Appendix A

Proof that cluster states and coset states need exponentially large QMDDs

In this appendix, we show that QMDDs which represent both clusters states, and coset states, are exponentially large in the worst case (respectively, Theorem 3.2 and Corollary A.1). On the other hand, in App. B, we will show that these states can be represented using only $\mathcal{O}(n)$ nodes by $\langle X\rangle$-LIMDDs, showing that they are exponentially more succinct than QMDDs. We first fix notation and definitions, after which we prove the theorem using two lemmas.

Let G be an undirected graph with vertices $V_G = \{v_1, ..., v_n\}$ and edge set $E_G \subseteq V_G \times V_G$. For a subset of vertices $S \subseteq V_G$, the S-induced subgraph of G has vertices S and edge set $(S \times S) \cap E$. Given G, its graph state $|G\rangle$ is expressed as

$$|G\rangle = \sum_{\vec{x}\in\{0,1\}^n} (-1)^{f_G(\vec{x})} |\vec{x}\rangle \tag{A.1}$$

where $f_G(\vec{x})$ is the number of edges in the S-induced subgraph of G.

For a function $f : \{0,1\}^n \to \mathbb{C}$ and bit string $\vec{a} = a_1 \cdots a_k \in \{0,1\}^k$, we denote by $f_{\vec{a}}$

the subfunction of f restricted to $\vec{a}$:

$$f_{\vec{a}}(x_{k+1}, \ldots, x_n) := f(a_1, \ldots, a_k, x_{k+1}, \ldots, x_n) \tag{A.2}$$

We also say that $f_{\vec{a}}$ is a subfunction of f of *order* $|\vec{a}| = k$.

We will also need the notions of boundary and strong matching.

Definition A.1 (Boundary)**.** For a set $S \subseteq V_G$ of vertices in G, the *boundary* of S is the set of vertices in S adjacent to a vertex outside of S.

Definition A.2 (Strong Matching)**.** Let $G = (V, E)$ be an undirected graph. A *strong matching* is a subset of edges $M \subseteq E$ that do not share any vertices (i.e., it is a matching) and no two edges of M are incident to the same edge of G, i.e., an edge in $E \setminus M$. Alternatively, a strong matching is a matching M s.t. $G[V(M)] = M$. We say that M is an (S, T)-strong matching for two sets of vertices $S, T \subset V$ if $M \subseteq S \times T$. For a strong matching M and a vertex $v \in V(M)$, we let $M(v)$ denote the unique vertex to which v is matched by M.

Using these definitions and notation, we prove Theorem 3.2.

Theorem 3.2. Denote by $|G_n\rangle$ the two-dimensional cluster state, defined as a graph state on the $n \times n$ lattice. Each QMDD representing $|G_n\rangle$ has at least $2^{\lfloor n/12 \rfloor}$ nodes.

Proof. Let $G = \text{lattice}(n, n)$ be the undirected graph of the $n \times n$ lattice, with vertex set $V = \{v_1, \ldots, v_{n^2}\}$. Let $\sigma = v_1 v_2 \cdots v_{n^2}$ be a variable order, and let $S = \{v_1, v_2, \ldots, v_{\frac{1}{2}n^2}\} \subset V$ be the first $\frac{1}{2}n^2$ vertices in this order.

The proof proceeds broadly as follows. First, in Lemma A.1, we show that any $(S, \overline{S})$-strong matching M effects $2^{|M|}$ different subfunctions of f_G. Second, Lemma A.2 shows that the lattice contains a large $(S, \overline{S})$-strong matching for any choice of S. Put together, this will prove the lower bound on the number of QMDD nodes as in Theorem 3.2 by the fact that a QMDD for the cluster state G has a node per unique subfunction of the function f_G. Figure A.1 illustrates this setup for the 5×5 lattice.

Lemma A.1. Let M be a non-empty $(S, \overline{S})$-strong matching for the vertex set S chosen above. If $\sigma = v_1 v_2 \cdots v_{n^2}$ is a variable order where all vertices in S appear before all vertices in $\overline{S}$, then $f_G(x_1, \ldots, x_{n^2})$ has $2^{|M|}$ different subfunctions of order $|S|$.

Proof. Let $S_M := S \cap V(M)$ and $\overline{S}_M := \overline{S} \cap M$ be the sets of vertices that are involved in the strong matching. Write $\chi(x_1, ..., x_n)$ for the indicator function for vertices: $\chi(x_1, ..., x_n) := \{v_i \mid x_i = 1, i \in [n]\}$. Choose two different subsets $A, B \subseteq S_M$ and let $\vec{a} = \chi^{-1}(A)$ and $\vec{b} = \chi^{-1}(B)$ be the corresponding length-$|S|$ bit strings. These two strings induce the two subfunctions $f_{G,\vec{a}}$ and $f_{G,\vec{b}}$. We will show that these subfunctions differ in at least one point.

First, if $f_{G,\vec{a}}(0,\ldots,0) \neq f_{G,\vec{b}}(0,\ldots,0)$, then we are done. Otherwise, take a vertex $s \in A \oplus B$ and say w.l.o.g. that $s \in A \setminus B$. Let $t = M(s)$ be its partner in the strong matching. Then we have, $|E[A \cup \{t\}]| = |E[A]| + 1$ but $|E[B \cup \{t\}]| = |E[B]|$. Therefore we have

$$f_{G,\vec{a}}(0,\ldots,0,x_t = 0,0,\ldots,0) \neq f_{G,\vec{a}}(0,\ldots,0,x_t = 1,0,\ldots,0) \tag{A.3}$$

$$f_{G,\vec{b}}(0,\ldots,0,x_t = 0,0,\ldots,0) = f_{G,\vec{b}}(0,\ldots,0,x_t = 1,0,\ldots,0) \tag{A.4}$$

We see that each subset of S_M corresponds to a different subfunction of f_G. Since there are $2^{|M|}$ subsets of M, f_G has at least that many subfunctions. □

We now show that the $n \times n$ lattice contains a large enough strong matching.

Lemma A.2. Let $S = \{v_1, \ldots, v_{\frac{1}{2}n^2}\}$ be a set of $\frac{1}{2}n^2$ vertices of the $n \times n$ lattice, as above. Then the graph contains a $(S, \overline{S})$-strong matching of size at least $\lfloor \frac{1}{12}n \rfloor$.

Proof. Consider the boundary B_S of S. This set contains at least $n/3$ vertices, by Theorem 11 in [204]. Each vertex of the boundary of S has degree at most 4. It follows that there is a set of $\lfloor \frac{1}{4}|B_S| \rfloor$ vertices which share no neighbors. In particular, there is a set of $\lfloor \frac{1}{4}|B_S| \rfloor \geq \lfloor \frac{1}{12}n \rfloor$ vertices in B_S which share no neighbors in $\overline{S}$. □

Put together, every choice of half the vertices in the lattice yields a set with a boundary of at least $n/3$ nodes, which yields a strong matching of at least $\lfloor \frac{1}{12}n \rfloor$ edges, which shows that f_G has at least $2^{\lfloor \frac{1}{12}n \rfloor}$ subfunctions of order $\frac{1}{2}n^2$. □

Proof that coset states need exponentially large QMDDs. We now show that QMDDs which represent coset states are exponentially large in the worst case. We will use the following result by Ďuriš et al. on binary decision diagrams (BDDs), which are QMDDs with codomain $\{0, 1\}$. This result concerns vector spaces, but of course, every vector space of $\{0, 1\}^n$ is, in particular, a coset.

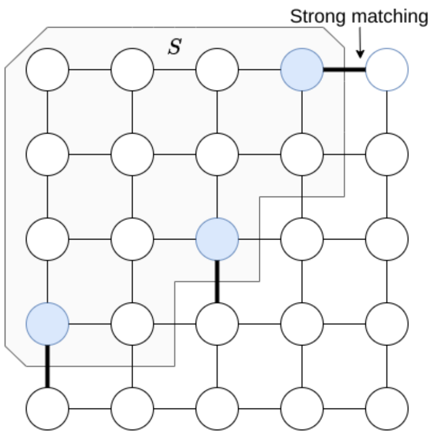

Figure A.1: The 5×5 lattice, partitioned in a vertex set S and its complement $\overline{S}$. A strong matching between S and $\overline{S}$ is indicated by thick black edges. The nodes in S are highlighted.

Theorem A.1 (Ďuriš et al. [108])**.** The characteristic function $f_V : \{0,1\}^n \to \{0,1\}$ of a randomly chosen vector space V in $\{0,1\}^n$, defined as $f_V(x) = 1$ if $x \in V$ and 0 otherwise, needs a BDD of size $2^{\Omega(n)}/(2n)$ with high probability.

Our result follows by noting that if f has codomain $\{0,1\}$ as above, then the QMDD of the state $|f\rangle = \sum_x f(x)\,|x\rangle$ has the same structure as the BDD of f. Consequently, in particular the BDD and QMDD have the same number of nodes.

Corollary A.1. For a random vector space $V \subseteq \{0,1\}^n$, the coset state $|V\rangle$ requires QMDDs of size $2^{\Omega(n)}/(2n)$ with high probability.

Proof. We will show that the QMDD has the same number of nodes as a BDD. A BDD encodes a function $f\colon \{0,1\}^n \to \{0,1\}$. In this case, the BDD encodes f_V, the characteristic function of V. A BDD is a graph which contains one node for each subfunction of f. (In the literature, such a BDD is sometimes called a Full BDD, so that the term BDD is reserved for a variant where the nodes are in one-to-one correspondence with the subfunctions f which satisfy $f_0 \neq f_1$).

Similarly, a QMDD representing a state $|\varphi\rangle = \sum_x f(x)\,|x\rangle$ can be said to represent the function $f\colon \{0,1\}^n \to \mathbb{C}$, and contains one node for each subfunction of f modulo scalars. We will show that, two distinct subfunctions of f_V are never equal up to a scalar. To this end, let $f_{V,a}, f_{V,b}$ be distinct subfunctions of f_V induced by partial assignments $a, b \in \{0,1\}^k$. We will show that there is no $\lambda \in \mathbb{C}^*$ such that $f_{V,a} = \lambda f_{V,b}$. Since the two subfunctions are not pointwise equal, say that the two subfunctions differ in the point $x \in \{0,1\}^{n-k}$, i.e., $f_{V,a}(x) \neq f_{V,b}(x)$. Say without loss of generality that $f_{V,a}(x) = 0$ and $f_{V,b}(x) = 1$. Then, since $\lambda \neq 0$, we have $\lambda = \lambda f_{S,b}(x) \neq f_{V,a}(x) = 0$, so $f_{V,a} \neq \lambda f_{B,b}$.

Because distinct subfunctions of f_V are not equal up to a scalar, the QMDD of $|V\rangle$ contains a node for every unique subfunction of f_V. We conclude that, since by Theorem A.1 with high probability the BDD representing f_V has exponentially many nodes, so does the QMDD representing $|V\rangle$. $\square$

Appendix B

How to write graph states, coset states and stabilizer states as Tower-LIMDDs

In this appendix, we prove that the families of $\langle Z\rangle$-, $\langle X\rangle$-, and $\langle \text{PAULI}\rangle$-Tower-LIMDDs correspond to graph states, coset states, and stabilizer states, respectively, in Theorem B.1, Theorem B.2 and Theorem 3.1 below. Definition 3.5 for reduced PAULI-LIMDDs requires modification for $G = \langle Z\rangle$-LIMDDs because of the absence of X as discussed below the definition. Note that the proofs do not rely on the specialized definition of reduced LIMDDs, but only on Definition 3.2 which allows parameterization of the LIM G. They only rely on the Tower LIMDD in Definition 3.3.

Before we give the proof, we remark that graph states present an interesting special case because the LIMDD's edge labels contain meaningful information. Namely, the labels on the high edges of a graph state's LIMDD are precisely the edges in the original graph. Specifically, suppose a graph G gives rise to a graph state $|\varphi_G\rangle$ represented by a LIMDD. Let $P = P_{k-1} \otimes \cdots \otimes P_1$ be the label on the high edge out of the LIMDD node at level k. Then G contains an edge (v_k, v_j) if and only if $P_j = Z$ (with the roles of k and j reversed if $k < j$). These edge labels come about in a straightforward manner during the construction of the graph state. Namely, the graph state $|\varphi_G\rangle$ is produced by starting from the state $|+\rangle^{\otimes n}$, and applying controlled-Z gates to qubit pairs (u, v) for every edge (u, v) in the graph. Applying such a controlled-Z gate to qubit pair (u, v) has the effect of setting P_v to Z in the high edge outgoing from the vertex at level u. In general, however, the labels on the high edges cannot be easily inferred from the stabilizer state.

A G-Tower-LIMDD representing an n-qubit state is a LIMDD which has n nodes, not counting the leaf. It has G-LIMs on its high edges. Definition 3.3 gives an exact definition.

Theorem B.1 (Graph states are $\langle Z\rangle$-Tower-LIMDDs)**.** Let $n \geq 1$. Denote by $\mathcal{G}_n$ the set of n-qubit graph states and write $\mathcal{Z}_n$ for the set of n-qubit quantum states which are represented by $\langle Z\rangle$-Tower-LIMDDs a defined in Definition 3.3, i.e, a tower with low-edge-labels $\mathbb{I}$ and high-edge labels $\lambda \bigotimes_j P_j$ with $P_j \in \{\mathbb{I}, Z\}$ and $\lambda = 1$, except for the root edge where $\lambda \in \mathbb{C} \setminus \{0\}$. Then $\mathcal{G}_n = \mathcal{Z}_n$.

Proof. We establish $\mathcal{G}_n \subseteq \mathcal{Z}_n$ by providing a procedure to convert any graph state in $\mathcal{G}_n$ to a $\langle Z\rangle$-Tower-LIMDD in $\mathcal{Z}_n$. See Figure B.1 for an example of a 4-qubit graph state. We describe the procedure by induction on the number n of qubits in the graph state.

Base case: $n = 1$**.** We note that there is only one single-qubit graph state by definition (see Equation A.1), which is $|+\rangle := (|0\rangle + |1\rangle)/\sqrt{2}$ and can be represented as LIMDD by a single node (in addition to the leaf node): see Figure B.1(a).

Induction case. We consider an $(n+1)$-qubit graph state $|G\rangle$ corresponding to the graph G. We isolate the $(n+1)$-th qubit by decomposing the full state definition from Equation A.1:

$$|G\rangle = \frac{1}{\sqrt{2}}\left(|0\rangle \otimes |G_{1..n}\rangle + |1\rangle \otimes \underbrace{\left[\bigotimes_{(n+1,j)\in E} Z_j\right]}_{\text{Isomorphism B}} |G_{1..n}\rangle\right) \tag{B.1}$$

where E is the edge set of G and $G_{1..n}$ is the induced subgraph of G on vertices 1 to n. Thus, $|G_{1..n}\rangle$ is an n-qubit graph state on qubits 1 to n. Since $|G_{1..n}\rangle$ is a graph state on n qubits, by the induction hypothesis, we have a procedure to convert it to a $\langle Z\rangle$-Tower-LIMDD $\in \mathcal{Z}_n$. Now we construct a $\langle Z\rangle$-Tower-LIMDD for $|G\rangle$ as follows. The root node has two outgoing edges, both going to the node representing $|G_{1..n}\rangle$.

The node's low edge has label $\mathbb{I}$, and the node's high edge has label B, as follows,

$$B = \bigotimes_{(n+1,j)\in E} Z_j \tag{B.2}$$

Thus the root node represents the state $|0\rangle |G_{1..n}\rangle + |1\rangle B |G_{1..n}\rangle$, satisfying Equation B.1.

To prove $\mathcal{Z}_n \subseteq \mathcal{G}_n$, we show how to construct the graph corresponding to a given $\langle Z\rangle$-Tower LIMDD. Briefly, we simply run the algorithm outlined above in reverse, constructing the graph one node at a time. Here we assume without loss of generality that the low edge of every node is labeled $\mathbb{I}$.

Base case. The LIMDD node above the Leaf node, representing the state $|+\rangle$, always represents the singleton graph, containing one node.

Induction case. Suppose that the LIMDD node $k+1$ levels above the Leaf has a low edge labeled $\mathbb{I}$, and a high edge labeled $P_k \otimes \cdots \otimes P_1$, with $P_j = Z^{a_j}$ for $j = 1\ldots k$. Here by Z^{a_j} we mean $Z^0 = \mathbb{I}$ and $Z^1 = Z$. Then we add a node labeled $k+1$ to the graph, and connect it to those nodes j with $a_j = 1$, for $j = 1\ldots k$. The state represented by this node is of the form given in Equation B.1, so it represents a graph state.

A simple counting argument based on the above construction shows that $|\mathcal{Z}_n| = |\mathcal{G}_n| = 2^{\binom{n}{2}}$, so the conversion is indeed a bijection. Namely, there are $2^{\binom{n}{2}}$ graphs, since there are $\binom{n}{2}$ edges to choose, and there are $2^{\binom{n}{2}}$ $\langle Z\rangle$-Tower-LIMDDs, because the total number of single-qubit operators of the LIMs on the high edges is $\binom{n}{2}$, each of which can be independently chosen to be either $\mathbb{I}$ or Z. $\square$

We now prove that coset states are represented by $\langle X\rangle$-Tower-LIMDDs.

Theorem B.2 (coset states are $\langle X\rangle$-Tower-LIMDDs)**.** Let $n \geq 1$. Denote by $\mathcal{V}_n$ the set of n-qubit coset states and write $\mathcal{X}_n$ for the set of n-qubit quantum states which are represented by $\langle X\rangle$-Tower-LIMDDs as per Definition 3.3, i.e., a tower with low edge labels $\mathbb{I}$ and high edge labels $\lambda \bigotimes_j P_j$ with $P_j \in \{\mathbb{I}, X\}$ and $\lambda \in \{0, 1\}$, except for the root edge where $\lambda \in \mathbb{C} \setminus \{0\}$. Then $\mathcal{V}_n = \mathcal{X}_n$.

Proof. We first prove $\mathcal{V}_n \subseteq \mathcal{X}_n$ by providing a procedure for constructing a Tower-LIMDD for a coset state. We prove the statement for the case when C is a group rather than a coset; the result will then follow by noting that, by placing the label

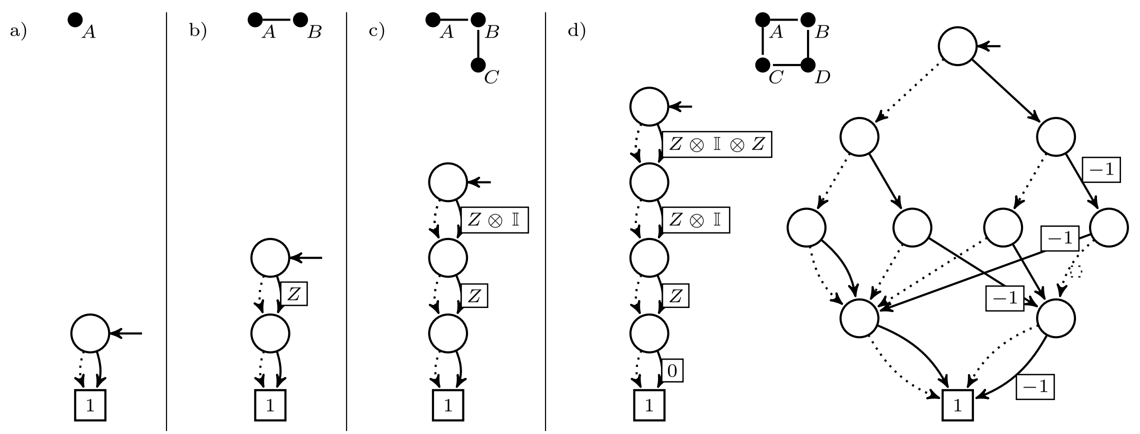

Figure B.1: Construction of the $\langle Z \rangle$-Tower LIMDD for the 4-qubit cluster state, by iterating over the vertices in the graph, as described in the proof of Theorem B.1. (a) First, we consider the single-qubit graph state, which corresponds to a the subgraph containing only vertex A. (b) Then, we add vertex B, which is connected to A by an edge. The resulting LIMDD is constructed from the LIMDD from (a) by adding a new root node. In the figure, the isomorphism is $Z_B \otimes \mathbb{I}[A]$, since vertex C is connected to vertex B (yielding the Z operator) but not to A (yielding the identity operator $\mathbb{I}$). (c) This process is repeated for a third vertex C until we reach the LIMDD of the full 4-qubit cluster state (d). For comparison, (d) also depicts a regular QMDD for the same graph state, which has width 4 instead of 1 for the LIMDD.

$X^{a_n} \otimes \cdots \otimes X^{a_1}$ on the root edge, we obtain the coset state $|C + a\rangle$. The procedure is recursive on the number of qubits.

Base case: $n = 1$**.** In this case, there are two coset states: $|0\rangle$ and $(|0\rangle + |1\rangle)/\sqrt{2}$, which are represented by a single node which has a low and high edge pointing to the leaf node with low/high edge labels 1/0 and 1/1, respectively.

Induction case. Now consider an $(n + 1)$-qubit coset state $|S\rangle$ for a group $S \subseteq \{0, 1\}^{n+1}$ for some $n \geq 1$ and assume we have a procedure to convert any n-qubit coset state into a Tower-LIMDD in $\mathcal{X}_n$. We consider two cases, depending on whether the first bit of each element of S is zero:

(a) The first bit of each element of S is 0. Thus, we can write $S = \{0x \mid x \in S_0\}$ for some set $S_0 \subseteq \{0, 1\}^n$. Then $0a, 0b \in S \implies 0a \oplus 0b \in S$ implies $a, b \in S_0 \implies a \oplus b \in S_0$ and thus S_0 is an length-n bit string vector space. Thus by assumption, we have a procedure to convert it to a Tower-LIMDD in $\mathcal{X}_n$. Convert it into a Tower-LIMDD in $\mathcal{X}_{n+1}$ for $|S\rangle$ by adding a fresh node on top with low edge label $\mathbb{I}^{\otimes n}$ and high edge label 0, both pointing to the the root S.

(b) There is some length-n bit string u such that $1u \in S$. Write S as the union of the sets $\{0x \mid x \in S_0\}$ and $\{1x \mid x \in S_1\}$ for sets $S_0, S_1 \subseteq \{0, 1\}^n$. Since S is closed under element-wise XOR, we have $1u \oplus 1x = 0(u \oplus x) \in S$ for each $x \in S_1$ and therefore $u \oplus x \in S_0$ for each $x \in S_1$. This implies that $S_1 = \{u \oplus x \mid x \in S_0\}$ and thus S is the union of $\{0x \mid x \in S_0\}$ and $\{1u \oplus 0x \mid x \in S_0\}$. By similar reasoning as in case (a), we can show that S_0 is a vector space on length-n bit strings.

   We build a Tower-LIMDD for $|S\rangle$ as follows. By the induction hypothesis, there is a Tower-LIMDD with root node v which represents $|v\rangle = |S_0\rangle$. We construct a new node whose two outgoing edges both go to this node v. Its low edge has label $\mathbb{I}^{\otimes n}$ and its high edge has label $P = P_n \otimes \cdots \otimes P_1$ where $P_j = X$ if $u_j = 1$ and $P_j = \mathbb{I}$ if $u_j = 0$.

We now show $\mathcal{V}_n \subseteq \mathcal{X}_n$, also by induction.

Base case: $n = 1$**.** There are only two Tower-LIMDDs on 1 qubit satisfying the description above, namely

(1) A node whose two edges point to the leaf. Its low edge has label 1, and its high

edge has label 0. This node represents the coset state $|0\rangle$, corresponding to the vector space $V = \{0\} \subseteq \{0,1\}^1$.

(2) A node whose two edges point to the leaf. Its low edge has label 1 and its high edge also has label 1. This node represents the coset state $|0\rangle + |1\rangle$, corresponding to the vector space $V = \{0,1\}$.

Induction case. Let v be the root node of an $n+1$-qubit Tower $\langle X\rangle$-LIMDD as described above. We distinguish two cases, depending on whether v's high edge has label 0 or not.

(a) The high edge has label 0. Then $|v\rangle = |0\rangle |v_0\rangle$ for a node v_0, which represents a coset state $|v_0\rangle$ corresponding to a coset $V_0 \subseteq \{0,1\}^n$, by the induction hypothesis. Then v corresponds to the coset $\{0x \mid x \in V_0\}$.

(b) the high edge has label $P = P_n \otimes \cdots \otimes P_1$ with $P_j \in \{\mathbb{I}, X\}$. Then $|v\rangle = |0\rangle |v_0\rangle + |1\rangle \otimes P |v_0\rangle$. By the observations above, this is a coset state, corresponding to the vector space $V = \{0x | x \in V_0\} \cup \{1(ux) | x \in V_0\}$ where $u \in \{0,1\}^n$ is a string whose bits are $u_j = 1$ if $P_j = X$ and $u_j = 0$ if $P_j = \mathbb{I}$, and V_0 is the vector space corresponding to the coset state $|v_0\rangle$.

□

Lastly, we prove the stabilizer-state case, showing that they are exactly equivalent to the $\langle$Pauli$\rangle$-Tower-LIMDD, as defined in Definition 3.3. For this, we first need Lemma B.1 and Lemma B.2, which state that, if one applies a Clifford gate to a $\langle$Pauli$\rangle$-Tower-LIMDD, the resulting state is another $\langle$Pauli$\rangle$-Tower-LIMDD. First, Lemma B.1 treats the special case of applying a gate to the top qubit; then Lemma B.2 treats the general case of applying a gate to an arbitrary qubit.

Lemma B.1. Let $|\varphi\rangle$ be an n-qubit stabilizer state which is represented by a $\langle$Pauli$\rangle$-Tower-LIMDD as defined in Definition 3.3. Let U be either a Hadamard gate or S gate on the top qubit (n-th qubit), or a downward CNOT with the top qubit as control. Then $U|\varphi\rangle$ is still represented by a $\langle$Pauli$\rangle$-Tower-LIMDD.

Proof. The proof is on the number n of qubits.

Base case: $n = 1$. For $n = 1$, there are six single-qubit stabilizer states $|0\rangle, |1\rangle$ and $(|0\rangle + \alpha|1\rangle)/\sqrt{2}$ for $\alpha \in \{\pm 1, \pm i\}$. There are precisely represented by Pauli-Tower-LIMDDs with high edge label factor $\in \{0, \pm 1, \pm i\}$ as follows:

- for $|0\rangle$: $\textcircled{1} \overset{1}{\cdots} \bigcirc \overset{0}{—} \textcircled{1}$
- for $|1\rangle$: $A \cdot \textcircled{1} \overset{1}{\cdots} \bigcirc \overset{0}{—} \textcircled{1}$ where $A \propto X$ or $A \propto Y$
- for $(|0\rangle + \alpha\,|1\rangle)/\sqrt{2})$: $\textcircled{1} \overset{1}{\cdots} \bigcirc \overset{\alpha}{—} \textcircled{1}$

Since the H and S gate permute these six stabilizer states, $U\,|\varphi\rangle$ is represented by a $\langle$Pauli$\rangle$-Tower-LIMDD if $|\varphi\rangle$ is.

Induction case. For $n > 1$, we first consider $U = S$ and $U = \text{CNOT}$. Let R be the label of the root edge. If $U = S$, then the high edge of the top node is multiplied with i, while a downward CNOT (target qubit with index k) updates the high edge label $A \mapsto X_k A$. Next, the root edge label is updated to $URU^\dagger$, which is still a Pauli string, since U is a Clifford gate. Since the high labels of the top qubit in the resulting diagram is still a Pauli string, and the high edge's weights are still $\in \{0, \pm 1, \pm i\}$, we conclude that both these gates yield a $\langle$Pauli$\rangle$-Tower-LIMDD. Finally, for the Hadamard, we decompose $|\varphi\rangle = |0\rangle \otimes |\psi\rangle + \alpha\,|1\rangle \otimes P\,|\psi\rangle$ for some $(n-1)$-qubit stabilizer state $|\psi\rangle$, $\alpha \in \{0, \pm 1, \pm i\}$ and P is an $(n-1)$-qubit Pauli string. Now we note that $H\,|\varphi\rangle \propto |0\rangle \otimes |\psi_0\rangle + |1\rangle \otimes |\psi_1\rangle$ where $|\psi_x\rangle := (\mathbb{I} + (-1)^x \alpha P)\,|\psi\rangle$ with $x \in \{0, 1\}$. Now we consider two cases, depending on whether P commutes with all stabilizers of $|\psi\rangle$:

(a) There exist a stabilizer g of $|\psi\rangle$ which anticommutes with P. We note two things. First, $\langle\psi|P|\psi\rangle = \langle\psi|Pg|\psi\rangle = \langle\psi|g \cdot (-P)|\psi\rangle = -\langle\psi|P|\psi\rangle$, hence $\langle\psi|P|\psi\rangle = 0$. It follows from Lemma 15 of [125] that $|\psi_x\rangle$ is a stabilizer state, so by the induction hypothesis it can be written as a $\langle$Pauli$\rangle$-Tower-LIMDD. Let v be the root node of this LIMDD. Next, we note that $g\,|\psi_0\rangle = g(\mathbb{I} + \alpha P)\,|\psi\rangle = (\mathbb{I} - \alpha P)g\,|\psi\rangle = |\psi_1\rangle$. Hence, $\textcircled{v} \overset{\mathbb{I}}{\cdots} \bigcirc \overset{g}{—} \textcircled{v}$ is the root node of a $\langle$Pauli$\rangle$-Tower-LIMDD for $H\,|\varphi\rangle$.

(b) All stabilizers of $|\psi\rangle$ commute with P. Then $(-1)^y P$ is a stabilizer of $|\psi\rangle$ for either $y = 0$ or $y = 1$. Hence, $|\psi_x\rangle = (\mathbb{I} + (-1)^x \alpha P)\,|\psi\rangle = (1 + (-1)^{x+y}\alpha)\,|\psi\rangle$. Therefore, $|\varphi\rangle = |a\rangle \otimes |\psi\rangle$ where $|a\rangle := (1 + (-1)^y \alpha)\,|0\rangle + (1 + (-1)^{y+1}\alpha\,|1\rangle)$. It is not hard to see that $|a\rangle$ is a stabilizer state for all choices of $\alpha \in \{0, \pm 1, \pm i\}$. By the induction hypothesis, both $|a\rangle$ and $|\psi\rangle$ can be represented as $\langle$Pauli$\rangle$-Tower-LIMDDs. We construct a $\langle$Pauli$\rangle$-Tower-LIMDD for $H\,|\varphi\rangle$ by replacing the leaf of the LIMDD of $|a\rangle$ by the root node of the LIMDD of $|\psi\rangle$, and propagating the root edge label of $|\psi\rangle$ upwards. Specifically, if the root edge of $|a\rangle$ is $\overset{A}{—}\textcircled{v}$

with $v = \textcircled{1}\overset{1}{\cdots}\bigcirc\overset{\beta}{\text{——}}\textcircled{1}$, and if the root edge of $|\psi\rangle$ is $\overset{B}{\text{——}}\textcircled{w}$, then a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD for $H|\varphi\rangle$ has root node $\textcircled{\mathbb{I}}\overset{w}{\cdots}\bigcirc\overset{\beta\mathbb{I}}{\text{——}}\textcircled{w}$ and has root edge label $A \otimes B$.

$\square$

Lemma B.2. Let $|\varphi\rangle$ be an n-qubit state state represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD, as defined in Definition 3.3. Let U be either a Hadamard gate, an S gate or a CNOT gate. Then $U|\varphi\rangle$ is a state which is also represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD.

Proof. The proof is by induction on n. The case $n = 1$ is covered by Lemma B.1. Suppose that the induction hypothesis holds, and let $|\varphi\rangle$ be an $n+1$-qubit state represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD. First, we note that a CNOT gate CX_c^t can be written as $CX_c^t = (H \otimes H)CX_t^c(H \otimes H)$, so without loss of generality we may assume that $c > t$. We treat two cases, depending on whether U affects the top qubit or not.

(a) U affects the top qubit. Then $U|\varphi\rangle$ is represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD, according to Lemma B.1.

(b) U does not affect the top qubit. Suppose $|\varphi\rangle = |0\rangle \otimes |\varphi_0\rangle + |1\rangle \otimes \alpha P|\varphi_0\rangle$ (with P a Pauli string and $\alpha \in \{0, \pm 1, \pm i\}$). Then $U|\varphi\rangle = |0\rangle \otimes U|\varphi_0\rangle + |1\rangle \otimes (\alpha UPU^\dagger)U|\varphi_0\rangle$. Since U is either a Hadamard, S gate or CNOT, and $|\varphi_0\rangle$ is an n-qubit state, the induction hypothesis states that the state $U|\varphi_0\rangle$ is represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD. Let $\overset{A}{\text{——}}\textcircled{v}$ be the root edge of this $\langle\textsc{Pauli}\rangle$-Tower-LIMDD, representing $U|\varphi_0\rangle$. Then $U|\varphi\rangle$ is represented by the root edge $\overset{\mathbb{I}\otimes A}{\text{———}}\textcircled{w}$, where w is the node $\textcircled{v}\overset{\mathbb{I}}{\cdots}\bigcirc\overset{\alpha A^{-1}UPU^\dagger A}{\text{—————}}\textcircled{v}$. The label $\alpha A^{-1}UPU^\dagger A$ is a Pauli LIM, and may therefore be used as the label on the high edge of w.

$\square$

Finally, we show that stabilizer states are precisely the $\langle\textsc{Pauli}\rangle$-Tower-LIMDDs.

Theorem 3.1. Let $n > 0$. Each n-qubit stabilizer state is represented up to normalization by a $\langle\textsc{Pauli}\rangle$-Tower LIMDDs of Definition 3.3, e.g., where the scalars λ of the PAULILIMs λP on high edges are restricted as $\lambda \in \{0, \pm 1, \pm i\}$. Conversely, every such LIMDD represents a stabilizer state.

Proof. We first prove that each stabilizer state is represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD. We recall that each stabilizer state can be obtained as the output state of a Clifford circuit on input state $|0\rangle^{\otimes n}$. Each Clifford circuit can be decomposed into solely the gates H, S and CNOT. The state $|0\rangle^{\otimes n}$ is represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD. According to Lemma B.2, applying an H, S or CNOT gate to a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD results a state represented by another $\langle\textsc{Pauli}\rangle$-Tower-LIMDD. One can therefore apply the gates of a Clifford circuit to the initial state $|0\rangle$, and obtain a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD for every intermediate state, including the output state. Therefore, every stabilizer state is represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD.

For the converse direction, the proof is by induction on n. We only need to note that a state represented by a $\langle\textsc{Pauli}\rangle$-Tower-LIMDD can be written as $|\varphi\rangle = |0\rangle \otimes |\varphi_0\rangle + |1\rangle \otimes \alpha P |\varphi_0\rangle = C(P)(|0\rangle + \alpha |1\rangle) \otimes |\varphi_0\rangle$ where $C(P) := |0\rangle\langle 0| \otimes \mathbb{I} + |1\rangle\langle 1| \otimes P$ is the controlled-(P) gate. Using the relations $Z = HXH$, $Y = SXS^\dagger$ and $S = Z^2$, we can decompose $C(P)$ as CNOT, H and S, hence $C(P)$ is a Clifford gate. Since both $|0\rangle + \alpha |1\rangle$ and $|\varphi_0\rangle$ can be written as $\langle\textsc{Pauli}\rangle$-Tower-LIMDDs, they are stabilizer states by the induction hypothesis. Therefore, the state $|\psi\rangle = (|0\rangle + \alpha |1\rangle) \otimes |\varphi_0\rangle$ is also a stabilizer state. Thus, the state $|\varphi\rangle = C(P) |\psi\rangle$ is obtained by applying the Clifford gate $C(P)$ to the stabilizer state $|\varphi\rangle$. Therefore, $|\varphi\rangle$ is a stabilizer state. $\square$

Appendix C

Advanced LIMDD algorithms

C.1 Measuring an arbitrary qubit

Algorithm 18 allows one to measure a given qubit. Specifically, given a quantum state $|e\rangle$ represented by a LIMDD edge e, a qubit index k and an outcome $b \in \{0, 1\}$, it computes the probability of observing $|b\rangle$ when measuring the k-th significant qubit of $|e\rangle$. The algorithm proceeds by traversing the LIMDD with root edge e at Line 7. Like Algorithm 5, which measured the top qubit, this algorithm finds the probability of a given outcome by computing the squared norm of the state when the k-th qubit is projected onto $|0\rangle$, or $|1\rangle$. The case that is added, relative to Algorithm 5, is the case when $n > k$, in which case it calls the procedure SQUAREDNORMPROJECTED. On input e, y, k, the procedure SQUAREDNORMPROJECTED outputs the squared norm of $\Pi_k^y |e\rangle$, where $\Pi_k^y = \mathbb{I}[n-k] \otimes |y\rangle\langle y| \otimes \mathbb{I}[k-1]$ is the projector which projects the k-th qubit onto $|y\rangle$.

After measurement of a qubit k, a quantum state is typically projected to $|0\rangle$ or $|1\rangle$ ($b = 0$ or $b = 1$) on that qubit, depending on the outcome. Algorithm 19 realizes this. It does so by traversing the LIMDD until a node v with $\mathsf{idx}(v) = k$ is reached. It then returns an edge to a new node by calling $\text{MAKEEDGE}(\text{FOLLOW}(0, e), 0)$ to project onto $|0\rangle$ or $\text{MAKEEDGE}(0, \text{FOLLOW}(1, e))$ to project onto $|1\rangle$, on Line 6, recreating a node on level k in the backtrack on Line 8. The projection operator Π_k^b commutes with any LIM P when P_k is a diagonal operator (i.e., $P_k \in \{\mathbb{I}[2], Z\}$). Otherwise, if P_k is

Algorithm 18 Compute the probability of observing $|y\rangle$ when measuring the k-th qubit of the state $|e\rangle$. Here e is given as LIMDD on n qubits, y is given as a bit, and k is an integer index. For example, to measure the top-most qubit, one calls $\textsc{Measure}(e, 0, n)$. The procedure $\textsc{SquaredNorm}(e, y, k)$ computes the scalar $\langle e| (\mathbb{I} \otimes |y\rangle \langle y| \otimes \mathbb{I}) |e\rangle$, i.e., computes the squared norm of the state $|e\rangle$ after the k-th qubit is projected to $|y\rangle$. For readability, we omit calls to the cache, which implement dynamic programming.

```
1:  procedure MeasurementProbability(Edge $e \xrightarrow{\lambda P_n \otimes P'} \textcircled{v}$, $y \in \{0,1\}$, $k \in [1...\mathsf{idx}(v)]$)
2:      if $n = k$ then
3:          $p_0 :=$ SquaredNorm(follow$(0, e)$)
4:          $p_1 :=$ SquaredNorm(follow$(1, e)$)
5:          return $p_j/(p_0 + p_1)$ where $j = 0$ if $P_n \in \{\mathbb{I}, Z\}$ and $j = 1$ if $P_n \in \{X, Y\}$
6:      else
7:          $p_0 :=$ SquaredNormProjected(follow$(0, e), y, k$)
8:          $p_1 :=$ SquaredNormProjected(follow$(1, e), y, k$)
9:          return $(p_0 + p_1)/$SquaredNorm$(e)$
10: procedure SquaredNorm(Edge $\xrightarrow{\lambda P} \textcircled{v}$)
11:     if $n = 0$ then return $|\lambda|^2$
12:     $s :=$ Add(SquaredNorm(follow$(0, \xrightarrow{\mathbb{I}} \textcircled{v})$), SquaredNorm(follow$(1, \xrightarrow{\mathbb{I}} \textcircled{v})$))
13:     return $|\lambda|^2 s$
14: procedure SquaredNormProjected(Edge $e \xrightarrow{\lambda P_n \otimes P'} \textcircled{v}$, $y \in \{0,1\}$, $k \in [1...\mathsf{idx}(v)]$)
15:     $b := (P_n \in \{X, Y\})$                         ▷ i.e., $b = 1$ iff $P_n$ is Anti-diagonal
16:     if $n = 0$ then
17:         return $|\lambda|^2$
18:     else if $n = k$ then
19:         return SquaredNorm(follow$(b \oplus y, e)$)
20:     else
21:         $\alpha_0 :=$ SquaredNormProjected(follow$(0, \xrightarrow{\mathbb{I}} \textcircled{v}), b \oplus y, k$)
22:         $\alpha_1 :=$ SquaredNormProjected(follow$(1, \xrightarrow{\mathbb{I}} \textcircled{v}), b \oplus y, k$)
23:         return $|\lambda|^2 \cdot (\alpha_0 + \alpha_1)$
```

an antidiagonal operator (i.e, $P_k \in \{X, Y\}$), have $\Pi_k^b \cdot P = P\Pi_k^{(1-b)}$. The algorithm applies this correction on Line 2. The resulting state should still be normalized as shown in Sec. 3.3.3.1.

Sampling. To sample from a quantum state in the computational basis, simply repeat the three-step measurement procedure outlined in Sec. 3.3.3.1 n times: once for each

Algorithm 19 Project the state given by LIMDD $\xrightarrow{A}\textcircled{v}$ to state $|b\rangle$ for qubit k, i.e., produce a LIMDD representing the state $(\mathbb{I}[n-k] \otimes |b\rangle\langle b| \otimes \mathbb{I}[k-1]) \cdot A|v\rangle$, with $A = \lambda P_n \otimes \cdots \otimes P_1$.

```
1: procedure UpdatePostMeas(Edge $\xrightarrow{\lambda P_n \otimes \cdots \otimes P_1}\textcircled{v}$, $k \in [1...\mathsf{idx}(v)]$, $b \in \{0,1\}$)
2:     $b' := x \oplus b$ where $x = 0$ if $P_k \in \{\mathbb{I}, Z\}$ and $x = 1$ if $P_k \in \{X, Y\}$ ▷ flip $b$ if $P_k$ is anti-diagonal
3:     if $(v, k, b') \in$ Cache then return Cache$[v, k, b']$
4:     $n := \mathsf{idx}(v)$
5:     if $n = k$ then
6:         $e :=$ MakeEdge$((1-b') \cdot \mathsf{low}_v,\ b' \cdot \mathsf{high}_v)$ ▷ Project $|v\rangle$ to $|b'\rangle\langle b'| \otimes \mathbb{I}[2]^{\otimes n-1}$
7:     else                                                        ▷ $n \neq k$:
8:         $e :=$ MakeEdge(UpdatePostMeas$(\mathsf{low}_v, k, b')$, UpdatePostMeas$(\mathsf{high}_v, k, b')$)
9:     Cache$[v, k, b'] := e$
10:    return $e$
```

qubit.

Strong simulation. To compute the probability of observing a given bit-string $x = x_n \ldots x_1$, first compute the probability p_n of observing $|x_n\rangle$; then update the LIMDD to outcome x_n, obtaining a new, smaller LIMDD. On this new LIMDD, compute the probability p_{n-1} of observing $|x_{n-1}\rangle$, and so forth. Note that, because the LIMDD was updated after observing the measurement outcome $|x_n\rangle$, p_{n-1} is the probability of observing x_{n-1} given that the top qubit is measured to be x_n. Then the probability of observing the string x is the product $p = p_1 \cdots p_n$.

C.2 Applying Hadamards to stabilizer states in polynomial time

We show that, using the algorithms that we have given,* a Hadamard can be applied to a stabilizer state in polynomial time (Theorem C.1). Together with the algorithms for the other Clifford gates, presented in Sec. 3.3.3.2, this shows that all Clifford gates can be applied to stabilizer states in polynomial time. We emphasize that our algorithms do not invoke existing algorithms that are tailored to applying a Hadamard

*We make minor modifications to the Add algorithm, which are presented in Theorem C.1

to a stabilizer state; instead, the LIMDD algorithms are inherently polynomial-time for this use case. The key ingredient is Lemma C.3, which describes situations in which the ADD procedure adds two stabilizer states in polynomial time. It shows that only $5n$ distinct calls to ADD are made. Our algorithms are polynomial-time because of the dynamic programming effected by the caching of previously computed results, as described in Sec. 3.3.3.3, which, in this case, makes sure only $5n$ recursive calls are made.

Theorem C.1. The algorithm HGATE($\xrightarrow{A}\textcircled{v}, k$) (Algorithm 10) takes polynomial time when the input edge $\xrightarrow{A}\textcircled{v}$ represents a stabilizer state.

Proof. Due to the cache, the algorithm HGATE effects only one recursive call per node. The LIMDD of a stabilizer state has one node on each of the n layers, so there are at most n recursive calls.

When the algorithm arrives at layer k, it makes two calls to ADD. Both calls are of the form ADD($\xrightarrow{\lambda P}\textcircled{v'}$, $\xrightarrow{\omega Q}\textcircled{v'}$) where $\lambda, \omega \in \{0, \pm 1, \pm i\}$, where P and Q are Pauli strings, and v' is a node representing a stabilizer state, namely v' is the node at the $(k-1)$-th level of the LIMDD. This satisfies the conditions of Lemma C.3; therefore, both calls to ADD make at most $5k = \mathcal{O}(n)$ recursive calls in total. Each recursive call to ADD may invoke the MAKEEDGE procedure, which runs in time $\mathcal{O}(n^3)$, yielding a total worst-case runtime of $\mathcal{O}(n^4)$. Since there are two calls to ADD, the total runtime of HGATE is also $\mathcal{O}(n^4)$.

Lastly, for completeness we note that the call to ADD($\xrightarrow{\lambda P}\textcircled{v'}$, $\xrightarrow{\omega Q}\textcircled{v'}$) may have $\lambda = 0$ or $\omega = 0$, i.e., one of the operands may be the zero vector. For readability, we have presented the ADD algorithm (Algorithm 9) without treatment of this case, when one of the edges is the zero vector. For the purposes of this proof, we therefore add the following two lines to the ADD algorithm:

```
1: procedure ADD(EDGE $e = \xrightarrow{\alpha P}\textcircled{v}$, EDGE $f = \xrightarrow{\beta Q}\textcircled{w}$)
2:     ...
3:     if $\alpha = 0$ then return $\xrightarrow{\beta Q}\textcircled{w}$
4:     if $\beta = 0$ then return $\xrightarrow{\alpha P}\textcircled{v}$
5:     ...
```

These simple checks are also present in the C++ implementation presented in Chapter 4 and is routine in DD implementations, such as the matrix addition algorithm described by Miller and Thornton [227]. Consequently, a call to ADD($\xrightarrow{\alpha P}\textcircled{v}$, $\xrightarrow{\beta Q}\textcircled{w}$) runs in $\mathcal{O}(1)$ time if $\alpha = 0$ or $\beta = 0$. $\square$

We now prepare Lemma C.3, which is the main technical ingredient. It states that all the recursive calls to ADD effect only five different cache entries at any given level of the LIMDD. To this end, the strategy is (1) to look closely at which recursive calls made by ADD; (2) to look closely at when a cache hit is achieved; and (3) to inspect the FOLLOW procedure.

The recursive calls of ADD. First, we will find a good description of the set of recursive calls made by a call to ADD. We note that each call to ADD makes two recursive calls. Specifically, when it is called with parameters ADD($\xrightarrow{\alpha P}\textcircled{v}$, $\xrightarrow{\beta Q}\textcircled{v}$), it makes two recursive calls, of the following form,

$$\text{ADD}(\text{FOLLOW}(x, \xrightarrow{\alpha P}\textcircled{v}), \text{FOLLOW}(x, \xrightarrow{\beta Q}\textcircled{v})) \qquad \text{for } x \in \{0,1\} \tag{C.1}$$

These calls subsequently call ADD again, recursively. Let us temporarily forget that some of these calls may not happen because a cache hit preempts them (namely, the ADD does not recurse in the cache of a cache hit). Then the set of recursive calls to ADD is described by calls of the following form,

$$\text{ADD}(\text{FOLLOW}(x, \xrightarrow{\alpha P}\textcircled{v}), \text{FOLLOW}(x, \xrightarrow{\beta Q}\textcircled{v})) \quad \text{for } x \in \{0,1\}^{\ell} \text{ for } 0 \leq \ell \leq n \tag{C.2}$$

Cache hits of ADD. Inspecting the algorithm ADD (Algorithm 9) in Sec. 3.3.3.3, we see that a call to ADD with parameters ($\xrightarrow{A}\textcircled{v}$, $\xrightarrow{B}\textcircled{v}$) effects a cache hit if and only if ADD was previously called with ($\xrightarrow{C}\textcircled{v}$, $\xrightarrow{D}\textcircled{v}$) satisfying $A^{-1}B\,|v\rangle = C^{-1}D\,|v\rangle$. Therefore, let us associate a given call to ADD($\xrightarrow{\alpha P}\textcircled{v}$, $\xrightarrow{\beta Q}\textcircled{v}$) with the operator $\alpha^{-1}\beta P^{-1}Q$. Then a call to ADD with associated operator U will effect a cache hit if a previous call to ADD was associated with the same operator U.[†]

The FOLLOW procedure. We now turn to the FOLLOW procedure. The procedure FOLLOW$(x, \xrightarrow{\alpha P}\textcircled{v})$ outputs an edge $\xrightarrow{A}\textcircled{t}$, labeled with some label A. Let $L(x, \xrightarrow{\alpha P}\textcircled{v})$ be the function which outputs this label, i.e., $L(x, \xrightarrow{\alpha P}\textcircled{v}) = A$. In this paragraph, we aim to find a closed-form expression for $L(x, \xrightarrow{\alpha P}\textcircled{v})$ in the case

[†]More precisely, a call to ADD associated with U effects a cache hit if and only if a previous call was associated with an operator U' satisfing $U \cdot \text{Stab}(\varphi) = U' \cdot \text{Stab}(\varphi)$. Here $U \cdot \text{Stab}(\varphi)$ is the coset obtained by left-multiplying the group $\text{Stab}(\varphi)$ with U. Therefore, the condition $U = U'$, named above, is sufficient, but not necessary.

of Tower PAULI-LIMDDs. If the node v is clear from context, we will write simply $L(x, P)$.

It is useful to conceive of the FOLLOW procedure as traversing a path from v to t of length $\ell = |x|$. Then the label $L(x, \xrightarrow{\alpha P}\textcircled{v})$ is the product of the LIMs on the edges that were traversed (including the label P), after which we discard the most significant ℓ qubits. More precisely, for any Pauli string $A = \alpha P_n \otimes \cdots \otimes P_1$, denote with $A^{(\ell)} = \alpha P_{n-\ell} \otimes \cdots \otimes P_1$ the least significant $(n-\ell)$ gates of A, so that, e.g., $A = P_n \otimes P_{n-1} \otimes P^{(2)}$. (In other words, $A^{(\ell)}$ *discards* the ℓ most significant qubits of A). Then, if the FOLLOW procedure traverses edges $e_1, e_2, \ldots, e_\ell$, labeled with LIMs $A_1, A_2, \ldots, A_\ell$, respectively, then

$$L(x, \xrightarrow{A_1}\textcircled{v}) = \lambda A_1^{(\ell)} \cdot A_2^{(\ell)} \cdots A_\ell^{(\ell)} \qquad \text{for some } \lambda \in \mathbb{C} \tag{C.3}$$

Here the factor λ depends only on x and on the operators of A_1 that were discarded; we give a closed formula for λ below. For example, if $x = 1$ and $P_n = Z$, then $\lambda = -1$. In summary, $L(x, \xrightarrow{A}\textcircled{v})$ is the product of (1) the labels on the traversed edges and (2) a phase λ.

Moreover, the ℓ most significant operators of P influence which path is traversed in the following way. For a pauli string P, let $\chi(P) = \chi_1(P) \ldots \chi_\ell(P) \in \{0,1\}^\ell$ be the string defined by $\chi_j(P) = 0$ if $P_{n-j+1} \in \{I, Z\}$ and $\chi_j(P) = 1$ otherwise, i.e., if $P_{n-j+1} \in \{X, Y\}$. To be clear, $P^{(\ell)}$ isolates the $n - \ell$ *least significant* qubits, whereas $\chi(P)$ depends on the ℓ *most significant* qubits:

$$P = \underbrace{P_n \otimes \cdots \otimes P_{n-\ell+1}}_{\chi(P) \text{ depends on this part}} \otimes \overbrace{P_{n-\ell} \otimes \cdots \otimes P_1}^{P^{(\ell)} \text{ yields this part}} \tag{C.4}$$

Then we have

$$L(x, P) = \lambda L(x \oplus \chi(P), \mathbb{I}^{\otimes \ell} \otimes P^{(\ell)}) \tag{C.5}$$

where $\lambda = \langle x| P_n \otimes \cdots \otimes P_{n-\ell+1} |x \oplus \chi(P)\rangle$. Therefore,

$$\text{FOLLOW}(x, \xrightarrow{P}\textcircled{v}) = \xrightarrow{\lambda L(x \oplus \chi(P), \mathbb{I}^{\otimes \ell} \otimes P^{(\ell)})}\textcircled{t} \quad \text{for some } \lambda \in \{0, \pm 1, \pm i\} \tag{C.6}$$

where t is the destination of the path traversed by $\text{FOLLOW}(x, \xrightarrow{P}\textcircled{v})$. Lastly, we note that

$$L(x, \mathbb{I}^{\otimes \ell} \otimes P^{(\ell)}) = P^{(\ell)} \cdot L(x, \mathbb{I}^{\otimes n}) \tag{C.7}$$

We have thus reduced the problem of finding a closed-form expression for $L(x, \xrightarrow{\alpha P}\textcircled{v})$ to the problem of obtaining a closed-form expression for $L(x, \mathbb{I})$, to which we now turn. In the following, we let $v_0, \ldots, v_n$ be the nodes in the Tower Pauli-limdd, with v_0 the top node and v_n the Leaf node (we say that node v_ℓ is on layer ℓ). For a bit $a \in \{0,1\}$ and Pauli string P, we use the notation $P^a = P$ if $a = 1$ and $P^a = \mathbb{I}$ if $a = 0$. To avoid multiple superscripts, we write $P^{a,(\ell)} = (P^a)^{(\ell)}$ for a bit a and an integer ℓ.

Lemma C.1. Let v be the root node of an n-qubit Tower LIMDD and denote with A_j the label of the (unique) high edge from layer $j-1$ to layer j in this Tower LIMDD. Let $x \in \{0,1\}^\ell$. Then there are predicates $V_1, \ldots, V_\ell$ such that (1) for each $1 \le j \le \ell$, the predicate $V_j(x)$ can be expressed as the XOR of (a subset of) the variables $x_1, \ldots, x_j$; and (2) it holds that

$$L(x, \xrightarrow{\mathbb{I}}\textcircled{v}) = A_1^{V_1(x),(\ell)} \cdots A_\ell^{V_\ell(x),(\ell)} \tag{C.8}$$

Proof. We observed above (in Equation C.3) that $L(x, \mathbb{I})$ is the product of the $P^{(\ell)}$ for each label P encountered on the edges traversed by FOLLOW$(x, \mathbb{I})$. For a layer $1 \le j \le \ell$, let $V_j(x)$ be the predicate which is true iff the high edge from layer $j-1$ to j is traversed by FOLLOW$(x, \xrightarrow{\mathbb{I}}\textcircled{v})$. Recall that the low edges of a LIMDD are labeled with the identity operator $\mathbb{I}$. It follows that, in a Tower-LIMDD, $L(x, \mathbb{I}) = A_1^{V_1(x),(\ell)} \cdots A_\ell^{V_\ell(x),(\ell)}$, thus settling claim (1).

We now show that $V_j(x)$ can be expressed as the XOR of (a subset of) the variables $x_1, \ldots, x_j$, which proves the lemma. The proof is by induction on the layer index. The induction hypothesis in step j is that the predicates $V_1(x), \ldots, V_j(x)$ can each be written as a XOR over the variables $x_1, \ldots, x_j$.

Base case. For the base case, we observe that $V_1(x) = x_1$; namely, if $x_1 = 1$, then from layer 0 to layer 1, the path traverses the high edge; otherwise the low edge.

Induction step. Assume the induction hypothesis and consider V_{j+1}. We claim that

$$V_{j+1} = x_{j+1} \oplus (\chi_{j+1}(A_1) \wedge V_1) \oplus \cdots \oplus (\chi_{j+1}(A_j) \wedge V_j) \tag{C.9}$$

Namely, for each visited high edge with label A, the bit $\chi_{j+1}(A)$ "flips" the instruction for the path to traverse the high or low edge at layer $j+1$. Lastly, since the bits $\chi_{j+1}(A) \in \{0,1\}$ are constants defined by the LIMDD, and the expressions $V_1, \ldots, V_j$ are XORs over the variables $x_1, \ldots, x_j$, it follows that V_{j+1} is a XOR over the variables $x_1, \ldots, x_{j+1}$. $\square$

Lemma C.2. Let v be the root node of an n-qubit Tower PAULI-LIMDD. Let $x, y \in \{0,1\}^\ell$ for some $0 \le \ell \le n$. Then $L(x, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) \cdot L(y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v})^{-1} = \pm L(x \oplus y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v})$.

Proof. Let $V_1, \ldots, V_\ell$ be the predicates determining L as in Equation C.8. Then,

$$L(x, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) = A_1^{V_1(x),(\ell)} \cdots A_\ell^{V_\ell(x),(\ell)} \tag{C.10}$$

$$L(y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) = A_1^{V_1(y),(\ell)} \cdots A_\ell^{V_\ell(y),(\ell)} \tag{C.11}$$

$$\begin{aligned} L(x \oplus y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) &= A_1^{V_1(x\oplus y),(\ell)} \cdots A_\ell^{V_\ell(x\oplus y),(\ell)} && \text{(C.12)}\\ &= A_1^{V_1(x)\oplus V_1(y),(\ell)} \cdots A_\ell^{V_\ell(x)\oplus V_\ell(x),(\ell)} && \text{(C.13)}\\ &= \pm\, A_1^{V(x),(\ell)} A_1^{V_1(y),(\ell)} \cdots A_\ell^{V_\ell(y),(\ell)} A_\ell^{V_\ell(y),(\ell)} && \text{(C.14)} \end{aligned}$$

Here, in Equation C.13, we have used the fact that, since $V_j(x \oplus y)$ is simply a XOR over some of its inputs, we have

$$V_j(x \oplus y) = V_j(x) \oplus V_j(y) \tag{C.15}$$

In Equation C.14, we have used the fact that $A^{a\oplus b} = \pm A^a \cdot A^b$. Namely, we have $A = \lambda P$ for some $\lambda \in \{0, \pm 1, \pm i\}$; thus, if $a = b = 1$ and $\lambda = \pm i$ then $A^2 = -\mathbb{I}$ so $A^{a\oplus b} = \mathbb{I} = -A^a \cdot A^b$; otherwise, if $\lambda \in \{0, \pm 1\}$ or if $a = 0$ or $b = 0$ we have $A^{a\oplus b} = A^a \cdot A^b$. We now obtain $L(x, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) \cdot L(x \oplus y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) = L(y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v})$ by simple algebraic manipulation:

$$\begin{aligned} & L(x, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) \cdot L(x \oplus y, \overset{\mathbb{I}}{\text{—}}\!\textcircled{v}) && \text{(C.16)}\\ &= \pm\, \underbrace{A_1^{V_1(x),(\ell)} \cdots A_\ell^{V_\ell(x),(\ell)}}_{L(x,\mathbb{I})} \cdot \underbrace{A_1^{V_1(x),(\ell)} \cdot A_1^{V_1(y),(\ell)} \cdots A_\ell^{V_\ell(x),(\ell)} \cdot A_\ell^{V_\ell(y),(\ell)}}_{L(x\oplus y)} && \text{(C.17)}\\ &= \pm\, A_1^{V_1(x),(\ell)} \cdot A_1^{V_1(x),(\ell)} \cdot A_1^{V_1(y),(\ell)} \cdots A_\ell^{V_\ell(x),(\ell)} \cdot A_\ell^{V_\ell(x),(\ell)} \cdot A_\ell^{V_\ell(y),(\ell)} && \text{(C.18)}\\ &= \pm\, A_1^{V_1(x)\oplus V_1(x)\oplus V_1(y),(\ell)} \cdots A_\ell^{V_\ell(x)\oplus V_\ell(x)\oplus V_\ell(y),(\ell)} && \text{(C.19)}\\ &= \pm\, A_1^{V_1(y),(\ell)} \cdots A_\ell^{V_\ell(y),(\ell)} = \pm L(y, \mathbb{I}) && \text{(C.20)} \end{aligned}$$

We obtain Equation C.18 by grouping like terms; this "shuffling" is possible because

Pauli operators either commute or anticommute. The statement $L(x, \mathbb{I}) \cdot L(y, \mathbb{I})^{-1} = \pm L(x \oplus y, \mathbb{I})$ follows from Equation C.20. $\square$

Lemma C.3. Let $v = v_0$ be a node in a Tower Pauli-LIMDD representing a stabilizer state and let P, Q Pauli strings. Then a call to ADD($\xrightarrow{\alpha P}\textcircled{v}$, $\xrightarrow{\beta Q}\textcircled{v}$) invokes only at most $5n$ recursive calls to ADD.

Proof. We observed in Equation C.2 that when ADD is called with parameters ADD($\xrightarrow{\alpha P}\textcircled{v_0}$, $\xrightarrow{\beta Q}\textcircled{v_0}$), the parameters to the recursive calls are all of the form

$$\text{ADD}(\text{FOLLOW}(x, \xrightarrow{\alpha P}\textcircled{v_0}), \text{FOLLOW}(x, \xrightarrow{\beta Q}\textcircled{v_0})) \quad \text{for some } x \in \{0,1\}^\ell \text{ and } 0 \leq \ell \leq n \tag{C.21}$$

Using the insights above, we have, for any $x \in \{0,1\}^\ell$,

$$\text{FOLLOW}(x, \xrightarrow{\alpha P}\textcircled{v_0}) = \xrightarrow{\alpha\lambda L(x \oplus \chi(P), \mathbb{I}^{\otimes \ell} \otimes P^{(\ell)})}\textcircled{v_ℓ} = \xrightarrow{\alpha\lambda P^{(\ell)} L(x \oplus \chi(P), \mathbb{I})}\textcircled{v_ℓ} \tag{C.22}$$

$$\text{FOLLOW}(x, \xrightarrow{\beta Q}\textcircled{v_0}) = \xrightarrow{\beta\omega L(x \oplus \chi(Q), \mathbb{I}^{\otimes \ell} \otimes R^{(\ell)})}\textcircled{v_ℓ} = \xrightarrow{\beta\omega Q^{(\ell)} L(x \oplus \chi(Q), \mathbb{I})}\textcircled{v_ℓ} \tag{C.23}$$

with $\lambda, \omega \in \{0, \pm 1, \pm i\}$. Thus, ADD is called with parameters

$$\text{ADD}\left(\xrightarrow{\alpha\lambda P^{(\ell)} L(x \oplus \chi(P), \mathbb{I})}\textcircled{v_ℓ}, \xrightarrow{\beta\omega R^{(\ell)} L(x \oplus \chi(Q), \mathbb{I})}\textcircled{v_ℓ}\right) \quad \text{for some } \lambda, \omega \in \{0, \pm 1, \pm i\} \tag{C.24}$$

Therefore, this call to ADD can be associated with the following operator,

$$(\alpha\lambda P^{(\ell)} L(x \oplus \chi(P), \mathbb{I}))^{-1} \cdot (\omega Q^{(\ell)} L(x \oplus \chi(Q), \mathbb{I})) \tag{C.25}$$

$$= \alpha^{-1}\lambda^{-1}\beta\omega P^{(\ell)} Q^{(\ell)} L(x \oplus \chi(P), \mathbb{I}) L(x \oplus \chi(Q), \mathbb{I})^{-1} \tag{C.26}$$

$$= \theta P^{(\ell)} Q^{(\ell)} L(\chi(P) \oplus \chi(Q), \mathbb{I}) \tag{C.27}$$

for some $\theta \in \mathbb{C}$. In Equation C.27 we have used Lemma C.2 to obtain

$$L(x \oplus \chi(P), \mathbb{I})^{-1} \cdot L(x \oplus \chi(Q), \mathbb{I}) = \pm L(\chi(P) \oplus \chi(Q), \mathbb{I}) \tag{C.28}$$

Recall that in a Tower PAULI-LIMDD, all edge weights are in $\{0, \pm 1, \pm i\}$, so in particular we have $\theta \in \{0, \pm 1, \pm i\}$. We observe that this operator depends on the level ℓ and only the phase θ depends on x. That is to say, $P^{(\ell)}$, $Q^{(\ell)}$ and $L(\chi(P) \oplus \chi(Q), \mathbb{I})$ are fixed for a given level ℓ. It follows that each recursive call to ADD at some level ℓ

is associated with the same operator, modulo some phase $\theta \in \{0, \pm 1, \pm i\}$. Therefore, the cache only stores at most five distinct recursive calls, and will achieve a cache hit on all other recursive calls, at level ℓ. When a cache hit is achieved, the algorithm does not recurse further, and instead terminates the current call. Since the diagram contains n levels, there are at most $5n$ recursive calls in total. $\square$

Appendix D

LIMDDs prepare the W state efficiently

In this section, we prove Theorem 3.6. To this end, we show that LIMDDs can efficiently simulate a circuit family given by McClung [220], which prepares the $|W\rangle$ state when initialized to the $|0\rangle$ state. We thereby show a separation between LIMDD and the Clifford$+T$ simulator, as explained in Sec. 3.3.4.3. Figure Figure D.1 shows the circuit for the case of 8 qubits.

Theorem 3.6. There exists a circuit family C_n such that $C_n |0\rangle^{\otimes n} = |W_n\rangle$, that Pauli-LIMDDs can efficiently simulate. Here simulation means that it constructs representations of all intermediate states, in a way which allows one to, e.g., efficiently simulate any single-qubit computational-basis measurement or compute any computational basis amplitude on any intermediate state and the output state.

Proof. The proof outline is as follows. First, we establish that the LIMDD of each intermediate state (Lemma D.3), as well as of each gate (Lemma D.4), has polynomial size. Second, we establish that the algorithms presented in Sec. 3.3.3 can apply each gate to the intermediate state in polynomial time (Lemma D.8). To this end, we observe that the circuit only produces relatively simple intermediate states. Specifically, each intermediate and output state is of the form $|\psi_t\rangle = \frac{1}{\sqrt{n}} \sum_{k=1}^{n} |x_k\rangle$ where the $x_k \in \{0,1\}^n$ are computational basis vectors (Lemma D.2). For example, the output state has $|x_k\rangle = |0\rangle^{k-1} |1\rangle |0\rangle^{\otimes n-k}$. The main technical tool we will use to reason about the size of the LIMDDs of these intermediate states, are the *subfunction rank* and *computational basis rank* of a state. Both these measures are upper bounds of the size of a LIMDD (in Lemma D.1), and also allow us to upper bound the time taken by the ApplyGate and Add algorithms (in Lemma D.5 for ApplyGate and Lemma D.6 Add).

The theorem follows from Lemma D.8 and Corollary D.1. □

Figure D.1 shows the circuit for the case of $n = 8$ qubits. For convenience and without loss of generality, we only treat the case when the number of qubits is a power of 2, since the circuit is simplest in that case. In general, the circuit works as follows. The qubits are divided into two registers; register A, with $\log n$ qubits, and register B, with the remaining $n - \log n$ qubits. First, the circuit applies a Hadamard gate to each qubit in register A, to bring the state to the superposition $|+\rangle^{\otimes \log n} |0\rangle^{n-\log n}$. Then it applies $n - \log n$ Controlled-X gates, where, in each gate, each qubit of register A acts as the control qubits and one qubit in register B is the target qubit. Lastly, it applies $n - \log n$ Controlled-X gates, where, in each gate, one qubit in register B is the control qubit and one or more qubits in register A are the target qubits. Each of the three groups of gates is highlighted in a dashed rectangle in Figure D.1. On input $|0\rangle^{\otimes n}$, the circuit's final state is $|W_n\rangle$. We emphasize that the Controlled-X gates are

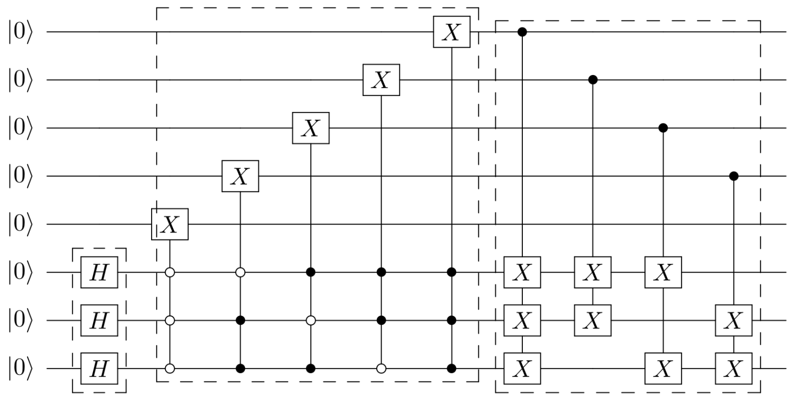

Figure D.1: Reproduced from McClung [220]. A circuit on eight qubits ($n = 8$) which takes as input the $|0\rangle^{\otimes 8}$ state and outputs the $|W_8\rangle$ state. In the general case, it contains $\log n$ Hadamard gates, and its Controlled-X gates act on one target qubit and at most $\log n$ control qubits.

permutation gates (i.e., their matrices are permutation matrices). Therefore, these gates do not influence the number of non-zero computational basis state amplitudes of the intermediate states. We refer to the t-th gate of this circuit as U_t, and the t-th intermediate state as $|\psi_t\rangle$, so that $|\psi_{t+1}\rangle = U_t\,|\psi_t\rangle$ and $|\psi_0\rangle = |0\rangle$ is the initial state.

We refer to the number of computational basis states with nonzero amplitude as a state's *computational basis rank*, denoted $\chi_{\text{comp}}(|\psi\rangle)$.

Definition D.1. (Computational basis rank) Let $|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha(x)\,|x\rangle$ be a quantum state defined by the amplitude function $\alpha\colon \{0,1\}^n \to \mathbb{C}$. Then the *computational basis rank* of $|\psi\rangle$ is $\chi_{\text{comp}}(|\psi\rangle) = |\,\{x \mid \alpha(x) \neq 0\}\,|$, the number of nonzero computational basis amplitudes.

Recall that, for a given function $\alpha\colon \{0,1\}^n \to \mathbb{C}$, a string $a \in \{0,1\}^\ell$ induces a *subfunction* $\alpha_y\colon \{0,1\}^{n-\ell} \to \mathbb{C}$, defined as $\alpha_y(x) = \alpha(y,x)$. We refer to the number of subfunctions of a state's amplitude function as its *subfunction rank*. The following definition makes this more precise.

Definition D.2. (Subfunction rank) Let $|\psi\rangle = \sum_{x\in\{0,1\}^n} \alpha^\psi(x)\,|x\rangle$ be a quantum state defined by the amplitude function $\alpha^\psi\colon \{0,1\}^n \to \mathbb{C}$, as above. Let $\chi_{\text{sub}}(|\psi\rangle,\ell)$ be the number of unique non-zero subfunctions induced by strings of length ℓ, as follows,

$$\chi_{\text{sub}}(|\psi\rangle,\ell) = |\,\{\alpha^\psi_y\colon \{0,1\}^{n-\ell} \to \mathbb{C} \mid \alpha_y \neq 0, y \in \{0,1\}^\ell\}\,| \tag{D.1}$$

We define the *subfunction rank* of $|\psi\rangle$ as $\chi_{\text{sub}}(|\psi\rangle) = \max_{\ell=0,\ldots n} \chi_{\text{sub}}(|\psi\rangle,\ell)$. We extend these definitions in the natural way for an n-qubit matrix $U = \sum_{r,c\in\{0,1\}^n} \alpha^U(r,c)\,|r\rangle\langle c|$ defined by the function $\alpha^U : \{0,1\}^{2n} \to \mathbb{C}$.

It is easy to check that $\chi_{\text{sub}}(|\psi\rangle) \leq \chi_{\text{comp}}(|\psi\rangle)$ holds for any state.

For the next lemma, we use the notion of a *prefix* of a LIMDD node. This lemma will serve as a tool which allows us to show that a LIMDD is small when its computational basis rank is low. We apply this tool to the intermediate states of the circuit in Lemma D.3.

Definition D.3 (Prefix of a LIMDD node)**.** For a given string $x \in \{0,1\}^\ell$, consider the path traversed by the FOLLOW(x, $\xrightarrow{R}\!\!\textcircled{r}$) subroutine, which starts at the diagram's root edge and ends at a node v on level ℓ. We will say that x is a *prefix* of the node v.

We let Labels(x) be the product of the LIMs on the edges of this path (i.e., including the root edge). The set of prefixes of a node v is denoted pre(v).

Lemma D.1. If a LIMDD represents the state $|\varphi\rangle$, then its width at any given level (i.e., the number of nodes at that level) is at most $\chi_{\text{comp}}(|\varphi\rangle)$.

Proof. For notational convenience, let us number the levels so that the root node is on level 0, its children are on level 1, and so on, with the Leaf on level n (contrary to Figure 3.3). Let r be the root node of the LIMDD, and R the root edge's label. By construction of a LIMDD, the state represented by the LIMDD can be expressed as follows, for any level $\ell \geq 0$,

$$R\,|r\rangle = \sum_{x\in\{0,1\}^{\ell}} |x\rangle \otimes \textsc{follow}(x, \xrightarrow{R}\textcircled{r}) \tag{D.2}$$

Since $\xrightarrow{R}\textcircled{r}$ is the root of our diagram, if x is a prefix of v, then

$$\textsc{follow}(x, \xrightarrow{R}\textcircled{r}) = \text{Labels}(x)\cdot|v\rangle \tag{D.3}$$

A string $x \in \{0,1\}^{\ell}$ can be a prefix of only one node; consequently, the prefix sets of two nodes on the same level are disjoint, i.e., $\text{pre}(v_p)\cap\text{pre}(v_q) = \emptyset$ for $p \neq q$. Moreover, each string x is a prefix of *some* node on level ℓ (namely, simply the node at which the $\textsc{follow}(x, \xrightarrow{R}\textcircled{r})$ subroutine arrives). Say that the ℓ-th level contains m nodes, $v_1,\ldots,v_m$. Therefore, the sets $\text{pre}(v_1),\ldots,\text{pre}(v_m)$ partition the set $\{0,1\}^{\ell}$. Therefore, by putting Equation D.3 and Equation D.2 together, we can express the root node's state in terms of the nodes $v_1,\ldots,v_m$ on level ℓ:

$$\begin{aligned} R\,|r\rangle &= \sum_{k=1}^{m}\sum_{x\in\text{pre}(v_k)} |x\rangle \otimes \textsc{follow}(x, \xrightarrow{R}\textcircled{r}) && \text{(D.4)}\\ &= \sum_{k=1}^{m}\sum_{x\in\text{pre}(v_k)} |x\rangle \otimes \text{Labels}(x)\cdot|v_k\rangle && \text{(D.5)}\end{aligned}$$

We now show that each term $\sum_{x\in\text{pre}(v_k)}|x\rangle\otimes\text{Labels}(x)\cdot|v_k\rangle$ contributes a non-zero vector. It then follows that the state has computational basis rank at least m, since these terms are vectors with pairwise disjoint support, since the sets pre(v_k) are pairwise disjoint. Specifically, we show that each node has at least one prefix x such that Labels$(x)\cdot|v\rangle$ is not the all-zero vector. In principle, this can fail in one of three ways: either v has no prefixes, or all prefixes $x \in \text{pre}(v_k)$ have Labels$(x) = 0$ because the path contains an edge labeled with the 0 LIM, or the node v represents the all-zero vector (i.e., $|v\rangle = \vec{0}$). First, we note that each node has at least one prefix, since each node is reachable from the root, as a LIMDD is a connected graph. Second, due to the zero edges rule (see Definition 3.5), for any node, at least one of its prefixes has only non-zero LIMs on the edges. Namely, each node v has at least one incoming edge labeled with a non-zero LIM, since, if it has an incoming edge from node w labeled with 0, then this must be the high edge of w and by the zero edges rule the low edge of w must also point to v and moreover must be labeled with $\mathbb{I}$ by the low factoring rule. Together, via a simple inductive argument, there must be at least one non-zero path from v to the root. Lastly, no node represents the all-zero vector, due to the low factoring rule (in Definition 3.5). Namely, if v is a node, then by the low factoring rule, the low edge has label $\mathbb{I}$. Therefore, if this edge points to node v_0, and the high edge is $\xrightarrow{A}\textcircled{v_1}$, then the node v represents $|v\rangle = |0\rangle|v_0\rangle + |1\rangle A|v_1\rangle$ with possibly $A = 0$, so, if $|v_0\rangle \neq \vec{0}$, then $|v\rangle \neq \vec{0}$. An argument by induction now shows that no node in the reduced LIMDD represents the all-zero vector.

Therefore, each node has at least one prefix x such that $\textsc{follow}(x, \xrightarrow{R}\textcircled{r}) \neq \vec{0}$. We conclude that the equation above contains at least m non-zero contributions. Hence $m \leq \chi_{\text{comp}}(R|r\rangle)$, at any level $0 \leq \ell \leq n$. □

Lemma D.2. Each intermediate state in the circuit in Figure D.1 (with $n = 2^c$) has $\chi_{\text{comp}}(|\psi\rangle) \leq n$.

Proof. The initial state is $|\psi_0\rangle = |0\rangle^{\otimes n}$, which is a computational basis state, so $\chi_{\text{comp}}(\psi_0) = 1$. The first $\log n$ gates are Hadamard gates, which produce the state

$$|\psi_{\log n}\rangle = H^{\otimes \log n} \otimes \mathbb{I}^{n-\log n}|0\rangle = |+\rangle^{\otimes \log n} \otimes |0\rangle^{\otimes n-\log n} = \frac{1}{\sqrt{n}}\sum_{x=0}^{n-1}|x\rangle|0\rangle^{\otimes n-\log n} \tag{D.6}$$

This is a superposition of n computational basis states, so we have $\chi_{\text{comp}}(|\psi_{\log n}\rangle) = n$. All subsequent gates are controlled-X gates; these gates permute the computational basis states, but they do not increase their number. □

Lemma D.3. The reduced LIMDD of each intermediate state in the circuit in Figure D.1 has polynomial size.

Proof. By Lemma D.1, the width of a LIMDD representing $|\varphi\rangle$ is at most $\chi_{\text{comp}}(|\varphi\rangle)$

at any level. Since there are n levels, the total size is at most $n\chi_{\text{comp}}(|\varphi\rangle)$. By Lemma D.2, the intermediate states in question have polynomial χ_{comp}, so the result follows. □

Lemma D.4. The LIMDD of each gate in the circuit in Figure D.1 (with $n = 2^c$) has polynomial size.

Proof. Each gate acts on at most $k = \log n + 1$ qubits. Therefore, the width of any level of the LIMDD is at most $4^k = 4n^2$. The height of the LIMDD is n by definition, so the LIMDD has at most $4n^3$ nodes. □

The ApplyGate procedure handles the Hadamard gates efficiently, since they apply a single-qubit gate to a product state. The difficult part is to show that the same holds for the controlled-X gates. To this end, we show a general result for the speed of LIMDD operations (Lemma D.5). Although this worst-case upper bound is tight, it is exponentially far removed from the best case, e.g., in the case of Clifford circuits, in which case the intermediate states can have exponential χ_{sub}, yet the LIMDD simulation is polynomial-time, as shown in Sec. 3.3.3.4.

Lemma D.5. The number of recursive calls made by subroutine ApplyGate$(U, |\psi\rangle)$, is at most $n\chi_{\text{sub}}(U)\chi_{\text{sub}}(|\psi\rangle)$, for any gate U and any state $|\psi\rangle$.

Proof. Inspecting Algorithm 8, we see that every call to ApplyGate$(U, |\psi\rangle)$ produces four new recursive calls, namely ApplyGate(follow(rc, U), follow$(c, |\psi\rangle)$) for $r, c \in \{0, 1\}$. Therefore, the set of parameters in all recursive calls of ApplyGate$(U, |\psi\rangle)$ is precisely the set of tuples (follow(rc, U), follow$(c, |\psi\rangle)$), with $r, c \in \{0, 1\}^\ell$ with $\ell = 0 \ldots n$. The terms follow(rc, U) and follow$(c, |\psi\rangle)$ are precisely the subfunctions of U and $|\psi\rangle$, and since there are at most $\chi_{\text{sub}}(U)$ and $\chi_{\text{sub}}(|\psi\rangle)$ of these, the total number of distinct parameters passed to ApplyGate in recursive calls at level ℓ, is at most $\chi_{\text{sub}}(U, \ell) \cdot \chi_{\text{sub}}(|\psi\rangle, \ell) \le \chi_{\text{sub}}(U) \cdot \chi_{\text{sub}}(|\psi\rangle)$. Summing over the n levels of the diagram, we see that there are at most $n\chi_{\text{sub}}(U)\chi_{\text{sub}}(|\psi\rangle)$ distinct recursive calls in total. As detailed in Sec. 3.3.3.3, the ApplyGate algorithm caches its inputs in such a way that it will achieve a cache hit on a call ApplyGate$(U', |\psi'\rangle)$ when it has previously been called with parameters $U, |\psi\rangle$ such that $U = U'$ and $|\psi\rangle = |\psi'\rangle$. Therefore, the total number of recursive calls that is made, is equal to the number of *distinct* calls, and the result follows. □

In our case, both $\chi_{\text{sub}}(U)$ and $\chi_{\text{sub}}(|\psi\rangle)$ are polynomial, so a polynomial number of recursive calls to ApplyGate is made. We now show that also the Add subroutine makes only a small number of recursive calls every time it is called from ApplyGate. First, Lemma D.6 shows expresses a worst-case upper bound on the number of recursive calls to Add in terms of χ_{sub}. Then Lemma D.7 uses this result to show that, in our circuit, the number of recursive calls is polynomial in n.

Lemma D.6. The number of recursive calls made by the subroutine $\text{Add}(|\alpha\rangle, |\beta\rangle)$ is at most $n\chi_{sub}(|\alpha\rangle) \cdot \chi_{sub}(|\beta\rangle)$, if $|\alpha\rangle, |\beta\rangle$ are n-qubit states.

Proof. Inspecting Algorithm 9, every call to $\text{Add}(|\alpha\rangle, |\beta\rangle)$ produces two new recursive calls, namely $\text{Add}(\text{follow}(0, |\alpha\rangle), \text{follow}(0, |\beta\rangle))$ and $\text{Add}(\text{follow}(1, |\alpha\rangle), \text{follow}(1, |\beta\rangle))$. It follows that the set of parameters on $n - \ell$ qubits with which Add is called is the set of tuples $(\text{follow}(x, |\alpha\rangle), \text{follow}(x, |\beta\rangle))$, for $x \in \{0,1\}^\ell$. This corresponds precisely to the set of subfunctions of α and β induced by length-ℓ strings, of which there are $\chi_{\text{sub}}(|\alpha\rangle, \ell)$ and $\chi_{\text{sub}}(|\beta\rangle, \ell)$, respectively. Because the results of previous computations are cached, as explained in Sec. 3.3.3.3, the total number of recursive calls is the number of *distinct* recursive calls. Therefore, we get the upper bound of $\chi_{\text{sub}}(|\alpha\rangle) \cdot \chi_{\text{sub}}(|\beta\rangle)$ for each level of the LIMDD. Since the LIMDD has n levels, the upper bound $n\chi_{\text{sub}}(|\alpha\rangle) \cdot \chi_{\text{sub}}(|\beta\rangle)$ follows. □

Lemma D.7. The calls to $\text{Add}(|\alpha\rangle, |\beta\rangle)$ that are made by the recursive calls to $\text{ApplyGate}(U_t, |\psi_t\rangle)$, satisfy $\chi_{\text{sub}}(|\alpha\rangle), \chi_{\text{sub}}(|\beta\rangle) = \text{poly}(n)$.

Proof. We have established that the recursive calls to ApplyGate are all called with parameters of the form $\text{ApplyGate}(\text{follow}(r, c, U_t), \text{follow}(c, |\psi_t\rangle))$ for some $r, c \in \{0,1\}^\ell$. Inspecting Algorithm 8, we see that, within such a call, each call to $\text{Add}(|\alpha\rangle, |\beta\rangle)$ has parameters which are both of the form $|\alpha\rangle, |\beta\rangle = \text{ApplyGate}(\text{follow}(rx, cy, U_t), \text{follow}(cy, |\psi_t\rangle))$ for some $x, y \in \{0,1\}$; therefore, the parameters $|\alpha\rangle, |\beta\rangle$ are of the form $|\alpha\rangle, |\beta\rangle = \text{follow}(r, c, U_t) \cdot \text{follow}(r, |\psi_t\rangle)$. Here $\text{follow}(cy, |\psi_t\rangle)$ is a quantum state on $n - (\ell + 1)$ qubits.

The computational basis rank of a state is clearly non-increasing under taking subfunctions; that is, for any string x, it holds that, $\chi_{\text{comp}}(\text{follow}(x, |\psi\rangle)) \le \chi_{\text{comp}}(|\psi\rangle)$. In particular, we have $\chi_{\text{comp}}(\text{follow}(cy, |\psi_t\rangle)) \le \chi_{\text{comp}}(|\psi_t\rangle) = \mathcal{O}(n)$. The matrix $\text{follow}(rx, cy, U_t)$ is a subfunction of a permutation gate, and applying such a matrix to a vector cannot increase its computational basis rank, so we have

$$\chi_{\text{sub}}(|\alpha\rangle) = \chi_{\text{sub}}(\text{FOLLOW}(rx, cy, U_t) \cdot \text{FOLLOW}(cy, |\psi_t\rangle)) \tag{D.7}$$

$$\leq \chi_{\text{comp}}(\text{FOLLOW}(rx, cy, U_t) \cdot \text{FOLLOW}(cy, |\psi_t\rangle)) \leq \chi_{\text{comp}}(\text{FOLLOW}(cy, |\psi_t\rangle)) \tag{D.8}$$

$$\leq \chi_{\text{comp}}(|\psi_t\rangle) = \mathcal{O}(n) \tag{D.9}$$

This proves the lemma. □

Lemma D.8. Each call to ApplyGate$(U_t, |\psi_t\rangle)$ runs in polynomial time, for any gate U_t in the circuit in Figure D.1 (with $n = 2^c$).

Proof. If U_t is a Hadamard gate, then LIMDDs can apply this in polynomial time by Theorem 3.5, since $|\psi_t\rangle$ is a stabilizer state. Otherwise, U_t is one of the controlled-X gates. In this case there are a polynomial number of recursive calls to ApplyGate, by Lemma D.5. Each recursive call to ApplyGate makes two calls to Add$(|\alpha\rangle, |\beta\rangle)$, where both α and β are states with polynomial subfunction rank, by Lemma D.7. By Lemma D.6, these calls to Add all complete in time polynomial in the subfunction rank of its arguments. □

Corollary D.1. The circuit in Figure D.1 (with $n = 2^c$) can be simulated by LIMDDs in polynomial time.

Appendix E

Proofs of Section 5.3

In this appendix, we prove Theorem 5.1 as reproduced below. We also reproduce Figure 5.2 in Figure E.1, which additionally includes references to the respective lemmas. Chapter 2 and Section 5.2 contain relevant preliminaries on quantum information and QDDs.

Theorem 5.1. The succinctness results in Figure 5.2 hold.

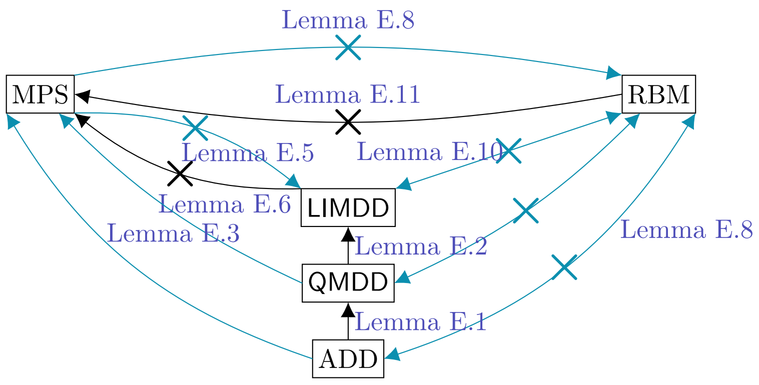

Figure E.1: Succinctness relations between various classical data structures for representing quantum states. Solid arrows $A \to B$ denote $B \prec_s A$, i.e., B is strictly more succinct than A. Crossed arrows $A \nleftrightarrow B$ denote a separation $B \not\preceq_s A$; a bidirectional crossed arrow implies incomparability. Blue arrows indicate novel relations that we identified.

Proof. The proofs for individual relations are stated in the lemmas referenced by Figure E.1.

Note that we do not include a proof for every arrow (direction), since several can be derived through transitivity properties. All unlabeled edge (directions) can be derived as follows:

- MPS $\prec_s$ ADD follows from MPS $\prec_s$ QMDD and QMDD $\prec_s$ ADD
- LIMDD $\prec_s$ ADD follows from LIMDD $\prec_s$ QMDD and QMDD $\prec_s$ ADD
- QMDD $\not\preceq_s$ RBM follows from LIMDD $\not\preceq_s$ RBM and LIMDD $\prec_s$ QMDD
- ADD $\not\preceq_s$ RBM follows from LIMDD $\not\preceq_s$ RBM and LIMDD $\prec_s$ ADD
- RBM $\not\preceq_s$ MPS follows from RBM $\not\preceq_s$ ADD and MPS $\prec_s$ ADD
- RBM $\not\preceq_s$ QMDD follows from RBM $\not\preceq_s$ ADD and QMDD $\prec_s$ ADD
- RBM $\not\preceq_s$ LIMDD follows from RBM $\not\preceq_s$ ADD and LIMDD $\prec_s$ ADD

This completes the proof of all stated succinctness relations. □

Lemma E.1. QMDD is exponentially more succinct than ADD.

Proof. Since ADD is a special case of QMDD (Sec. 5.2.2), QMDD is at least as succinct.

Fargier et al. [114] prove an exponential separation in Prop. 10. The proposition itself only mentions a superpolynomial separation; the fact that the separation is in fact exponential is contained in the proof. □

Lemma E.2. LIMDD is exponentially more succinct than QMDD.

Proof. Since QMDD is a special case of LIMDD (Sec. 5.2.2), LIMDD is at least as succinct.

Vinkhuijzen et al. [337] show an exponential separation for so-called 'cluster states.' □

Lemma E.3. MPS is exponentially more succinct than QMDD.

Proof. We show in Section G.4 that MPS is at least as succinct as QMDD, by showing that every QMDD can be translated to MPS in linear time.

We provide a state $|\varphi\rangle$ on n qubits, which has an exponential-sized QMDD, but a polynomial-sized MPS. Let $(x)_2 \in \mathbb{Z}$ be the integer represented by a bit-string $x \in \{0,1\}^n$. The state of interest is

$$|\varphi\rangle = \sum_{x\in\{0,1\}^n} (x)_2 |x\rangle = \sum_{x\in\{0,1\}^n} \left(\sum_{j=1}^{n} 2^{j-1}x_j\right) |x\rangle \tag{E.1}$$

Fargier et al. [114] show that this state has exponential-sized QMDD (Prop. 10). On the other hand, it can be efficiently represented by the following MPS of bond dimension 2:

$$A_n^0 = \begin{bmatrix} 1 & 0 \end{bmatrix} \qquad A_j^0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad A_1^0 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{E.2}$$

$$A_n^1 = \begin{bmatrix} 1 & 2^{n-1} \end{bmatrix} \qquad A_j^1 = \begin{bmatrix} 1 & 2^{j-1} \\ 0 & 1 \end{bmatrix} \qquad A_1^1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{E.3}$$

Here j ranges from $2 \dots n-1$. To show this, we can write

$$A_n^{x_n} = \begin{bmatrix} 1 & x_n \cdot 2^{n-1} \end{bmatrix} \qquad A_j^{x_j} = \begin{bmatrix} 1 & x_j \cdot 2^{j-1} \\ 0 & 1 \end{bmatrix} \text{ for } j = 2, \dots, n-1 \qquad A_1^{x_1} = \begin{bmatrix} x_1 \\ 1 \end{bmatrix}$$

Hence we can write

$$A_n^{x_n} \cdot \dots \cdot A_1^{x_1} = \begin{bmatrix} 1 & x_n \cdot 2^{n-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & \sum_{j=2}^{n-1} x_j \cdot 2^{j-1} \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ 1 \end{bmatrix} \tag{E.4}$$

$$= \begin{bmatrix} 1 & x_n \cdot 2^{n-1} \end{bmatrix} \cdot \begin{bmatrix} \sum_{j=1}^{n-1} x_j \cdot 2^{j-1} \\ 1 \end{bmatrix} \tag{E.5}$$

$$= \begin{bmatrix} \sum_{j=1}^{n} 2^{j-1} \cdot x_j \end{bmatrix} \tag{E.6}$$

□

The following quantum state, called $|\text{Sum}\rangle$, will feature in several of the below proofs. Specifically, we will show that RBM and MPS can represent this state efficiently, whereas LIMDDs cannot. A similar state will be used to show that LIMDD does not support the Swap operation. We omit normalization factors, as all data structures are oblivious to them.

$$|\text{Sum}\rangle = |+\rangle^{\otimes n} + \bigotimes_{j=1}^{n}(|0\rangle + e^{i\pi 2^{-j-1}}|1\rangle) \tag{E.7}$$

Lemma E.4. The LIMDD of $|\text{Sum}\rangle$ has size $2^{\Omega(n)}$ for every variable order.

Proof. We compute that the amplitude function for $|\text{Sum}\rangle$ is

$$f(\vec{x}) = 1 + e^{i\pi \sum_{j=1}^{n} x_j \cdot 2^{-j-1}}. \tag{E.8}$$

We note that f is injective and never zero, and indeed that the function $(\vec{x}, \vec{y}) \mapsto \frac{f(\vec{x})}{f(\vec{y})}$ is injective on the domain where $\vec{x} \neq \vec{y}$.

We now study the nodes v at level 1 (with $\mathsf{idx}(v) = 1$) via the subfunctions they represent, considering all variable orders. These nodes represent subfunctions on one variable. So we take out one variable $x_k \in \vec{x} = \{x_1, ..., x_n\}$. Without loss of generality, we may pick x_1 because the summation in Equation E.8 is commutative. For each assignment $\vec{a} \in \{0,1\}^{n-1}$, we obtain the function:

$$f_{\vec{a}}(x_1) = 1 + e^{i\pi \sum_{j=2}^{n} a_j \cdot 2^{-j-1}} \cdot e^{i\pi \cdot 1/4 x_1}.$$

We now show that for any $\vec{a} \neq \vec{c} \in \{0,1\}^{n-1}$ there is no $Q \in \text{PAULILIM}_1$ such that $f_{\vec{a}} = Q f_{\vec{c}}$.

Let $Q = \alpha P$ for $\alpha \in \mathbb{C} \setminus \{0\}, P \in \{\mathbb{I}, X, Y, Z\}$, so $f_{\vec{a}} = \alpha P f_{\vec{c}}$. Furthermore, define $\alpha = \alpha(z, x, \vec{a}, x_1) = (-1)^z \cdot \frac{f_{\vec{a}}(x_1 \oplus x = 0)}{f_{\vec{a}}(x_1 \oplus x = 1)}$ for $P = X^x \cdot Z^z$ with $x, z \in \{0,1\}$, absorbing the factor i of Y and -1 of ZX in α. The function α is injective, i.e., $\alpha(s) = \alpha(t)$ implies $s = t$, based on our earlier observations about f.

It follows that each subfunction $f_{\vec{a}}$ requires a separate node at level 1. So there are $\Omega(2^{n-1})$ nodes. □

Lemma E.5. There is a family of quantum states with polynomial-size MPS but exponential-size LIMDD.

Proof. MPS require only bond dimension 2 to represent the state $|\text{Sum}\rangle$ as shown by Lemma E.3. However, Lemma E.4 shows that LIMDDs require exponential size to represent the same state. □

Lemma E.6. There is a family of quantum states with polynomial-size LIMDD but exponential-size MPS.

Proof. LIMDD can efficiently represent any stabilizer state, but some stabilizer states require exponential-size MPS (in particular, the cluster state, among others [337]. □

Lemma E.7. MPS is at least as succinct as QMDD.

Proof. Section G.4 provides a polynomial-time transformation from QMDD to MPS. □

For proving the separation between RBM and ADD, we use the seminal Boolean function $IP : \{0,1\}^n \to \{0,1\}, \vec{x} \mapsto \sum_{k=1}^{n/2} x_k x_{k+n/2} \mod 2$ for even n, which computes the inner product between the first half of the input with the second half. Martens et al. [214] show that any RBM requires a number of hidden weights m which is necessarily exponential in m.

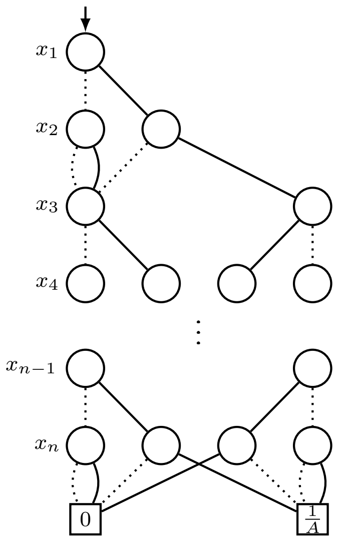

Figure E.2: An ADD for the inner product function IP' from Lemma E.8 made up of stacked blocks, each consisting of a layer of 2 nodes and a layer of 4 nodes. A in the right leaf is the normalization constant from Lemma E.8.

Lemma E.8. There is a quantum state that has linear representation both as ADD, and QMDD, and LIMDD, and MPS, but requires exponential space when represented as RBM under any qubit order.

Proof. We will give the proof for the ADD; the result will then follow for QMDD, LIMDD and MPS, since these are at least as succinct as ADD.

Since we consider the representation size under any qubit variable order, we may as well interleave the order. That is, we consider IP' which equals IP with x_{k+1}

and $x_{k+n/2}$ swapped, i.e. $IP'(x) = x_1x_2 + x_3x_4 + ... + x_{n-1}x_n$. Consider the n-qubit quantum state $|\varphi\rangle$ where $\langle x|\varphi\rangle = IP'(x)/A$ for $x \in \{0,1\}$, where the normalization factor is $A = \sqrt{\sum_{x\in\{0,1\}} IP'(x)}$. Martens et al. [214] show that any RBM requires a number of hidden weights m which is necessarily exponential in m.

There exists an ADD which represents $|\varphi\rangle$ in $O(n)$ space. This ADD is constructed from stacked blocks of two layers (of 2 and 4 nodes, respectively). The $(k+1)/2$-th block (counting from 1 from the top) for odd $k = 1, 3, 5, ...$ corresponds to computing the value $x_k \cdot x_{k+1}$ and adding it to the running value of $IP'(x_1, x_2, ..., x_{k-1})$. See Figure E.2. □

Lemma E.9. RBM can represent the state $|\text{Sum}\rangle$ with a single hidden node.

Proof. All nodes have bias 0, i.e., $\beta = [0]$ and $\alpha = [0, ..., 0]^T$ (a length-n vector). The weight on the edge between the hidden node and the j-th visible node is $e^{i\pi 2^{-j-1}}$. Then the RBM is defined by the multiplicative term of this hidden node, yielding

$$\psi(\vec{x}) = 1 + e^{w\cdot\vec{x}} = 1 + \prod_{j=1}^{n} e^{x_j i\pi 2^{-j-1}} \tag{E.9}$$

This corresponds exactly with the sum state: $|\psi\rangle = |\text{Sum}\rangle$. □

Lemma E.10. There is a state with a RBM of size $\mathcal{O}(n)$ but which requires LIMDD of size $2^{\Theta(n)}$, for every variable order.

Proof. RBM can represent the state $|\text{Sum}\rangle$, by Lemma E.9. However, Lemma E.4 shows that LIMDDs require exponential size to represent this state. □

Lemma E.11. There is a family of states with polynomial-size RBM but exponential-size MPS.

Proof. RBM can efficiently represent stabilizer states, as shown by Zhang et al. [365]. Vinkhuijzen et al. [337] show that some stabilizer states require exponential-size MPS (in particular, the cluster state, among others). □

Appendix F

Proofs of Section 5.4

In this appendix, we prove Theorem 5.2 and Theorem 5.3 from Section 5.4.

Theorem 5.2 is restated below. The proofs are organized per row of the table, so there is one section for each data structure. Section 5.2 and Chapter 2 contain relevant preliminaries on quantum information and QDDs.

Theorem 5.2. The tractability results in Table 5.2 hold.

We restate the other main result Theorem 5.3 here and provide a proof.

Theorem 5.3. Assuming the exponential time hypothesis, the fidelity of two states represented as LIMDDs or RBMs cannot be computed in polynomial time. The proof uses a reduction from the #EVEN SUBGRAPHS problem [169].

Proof. Lemma F.19 proves that LIMDD does not admit a polynomial time algorithm unless the exponential time hypothesis fails. Corollary F.1 concludes the same for RBM. □

F.1 Easy and hard operations for ADD

As noted in Sec. 5.2.2, the decision diagrams are special cases of each other. In particular, ADD specializes QMDD, which specializes LIMDD. From this, it immediately follows that LIMDD $\preceq_s$ QMDD $\preceq_s$ ADD. We also use this fact in the below proofs.

Easy and hard operations for ADD

Lemma F.1. ADD supports **Sample** and **Measure**.

Proof. LIMDD supports these operations (see Lemma F.15). Since ADD specializes LIMDD, it inherits the tractability of these operations □

Lemma F.2. ADD supports inner-product $\langle\varphi|\psi\rangle$.

Proof. QMDD supports these operations (see Lemma F.6). Since ADD is a specialization of QMDD, it inherits the tractability of these operations. □

Lemma F.3. ADD supports **Addition** and **Equal**.

Proof. See Fargier et al. [113] Table 1 (EQ) and Table 2 (+**BC**). □

Lemma F.4. ADD supports **Local**, and hence also **Hadamard**, **X,Y,Z**, T, **Swap** and **CZ**.

Proof. Suppose U is a local gate on k qubits. Then U can be expressed as the sum of 4^k terms, $U = \sum_{x,y\in\{0,1\}^k} a_{xy} |x\rangle\langle y| \otimes \mathbb{I}_{n-k}$. Each of these terms individually can be applied to an ADD in polynomial time ([113] Table 1 **CD**), since they are projections, followed by X gates. Since a constant number of states can be added in polynomial

Table 5.2: Tractability of queries and manipulations on the data structures analyzed in this chapter (single application of the operation). A ✓ means the data structure supports the operation in polytime, a ✓' means supported in randomized polytime, and ✖ means the data structure does not support the operation in polytime. A ○ means the operation is not supported in polytime unless $P = NP$. ? means unknown. The table only considers deterministic algorithms (for some ? a probabilistic algorithm exists, e.g., for **InnerProd** on RBM). Novel results are blue and underlined.

		Queries					Manipulation operations						
		Sample	**Measure**	**Equal**	**InnerProd**	**Fidelity**	**Addition**	**Hadamard**	**X,Y,Z**	**CZ**	**Swap**	**Local**	**T**-gate
Vector		✓'	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ADD	Section F.1	✓'	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
QMDD	Section F.2	✓'	✓	✓	✓	✓	✖	✖	✓	✓	✖	✖	✓
LIMDD	Section F.3	✓'	✓	✓	○	○	✖	✖	✓	✓	✖	✖	✓
MPS	Section F.4	✓'	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBM	Section F.5	✓'	?	?	○	○	?	?	✓	✓	✓	?	✓

time in ADDs (Lemma F.3), the result can be computed in polynomial time. Since ADD supports arbitrary k-local gates, in particular it supports all other gates that are mentioned: H, X, Y, Z, T, Swap CZ. □

F.2 Easy and hard operations for QMDD

Lemma F.5. QMDD supports **Equal**, **Sample** and **Measure**.

Proof. LIMDD supports these operations (see Lemma F.15). Since QMDD specializes LIMDD, it inherits the tractability of these operations. □

Lemma F.6. QMDD supports inner product (**InnerProd**) and fidelity (**Fidelity**).

Proof. We show in Section G.4 that a QMDD can be efficiently and exactly translated to an MPS. Since MPS supports inner product and fidelity, the result follows. □

Lemma F.7. QMDD does not support **Addition** in polynomial time.

Proof. Fargier et al. [113] (Thm. 4.9) show that **Addition** is hard for QMDD. □

Lemma F.8. QMDD does not support **Hadamard** in polynomial time and hence neither **Local**.

Proof. By reduction from addition: Take a QMDD root node v with left child a and right child b, then **Hadamard**$(v) = H\left|v\right\rangle$ is a new node with a left child $\left|a\right\rangle + \left|b\right\rangle$. By choosing $\left|a\right\rangle, \left|b\right\rangle$ to be the states from Fargier et al.'s proof showing that addition is intractable for QMDDs, the state $\left|a\right\rangle + \left|b\right\rangle$ requires an exponential-size QMDD. Since QMDD does not support the Hadamard gate, neither does it support arbitrary local gates (**Local**). □

Lemma F.9. QMDD supports Pauli gates **X,Y,Z** and **T** in polynomial time.

Proof. We will show that we can apply any single-qubit diagonal or anti-diagonal operator $A = \left[\begin{smallmatrix}\alpha & 0\\ 0 & \beta\end{smallmatrix}\right]$ to an QMDD in polynomial time. The result then immediately follows for the special cases of the gates X, Y, Z and T. Applying any diagonal local operator $A = \left[\begin{smallmatrix}\alpha & 0\\ 0 & \beta\end{smallmatrix}\right]$ to the top qubit is easy: simply multiply the weights of the low and high edges of the diagram's root node with respectively α and β. For the anti-diagonal operator A^T, we also swap low with high edges. To apply the local operator on any qubit, simply do the above for all nodes on the corresponding level.

To see that the resulting QMDD indeed represents the state $A \cdot |\psi\rangle$ (or $A^T |\psi\rangle$), consider the amplitude of any basis state $x \in \{0,1\}^n$. The amplitude of $|x\rangle$ in an QMDD is the product of the labels found on the edges while traversing the diagram from root to leaf. In the new diagram, only the weights have changed, whereas the topology has remained the same. If $x_k = 0$ (resp. $x_k = 1$), then, the k-th edge encountered during this traversal is the same in the new diagram as in the old diagram, but the label has been multiplied by α. Otherwise, if $x_k = 1$ (resp. $x_k = 0$), then the label is multiplied by β. All the other weights remain the same. Therefore, the amplitude of x in the new diagram is equal to the old amplitude multiplied by α (resp. β). $\square$

Lemma F.10. QMDD supports controlled-Z in polynomial time.

Proof. Algorithm 20 applies a controlled-Z gate to a QMDD in time linear in the number of nodes in the QMDD. To show that this is the runtime, we consider the number of times the algorithms ApplyControlledZ and ApplyZ are called.

For both these algorithms, say that a call is *trivial* if the result is already in the cache, otherwise a call is *non-trivial*. Then a trivial call completes in constant time (i.e., in time $\mathcal{O}(1)$). Moreover, the number of trivial calls is at most twice the number of non-trivial calls. Therefore, for the purposes of obtaining an asymptotic upper bound on the running time, it suffices to count the number of non-trivial calls to the algorithm.

Thanks to the cache, a given setting of the input parameters (v, a, b) (or (v, t) in the case of ApplyZ) will trigger only one non-trivial call. Therefore, the number of non-trivial calls is equal to the number of distinct input parameters. But here only v varies, so the number of non-trivial calls is at most the number of nodes in the QMDD. This reasoning holds for both algorithms ApplyControlledZ and ApplyZ. Therefore, both subroutines run in time $\mathcal{O}(m)$, for an QMDD which contains m nodes.

The correctness of this algorithm follows from the fact that $CZ_{a,b}|v\rangle = \lambda_0 |v_0\rangle + \lambda_1 Z_b |v_1\rangle$ where node v is represents an $a - qubit$ state $\begin{tikzcd} v_0 & v \arrow[l, dotted, "\lambda_0"'] \arrow[r, "\lambda_1"] & v_1 \end{tikzcd}$ and Z_b means applying the Z gate to the b-th qubit. This behavior is implemented by Line 4. By linearity, the algorithm is correct for nodes representing k-qubit states with $k > a$. This is implemented by Line 5. $\square$

Algorithm 20 Applies a controlled-Z gate to a QMDD node v, with control qubit a and target qubit b. More specifically, given a QMDD node v, representing the state $|v\rangle$, this subroutine returns a QMDD edge e representing $|e\rangle = CZ_a^b |v\rangle$. We assume wlog that $a > b$, since $CZ_a^b = CZ_b^a$. Here idx denotes the index of the qubit of v, and CACHE denotes a hashmap which maps triples to QMDD nodes. The subroutine APPLYZ applies a Z gate to a given target qubit t.

```
procedure APPLYCONTROLLEDZ(QMDD node $v$, qubit indices $a, b$)
    Say that node $v$ is $\textcircled{v_0} \overset{\lambda_0}{\cdots\cdots} \textcircled{v} \overset{\lambda_1}{\text{——}} \textcircled{v_1}$
    if the CACHE contains the tuple $(v, a, b)$ then return CACHE$[v, a, b]$
    else if idx$(v) = a$ then $r :=$ MAKENODE( $\xrightarrow{\lambda_0} \textcircled{v_0}$, $\lambda_1 \cdot$ APPLYZ$(v_1, b)$)
    else $r :=$ MAKENODE($\lambda_0 \cdot$ APPLYCZ$(v_0, a, b), \lambda_1 \cdot$ APPLYCZ$(v_1, a, b)$)
    CACHE$[v, a, b] := r$
    return $r$
procedure APPLYZ(QMDD node $v$, qubit index $t$)
    Say that node $v$ is $\textcircled{v_0} \overset{\lambda_0}{\cdots\cdots} \textcircled{v} \overset{\lambda_1}{\text{——}} \textcircled{v_1}$
    if the CACHE contains the tuple $(v, t)$ then return Z-CACHE$[v, t]$
    else if idx$(v) = t$ then $r :=$ MAKENODE( $\xrightarrow{\lambda_0} \textcircled{v_0}$, $\xrightarrow{-\lambda_1} \textcircled{v_1}$)
    else $r :=$ MAKENODE($\lambda_0 \cdot$ APPLYZ$(v_0, t), \lambda_1 \cdot$ APPLYZ$(v_1, t)$)
    Z-CACHE$[v, t] := r$
    return $r$
```

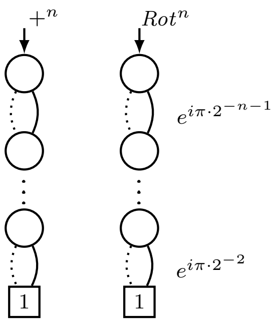

Figure F.1: The states $|+^n\rangle$ and $|Rot^n\rangle$ state as QMDD.

To prove that a single swap operation can explode the LIMDD or QMDD, we first provide two lemmas.

Lemma F.11. For $n \geq 1$, let $|Rot^n\rangle = \bigotimes_{j=1}^{n}\left(|0\rangle + e^{i\pi 2^{-j-1}}|1\rangle\right)$ and $|+^n\rangle = |+\rangle^{\otimes n}$.Then the states $|Rot^n\rangle$ and $|+^n\rangle$ have a linear-size QMDD.

Proof. Figure F.1 provides the QMDD representing both states. □

Lemma F.12. The following state has large LIMDD, for any variable order in which the qubit in register B comes after the qubits in register A.

$$|Sum'\rangle = |+\rangle_A^{\otimes n}|0\rangle_B + \bigotimes_{j=1}^{n}(|0\rangle_A + e^{i\pi 2^{-j-1}}|1\rangle_A) \otimes |1\rangle_B \tag{F.1}$$

Proof. The proof is similar to that of Lemma E.4 except that we reason about level 2. □

Lemma F.13. QMDD does not support **Swap** in polynomial time.

Proof. Let $|+^n\rangle$ and $|Rot^n\rangle$ be the states from Lemma F.11, and define the following state $|\rho\rangle$ on $n+2$ qubits,

$$|\rho\rangle = |0\rangle|+^n\rangle|0\rangle + |1\rangle|Rot^n\rangle|0\rangle \tag{F.2}$$

Then $|\rho\rangle$ has a small QMDD, of only size $\mathcal{O}(n)$. When we swap the first and last qubits, we obtain a state that includes $|Sum'\rangle$ from Lemma F.12:

$$\text{Swap}_1^n \cdot |\rho\rangle = |0\rangle \otimes (|+^n\rangle|0\rangle + |Rot^n\rangle|1\rangle) \tag{F.3}$$

The QMDD of $\text{Swap}_1^n \cdot |\rho\rangle$ is at least as large as that of $|\text{Sum}\rangle$: First, Wegener [344], Th. 2.4.1, shows that constraining can never increase the DD size, so we can discard the $|0\rangle\otimes$ part (regardless of variable order), as $\text{Swap}_1^n \cdot |\rho\rangle$ is at least as large as $|Sum'\rangle$. Then Lemma F.12 shows that this LIMDD has size at least $2^{\Omega(n)}$ for any variable order. Since, LIMDD is at least as succinct as QMDD (see Figure 5.2), this also holds for QMDD. □

Lemma F.14. QMDD does not support **Local** in polynomial time.

Proof. This is implied by Lemma F.13, since **Swap** is a 2-local gate (namely, it involves 2 qubits). □

F.3 Easy and hard operations for LIMDD

Lemma F.15. LIMDD supports **Sample**, **Measure**, **Equal** and **X,Y,Z**.

Proof. Vinkhuijzen et al. [337] show that LIMDD supports **Sample**, **Measure**, **Equal** and **X,Y,Z**. □

Here we show that LIMDD also support applying a T gate, but does not support **Addition**, H and **Swap**. In this work, we show that computing the fidelity (and hence the inner product, as we can reduce fidelity to inner product) between two states represented by LIMDDs is NP-hard.

Lemma F.16. LIMDD supports Controlled-Z in polynomial time.

Proof. Vinkhuijzen et al. [337] show how to apply any controlled Pauli gate to a state represented by a LIMDD in polynomial time, in the case where the target qubit comes after the control qubit in the variable order of that LIMDD. However, in the case of the controlled-Z, there is no distinction between control and target qubit, since the gate is symmetric. Therefore, their analysis applies to all controlled-Z gates. In fact, inspecting their method, we see that the LIMDD of the resulting state is never larger in size than the LIMDD we started with. □

It is known that addition is hard for QMDD (see Table 2 in [113]). For LIMDD, the same was suspected, but not proved in [337]. We show it here by showing that $\langle Z\rangle$-LIMDD does not support addition in polytime.

Lemma F.17. LIMDD does not support **Addition** in polytime.

Proof. Consider the states $|+^n\rangle$ and $|Rot^n\rangle$ as defined in Lemma F.12. Both states have polynomially sized QMDDs as shown in Lemma F.11. Since LIMDD is at least as succinct as QMDD (see Figure 5.2), the LIMDDs representing these states are also small. However, their sum is the state $|\text{Sum}\rangle = |+^n\rangle + |Rot^n\rangle$, which has an exponential-size LIMDD relative to every variable order by Lemma E.4. □

Lemma F.18. LIMDD does not support **Hadamard** in polynomial time, and hence neither does it support **Local**.

Proof. Since Hadamard can be used together with measurement (called *conditioning* by Fargier [113]) to realize state addition as explained in Section 5.4, it is also intractable. (Recall also from [344] Th. 2.4.1 that conditioning never increases DD size; this is true in particular for LIMDDs) □

We now prove that the fidelity of LIMDDs cannot be computed in polynomial time, under common assumptions of complexity theory.

LIMDD FIDELITY is hard to compute. We show that LIMDD FIDELITY cannot be computed in polynomial time, unless the Exponential Time Hypothesis (ETH) is false. This proof implies that inner product is hard, since fidelity reduces to inner product. Proving hardness of inner product is also a specialized case of the below construction, which does not require our newly defined EOSD problem (see below) but only the well-known hard problem of counting even subgraphs of a certain size (#EVEN SUBGRAPHS).

The proof of LIMDD FIDELITY hardness proceeds in several steps. The starting point is Jerrum and Meeks' result that the problem #EVEN SUBGRAPHS cannot be solved in polynomial time unless ETH is false (Lemma F.19). We introduce a problem we call EVEN ODD SUBGRAPHS DIFFERENCE (EOSD). We give a reduction from #EVEN SUBGRAPHS to EOSD, thus showing that EOSD cannot be solved in polynomial time, under the same assumptions (Lemma F.21). This step is the most technical part of the proof. Finally, we give a reduction form EOSD to LIMDD FIDELITY, thus obtaining the desired result, that LIMDD FIDELITY cannot be computed in polynomial time (to a certain precision), unless ETH is false (Lemma F.20). In this step, we use the fact that LIMDDs can efficiently represent Dicke states and graph states (a type of stabilizer state). Specifically, we will show that computing the fidelity between these states essentially amounts to solving EOSD for the given graph state. Dicke states were first studied by Dicke [101]; see also Bärtschi et al. [32].

We first formally define the three problems above, including computing the fidelity of two LIMDDs. We will need the following terminology for graphs. For an undirected graph $G = (V, E)$ and a set of vertices $S \subseteq V$, we denote by $G[S]$ the subgraph induced by S. If $|S| = k$, then we say that $G[S]$ is a k-induced subgraph, and we say that it is an *even* (resp. *odd*) subgraph if $G[S]$ has an even (resp. odd) number of edges. We let $e(G, k)$ (resp. $o(G, k)$) denote the number of even (resp. odd) k-induced subgraphs of G.

LIMDD FIDELITY.

Input: Two LIMDDs, representing the states $|\varphi\rangle, |\psi\rangle$

Output: The value $|\langle\varphi|\psi\rangle|^2$ to $2n$ bits of precision.

#EVEN SUBGRAPHS

Input: A graph $G = (V, E)$, an an integer k

Output: The value $e(G, k)$.

EVEN ODD SUBGRAPH DIFFERENCE (EOSD).

Input: A graph $G = (V, E)$, and an integer k.

Output: The value $|e(G, k) - o(G, k)|$, i.e., the absolute value of the difference between the number of even and odd induced k-subgraphs of G.

Lemma F.19 (Jerrum and Meeks [169])**.** If #EVEN SUBGRAPHS is polytime, then ETH is false.

Proof. Jerrum and Meeks [169] showed that counting the number of even induced subgraphs with k vertices is #W[1]-hard. Consequently, there is no algorithm running in time $poly(n)$ (independent of k) unless the exponential time hypothesis fails. □

Lemma F.20. There is no polynomial-time algorithm for LIMDD FIDELITY, i.e., for computing fidelity between two LIMDDs to $2n$ bits of precision, unless the Exponential Time Hypothesis (ETH) fails.

Proof. Suppose there was such a polynomial-time algorithm, running in time $\mathcal{O}(n^c)$ for some constant $c \geq 1$. We will show that then EOSD can be solved in time $\mathcal{O}(n^c)$ (independent of k), by giving a reduction from EOSD to LIMDD FIDELITY. From Lemma F.21, it would then follow that ETH is false.

The reduction from EOSD to LIMDD FIDELITY is as follows.

Let G be an input graph on n vertices V and $0 \leq k \leq n$ an integer. Let $|G\rangle$ be the graph state corresponding to G [324], so that

$$|G\rangle = \frac{1}{2^{n/2}} \sum_{S \subseteq V} (-1)^{|G[S]|} |S\rangle \tag{F.4}$$

where $|G[S]|$ denotes the number of edges in the S-induced subgraph of G, and $|S\rangle$ denotes the computational-basis state $|S\rangle = |x_1\rangle \otimes |x_2\rangle \otimes ... \otimes |x_n\rangle$ with $x_j = 1$ if $j \in S$ and $x_j = 0$ otherwise. Let $|D_n^k\rangle$ be the Dicke state [101].

$$|D_n^k\rangle = \frac{1}{\sqrt{\binom{n}{k}}} \sum_{\vec{x}\in\{0,1\}^n \text{ with } |\vec{x}|=k} |\vec{x}\rangle \tag{F.5}$$

Both these states have small LIMDDs:

Dicke state. Bryant [67] gives a construction for BDDs to represent the function $f_k\colon \{0,1\}^n \to \{0,1\}$, with $f_k(x) = 1$ iff $|x| = k$. This is precisely the amplitude function of the Dicke state $|D_n^k\rangle$ (up to a factor $1/\sqrt{\binom{n}{k}}$). This construction also works for LIMDDs, by simply setting all the edge labels to the identity, and using root label $1/\sqrt{\binom{n}{k}} \cdot \mathbb{I}^{\otimes n}$.

Graph state. Vinkhuijzen et al. [337] show how to efficiently construct a LIMDD for any graph state.

It is straightforward to verify that the fidelity between $|D_n^k\rangle$ and $|G\rangle$ is related to the subgraphs of G, as follows,

$$\langle D_n^k|G\rangle = \frac{1}{\sqrt{\binom{n}{k}2^n}} \sum_{S\subseteq V:|S|=k} (-1)^{|G[S]|} = \frac{1}{\sqrt{\binom{n}{k}2^n}} (e(G,k) - o(G,k)) \tag{F.6}$$

Hence,

$$\underbrace{|e(G,k) - o(G,k)|}_{\text{solution to EOSD}} = \sqrt{\binom{n}{k} 2^n \underbrace{|\langle D_n^k|G\rangle|^2}_{\text{Fidelity}}} \tag{F.7}$$

Since $|\langle D_n^k|G\rangle|^2$ denotes the fidelity between $|D_n^k\rangle$ and $|G\rangle$, and $|e(G,k) - o(G,k)|$ denotes the quantity asked for by the EOSD problem, this completes the reduction. The overhead of constructing the LIMDDs from the description of the Dicke and graph states takes linear time in the size of the resulting LIMDD. So, if the fidelity of two LIMDDs is computed in polynomial time, say, in time $\mathcal{O}(n^c)$, then also the quantity $|e(G,k) - o(G,k)|$ is computed in time $\mathcal{O}(n^c)$; thus, EOSD is solved in time $\mathcal{O}(n^c)$. Lastly, we address the number of bits of precision required. In order to exactly compute the integer $|e(G,k) - o(G,k)|$, it is necessary to compute the fidelity $|\langle D_n^k|G\rangle|^2$ with a precision of at least one part in $\binom{n}{k}2^n$. Put another way, the required number of bits of precision is $\log_2(\binom{n}{k} \cdot 2^n) \leq \log_2(2^n \cdot 2^n) = 2n$. Summarizing, computing the fidelity of (the states represented by) two LIMDDs representing a graph state and a Dicke state, to $2n$ bits of precision, is not possible in polynomial time, unless ETH fails. □

Lemma F.21. There is no polynomial-time algorithm for EOSD, unless ETH is false.

Proof. We provide an efficient reduction (in Algorithm 21) from #EVEN SUBGRAPHS: the problem, on input an undirected graph G and a parameter $k \in \{0, 1, 2, ..., |V|\}$, of computing the number of k-vertex induced subgraphs which have an even number of edges. It follows that, if EOSD can be computed in polynomial time, then Algorithm 21, which computes #EVEN SUBGRAPHS, also runs in polynomial time. Jerrum and Meeks [169] show that #EVEN-SUBGRAPHS cannot be computed in polynomial time unless ETH is false (Lemma F.19). Therefore, if EOSD could be computed in polynomial time, then ETH would be false.

The algorithm CountEvenSubgraphs (Algorithm 21) takes as parameters a graph G and an integer $k \geq 0$, and outputs $e(G, k)$, the number of even k-induced subgraphs of G, thus solving #EVEN SUBGRAPHS. This algorithm uses at most $2n$ invocations of a subroutine EvenOddSubgraphsDifference; therefore, if the subroutine EvenOddSubgraphsDifference runs in polynomial time, then so does CountEvenSubgraphs.

Let us briefly sketch the idea behind the algorithm, before we give a formal proof of correctness. First, we know that $e(G, k) + o(G, k) = \binom{n}{k}$, since each subgraph is either even or odd, and G has $\binom{n}{k}$ different k-induced subgraphs in total. Thus, if we knew the (possibly negative) difference $\zeta_k = e(G, k) - o(G, k)$, then we know the sum and difference of $e(G, k)$ and $o(G, k)$, so we could compute the desired value $e(G, k) = \frac{1}{2}(\binom{n}{k} + \zeta_k)$. Unfortunately, EvenOddSubgraphsDifference only tells us the absolute value, $|\zeta_k|$. Fortunately, we know that $e(G, 0) = 1$ and $o(G, 0) = 0$, so $\zeta_0 = 1 - 0 = 1$ (namely, there is only one induced subgraph with 0 vertices, and it has 0 edges, which is even). We now bootstrap our way up, computing ζ_j for $j = 1, \ldots, k$ using the previously known results. The key ingredient is that, by adding isolated vertices to the graph and querying EvenOddSubgraphsDifference on this new graph, we can discover the the absolute difference $|\zeta_j + \zeta_{j-1}|$, which allows us to compute the values ζ_j.

Correctness of the algorithm. We now prove that the algorithm CountEven-

SUBGRAPHS outputs the correct value. Let (G, k) be the input to the algorithm. For $j = 0, \dots, k$, let $\zeta_j = e(G, j) - o(G, j)$. We will show that, for each $j = 1, \dots, k$, the algorithm sets the variable d_j to the value ζ_j in the j-th iteration of the for-loop. The proof is by induction on j. In the induction hypothesis, we include also that the variable ℓ is always the largest value below j satisfying $\zeta_\ell \neq 0$ as in Equation F.8 (this value is well-defined, since $1 = \zeta_0 \neq 0$, so we have $0 \leq \ell < j$).

$$\ell = \max\{0 \leq \ell < j \mid \zeta_\ell \neq 0\} \tag{F.8}$$

For the base case, where $j = 0$, it suffices to note that there is only one set with zero vertices – the empty set – which induces the empty graph, which contains an even number of edges. Therefore, $\zeta_0 = 1$, which the algorithm sets on Line 2. Finally, ℓ is correctly set to 0.

For the induction case $j \geq 1$, the variables d_t have been set to $d_t = \zeta_t$ for $t = 0, \dots, j-1$ and ℓ satisfies Equation F.8 from the induction hypothesis. Consequently, we have $\zeta_{\ell+1} = \dots = \zeta_{j-1} = 0$. If $\zeta_j = 0$, then the algorithm sets $q := |\zeta_j| = |0| = 0$ on Line 5, so the algorithm sets d_j correctly on Line 7, and correctly leaves ℓ untouched (ℓ remains unchanged from the $j-1$-th to the j-th iteration). Otherwise, if $\zeta_j \neq 0$,

Algorithm 21 An algorithm which computes the number of even k-induced subgraphs using at most $2n$ calls to a subroutine EVENODDSUBGRAPHSDIFFERENCE, which returns $|e(G, k) - o(G, k)|$ on input (G, k).

```
1:  procedure CountEvenSubgraphs($G = (V, E), k$)
    Output: The number of even induced subgraphs of $G$ with $k$ vertices
2:      $d_0 := 1$                                    ▷ $d$ is an array of $k+1$ integers
3:      $\ell := 0$                                   ▷ Last iteration when $\zeta_j = 1$
4:      for $j := 1, \dots, k$ do
5:          $q :=$ EvenOddSubgraphsDifference($G, j$)
6:          if $q = 0$ then                           ▷ There are equally many even as odd subgraphs
7:              $d_j := 0$
8:          else          ▷ Else we have to figure out whether there more even or odd subgraphs:
9:              $G' := (V \cup \{v'_1, \dots, v'_{j-\ell}\}, E)$     ▷ Add $j - \ell$ new isolated vertices
10:             $p :=$ EvenOddSubgraphsDifference($G', j$)
11:             $d_j := \begin{cases} q & \text{if } |d_\ell + q| = p \\ -q & \text{if } |d_\ell - q| = p \end{cases}$
12:             $\ell := j$                           ▷ Since iteration $j$ is the latest iteration having $\zeta_j = 1$
13:     return $\frac{1}{2}\left(\binom{n}{k} + d_k\right)$
```

the algorithm adds $j-\ell$ new, isolated vertices to G, obtaining the new graph $G' = (V_G \cup \{v'_1, \ldots, v'_{j-\ell}\}, E_G)$. On Line 10, it computes the value $|e(G', j) - o(G', j)|$ of this graph. Since this expression also sums over induced subgraphs of G' that contain isolated vertices, this value can be expressed as follows:

$$p := |e(G', j) - o(G', j)| \tag{F.9}$$

$$= \left| \sum_{a=\ell}^{j} \binom{j-\ell}{j-a} (e(G,a) - o(G,a)) \right| = \left| \sum_{a=\ell}^{j} \binom{j-\ell}{j-a} \zeta_a \right| \tag{F.10}$$

$$= \left| \binom{j-\ell}{j-\ell} \zeta_\ell + \binom{j-\ell}{0} \zeta_j \right| = |\zeta_\ell + \zeta_j| \tag{F.11}$$

We noted that $\zeta_{\ell+1} = \cdots = \zeta_{j-1} = 0$; therefore, these terms vanish from the summation (step from Equation F.10 to Equation F.11), so that only $p = |\zeta_j + \zeta_\ell|$ remains. Since we now know the values of $\zeta_\ell, |\zeta_j|$ and $|\zeta_j + \zeta_\ell|$ and since $\zeta_\ell \neq 0$, we can infer the value of ζ_j, which is done on Line 11. We conclude that each variable d_j is correctly set to ζ_j, concluding the proof by induction. Also, since $\zeta_j \neq 0$, ℓ is correctly set to j.

Lastly, we show that the value returned by the algorithm is indeed the number $e(G,k)$. Suppose that $d_k = \zeta_k = e(G,k) - o(G,k)$. We know that $e(G,k) + o(G,k) = \binom{n}{k}$. That is, we know both the sum and the difference of $e(G,k), o(G,k)$; therefore we can compute them both. By adding these two equations and solving for $e(G,k)$, we obtain $e(G,k) = \frac{1}{2}\left(\binom{n}{k} + d_k\right)$, which is the value returned by the algorithm. □

Lemma F.22. LIMDD supports the T-gate.

Proof. Algorithm 22 applies an arbitrary diagonal gate $D = \left[\begin{smallmatrix} \rho & 0 \\ 0 & \omega \end{smallmatrix}\right]$ to a state represented by a LIMDD. To apply a T-gate, one calls the algorithm with $D = T = \left[\begin{smallmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{smallmatrix}\right]$. We now show that the algorithm runs in polynomial time. First, since each recursive call takes $\mathcal{O}(1)$ time, for the purposes of estimating runtime it suffices to count the number of recursive calls. The cache stores all tuples of nodes and matrices with which the algorithm is called, so for the purposes of estimating the runtime it suffices to count the number of *distinct* recursive calls. To this end, we note that the recursive calls to the algorithm only receive two different matrices, namely $\left[\begin{smallmatrix} \rho & 0 \\ 0 & \omega \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} \omega & 0 \\ 0 & \rho \end{smallmatrix}\right]$. The nodes that are passed as argument v are nodes that are already in the diagram. Therefore, if the diagram contains m nodes, then at most $2m$ distinct recursive calls are made. We conclude that the runtime is polynomial (indeed, linear), in the size of the diagram. □

Algorithm 22 Applies a diagonal gate D to qubit k of a state represented by a LIMDD.

```
1: procedure ApplyDiagGate(LIMDD Node $v = (v_0) \overset{\lambda_0 A_0}{\cdots\cdots} \bigcirc \overset{\lambda_1 A_1}{\text{———}} (v_1)$, gate $D$, qubit $k$)
   with $D = \left[\begin{smallmatrix} \rho & 0 \\ 0 & \omega \end{smallmatrix}\right]$
   Node $v$ represents a state on $n$ qubits
2:     if Cache contains the tuple $(v, D)$ then return Cache$[v, D]$
3:     else if $k = n$ then
4:         return $(v_0) \overset{\rho\lambda_0 A_0}{\cdots\cdots} \bigcirc \overset{\omega\lambda_1 A_1}{\text{———}} (v_1)$
5:     else
6:         gate $E := \left[\begin{smallmatrix} \omega & 0 \\ 0 & \rho \end{smallmatrix}\right]$
7:         for $i = 0, 1$ do
8:             gate $F_i := \begin{cases} D & \text{if } A_i^k \in \{I, Z\} \\ E & \text{if } A_i^k \in \{X, Y\} \end{cases}$    ▷ Here $A_i^k$ denotes the $k$-th qubit of the Pauli operator $A^i$
9:             Node $u_i :=$ ApplyTGateToLIMDD$(v_0, F_i, k)$
10:        Node $r := (u_0) \overset{\lambda_0 A_0}{\cdots\cdots} \bigcirc \overset{\lambda_1 A_1}{\text{———}} (u_1)$
11:        Cache$[v, D] := r$
12:        return $r$
```

F.4 Easy operations for MPS

Vidal [335] shows that MPS supports efficient application of a single one-qubit gate or two-qubit gate on consecutive qubits, which includes **X,Y,Z**, **Hadamard**, **T**. This extends to any two-qubit operation on any pair of qubits [262], particularly including **Swap** and **CZ** gates. These algorithms are extendable to k-local gates on adjacent qubits, which does not increase the largest matrix dimension D to more than D^k. The algorithm consists of merging the k tensors (the j-th tensor combines the two matrices A_j^0 and A_j^1) into a single large one, applying the gate to the large tensor, followed by splitting the tensor again into k matrices A_j^0 and A_j^1 again by use of the singular-value decomposition (for details on the merging and splitting see e.g. Dang et al. [93]). The largest matrix dimension during this process does not increase above D^k. Using **Swap** gates, one thus implements **Local** on any qubits. We give a direct proof of the support for addition below (Lemma F.23).

Orus [248] gives an accessible exposition of TN, of which MPS is a special case. He explains how to compute the inner product in polynomial time. Thus, MPS also supports **Measure**. **Sample** can be done by a Markov Chain Monte Carlo approach, invoking **Measure** as subroutine. Since inner product is supported, so is **Equal**: MPS M and M' are equivalent iff $\frac{|\langle M|M'\rangle|^2}{\langle M|M\rangle \cdot \langle M'|M'\rangle} = 1$.

Lemma F.23. MPS supports addition in polynomial time.

Proof. Let A, B be MPSs. Then a new MPS C representing $|C\rangle = |A\rangle + |B\rangle$ can be efficiently constructed as follows, for $x = 0, 1$ and $j = 2, \ldots, n-1$:

$$C_n^x = \begin{bmatrix} A_n^x & B_n^x \end{bmatrix} \qquad C_j^x = \begin{bmatrix} A_j^x & 0 \\ 0 & B_j^x \end{bmatrix} \qquad C_1^x = \begin{bmatrix} A_1^x \\ B_1^x \end{bmatrix} \tag{F.12}$$

□

F.5 Easy and hard operations for RBM

Jonsson et al. [174] show that RBM supports Pauli gates, the controlled-Z gate and the T-gate (and, in fact, arbitrary phase gates). There is at the moment no efficient exact algorithm for the **Hadamard** gate, which would make the list of supported gates universal. Hence there is at the moment no exact efficient algorithm for **Local** either. **Sample** is supported for any n-qubit RBM M, see e.g. Appendix B of [174] and references therein, by performing a Markov Chain Monte Carlo algorithm (e.g. Metropolis algorithm) where the Markov Chain state space consists of all bit strings $x \in \{0, 1\}^n$, and the corresponding unnormalized probability $|\langle x|M\rangle|^2$ of each state is efficiently computed using Equation 5.1. No exact algorithm for **Equal** is known (in fact, the related problem of identity testing when one only has sampling access to one of the two RBMs is already computationally hard [52]). Although no exact algorithm for **InnerProd** is known, it can be approximated using **Sample** as subroutine (see e.g. Wu et al. [354]). Furthermore, **Measure** can be approximated by computing the normalization factor $1/\langle M|M\rangle$ using the (exact or approximate) algorithm for **InnerProd**, while the relative outcome probabilities are defined in Equation 5.1.

Lemma F.24. RBM supports **Swap**.

Proof. In order to effect a swap between qubits q_1 and q_2, we simply exchange rows q_1 and q_2 in the matrix W and the vector $\vec{\alpha}$, obtaining W' and $\vec{\alpha}'$. Then $\mathcal{M}' = (\vec{\alpha}', \vec{\beta}, W', m)$ has $|\mathcal{M}'\rangle = \textsc{Swap}(q_1, q_2) \cdot |\mathcal{M}\rangle$. □

Torlai et al. [315] note that RBMs can exactly represent Dicke states. In Lemma F.25, we give another construction of succinct RBMs for Dicke states, where the number of hidden nodes grows linearly with the number of visible nodes.

Lemma F.25. An RBM can exactly represent any Dicke state, using only $2n$ hidden nodes.

Proof. We will construct an RBM with $2n$ hidden nodes representing $|D_n^k\rangle$.

For each $j \in \{0, 1, \ldots, n\} \setminus \{k\}$, our construction will use two hidden nodes. Fix such a j. Then the first hidden node is connected to each visible node with weight $i\pi/n$, and has bias $b_j = i\pi(1 - j/n)$. The second hidden node is connected to each visible node with weight $-i\pi/n$ and has bias $b_j = -i\pi(1 - j/n)$. Since the weights on all edges incident to a given hidden node are the same, the term it contributes depends only on the weight of the input (i.e., the number of zeroes and ones). Thus, these two nodes contribute a multiplicative factor $(1 + e^{i\pi(1+|x|/n+j/n)})$ and $(1 + e^{-i\pi(1+|x|/n-j/n)})$, respectively. Multiplying these together, the two terms collectively contribute a multiplicative term of $2+2\cos(\pi(1+|x|/n-j/n) = 2-2\cos(\pi(|x|-j)/n)$, which is 0 iff $|x| = j$ and nonzero otherwise. Let a be the constant $a = \prod_{j=0,j\neq k}^{n} 2 - 2\cos(\pi(k-j)/n)$, i.e., the product of all terms when $|x| = k$. Then the RBM represents the following state unnormalized function $\Psi\colon \{0,1\}^n \to \mathbb{C}$, which we then normalize to obtain the state $|\Psi\rangle$:

$$\Psi(x) = \begin{cases} a & \text{if } |x| = k \\ 0 & \text{otherwise} \end{cases} \qquad\qquad |\Psi\rangle = \frac{1}{a\sqrt{\binom{n}{k}}} \sum_x \Psi(x)\,|x\rangle \tag{F.13}$$

The normalized state $|\Psi\rangle$ represents exactly the Dicke state $|D_n^k\rangle$, since its amplitudes are equal to $1/\sqrt{\binom{n}{k}}$ when $|x| = k$ and zero otherwise. □

Since RBM can succinctly represent both Dicke states (Lemma F.25) and graph states, a subset of stabilizer states [365], the proof for hardness of LIMDD FIDELITY (Lemma F.20) is also applicable to RBM.

Corollary F.1. There is no polynomial-time algorithm for RBM FIDELITY, i.e., for computing fidelity between two RBM to $2n$ bits of precision, unless the Exponential Time Hypothesis (ETH) fails.

Appendix G

Proofs of Section 5.5

Section G.1 proves that our rapidity definition is a preorder and is equivalent to the one given by Lai et al. [194] for canonical data structures.

Section G.2 provides the proof for the sufficient condition for rapidity. Section G.3-G.5 apply this sufficient condition to the data structures studied in this work.

G.1 Rapidity is a preorder and generalizes earlier definitions

We now show that rapidity is a preorder over data structures and that the definition of Lai et al. [194] can be considered a special case for canonical data structures. For convenience, we restate the definition of rapidity.

Definition 5.3 (Rapidity for non-canonical data structures)**.** Let D_1, D_2 be two data structures and consider some c-ary operation OP on these data structures. In the below, ALG_1 (ALG_2) is an algorithm implementing OP for D_1 (D_2).

(a) We say that ALG_1 is *at most as rapid as* ALG_2 iff there exists a polynomial p such that for each input $\varphi = (\varphi_1, \dots, \varphi_c)$ there exists an equivalent input $\psi = (\psi_1, \dots, \psi_c)$, i.e., with $|\varphi_j\rangle = |\psi_j\rangle$ for $j = 1 \dots c$, for which $\mathit{time}(ALG_2, \psi) \leq p\,(\mathit{time}(ALG_1, \varphi))$. We say that ALG_2 is *at least as rapid as* ALG_1.

(b) We say that $OP(D_1)$ is *at most as rapid as* $OP(D_2)$ if for each algorithm ALG_1 performing $OP(D_1)$, there is an algorithm ALG_2 performing $OP(D_2)$ such that ALG_1 is at most as rapid as ALG_2.

Theorem 5.4. Rapidity is a preorder over data structures.

Proof. We first show that rapidity is reflexive, next we show that it is transitive.

Rapidity is reflexive. It suffices to show that rapidity is a reflexive relation on algorithms performing a given operation. Let D be a data structure, OP an operation and ALG an algorithm performing $OP(D)$. Then ALG is at most as rapid as itself if there exists a polynomial p such that for each input φ there exists an equivalent input ψ with $time(ALG, \varphi) \leq p(ALG, \psi)$. We may choose the polynomial $p(x) = x$, and we may choose $\psi := \varphi$. Then the statement reduces to the trivial statement $time(ALG, \varphi) = time(ALG, \psi) = p(ALG, \psi)$.

Rapidity is transitive. It suffices to show that rapidity is a transitive relation on algorithms. To this end, let D_1, D_2, D_3 be data structures, OP an operation and ALG_1, ALG_2, ALG_3 algorithms performing $OP(D_1), OP(D_2), OP(D_3)$, respectively. Suppose that ALG_1 is at most as rapid as ALG_2 and ALG_2 is at most as rapid as ALG_3. We will show that ALG_1 is at most as rapid as ALG_3. By the assumptions above, there are polynomials p and q such that (i) for each input φ there exists an equivalent input ψ such that $time(ALG_2, \psi) \leq p(time(ALG_1, \varphi))$; and (ii) for each input ψ there exists an equivalent input γ such that $time(ALG_3, \gamma) \leq q(time(ALG_2, \psi))$.

Put together, for every input φ there exist equivalent inputs ψ and γ such that $time(ALG_3, \gamma) \leq q(time(ALG_2, \psi)) \leq q(p(time(ALG_1, \varphi)))$. Letting the polynomial $\ell(x) = q(p(x))$, we obtain that for every φ there exists an equivalent γ such that $time(ALG_3, \gamma) \leq \ell(ALG_1, \varphi)$. □

We note that an alternative definition of rapidity [194], which always allows ALG_2 to read its input by requiring $\text{time}(ALG_2, y) \leq p(\text{time}(ALG_1, x) + |y|)$ instead of $\text{time}(ALG_2, y) \leq p(\text{time}(ALG_1, x))$, is not transitive for query operations:

Consider the data structure Padded QMDD, (PQMDD) which is just a QMDD, except that a string of 2^{2^n} "0"'s have been concatenated to the end of the QMDD representation, where n is the number of qubits.

Under the alternative rapidity relation $\geq_r^{\text{alt}}$, both ADD and QMDD are at least as rapid as PQMDD, because the ADD algorithm is allowed to run for $poly(2^{2^n})$ time. But PQMDD is also at least as rapid as QMDD, because algorithms for PQMDD don't need to read the whole 2^{2^n}-length input — they only read the QMDD at the beginning of the string. Put together, this leads to:

$$\mathsf{ADD} \geq_r^{\text{alt}} \mathsf{PQMDD} \geq_r^{\text{alt}} \mathsf{QMDD} \quad \text{and} \quad \mathsf{ADD} \not\geq_r^{\text{alt}} \mathsf{QMDD}.$$

Next, we show that our definition of rapidity is equivalent to Lai et al.'s definition of rapidity in the case when both data structures are canonical and we restrict our attention to only those algorithms which run in time at least m where m is the size of the input. For convenience, we restate Lai et al.'s definition here.

Definition G.1 (Rapidity for canonical data structures [194])**.** A c-ary operation OP on a canonical language L_1 is *at most as rapid as* OP on another canonical language L_2, iff for each algorithm ALG performing OP on L_1 there exists some polynomial p and some algorithm ALG_2 performing OP on L_2 such that for every valid input $(\varphi_1, \dots, \varphi_c, \alpha)$ of OP on L_1 and every valid input $(\psi_1, \dots, \psi_c, \alpha)$ of OP on L_2 satisfying $\varphi_i \equiv \psi_i$ $(1 \leq i \leq c)$, $ALG_2(\psi_1, \dots, \psi_c, \alpha)$ can be done in time $p(t + |\varphi_1| + \dots + |\varphi_c| + |\alpha|)$, where α is any element of supplementary information and t is the running time of $ALG(\varphi_1, \dots, \varphi_c, \alpha)$.

Lai et al. use several minor differences in notation. First, they speak of *valid* inputs (because they consider data structures which cannot represent all objects), whereas we do not; they use an element of supplementary information α as part of the input, whereas we omit such an element; they write $\varphi_i \equiv \psi_i$ where we write $|\varphi_i\rangle = |\psi_i\rangle$; lastly they speak of a *language* whereas we speak of a *data structure*. Since these differences between the notation are inconsequential, it will be convenient to rephrase the definition of Lai et al. using the notation of this work, as follows:

Definition G.2 (Rapidity of canonical data structures, rephrased)**.** In the following, ALG_1, ALG_2 are algorithms which perform OP on canonical data structures D_1, D_2, respectively.

(a) An algorithm ALG_1 is *at most as rapid as* an algorithm ALG_2 iff there is a polynomial p such that for each input φ and for each equivalent input ψ, it holds that $time(ALG_2, \psi) \leq p(time(ALG_1, \varphi) + |\varphi|)$.

(b) A canonical data structure D_1 is *at most as rapid as* a canonical data structure D_2 for an operation OP if for each algorithm ALG_1 performing OP on D_1 there is an algorithm ALG_2 performing OP on D_2 such that ALG_1 is at most as rapid as ALG_2.

Lemma G.1. Definition 5.3 is equivalent to the definition of [194] (Definition G.2) in the case when two data structures D_1, D_2 are both canonical and where we restrict our attention to algorithms whose runtime is at least m, where m is the size of the input.

Proof. Let D_1, D_2 be two canonical data structures. We will show that D_1 is at most as rapid as D_2 according to Definition 5.3 if and only if the same is true according tot Definition G.2. Since items 5.3.(b) and G.2.(b) are equivalent, it suffices to show that the two definitions are equivalent for *algorithms* rather than *data structures*. That is, we will show that an algorithm ALG_1 is at most as rapid as ALG_2 according to Definition 5.3 if and only if the same is true according to Definition G.2.

Abusing notation, we write $|(\varphi_1, \dots, \varphi_c)|$ instead of $|\varphi_1| + \ldots + |\varphi_c|$, etc. In this proof, we will assume without loss of generality that all polynomials p are monotonically increasing (i.e., $p(x) \leq p(y)$ if $x \leq y$). Namely, if p is a polynomial which does not monotonically increase, then use instead the polynomial $p'(x) = p(x) + x^k$ for sufficiently large k.

Direction *if*. Let ALG_1, ALG_2 be algorithms performing OP on canonical data structures D_1, D_2, respectively, such that ALG_1 is at most as rapid as ALG_2 according to Definition G.2. Then there is a polynomial p such that $time(ALG_2, \psi) \leq p(time(ALG_1, \varphi) + |\varphi|)$ for all equivalent inputs φ, ψ. Since the data structures D_1, D_2 can represent all quantum state vectors, there certainly *exists* an equivalent ψ to any φ; indeed, since D_2 is canonical, there is a unique such instance ψ. Since we restrict our attention to algorithms with runtime at least m where m is the size of the input, we get that $|\varphi| \leq time(ALG_1, \varphi)$, so $p(time(ALG_1, \varphi) + |\varphi|) \leq p(2 \cdot time(ALG_1, \varphi))$.

Therefore, let $q(x) = p(2x)$. Now we get that, for every input φ, there exists an equivalent input ψ such that $time(ALG_2, \psi) \leq q(ALG_1, \varphi)$. Therefore, ALG_1 is at most as rapid as ALG_2 according to Definition 5.3.

Direction *only if*. Suppose that ALG_1 is at most as rapid as ALG_2 according to Definition 5.3. Then there is a polynomial p such that for each input φ, there is an equivalent input ψ such that $time(ALG_2, \psi) \leq p(time(ALG_1, \varphi))$. Using the monotonicity of p which we assume without loss of generality, we get that $p(time(ALG_1, \varphi)) \leq p(time(ALG_1, \varphi) + |\varphi|)$. Lastly, since D_2 is canonical, any instance ψ which is equivalent to φ must be the *only* input instance that is equivalent to φ. Therefore, we obtain that there exists a polynomial p such that for each input φ and for all equivalent inputs ψ (i.e., for the unique equivalent instance ψ of D_2), it holds that $time(ALG_2, \psi) \leq p(time(ALG_1, \varphi) + |\varphi|)$. Therefore, ALG_1 is at most as rapid as ALG_2 according to Definition G.2. □

G.2 A Sufficient Condition for Rapidity

Here, we prove Theorem 5.5, which we restate below.

Theorem 5.5 (A sufficient condition for rapidity)**.** Let D_1, D_2 be data structures with $D_1 \preceq_s D_2$ and OP a c-ary operation. Suppose that,

A1 $OP(D_2)$ requires time $\Omega(m)$ where m is the sum of the sizes of the operands; and

A2 for each algorithm ALG implementing $OP(D_2)$, there is a runtime monotonic algorithm ALG^{rm}, implementing the same operation $OP(D_2)$, which is at least as rapid as ALG; and

A3 there exists a transformation from D_1 to D_2 which is (i) weakly minimizing and (ii) runs in time polynomial in the output size (i.e, in time $\mathsf{poly}(|\psi|)$ for transformation output $\psi \in D_2$); and

A4 if OP is a manipulation operation (as opposed to a query), then there also exists a polynomial time transformation from D_2 to D_1 (polynomial time in the input size, i.e, in $|\rho|$ for transformation input $\rho \in D_2$).

Then D_1 is at least as rapid as D_2 for operation OP.

Proof. We prove the theorem for $c = 1$. This can be easily extended to the case with multiple operands by treating the operands point-wise and summing their sizes. We show that $OP(D_2)$ is at most as rapid as $OP(D_1)$, assuming that the conditions in Theorem 5.5 hold. (Note that this swaps the roles of D_1 and D_2 relative to Definition 5.3). In this proof, we will assume without loss of generality that all polynomials p are monotone, i.e., if $x \leq y$ then $p(x) \leq p(y)$.

We prove the theorem for a manipulation operation OP. The proof for a query operation OP follows as a special case, which we treat at the end of the proof.

Let ALG_2 be an $\Omega(m)$ algorithm implementing $OP(D_2)$. By A2, we may assume without loss of generality that ALG_2 is runtime monotonic. Let $f \colon D_1 \to D_2$ be the polynomial-time weakly minimizing transformation (A3), and $g \colon D_2 \to D_1$ the polynomial-time transformation in the other direction satisfying the criteria in A4.

We set $ALG_1 = f \circ ALG_2 \circ g$, i.e., ALG_1 is as follows.

```
1: procedure $ALG_1(\varphi)$
2:     $\psi := f(\varphi)$
3:     $\rho := ALG_2(\psi)$
4:     return $g(\rho)$
```

ALG_1 is *complete* (i.e., works on all inputs), since f, g and ALG_2 are. The remainder of the proof shows that ALG_2 is at most as rapid as ALG_1, i.e., there exists a polynomial p such that for all operands $\psi \in D_2$, there exists in input $\varphi \in D_1$ with $|\varphi\rangle = |\psi\rangle$ for which $time(ALG_1, \varphi) \leq p\,(time(ALG_2, \psi))$.

Let $\psi \in D_2$. We take $\varphi \in D_1$ such that $|\varphi\rangle = |\psi\rangle$ and $|\varphi| \leq s(|\psi|)$ for the polynomial s ensuring the succinctness relation $D_1 \preceq_s D_2$. Such a φ exists, because D_1 is more succinct than D_2.

It remains to show that $\exists p \colon time(ALG_1, \varphi) \leq p\,(time(ALG_2, \psi))$, where p is independent of φ and ψ. To this end, we can express the time required by ALG_1 by summing the runtimes of its three steps as follows.

$$time(ALG_1, \varphi) = time(f, \varphi) + time(ALG_2, f(\varphi)) + time(g, ALG_2(f(\varphi))) \tag{G.1}$$

It now suffices to prove that each summand of Equation G.1 is polynomial in the runtime of $ALG_2(\psi)$.

1. We show $time(f, \varphi) \leq \mathsf{poly}(time(ALG_2, \psi))$. Since f runs in polynomial time in its output (A3) and $|\varphi| \leq s(|\psi|)$ (see above), we have $time(f, \varphi) \leq \mathsf{poly}(|f(\varphi)|)$. Let t be the polynomial such that $time(f, \varphi) \leq t(|f(\varphi)|)$. Since f is weakly minimizing (A3), it is guaranteed that $|f(\varphi)| \leq m(|\psi|)$ for some polynomial m. Lastly, by A1, we have $|\psi| = \mathcal{O}(time(ALG_2^{rm}, \psi))$, so $|\psi| \leq k(time(ALG_2, \psi))$ for some polynomial k. Put together, we have $time(f, \varphi) \leq t(|f(\varphi)|) \leq t(m(|\psi|)) \leq$

$t(m(k(\mathit{time}(ALG_2, \psi))))$, which proves the claim.

2. We show $\mathit{time}(ALG_2, f(\varphi)) \leq \mathsf{poly}(\mathit{time}(ALG_2, \psi))$. Because f is a weakly minimizing transformation (A3), we have $|f(\varphi)| \leq s(|\psi|)$ for some s. Since ALG_2 is runtime monotonic (A2), and because $|f(\varphi)| \leq s(|\psi|)$, we have $\mathit{time}(ALG_2, f(\varphi)) \leq t(ALG_2, \psi)$ for some t, which proves the claim.

3. We show $\mathit{time}(g, ALG_2(f(\varphi))) \leq \mathsf{poly}(\mathit{time}(ALG_2, \psi))$. Since g runs in time polynomial in the input (A4); and the input to g is $ALG_2(f(\varphi))$, we have $\mathit{time}(g, ALG_2(f(\varphi))) \leq p(|ALG_2(f(\varphi))|)$ for some polynomial p. Next, we have trivially $\mathit{time}(ALG_2, f(\varphi)) \geq |ALG_2(f(\varphi))|$, since the time ALG_2 spends writing the output is included in the total time, thus we obtain $\mathit{time}(g, ALG_2(f(\varphi))) \leq p(\mathit{time}(ALG_2, f(\varphi)))$. As we have seen above in item 2, $\mathit{time}(ALG_2, f(\varphi)) \leq t(\mathit{time}(ALG_2, \psi))$ for some polynomial t. Putting this together, we obtain $\mathit{time}(g, ALG_2(f(\varphi))) \leq p(t(\mathit{time}(ALG_2, \psi)))$, which proves the claim.

This proves the theorem for the case when OP is a manipulation operation.

Lastly, if OP is a query operation rather than a manipulation operation, then the transformation from D_2 back to D_1 using g is no longer necessary. This is the only change needed in ALG_1; in the proof above, we may use $\mathit{time}(g, ALG_2(f(x_1))) = 0$. The requirement that $\mathit{time}(g, ALG_2(f(x_1))) \leq p(\mathit{time}(ALG_2, x_2))$ now holds vacuously. □

G.3 Rapidity Relations between Data Structures

Here we prove the rapidity relations between data structures studied in the paper as stated in Theorem 5.6, restated below with proof.

Theorem 5.6. The rapidity relations in Figure 5.4 hold.

Proof. The relation between QMDD and MPS is proved in Theorem 5.7 as restated in Section G.4. Finally, Section G.5 provides the transformations between QDDs that fulfill the conditions of Theorem 5.5. □

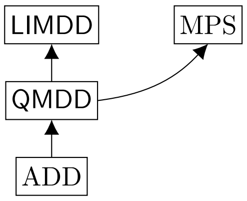

Figure G.1: Rapidity relations between data structures considered here. A solid arrow $D_1 \to D_2$ means D_2 is at least as rapid as D_1 for all operations satisfying A1 and A2 of Theorem 5.5.

G.4 MPS is at least as Rapid as QMDD

This appendix proves Theorem 5.7 from Sec. 5.5.2 by providing transformations between MPS and QMDD that realize the sufficient conditions of Theorem 5.5. The introduction to QDDs, given in Section 2.3, is relevant here.

Theorem 5.7. MPS is at least as rapid as QMDD for all operations satisfying A1 and A2.

Proof. Let f be the polynomial-time transformation from Lemma G.2. Let g be the weakly minimizing transformation from MPS to QMDD of Lemma G.3, that runs in time polynomial in the size of the input MPS and the resulting QMDD. These transitions satisfy requirements A3 and A4 of Theorem 5.5 respectively. Since QDDs are canonical data structures as explained in Section 5.2, all algorithms are by definition runtime monotonic, as for any state $|\varphi\rangle$ there is only one structure representing it, i.e., D^φ is a singleton set. This satisfies A2. Since its premise fulfills A1, the theorem follows. □

Lemma G.2 (QMDD to MPS)**.** In polynomial time, a QMDD can be converted to an MPS representing the same state.

Proof. Consider a QMDD with root edge $\xrightarrow{\lambda} \textcircled{v}$ describing a state $|\varphi\rangle = \sum_{\vec{x}\in\{0,1\}^n} \alpha(\vec{x}) |\vec{x}\rangle$. We will construct an MPS A describing the same state. For the purposes of this proof, we will call low edges *0-edges* and high edge *1-edges*.

First, without loss of generality, we may assume that the root edge label is $\lambda = 1$. Namely, we may multiply the labels on the root's low and high edges with λ, and then set the root edge label to 1; this operation preserves the state represented by the QMDD.

Denote by D_ℓ the number of nodes at the ℓ-th layer in QMDD v, i.e. $D_n = 1$ (the root node v) and $D_0 = 1$ (the leaf $\boxed{1}$). Recall that the QMDD is a directed, weighted graph whose vertices are divided into $n+1$ layers, i.e., the edges only connect nodes from consecutive layers. Therefore, we may speak of the $D_\ell \times D_{\ell-1}$ bipartite adjacency matrix between layer ℓ and layer $\ell-1$ of the diagram. For layer $1 \le \ell \le n$ and $x = 0, 1$, let A_ℓ^x be the $D_\ell \times D_{\ell-1}$ bipartite adjacency matrix obtained in this way using only the low edges if $x = 0$, and only the high edges if $x = 1$. That is, assuming some order on nodes within each level, the entry of the matrix A_ℓ^x in row r and column c is defined as

$$(A_\ell^x)_{r,c} = \begin{cases} \text{label}(e) & \text{if node with index } r \text{ in level } \ell \text{ has a } x\text{-edge } e \\ & \text{to node with index } c \text{ in level } \ell-1 \\ & \\ 0 & \text{otherwise} \end{cases} \tag{G.2}$$

We claim that the following MPS A describes the same state as the QMDD:

$$A = (A_1^0, A_1^1, \ldots, A_n^0, A_n^1) \tag{G.3}$$

Following the MPS definition in Section 5.2, our claim is proven by showing that for QMDD root node v representing $|v\rangle$, we have

$$\langle \vec{x} | v \rangle = A_n^{x_n} \cdot A_{n-1}^{x_{n-1}} \cdots A_1^{x_1} \qquad \text{for all } \vec{x} \in \{0,1\}^n \tag{G.4}$$

For an n-qubit QMDD v, the amplitude $\langle \vec{x} | v \rangle$ for $\vec{x} \in \{0,1\}^n$ is equal to the product of the weights found on the single path from the root node node to leaf effected by $\vec{x}$ (this path is found as follows: go down from root to leaf; at a vertex at layer j, choose to traverse the low edge if $x_j = 0$ and the high edge if $x_j = 1$). We next reason that this product equals the single entry of the product $y := A_n^{x_n} \cdot A_{n-1}^{x_{n-1}} \cdots A_1^{x_1}$ from Equation G.4.

We recall several useful facts from graph theory. If G (G') is a weighted, directed bipartite graph on the bipartition $M \cup M''$ ($M'' \cup M'$) vertices, with weighted adjacency matrix A_G ($A_{G'}$), then it is not hard to see that the element $(A_G \cdot A_{G'})_{r,c}$ is the sum, over all two-step paths $r - a - c$ starting at vertex $r \in M$ and going through vertex $a \in M''$ to vertex $c \in M'$, of products of the two weights $w_{r\to a}$ and $w_{a\to c}$. More generally, for a sequence of weighted, directed bipartite graphs G_j with vertex set $M_j \cup M_{j+1}$, the (r, c)-th entry of the product of adjacency matrices $A_{G_1} \cdot A_{G_2} \cdot \ldots \cdot A_{G_n}$ equals $\sum_{\text{paths } \pi \text{ from } r \text{ to } c} \prod_{\text{edge}\epsilon\in\pi} \text{weight}(\epsilon)$.

Now note that the matrix y has dimensions 1×1 (since $D_0 = D_n = 1$), corresponding to a single root and single leaf. By the reasoning above, since y is the product of all bipartite adjacency matrices of the QMDD, the single element of this matrix is equal to the product of weights found on the single path from root to leaf as represented by $\vec{x}$. □

Lemma G.3 (MPS to QMDD)**.** There is a weakly minimizing transformation from MPS to QMDD, that runs in time polynomial in the size of the input MPS and the resulting QMDD.

Proof. Algorithm 23 shows the algorithm which converts an MPS to a QMDD. The idea is to use perform backtracking to construct the QMDD bottom-up. Specifically, given an MPS $\{A_n^0, A_n^1, \ldots, A_1^0, A_1^1\}$ representing a state $|\varphi\rangle = |0\rangle |\varphi_0\rangle + |1\rangle |\varphi_1\rangle$, the MPS for $|\varphi_0\rangle$ is easily constructed by setting the first open index to 0 and contracting these two blocks, i.e., $A_{n-1}^0 := A_n^0 \cdot A_{n-1}^0$ and $A_{n-1}^1 := A_n^0 \cdot A_{n-1}^1$, and similarly for $|\varphi_1\rangle$. We then recurse, constructing MPS for states $|\varphi_{00}\rangle, |\varphi_{01}\rangle$, etc. When we find a state whose QMDD node we have already constructed, then we may simply return an edge to that QMDD node without recursing further. This dynamic programming behavior is implemented through the check at Line 3.

Through the use of dynamic programming with the cache set D, it is clear that the number of recursive calls to MPS2QMDD is bound by the number of edges in the resulting QMDD. Dynamic programming is implemented by checking, for each call with MPS M, whether some QMDD node $v \in D$ already represents $|M\rangle$ up to a complex factor. To this end, the subroutine Equivalent, on Line 3, checks whether $|M\rangle = \lambda \cdot |v\rangle$ for some $\lambda \in \mathbb{C}$. It is straightforward to see that it runs in polynomial time in the sizes of QMDD v and MPS M: first, it creates an MPS for the given QMDD node v using the efficient transformation in Section G.4. Next, it computes several inner products on MPS, which can also be done in polynomial time, using the results in App. F. This Equivalent operation is called $|D|$ time, which dominates the runtime of each call MPS2QMDD. Therefore the entire runtime is polynomial in the sizes of the MPS and the resulting QMDD.

Since QMDD is canonical, the transformation is weakly minimizing by definition. ☐

Algorithm 23 An algorithm which converts an MPS into a QMDD. It runs in time polynomial in $s + d$, where s is the size of the QMDD, and d is the bond dimension of the MPS. Here D is the diagram representing the state. The subroutine EQUIVALENT(v, M) computes whether the vectors $|v\rangle, |M\rangle$ are co-linear, i.e., whether there exists $\lambda \in \mathbb{C}$ such that $|M\rangle = \lambda |v\rangle$.

```
 1: $D := \{\boxed{1}\}$          ▷ Initiate diagram $D$ with only a QMDD leaf node representing 1
 2: procedure MPS2QMDD(MPS $M = \{A_j^x\}$)          ▷ Returns a root edge $e_R$ such that $|e_R\rangle = |M\rangle$
 3:     if $D$ contains a node $v$ with $|M\rangle = \lambda |v\rangle$ then return $\overset{\lambda}{\text{—}}\textcircled{v}$          ▷ Implemented with EQUIVALENT$(v, M)$ for all $v \in D$
 4:     EDGE $e_0$ := MPS2QMDD($\{A_n^0 \cdot A_{n-1}^0, A_n^0 \cdot A_{n-1}^1\} \cup \{A_{n-2}^0, A_{n-2}^1, \dots, A_1^0, A_1^1\}$)
 5:     EDGE $e_1$ := MPS2QMDD($\{A_n^1 \cdot A_{n-1}^0, A_n^1 \cdot A_{n-1}^1\} \cup \{A_{n-2}^0, A_{n-2}^1, \dots, A_1^0, A_1^1\}$)
 6:     NODE $w := \bigcirc \overset{e_0}{\cdots\cdots} \bigcirc \overset{e_1}{\text{———}} \bigcirc$          ▷ Create new node $w$ with MAKENODE
 7:     $D := D \cup \{w\}$
 8:     return EDGE $\overset{1}{\text{—}}\textcircled{w}$
 9: procedure EQUIVALENT( QMDD NODE $v$, MPS $M = \{A_j^x\}$)
10:     $V :=$ QMDD2MPS$(v)$          ▷ Using transformation in Section G.4
11:     $s_V := \sqrt{|\langle V|V\rangle|}$          ▷ Compute inner product
12:     $s_M := \sqrt{|\langle M|M\rangle|}$          ▷ Compute inner product
13:     $\lambda := {}^1/_{s_V \cdot s_M} \langle V|M\rangle$          ▷ Compute inner product
14:     if $|\lambda| = 1$ then return "$|M\rangle = \frac{s_M}{s_V} \lambda |v\rangle$"
15:     else return "$|v\rangle$ is not equivalent to $|M\rangle$"
```

G.5 Transformations between QDDs

QDDs are canonical data structures as explained in Section 5.2 and Chapter 2. Therefore, (i) all algorithms are by definition runtime monotonic, as for any state $|\varphi\rangle$ there is only one structure representing it, i.e., D^φ is a singleton set; and (ii) all transformations given below are therefore weakly minimizing since they convert to a canonical data structure (namely, since they map to the unique element in D^φ, in particular they map to the minimum-size element of D^φ).

Algorithm 24 An algorithm which converts a LIMDD into an QMDD.

```
1:  QMDD $D := \{ \xrightarrow{1}\textcircled{1} \}$        ▷ The QMDD is initialized to contain only the Leaf
2:  procedure LIMDD 2QMDD(LIMDD edge $\xrightarrow{\lambda P_n \otimes P'}\textcircled{v}$)
    Returns (a pointer to) an edge to a QMDD node
3:      if $v$ is the Leaf node then return $\xrightarrow{\lambda}\textcircled{v}$
4:      $R :=$ getLexminLabel$(P_n \otimes P', v)$
5:      if the Cache contains tuple $(R, v)$ then return $\lambda \cdot$ Cache$[R, v]$
6:      for $x = 0, 1$ do
7:          QMDD edge $r_x :=$ Follow$_x$($\xrightarrow{P_n \otimes P'}\textcircled{v}$)
8:      QMDD edge $r :=$ MakeEdge$(r_0, r_1)$
9:      Cache$[R, v] := r$
10:     $D := D \cup \{r\}$        ▷ Add the new edge to the diagram
11:     return $\lambda \cdot r$
```

G.5.1 Transforming LIMDD to QMDD

Algorithm 24 converts a LIMDD to a QMDD in time linear in the size of the output. The diagram is the set of edges D, which is initialized to contain the Leaf (i.e., the node $\xrightarrow{1}\textcircled{1}$), and is filled with the other edges during the recursive calls to LIMDD 2QMDD. The function getLexminLabel is taken from Vinkhuijzen et al. [337]; it returns a canonical edge label.

G.5.2 Transforming QMDD to LIMDD

By definition, a QMDD can be seen as a LIMDD in which every edge is labeled with a complex number and the n-qubit identity tensor $\mathbb{I}^{\otimes n}$. Thus, a transformation does not need to do anything. Optionally, it is possible to convert a given LIMDD to one of minimum size, as described by [337].

G.5.3 Transforming ADD to QMDD

To convert an ADD into a QMDD, we add a Leaf node labelled with 1; then, for each Leaf node labelled with $\lambda \neq 1$, we label each incoming edge with λ, and then reroute this edge to the (new) Leaf node labelled with 1. Optionally, the resulting QMDD can be minimized to obtain the canonical instance for this state, using, e.g., techniques from [59, 227].

G.5.4 Transforming QMDD to ADD

Algorithm 25 gives a method which converts an QMDD to an ADD. It is very similar to the ones used in the transformation LIMDD to QMDD above, . We here check whether the diagram already contains a function which is pointwise equal to the one we are currently considering. If so, we reuse that node; otherwise, we recurse.

Algorithm 25 An algorithm which converts an QMDD to an ADD. Its input is an QMDD edge e representing a state $|e\rangle$ on n qubits. Here the method $\textsc{Follow}_x(e)$ returns an QMDD edge representing the state $\langle x| \otimes \mathbb{I}^{\otimes n-1} \cdot |e\rangle$. It outputs an QMDD node w representing $|w\rangle = |e\rangle$.

```
procedure QMDD 2ADD(QMDD edge $e = \xrightarrow{\lambda}\textcircled{v}$ on $n$ qubits)
    if $n = 0$ then
        $w := -\textcircled{\lambda}$
    else if $A$ contains a node $w$ with $|v\rangle = |w\rangle$ then
        return $w$
    else
        for $x = 0, 1$ do
            $w_x :=$ SLDD2ADD($\textsc{Follow}_x(e)$)
        QMDD Node $w := \textcircled{w_0} \cdots \bigcirc - \textcircled{w_1}$
    $A := A \cup \{w\}$
    return $w$
```

Bibliography

[1] Scott Aaronson. Multilinear formulas and skepticism of quantum computing. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, page 118–127, New York, NY, USA, 2004. Association for Computing Machinery.

[2] Scott Aaronson. *Quantum computing since Democritus.* Cambridge University Press, 2013.

[3] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), nov 2004.

[4] Afshin Abdollahi and Massoud Pedram. Analysis and synthesis of quantum circuits by using quantum decision diagrams. In *Design, Automation and Test in Europe*, pages 317–322, 2006.

[5] Farid Ablayev, Aida Gainutdinova, and Marek Karpinski. On computational power of quantum branching programs. In *Fundamentals of Computation Theory: 13th International Symposium, FCT 2001 Riga, Latvia, August 22–24, 2001 Proceedings 13*, pages 59–70. Springer, 2001.

[6] Smaran Adarsh, Lukas Burgholzer, Tanmay Manjunath, and Robert Wille. SyReC synthesizer: An MQT tool for synthesis of reversible circuits. *Software Impacts*, 14:100451, 2022.

[7] Sheldon B. Akers. Binary decision diagrams. *IEEE Computer Architecture Letters*, 27(06):509–516, 1978.

[8] Anas N. Al-Rabadi, Marek Perkowski, and Martin Zwick. A comparison of modified reconstructability analysis and Ashenhurst-Curtis decomposition of Boolean functions. *Kybernetes*, 2004.

[9] F Aloul, I Markov, and K Sakallah. Mince: A static global variable-ordering for SAT and BDD. In *International Workshop on Logic and Synthesis*, pages 1167–1172, 2001.

[10] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 202–211, New York, NY, USA, 2004. Association for Computing Machinery.

[11] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. BDS-MAJ: A BDD-based logic synthesis tool exploiting majority logic decomposition. In *Proceedings of the 50th Annual Design Automation Conference*, pages 1–6, 2013.

[12] Andris Ambainis, András Gilyén, Stacey Jeffery, and Martins Kokainis. Quadratic speedup for finding marked vertices by quantum walks. In *Symp. on Theory of Computing*, pages 412–424, 2020.

[13] Matthew Amy. *Formal methods in quantum circuit design.* PhD thesis, 2019.

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

[15] Matthew Amy, Martin Roetteler, and Krysta M Svore. Verified compilation of space-efficient reversible circuits. In *International Conference on Computer Aided Verification*, pages 3–21. Springer, 2017.

[16] Linda Anticoli, Carla Piazza, Leonardo Taglialegne, and Paolo Zuliani. Towards quantum programs verification: from quipper circuits to QPMC. In *Reversible Computation: 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings 8*, pages 213–219. Springer, 2016.

[17] Ebrahim Ardeshir-Larijani, Simon J Gay, and Rajagopal Nagarajan. Equivalence checking of quantum protocols. In *Tools and Algorithms for the Construction and Analysis of Systems: 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 19*, pages 478–492. Springer, 2013.

[18] Ebrahim Ardeshir-Larijani, Simon J Gay, and Rajagopal Nagarajan. Verification of concurrent quantum protocols by equivalence checking. In *TACAS*, pages 500–514. Springer, 2014.

[19] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[20] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[21] Srinivasan Arunachalam, Sergey Bravyi, Chinmay Nirkhe, and Bryan O'Gorman. The parameterized complexity of quantum verification. *arXiv preprint arXiv:2202.08119*, 2022.

[22] Robert L. Ashenhurst. The decomposition of switching functions. In *Proceedings of an International Symposium on the theory of Switching, April 1957*, 1957.

[23] Gilles Audemard, Frédéric Koriche, and Pierre Marquis. On tractable XAI queries based on compiled representations. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 17, pages 838–849, 2020.

[24] Koenraad M R Audenaert and Martin B Plenio. Entanglement on mixed stabilizer states: normal forms and reduction procedures. *New Journal of Physics*, 7(1):170, 2005.

[25] UCLA Automated Reasoning Group. The SDD package. http://reasoning.cs.ucla.edu/sdd/, 2018.

[26] Junaid Babar, Chuan Jiang, Gianfranco Ciardo, and Andrew Miner. Binary decision diagrams with edge-specified reductions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 303–318. Springer, 2019.

[27] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 188–191, 1993.

[28] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[29] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Comput. Surv.*, 51(3), 2018.

[30] Pedro Baltazar, Rohit Chadha, and Paulo Mateus. Quantum computation tree logic—model checking and complete calculus. *International Journal of Quantum Information*, 6(02):219–236, 2008.

[31] Zuzana Baranová, Jiří Barnat, Katarína Kejstová, Tadeáš Kučera, Henrich Lauko, Jan Mrázek, Petr Ročkai, and Vladimír Štill. Model checking of C and C++ with DIVINE 4. In *ATVA 2017*, volume 10482 of *LNCS*, pages 201–207. Springer, 2017.

[32] Andreas Bärtschi and Stephan Eidenbenz. Deterministic preparation of Dicke states. In *Fundamentals of Computation Theory: 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12-14, 2019, Proceedings 22*, pages 126–139. Springer, 2019.

[33] Jon Barwise. *Handbook of mathematical logic.* Elsevier, 1982.

[34] Fabian Bauer-Marquart, Stefan Leue, and Christian Schilling. symQV: Automated symbolic verification of quantum programs. In *Formal Methods: 25th International Symposium, FM 2023, Lübeck, Germany, March 6–10, 2023, Proceedings*, pages 181–198. Springer, 2023.

[35] Stephane Beauregard. Circuit for Shor's algorithm using $2n + 3$ qubits. *arXiv preprint quant-ph/0205095*, 2002.

[36] John S Bell. On the Einstein Podolsky Rosen paradox. *Physics Physique Fizika*, 1(3):195, 1964.

[37] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, nov 2019.

[38] Charles H. Bennet. Quantum cryptography: public key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, Dec. 1984*, pages 175–179, 1984.

[39] Charles H. Bennett, Herbert J. Bernstein, Sandu Popescu, and Benjamin Schumacher. Concentrating partial entanglement by local operations. *Physical Review A*, 53(4):2046, 1996.

[40] Charles H. Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K. Wootters. Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. *Physical Review Letters*, 70(13):1895, 1993.

[41] Charles H. Bennett, David P. DiVincenzo, John A. Smolin, and William K Wootters. Mixed-state entanglement and quantum error correction. *Physical Review A*, 54(5):3824, 1996.

[42] Charles H. Bennett and Stephen J. Wiesner. Communication via one-and two-particle operators on Einstein-Podolsky-Rosen states. *Physical Review Letters*, 69(20):2881, 1992.

[43] Lucas Berent, Lukas Burgholzer, and Robert Wille. Towards a SAT encoding for quantum circuits: A journey from classical circuits to Clifford circuits and beyond. *arXiv:2203.00698*, 2022.

[44] David Bergman, Andre A Cire, Willem-Jan van Hoeve, and John N Hooker. Discrete optimization with decision diagrams. *INFORMS Journal on Computing*, 28(1):47–66, 2016.

[45] Valeria Bertacco. The disjunctive decomposition of logic functions. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD'97), November 1997*, pages 78–82, 1997.

[46] Valeria Bertacco and Maurizio Damiani. Boolean function representation based on disjoint-support decompositions. In *Proceedings International Conference on Computer Design. VLSI in Computers and Processors*, pages 27–32. IEEE, 1996.

[47] Yves Bertot and Pierre Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[48] Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S Kottmann, Tim Menke, et al. Noisy intermediate-scale quantum algorithms. *Reviews of Modern Physics*, 94(1):015004, 2022.

[49] Debjyoti Bhattacharjee and Anupam Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies. *arXiv preprint arXiv:1703.08540*, 2017.

[50] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 893–902. SIAM, 2016.

[51] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 193–207. Springer, 1999.

[52] Antonio Blanca, Zongchen Chen, Daniel Štefankovič, and Eric Vigoda. Hardness of identity testing for restricted Boltzmann machines and Potts models. *The Journal of Machine Learning Research*, 22(1):6727–6782, 2021.

[53] Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

[54] Beate Bollig and Matthias Buttkus. On the relative succinctness of sentential decision diagrams. *Theory of Computing Systems*, 63(6):1250–1277, 2019.

[55] Beate Bollig and Martin Farenholtz. On the relation between structured d-DNNFs and SDDs. *Theory of Computing Systems*, 65(2):274–295, 2021.

[56] Beate Bollig and Ingo Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, 45:993–1002, 1996.

[57] Adi Botea, Akihiro Kishimoto, and Radu Marinescu. On the complexity of quantum circuit compilation. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, pages 138–142, 2018.

[58] Simone Bova. SDDs are exponentially more succinct than OBDDs. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[59] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a BDD package. In *27th ACM/IEEE Design Automation Conference*, pages 40–45. IEEE, 1990.

[60] Aaron R. Bradley. SAT-based model checking without unrolling. In *Verification, Model Checking, and Abstract Interpretation: 12th International Conference, VMCAI 2011, Austin, Texas, USA, January 23-25, 2011. Proceedings 12*, pages 70–87. Springer, 2011.

[61] Sergey Bravyi, Dan Browne, Padraic Calpin, Earl Campbell, David Gosset, and Mark Howard. Simulation of quantum circuits by low-rank stabilizer decompositions. *Quantum*, 3:181, September 2019.

[62] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Physical Review Letters*, 116:250501, Jun 2016.

[63] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71:022316, Feb 2005.

[64] Sergey Bravyi, Graeme Smith, and John A. Smolin. Trading classical and quantum computational resources. *Physical Review X*, 6:021043, Jun 2016.

[65] Hans J. Briegel and Robert Raussendorf. Persistent entanglement in arrays of interacting particles. *Physical Review Letters*, 86:910–913, Jan 2001.

[66] Dan Browne and Hans Briegel. One-way quantum computation. *Quantum information: From foundations to quantum technology applications*, pages 449–473, 2016.

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

[68] Randal E Bryant. Chain reduction for binary and zero-suppressed decision diagrams. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 81–98. Springer, 2018.

[69] Yirng-An Chen Randal E Bryant. Verification of arithmetic circuits with binary moment diagrams. In *32nd Design Automation Conference*, pages 535–541. IEEE, 1995.

[70] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

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

[72] Lukas Burgholzer, Alexander Ploier, and Robert Wille. Tensor networks or decision diagrams? Guidelines for classical quantum circuit simulation. *arXiv preprint arXiv:2302.06616*, 2023.

[73] Lukas Burgholzer, Rudy Raymond, and Robert Wille. Verifying results of the IBM Qiskit quantum circuit compilation flow. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 356–365. IEEE, 2020.

[74] Lukas Burgholzer and Robert Wille. Improved DD-based equivalence checking of quantum circuits. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 127–132. IEEE, 2020.

[75] Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Transactions on CAD of Integrated Circuits and Systems*, 40(9):1810–1824, 2021.

[76] Padraic Calpin. *Exploring Quantum Computation Through the Lens of Classical Simulation*. PhD thesis, UCL (University College London), 2020.

[77] Jacques Carette, Gerardo Ortiz, and Amr Sabry. Symbolic execution of Hadamard-Toffoli quantum circuits. In *Proceedings of the 2023 ACM SIGPLAN International Workshop on Partial Evaluation and Program Manipulation*, pages 14–26, 2023.

[78] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, 2017.

[79] Rohit Chadha, Paulo Mateus, and Amílcar Sernadas. Reasoning about imperative quantum programs. *Electronic Notes in Theoretical Computer Science*, 158:19–39, 2006.

[80] Jing Chen, Song Cheng, Haidong Xie, Lei Wang, and Tao Xiang. Equivalence of restricted Boltzmann machines and tensor network states. *Physical Review B*, 97(8):085104, 2018.

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

[82] Eric Chitambar, Debbie Leung, Laura Mančinska, Maris Ozols, and Andreas Winter. Everything you always wanted to know about LOCC (but were afraid to ask). *Communications in Mathematical Physics*, 328(1):303–326, 2014.

[83] Arthur Choi and Adnan Darwiche. Dynamic minimization of sentential decision diagrams. In *27th AAAI Conference on Artificial Intelligence*, 2013.

[84] E. M. Clarke, K. L. McMillan, X Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, pages 54–60, New York, NY, USA, 1993. Association for Computing Machinery.

[85] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[86] Edmund M Clarke, M. Fujita, P C McGeer, K. McMillan, J C-Y Yang, and X Zhao. Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. 2 2001.

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

[88] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

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

[90] Elizabeth Cuthill and James McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th national conference*, pages 157–172. ACM, 1969.

[91] Giso H. Dal, Alfons W Laarman, Arjen Hommersom, and Peter JF Lucas. A compositional approach to probabilistic knowledge compilation. *International Journal of Approximate Reasoning*, 138:38–66, 2021.

[92] Maurizio Damiani and Valeria Bertacco. Finding complex disjunctive decompositions of logic functions. In *Proc of the International Workshop on Logic & Synthesis*, pages 478–483, 1998.

[93] Aidan Dang, Charles D. Hill, and Lloyd C. L. Hollenberg. Optimising Matrix Product State Simulations of Shor's Algorithm. *Quantum*, 3:116, January 2019.

[94] Adnan Darwiche. Compiling knowledge into decomposable negation normal form. In *IJCAI*, volume 99, pages 284–289. Citeseer, 1999.

[95] Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM (JACM)*, 48(4):608–647, 2001.

[96] Adnan Darwiche. SDD: a new canonical representation of propositional knowledge bases. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 819–826. AAAI Press, 2011.

[97] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[98] JW De Bakker and Lambert G. L. T. Meertens. On the completeness of the inductive assertion method. *Journal of Computer and System Sciences*, 11(3):323–357, 1975.

[99] Luc De Raedt, Kristian Kersting, Angelika Kimmig, Kate Revoredo, and Hannu Toivonen. Compressing probabilistic prolog programs. *Machine learning*, 70:151–168, 2008.

[100] Ellie D'hondt and Prakash Panangaden. Quantum weakest preconditions. *Mathematical Structures in Computer Science*, 16(3):429–451, 2006.

[101] Robert H Dicke. Coherence in spontaneous radiation processes. *Physical Review*, 93(1):99, 1954.

[102] Edsger W Dijkstra. The humble programmer. *Communications of the ACM*, 15(10):859–866, 1972.

[103] Rolf Drechsler, Andisheh Sarabi, Michael Theobald, Bernd Becker, and Marek A Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Proceedings of the 31st annual Design Automation Conference*, pages 415–419, 1994.

[104] Vincent Dumoulin, Ian Goodfellow, Aaron Courville, and Yoshua Bengio. On the challenges of physical implementations of RBMs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), Jun. 2014.

[105] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic Simplification of Quantum Circuits with the ZX-calculus. *Quantum*, 4:279, June 2020.

[106] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

[107] Wolfgang Dür, Guifre Vidal, and J Ignacio Cirac. Three qubits can be entangled in two inequivalent ways. *Physical Review A*, 62(6):062314, 2000.

[108] Pavol Ďuriš, Juraj Hromkovič, Stasys Jukna, Martin Sauerhoff, and Georg Schnitger. On multi-partition communication complexity. *Information and computation*, 194(1):49–75, 2004.

[109] Matthias Englbrecht and Barbara Kraus. Symmetries and entanglement of stabilizer states. *Physical Review A*, 101:062302, Jun 2020.

[110] Glen Evenbly and Guifré Vidal. Tensor network states and geometry. *Journal of Statistical Physics*, 145:891–918, 2011.

[111] Liangda Fang, Biqing Fang, Hai Wan, Zeqi Zheng, Liang Chang, and Quan Yu. Tagged sentential decision diagrams: Combining standard and zero-suppressed compression and trimming rules. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[112] Hélene Fargier and Pierre Marquis. Extending the knowledge compilation map: Krom, horn, affine and beyond. In *AAAI*, pages 442–447, 2008.

[113] Hélene Fargier, Pierre Marquis, Alexandre Niveau, and Nicolas Schmidt. A knowledge compilation map for ordered real-valued decision diagrams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[114] Hélene Fargier, Pierre Marquis, and Nicolas Schmidt. Semiring labelled decision diagrams, revisited: Canonicity and spatial efficiency issues. In *IJCAI*, pages 884–890, 2013.

[115] David Y Feinstein and Mitchell A Thornton. On the skipped variables of quantum multiple-valued decision diagrams. In *2011 41st IEEE International Symposium on Multiple-Valued Logic*, pages 164–169. IEEE, 2011.

[116] Yuan Feng, Ernst Moritz Hahn, Andrea Turrini, and Lijun Zhang. QPMC: A model checker for quantum programs and protocols. In *FM 2015: Formal Methods: 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings 20*, pages 265–272. Springer, 2015.

[117] Yuan Feng, Nengkun Yu, and Mingsheng Ying. Model checking quantum Markov chains. *Journal of Computer and System Sciences*, 79(7):1181–1198, 2013.

[118] Felipe Cavalcanti Ferreira. *An Exploratory Study on the Usage of Quantum Programming Languages*. PhD thesis, 2022.

[119] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[120] Richard P. Feynman. Quantum mechanical computers. *Optics news*, 11(2):11–20, 1985.

[121] Robert W Floyd. Assigning meanings to programs. In *Program Verification*, pages 65–81. Springer, 1993.

[122] WMC Foulkes, Lubos Mitas, RJ Needs, and Guna Rajagopal. Quantum Monte Carlo simulations of solids. *Reviews of Modern Physics*, 73(1):33, 2001.

[123] Lars-Hendrik Frahm and Daniela Pfannkuche. Ultrafast ab initio quantum chemistry using matrix product states. *Journal of Chemical Theory and Computation*, 15(4):2154–2165, 2019.

[124] Masahiro Fujita, Patrick C. McGeer, and JC-Y Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2-3):149–169, 1997.

[125] Hector J Garcia, Igor L Markov, and Andrew W Cross. Efficient inner-product algorithm for stabilizer states. *arXiv preprint arXiv:1210.6646*, 2012.

[126] Sunita Garhwal, Maryam Ghorani, and Amir Ahmad. Quantum programming language: A systematic review of research topic and top cited languages. *Archives of Computational Methods in Engineering*, 28:289–310, 2021.

[127] Simon J Gay and Rajagopal Nagarajan. Communicating quantum processes. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages*, pages 145–157, 2005.

[128] Simon J Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou. QMC: A model checker for quantum systems: Tool paper. In *Computer Aided Verification: 20th International Conference, CAV 2008 Princeton, NJ, USA, July 7-14, 2008 Proceedings 20*, pages 543–547. Springer, 2008.

[129] Jordan Gergov and Christoph Meinel. Efficient boolean manipulation with obdd's can be extended to fbdd's. *IEEE Transactions on Computers*, 43(10):1197–1209, 1994.

[130] Jordan Gergov and Christoph Meinel. Mod-2-OBDDs—a data structure that generalizes exor-sum-of-products and ordered binary decision diagrams. *Formal Methods in System Design*, 8:273–282, 1996.

[131] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden. Quantum cryptography. *Reviews of Modern Physics*, 74(1):145, 2002.

[132] Ivan Glasser, Nicola Pancotti, Moritz August, Ivan D Rodriguez, and J Ignacio Cirac. Neural-network quantum states, string-bond states, and chiral topological states. *Physical Review X*, 8(1):011006, 2018.

[133] Ivan Glasser, Ryan Sweke, Nicola Pancotti, Jens Eisert, and Ignacio Cirac. Expressive power of tensor-network factorizations for probabilistic modeling. *Advances in Neural Information Processing Systems*, 32, 2019.

[134] Patrice Godefroid. Using partial orders to improve automatic verification methods. In *Computer-Aided Verification: 2nd International Conference, CAV'90 New Brunswick, NJ, USA, June 18–21, 1990 Proceedings 2*, pages 176–185. Springer, 1991.

[135] Daniel Gottesman. *Stabilizer codes and quantum error correction.* PhD thesis, California Institute of Technology, 1997.

[136] Daniel Gottesman. The Heisenberg representation of quantum computers. In *Proc. XXII International Colloquium on Group Theoretical Methods in Physics, 1998*, pages 32–43, 1998.

[137] Daniel Gottesman. Theory of fault-tolerant quantum computation. *Physical Review A*, 57:127–137, 1998.

[138] Daniel M Greenberger, Michael A Horne, Abner Shimony, and Anton Zeilinger. Bell's theorem without inequalities. *American Journal of Physics*, 58(12):1131–1143, 1990.

[139] David J Griffiths and Darrell F Schroeter. *Introduction to quantum mechanics.* Cambridge university press, 2018.

[140] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Symp. on Theory of Computing*, pages 212–219, 1996.

[141] Thomas Grurl, Jürgen Fuß, and Robert Wille. Considering decoherence errors in the simulation of quantum circuits using decision diagrams. In *Proceedings of the 39th International Conference on Computer-Aided Design*, pages 1–7, 2020.

[142] Thomas Grurl, Jürgen Fuß, and Robert Wille. Lessons learnt in the implementation of quantum circuit simulation using decision diagrams. In *2021 IEEE 51st International Symposium on Multiple-Valued Logic (ISMVL)*, pages 87–92. IEEE, 2021.

[143] Thomas Grurl, Jürgen Fuß, and Robert Wille. Noise-aware quantum circuit simulation with decision diagrams. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[144] Thomas Grurl, Richard Kueng, Jürgen Fuß, and Robert Wille. Stochastic quantum circuit simulation using decision diagrams. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 194–199. IEEE, 2021.

[145] G. G. Guerreschi and A. Y. Matsuura. QAOA for max-cut requires hundreds of qubits for quantum speed-up. *Scientific Reports*, 9(1):6903, 2019.

[146] Wolfgang Günther and Rolf Drechsler. BDD minimization by linear transformations. In *In Advanced Computer Systems.* University Szczecin, 1998.

[147] Ernst Moritz Hahn, Yi Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. ISCAS MC: a web-based probabilistic model checker. In *FM 2014: Formal Methods: 19th International Symposium, Singapore, May 12-16, 2014. Proceedings 19*, pages 312–317. Springer, 2014.

[148] Sean Hallgren. Fast quantum algorithms for computing the unit group and class group of a number field. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 468–474, 2005.

[149] Sean Hallgren. Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. *Journal of the ACM (JACM)*, 54(1):1–19, 2007.

[150] Thomas Häner and Damian S. Steiger. 5 petabyte simulation of a 45-qubit quantum circuit. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–10, 2017.

[151] Klaus Havelund, Mike Lowry, and John Penix. Formal analysis of a space-craft controller using spin. *IEEE Transactions on Software Engineering*, 27(8):749–765, 2001.

[152] Martin Hebenstreit, Richard Jozsa, Barbara Kraus, and Sergii Strelchuk. Computational power of matchgates with supplementary resources. *Physical Review A*, 102(5):052604, 2020.

[153] Marc Hein, Wolfgang Dür, Jens Eisert, Robert Raussendorf, M Nest, and H-J Briegel. Entanglement in graph states and its applications. In *Proceedings of the International School of Physics "Enrico Fermi"*, volume 162: Quantum Computers, Algorithms and Chaos. IOS Press, 2006.

[154] Marc Herbstritt. wld: A C++ library for decision diagrams. https://ira.informatik.uni-freiburg.de/software/wld/, 2004.

[155] Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. A verified optimizer for quantum circuits. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–29, 2021.

[156] Stefan Hillmich, Lukas Burgholzer, Florian Stögmüller, and Robert Wille. Reordering decision diagrams for quantum computing is harder than you might think. In *Reversible Computation: 14th International Conference, RC 2022, Urbino, Italy, July 5–6, 2022, Proceedings*, pages 93–107. Springer, 2022.

[157] Stefan Hillmich, Richard Kueng, Igor L. Markov, and Robert Wille. As accurate as needed, as efficient as possible: Approximations in DD-based quantum circuit simulation. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 188–193. IEEE, 2021.

[158] Stefan Hillmich, Igor L. Markov, and Robert Wille. Just like the real thing: Fast weak simulation of quantum computation. In *Design Automation Conference*, pages 1–6. IEEE, 2020.

[159] Gerard J. Holzmann. The model checker SPIN. *IEEE TSE*, 23:279–295, 1997.

[160] Gerard J. Holzmann, Eli Najm, and Ahmed Serhrouchni. SPIN model checking: An introduction. *International Journal on Software Tools for Technology Transfer*, 2:321–327, 2000.

[161] Shahin Honarvar, Mohammad Reza Mousavi, and Rajagopal Nagarajan. Property-based testing of quantum programs in Q#. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, pages 430–435, 2020.

[162] Xin Hong, Mingsheng Ying, Yuan Feng, Xiangzhen Zhou, and Sanjiang Li. Approximate equivalence checking of noisy quantum circuits. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 637–642, 2021.

[163] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying. A tensor network based decision diagram for representation of quantum circuits. *ACM Trans. Des. Autom. Electron. Syst.*, 27(6), 2022.

[164] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.

[165] Yifei Huang and Peter Love. Approximate stabilizer rank and improved weak simulation of Clifford-dominated circuits for qudits. *Physical Review A*, 99:052307, May 2019.

[166] Yipeng Huang and Margaret Martonosi. QDB: from quantum algorithms towards correct quantum programs. *arXiv preprint arXiv:1811.05447*, 2018.

[167] Yipeng Huang and Margaret Martonosi. Statistical assertions for validating patterns and finding bugs in quantum programs. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 541–553, 2019.

[168] Vít Jelínek. The rank-width of the square grid. *Discrete Applied Mathematics*, 158(7):841–850, 2010.

[169] Mark Jerrum and Kitty Meeks. The parameterised complexity of counting even and odd induced subgraphs. *Combinatorica*, 37(5):965–990, 2017.

[170] Wang Jian, Zhang Quan, and Tang Chao-Jing. Quantum secure communication scheme with W state. *Communications in Theoretical Physics*, 48(4):637, 2007.

[171] Phillip Johnston and Rozi Harris. The boeing 737 max saga: lessons for software organizations. *Software Quality Professional*, 21(3):4–12, 2019.

[172] N Cody Jones, James D Whitfield, Peter L McMahon, Man-Hong Yung, Rodney Van Meter, Alán Aspuru-Guzik, and Yoshihisa Yamamoto. Faster quantum chemistry simulation on fault-tolerant quantum computers. *New Journal of Physics*, 14(11):115023, 2012.

[173] Tyson Jones, Anna Brown, Ian Bush, and Simon C Benjamin. Quest and high performance simulation of quantum computers. *Scientific reports*, 9(1):1–11, 2019.

[174] Bjarni Jónsson, Bela Bauer, and Giuseppe Carleo. Neural-network states for the classical simulation of quantum computing. *arXiv preprint arXiv:1808.05232*, 2018.

[175] Stephen Jordan. Quantum algorithm zoo. https://quantumalgorithmzoo.org/. Accessed: 20-05-2023.

[176] Richard Jozsa. Quantum algorithms and the Fourier transform. *Royal Society of London. Series A*, 454(1969):323–337, 1998.

[177] Richard Jozsa and Akimasa Miyake. Matchgates and classical simulation of quantum circuits. *Proceedings: Mathematical, Physical and Engineering Sciences*, pages 3089–3106, 2008.

[178] Gijs Kant et al. LTSmin: High-performance language-independent model checking. In *Tool and Algorithms for the Construction and Analysis of Systems (TACAS), TACAS'15*, volume 9035 of *LNCS*, pages 692–707. Springer, 2015.

[179] Pawel Kerntopf. A new heuristic algorithm for reversible logic synthesis. In *Proceedings of the 41st annual Design Automation Conference*, pages 834–837, 2004.

[180] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

[181] Aleks Kissinger and John van de Wetering. Reducing T-count with the ZX-calculus. *arXiv preprint arXiv:1903.10477*, 2019.

[182] Alexei Yu Kitaev, Alexander Shen, and Mikhail N Vyalyi. *Classical and quantum computation*. Number 47. American Mathematical Soc., 2002.

[183] Donald E Knuth. *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional, 2009.

[184] Lucas Kocia and Peter Love. Stationary phase method in discrete wigner functions and classical simulation of quantum circuits. *Quantum*, 5:494, 2021.

[185] Lucas Kocia and Mohan Sarovar. Improved simulation of quantum circuits by fewer gaussian eliminations. *arXiv:2003.01130*, 2020.

[186] Attila Kondacs and John Watrous. On the power of quantum finite state automata. In *Proceedings 38th annual symposium on foundations of computer science*, pages 66–75. IEEE, 1997.

[187] Fabrice Kordon, Hubert Garavel, Lom-Messan Hillah, Emmanuel Paviot-Adet, Loïg Jezequel, Francis Hulin-Hubard, Elvio Gilberto Amparore, Marco Beccuti, Bernard Berthomieu, Hugues Evrard, Peter Gjøl Jensen, Didier Le Botlan, Torsten Liebke, Jeroen Meijer, Jirí Srba, Yann Thierry-Mieg, Jaco van de Pol, and Karsten Wolf. Mcc'2017 - the seventh model checking contest. *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, XIII:181–209, 2018.

[188] Fabrice Kordon, Hubert Garavel, Lom-Messan Hillah, Emmanuel Paviot-Adet, Loïg Jezequel, César Rodríguez, and Francis Hulin-Hubard. MCC'2015 - The Fifth Model Checking Contest. *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC)*, XI:262–273, 2016.

[189] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical computer science*, 27(3):333–354, 1983.

[190] Richard Kueng and David Gross. Qubit stabilizer states are complex projective 3-designs. *arXiv preprint arXiv:1510.02767*, 2015.

[191] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 585–591. Springer, 2011.

[192] Jean-Marie Lagniez and Pierre Marquis. An improved decision-dnnf compiler. In *IJCAI*, volume 17, pages 667–673, 2017.

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

[194] Yong Lai, Dayou Liu, and Minghao Yin. New canonical representations by augmenting OBDDs with conjunctive decomposition. *Journal of Artificial Intelligence Research*, 58:453–521, 2017.

[195] Yong Lai, Kuldeep S. Meel, and Roland H.C. Yap. CCDD: A tractable representation for model counting and uniform sampling. *arXiv preprint arXiv:2202.10025*, 2022.

[196] Benjamin P Lanyon, James D Whitfield, Geoff G Gillett, Michael E Goggin, Marcelo P Almeida, Ivan Kassal, Jacob D Biamonte, Masoud Mohseni, Ben J Powell, Marco Barbieri, et al. Towards quantum chemistry on a quantum computer. *Nature Chemistry*, 2(2):106, 2010.

[197] Anna LD Latour, Behrouz Babaki, Anton Dries, Angelika Kimmig, Guy Van den Broeck, and Siegfried Nijssen. Combining stochastic constraint optimization and probabilistic programming: from knowledge compilation to constraint solving. In *Principles and Practice of Constraint Programming: 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28–September 1, 2017, Proceedings 23*, pages 495–511. Springer, 2017.

[198] Anna Louise D Latour, Behrouz Babaki, and Siegfried Nijssen. Stochastic constraint propagation for mining probabilistic networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 1137–1145, 2019.

[199] Chang-Yeong Lee. Representation of switching circuits by binary-decision programs. *The Bell System Technical Journal*, 38(4):985–999, 1959.

[200] Nancy G Leveson and Clark S Turner. An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[201] Daniel A Lidar and Haobin Wang. Calculating the thermal rate constant with exponential speedup on a quantum computer. *Physical Review E*, 59(2):2429, 1999.

[202] Jacques-Louis Lions, Lennart Luebeck, Jean-Luc Fauquembergue, Gilles Kahn, Wolfgang Kubbat, Stefan Levedag, Leonardo Mazzini, Didier Merle, and Colin O'Halloran. Ariane 5 flight 501 failure report by the inquiry board, 1996.

[203] Victoria Lipinska, Gláucia Murta, and Stephanie Wehner. Anonymous transmission in a noisy quantum network using the W state. *Physical Review A*, 98:052320, Nov 2018.

[204] Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979.

[205] Junyi Liu, Bohua Zhan, Shuling Wang, Shenggang Ying, Tao Liu, Yangjia Li, Mingsheng Ying, and Naijun Zhan. Formal verification of quantum algorithms using quantum Hoare logic. In *Computer Aided Verification: 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II 31*, pages 187–207. Springer, 2019.

[206] Wen Liu, Yong-Bin Wang, and Zheng-Tao Jiang. An efficient protocol for the quantum private comparison of equality with W state. *Optics Communications*, 284(12):3160–3163, 2011.

[207] Benjamin Livshits, Manu Sridharan, Yannis Smaragdakis, Ondřej Lhoták, J Nelson Amaral, Bor-Yuh Evan Chang, Samuel Z Guyer, Uday P Khedker, Anders Møller, and Dimitrios Vardoulakis. In defense of soundiness: A manifesto. *Communications of the ACM*, 58(2):44–46, 2015.

[208] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–1078, 1996.

[209] Benjamin Lovitz and Vincent Steffan. New techniques for bounding stabilizer rank. *Quantum*, 6:692, 2022.

[210] LTSmin with sentential decision diagrams. https://zenodo.org/record/3940936.

[211] Chin-Yung Lu, Shiou-An Wang, and Sy-Yen Kuo. An extended XQDD representation for multiple-valued quantum logic. *IEEE Transactions on Computers*, 60(10):1377–1389, 2011.

[212] Eugene M Luks, Ferenc Rákóczi, and Charles RB Wright. Some algorithms for nilpotent permutation groups. *Journal of Symbolic Computation*, 23(4):335–354, 1997.

[213] Guanfeng Lv, Yao Chen, Yachao Feng, Qingliang Chen, and Kaile Su. A succinct and efficient implementation of a 2^{32} BDD package. In Tiziana Margaria, Zongyan Qiu, and Hongli Yang, editors, *Int'l Symp. on Theoretical Aspects of Software Engineering*, pages 241–244, 2012.

[214] James Martens, Arkadev Chattopadhya, Toni Pitassi, and Richard Zemel. On the representational efficiency of restricted Boltzmann machines. *Advances in Neural Information Processing Systems*, 26, 2013.

[215] Robert Mateescu, Rina Dechter, and Radu Marinescu. And/or multi-valued decision diagrams (AOMDDs) for graphical models. *Journal of Artificial Intelligence Research*, 33:465–519, 2008.

[216] Paulo Mateus, Jaime Ramos, Amílcar Sernadas, and Cristina Sernadas. Temporal logics for reasoning about quantum systems. *Semantic techniques in quantum computation*, pages 389–413, 2009.

[217] Paulo Mateus and Amílcar Sernadas. Weakly complete axiomatization of exogenous quantum propositional logic. *Information and Computation*, 204(5):771–794, 2006.

[218] Yusuke Matsunaga. An exact and efficient algorithm for disjunctive decomposition. *Proceedings of Synthesis and System Integration of Mixed Technologies (SASIMI'98, Japan), Oct.*, 1998.

[219] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. Quantum computational chemistry. *Reviews of Modern Physics*, 92(1):015003, 2020.

[220] James McClung. *Constructions and Applications of W-States.* PhD thesis, Worcester Polytechnic Institute, 2020.

[221] K.L. McMillan. *Symbolic model checking: an approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, 1992. UMI No. GAX92-24209.

[222] Saeed Mehraban and Mehrdad Tahmasbi. Lower bounds on the approximate stabilizer rank: A probabilistic approach. *arXiv preprint arXiv:2305.10277*, 2023.

[223] Jeroen Meijer, Gijs Kant, Stefan Blom, and Jaco van de Pol. Read, write and copy dependencies for symbolic model checking. In *Haifa Verification Conference*, pages 204–219. Springer, 2014.

[224] Jeroen Meijer and Jaco van de Pol. Bandwidth and wavefront reduction for static variable ordering in symbolic reachability analysis. In *NASA Formal Methods Symposium*, pages 255–271. Springer, 2016.

[225] Roger G Melko, Giuseppe Carleo, Juan Carrasquilla, and J Ignacio Cirac. Restricted Boltzmann machines in quantum physics. *Nature Physics*, 15(9):887–892, 2019.

[226] D Michael Miller, David Y Feinstein, and Mitchell A Thrornton. QMDD minimization using sifting for variable reordering. *Journal of Multiple-Valued Logic & Soft Computing*, 13, 2007.

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

[228] D Michael Miller, Robert Wille, and Zahra Sasanian. Elementary quantum gate realizations for multiple-control toffoli gates. In *2011 41st IEEE International Symposium on Multiple-Valued Logic*, pages 288–293. IEEE, 2011.

[229] Shin-ichi Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *30th ACM/IEEE Design Automation Conference*, pages 272–277. IEEE, 1993.

[230] Shin-ichi Minato. Finding simple disjoint decompositions in frequent itemset data using zero-suppressed BDD. In *Proc. of IEEE ICDM 2005 workshop on Computational Intelligence in Data Mining*, pages 3–11, 2005.

[231] Ashley Montanaro. Quantum-walk speedup of backtracking algorithms. *Theory of Computing*, 14(1):1–24, 2018.

[232] Tomoyuki Morimae and Suguru Tamaki. Fine-grained quantum computational supremacy. *arXiv preprint arXiv:1901.01637*, 2019.

[233] MQT DDSIM - A quantum circuit simulator based on decision diagrams written in C++. https://github.com/cda-tum/ddsim/tree/limdd.

[234] Ece C Mutlu. Quantum probabilistic models using Feynman diagram rules for better understanding the information diffusion dynamics in online social networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 13730–13731, 2020.

[235] Kengo Nakamura, Shuhei Denzumi, and Masaaki Nishino. Variable shift SDD: a more succinct sentential decision diagram. *arXiv preprint arXiv:2004.02502*, 2020.

[236] Naoki Nakatani and Garnet Kin Chan. Efficient tree tensor network states (TTNS) for quantum chemistry: Generalizations of the density matrix renormalization group algorithm. *The Journal of chemical physics*, 138(13), 2013.

[237] Amit Narayan, Jawahar Jain, Masahiro Fujita, and Alberto Sangiovanni-Vincentelli. Partitioned ROBDDs-a compact, canonical and efficiently manipulable representation for boolean functions. In *Proceedings of International Conference on Computer Aided Design*, pages 547–554. IEEE, 1996.

[238] Hendrik Poulsen Nautrup, Nicolas Delfosse, Vedran Dunjko, Hans J Briegel, and Nicolai Friis. Optimizing quantum error correction codes with reinforcement learning. *Quantum*, 3:215, 2019.

[239] Xiaotong Ni, Oliver Buerschaper, and Maarten Van den Nest. A non-commuting stabilizer formalism. *Journal of Mathematical Physics*, 56(5):052201, 2015.

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

[241] Philipp Niemann, Robert Wille, and Rolf Drechsler. Equivalence checking in multi-level quantum systems. In *Reversible Computation: 6th International Conference, RC 2014, Kyoto, Japan, July 10-11, 2014. Proceedings 6*, pages 201–215. Springer, 2014.

[242] Philipp Niemann, Robert Wille, David Michael Miller, Mitchell A Thornton, and Rolf Drechsler. QMDDs: Efficient quantum function representation and manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(1):86–99, 2015.

[243] Philipp Niemann, Alwin Zulehner, Rolf Drechsler, and Robert Wille. Overcoming the trade-off between accuracy and compactness in decision diagrams for quantum computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[244] Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer, 2002.

[245] Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Zero-suppressed sentential decision diagrams. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[246] Weber Noah, Niranjan Balasubramanian, and Nathanael Chambers. Event representations with tensor-based compositions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[247] Yusuke Nomura, Andrew S Darmawan, Youhei Yamaji, and Masatoshi Imada. Restricted boltzmann machine learning for solving strongly correlated quantum systems. *Physical Review B*, 96(20):205152, 2017.

[248] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014.

[249] Umut Oztok and Adnan Darwiche. CV-width: A new complexity parameter for CNFs. In *ECAI*, pages 675–680, 2014.

[250] Sebastian Paeckel, Thomas Köhler, Andreas Swoboda, Salvatore R Manmana, Ulrich Schollwöck, and Claudius Hubig. Time-evolution methods for matrix-product states. *Annals of Physics*, 411:167998, 2019.

[251] Matteo Paltenghi and Michael Pradel. Bugs in quantum computing platforms: an empirical study. *Proceedings of the ACM on Programming Languages*, 6(OOPSLA1):1–27, 2022.

[252] Christos H Papadimitriou et al. Computational complexity. 1994.

[253] Lawrence C Paulson. *Isabelle: A generic theorem prover*. Springer, 1994.

[254] Lawrence C Paulson. Isabelle: The next 700 theorem provers. *arXiv preprint cs/9301106*, 2000.

[255] Edwin Pednault, John A. Gunnels, Giacomo Nannicini, Lior Horesh, and Robert Wisnieff. Leveraging secondary storage to simulate deep 54-qubit sycamore circuits, 2019.

[256] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking of quantum circuits with the ZX-calculus. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 12(3):662–675, 2022.

[257] Tom Peham, Lukas Burgholzer, and Robert Wille. Equivalence checking paradigms in quantum circuit design: A case study. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 517–522, 2022.

[258] R. Pelánek. BEEM: Benchmarks for Explicit Model Checkers. In *SPIN*, volume 4595 of *LNCS*, pages 263–267. Springer, 2007.

[259] Doron Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8:39–64, 1996.

[260] Shir Peleg, Amir Shpilka, and Ben Lee Volk. Lower bounds on stabilizer rank. *Quantum*, 6:652, 2022.

[261] Yuxiang Peng, Kesha Hietala, Runzhou Tao, Liyi Li, Robert Rand, Michael Hicks, and Xiaodi Wu. A formally certified end-to-end implementation of Shor's factorization algorithm. *Proceedings of the National Academy of Sciences*, 120(21):e2218775120, 2023.

[262] D. Perez-Garcia, F. Verstraete, M. M. Wolf, and J. I. Cirac. Matrix product state representations. *Quantum Information & Computation*, 7(5):401–430, jul 2007.

[263] Stefano Pirandola, Ulrik L Andersen, Leonardo Banchi, Mario Berta, Darius Bunandar, Roger Colbeck, Dirk Englund, Tobias Gehring, Cosmo Lupo, Carlo Ottaviani, et al. Advances in quantum cryptography. *Advances in Optics and Photonics*, 12(4):1012–1236, 2020.

[264] Stephen Plaza and Valeria Bertacco. Boolean operations on decomposed functions. *Proceedings of the 24th International Workshop on Logic & Synthesis*, pages 310–317, 2005.

[265] Stephen Plaza and Valeria Bertacco. STACCATO: disjoint support decompositions from BDDs through symbolic kernels. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, pages 276–279, 2005.

[266] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[267] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, 2018.

[268] Hammam Qassim, Hakop Pashayan, and David Gosset. Improved upper bounds on the stabilizer rank of magic states. *Quantum*, 5:606, 2021.

[269] Mohamed Raed El Aoun, Heng Li, Foutse Khomh, and Lionel Tidjon. Bug characteristics in quantum software ecosystem. *arXiv preprint arXiv:2204.11965*, 2022.

[270] Robert Rand, Jennifer Paykin, and Steve Zdancewic. QWIRE practice: Formal verification of quantum circuits in Coq. *arXiv preprint arXiv:1803.00699*, 2018.

[271] Robert Raussendorf and Hans J. Briegel. A one-way quantum computer. *Physical Review Letters*, 86:5188–5191, May 2001.

[272] Robert Raussendorf, Daniel E Browne, and Hans J Briegel. Measurement-based quantum computation on cluster states. *Physical Review A*, 68(2):022312, 2003.

[273] Igor Razgon. On OBDDs for CNFs of bounded treewidth. *arXiv preprint arXiv:1308.3829*, 2013.

[274] Markus Reiher, Nathan Wiebe, Krysta M Svore, Dave Wecker, and Matthias Troyer. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences*, 114(29):7555–7560, 2017.

[275] José Ignacio Requeno and José Manuel Colom. Compact representation of biological sequences using set decision diagrams. In *6th International Conference on Practical Applications of Computational Biology & Bioinformatics*, pages 231–239. Springer, 2012.

[276] Michael Rice and Sanjay Kulhari. A survey of static variable ordering heuristics for efficient BDD/MDD construction. *University of California, Tech. Rep*, 2008.

[277] Leanna Rierson. *Developing safety-critical software: a practical guide for aviation software and DO-178C compliance.* CRC Press, 2017.

[278] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 42–47. IEEE, 1993.

[279] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits—a survey. *ACM Computing Surveys (CSUR)*, 45(2):1–34, 2013.

[280] Tuhin Sahai, Anurag Mishra, Jose Miguel Pasini, and Susmit Jha. Estimating the density of states of boolean satisfiability problems on classical and quantum computing platforms. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1627–1635, Apr. 2020.

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

[282] Tsutomu Sasao. FPGA design by generalized functional decomposition. In *Logic synthesis and optimization*, pages 233–258. Springer, 1993.

[283] Tsutomu Sasao and Munehiro Matsuura. DECOMPOS: An integrated system for functional decomposition. In *1998 International Workshop on Logic Synthesis*, pages 471–477, 1998.

[284] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of physics*, 326(1):96–192, 2011.

[285] Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

[286] Peter Selinger. Quantum circuits of T-depth one. *Physical Review A*, 87(4):042302, 2013.

[287] Irfansha Shaik and Jaco van de Pol. Optimal layout synthesis for quantum circuits as classical planning. *arXiv preprint arXiv:2304.12014*, 2023.

[288] Claude E Shannon. A symbolic analysis of relay and switching circuits. *Electrical Engineering*, 57(12):713–723, 1938.

[289] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. Knowledge compilation meets uniform sampling. In *LPAR*, pages 620–636, 2018.

[290] Y-Y Shi, L-M Duan, and Guifre Vidal. Classical simulation of quantum many-body systems with a tree tensor network. *Physical Review A*, 74(2):022320, 2006.

[291] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Jour. of Comp.*, 26(5):1484–1509, 1997.

[292] Radu I Siminiceanu and Gianfranco Ciardo. New metrics for static variable ordering in decision diagrams. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 90–104. Springer, 2006.

[293] Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. CFLOBDDs: Context-free-language ordered binary decision diagrams. *arXiv preprint arXiv:2211.06818*, 2022.

[294] Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. Symbolic quantum simulation with quasimodo. *arXiv preprint arXiv:2302.04349*, 2023.

[295] Meghana Sistla, Swarat Chaudhuri, and Thomas Reps. Symbolic quantum simulation with quasimodo. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 213–225, Cham, 2023. Springer Nature Switzerland.

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

[297] SW Sloan. A FORTRAN program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering*, 28(11):2651–2679, 1989.

[298] Graeme Smith and Debbie Leung. Typical entanglement of stabilizer states. *Physical Review A*, 74(6):062314, 2006.

[299] Kaitlin N Smith and Mitchell A Thornton. A quantum computational compiler and design tool for technology-specific targets. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 579–588, 2019.

[300] Mathias Soeken, Stefan Frehse, Robert Wille, and Rolf Drechsler. RevKit: A toolkit for reversible circuit design. *J. Multiple Valued Log. Soft Comput.*, 18(1):55–65, 2012.

[301] Mathias Soeken, Laura Tague, Gerhard W. Dueck, and Rolf Drechsler. Ancilla-free synthesis of large reversible functions using binary decision diagrams. *Journal of Symbolic Computation*, 73:1–26, 2016.

[302] Mathias Soeken, Robert Wille, Christoph Hilken, Nils Przigoda, and Rolf Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *17th Asia and South Pacific Design Automation Conference*, pages 85–92. IEEE, 2012.

[303] Fabio Somenzi. Efficient manipulation of decision diagrams. *International Journal on Software Tools for Technology Transfer*, 3(2):171–181, 2001.

[304] Fabio Somenzi. CUDD: CU decision diagram package release 3.0.0. http://vlsi.colorado.edu/~fabio/, 2015.

[305] Stabranksearcher: code for finding (upper bounds to) the stabilizer rank of a quantum state. https://github.com/timcp/StabRankSearcher, 2021.

[306] Krysta Svore, Alan Geller, Matthias Troyer, John Azariah, Christopher Granade, Bettina Heim, Vadym Kliuchnikov, Mariia Mykhailova, Andres Paz, and Martin Roetteler. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pages 1–10, 2018.

[307] Ryan Sweke, Markus S Kesselring, Evert P.L. van Nieuwenburg, and Jens Eisert. Reinforcement learning decoders for fault-tolerant quantum computation. *Machine Learning: Science and Technology*, 2(2):025005, 2020.

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

[309] Yasuhiro Takahashi and Noboru Kunihiro. A quantum circuit for Shor's factoring algorithm using $2n+2$ qubits. *Quantum Information & Computation*, 6(2):184–192, 2006.

[310] Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307, 2015.

[311] Barbara M. Terhal and David P. DiVincenzo. Classical simulation of noninteracting-fermion quantum circuits. *Physical Review A*, 65:032325, Mar 2002.

[312] Dimitrios Thanos, Tim Coopmans, and Alfons Laarman. Fast equivalence checking of quantum circuits of clifford gates. *ATVA 2023 (accepted for publication)*, 2023.

[313] Himanshu Thapliyal, Edgard Munoz-Coreas, TSS Varun, and Travis S Humble. Quantum circuit designs of integer division optimizing T-count and T-depth. *IEEE Transactions on Emerging Topics in Computing*, 9(2):1045–1056, 2019.

[314] Jules Tilly, Hongxiang Chen, Shuxiang Cao, Dario Picozzi, Kanav Setia, Ying Li, Edward Grant, Leonard Wossnig, Ivan Rungger, George H. Booth, et al. The variational quantum eigensolver: a review of methods and best practices. *Physics Reports*, 986:1–128, 2022.

[315] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Neural-network quantum state tomography. *Nature Physics*, 14(5):447–450, 2018.

[316] Jan Tretmans, Klaas Wijbrans, and Michel Chaudron. Software engineering with formal methods: The development of a storm surge barrier control system revisiting seven myths of formal methods. *Formal Methods in System Design*, 19:195–215, 2001.

[317] Andrea Turrini. An introduction to quantum model checking. *Applied Sciences*, 12(4):2016, 2022.

[318] Tomás E. Uribe and Mark E. Stickel. Ordered binary decision diagrams and the Davis-Putnam procedure. In *International Conference on Constraints in Computational Logics*, pages 34–49. Springer, 1994.

[319] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990 10*, pages 491–515. Springer, 1991.

[320] John van de Wetering. ZX-calculus for the working quantum computer scientist. *arXiv preprint arXiv:2012.13966*, 2020.

[321] Ewout van den Berg and Kristan Temme. Circuit optimization of Hamiltonian simulation by simultaneous diagonalization of pauli clusters. *Quantum*, 4:322, 2020.

[322] Guy Van den Broeck and Adnan Darwiche. On the role of canonicity in knowledge compilation. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[323] M. Van den Nest, W. Dür, G. Vidal, and H. J. Briegel. Classical simulation versus universality in measurement-based quantum computation. *Physical Review A*, 75:012337, Jan 2007.

[324] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Graphical description of the action of local clifford transformations on graph states. *Physical Review A*, 69(2):022316, 2004.

[325] Maarten Van den Nest, Jeroen Dehaene, and Bart De Moor. Local unitary versus local clifford equivalence of stabilizer states. *Physical Review A*, 71:062323, Jun 2005.

[326] Tom Van Dijk, Alfons Laarman, and Jaco Van De Pol. Multi-core BDD operations for symbolic reachability. *Electronic Notes in Theoretical Computer Science*, 296:127–143, 2013.

[327] Tom van Dijk and Jaco van de Pol. Sylvan: Multi-core decision diagrams. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 677–691. Springer, 2015.

[328] Tom van Dijk, Robert Wille, and Robert Meolic. Tagged BDDs: combining reduction rules from different decision diagram types. In *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, pages 108–115. FMCAD Inc, 2017.

[329] Vivien Vandaele, Simon Martiel, Simon Perdrix, and Christophe Vuillot. Optimal Hadamard gate count for Clifford $+T$ synthesis of Pauli rotations sequences. *arXiv preprint arXiv:2302.07040*, 2023.

[330] Frank Verstraete, Diego Porras, and J. Ignacio Cirac. Density matrix renormalization group and periodic boundary conditions: A quantum information perspective. *Physical Review Letters*, 93(22):227205, 2004.

[331] George F. Viamontes, Igor L. Markov, and John P. Hayes. Improving gate-level simulation of quantum circuits. *Quantum Information Processing*, 2(5):347–380, 2003.

[332] George F Viamontes, Igor L Markov, and John P Hayes. *Quantum circuit simulation.* Springer Science & Business Media, 2009.

[333] George F. Viamontes, Manoj Rajagopalan, Igor L. Markov, and John P. Hayes. Gate-level simulation of quantum circuits. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, pages 295–301, 2003.

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

[335] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. *Physical Review Letters*, 91(14):147902, 2003.

[336] Renaud Vilmart. Quantum multiple-valued decision diagrams in graphical calculi, 2021.

[337] Lieuwe Vinkhuijzen, Tim Coopmans, David Elkouss, Vedran Dunjko, and Alfons Laarman. LIMDD: A decision diagram for simulation of quantum computing including stabilizer states. *Quantum*, 7:1108, 2023.

[338] Lieuwe Vinkhuijzen, Thomas Grurl, Stefan Hillmich, Sebastiaan Brand, Robert Wille, and Alfons Laarman. Efficient implementation of LIMDDs for quantum circuit simulation. In *International Symposium on Model Checking Software*, pages 3–21. Springer, 2023.

[339] Lieuwe Vinkhuijzen and Alfons Laarman. Symbolic model checking with sentential decision diagrams. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pages 124–142. Springer, 2020.

[340] Hefeng Wang, Sabre Kais, Alán Aspuru-Guzik, and Mark R Hoffmann. Quantum algorithm for obtaining the energy spectrum of molecular systems. *Physical Chemistry Chemical Physics*, 10(35):5388–5393, 2008.

[341] Shiou-An Wang, Chin-Yung Lu, I-Ming Tsai, and Sy-Yen Kuo. An XQDD-based verification method for quantum circuits. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(2):584–594, 2008.

[342] Shiou-An Wang, Chin-Yung Lu, I-Ming Tsai, and Sy-Yen Kuo. An xqdd-based verification method for quantum circuits. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(2):584–594, 2008.

[343] Tianshu Wang, Yuexian Hou, Panpan Wang, and Xiaolei Niu. Exploring relevance judgement inspired by quantum weak measurement. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[344] Ingo Wegener. *Branching programs and binary decision diagrams: theory and applications.* SIAM, 2000.

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

[346] Steven R. White. Density matrix formulation for quantum renormalization groups. *Physical Review Letters*, 69:2863–2866, Nov 1992.

[347] Robert Wille and Rolf Drechsler. BDD-based synthesis of reversible logic for large functions. In *Proceedings of the 46th Annual Design Automation Conference*, pages 270–275, 2009.

[348] Robert Wille and Rolf Drechsler. Effect of BDD optimization on synthesis of reversible and quantum logic. *Electronic Notes in Theoretical Computer Science*, 253(6):57–70, 2010.

[349] Robert Wille, Daniel Große, D Michael Miller, and Rolf Drechsler. Equivalence checking of reversible circuits. In *2009 39th International Symposium on Multiple-Valued Logic*, pages 324–330. IEEE, 2009.

[350] Robert Wille, Stefan Hillmich, and Lukas Burgholzer. JKQ: JKU tools for quantum computing. In *Int'l Conf. on CAD*, pages 154:1–154:5, 2020.

[351] Robert Wille, Nils Przigoda, and Rolf Drechsler. A compact and efficient SAT encoding for quantum circuits. In *2013 Africon*, pages 1–6. IEEE, 2013.

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

[353] Yu Wu, Per Austrin, Toniann Pitassi, and David Liu. Inapproximability of treewidth and related problems. *Journal of Artificial Intelligence Research*, 49:569–600, 2014.

[354] Yukai Wu, L.-M. Duan, and Dong-Ling Deng. Artificial neural network based computation for out-of-time-ordered correlators. *Physical Review B*, 101:214308, Jun 2020.

[355] Ming Xu, Jianling Fu, Jingyi Mei, and Yuxin Deng. Model checking QCTL plus on quantum markov chains. *Theoretical Computer Science*, 913:43–72, 2022.

[356] Shigeru Yamashita and Igor L Markov. Fast equivalence-checking for quantum circuits. In *2010 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 23–28. IEEE, 2010.

[357] Shigeru Yamashita, Shin-ichi Minato, and D Michael Miller. DDMF: An efficient decision diagram structure for design verification of quantum circuits under a practical restriction. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(12):3793–3802, 2008.

[358] Feidiao Yang, Jiaqing Jiang, Jialin Zhang, and Xiaoming Sun. Revisiting online quantum state learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6607–6614, Apr. 2020.

[359] Mingsheng Ying. Floyd–Hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6):1–49, 2012.

[360] Mingsheng Ying and Yuan Feng. Model checking quantum systems—a survey. *arXiv preprint arXiv:1807.09466*, 2018.

[361] Mingsheng Ying and Yuan Feng. *Model Checking Quantum Systems: Principles and Algorithms*. Cambridge University Press, 2021.

[362] Mingsheng Ying, Yuan Feng, Runyao Duan, and Zhengfeng Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic (TOCL)*, 10(3):1–36, 2009.

[363] Mingsheng Ying, Yangjia Li, Nengkun Yu, and Yuan Feng. Model-checking linear-time properties of quantum systems. *ACM Transactions on Computational Logic (TOCL)*, 15(3):1–31, 2014.

[364] Christof Zalka. Simulating quantum systems on a quantum computer. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):313–322, 1998.

[365] Yuan-Hang Zhang, Zhian Jia, Yu-Chun Wu, and Guang-Can Guo. An efficient algorithmic way to construct Boltzmann machine representations for arbitrary stabilizer code. *arXiv:1809.08631*, 2018.

[366] Li Zhou, Gilles Barthe, Justin Hsu, Mingsheng Ying, and Nengkun Yu. A quantum interpretation of bunched logic & quantum separation logic. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE, 2021.

[367] Li Zhou, Nengkun Yu, and Mingsheng Ying. An applied quantum Hoare logic. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1149–1162, 2019.

[368] Alwin Zulehner, Stefan Hillmich, Igor L Markov, and Robert Wille. Approximation of quantum states using decision diagrams. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 121–126. IEEE, 2020.

[369] Alwin Zulehner, Stefan Hillmich, and Robert Wille. How to efficiently handle complex values? implementing decision diagrams for quantum computing. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7. IEEE, 2019.

[370] Alwin Zulehner, Philipp Niemann, Rolf Drechsler, and Robert Wille. Accuracy and compactness in decision diagrams for quantum computation. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 280–283. IEEE, 2019.

[371] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2018.

[372] Alwin Zulehner and Robert Wille. Improving synthesis of reversible circuits: Exploiting redundancies in paths and nodes of QMDDs. In *International Conference on Reversible Computation*, pages 232–247. Springer, 2017.

[373] Alwin Zulehner and Robert Wille. One-pass design of reversible circuits: Combining embedding and synthesis for reversible logic. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(5):996–1008, 2017.

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

[375] Alwin Zulehner and Robert Wille. Compiling $SU(4)$ quantum circuits to IBM QX architectures. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 185–190, 2019.

[376] Alwin Zulehner and Robert Wille. *Introducing design automation for quantum computing*, volume 11. Springer, 2020.