The Classification of Reversible Bit Operations

Scott Aaronson[*1], Daniel Grier[†2], and Luke Schaeffer[3]

1 **University of Texas, Austin, USA**
aaronson@cs.utexas.edu

2 **Massachusets Institute of Technology, Cambridge, USA**
grierd@mit.edu

3 **Massachusets Institute of Technology, Cambridge, USA**
lrs@mit.edu

Abstract

We present a complete classification of all possible sets of classical reversible gates acting on bits, in terms of which reversible transformations they generate, assuming swaps and ancilla bits are available for free. Our classification can be seen as the reversible-computing analogue of *Post's lattice*, a central result in mathematical logic from the 1940s. It is a step toward the ambitious goal of classifying all possible quantum gate sets acting on qubits.

Our theorem implies a linear-time algorithm (which we have implemented), that takes as input the truth tables of reversible gates G and H, and that decides whether G generates H. Previously, this problem was not even known to be decidable (though with effort, one can derive from abstract considerations an algorithm that takes triply-exponential time). The theorem also implies that any n-bit reversible circuit can be "compressed" to an equivalent circuit, over the same gates, that uses at most $2^n \operatorname{poly}(n)$ gates and $O(1)$ ancilla bits; these are the first upper bounds on these quantities known, and are close to optimal. Finally, the theorem implies that every non-degenerate reversible gate can implement either every reversible transformation, or every affine transformation, when restricted to an "encoded subspace."

Briefly, the theorem says that every set of reversible gates generates either all reversible transformations on n-bit strings (as the Toffoli gate does); no transformations; all transformations that preserve Hamming weight (as the Fredkin gate does); all transformations that preserve Hamming weight mod k for some k; all affine transformations (as the Controlled-NOT gate does); all affine transformations that preserve Hamming weight mod 2 or mod 4, inner products mod 2, or a combination thereof; or a previous class augmented by a NOT or NOTNOT gate. Prior to this work, it was not even known that every class was finitely generated. Ruling out the possibility of additional classes, not in the list, requires involved arguments about polynomials, lattices, and Diophantine equations.

Due to the length of the proof, some parts of it have been omitted and may be found in the full version of the paper online.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Reversible computation, Reversible gates, Circuit synthesis, Gate classification, Boolean logic, Post's lattice

Digital Object Identifier 10.4230/LIPIcs.ITCS.2017.23

* This work was done while the author was at MIT. Supported by an Alan T. Waterman Award from the National Science Foundation, under grant no. 1249349.
† Supported by an NSF Graduate Research Fellowship under grant no. 1122374.

1 Introduction

The *pervasiveness of universality* – that is, the likelihood that a small number of simple operations already generate all operations in some relevant class – is one of the central phenomena in computer science. It appears, among other places, in the ability of simple logic gates to generate all Boolean functions (and of simple quantum gates to generate all unitary transformations); and in the simplicity of the rule sets that lead to Turing-universality, or to formal systems to which Gödel's theorems apply. Yet precisely because universality is so pervasive, it is often more interesting to understand the ways in which systems can *fail* to be universal.

In 1941, the great logician Emil Post [23] published a complete classification of all the ways in which sets of Boolean logic gates can fail to be universal: for example, by being monotone (like the AND and OR gates) or by being affine over $\mathbb{F}_2$ (like NOT and XOR). In universal algebra, closed classes of functions are known, somewhat opaquely, as *clones*, while the inclusion diagram of all Boolean clones is called *Post's lattice*. Post's lattice is surprisingly complicated, in part because Post did not assume that the constant functions 0 and 1 were available for free.[1]

This paper had its origin in our ambition to find the analogue of Post's lattice for all possible sets of *quantum* gates acting on qubits. We view this as a large, important, and underappreciated goal: something that could be to quantum computing theory almost what the Classification of Finite Simple Groups was to group theory. To provide some context, there are many sets of 1-, 2- and 3-qubit quantum gates that are known to be universal – either in the strong sense that they can be used to approximate any n-qubit unitary transformation to any desired precision, or in the weaker sense that they suffice to perform universal quantum computation (possibly in an encoded subspace). To take two examples, Barenco et al. [6] showed universality for the CNOT gate plus the set of all 1-qubit gates, while Shi [27] showed universality for the Toffoli and Hadamard gates.

There are also sets of quantum gates that are known *not* to be universal: for example, the basis-preserving gates, the 1-qubit gates, and most interestingly, the so-called *stabilizer gates* [12, 3] (that is, the CNOT, Hadamard, and $\pi/4$-Phase gates), as well as the stabilizer gates conjugated by 1-qubit unitary transformations[2]. What is *not* known is whether the preceding list basically exhausts the ways in which quantum gates on qubits can fail to be universal. Are there other elegant discrete structures, analogous to the stabilizer gates, waiting to be discovered? Are there any gate sets, other than conjugated stabilizer gates, that might give rise to intermediate complexity classes, neither contained in P nor equal to BQP?[3] How can we claim to understand quantum circuits – the bread-and-butter of quantum computing textbooks and introductory quantum computing courses – if we do not know the answers to such questions?

---

[1] If one *does* assume constants are free, then Post's lattice dramatically simplifies, with all non-universal gate sets either monotone or affine.

[2] In fact, Grier and Schaeffer [13] have extended our classification to these quantum stabilizer operations under the same model and with a similar proof structure. In the same paper, the authors classify the classical reversible transformations in the quantum regime (i.e., with quantum ancillas), heavily relying on the classification in this paper.

[3] To clarify, there are many restricted models of quantum computing known that are plausibly "intermediate" in that sense, including BosonSampling [1], the one-clean-qubit model [18], and log-depth quantum circuits [9]. However, with the exception of conjugated stabilizer gates, none of those models arises from simply considering which unitary transformations can be generated by some set of k-qubit gates. They all involve non-standard initial states, building blocks other than qubits, or restrictions on how the gates can be composed.

Unfortunately, working out the full "quantum Post's lattice" appears out of reach at present. This might surprise readers, given how much is known about particular quantum gate sets (e.g., those containing CNOT gates), but keep in mind that what is asked for is an accounting of *all* possibilities, no matter how exotic. Indeed, even classifying 1- and 2-qubit quantum gate sets remains wide open (!), and seems, without a new idea, to require studying the irreducible representations of thousands of groups. Recently, Aaronson and Bouland [2] completed a much simpler task, the classification of 2-mode beamsplitters; that was already a complicated undertaking.

Due to its length, this paper has been shortened for these proceedings. Its full version can be found on the arXiv [4].

1.1 Classical Reversible Gates

So one might wonder: can we at least understand all the possible sets of *classical reversible gates* acting on bits, in terms of which reversible transformations they generate? This is an obvious precursor to the quantum case, since every classical reversible gate is also a unitary quantum gate. But beyond that, the classical problem is extremely interesting in its own right, with (as it turns out) a rich algebraic and number-theoretic structure, and with many implications for reversible computing as a whole.

The notion of reversible computing [11, 29, 19, 8, 21, 24] arose from early work on the physics of computation, by such figures as Feynman, Bennett, Benioff, Landauer, Fredkin, Toffoli, and Lloyd. This community was interested in questions like: does universal computation inherently require the generation of entropy (say, in the form of waste heat)? Surprisingly, the theory of reversible computing showed that, in principle, the answer to this question is "no." *Deleting* information unavoidably generates entropy, according to *Landauer's principle* [19], but deleting information is not necessary for universal computation.

Formally, a reversible gate is just a permutation $G : \{0,1\}^k \rightarrow \{0,1\}^k$ of the set of k-bit strings, for some positive integer k. The most famous examples are:

- the 2-bit CNOT (Controlled-NOT) gate, which flips the second bit if and only if the first bit is 1;
- the 3-bit Toffoli gate, which flips the third bit if and only if the first two bits are both 1;
- the 3-bit Fredkin gate, which swaps the second and third bits if and only if the first bit is 1.

These three gates already illustrate some of the concepts that play important roles in this paper. The CNOT gate can be used to copy information in a reversible way, since it maps $x0$ to xx; and also to compute arbitrary affine functions over the finite field $\mathbb{F}_2$. However, because CNOT is *limited* to affine transformations, it is not computationally universal. Indeed, in contrast to the situation with irreversible logic gates, one can show that *no* 2-bit classical reversible gate is computationally universal. The Toffoli gate is computationally universal, because (for example) it maps $x, y, 1$ to $x, y, \overline{xy}$, thereby computing the NAND function. Moreover, Toffoli showed [29] – and we prove for completeness in Section 7.1 – that the Toffoli gate is universal in a stronger sense: it generates all possible reversible transformations $F : \{0,1\}^n \rightarrow \{0,1\}^n$ if one allows the use of ancilla bits, which must be returned to their initial states by the end.

But perhaps the most interesting case is that of the Fredkin gate. Like the Toffoli gate, the Fredkin gate is computationally universal: for example, it maps $x, y, 0$ to $x, \overline{x}y, xy$, thereby computing the AND function. But the Fredkin gate is *not* universal in the stronger sense. The reason is that it is *conservative*: that is, it never changes the total Hamming weight of

the input. Far from being just a technical issue, conservativity was regarded by Fredkin and the other reversible computing pioneers as a sort of discrete analogue of the conservation of energy – and indeed, it plays a central role in certain physical realizations of reversible computing (for example, billiard-ball models, in which the total number of billiard balls must be conserved).

However, all we have seen so far are three specific examples of reversible gates, each leading to a different behavior. To anyone with a mathematical mindset, the question remains: what are all the *possible* behaviors? For example: is Hamming weight the only possible "conserved quantity" in reversible computation? Are there other ways, besides being affine, to fail to be computationally universal? Can one *derive*, from first principles, why the classes of reversible transformations generated by CNOT, Fredkin, etc. are somehow special, rather than just pointing to the sociological fact that these are classes that people in the early 1980s happened to study?

1.2 Ground Rules

In this work, we achieve a complete classification of all possible sets of reversible gates acting on bits, in terms of which reversible transformations $F : \{0,1\}^n \rightarrow \{0,1\}^n$ they generate. Before describing our result, let us carefully explain the ground rules.

First, we assume that swapping bits is free. This simply means that we do not care how the input bits are labeled – or, if we imagine the bits carried by wires, then we can permute the wires in any way we like. The second rule is that an unlimited number of ancilla bits may be used, *provided* the ancilla bits are returned to their initial states by the end of the computation. This second rule might look unfamiliar, but in the context of reversible computing, it is the right choice.

We need to allow ancilla bits because if we do not, then countless transformations are disallowed for trivial reasons. (Restricting a reversible circuit to use *no* ancillas is like restricting a Turing machine to use no memory, besides the n bits that are used to write down the input.) We are forced to say that, although our gates might generate some reversible transformation $F(x,0) = (G(x),0)$, they do not generate the smaller transformation G. The exact value of n then also takes on undeserved importance, as we need to worry about "small-n effects": e.g., that a 3-bit gate cannot be applied to a 2-bit input.

As for the number of ancilla bits: it will *turn out*, because of our classification theorem, that every reversible gate needs only $O(1)$ ancilla bits[4] to generate every n-bit reversible transformation that it can generate at all. However, we do not wish to prejudge this question; if there had been reversible gates that could generate certain transformations, but only by using (say) 2^{2^n} ancilla bits, then that would have been fascinating to know. For the same reason, we do not wish prematurely to restrict the number of ancilla bits that can be 0, or the number that can be 1.

On the other hand, the ancilla bits must be returned to their original states because if they are not, then the computation was not really reversible. One can then learn something about the computation by examining the ancilla bits – if nothing else, then the fact that the computation was done at all. The symmetry between input and output is broken; one cannot then run the computation backwards without setting the ancilla bits differently. This is not just a philosophical problem: if the ancilla bits carry away information about the input

[4] Since it is easy to show that a constant number of ancilla bits are sometimes needed (see Proposition 9), this is the optimal answer, up to the value of the constant (which might depend on the gate set).

x, then *entropy*, or waste heat, has been leaked into the computer's environment. Worse yet, if the reversible computation is a subroutine of a quantum computation, then the leaked entropy will cause *decoherence*, preventing the branches of the quantum superposition with different x values from interfering with each other, as is needed to obtain a quantum speedup. In reversible computing, the technical term for ancilla bits that still depend on x after a computation is complete is *garbage*.[5]

1.3 Our Results

Even after we assume that bit swaps and ancilla bits are free, it remains a significant undertaking to work out the complete list of reversible gate classes, and (especially!) to prove that the list is complete. Doing so is this paper's main technical contribution.

We give a formal statement of the classification theorem in Section 3, and we show the lattice of reversible gate classes in Figure 3. For now, let us simply state the main conclusions informally.

1. **Conserved Quantities.** The following is the complete list of the "global quantities" that reversible gate sets can conserve (if we restrict attention to non-degenerate gate sets, and ignore certain complications caused by linearity and affineness): Hamming weight, Hamming weight mod k for any $k \geq 2$, and inner product mod 2 between pairs of inputs.
2. **Anti-Conservation.** There are gates, such as the NOT gate, that "anti-conserve" the Hamming weight mod 2 (i.e., always change it by a fixed nonzero amount). However, there are no analogues of these for any of the other conserved quantities.
3. **Encoded Universality.** In terms of their "computational power," there are only three kinds of reversible gate sets: degenerate (e.g., NOTs, bit-swaps), non-degenerate but affine (e.g., CNOT), and non-affine (e.g., Toffoli, Fredkin). More interestingly, every non-affine gate set can implement every reversible transformation, and every non-degenerate affine gate set can implement every affine transformation, *if* the input and output bits are encoded by longer strings in a suitable way. For details about "encoded universality," see Section 4.4.
4. **Sporadic Gate Sets.** The conserved quantities interact with linearity and affineness in complicated ways, producing "sporadic" affine gate sets that we have classified. For example, non-degenerate affine gates can preserve Hamming weight mod k, but only if $k = 2$ or $k = 4$. All gates that preserve inner product mod 2 are linear, and all linear gates that preserve Hamming weight mod 4 also preserve inner product mod 2. As a further complication, for an affine transformation $F(x) = Ax + b$, it is possible for A to be orthogonal, mod-2-preserving, or mod-4-preserving without F being orthogonal, mod-2-preserving, or mod-4-preserving, respectively.
5. **Finite Generation.** For each closed class of reversible transformations, there is a single gate that generates the entire class. (*A priori*, it is not even obvious that every class is finitely generated, or that there is "only" a countable infinity of classes!) For more, see Section 4.1.
6. **Symmetry.** Every reversible gate set is symmetric under interchanging the roles of 0 and 1. For more, see Section 4.1.

[5] In Section 2.3, we will discuss a modified rule, which allows a reversible circuit to change the ancilla bits, as long as they change in a way that is independent of the input x. We will show that this "loose ancilla rule" causes only a small change to our classification theorem.

1.4 Algorithmic and Complexity Aspects

Perhaps most relevant to theoretical computer scientists, our classification theorem leads to new algorithms and complexity results about reversible gates and circuits: results that follow easily from the classification, but that we have no idea how to prove otherwise.

Let RevGen (Reversible Generation) be the following problem: we are given as input the truth tables of reversible gates $G_1, \ldots, G_K$, as well as of a target gate H, and wish to decide whether the G_i's generate H. Then we obtain a linear-time algorithm for RevGen. Here, of course, "linear" means linear in the sizes of the truth tables, which is $n2^n$ for an n-bit gate. However, if just a tiny amount of "summary data" about each gate G is provided – namely, the possible values of $|G(x)| - |x|$, where $|\cdot|$ is the Hamming weight, as well as which affine transformation G performs if it is affine – then the algorithm actually runs in $O(n^\omega)$ time, where ω is the matrix multiplication exponent.

We have implemented this algorithm; code is available for download at [25]. For more details see Section 4.2.

Our classification theorem also implies the first general upper bounds (i.e., bounds that hold for all possible gate sets) on the number of gates and ancilla bits needed to implement reversible transformations. In particular, we show (see Section 4.3) that if a set of reversible gates generates an n-bit transformation F at all, then it does so via a circuit with at most $2^n \operatorname{poly}(n)$ gates and $O(1)$ ancilla bits. These bounds are close to optimal.

By contrast, let us consider the situation for these problems without the classification theorem. Suppose, for example, that we want to know whether a reversible transformation $H : \{0,1\}^n \to \{0,1\}^n$ can be synthesized using gates $G_1, \ldots, G_K$. If we knew some upper bound on the number of ancilla bits that might be needed by the generating circuit, then if nothing else, we could of course solve this problem by brute force. The trouble is that, without the classification, it is not obvious how to prove *any* upper bound on the number of ancillas – not even, say, Ackermann(n). This makes it unclear, *a priori*, whether RevGen is even *decidable*, never mind its complexity!

One *can* show on abstract grounds that RevGen is decidable, but with an astronomical running time. To explain this requires a short digression. In universal algebra, there is a body of theory (see e.g. [20]), which grew out of Post's original work [23], about the general problem of classifying closed classes of functions (clones) of various kinds. The upshot is that every clone is characterized by an *invariant* that all functions in the clone preserve: for example, affineness for the NOT and XOR functions, or monotonicity for the AND and OR functions. The clone can then be shown to contain *all* functions that preserve the invariant. (There is a formal definition of "invariant," involving polymorphisms, which makes this statement not a tautology, but we omit it.) Alongside the lattice of clones of functions, there is a dual lattice of *coclones* of invariants, and there is a Galois connection relating the two: as one adds more functions, one preserves fewer invariants, and vice versa.

In response to an inquiry by us, Emil Jeřábek recently showed [15] that the clone/coclone duality can be adapted to the setting of reversible gates. This means that we know, even without a classification theorem, that every closed class of reversible transformations is uniquely determined by the invariants that it preserves.

Unfortunately, this elegant characterization does not give rise to feasible algorithms. The reason is that, for an n-bit gate $G : \{0,1\}^n \to \{0,1\}^n$, the invariants could in principle involve all 2^n inputs, as well arbitrary polymorphisms mapping those inputs into a commutative monoid. Thus the number of polymorphisms one needs to consider grows at least like $2^{2^{2^n}}$. Now, the word problem for commutative monoids is decidable, by reduction to the ideal membership problem (see, e.g., [17, p. 55]). And by putting these facts together, one can

derive an algorithm for RevGen that uses doubly-exponential space and triply-exponential time, as a function of the truth table sizes: in other words, $\exp(\exp(\exp(\exp(n))))$ time, as a function of n. We believe it should also be possible to extract $\exp(\exp(\exp(\exp(n))))$ upper bounds on the number of gates and ancillas from this algorithm, although we have not verified the details.

1.5 Proof Ideas

We hope we have made the case that the classification theorem improves the complexity situation for reversible circuit synthesis! Even so, some people might regard classifying all possible reversible gate sets as a complicated, maybe worthwhile, but fundamentally tedious exercise. Can't such problems be automated via computer search? On the contrary, there are specific aspects of reversible computation that make this classification problem both unusually rich, and unusually hard to reduce to any finite number of cases.

We already discussed the astronomical number of possible invariants that even a tiny reversible gate (say, a 3-bit gate) might satisfy, and the hopelessness of enumerating them by brute force. However, even if we could cut down the number of invariants to something reasonable, there would still be the problem that the size, n, of a reversible gate can be arbitrarily large – and as one considers larger gates, one can discover more and more invariants. Indeed, that is precisely what happens in our case, since the Hamming weight mod k invariant can only be "noticed" by considering gates on k bits or more. There are also "sporadic" affine classes that can only be found by considering 6-bit gates.

Of course, it is not hard just to *guess* a large number of reversible gate classes (affine transformations, parity-preserving and parity-flipping transformations, etc.), prove that these classes are all distinct, and then prove that each one can be generated by a simple set of gates (e.g., CNOT or Fredkin + NOT). Also, once one has a sufficiently powerful gate (say, the CNOT gate), it is often straightforward to classify all the classes *containing* that gate. So for example, it is relatively easy to show that CNOT, together with any non-affine gate, generates all reversible transformations.

As usual with classification problems, the hard part is to rule out exotic additional classes: most of the work, one might say, is not about what is there, but about what isn't there. It is one thing to synthesize some random 1000-bit reversible transformation using only Toffoli gates, but quite another to synthesize a Toffoli gate using only the random 1000-bit transformation!

Thinking about this brings to the fore the central issue: that in reversible computation, it is not enough to output some desired string $F(x)$; one needs to output nothing else *besides* $F(x)$. And hence, for example, it does not suffice to look inside the random 1000-bit reversible gate G, to show that it contains a NAND gate, which is computationally universal. Rather, one needs to deal with *all* of G's outputs, and show that one can eliminate the undesired ones.

The way we do that involves another characteristic property of reversible circuits: that they can have "global conserved quantities," such as Hamming weight. Again and again, we need to prove that if a reversible gate G *fails* to conserve some quantity, such as the Hamming weight mod k, then that fact alone implies that we can use G to implement a desired behavior. This is where elementary algebra and number theory come in.

There are two aspects to the problem. First, we need to understand something about the possible quantities that a reversible gate can conserve. For example, we will need the following three results:

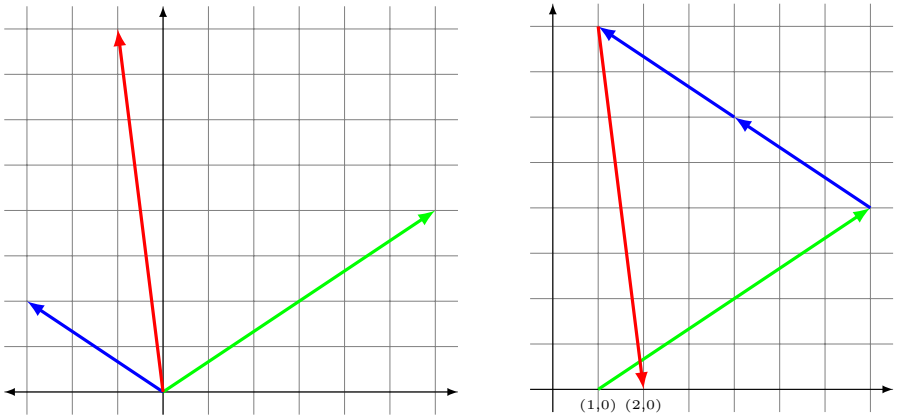

Figure 1 Moving within first quadrant of lattice to construct a COPY gate.

- No reversible gate can change the Hamming weight of some input and also conserve inner products mod k, unless $k = 2$.
- No reversible gate can change Hamming weight mod k by a fixed, nonzero amount, unless $k = 2$.
- No nontrivial linear gate can conserve Hamming weight mod k, unless $k = 2$ or $k = 4$.

We prove each of these statements in Section 6, using arguments based on complex polynomials.

Next, using our knowledge about the possible conserved quantities, we need procedures that take any gate G that fails to conserve some quantity, and that use G to implement a desired behavior (say, making a single copy of a bit, or changing an inner product by exactly 1). We then leverage that behavior to generate a desired gate (say, a Fredkin gate). The two core tasks turn out to be the following:

- Given any non-affine gate, we need to construct a Fredkin gate.
- Given any non-orthogonal linear gate, we need to construct a CNOTNOT gate, a parity-preserving version of CNOT that maps x, y, z to $x, y \oplus x, z \oplus x$.

These proofs are quite lengthy and are included in the full version of our paper [4]. The solution involves 3-dimensional lattices: that is, subsets of $\mathbb{Z}^3$ closed under integer linear combinations. We argue, in essence, that the only possible obstruction to the desired behavior is a "modularity obstruction," but the assumption about the gate G rules out such an obstruction.

We can illustrate this with an example that ends up *not* being needed in the final classification proof, but that we worked out earlier in this research.[6] Let G be any gate that does not conserve (or anti-conserve) the Hamming weight mod k for any $k \geq 2$, and suppose we want to use G to construct a CNOT gate.

[6] In general, after completing the classification proof, we were able to go back and simplify it substantially, by removing results – for example, about the generation of CNOT gates – that were important for working out the lattice in the first place, but which then turned out to be subsumed (or which *could* be subsumed, with modest additional effort) by later parts of the classification. Our current proof reflects these simplifications.

Then we examine how G behaves on restricted inputs: in this case, on inputs that consist entirely of some number of copies of x and $\overline{x}$, where $x \in \{0,1\}$ is a bit, as well as constant 0 and 1 bits. For example, perhaps G can increase the number of copies of x by 5 while decreasing the number of copies of $\overline{x}$ by 7, and can *also* decrease the number of copies of x by 6 without changing the number of copies of $\overline{x}$. Whatever the case, the set of possible behaviors generates some lattice: in this case, a lattice in $\mathbb{Z}^2$ (see Figure 1). We need to argue that the lattice contains a distinguished point encoding the desired "copying" behavior. In the case of the CNOT gate, the point is $(1,0)$, since we want one more copy of x and no more copies of $\overline{x}$. Showing that the lattice contains $(1,0)$, in turn, boils down to arguing that a certain system of Diophantine linear equations must have a solution. One can do this, finally, by using the assumption that G does not conserve or anti-conserve the Hamming weight mod k for any k.

To generate the Fredkin gate, we instead use the Chinese Remainder Theorem to combine gates that change the inner product mod p for various primes p into a gate that changes the inner product between two inputs by exactly 1; while to generate the CNOTNOT gate, we exploit the assumption that our generating gates are linear. In all these cases, it is crucial that we know, from Section 6, that certain quantities *cannot* be conserved by any reversible gate.

There are a few parts of the classification proof that basically *do* come down to enumerating cases, but we hope to have given a sense for the interesting parts.

1.6 Related Work

Surprisingly, the general question of classifying reversible gates such as Toffoli and Fredkin appears never to have been asked, let alone answered, prior to this work.

In the reversible computing literature, there are hundreds of papers on synthesizing reversible circuits (see [24] for a survey), but most of them focus on practical considerations: for example, trying to minimize the number of Toffoli gates or other measures of interest, often using software optimization tools. We found only a tiny amount of work relevant to the classification problem: notably, an unpublished preprint by Lloyd [21], which shows that every non-affine reversible gate is computationally universal, if one does not care what garbage is generated in addition to the desired output. Lloyd's result was subsequently rediscovered by Kerntopf et al. [16] and De Vos and Storme [30].

There is also work by Morita et al. [22] that uses brute-force enumeration to classify certain reversible computing elements with 2, 3, or 4 wires, but the notion of "reversible gate" there is very different from the standard one (the gates are for routing a single "billiard ball" element rather than for transforming bit strings, and they have internal state). Finally, there is work by Strazdins [28], not motivated by reversible computing, which considers classifying reversible Boolean functions, but which imposes a separate requirement on each output bit that it belong to one of the classes from Post's original lattice, and which thereby misses all the reversible gates that conserve "global" quantities, such as the Fredkin gate.[7]

[7] Because of different rules regarding constants, developed with Post's lattice rather than reversible computing in mind, Strazdins also includes classes that we do not (e.g., functions that always map 0^n or 1^n to themselves, but are otherwise arbitrary). To use our notation, his 13-class lattice ends up intersecting our infinite lattice in just five classes: $\langle\varnothing\rangle$, $\langle\text{NOT}\rangle$, $\langle\text{CNOTNOT}, \text{NOT}\rangle$, $\langle\text{CNOT}\rangle$, and $\langle\text{Toffoli}\rangle$.

2 Notation and Definitions

$\mathbb{F}_2$ means the field of 2 elements. $[n]$ means $\{1,\ldots,n\}$. We denote by $e_1,\ldots,e_n$ the standard basis for the vector space $\mathbb{F}_2^n$: that is, $e_1=(1,0,\ldots,0)$, etc.

Let $x=x_1\ldots x_n$ be an n-bit string. Then $\overline{x}$ means x with all n of its bits inverted. Also, $x\oplus y$ means bitwise XOR, x,y or xy means concatenation, x^k means the concatenation of k copies of x, and $|x|$ means the Hamming weight. The *parity* of x is $|x| \bmod 2$. The *inner product* of x and y is the integer $x\cdot y=x_1y_1+\cdots+x_ny_n$. Note that

$$x\cdot(y\oplus z)\equiv x\cdot y+x\cdot z\,(\mathrm{mod}\,2)\,,$$

but the above need not hold if we are not working mod 2.

By $\mathrm{gar}\,(x)$, we mean garbage depending on x: that is, "scratch work" that a reversible computation generates along the way to computing some desired function $f\,(x)$. Typically, the garbage later needs to be *uncomputed*. Uncomputing, a term introduced by Bennett [8], simply means running an entire computation in reverse, after the output $f\,(x)$ has been safely stored.

2.1 Gates

By a *(reversible) gate*, throughout this paper we will mean a reversible transformation G on the set of k-bit strings: that is, a permutation of $\{0,1\}^k$, for some fixed k. Formally, the terms 'gate' and 'reversible transformation' will mean the same thing; 'gate' just connotes a reversible transformation that is particularly small or simple.

A gate is *nontrivial* if it does something other than permute its input bits, and *non-degenerate* if it does something other than permute its input bits and/or apply NOT's to some subset of them.

A gate G is *conservative* if it satisfies $|G\,(x)|=|x|$ for all x. A gate is *mod-k-respecting* if there exists a j such that

$$|G\,(x)|\equiv|x|+j\,(\mathrm{mod}\,k)$$

for all x. It's *mod-k-preserving* if moreover $j=0$. It's *mod-preserving* if it's mod-k-preserving for some $k\geq 2$, and *mod-respecting* if it's mod-k-respecting for some $k\geq 2$.

As special cases, mod-2-respecting gates and mod-2-preserving gates are called is also called *parity-respecting* and *parity-preserving* respectively. A gate G such that

$$|G\,(x)|\not\equiv|x|\,(\mathrm{mod}\,2)$$

for all x is called *parity-flipping*. In Theorem 12, we will prove that parity-flipping gates are the *only* examples of mod-respecting gates that are not mod-preserving.

The *respecting number* of a gate G, denoted $k\,(G)$, is the largest k such that G is mod-k-respecting. (By convention, if G is conservative then $k\,(G)=\infty$, while if G is non-mod-respecting then $k\,(G)=1$.) We have the following fact:

▶ **Proposition 1.** *G is mod-ℓ-respecting if and only if ℓ divides $k\,(G)$.*

Proof. If ℓ divides $k\,(G)$, then certainly G is mod-ℓ-respecting. Now, suppose G is mod-ℓ-respecting but ℓ does not divide $k\,(G)$. Then G is both mod-ℓ-respecting and mod-$k(G)$-respecting. So by the Chinese Remainder Theorem, G is mod-$\mathrm{lcm}\,(\ell,k\,(G))$-respecting. But this contradicts the definition of $k\,(G)$. ◀

A gate G is *affine* if it implements an affine transformation over $\mathbb{F}_2$: that is, if there exists an invertible matrix $A \in \mathbb{F}_2^{k \times k}$, and a vector $b \in \mathbb{F}_2^k$, such that $G(x) = Ax \oplus b$ for all x. A gate is *linear* if moreover $b = 0$. A gate is *orthogonal* if it satisfies

$$G(x) \cdot G(y) \equiv x \cdot y \pmod 2$$

for all x, y. (We will observe, in Lemma 14, that every orthogonal gate is linear.) Also, if $G(x) = Ax \oplus b$ is affine, then the *linear part of* G is the linear transformation $G'(x) = Ax$. We call G orthogonal in its linear part, mod-k-preserving in its linear part, etc. if G' satisfies the corresponding invariant. A gate that is orthogonal in its linear part is also called an *isometry*.

Given two gates G and H, their *tensor product*, $G \otimes H$, is a gate that applies G and H to disjoint sets of bits. We will often use the tensor product to produce a single gate that combines the properties of two previous gates. Also, we denote by $G^{\otimes t}$ the tensor product of t copies of G.

2.2 Gate Classes

Let $S = \{G_1, G_2, \ldots\}$ be a set of gates, possibly on different numbers of bits and possibly infinite. Then $\langle S \rangle = \langle G_1, G_2, \ldots \rangle$, *the class of reversible transformations generated by* S, can be defined as the smallest set of reversible transformations $F : \{0,1\}^n \to \{0,1\}^n$ that satisfies the following closure properties:

1. **Base case.** $\langle S \rangle$ contains S, as well as the identity function $F(x_1 \ldots x_n) = x_1 \ldots x_n$ for all $n \geq 1$.
2. **Composition rule.** If $\langle S \rangle$ contains $F(x_1 \ldots x_n)$ and $G(x_1 \ldots x_n)$, then $\langle S \rangle$ also contains $F(G(x_1 \ldots x_n))$.
3. **Swapping rule.** If $\langle S \rangle$ contains $F(x_1 \ldots x_n)$, then $\langle S \rangle$ also contains all possible functions $\sigma\left(F\left(x_{\tau(1)} \ldots x_{\tau(n)}\right)\right)$ obtained by permuting F's input and output bits.
4. **Extension rule.** If $\langle S \rangle$ contains $F(x_1 \ldots x_n)$, then $\langle S \rangle$ also contains the function

   $$G(x_1 \ldots x_n, b) := (F(x_1 \ldots x_n), b),$$

   in which b occurs as a "dummy" bit.
5. **Ancilla rule.** If $\langle S \rangle$ contains a function F that satisfies

   $$F(x_1 \ldots x_n, a_1 \ldots a_k) = (G(x_1 \ldots x_n), a_1 \ldots a_k) \quad \forall x_1 \ldots x_n \in \{0,1\}^n,$$

   for some smaller function G and fixed "ancilla" string $a_1 \ldots a_k \in \{0,1\}^k$ that does not depend on x, then $\langle S \rangle$ also contains G. (Note that, if the a_i's are set to other values, then F need not have the above form.)

Note that because of reversibility, the set of n-bit transformations in $\langle S \rangle$ (for any n) always forms a group. Indeed, if $\langle S \rangle$ contains F, then clearly $\langle S \rangle$ contains all the iterates $F^2(x) = F(F(x))$, etc. But since there must be some positive integer m such that $F^m(x) = x$, this means that $F^{m-1}(x) = F^{-1}(x)$. Thus, we do not need a separate rule stating that $\langle S \rangle$ is closed under inverses.

We say S *generates* the reversible transformation F if $F \in \langle S \rangle$. We also say that S generates $\langle S \rangle$.

Given an arbitrary set $\mathcal{C}$ of reversible transformations, we call $\mathcal{C}$ a *reversible gate class* (or *class* for short) if $\mathcal{C}$ is closed under rules (1)-(5) above: in other words, if there exists an S such that $\mathcal{C} = \langle S \rangle$.

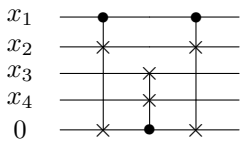

Figure 2 Generating a Controlled-Controlled-Swap gate from Fredkin.

A *reversible circuit* for the function F, over the gate set S, is an explicit procedure for generating F by applying gates in S, and thereby showing that $F \in \langle S \rangle$. An example is shown in Figure 2. Reversible circuit diagrams are read from left to right, with each bit that occurs in the circuit (both input and ancilla bits) represented by a horizontal line, and each gate represented by a vertical line.

If every gate $G \in S$ satisfies some invariant, then we can also describe S and $\langle S \rangle$ as satisfying that invariant. So for example, the set $\{\text{CNOTNOT}, \text{NOT}\}$ is affine and parity-respecting, and so is the class that it generates. Conversely, S violates an invariant if any $G \in S$ violates it.

Just as we defined the respecting number $k(G)$ of a gate, we would like to define the respecting number $k(S)$ of an entire gate set. To do so, we need a proposition about the behavior of $k(G)$ under tensor products.

▶ **Proposition 2.** *For all gates G and H,*

$$k(G \otimes H) = \gcd(k(G), k(H)).$$

Proof. Letting $\gamma = \gcd(k(G), k(H))$, clearly $G \otimes H$ is mod-γ-respecting. To see that $G \otimes H$ is not mod-ℓ-respecting for any $\ell > \gamma$: by definition, ℓ must fail to divide either $k(G)$ or $k(H)$. Suppose it fails to divide $k(G)$ without loss of generality. Then G cannot be mod-ℓ-respecting, by Proposition 1. But if we consider pairs of inputs to $G \otimes H$ that differ only on G's input, then this implies that $G \otimes H$ is not mod-ℓ-respecting either. ◀

If $S = \{G_1, G_2, \ldots\}$, then because of Proposition 2, we can define k on the set S as $\gcd(k(G_1), k(G_2), \ldots)$. For then not only will every transformation in $\langle S \rangle$ be mod-$k(S)$-respecting, but there will exist transformations in $\langle S \rangle$ that are not mod-ℓ-respecting for any $\ell > k(S)$.

We then have that S is mod-k-respecting if and only if k divides $k(S)$, and mod-respecting if and only if S is mod-k-respecting for some $k \geq 2$.

2.3 Alternative Kinds of Generation

We now discuss four alternative notions of what it can mean for a reversible gate set to "generate" a transformation. Besides being interesting in their own right, some of these notions will also be used in the proof of our main classification theorem.

Partial Gates. A *partial reversible gate* is an injective function $H : D \to \{0,1\}^n$, where D is some subset of $\{0,1\}^n$. Such an H is *consistent* with a full reversible gate G if $G(x) = H(x)$ whenever $x \in D$. Also, we say that a reversible gate set S *generates* H if S generates any G with which H is consistent. As an example, COPY is the 2-bit partial reversible gate defined by the following relations:

$$\text{COPY}(00) = 00, \qquad \text{COPY}(10) = 11.$$

If a gate set S can implement the above behavior, using ancilla bits that are returned to their original states by the end, then we say S "generates COPY"; the behavior on inputs 01

and 11 is irrelevant. Note that COPY is consistent with CNOT. One can think of COPY as a bargain-basement CNOT, but one that might be bootstrapped up to a full CNOT with further effort.

Generation With Garbage. Let $D \subseteq \{0,1\}^m$, and $H : D \to \{0,1\}^n$ be some function, which need not be injective or surjective, or even have the same number of input and output bits. Then we say that a reversible gate set S *generates* H *with garbage* if there exists a reversible transformation $G \in \langle S \rangle$, as well as an ancilla string a and a function gar, such that $G(x,a) = (H(x), \text{gar}(x))$ for all $x \in D$. As an example, consider the ordinary 2-bit AND function, from $\{0,1\}^2$ to $\{0,1\}$. Since AND destroys information, clearly no reversible gate can generate it in the usual sense, but many reversible gates can generate AND with garbage: for instance, the Toffoli and Fredkin gates, as we saw in Section 1.1.

Encoded Universality. This is a concept borrowed from quantum computing [5]. In our setting, encoded universality means that there is some way of encoding 0's and 1's by longer strings, such that our gate set can implement any desired transformation on the encoded bits. Note that, while this is a weaker notion of universality than the ability to generate arbitrary permutations of $\{0,1\}^n$, it is stronger than "merely" computational universality, because it still requires a transformation to be performed reversibly, with no garbage left around. Formally, given a reversible gate set S, we say that S *supports encoded universality* if there are k-bit strings $\alpha(0)$ and $\alpha(1)$ such that for every n-bit reversible transformation $F(x_1 \ldots x_n) = y_1 \ldots y_n$, there exists a transformation $G \in \langle S \rangle$ that satisfies

$$G(\alpha(x_1) \ldots \alpha(x_n)) = \alpha(y_1) \ldots \alpha(y_n)$$

for all $x \in \{0,1\}^n$. Also, we say that S *supports affine encoded universality* if this is true for every affine F.

As a well-known example, the Fredkin gate is not universal in the usual sense, because it preserves Hamming weight. But it is easy to see that Fredkin supports encoded universality, using the so-called *dual-rail encoding*, in which every 0 bit is encoded as 01, and every 1 bit is encoded as 10. In Section 4.4, we will show, as a consequence of our classification theorem, that *every* reversible gate set (except for degenerate sets) supports either encoded universality or affine encoded universality.

Loose Generation. Finally, we say that a gate set S *loosely generates* a reversible transformation $F : \{0,1\}^n \to \{0,1\}^n$, if there exists a transformation $G \in \langle S \rangle$, as well as ancilla strings a and b, such that

$$G(x,a) = (F(x), b)$$

for all $x \in \{0,1\}^n$. In other words, G is allowed to change the ancilla bits, so long as they change in a way that is independent of the input x. Under this rule, one could perhaps tell by examining the ancilla bits *that* G was applied, but one could not tell to which input. This suffices for some applications of reversible computing, though not for others.[8]

3 Stating the Classification Theorem

In this section we state our main result, and make a few preliminary remarks about it. First let us define the gates that appear in the classification theorem.

[8] For example, if G were applied to a quantum superposition, then it would still maintain coherence among all the inputs to which it was applied – though perhaps not between those inputs and other inputs in the superposition to which it was *not* applied.

- NOT is the 1-bit gate that maps x to $\overline{x}$.
- NOTNOT, or $\mathrm{NOT}^{\otimes 2}$, is the 2-bit gate that maps x, y to $\overline{x}, \overline{y}$. NOTNOT is a parity-preserving variant of NOT.
- CNOT (Controlled-NOT) is the 2-bit gate that maps x, y to $x, y \oplus x$. CNOT is affine.
- CNOTNOT is the 3-bit gate that maps x, y, z to $x, y \oplus x, z \oplus x$. CNOTNOT is affine and parity-preserving.
- Toffoli (also called Controlled-Controlled-NOT, or CCNOT) is the 3-bit gate that maps x, y, z to $x, y, z \oplus xy$.
- Fredkin (also called Controlled-SWAP, or CSWAP) is the 3-bit gate that maps x, y, z to $x, y \oplus x\,(y \oplus z)\,, z \oplus x\,(y \oplus z)$. In other words, it swaps y with z if $x = 1$, and does nothing if $x = 0$. Fredkin is conservative: it never changes the Hamming weight.
- C_k is a k-bit gate that maps 0^k to 1^k and 1^k to 0^k, and all other k-bit strings to themselves. C_k preserves the Hamming weight mod k. Note that $\mathrm{C}_1 = \mathrm{NOT}$, while C_2 is equivalent to NOTNOT, up to a bit-swap.
- T_k is a k-bit gate (for even k) that maps $x = (x_1, x_2, \ldots, x_k)$ to $\overline{x} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_k)$ if $|x|$ is odd, or to x if $|x|$ is even. A different definition is

  $$\mathrm{T}_k\,(x_1, \ldots, x_k) = (x_1 \oplus b_x, \ldots, x_k \oplus b_x)\,,$$

  where $b_x := x_1 \oplus \cdots \oplus x_k$. This shows that T_k is linear. Indeed, we also have

  $$\mathrm{T}_k\,(x) \cdot \mathrm{T}_k\,(y) \equiv x \cdot y \,(\mathrm{mod}\, 2)\,,$$

  which shows that T_k is orthogonal. Note also that, if $k \equiv 2 \,(\mathrm{mod}\, 4)$, then T_k preserves Hamming weight mod 4: if $|x|$ is even then $|\mathrm{T}_k\,(x)| = |x|$, while if $|x|$ is odd then

  $$|\mathrm{T}_k\,(x)| \equiv k - |x| \equiv 2 - |x| \equiv |x| \,(\mathrm{mod}\, 4)\,.$$

- F_k is a k-bit gate (for even k) that maps x to $\overline{x}$ if $|x|$ is even, or to x if $|x|$ is odd. A different definition is

  $$\mathrm{F}_k\,(x_1 \ldots x_k) = \overline{\mathrm{T}_k\,(x_1 \ldots x_k)} = (x_1 \oplus b_x \oplus 1, \ldots, x_k \oplus b_x \oplus 1)$$

  where b_x is as above. This shows that F_k is affine. Indeed, if k is a multiple of 4, then F_k preserves Hamming weight mod 4: if $|x|$ is odd then $|\mathrm{F}_k\,(x)| = |x|$, while if $|x|$ is even then

  $$|\mathrm{F}_k\,(x)| \equiv k - |x| \equiv |x| \,(\mathrm{mod}\, 4)\,.$$

  Since F_k is equal to T_k in its linear part, F_k is also an isometry.

We can now state the classification theorem.

▶ **Theorem 3** (Main Result). *Every set of reversible gates generates one of the following classes:*

1. *The trivial class (which contains only bit-swaps).*
2. *The class of all transformations (generated by* Toffoli*).*
3. *The class of all conservative transformations (generated by* Fredkin*).*
4. *For each $k \geq 3$, the class of all mod-k-preserving transformations (generated by C_k).*
5. *The class of all affine transformations (generated by* CNOT*).*
6. *The class of all parity-preserving affine transformations (generated by* CNOTNOT*).*
7. *The class of all mod-4-preserving affine transformations (generated by F_4).*

8. *The class of all orthogonal linear transformations (generated by* T_4*).*
9. *The class of all mod-*4*-preserving orthogonal linear transformations (generated by* T_6*).*
10. *Classes 1, 3, 7, 8, or 9 augmented by a* NOTNOT *gate (note: 7 and 8 become equivalent this way).*
11. *Classes 1, 3, 6, 7, 8, or 9 augmented by a* NOT *gate (note: 7 and 8 become equivalent this way).*

Furthermore, all the above classes are distinct except when noted otherwise, and they fit together in the lattice diagram shown in Figure 3.[9]

Let us make some comments about the structure of the lattice. The lattice has a countably infinite number of classes, with the one infinite part given by the mod-k-preserving classes. The mod-k-preserving classes are partially ordered by divisibility, which means, for example, that the lattice is not planar.[10] While there are infinite descending chains in the lattice, there is no infinite ascending chain. This means that, if we start from some reversible gate class and then add new gates that extend its power, we must terminate after finitely many steps with the class of all reversible transformations.

In the full version [4], we prove that if we allow loose generation, then the only change to Theorem 3 is that every class $\mathcal{C}$ containing a NOTNOT gate collapses with $\mathcal{C} + \mathrm{NOT}$.

4 Consequences of the Classification

To illustrate the power of the classification theorem, in this section we use it to prove four general implications for reversible computation. While these implications are easy to prove with the classification in hand, we do not know how to prove any of them without it.

4.1 Nature of the Classes

Here is one immediate (though already non-obvious) corollary of Theorem 3.

▶ **Corollary 4.** *Every reversible gate class* $\mathcal{C}$ *is finitely generated: that is, there exists a finite set* S *such that* $\mathcal{C} = \langle S \rangle$*.*

Indeed, we have something stronger.

▶ **Corollary 5.** *Every reversible gate class* $\mathcal{C}$ *is generated by a single gate* $G \in \mathcal{C}$*.*

Proof. This is immediate for all the classes listed in Theorem 3, except the ones involving NOT or NOTNOT gates. For classes of the form $\mathcal{C} = \langle G, \mathrm{NOT} \rangle$ or $\mathcal{C} = \langle G, \mathrm{NOTNOT} \rangle$, we just need a single gate G' that is clearly generated by $\mathcal{C}$, and clearly *not* generated by a smaller class. We can then appeal to Theorem 3 to assert that G' *must* generate $\mathcal{C}$. For each of the relevant G's – namely, Fredkin, CNOTNOT, F_4, and T_6 – one such G' is the tensor product, $G \otimes \mathrm{NOT}$ or $G \otimes \mathrm{NOTNOT}$. ◀

[9] Let us mention that Fredkin + NOTNOT generates the class of all parity-preserving transformations, while Fredkin + NOT generates the class of all parity-respecting transformations. We could have listed the parity-preserving transformations as a special case of the mod-k-preserving transformations: namely, the case $k = 2$. If we had done so, though, we would have had to include the caveat that C_k only generates all mod-k-preserving transformations when $k \geq 3$ (when $k = 2$, we also need Fredkin in the generating set). And in any case, the parity-respecting class would still need to be listed separately.

[10] For consider the graph with the integers 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 15, 18, 20, 21, 24, and 28 as its vertices, and with an edge between each pair whose ratio is a prime. One can check that this graph contains $K_{3,3}$ as a minor.

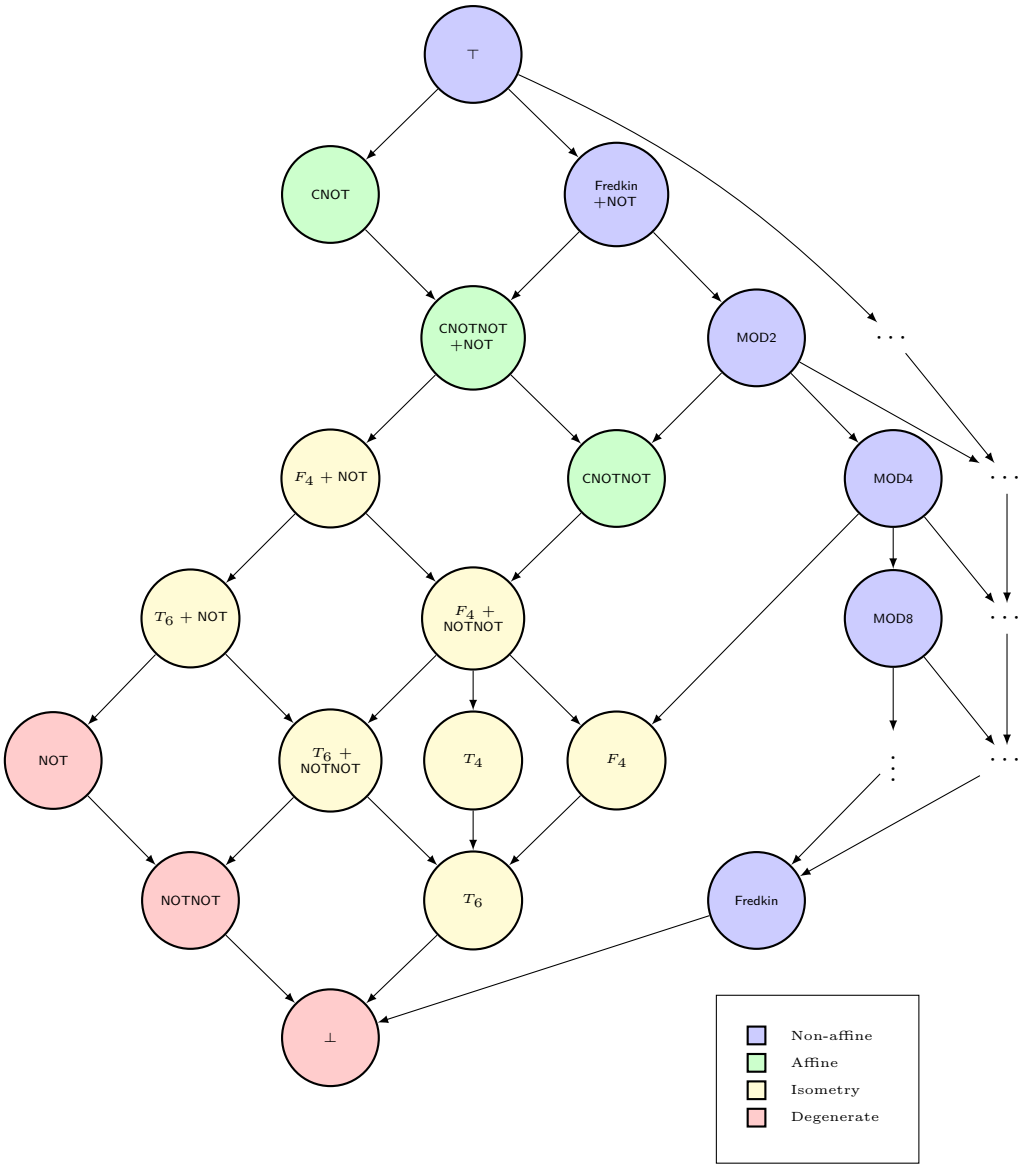

Figure 3 The inclusion lattice of reversible gate classes.

We also wish to point out a non-obvious symmetry property that follows from the classification theorem. Given an n-bit reversible transformation F, let F^*, or the *dual* of F, be $F^*(x_1 \dots x_n) := \overline{F(\overline{x}_1 \dots \overline{x}_n)}$. The dual can be thought of as F with the roles of 0 and 1 interchanged: for example, Toffoli$^*(xyz)$ flips z if and only if $x = y = 0$. Also, call a gate F *self-dual* if $F^* = F$, and call a reversible gate class $\mathcal{C}$ *dual-closed* if $F^* \in \mathcal{C}$ whenever $F \in \mathcal{C}$. Then:

▶ **Corollary 6.** *Every reversible gate class $\mathcal{C}$ is dual-closed.*

Proof. This is obvious for all the classes listed in Theorem 3 that include a NOT or NOTNOT gate. For the others, we simply need to consider the classes one by one: the notions of "conservative," "mod-k-respecting," and "mod-k-preserving" are manifestly the same after we interchange 0 and 1. This is less manifest for the notion of "orthogonal," but one can check that T_k and F_k are self-dual for all even k. ◀

4.2 Linear-Time Algorithm

If one wanted, one could interpret this entire paper as addressing a straightforward *algorithms* problem: namely, the RevGen problem defined in Section 1.4, where we are given as input a set of reversible gates $G_1, \ldots, G_K$, as well as a target reversible transformation H, and we want to know whether the G_i's generate H. From that perspective, our contribution is to reduce the known upper bound on the complexity of RevGen: from recursively-enumerable (!), or triply-exponential time if we use Jeřábek's recent clone/coclone duality for reversible gates [15], all the way down to linear time.

▶ **Theorem 7.** *There is a linear-time algorithm for* RevGen.

Proof. It suffices to give a linear-time algorithm that takes as input the truth table of a single reversible transformation $G : \{0,1\}^n \to \{0,1\}^n$, and that decides which class it generates. For we can then compute $\langle G_1, \ldots, G_K \rangle$ by taking the least upper bound of $\langle G_1 \rangle, \ldots, \langle G_K \rangle$, and can also solve the membership problem by checking whether

$$\langle G_1, \ldots, G_K \rangle = \langle G_1, \ldots, G_K, H \rangle .$$

The algorithm is as follows: first, make a single pass through G's truth table, in order to answer the following two questions.

- Is G affine, and if so, what is its matrix representation, $G(x) = Ax \oplus b$?
- What is $W(G) := \{|G(x)| - |x| : x \in \{0,1\}^n\}$?

In any reasonable RAM model, both questions can easily be answered in $O(n2^n)$ time, which is the number of bits in G's truth table.

If G is non-affine, then Theorem 3 implies that we can determine $\langle G \rangle$ from $W(G)$ alone. If G is affine, then Theorem 3 implies we can determine $\langle G \rangle$ from (A, b) alone, though it is also convenient to use $W(G)$. We need to take the gcd of the numbers in $W(G)$, check whether A is orthogonal, etc., but the time needed for these operations is only $\text{poly}(n)$, which is negligible compared to the input size of $n2^n$. ◀

We have implemented the algorithm described in Theorem 7, and Java code is available for download [25].

4.3 Compression of Reversible Circuits

We now state a "complexity-theoretic" consequence of Theorem 3.

▶ **Theorem 8.** *Let R be a reversible circuit, over any gate set S, that maps $\{0,1\}^n$ to $\{0,1\}^n$, using an unlimited number of gates and ancilla bits. Then there is another reversible circuit, over the same gate set S, that applies the same transformation as R does, and that uses only $2^n \, \text{poly}(n)$ gates and $O(1)$ ancilla bits.*[11]

Proof. If S is one of the gate sets listed in Theorem 3, then this follows immediately by examining the reversible circuit constructions in Section 7, for each class in the classification. Building, in relevant parts, on results by others [26, 7], we will take care in Section 7 to ensure that each non-affine circuit construction uses at most $2^n \, \text{poly}(n)$ gates and $O(1)$ ancilla bits, while each affine construction uses at most $O(n^2)$ gates and $O(1)$ ancilla bits (most actually use no ancilla bits).

[11] Here the big-O's suppress constant factors that depend on the gate set in question.

Now suppose S is *not* one of the sets listed in Theorem 3, but some other set that generates one of the listed classes. So for example, suppose $\langle S \rangle = \langle \text{Fredkin}, \text{NOT} \rangle$. Even then, we know that S *generates* Fredkin and NOT, and the number of gates and ancillas needed to do so is just some constant, independent of n. Furthermore, each time we need a Fredkin or NOT, we can reuse the same ancilla bits, by the assumption that those bits are returned to their original states. So we can simply simulate the appropriate circuit construction from Section 7, using only a constant factor more gates and $O(1)$ more ancilla bits than the original construction. ◀

As we said in Section 1.4, without the classification theorem, it is not obvious how to prove *any upper bound whatsoever* on the number of gates or ancillas, for arbitrary gate sets S. Of course, any circuit that uses T gates also uses at most $O(T)$ ancillas; and conversely, any circuit that uses M ancillas needs at most $\left(2^{n+M}\right)!$ gates, for counting reasons. But the best upper bounds on either quantity that follow from clone theory and the ideal membership problem appear to have the form $\exp(\exp(\exp(\exp(n))))$.

A constant number of ancilla bits *is* sometimes needed, and not only for the trivial reasons that our gates might act on more than n bits, or only (e.g.) be able to map 0^n to 0^n if no ancillas are available.

▶ **Proposition 9** (Toffoli [29]). *If no ancillas are allowed, then there exist reversible transformations of* $\{0,1\}^n$ *that cannot be generated by any sequence of reversible gates on* $n-1$ *bits or fewer.*

Proof. For all $k \geq 1$, any $(n-k)$-bit gate induces an even permutation of $\{0,1\}^n$ – since each cycle is repeated 2^k times, once for every setting of the k bits on which the gate doesn't act. But there are also odd permutations of $\{0,1\}^n$. ◀

It is also easy to show, using a Shannon counting argument, that there exist n-bit reversible transformations that require $\Omega(2^n)$ gates to implement, and n-bit affine transformations that require $\Omega\left(n^2/\log n\right)$ gates. Thus the bounds in Theorem 8 on the number of gates T are, for each class, off from the optimal bounds only by $\text{polylog}\, T$ factors.

4.4 Encoded Universality

If we only care about which Boolean functions $f : \{0,1\}^n \to \{0,1\}$ can be computed, and are completely uninterested in what garbage is output along with f, then it is not hard to see that all reversible gate sets fall into three classes. Namely, non-affine gate sets (such as Toffoli and Fredkin) can compute all Boolean functions;[12] non-degenerate affine gate sets (such as CNOT and CNOTNOT) can compute all affine functions; and degenerate gate sets (such as NOT and NOTNOT) can compute only 1-bit functions. However, the classification theorem lets us make a more interesting statement. Recall the notion of *encoded universality* from Section 2.3, which demands that every reversible transformation (or every affine transformation) be implementable without garbage, once 0 and 1 are "encoded" by longer strings $\alpha(0)$ and $\alpha(1)$ respectively.

▶ **Theorem 10.** *Besides the trivial,* NOT*, and* NOTNOT *classes, every reversible gate class supports encoded universality if non-affine, or affine encoded universality if affine.*

[12] This was proven by Lloyd [21], as well as by Kerntopf et al. [16] and De Vos and Storme [30]; we include a proof for completeness in the full version of this paper [4].

Proof. For $\langle \text{Fredkin} \rangle$, and for all the non-affine classes above $\langle \text{Fredkin} \rangle$, we use the so-called "dual-rail encoding," where 0 is encoded by 01 and 1 is encoded by 10. Given three encoded bits, $x\overline{x}y\overline{y}z\overline{z}$, we can simulate a Fredkin gate by applying one Fredkin to xyz and another to $x\overline{y}\overline{z}$, and can also simulate a CNOT by applying a Fredkin to $xy\overline{y}$. But Fredkin + CNOT generates everything.

The dual-rail encoding also works for simulating all affine transformations using an F_4 gate. For note that

$$\begin{aligned} \mathrm{F}_4\left(xy\overline{y}1\right) &= \left(1, \overline{x \oplus y}, x \oplus y, x\right) \\ &= \left(x, x \oplus y, \overline{x \oplus y}, 1\right), \end{aligned}$$

where we used that we can permute bits for free. So given two encoded bits, $x\overline{x}y\overline{y}$, we can simulate a CNOT from x to y by applying F_4 to x, y, $\overline{y}$, and one ancilla bit initialized to 1.

For $\langle \text{CNOTNOT} \rangle$, we use a repetition encoding, where 0 is encoded by 00 and 1 is encoded by 11. Given two encoded bits, $xxyy$, we can simulate a CNOT from x to y by applying a CNOTNOT from either copy of x to both copies of y. This lets us perform all affine transformations on the encoded subspace.

The repetition encoding also works for $\langle \mathrm{T}_4 \rangle$. For notice that

$$\begin{aligned} \mathrm{T}_4\left(xyy0\right) &= \left(0, x \oplus y, x \oplus y, x\right) \\ &= \left(x, x \oplus y, x \oplus y, 0\right). \end{aligned}$$

Thus, to simulate a CNOT from x to y, we use one copy of x, both copies of y, and one ancilla bit initialized to 0.

Finally, for $\langle \mathrm{T}_6 \rangle$, we encode 0 by 0011 and 1 by 1100. Notice that

$$\begin{aligned} \mathrm{T}_6\left(xyy\overline{y}\overline{y}0\right) &= \left(0, x \oplus y, x \oplus y, \overline{x \oplus y}, \overline{x \oplus y}, x\right) \\ &= \left(x, x \oplus y, x \oplus y, \overline{x \oplus y}, \overline{x \oplus y}, 0\right). \end{aligned}$$

So given two encoded bits, $xx\overline{x}\overline{x}yy\overline{y}\overline{y}$, we can simulate a CNOT from x to y by using one copy of x, all four copies of y and $\overline{y}$, and one ancilla bit initialized to 0. ◀

In the proof of Theorem 10, notice that, every time we simulated Fredkin (xyz) or CNOT (xy), we had to examine only a single bit in the encoding of the control bit x. Thus, Theorem 10 actually yields a stronger consequence: that given an ordinary, unencoded input string $x_1 \ldots x_n$, we can use any non-degenerate reversible gate first to *translate* x into its encoded version $\alpha\left(x_1\right) \ldots \alpha\left(x_n\right)$, and then to perform arbitrary transformations or affine transformations on the encoding.

5 Structure of the Proof

The proof of Theorem 3 naturally divides into four components. First, we need to verify that all the gates mentioned in the theorem really do satisfy the invariants that they are claimed to satisfy – and as a consequence, that any reversible transformation they generate also satisfies the invariants. This is completely routine.

Second, we need to verify that all pairs of classes that Theorem 3 says are distinct, *are* distinct. We handle this in Theorem 11 below (there are only a few non-obvious cases).

Third, we need to verify that the "gate definition" of each class coincides with its "invariant definition"– i.e., that each gate really does generate all reversible transformations that satisfy its associated invariant. For example, we need to show that Fredkin generates all conservative

transformations, that C_k generates all transformations that preserve Hamming weight mod k, and that T_4 generates all orthogonal linear transformations. Many of these results are already known, but for completeness, we prove all of them in Section 7, by giving explicit constructions of reversible circuits.[13]

Finally, we need to show that there are no *additional* reversible gate classes, besides the ones listed in Theorem 3. This is by far the most interesting part, but it appears only in the full version of the paper [4] due to its length. Nevertheless, the organization of the complete proof is as follows:

- In Section 6, we collect numerous results about what reversible transformations can and cannot do to Hamming weights mod k and inner products mod k, in both the affine and the non-affine cases; these results are then drawn on in the rest of the paper. (Some of them are even used for the circuit constructions in Section 7.)
- In the full version, we complete the classification of all non-affine gate sets. We show that the only classes that contain a Fredkin gate (equivalently, the classes above $\langle \text{Fredkin} \rangle$ in the lattice) are $\langle \text{Fredkin} \rangle$ itself, $\langle \text{Fredkin}, \text{NOTNOT} \rangle$, $\langle \text{Fredkin}, \text{NOT} \rangle$, $\langle C_k \rangle$ for $k \geq 3$, and $\langle \text{Toffoli} \rangle$. Next, we show that every nontrivial conservative gate generates Fredkin. We then build on that result to show that every non-affine gate set generates Fredkin.
- The complete classification of all affine gate sets is also contained in the full version. For simplicity, we start with *linear* gate sets only, and show that every nontrivial mod-4-preserving linear gate generates T_6, and that every nontrivial, *non*-mod-4-preserving orthogonal gate generates T_4. Next, we show that every non-orthogonal linear gate generates CNOTNOT. Then, we show that every non-parity-preserving linear gate generates CNOT. Since CNOT generates all linear transformations, this completes the classification of linear gate sets. Finally, we "put back the affine part," showing that it can lead to only 8 additional classes besides the linear classes $\langle \varnothing \rangle$, $\langle T_6 \rangle$, $\langle T_4 \rangle$, $\langle \text{CNOTNOT} \rangle$, and $\langle \text{CNOT} \rangle$.

▶ **Theorem 11.** *All pairs of classes asserted to be distinct by Theorem 3, are distinct.*

Proof. In each case, one just needs to observe that the gate that generates a given class A, satisfies some invariant violated by the gate that generates another class B. (Here we are using the "gate definitions" of the classes, which will be proven equivalent to the invariant definitions in Section 7.) So for example, $\langle \text{Fredkin} \rangle$ cannot contain CNOT because Fredkin is conservative; conversely, $\langle \text{CNOT} \rangle$ cannot contain Fredkin because CNOT is affine.

The only tricky classes are those involving NOT and NOTNOT gates: indeed, these classes *do* sometimes coincide, as noted in Theorem 3. However, in all cases where the classes are distinct, their distinctness is witnessed by the following invariants:

- $\langle \text{Fredkin}, \text{NOT} \rangle$ and $\langle \text{Fredkin}, \text{NOTNOT} \rangle$ are conservative in their linear part.
- $\langle \text{CNOTNOT}, \text{NOT} \rangle$ is parity-preserving in its linear part.
- $\langle F_4, \text{NOT} \rangle = \langle T_4, \text{NOT} \rangle$ and $\langle F_4, \text{NOTNOT} \rangle = \langle T_4, \text{NOTNOT} \rangle$ are orthogonal in their linear part (isometries).
- $\langle T_6, \text{NOT} \rangle$ and $\langle T_6, \text{NOTNOT} \rangle$ are orthogonal and mod-4-preserving in their linear part.

[13] The upshot of the Galois connection for clones [15] is that, if we could prove that a list of invariants for a given gate set S was the *complete* list of invariants satisfied by S, then this second part of the proof would be unnecessary: it would follow automatically that S generates all reversible transformations that satisfy the invariants. But this raises the question: how do we prove that a list of invariants for S is complete? In each case, the easiest way we could find to do this, was just by explicitly describing circuits of S-gates to generate all transformations that satisfy the stated invariants.

As a final remark, even if a reversible transformation is implemented with the help of ancilla bits, as long as the ancilla bits start and end in the same state $a_1 \dots a_k$, they have no effect on any of the invariants discussed above, and for that reason are irrelevant. ◀

6 Hamming Weights and Inner Products

The purpose of this section is to collect various mathematical results about what a reversible transformation $G : \{0,1\}^n \to \{0,1\}^n$ can and cannot do to the Hamming weight of its input, or to the inner product of two inputs. That is, we study the possible relationships that can hold between $|x|$ and $|G(x)|$, or between $x \cdot y$ and $G(x) \cdot G(y)$ (especially modulo various positive integers k). Not only are these results used heavily in the rest of the classification, but some of them might be of independent interest.

6.1 Ruling Out Mod-Shifters

Call a reversible transformation a *mod-shifter* if it always shifts the Hamming weight mod k of its input string by some fixed, nonzero amount. When $k = 2$, clearly mod-shifters exist: indeed, the humble NOT gate satisfies $|\text{NOT}(x)| \equiv |x| + 1 \pmod 2$ for all $x \in \{0,1\}$, and likewise for any other parity-flipping gate. However, we now show that $k = 2$ is the *only* possibility: mod-shifters do not exist for any larger k.

▶ **Theorem 12.** *There are no mod-shifters for $k \geq 3$. In other words: let G be a reversible transformation on n-bit strings, and suppose*

$$|G(x)| \equiv |x| + j \pmod k$$

for all $x \in \{0,1\}^n$. Then either $j = 0$ or $k = 2$.

Proof. Suppose the above equation holds for all x. Then introducing a new complex variable z, we have

$$z^{|G(x)|} \equiv z^{|x|+j} \left(\text{mod } \left(z^k - 1\right)\right)$$

(since working mod $z^k - 1$ is equivalent to setting $z^k = 1$). Since the above is true for all x,

$$\sum_{x \in \{0,1\}^n} z^{|G(x)|} \equiv \sum_{x \in \{0,1\}^n} z^{|x|} z^j \left(\text{mod } \left(z^k - 1\right)\right). \tag{1}$$

By reversibility, we have

$$\sum_{x \in \{0,1\}^n} z^{|G(x)|} = \sum_{x \in \{0,1\}^n} z^{|x|} = (z+1)^n.$$

Therefore equation (1) simplifies to

$$(z+1)^n \left(z^j - 1\right) \equiv 0 \left(\text{mod } \left(z^k - 1\right)\right).$$

Now, since $z^k - 1$ has no repeated roots, it can divide $(z+1)^n \left(z^j - 1\right)$ only if it divides $(z+1)\left(z^j - 1\right)$. For this we need either $j = 0$, causing $z^j - 1 = 0$, or else $j = k - 1$ (from degree considerations). But it is easily checked that the equality

$$z^k - 1 = (z+1)\left(z^{k-1} - 1\right)$$

holds only if $k = 2$. ◀

6.2 Inner Products Mod k

We have seen that there exist *orthogonal* gates (such as the T_k gates), which preserve inner products mod 2. In this section, we first show that no reversible gate that changes Hamming weights can preserve inner products mod k for any $k \geq 3$. We then observe that, if a reversible gate is orthogonal, then it must be linear, and we give necessary and conditions for orthogonality.

▶ **Theorem 13.** *Let G be a non-conservative n-bit reversible gate, and suppose*

$$G(x) \cdot G(y) \equiv x \cdot y \pmod{k}$$

for all $x, y \in \{0,1\}^n$. Then $k = 2$.

Proof. As in the proof of Theorem 12, we promote the congruence to a congruence over complex polynomials:

$$z^{G(x)\cdot G(y)} \equiv z^{x\cdot y} \pmod{z^k - 1}$$

Fix a string $x \in \{0,1\}^n$ such that $|G(x)| > |x|$, which must exist because G is non-conservative. Then sum the congruence over all y:

$$\sum_{y\in\{0,1\}^n} z^{G(x)\cdot G(y)} \equiv \sum_{y\in\{0,1\}^n} z^{x\cdot y} \pmod{z^k - 1}.$$

The summation on the right simplifies as follows.

$$\begin{aligned}\sum_{y\in\{0,1\}^n} z^{x\cdot y} &= \sum_{y\in\{0,1\}^n} \prod_{i=1}^{n} z^{x_i y_i} = \prod_{i=1}^{n} \sum_{y_i\in\{0,1\}} z^{x_i y_i} = \prod_{i=1}^{n} (1 + z^{x_i}) = (1+z)^{|x|}\, 2^{n-|x|}, \\ &= \left(\frac{1+z}{2}\right)^{|x|} 2^n.\end{aligned}$$

Similarly,

$$\sum_{y\in\{0,1\}^n} z^{G(x)\cdot G(y)} = \left(\frac{1+z}{2}\right)^{|G(x)|} 2^n,$$

since summing over all y is the same as summing over all $G(y)$. So we have

$$\begin{aligned}\left(\frac{1+z}{2}\right)^{|G(x)|} 2^n &\equiv \left(\frac{1+z}{2}\right)^{|x|} 2^n \pmod{z^k - 1}, \\ 0 &\equiv (1+z)^{|x|} 2^{-|G(x)|} \left(2^{|G(x)|-|x|} - (1+z)^{|G(x)|-|x|}\right) \pmod{z^k - 1},\end{aligned}$$

or equivalently, letting

$$p(x) := 2^{|G(x)|-|x|} - (1+z)^{|G(x)|-|x|},$$

we find that $z^k - 1$ divides $(1+z)^{|x|} p(x)$ as a polynomial. Now, the roots of $z^k - 1$ lie on the unit circle centered at 0. Meanwhile, the roots of $p(x)$ lie on the circle in the complex plane of radius 2, centered at -1. The only point of intersection of these two circles is $z = 1$, so that is the only root of $z^k - 1$ that can be covered by $p(x)$. On the other hand, clearly $z = -1$ is the only root of $(1+z)^{|x|}$. Hence, the only roots of $z^k - 1$ are 1 and -1, so we conclude that $k = 2$. ◀

We now study reversible transformations that preserve inner products mod 2.

▶ **Lemma 14.** *Every orthogonal gate G is linear.*

Proof. Suppose

$$G(x) \cdot G(y) \equiv x \cdot y \pmod 2.$$

Then for all x, y, z,

$$\begin{aligned} G(x \oplus y) \cdot G(z) &\equiv (x \oplus y) \cdot z \\ &\equiv x \cdot z + y \cdot z \\ &\equiv G(x) \cdot G(z) + G(y) \cdot G(z) \\ &\equiv (G(x) \oplus G(y)) \cdot G(z) \pmod 2. \end{aligned}$$

But if the above holds for all possible z, then

$$G(x \oplus y) \equiv G(x) \oplus G(y) \pmod 2. \quad \blacktriangleleft$$

Theorem 13 and Lemma 14 have the following corollary.

▶ **Corollary 15.** *Let G be any non-conservative, nonlinear gate. Then for all $k \geq 2$, there exist inputs x, y such that*

$$G(x) \cdot G(y) \not\equiv x \cdot y \pmod k.$$

Also:

▶ **Lemma 16.** *A linear transformation $G(x) = Ax$ is orthogonal if and only if $A^T A$ is the identity: that is, if A's column vectors satisfy $|v_i| \equiv 1 \pmod 2$ for all i and $v_i \cdot v_j \equiv 0 \pmod 2$ for all $i \neq j$.*

Proof. This is just the standard characterization of orthogonal matrices; that we are working over $\mathbb{F}_2$ is irrelevant. First, if G preserves inner products mod 2 then for all $i \neq j$,

$$\begin{aligned} 1 &\equiv e_i \cdot e_i \equiv (Ae_i) \cdot (Ae_i) \equiv |v_i| \pmod 2, \\ 0 &\equiv e_i \cdot e_j \equiv (Ae_i) \cdot (Ae_j) \equiv v_i \cdot v_j \pmod 2. \end{aligned}$$

Second, if G satisfies the conditions then

$$Ax \cdot Ay \equiv (Ax)^T Ay \equiv x^T (A^T A) y \equiv x^T y \equiv x \cdot y \pmod 2. \quad \blacktriangleleft$$

6.3 Why Mod 2 and Mod 4 Are Special

Recall that $\wedge$ denotes bitwise AND. We first need an "inclusion/exclusion formula" for the Hamming weight of a bitwise sum of strings.

▶ **Lemma 17.** *For all $v_1, \ldots, v_t \in \{0,1\}^n$, we have*

$$|v_1 \oplus \cdots \oplus v_t| = \sum_{\emptyset \subset S \subseteq [t]} (-2)^{|S|-1} \left| \bigwedge_{i \in S} v_i \right|.$$

Proof. It suffices to prove the lemma for $n = 1$, since in the general case we are just summing over all $i \in [n]$. Thus, assume without loss of generality that $v_1 = \cdots = v_t = 1$. Our problem then reduces to proving the following identity:

$$\sum_{i=1}^{t}(-2)^{i-1}\binom{t}{i} = \begin{cases} 0 & \text{if } t \text{ is even} \\ 1 & \text{if } t \text{ is odd,} \end{cases}$$

which follows straightforwardly from the binomial theorem. ◀

▶ **Lemma 18.** *No nontrivial affine gate G is conservative.*

Proof. Let $G(x) = Ax \oplus b$; then $|G(0^n)| = |0^n| = 0$ implies $b = 0^n$. Likewise, $|G(e_i)| = |e_i| = 1$ for all i implies that A is a permutation matrix. But then G is trivial. ◀

▶ **Theorem 19.** *If G is a nontrivial linear gate that preserves Hamming weight mod k, then either $k = 2$ or $k = 4$.*

Proof. For all x, y, we have

$$\begin{aligned} |x| + |y| - 2(x \cdot y) &\equiv |x \oplus y| \\ &\equiv |G(x \oplus y)| \\ &\equiv |G(x) \oplus G(y)| \\ &\equiv |G(x)| + |G(y)| - 2(G(x) \cdot G(y)) \\ &\equiv |x| + |y| - 2(G(x) \cdot G(y)) \pmod{k}, \end{aligned}$$

where the first and fourth lines used Lemma 17, the second and fifth lines used that G is mod-k-preserving, and the third line used linearity. Hence

$$2(x \cdot y) \equiv 2(G(x) \cdot G(y)) \pmod{k}. \tag{2}$$

If k is odd, then equation (2) implies

$$x \cdot y \equiv G(x) \cdot G(y) \pmod{k}.$$

But since G is nontrivial and linear, Lemma 18 says that G is non-conservative. So by Theorem 13, the above equation cannot be satisfied for any odd $k \geq 3$. Likewise, if k is even, then (2) implies

$$x \cdot y \equiv G(x) \cdot G(y) \left(\mod \frac{k}{2}\right).$$

Again by Theorem 13, the above can be satisfied only if $k = 2$ or $k = 4$. ◀

▶ **Theorem 20.** *Let $\{o_i\}_{i=1}^n$ be an orthonormal basis over $\mathbb{F}_2$. An affine transformation $F(x) = Ax \oplus b$ is mod-4-preserving if and only if $|b| \equiv 0 \pmod 4$, and the vectors $v_i := Ao_i$ satisfy $|v_i| + 2(v_i \cdot b) \equiv |o_i| \pmod 4$ for all i and $v_i \cdot v_j \equiv 0 \pmod 2$ for all $i \neq j$.*

Proof. First, if F is mod-4-preserving, then

$$0 \equiv |F(0^n)| \equiv |A0^n \oplus b| \equiv |b| \pmod 4,$$

and hence

$$|o_i| \equiv |F(o_i)| \equiv |Ao_i \oplus b| \equiv |v_i \oplus b| \equiv |v_i| + |b| - 2(v_i \cdot b) \equiv |v_i| + 2(v_i \cdot b) \pmod 4$$

for all i, and hence

$$\begin{aligned}|o_i + o_j| &\equiv |F(o_i \oplus o_j)| \\ &\equiv |v_i \oplus v_j \oplus b| \\ &\equiv |v_i| + |v_j| + |b| - 2(v_i \cdot v_j) - 2(v_i \cdot b) - 2(v_j \cdot b) + 4|v_i \wedge v_j \wedge b| \\ &\equiv |v_i| + |v_j| + 2(v_i \cdot v_j) + 2(v_i \cdot b) + 2(v_j \cdot b) \pmod 4 \\ &\equiv |o_i| + |o_j| + 2(v_i \cdot v_j) \pmod 4\end{aligned}$$

for all $i \neq j$, from which we conclude that $v_i \cdot v_j \equiv 0 \pmod 2$.

Second, if F satisfies the conditions, then for any $x = \sum_{i \in S} o_i$, we have

$$\begin{aligned}|F(x)| &= \left| b \oplus \sum_{i \in S} v_i \right| \\ &= |b| + \sum_{i \in S} |v_i| - 2\sum_{i \in S} (b \cdot v_i) - 2 \sum_{i \in S \,<\, j \in S} (v_i \cdot v_j) + 4(\cdots) \\ &\equiv \sum_{i \in S} |v_i| - 2(b \cdot v_i) \\ &\equiv \sum_{i \in S} |o_i| \pmod 4,\end{aligned}$$

where the second line follows from Lemma 17. Furthermore, we have that

$$|x| = \left| \sum_{i \in S} o_i \right| = \sum_{i \in S} |o_i| - 2 \sum_{i \in S < j \in S} (o_i \cdot o_j) + 4(\ldots) \equiv \sum_{i \in S} |o_i| \pmod 4,$$

where the last equality follows from the fact that $\{o_i\}_{i=1}^n$ is an orthonormal basis. Therefore, we conclude that $|F(x)| \equiv |x| \pmod 4$. ◀

We note two corollaries of Theorem 20 for later use.

▶ **Corollary 21.** *Any linear transformation $A \in \mathbb{F}_2^{n \times n}$ that preserves Hamming weight mod 4 is also orthogonal.*

▶ **Corollary 22.** *An orthogonal transformation $A \in \mathbb{F}_2^{n \times n}$ preserves Hamming weight mod 4 if and only if all of its columns have Hamming weight 1 mod 4.*

7 Reversible Circuit Constructions

In this section, we show that all the classes of reversible transformations listed in Theorem 3, are indeed generated by the gates that we claimed, by giving explicit synthesis procedures. In order to justify Theorem 8, we also verify that in each case, only $O(1)$ ancilla bits are needed, even though this constraint makes some of the constructions more complicated than otherwise.

Many of our constructions – those for Toffoli and CNOT, for example – have appeared in various forms in the reversible computing literature, and are included here only for completeness. Others – those for C_k and F_4, for example – are new as far as we know, but not hard.

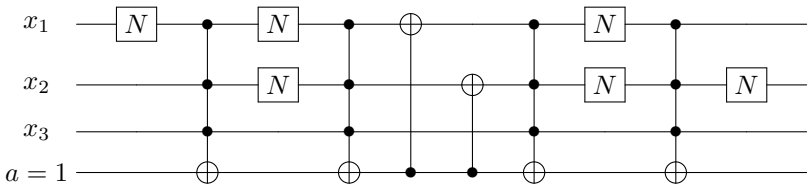

Figure 4 Circuit for the transposition $\sigma_{011,101}$ after simplification.

7.1 Non-Affine Circuits

We start with the non-affine classes: $\langle \text{Toffoli} \rangle$, $\langle \text{Fredkin} \rangle$, $\langle \text{Fredkin}, \text{C}_k \rangle$, and $\langle \text{Fredkin}, \text{NOT} \rangle$.

▶ **Theorem 23** (variants in [29, 26]). Toffoli *generates all reversible transformations on n bits, using only 2 ancilla bits.*[14]

Proof. Any reversible transformation $F : \{0,1\}^n \to \{0,1\}^n$ is a permutation of n-bit strings, and any permutation can be written as a product of transpositions. So it suffices to show how to use Toffoli gates to implement an arbitrary transposition $\sigma_{y,z}$: that is, a mapping that sends $y = y_1 \dots y_n$ to $z = z_1 \dots z_n$ and z to y, and all other n-bit strings to themselves.

Given any n-bit string w, let us define w-CNOT to be the $(n+1)$-bit gate that flips its last bit if its first n bits are equal to w, and that does nothing otherwise. (Thus, the Toffoli gate is 11-CNOT, while CNOT itself is 1-CNOT.) Given y-CNOT and z-CNOT gates, we can implement the transposition $\sigma_{y,z}$ as follows on input x:

1. Initialize an ancilla bit, $a = 1$.
2. Apply y-CNOT (x, a).
3. Apply z-CNOT (x, a).
4. Apply NOT gates to all x_i's such that $y_i \neq z_i$.
5. For each i such that $y_i \neq z_i$, apply CNOT (a, x_i).
6. Apply z-CNOT (x, a).
7. Apply y-CNOT (x, a).

Thus, all that remains is to implement w-CNOT using Toffoli. Observe that we can simulate any w-CNOT using 1^n-CNOT by negating certain input bits (namely, those for which $w_i = 0$) before and after we apply the 1^n-CNOT. An example of the transposition $\sigma_{011,101}$ is given in Figure 4.

So it suffices to implement 1^n-CNOT, with control bits $x_1 \dots x_n$ and target bit y. The base case is $n = 2$, which we implement directly using Toffoli. For $n \geq 3$, we do the following.

- Let a be an ancilla.
- Apply $1^{\lceil n/2 \rceil}$-CNOT $\left(x_1 \dots x_{\lceil n/2 \rceil}, a\right)$.
- Apply $1^{\lfloor n/2 \rfloor + 1}$-CNOT $\left(x_{\lceil n/2 \rceil + 1} \dots x_n, a, y\right)$.
- Apply $1^{\lceil n/2 \rceil}$-CNOT $\left(x_1 \dots x_{\lceil n/2 \rceil}, a\right)$.
- Apply $1^{\lfloor n/2 \rfloor + 1}$-CNOT $\left(x_{\lceil n/2 \rceil + 1} \dots x_n, a, y\right)$.

The crucial point is that this construction works whether the ancilla is initially 0 or 1. In other words, we can use *any* bit which is not one of the inputs, instead of a new ancilla. For instance, we can have one bit dedicated for use in 1^n-CNOT gates, which we use in the

[14] Notice that we need at least 2 so that we can generate CNOT and NOT using Toffoli.

recursive applications of $1^{\lceil n/2 \rceil}$-CNOT and $1^{\lfloor n/2 \rfloor + 1}$-CNOT, and the recursive applications within them, and so on.[15]

Carefully inspecting the above proof shows that $O\left(n^2 2^n\right)$ gates and 2 ancilla bits suffice to generate any transformation. ◀

The particular construction above was inspired by a result of Ben-Or and Cleve [7], in which they compute algebraic formulas in a straight-line computation model using a constant number of registers. We note that Toffoli [29] proved a version of Theorem 23, but with $O(n)$ ancilla bits rather than $O(1)$. More recently, Shende et al. [26] gave a slightly more complicated construction that uses only 1 ancilla bit (assuming that we have CNOT and NOT gates in addition to Toffoli gates), and that also gives explicit bounds on the number of Toffoli gates required based on the number of fixed points of the permutation. Recall that at least 1 ancilla bit is needed by Proposition 9.

Next, let CCSWAP, or Controlled-Controlled-SWAP, be the 4-bit gate that swaps its last two bits if its first two bits are both 1, and otherwise does nothing.

▶ **Proposition 24.** Fredkin *generates* CCSWAP.

Proof. Let a be an ancilla bit initialized to 0. We implement CCSWAP (x, y, z, w) by applying Fredkin (x, y, a), then Fredkin (a, z, w), then again Fredkin (x, y, a). ◀

We can now prove an analogue of Theorem 23 for conservative transformations.

▶ **Theorem 25.** Fredkin *generates all conservative transformations on n bits, using only* 5 *ancilla bits.*

Proof. In this proof, we will use the *dual-rail representation*, in which 0 is encoded as 01 and 1 is encoded as 10. We will also use Proposition 24, that Fredkin generates CCSWAP.

As in Theorem 23, we can decompose any reversible transformation $F : \{0,1\}^n \to \{0,1\}^n$ as a product of transpositions $\sigma_{y,z}$. In this case, each $\sigma_{y,z}$ transposes two n-bit strings $y = y_1 \ldots y_n$ and $z = z_1 \ldots z_n$ of the same Hamming weight.

Given any n-bit string w, let us define w-CSWAP to be the $(n+2)$-bit gate that swaps its last two bits if its first n bits are equal to w, and that does nothing otherwise. (Thus, Fredkin is 1-CSWAP, while CCSWAP is 11-CSWAP.) Then given y-CSWAP and z-CSWAP gates, where $|y| = |z|$, as well as CCSWAP gates, we can implement the transposition $\sigma_{y,z}$ on input x as follows:

1. Initialize two ancilla bits (comprising one dual-rail register) to $a\overline{a} = 01$.
2. Apply y-CSWAP $(x_1 \ldots x_n, a, \overline{a})$.
3. Apply z-CSWAP $(x_1 \ldots x_n, a, \overline{a})$.
4. Pair off the i's such that $y_i = 1$ and $z_i = 0$, with the equally many j's such that $z_j = 1$ and $y_j = 0$. For each such (i, j) pair, apply Fredkin (a, x_i, x_j).
5. Apply z-CSWAP $(x_1 \ldots x_n, a, \overline{a})$.
6. Apply y-CSWAP $(x_1 \ldots x_n, a, \overline{a})$.

[15] The number of Toffoli gates $T(n)$ needed to implement a 1^n-CNOT (which dominates the cost of a transposition) by this recursive scheme, is given by the recurrence

$$T(n) = 2T(1 + \lfloor n/2 \rfloor) + 2T(\lceil n/2 \rceil)$$

which we solve to obtain $T(n) = O\left(n^2\right)$.

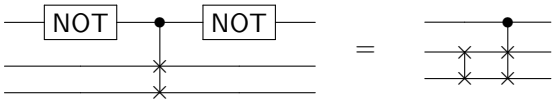

Figure 5 Removing NOT gates from the Fredkin circuit.

The logic here is exactly the same as in the construction of transpositions in Theorem 23; the only difference is that now we need to conserve Hamming weight.

All that remains is to implement w-CSWAP using CCSWAP. First let us show how to implement 1^n-CSWAP using CCSWAP. Once again, we do so using a recursive construction. For the base case, $n = 2$, we just use CCSWAP. For $n \geq 3$, we implement 1^n-CSWAP $(x_1, \ldots, x_n, y, z)$ as follows:

- Initialize two ancilla bits (comprising one dual-rail register) to $a\overline{a} = 01$.
- Apply $1^{\lceil n/2 \rceil}$-CSWAP $\left(x_1 \ldots x_{\lceil n/2 \rceil}, a, \overline{a}\right)$.
- Apply $1^{\lfloor n/2 \rfloor + 1}$-CSWAP $\left(x_{\lceil n/2 \rceil + 1} \ldots x_n, a, y, z\right)$.
- Apply $1^{\lceil n/2 \rceil}$-CSWAP $\left(x_1 \ldots x_{\lceil n/2 \rceil}, a, \overline{a}\right)$.
- Apply $1^{\lfloor n/2 \rfloor + 1}$-CSWAP $\left(x_{\lceil n/2 \rceil + 1} \ldots x_n, a, y, z\right)$.

The logic is the same as in the construction of 1^n-CNOT in Theorem 23 except we now use 2 ancilla bits for the dual-rail representation.

Finally, we need to implement w-CSWAP $(x_1 \ldots x_n, y, z)$, for arbitrary w, using 1^n-CSWAP. We do so by first constructing w-CSWAP from NOT gates and 1^n-CSWAP. Observe that we only use the NOT gate on the control bits of the Fredkin gates used during the construction so the equivalence given in Figure 5 holds (i.e., we can remove the NOT gates).

Hence, we can build a w-CSWAP out of CCSWAPs using only 5 ancilla bits: 1 for CCSWAP, 2 for the 1^n-CSWAP, and 2 for a transposition. ◀

We note that, before the above construction was found by the authors, unpublished and independent work by Siyao Xu and Qian Yu first showed that $O(1)$ ancillas were sufficient and has since been improved to exactly 1 ancilla [31].

In [11], the result that Fredkin generates all conservative transformations is stated without proof, and credited to B. Silver. We do not know how many ancilla bits Silver's construction used.

Next, we prove an analogue of Theorem 23 for the mod-k-respecting transformations, for all $k \geq 2$. First, let CC_k, or Controlled-C_k, be the $(k+1)$-bit gate that applies C_k to the final k bits if the first bit is 1, and does nothing if the first bit is 0.

▶ **Proposition 26.** Fredkin $+ \mathrm{C}_k$ *generates* CC_k*, using* 2 *ancilla bits, for all* $k \geq 2$*.*

Proof. To implement CC_k on input bits $x, y_1 \ldots y_k$, we do the following:

1. Initialize ancilla bits a, b to $0, 1$ respectively.
2. Use Fredkin gates and swaps to swap y_1, y_2 with a, b, conditioned on $x = 0$.[16]
3. Apply C_k to $y_1 \ldots y_k$.
4. Repeat step 2. ◀

Then we have the following.

[16] In more detail, use Fredkin gates to swap y_1, y_2 with a, b, conditioned on $x = 1$. Then swap y_1, y_2 with a, b unconditionally.

▶ **Theorem 27.** Fredkin + CC_k *generates all mod-k-preserving transformations, for* $k \geq 1$*, using only* 5 *ancilla bits.*

Proof. The proof is exactly the same as that of Theorem 25, except for one detail. Namely, let y and z be n-bit strings such that $|y| \equiv |z| \pmod{k}$. Then in the construction of the transposition $\sigma_{y,z}$ from y-CSWAP and z-CSWAP gates, when we are applying step 5, it is possible that $|y| - |z|$ is some nonzero multiple of k, say qk. If so, then we can no longer pair off each i such that $y_i = 1$ and $z_i = 0$ with a unique j such that $z_j = 1$ and $y_j = 0$: after we have done that, there will remain a surplus of '1' bits of size qk, either in y or in z, as well as a matching surplus of '0' bits of size qk in the other string. However, we can get rid of both surpluses using q applications of a CC_k gate (which we have by Proposition 26), with c as the control bit. ◀

As a special case of Theorem 27, note that Fredkin + CC_1 = Fredkin + CNOT generates all mod-1-preserving transformations – or in other words, all transformations.

We just need one additional fact about the C_k gate.

▶ **Proposition 28.** C_k *generates* Fredkin*, using* $k-2$ *ancilla bits, for all* $k \geq 3$*.*

Proof. Let $a_1 \ldots a_{k-2}$ be ancilla bits initially set to 1. Then to implement Fredkin on input bits x, y, z, we apply:

$$\mathrm{C}_k(x, y, a_1 \ldots a_{k-2}), \mathrm{C}_k(x, z, a_1 \ldots a_{k-2}), \mathrm{C}_k(x, y, a_1 \ldots a_{k-2}).$$ ◀

Combining Theorem 27 with Proposition 28 now yields the following.

▶ **Corollary 29.** C_k *generates all mod-k-preserving transformations for* $k \geq 3$*, using only* $k+3$ *ancilla bits.*

Finally, we handle the parity-flipping case.

▶ **Proposition 30.** Fredkin + NOTNOT *(and hence,* Fredkin + NOT*) generates* CC_2*.*

Proof. This follows from Proposition 26, if we recall that C_2 is equivalent to NOTNOT up to an irrelevant bit-swap. ◀

▶ **Theorem 31.** Fredkin + NOT *generates all parity-respecting transformations, using only* 5 *ancilla bits.*

Proof. Let F be any parity-flipping transformation, and let F' be F followed by NOT on one of the output bits. Then F' is parity-preserving. So by Theorem 27, we can implement F' using Fredkin + CC_2 (and we have CC_2 by Proposition 30). We can then apply another NOT gate to get F itself. ◀

One consequence of Theorem 31 is that every parity-flipping transformation can be constructed from parity-preserving gates and exactly one NOT gate.

7.2 Affine Circuits

It is well-known that CNOT is a "universal affine gate":

▶ **Theorem 32.** CNOT *generates all affine transformations, with only* 1 *ancilla bit (or* 0 *for linear transformations).*

Proof. Let $G(x) = Ax \oplus b$ be the affine transformation that we want to implement, for some invertible matrix $A \in \mathbb{F}_2^{n \times n}$. Then given an input $x = x_1 \dots x_n$, we first use CNOT gates (at most $\binom{n}{2}$ of them) to map x to Ax, by reversing the sequence of row-operations that maps A to the identity matrix in Gaussian elimination. Finally, if $b = b_1 \dots b_n$ is nonzero, then for each i such that $b_i = 1$, we apply a CNOT from an ancilla bit that is initialized to 1. ◀

A simple modification of Theorem 32 handles the parity-preserving case.

▶ **Theorem 33.** CNOTNOT *generates all parity-preserving affine transformations with only* 1 *ancilla bit (or* 0 *for linear transformations).*

Proof. Let $G(x) = Ax \oplus b$ be a parity-preserving affine transformation. We first construct the linear part of G using Gaussian elimination. Notice that for G to be parity-preserving, the columns v_i of A must satisfy $|v_i| \equiv 1 \pmod 2$ for all i. For this reason, the row-elimination steps come in pairs, so we can implement them using CNOTNOT. Notice further that since G is parity-preserving, we must have $|b| \equiv 0 \pmod 2$. So we can map Ax to $Ax \oplus b$, by using CNOTNOT gates plus one ancilla bit set to 1 to simulate NOTNOT gates. ◀

Likewise (though, strictly speaking, we will not need this for the proof of Theorem 3):

▶ **Theorem 34.** CNOTNOT + NOT *generates all parity-respecting affine transformations using no ancilla bits.*

Proof. Use Theorem 33 to map x to Ax, and then use NOT gates to map Ax to $Ax \oplus b$. ◀

We now move on to the more complicated cases of $\langle \mathrm{F}_4 \rangle$, $\langle \mathrm{T}_6 \rangle$, and $\langle \mathrm{T}_4 \rangle$.

▶ **Theorem 35.** F_4 *generates all mod-4-preserving affine transformations using no ancilla bits.*

Proof. Let $F(x) = Ax \oplus b$ be an n-bit affine transformation, $n \geq 2$, that preserves Hamming weight mod 4. Using F_4 gates, we will show how to map $F(x) = y_1 \dots y_n$ to $x = x_1 \dots x_n$. Reversing the construction then yields the desired map from x to $F(x)$.

At any point in time, each y_j is some affine function of the x_i's. We say that x_i "occurs in" y_j, if y_j depends on x_i. At a high level, our procedure will consist of the following steps, repeated up to $n - 3$ times:

1. Find an x_i that does not occur in every y_j.
2. Manipulate the y_j's so that x_i occurs in exactly *one* y_j.
3. Argue that no *other* $x_{i'}$ can then occur in that y_j. Therefore, we have recursively reduced our problem to one involving a reversible, mod-4-preserving, affine function on $n - 1$ variables.

It is not hard to see that the only mod-4-preserving affine functions on 3 or fewer variables, are permutations of the bits. So if we can show that the three steps above can always be carried out, then we are done.

First, since A is invertible, it is not the all-1's matrix, which means that there must be an x_i that does not occur in every y_j.

Second, if there are at least three occurrences of x_i, then apply F_4 to three positions in which x_i occurs, plus one position in which x_i does not occur. The result of this is to decrease the number of occurrences of x_i by 2. Repeat until there are at most two occurrences of x_i. Since F_4 is mod-4-preserving and affine, the resulting transformation $F'(x) = A'x + b'$ must still be mod-4-preserving and affine, so it must still satisfy the conditions of Lemma 20. In

particular, no column vector of A' can have even Hamming weight. Since two occurrences of x_i would necessitate such a column vector, we know that x_i must occur only once.

Third, if x_i occurs only once in $F'(x)$, then the corresponding column vector v_i has exactly one nonzero element. Since $|v_i| = 1$, we know by Lemma 20 that $v_i \cdot b \equiv 0 \pmod 2$, which means that b has a 0 in the position where v_i has a 1. Now consider the row of A' that includes the nonzero entry of v_i. If any other column $v_{i'}$ is also nonzero in that row, then $v_i \cdot v_{i'} \equiv 1 \pmod 2$, which once again contradicts the conditions of Lemma 20. Thus, no other $x_{i'}$ occurs in the same y_j that x_i occurs in. Indeed no constant occurs there either, since otherwise F' would no longer be mod-4-preserving. So we have reduced to the $(n-1) \times (n-1)$ case. ◀

The same argument, with slight modifications, handles $\langle \mathrm{T}_4 \rangle$ and $\langle \mathrm{T}_6 \rangle$.

▶ **Theorem 36.** T_4 *generates all orthogonal transformations, using no ancilla bits.*

Proof. The construction is identical to that of Theorem 35, except with T_4 instead of F_4. When reducing the number of occurrences of x_i to at most 2, Lemma 16 assures us that $|v_i| \equiv 1 \pmod 2$. ◀

▶ **Theorem 37.** T_6 *generates all mod-4-preserving linear transformations, using no ancilla bits.*

Proof. The construction is identical to that of Theorem 35, except for the following change. Rather than using F_4 to reduce the number of occurrences of some x_i to at most 2, we now use T_6 to reduce the number of occurrences of x_i to at most 4. (If there are 5 or more occurrences, then T_6 can always decrease the number by 4.) We then appeal to Corollary 22, which says that $|v_i| \equiv 1 \pmod 4$ for each i. This implies that no x_i can occur 2, 3, or 4 times in the output vector. But that can only mean that x_i occurs once. ◀

By Lemma 14 and Corollary 21, an equivalent way to state Theorem 37 is that T_6 generates all affine transformations that are both mod-4-preserving and orthogonal.

All that remains is some "cleanup work" (which, again, is not even needed for the proof of Theorem 3).

▶ **Theorem 38.** $\mathrm{T}_6 + \mathrm{NOT}$ *generates all affine transformations that are mod-4-preserving (and therefore orthogonal) in their linear part.*

$\mathrm{T}_6 + \mathrm{NOTNOT}$ *generates all parity-preserving affine transformations that are mod-4-preserving (and therefore orthogonal) in their linear part.*

$\mathrm{F}_4 + \mathrm{NOT}$ *(or equivalently,* $\mathrm{T}_4 + \mathrm{NOT}$*) generates all isometries.*

$\mathrm{F}_4 + \mathrm{NOTNOT}$ *(or equivalently,* $\mathrm{T}_4 + \mathrm{NOTNOT}$*) generates all parity-preserving isometries.*

NOT *generates all degenerate transformations.*

NOTNOT *generates all parity-preserving degenerate transformations.*

In none of these cases are any ancilla bits needed.

Proof. As in Theorem 34, we simply apply the relevant construction for the linear part (e.g., Theorem 36 or 37), then handle the affine part using NOT or NOTNOT gates. ◀

8 Open Problems

As discussed in Section 1, the central challenge we leave is to give a complete classification of all *quantum* gate sets acting on qubits, in terms of which unitary transformations they can generate or approximate. Here, just like in this paper, one should assume that qubit-swaps are free, and that arbitrary ancillas are allowed as long as they are returned to their initial states.

A possible first step in the direction we want, which would involve Lie algebras, would be to classify all sets of 1- and 2-qubit gates. A second step would be to classify qubit Hamiltonians (i.e., the infinitesimal-time versions of unitary gates), in terms of which n-qubit Hamiltonians they can be used to generate. Here the recent work of Cubitt and Montanaro [10], which classifies qubit Hamiltonians in terms of the complexity of approximating ground state energies, might be relevant. Yet a third possibility would be to classify quantum gates under the assumption that intermediate measurements are allowed. Of course, these simplifications can also be combined.

On the classical side, we have left completely open the problem of classifying reversible gate sets over non-binary alphabets. In the non-reversible setting, it was discovered in the 1950s (see [20]) that Post's lattice becomes dramatically different and more complicated when we consider gates over a 3-element set rather than Boolean gates: for example, there is now an uncountable infinity of clones, rather than "merely" a countable infinity. Does anything similar happen in the reversible case? We know from Gu [14] that for alphabets of size greater than three, there exists a class that is not finitely generated. Recall that for binary alphabets, one gate always suffices to generate a particular class.

Even for reversible gates over (say) $\{0,1,2\}^n$, we cannot currently give an algorithm to decide whether a given gate G generates another gate H any better than the triple-exponential-time algorithm that comes from clone theory, nor can we give reasonable upper bounds on the number of gates or ancillas needed in the generating circuit, nor can we answer basic questions like whether every class is finitely generated.

Finally, can one reduce the number of gates in each of our circuit constructions to the limits imposed by Shannon-style counting arguments? What are the tradeoffs, if any, between the number of gates and the number of ancilla bits?

Acknowledgments. At the very beginning of this project, Emil Jeřábek [15] brought the $\langle \mathrm{C}_k \rangle$ and $\langle \mathrm{T}_6 \rangle$ classes to our attention, and also proved that every reversible gate class is characterized by invariants (i.e., that the "clone-coclone duality" holds for reversible gates). Also, Matthew Cook gave us encouragement, asked pertinent questions, and helped us understand the $\langle \mathrm{T}_4 \rangle$ class. We are grateful to both of them. We also thank Harry Altman, Adam Bouland, Seth Lloyd, Igor Markov, and particularly Siyao Xu for helpful discussions.

References

1 S. Aaronson and A. Arkhipov. The computational complexity of linear optics. *Theory of Computing*, 9(4):143–252, 2013. Conference version in Proceedings of ACM STOC 2011. ECCC TR10-170, arXiv:1011.3245.

2 S. Aaronson and A. Bouland. Generation of universal linear optics by any beam splitter. *Phys. Rev. A*, 89(6):062316, 2014. arXiv:1310.6718.

3 S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70(052328), 2004. arXiv:quant-ph/0406196.

4 Scott Aaronson, Daniel Grier, and Luke Schaeffer. The classification of reversible bit operations. *arXiv:1504.05155*, 2015.

5 D. Bacon, J. Kempe, D. P. DiVincenzo, D. A. Lidar, and K. B. Whaley. Encoded universality in physical implementations of a quantum computer. In R. Clark, editor, *Proceedings of the 1st International Conference on Experimental Implementations of Quantum Computation*, page 257. Rinton, 2001. arXiv:quant-ph/0102140.

6 A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52(3457), 1995. arXiv:quant-ph/9503016.

7 M. Ben-Or and R. Cleve. Computing algebraic formulas with a constant number of registers. In *Proc. ACM STOC*, pages 254–257, 1988.

8 C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

9 R. Cleve and J. Watrous. Fast parallel circuits for the quantum Fourier transform. In *Proc. IEEE FOCS*, pages 526–536, 2000. arXiv:quant-ph/0006004.

10 T. Cubitt and A. Montanaro. Complexity classification of local Hamiltonian problems. In *Proc. IEEE FOCS*, pages 120–129, 2014. arXiv:1311.3161.

11 E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3-4):219–253, 1982.

12 D. Gottesman. Class of quantum error-correcting codes saturating the quantum Hamming bound. *Phys. Rev. A*, 54:1862–1868, 1996. arXiv:quant-ph/9604038.

13 D. Grier and L. Schaeffer. The classification of stabilizer operations over qubits. *ArXiv e-prints*, March 2016. arXiv:1603.03999.

14 Y. Gu. Some results on reversible gate classes over non-binary alphabets. *CoRR*, abs/1606.00804, 2016. URL: `http://arxiv.org/abs/1606.00804`.

15 E. Jeřábek. Answer to CS Theory StackExchange question on "classifying reversible gates". At `http://cstheory.stackexchange.com/questions/25730/classifying-reversible-gates`, 2014.

16 P. Kerntopf, M. A. Perkowski, and M. Khan. On universality of general reversible multiple-valued logic gates. In *IEEE International Symposium on Multiple-Valued Logic*, pages 68–73, 2004.

17 O. G. Kharlampovich and M. V. Sapir. Algorithmic problems in varieties. *International Journal of Algebra and Computation*, 5(04n05):379–602, 1995. `http://www.math.vanderbilt.edu/~msapir/ftp/pub/survey/survey.pdf`.

18 E. Knill and R. Laflamme. Power of one bit of quantum information. *Phys. Rev. Lett.*, 81(25):5672–5675, 1998. arXiv:quant-ph/9802037.

19 R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.

20 D. Lau. *Function Algebras on Finite Sets: Basic Course on Many-Valued Logic and Clone Theory*. Springer, 2006.

21 S. Lloyd. Any nonlinear one-to-one binary logic gate suffices for computation. Technical Report LA-UR-92-996, Los Alamos National Laboratory, 1992. arXiv:1504.03376.

22 K. Morita, T. Ogiro, K. Tanaka, and H. Kato. Classification and universality of reversible logic elements with one-bit memory. In *Proceedings of the 4th International Conference on Machines, Computations, and Universality*, pages 245–256. Springer-Verlag, 2005.

23 E. L. Post. *The two-valued iterative systems of mathematical logic*. Number 5 in Annals of Mathematics Studies. Princeton University Press, 1941.

24 M. Saeedi and I. L. Markov. Synthesis and optimization of reversible circuits–a survey. *ACM Computing Surveys*, 45(2):21, 2013. arXiv:1110.2574.

25 L. Schaeffer. Reversible Gate Classifier, 2015. https://github.com/lrschaeffer/Gate-Classifier.

26 V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722, 2003. arXiv:quant-ph/0207001.

27 Y. Shi. Both Toffoli and controlled-NOT need little help to do universal quantum computation. *Quantum Information and Computation*, 3(1):84–92, 2002. quant-ph/0205115.

28 I. Strazdins. Universal affine classification of Boolean functions. *Acta Applicandae Mathematica*, 46(2):147–167, 1997.

29 T. Toffoli. Reversible computing. In *Proc. Intl. Colloquium on Automata, Languages, and Programming (ICALP)*, pages 632–644. Springer, 1980.

30 A. De Vos and L. Storme. r-universal reversible logic gates. *Journal of Physics A: Mathematical and General*, 37(22):5815–5824, 2004.

31 S. Xu. Reversible logic synthesis with minimal usage of ancilla bits. *CoRR*, abs/1506.03777, 2015. URL: http://arxiv.org/abs/1506.03777.