Decision Diagrams for Symbolic Verification of Quantum Circuits

1st Xin Hong
Centre for Quantum Software and Information
University of Technology Sydney
Sydney, Australia
xin.hong@student.uts.edu.au

2nd Wei-Jia Huang
Hon Hai Quantum Computing Research Center
Taipei, Taiwan
wei-jia.huang@foxconn.com

3rd Wei-Chen Chien
MediaTek, Inc.
Hsinchu, Taiwan
owen.chien@mediatek.com

4th Yuan Feng
Centre for Quantum Software and Information
University of Technology Sydney
Sydney, Australia
yuan.feng@uts.edu.au

5th Min-Hsiu Hsieh
Hon Hai Quantum Computing Research Center
Taipei, Taiwan
minhsiuh@gmail.com

6th Sanjiang Li
Centre for Quantum Software and Information
University of Technology Sydney
Sydney, Australia
sanjiang.li@uts.edu.au

7th Chia-Shun Yeh
MediaTek, Inc.
Hsinchu, Taiwan
jason.yeh@mediatek.com

8th Mingsheng Ying
Institute of Software
Chinese Academy of Sciences
Beijing, China
yingms@ios.ac.cn

Abstract—With the rapid development of quantum computing, automatic verification of quantum circuits becomes more and more important. While several decision diagrams (DDs) have been introduced in quantum circuit simulation and verification, none of them supports symbolic computation. Algorithmic manipulations of symbolic objects, however, have been identified as crucial, if not indispensable, for several verification tasks. This paper proposes the first decision-diagram approach for operating symbolic objects and verifying quantum circuits with symbolic terms. As a notable example, our symbolic tensor decision diagrams (symbolic TDD) could verify the functionality of the 160-qubit quantum Fourier transform circuit within three minutes. Moreover, as demonstrated on Bernstein-Vazirani algorithm, Grover's algorithm, and the bit-flip error correction code, the symbolic TDD enables efficient verification of quantum circuits with user-supplied oracles and/or classical controls.

Index Terms—Decision Diagram, Symbolic Verification, Quantum Circuits

I. INTRODUCTION

In the past several years, quantum hardware has experienced rapid development. In particular, IBM has recently announced their 127-qubit quantum processor "Eagle", 433-qubit "Osprey" [1] as well as the planned 1121-qubit "Condor" in their aggressive roadmap [2]. As the scale of quantum devices becomes larger and larger, the automatic verification of quantum circuits becomes more and more important.

Decision diagram-based methods are perhaps the most successful verification methods for quantum circuits. Successful examples include, among others, the quantum information decision diagram (QuIDD) [3], the quantum multi-value decision diagram (QMDD) [4], and tensor decision diagram (TDD) [5]. A recent work [6] even used binary decision diagrams (BDD) in the verification of quantum circuits. All these decision diagrams can be used to represent quantum states, simulate quantum circuits, and represent the functionality of quantum circuits. However, many quantum circuits still have no compact decision diagram representations.

Take the well-known quantum Fourier transform (QFT) as an example [7], [8]. QFT can be classically simulated in polynomial time [9] and, for any given input state in the computational basis, the size of its decision diagram representation is linear in the number of qubits. However, it requires exponentially more memory resources to fulfil the complete functionality. Indeed, for an n-qubit QFT, its QuIDD, TDD, or QMDD representation all have $O(2^n)$ nodes [5], [10]. This is because they have very different output amplitudes on 2^n different computational basis states. For example, if $n = 27$, it needs 8 GB of memory to store such a decision diagram.

It is worth noting that the decision diagram representations of different input basis states have similar patterns. Instead of simulating on all 2^n computational basis states, we propose to interpret an arbitrary input basis state on each qubit as a symbol and simulate the circuit on this symbolised basis state. Fortunately, for QFT, the simulation on this symbolised basis state consumes only polynomial resources. This implies that we may verify the functionality of a quantum circuit with a single (symbolised) simulation. The output state for such a symbolised input state can be elegantly represented by a single

TDD-like diagram whose edge weights are symbolic objects instead of complex values.

Perhaps more importantly, symbolic objects appear naturally as classical control signals in many quantum algorithms, such as Grover's algorithm [11] and the Bernstein-Vazirani algorithm (BV) [12]. To completely verify the correctness of these algorithms, all possible classical control signals need to be considered. One way of achieving this is to regard classical control signals as symbolic objects and then check the circuit on the symbolic level.

This work formally presents the symbolic tensor decision diagrams (or simply symTDDs) for symbolically executing and representing a quantum circuit, establishes the canonicity, and then demonstrates the efficiency of symTDDs in the simulation and verification of quantum circuits such as QFT, BV, Grover's algorithm, and the bit-flip error correction code circuit. In this paper, to fully utilise the efficiency of TDD, we regard symbolic objects as tensors and represent them as TDDs.

Extending TDDs with symbolic (tensor) weights makes it possible to leverage the power of symbolic logic in a systematic way. On the one hand, Boolean algebra laws such as $x \cdot x = x$, $x \cdot x' = 0$, and $x + x' = 1$ are used in the normalisation and reduction of symTDDs so that canonicity is guaranteed. On the other hand, the generated symTDDs can be used in extracting distributions of the output states by symbolic computations with these symbolic weights. It is also possible to employ software tools such as SMT solvers like Z3 in the generation of symTDD to check, for example, if some given output state or subspace is reachable.

Related works. Symbolic verification of quantum circuits has also been explored in [13], where a symbolic approach for representing quantum circuits with more general matrix-valued Boolean expressions was proposed. Their approach provides a way to verify quantum circuits with existing techniques and tools developed for the verification of classical logic circuits. In contrast, our work is based on TDDs, which combine the merits of both tensor networks and decision diagrams for representing quantum circuits. In [14], the author also discussed the verification of quantum circuits in a symbolic way. Potentially, it could efficiently represent a large class of quantum circuits, while sacrificing uniqueness (i.e., canonicity). Note that the canonicity of symTDD is guaranteed (cf. Theorem 2).

Classical simulation of QFT circuit as an important task has been considered in works such as [9], [15], [16]. These works simulate either on a subset of all possible input states [16] or in an approximate way [9], [15]. Our scheme, in contrast, can reveal information of the circuit for all computational basis states.

Other verification methods such as equivalence checking and model checking have been considered in [17], [18], [19], [20], etc. However, none of them considers executing a quantum circuit with a symbolic input state and verifying a quantum circuit using a decision diagram with symbolic weights.
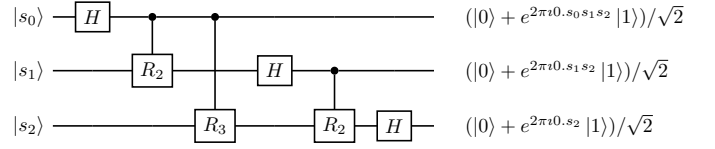

Fig. 1. The circuit for 3-qubit quantum Fourier transform.

II. BACKGROUND

Quantum Computing. The most fundamental concept in quantum computing is qubit, the counterpart of the classical bit. While a classical bit can be either 0 or 1, a qubit can be in a superposition of two basis states. Using the Dirac notation, a qubit state can be represented as $|\psi\rangle = a|0\rangle + b|1\rangle$, where a, b are complex numbers with $|a|^2 + |b|^2 = 1$. It is often represented by the vector $|\psi\rangle = [a, b]^T$. In general, an n-qubit state can be represented by a 2^n-dim vector $|\psi\rangle = [\alpha_0, \cdots, \alpha_{2^n-1}]^T$.

The evolution of quantum states is according to unitary operators, which are also called quantum (logic) gates. The quantum gates used in our paper mainly include the Hadamard gate, the R_k gates, and the Controlled-X gate (aka CNOT or CX gate) defined as follows:

$$H = \tfrac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad R_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{bmatrix} \quad CX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The reader is referred to, e.g., [8] for other standard gates used in this paper. Mathematically, a quantum gate on n qubits can be represented as a $2^n \times 2^n$ unitary matrix.

A quantum circuit is a sequence of quantum gates. For example, Fig. 1 depicts a quantum circuit of the QFT on three qubits. Given a computational basis input $|s_0\rangle |s_1\rangle |s_2\rangle$, the expected output is $\frac{1}{2\sqrt{2}}(|0\rangle + e^{2\pi i 0.s_0 s_1 s_2}|1\rangle)(|0\rangle + e^{2\pi i 0.s_1 s_2}|1\rangle)(|0\rangle + e^{2\pi i 0.s_2}|1\rangle)$. Note that we use i to represent the imaginary unit.

Tensor Decision Diagram (TDD). Tensor network methods play a fundamental role in machine learning and quantum physics. By combining the merits of both tensor networks and decision diagram, TDDs [5] provide a compact and canonical way to represent quantum circuits, which are a special class of tensor networks. TDDs have also been applied in equivalence checking of quantum circuits [21].

A rank n complex-valued tensor is a map $\phi(q_1 \ldots q_n) : \{0,1\}^n \to \mathbb{C}$, where $I = \{q_1, \cdots, q_n\}$ is the set of indices of the tensor and $\phi(a_1, \ldots, a_n)$ its value for the evaluation $q_1 = a_1, \cdots, q_n = a_n$. In the following, we also denote it as $\phi(\mathbf{q})$ for simplicity. The classical Boole-Shannon expansion [22] also extends to tensors: $\phi = q_k' \cdot \phi|_{q_k=0} + q_k \cdot \phi|_{q_k=1}$, where $\phi|_{q_k=a_k}$ is a rank $n-1$ tensor with indices $\{q_1, \cdots, q_{k-1}, q_{k+1}, \cdots, q_n\}$ and $\phi|_{q_k=a_k}(a_1, \cdots, a_{k-1}, a_{k+1}, \cdots, a_n) = \phi(a_1, \cdots, a_n)$, and q_k' represents the complement of q_k. Obviously, an n-qubit quantum state can be seen as a rank n tensor, and an n-qubit quantum gate can be seen as a rank $2n$ tensor. When
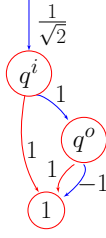
Fig. 2. The canonical TDD of the H gate w.r.t. the index order $q^i \prec q^o$, where q^i and q^o are two internal nodes and we have a unique terminal node.
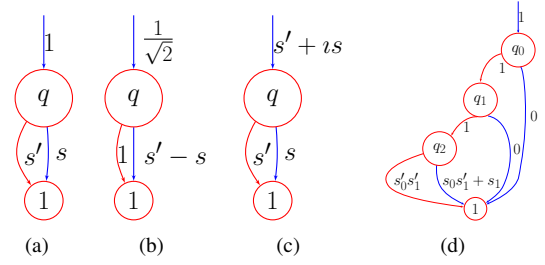

Fig. 3. symTDD representations of (a) a 1-qubit symbolised computational basis state $|s\rangle$; (b) $H|s\rangle$; (c) $R_2|s\rangle$; (d) the output state of the 2-qubit Grover oracle applied on $|001\rangle$, where $|1\rangle$ is the state of the ancilla qubit.

associating to quantum states, q_k normally represents the index of the k-th qubit and we use q_k^i and q_k^o to represent the indices of the input and output of a quantum gate corresponding to the k-th qubit.

There are mainly two tensor operations: addition and contraction. Addition is defined for two tensors ϕ, ψ with the same index set, and $(\phi + \psi)(\mathbf{a}) = \phi(\mathbf{a}) + \psi(\mathbf{a})$ for any assignments $\mathbf{a}$ of the indices. Tensor contraction can be defined for two tensors with different index sets. Let $\gamma(\mathbf{q}, \mathbf{r})$ and $\xi(\mathbf{r}, \mathbf{s})$ be two tensors of which the indices share a common part $\mathbf{r}$. Their contraction is the sum of product over every assignment $\mathbf{c}$ of indices in $\mathbf{r}$. The new tensor $\phi(\mathbf{q}, \mathbf{s})$ with assignments $\mathbf{a}$ and $\mathbf{b}$ hence becomes

$$\phi(\mathbf{a}, \mathbf{b}) = \sum_{\mathbf{c} \in \{0,1\}^{\mathbf{r}}} \gamma(\mathbf{a}, \mathbf{c}) \cdot \xi(\mathbf{c}, \mathbf{b}). \quad (1)$$

In this paper, we will also use the Hadamard product $\odot$ of two tensors, which is defined for two tensors with the same indices and $(\phi \odot \psi)(\mathbf{a}) = \phi(\mathbf{a}) \cdot \psi(\mathbf{a})$. All complex-valued tensors with the same index set form a commutative ring with the two operations $+$ and $\odot$.

A TDD over an index set I is a directed acyclic graph $\mathcal{T} = (V, E, idx, val, w)$, where the node set V consists of non-terminal nodes in V_N and terminal ones in V_T and each non-terminal node v has two child nodes $low(v)$ and $high(v)$ (also called the 0- and 1-successors of v); the associated functions are $idx : V_N \to I$, $val : V_T \to \mathbb{C}$, and $w : E \to \mathbb{C}$ which are respectively assigned node index, terminal node value, and edge weight [5]. For any internal node v, we also call the edge from v to $low(v)$ (resp. $high(v)$) the low-edge or 0-edge (resp. the high-edge or 1-edge). A general TDD may be transformed into a reduced and normalised one, which is canonical w.r.t. a fixed index order, by a series of normalisation and reduction procedures. Usually, all terminal nodes in a TDD are merged to a unique one labelled with 1. Then the tensor value of an evaluation $q_1 = a_1, \cdots, q_n = a_n$ can be obtained by multiplying the weights of the edges along the path assigned by the evaluation.

Example 1. *Fig. 2 gives the canonical TDD of the H gate w.r.t. the index order $q^i \prec q^o$. Each matrix element of H corresponds to the product of the weights along a maximal path from the root to the terminal. For any internal node u, the outgoing red (blue, resp.) edge connects its 0-successor (1-successor, resp.), denoting that the index takes value 0 (1,*

resp.). The red edge of node q^i implies that all elements of the first column of this matrix are $\frac{1}{\sqrt{2}} \cdot 1 = \frac{1}{\sqrt{2}}$. Similarly, the blue edge leading to node q^o implies that the second column of the matrix is $\frac{1}{\sqrt{2}} \cdot 1 \cdot [1, -1]^T = [\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}]^T$.

III. SYMBOLIC TDD

Symbolic TDD (symTDD) is TDD with all weights being symbolic objects instead of complex values. In this paper, these symbolic objects are complex-valued tensors whose indices are symbols from either the input symbolised basis state or classical control signals.

A. Motivating Examples

Similar to the 3-qubit QFT case shown in Fig. 1, the output states (functionality) of a quantum circuit C on all 2^n computational basis states can be (symbolically) computed by executing C on a single symbolised basis state. Furthermore, symTDD can also be used to represent the oracle of many quantum algorithms, including Grover's algorithm and Bernstein-Vazirani algorithm.

Fig. 3 illustrates several symTDDs, which represent (a) the symbolic input basis state $|s\rangle = s'|0\rangle + s|1\rangle$, (b) $H|s\rangle$, the resultant state of applying a Hadamard gate on $|s\rangle$, (c) $R_2|s\rangle$, the resultant state of applying a rotation gate R_2 on $|s\rangle$, and (d) the resultant state of applying a 2-qubit Grover oracle on $|001\rangle$, where $|1\rangle$ is the state of the ancilla qubit. Note that $H|s\rangle = \frac{1}{\sqrt{2}}(|0\rangle + (s' - s)|1\rangle)$. If we take $s = 0$, then Fig. 3(a,b) exactly give the TDD representations of the states $|0\rangle$ and $|+\rangle = H|0\rangle$. A similar observation applies to the case $s = 1$. This implies that the information of the circuit for all input computational basis states is captured by a single simulation on the symbolised basis state. Note that all weights here are tensors: s and s' (the complement of s) represent the tensors with only one index s, and $\phi(s) = s$ and $\xi(s) = 1 - s$; $s'_0 s'_1$ and $s_0 s'_1 + s_1$ represent the rank 2 tensors with indices s_0, s_1, where $\phi(s_0, s_1) = (1 - s_0) \cdot (1 - s_1)$ and $\xi(s_0, s_1) = s_0 \cdot (1 - s_1) + s_1$.

B. Symbolic TDD and Operations

SymTDD is an extension of TDD [5] in that its weights are, instead of complex values, complex-valued tensors over a set of indices (i.e., Boolean symbols) $S = \{s_0, \cdots, s_{m-1}\}$. Since a TDD represents a tensor, a symTDD represents a tensor-valued tensor ϕ (a map from $\{0, 1\}^n$ to tensors), or more

precisely, a map in $\{0,1\}^n \rightarrow \{0,1\}^m \rightarrow \mathbb{C}$. For clarity, let $I = \{q_0, \ldots, q_{n-1}\}$ be the set of indices other than those in S. For convenience, we often address indices in I and S as, respectively, quantum and classical indices. In addition, we call ϕ an (I,S)-tensor with rank n or simply an (m,n)-tensor. This representation gives us more flexibility for representing and analysing a quantum circuit. For example, let $\mathbf{s}$ be the indices that appear in both an n-qubit input state $|\psi\rangle$ and an n-qubit quantum circuit U (note that a tensor with index set S can always be regarded as a tensor with index set S' with $S' \supseteq S$). That is, $|\psi\rangle = [f_0(\mathbf{s}), \cdots, f_{2^n-1}(\mathbf{s})]^T$ and $U = [u_{ij}(\mathbf{s}) : i,j = 0, \ldots, 2^n-1]$ where both f_j and u_{ij} are tensors in $\{0,1\}^m \rightarrow \mathbb{C}$. Note that $U|\psi\rangle = [g_0(\mathbf{s}), \cdots, g_{2^n-1}(\mathbf{s})]^T$ where for each i,

$$g_i(\mathbf{s}) = \sum_{j=0}^{2^n-1} u_{ij}(\mathbf{s}) \odot f_j(\mathbf{s}).$$

Then g_i is again a tensor in $\{0,1\}^m \rightarrow \mathbb{C}$, and finally, $U|\psi\rangle$ is a symbolic tensor in $\{0,1\}^n \rightarrow \{0,1\}^m \rightarrow \mathbb{C}$.

The overall representation and calculation are similar to that in TDD, except that the addition and multiplication of complex numbers used in TDD (Alg.s 1~3 in [5]) should be replaced by the addition and multiplication of complex-valued tensors.

C. Normalisation

Normalisation is required to make symTDD representations canonical. The main idea is to extract a (greatest) common part from the weights on the two outgoing edges of every internal node and multiply it to their incoming edges, such that no more common part can be further extracted from the two remaining weights.

Assume that all tensors are over the same index set S. Let f, g be the weights on the outgoing edges of internal node v. We define the extracted tensor h and the remaining parts f^* and g^* in a term-by-term manner. For any assignment $\mathbf{a}$ of S, we set $h(\mathbf{a})$ as $f(\mathbf{a})$ if $f(\mathbf{a}) \neq 0$, and $g(\mathbf{a})$ otherwise. Furthermore,

$$f^*(\mathbf{a}) = \begin{cases} 1 & \text{if } f(\mathbf{a}) \neq 0 \\ 0 & \text{otherwise,} \end{cases} \quad g^*(\mathbf{a}) = \begin{cases} g(\mathbf{a})/f(\mathbf{a}) & \text{if } f(\mathbf{a}) \neq 0 \\ 1 & \text{if } f(\mathbf{a}) = 0, \ g(\mathbf{a}) \neq 0 \\ 0 & \text{if } f(\mathbf{a}) = g(\mathbf{a}) = 0 \end{cases}$$

We call this the local normalisation procedure and denote the result as $loc\_norm(f,g) = (h, f^*, g^*)$. For example, let $f = [2\imath, 0, 1+\imath, 0]$ and $g = [0, 1, 1-\imath, 0]$ be two tensors over S in vector representations. They will be normalised to (h, f^*, g^*) with $h = [2\imath, 1, 1+\imath, 0]$, $f^* = [1, 0, 1, 0]$, and $g^* = [0, 1, \frac{1-\imath}{1+\imath}, 0]$.

Let $\text{supp}(f) = \{\mathbf{a} \in \{0,1\}^n : f(\mathbf{a}) \neq 0\}$. For a local normalisation $loc\_norm(f,g) = (h, f^*, g^*)$, we have $\text{supp}(h) = \text{supp}(f) \cup \text{supp}(g)$, $\text{supp}(f^*) = \text{supp}(f)$, and $\text{supp}(g^*) = \text{supp}(g)$. If $\text{supp}(f) = \{0,1\}^n$, then $h = f$, $f^* = 1$ and $g^* = \left[\frac{g_0}{f_0}, \cdots, \frac{g_{2^n-1}}{f_{2^n-1}}\right]$. We also have the following useful lemma.

Lemma 1. *If $loc\_norm(f,g) = (h, f^*, g^*)$ and $\text{supp}(h) \subseteq \text{supp}(h^*)$, then $loc\_norm(h^* \odot f, h^* \odot g) = (h^* \odot h, f^*, g^*)$; if*

$loc\_norm(h \odot f, h \odot g) = (h, f, g)$ *and $\text{supp}(h) = \text{supp}(h^*)$, then $loc\_norm(h^* \odot f, h^* \odot g) = (h^*, f, g)$.*

By using this local normalisation scheme, we can *fully* normalise a symTDD $\mathcal{T}$. A symTDD $\mathcal{T}$ is called **fully normalised** if for each internal node v with two outgoing edges weights f_v and g_v and any path leading to it with accumulated weights h_v we have $loc\_norm(h_v \odot f_v, h_v \odot g_v) = (h_v, f_v, g_v)$. The fully normalised form reveals the invariance under local normalisation. Typically, a symTDD can be fully normalised by pushing all weights down to the bottom and then performing the local normalisation procedure on each internal node from bottom to top. During this process, however, we may need to split a lot of nodes causing additional time and memory overhead. But the following theorem shows that the normalisation procedure can be done more efficiently for the cases considered in this paper.

Theorem 1 (normalisation). *Let $\mathcal{T}$ be a symTDD with quantum and classical index sets I and S. For each internal node v of $\mathcal{T}$ and any incoming edge e of v, write h_v, f_v, g_v for, respectively, the weights of e and the two outgoing edges of v. Then $\mathcal{T}$ is fully normalised if and only if $loc\_norm(h_v \odot f_v, h_v \odot g_v) = (h_v, f_v, g_v)$ for each internal node v and any incoming edge e of v.*

Proof. If $\mathcal{T}$ has no internal nodes, this result is clearly true.

In the following, let $\mathcal{T}$ be a symTDD and v an internal node of $\mathcal{T}$. For any incoming edge e of v, suppose h_v is weight on $e = (u,v)$ and h_v^* the corresponding weight accumulated by multiplying all the weights leading to v through e. In addition, write f_v and g_v for the weights on the two outgoing edges of v.

If v is the root node of $\mathcal{T}$, we have $h_v = h_v^*$. Thus, $loc\_norm(h_v \odot f_v, h_v \odot g_v) = (h_v, f_v, g_v)$ iff $loc\_norm(h_v^* \odot f_v, h_v^* \odot g_v) = (h_v^*, f_v, g_v)$. Otherwise, suppose $h_v^* = h_0 \odot \cdots \odot h_k \odot h_v$. Note that here u is the parent node of v s.t. $e = (u,v)$. Then $h_u^* = h_0 \odot \cdots \odot h_k$.

Then, suppose $\mathcal{T}$ is fully normalised. We have $loc\_norm(h_v^* \odot f_v, h_v^* \odot g_v) = (h_v^*, f_v, g_v)$. Then $\text{supp}(h_v^*) = \text{supp}(f_v) \cup \text{supp}(g_v)$. By applying this property for the node u, we have $\text{supp}(h_u^*) \supseteq \text{supp}(h_v)$. As a result, we must have $\text{supp}(h_v) = \text{supp}(h_v^*)$. According to Lemma 1, we have $loc\_norm(h_v \odot f_v, h_v \odot g_v) = (h_v, f_v, g_v)$.

For the sufficiency part, we have $loc\_norm(h_v \odot f_v, h_v \odot g_v) = (h_v, f_v, g_v)$. According to the local normalisation scheme, $\text{supp}(f_v) = \text{supp}(h_v \odot f_v) \subseteq \text{supp}(h_v)$. Also, we have $\text{supp}(h_v) \subseteq \text{supp}(h_k) \subseteq \cdots \subseteq \text{supp}(h_0)$. Thus, $\text{supp}(h_v^*) = \text{supp}(h_0 \odot \cdots \odot h_k \odot h_v) = \text{supp}(h_v)$. Then according to lemma 1, we have $loc\_norm(h_v^* \odot f_v, h_v^* \odot g_v) = (h_v^*, f_v, g_v)$. $\qquad\square$

Now, we have a more efficient procedure for normalising a symTDD $\mathcal{T}$. That is, we first do a local normalisation for every internal node of $\mathcal{T}$ from top to bottom and then do another local normalisation for every internal node of $\mathcal{T}$ from bottom to top. The top-down procedure ensures that, for every internal
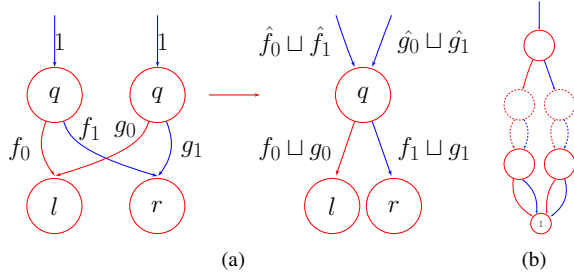
Fig. 4. Reduction rule RR3.

node v, the support of the weight on any incoming edge of v contains the supports of the weights of v's outgoing edges; and the bottom-up procedure ensures that $loc\_norm(h_v \odot f_v, h_v \odot g_v) = (h_v, f_v, g_v)$ for every internal node v. As we don't need to push every weight down to the bottom, the probability of node splitting and hence the space and time consumption is significantly reduced.

D. Reduction

Reduction is a process to merge nodes that represent the same tensors so that the decision diagram can be as compact as possible. The following two reduction rules are commonly used for various kinds of decision diagrams, including BDDs and TDDs.

- RR1: Delete a node v if its 0- and 1-successors are both w and its low- and high-edges have the same weight f. Meanwhile, redirect the incoming edge of v to w.
- RR2: Merge two nodes if they have the same index, the same 0- and 1-successors, and the same weights on the corresponding edges.

Note in RR1 we have $h \odot f = h$ if the local normalisation has been conducted on v, where h is the weight on the incoming edge of v.

While the above two reduction rules are sufficient, the reduction rule RR3 as specified in Fig. 4 (a) is sometimes more powerful. To apply RR2, we require the weights on the corresponding low- and high edges to be identical, which are often too restricted for tensors. From the left side of Fig. 4 (a), we can see that RR3 is applicable when two nodes (i) have the same index and the same 0- and 1-successors; (ii) $g_i(\mathbf{a}) = f_i(\mathbf{a})$ for all $\mathbf{a} \in \operatorname{supp}(f_i) \cap \operatorname{supp}(g_i)$, where f_i and g_i are the weights on the outgoing edge to the i-successor for $i \in \{0, 1\}$; and (iii) $\operatorname{supp}(f_0) \cap \operatorname{supp}(g_1) = \operatorname{supp}(f_1) \cap \operatorname{supp}(g_0)$. In the right side of Fig. 4 (a), $\hat{f}$ denotes the weight tensor which takes value 1 at each element in $\operatorname{supp}(f)$ and 0 otherwise; $f \sqcup g$ denotes the tensor which takes the value as $f(\mathbf{a})$ at each $\mathbf{a} \in \operatorname{supp}(f)$ and $g(\mathbf{a})$ otherwise. Here, we assume that f and g have the same value on their common support when using this operation. The basic idea here is that $(\hat{f_0} \sqcup \hat{f_1}) \odot (f_0 \sqcup g_0) = f_0$, $(\hat{f_0} \sqcup \hat{f_1}) \odot (f_1 \sqcup g_1) = f_1$, $(\hat{g_0} \sqcup \hat{g_1}) \odot (f_0 \sqcup g_0) = g_0$ and $(\hat{g_0} \sqcup \hat{g_1}) \odot (f_1 \sqcup g_1) = g_1$. In RR3, the weights $\hat{f_0} \sqcup \hat{f_1}$ and $\hat{g_0} \sqcup \hat{g_1}$ are used as filters indicating when the corresponding edges will lead to a node which represents a tensor with non-zero values. Then $f_0 \sqcup g_0$ and $f_1 \sqcup g_1$ specify the corresponding

values. RR3 is more powerful than RR2 and is essential for making quantum circuits, such as the Toffoli circuit, in tower forms. Since RR3 may change the weights on the edges and destroy the normalisation of a symTDD, we only use it at specific times as indicated in Fig. 4 (b). That is, we will only merge two nodes using RR3 when all the paths leading to these two nodes come from the same node. The nodes with dotted lines mean that there can be zero or many such nodes.

E. Canonicity

By using the normalisation and reduction rules, we are able to construct a canonical symTDD representation for every tensor-valued tensor. In the following, we only consider the use of RR1 and RR2 for simplicity. A fully normalised symTDD is reduced when no reduction rule (RR1 or RR2) can be further applied. The following theorem shows that there is, up to isomorphism and a given index order, a unique reduced symTDD for each tensor. Two symTDDs $\mathcal{F}$ and $\mathcal{G}$ w.r.t. the same index order are *isomorphic* if there is a bijection σ between the node sets of $\mathcal{F}$ and $\mathcal{G}$ such that, for each node v, $v' = \sigma(v)$ and v have the same index, the same weights on their incoming and outgoing edges, and $\sigma(low(v)) = low(v')$, $\sigma(high(v)) = high(v')$.

Theorem 2 (canonicity). *Let $S = \{s_0, \ldots, s_{m-1}\}$ be a set of classical indices and $I = \{q_0, \ldots, q_{n-1}\}$ a set of quantum indices. Suppose ϕ is an (I, S)-tensor. Given any index order $\prec$ of I, ϕ has a unique reduced symTDD representation w.r.t. $\prec$ up to isomorphism.*

Proof. We prove this by using induction on the rank of the tensor. Let $\phi = s \in [2^m \to \mathbb{C}]$ be a rank 0 tensor. Any reduced symTDD of ϕ has only one node labelled with 1 and has weight s. Suppose the statement holds for any rank k tensor. Let ϕ be a rank $k+1$ tensor and $\prec$ any index order on I. Suppose $\mathcal{F}, \mathcal{G}$ are two reduced symTDDs that represent ϕ w.r.t. $\prec$. We show they are isomorphic. Let q be the first index under $\prec$. Without loss of generality, we assume $\phi|_{q=0} \neq \phi|_{q=1}$. That is, q is essential in ϕ.

Write $r_{\mathcal{F}}$ and $r_{\mathcal{G}}$ for the root nodes of $\mathcal{F}$ and $\mathcal{G}$. Then q is the index of both $r_{\mathcal{F}}$ and $r_{\mathcal{G}}$. Let h_1, f_1, g_1 and h_2, f_2, g_2 be the weights of the incoming and two outgoing edges of, respectively, $r_{\mathcal{F}}$ and $r_{\mathcal{G}}$. We write $\mathcal{F}_0$ and $\mathcal{F}_1$ for the sub-DDs of $\mathcal{F}$ with root nodes the 0- and 1-successors of $r_{\mathcal{F}}$ and weights $h_1 \odot f_1$ and $h_1 \odot g_1$, respectively. The two sub-DDs $\mathcal{G}_0$ and $\mathcal{G}_1$ of $\mathcal{G}$ are defined in a similar way. We assert that $\mathcal{F}_i$ and $\mathcal{G}_i$ are all reduced. Apparently, no reduction rule can be applied as they are sub-DDs of reduced symTDDs. Thus we need only show that they are normalised. By Theorem 1, we need only show their root nodes are locally normalised. Take $\mathcal{F}_0$ as an example. Let $v = low(r_{\mathcal{F}})$ (the 0-successor of $r_{\mathcal{F}}$) and f_v, g_v be the weights on the two outgoing edges of v in $\mathcal{F}$. Note that f_1 is the weight on the incoming edge of v. We need only show $loc\_norm(h_1 \odot f_1 \odot f_v, h_1 \odot f_1 \odot g_v) = (h_1 \odot f_1, f_v, g_v)$. Since $\mathcal{F}$ is normalised, we have $\operatorname{supp}(f_1) \subseteq \operatorname{supp}(h_1)$ and $loc\_norm(f_1 \odot f_v, f_1 \odot g_v) = (f_1, f_v, g_v)$. By Lemma 1, we have $loc\_norm(h_1 \odot f_1 \odot f_v, h_1 \odot f_1 \odot g_v) = (h_1 \odot f_1, f_v, g_v)$.

Thus $\mathcal{F}_i$ and $\mathcal{G}_i$ are all normalised and reduced. It is easy to see that $\mathcal{F}_0$ and $\mathcal{G}_0$ represent the same tensor $\phi|_{q=0}$ and hence are isomorphic by induction hypothesis. Analogously, $\mathcal{F}_1$ and $\mathcal{G}_1$ represent the same tensor $\phi|_{q=1}$ and are also isomorphic.

Let σ_i be the isomorphism between $\mathcal{F}_i$ and $\mathcal{G}_i$ for $i = 1, 2$. Because $\mathcal{F}$ is reduced, for any node v in both $\mathcal{F}_1$ and $\mathcal{F}_2$, we have $\sigma_1(v) = \sigma_2(v)$. Define σ as the extension of σ_1 and σ_2 by further mapping $r_{\mathcal{F}}$ to $r_{\mathcal{G}}$. We show $\sigma : \mathcal{F} \to \mathcal{G}$ is an isomorphism. To this end, we need only show $h_1 = h_2$, $f_1 = f_2$, and $g_1 = g_2$. Since $\mathcal{F}_i \cong \mathcal{G}_i$, we have $h_1 \odot f_1 = h_2 \odot f_2$ and $h_1 \odot g_1 = h_2 \odot g_2$. Because $\mathcal{F}, \mathcal{G}$ are normalised, we have $(h_1, f_1, g_1) = loc\_norm(h_1 \odot f_1, h_1 \odot g_1) = loc\_norm(h_2 \odot f_2, h_2 \odot g_2) = (h_2, f_2, g_2)$. This shows $\mathcal{F} \cong \mathcal{G}$. $\square$

IV. CASE STUDY

In this section, we show by examples how symTDD can be used in the verification of quantum circuits.

Quantum Fourier Transform. Suppose the input state is a computational basis state. The output state of QFT can be represented as a QMDD or TDD with $\mathcal{O}(n)$ nodes. The functionality of the circuit, however, requires $\mathcal{O}(2^n)$ nodes to represent in either QMDD or TDD.
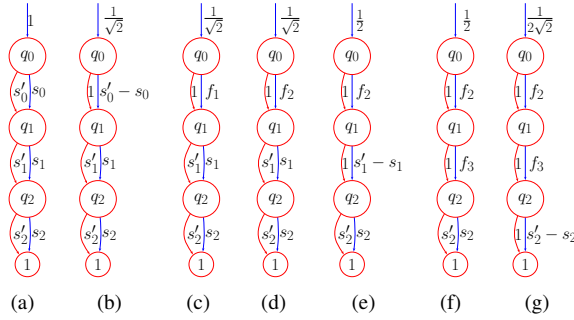

Fig. 5. symTDD for simulation of 3-qubit QFT, where (a)-(g) represent the symTDD of the input state and the states obtained after applying the gates shown in Fig. 1 in sequence, and $f_1 = (s'_0 - s_0)(s'_1 - \imath s_1)$, $f_2 = (s'_0 - s_0)(s'_1 + \imath s_1)(s'_2 + \frac{1+\imath}{\sqrt{2}} s_2)$, $f_3 = (s'_1 - s_1)(s'_2 + \imath s_2)$.

To verify the functionality of the QFT circuit, we first execute the circuit on the symbolised input state $|s_0\rangle \cdots |s_{n-1}\rangle$. Fig. 5 illustrates the detailed process of executing the 3-qubit QFT on the symbolised input state $|s_0\rangle |s_1\rangle |s_2\rangle$. All symTDDs generated have a tower form with only four nodes. In this process, applying the Hadamard gate on q_0 changes the two weights s'_0 and s_0 to 1 and $s'_0 - s_0$ respectively. Then applying the Controlled-R_2 gate on q_0, q_1 further changes the weights to 1 and $(s'_0 - s_0)(s'_1 - \imath s_1)$ respectively. This is because, no matter whether R_2 is applied on the target qubit q_1, the corresponding node is not changed (cf. Fig. 3(a,c)) and only a weight is multiplied to the weight on the high-edge of the control qubit q_0. Similarly, applying the Controlled-R_3 gate on q_0, q_2 changes the weights to 1 and $(s'_0 - s_0)(s'_1 + \imath s_1)(s'_2 + (\frac{1}{\sqrt{2}} + \frac{1}{\sqrt{2}}\imath)s_2)$ respectively. The similar pattern will repeat on the last two qubits. The symTDD shown in Fig. 5(g) contains

the information of all possible output states of the circuit. Since $s'_2 - s_2 = e^{2\pi\imath 0.s_2}$, $f_3 = (s'_1 - s_1)(s'_2 + \imath s_2) = e^{2\pi\imath 0.s_1 s_2}$, and $f_2 = (s'_0 - s_0)(s'_1 + \imath s_1)(s'_2 + \frac{1+\imath}{\sqrt{2}} s_2) = e^{2\pi\imath 0.s_0 s_1 s_2}$, this verified the functionality of the QFT circuit.
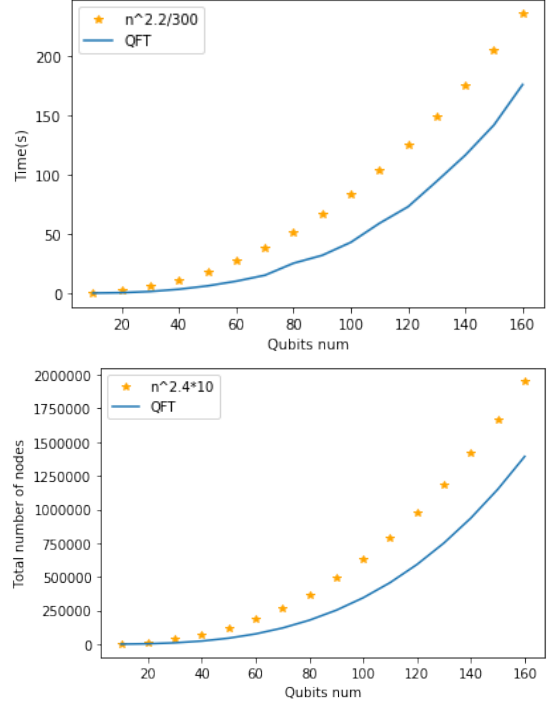

Fig. 6. The time and space consumption of symTDD execution of 10-160 qubits QFT circuits. In this figure, we use ˆ to represent the exponent and nˆ 2.2/300, nˆ 2.4*10 represent $n^{2.2}/300$ and $10n^{2.4}$ respectively.

We symbolically execute the QFT circuit from 10 to 160 qubits and depict in Fig. 6 the time and space consumption curve. The space consumption was measured as the total number of nodes used for representing both the symTDD and all tensor weights. As a comparison, we also give the curve for $n^{2.2}/300$ and $10n^{2.4}$ in the time and space figures. It can be seen that both the time and space consumption grow polynomially in the number of qubits. In fact, it only requires about ten thousand nodes to represent a 30-qubit QFT circuit in symTDD other than using around two billion nodes in a single TDD. Remarkably, QFT circuit with 160 qubits can be executed within three minutes on our laptop with 8GB memory.

Bernstein-Vazirani Algorithm. Suppose O is an oracle such that $O |\mathbf{x}\rangle |\mathbf{y}\rangle = |\mathbf{x}\rangle |\mathbf{y} \oplus f(\mathbf{x})\rangle$, where $f(\mathbf{x}) = \mathbf{s} \cdot \mathbf{x} = s_0 x_0 \oplus \cdots \oplus s_{n-1} x_{n-1}$ and $\mathbf{s}$ is a hidden Boolean string. The Bernstein-Vazirani algorithm (BV) [12] can find s with a single call of the oracle. The circuit for the 3-bit BV is shown in Fig. 7, where the oracle is implemented by a set of Controlled-X^{s_i} gates.

While for every $\mathbf{s}$ this circuit can be verified with a single run of TDD simulation, it needs exponential time in total to verify this for all $\mathbf{s}$. Executing the circuit symbolically, we obtain the symTDD of $|s_0\rangle \cdots |s_{n-1}\rangle |1\rangle$ as the output
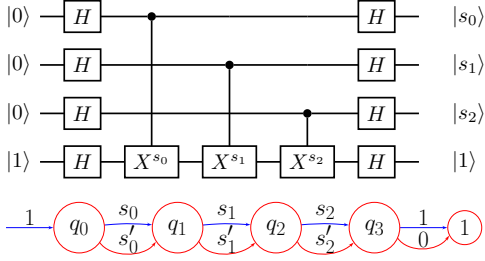
Fig. 7. The circuit (top) and symTDD (bottom) for verification of the Bernstein-Vazirani algorithm.

state (cf. Fig. 7 (bottom)), indicating that the hidden string **s** has been successfully computed. It is also observed in our experiments that all symTDDs generated during this process are in a tower form.
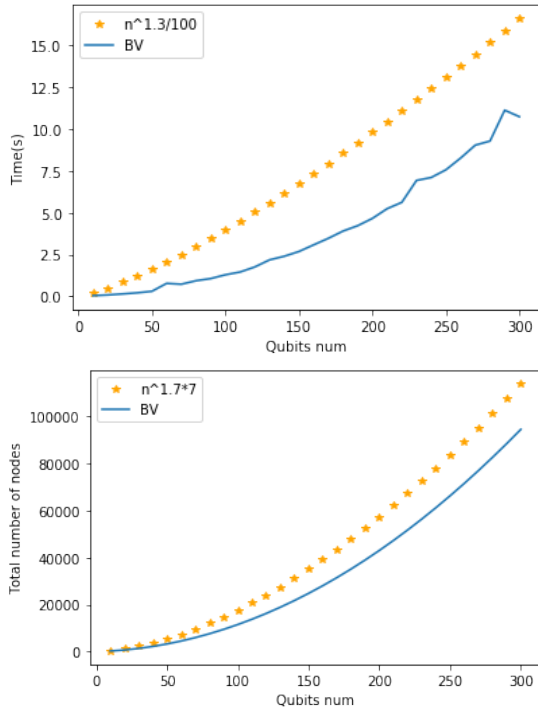

Fig. 8. The time and space consumption of using symTDDs to verify the Bernstein-Vazirani algorithm. In the figure, we use ˆ to represent the exponent and n^ 1.3/100, n^ 1.7*7 represent $n^{1.3}/100$ and $7n^{1.7}$ respectively.

Fig. 8 gives the time and space consumption needed for verifying BV using the symTDD. As a comparison, we also provide the data for $n^{1.3}/100$ and $7n^{1.7}$. It can be seen that both the time and space consumption grow polynomially in the number of qubits. In fact, 300-qubit BV can be verified within 10 seconds.

Grover's Algorithm. Similar to BV, Grover's algorithm [8] can also be verified by symTDD. For the 3-qubit Grover's algorithm, an oracle O as specified by $O |\mathbf{x}\rangle |y\rangle = |\mathbf{x}\rangle |y \oplus f(\mathbf{x})\rangle$ with $f(\mathbf{x}) = 1$ iff $x = s$, where s is the solution that the algorithm aims to find. Similar to BV, it also needs an

exponential number of runs to verify for all possible **s** by using TDD. In contrast, a single run of the symTDD suffices.
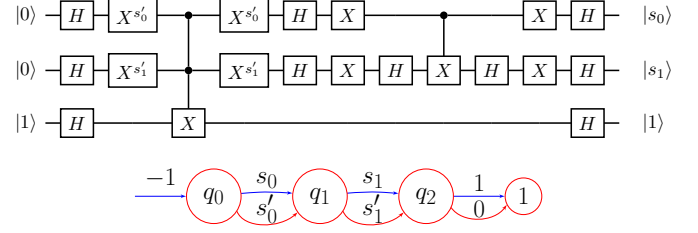

Fig. 9. The circuit (top) and symTDD (bottom) for verification of the 3-qubit Grover's algorithm.

The output of symbolic execution of the 3-qubit Grover circuit is shown in Fig. 9 (bottom), which exactly represents the state $|s_0\rangle |s_1\rangle |1\rangle$ up to a global phase -1.

For Grover's algorithm with more than 4 qubits, the output state is not exactly $|s_0\rangle \cdots |s_{n-1}\rangle |1\rangle$. Nevertheless, it can be analysed using symTDD as well. We symbolically execute the algorithm by applying the Grover iteration $\lfloor \sqrt{2^n} \pi/4 \rfloor$ times, and then compute the probability of successfully finding the solution s. Note that the desirable output $|s_0\rangle \cdots |s_{n-1}\rangle |1\rangle$ can also be represented as a symTDD ϕ_{suc}. The calculation of the success probability boils down to the contraction of ϕ_{suc} with the resulting symTDD of executing Grover's algorithm.

Although the whole complexity is not polynomial, it shows an exponential advantage over TDDs. For example, it takes 29 seconds and 53,892 nodes to simulate the 8-qubit Grover's algorithm (with 8 iterations) using symTDD and find the success probability to be 0.9956 no matter what the solution is. In comparison, it takes 23 seconds and 1753 nodes to obtain the same answer for a given solution, which means that the expenses will sum up to $23 \cdot 2^7 = 2,944$ seconds and $1,753 \cdot 2^7 = 224,384$ nodes when the task is to analyse all possible **s**. When we consider the 9-qubit Grover's algorithm with 12 iterations, the corresponding data will be $153 \ vs \ 34,371$ seconds and $164,377 \ vs \ 829,184$ nodes respectively, where the success probability is 0.9999.

Toffoli Circuits. In addition, symTDD can be used to analyse Toffoli circuits [23], i.e., circuits consisting of only $C^k X$ ($k \in \mathbb{N}$) gates. Every Toffoli circuit has a symTDD representation in tower form (cf. the bottom of Fig. 10). This is because, for any $C^k X$ which splits the symTDD, the two branches have disjoint supports, and thus can be merged by using RR3 introduced in Sec. III-D. Moreover, the two weights on the outgoing edges of every internal node have forms f' and f respectively, where f is a tensor that states exactly the relations between the output state of this qubit and the input signals while f' represents its complement. By contrast, using TDD will interleave the input and output indices, making the analysis difficult.

As an example of Toffoli circuits, let us consider the bit-flip code circuit [8] shown in Fig. 10 (top), which can correct the input quantum state if there is at most one bit-flip error occurred. Let $q_0, \cdots, q_5$ be the 6 qubits from top to bottom. Symbolically executing this circuit, the corresponding
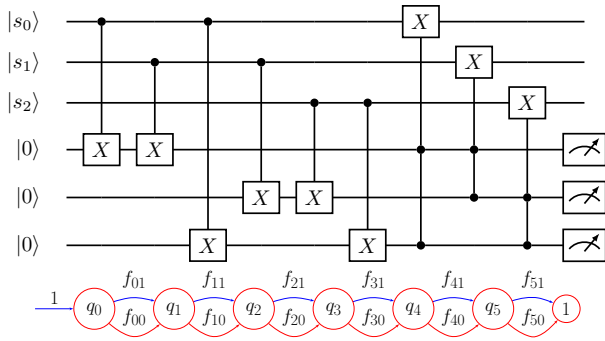
Fig. 10. The circuit (top) and the symTDD (bottom) for verification of the bit-flip code circuit. Principle of deferred measurement [8] is used.

symTDD is shown in Fig. 10 (bottom) where $f_{00} = f_{10} = f_{20} = s_0's_1's_2 + s_0's_1s_2' + s_1's_2'$ and $f_{01} = f_{11} = f_{21} = f_{00}'$. This means that the output of q_0, q_1, q_2 will be $|000\rangle$ when at least two of s_0, s_1, s_2 are 0 and the output of q_0, q_1, q_2 will be $|111\rangle$ when at least two of s_0, s_1, s_2 are 1. Furthermore, the weights $f_{30} = f_{31}' = s_0s_1 + s_0's_1'$, $f_{40} = f_{41}' = s_1s_2 + s_1's_2'$ and $f_{50} = f_{51}' = s_0s_2 + s_0's_2'$, meaning that measuring q_3, q_4, q_5 at the end of the circuit will obtain result 0 iff $s_0 = s_1$, $s_1 = s_2$ and $s_0 = s_2$, respectively. Note that to obtain the same information, we have to traverse all 56 paths in a TDD of 34 nodes, if TDD is employed for the verification task.

V. Conclusion

This paper proposed the first decision-diagram approach for operating symbolic objects. The proposed symTDD is an extension of the tensor decision diagram and makes it possible to leverage the power of symbolic logic and tensor networks in a systematic way. Our experiments on QFT, BV, Grover's Algorithm have partially demonstrated the utility of symTDD. In particular, it provides an efficient approach for extracting information of the output states of quantum circuits. It can also be used to verify the correctness of quantum algorithms with classical control signals. Future work will employ SMT solvers like Z3 in extracting more useful output state information of quantum circuits.

Acknowledgements

References

[1] J. Chow, O. Dial, and J. Gambetta, "IBM quantum breaks the 100-qubit processor barrier," *IBM Research Blog*, 2021.

[2] Jay Gambetta, "IBM's roadmap for scaling quantum technology," https://research.ibm.com/blog/ibm-quantum-roadmap, 2020, accessed: 2022-04-27.

[3] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Improving gate-level simulation of quantum circuits," *Quantum Information Processing*, vol. 2, no. 5, pp. 347–380, 2003.

[4] P. Niemann, R. Wille, D. M. Miller, M. A. Thornton, and R. Drechsler, "QMDDs: Efficient quantum function representation and manipulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 1, pp. 86–99, 2015.

[5] X. Hong, X. Zhou, S. Li, Y. Feng, and M. Ying, "A tensor network based decision diagram for representation of quantum circuits," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 6, pp. 1–30, 2022.

[6] Y.-H. Tsai, J.-H. R. Jiang, and C.-S. Jhang, "Bit-slicing the Hilbert space: scaling up accurate quantum circuit simulation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 439–444.

[7] D. Coppersmith, "An approximate fourier transform useful in quantum factoring," *arXiv preprint quant-ph/0201067*, 2002.

[8] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016.

[9] D. Aharonov, Z. Landau, and J. A. Makowsky, "The quantum FFT can be classically simulated," *arXiv: Quantum Physics*, 2008.

[10] A. Zulehner and R. Wille, "Advanced simulation of quantum computations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 848–859, 2018.

[11] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

[12] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[13] M. Ying and Z. Ji, "Symbolic verification of quantum circuits," *arXiv preprint arXiv:2010.03032*, 2020.

[14] M. Amy, "Towards large-scale functional verification of universal quantum circuits," *arXiv preprint arXiv:1805.06908*, 2018.

[15] N. Yoran and A. J. Short, "Efficient classical simulation of the approximate quantum fourier transform," *Physical Review A*, 2007.

[16] D. E. Browne, "Efficient classical simulation of the quantum Fourier transform," *New Journal of Physics*, vol. 9, 2007.

[17] L. Burgholzer and R. Wille, "QCEC: A jkq tool for quantum circuit equivalence checking," *Software Impacts*, vol. 7, p. 100051, 2021.

[18] ——, "Improved DD-based equivalence checking of quantum circuits," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 127–132.

[19] T. Peham, L. Burgholzer, and R. Wille, "Equivalence checking of quantum circuits with the ZX-calculus," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2022.

[20] M. Ying and Y. Feng, *Model Checking Quantum Systems: Principles and Algorithms*. Cambridge University Press, 2021.

[21] X. Hong, M. Ying, Y. Feng, X. Zhou, and S. Li, "Approximate equivalence checking of noisy quantum circuits," in *2021 58th ACM/IEEE DAC*. IEEE, 2021, pp. 1–6.

[22] C. E. Shannon, "A symbolic analysis of relay and switching circuits," *Transactions of The American Institute of Electrical Engineers*, 1938.

[23] M. Szyprowski and P. Kerntopf, "Reducing quantum cost in reversible toffoli circuits," *arXiv preprint arXiv:1105.5831*, 2011.