Chapter 1

Introduction

Our world contains more and more software. Moreover, we are putting this software in charge of ever more important tasks in our daily lives. Software now operates or helps to operate traffic lights [277], medical devices [200], airplanes [171], spacecraft [151,160, 202] and innumerable more devices. A bug in this software can, in an innocent case, cause longer queues at the traffic light, and in the worst case can fail catastrophically when lives depend on it.

Quantum computing is a new and emerging field which may in the near or medium term deliver devices which can solve problems that are believed to be intractable on our current computers. To make quantum computing a reality, we need to be able to compile, optimize, design, synthesize, simulate and verify the correctness of quantum algorithms. Correctness verification in particular is a crucial part of the compilation and optimization of quantum algorithms (e.g., [13, 299]). However, verifying that software is correct is a tremendously difficult task, and becomes even more challenging if we wish to analyze software written for quantum computers instead of conventional ones. We therefore need powerful tools which can verify whether a proposed piece of software works as intended. In many of these approaches, the key ingredient is a data structure which can compactly store a quantum state. Therefore, this thesis has the following goal:

In this thesis, we introduce and analyze data structures for simulation and verification of software for quantum computers.

1.1 Context

Quantum computing. Physics tells us that nature is not classical, but *quantum* [139, 240]. This means that subatomic particles such as photons and electrons behave in ways that are different from the behaviour of large, everyday objects such as desks and coffee mugs. Specifically, particles can, in some sense, be in multiple places at once (called "superposition"), and they can interact and interfere with one another in ways that everyday objects do not (called "entanglement"). Current computers do not make use of these properties of matter, but scientists have speculated since the 1980s these phenomena may be useful for performing certain computations [119, 120]. A device which uses superposition and entanglement to its advantage would be called a *quantum computer*.

Quantum computing is a new and drastically different computing paradigm promising to efficiently solve several important problems that are believed to be intractable for classical computers. Examples of such problems include[*] integer factorization and discrete logarithm using Shor's algorithm [291], solving Pell's equation [149], computing the ideal class group and solving the principal ideal problem for a number field [50, 148], unstructured search using Grover's algorithm [12, 140, 231] and problems in quantum chemistry [172, 196, 201, 208, 219, 274, 314, 340, 364]. In the near term, so-called *Noisy Intermediate-Scale Quantum* (NISQ) devices are expected to both solve practical problems as well as deliver empirical evidence of quantum advantage over classical computers (see [267] for a recent perspective and [48] for a survey of NISQ devices and algorithms).

In order to achieve these lofty goals, it is not enough to be able to build large quantum computers. We will also need tools to, among other things, verify that the algorithms we design are correct; and to compile these quantum algorithms into correct implementations on hardware. These techniques are necessary to make quantum computing a practical reality; for example, compilation is necessary because the hardware typically imposes constraints on the connectivity of the qubits (e.g., [49, 287, 299, 371, 375]), or

[*]The Quantum Algorithm Zoo maintains an up-to-date list [175].

may support only a limited set of native quantum gates [75]; and circuit optimization is necessary to reduce the size and depth of a circuit, so that a computation can be performed using the limited resources of a small quantum computer [14, 286]. We speak of *analyzing quantum algorithms* as an umbrella term which includes the compilation, synthesis, optimization, simulation and verification, including equivalence checking and model checking, of quantum circuits. These tasks are interrelated; for example, some approaches to circuit optimization employ circuit equivalence checking to check whether a given optimization preserved the functionality of the circuit (e.g., [228]); and circuit equivalence checking in turn is sometimes done using circuit simulation (e.g., [71, 312]). Of these tasks, simulation is arguably the simplest, and is used in service of the other tasks; this makes it especially suited as a target application, and as a testbed for new techniques. In this thesis, therefore, we focus on quantum circuit simulation.

However, a major challenge in analyzing quantum algorithms using a classical computer is that the amount of memory required to describe an arbitrary quantum state grows exponentially in the number of qubits.[†] This is because, contrary to the classical world, where representing a system state consisting of m classical bits requires only a linear amount of memory, the state of an n-qubit quantum system is described by a vector of 2^n complex numbers, i.e., a vector in $\mathbb{C}^{2^n}$. Current estimates indicate that at least hundreds of qubits are required to perform useful tasks on a quantum computer [145]. However, even current super-computing clusters can only analyze systems with between 50 and 60 qubits represented as vectors [150, 173, 255].

Therefore, dedicated data structures and design methods which can tackle the exponential complexity of quantum computing need to be developed. By encoding quantum states using only a modest amount of memory, these data structures make quantum algorithm analysis tractable in many cases and thereby form the cornerstone on which many of these techniques are built. This motivates the main goal of this thesis: to analyze existing and design new dedicated data structures and apply them to the simulation and verification – specifically, equivalence checking – of quantum circuits.

As we will explain below, the various data structures we investigate in principle all support the same set of operations, namely, those operations necessary to simulate quantum circuits. Therefore, although in this thesis we apply our techniques primarily

[†]Specifically, a simple counting argument shows that any classical method requires at least $\Omega((1-\varepsilon)2^n)$ bits on average to describe arbitrary quantum states to a fidelity (informally, the "accuracy") of $1-\varepsilon$. See Section 2.2 for a definition of fidelity.

to the tasks of simulation and verification, we are optimistic that in the future they can be applied to quantum algorithm analysis more broadly and to other tasks to which these data structures have been applied, such as synthesis [6, 179, 279, 302, 347, 348, 373], uniform sampling and model counting [192, 195, 289], stochastic constraint optimization [99, 197, 198] and solving quantum many-body problems, such as those arising in condensed matter physics and quantum chemistry (e.g., [123, 250, 346] use matrix product states and tree tensor networks, and [78,132,225,236,247] use restricted Boltzmann machines; we will see some of these in Chapter 5).

Verification of software. In the formal verification of software, we wish to develop automated tools which guarantee that a given program works as intended. Formal verification goes beyond *testing*. Paraphrasing Dijkstra: testing can show the presence of bugs, but formal verification can show the absence of bugs [102]. Our aim will be to carry out the verification automatically, as opposed to pen-and-paper techniques (e.g., [205, 366]), and techniques in which a proof assistant, such as Isabelle [253, 254] or Coq [47,270], assists a human prover. In our setting, the advantage of an automated approach is that no human expertise is required for this step, so that these tools can be embedded in a toolchain and, e.g., process code alongside a compiler.

Today, formal verification is often an important part of the design process of software and hardware. Baier and Katoen [28] list many real-world examples (for example, the analysis of software of Rotterdam's storm surge barrier [316] and NASA's Deep Space 1 probe [151], among others).

The principal bottleneck in formal verification is the *state space explosion*: the simple observation that the number of states of a system grows exponentially with the size of the program. Among other things, this is caused by the number of different possible inputs, and the number of possible interleavings of parallel processes. Indeed, most approaches to verification have as their principal aim to somehow tame this state space explosion. They do so by exploiting structure in the system; for example, partial order reduction exploits the commutativity of (concurrent) processes [134,259,319], symbolic reachability analysis uses decision diagrams to exploit the structure of the reachable state space when viewed as a Boolean function [29, 70] and IC3 uses SAT solvers [60]. We will see that the effectiveness of verification methods for quantum software likewise hinges on their ability to harness the structure that is present in quantum states and systems. In fact, in Section 1.2 we will see that data structures previously used to address the state space explosion in classical systems are repurposed to represent quantum states.

Verification of quantum software. Our aim will be to verify whether two given quantum circuits are equivalent, i.e., whether they produce the same output state when given the same input state. This allows a user to, for example, check whether their (efficient but complicated) algorithm is equivalent to an (inefficient but obviously or provably correct) reference implementation. A quantum circuit on n qubits is described by a $2^n \times 2^n$ complex-valued matrix, so, concretely, checking the equivalence of two quantum circuits entails checking whether their matrices are equal.[‡]

This method is complementary but orthogonal to approaches, like model checking, in which one checks whether a (quantum) system satisfies a specification formulated in for instance a temporal logic. An advantage of our approach is that an equivalence checker can be integrated directly into a compiler or an optimizer (e.g., as in [228]) without requiring the user to supply a specification in a temporal logic. Therefore, in Section 2.4, we put these two approaches in context, highlighting similarities and differences. For now, we briefly mention that there are several quantum extensions of temporal logics [30, 79, 100, 216, 217, 355, 359, 363, 366, 367], and several software tools provide support for these logics [16, 18, 116, 128, 161, 205].

The challenge of verifying quantum software inherits the difficulties of the classical case and in addition poses several new ones, both practical and conceptual. We briefly list two difficulties below, drawing inspiration from Ying et al. [360, 363] and Turrini [317].

1. A quantum state requires an exponential amount of data to describe in general. This is a practical problem which can be addressed by using data structures which can describe some states using only a modest amount of memory. We review existing data structures in Section 1.2, and describe new ones introduced in this thesis in Section 1.4.

2. A quantum algorithm admits an infinite number of possible inputs, which raises the conceptual challenge of inventing methods to guarantee that a quantum algorithm is correct on all input states. Burgholzer et al. articulate why enumerating and checking a property for all (finitely many) basis vectors is not sufficient [71]. We describe our chosen approach to this second difficulty in Section 1.4 and survey some existing approaches in Section 2.4.

[‡]In Sec. 2.2.1 we describe in more detail how such a matrix comes about. For the purposes of this introduction, it suffices to assume that, if we know the gates that constitute the circuit, then we can find the circuit's matrix.

To address the second difficulty, existing equivalence checkers have used two main approaches. Either the circuit is simulated on several randomly chosen input states (e.g., as in [71]), or the circuits' matrices are constructed and then we simply check whether the two matrices are equal[§] [74, 81, 162, 163, 241, 256, 345, 349, 356, 357]. Both approaches encounter the first difficulty listed above. Namely, during simulation, the intermediate states encountered are described by exponentially long vectors; and in the second approach, when building a circuit's matrix, such a matrix is exponentially large. Since the problem in both cases is that an object does not fit in memory, a natural solution is to use compression. Indeed, our solution of choice is to use data structures which compress either a quantum state vector, or a circuit's matrix. Therefore, we turn to this topic next.

1.2 State-of-the-art data structures for quantum algorithm analysis

The previous section argues that dedicated data structures are the key to effective quantum algorithm analysis. In this thesis, therefore, we investigate many popular state-of-the-art data structures that are used for the analysis of quantum algorithms, especially verification and simulation. These include the (extended) stabilizer formalism [3, 135], decision diagrams [211, 227, 331, 341, 374], matrix product states [248, 330, 346] and restricted Boltzmann machines [78, 174, 225]; data structures that we do not treat in thesis include tensor networks [284] and the ZX-calculus [89, 105, 256, 320], as well as SAT-based methods [34, 43, 351]. Each of these data structures can (i) represent the state of a quantum system at a given moment in time (among other things), and (ii) can simulate the execution of a quantum circuit on a given input.

The principal difficulty these methods are intended to address is the amount of memory required to represent an arbitrary quantum state. To this end, these methods aim to encode (i.e., compress) a quantum state using only a modest amount of computer memory by, broadly speaking, exploiting the structure that is often present in quantum states that arise in practice in quantum algorithms. Given such a compact representation, a quantum circuit can then be simulated "simply" by starting with a

[§]More precisely, we check whether two matrices are equal up to a complex constant; we explain this detail in Section 2.2 and sweep it under the rug for now.

data structure, e.g., a decision diagram (DD), representing the initial state and then repeatedly finding the next state by applying the next gate of the circuit to the current state, i.e., building a DD representing this next state. A given approach then succeeds in simulating a given circuit if the representations of the intermediate states fit in the available memory, which is achieved when, informally speaking, these intermediate states contain enough of the kind of structure that the data structure exploits. Comparing the degrees to which different data structures succeed in compressing a state, and simulating a circuit, is done using tools from *knowledge compilation*.

For the purposes of this introduction, let us first highlight two of these methods: the *stabilizer formalism* [3,135] and *decision diagrams* [227,374]. Afterward, we introduce knowledge compilation as a way to study the relative strengths of such data structures.

The stabilizer formalism. Stabilizer states are a subset of quantum states [135]. They play a fundamental role in quantum computing as they form the basis of many quantum protocols, such as communication protocols like superdense coding [42], cryptography [38,131,263], error-correcting codes [41,135,137,310] and measurement-based quantum computing [272]. Aaronson and Gottesman [3] note that, "Stabilizer states are expressive enough to encompass most "paradoxes" of quantum mechanics, including the GHZ experiment [138], quantum dense coding [42], and quantum teleportation [40]"; to this list we might add entanglement [298] and Bell's experiment [36]. Given their importance, it may be surprising that stabilizer states can be efficiently simulated on a classical computer. On the other hand, stabilizer states are not rich enough to encompass all of quantum computing, so this efficient simulation does not allow one to simulate arbitrary quantum circuits.

Stabilizer states are created by so-called stabilizer circuits, which are quantum circuits consisting of only a restricted gate set – the Clifford gates.[¶] Aaronson and Gottesman [3], Bravyi et al. [21, 61, 62, 63, 64] and Qassim et al. [268] show how to simulate a quantum circuit even when it contains non-Clifford gates. This *extended stabilizer formalism* can simulate any quantum circuit. This method expresses an arbitrary quantum state vector as a linear combination of stabilizer states. This way, a circuit on n qubits containing only k non-Clifford gates can be simulated in $\exp(k)\text{poly}(n)$ time, i.e., simulation is fixed-parameter tractable in the number of non-Clifford gates.[‖] Subsequently, Aranuchalam et al. [21] showed that even the more difficult problem

[¶]Stabilizer states, Clifford circuits and Clifford gates are defined in Sec. 2.2.3.

[‖]This bound can be improved if we consider a specific gate set. For example, a circuit containing only Clifford gates and k T-gates can be simulated in $\mathcal{O}(2^{\gamma k}\text{poly}(n))$ time with $\gamma \leq 0.228$ [62].

of checking whether two quantum circuits are equal is fixed-parameter tractable in the number of non-Clifford gates. The principal limitation of this method is that many interesting quantum circuits contain many non-Clifford gates. Moreover, even in theory, expressing a given quantum state as a linear combination of stabilizer states requires some minimum number of terms, called its stabilizer rank. Unfortunately, the stabilizer rank of interesting states may be expected to grow super-polynomially in the number of qubits on complexity-theoretic grounds,** although the current best explicit lower bounds are merely linear in the number of qubits [64, 209, 222, 260].††

Decision diagrams. A decision diagram (DD) aims to provide a compact representation of quantum state vectors (and quantum circuits) by exploiting the observation that a state vector often contains repeated subvectors, i.e., a state vector may contain multiple copies of the same block. By recognizing these repeated occurrences, a decision diagram can achieve lossless compression. This way, some state vectors can be represented very succinctly even for very large numbers of qubits. For example, any product state (i.e., a state without entanglement) on n qubits can be represented by certain DDs using only $\mathcal{O}(n)$ memory.

Decision diagrams were first introduced and developed by Lee [199], Akers [7] and Bryant [67] to analyze classical systems. Viamontes, Markov and Hayes first used a DD-based approach to simulate quantum circuits by introducing QuIDD, a decision diagram which can represent a vector of complex numbers, i.e., a quantum state [331, 333]. Miller and Thornton [227] introduced QMDDs, which specialize SLDDs [352] and AADDs [281] and which improve on QuIDDs by achieving greater compression. Since then, implementations of QMDDs and related DDs have achieved striking performance on a varied set of benchmarks, both for simulating quantum circuits [143, 156, 294, 368, 374] and for checking equivalence of quantum circuits [71, 74, 81, 162, 163, 241, 345, 349, 356, 357]. Based on empirical evaluations, Peham et al. conclude that "decision diagrams and the ZX-calculus can serve as complementary approaches" for quantum circuit equivalence checking [257].

Recent work has seen QMDDs applied to various subproblems of quantum simulation and verification and has improved many practical aspects of working with decision diagrams [142, 243, 294, 369, 370]. For example, Grurl et al. [141, 143, 144] and Hong

**For example, Morimae and Tamaki show that the stabilizer rank of the so-called magic state on n qubits grows as $2^{\Omega(n)}$ assuming the Exponential Time Hypothesis [232]. Mehraban and Tahmasbi show that the stabilizer rank of this magic state must grow super-polynomially, unless $\mathsf{MA} = \mathsf{P}^{\#\mathsf{P}}$ and the polynomial hierarchy collapses, which is widely thought to be unlikely [222].

††Recent work by Mehraban and Tahmasbi claims a quadratic lower bound [222].

et al. [162] develop DD-based techniques to address the challenges posed by noise and decoherence in quantum circuits. Second, Zulehner et al. [368] and Hillmich et al. [157] achieve better compression by periodically deleting parts of the diagram during simulation of a circuit, while guaranteeing that the fidelity (informally, the "accuracy") does not drop below a given threshold. Hong et al. use DD-based techniques to perform tensor network contraction, which is a general problem which includes quantum circuit simulation as a special case [163]. Lastly, multiple authors implement heuristics for dynamic variable reordering, which is a fundamental operation on DDs [156, 226]. Hillmich et al. find that variable reordering introduces challenges of numerical accuracy [156].

We highlight one QMDD-based method for circuit equivalence checking by Burgholzer et al. [71] in which the outputs of two given circuits are compared on random inputs. In this approach, QMDDs are used to simulate the two circuits on a randomly chosen input state; subsequently, we check whether the two circuits produced the same output state. The idea is that, if two circuits are different, then a random input likely reveals this. Specifically, the input states are random stabilizer states; this is necessary (as opposed to using simpler states, such as basis states, as inputs) in order to achieve good guarantees on the probability of finding such a counterexample.

Limitations of state-of-the-art tools for verification of quantum circuits. Early in our investigations, we discovered and proved that existing DDs cannot efficiently handle stabilizer states and Clifford circuits. This significantly limits their applicability to the analysis and verification of many common quantum protocols. Specifically, we proved the following about existing types of DDs:

> Existing types of decision diagrams require an exponential amount of memory to store some stabilizer states and (the unitary matrices of) some stabilizer circuits (Corollary 3.1). That is, stabilizer quantum circuits are a worst-case input for existing decision diagrams.

This may come as a surprise, because DDs are, ostensibly, designed to excel at analyzing states that possess structure, and stabilizer states certainly possess structure. This had been observed empirically earlier by Burgholzer et al. [71] for small numbers of qubits but had not been explained analytically. This discovery motivates our goal to find better decision diagrams, which are able to analyze stabilizer circuits in particular and common quantum protocols more broadly.

Although there are several available software tools for quantum verification, each suffers from certain limitations: those that are based on decision diagrams cannot efficiently handle stabilizer circuits (e.g., [71, 73, 75, 162]); some tools can handle only stabilizer circuits (e.g., [18, 128, 312]); other tools (e.g., [17, 43, 116, 161]) can handle arbitrary circuits but the chosen simulation method significantly limits the number of qubits the tool can handle (e.g., the tool of [43] generates SAT formulas that are exponentially large, except when analyzing stabilizer circuits).

Knowledge compilation. The approaches to verification that we study in this thesis each address the problem that quantum state descriptions are in general too large by using some data structure (DS) to (losslessly) compress a quantum state. These data structures are built on different architectures, which make some more suited to and more effective at certain tasks than others.

Knowledge compilation is the systematic study of such data structures in order to understand how effective a given DS is at a given task [97, 112, 113, 194]. To this end, researchers have primarily compared data structures on three criteria: *succinctness*, which tells us whether a data structure achieves good compression, and *tractability* and *rapidity*, which tell us how efficiently we can operate on the data in compressed form. A major goal of knowledge compilation is to help users choose a DS which is best suited to the task at hand. To this end, Darwiche and Marquis [97] formulate the following advice: when considering which DS to use, one should choose the most succinct DS which performs the query of interest in polynomial time.

However, the tools of knowledge compilation have not yet been systematically applied to data structures used for analysis of quantum computing. We now sketch four shortcomings of the literature on this topic.

First, the succinctness relations between popular data structures were unknown, e.g., it was not known whether matrix product states were more succinct than QMDDs. Therefore, it is not obvious how a practitioner should follow Darwiche's advice, which relies on the succinctness relations.

Second, it is not obvious which operations are important for such data structures in the context of quantum algorithm analysis (i.e., in terms of tractability and rapidity). For example, it seems useful that a data structure be able to efficiently compute the probability of a given measurement outcome for a given quantum state. However, tractability of one operation (such as measurement) is sometimes traded off against another operation (such as applying certain gates). Although for several data structures efficient algorithms are known for some of these operations, to the best of our knowledge there has been no systematic comparison of which tradeoffs are available, or how to prioritize such tradeoffs. Moreover, the traditional operations considered in knowledge compilation (e.g., in [97, 112, 194]), such as conjunction, disjunction and existential quantification, are inherently Boolean operations. However, quantum state vectors are not Boolean but complex vectors, so it is not obvious what would be natural quantum analogues for these operations. For example, Fargier et al. [113] consider the tractability of *pointwise addition* of two vectors, which in principle can be applied to complex-valued vectors such as quantum state vectors; however, the addition of two quantum states is not a meaningful quantum operation.

Third, the rapidity criterion, introduced by Lai et al. [194], is defined only for *canonical* data structures, i.e., for those data structures which provide a unique normal form for each quantum state. However, many data structures in popular use for quantum algorithm analysis are not canonical, such as restricted Boltzmann machines and matrix product states. Therefore, the criterion cannot be applied to these data structures, which limits our ability to compare them. Consequently, we cannot say that, e.g., matrix product states are more rapid than QMDDs, because MPS are not a canonical data structure.

Fourth, considering only succinctness and tractability sometimes leads to apparently incongruent results. For example, representing an n-(qu)bit state vector using an explicit vector representation always uses $\mathcal{O}(2^n)$ memory, whereas using a QMDD uses much less memory in some cases. However, applying a certain quantum gate called a Hadamard gate to a quantum state is considered tractable for the vector representation (because it can be performed in time polynomial in the length of the vector) but is considered intractable for the QMDD representation. This is counterintuitive: using QMDDs for this task is never much slower than using the state vector representation, and indeed is sometimes much faster. Therefore, it appears as though there is a tradeoff between succinctness and tractability in this case, but it is not obvious whether the implied tradeoff between speed and memory is real; or to put it more mildly, it is not clear what conclusions one should draw from this data point.

1.3 Research questions

In the previous section, we have sketched the current frontier of research into quantum verification methods. To some, it may come as a surprise that analyzing quantum computers on classical hardware is possible even in principle, but in fact, as we have seen above, there are today a variety of methods and tools to verify the correctness of quantum algorithms. However, these tools are still in their infancy relative to the mature field of classical verification. For example, several data structures, such as matrix product states and restricted Boltzmann machines, have not yet been applied to the verification of quantum algorithms to the best of our knowledge. Moreover, the limitations of several state-of-the-art methods, such as decision diagrams and the stabilizer formalism, leave opportunities for improvement. To push this frontier forward, we now ask three research questions which will guide the work in this thesis.

First, we seek to address the limitations of decision diagrams and the stabilizer formalism. In particular, we have remarked that stabilizer circuits are a worst-case scenario for existing decision diagrams because they need exponential size to represent most stabilizer states. Therefore, we ask:

> **Research question 1.** *Can we unite the strengths of decision diagrams and the stabilizer formalism?*

We would consider this question answered if we found a data structure which incorporates the strengths of the stabilizer formalism and of existing decision diagrams.

We have argued that the existing literature on knowledge compilation insufficiently covers the case of quantum algorithm analysis. In particular, the data structures in popular usage have not yet been compared on their succinctness, many non-canonical data structures cannot be compared on their rapidity and it is not clear how to prioritize the tractability of operations when designing new data structures. Therefore, we ask:

> **Research question 2.** *How can we analytically compare the relative strengths of data structures which represent quantum states?*

To date, not all the different DDs have been adapted to the quantum setting.[‡‡] For

[‡‡]A DD encodes a function $f\colon \{0,1\}^n \to R$ for some range R; we say a DD is *classical* if $R \subseteq [0,1]$

example, there are no quantum SDDs [96], DSDBDDs [46, 264], FBDDs [129], or ZDDs [229]. Given that DDs have been widely applied to and have shown striking success in the verification of classical software, it is interesting to ask which DDs would be effective in this new setting, if they were suitably adapted.

> **Research question 3.** *Which classical decision diagrams might be effective for the analysis of quantum algorithms, if they were suitably adapted?*

1.4 Contributions

In light of the bottlenecks of current state-of-the-art methods described above in Section 1.2, we introduce a novel data structure called *Local Invertible Map Decision Diagram* (LIMDD), which unites the strengths of DDs and the stabilizer formalism, in Chapter 3, thus addressing Research question 1. Specifically, LIMDDs can efficiently analyze both (i) all stabilizer circuits, and (ii) all circuits that existing DD-based tools can analyze, whereas existing DD-based tools required exponential resources to analyze stabilizer circuits. In fact, LIMDDs can efficiently analyze some non-stabilizer circuits that existing DD-based tools cannot efficiently analyze; therefore, LIMDDs are more than merely the "union" of existing DDs with the stabilizer formalism. In general, LIMDDs can simulate any quantum circuit (regardless of the gate set) on any initial state; and they can represent (the unitary matrix of) any quantum circuit. Many important quantum operations take only polynomial time in the size of the LIMDD, such as checking the equality of two quantum circuits, computing the probability of obtaining a given measurement outcome, sampling from a quantum state, and applying various gates, such as the (controlled) Pauli X, Y, Z gates. We explain Burgholzer et al.'s [71] empirical finding that QMDDs are unable to efficiently handle stabilizer states by proving that there are stabilizer states which can only be represented by exponentially-sized QMDDs. In the other direction, we show that QMDDs can represent quantum states that are not stabilizer states. Next, we compare LIMDDs to the extended stabilizer formalism and find that we have an exponential advantage over a popular instantiation of this formalism, which we call the Clifford+T simulator, assuming the exponential time hypothesis.

We implement LIMDDs in the publicly available software package MQT [233]. In

and *quantum* if $R = \mathbb{C}$.

Chapter 4, we show empirically that we are able to make good on the promises made in Chapter 3: our tool outperforms a state-of-the-art method when analyzing quantum circuits that implement the Quantum Fourier Transform (QFT). The QFT is a subroutine of many quantum algorithms, so it is important that proposed circuits are correct. We use the method of Burgholzer et al. [71], who check the equivalence of two quantum circuits by simulating them on a set of randomly chosen input states. The idea is that if two circuits are different, then one may expect that a random input will reveal this, since the outputs of the circuits will differ with high probability.

To answer Research question 2, we give a quantum knowledge compilation map in Chapter 5. We map the succinctness, tractability and rapidity of some of the currently popular data structures for analyzing quantum systems: decision diagrams, matrix product states and restricted Boltzmann machines. We show that LIMDDs and matrix product states are more rapid than QMDDs, which in turn are more rapid than the state vector representation. We argue that the most important operations in the context of quantum algorithm analysis are measurement, computing inner products, and applying various common quantum gates; we investigate the tractability of these operations for the data structures listed above. Lastly, we contribute to the *methodology* of knowledge compilation (i) by extending the definition of rapidity by removing the constraint that the data structures are canonical, which allows us to compare data structures that previously could not be compared; (ii) by giving a simple sufficient condition for when one (non-canonical) data structure is more rapid than another; (iii) by establishing the rapidity relations between various data structures, thus showcasing that the condition is simple to use and (iv) more broadly, by making a knowledge compilation map involving, for the first time, data structures that are not a variant of a so-called Negation Normal Form structure (see [33, 94]).

In Research question 3, we ask which classical decision diagrams might contribute to the analysis of quantum algorithms. To this end, we investigate two DDs: the Sentential Decision Diagram (SDD) [96] and the Disjoint Support Decomposition Binary Decision Diagram (DSDBDD) [46, 264]. For the time being, we test them out in the classical domain. It is instructive to do this, because the quantum setting poses several additional challenges relative to the classical setting. Notably, a quantum diagram needs to account for complex numbers, adding complexity to its architecture. Moreover, the finite precision of arithmetic leads to numerical inaccuracies and to questions of numerical stability. Meeting these challenges entails increasing the conceptual, and sometimes computational, complexity of the algorithms. For example, variable reordering is a fundamental and conceptually simple task for DDs. However, Hillmich et al. [156] find that this task becomes significantly error-prone for quantum DDs due to rounding errors that are inherent to finite precision floating-point arithmetic. By deferring these challenges to future work, we obtain diagrams which are conceptually simpler. This allows us to go through more prototyping cycles, e.g., in our case, for testing heuristics for setting an SDD's hyperparameters (specifically, its *variable tree*). We speculate about how to extend these two DDs to the quantum domain in Section 8.3.

In this context of future diagrams, we contribute the following two results. First, in Chapter 6, we prove that the DSDBDD sometimes achieves exponentially better compression than many other DDs. This solves succinctness but leaves unsolved the questions of tractability and rapidity.

Second, in Chapter 7, we extend the LTSmin model checking framework with SDDs, so that users can take advantage of the better compression that it offers over other DDs that were previously integrated with LTSmin. Effectively deploying SDDs requires us to propose the first heuristics for SDDs: we design the structure of a certain binary tree (a "variable tree") based on limited available information about which variables are read and written by the software program that we are trying to verify. Our experiments show that SDDs allow us to make a well-known tradeoff between time and space: the SDDs are slower than BDDs, but they achieve better compression; therefore, a patient user may find that these DDs allow her to solve larger problems without running out of memory.

Although we have pitched these two chapters as preparing the way for quantum applications, we note that these contributions may be valuable in their own right. For example, users of the LTSmin toolset can already use SDDs, regardless of their future applications to quantum computing.

1.5 Outline

The chapters of this thesis can be summarized as follows.

Chapter 2 We give the necessary background on quantum computing, decision diagrams and the stabilizer formalism.

Table 1.1: The chapters of this thesis and their methods.

Chapter	Domain	Method	Data structure	Artefact
Chapter 3	Quantum	Analytical	LIMDD	
Chapter 4	Quantum	Empirical	LIMDD	MQT [233]
Chapter 5	Quantum	Analytical	Various	
Chapter 6	Classical	Analytical	DSDBDD	
Chapter 7	Classical	Empirical	SDD	LTSmin [210]

Chapter 3 We introduce the LIMDD data structure, give algorithms for quantum simulation using this data structure and prove that it is faster than matrix product states and the extended stabilizer formalism for simulating certain quantum circuits.

Chapter 4 We describe our implementation of the LIMDD. We evaluate our implementation empirically by analyzing a circuit which implements Quantum Fourier Transform.

Chapter 5 We provide a knowledge compilation map for quantum computing: we compare five data structures on their succinctness, tractability and rapidity and we give a sufficient condition for one data structure to be more rapid than another.

Chapter 6 We prove that DSDBDDs are exponentially more succinct than BDDs.

Chapter 7 We use SDDs to do model checking on a large standard problem set. To this end, we extend the LTSmin package to provide support for SDDs.

Chapter 8 We conclude by reflecting on the work in this thesis and laying out some open problems for future research.

Table 1.1 lists, for each chapter in this thesis, its main data structure, the method by which it analyzes it and a link to the resulting software package, if any.

The proofs of many theorems appear in the appendices.

1.6 Publications and artefacts

This dissertation is based on the following publications.

1. **Lieuwe Vinkhuijzen** and Alfons Laarman. Symbolic Model Checking with Sentential Decision Diagrams. International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. Springer, Cham, 2020.
2. **Lieuwe Vinkhuijzen** and Alfons Laarman. The Power of Disjoint Support Decompositions in Decision Diagrams. NASA Formal Methods Symposium. Springer, Cham, 2022.
3. **Lieuwe Vinkhuijzen**, Tim Coopmans, David Elkouss, Vedran Dunjko and Alfons Laarman. LIMDD a decision diagram for simulation of quantum computing including stabilizer states. Quantum, 2023.
4. **Lieuwe Vinkhuijzen**, Thomas Grurl, Stefan Hillmich, Sebastiaan Brand, Robert Wille and Alfons Laarman. Efficient implementation of LIMDDs for Quantum Circuit Simulation. 29th International Symposium on Model Checking of Software, 2023.
5. **Lieuwe Vinkhuijzen**, Tim Coopmans and Alfons Laarman. A Quantum Knowledge Compilation Map. arXiv preprint, arXiv:2401.01322 (2024).

This dissertation features contributions to the following software packages.

1. **LIMDD.** We implemented the LIMDD in the MQT DDSIM package, allowing a user to simulate arbitrary quantum circuits [233].
2. **LTSmin.** We augmented the LTSmin package, giving the user the option to use Sentential Decision Diagrams [210]. This software package forms the basis of Chapter 7.

Besides the publications included in this thesis, we co-authored the following paper.

1. **Lieuwe Vinkhuijzen** and Andre Deutz. A simple Proof of Vyalyi's Theorem and Generalizations. Electronic Colloquim on Computational Complexity, 2019.