, we extend this encoding to *cyclic encoding* and *linear-cyclic encoding*, two new encodings that solve the problem with a single call to the weighted model counter, *as we require in the proposed synthesis approach*.

Theorem 4.1 (Exact Equivalence Checking in **PB**)**.** Let $\mathcal{C}$ be an n-qubit circuit encoded by $F_{\mathcal{C}}$ in the **PB**, with the corresponding weight function W. Then, the following four statements are equivalent:

- $\mathcal{C} \equiv I^{\otimes n}$
- **Linear encoding [82] (L):**

$$\forall \mathcal{P} \in \mathbb{P}_n : \#SAT_W(F_{\mathcal{P}}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{\mathcal{P}}(\boldsymbol{q'})) = 1 \tag{4.9}$$

- **Linear-Cyclic encoding (LC)**:

$$\#SAT_W(F_{\mathbb{P}_n}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{I^{\otimes n}}(\boldsymbol{q}, \boldsymbol{q'})) = 2n \tag{4.10}$$

- **Cyclic encoding (C):**

$$\#SAT_W(F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'}) \wedge F_{I^{\otimes n}}(\boldsymbol{q},\boldsymbol{q'})) = 4^n \tag{4.11}$$

where $F_{I^{\otimes n}}(\boldsymbol{q},\boldsymbol{q'}) = \bigwedge_{i\in[n]}((x_i \Leftrightarrow x'_i) \wedge (z_i \Leftrightarrow z'_i))$, $F_{\mathbb{P}_n}(\boldsymbol{q}) = \bigvee_{\mathcal{P}\in\mathbb{P}_n} F_{\mathcal{P}}(\boldsymbol{q})$, and $F_{\mathcal{P}}$ is the **PB** encoding of a multiplicative Pauli basis string $\mathcal{P} \in \mathbb{P}_n = \{X_j, Z_j \mid j \in [n]\}$.

Proof. From [82, Cor. 1], $\mathcal{C}$ and $I^{\otimes n}$ are equivalent if and only if Equation 4.9 holds. We now show that Equation 4.10 and Equation 4.11 are both equivalent to Equation 4.9.

For Equation 4.10, as stated in Corollary 1 and the proof of Lemma 2 in [82], we have

$$\#SAT_W(F_{\mathcal{P}}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'}) \wedge F_{\mathcal{P}}(\boldsymbol{q'})) \leq 1.$$

for all $\mathcal{P} \in \mathbb{P}_n$, and the WMC is 1 if and only if $\mathcal{C}$ is equivalent to $I^{\otimes n}$ over $\mathcal{P}$. Thus, we infer that

$$\sum_{\mathcal{P}\in\mathbb{P}_n} \#SAT_W(F_{\mathcal{P}}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'}) \wedge F_{\mathcal{P}}(\boldsymbol{q'})) \leq 2n,$$

where the value achieves $2n$ if and only if each of the summands achieves 1. Therefore Equation 4.10 is true if and only if Equation 4.9 is true, as demonstrated in [82, Prop. 1].

For Equation 4.11, the idea is similar. Since the state variables $\boldsymbol{q}$ and $\boldsymbol{q'}$ are equivalent but free, they can be assigned any Pauli string $\mathcal{P} \in \mathcal{P}_n = \{I, X, Y, Z\}^{\otimes n}$. Thus, for Equation 4.11 we have

$$\#SAT_W(F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'}) \wedge F_{I^{\otimes n}}(\boldsymbol{q},\boldsymbol{q'})) = \sum_{\mathcal{P}\in\mathcal{P}_n} \#SAT_W(F_{\mathcal{P}}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q}) \wedge F_{\mathcal{P}}(\boldsymbol{q})),$$

As stated before, each of the summands can contribute at most 1 to the summation, so the summation achieves 4^n if and only if each summand achieves 1. Since $\mathbb{P}_n \subseteq \mathcal{P}_n$, we have *Equation* 4.11 $\Rightarrow$ *Equation* 4.9. From [110], if two unitaries are equivalent over $\mathbb{P}_n$, they are equivalent over $\mathcal{P}_n$. Thus, we have Equation 4.9$\Rightarrow$ Equation 4.11. Therefore Equation 4.9$\Leftrightarrow$ Equation 4.11. □

Example 4.2. Consider two single-qubit circuits $\mathcal{C}_{in} = (S)$ and $\mathcal{C}_{out} = (T, T)$. To check their equivalence, we first encode the circuit $\mathcal{C} = \mathcal{C}_{out} \cdot \mathcal{C}_{in}^{\dagger}$:

$$F_{\mathcal{C}} = F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'}) = F_{\mathcal{C}}(\boldsymbol{q}^0,\boldsymbol{q}^3) = F_{S^\dagger}(\boldsymbol{q}^0,\boldsymbol{q}^1) \wedge F_T(\boldsymbol{q}^1,\boldsymbol{q}^2) \wedge F_T(\boldsymbol{q}^2,\boldsymbol{q}^3)$$

For readability, we let $F = F(\boldsymbol{q})$, $F' = F(\boldsymbol{q'})$ and $F_I = F_I(\boldsymbol{q},\boldsymbol{q'})$ Then we check if $\mathcal{C} \equiv I$ in one of the following ways:

- The linear encoding: $\#SAT_W(F_{\mathcal{P}} \wedge F_{\mathcal{C}} \wedge F'_{\mathcal{P}}) = 1$ for $\mathcal{P} \in \{X, Z\}$.
- The linear-cyclic encoding: $\#SAT_W((F_X \vee F_Z) \wedge F_{\mathcal{C}} \wedge F_I) = 2$.
- The cyclic encoding: $\#SAT_W(F_{\mathcal{C}} \wedge F_I) = 4$. △

4.3 Encoding Exact Synthesis

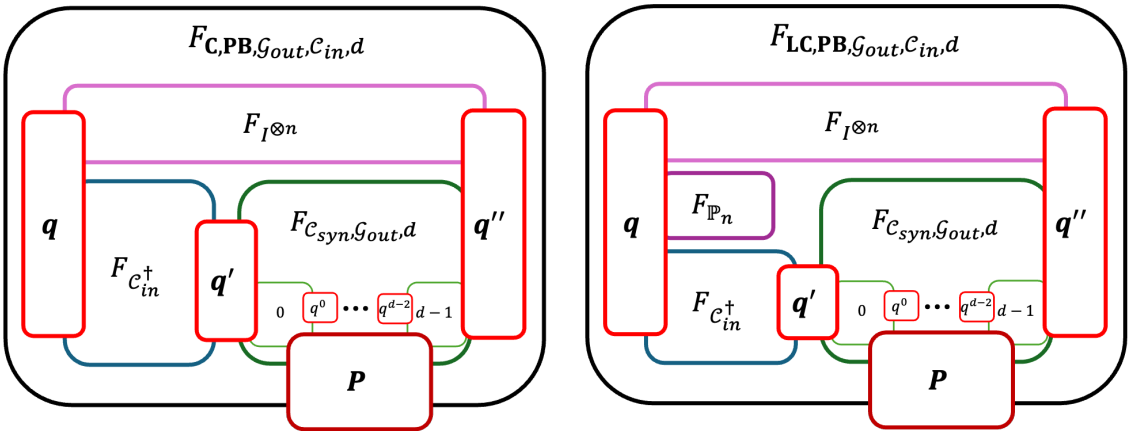
We now show the encoding of exact synthesis, using the encoding of exact equivalence checking and the gate layers given above. The main idea is to determine a minimal sequence of gate layers $\mathcal{C}_{out} = (D^1, \ldots, D^d)$ such that $\mathcal{C}_{out} \equiv \mathcal{C}_{in}$. For a depth d, we can encode the layers with the cyclic encoding for the equality check as follows:

$$Syn_{\mathbf{C},\mathbf{PB},\mathcal{G}_{out},\mathcal{C}_{in},d}(P,Q) \quad = \quad \overbrace{F_{\mathcal{C}_{in}^\dagger}(\boldsymbol{q},\boldsymbol{q}') \wedge F_{\mathcal{C}_{syn},\mathcal{G}_{out},d}(\boldsymbol{q}',\boldsymbol{q}'',\boldsymbol{p})}^{\mathcal{C}_{out}\cdot\mathcal{C}_{in}^\dagger} \quad \wedge \quad \overbrace{(\boldsymbol{q} \Leftrightarrow \boldsymbol{q}'')}^{F_{I^{\otimes n}}}, \tag{4.12}$$

where $\mathbf{C}$ denotes the encoding is cyclic, $\mathbf{PB}$ denotes the Pauli basis encoding is used, $\mathcal{G}_{out}$ is the target gate set for the synthesis, $P = \boldsymbol{p}$ is the set of gate-selecting variables, and Q is the set of all other variables, consisting of the state variables ($\boldsymbol{q}$, $\boldsymbol{q}'$, $\boldsymbol{q}''$, and intermediate states) and auxiliary variables (within the gate encodings). For linear-cyclic checking, denoted by $\mathbf{LC}$, the encoding is done by adding constraints to the initial variables

$$Syn_{\mathbf{LC},\mathbf{PB},\mathcal{G}_{out},\mathcal{C}_{in},d}(P,Q) = F_{\mathbb{P}_n}(\boldsymbol{q}) \wedge Syn_{\mathbf{C},\mathbf{PB},\mathcal{G}_{out},\mathcal{C}_{in},d}(P,Q). \tag{4.13}$$

Figure 4.2 illustrates the cyclic and linear-cyclic synthesis encodings.



(a) Cyclic synthesis encodings (b) Linear-Cyclic synthesis encodings

Figure 4.2: An illustration for the synthesis encodings, as described in Equation 4.12 and Equation 4.13. Formulas are defined over the variables they overlap with. The rectangles represent: Maroon - gate-selecting variables; Red - state variables; Orange - weighted auxiliary variables; Light Green - synthesis layers formulas (Equation 4.7); Dark Green - the synthesis circuit formula; Blue - the circuit formula; Cyan - gate formulas; Purple - state formula; Pink - equality formulas.

Proposition 4.3 shows how the exact synthesis problem is reduced to the MWMC problem by finding an assignment for the gate-selecting variables that maximizes the fidelity between the input and output circuits. As noted earlier, each satisfying assignment to the gate-selection variables corresponds to a unique circuit of depth d. For each such assignment, the encoding represents exactly that circuit. This establishes a one-to-one correspondence between assignments and possible circuits, ensuring the completeness of the construction. The correctness of the proposition follows directly from the cyclic and linear cyclic cases of Theorem 4.1.

Proposition 4.3 (Exact synthesis in **PB**)**.** Given a quantum circuit $\mathcal{C}_{in}$ in a gate set $\mathcal{G}_{in}$ and an integer d, there exists a d-depth circuit $\mathcal{C}_{out}$ in gate set $\mathcal{G}_{out}$ such that $\mathcal{C}_{in} \equiv \mathcal{C}_{out}$ if and only if, for the **PB** encoding, it holds that

$$Max\#SAT_W\Big(Syn_{\mathbf{E},\mathbf{PB},\mathcal{G}_{out},\mathcal{C}_{in},d}(P,Q)\Big) = (c, \tau(P)),$$

such that $c = 2n$ when $\mathbf{E}$ is $\mathbf{LC}$ and $c = 4^n$ when $\mathbf{E}$ is $\mathbf{C}$. When the maximum value is achieved, $\mathcal{C}_{out}$ is directly defined by the satisfying assignment $\tau(P)$.

By checking for incremental depths d, we can synthesize a depth-optimal quantum circuit. The following shows an example.

Example 4.4. Let us consider the circuit $\mathcal{C}_{in} = (S)$. Since it is a single-qubit circuit, the encoding of the general layer is:

$$\begin{aligned} F_{\mathcal{D},\mathcal{G}_{CT}}(\boldsymbol{q},\boldsymbol{q}',\boldsymbol{p}) = &\Big(p_{I,0} \Rightarrow F_I(q_0,q_0') \wedge p_{H,0} \Rightarrow F_H(q_0,q_0') \wedge \\ &\quad p_{T,0} \Rightarrow F_T(q_0,q_0') \wedge p_{T^\dagger,0} \Rightarrow F_{T^\dagger}(q_0,q_0')\Big) \\ &\wedge \bigwedge_{i\in[n]} \texttt{EXO}(\boldsymbol{p}_0), \end{aligned}$$

where $\boldsymbol{p} = \boldsymbol{p}_0 = \{p^0_{I,0}, p^0_{H,0}, p^0_{T,0}, p^0_{T^\dagger,0}\}$. We synthesize the circuit by calling an MWMC to find an optimal assignment for the gate-selection variables:

$$\begin{aligned} &Max\#SAT_W(Syn_{\mathbf{LC},\mathbf{PB},\mathcal{G}_{CT},\mathcal{C}_{in},1}(P,Q)) = (0.8535533906,\ \tau_1 = \{\overline{p}^0_{I,0}\overline{p}^0_{H,0}p^0_{T,0}\overline{p}^0_{T^\dagger,0}\}) \\ &Max\#SAT_W(Syn_{\mathbf{LC},\mathbf{PB},\mathcal{G}_{CT},\mathcal{C}_{in},2}(P,Q)) = (1,\ \tau_2 = \{\overline{p}^0_{I,0}\overline{p}^0_{H,0}p^0_{T,0}\overline{p}^0_{T^\dagger,0}, \\ &\qquad\qquad\qquad\qquad\qquad\qquad \overline{p}^0_{I,0}\overline{p}^1_{H,0}p^1_{T,0}\overline{p}^1_{T^\dagger,0}\}) \end{aligned}$$

As we can see, for one synthesis layer, we achieve a fidelity of less than 1, proving that there is no circuit of depth one (or less) equivalent to the input circuit. On the other hand, for two synthesis layers we achieve a fidelity of 1 with the synthesized circuit $\mathcal{C}_{out} = (T,T)$ and we can conclude that $(S) \equiv (T,T)$. △

Chapter 5

Approximate Synthesis

In this chapter, we focus on approximate synthesis as given in Definition 3.2. The input and output of the problem are now as follows:

- **Input:** A quantum circuit $\mathcal{C}_{in}$ in a gate set $\mathcal{G}_{in}$, a target gate set $\mathcal{G}_{out}$, and an accuracy parameter $\epsilon \in (0, 1]$.
- **Output:** A depth-optimal circuit $\mathcal{C}_{out}$ in the $\mathcal{G}_{out}$ gate set such that $\mathcal{C}_{out} \simeq_{\epsilon} \mathcal{C}_{in}$.

Since synthesis relies on equivalence checking, our first aim is to lift the latter to approximate equivalence checking. Theorem 5.1 shows that the cyclic encoding in **PB** computes the Jamiołkowski fidelity between two circuits, thus implementing Definition 3.1.

Theorem 5.1 (Approximate Equivalence Checking in **PB**)**.** Given two n-qubit circuits $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$, their Jamiołkowski fidelity is computed by the **PB** cyclic encoding for $\mathcal{C} = \mathcal{C}_{out} \cdot \mathcal{C}_{in}^{\dagger}$ as follows:

$$\mathrm{Fid}_J(\mathcal{C}_{in}, \mathcal{C}_{out}) \quad = \quad \frac{1}{4^n} \cdot |\#SAT_W(F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{I^{\otimes n}}(\boldsymbol{q}, \boldsymbol{q'}))|$$

Proof. To compute the Jamiołkowski fidelity of two n-qubit circuits $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$, one takes the maximally entangled state $|\Psi_n\rangle$ as input and computes the fidelity between the output states $(U_{\mathcal{C}_{in}} \otimes I^{\otimes n}) |\Psi_n\rangle$ and $(U_{\mathcal{C}_{out}} \otimes I^{\otimes n}) |\Psi_n\rangle$.

We start by representing $|\Psi_n\rangle$ and the Jamiołkowski fidelity in the Pauli basis. We will then demonstrate how the computations correspond to the representations.

As shown in [31], the density operator of the maximally entangled state can be decomposed in the Pauli basis as:

$$|\Psi_n\rangle\langle\Psi_n| = \frac{1}{2^n} \sum_{i \in [n], j \in [n]} |ii\rangle\langle jj| = \frac{1}{4^n} \sum_{\mathcal{P} \in \{X,Y,Z,I\}^{\otimes n}} \mathcal{P} \otimes \mathcal{P}^T$$

It follows that, for $|\Psi_{\mathcal{C}}\rangle = (U_{\mathcal{C}} \otimes I^{\otimes n}) |\Psi_n\rangle$, we have:

$$|\Psi_{\mathcal{C}}\rangle\langle\Psi_{\mathcal{C}}| = (U_{\mathcal{C}} \otimes I^{\otimes n}) |\Psi_n\rangle\langle\Psi_n| (U_{\mathcal{C}}^{\dagger} \otimes I^{\otimes n}) = \frac{1}{4^n} \sum_{\mathcal{P} \in \{X,Y,Z,I\}^{\otimes n}} U\mathcal{P}U^{\dagger} \otimes \mathcal{P}^T$$

Thus, the Jamiołkowski fidelity for $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$ can be defined in the Pauli basis as:

$$\begin{aligned}\mathrm{Fid}_J(\mathcal{C}_{in},\mathcal{C}_{out}) &= \mathrm{Fid}_J(\mathcal{C}_{in}^\dagger,\mathcal{C}_{out}^\dagger) = \mathrm{Tr}\Big(\Big|\Psi_{\mathcal{C}_{in}^\dagger}\Big\rangle\Big\langle\Psi_{\mathcal{C}_{in}^\dagger}\Big|\Big|\Psi_{\mathcal{C}_{out}^\dagger}\Big\rangle\Big\langle\Psi_{\mathcal{C}_{out}^\dagger}\Big|\Big)\\ &= \frac{1}{16^n}\sum_{\mathcal{P},\mathcal{P}'\in\{X,Y,Z,I\}^{\otimes n}} \mathrm{Tr}\Big(U_{\mathcal{C}_{in}}^\dagger\mathcal{P}U_{\mathcal{C}_{in}}U_{\mathcal{C}_{out}}^\dagger\mathcal{P}'U_{\mathcal{C}_{out}}\Big)\cdot\mathrm{Tr}\Big(\mathcal{P}^T\mathcal{P}'^T\Big)\\ &= \frac{1}{8^n}\sum_{\mathcal{P}\in\{X,Y,Z,I\}^{\otimes n}} \mathrm{Tr}\Big(U_{\mathcal{C}_{out}}U_{\mathcal{C}_{in}}^\dagger\mathcal{P}U_{\mathcal{C}_{in}}U_{\mathcal{C}_{out}}^\dagger\mathcal{P}\Big)\end{aligned} \tag{5.1}$$

where the equality $\mathrm{Fid}_J(\mathcal{C}_{in},\mathcal{C}_{out}) = \mathrm{Fid}_J(\mathcal{C}_{in}^\dagger,\mathcal{C}_{out}^\dagger)$ will be proven in detail in Theorem 6.2. Based on Proposition 1 in [82], for $U_{\mathcal{C}} = U_{\mathcal{C}_{out}}U_{\mathcal{C}_{in}}^\dagger$ each of the summands $\frac{1}{2^n}\cdot\mathrm{Tr}\Big(U_{\mathcal{C}}^\dagger\mathcal{P}U_{\mathcal{C}}\mathcal{P}\Big)$ can be computed by the weighted model counting of $F_{\mathcal{P}}(\boldsymbol{q})\wedge F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'})\wedge F_{\mathcal{P}}(\boldsymbol{q'})$, or equivalently, $F_{\mathcal{P}}(\boldsymbol{q})\wedge F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'})\wedge(\boldsymbol{q}\Leftrightarrow\boldsymbol{q'})$. Since the summation is over all possible Pauli strings, it can be computed by leaving the variables in $\boldsymbol{q}$ to be free. Hence, the Jamiołkowski fidelity is obtained by the cyclic encoding:

$$\mathrm{Fid}_J(\mathcal{C}_{in},\mathcal{C}_{out}) = \frac{1}{4^n}\#SAT_W(F_{\mathcal{C}}(\boldsymbol{q},\boldsymbol{q'})\wedge F_{I^{\otimes n}}(\boldsymbol{q},\boldsymbol{q'})).$$

□

Since the cyclic encoding computes the Jamiołkowski fidelity, we can decide approximate equivalence checking according to Definition 3.1. Thus, we can reuse the exact synthesis cyclic encoding from Equation 4.12 for approximate synthesis, as is shown in Proposition 5.2. Similarly to Proposition 4.3, the completeness follows from the constructions, and the correctness from Definition 3.1.

Proposition 5.2 (Approximate synthesis in **PB**)**.** Given a quantum circuit $\mathcal{C}_{in}$, an integer d and an accuracy parameter $\epsilon\in(0,1]$, there exists a d-depth circuit $\mathcal{C}_{out}$ in gate set $\mathcal{G}$ such that $\mathcal{C}_{out}\simeq_\epsilon\mathcal{C}_{in}$ if and only if, for the **PB** encoding, it holds that

$$Max\#SAT_W\Big(Syn_{\mathbf{C},\mathbf{PB},\mathcal{G},\mathcal{C}_{in},d}(P,Q)\Big) = (c,\tau(P))$$

such that $\frac{c}{4^n}\geq 1-\epsilon$. In that case, $\mathcal{C}_{out}$ is directly defined by the satisfying assignment $\tau(P)$.

Like in exact synthesis, to get the depth-optimal ϵ-approximate circuit, we apply the above Proposition 5.2 for an increasing depth d. Example 5.3 demonstrates approximate synthesis.

Example 5.3. Consider the circuit $\mathcal{C}_{in} = (R_Z(\frac{\pi}{8}))$ with $\epsilon = 0.05$. Synthesizing one gate layer, we get:

$$Max\#SAT_W(Syn_{\mathbf{C},\mathbf{PB},\mathcal{G}_{CT},\mathcal{C}_{in},1}(P,Q)) = (3.848,\ \tau_1 = \{\overline{p}_{I,0}^0\overline{p}_{H,0}^0p_{T,0}^0\overline{p}_{T^\dagger,0}^0\})$$

From the above result, we can determine the output circuit is $\mathcal{C}_{out} = (T)$ and the fidelity $\mathrm{Fid}_J(\mathcal{C}_{in},\mathcal{C}_{out}) = \frac{1}{4}\times 3.848 = 0.962 > 1-\epsilon$. Thus, the synthesis procedure stops and outputs the circuit (T). △

Chapter 6

Synthesis in the Computational Basis

In addition to the previous encoding in the Pauli basis ($\mathbf{PB}$), we introduce a new encoding that represents quantum states directly in the computational basis ($\mathbf{CB}$). The new encoding is more comprehensible and requires fewer variables, but utilizes complex weights.

6.1 Computational Basis Encoding

Encoding Quantum States. In $\mathbf{CB}$, an n-qubit quantum state $|\psi\rangle$ is encoded by a Boolean formula $F_{|\psi\rangle}$ on n distinct variables $\boldsymbol{q} = \{q_0, \ldots, q_{n-1}\}$ and some auxiliary variables (which we will specify later) together with a weight function W on those variables. Semantically, $F_{|\psi\rangle}$ should be defined such that when projected onto a bitstring $b \in \{0,1\}^n$, its weighted model count is precisely the amplitude $\langle b|\psi\rangle$. To express this, we write:

$$\#SAT_W(F_{|\psi\rangle}(\boldsymbol{q}) \wedge G_{|b\rangle}(\boldsymbol{q})) = \langle b|\psi\rangle \tag{6.1}$$

where $G_{|b\rangle} = \bigwedge_{j=1}^{n} l_j$ such that $l_j = q_j$ if $b_j = 1$ and $l_j = \overline{q_j}$ if $b_j = 0$ and the weights of the variables are unbiased.

To build the construction for $F_{|\psi\rangle}$ and W, we start with the simplest case: an n-qubit state $|\psi\rangle = |b\rangle$ for some bitstring $b \in \{0,1\}^n$. This state is encoded using a Boolean formula $F_{|\psi\rangle}$ on n distinct variables $\boldsymbol{q} = \{q_0, \ldots, q_{n-1}\}$. Because the inner product between $|b\rangle$ and $|b'\rangle$ equals 1 only if $b = b'$ and 0 otherwise, we infer from Equation 6.1 that the formula $F_{|\psi\rangle}$ should only allow for $q_j = b_j$ for $j \in [n]$, i.e., that the variable q_j is assigned 1 if $b_j = 1$ and 0 if $b_j = 0$. We will denote such an assignment by $\boldsymbol{q} = b$. For this reason, we can choose $F_{|\psi\rangle}$ to be a cube dictating this assignment, namely $F_{|\psi\rangle} = G_{|b\rangle}$. For example, the state $|0\rangle$ is encoded as $F_{|0\rangle}(\boldsymbol{q} = (q_0)) = \overline{q_0}$ and the state $|1\rangle$ as $F_{|1\rangle}(\boldsymbol{q} = (q_0)) = q_0$. For the 2-qubit state $|10\rangle$, we have $F_{|10\rangle}(\boldsymbol{q} = (q_0, q_1)) = q_0\overline{q_1}$.

For a general quantum state, which is a superposition of computational basis states, we define $F_{|\psi\rangle}$ such that each term in the summation corresponds to an assignment describing a basis state, and use auxiliary variables to encode the weight such that it corresponds to the relevant coefficient. We will consider for example the state $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$. We want to construct F such that each term in the state corresponds to a satisfying assignment of

F, i.e., both $q_1 = 0$ and $q_1 = 1$ are satisfying assignments of F (corresponding to the terms $|0\rangle$ and $|1\rangle$, respectively). To encode the weight $\frac{1}{\sqrt{2}}$, we introduce an auxiliary variable h with weight $W(\overline{h}) = \frac{1}{\sqrt{2}}$. Now, the formula $F(q_1, h) = \overline{h}$ encodes $|+\rangle$, because the amplitude $\frac{1}{\sqrt{2}}$ belonging to $|0\rangle$ is found as the weighted model count of $F(q_1, h) \wedge \overline{q_1} \equiv F(q_1 = 0, h)$, which equals $W(\overline{q_1}) \cdot W(\overline{h}) = 1 \cdot \frac{1}{\sqrt{2}}$. Similarly, the amplitude $\frac{1}{\sqrt{2}}$ of $|1\rangle$ is found as the weighted model count of $F(q_1, h) \wedge q_1 \equiv F(q_1 = 1, h)$. By extending the weight function on h as $W(h) = -\frac{1}{\sqrt{2}}$, we can encode for example $|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$ as $F(q_1, h) = q_1 \leftrightarrow h$. Two-qubit states are encoded similarly, e.g., $\frac{1}{\sqrt{2}}|01\rangle - \frac{1}{\sqrt{2}}|10\rangle$ is encoded by $F(q_1, q_2, h) = \left(\overline{q_1} \wedge q_2 \wedge \overline{h}\right) \vee (q_1 \wedge \overline{q_2} \wedge h)$.

Encoding Quantum Gates. To encode the action of a gate G in **CB**, we construct a formula F_G dictating the relation between $\boldsymbol{q}$, the state before applying the gate, and $\boldsymbol{q}'$, the state after applying the gate. Namely, for for each state $|\psi\rangle$ and gate G, for which $G|\psi\rangle = |\psi'\rangle$, the following holds:

$$F_{|\psi'\rangle}(\boldsymbol{q}') \equiv F_{|\psi\rangle}(\boldsymbol{q}) \wedge F_G(\boldsymbol{q}, \boldsymbol{q}') \tag{6.2}$$

To this end, $F_G(\boldsymbol{q}, \boldsymbol{q}')$ restricts the allowed combinations of assignments to $\boldsymbol{q}$ and $\boldsymbol{q}'$, introducing a constant number of auxiliary variables to assign weights to specific combinations of before and after states where the gate contributes factors.

	Gate Action	**Encoding**	**Weights**
$I = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$	$I\|0\rangle = \|0\rangle$, $I\|1\rangle = \|1\rangle$	$q' \Leftrightarrow q$	
$H = \frac{1}{\sqrt{2}}\left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$	$H\|0\rangle = \frac{1}{\sqrt{2}}(\|0\rangle + \|1\rangle)$, $H\|1\rangle = \frac{1}{\sqrt{2}}(\|0\rangle - \|1\rangle)$	$h \Leftrightarrow qq'$	$W(h) = -\frac{1}{\sqrt{2}}$, $W(\overline{h}) = \frac{1}{\sqrt{2}}$
$T = \left[\begin{smallmatrix} 1 & 0 \\ 0 & \frac{i+1}{\sqrt{2}} \end{smallmatrix}\right]$	$T\|0\rangle = \|0\rangle$, $T\|1\rangle = \frac{i+1}{\sqrt{2}}\|1\rangle$	$(q' \Leftrightarrow q) \wedge$ $(h \Leftrightarrow q)$	$W(h) = \frac{1+i}{\sqrt{2}}$ $W(\overline{h}) = 1$
$CX_{c,t} = \left[\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{smallmatrix}\right]$	$CX\|00\rangle = \|00\rangle, CX\|01\rangle = \|01\rangle$ $CX\|10\rangle = \|11\rangle, CX\|11\rangle = \|10\rangle$	$(q'_c \Leftrightarrow q_c) \wedge$ $(q'_t \Leftrightarrow (q_c \oplus q_t))$	

Table 6.1: Overview of the **CB** encoding for the gates I, H, T, and CX. The **Gate Action** column shows how each gate transforms the computational basis states. The **Encoding** column describes the **CB** encoding, with **Weights** describing the weight on the relevant variables. For a gate G we show the encoding of $F_G(q, q')$ if it is single-qubit and of $F_G(\boldsymbol{q} = (q_c, q_t), \boldsymbol{q}' = (q'_c, q'_t))$ if it is two-qubit.

The encodings of the $\mathcal{G}_{CT}$ gates are summarized in Table 6.1. We now describe in detail the construction of these encodings, illustrating how each gate in the set is handled. The encodings of additional gates can be derived using similar principles.

- I **gate:** The gate does not affect the state, and no factors are applied. Thus, we have the encoding of I:
$$F_I(\boldsymbol{q} = (q), \boldsymbol{q}' = (q')) = q \Leftrightarrow q'$$
- H **gate:** The gate allows for all combinations of before and after states, and factors are applied. If the before and after states are both $|1\rangle$ the factor is $-\frac{1}{\sqrt{2}}$, and otherwise

it is $\frac{1}{\sqrt{2}}$. Thus, we will introduce an auxiliary variable h, with the weights $W(h) = -\frac{1}{\sqrt{2}}, W(\overline{h}) = \frac{1}{\sqrt{2}}$, and define:

$$F_H(\boldsymbol{q} = (q), \boldsymbol{q}' = (q')) = h \Leftrightarrow qq' \text{ with } W(h) = -\frac{1}{\sqrt{2}}, W(\overline{h}) = \frac{1}{\sqrt{2}}$$

- T **gate:** While the gate does not affect the state of the qubit, a factor is applied if the qubit is in the $|1\rangle$ state. We will define an auxiliary variable h and encode the factor into it by defining its weight to be $W(h) = \frac{1+i}{\sqrt{2}}, W(\overline{h}) = 1$. Then we encode the weight to be applied if and only if the qubit is in the $|1\rangle$ state by defining $h \Leftrightarrow q$. Thus, the encoding of the T gate is:

$$F_T(\boldsymbol{q} = (q), \boldsymbol{q}' = (q')) = (q \Leftrightarrow q') \wedge (h \Leftrightarrow q) \text{ with } W(h) = \frac{1+i}{\sqrt{2}}$$

- CX **gate:** The gate operates on two qubits: a control qubit, denoted q_c, and a target qubit, denoted q_t. The control qubit remains unchanged, i.e., $q'_c \Leftrightarrow q_c$. The target qubit flips its value if and only if the control qubit is in state 1, which is captured by the equivalence: $q'_t \Leftrightarrow (q_c \oplus q_t)$. Since the CX gate introduces no weighting factors, no auxiliary variables are required. Thus, the full encoding of the CX gate is given by:

$$F_{CX}(\boldsymbol{q} = (q_c, q_t), \boldsymbol{q}' = (q'_c, q'_t)) = (q'_c \Leftrightarrow q_c) \wedge (q'_t \Leftrightarrow (q_c \oplus q_t))$$

Note. Each gate and state instance introduces new auxiliary variables with weights when necessary, which are omitted from the function signature. Moreover, as stated before, if the weight of a literal is not explicitly mentioned, it should be assumed to be 1.

Encoding Quantum Layers and Circuits. As with the **PB** encoding, a layer is encoded by concatenating all the gate encodings, such that each is applied to the relevant state variables, and a circuit is encoded by conjoining the layer encodings, such that each is applied to consecutive state variables.

Encoding Strong Simulation. Let us start by encoding a projector $\mathcal{M}$. Since $\mathcal{M}$ is a matrix, we can encode it like a gate, in the form $F_{\mathcal{M}}(\boldsymbol{q}, \boldsymbol{q}')$. Given the encoding of states, circuits and projectors, we can compute the probability of measuring the output for a projector $\mathcal{M}$, for a given circuit $\mathcal{C}$ applied to the initial state $|\psi\rangle$, as follows:

Proposition 6.1 (Strong Simulation in **CB**)**.** Let $\mathcal{C}$ be an n-qubit quantum circuit and $|b\rangle$ a computational basis state over n qubits, and $\mathcal{M}$ a projector. We will denote their **PB** encodings $F_{|b\rangle}, F_{\mathcal{C}}$, and $F_{\mathcal{M}}$ respectively, with an according weight function W. It holds that:

$$\langle b| \mathcal{C}^\dagger \mathcal{M} \mathcal{C} |b\rangle = \#SAT_W(F_{|b\rangle}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q}') \wedge F_{\mathcal{M}}(\boldsymbol{q}', \boldsymbol{q}'') \wedge F_{\mathcal{C}^\dagger}(\boldsymbol{q}'', \boldsymbol{q}''') \wedge F_{|b\rangle}(\boldsymbol{q}''')).$$

In the case of $\mathcal{M} = |b'\rangle\langle b'|$, we have:

$$\langle b| \mathcal{C}^\dagger \mathcal{M} \mathcal{C} |b\rangle = |\langle b'| \mathcal{C} |b\rangle|^2 = |\#SAT_W(F_{|b\rangle}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q}') \wedge F_{|b'\rangle}(\boldsymbol{q}'))|^2.$$

Proof. From Equation 6.2 we have that $F_{|b\rangle}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) = F_{\mathcal{C}|b\rangle}(\boldsymbol{q'})$ and from Equation 6.1 we have that $\#SAT_W(F_{\mathcal{C}|b\rangle}(\boldsymbol{q'}) \wedge F_{|b'\rangle}(\boldsymbol{q'})) = \langle b'| \mathcal{C} |b\rangle$. Similarly,

$$\begin{aligned}
&\#SAT_W(F_{|b\rangle}(\boldsymbol{q}) \wedge F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{\mathcal{M}}(\boldsymbol{q'}, \boldsymbol{q''}) \wedge F_{\mathcal{C}^\dagger}(\boldsymbol{q''}, \boldsymbol{q'''}) \wedge F_{|b\rangle}(\boldsymbol{q'''})) \\
&\qquad = \#SAT_W(F_{\mathcal{C}^\dagger\mathcal{M}\mathcal{C}|b\rangle}(\boldsymbol{q'''}) \wedge F_{|b\rangle}(\boldsymbol{q'''})) \\
&\qquad = \langle b| \mathcal{C}^\dagger\mathcal{M}\mathcal{C} |b\rangle .
\end{aligned}$$

□

6.2 Encoding Equivalence Checking in CB

As with the **PB** encoding, equivalence checking and computation of the Jamiołkowski fidelity in the **CB** can be carried out using the *Cyclic encoding*. However, since the computational basis lacks a linear multiplicative structure like that of the Pauli basis, there are no counterparts to the *Linear encoding* or *Linear-Cyclic encoding* in this setting.

Theorem 6.2 (Equivalence Checking in **CB**)**.** Given two n-qubit circuits $\mathcal{C}_{in}$ and $\mathcal{C}_{out}$ encoded in **CB** by $F_{\mathcal{C}_{in}}$ and $F_{\mathcal{C}_{out}}$ respectively, with the corresponding weight function W, their Jamiołkowski fidelity can be computed using the **cyclic encoding (C)** on $\mathcal{C} = \mathcal{C}_{out} \cdot \mathcal{C}_{in}^\dagger$:

$$\mathrm{Fid}_J(\mathcal{C}_{in}, \mathcal{C}_{out}) \quad = \quad \frac{1}{4^n} \cdot |\#SAT_W(F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{I^{\otimes n}}(\boldsymbol{q}, \boldsymbol{q'}))|^2 \tag{6.3}$$

where $\boldsymbol{q}$ and $\boldsymbol{q'}$ are state variables encoding the initial and final quantum state of $\mathcal{C}$ respectively, and $F_{I^{\otimes n}}(\boldsymbol{q}, \boldsymbol{q'}) = \bigwedge_{i\in[n]}(q_i \Leftrightarrow q'_i)$. Thus, $\mathcal{C}_{in} \equiv \mathcal{C}_{out}$ if and only if:

$$|\#SAT_W(F_{\mathcal{C}}\left(\boldsymbol{q}^0, \boldsymbol{q}^m\right) \wedge F_{I^{\otimes n}}(\boldsymbol{q}^0, \boldsymbol{q}^m))| = 2^n \tag{6.4}$$

Proof. The Jamiołkowski fidelity computations are as follows:

$$\begin{aligned}
\mathrm{Fid}_J\left(\mathcal{C}_{in}, \mathcal{C}_{out}\right) &= \mathrm{Fid}\left(\left(U_{\mathcal{C}_{out}} \otimes I^{\otimes n}\right)|\Psi_n\rangle, \left(U_{\mathcal{C}_{in}} \otimes I^{\otimes n}\right)|\Psi_n\rangle\right) \\
&= \left|\langle\Psi_n| \left(U_{\mathcal{C}_{out}^\dagger} \otimes I^{\otimes n}\right)\left(U_{\mathcal{C}_{in}} \otimes I^{\otimes n}\right)|\Psi_n\rangle\right|^2 \\
&= \left|\langle\Psi_n| \left(U_{\mathcal{C}_{out}^\dagger} U_{\mathcal{C}_{in}} \otimes I^{\otimes n}\right)|\Psi_n\rangle\right|^2 \\
&= \left|\frac{1}{2^n} \sum_{b,b'\in\{0,1\}^n} \langle bb| \left(U_{\mathcal{C}_{out}^\dagger} U_{\mathcal{C}_{in}} \otimes I^{\otimes n}\right)|b'b'\rangle\right|^2 \\
&= \frac{1}{4^n}\left|\sum_{b,b'\in\{0,1\}^n} \langle b| U_{\mathcal{C}_{out}^\dagger} U_{\mathcal{C}_{in}} |b'\rangle \cdot \langle b| I^{\otimes n} |b'\rangle\right|^2 \\
&= \frac{1}{4^n}\left|\sum_{b\in\{0,1\}^n} \langle b| U_{\mathcal{C}_{out}^\dagger} U_{\mathcal{C}_{in}} |b\rangle\right|^2 = \frac{1}{4^n}\left|\mathrm{Tr}\left(U_{\mathcal{C}_{out}^\dagger} U_{\mathcal{C}_{in}}\right)\right|^2
\end{aligned}$$

Similarly, we have $\mathrm{Fid}_J\left(\mathcal{C}_{in}^\dagger, \mathcal{C}_{out}^\dagger\right) = \frac{1}{4^n}\left|\mathrm{Tr}\left(U_{\mathcal{C}_{out}} U_{\mathcal{C}_{in}^\dagger}\right)\right|^2$.

Since a matrix and its transpose have the same trace, and the matrices in the trace of a product can be switched without changing the result, we have:

$$\mathrm{Tr}\Big(U_{\mathcal{C}_{out}^\dagger} U_{\mathcal{C}_{in}}\Big) = \mathrm{Tr}\Big(U_{\mathcal{C}_{in}^\dagger} U_{\mathcal{C}_{out}}\Big) = \mathrm{Tr}\Big(U_{\mathcal{C}_{out}} U_{\mathcal{C}_{in}^\dagger}\Big)$$

Thus, we can conclude that:

$$\mathrm{Fid}_J\left(\mathcal{C}_{in}, \mathcal{C}_{out}\right) = \mathrm{Fid}_J\left(\mathcal{C}_{in}^\dagger, \mathcal{C}_{out}^\dagger\right) = \frac{1}{4^n}\left|\mathrm{Tr}(U_{\mathcal{C}})\right|^2$$

Based on Proposition 6.1, we have that:

$$\begin{aligned}\#SAT_W(F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{I^{\otimes n}}(\boldsymbol{q}, \boldsymbol{q'})) &= \sum_{b\in\{0,1\}^n} \#SAT_W(F_{\mathcal{C}}(\boldsymbol{q}, \boldsymbol{q'}) \wedge F_{|b\rangle}(\boldsymbol{q}) \wedge F_{|b\rangle}(\boldsymbol{q'})) \\ &= \sum_{b\in\{0,1\}^n} \langle b| U_{\mathcal{C}} |b\rangle = \mathrm{Tr}(U_{\mathcal{C}}),\end{aligned}$$

giving us, as we wanted to show, that:

$$\mathrm{Fid}_J(\mathcal{C}_{in}, \mathcal{C}_{out}) = \frac{1}{4^n} \cdot \left|\#SAT_W(F_{\mathcal{C}}\left(\boldsymbol{q}, \boldsymbol{q'}\right) \wedge F_{I^{\otimes n}}(\boldsymbol{q}, \boldsymbol{q'}))\right|^2.$$

Lastly, $\mathcal{C}_{in} \equiv \mathcal{C}_{out}$ if and only if $\mathrm{Fid}_J(\mathcal{C}_{in}, \mathcal{C}_{out}) = 1$, giving us the exact equivalence checking condition:

$$\left|\#SAT_W(F_{\mathcal{C}}\left(\boldsymbol{q}^0, \boldsymbol{q}^m\right) \wedge F_{I^{\otimes n}}(\boldsymbol{q}^0, \boldsymbol{q}^m))\right| = 2^n.$$

□

6.3 Encoding Synthesis in CB

Using the same construction as in the **PB** synthesis encoding, we can represent general gate layers using the **CB** gate encodings. By combining this with the **CB** circuit encoding and the **CB** cyclic equality encoding, we define the synthesis formula $Syn_{\mathbf{C},\mathbf{PB},\mathcal{G},\mathcal{C}_{in},d}(P, Q)$. We then propose the following:

Proposition 6.3 (Synthesis in **CB**)**.** Given a quantum circuit $\mathcal{C}_{in}$, an integer d and an error bound $\epsilon \in [0, 1]$, there exists a d-depth circuit $\mathcal{C}_{out}$ in gate set $\mathcal{G}$ such that $U_{\mathcal{C}_{out}} \simeq_\epsilon U_{\mathcal{C}_{in}}$ iff

$$Max\#SAT_W\Big(Syn_{\mathbf{C},\mathbf{CB},\mathcal{G},\mathcal{C}_{in},d}(P, Q)\Big) = (c, \tau(P))$$

such that $\frac{|c|^2}{4^n} \geq 1 - \epsilon$. In that case, $\mathcal{C}_{out}$ is directly defined by the satisfying assignment $\tau(P)$.

Similarly to Proposition 4.3, the completeness follows from the constructions of the encoding, and the correctness from Theorem 6.2. By gradually increasing the depth d, we can find a depth-optimal approximate synthesis, and by choosing $\epsilon = 0$, we get exact synthesis.

Chapter 7

Comparing the Encodings

In this chapter, we provide a detailed comparison of the two encoding bases — the Pauli basis ($\mathbf{PB}$) and the computational basis ($\mathbf{CB}$). We examine their representational differences and present the asymptotical encoding sizes across the different functionalities. This comparison reveals tradeoffs between encoding, offering practical guidance for selecting the most suitable encoding depending on the functionality and circuit characteristics.

7.1 Comparing the Encoding Bases

While the $\mathbf{PB}$ and $\mathbf{CB}$ encodings represent the same quantum computations and share many similarities, there are a few key differences between them that we highlight below.

State Representation. The most fundamental difference lies in how each basis represents quantum states. In the $\mathbf{PB}$ encoding, a quantum state is represented by decomposing its density matrix into a linear combination of Pauli strings. In contrast, the $\mathbf{CB}$ represents quantum states as computational basis vectors, encoding the state vector directly, which offers a more intuitive and natural representation. To encode an n-qubit quantum state, the $\mathbf{PB}$ requires $2n$ Boolean variables — one for each X and Z component in the Pauli string — while the $\mathbf{CB}$ requires only n variables, corresponding to the computational basis indices.

Weight Types. The choice of encoding basis directly impacts the type of weights used in the model counting formulation. The $\mathbf{PB}$ encodes quantum states through real-valued Pauli coefficients, requiring only real weights. In contrast, the $\mathbf{CB}$ must represent complex amplitudes from the state vector, introducing the need for complex weights. This distinction has important implications for solver compatibility. Our encodings fall under the framework of algebraic model counting (AMC) [62], a generalized approach where weights are combined using customizable addition and multiplication rules. While AMC provides a flexible foundation, complex-weighted formulations are significantly more difficult to support, as most existing solvers are optimized for real, non-negative weights (as commonly used in probabilistic inference). Enabling complex arithmetic in this context requires either extending current tools or designing new solvers with native complex support — posing both theoretical and engineering challenges, essential for supporting the $\mathbf{CB}$ representation.

Gate Encodings. The size of the n-qubit gate encodings is also impacted by the number of

variables used to represent the quantum state. Since the $\mathbf{PB}$ requires $2n$ state variables (one for each X and Z component), compared to n in the $\mathbf{CB}$, the number of clauses needed to define the effect of each gate in the $\mathbf{PB}$ is naturally about twice as large. This contributes to a higher clause count in $\mathbf{PB}$ encodings, especially in deep circuits. Moreover, for specific gates, the encoding cost can differ dramatically between representations: for instance, the $Toffoli$ (CCX) gate can be encoded very compactly in the $\mathbf{CB}$, but leads to a large and complex encoding in the $\mathbf{PB}$ [82].

Gate-Induced Branching. Solver performance is influenced by how individual gates affect the structure of the encoding and, in particular, the number of satisfying assignments. Since circuits are encoded step by step over time, each gate introduces new constraints that evolve the quantum state. Some gates cause a "branching" effect, where a single assignment to the previous time step can lead to multiple consistent extensions in the next step — effectively increasing the model count. In the $\mathbf{PB}$, this behavior occurs primarily with the T gate, which introduces a superposition in the Pauli basis, as seen in Table 2.1. In the $\mathbf{CB}$, it is the H gate that leads to such branching, due to mixing computational basis states, as seen in Table 6.1. This structural difference has significant implications: circuits with many branching gates result in a rapidly growing space of satisfying assignments, which typically increases the solver's workload. As a result, the gate distribution in the input circuit can influence solver performance differently depending on the encoding used, as is shown in Section 10.1.

Simulation Handling. Another implication of the state representation is how simulation is implemented. In the $\mathbf{PB}$, measurement outcomes are naturally expressed via inner products in the density matrix formalism, allowing simulation to be performed with a single circuit traversal. In contrast, the $\mathbf{CB}$ typically requires encoding the circuit twice — once for the state and once for its adjoint — when handling measurements that are not of the form $|b\rangle\langle b|$, as discussed in Proposition 6.1. This duplication increases the encoding size and leads to higher solver overhead. However, while this duplication may be necessary in the $\mathbf{CB}$, the measurement operator itself can often be encoded far more compactly in one basis than in the other, depending on how naturally it decomposes in that basis. As a result, the relative efficiency of simulation can vary significantly with the type of measurement being performed.

Exact Equality Constraints. An important distinction between the bases lies in how circuit equality is encoded. In the $\mathbf{PB}$, we can apply a linear identity check, as presented in Theorem 4.1, which constrains the initial state to a carefully chosen subset of the basis with only $2n$ possible satisfying assignments, as opposed to full 4^n assignments. While the clause-level encoding is somewhat larger, this approach dramatically reduces the search space and improves performance, as presented in Section 10.2. In contrast, the $\mathbf{CB}$ lacks a similar multiplicative structure and must evaluate circuit equivalence over the entire computational basis, comprising 2^n satisfying assignments to the initial state variables.

In conclusion, these structural trade-offs between the two bases — compactness versus weight complexity, linear versus exponential bases for equality checking, and differing gate-induced branching behavior — help explain the solver performance variations observed in Chapter 10. They also offer practical guidance for choosing the most suitable encoding basis depending on the functionality and circuit characteristics.

7.2 Encoding Complexity

To better understand the trade-offs between the two encoding bases, this section analyzes the complexity of encoding various circuit components and functionalities. We summarize these trends asymptotically in Table 7.1.

State. As discussed earlier, the $\mathbf{CB}$ uses n variables to represent the quantum state, while the $\mathbf{PB}$ requires $2n$ variables, to capture both the X and Z components. In most scenarios, state variables do not contribute clauses on their own. For example, in a circuit, intermediate state variables between layers are implicitly constrained by adjacent gate encodings, rather than being directly assigned. When the state is explicitly fixed — as in simulation — the typical choice is the all-zero state $|0\rangle^{\otimes n}$. In that case, both encodings require n single-variable clauses: all state variables are set to 0 in $\mathbf{CB}$, and all X variables are set to 0 in $\mathbf{PB}$. However, in the worst case, restricting a state can require exponentially many clauses, with complexity varying significantly between bases depending on the decomposition.

Gate. Gates that act on a constant number of qubits contribute only a constant number of clauses in either basis. However, the actual encoding size for a given gate can differ substantially between the two, depending on how it decomposes in each basis. More generally, gates in $\mathbf{PB}$ often incur a $\times 2$ overhead due to the doubled number of state variables. Furthermore, for gates acting on multiple qubits, the encoding can grow exponentially with qubit count, and the efficiency of the decomposition can differ sharply between bases.

Circuit. At the circuit level, both encodings scale linearly with the number of qubits and non-identity gates. This is because identity operations are typically implemented as variable equalities like $x \Leftrightarrow y$, which are resolved via direct variable reuse rather than through explicit clause generation, avoiding unnecessary overhead.

Simulation. Simulation involves encoding an initial state, the circuit, and the measurement projector. When the projector corresponds to a simple measurement, such as the *All Zero* projector (checking whether the output state is $|0\rangle^{\otimes n}$), the encoding remains linear in both bases. In even simpler cases —like the *First Zero* projector, which only checks if the first output qubit is $|0\rangle$ — the encoding may even be constant. On the other hand, projectors with complex decompositions can require exponentially many clauses, and the size can vary greatly depending on the basis. Notably, the same measurement may admit a compact encoding in one basis but not the other. Additionally, $\mathbf{CB}$ incurs extra overhead when the projector is not a computational basis state $|b\rangle\langle b|$: in such cases, such as with the First Zero projector, the entire circuit must be duplicated.

Equivalence Checking. Equivalence checking involves encoding two circuits and asserting constraints that enforce their functional equality. In cyclic and cyclic-linear encodings, this includes an identity constraint between the initial and final states, which is linear in the number of qubits even when written explicitly. Additionally, in the cyclic-linear encoding, the initial state space is restricted to $2n$ assignments using the EXO constraint (Equation 4.4) on the $2n$ variables, requiring $O(n^2)$ clauses. This contrasts with the cyclic encodings, which evaluate the circuit over the entire basis: 4^n initial state assignments in $\mathbf{PB}$ and 2^n in $\mathbf{CB}$. In the linear $\mathbf{PB}$ encoding, the WMC is called $2n$ times, once per assignment to the initial state variables, each requiring $2n$ single-variable clauses enforcing the assignment.

Synthesis. Synthesis builds on equivalence checking by encoding d synthesis layers in place

of the second circuit, as described in Section 4.1. Each layer introduces a new set of state variables — n in **CB** and $2n$ in **PB**— and $O(n^2)$ clauses, assuming a gate set composed of only single-qubit and two-qubit gates (otherwise the cost is polynomial in n). These are combined with the input circuit encoding and equivalence constraints described earlier.

In summary, these comparisons illustrate key trade-offs between the two encoding bases. **PB** typically incurs higher clause and variable counts but enables more efficient encodings for equality checks and certain measurements. In contrast, **CB** uses fewer variables and can be more efficient for circuits with simple measurements, but struggles with non-basis projections. Consequently, the choice of encoding should be informed mostly by the intended functionality and the specific gate set used.

	PB	**CB**
State	$2n$ variables	n variables
Gate*	$O(1)$	
Circuit	$O(n+m)$	
Simulation**	$O(n+m)$	
Equivalence Checking (cyclic)	$O(n+m_1+m_2)$ 4^n i.s.a.	2^n i.s.a.
Equivalence Checking (linear-cyclic)	$O(n^2+m_1+m_2)$ $2n$ i.s.a.	—
Equivalence Checking (linear)	$O(n+m_1+m_2)$ 1 i.s.a., $2n$ calls	—
Synthesis***	$O(n \cdot m + n^2 \cdot d)$	

Table 7.1: Encoding complexity across components and functionalities. Notations: n = number of qubits; m (m_1, m_2) = number of gates in the circuit (compared circuits); d = synthesis depth; i.s.a. = the number of valid assignments to the initial state variables. * Assuming gates are applied to a constant number of qubits. ** Assuming simple projectors, such as the All Zero and First Zero, and initial state $|0\rangle^{\otimes n}$. *** Assuming that the target gate set consists of single-qubit and two-qubit gates.

Chapter 8

Implementation: Quokka#

This chapter introduces Quokka#, a comprehensive software library developed to realize the SAT-based quantum circuit synthesis techniques described throughout this thesis. While synthesis is the central goal, Quokka# also incorporates related functionalities such as circuit simulation and equivalence checking, which share a great deal in common and even build upon one another. Simulation encodes a given quantum circuit to analyze the probabilities of different measurement outcomes for a specified input state. Equivalence checking builds on this by comparing the behaviors of two circuits to determine whether they produce the same outcomes under all inputs. Synthesis further generalizes both tasks by encoding a partially specified output circuit and using equivalence constraints to ensure it behaves like the input circuit (or given specification) across all relevant inputs. Crucially, each of these functionalities reduces its respective problems to instances of Weighted Model Counting (WMC) or Maximum Weighted Model Counting (MWMC), enabling Quokka# to leverage powerful SAT-based solvers for practical quantum circuit analysis and synthesis. This chapter describes the design of Quokka# and how its different functionalities can be used.

8.1 Design of Quokka#

The design of the Quokka# is shown in Figure 8.1, displaying a high-level sketch for the simulation, equivalence checking, and synthesis modules.

In all modules, Quokka# takes as input a circuit U (and V) in QASM format [38]. The circuit encoder then transforms the QASM circuit to a Boolean formula with weights on literals as explained in Section 2.3. In a post-processing phase, Quokka# takes extra inputs to augment the encoding depending on the functionality, as will be elaborated in Section 8.2.

Finally, the encodings are translated into weighted CNF (wCNF) in (weighted) DIMACS format [1] via the Python library SymPy and given to an (M)WMC solver. From the solver's output, the results are concluded either directly (for simulation and equivalence checking) or after a simple post-processing (for synthesis).

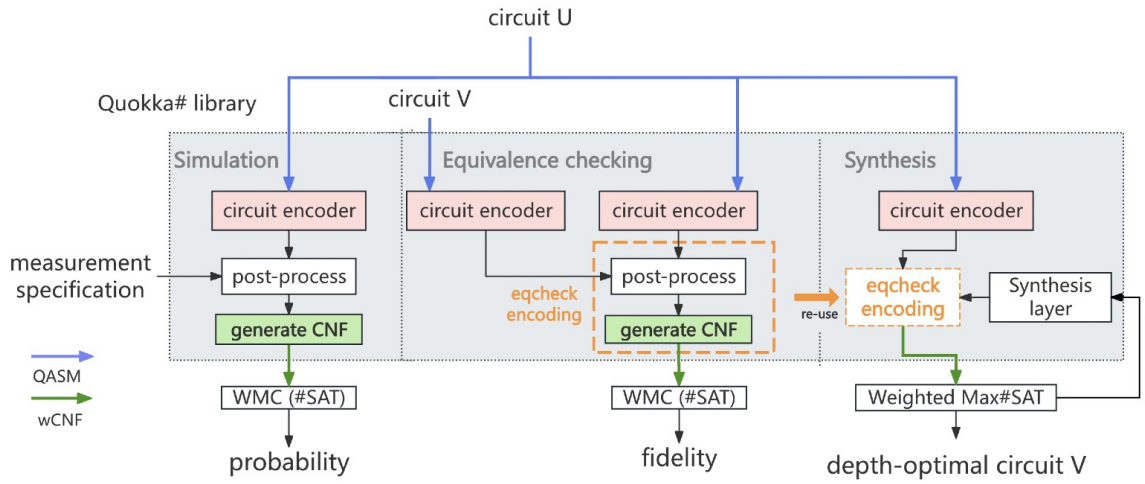


Figure 8.1: The architecture of `Quokka#`.

8.2 Usage of `Quokka#`

This section gives an overview of how to use the `Quokka#` library for simulation, equivalence checking, and exact and approximate synthesis.

`Quokka#` is implemented as a Python library and is publicly available through Python's `pip` package manager under the name 'quokka-sharp'. The full source code, along with additional resources such as benchmark datasets, experiment scripts, and comprehensive usage instructions, can be found on on GitHub[1].

To use `Quokka#`, a WMC solver is required for all functionalities, except for synthesis, which requires an MWMC solver. Since the computational basis encoding involves complex numbers, and no existing tool supports them, we extended the WMC solver `GPMC`[2] [54] and the MWMC solver `d4Max`[3] [57] for this cause. Depending on the functionality being used, either a WMC or an MWMC solver is a prerequisite for running `Quokka#`. A configuration file must be provided to specify the paths to the required solvers.

In Figure 8.2, we show how to import `Quokka#` and demonstrate how to invoke each functionality. In all invocations, the first parameters are the circuits, provided as paths to QASM files, along with the basis in which to encode them (`"comp"` for **CB** and `"pauli"` for **PB**). For each functionality, the additional inputs and the corresponding outputs are detailed as follows.

Simulation. The simulation functionality is invoked using the function `sim`, as shown in Line 3. It returns the probability of measuring a specific outcome, specified via the `measurement` parameter. The `measurement` can be one of the following: `"allzero"` (all qubits in the $|0\rangle$ state), `"firstzero"` (the first qubit in the $|0\rangle$ state), or a dictionary mapping qubit indices to their expected values, representing a computational basis state on a subset or all of the qubits.

[1] Available at `https://github.com/System-Verification-Lab/Quokka-Sharp`
[2] Available at `https://github.com/System-Verification-Lab/GPMC`.
[3] Available at `https://github.com/crillab/d4v2`.

```
1 import quokka_sharp.functionalities as qk_funcs
2
3 probability = qk_funcs.sim("path/to/circ.qasm", "comp",
                                    measurement="allzero")
4 are_equal = qk_funcs.eq("path/to/circ.qasm",
                                    "path/to/alternative_circ.qasm",
                                    "pauli", check="liniar", N=16)
5 syn_results = qk_funcs.syn("path/to/circ.qasm", "pauli",
                                    cyc_lin_encoding=True, fid=1)
```

Figure 8.2: Importing `Quokka#` and examples invoking its functionalities on circuits defined in QASM format in files `path/to/circ.qasm` and `path/to/alternative_circ.qasm`.

Equivalence Checking. The equivalence checking functionality is invoked using the function `eq`, as demonstrated in Line 4. The function returns a Boolean value: `True` if and only if the two circuits are equivalent. The `check` parameter specifies the encoding used for equality checking and can be set to `"cyclic"`, `"linear"`, or `"cyclic-linear"`. Note that with **CB**, only `"cyclic"` can be used.

Synthesis. The synthesis functionality is invoked using the function `syn`, as demonstrated in Line 5. The function returns a tuple with four elements: (1) the synthesis outcome, which can be `"FOUND"`, `"TIMEOUT"`, or `"CRASH"`; (2) the weight of the best synthesized circuit; (3) a QASM representation of the synthesized circuit; and (4) the number of layers in the synthesized circuit. The user can specify the fidelity threshold via the `fid` parameter and, if using the Pauli basis, enable the cyclic-linear encoding with the `cyc_lin_encoding` parameter.

8.3 Repository Structure

The main folders and files in the codebase that users can test, modify, and expand upon to enhance the tool are as follows:

- `experiment` folder: containing the source code to test the tool and experiment with it.
  - `benchmark` folder: containing multiple benchmarks to test on, including the folder `modifications` with scripts to create the benchmark modifications.
  - `run_benchmarks` folder: containing files for running experiments, such as scripts to create the results in this work.
- `quokka_sharp` folder: containing the source code of the tool.
  - `encoding` folder: containing files defining core classes to encode quantum states, gates, and circuits.
  - `eq.py`, `sim.py`, `syn.py` files: implement the functionalities of equivalence checking, simulation, and synthesis, respectively.
  - `functionalities.py` file: contains a function for each of the functionalities, defining the user interface for the tool.

8.3.1 Encoding optimization

While the encoding in Equation 4.8 allows for any circuit of depth d, many encoded circuits are redundant as they fall in the same equivalence class. For example, two consecutive Hadamard gates can be reduced to the identity. So if the H is applied to the j-th qubit at depth t, we can safely exclude the case where another H is applied to the same qubit at depth $t+1$. In the encoding, this corresponds to enforcing that if $p^t_{H_j}$ is set to 1, then $p^{t+1}_{H_j}$ must be 0. We apply similar reasoning to patterns like $T^8 = I$ and $CX_{i,j}CX_{i,j} = I^{\otimes 2}$, which are never part of an optimal circuit. Eliminating such cases does not affect optimality and helps reduce the search space. The corresponding rules are as follows:

- **Rule 1**: No two H gates in a row on the same qubit, since $HH = I$:

$$F^d_{R1}(P) = \bigwedge_{k\in[d-1]} \bigwedge_{i\in[n]} (\overline{p}^k_{H,i} \vee \overline{p}^{k+1}_{H,i})$$

- **Rule 2**: No 8 T gates in a row on the same qubit since $T^8 = I$:

$$F^d_{R2}(P) = \bigwedge_{k\in[d-7]} \bigwedge_{i\in[n]} \bigvee_{j\in[8]} \overline{p}^{k+j}_{T,i}$$

- **Rule 3**: No two CX gates in a row on the same qubits since $CX_{i,j} \cdot CX_{i,j} = I$:

$$F^d_{R3}(P) = \bigwedge_{k\in[d-1]} \bigwedge_{i,j\in[n],j\neq i} (\overline{p}^k_{CX,i,j} \vee \overline{p}^{k+1}_{CX,i,j})$$

In addition to the constraints in the rules above, we aim to have a canonical representation for a given set of gates. Our guideline is that every non-I is pushed forward as far as possible. This means not allowing any single qubit gate to follow an I gate, other than I itself. For two-qubit gates, we do not allow following I on both qubits. The corresponding rules are as follows:

- **Rule 4**: No single qubit gates other than I after an I gate:

$$F^d_{R4}(P) = \bigwedge_{k\in[d-1]} \bigwedge_{i\in[n]} (p^k_{I,i} \Rightarrow (p^{k+1}_{I,i} \vee \bigvee_{j\in[n],j\neq i} (p^{k+1}_{CX,i,j} \vee p^{k+1}_{CX,j,i})))$$

- **Rule 5**: No CX gate following I gate on both qubits:

$$F^d_{R5}(P) = \bigwedge_{k\in[d-1]} \bigwedge_{i,j\in[n],j\neq i} (p^{k+1}_{CX,i,j} \Rightarrow (\overline{p}^k_{I,i} \vee \overline{p}^k_{I,j}))$$

Chapter 9

Related Work

In this chapter, we provide an overview of related work on classical simulation, equivalence checking, and synthesis in quantum computing, with a focus on methods that employ SAT-based solvers. Table 9.1 summarizes the functionalities supported by `Quokka#` and compares them to those offered by other tools.

Table 9.1 displays seven different functionalities. Of those, simulation, exact and approximate equivalence checking, and exact and approximate synthesis are all defined and discussed in depth throughout this thesis. The two remaining functionalities, partial equivalence and Hoare verification, are included in the table to provide a comprehensive comparison of tool capabilities, but are not covered in detail in this work.

Partial equivalence, as formally defined in [32], refers to a relaxed notion of circuit equivalence: two circuits are partially equivalent if, for every possible input state, they yield the same probability distribution over measurement outcomes. Consequently, partial equivalence allows phase differences to vary across different input-output combinations. Moreover, because measurements may be restricted to a subset of qubits, differences on unmeasured qubits can be ignored, and the circuits may even differ in the total number of qubits.

Hoare verification refers to a method of program correctness checking based on *Hoare triples* of the form $\{P\}\ C\ \{Q\}$, where P is a precondition, C is a program (or circuit, in our context), and Q is a postcondition. This asserts that if P holds before executing C, then Q will hold afterward. In quantum circuit verification [125], this approach is adapted to reason about how certain input conditions (often on individual qubits or subspaces) affect the outputs, allowing one to verify correctness properties of quantum programs in a structured, modular way.

We can see from Table 9.1 that `Quokka#` offers a much more varied tool set than most, and is the only one to support universal optimal synthesis that can be both exact and approximate. We see in Chapter 10, how the performances of the tools compare.

9.1 Simulation and Equivalence Checking

SAT-Based. SAT-based solvers have proven successful in navigating the huge search spaces encountered in various problems in quantum computing [119, 84]. For instance, [16] implements a simulator for Clifford circuits based on a SAT encoding (our encoding of H, S, CX in

Tool	Quokka#	MQT	SliQSim	SliQEC	Quasimodo	AutoQ	QuiZX	Synthetiq	MITMS
Simulation	✓	✓ [127]	✓ [112]	×	✓ [107]	✓ [34]	✓ [64]	×	×
Exact Eq. Check.	✓	✓ [25]	×	✓ [118]	✓ [106]	✓ [34]	×	×	×
Approx. Eq. Check.	✓	✓ [23]	×	✓ [118]	×	×	×	×	×
Partial Eq. Check.	×	✓ [24]	×	✓ [32]	×	×	×	×	×
Hoare Verification	□	×	×	×	×	✓ [34]	×	×	×
Exact Synthesis	✓	◯ [103]	×	×	×	×	×	△ [89]	✓ [7]
Approx. Synthesis	✓	×	×	×	×	×	×	△ [89]	×

Table 9.1: Functionalities supported by Quokka# compared with other related state-of-the-art tools, where ✓ (resp. ×) denotes the tool and functionality combination is (resp. is not) supported. MQT only supports optimal Clifford circuit synthesis, not universal, which is denoted by ◯. Synthetiq supports non-optimal (exact and approximate) synthesis, which is denoted by △. While both Quokka# and AutoQ support Hoare-style verification, Quokka# imposes stricter limitations on the expressiveness of pre- and postconditions, denoted by □.

Table 2.1 is similar to theirs). The authors also discuss a SAT encoding for universal quantum circuits, which, however, requires exponentially large representations, making it impractical. Berent et al. [16] realize a Clifford circuit simulator and equivalence checker based on a SAT encoding. The equivalence checker was superseded by the deterministic polynomial-time algorithm proposed and implemented in [110]. Amy [5] uses path integrals to check the equivalence of circuits, which is complete for Clifford circuits and can prove the equivalence of Clifford+T and Clifford+R circuits.

Decision Diagram-Based Methods. Another method is based on decision diagrams (DDs) [3, 22], which represent many Boolean functions succinctly, while allowing manipulation operations without decompression. DD methods for pseudo-Boolean functions include Algebraic DDs (ADD) [12, 35, 114] and various "edge-valued" ADDs [72, 109, 120, 102]. The application of DDs to quantum circuit simulation, by viewing a quantum state as a pseudo-Boolean function, was pioneered with QuiDDs [115] and further developed with Quantum Multi-valued DDs [85], Tensor DDs [56] and CFLOBDDs [106]. All but CFLOBDD are essentially ADDs with complex numbers. They have also been used for checking the equivalence [25] and synthesis [126] of quantum circuits. Jimenez et al. use bisimulation for circuit reduction, reducing simulation time compared to DDs in some cases [59].

Abstract Interpretation and Stabilizers. Yu and Palsberg [123] use an abstract interpretation to simulate and automatically verify quantum circuits. Abstraqt [18] defines a different abstract interpretation using the stabilizer basis. SAT solvers have proven successful in quantum compilation [111], e.g., for reversible simulation of circuits [119] and optimizing space requirements of quantum circuits [84, 96].

ZX-Calculus. Another way is to translate quantum circuits into ZX-diagrams [36], which is a graphical calculus for quantum circuits equipped with powerful rewrite rules. It offers a diagrammatic approach to manipulate and analyze quantum circuits. A circuit is almost trivially expressible as a diagram, but the diagram language is more powerful and circuit extraction

is consequently #P-complete [40]. It has proven enormously successful in applications from equivalence checking [91, 92], to circuit optimization [63] and simulation [64].

Verification. Classical simulation is commonly used for the verification of quantum circuits, with extensive research focused on their equivalence checking [117, 9, 8]. It can also be applied in bug hunting in quantum circuits. In [34], the authors proposed a tree automaton to compactly represent quantum states and gates algebraically, framing the verification problem as a Hoare triple.

9.2 Synthesis

Clifford Circuit Synthesis. Synthesis of Clifford circuits is substantially simpler than that of universal quantum circuits due to the ability to exploit the algebraic structure of the symplectic group, which significantly constrains the search space. Any Clifford operation on n qubits can be efficiently represented by a $2n \times 2n$ symplectic matrix, rather than the exponentially larger $2^n \times 2^n$ unitary matrix typically required for general quantum operations [27]. Building on this, Maslov and Roetteler [78] employ the Bruhat decomposition of the symplectic group to generate shorter Clifford circuits. Similarly, Rengaswamy et al. [99] develop Clifford synthesis algorithms via symplectic geometry, targeting logical-level Clifford operations with an emphasis on practical implementations on physical qubits. While these approaches produce efficient Clifford circuits, they do not guarantee optimality in terms of depth or gate count. To address this limitation, Schneider et al. [103] reformulate Clifford synthesis as a satisfiability problem. This encoding enables the use of SAT and MaxSAT solvers to identify optimal Clifford circuits for a fixed depth, providing a rigorous method for achieving minimality.

Exact Clifford+T Circuit Synthesis In error-corrected quantum computing, the relevant universal gate set is Clifford+T. There are many works considering the exact synthesis of quantum circuits in the gate set Clifford+T[7, 79, 49, 50, 68, 87], i.e., the desired specification is realized without any rounding errors. Approaches like [49, 87] synthesize the unitary matrix representing the specification in a local fashion, i.e., column by column. However, they do not give the optimal solution and leave significant room for improvement. To achieve optimality, the meet-in-the-middle algorithm [79, 50] performs an exhaustive search over the space of all Clifford+T circuits up to a given depth.

Approximate Clifford+T Circuit Synthesis. Since not all unitaries can be implemented exactly in the Clifford+T gate set, other works have focused on synthesizing circuits under different approximation metrics. Most of these works [69, 100, 104, 45] focus on single-qubit operators, especially rotation gates, while [48, 89] consider multi-qubit operators.

Chapter 10

Experimental Evaluation

In this chapter, we evaluate the performance of our approach as implemented in `Quokka#`, focusing on its scalability and effectiveness across the core functionalities of simulation, equivalence checking, and synthesis. Our goal is to understand how different encoding strategies and design choices affect overall performance, and to identify the strengths and limitations of the approach in practical settings.

For each functionality, we begin by evaluating our proposed encodings on randomly generated circuits to establish a baseline for understanding their relative behavior in a controlled setting. We then proceed to compare them with existing state-of-the-art approaches on established benchmarks, in order to assess their practical effectiveness and competitiveness in real-world scenarios.

We first compare the two supported basis representations, the Pauli basis ($\mathbf{PB}$) and the computational basis ($\mathbf{CB}$), using simulation on random circuits. This offers initial insight into the practical trade-offs between different encoding choices. We then examine the performance of our equivalence checking methods across all proposed encodings: linear, cyclic, and linear-cyclic encodings in the $\mathbf{PB}$, as well as the cyclic encoding in the $\mathbf{CB}$. This analysis highlights the computational cost associated with each representation strategy. Next, we assess exact synthesis performance across these encodings, again using random circuits to isolate the effect of the encoding independent of external complexity.

After establishing these baseline results, we compare our approaches against state-of-the-art tools for each functionality. These evaluations are carried out using circuits from the MQTBench benchmark suite [95]. Additionally, we explore the impact of different weighted model counters to better understand their role in optimizing circuit-level tasks.

Through this layered analysis, we aim to provide a comprehensive picture of how well the maximum weighted model counting supports the core tasks of quantum circuit synthesis and where further improvements might be most impactful.

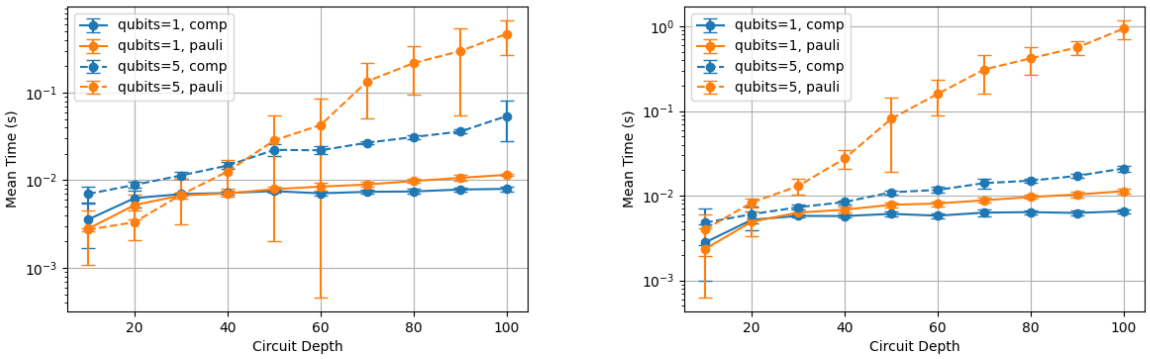
Random Circuit Generation. For the experiments, we generate random circuits for a given number of qubits n and a specified depth d, using a predefined probability distribution over the gate set $\mathcal{G}_{CST} = \{CX, H, S, S^\dagger, T, T^\dagger\}$. Each layer is constructed by iteratively selecting gates according to the distribution and assigning them to randomly chosen unassigned qubits. The two-qubit gate CX is only selected when at least two unassigned qubits remain.

This process continues until all qubits in the layer are assigned a gate — no identity (I) gates are inserted, and every qubit receives exactly one gate per layer.

Model Counters. For simulation and equivalence checking, a WMC is required, while synthesis relies on an MWMC. We use `GPMC` and `d4Max` for these purposes, respectively, in the random circuit experiments. Both tools were extended to support complex numbers, as discussed in Section 8.2.

10.1 Simulation

To compare the different basis encodings, we generate random circuits with 1 and 5 qubits and depths ranging from 5 to 50 in increments of 5, using a uniform probability distribution over the gate set $\mathcal{G}_{CST}$. We conduct simulations using both the **PB** and the **CB**, evaluating performance under two projectors: (1) *First Zero* — measuring the probability that the first qubit of the output state is in the $|0\rangle$ state, and (2) *All Zero* — measuring the probability that the entire output state is $|0\ldots0\rangle$. The results appear in Figure 10.1. The data shows that **CB** scales better than **PB**, outperforming it as circuit depth and qubit count increase. Notably, this holds even with the First Zero projector, which is more challenging for **CB**, as explained in Section 7.1.



(a) First Zero Projector (b) All Zero Projector

Figure 10.1: Runtime as a function of circuit depth for Simulation of circuits with 1 and 5 qubits with First Zero and All Zero projectors.

One aspect of the circuits that we believe influences performance is their gate composition. As discussed in Section 7.1, the **PB** is presumed to be sensitive to the presence of T gates, while the **CB** is affected by H gates. To investigate this, we conducted additional experiments on random circuits generated with varying probabilities of applying H versus T gates. Specifically, for a ratio r, the probability of applying a T or $T^\dagger$ gate is $2*r$ times the probability of applying an H gate, such that for $r = 0.5$, this corresponds to a uniform probability distribution over the $\mathcal{G}_{CST}$ gate set. The results of these experiments are shown in Figure 10.2. We observe that the performance of the **CB** deteriorates as the proportion of T gates increases, while the **PB** exhibits a less consistent behavior, with performance surprisingly improving for $r > 0.5$.

This irregularity can be explained by the structure of the Pauli basis. A T gate only causes a branching in satisfying assignments when the X component is active. However, since we use the all-zero measurement projector, the initial state is represented using only I and Z terms — meaning the X component is inactive at the outset. It is the H gate that introduces X terms into the state. Therefore, the impact of T gates in the **PB** becomes significant only when preceded or accompanied by a sufficient number of H gates. As a result, the most challenging configurations arise when both H and T gates are present in balanced quantities (e.g., $r = 0.5$). In contrast, when the ratio skews heavily toward T gates with fewer H gates to activate X components, the T gates become less disruptive, leading to a slight improvement in performance.

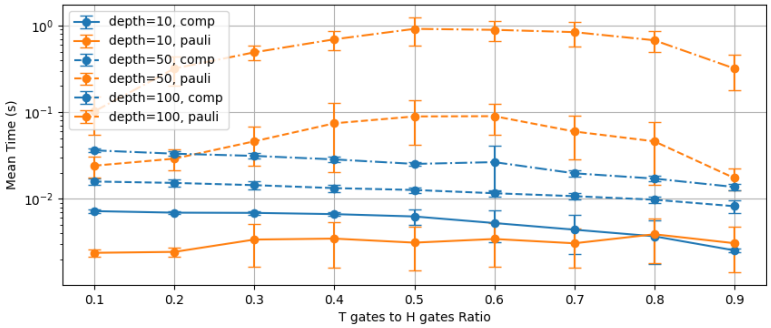


Figure 10.2: Runtime of simulation with All Zero measurements as a function of T gates to H gate probability ratio in random circuits with 5 qubits.

Comparison with State of the Art.

We empirically evaluated `Quokka#` on the MQTBench benchmark with the All Zero projector, as is typical in most quantum algorithms. We compare `Quokka#`, using both **CB** and **PB**, with two state-of-the-art tools for simulation: `QuiZX` [64] based on ZX calculus [36] and `Quasimodo` [107] based on CFLOBDD [106]. We present a representative subset of results in Table 10.3. Across all cases, `Quokka#` with the computational basis consistently outperforms `Quasimodo`. Moreover, `Quokka#` **CB** also outperforms both `QuiZX` and `Quokka#` **PB** in most scenarios. When it does, the performance gains are often substantial; in the few cases where it does not, the runtime differences are relatively minor.

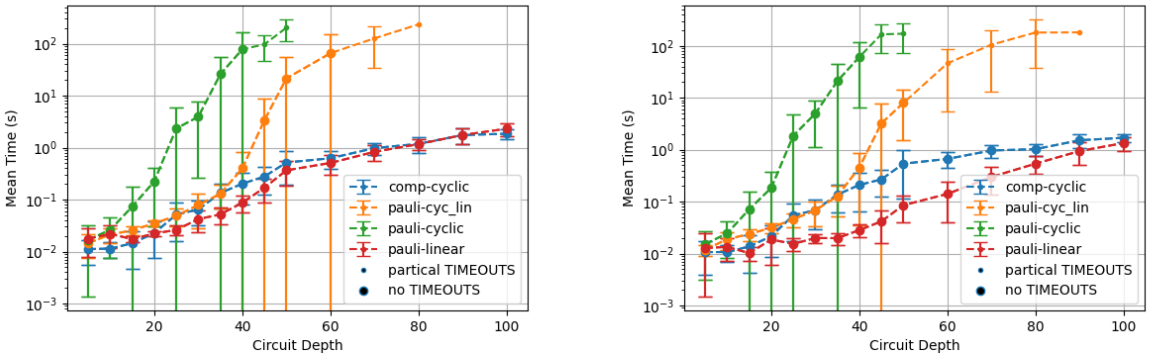
10.2 Equivalence Checking

As proposed in Theorem 4.1 and Theorem 6.2, there are multiple techniques to compare the equivalence of two circuits. Before proceeding with synthesis testing, we aim to gain a better understanding of the performance of equivalence checking on its own. To compare the performance of the different techniques, we run experiments on random circuits and their modified copies. Two types of modifications are performed. The first, referred to as *Optimized*, is an optimization of the circuit — equivalent to the original — generated using the Python library `PyZX` [63]. The second modification, referred to as *Gate Missing*, removes a random gate

Algorithm	n	Quokka# (**PB**)	Quokka# (**CB**)	QuiZX	Quasimodo
Grover's (noancilla)	5	8.626	0.025	**0.019**	0.104
	6	> 300	**0.052**	0.495	0.25
	7	> 300	**0.089**	4.345	0.517
QAOA	7	0.15	**0.025**	0.026	0.087
	9	1.356	0.034	**0.031**	0.05
	11	1.086	0.038	**0.033**	0.06
QFT	16	0.067	**0.026**	> 300	7.826
	32	0.261	**0.069**	> 300	> 300
	64	**0.11**	0.15	0.57	> 300
QNN	16		**51.83**		57.36
	32	> 300	> 300	✕	> 300
	64		> 300		> 300
VQE	5	0.059	**0.022**	1.005	7.368
	10	0.119	**0.030**	0.031	9.905
	15	0.191	**0.036**	> 300	40.233
W-state	16	0.107	**0.023**	33.619	15.241
	32	0.279	**0.032**	32.947	> 300
	64	0.878	**0.041**	33.552	> 300

Table 10.3: Simulation runtime (sec) comparison for Quokka#, QuiZX, and Quasimodo. > 300 represents a timeout (5 min) and ✕ means that the result was 'unknown'.

from the Optimized circuit, ensuring that the resulting circuit is not equivalent to the original one. Figure 10.4 shows the performance of the different equivalence checking encodings on circuits compared with their modified counterparts. As with simulation, we observe that **CB** scales better than the **PB** encoding overall. That said, among the **PB** variants, the linear encoding performs significantly better than the others and is only outperformed by **CB** at large depths (around 100).



(a) Optimized Modification (b) Gate Missing Modification

Figure 10.4: Runtime as a function of circuit depth for equivalence checking of 5-qubit circuits against their modified counterparts. For data points with at least one timeout, the average excludes timeouts and is marked with a smaller dot.

Comparison with State of the Art.

We now compare the **PB** and **CB** encodings to a state-of-the-art tool that performs the same task. For the **PB**, we select the linear encoding, as it was the fastest based on the results from the random circuit experiments. The state-of-the-art reference is `QCEC`[26] from the Munich Quantum Toolkit (`MQT`), which integrates decision diagrams (DDs) and ZX-calculus. In `QCEC` (v2.2.4), the ZX-calculus can be disabled to yield a purely DD-based variant, which we denote as `QCEC (DD)`.

For consistency, we use the same benchmark suite, MQTBench[95], as in the simulation experiments to compare the two `Quokka#` encodings against both variants of `QCEC`. As with the random experiments, for each benchmark we construct a modified version to compare to. The modification here, referred to as *Phase shift*, takes the optimized circuit, generated by PyZX [63], and inserts a error of 10^{-4} to the rotation angle of a randomly chosen rotation gate. So, by construction, the circuits are not equivalent.

The performance on equivalence checking is shown in Table 10.5. Regarding running time, `Quokka#` is mostly faster than `QCEC` and `QCEC (DD)` for circuits that feature rotation gates, such as W-state and QFT. Overall, the linear **PB** encoding performs better than the **CB**, where the differences are significant (QAOA, QFT, VQE). This can be explained by the $2n$ separate calls of the linear **PB** to WMC, allowing it to stop when it finds a non-equivalent case, while the computational basis method always has to range over all basis states. However, it is worth noting that `Quokka#` is always more accurate than `QCEC` when it comes to phase shift error. `QCEC` reports many wrong results, and there are results of unknown due to the decision diagram method and the ZX-calculus method giving different results.

10.3 Synthesis

Exact Synthesis.

To evaluate the scalability and performance of our proposed synthesis method, we run the different encodings on random circuits with 2 to 6 qubits. The random circuits are generated with a uniform distribution over the target gate set $\mathcal{G}_{CT} = \{H, CX, T, T^\dagger\}$. Since a random circuit of depth d could be equivalent to a shallower circuit, we retain only those circuits that did not produce a shallower result when synthesized, until we obtain 10 samples. We synthesize those 10 samples, using each of the three encodings presented in Proposition 4.3 and Proposition 6.3, with a time limit of 300 seconds. We report the success rate, defined as the ratio of samples solved within the time limit, the average runtime of the solved samples, and the average memory usage for those runs as reported by `d4max`. For each encoding, we increase the circuit depth until the success rate drops below 0.5. The results are displayed in Table 10.6.

It is important to note that when the success rate is below 1, the reported run times and memory usage in the table are naturally lower than the average required to solve all cases, as failed instances, which would exceed the time limit, are excluded from the averages. Furthermore, since the maximum weight corresponding to a successful synthesis is known (as shown

Algorithm	n	Quokka# (**LPB**)	Quokka# (**CB**)	QCEC (DD)	QCEC
Grover's (noancilla)	5	24.37	11.8	0.12	**0.04**
	6	> 300	> 300	**6.04**	✗
	7	> 300	> 300	> 300	**1.97**
QAOA	7	0.15	0.04	**0.03**	✗
	9	**0.29**	283.41	wrong	✗
	11	0.33	> 300	0.12	**0.1**
QFT	2	0.02	**0.007**	0.01	0.01
	8	**0.2**	283.54	wrong	✗
	16	**0.79**	> 300	wrong	✗
QNN	2	0.04	**0.009**	wrong	wrong
	8	> 300	> 300	**0.24**	wrong
	16	> 300	> 300	> 300	✗
VQE	5	0.15	**0.06**	wrong	wrong
	10	**0.79**	19.94	wrong	wrong
	15	0.63	66.16	**0.15**	**0.15**
W-state	16	0.33	**0.07**	12.85	✗
	32	1.55	**0.17**	> 300	✗
	64	5.4	**0.55**	> 300	> 300

Table 10.5: Equivalence checking runtime (sec) comparison for Quokka# in the **CB** cyclic encoding and **PB** linear encoding (**LPB**) with QCEC in the regular mode and DD only mode (QCEC (DD)). > 300 represents a timeout and ✗ means that the result was 'unknown'.

in Theorem 4.1), Quokka# provides a threshold to the MWMC solver, allowing it to terminate as soon as the optimal value is reached. As a result, when a circuit can be synthesized within the given depth, i.e., in the successful cases used to produce the data in Table 10.6, the solver may terminate early. Conversely, in unsuccessful cases, where no circuit of the given depth exists, the solver must exhaustively explore the search space, resulting in longer run times. For example, for the cases with 3 qubits and depth 4, when running the same sample circuits as in Table 10.6 without the threshold using the linear-cyclic encoding in the **PB**, the average runtime is 30.865 ± 11.479 instead of 11.26 ± 4.95.

Approximate Synthesis.

Important gates that often need to be synthesized approximately in Clifford+T are the general rotation gates: R_x, R_y, R_z [69]. In our encoding, these gates correspond to fixed Boolean formulas, with the rotation angle only influencing the weight function [81]. Thus, different rotation angles do not significantly affect the performance per depth of the approximate synthesis. To demonstrate the use of approximate synthesis, we consider the rotation gate

$$R_z\left(\frac{\pi}{8}\right) = \begin{pmatrix} e^{-i\pi/16} & 0 \\ 0 & e^{i\pi/16} \end{pmatrix}.$$

We choose to use the **CB** with the cyclic encoding as it outperforms the **PB** with the cyclic encoding, according to the results in Table 10.6 and Figure 10.4 (recall that **PB** with the linear-cyclic encoding cannot perform *approximate* synthesis).

#Qb	Depth	Rate			Time (s)			Memory (GB)		
		CB	**LPB**	**PB**	**CB**	**LPB**	**PB**	**CB**	**LPB**	**PB**
2	1	1.0	1.0	1.0	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	< 2	< 2	< 2
	2	1.0	1.0	1.0	0.07 ± 0.00	0.08 ± 0.00	0.08 ± 0.00	< 2	< 2	< 2
	3	1.0	1.0	1.0	0.11 ± 0.01	0.11 ± 0.01	0.14 ± 0.02	< 2	< 2	< 2
	4	1.0	1.0	1.0	0.29 ± 0.09	0.22 ± 0.07	0.40 ± 0.10	< 2	< 2	< 2
	5	1.0	1.0	1.0	2.46 ± 1.51	1.46 ± 0.72	4.53 ± 3.16	2.13 ± 0.03	2.09 ± 0.00	2.11 ± 0.02
	6	1.0	1.0	1.0	21.25 ± 12.54	14.26 ± 10.33	55.69 ± 42.54	2.46 ± 0.28	2.16 ± 0.07	2.38 ± 0.27
	7	0.9	0.9	0.3	115.31 ± 64.99	97.28 ± 46.98	206.37 ± 17.36	3.84 ± 1.31	2.48 ± 0.21	3.41 ± 0.54
	8	0.0	0.0	∘	-	-	∘	-	-	∘
3	1	1.0	1.0	1.0	0.05 ± 0.00	0.05 ± 0.00	0.05 ± 0.00	< 2	< 2	< 2
	2	1.0	1.0	1.0	0.08 ± 0.01	0.10 ± 0.01	0.10 ± 0.02	< 2	< 2	< 2
	3	1.0	1.0	1.0	1.43 ± 0.97	0.44 ± 0.24	2.45 ± 1.91	2.11 ± 0.02	< 2	2.11 ± 0.01
	4	1.0	1.0	1.0	51.56 ± 57.11	11.26 ± 4.95	105.40 ± 64.98	3.15 ± 1.25	2.15 ± 0.04	2.87 ± 0.67
	5	0.1	0.7	0.0	145.96	147.37 ± 49.30	-	5.08	3.20 ± 0.53	-
	6	∘	0.0	∘	∘	-	∘	∘	-	∘
4	1	1.0	1.0	1.0	0.05 ± 0.00	0.06 ± 0.00	0.05 ± 0.00	< 2	< 2	< 2
	2	1.0	1.0	1.0	0.37 ± 0.20	0.30 ± 0.14	0.62 ± 0.48	2.09	< 2	2.09 ± 0.01
	3	1.0	1.0	0.5	102.55 ± 85.73	22.65 ± 16.03	85.70 ± 42.76	4.44 ± 2.07	2.26 ± 0.13	2.69 ± 0.34
	4	0.0	0.2	0.0	-	246.80 ± 21.38	-	-	4.51 ± 0.20	-
5	1	1.0	1.0	1.0	0.05 ± 0.00	0.06 ± 0.01	0.06 ± 0.00	< 2	< 2	< 2
	2	1.0	1.0	1.0	2.93 ± 2.17	3.01 ± 1.62	11.85 ± 11.15	2.14 ± 0.04	2.11 ± 0.02	2.18 ± 0.10
	3	0.0	0.1	0.0	-	243.71	-	-	4.67	-
6	1	1.0	1.0	1.0	0.05 ± 0.00	0.12 ± 0.06	0.06 ± 0.00	< 2	< 2	< 2
	2	0.9	1.0	0.7	70.49 ± 82.13	54.47 ± 44.75	72.71 ± 69.85	3.74 ± 2.03	2.69 ± 0.61	2.80 ± 0.81
	3	0.0	0.0	0.0	-	-	-	-	-	-

Table 10.6: Synthesis benchmarks for random circuits using the target gate set $\{CX, H, T, T^\dagger\}$. We report the success rate, average runtime, and average memory usage of the optimal circuit synthesized within 300 seconds. **CB** and **PB** denote the cyclic encoding in each basis; L**PB** refers to the linear-cyclic encoding in the **PB**. Memory usage is reported by `d4max`; for short runtimes where no value is reported, we estimate usage to be under 2GB and denote this with < 2. Averages are shown with $\pm$ standard deviation; if all 10 samples failed, the average is marked as **-**. Untested depths (due to low success rate in the previous depth) are marked with ∘.

We report statistics for each synthesis layer where an improvement in the achieved fidelity is observed. The results are presented in Figure 10.7. We also present the runtime of the MWMC for each layer in Figure 10.9. The corresponding output circuits are as follows:

- **1 Layer:** $\mathcal{C}_{out} = (T)$
- **10 Layer:** $\mathcal{C}_{out} = (T^\dagger, H, T^\dagger, H, T^\dagger, H, T^\dagger, H, T^\dagger, H)$
- **15 Layer:** $\mathcal{C}_{out} = (H, T^\dagger, H, T, H, T, H, T^\dagger, H, T, H, T, H, T^\dagger, H)$

Comparison with State of the Art.

As discussed in Chapter 9, there are numerous synthesis tools available, but they differ significantly in the specific tasks they address. Some focus only on a subset of quantum circuits — for example, restricting to single-qubit or Clifford-only circuits. In addition to differences in input types, there is also variation in the desired output. This can include targeting different gate sets, different optimality metrics, or, in the case of approximate synthesis, differing approximation metrics. Furthermore, some tools sacrifice optimality in favor of faster runtimes and better scalability.

depth	1	10	15	24
#variables	11	92	137	218
#clauses	37	382	577	928
#literals	90	960	1460	2360
#Selecting variables	4	40	60	96
fidelity	0.962	0.975	0.997	-
Time (s)	0.023	0.092	2.715	> 1543.246
mem (GB)	< 2	< 2	2.14	> 33.31

Figure 10.7: Synthesis performance for approximating $R_Z(\pi/8)$ using the cyclic encoding in **CB**. The data is given for each MWMC call, for incremental target depths, until the program crashes at 24 layers due to limited resources. Memory usage for short runtime is not given (< 2). Only iterations where the fidelity improved are shown.

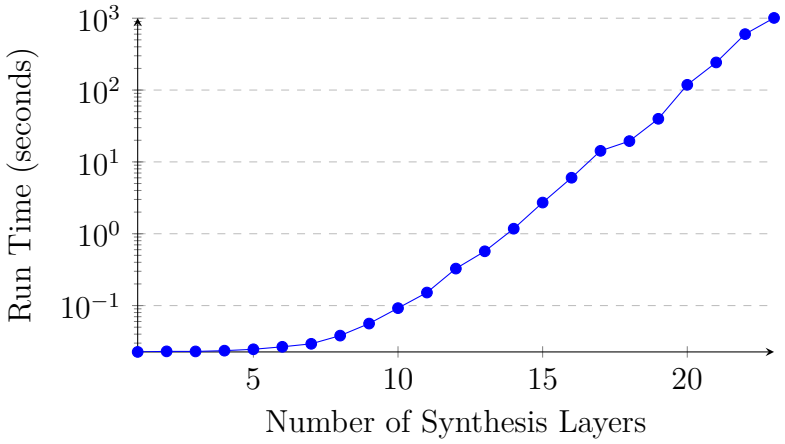


Figure 10.8

Figure 10.9: Runtimes as a function of target depth, for approximate synthesis of $R_Z(\pi/8)$, using the cyclic encoding in **CB**. The data is given for each MWMC call, for incremental target depths, until the program crashed at 24 layers, due to limited resources.

As a result, identifying a tool for comparison that performs exactly the same task — and thus allows for a fair evaluation — is not straightforward.

One tool that performs a task similar to the synthesis functionality in `Quokka#` is `MITMS`, which targets the gate set $\{H, CX, S, S^\dagger, T, T^\dagger\}$. However, we were unable to compile `MITMS` in our experimental setup due to its reliance on outdated libraries. Instead, we refer to the performance data reported in [7] as a comparative reference.

The results taken from [7], and the corresponding data form Table 10.6, are shown in Table 10.10. While the comparison is not entirely fair — due to differences in gate set, benchmark design, and platform — we observe that `Quokka#` uses more memory, but achieves lower run times, compared to `MITMS`.

#Qubits	2	3	4
Depth	Time (s)		
1	0.002	0.027	1.173
2	0.019	1.409	194.386
3	0.188	53.238	31,583.809
4	1.602	2311.023	-
5	12.433	85,055.065	-
6	84.622	-	-
Depth	Mem (MB)		
1	0.002	0.006	0.013
2	0.016	0.175	1.580
3	0.143	6.580	277.9256
4	0.990	210.907	-
5	6.103	7557.210	-
6	32.00	-	-

(a) `MITMS` results from [7]

#Qubits	2	3	4
Depth	Time (s)		
1	0.05 ± 0.00	0.05 ± 0.00	0.06 ± 0.00
2	0.08 ± 0.00	0.10 ± 0.01	0.30 ± 0.14
3	0.11 ± 0.01	0.44 ± 0.24	22.65 ± 16.03
4	0.22 ± 0.07	11.26 ± 4.95	(246.80 ± 21.38)
5	1.46 ± 0.72	(147.37 ± 49.30)	-
6	13.26 ± 10.33	-	-
7	(97.28 ± 46.98)	-	-
Depth	Mem (MB)		
1	< 2000	< 2000	< 2000
2	< 2000	< 2000	< 2000
3	< 2000	< 2000	2314.24 ± 133.12
4	< 2000	2201.60 ± 40.96	(4618.24 ± 204.8)
5	2140.16 ± 0.00	(3276.8 ± 542.72)	-
6	2211.84 ± 71.68	-	-
7	(2539.52 ± 215.04)	-	-

(b) corresponding Linear Pauli Basis results from Table 10.6. Where success rate is < 0.1 (timeout of 300 seconds) we denote the results with parentheses.

Table 10.10: Comparison of runtime and memory usage between (a) `MITMS` and (b) our synthesis results using the Linear Pauli Basis encoding.

Chapter 11

Conclusion

11.1 Summary

As quantum computing pushes toward demonstrating real-world utility, the ability to generate optimized, hardware-compatible quantum circuits becomes increasingly vital. Circuit synthesis — the process of converting a desired quantum behavior into a low-level gate sequence — is essential for bridging high-level quantum algorithms with the physical realities of quantum hardware. However, existing methods that guarantee optimality remain highly limited in scale.

Motivated by the recent successes of SAT-based methods in quantum simulation and verification, this thesis explored whether these symbolic techniques could also be used to synthesize quantum circuits. In particular, we focused on leveraging maximum weighted model counting (MWMC) to support exact and approximate depth-optimal synthesis over the Clifford+T universal gate sets.

We began by extending the recent SAT-based equivalence checking technique [82], adapting it to operate within a single call to a weighted model counter and further to compute the fidelity between quantum circuits. This allows us to support not only exact reasoning but also enable approximate equivalence checking by quantifying how similar two circuits are under fidelity. Crucially, we showed that cyclic encodings naturally support this functionality, as they directly compute the trace overlap between two circuits.

Building on these foundations, we formulated quantum circuit synthesis as a model counting problem, enabling us to construct candidate circuits and verify their correctness (or proximity) to a specification via a single MWMC call. We introduced a depth-bounded synthesis strategy that incrementally searches for optimal solutions by increasing the circuit depth. This synthesis framework supports both exact and approximate settings by tuning the required fidelity threshold.

In addition, we explored an alternative encoding basis, the computational basis ($\mathbf{CB}$), to the Pauli basis ($\mathbf{PB}$) used in [82], and analyzed their respective strengths and limitations. While the $\mathbf{PB}$ offers the multiplicative basis for streamlined equality checking encodings, we investigated whether the $\mathbf{CB}$ could provide advantages due to its smaller variable count. Our comparative analysis reveals trade-offs in expressiveness, overhead, and solver performance across tasks like simulation, equivalence checking, and synthesis.

These ideas were implemented in a prototype tool, `Quokka#`, which unifies SAT-based simulation, equivalence checking, and synthesis into a single framework. It supports both exact and approximate reasoning, allows toggling between encoding strategies, and demonstrates the practical viability of SAT-based synthesis in quantum computing.

11.2 Key Findings

The results of this thesis indicate that the answer to our central question — whether SAT-based techniques can be effectively adapted to quantum circuit synthesis — is a yes, with promising outcomes.

We extend prior work on model counting for equivalence checking to support one-shot equivalence, enabling effective integration with synthesis workflows. We also further adapt the encoding to allow for variations that compute fidelity, allowing for approximate equivalence checking and, by extension, approximate synthesis. By generalizing this equivalence checking process to also encompass gate selection and circuit structure, we formulate quantum synthesis as a Maximum-Weighted Model Counting (MWMC) problem that supports both exact and approximate fidelity objectives. To achieve depth optimality, we introduce a depth-bounded incremental search strategy that discovers minimal-depth circuits within the synthesis space. Additionally, we propose another encoding strategy, following the computational basis representation, that improves performance and scalability in many cases.

These techniques are embodied in our prototype tool, `Quokka#`, which demonstrates the feasibility and versatility of the proposed approach. `Quokka#` supports multiple core functionalities of quantum circuit design — including simulation, equivalence checking, and synthesis — within a single, unified symbolic framework. To our knowledge, it is the first tool to enable depth-optimal synthesis for both exact and approximate objectives. In evaluations, `Quokka#` significantly outperforms state-of-the-art tools in several benchmarks for simulation and synthesis. Moreover, in the domain of depth-optimal synthesis, its performance is competitive with the only existing tool that supports such guarantees, while uniquely offering fidelity-aware approximate synthesis with formal correctness bounds.

These findings not only demonstrate the technical viability of SAT-based synthesis but also highlight its potential for broader impact within the field of quantum computing. In particular, this work shows that SAT-based techniques can unify simulation, equivalence checking, and synthesis within a single formalism, offering flexibility, generality, and theoretical rigor. Although current synthesis scalability is limited to circuits of moderate size (from 1 qubit with depth ≤ 32 to 4 qubits with depth 4), this approach marks a shift toward symbolic, solver-driven workflows that complement traditional algebraic and heuristic methods. Moreover, it creates a promising feedback loop: advances in model counting — particularly in Maximum Weighted Model Counting (MWMC) — will directly enhance synthesis performance, while our framework, in turn, provides new motivation and use cases for improving MWMC techniques. This synergy offers a hopeful path toward greater performance and scalability.

Furthermore, the techniques developed here align with broader goals in quantum computing: enabling fault-tolerant execution, reducing circuit depth to mitigate decoherence, and adapting to hardware constraints. By supporting approximate synthesis, our methods can also aid in practical deployment where precision and resource constraints must be carefully balanced.

11.3 Future Directions

This thesis demonstrates that maximum weighted model counting (MWMC) is not only theoretically sound but also practically viable as a unifying formalism for quantum circuit simulation, equivalence checking, and synthesis. The success of this approach — realized in the `Quokka#` prototype — opens several promising avenues for future work, which we list below.

Solver-Level Innovations: For the evaluation of our methods, we rely on a prototype extension of `d4Max`, which currently lacks many of the optimizations found in state-of-the-art model counters. A critical next step is to design dedicated MWMC solvers with direct complex weight support and tailored heuristics. Incorporating incremental solving techniques — widely used in SAT-based bounded model checking [19, 51] and recently explored for sampling and weighted model counting [121, 122] — remains an open and unexplored direction for maximum weighted model counting. Adapting such techniques to the MWMC setting could significantly enhance synthesis performance by reusing solver state across depth-bounded iterations or layered circuit constructions. Similarly, exploiting problem structure in the CNF encodings (e.g., XOR clauses [83], repeated motifs, or circuit symmetries) could dramatically improve performance [70, 14]. Symmetry-aware inference and caching, already successful in other reasoning domains, could further accelerate solving in quantum contexts.

Synthesis Strategy and Encoding Refinements: At the synthesis level, there are opportunities to improve scalability and generality through enhanced encoding strategies. Our current depth-bounded incremental approach could be extended with partial reuse across layers or heuristic guidance from structural features of the specification. While this work focused on the Clifford+T gate set, the encoding naturally generalizes to other bases, including hardware-native sets. The method could also be adapted to optimize alternative metrics such as T-count or gate count, and support additional features, such as connectivity or ancilla usage. Specialized tasks — such as single-qubit rotation synthesis or Clifford-only compilation — could be isolated and encoded modularly to improve efficiency.

Integration into Hardware-Aware Compilation Pipelines: Embedding symbolic synthesis into a broader quantum compilation stack is another important frontier. This includes mapping to physical hardware topologies, respecting native gate sets, and integrating with error-correcting codes or logical-to-physical transpilation workflows. Depth-optimal synthesis is especially critical in fault-tolerant settings [60], where multi-qubit gates and T-gates are costly [69]. Continued refinement of these techniques may also enable the minimization of important cost metrics under physical constraints.

Scalability through Approximation and Learning: While exact MWMC-based synthesis remains challenging beyond a few qubits and layers, promising directions for scaling include approximate model counting, variational synthesis, stochastic SAT formulations, and learning-based guidance [76, 73] . For example, casting synthesis as a stochastic optimization problem could allow probabilistic balancing between competing objectives such as fidelity, depth, and gate cost. Learning-guided circuit prediction or abstraction-refinement loops could bridge symbolic synthesis with data-driven methods.

Benchmarking and Community Engagement: This thesis presents the first systematic application of MWMC-based quantum synthesis. The accompanying tool, `Quokka#`, provides a foundation for defining future benchmark sets and offers motivation for developing more capable MWMC solvers. This work encourages further progress in symbolic quantum compilation

and weighted model counting research by highlighting concrete synthesis challenges and exposing solver limitations. This feedback loop between application-driven problem instances and solver development is well-established in classical SAT/SMT — through venues such as the Model Counting Competition [1] — and may similarly benefit symbolic quantum compilation.

Theoretical and Foundational Implications: Beyond engineering, this work raises interesting questions at the interface of formal methods, quantum compilation, and theoretical computer science. For instance, it suggests new ways to derive lower bounds for quantum synthesis problems based on results from reasoning and complexity theory [13]. The encoding techniques and search paradigms developed here could also inform broader symbolic AI efforts, particularly those involving weighted reasoning, probabilistic inference, and constrained optimization.

In summary, this research lays the groundwork for a unified, SAT-based approach to quantum circuit design. By showing that simulation, equivalence checking, and synthesis can all be captured within a model counting framework, it opens the door to symbolic, rigorous, and scalable quantum compilers. As quantum hardware matures and solver technology advances, MWMC-based techniques may become a central component in the development of practical, high-performance quantum computing.

[1] `https://mccompetition.org/`

Bibliography

[1] Dimacs cnf. https://jix.github.io/varisat/manual/0.2.0/formats/dimacs.html. Accessed: 2025-01-27.

[2] Scott Aaronson. Quantum computing, postselection, and probabilistic polynomial-time. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 461(2063):3473–3482, 2005.

[3] Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978. doi:10.1109/TC.1978.1675141.

[4] Panos Aliferis, Daniel Gottesman, and John Preskill. Quantum accuracy threshold for concatenated distance-3 codes. *Quantum Info. Comput.*, 6(2):97–165, March 2006.

[5] Matthew Amy. Towards large-scale functional verification of universal quantum circuits. *arXiv:1805.06908*, 2018.

[6] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time t-depth optimization of clifford+ t circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

[7] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013. doi:10.1109/TCAD.2013.2244643.

[8] Ebrahim Ardeshir-Larijani, Simon J. Gay, and Rajagopal Nagarajan. Equivalence checking of quantum protocols. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 478–492, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[9] Ebrahim Ardeshir-Larijani, Simon J. Gay, and Rajagopal Nagarajan. Verification of concurrent quantum protocols by equivalence checking. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 500–514, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[10] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574:505âĂŞ510, 2019. URL: https://www.nature.com/articles/s41586-019-1666-5.

[11] Gilles Audemard, Jean-Marie Lagniez, and Marie Miceli. A New Exact Solver for (Weighted) Max#SAT. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:20, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2022.28, doi:10.4230/LIPIcs.SAT.2022.28.

[12] R. Iris Bahar, Erica A. Frohm, Charles M. Gaona, Gary D. Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 188–191, 1993.

[13] Max Bannach and Markus Hecher. On Weighted Maximum Model Counting: Complexity and Fragments. In *36th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2024, Herndon, Virginia, USA, October 28-30, 2024*, 2024.

[14] Anicet Bart, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. Symmetry-driven decision diagrams for knowledge compilation. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 51–56. IOS Press, 2014. doi:10.3233/978-1-61499-419-0-51.

[15] Charles H Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. *Theoretical computer science*, 560:7–11, 2014.

[16] Lucas Berent, Lukas Burgholzer, and Robert Wille. Towards a SAT Encoding for Quantum Circuits: A Journey From Classical Circuits to Clifford Circuits and Beyond. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing (SAT 2022)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:17, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. URL: https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2022.18, doi:10.4230/LIPIcs.SAT.2022.18.

[17] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20, 1993.

[18] Benjamin Bichsel, Anouk Paradis, Maximilian Baader, and Martin Vechev. Abstraqt: Analysis of quantum circuits via abstract stabilizer simulation. *Quantum*, 7:1185, 2023.

[19] Armin Biere. Bounded model checking. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 739–764. IOS Press, 2021. doi:10.3233/FAIA201002.

[20] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*. IOS Press, 2021.

[21] Armin Biere, Matti Järvisalo, and Benjamin Kiesl. Preprocessing in sat solving. In *Handbook of Satisfiability*, pages 391–435. IOS press, 2021.

[22] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, 35(8):677–691, 1986.

[23] Lukas Burgholzer, Richard Kueng, and Robert Wille. Random stimuli generation for the verification of quantum circuits. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 767–772, 2021.

[24] Lukas Burgholzer, Tom Peham, and Robert Wille. MQT QCEC: Equivalence checking for quantum circuits, 2020. https://arxiv.org/abs/2004.08420. URL: https://mqt.readthedocs.io/projects/qcec, doi:10.1109/TCAD.2020.3032630.

[25] Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(9):1810–1824, sep 2020. URL: http://dx.doi.org/10.1109/TCAD.2020.3032630, doi:10.1109/tcad.2020.3032630.

[26] Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(9):1810âĂŞ1824, September 2021. URL: http://dx.doi.org/10.1109/TCAD.2020.3032630, doi:10.1109/tcad.2020.3032630.

[27] A. R. Calderbank, E. M. Rains, P. W. Shor, and N. J. A. Sloane. Quantum error correction and orthogonal geometry. *Phys. Rev. Lett.*, 78:405–408, Jan 1997. URL: https://link.aps.org/doi/10.1103/PhysRevLett.78.405, doi:10.1103/PhysRevLett.78.405.

[28] A. Robert Calderbank and Peter W. Shor. Good quantum error-correcting codes exist. *Phys. Rev. A*, 54:1098–1105, Aug 1996. URL: https://link.aps.org/doi/10.1103/PhysRevA.54.1098, doi:10.1103/PhysRevA.54.1098.

[29] Yudong Cao, Jonathan Romero, Jonathan P Olson, Matthias Degroote, Peter D Johnson, Mária Kieferová, Ian D Kivlichan, Tim Menke, Borja Peropadre, Nicolas PD Sawaya, et al. Quantum chemistry in the age of quantum computing. *Chemical reviews*, 119(19):10856–10915, 2019.

[30] Supratik Chakraborty, Dror Fried, Kuldeep S Meel, and Moshe Y Vardi. From weighted to unweighted model counting. In *Proceedings of IJCAI*, pages 689–695, 2015.

[31] Senrui Chen, Sisi Zhou, Alireza Seif, and Liang Jiang. Quantum advantages for pauli channel estimation. *Physical Review A*, 2021. URL: https://api.semanticscholar.org/CorpusID:237213441.

[32] Tian-Fu Chen, Jie-Hong R Jiang, and Min-Hsiu Hsieh. Partial equivalence checking of quantum circuits. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 594–604. IEEE, 2022.

[33] Yanlin Chen, Yilei Chen, Rajendra Kumar, Subhasree Patro, and Florian Speelman. Qseth strikes again: finer quantum lower bounds for lattice problem, strong simulation, hitting set problem, and more. *arXiv preprint arXiv:2309.16431*, 2023.

[34] Yu-Fang Chen, Kai-Min Chung, Ondřej Lengál, Jyun-Ao Lin, Wei-Lun Tsai, and Di-De Yen. An automata-based framework for verification and bug hunting in quantum circuits. *Proc. ACM Program. Lang.*, 7(PLDI), jun 2023. doi:10.1145/3591270.

[35] Edmund M. Clarke, Kenneth L. McMillan, Xudong Zhao, Masahiro Fujita, and Jerry Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proceedings of the 30th international Design Automation Conference*, pages 54–60, 1993.

[36] Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

[37] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons, and Seyon Sivarajah. On the qubit routing problem. *arXiv preprint arXiv:1902.08091*, 2019.

[38] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, Lev S. Bishop, Steven Heidel, Colm A. Ryan, Prasahnt Sivarajah, John Smolin, Jay M. Gambetta, and Blake R. Johnson. OpenQASM3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3):1–50, September 2022. URL: http://dx.doi.org/10.1145/3505636, doi:10.1145/3505636.

[39] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quantum Inf. Comput.*, 6(1):81–95, 2006. doi:10.26421/QIC6.1-6.

[40] Niel de Beaudrap, Aleks Kissinger, and John van de Wetering. Circuit extraction for ZX-diagrams can be #P-hard. In *ICALP 2022*. Schloss Dagstuhl - Leibniz-Zentrum fÃijr Informatik, 2022. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ICALP.2022.119, doi:10.4230/LIPICS.ICALP.2022.119.

[41] M. Van den Nest. Classical simulation of quantum computation, the gottesman-knill theorem, and slightly beyond. *arXiv:0811.0898*, 2008.

[42] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[43] Johannes K. Fichte, Markus Hecher, and Stefan Szeider. A time leap challenge for sat solving, 2023. URL: https://arxiv.org/abs/2008.02215, arXiv:2008.02215.

[44] Fulvio Flamini, Nicolo Spagnolo, and Fabio Sciarrino. Photonic quantum information processing: a review. *Reports on Progress in Physics*, 82(1):016001, 2018.

[45] Austin G. Fowler. Constructing arbitrary steane code single logical qubit fault-tolerant gates. *Quantum Inf. Comput.*, 11(9&10):867–873, sep 2011. doi:10.26421/QIC11.9-10-10.

[46] 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.

[47] Simon J. Gay. Stabilizer states as a basis for density matrices. *CoRR*, abs/1112.2156, 2011. URL: http://arxiv.org/abs/1112.2156, arXiv:1112.2156.

[48] Vlad Gheorghiu, Michele Mosca, and Priyanka Mukhopadhyay. T-count and t-depth of any multi-qubit unitary. *npj Quantum Information*, 8(1), November 2022. URL: http://dx.doi.org/10.1038/s41534-022-00651-y, doi:10.1038/s41534-022-00651-y.

[49] Brett Giles and Peter Selinger. Exact synthesis of multiqubit clifford+ t circuits. *Physical Review A*, 87(3):032332, 2013.

[50] David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. An algorithm for the t-count, 2013. URL: https://arxiv.org/abs/1308.4134, arXiv:1308.4134.

[51] Henning Günther and Georg Weissenbacher. Incremental bounded software model checking. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 40–47. ACM, 2014. doi:10.1145/2632362.2632374.

[52] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modelling and Computation*, 6(4):245–262, 2010.

[53] Matthew P Harrigan, Kevin J Sung, Matthew Neeley, Kevin J Satzinger, Frank Arute, Kunal Arya, Juan Atalaya, Joseph C Bardin, Rami Barends, Sergio Boixo, et al. Quantum approximate optimization of non-planar graph problems on a planar superconducting processor. *Nature Physics*, 17(3):332–336, 2021.

[54] Kenji Hashimoto. GPMC. https://git.trs.css.i.nagoya-u.ac.jp/k-hasimt/GPMC, 2020.

[55] Loïc Henriet, Lucas Beguin, Adrien Signoles, Thierry Lahaye, Antoine Browaeys, Georges-Olivier Reymond, and Christophe Jurczak. Quantum computing with neutral atoms. *Quantum*, 4:327, 2020.

[56] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying. A tensor network based decision diagram for representation of quantum circuits. *ACM Trans. Design Autom. Electr. Syst.*, 27(6):60:1–60:30, 2022. doi:10.1145/3514355.

[57] Marie Miceli Jean-Marie Lagniez. d4. https://github.com/crillab/d4v2, 2024.

[58] Zhengfeng Ji and Xiaodi Wu. Non-identity check remains QMA-complete for short circuits. *arXiv:0906.5416*, 2009.

[59] Antonio Jiménez-Pastor, Kim G. Larsen, Mirco Tribastone, and Max Tschaikowski. Forward and backward constrained bisimulations for quantum circuits, 2024. arXiv:2308.09510.

[60] N. Cody Jones. Logic synthesis for fault-tolerant quantum computers, 2013. URL: https://arxiv.org/abs/1310.7290, arXiv:1310.7290, doi:10.48550/arXiv.1310.7290.

[61] Tyson Jones. Decomposing dense matrices into dense Pauli tensors. *arXiv:2401.16378*, 2024.

[62] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *Journal of Applied Logic*, 22:46–62, 2017.

[63] Aleks Kissinger and John van de Wetering. Pyzx: Large scale automated diagrammatic reasoning. volume 318, pages 229–241. Open Publishing Association, May 2020. URL: http://dx.doi.org/10.4204/EPTCS.318.14, doi:10.4204/eptcs.318.14.

[64] Aleks Kissinger and John van de Wetering. Simulating quantum circuits with ZX-calculus reduced stabiliser decompositions. *Quantum Science and Technology*, 7(4):044001, July 2022. URL: http://dx.doi.org/10.1088/2058-9565/ac5d20, doi:10.1088/2058-9565/ac5d20.

[65] A Yu Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191, 1997.

[66] Alexei Yu Kitaev, Alexander Shen, and Mikhail N Vyalyi. *Classical and quantum computation*. American Mathematical Society, 2002.

[67] Morten Kjaergaard, Mollie E Schwartz, Jochen Braumüller, Philip Krantz, Joel I-J Wang, Simon Gustavsson, and William D Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11(1):369–395, 2020.

[68] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates, 2013. URL: https://arxiv.org/abs/1206.5236, arXiv:1206.5236.

[69] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Practical approximation of single-qubit unitaries by single-qubit quantum clifford and t circuits. *IEEE Transactions on Computers*, 65(1):161–172, 2016. doi:10.1109/TC.2015.2409842.

[70] Frédéric Koriche, Jean-Marie Lagniez, Pierre Marquis, and Samuel Thomas. Knowledge compilation for model counting: Affine decision trees. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 947–953. IJCAI/AAAI, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6574.

[71] Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, and Philippe Suter. Complete functional synthesis. *ACM Sigplan Notices*, 45(6):316–329, 2010.

[72] Yung-Te Lai, Massoud Pedram, and Sarma BK Vrudhula. EVBDD-based algorithms for integer linear programming, spectral transformation, and function decomposition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):959–975, 1994.

[73] Nian-Ze Lee, Yen-Shi Wang, and Jie-Hong R Jiang. Solving stochastic boolean satisfiability under random-exist quantification. In *IJCAI*, pages 688–694, 2017. doi:10.24963/ijcai.2017/96.

[74] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki. Learning rate based branching heuristic for sat solvers. In *Theory and Applications of Satisfiability Testing–SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings 19*, pages 123–140. Springer, 2016.

[75] Norbert M Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[76] Michael L Littman, Stephen M Majercik, and Toniann Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27:251–296, 2001. doi:10.1023/A:1017584715408.

[77] Joao Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-driven clause learning sat solvers. In *Handbook of satisfiability*, pages 133–182. ios Press, 2021.

[78] Dmitri Maslov and Martin Roetteler. Shorter stabilizer circuits via bruhat decomposition and quantum circuit transformations. *IEEE Transactions on Information Theory*, 64(7):4729–4738, July 2018. URL: http://dx.doi.org/10.1109/TIT.2018.2825602, doi:10.1109/tit.2018.2825602.

[79] Olivia Di Matteo and Michele Mosca. Parallelizing quantum circuit synthesis. *Quantum Science and Technology*, 1(1):015003, oct 2016. URL: https://dx.doi.org/10.1088/2058-9565/1/1/015003, doi:10.1088/2058-9565/1/1/015003.

[80] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C. Benjamin, and Xiao Yuan. Quantum computational chemistry. *Rev. Mod. Phys.*, 92:015003, Mar 2020. URL: https://link.aps.org/doi/10.1103/RevModPhys.92.015003, doi:10.1103/RevModPhys.92.015003.

[81] Jingyi Mei, Marcello Bonsangue, and Alfons Laarman. Simulating quantum circuits by model counting. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification*, pages 555–578, Cham, 2024. Springer Nature Switzerland.

[82] Jingyi Mei, Tim Coopmans, Marcello Bonsangue, and Alfons Laarman. Equivalence checking of quantum circuits by model counting. In Christoph Benzmüller, Marijn J.H. Heule, and Renate A. Schmidt, editors, *Automated Reasoning*, pages 401–421. Springer, 2024.

[83] Jingyi Mei, Jan Martens, and Alfons Laarman. Disentangling the gap between quantum and #SAT. In *Theoretical Aspects of Computing - ICTAC 2024: 21st International Colloquium, Bangkok, Thailand, November 25-29, 2024, Proceedings*, pages 17–40, Berlin, Heidelberg, 2024. Springer-Verlag. doi:10.1007/978-3-031-77019-7_2.

[84] Giulia Meuli, Mathias Soeken, and Giovanni De Micheli. SAT-based {CNOT, T} quantum circuit synthesis. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation*, pages 175–188, Cham, 2018. Springer International Publishing.

[85] D Michael Miller and Mitchell A Thornton. QMDD: A decision diagram structure for reversible and quantum circuits. In *36th International Symposium on Multiple-Valued Logic (ISMVL'06)*, pages 30–30. IEEE, 2006.

[86] Michael A Nielsen and Isaac L Chuang. Quantum information and quantum computation. *Cambridge: Cambridge University Press*, 2(8):23, 2000.

[87] Philipp Niemann, Robert Wille, and Rolf Drechsler. Advanced exact synthesis of clifford+t circuits. *Quantum Information Processing*, 19(9):317, Aug 2020. doi:10.1007/s11128-020-02816-0.

[88] Román Orús, Samuel Mugel, and Enrique Lizaso. Quantum computing for finance: Overview and prospects. *Reviews in Physics*, 4:100028, 2019.

[89] Anouk Paradis, Jasper Dekoninck, Benjamin Bichsel, and Martin Vechev. Synthetiq: Fast and versatile quantum circuit synthesis. *Proc. ACM Program. Lang.*, 8(OOPSLA1), April 2024. doi:10.1145/3649813.

[90] Edwin Pednault, John Gunnels, Dmitri Maslov, and Jay Gambetta. On quantum supremacy. *IBM Research Blog*, 21, 2019.

[91] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking of quantum circuits with the ZX-calculus. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(3):662–675, September 2022. URL: http://dx.doi.org/10.1109/JETCAS.2022.3202204, doi:10.1109/jetcas.2022.3202204.

[92] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking of parameterized quantum circuits: Verifying the compilation of variational quantum algorithms. In *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 702–708, 2023.

[93] John Preskill. Quantum computing and the entanglement frontier, 2012. URL: https://arxiv.org/abs/1203.5813, arXiv:1203.5813.

[94] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, August 2018. URL: http://dx.doi.org/10.22331/q-2018-08-06-79, doi:10.22331/q-2018-08-06-79.

[95] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. MQT bench: Benchmarking software and design automation tools for quantum computing. *Quantum*, 7:1062, July 2023. URL: http://dx.doi.org/10.22331/q-2023-07-20-1062, doi:10.22331/q-2023-07-20-1062.

[96] Arend-Jan Quist and Alfons Laarman. Optimizing quantum space using spooky pebble games. In *International Conference on Reversible Computation*, pages 134–149. Springer, 2023.

[97] Maxim Raginsky. A fidelity measure for quantum channels. *Physics Letters A*, 290(1-2):11–18, 2001.

[98] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Phys. Rev. Lett.*, 86:5188–5191, May 2001. URL: https://link.aps.org/doi/10.1103/PhysRevLett.86.5188, doi:10.1103/PhysRevLett.86.5188.

[99] Narayanan Rengaswamy, Robert Calderbank, Henry D. Pfister, and Swanand Kadhe. Synthesis of logical clifford operators via symplectic geometry. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 791–795, 2018. doi:10.1109/ISIT.2018.8437652.

[100] Neil J. Ross and Peter Selinger. Optimal ancilla-free clifford+t approximation of z-rotations, 2016. URL: https://arxiv.org/abs/1403.2975, arXiv:1403.2975.

[101] Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005.

[102] Scott Sanner and David McAllester. Affine algebraic decision diagrams (AADDs) and their application to structured probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1384–1390, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[103] Sarah Schneider, Lukas Burgholzer, and Robert Wille. A sat encoding for optimal clifford circuit synthesis. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, ASPDAC '23, pages 190–195, New York, NY, USA, 2023. Association for Computing Machinery. doi:10.1145/3566097.3567929.

[104] Peter Selinger. Efficient clifford+t approximation of single-qubit operators, 2014. URL: https://arxiv.org/abs/1212.6253, arXiv:1212.6253.

[105] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[106] Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. Weighted context-free-language ordered binary decision diagrams. *arXiv preprint arXiv:2305.13610*, 2023.

[107] Meghana Sistla, Swarat Chaudhuri, and Thomas W. Reps. Symbolic quantum simulation with quasimodo. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 213–225. Springer, Springer, 2023. URL: https://doi.org/10.1007/978-3-031-37709-9_11, doi:10.1007/978-3-031-37709-9_11.

[108] A. M. Steane. Error correcting codes in quantum theory. *Phys. Rev. Lett.*, 77:793–797, Jul 1996. URL: https://link.aps.org/doi/10.1103/PhysRevLett.77.793, doi:10.1103/PhysRevLett.77.793.

[109] Paul Tafertshofer and Massoud Pedram. Factored edge-valued binary decision diagrams. *Formal Methods in System Design*, 10(2):243–270, 1997.

[110] Dimitrios Thanos, Tim Coopmans, and Alfons Laarman. Fast equivalence checking of quantum circuits of Clifford gates. In Étienne André and Jun Sun, editors, *Automated Technology for Verification and Analysis*, pages 199–216, Cham, 2023. Springer Nature Switzerland.

[111] Dimitrios Thanos, Alejandro Villoria, Sebastiaan Brand, Arend-Jan Quist Jingyi Mei, Tim Coopmans, and Alfons Laarman. Automated reasoning in quantum circuit compilation. In *Model Checking Software (SPIN) 2024, (accepted for publication)*. Springer, 2024.

[112] Yuan-Hung Tsai, Jie-Hong R Jiang, and Chiao-Shan Jhang. Bit-slicing the Hilbert space: Scaling up accurate quantum circuit simulation. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 439–444. IEEE, 2021.

[113] Richard Versluis, Stefano Poletto, Nader Khammassi, Brian Tarasinski, Nadia Haider, David J Michalak, Alessandro Bruno, Koen Bertels, and Leonardo DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *Physical Review Applied*, 8(3):034021, 2017.

[114] George F Viamontes, Igor L Markov, and John P Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003. doi:10.1023/B:QINP.0000022725.70000.4a.

[115] G.F. Viamontes, I.L. Markov, and J.P. Hayes. High-performance QuIDD-based simulation of quantum circuits. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 1354–1355 Vol.2, 2004. doi:10.1109/DATE.2004.1269084.

[116] Lieuwe Vinkhuijzen, Tim Coopmans, David Elkouss, Vedran Dunjko, and Alfons Laarman. LIMDD: A decision diagram for simulation of quantum computing including stabilizer states. *Quantum*, 7:1108, 2023. doi:10.22331/q-2023-09-11-1108.

[117] Qisheng Wang, Riling Li, and Mingsheng Ying. Equivalence checking of sequential quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(9):3143–3156, 2022. doi:10.1109/TCAD.2021.3117506.

[118] Chun-Yu Wei, Yuan-Hung Tsai, Chiao-Shan Jhang, and Jie-Hong R Jiang. Accurate BDD-based unitary operator manipulation for scalable and robust quantum circuit verification. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 523–528, 2022.

[119] Robert Wille, Hongyan Zhang, and Rolf Drechsler. ATPG for reversible circuits using simulation, Boolean satisfiability, and pseudo Boolean optimization. In *2011 IEEE Computer Society Annual Symposium on VLSI*, pages 120–125, 2011. doi:10.1109/ISVLSI.2011.77.

[120] Nic Wilson. Decision diagrams for the computation of semiring valuations. In *Proceedings of the 19th international joint conference on Artificial intelligence*, pages 331–336, 2005.

[121] Suwei Yang, Victor C. Liang, and Kuldeep S. Meel. INC: A scalable incremental weighted sampler. In Alberto Griggio and Neha Rungta, editors, *22nd Formal Methods in Computer-Aided Design, FMCAD 2022, Trento, Italy, October 17-21, 2022*, volume 3, pages 205–213. TU Wien Academic Press, IEEE, 2022. URL: https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_27, doi:10.34727/2022/isbn.978-3-85448-053-2_27.

[122] Suwei Yang and Kuldeep S. Meel. Towards projected and incremental pseudo-boolean model counting. *CoRR*, abs/2412.14485, 2024. arXiv:2412.14485, doi:10.48550/arXiv.2412.14485.

[123] Nengkun Yu and Jens Palsberg. Quantum abstract interpretation. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 542–558, 2021.

[124] Dekel Zak, Jingyi Mei, Jean-Marie Lagniez, and Alfons Laarman. Reducing quantum circuit synthesis to #SAT. In *31th International Conference on Principles and Practice of Constraint Programming (CP 2025)*, 2025. Accepted for publication.

[125] Li Zhou, Nengkun Yu, and Mingsheng Ying. An applied quantum hoare logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 1149âĂŞ1162, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3314221.3314584.

[126] Alwin Zulehner and Robert Wille. Improving synthesis of reversible circuits: Exploiting redundancies in paths and nodes of QMDDs. In *Reversible Computation: 9th International Conference, RC 2017, Kolkata, India, July 6-7, 2017, Proceedings 9*, pages 232–247. Springer, 2017.

[127] Alwin Zulehner and Robert Wille. Advanced simulation of quantum computations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(5):848–859, 2018.