Qubit Recycling Revisited

ANONYMOUS AUTHOR(S)

Reducing the width of quantum circuits is crucial due to limited number of qubits in quantum devices. This paper revisit an optimization strategy known as *qubit recycling* (alternatively *wire-recycling* or *measurement-and-reset*), which leverages gate commutativity to reuse discarded qubits, thereby reducing circuit width. We introduce *qubit dependency graphs* (QDGs) as a key abstraction for this optimization. With QDG, we isolate the computationally demanding components, and observe that qubit recycling is essentially a matrix triangularization problem. Based on QDG and this observation, we study qubit recycling with a focus on complexity, algorithmic, and verification aspects. Firstly, we establish qubit recycling's NP-hardness through reduction from Wilf's question, another matrix triangularization problem. Secondly, we propose a QDG-guided solver featuring multiple heuristic options for effective qubit recycling. Benchmark tests conducted on RevLib illustrate our solver's superior or comparable performance to existing alternatives. Notably, it achieves optimal solutions for the majority of circuits. Finally, we develop a certified qubit recycler that integrates verification and validation techniques, with its correctness proof mechanized in Coq.

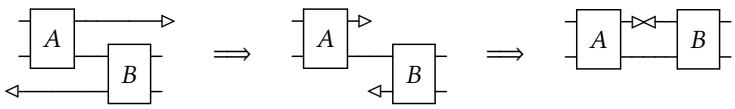
Additional Key Words and Phrases: quantum circuit, circuit width, complexity, certified compilation

1 INTRODUCTION

Reducing the cost of a quantum circuit is crucial, particularly for near-term quantum computers with limited computational resources [Preskill 2018]. One commonly used metric for assessing cost is the circuit *width*, which corresponds to the number of qubits used in a quantum circuit. In fault-tolerant quantum computing, where the overhead of a logical qubit using quantum error-correction is substantial [Fowler et al. 2012], circuit width becomes particularly important.

Among the various approaches to minimizing circuit width, qubit recycling, also known as wire recycling or reclaiming qubit via measurement-and-reset, has been found to be effective and intuitive. Analytic analysis [DeCross et al. 2022] on well-structured circuit families showcased its capacity to significantly reduce circuit width, sometimes exponentially or to a constant size. Empirical evaluation [DeCross et al. 2022; Hua et al. 2023; Paler et al. 2016] further indicated that qubit recycling can achieve reductions in circuit width up to 80 $\sim$ 90%. Moreover, recent research [Hua et al. 2023] illustrated that leveraging mid-circuit measurement for qubit recycling might enhance fidelity in specific circuits executed on real quantum hardware.

The idea behind qubit recycling is that once a qubit is measured and discarded, it becomes disentangled from the other qubits and can be reused as a fresh qubit by resetting it. Qubit recycling further leverages the commutativity of local quantum operations to create more opportunities for qubit reuse, allowing for earlier measurements or deferring allocation.



The above is an example illustrating a qubit recycling procedure, which rewrites the left most circuit into the right most one. In each circuit, a horizontal line represents a qubit, a box represents a quantum gate applied to the qubits it covers, and the gates are applied from left to right. Starting from the left-most circuit, we first postpone the allocation (denoted by $\triangleleft$) and bring forward the measurement (denoted by $\triangleright$). When the measurement is earlier than the allocation, we reuse the top qubit as the bottom one. As a result, the circuit width is reduced by one.

2018. 0004-5411/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

Despite its efficacy, qubit recycling remains an underexplored topic with several fundamental questions yet to be answered, including:

(1) *What is the computational complexity of minimizing circuit width using qubit recycling?* Previous works [DeCross et al. 2022; Hua et al. 2023; Paler et al. 2016] develop (exponential-time) exact or heuristic-based algorithms for qubit recycling. However, the intrinsic difficulty of this problem remains unclear.
(2) *How can we ensure the correctness of a qubit recycler?* Existing approaches on verified compilation for quantum programs [Hietala et al. 2021; Tao et al. 2022] do not directly apply to qubit recycling. These approaches primarily concentrate on rewriting-based optimizations that *preserve* circuit width, while qubit recycling aims to *reduce* circuit width.

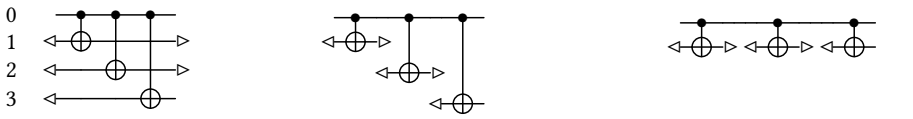
To better understand qubit recycling and address these questions, it is essential to have a problem abstraction that isolates the computationally intensive part from simpler circuit rewritings based on gate commutativity. With this in mind, we revisit qubit recycling and address the aforementioned questions. The contribution includes:

- We identify computational dependencies between *qubits* as the key factor in qubit recycling and formalize these dependencies using *qubit dependency graphs* (QDGs). A QDG concisely captures all the necessary information to determine valid qubit *recycling strategies*, specifying which qubit can be reused by another. With the matrix representation of QDGs, we observe that qubit recycling is essentially a matrix triangularization problem. This observation proves valuable in studying complexity, algorithm design, and verification of qubit recyclers.
- We prove that qubit recycling is NP-hard. The proof is based on a reduction from Wilf's question, another triangularization problem which is known to be NP-complete.
- Based on the structure of the matrix triangularization problem, we address the qubit recycling problem from three perspectives: (i) an efficient solver featuring multiple heuristic options; (ii) an integer linear programming (ILP) model that can be solved using optimizers like Gurobi; and (iii) an upper-bound estimator of the size of optimal solution.
  We evaluate our solver, ILP model, and upper-bound estimator using the RevLib [Wille et al. 2008] benchmark. Our solver consistently yields equally good or superior solutions when compared to existing methods. For the majority of the circuits, our solver achieves optimal solutions, and our upper-bound estimator is tight. Our ILP model is of smaller size ($O(n^2)$ variables and constraints) compared with existing work [DeCross et al. 2022]($O(n^2)$ variables and $O(n^4)$ constraints), which is potentially easier for finding optimal solutions.
- We formalize a weaker correctness criterion for quantum circuit optimizations, allowing for qubit renaming and reuse. To support dynamic qubit allocation and discard, our semantics decouple qubit identity from their physical location by explicitly incorporating I/O qubits in the circuit. Our correctness formulation is transitive and congruent to sequential composition, facilitating modular verification of optimization passes.
- We propose a certified qubit recycler design that integrates compiler verification and validation techniques. This recycler comprises an untrusted solver handling computationally intensive tasks and a verified circuit rewriter which also validates the recycling strategy. This approach circumvents the need to individually verify the solver and ensures the durability of our proof against potential future updates to it. The validation process aligns with part of the circuit transformations, namely topological sorting. This is the point where we integrate verification and validation techniques in a single module.
- We implement the certified qubit recycler in Coq. Byproducts of the Coq development include a verified Kahn's algorithm for topological sorting, and a version of the coherence theorem for symmetric monoidal categories.

Outline. Sec. 2 overview the main results. Sec.3 presents the necessary language and problem settings. Subsequently, Sec.4 formalizes QDGs, and Sec.5 proves the NP-hardness of qubit recycling. We then proceed to develop the certified qubit recycler in Sec.6. Finally, we present and evaluate our algorithms and ILP model for qubit recycling in Sec. 7, and discuss related work in Sec.8. Proofs that are not presented can be found in the supplementary material.

2 INFORMAL DEVELOPMENT

In this section, we briefly overview qubit recycling (Sec.2.1), and outline the challenges associated with addressing the questions raised in Sec.1 and present our approaches to tackle them (Sec.2.2). Throughout this section, we use Fig.1 as a working example.

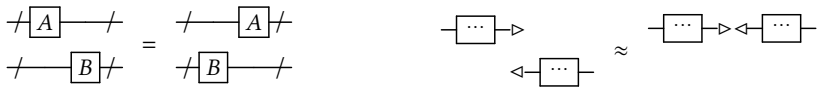


(a) Input circuit. (b) Topological deformation. (c) Renaming and reusing.

Fig. 1. An example of qubit recycling.

2.1 Background: Quantum Circuits and the Qubit Recycling Problem

Quantum circuits. Quantum programs are commonly represented using quantum circuits, which describe a sequence of quantum operations or gates. For instance, Fig.1a is a 4-qubit quantum circuit. Each horizontal line in the circuit corresponds to a qubit, numbered as 0, 1, 2, and 3. Quantum gates are applied from left to right. A $\triangleleft$ symbol represents the allocation of the qubit, while a $\triangleright$ symbol represents a discard gate. The discard gate effectively measures the qubit and discards the measurement result. A qubit used without allocation (e.g, qubit 0) is considered an *input qubit*, while a qubit lacking a discard gate (e.g. qubit 0 and 3) is referred to as *output qubits*. Input and output qubits may differ. For unitary gates, we consider only arbitrary 2-qubit gates for now. The gate used in Fig.1, is the *CX* gate, which is an arbitrary and irrelevant choice. It may help understanding if we temporarily forget the semantics of *CX*. In Fig.1, the *CX* gate is denoted by a vertical line connecting $\bullet$ and $\oplus$, indicating the two qubits it operates on. The vertical line has no effect on a qubit if the crossing is not associated with $\bullet$ or $\oplus$.



(a) Topological deformation. Here $-\!\!/\!\!-$ means one or more qubits.

(b) Qubit reusing

Fig. 2. Semantic preserving circuit transformations.

In this section, we do not delve into the semantics of a circuit but instead introduce two transformations that preserve semantics.

The first transformation is *topological deformation*, as demonstrated in Fig. 2a. It states that two gates commute if they are applied on disjoint sets of qubits. Two circuits are considered *topologically identical* if they are identical modulo topological deformations.

The second transformation is *qubit reusing*, depicted in Fig. 2b. It states that if the discard of a qubit precedes the allocation of another qubit, the former can be reused as the latter. We use the symbol $\approx$ instead of $=$ to indicate that these circuits are not identical but observably equivalent modulo renaming of I/O qubits. Qubit reusing leads to a reduction in circuit width by 1.

Qubit recycling. Given a quantum circuit, qubit recycling aims to *find a topologically identical circuit that maximizes qubit reusing*, effectively minimizing the circuit width. Consider the circuit in Fig.1a. One can first reshape the circuit using topological deformation and obtain Fig.1b. Then, by renaming qubit 3 into 2 and qubit 2 into 1, we obtain Fig.1c with width reduced by 2.

2.2 Challenges and Our Approach

Our goal is to solve the qubit recycling problem correctly and efficiently. Specifically, we aim to understand the problem's complexity, develop a solution that runs in a reasonable time, and create a certified optimizer for qubit recycling. In the following, we outline the challenges we face and present our approaches to achieving these goals.

2.2.1 The search space is too large to analyze. Given a circuit, the set of its topologically identical circuits becomes unmanageably large as the number of gates increases. This set comprises all the topological orderings of the circuit's directed acyclic graph (DAG) representation, which are the total orders compatible with the DAG. Even computing the size of this set is #P-complete [Brightwell and Winkler 1991]. For instance, the number of all topological orderings for the DAG repr. of the circuit in Fig.1a is more than 100, which is significantly larger than 4, the qubit count.

Our approach: decomposition using recycling strategies. Our first observation is that by employing *recycling strategies*, we can decompose the qubit recycling problem and avoid directly analyzing the set of topologically identical circuits. A recycling strategy is an injective partial map on qubits, where each pair $q \hookrightarrow q'$ in the map denotes that q' reuses q. Every solution to the qubit recycling problem corresponds to a recycling strategy $\hookrightarrow$, and its size $|\hookrightarrow|$ represents the number of reused qubits. Conversely, given a valid $\hookrightarrow$, we can efficiently construct a solution of the same width.

Therefore, to find an optimal solution for qubit recycling, it suffices to:

(i) Find the largest *valid* recycling strategy $\hookrightarrow$ for the input circuit
(ii) Construct a solution using $\hookrightarrow$ and the input circuit.

Sub-problem (ii) can be solved in linear time, which is essentially topological sorting. Regarding sub-problem (i), it is evident that the number of all recycling strategies (valid or invalid) is only relevant to the circuit's width. In practice, the search space for sub-problem (i) is much smaller than that for the original qubit recycling problem.

Returning to our example circuit, since there are only 2 discarded qubits and 3 initialized qubits, there are at most 12 recycling strategies of interest, and only 4 of them are valid: $\{1 \hookrightarrow 2\}$, $\{2 \hookrightarrow 3\}$, $\{1 \hookrightarrow 3\}$, and $\{1 \hookrightarrow 2, 2 \hookrightarrow 3\}$. With the largest strategy, we can obtain a solution for qubit recycling by connecting discard of qubit 1 and 2 with allocation of qubit 2 and 3, respectively, then perform a topological sort on it.

2.2.2 Gap 1: What is a valid recycling strategy? A straightforward answer is that a recycling strategy is valid if it corresponds to a solution of the original problem, but this requires finding a solution in the first place. Intuitively, there should be a simpler way: a recycling strategy is essentially a relation over qubits, it should be possible to check its validity knowing only qubit relationships.

Our approach: We introduce *qubit dependency graphs* (QDGs) as the key abstraction for analyzing recycling strategies and verifying their validity. The QDG of a circuit C, denoted as $QDG(C)$, is a directed graph whose vertices represent the qubits in C. An edge $q \to q'$ in $QDG(C)$ indicates a

computational dependency, it means that some computations involving q must occur before those involving q'. Concretely, $q \to q'$ if either (i) there is a path in the DAG representation of C from the allocation of q to discard of q', or (ii) q/q' is an input/output qubit. Such a dependency prevents reusing q' as q. Notably, each vertex in a QDG has a self-loop.

A recycling strategy $\hookrightarrow$ is valid for C turns out to be equivalent to

$$\to\hookrightarrow \text{ is acyclic.}$$

Here $\to$ is the edges of $QDG(C)$, and $\to\hookrightarrow$ is a composed graph where $q \to\hookrightarrow q''$ if there exists q' such that $q \to q' \hookrightarrow q''$. This criteria is intuitively necessary: a cycle in $\to\hookrightarrow$ means there is a circular dependency if we are to use this recycling strategy.

An example of a QDG is shown in Fig.3, where self-loops are omitted for clarity. This QDG effectively abstracts away irrelevant information in the circuit. It is easy to verify that $\{1 \hookrightarrow 2, 2 \hookrightarrow 3\}$ is a valid recycling strategy, while $2 \hookrightarrow 1$ is not due to the presence of a cycle $1 \to 2 \hookrightarrow 1$.

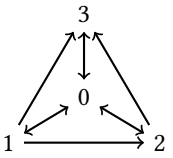


Fig. 3. *QDG*(Fig. 1a).

2.2.3 Gap 2: How to find a largest valid recycling strategy? Although QDGs provide a more concise criteria of valid recycling strategies, the set of recycling strategies is still quite large, with $O(n!)$ possible injective maps over n qubits, and an even larger number of partial injective maps. Brute-force search becomes unfeasible as the number of qubits grows.

This raises the following natural questions:

(a) Is it possible to efficiently find the largest recycling strategy?
(b) If not, can we find sufficiently good recycling strategies in a reasonable amount of time?

Answering question (a): hardness result. We prove that finding the maximal recycling strategy for a given QDG is NP-hard. This result is derived from the NP-completeness of the decision version of the problem: determining if there exists a valid recycling strategy of size k for a given QDG.

Our key observation is that finding the largest valid strategy over a QDG is equivalent to a matrix triangularization problem. Note that a QDG $\to$ and a recycling strategy $\hookrightarrow$ are directed graphs. Suppose their adjacency matrices are A and R, respectively, then

$$\begin{aligned} & \to\hookrightarrow \text{ is acyclic} \\ \Leftrightarrow\ & AR \text{ is nilpotent} \\ \Leftrightarrow\ & \text{there is a permutation } P \text{ such that } P^T A(RP) \text{ strictly upper triangular.} \end{aligned} \tag{1}$$

If R represents a total injective map, then RP is also a permutation matrix. This equivalent form remind us of another problem: *can a square matrix be made upper triangular by independently permuting its rows and columns?*

The problem, known as Wilf's question [Wilf 1997], has been studied extensively in Haddad's dissertation [Haddad 1990], and is known to be NP-complete [Fertin et al. 2015] recently. We prove that Wilf's question can be reduced to qubit recycling problem, demonstrating the latter is NP-hard. The main difficulty in this reduction is that the QDG of a circuit is not an arbitrary square 0-1 matrix, for example: it has 1s on its diagonal. The reduction is achieved by appropriately padding a matrix with 0s and 1s, such that it becomes the QDG of some circuit.

Answering question (b): QDG-based algorithms. Based on the previous observation, we develop a solver for this NP-hard problem. The key insight for this solver is that: once the permutation P in Eq. (1) is given, a recycling strategy R with maximal size (that is, the rank of matrix R) can be computed efficiently. The problem is then reduced to finding an appropriate permutation P, which can be done either by a heuristic, or by exhaustive search.

We demonstrate the procedure of solving R given P over the adjacency matrix A of the QDG of the circuit in Fig. 1a. The matrix A together with the indices of its rows and columns is shown in Fig. 4a. Suppose we are given a permutation P^T that maps rows 0123 into 3210, then P^TA is shown in Fig. 4b. The solver then try to put the 0s of each row to the right most columns. For example, starting from the first row (with index 3), we put the two 0s to the right, as shown in Fig. 4c. We then continue the process with the remaining submatrix with no background color, that is, we remove the first row, and columns with value 1 in the first row. One more iteration gives the matrix in Fig. 4d, and the procedure stops because no 0s can be found in the remaining submatrix.

	0	1	2	3
0	1	1	1	1
1	1	1	0	1
2	1	0	1	1
3	1	0	0	1

(a)

	0	1	2	3
3	1	0	0	1
2	1	0	1	1
1	1	1	0	1
0	1	1	1	1

(b)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(c)

	0	3	2	1
3	1	1	0	0
2	1	1	1	0
1	1	1	0	1
0	1	1	1	1

(d)

Fig. 4. Demonstrating the solving procedure using the circuit in Fig. 1a

To get R, we read out the column and row indices of each diagonal elements (in green background) in the strictly upper triangular submatrix at the right upper corner: $(2, 3), (1, 2)$. It coincides with the previous solution $\{1 \hookrightarrow 2, 2 \hookrightarrow 3\}$.

By far, the solver is parameterized with a permutation P. To find P, it suffices to choose one row at a time, which aligns with the solving procedure. For example, we can choose a row that maximizing the remaining submatrix (greedy), or choose a row such that maximizing the number of 0s in the remaining submatrix (heuristics). In addition, we may consult an additional iteration to break a tie (look ahead).

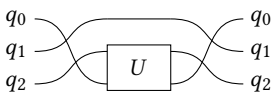
To evaluate these algorithms, we also design an ILP model for solving optimal solutions, and an upper-bound estimator for the cases when solving optimal solution is unfeasible. Evaluation (Sec.7.3) over the RevLib [Wille et al. 2008] benchmark shows that these methods can find optimal solutions for most cases.

2.2.4 Generalizing the notion of semantic preservation. Now that we have an efficient algorithm to find good solutions to the qubit recycling problem, it becomes crucial to ensure the correctness of its implementation. Therefore, we aim to develop a certified qubit recycler that guarantees its correctness. To achieve this, we need to define the correctness for a qubit recycler in the first place.

The correctness of a circuit optimization is typically defined in terms of semantic preservation. However, for qubit recycling, the existing notions of semantic preservation in compiler verification for quantum programs, such as those used in VOQC [Hietala et al. 2021] and Giallar [Tao et al. 2022], do not directly apply.

VOQC's notion of semantic preservation is based on superoperators, with equality defined over the denotations of circuits. However, VOQC explicitly ties qubit identity to their physical location in a density matrix and maintains the dimension of quantum states during execution. This approach makes it challenging to formalize the semantics of dynamic qubit allocation and discard, and it does not support qubit renaming.

Similarly, CertiQ focuses on local unitary rewritings and has a limited notion of semantic preservation for unitary circuits. Since qubit recycling involves non-unitary operations like discard and allocation, their notion of semantic preservation does not directly apply.



(a) Lifting

$$\frac{C_1 \sqsupseteq C_1' \qquad C_2 \sqsupseteq C_2'}{C_1; C_2 \sqsupseteq C_1'; C_2'} \qquad \frac{C \sqsupseteq C' \qquad C' \sqsupseteq C''}{C \sqsupseteq C''}$$

(b) Compositionality

Fig. 5. Semantic lifting and semantic preservation.

Our approach: instrumented circuits, semantic lifting and equivalence. We propose a weaker criterion that allows for qubit renaming and reusing. We achieve this by decoupling qubit identity from their physical location at the beginning or end of execution. This decoupling is accomplished by explicitly instrumenting a circuit C with lists of input/output (I/O) qubits, which serve as maps from qubit identities to their locations in the input/output states.

To define the semantics of a quantum gate with respect to these specific inputs and outputs, we introduce a semantic lifting mechanism. This lifting is defined using permutations and is applied to the qubits before and after the gate operation. For example, when applying a two-qubit unitary gate U on qubits $[q_2, q_0]$ in a state containing qubits $[q_0, q_1, q_2]$, we first permute q_2 and q_0 to adjacent positions, apply the gate U, and then reverse the permutation. Fig. 5a depicts this lifting. This allows us to define semantic preservation that is transitive and congruent to sequential composition.

In our definition of semantic preservation, an instrumented circuit (C', In', Out') preserves the semantics of (C, In, Out), denoted as $C \sqsupseteq C'$ omitting the I/O for short, if there exist bijections f and g (for renaming the qubits), and permutations σ and τ (for positioning the qubits), satisfying:

$$\begin{array}{ccc} In & \xrightarrow{C} & Out \\ \downarrow{\scriptstyle \sigma \circ f} & & \downarrow{\scriptstyle \tau \circ g} \\ In' & \xrightarrow{C'} & Out' \end{array}$$

(1) $\sigma \circ f(In) = In'$, and $\tau \circ g(Out) = Out'$, and
(2) if the input states are equivalent modulo σ, the output states are equivalent modulo τ.

This generalized notion can be roughly interpreted as the above commutative diagram. It allows for qubit renaming and reusing while still preserving the effects on the input and output states modulo permutation. It is congruent with sequential composition and is transitive, as shown in Fig. 5b. This enables modular verification of an optimizer.

2.2.5 Verification versus validation? When it comes to building a certified optimizing compiler [Leroy 2009; Rideau and Leroy 2010], a similar choice arises in the context of qubit recycling. We can either directly verify the qubit recycler, or employ an untrusted optimizer alongside a verified validator to verify the result. The former ensures that every output of the recycler is correct, while the latter typically reduces the amount of proof work required and remains resilient to optimizer upgrades.

However, relying solely on verification or validation seems to be unsatisfying. On one hand, verifying the correctness of the qubit recycler necessitates mechanically proving the previously introduced algorithms concerning QDG. This process can be time-consuming and susceptible to updates. On the other hand, validation approaches may fall short in terms of completeness, as they may reject correct results. The task of checking the identity of two quantum circuits is generally challenging [Janzing et al. 2003], and existing translation validation methods [Kissinger and van de Wetering 2020] may fail to identify equivalent circuits.

Our approach: integrating verification and validation. We have designed our optimizer to facilitate the integration of verification and validation techniques, leveraging the benefits of both approaches. As described in Section 2.2.1, our qubit recycler consists of two components: a solver and a rewriting

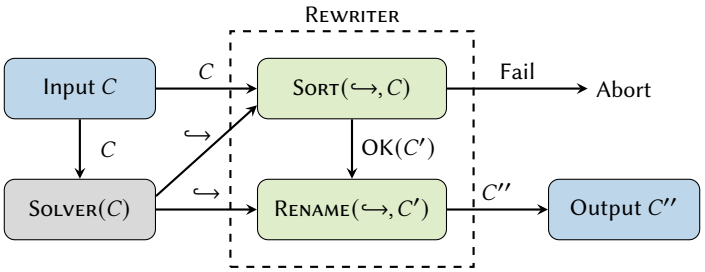


Fig. 6. Structure of the certified qubit recycler.

module that operates based on the solver's results. The solver handles complicated computations, while the rewriting module is more amenable to verification.

The structure of our certified qubit recycler is illustrated in Fig. 6. The green nodes represent the formally verified Rewriter module, while the gray node represents the untrusted Solver. An input circuit C is passed to both the Solver and the Sort module. The Solver attempts to compute a recycling strategy $\hookrightarrow$ and sends it to both the sorting and renaming modules. The Sort module then aims to reorder C into C' using topological deformation (Fig. 2a), ensuring that C' adheres to the recycling strategy $\hookrightarrow$. If Sort succeeds, the Rename module simply renames the qubits in C' according to $\hookrightarrow$. If Sort fails, the compilation is aborted.

Notably, the Sort module serves a dual role as both a validator and a circuit rewriter. This is the point where we integrate validation and verification techniques in certifying a single module. In essence, Sort performs a topological sorting on the circuit with additional recycling edges, utilizing Kahn's algorithm [Kahn 1962]. It's important to note that topological sorting simultaneously sorts the circuit and detects cycles in a digraph. As a result, Sort effectively checks the validity of the recycling strategy $\hookrightarrow$ with respect to circuit C, while transforming C into C'.

From a verification perspective, we verify that for any given $\hookrightarrow$, the function Sort$(\hookrightarrow, -)$ correctly transforms the circuit when it succeeds:

$$\textsc{Sort}(\hookrightarrow, C) = \mathit{OK}(C') \implies C \sqsupseteq C'.$$

From a validation perspective, we verify that Sort$(-, C)$ serves as a correct validator that checks the desirable properties of a recycling strategy:

$$\textsc{Sort}(\hookrightarrow, C) = \mathit{OK}(C') \implies \hookrightarrow \text{ is good w.r.t. } C'.$$

These desirable properties ensure that $\hookrightarrow$ is a suitable input for the Rename$(-, C')$ module:

$$\hookrightarrow \text{ is good w.r.t. } C' \implies C' \sqsupseteq \textsc{Rename}(\hookrightarrow, C').$$

Putting these together, we obtain a certified qubit recycler (Sec. 6). Since topological sorting always succeeds on a DAG, our recycler will not reject a valid recycling strategy.

3 BASIC SETTINGS

This section presents the syntax of a quantum circuit description language and the qubit recycling problem on it. The instrumented circuits and their semantics are introduced in Sec. 6 when needed.

3.1 Syntax of Quantum Circuits

The left half of Fig. 7 shows the syntax of a simple quantum circuit description language. A (uninstrumented) circuit C is a list of instructions, we use ";" to denote both cons and list concatenation. An

$$
\begin{array}{llcll}
(\textit{Circuit}) & C & ::= & \mathsf{nil} & \textit{instr} \,\#\#\, \textit{instr}' \;\text{iff}\; \textit{args}(\textit{instr}) \cap \textit{args}(\textit{instr}') = \emptyset \\
 & & \mid & \textit{instr}; C & C \,\#\#\, C' \;\text{iff}\; \forall \textit{instr} \in C, \textit{instr}' \in C'.\ \textit{instr} \,\#\#\, \textit{instr}' \\
(\textit{Instr}) & \textit{instr} & ::= & \mathbf{alloc}[q] & \textit{args}(\mathbf{alloc}[q]) = \{q\} \\
 & & \mid & \mathbf{discard}[q] & \textit{args}(\mathbf{discard}[q]) = \{q\} \\
 & & \mid & U[\vec{q}] & \textit{args}(U[\vec{q}]) = \{q \in \vec{q}\}
\end{array}
$$

Fig. 7. Syntax and disjoint instructions.

instr can be one of the followings: $\mathbf{alloc}[q]$, which allocates a fresh qubit q in state $|0\rangle$; $\mathbf{discard}[q]$, which measures qubit q and discards the outcome; or $U[\vec{q}]$, which applies a unitary operator U to a list $\vec{q}$ of qubits. The qubit identities ranges from a set *Qid*. We do not instantiate a gate set, since it is mostly irrelevant to qubit recycling. As an example, the circuit in Fig. 1a is

$$\mathbf{alloc}[1]; \mathbf{alloc}[2]; \mathbf{alloc}[3]; CX[0,1]; CX[0,2]; CX[0,3]; \mathbf{discard}[1]; \mathbf{discard}[2].$$

The right half of Fig. 7 defines *disjoint* instructions. Two instructions *instr* and *instr′* are disjoint, denoted by $\textit{instr} \,\#\#\, \textit{instr}'$, if their arguments are disjoint, i.e., $\textit{args}(\textit{instr}) \cap \textit{args}(\textit{instr}') = \emptyset$. Here *args* returns the arguments occurring in an instruction or a circuit. Two circuits C and C' are disjoint, denoted by $C \,\#\#\, C'$, if their instructions are disjoint.

3.2 The Qubit Recycling Problem

When considering the qubit recycling problem, we assume every circuit to be *simple*, that is, a discarded qubit won't be initialized later. For general cases where q is allocated after discard, we may either rename q into a fresh qubit after discard, or combine $\mathbf{discard}[q]$ and $\mathbf{alloc}[q]$ into a measure-and-reset gate, to obtain a simple circuit.

Definition 3.1 (Simple circuits). A circuit C is *simple* if and only if for any $q \in \textit{args}(C)$,

(1) there is at most one $\mathbf{alloc}[q]$ in C, and $\mathbf{alloc}[q]$ (if it exists) is the first gate on q, and
(2) there is at most one $\mathbf{discard}[q]$ in C, and $\mathbf{discard}[q]$ (if it exists) is the last gate on q.

Given an input circuit, the qubit recycling problem is to find a topologically identical circuit that maximizes the number of reusable qubits. We formulate a decision version of this problem to study its complexity. Below we introduce notions used in formalizing the qubit recycling problem.

Reusable qubit. Given two qubits q and q' in a simple circuit C, q' *can reuse* q if and only if $\mathbf{discard}[q]$ occurs before $\mathbf{alloc}[q']$ in C. Formally, q' can reuse q in C if there exists n and n' such that $n < n'$, $C[n] = \mathbf{discard}[q]$ and $C[n'] = \mathbf{alloc}[q']$. Here $C[n]$ is the n-th element in C.

Topologically identical circuits. Two simple circuits C and C' are *topologically identical*, denoted by $C \sim C'$, if we can obtain C' from C by swapping adjacent disjoint instructions:

$$\textsc{Nil}\ \ \mathsf{nil} \sim \mathsf{nil} \qquad \textsc{Skip}\ \frac{C \sim C'}{\textit{instr}; C \sim \textit{instr}; C'} \qquad \textsc{Swap}\ \frac{\textit{instr} \,\#\#\, \textit{instr}'}{\textit{instr}; \textit{instr}'; C \sim \textit{instr}'; \textit{instr}; C} \qquad \textsc{Trans}\ \frac{C \sim C' \qquad C' \sim C''}{C \sim C''}$$

Clearly $\sim$ is an equivalence relation. As we will see in Sec 6, if $C \sim C'$, they are semantically equivalent, so we can safely transform C into C' to find more reusable qubits.

Recycling strategy and validity. We call a set of pairs of qubits $\{(q_i, q_i') \mid i = 1, \ldots k\}$ a *recycling strategy* if it defines an injective partial map over qubits, i.e., for any $i \neq j$ we have $q_i \neq q_j$ and $q_i' \neq q_j'$. The *size* of a recycling strategy is the number of pairs in it. We often use the notation $\hookrightarrow$ for the relation defined by a recycling strategy, i.e., $q_i \hookrightarrow q_i'$ if (q_i, q_i') is in the strategy. A recycling

strategy $\hookrightarrow$ is *valid w.r.t. C*, if there exists $C' \sim C$ such that for any $q_i \hookrightarrow q_i'$, q_i' can reuse q_i. Later we will see that a given a valid recycling strategy, we can construct C' in polynomial time.

Definition 3.2 (Qubit recycling problem (decision)). Given a simple circuit C and $k \in \mathbb{N}$, decide if there is a valid recycling strategy w.r.t. C of size k.

The original qubit recycling problem is an optimization problem to find a valid recycling strategy with the largest size. The decision problem in Def. 3.2 is no harder than the optimization problem.

4 QUBIT DEPENDENCY GRAPHS

A qubit dependency graph (QDG) makes the qubit recycling problem more manageable. It abstracts away irrelevant details and lets us focus on the computational dependencies that decide whether a qubit can be reused. Below we formalize QDG, then show that the validity of a recycling strategy for a circuit can be determined using its corresponding QDG only. To refer to each instruction in C easily, we implicitly label a instruction by its location in C, such that it is unique in C.

Dependency between instructions. The computation of *instr′* depends on *instr* in circuit C, denoted by $\mathit{instr} \prec_C \mathit{instr}'$, if $C = C_1; \mathit{instr}; C_2; \mathit{instr}'; C_3$ such that $(\mathit{args}(\mathit{instr}) \cap \mathit{args}(\mathit{instr}')) - \mathit{args}(C_2)$ is not empty. Equivalently, it means there is an edge from *instr* to *instr′* in the DAG representation of C. In particular, if $\mathbf{alloc}[q] \prec_C \mathbf{discard}[q']$, then q cannot reuse q' in any $C' \sim C$.

Definition 4.1 (Qubit dependency graph). The QDG of a simple circuit C, denoted by $\mathit{QDG}(C)$, is the digraph $(\mathit{args}(C), \rightarrow)$, where $q \rightarrow q'$ if and only if

- q is an input qubit ($\mathbf{alloc}[q] \notin C$), or q' is an output qubit ($\mathbf{discard}[q'] \notin C$), or
- $\mathbf{alloc}[q] \prec_C{}^* \mathbf{discard}[q']$. Here $\prec_C{}^*$ is the reflexive transitive closure of $\prec_C$.

For example, for circuit C in Fig. 1a, we have $1 \rightarrow 2$ in its QDG, because $\mathbf{alloc}[1] \prec_C CX[0,1] \prec_C CX[0,2] \prec_C \mathbf{discard}[2]$; and $0 \leftrightarrow 1$ because $\mathbf{alloc}[0]$ and $\mathbf{discard}[0]$ are not in C.

Notably, QDG is invariant under topological deformation.

Lemma 4.2. *Given simple circuits C and C', if $C \sim C'$, then $\mathit{QDG}(C) = \mathit{QDG}(C')$.*

Validity of recycling strategy w.r.t. QDG. A recycling strategy $\hookrightarrow$ is *valid w.r.t a digraph* $(V, \rightarrow)$, if $\hookrightarrow \subseteq V \times V$ and $\rightarrow\hookrightarrow$ is acyclic. Here $v \rightarrow\hookrightarrow v'$ if there is v'' such that $v \rightarrow v''$ and $v'' \hookrightarrow v'$. If $(V, \rightarrow)$ is the QDG of simple circuit C, the validity of $\hookrightarrow$ w.r.t. C coincides with that w.r.t. QDG.

Lemma 4.3 (Adequacy of QDG). *Given a simple circuit C and a recycling strategy $\hookrightarrow$,*

$$\hookrightarrow \text{ is valid w.r.t } C \iff \hookrightarrow \text{ is valid w.r.t. } \mathit{QDG}(C).$$

Validity w.r.t. QDG is irrelevant to *universal vertices*. Here a vertex is universal if it is connected to every vertex in both directions, modeling a qubit that serves as both an input and an output.

Proposition 4.4. *Given a digraph $G = (V, \rightarrow)$ and a set $V' \subseteq V$ of universal vertices in G. A recycling strategy is valid w.r.t. G if and only if it is valid w.r.t. the induced subgraph $G[V \setminus V']$.*

5 QUBIT RECYCLING IS NP-HARD

We prove the NP-hardness of the qubit recycling problem by showing the corresponding decision problem (Def. 3.2) is NP-complete. This proof involves two reductions, with recycling on QDGs (Definition 5.2) serving as the link between an NPC problem and the qubit recycling problem.

Theorem 5.1. *The decision version of the qubit recycling problem is NP-complete.*

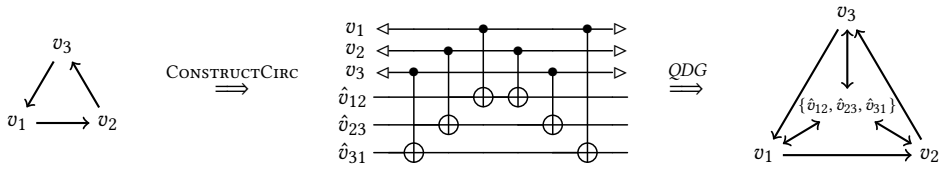


Fig. 8. An example digraph and the constructed circuit. Self loops are omitted.

Proof. The qubit recycling problem is in NP, because detecting cycles in a digraph has linear time algorithms. To show it is NP-complete, it suffices to prove the following reductions, since Wilf's question is NP-complete [Fertin et al. 2015].

Wilf's question	$\leq_p$	Recycling on QDGs	$\leq_p$	Qubit recycling
(Def. 5.6)	(Lm. 5.7,5.5)	(Def. 5.2)	(Lm. 5.3)	(Def. 3.2)

□

In the following subsections, we provide explain the reduction steps from right to left.

5.1 Recycling on QDGs is No Harder than Qubit Recycling

We first formalize the intermediate problem using QDG. Observe from Def. 4.1 that for each qubit q occur in C, $q \longrightarrow q$ in $QDG(C)$, we consider digraph with self loops only.

Definition 5.2 (Recycling problem on QDG). Given a digraph $G = (V, \longrightarrow)$ such that $\forall v \in V. v \longrightarrow v$, and a natural number $k > 0$, decide whether there is a valid recycling strategy of size k w.r.t. G.

To reduce Def. 5.2 to qubit recycling problem, we construct a circuit based on a digraph using Alg. 1. The intuition is that for each edge $v \longrightarrow v'$ in a digraph G, we introduce a fresh qubit $\hat{v}$ and two CX gates [1], such that $\mathbf{alloc}[v] \prec CX[v, \hat{v}] \prec CX[v', \hat{v}] \prec \mathbf{discard}[v']$, i.e., $v \longrightarrow v'$ is also in the constructed circuit's QDG. Since $\hat{v}$ are I/O qubits, they are universal vertices, and do not introduce new valid recycling strategies.

Algorithm 1 ConstructCirc$(V, \longrightarrow)$

```
1: C ← nil
2: for e ∈ {(v, v') | v → v' ∧ v ≠ v'} do
3:     C ← CX[v, v̂_e]; C; CX[v', v̂_e]
4:     // v̂_e is fresh
5: end for
6: for v ∈ V do
7:     C ← alloc[v]; C; discard[v]
8: end for
9: return C
```

For example, in Fig. 8, the left-most of is a 3-node digraph, in the middle is the constructed circuit. The right-most is the QDG of the circuit, for clarity, we merge the $\hat{v}$ vertices. A recycling strategy is valid w.r.t. the left-most digraph if and only if it is valid w.r.t. the right most digraph, since $\hat{v_{ij}}$ are universal vertices.

Lemma 5.3. *The qubit recycling problem is no harder than the recycling problem on QDGs.*

5.2 Wilf's Question is No Harder than Recycling on QDGs

This subsection briefly explains how to reduce Wilf's question to recycling on QDGs. We rephrase Def. 5.2 using adjacency matrices, and observe that a digraph is acyclic if and only if its adjacency matrix A can be made strictly upper triangular by permuting its columns and rows simultaneously. That is, there is a permutation matrix P such that $P^T AP$ is strictly upper triangular.

Definition 5.4 (A triangularization problem). Given a square 0, 1-matrix A whose diagonal elements are all 1s, and a natural number $k > 0$, decide if there exists permutation matrices P and Q such that $PAQ = \left[\begin{smallmatrix} * & * \\ B & * \end{smallmatrix}\right]$ for some $k \times k$ strictly upper-triangular matrix B.

[1] We can use any two-qubit gate in the place of CX.

Lemma 5.5. *Recycling problem on QDGs is equivalent to the triangularization problem (Def. 5.4).*

The formulation in Def. 5.4 is close enough to Wilf's question defined below.

Definition 5.6 (Wilf's question [Wilf 1997]). Given a square 0, 1-matrix A, Wilf's question, denoted by $\textsc{Wilf}(A)$, asks the existence of permutations P and Q such that PAQ is upper triangular.

Lemma 5.7. *Wilf's question is no harder than the triangularization problem.*

Given an instance of Wilf's question, we can construct a matrix as an instance of problem Def. 5.4 that has the same answer. The construction is by appropriately padding a matrix with 0s and 1s , which is put in the supplementary material due to space limit.

6 CERTIFIED QUBIT RECYCLER

This section presents the verified qubit recycler

$$\textsc{Rewriter}(\hookrightarrow, \widetilde{C}) ::= (\textsc{Rename}(\hookrightarrow, -) \circ \textsc{Sort}(\hookrightarrow, -))(\widetilde{C}).$$

We break down the correctness proof into syntactic and semantic properties, as shown below.

$$\textsc{Rewriter}(\hookrightarrow, \widetilde{C}) = \mathsf{OK}(\widetilde{C'}) \overset{\text{(Lm. 6.4,6.5,6.7)}}{\Longrightarrow} \widetilde{C} \rightarrowtail^* \widetilde{C'} \overset{\text{(Lm. 6.3,6.2)}}{\Longrightarrow} \widetilde{C} \sqsupseteq \widetilde{C'}$$

The philosophy is to break down semantic preservation into as many syntactic properties as possible, minimizing reliance on semantic properties in proofs. This approach stems from the fact that syntactic properties are often considerably easier to prove than semantic ones.

Syntactically, we establish that (i) $\textsc{Sort}(-, \widetilde{C})$ is a correct validator for the validity of a recycling strategy; and (ii) when $\hookrightarrow$ is a valid recycling strategy, $\textsc{Sort}(\hookrightarrow, -)$ and $\textsc{Rename}(\hookrightarrow, -)$ equates to a series of atomic circuit rewrites ($\rightarrowtail$). Semantically, we demonstrate that these rewrites preserve semantics ($\sqsupseteq$), akin to a soundness proof of rewrite rules.

Results presented in this section are formalized in Coq, unless explicitly stated otherwise.

6.1 Instrumented Circuits and Semantic Preservation

6.1.1 Instrumented circuits. To decouple qubit identities from their locations and support dynamic qubit allocation/discard, we instrument a circuit with lists of I/O qubits, which serve as maps from qubit to its locations. These I/O qubit lists should be consistent with the instructions in the circuit.

Formally, the instrumented circuits, denoted by $C : \mathit{In} \rightsquigarrow \mathit{Out}$[2], is inductively defined as follows.

$$\frac{\mathsf{NoDup}(\mathit{In}) \qquad \mathit{In} \equiv_p \mathit{Out}}{\mathsf{nil} : \mathit{In} \rightsquigarrow \mathit{Out}} \qquad \frac{\mathsf{NoDup}(\vec{q}) \qquad \vec{q} \subseteq \mathit{In} \qquad C : \mathit{In} \rightsquigarrow \mathit{Out}}{U[\vec{q}]; C : \mathit{In} \rightsquigarrow \mathit{Out}}$$

$$\frac{q \in \mathit{In} \qquad C : \mathit{In} \rightsquigarrow \mathit{Out}}{\mathbf{alloc}[q]; C : \mathit{In} \setminus \{q\} \rightsquigarrow \mathit{Out}} \qquad \frac{q \in \mathit{In} \qquad C : A \setminus \{q\} \rightsquigarrow \mathit{Out}}{\mathbf{discard}[q]; C : \mathit{In} \rightsquigarrow \mathit{Out}}$$

Here $A \equiv_p B$ means list A is a permutation of list B.

Intuitively, the I/O list of qubits represents the *living* qubit at the beginning/end of the circuit, respectively. An empty circuit or a unitary gate does not change living qubits. An $\mathbf{alloc}[q]$ instruction brings a dead qubit q alive, while $\mathbf{discard}[q]$ kills an alive q. Instructions other than $\mathbf{alloc}[q]$ must operate on living qubits. We often denote $(C : \mathit{In} \rightsquigarrow \mathit{Out})$ by $\widetilde{C}$ when In and Out are irrelevant.

[2]In our Coq development, an instrumented circuit is a triple $(C, \mathit{In}, \mathit{Out})$ that satisfies a similarly defined property.

6.1.2 Denotational semantics. We consider a symmetric monoidal category as the semantic domain. This approach follows the work [Selinger 2004], where the semantic domain is a category of superoperators. However, since our focus is on the symmetric monoidal structure rather than a specific instantiation, we do not assume a concrete category instance. Nevertheless, we introduce a specific instantiation, **SuperOp**, to aid in understanding.

Semantic domain: symmetric monoidal categories. Concretely, the category **SuperOp** has its objects natural numbers. Each object n can be interpreted as the number of qubits in a system, or the Hilbert space $\mathcal{H}_{2^n}$ where an n-qubit system resides. Special objects in **SuperOp** includes $\mathbf{I} = 0$, and $\mathbf{qbit} = 1$. A morphism from m to n is a superoperator $\mathcal{E} : \mathcal{H}_{2^m} \to \mathcal{H}_{2^n}$.

Additionaly, **SuperOp** has a symmetric monoidal structure. The product $\otimes$ over objects m and n is defined as $m \otimes n = m + n$. The unit of this product is the object $\mathbf{I} = 0$. Given morphisms $\mathcal{E}_1 \in \mathrm{Hom}(m_1, n_1)$, and $\mathcal{E}_2 \in \mathrm{Hom}(m_2, n_2)$, the product $\otimes$ over morphisms $\mathcal{E}_1 \otimes \mathcal{E}_2 \in \mathrm{Hom}(m_1 \otimes m_2, n_1 \otimes n_2)$ is defined on a basis elements $e_1 \otimes e_2$ via $(\mathcal{E}_1 \otimes \mathcal{E}_2)(e_1 \otimes e_2) = \mathcal{E}_1(e_1) \otimes \mathcal{E}_2(e_2)$, and extends to arbitrary elements by linearity. Finally, the symmetric braiding (or twist) β is defined on basis elements $e_1 \otimes e_2$ via $\beta(e_1 \otimes e_2) = e_2 \otimes e_1$.

Using the category **SuperOp**, the denotation of each individual instruction is shown below, originally formalized in [Selinger 2004]. A qubit q is associated with the object **qbit**: $[\![q]\!] = \mathbf{qbit}$; and a list of n qubits is associated with the tensor product of **qbit**s: $[\![\vec{q}]\!] = [\![\vec{q}[1]]\!] \otimes \ldots \otimes [\![\vec{q}[n]]\!] = \mathbf{qbit}^n$. In particular, and empty list of qubits nil is associated with the tensor unit: $[\![\mathsf{nil}]\!] = \mathbf{I}$.

$$[\![\mathbf{alloc}[q]]\!] \in \mathrm{Hom}([\![\mathsf{nil}]\!], [\![[q]]\!]) \quad : \quad a \mapsto \left[\begin{smallmatrix} a & 0 \\ 0 & 0 \end{smallmatrix}\right]$$

$$[\![\mathbf{discard}[q]]\!] \in \mathrm{Hom}([\![[q]]\!], [\![\mathsf{nil}]\!]) \quad : \quad \left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right] \mapsto a + d$$

$$[\![U[\vec{q}]]\!] \in \mathrm{End}([\![\vec{q}]\!]) \quad : \quad A \mapsto UAU^\dagger$$

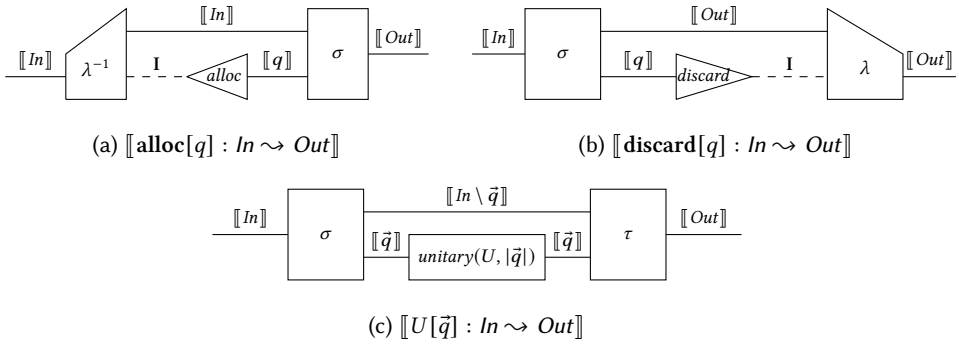
$$\mathit{permute}_\sigma \in \mathrm{Hom}(X_1 \otimes X_2 \otimes \ldots \otimes X_n,\ X_{\sigma(1)} \otimes X_{\sigma(2)} \otimes \ldots \otimes X_{\sigma(n)})$$

Here, we also introduce a family of special morphisms $\mathit{permute}_\sigma$ for later defining semantic lifting. This is the natural permutation map based on the symmetric tensor $\otimes$.

We further generalize our semantic domain to an arbitrary symmetric monoidal category, since a symmetrical monoidal structure is sufficient for proving correctness of qubit recycling. We refer to the book [Etingof et al. 2016] for more details about symmetric monoidal categories. This generalization is reasonable because, commonly used semantic domains for quantum programs [Abramsky and Coecke 2004; Heunen and Vicary 2019; Selinger 2004, 2005] share a symmetric monoidal structure. The remainder contents of this section depends only on the properties of a symmetric monoidal category, without referring to the concrete instantiation **SuperOp**.

In detail, we parameterize over an arbitrary symmetric monoidal category **SMC** as the semantic domain, and assume a special object $\mathbf{qbit} \in \mathbf{SMC}$. In defining denotations for **alloc** and **discard**, we assume morphisms $\mathit{alloc} \in \mathrm{Hom}(\mathbf{I}, \mathbf{qbit})$ and $\mathit{discard} \in \mathrm{Hom}(\mathbf{qbit}, \mathbf{I})$. For unitary operations, we assume a partial map $\mathit{unitary}(U, n) : \mathrm{End}(\mathbf{qbit}^n) \cup \{\bot\}$, where an endomorphism is defined only if U can be applied over n qubits. The $\mathit{permute}_\sigma$ morphism generalizes to any morphism in **SMC** that implements the permutation σ, constructed using associator $\alpha_{X,Y,Z} : (X \otimes Y) \otimes Z \cong X \otimes (Y \otimes Z)$, left and right unitors $\lambda_X : \mathbf{I} \otimes X \cong X$ and $\rho_X : X \otimes \mathbf{I} \cong X$, or twist $\beta_{X,Y} : X \otimes Y \cong Y \otimes X$. By the coherence theorem of symmetric monoidal categories, these morphisms are identical, thus the generalized $\mathit{permute}_\sigma$ is well defined.

Semantic lifting. On top of an **SMC** and the parameterized objects and morphisms, we define the denotation of an instrumented circuit $(C : \mathit{In} \rightsquigarrow \mathit{Out})$ by lifting the morphisms of individual instructions to a morphism between the I/O qubits, using permutation and left/right unitors. This is to permute the input qubits In to make the arguments of an instruction adjacent and aligned in



(a) $[\![\textbf{alloc}[q] : In \rightsquigarrow Out]\!]$ (b) $[\![\textbf{discard}[q] : In \rightsquigarrow Out]\!]$

(c) $[\![U[\vec{q}] : In \rightsquigarrow Out]\!]$

Fig. 9. Graphical repr. of the lifting. Here λ is the left unitor, σ and τ are permutation morphisms.

order, such that the morphisms of the corresponding instruction is applicable. After applying the morphism, we again permutes the living qubits such that they are in the order defined by *Out*.

The denotational semantics lifted w.r.t. I/O is formalized below [3].

$$
\begin{aligned}
[\![\mathsf{nil} : In \rightsquigarrow Out]\!] &= permute_\sigma && \text{where } Out = \sigma(In)\\
[\![\textbf{alloc}[q] : In \rightsquigarrow Out]\!] &= permute_\sigma \circ alloc \otimes \mathsf{id} \circ \lambda^{-1} && \text{where } Out = \sigma(q; In)\\
[\![\textbf{discard}[q] : In \rightsquigarrow Out]\!] &= \lambda \circ discard \otimes \mathsf{id} \circ permute_\sigma && \text{where } q; Out = \sigma(In)\\
[\![U[\vec{q}] : In \rightsquigarrow Out]\!] &= permute_\tau \circ unitary(U, |\vec{q}|) \otimes \mathsf{id} \circ permute_\sigma \\
&\qquad \text{where } \sigma(In) = (\vec{q}; Mid) = \tau^{-1}(Out) \text{ for some } Mid\\
[\![C_1; C_2 : In \rightsquigarrow Out]\!] &= [\![C_2 : Mid \rightsquigarrow Out]\!] \circ [\![C_1 : In \rightsquigarrow Mid]\!] && \text{for some } Mid
\end{aligned}
$$

A denotation is defined only if the intermediate terms are all defined. It is straightforward to show that the permutations involved are uniquely determined, and the choice of the intermediate qubit lists *Mid* does not affect the results. Thus the above indeed defines a partial function over instrumented circuits.

Graphical representations for the lifted denotation of **alloc**, **discard** and U are depicted in Fig. 9.A line represents an object (or the identity morphism over that object) in **SMC**, which is labeled above the line. The **I** object is specially represented by a dashed line. Lines placed in juxtaposition means a tensor product of objects. A morphism is represented by a box or triangle, whose domain is the line coming from the left, and the codomain is the line outgoing to the right.

6.1.3 Semantic preservation. On top of the denotational semantics, semantic preservation is naturally defined as equivalence over the denotations modulo renaming and permutation.

Definition 6.1 (Semantic preservation). For any instrumented circuits $C : In \rightsquigarrow Out$ and $C' : In' \rightsquigarrow Out'$, the latter preserves the semantics of the former, denoted as $\widetilde{C} \sqsupseteq \widetilde{C'}$, if there exists bijections f, g, and permutations σ, τ, such that

(1) (I/O equivalent up to renaming and permutation) $In' = \sigma \circ f(In)$ and $\tau(Out') = g(Out)$; and
(2) (Safety preservation) if $[\![\widetilde{C}]\!]$ is defined, then $[\![\widetilde{C'}]\!]$ is defined; and
(3) (Semantics equal modulo permutation) $[\![\widetilde{C}]\!] = permute_\tau \circ [\![\widetilde{C'}]\!] \circ permute_\sigma$.

We prove that this semantic preservation is compositional.

[3]We omit the associators for clarity, since they are irrelevant by the coherence theorem of a monoidal category

Lemma 6.2 (Compositionality). *The relation $\sqsupseteq$ is transitive, and is congruent to sequential composition. The latter is, for any $\widetilde{C}_1, \widetilde{C}_1', \widetilde{C}_2, \widetilde{C}_2'$, we have $\widetilde{C}_1 \sqsupseteq \widetilde{C}_1' \wedge \widetilde{C}_2 \sqsupseteq \widetilde{C}_2' \implies \widetilde{C}_1;\widetilde{C}_2 \sqsupseteq \widetilde{C}_1';\widetilde{C}_2'$.*

Here $\widetilde{C};\widetilde{C}'$ is defined only if there is a Mid such that $\widetilde{C} = (C : In \rightsquigarrow Mid)$ and $\widetilde{C}' = (C' : Mid \rightsquigarrow Out)$.

6.2 Rewriting rules

We introduce two rewriting rules over instrumented circuits, and use relation $\widetilde{C} \rightarrowtail \widetilde{C}'$ to denote $\widetilde{C}'$ is obtained from $\widetilde{C}$ via a rewriting step.

$$\textsc{TopoDeform}\ \frac{In \equiv_p In' \qquad C \sim C' \qquad Out \equiv_p Out'}{(C : In \rightsquigarrow Out) \rightarrowtail (C' : In' \rightsquigarrow Out')} \qquad \textsc{Reuse}\ \frac{q' \notin (C_1 : In \rightsquigarrow Mid) \qquad q \notin (C_2 : Mid \rightsquigarrow Out)}{(C_1;C_2 : In \rightsquigarrow Out) \rightarrowtail (C_1;C_2 : In \rightsquigarrow Out)[q/q']}$$

Here $q \notin (C : In \rightsquigarrow Out)$ indicates that qubit q is not present in the sets *In* and *Out*, nor is it used as an argument in any instructions within C. This signifies that q is entirely irrelevant in the instrumented circuit, and its lifetime does not overlap with the execution of C. The notation $[q/q']$ denotes the substitution of each occurrence of q' with q.

The first rule involves topological deformation, and permits permutation on I/O qubits, while the second rule allows for the reuse of qubit q as q', given that the lifetime of q precedes that of q'.

We prove that these rewriting rules are sound. The proof relies on fundamental properties such as the interchange law and coherence theorem of a symmetric monoidal category.

Lemma 6.3 (Soundness of rewriting rules). *If $\widetilde{C} \rightarrowtail \widetilde{C}'$, then $\widetilde{C} \sqsupseteq \widetilde{C}'$.*

In the following subsections, we show that the overall behavior of our recycler is equivalent to a series of circuit rewritings using these two rules.

6.3 Sorting and Validity Checking

The Sort function takes a recycling strategy $\hookrightarrow$ and a circuit C as inputs. It tries to do topological deformation over C such that the reordered circuit C' is compatible with $\hookrightarrow$. That is, for each $q \hookrightarrow q'$, q' can reuse q in C'.

The algorithm Sort is outlined in Alg. 2. In line 1, the algorithm first checks two conditions for the input $\hookrightarrow$: (i) $\hookrightarrow$ is an injective function, and (ii) for each $q \hookrightarrow q'$, both **discard**$[q]$ and **alloc**$[q]$ are present in C. If both conditions are met, the circuit is translated into a DAG representation, denoted as G (line 2). Next, edges (**discard**$[q]$, **alloc**$[q']$) are added to G for each $q \hookrightarrow q'$, resulting in G' (line 3). The algorithm then performs a topological sorting using Kahn's algorithm (line 4). Kahn's algorithm guarantees that the circuit is reordered without violating computational dependencies, ensuring that **discard**$[q]$ occurs before **alloc**$[q']$ for each $q \hookrightarrow q'$. If the topological sorting succeeds, the algorithm returns the reordered circuit (line 5); otherwise, it detects a cycle in G' that invalidates $\hookrightarrow$.

Algorithm 2 Sort$(\hookrightarrow, \widetilde{C})$

```
1: if ↪ is a recycling strategy for C then
2:     G ← DAG of C
3:     G' ← G with edges defined by ↪
4:     if TopoSort(G') = OK(C') then
5:         return OK(C' : In ⇝ Out)
6:     end if
7: end if
8: return Invalid recycling strategy
```

We prove that Sort has the desired properties.

Lemma 6.4 (Sort returns a topological deformation of C.). *Given $C : In \rightsquigarrow Out$ and $\hookrightarrow$, if Sort$(\hookrightarrow, (C : In \rightsquigarrow Out)) = OK(C' : In \rightsquigarrow Out)$, then $C' \sim C$. By definition, $\widetilde{C} \rightarrowtail \widetilde{C}'$.*

Lemma 6.5 (Sort is a correct validator). *Given $\widetilde{C}$ and $\hookrightarrow$, if $\textsc{Sort}(\hookrightarrow, \widetilde{C}) = OK(\widetilde{C'})$ for some C', then for any $q \hookrightarrow q'$, q' can reuse q in C'.*

The inverse of the previous lemma also holds, which says Sort is a "complete" validator.

Lemma 6.6 (Sort succeeds when recycling strategy is valid). *Given $\widetilde{C}$ and $\hookrightarrow$, then $\hookrightarrow$ is valid w.r.t. C if and only if $\textsc{Sort}(\hookrightarrow, \widetilde{C})$ succeeds.*

The proof of Lm. 6.6 is not mechanized in Coq because it is irrelevant to semantic preservation. We sketch a proof in the supplementary material.

6.4 Reusing by Renaming

The $\textsc{Rename}(\hookrightarrow, \widetilde{C})$ function operates by iteratively popping $q \hookrightarrow q'$ from $\hookrightarrow$, and continue to the next iteration with $\textsc{Rename}(\hookrightarrow [q/q'], \widetilde{C}[q/q'])$, until $\hookrightarrow$ is empty and returns $\widetilde{C}$. Notably, Rename is a total function, it preserves semantics only if the inputs are appropriate. The following lemma establishes that when the recycling strategy $\hookrightarrow$ is valid, the Rename function is equivalent to a series of atomic rewrites (using the Reuse rule) over the instrumented circuit $\widetilde{C}$.

Lemma 6.7 (Rename is equivalent to a series rewrites). *For any $\hookrightarrow$ and $\widetilde{C}$, if q' can reuse q in C for any $q \hookrightarrow q'$, then $\widetilde{C} \rightarrowtail^* \textsc{Rename}(\hookrightarrow, \widetilde{C})$.*

6.5 Final Theorem and Coq Development

Putting the previous results together, we obtain the final theorem for our certified qubit recycler.

Theorem 6.8. *For any recycling strategy $\hookrightarrow$ and instrumented circuit $\widetilde{C}$ and $\widetilde{C'}$,*

$$\textsc{Rewriter}(\hookrightarrow, \widetilde{C}) = \mathsf{OK}(\widetilde{C'}) \implies \widetilde{C} \sqsupseteq \widetilde{C'}.$$

We implemented Rewriter and mechanized Thm. 6.8 in the Coq proof assistant, with ~6k lines of Coq code. The mechanization is built on top of an axiom-free formalization of category theory in Coq [Wiegley 2022]. Notable byproducts of this Coq formalization include the implementation and verification of Kahn's algorithm for topological sorting, and version of the coherence theorem for symmetric monoidal categories.

We extracted Rewriter to OCaml such that it can work together with the Solver in Sec. 7.

7 SOLVER FOR FINDING RECYCLING STRATEGIES

This section presents our algorithm for finding recycling strategies, and an empirical evaluation on the RevLib [Wille et al. 2008] benchmark.

7.1 The Algorithm

Our algorithm is outlined in Alg. 3. The Solver is inspired by Lm. 5.5, that finding recycling strategies on QDGs is equivalent to a triangularization problem. The Solver is essentially a polynomial time algorithm for the triangularization problem given a row permutation P. It implements the procedure demonstrated in Fig. 4: given a heuristic function f (which essentially computes a permutation P one row at a time), and an $n \times n$ matrix A representing a QDG, and a current state σ, it iteratively choose a row r using f, take a step to state σ', and continue the procedure.

A state σ is a tuple $(\texttt{rows}, \texttt{cols}, \texttt{cols\_del}, \texttt{k})$. Here `rows` is a list of row indices already chosen, `cols` is the list of remaining columns, `cols_del` is the list of deleted columns, and `k` is an upper-bound of the number of further steps.

Algorithm 3 Solver(f, A, σ)

1: **if** not $halt(\sigma)$ **then**
2: $\quad r \leftarrow f(A, \sigma)$
3: $\quad \sigma \xrightarrow{r}_A \sigma'$
4: $\quad$ Solver(f, A, σ')
5: **else return** Extract(σ)
6: **end if**

Algorithm 4 Extract(σ)

1: $rl \leftarrow \sigma.\texttt{rows}$
2: $k \leftarrow |rl|$
3: $cl \leftarrow \sigma.\texttt{cols\_del} +\!\!+ \sigma.\texttt{cols}$
4: **return** $\{(cl[n-k+i], rl[i]) \mid i = 0, 1, ..., k-1\}$

The initial state is $\sigma_0 = ([], [0, ..., n-1], [], n)$, and $halt(\sigma)$ if $\sigma.\texttt{k} \le 0$. The stepping rule is:

$$\frac{\begin{array}{c} r \notin \texttt{rows} \qquad \texttt{cols}' = \texttt{cols} \backslash \{j \mid A_{rj} = 1\} \qquad \texttt{cols\_del}' = \texttt{cols\_del} +\!\!+ (\texttt{cols} \backslash \texttt{cols}') \\ \texttt{rows}' = (\texttt{cols}' = [] \ ? \ \texttt{rows} : (\texttt{rows} +\!\!+ [r])) \qquad \texttt{k}' = \min(\texttt{k}, |\texttt{cols}'|) - 1 \end{array}}{(\texttt{rows}, \texttt{cols}, \texttt{cols\_del}, \texttt{k}) \xrightarrow{r}_A (\texttt{rows}', \texttt{cols'}, \texttt{cols\_del}', \texttt{k}')}$$

In each step, given a next row r, it delete those columns j such that $A_{rj} = 1$, and update the list for remaining columns (`col`) and deleted columns (`col_del`). It then append r to `rows` if the remaining columns `col'` is not empty. Finally, it updates `k` to the minimum of `k` $- 1$ and $|$`cols'`$| - 1$.

When reaching a halting state σ_h, the solver extracts the solution from σ_h, as described in Alg. 4. The size of the extracted recycling strategy corresponds to the number of overall steps.

We illustrate an execution of this algorithm using the QDG presented in Sec. 2.2 as an example. We assume the heuristic f chooses rows $(3, 2, 1, 0)$ in order.

	0	1	2	3
0	1	1	1	1
1	1	1	0	1
2	1	0	1	1
3	1	0	0	1

	r	row	col	col_del	k
σ_0	3	[]	$[0, 1, 2, 3]$	[]	4
σ_1	2	$[3]$	$[1, 2]$	$[0, 3]$	1
σ_2	–	$[3, 2]$	$[1]$	$[0, 3, 2]$	-1

The algorithm halts after 2 steps. Extracting the result from σ_2 yields $\{(2, 3), (1, 2)\}$, a solution of size 2 and coincides with our previous solution.

We also introduce the "dual-circuit" technique introduced in [DeCross et al. 2022] to our settings: by recognizing the symmetry in rows and columns, we apply the solver to the transpose of A. Extracting a solution involves swapping the resulting pairs. Comparing the solutions obtained from A and its transpose A^T, we select the larger one.

7.1.1 Designs of the heuristic. The only remaining problem is to find an appropriate heuristic f such that the number of overall steps is maximized. We design heuristics based on the following observations:

(1) (`Greedy`) The bound `k` in the state strictly decreases as the algorithm proceeds, and the algorithm halts if `k` ≤ 0. Therefore, we may choose the next row that maximizes the bound `k` in the next step, such that the algorithm is likely to step more.
(2) (`Max0s`) The more 0s there are in the submatrix A[`rows'`][`cols'`], it is more likely for the algorithm to continue stepping. Therefore, we may choose the next row that maximizes the number of 0s in A[`rows'`][`cols'`].
(3) (`LA`) In experiments, it is common that several rows perform equally. In these cases, we may break a tie by taking a further step (look-ahead).

Later in this section, we evaluate these heuristics, which yield optimal results most of the time.

7.2 An ILP Model and Upper-bound Estimator for Qubit Recycling

To evaluate the quality of the solutions produced by our heuristic-based solver, we first try to find an optimal solution by translating qubit recycling problem into an ILP model, and solve the model using Gurobi optimizer. When the problem size is unfeasible for finding optimal solutions, we provide an upper-bound estimation for the size of an optimal solution.

7.2.1 ILP model. Recall that given a QDG in form of a $n \times n$ 0-1 matrix A, to find a maximal valid recycling strategy is to find a partial permutation matrix R (representing an injective partial map) such that AR is nilpotent, and the rank of R is maximized. This description can be roughly transformed into an ILP problem on the right. Here R consists of $n \times n$ 0-1 variables, and the constraints $\mathbf{1}^T \cdot R \le \mathbf{1}$ and $R \cdot \mathbf{1} \le \mathbf{1}$ is to guarantee that R is a partial permutation. The main difficulty is to translate "nilpotent" into a linear constraint.

maximize	$\mathbf{1}^T \cdot R \cdot \mathbf{1}$
subject to	AR nilpotent
	$\mathbf{1}^T \cdot R \le \mathbf{1}$
	$R \cdot \mathbf{1} \le \mathbf{1}$

We achieve this using the fact [Bang-Jensen and Gutin 2008] that: every acyclic graph has an acyclic ordering. That is, an ordering $p_1, ..., p_n$ such that for every $p_i \to p_j$, we have $i < j$. Since the nilpotent constraint is equivalent to requiring the digraph represented by AR is nilpotent, we introduce another n integer variables p_i representing the i-th qubit in an acyclic ordering, and the nilpotent constraint becomes: $\forall i, j.\ (AR)_{i,j} = 1 \implies p_i < p_j$, and additional constraints $\forall i.\ 1 \le p_i \le n$ and $\forall i, j.\ p_i \ne p_j$. These constraints can be easily translated into $O(n^2)$ linear constraints at a cost of additional $O(n^2)$ variables.

The full ILP model can be found in the supplementary material. Our model is more compact ($O(n^2)$ variables and constraints) compared with that of [DeCross et al. 2022] ($O(n^2)$ variables and $O(n^4)$ constraints), thus is potentially easier to solve.

7.2.2 Upper-bound estimator. If we are to have a size k solution for a QDG A, by Lm. 5.5, there must be a strictly upper-triangular submatrix B of order k. Therefore, there must be a row i_k with more than k 0s, and a column j_k with more than k 0s. Among the rest of rows and columns, there must be a row i_{k-1} with more than $k-1$ 0s, etc. Based on this idea, we have an upper-bound estimation for the size of the optimal solution. We count the number of 0s of each rows, and sort the counts in descending order obtaining $r_1, r_2, ..., r_n$. We do the same for the columns, and obtain $c_1, c_2, ..., c_n$. The upper bound is $\hat{k} = \min_{i=1,2,...,n}(\min(r_i, c_i) + 2(i-1))$.

7.3 Experimental Evaluation

We implement Alg. 3 and the heuristics, namely, `Greedy`, `Max0s`, and their look-ahead version `Greedy+LA` and `Max0s+LA`. We evaluated these algorithms on the 84 circuits in the RevLib [Wille et al. 2008] benchmark reported in [Paler et al. 2016]. For small circuits, we solve their ILP model using Gurobi optimizer to find optimal solutions. We also compute the upper-bound estimation in case when optimal solutions are unfeasible to find,

Selected results, presented in Tab. 1, are chosen due to either the unavailability of an optimal solution within a reasonable timeframe (10 hours on a laptop) or the inconsistent performance of the four heuristics. The results shows that our algorithms outperform [Paler et al. 2016]'s, by up to 49% or 44 qubits. Several notable observations emerge from these results:

- Both `Greedy` and `Max0s` outperform the other.
- `LA` improves `Greedy` often but not `Max0s`.
- Our upper-bound estimation is tight for certain circuits that Gurobi cannot efficiently solve.

For the remaining 75 circuits out of 84, all four of our heuristics consistently reach the optimal solution, and our upper-bound estimation is tight for more than 80% circuits. In contrast, [Paler et al. 2016] fails to find an optimal solution in 12 out of these 75 circuits, and erroneously surpasses

Table 1. Selected results of the evaluation on the RevLib benchmark. For each circuit, we list the total number of qubits (# Qubits), the number of recycled qubits in [Paler et al. 2016]'s work, using `Greedy`, `Max0s`, and these two heuristics combined with look-ahead `LA` to break a tie. We also list the optimal solution (if any) given by the Gurobi optimizer, and our estimated upper-bound. The best results among the methods are in green background. A full table of results is in the supplementary material.

Circuit	#Qubits	[Paler et al. 2016]	Greedy	Max0s	Greedy+LA	Max0s+LA	Gurobi	Bound
pdc_307	619	464	505	508	505	508	N/A	575
spla_315	489	401	407	407	407	407	N/A	433
lu_326	299	194	196	196	196	196	N/A	196
hwb9_304	170	81	121	119	119	119	N/A	144
ex5p_296	206	107	127	125	127	125	N/A	137
e64-bdd_295	195	114	126	126	126	126	N/A	128
hwb8_303	112	52	73	73	73	73	N/A	89
hwb7_302	73	31	45	44	45	44	45	53
hwb6_301	46	20	22	22	23	22	23	27

the optimal solution in 2 circuits. We elaborate on these erroneous results in the supplementary material, which stem from a bug in their implementation.

Evaluation of randomly generated adjacency matrices reaffirmed the first two observations. For example, on 100 instances of 320×320 matrices with 0.8 sparsity, `Max0s`, `Max0s+LA`, and `Greedy+LA` exhibit an average solution size improvement of 0.46, 0.49, and 0.78 respectively compared with `Greedy`. However, there are certain instances where `Greedy` outperforms the other three.

We also generalize and implement the algorithm proposed in another previous work [DeCross et al. 2022]. The generalized algorithm performs identically to `Greedy` on the RevLib benchmark. This outcome is intuitively reasonable since their algorithm essentially selects rows to maximize the cardinality of $|\texttt{cols}'|$, while `Greedy` chooses rows to maximize $\min(|\texttt{cols}'|, k)$.

8 RELATED WORK

Quantum circuit optimization. There is rich literature on quantum circuit optimizations for reducing various kinds of costs. Many works [Häner et al. 2020; Nam et al. 2018; Xu et al. 2023, 2022] focus on minimizing the total gate count or circuit depth. In fault-tolerant quantum computing, where T-gates are expensive resources, the reduction of T-gate count is highly desirable, as demonstrated in works like [Gosset et al. 2014; Häner and Soeken 2022].

Regarding the specific problem of minimizing circuit width, the most related works include REVS [Parent et al. 2015], SQUARE [Ding et al. 2020], CaQR [Hua et al. 2023], and two other works [DeCross et al. 2022; Paler et al. 2016] compared in Sec. 7.3. These works address qubit allocation and reclamation problems from various perspectives: REVS focuses on minimizing memory footprint of reversible networks, SQUARE focuses on uncomputation for creating opportunities for qubit reuse, CaQR focuses on improving circuit efficacy and fidelity leveraging mid-circuit measurement, [DeCross et al. 2022; Paler et al. 2016] studies qubit recycling in a similar setting to ours with a focus on algorithm design. Notably, our evaluation results demonstrates that the generalized version of the algorithm in [DeCross et al. 2022] performs well on realistic benchmarks. In contrast, our work can be viewed as an improvement over [DeCross et al. 2022] in that we provide a systematic way of designing heuristic based algorithms that performs even better, and an improved ILP model for finding optimal solutions. Moreover, none of the above works study qubit recycling from complexity or verification perspectives as we did.

Related problems in classical compiler optimizations. Two closely related problems in classical compiler optimizations are minimum register instruction sequencing (MRIS) [Govindarajan et al. 2003], and register saturation (RS) [Touati 2005]. MRIS is very similar to qubit recycling, it asks to find a scheduling of instructions that requires minimum registers. However, the complexity of MRIS is not known. Opposite to qubit recycling, RS is an NP-hard problem that asks for the upper bound of needed registers in all possible scheduling. Another closely related problem is optimal code generation for DAGs. The problem is known to be NP-complete [Bruno and Sethi 1976], but its objective is to minimize code length instead of the number of registers.

Related problem in the language of monoidal categories. In the context of monoidal categories, a closely related problem is computing the monoidal width [Lavore and Sobociński 2023]. Monoidal width quantifies the complexity of decomposing morphisms within monoidal categories, encompassing structural width measures for graphs like tree width and rank width. While monoidal width penalizes the composition operation along "large" objects and encourages the use of monoidal products, the qubit recycling problem seeks to minimize circuit width by penalizing the usage of monoidal products. For instance, in monoidal width, $w(f \otimes g)$ is determined by the $\max(w(f), w(g))$ using a weight function w, whereas in circuit analysis, a proper definition capturing circuit width is given by $w(f) + w(g)$. Exploring the generalization of the qubit recycling problem to other settings presents an intriguing avenue for further investigation.

Compiler verification for quantum programs. There are numerous works on compiler verification and formulation of compiler correctness for various scenarios [Patterson and Ahmed 2019]. Few work targets the quantum settings. Amy et. al. [Amy et al. 2017] verified a compiler from Boolean expressions to reversible circuits in F*, with an aim to reduce circuit width. However, their verification uses a classical semantic model, where a state is of type $\mathbb{N} \to \mathbb{B}$. ReQWIRE [Rand et al. 2018], presents methods for verifying that ancillae are discarded in the desired state, and implements a verified compiler from classical functions to quantum oracles. Their semantics is parameterized with a context that maps variables to wire indices, serving a similar purpose as our *In*, *Out* lists. However, there is no notion for semantic preservation between quantum circuits in ReQWIRE, and it is not clear to us how their approach facilitates qubit recycling. VOQC [Hietala et al. 2021] and Giallar [Tao et al. 2022] aim to verify practical quantum circuit optimizers [Aleksandrowicz et al. 2019; Nam et al. 2018]. VOQC follows a CompCert-like approach that verifies each single optimization pass manually using sophisticated tactics; while Giallar seeks an almost automatic solution. As discussed previously, the verification techniques in these two works are not directly applicable to qubit recycling, where renaming and dynamic qubit allocation/discard are involved.

9 CONCLUSION AND FUTURE WORK

Qubit recycling involves finding a topologically identical quantum circuit that maximizes qubit reuse. By translating the problem to a matrix triangularization problem based on qubit dependency graphs, we demonstrated the NP-hardness of this problem. Additionally, we have developed a certified qubit recycler in Coq, This qubit recycler integrates validation and verification approaches. Byproducts of the certification include a verified implementation of Kahn's topological sort algorithm, and a mechanized proof of a version of the coherence theorem of symmetric monoidal categories. Our qubit recycler reaches optimal solutions for the majority circuits in the RevLib benchmark.

While our focus has been on topologically identical quantum circuits, where qubit recycling opportunities arise from disjoint instruction swaps, there is potential to further enhance qubit reuse when a broader class of semantically equivalent quantum circuits is introduced. This requires the use of a semantic domain with richer structure, e.g., the ZX-calculus [Coecke and Kissinger 2017] or Quon [Liu et al. 2017].

REFERENCES

Samson Abramsky and Bob Coecke. 2004. A Categorical Semantics of Quantum Protocols. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*. IEEE Computer Society, 415–425. https://doi.org/10.1109/LICS.2004.1319636

Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernández, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. *Qiskit: An Open-source Framework for Quantum Computing*. https://doi.org/10.5281/zenodo.2562111

Matthew Amy, Martin Roetteler, and Krysta M. Svore. 2017. Verified Compilation of Space-Efficient Reversible Circuits. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 10427)*, Rupak Majumdar and Viktor Kuncak (Eds.). Springer, 3–21. https://doi.org/10.1007/978-3-319-63390-9_1

Jrgen Bang-Jensen and Gregory Z. Gutin. 2008. *Digraphs: Theory, Algorithms and Applications* (2nd ed.). Springer Publishing Company, Incorporated.

Graham R. Brightwell and Peter Winkler. 1991. Counting Linear Extensions is #P-Complete. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*, Cris Koutsougeras and Jeffrey Scott Vitter (Eds.). ACM, 175–181. https://doi.org/10.1145/103418.103441

John Bruno and Ravi Sethi. 1976. Code Generation for a One-Register Machine. *J. ACM* 23, 3 (jul 1976), 502–510. https://doi.org/10.1145/321958.321971

Bob Coecke and Aleks Kissinger. 2017. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press. https://doi.org/10.1017/9781316219317

Matthew DeCross, Eli Chertkov, Megan Kohagen, and Michael Foss-Feig. 2022. Qubit-reuse compilation with mid-circuit measurement and reset. arXiv:2210.08039 [quant-ph]

Yongshan Ding, Xin-Chuan Wu, Adam Holmes, Ash Wiseth, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. 2020. SQUARE: Strategic Quantum Ancilla Reuse for Modular Quantum Programs via Cost-Effective Uncomputation. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 570–583. https://doi.org/10.1109/ISCA45697.2020.00054

Pavel Etingof, Shlomo Gelaki, Dmitri Nikshych, and Victor Ostrik. 2016. *Tensor categories*. Vol. 205. American Mathematical Soc.

Guillaume Fertin, Irena Rusu, and Stéphane Vialette. 2015. Obtaining a Triangular Matrix by Independent Row-Column Permutations. In *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9472)*, Khaled M. Elbassioni and Kazuhisa Makino (Eds.). Springer, 165–175. https://doi.org/10.1007/978-3-662-48971-0_15

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A* 86 (Sep 2012), 032324. Issue 3. https://doi.org/10.1103/PhysRevA.86.032324

David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. 2014. An Algorithm for the T-Count. *Quantum Info. Comput.* 14, 15–16 (nov 2014), 1261–1276.

Ramaswamy Govindarajan, Hongbo Yang, José Nelson Amaral, Chihong Zhang, and Guang R. Gao. 2003. Minimum Register Instruction Sequencing to Reduce Register Spills in Out-of-Order Issue Superscalar Architectures. *IEEE Trans. Computers* 52, 1 (2003), 4–20. https://doi.org/10.1109/TC.2003.1159750

Ramsey W. Haddad. 1990. *Triangularization: a two-processor scheduling problem*. Ph. D. Dissertation. Stanford University, USA. https://searchworks.stanford.edu/view/507223

Thomas Häner, Torsten Hoefler, and Matthias Troyer. 2020. Assertion-based optimization of Quantum programs. *Proc. ACM Program. Lang.* 4, OOPSLA (2020), 133:1–133:20. https://doi.org/10.1145/3428201

Thomas Häner and Mathias Soeken. 2022. Lowering the T-Depth of Quantum Circuits via Logic Network Optimization. *ACM Transactions on Quantum Computing* 3, 2, Article 6 (Mar 2022), 15 pages. https://doi.org/10.1145/3501334

Chris Heunen and Jamie Vicary. 2019. *Categories for Quantum Theory: An Introduction.* Oxford University Press. https://doi.org/10.1093/oso/9780198739623.001.0001

Kesha Hietala, Robert Rand, Shih-Han Hung, Xiaodi Wu, and Michael Hicks. 2021. A Verified Optimizer for Quantum Circuits. *Proc. ACM Program. Lang.* 5, POPL, Article 37 (Jan 2021), 29 pages. https://doi.org/10.1145/3434318

Fei Hua, Yuwei Jin, Yan-Hao Chen, Suhas Vittal, Kevin Krsulich, Lev S. Bishop, John Lapeyre, Ali Javadi-Abhari, and Eddy Z. Zhang. 2023. CaQR: A Compiler-Assisted Approach for Qubit Reuse through Dynamic Circuit. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023, Vancouver, BC, Canada, March 25-29, 2023*, Tor M. Aamodt, Natalie D. Enright Jerger, and Michael M. Swift (Eds.). ACM, 59–71. https://doi.org/10.1145/3582016.3582030

Dominik Janzing, Pawel Wocjan, and Thomas Beth. 2003. Identity check is QMA-complete. https://doi.org/10.48550/ARXIV.QUANT-PH/0305050

A. B. Kahn. 1962. Topological Sorting of Large Networks. *Commun. ACM* 5, 11 (nov 1962), 558–562. https://doi.org/10.1145/368996.369025

Aleks Kissinger and John van de Wetering. 2020. PyZX: Large Scale Automated Diagrammatic Reasoning. *Electronic Proceedings in Theoretical Computer Science* 318 (May 2020), 229–241. https://doi.org/10.4204/eptcs.318.14

Elena Di Lavore and Paweł Sobociński. 2023. Monoidal Width. arXiv:2212.13229 [cs.LO]

Xavier Leroy. 2009. Formal Verification of a Realistic Compiler. *Commun. ACM* 52, 7 (Jul 2009), 107–115. https://doi.org/10.1145/1538788.1538814

Zhengwei Liu, Alex Wozniakowski, and Arthur M. Jaffe. 2017. Quon 3D language for quantum information. *Proceedings of the National Academy of Sciences* 114, 10 (2017), 2497–2502. https://doi.org/10.1073/pnas.1621345114 arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.1621345114

Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs, and Dmitri Maslov. 2018. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information* 4, 1 (2018), 23. https://doi.org/10.1038/s41534-018-0072-4

Alexandru Paler, Robert Wille, and Simon J. Devitt. 2016. Wire recycling for quantum circuit optimization. *Phys. Rev. A* 94 (Oct 2016), 042337. Issue 4. https://doi.org/10.1103/PhysRevA.94.042337

Alex Parent, Martin Roetteler, and Krysta M. Svore. 2015. Reversible circuit compilation with space constraints. arXiv:1510.00377 [quant-ph]

Daniel Patterson and Amal Ahmed. 2019. The next 700 Compiler Correctness Theorems (Functional Pearl). *Proc. ACM Program. Lang.* 3, ICFP, Article 85 (Jul 2019), 29 pages. https://doi.org/10.1145/3341689

John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (Aug. 2018), 79. https://doi.org/10.22331/q-2018-08-06-79

Robert Rand, Jennifer Paykin, Dong-Ho Lee, and Steve Zdancewic. 2018. ReQWIRE: Reasoning about Reversible Quantum Circuits. In *Proceedings 15th International Conference on Quantum Physics and Logic, QPL 2018, Halifax, Canada, 3-7th June 2018 (EPTCS, Vol. 287)*, Peter Selinger and Giulio Chiribella (Eds.). 299–312. https://doi.org/10.4204/EPTCS.287.17

Silvain Rideau and Xavier Leroy. 2010. Validating Register Allocation and Spilling. In *Compiler Construction, 19th International Conference, CC 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6011)*, Rajiv Gupta (Ed.). Springer, 224–243. https://doi.org/10.1007/978-3-642-11970-5_13

Peter Selinger. 2004. Towards a Quantum Programming Language. *Mathematical. Structures in Comp. Sci.* 14, 4 (aug 2004), 527–586. https://doi.org/10.1017/S0960129504004256

Peter Selinger. 2005. Dagger Compact Closed Categories and Completely Positive Maps: (Extended Abstract). In *Proceedings of the 3rd International Workshop on Quantum Programming Languages, QPL 2005, DePaul University, Chicago, USA, June 30 - July 1, 2005 (Electronic Notes in Theoretical Computer Science, Vol. 170)*, Peter Selinger (Ed.). Elsevier, 139–163. https://doi.org/10.1016/j.entcs.2006.12.018

Runzhou Tao, Yunong Shi, Jianan Yao, Xupeng Li, Ali Javadi-Abhari, Andrew W. Cross, Frederic T. Chong, and Ronghui Gu. 2022. Giallar: push-button verification for the qiskit Quantum compiler. In *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*, Ranjit Jhala and Isil Dillig (Eds.). ACM, 641–656. https://doi.org/10.1145/3519939.3523431

Sid Ahmed Ali Touati. 2005. On the Optimality of Register Saturation. *Electron. Notes Theor. Comput. Sci.* 132, 1 (2005), 131–148. https://doi.org/10.1016/j.entcs.2005.01.033

John Wiegley. 2022. An axiom-free formalization of category theory in Coq. https://github.com/jwiegley/category-theory. Version 1.0.0.

H.S. Wilf. 1997. *On Crossing Numbers, and some Unsolved Problems.* Cambridge University Press, 557–562. https://doi.org/10.1017/CBO9780511662034.049

Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. 2008. RevLib: An Online Resource for Reversible Functions and Reversible Circuits. In *38th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2008), 22-23 May 2008, Dallas, Texas, USA*. IEEE Computer Society, 220–225. https://doi.org/10.1109/ISMVL.2008.43

Amanda Xu, Abtin Molavi, Lauren Pick, Swamit Tannu, and Aws Albarghouthi. 2023. Synthesizing Quantum-Circuit Optimizers. *Proc. ACM Program. Lang.* 7, PLDI, Article 140 (jun 2023), 25 pages. https://doi.org/10.1145/3591254

Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. 2022. Quartz: superoptimization of Quantum circuits. In *PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022*, Ranjit Jhala and Isil Dillig (Eds.). ACM, 625–640. https://doi.org/10.1145/3519939.3523433