Advancing quantum computing with formal methods

Quist, A.; Mei, J.; Coopmans, T.J.; Laarman, A.W.; Platzer, A.; Rozier, K.Y.; ... ; Rossi, M.

Advancing Quantum Computing with Formal Methods

Arend-Jan Quist, Jingyi Mei, Tim Coopmans(✉), and Alfons Laarman

Leiden University, Leiden, The Netherlands
{a.quist,j.mei,t.j.coopmans,a.w.laarman}@liacs.leidenuniv.nl

Abstract. This tutorial introduces quantum computing with a focus on the applicability of formal methods in this relatively new domain. We describe quantum circuits and convey an understanding of their inherent combinatorial nature and the exponential blow-up that makes them hard to analyze. Then, we show how weighted model counting (#SAT) can be used to solve hard analysis tasks for quantum circuits.

This tutorial is aimed at everyone in the formal methods community with an interest in quantum computing. Familiarity with quantum computing is not required, but basic linear algebra knowledge (particularly matrix multiplication and basis vectors) is a prerequisite. The goal of the tutorial is to inspire the community to advance the development of quantum computing with formal methods.

1 Introduction

> *"Nature isn't classical, (...), and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy"*
>
> Richard Feynman [11]

Renowned physicist and Nobel prize winner Richard Feynman is said to be the first to coin the idea of a 'quantum computer' by inverting the difficulty he encountered when solving equations in quantum mechanics [11]. He proposed to make nature's complexity work for us, instead of trying to calculate nature's behavior using classical tools —which Feynman noticed requires performing exponentially many calculations. Fast forward to today, and the first quantum computers have been built arguably showing first signs of 'quantum advantage' [3].

We see that powerful techniques from formal methods are ideally suited to tackle some of the crucial problems on the road towards a full-scale quantum

A.-J. Quist, J. Mei and T. Coopmans–These authors contributed equally.

A. Platzer et al. (Eds.): FM 2024, LNCS 14934, pp. 420–446, 2025.
https://doi.org/10.1007/978-3-031-71177-0_25

advantage. This tutorial, therefore, provides a crash course into quantum computing, specifically geared towards a formal methods audience. Our goal is to inform the community of the challenges in handling quantum computations and point them in the right direction to apply their own favorite techniques on open problems in the field of quantum computing (Sect. 4).

For inspiration, we show how to formulate the semantics of quantum circuits using #SAT. At the same time, we keep the quantum background to an absolute minimum by introducing quantum computing as a minor extension of reversible and probabilistic computation,[1] and gradually introducing the required notation.

Nevertheless, our basic exposition manages to touch upon quantum algorithms, such as Grover search [15], by briefly discussing how the Boolean satisfiability problem can be incorporated into a quantum circuit as an oracle. While the satisfying assignments can then be found in the so-called *probability amplitudes* of the quantum state that is computed by the circuit, it is hard to extract such information using *measurements*, the only method to extract physical information from a quantum system (which is unfortunately rather crude). Such an example should inspire listeners to think about how to extract the solution to the satisfiability problem using measurements, which is the difficulty of coming up with so-desired quantum algorithms that outperform their classical alternatives.[2]

Our exposition here focuses on the task of (classically) simulating quantum circuits, which is formally defined in the background section. This ensures that the audience becomes familiar with the semantics of quantum circuits and algorithms (which can be formulated as uniform families of quantum circuits). At the very end, we show, based on related works, how the discussed encodings in #SAT can also be used to solve various other tasks, inter alia: (quantum circuit) equivalence checking, synthesis, optimization and quantum Hoare logic checking. We also point the audience in the direction of the current obstacles on the road to quantum supremacy, such as, circuit optimization and quantum error correction (which will crucially realize the ideal quantum circuit model introduced here). Here, we even point out connections beyond quantum computing since it is easy to show that progress in handling quantum circuits translates to progress in solving important and hard problems in quantum mechanics, like simulation of many-body physics and finding ground states.

[1] Surprisingly, not even complex numbers are needed, as the (classical) reversible Toffoli gate, which is universal for reversible computing, requires only an additional Hadamard (Walsh-Hadamard/Reed-Muller transform) gate to yield a universal gate set for quantum computation [1,34].

[2] While there is, for instance, an efficient quantum algorithm for factoring (Shor's [35]), we do not know for sure whether factoring is hard classically. So a true *separation* between the classical and quantum complexity classes is still open (formalized as the $\mathsf{BPP} = \mathsf{BQP}$ question).

2 Quantum Computing for Computer Scientists

"Quantum computing becomes easy once you take the physics out of it"

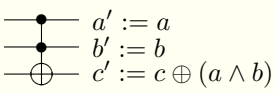
https://scottaaronson.blog

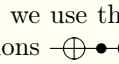
This section introduces quantum circuits as a generalization of (classical) reversible and probabilistic computing. Having provided the reader with an understanding of these classical building blocks, we then explore the semantics of quantum circuits. We finish by highlighting the crucial aspects of a quantum circuit that make formal methods amenable to it.

2.1 From Reversible, via Probabilistic, to Quantum Circuits

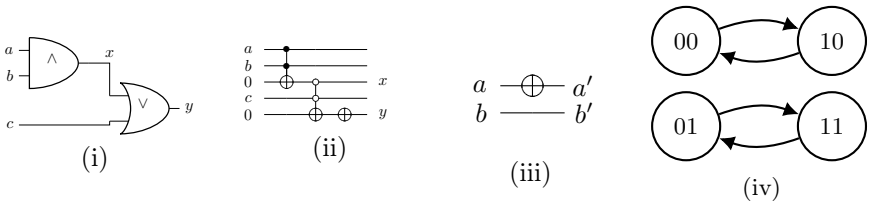
Each (irreversible) Boolean circuit can be written as a reversible circuit. One way to do this is using circuits with only the so-called Toffoli gate [37] (see the yellow box below).

Toffoli gate. The three-bit Toffoli gate, shown below, is both reversible and classically universal. It takes three bits a, b, c as inputs and outputs $a, b, c \oplus ab$. Informally, it only flips the c bit when both a and b are true (leaving a and b intact).



Applying a Toffoli twice will compute $c \oplus ab \oplus (a \wedge b) = c$, thus reversing the original computation by 'uncomputing' the result. Furthermore, by setting $a = b = 1$, the c bit is always flipped so that we can realize a NOT gate ($\neg$) as syntactic sugar (drawn as  in a circuit). Toffoli is classically universal, which follows from the fact that the gate set $\{\wedge, \neg\}$ is universal and that Toffoli implements an AND gate ($\wedge$). By setting $a = 1$, we obtain a (singly-)controlled-NOT, or CNOT gate (drawn as ), which flips the value of c only when b is set. As additional syntactic sugar, we use the negated control $\circ$, which is a control $\bullet$ surrounded by negations  . It checks whether the input is false, returning it to its original state.

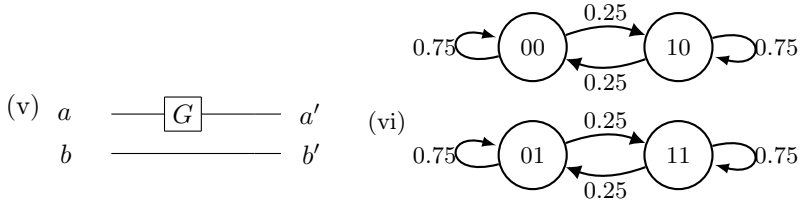
As an example of writing each Boolean circuit as a reversible circuit, consider circuit (i) on input $a, b, c \in \{0, 1\}$ below: We construct a reversible circuit by storing intermediate value x and the result y separately, adding wires for these variables. This leads to the reversible circuit (ii) containing NOT gates (drawn as $\oplus$ indicating a bit flip without control) and Toffoli gates (a bit flip $\oplus$ with two controls or).



Next, let us visualize the action of a classical reversible circuit. In (iii) above, we use a circuit on two bits a, b and give in (iv) an automaton of the four possible states on two bits. An arrow from state (a, b) to (a', b') indicates that the circuit maps (a, b) to (a', b'). (We use a circuit on two bits instead of five as in (ii) to avoid too large automata for easy readability.)

Exercise: Create a reversible circuit for the Boolean formula $((a \wedge b) \vee c) \wedge d$. Then extend the circuit to uncompute all wires but the result.

2.1.1 Probabilistic Classical (reversible) Computing with Linear Algebra To move closer towards quantum circuits, we will now see how the automaton changes when the logical gates used are probabilistic. Specifically, in example (iii) above, we replace the NOT gate ($\oplus$) with a probabilistic gate G, which does nothing with probability 75% and applies a NOT with 25%, resulting in (v) below. Then the circuit is modeled by a Markov chain, and the transition arrows in the automaton (vi) are labeled with the probabilities of mapping the input state (a, b) to output state (a', b') [16].



A Markov chain is typically analyzed by associating each state (a, b) (abbreviated 'ab') in the automaton to a basis vector and writing down the transition probabilities as an adjacency matrix M of the automaton above:

$$\text{`00'} \equiv \begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \text{`01'} \equiv \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \text{`10'} \equiv \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \text{`11'} \equiv \begin{bmatrix}0\\0\\0\\1\end{bmatrix}, \; M = \begin{bmatrix}0.75 & 0 & 0.25 & 0\\ 0 & 0.75 & 0 & 0.25\\ 0.25 & 0 & 0.75 & 0\\ 0 & 0.25 & 0 & 0.75\end{bmatrix} \quad (1)$$

Next, the probability distribution over the output states (a', b') on input (a, b) is computed using matrix-vector multiplication. For example, on input '00' ($a = 0, b = 0$), the output is

$$M \cdot \text{'00'} = \begin{bmatrix} 0.75 & 0 & 0.25 & 0 \\ 0 & 0.75 & 0 & 0.25 \\ 0.25 & 0 & 0.75 & 0 \\ 0 & 0.25 & 0 & 0.75 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.75 \\ 0 \\ 0.25 \\ 0 \end{bmatrix} = 0.75 \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + 0.25 \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

that is, '00' is mapped to itself with probability 75% and to '10' with probability 25%, as was already revealed by the Markov chain.

The advantage of using the transition matrix instead of writing down the Markov chain is that the input need not be a single bitstring itself but can also be a probability distribution over bitstrings. In this more general case, **matrix-vector multiplication correctly propagates the probabilities.** To see this, we first directly compute the probabilities of the output bitstrings for the circuit in (v) using the Markov chain. We use the example of an input state that is either set to the bitstring '00' with probability 0.4 or to the bitstring '10' with probability 0.6. In the first case, the output of the circuit in (v) is '00' with a probability of 0.75 and '10' with a probability of 0.25; in the second case, it is '00' ('10') with a probability of 0.25 (0.75). Aggregating these two, the output is '00' with probability $0.4 \cdot 0.75 + 0.6 \cdot 0.25 = 0.45$ and '10' with probability $0.4 \cdot 0.25 + 0.6 \cdot 0.75 = 0.55$. Equivalently, we obtain this result using linear algebra. The input, which represents '00' with probability 0.4 and '10' with probability 0.6, is the vector $\boldsymbol{v} = \begin{bmatrix} 0.4 & 0 & 0.6 & 0 \end{bmatrix}^\top$ (where the transpose $(.)^\top$ turns a row vector into a column vector). The output probability distribution is computed by applying the transition matrix to the input vector $\boldsymbol{v}$:

$$M \cdot \boldsymbol{v} = \begin{bmatrix} 0.4 \cdot 0.75 + 0.6 \cdot 0.25 \\ 0 \\ 0.4 \cdot 0.25 + 0.6 \cdot 0.75 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.45 \\ 0 \\ 0.55 \\ 0 \end{bmatrix},$$

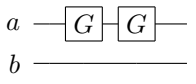
representing 45% in state '00' and 55% in state '10', which is precisely the same result we obtained before! Indeed, matrix-vector multiplication correctly propagates the probabilities.

We note that because the input vector represents a probability distribution over bitstrings, its entries should be probabilities, i.e., nonnegative values between zero and 1, and the vector should be *normalized* (all its entries should sum to 1). Also, each column in the transition matrix has the same property, as the sum of outgoing-edge labels of a node in the automaton constitute a probability distribution themselves too.

Key message: An n-bit reversible circuit with probabilistic gates is represented by a transition matrix. If the input to the circuit is a probability distribution over bitstrings, it can be represented by a 2^n-length vector containing the probabilities. Furthermore, the action of the circuit is computed by applying the transition matrix to the input vector.

Finally, in order to compute the probability of ending in a certain state, say '10', after starting in $\boldsymbol{v}$, we can compute the following: $\text{'10'}^\top \cdot M \cdot \boldsymbol{v} \equiv \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix} \cdot M \cdot \boldsymbol{v} = 0.55$. Note the use of a row vector ($\text{'10'}^\top$) and a column vector ($\boldsymbol{v}$).

Sequential Composition of Circuits Using Matrix Multiplication. Suppose that the input probability distribution over bitstrings $\boldsymbol{v}$ is inputted to circuit A, yielding the output vector $\boldsymbol{w} = M_A\boldsymbol{v}$, where M_A is the transition matrix of A. Suppose furthermore that we pass input $\boldsymbol{w}$ to another circuit B. Then the probability vector representing B's output is $M_B\boldsymbol{w} = M_B(M_A\boldsymbol{v})$, which can also be written as $(M_B \cdot M_A)\boldsymbol{v}$, where $\cdot$ denotes matrix multiplication. Since this holds for any input vector $\boldsymbol{v}$, we observe that a circuit C, consisting of first performing A followed by B, has a transition matrix $M_B \cdot M_A$. That is, the sequential composition of gates, and thus of circuits, corresponds to the matrix multiplication of their transition matrices. For example, in (v) the transition matrix is denoted as M and G is applied to the top bit a once, so applying G twice instead yields M^2:



$$M^2 = \begin{bmatrix}0.75 & 0 & 0.25 & 0\\ 0 & 0.75 & 0 & 0.25\\ 0.25 & 0 & 0.75 & 0\\ 0 & 0.25 & 0 & 0.75\end{bmatrix}^2 = \begin{bmatrix}0.625 & 0 & 0.375 & 0\\ 0 & 0.625 & 0 & 0.625\\ 0.375 & 0 & 0.375 & 0\\ 0 & 0.375 & 0 & 0.625\end{bmatrix}$$

Parallel Composition of Circuits Using the Kronecker Product. Above, we found that the sequential composition of circuits corresponds to the matrix-multiplication product of their transition matrices. For parallel composition, where one stacks two circuits 'on top' of each other, we need the Kronecker product of matrices.

Kronecker product. The Kronecker product of matrices A and B performs a giant case distinction: for each entry a of A, we create a copy of all possible entries b of B, and the resulting matrix contains the values $a \cdot b$. For example, the transition matrices on a single bit

$$A = \begin{bmatrix}0.1 & 0.2\\ 0.9 & 0.8\end{bmatrix}, \qquad B = \begin{bmatrix}0.3 & 0.4\\ 0.7 & 0.6\end{bmatrix}$$

is

$$A \otimes B = \begin{bmatrix} 0.1 \cdot B & 0.2 \cdot B \\ 0.9 \cdot B & 0.8 \cdot B \end{bmatrix} = \begin{bmatrix} 0.1 \cdot 0.3 & 0.1 \cdot 0.4 & 0.2 \cdot 0.3 & 0.2 \cdot 0.4 \\ 0.1 \cdot 0.7 & 0.1 \cdot 0.6 & 0.2 \cdot 0.7 & 0.2 \cdot 0.6 \\ 0.9 \cdot 0.3 & 0.9 \cdot 0.4 & 0.8 \cdot 0.3 & 0.8 \cdot 0.4 \\ 0.9 \cdot 0.7 & \underline{0.9 \cdot 0.6} & 0.8 \cdot 0.7 & 0.8 \cdot 0.6 \end{bmatrix}.$$

For example, the entry of $A \otimes B$ at column 2 ($01 = 0_A 1_B$ in binary) and row 4 ($1_A 1_B$), which is underlined above, contains the probability $0.9 \cdot 0.6$ which is the probability that *both* A maps bit 0 to 1 (which happens with probability 0.9) as well as B maps 1 to 1 (which happens with probability 0.6).

Formally, the Kronecker product on two general matrices (not necessarily transition matrices) acts 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

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

The product behavior of the Kronecker product of two matrices A and B can be interpreted as choosing an entry a from A and an entry b from B and multiplying them (see the yellow box above). When A and B are probability vectors of the individual systems, their Kronecker product yields precisely the vector on the combined system, where the values $a \cdot b$ are the occurrence probabilities of these automaton states/unit vectors. The same interpretation shows that the parallel composition of two gates is also given by their Kronecker product. For example, the transition matrix for G in (iii) (which acts on a single bit) is given by $M_G = \left[\begin{smallmatrix} 0.75 & 0.25 \\ 0.25 & 0.75 \end{smallmatrix}\right]$, the transition matrix on the second bit (doing nothing) is the identity matrix $I = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$; hence we may write $M_G \otimes I$ for the transition matrix on the two bits a, b.

Exercise: Using the definition of the Kronecker product sign $\otimes$ (see the yellow box above), verify that $M_G \otimes I = M$ from Eq. 1.

Exercise: For a single bit, the two basis vectors are '0' $\equiv \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$ and '1'$\equiv \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. Compute the 4 possible Kronecker products between these states ('0' $\otimes$ '0', '0 $\otimes$ '1', etc.) and verify that these are precisely the four basis vectors for two bits from Eq. 1.

Key message: Composing probabilistic circuits sequentially corresponds to multiplying their transition matrices, while their parallel composition is represented by the Kronecker product of the transition matrices.

2.1.2 Quantum Computations with Linear Algebra Now let us move to quantum bits or qubits. *Here, we limit the presentation to a simplified model of quantum computing using solely real numbers. This model is universal for quantum computing* [1] *and suffices to explain all aspects of quantum computing. Nevertheless, a reader should keep in mind that the usual quantum computing model contains complex numbers and, therefore, comprises the subtle differences noted below (also see reading material referenced at the end of Sect. 4).*

The state of two qubits is a generalization of a probability distribution over the four bitstrings, whose vectors, e.g., for '00', '01', '10' and '11' in Eq. 1, are referred to as the *computational-basis states* as they form a basis of the space of 4-dimensional vectors, and written as $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ (so-called Dirac notation). Just like in the Markov chain case above, the state on n quantum bits is represented by a vector of length 2^n. However, in contrast to the Markov chain case, the entries of the vector are not probabilities (which sum up to 1) but are so-called *probability amplitudes*, or simply *amplitudes*. So to represent a quantum state, the vector should now be normalized by letting the *squares*[3] of these amplitudes sum up to 1. Instead of a transition matrix, which contains probabilities such that the entries in each column add up to 1, the gate in the circuit is given by a *unitary matrix*: in a unitary matrix, the sum of *squares* of the entries (amplitudes) in a column add up to 1.[4] Sequential and parallel composition of gates, identically to the case of probabilistic computing in Sect. 2.1.1, correspond to matrix multiplication and Kronecker product, respectively.

An example is the circuit below: the input is computational-basis state $|a\rangle \otimes |b\rangle = |ab\rangle$ for $a, b \in \{0, 1\}$, and the single-qubit quantum gate U in the circuit yields the two-qubit action M' (to see why $M' = U \otimes I$, compare with $M_G \otimes I$ in Sect. 2.1.1):

$|a\rangle$ — $\boxed{U}$ —
$|b\rangle$ ———

$$U = \begin{bmatrix} \sqrt{0.75} & \sqrt{0.25} \\ \sqrt{0.25} & -\sqrt{0.75} \end{bmatrix} \qquad M' = U \otimes I = \begin{bmatrix} \sqrt{0.75} & 0 & \sqrt{0.25} & 0 \\ 0 & \sqrt{0.75} & 0 & \sqrt{0.25} \\ \sqrt{0.25} & 0 & -\sqrt{0.75} & 0 \\ 0 & \sqrt{0.25} & 0 & -\sqrt{0.75} \end{bmatrix}$$

For example, if the input state is $|11\rangle \equiv \left[0, 0, 0, 1\right]^\top$, then the output state is $(U \otimes I) \cdot |11\rangle = \left[0, \sqrt{0.25}, 0, -\sqrt{0.75}\right]^\top = \sqrt{0.25}|01\rangle - \sqrt{0.75}|11\rangle$. By squaring the amplitudes $\sqrt{0.25}$ and $\sqrt{0.75}$, we regain the interpretation as probability

[3] Or modulus square in the general case with complex numbers.
[4] The rows of a unitary matrix are also orthogonal; see Sect. 2.2 for full definition.

distribution over the states $|01\rangle$ and $|11\rangle$. *However*, the fact that the amplitudes can be *negative* (and complex numbers in general) is crucial: as we will see later, these give rise to *interference*, where amplitudes are canceled or amplified, a feature that is not present in probabilistic computing.

Key message: quantum computing is a generalization of probabilistic computing.
It will be useful to read a(n exponential-length) state vector as a probability (amplitude) distribution over classical (computational basis) states:

$$\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \vdots \\ \alpha_{2^n-2} \\ \alpha_{2^n-1} \end{bmatrix} = \begin{bmatrix} \alpha_{0\ldots00} \\ \alpha_{0\ldots01} \\ \vdots \\ \vdots \\ \alpha_{1\ldots10} \\ \alpha_{1\ldots11} \end{bmatrix} \begin{array}{ll} \rightarrow & \text{probability amplitude of } |0\ldots00\rangle \\ \rightarrow & \text{probability amplitude of } |0\ldots01\rangle \\ & \\ & \\ & \\ \rightarrow & \text{probability amplitude of } |1\ldots10\rangle \\ \rightarrow & \text{probability amplitude of } |1\ldots11\rangle \end{array}$$

2.2 Building Blocks of Quantum Circuits

In the previous section, we gave an example of a quantum state as a generalization of a probability distribution over bitstrings. Here we give the full definition of a general quantum state (state of a collection of qubits), and give the building blocks of quantum-computer circuits: the quantum analogs of logical *gates*, and *measurements* as a means to extract information from a quantum state. (For a full introduction, see [24].)

2.2.1 Quantum Bits As we have seen before, a quantum state on n qubits is a column vector of length 2^n, which constitutes a probability distribution over the length-n bitstrings $b \in \{0,1\}^n$ as the vector is a *linear combination* $\sum_{b\in\{0,1\}^n} \alpha_b|b\rangle$ over the corresponding computational-basis states $|b\rangle$ (i.e. the length-2^n vector with entry 1 at position b and 0 everywhere else). (The quantum-computing jargon is *superposition* instead of linear combination.) The vector entries (amplitudes) α_b are complex numbers in general. In this paper, we will always consider real vector entries. The square of the amplitude determines the probability. For example, the following vectors

$$\begin{bmatrix} -\sqrt{0.75} \\ \sqrt{0.25} \end{bmatrix}, \quad \begin{bmatrix} -\sqrt{0.75} \\ -\sqrt{0.25} \end{bmatrix}, \quad \begin{bmatrix} \sqrt{0.75} \\ \sqrt{0.25} \end{bmatrix} \tag{2}$$

all give rise to the probabilities 0.75 and 0.25 over the states $|0\rangle$ and $|1\rangle$, respectively. If the amplitudes α_b of a vector $\boldsymbol{v}$ are real numbers, then $\sqrt{\sum_{b\in\{0,1\}^n} \alpha_b^2}$

indicates the vector's length (for complex numbers, we would also have to take the modulus $|\alpha_b|$ instead of α_b). Since the values $|\alpha_b|^2$ form a probability distribution, we find that:

Key message: an n-qubit state is a vector of 2^n complex numbers with norm 1. (In this paper, we only consider real vectors.)

Example 1. The vectors $|\phi\rangle, |\psi\rangle$ and $|\eta\rangle$ are quantum states on 2, 2 and 3 qubits, respectively:

$$|\phi\rangle \triangleq \frac{1}{\sqrt{1^2+2^2+3^2+\sqrt{17}^2}} \cdot \begin{bmatrix} 1 \\ 2 \\ -3 \\ \sqrt{17} \end{bmatrix} = \frac{1}{\sqrt{31}} \left(1 \cdot |00\rangle + 2 \cdot |01\rangle - 3 \cdot |10\rangle + \sqrt{17}|11\rangle\right)$$

$$|\psi\rangle \triangleq \frac{1}{2} \cdot \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix}, \qquad |\eta\rangle \triangleq \frac{1}{\sqrt{2}} \cdot \left[1,0,0,0,0,0,0,1\right]^\top = \frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle.$$

◇

Consider a set of six qubits and suppose that the first two are jointly in some arbitrary state $|\phi\rangle$ (a vector of length $2^2 = 4$) and the remaining four in some state $|\psi\rangle$ (a vector of length $2^4 = 16$). Identically to the case of probabilistic computing (Sect. 2.1.1), we use the Kronecker product $\otimes$ to find the joint state on the six qubits, which is $|\phi\rangle \otimes |\psi\rangle$ (a vector of length $2^2 \cdot 2^4 = 2^6$).

Exercise: Show that the computational-basis states $|b\rangle$ for $b = (b_1, b_2, \ldots, b_n) \in \{0,1\}^n$ can be written as $|b\rangle = |b_1\rangle \otimes |b_2\rangle \otimes \cdots \otimes |b_n\rangle$.

2.2.2 Manipulating Quantum States Using Gates A quantum gate on n qubits is represented by a 2^n by 2^n unitary matrix U (unitarity defined below). A quantum state $|\phi\rangle$ is updated by a unitary matrix as $U \cdot |\phi\rangle$ where $\cdot$ denotes matrix-vector multiplication.

Recall from Sect. 2.1 that the three-(qu)bit $CCNOT$ gate (or doubly control-NOT, or Toffoli gate) is universal for classical computing, while the NOT and $CNOT$ gates can be added as syntactic sugar. Below we give their matrices. To obtain a universal gate set for quantum computing, we merely need to extend this gate set with the single-qubit *Hadamard gate*: $H \triangleq \frac{1}{\sqrt{2}} \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right]$. Example 2

illustrates its function: realizing superpositions.

$$NOT \triangleq \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad CNOT \triangleq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad CCNOT \triangleq \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Example 2. Applying the H gate to the state $|0\rangle$ or $|1\rangle$, we obtain

$$H \cdot |0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle),$$

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

◇

A square matrix U is unitary if it is invertible and its inverse is $U^\dagger$, where $U^\dagger$ is found by transposing U for real matrices.[5] Unitarity ensures that the matrix is norm-preserving, thus guaranteeing that the output quantum state has norm 1 if the input state does so too.[6] As unitary matrices are reversible, we directly see that all quantum gates are also reversible. This relates quantum computing to reversible computing. The Hadamard gate and NOT gate have adjoint operators $H^\dagger = \frac{1}{\sqrt{2}} \left[\begin{smallmatrix} 1 & 1 \\ 1 & -1 \end{smallmatrix}\right] = H$ and $NOT^\dagger = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right] = NOT$; it is not hard to check, indeed, that $H^\dagger \cdot H = NOT^\dagger \cdot NOT = I = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$.

To apply a single-qubit gate to one qubit of a state on $n > 1$ qubits, one uses the Kronecker product (see Sect. 2.1) for 'padding' the gate with an identity matrix I for each other qubit, i.e. the matrices which leave any vector unchanged. For example, applying H to the first qubit of $|010\rangle$ is computed as $(H \otimes I \otimes I)|010\rangle$.

Exercise: Compute the unitary matrix $H \otimes I \otimes I$. Remark that it has dimensions $2^3 \times 2^3$, as is needed for being able to apply it to a 3-qubit state.

[5] The adjoint $U^\dagger$ of a complex matrix U is found by transposing U, followed by replacing each matrix entry $u + wi$ with its complex conjugate $u - wi$. As we use only real matrices in this paper, one can always think of the adjoint as the transpose. The adjoint notation $U^\dagger$ is used in this paper, in favor of the inverse U^{-1}, as this is common in the quantum community.

[6] Quantum states should have norm 1 to be physically meaningful since the probabilites over the computational-basis states should sum up to 1. Further, quantum gates need to be unitary to obey the energy conservation law, due to quantum mechanical properties following from the famous Schödinger equation.

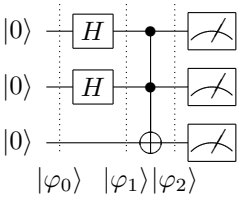
Finally, we remark that sequential composition of gates is represented by matrix multiplication of their unitaries, similar to matrix multiplication in the Markov chain case in Sect. 2.1: applying first U and then V to $|\phi\rangle$ yields the state $V \cdot (U \cdot |\phi\rangle) = (V \cdot U)|\phi\rangle$.

2.2.3 Measurement Measurement enables one to extract classical information (bits) from a quantum state. The theory of quantum measurement allows many ways to do this and here we only focus on a common one: measuring all qubits in the computational basis. Given an n-qubit quantum state $\sum_{b\in\{0,1\}^n} \alpha_b|b\rangle$, a measurement (on all qubits) is a probabilistic operation that returns one of the values $b \in \{0,1\}^n$, called the *measurement outcome*. The value b is returned with probability $(\alpha_b)^2$. (Or $|\alpha_b|^2$ in the case of complex amplitudes.) For example, the probability of measuring $00\ldots0$ is $(\alpha_{00\ldots0})^2$. Since each quantum state vector has unit norm, these probabilities add up to 1, as desired.

Example 3. Consider the state $\left[\frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, 0, \frac{1}{2}\right]^\top = \frac{1}{2}|000\rangle + \frac{1}{2}|010\rangle + \frac{1}{2}|100\rangle + \frac{1}{2}|111\rangle$. Upon measurement, the probability to obtain measurement outcome 000 is $(\alpha_{000})^2 = \left(\frac{1}{2}\right)^2 = \frac{1}{4}$ and the probability to obtain measurement outcome 001 is $(\alpha_{001})^2 = 0^2 = 0$. ◇

2.2.4 Quantum Circuit A quantum circuit is composed of qubits represented by horizontal lines (wires) and quantum gates represented by boxes, with each gate acting on one or more qubits. The circuit is finished with a measurement. We will always let the input state be $|0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle$ (which is usually written as $|0\rangle^{\otimes n}$ or $|00\ldots0\rangle$), and the output state is obtained after the sequential application of the circuit's gates. All gates can be combined with matrix multiplication into a single unitary operator describing the entire circuit.

Example 4. In the following circuit, we apply a Hadamard to the first two qubits of $|000\rangle$. Then we apply a Toffoli gate ($CCNOT$) to the three qubits, followed by an all-qubit measurement. The intermediate states are as follows; see Example 3 for evaluating measurement.



$$|\varphi_0\rangle = |000\rangle$$

$$|\varphi_1\rangle = (H^{\otimes 2} \otimes I)|\varphi_0\rangle = \frac{1}{2}(|000\rangle + |010\rangle + |100\rangle + |110\rangle)$$

$$|\varphi_2\rangle = CCNOT|\varphi_1\rangle = \frac{1}{2}(|000\rangle + |010\rangle + |100\rangle + |111\rangle)$$

◇

We emphasize that quantum circuits are reversible: each gate, and hence each circuit, has the same number of input and output qubits.

Suppose that we have an n-qubit circuit C initialized to the all-zero state $|00\ldots0\rangle$. Computing the probability of outcome $b \in \{0,1\}^n$ can now be written as follows: $|\langle b|C|00\ldots0\rangle|^2$. Here, $\langle b|$ is a row vector of the computational basis state $|b\rangle$, so $\langle b|C|00\ldots0\rangle$ can be seen as a product of a row vector, matrix and column vector, resulting in a scalar.

Example 5. Let C be the circuit from Example 4. Then $C|000\rangle$ equals $\frac{1}{2}|000\rangle + \frac{1}{2}|010\rangle + \frac{1}{2}|100\rangle + \frac{1}{2}|111\rangle = \left[\frac{1}{2}, 0, \frac{1}{2}, 0, \frac{1}{2}, 0, 0, \frac{1}{2}\right]^\top$. Hence, the probability of measuring 000 is $|\langle 000|C|000\rangle|^2 = \frac{1}{4}$ and the probability of measuring 001 is $|\langle 001|C|000\rangle|^2 = 0$.

Note that, as in Sect. 2.1.1, we use a row vector for the measured computational basis state and a column vector for the initial state of the circuit.

Key message: an n-qubit state is transformed by $2^n \times 2^n$ matrices through matrix-vector multiplication. Measurement allows one to extract information from the state.

We will now take a first step towards using quantum circuits to our advantage. We focus on Boolean satisfiability (SAT), and will provide a naive approach to solving SAT using a quantum computer. Although this approach will not work, it will show how the *single* evaluation of the Boolean circuit on a quantum state will evaluate the Boolean function on all *exponentially-many* bitstrings as input.

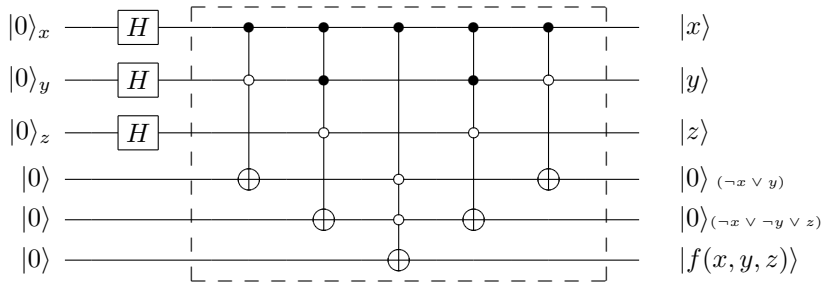
Consider the 3-CNF formula $f(x,y,z) = x \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y \vee z)$. We will use a reversible circuit to implement this function as a quantum circuit. We usually call such quantum circuit a *function oracle*. We want to compute f for every possible input $000, 001, \ldots, 111$. Therefore, we create a superposition of these states. This is done by applying a Hadamard gate to the qubits representing the input (see exercise below).

Now we will see what happens if we apply the function oracle f to a superposition of input states. Example 4 contains a simple example of a quantum circuit calculating $f = x \wedge y$. A more complicated example is the following one:

Example 6. Consider the 3-CNF formula

$$f(x,y,z) = x \wedge (\neg x \vee y) \wedge (\neg x \vee \neg y \vee z). \tag{3}$$

The following circuit first applies Hadamard gates to get a superposition on the input qubits. Then it applies the function oracle for f (in the dashed rectangle).



The resulting state is $\sum_{x,y,z\in\{0,1\}^3} |x,y,z\rangle \otimes |f(x,y,z)\rangle$, or, written out in full,

$$\frac{1}{2\sqrt{2}}(|0000\rangle + |0010\rangle + |0100\rangle + |0110\rangle + |1000\rangle + |1010\rangle + |1100\rangle + |1111\rangle). \quad (4)$$

(Here, we omitted the two auxiliary qubits uncomputed to $|0\rangle$). Note that the single satisfying assignment (111) represents the solution to the satisfiability problem for f, because in Example 4, the only term which has a '1' at the fourth qubit, corresponding to $f(x,y,z)$, is $|1111\rangle$.

Exercise: Evaluate the circuit above in three steps. **(a)** Show that applying $H \otimes H \otimes H$ to the input $|000\rangle$ (the top three qubits in the circuit) yields a superposition over all 3-qubit computational-basis states. **(b)** Next, find the state when adding the register of the bottom three qubits (which is in the state $|000\rangle$) using the Kronecker product. **(c)** Lastly, using the Boolean function f as given in Eq. 3 (the circuit in the dashed rectangle above), verify that the resulting state indeed is Eq. 4.

We would like to extract the satisfying assignment '111' using measurement. Unfortunately, we see that measuring gives outcome 1111 with a probability of only $\left|\frac{1}{2\sqrt{2}}\right|^2 = \frac{1}{8}$. This probability is $\frac{1}{2^n}$ in general, where n is the number of variables to f. Thus, finding a satisfiable instance this way only succeeds with exponentially-small probability, much like the classical naive approach of randomly guessing bitstrings as input and evaluating f on them. ◇

Boolean satisfiability & quantum computing. In Example 6 above, we saw a naive approach to solving Boolean satisfiability using quantum computing, which performed equally well as random guessing. However, using the famous quantum algorithm by Grover [15], the probability of measuring a satisfying assignment of f can be increased to arbitrarily high

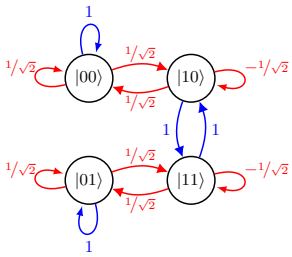
probability. Since every CNF formula f can be efficiently encoded as a reversible circuit, this solves the satisfiability problem for f. The Grover algorithm has runtime $O^*(\sqrt{2^n})$, where O^* omits polynomial factors. This is a quadratic speed-up compared to brute force and the best-known classical algorithm for k-SAT where k is unbounded.[a]

[a] Schöning's [32] and the PPSZ [26] algorithm deliver better guarantees for constant k, e.g., for 3-SAT, and can also be quantized [29].

2.3 Visualizing a Quantum Computation Using an Automaton

Armed with a well-defined notion of a quantum circuit, we can now continue the comparison with Markov chains in Sect. 2.1 and visualize an n-qubit quantum gate as an automaton: it has 2^n states, one for each computational-basis state, and we interpret the matrix of each gate U as the weighted adjacency matrix between these states. For example, consider 2 qubits, acted upon by the $CNOT$ gate and by the H gate on the first qubit:

$$H \otimes I = \begin{bmatrix} 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 0 & 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 1/\sqrt{2} & 0 & -1/\sqrt{2} & 0 \\ 0 & 1/\sqrt{2} & 0 & -1/\sqrt{2} \end{bmatrix}, \quad CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



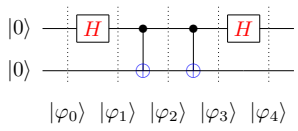
We will now use the following known result from graph theory:

> The entry in the product of adjacency matrices at row r and column c equals the sum of products of edge labels (a *path sum*) over all paths from node r to node c.

This somewhat complicated statement tells us that matrix multiplication can be visualized as path sums, a result we already implicitly used in the Markov chain case in Sect. 2.1.1 when observing that sequential composition of gates is represented by multiplication of transition matrices. To illustrate how this statement results in a visualization of quantum gates as paths in the automaton, we give the following example.

Example 7. Consider the circuit below; we will compute the entries in $(H \otimes I) \cdot CNOT \cdot CNOT \cdot (H \otimes I)$ corresponding to the transitions $|00\rangle \to |00\rangle$ and $|00\rangle \to |10\rangle$, first using matrix-vector multiplication, and then using paths

in the automaton. For the former, it is straightforward to derive that $|\varphi_1\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$, hence $|\varphi_3\rangle = \frac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot |00\rangle + \frac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot |10\rangle$ (the second CNOT uncomputes the first). We use this to compute $|\varphi_4\rangle = (H \otimes I)|\varphi_3\rangle$ below.



$$
\begin{aligned}
|\varphi_4\rangle &= \tfrac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot (\tfrac{1}{\sqrt{2}}|00\rangle + \tfrac{1}{\sqrt{2}}|10\rangle + \tfrac{1}{\sqrt{2}}|00\rangle - \tfrac{1}{\sqrt{2}}|10\rangle) \\
&= \left(\tfrac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot \tfrac{1}{\sqrt{2}} + \tfrac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot \tfrac{1}{\sqrt{2}}\right)|00\rangle \\
&\quad + \left(\tfrac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot \tfrac{1}{\sqrt{2}} - \tfrac{1}{\sqrt{2}} \cdot 1 \cdot 1 \cdot \tfrac{1}{\sqrt{2}}\right)|10\rangle \qquad (5) \\
&= \left(\tfrac{1}{2} + \tfrac{1}{2}\right)|00\rangle + \left(\tfrac{1}{2} - \tfrac{1}{2}\right)|10\rangle \qquad (6)
\end{aligned}
$$

Now the gate sequence $H - CNOT - CNOT - H$ means we should consider 'red-blue-blue-red' paths in the automaton. The factor $\frac{1}{2} + \frac{1}{2}$ in front of $|00\rangle$ in Eq. 6 is mirrored in the automaton by noting that there are two paths from $|00\rangle$ to itself: $(|00\rangle) \xrightarrow{1/\sqrt{2}} (|10\rangle) \xrightarrow{1} (|11\rangle) \xrightarrow{1} (|10\rangle) \xrightarrow{1/\sqrt{2}} (|00\rangle)$ and $(|00\rangle) \xrightarrow{1/\sqrt{2}} (|00\rangle) \xrightarrow{1} (|00\rangle) \xrightarrow{1} (|00\rangle) \xrightarrow{1/\sqrt{2}} (|00\rangle)$. These paths both have amplitude $1/\sqrt{2} \cdot 1 \cdot 1 \cdot 1/\sqrt{2} = 1/2$, and their sum $1/2 + 1/2$ is the amplitude of $|00\rangle$ in $|\varphi_4\rangle$ in Eq. 6 indeed (also note that the two products $1/2 \cdot 1 \cdot 1 \cdot 1/2$ are precisely the terms in front of $|00\rangle$ in Eq. 5). In contrast, the factor $1/2 - 1/2$ in front of $|10\rangle$ in Eq. 5 arises in the automaton as two paths from $|00\rangle$ to $|10\rangle$, one with amplitude $1/\sqrt{2} \cdot 1 \cdot 1 \cdot 1/\sqrt{2}$, and one with amplitude $1/\sqrt{2} \cdot 1 \cdot 1 \cdot -1/\sqrt{2}$, which are precisely the terms in front of $|10\rangle$ in Eq. 5. ◇

Note that in the example above, the path sums have opposite sign, so they precisely cancel each other, implying that the transition $|00\rangle \to |10\rangle$ has amplitude zero! This cannot happen in a Markov chain as probabilities are never negative.

We thus see that the transition amplitude from state $|x\rangle$ to state $|y\rangle$, which we first found through linear algebra, can be found in the automaton by summing path contributions from $(|x\rangle)$ to $(|y\rangle)$, where each path contribution is the product of the edge labels of the path.

The automaton is useful because it shows a few properties of quantum circuits:

1. an **exponentially-sized state space** as function of number of qubits
2. the **combinatorial nature of the evolution of a quantum state through a circuit:** there can be **many paths** from one node to another, sometimes exponentially-many, and we need to track all of them to compute the output state's amplitudes
3. the many paths from one node to another will **interfere** as amplitudes, either **constructively** (the amplitudes amplify through addition, as in the example above for computing the amplitude of $|00\rangle$) or **destructively** (the amplitudes cancel, as for $|10\rangle$ above).

Item 1 and 2 also occur for probabilistic computation (see the Markov chain in Sect. 2.1) but item 3 is where quantum computing differs. Automated reasoning methods were often developed to solve scenarios where items 1 and 2 are present; we will see one approach in Sect. 3 that also illustrates item 3.

Key message: during a run of a quantum circuit, there can be exponentially many paths leading from one state to another. The amplitude contributions of these paths can constructively or destructively interfere.

2.4 Towards a Quantum Advantage

Application of a gate G once to a superposition of many computational-basis states $|b\rangle$ yields a sum of terms of the form $G|b\rangle$. This realization is particularly astounding in case we start out with an exponentially-large superposition, which can be created already with only linearly many gates in the number of qubits (recall the exercise above to compute $H^{\otimes n}|0\rangle^{\otimes n}$ for $n = 3$). However, using measurement, we can only read off a single bitstring from this superposition.

A plethora of quantum algorithms[8] [28] has been found whose quantum circuits provably need polynomially-fewer or exponentially-fewer gates than their classical counterpart.[9] Real-world quantum devices suffer from noise, and counteracting that noise might require additional resources that cancel the complexity-theoretic and real-world advantages of quantum computing. Arguably **the main open problem of quantum technologies is therefore to provide a real-world demonstration of a quantum advantage**. On the road to this goal, there are several tasks where formal methods could help: performance prediction of real-world devices through classical simulation of quantum circuits (sampling or computing the measurement distribution); optimizing circuits (e.g. fewer gates, circuit layouts following the topology of real-world chips, etc.); verification, specifically checking if two quantum circuits implement the same unitary matrix; finding a circuit that outputs a desired quantum state; transpilation to a gate set that real-world devices can natively run; etc.

In the remainder of this tutorial, we will focus on using weighted model counting or #SAT (Sect. 3) to perform classical simulation of quantum circuits.

[8] A quantum algorithm is a uniform family of quantum circuits (like in circuit complexity [2]).

[9] The exact statements depend on the used model (for example, whether the input is promised to be picked from a certain set). Quantum algorithms are typically complex, so we decided they are out of scope for this tutorial, where we aim to focus on analyzing quantum circuits using formal methods tools.

2.5 Our Scope: Quantum Circuit Simulation

In this tutorial, we will mainly consider the task of *classically simulating a quantum circuit* (that is, to design classical algorithms for this task). Formally, the task of simulating an n-qubit quantum circuit C is to find the probability of outcome $b \in \{0,1\}^n$ when the output state of C is measured, assuming that $|0\rangle^{\otimes n}$ is the input quantum state to C. Although simulation is #P-hard in general [23], so are many problems in formal methods, where solutions have been found that work well in practice. We should thus not be discouraged from tackling simulation and will see how to do so with #SAT in Sect. 3. In Sect. 4, we refer to extensions that tackle other important quantum circuit analysis tasks.

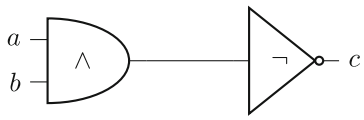
We focus here on *strong simulation* [7]: the problem of returning a probability for a certain computational basis state. In contrast, *weak simulation* asks to sample the probability distribution of measurement outcomes.

3 Reducing Quantum Computing to #SAT

Here we reduce quantum-circuit simulation to weighted model counting (weighted-#SAT). This section is partly based on [21], while the approach here effectively realizes the well-known path-sum approach [12].

3.1 SAT and #SAT

We denote $SAT(F) := \{\alpha \mid F(\alpha) = 1\}$ for the set of all satisfiable assignments of a propositional formula $F\colon \{0,1\}^V \to \{0,1\}$ over a finite set of Boolean variables V. We say that F is *satisfiable* if $SAT(F)$ is non-empty. We write an assignment α as a *cube* (a conjunction of literals, i.e., positive or negative variables), e.g., $a \wedge b$, or shorter ab.



The action of a classical circuit can be encoded by SAT constraints directly by representing each bit as a Boolean variable. For example, the Boolean constraint for the classical circuit C on the right is $F_C(V) = c \Leftrightarrow \neg(a \wedge b)$ over variables $V = \{a, b, c\}$. Given an input $a = 0$ and $b = 1$, the satisfying assignment is $\alpha = \overline{a}bc$, where $\alpha(c) = 1$ is the final state.

We denote $\#SAT(F) \triangleq |SAT(F)|$ for the *model count* of a formula F. A weight function $W\colon \{\overline{v}, v \mid v \in V\} \to \mathbb{R}$ assigns a real-valued weight to positive literals v (i.e., $v = 1$) and the negative literals $\overline{v}$ (i.e., $v = 0$). We say variable v is

unbiased iff $W(v) = W(\overline{v}) = 1$. Given an assignment $\alpha \in \mathbb{B}^V$, let $W(\alpha(v)) = W(v := \alpha(v))$ for $v \in V$. For a propositional formula F over V and a weight function W, we define *weighted model counting* (#SAT) as:

$$\#SAT_W(F) \triangleq \sum_{\alpha \in SAT(F)} W(\alpha) \text{ where } W(\alpha) = \prod_{v \in V} W(\alpha(v)).$$

Example 8. The propositional formula $F = (v_1 \vee v_2) \wedge (\overline{v_1} \vee v_2) \wedge v_3$ over $V = \{v_1, v_2, v_3\}$ has two satisfying assignments: $\alpha_1 = v_1 v_2 v_3$ and $\alpha_2 = \overline{v_1} v_2 v_3$. We define the weight function W as $W(v_1) = -\frac{1}{2}$, $W(\overline{v_1}) = \frac{1}{3}$ and $W(v_2) = \frac{1}{4}$, $W(\overline{v_2}) = \frac{3}{4}$, while v_3 remains unbiased. The weight of F can be computed as $MC_W(F) = -\frac{1}{2} \times \frac{1}{4} \times 1 + \frac{1}{3} \times \frac{1}{4} \times 1 = -\frac{1}{24}$.

Why #SAT for tackling simulation? First, analysis tasks on quantum circuits are inherently functional problems, as the outcome often is a measurement probability (or amplitude). For the same reason, inference on Bayesian Networks [30,31] was done with weighted model counting. Here we use a similar approach by reducing the simulation of quantum circuits to weighted model counting.
In our #SAT encoding, we let each satisfying assignment encode a path in the automaton as discussed in Sect. 2.3. We then add weighted variables so that the weight for each assignment (representing a path) equals a part of the circuit's final amplitude. We also let the measurements constrain the encoding so that only the required paths remain. The weighted model counter ensures that the weights of all paths contributing to the measurement outcome are summed up (positive and negative).
Our encoding uses only linearly many variables and clauses in the number of gates plus the number of qubits. So, to encode a single gate, we require only a constant amount of clauses (around four), while encoding measurements require n (unit) clauses, one per qubit.

3.2 Encoding Quantum States Using (Weighted) Boolean Variables

Recall from Sect. 2 that a quantum state $|\phi\rangle$ is a specific linear combination of classical states, i.e.,

$$|\phi\rangle = \sum_{b \in \{0,1\}^n} \alpha_b |b\rangle.$$

Accordingly, we can simply reserve a single Boolean variable for every qubit and let the satisfying assignments of our formulae represent the state vector. Example 9 illustrates this. In what follows, we will write $F_{|\phi\rangle}$ for a similar encoding of $|\phi\rangle$.

Example 9. Let $|B\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |11\rangle)$. Let x be the variable for the first qubit and y be the variable for the second qubit. Written differently, we have: $|B\rangle = (\frac{1}{\sqrt{2}}|0\rangle_x - \frac{1}{\sqrt{2}}|1\rangle_x) \otimes |1\rangle_y$. The corresponding Boolean constraint is $F_{|B\rangle} = (x \vee \overline{x}) \wedge y$ where we assign the $W(x) = -\frac{1}{\sqrt{2}}$ and $W(\overline{x}) = \frac{1}{\sqrt{2}}$, leaving y unbiased. The (weighted) satisfying assignments are: $\{\overline{x}y \equiv \frac{1}{\sqrt{2}}|01\rangle, xy \equiv -\frac{1}{\sqrt{2}}|11\rangle\}$. ◇

From now on, we will reserve the variables x, y, z for the first three qubits.

Exercise: Encode the so-called Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ in weighted model counting.

3.3 Encoding Quantum Gates and Circuits to #SAT

Since the NOT, $CNOT$ and $CCNOT$ (Toffoli) gates are classical (reversible) gates, their encoding is easy. For instance, the Toffoli gate on input bits x, y, z and output bits x', y', z', only flips z, i.e., sets $z' \Leftrightarrow \neg z$, when both x and y are set to true, as explained in Sect. 2. Nothing changes in the quantum setting, except that we reserve the same variables now for the qubits. The Boolean encoding of these gates is thus as follows.

$$\begin{array}{lrcl} F_{NOT} & (x, x') & \triangleq & x' \Leftrightarrow x \oplus 1 \quad = \quad x' \Leftrightarrow \neg x \\ F_{CNOT} & (x, y, x', y') & \triangleq & y' \Leftrightarrow y \oplus x \wedge x' \Leftrightarrow x \\ F_{CCNOT} & (x, y, z, x', y', z') & \triangleq & z' \Leftrightarrow z \oplus (y \wedge z) \wedge x' \Leftrightarrow x \wedge y' \Leftrightarrow y \end{array} \tag{7}$$

Observe that the above encoding determines the output variables x', y', z' in terms of the input variables, as illustrated in the following example.

Example 10. Let the input state be $|110\rangle$, where the corresponding Boolean constraint is $F_{|110\rangle} = x \wedge y \wedge \neg z$. After applying $CCNOT$ gate, the output state is described by the constraint

$$F_{|110\rangle} \wedge F_{CCNOT} = (x \wedge y \wedge \neg z) \quad \wedge \quad (z' \Leftrightarrow z \oplus (y \wedge z) \wedge x' \Leftrightarrow x \wedge y' \Leftrightarrow y)$$

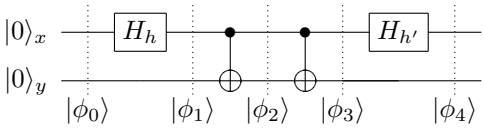
with satisfying assignment $x'y'z'$, which is interpreted as state $|111\rangle$.

In the (non-classical) case where we have superposition $\frac{1}{\sqrt{2}}(|011\rangle + |111\rangle)$, each basis state will be a satisfying assignment, e.g. yz with $W(x) = W(\overline{x}) = \frac{1}{\sqrt{2}}$, where the satisfying assignments are $\{xyz, \overline{x}yz\} \equiv \frac{1}{\sqrt{2}}(|011\rangle + |111\rangle)$. Applying a $CCNOT$ gate ends up with two computational basis states (satisfying assignments) $\{\overline{x}yz, xy\overline{z}\} \equiv \frac{1}{\sqrt{2}}|011\rangle + |110\rangle$. ◇

Recall that its gate semantics in Example 2. We will encode the gate again as a constraint $F_H(x, x', h)$, where x/x' is the qubit input/output variable and h is a separate variable representing the $\pm\frac{1}{\sqrt{2}}$ normalization. Notice in particular that the encoding should increase the number of satisfying assignments after introducing the x' variable. We achieve this by leaving x' unconstrained. The following encoding of the Hadamard gate also ensures that the negative weight only occurs for the case when $|1\rangle$ is the input and $|1\rangle$ is the output.

$$F_H(x, x', h) \quad \triangleq \quad h \iff (x \wedge x') \quad \text{with} \quad W(\overline{h}) = \frac{1}{\sqrt{2}} \quad W(h) = -\frac{1}{\sqrt{2}} \tag{8}$$

Example 11. The following circuit (identical to the one used in Sect. 2.3) computes the famous Bell state $|\phi_2\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ at time step 2, only to uncompute it again, ending up back in the $|00\rangle$ state. As will become apparent, this circuit nicely illustrates how the encoding handles constructive and destructive interference. In the circuit, we explicitly label the qubits with Boolean variables x, y and add a subscript to the Hadamard gates, to indicate that we will reserve an additional Boolean variable h or h' per gate.



We first show the satisfying assignments of the circuit encoding at each time step, where for the encoded Hadamard gates $F_H(x, x', h)$, we add an additional constraint $y \Leftrightarrow y'$ implementing the identity on the second qubit, i.e., $H \otimes I$.

$$|\phi_0\rangle \equiv F_{|\phi_0\rangle} = \neg x_0 \wedge \neg y_0 \qquad \{\overline{x}_0\overline{y}_0\}$$
$$|\phi_1\rangle \equiv F_{|\phi_1\rangle} = F_{|\phi_0\rangle} \wedge F_H(x_0, x_1, h) \wedge (y_0 \Leftrightarrow y_1) \qquad \{\overline{h}x_1\overline{y}_1, \overline{h}\overline{x}_1\overline{y}_1\}$$
$$|\phi_2\rangle \equiv F_{|\phi_2\rangle} = F_{|\phi_1\rangle} \wedge F_{CNOT}(x_1, y_1, x_2, y_2) \qquad \{\overline{h}x_1\overline{y}_1x_2y_2, \overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\}$$
$$|\phi_3\rangle \equiv F_{|\phi_3\rangle} = F_{|\phi_2\rangle} \wedge F_{CNOT}(x_2, y_2, x_3, y_3) \quad \{\overline{h}x_1\overline{y}_1x_2y_2x_3\overline{y}_3, \overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\overline{x}_3\overline{y}_3\}$$

It is worth noting that, in the final time step, the satisfying assignments of $|\phi_4\rangle \equiv F_{|\phi_4\rangle} = F_{|\phi_3\rangle} \wedge F_H(x_3, x_4, h') \wedge (y_3 \Leftrightarrow y_4)$ will be

$$\{h'\overline{h}x_1\overline{y}_1x_2y_2x_3\overline{y}_3x_4\overline{y}_4 \equiv -\tfrac{1}{2}|10\rangle, \quad \overline{h}'\overline{h}x_1\overline{y}_1x_2y_2x_3\overline{y}_3\overline{x}_4\overline{y}_4 \equiv \tfrac{1}{2}|00\rangle,$$
$$\overline{h}'\overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\overline{x}_3\overline{y}_3x_4\overline{y}_4 \equiv \tfrac{1}{2}|10\rangle, \quad \overline{h}'\overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\overline{x}_3\overline{y}_3\overline{x}_4\overline{y}_4 \equiv \tfrac{1}{2}|00\rangle\},$$

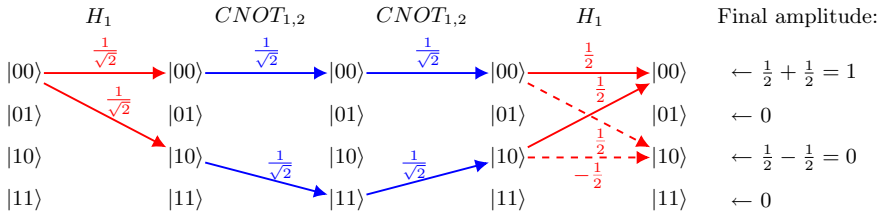
where we have $(\frac{1}{2} - \frac{1}{2})|10\rangle$ (destructive interference) and $(\frac{1}{2} + \frac{1}{2})|00\rangle$ (constructive interference). Recall in Sect. 2.3, we give the transition paths of states in the same circuit. Consider one of the paths from $|00\rangle$ to itself corresponding to the satisfying assignment $\overline{h}'\overline{h}\overline{x}_0\overline{y}_0x_1\overline{y}_1x_2y_2x_3\overline{y}_3\overline{x}_4\overline{y}_4$: $\underset{\overline{x}_0\overline{y}_0}{(|00\rangle)} \xrightarrow[\overline{h}]{1/\sqrt{2}} \underset{x_1\overline{y}_1}{(|10\rangle)} \xrightarrow{1} \underset{x_2y_2}{(|11\rangle)} \xrightarrow{1} \underset{x_3\overline{y}_3}{(|10\rangle)} \xrightarrow[\overline{h}']{1/\sqrt{2}} \underset{\overline{x}_4\overline{y}_4}{(|00\rangle)}$.

Here we omit the satisfying assignments for x_0 and y_0 in $SAT(F_{|\psi_t\rangle})$ with $t \in [1, 4]$ as each of them contains $\overline{x}_0\overline{y}_0$. We see that the satisfying assignments have a one-to-one mapping to the paths. ◇

This encoding effectively realizes the well-known path sum approach [12] to the classical simulation of quantum circuits. It can be written in linear algebra as follows, where $U_0, U_1 \ldots, U_m$ are the (n-qubit) gates in the circuit and $|\boldsymbol{b}_1\rangle, |\boldsymbol{b}_2\rangle, \ldots, |\boldsymbol{b}_m\rangle$ are computational basis states.

$$U_m \cdots U_1 U_0 |00\ldots0\rangle = \sum_{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_m \in \{0,1\}^n} U_m \cdot |\boldsymbol{b}_m\rangle\langle\boldsymbol{b}_m| \cdot \ldots \cdot |\boldsymbol{b}_2\rangle\langle\boldsymbol{b}_2| \cdot U_1 \cdot |\boldsymbol{b}_1\rangle\langle\boldsymbol{b}_1| \cdot U_0 \cdot |00\ldots0\rangle$$

The previous example shows that the #SAT encoding does not "merge" paths ending in the same computational basis state, as illustrated below (dashed edges cancel each other out).



Key message: Given a universal quantum circuit, consisting of Toffoli and Hadamard gates, we can construct a Boolean formula whose satisfying assignments represent the circuit's output quantum state.

3.4 Encoding Measurements

Recall in Sect. 2.2.3, measuring all qubits in computational basis obtains the probability of an outcome $|b\rangle$ for $b \in \{0,1\}^n$. In a circuit with gates $U_1, \ldots, U_m$ and an input state $|\varphi_0\rangle$, the output state $|\varphi_m\rangle = U_m \cdots U_1 |\varphi_0\rangle$ can be decomposed into basis states as $|\varphi_m\rangle = \sum_{b\in\{0,1\}^n} \alpha_b |b\rangle$, where the amplitudes can be computed by weighted model counting $\#SAT_W(F_{|\varphi_m\rangle} \wedge F_{|b\rangle}) = \alpha_b$, where $F_{|\varphi_m\rangle} = F_{U_m} \wedge \cdots \wedge F_{U_2} \wedge F_{U_1} \wedge F_{|\varphi_0\rangle}$ follows the encoding in Sect. 3.3 and $F_{|b\rangle}$ is the encoding of basis state $|b\rangle$ as shown in Sect. 3.2. The probability then equals $(\alpha_b)^2$.

Example 12. Reconsider Example 11 with measurements on all qubits of the output state. To get the probability of measuring basis state $|b\rangle = |00\rangle$, the constraint $F_{|b\rangle} = \overline{x}_4 \wedge \overline{y}_4$. should be conjoined to the final state: $F_{|\varphi_4\rangle} \wedge F_{|b\rangle}$. We then have $SAT(F_{|\varphi_4\rangle} \wedge F_{|b\rangle}) = \{\overline{h}'\overline{h}x_1\overline{y}_1\overline{x}_2\overline{y}_2x_3\overline{y}_3\overline{x}_4\overline{y}_4, \overline{h}'\overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\overline{x}_3\overline{y}_3\overline{x}_4\overline{y}_4\}$. The amplitude of $|00\rangle$ is $MC_W(F_{|\varphi_4\rangle} \wedge F_{|b\rangle}) = W(h)W(h') + W(h)W(h') = 1$, thus the resulting probability is $1^2 = 1$. ◇

Exercise: In the above example, compute the probability of measuring with basis state $|01\rangle$.

By adding the measurement constraint, only the "paths" to the basis state we measure would be left. In the sense of satisfying assignments, as shown in the previous example, we will keep the satisfying assignments with variables on the final time step encoding the measured basis state ($\overline{h}'\overline{h}x_1\overline{y}_1\overline{x}_2\overline{y}_2x_3\overline{y}_3\overline{x}_4\overline{y}_4 \equiv \frac{1}{2}|00\rangle, \overline{h}'\overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\overline{x}_3\overline{y}_3\overline{x}_4\overline{y}_4 \equiv \frac{1}{2}|00\rangle$) and discard others ($h'\overline{h}x_1\overline{y}_1x_2y_2x_3\overline{y}_3x_4\overline{y}_4 \equiv -\frac{1}{2}|10\rangle$, $\overline{h}'\overline{h}\overline{x}_1\overline{y}_1\overline{x}_2\overline{y}_2\overline{x}_3\overline{y}_3x_4\overline{y}_4 \equiv \frac{1}{2}|10\rangle$).

4 Wrap-Up

This tutorial detailed how the basic task of *classical simulation* of quantum computing can be done using #SAT, which illustrates the inherent combinatorial nature of quantum computing as well as interference, which amplifies or cancels amplitudes of the output quantum state. We now explain some important open problems in quantum computing and demonstrate, based on related work, how similar methods can solve them.

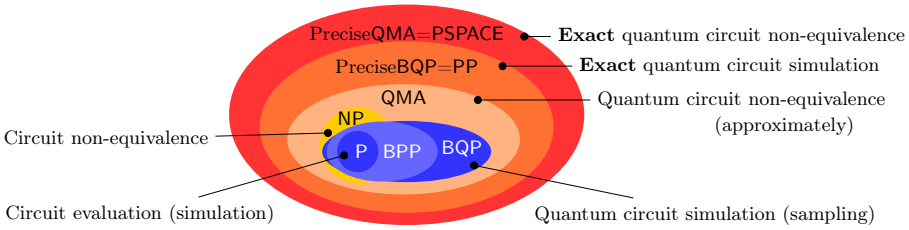
Current quantum computers will have limited numbers of qubits that are noisy [27]. For this reason, investment in the field only took off after the invention of quantum error correction [14], which realizes the ideal quantum circuit model on noisy hardware. However, the added complexity of error correction requires additional resources since the logical error-corrected qubits consist of many physical qubits. So to attain a quantum advantage earlier, we need to optimize quantum algorithms to use few qubits and respect the gate sets and topology (physical connectivity of qubits) of the many different quantum hardware types.

Therefore, we need to optimize quantum circuits, synthesize them in different gate sets and verify their correctness. The latter starts with checking whether an optimized circuit implements the same operations as its original (equivalence checking), but also involves checking quantum Hoare logic [41]. Finally, error-corrected quantum computers will work in tandem with classical computers to monitor the process [13], which requires solving tasks strongly related to the ones discussed above.

Formal methods can indeed be extended to solve the above tasks. For instance, decision diagrams have been used to check the equivalence of quantum circuits [6, 38] and model counting [22] as well. For a more detailed overview of the successes of formal methods in this domain, we refer readers to a historical overview [36] (in which all authors of this tutorial were involved). As an honorable mention, we note that there are also approaches based on diagrammatic reasoning using

the ZX calculus [8,40], which has also been used for circuit optimization [18], on planning [33] and on abstract interpretation [4].

However, the applications of our methods can be viewed in an even broader context. The features that make quantum compilation difficult are shared with the computationally-hard aspects of many problems in quantum physics and quantum chemistry, such as computing the energy of the ground state of a physical system [17] or simulating a many-body system [25]. Indeed, many of these problems in physics are in the quantum analogs of the complexity classes NP and P, i.e., in QMA [5] and BQP [19]; see the figure below for how these quantum complexity classes relate to classical ones (reproduced from [10]). Therefore, any progress in tackling hard problems for quantum circuits can be directly used to solve the very problems that quantum physicists and chemists struggle with on a daily basis (using the default reductions between BQP- and QMA-complete problems). This is important since even if the ideal error-corrected quantum computer never gets built, we are still stuck with the myriad of "quantum-hard" problems that nature poses—the very reason why Feynman proposed building a quantum computer by inverting the problem in the first place (see Sect. 1).



We finish by giving some references for further reading: Lipton and Regan's accessible introduction to quantum computing for computer scientists [20], a seminal textbook by Nielsen and Chuang [24], the lecture notes of Ronald de Wolf [9] and Watrous' detailed monograph on quantum information [39].

Acknowledgements. This publication is part of the project Divide & Quantum (with project number 1389.20.241) of the research program NWA-ORC which is (partly) financed by the Dutch Research Council (NWO). This work was supported by the Dutch National Growth Fund, as part of the Quantum Delta NL program. The third author acknowledges the support received through the NWO Quantum Technology program (project number NGF.1582.22.035).

References

1. Aharonov, D.: A simple proof that Toffoli and Hadamard are quantum universal. arXiv preprint quant-ph/0301040 (2003)
2. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridge (2009). https://doi.org/10.1017/cbo9780511804090

3. Arute, F., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**, 505–510 (2019). https://www.nature.com/articles/s41586-019-1666-5
4. Bichsel, B., Paradis, A., Baader, M., Vechev, M.: Abstraqt: analysis of quantum circuits via abstract stabilizer simulation. Quantum **7**, 1185 (2023). https://doi.org/10.22331/q-2023-11-20-1185, http://dx.doi.org/10.22331/q-2023-11-20-1185
5. Bookatz, A.D.: QMA-complete problems. arXiv:1212.6312 (2012)
6. Burgholzer, L., Wille, R.: Advanced equivalence checking for quantum circuits. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **40**(9), 1810–1824 (2020). https://doi.org/10.1109/tcad.2020.3032630
7. Chen, Y., Chen, Y., Kumar, R., Patro, S., Speelman, F.: Qseth strikes again: finer quantum lower bounds for lattice problem, strong simulation, hitting set problem, and more. arXiv preprint arXiv:2309.16431 (2023)
8. Coecke, B., Duncan, R.: Interacting quantum observables: categorical algebra and diagrammatics. New J. Phys. **13**(4), 043016 (2011). https://doi.org/10.1088/1367-2630/13/4/043016
9. De Wolf, R.: Quantum computing: lecture notes. arXiv preprint arXiv:1907.09415 (2019)
10. Deshpande, A., Gorshkov, A.V., Fefferman, B.: Importance of the spectral gap in estimating ground-state energies. PRX Quant. **3**(4), 040327 (2022)
11. Feynman, R.P.: Simulating physics with computers. Int. J. Theor. Phys. **21**(6), 467–488 (1982). https://doi.org/10.1007/BF02650179
12. Feynman, R., Hibbs, A., Styer, D.: Quantum Mechanics and Path Integrals. Dover Books on Physics, Dover Publications (2010). https://books.google.nl/books?id=JkMuDAAAQBAJ
13. Fowler, A.G., Mariantoni, M., Martinis, J.M., Cleland, A.N.: Surface codes: towards practical large-scale quantum computation. Phys. Rev. A **86**(3), 032324 (2012)
14. Gottesman, D.: Stabilizer codes and quantum error correction. Ph.D. thesis, California Institute of Technology (1997)
15. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219 (1996)
16. Kemeny, J.G., Snell, J.L., et al.: Finite Markov Chains, vol. 26. van Nostrand, Princeton (1969)
17. Kempe, J., Kitaev, A., Regev, O.: The complexity of the local Hamiltonian problem. SIAM J. Comput. **35**(5), 1070–1097 (2006)
18. Kissinger, A., van de Wetering, J.: Reducing the number of non-Clifford gates in quantum circuits. Phys. Rev. A **102**, 022406 (2020). https://doi.org/10.1103/PhysRevA.102.022406, https://link.aps.org/doi/10.1103/PhysRevA.102.022406
19. Kitaev, A.Y., Shen, A., Vyalyi, M.N.: Classical and quantum computation. Am. Math. Soc. (2002)
20. Lipton, R.J., Regan, K.W.: Introduction to Quantum Algorithms via Linear Algebra. MIT Press, Cambridge (2021)
21. Mei, J., Bonsangue, M., Laarman, A.: Simulating quantum circuits by model counting. In: CAV 2024, (accepted for publication). Springer (2024). Pre-print available at arXiv:2403.07197)
22. Mei, J., Coopmans, T., Bonsangue, M., Laarman, A.: Equivalence checking of quantum circuits by model counting. In: IJCAR (accepted for publication) (2024). Pre-print available at arXiv:2403.18813)

23. den Nest, M.V.: Classical simulation of quantum computation, the Gottesman-Knill theorem, and slightly beyond. arXiv:0811.0898 (2008)
24. Nielsen, M.A., Chuang, I.L.: Quantum Information and Quantum Computation, vol. 2, no. 8, p. 23. Cambridge University Press, Cambridge (2000)
25. Orús, R.: A practical introduction to tensor networks: matrix product states and projected entangled pair states. Ann. Phys. **349**, 117–158 (2014). https://doi.org/10.1016/j.aop.2014.06.013, https://www.sciencedirect.com/science/article/pii/S0003491614001596
26. Paturi, R., Pudlák, P., Saks, M.E., Zane, F.: An improved exponential-time algorithm for k-sat. J. ACM (JACM) **52**(3), 337–364 (2005)
27. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
28. Quantum algorithm zoo. https://quantumalgorithmzoo.org/. Accessed 25 Apr 2024
29. Rennela, M., Brand, S., Laarman, A., Dunjko, V.: Hybrid divide-and-conquer approach for tree search algorithms. Quantum **7**, 959 (2023). https://doi.org/10.22331/q-2023-03-23-959
30. Roth, D.: On the hardness of approximate reasoning. Artif. Intell. **82**(1–2), 273–302 (1996)
31. Sang, T., Beame, P., Kautz, H.A.: Performing Bayesian inference by weighted model counting. In: AAAI, vol. 5, pp. 475–481 (2005)
32. Schoning, T.: A probabilistic algorithm for k-sat and constraint satisfaction problems. In: 40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039), pp. 410–414. IEEE (1999)
33. Shaik, I., van de Pol, J.: Optimal layout synthesis for quantum circuits as classical planning. arXiv:2304.12014 (2023)
34. Shi, Y.: Both Toffoli and controlled-NOT need little help to do universal quantum computing. Quant. Inf. Comput. **3**(1), 84–92 (2003)
35. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134. IEEE (1994)
36. Thanos, D., et al.: Automated reasoning in quantum circuit compilation. In: Preproceedings of SPIN2024 (2024). https://spin-web.github.io/SPIN2024/assets/preproceedings/SPIN2024-paper6.pdf
37. Toffoli, T.: Reversible computing. In: de Bakker, J., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 632–644. Springer, Heidelberg (1980). https://doi.org/10.1007/3-540-10003-2_104
38. Viamontes, G.F., Markov, I.L., Hayes, J.P.: Checking equivalence of quantum circuits and states. In: 2007 IEEE/ACM International Conference on Computer-Aided Design, pp. 69–74 (2007). https://doi.org/10.1109/ICCAD.2007.4397246
39. Watrous, J.: The Theory of Quantum Information. Cambridge University Press, Cambridge (2018)
40. van de Wetering, J.: Zx-calculus for the working quantum computer scientist. arXiv:2012.13966 (2020)
41. Ying, M.: Floyd-Hoare logic for quantum programs. ACM Trans. Program. Lang. Syst. (TOPLAS) **33**(6), 1–49 (2012)