Electronic Colloquium on Computational Complexity, Report No. 66 (2015)

The Classification of Reversible Bit Operations

Scott Aaronson[*] Daniel Grier[†] Luke Schaeffer[‡]

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 some arguments about polynomials, lattices, and Diophantine equations.

Contents

[*]MIT. Email: aaronson@csail.mit.edu. Supported by an Alan T. Waterman Award from the National Science Foundation, under grant no. 1249349.
[†]MIT. Email: grierd@mit.edu. Supported by an NSF Graduate Research Fellowship under Grant No. 1122374.
[‡]MIT. Email: lrs@mit.edu.

ISSN 1433-8092

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 [22] 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 finite 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. [5] showed universality for the CNOT gate plus the set of all 1-qubit gates, while Shi [26] 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* [11, 3] (that is, the CNOT, Hadamard, and $\pi/4$-Phase gates), as well as the stabilizer gates conjugated by 1-qubit unitary transformations. 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?[2] 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?

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

[1]In Appendix 12, we prove for completeness that if one *does* assume constants are free, then Post's lattice dramatically simplifies, with all non-universal gate sets either monotone or affine.

[2]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 [15], and log-depth quantum circuits [8]. 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.

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.

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 an obvious prerequisite 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 [10, 28, 17, 7, 19, 23] 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* [17], but deleting information is not necessary for universal computation.

Formally, a reversible gate is just a permutation $G : \{0,1\}^k \to \{0,1\}^k$ 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 [28]—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 $F : \{0,1\}^n \to \{0,1\}^n$ 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[3] 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 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*.[4]

[3] 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).

[4] In Section 2.3 and Appendix 13, 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

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. (In Appendix 14, we also calculate the exact number of 3-bit gates that generate each class.) 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, affine gates can be orthogonal or mod-2-preserving or mod-4-preserving in their linear part, but not in their affine part.

(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.

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

rule" causes only a small change to our classification theorem.

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 [24]. 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 $H : \{0,1\}^n \rightarrow \{0,1\}^n$ 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, $\operatorname{Ackermann}(n)$. 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. [18]), which grew out of Post's original work [22], 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 [12] 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 $G : \{0,1\}^n \rightarrow \{0,1\}^n$, 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., [14, 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:

- No non-conservative reversible gate can 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. In Appendix 15, we give alternative, more "combinatorial" proofs for the second and third statements.

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. We do this in Sections 8.3 and 8.4.
- 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 $x, y \oplus x, z \oplus x$. We do this in Section 9.2.

In both of these cases, our 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.[5] 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.

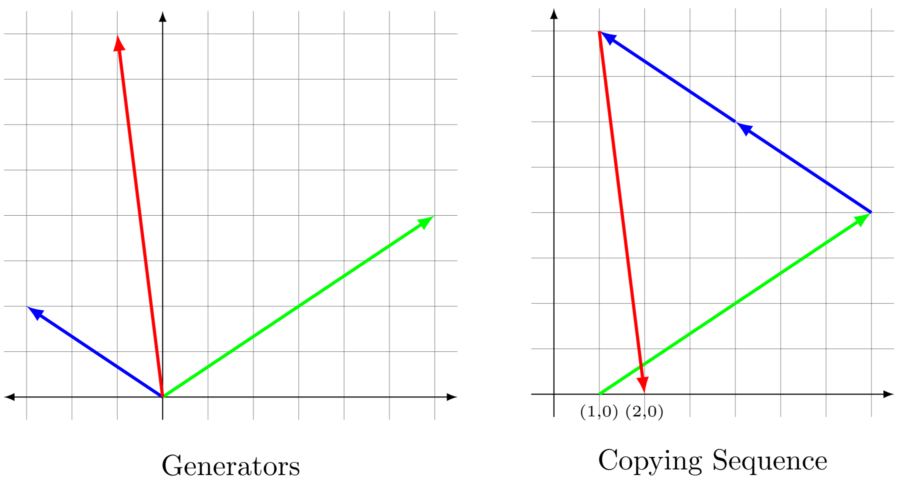

Figure 1: Moving within first quadrant of lattice to construct a COPY gate

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.

[5]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.

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 (for example, Section 9.4, on affine gate sets) 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 [23] 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 [19], 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. [13] and De Vos and Storme [29]. We will reprove this result for completeness in Section 8.2, as we use it as one ingredient in our proof.

There is also work by Morita et al. [21] 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 [27], 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.[6]

[6] 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

$$x\cdot(y\oplus z)\equiv x\cdot y+x\cdot z \pmod 2,$$

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

By $\operatorname{gar}(x)$, 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 [7], 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

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

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, a mod-2-respecting gate is also called *parity-respecting*, a mod-2-preserving gate is called *parity-preserving*, and a gate G such that

$$|G(x)|\not\equiv|x| \pmod 2$$

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-$\operatorname{lcm}(\ell,k(G))$-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 do 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 $\langle S \rangle$. If $\langle S \rangle$ equals the set of all permutations of $\{0,1\}^n$, for all $n \geq 1$, then we call S *universal*.

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 (2)-(5) above: in other words, if there exists an S such that $\mathcal{C} = \langle S \rangle$.

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.

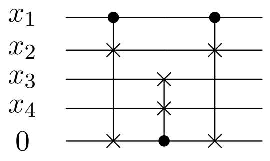

Figure 2: Generating a Controlled-Controlled-Swap gate from Fredkin

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 $k(S)$ 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 [4]. 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.[7]

[7]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.

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.

- NOT is the 1-bit gate that maps x to $\overline{x}$.
- NOTNOT, or NOT$^{\otimes 2}$, is the 2-bit gate that maps xy to $\overline{xy}$. 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 $x, y \oplus x\left(y \oplus z\right), z \oplus x\left(y \oplus z\right)$. 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 to $\overline{x}$ if $|x|$ is odd, or to x if $|x|$ is even. A different definition is
$$\mathrm{T}_k\left(x_1 \ldots x_k\right) = \left(x_1 \oplus b_x, \ldots, x_k \oplus b_x\right),$$
where $b_x := x_1 \oplus \cdots \oplus x_k$. This shows that T_k is linear. Indeed, we also have
$$\mathrm{T}_k\left(x\right) \cdot \mathrm{T}_k\left(y\right) \equiv x \cdot y + \left(k+2\right) b_x b_y \equiv x \cdot y \left(\operatorname{mod} 2\right),$$
which shows that T_k is orthogonal. Note also that, if $k \equiv 2 \left(\operatorname{mod} 4\right)$, then T_k preserves Hamming weight mod 4: if $|x|$ is even then $\left|\mathrm{T}_k\left(x\right)\right| = |x|$, while if $|x|$ is odd then
$$\left|\mathrm{T}_k\left(x\right)\right| \equiv k - |x| \equiv 2 - |x| \equiv |x| \left(\operatorname{mod} 4\right).$$
- 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
$$\mathrm{F}_k\left(x_1 \ldots x_k\right) = \overline{\mathrm{T}_k\left(x_1 \ldots x_k\right)} = \left(x_1 \oplus b_x \oplus 1, \ldots, x_k \oplus b_x \oplus 1\right)$$
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 $\left|\mathrm{F}_k\left(x\right)\right| = |x|$, while if $|x|$ is even then
$$\left|\mathrm{F}_k\left(x\right)\right| \equiv k - |x| \equiv |x| \left(\operatorname{mod} 4\right).$$
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.[8]

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.[9] 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 Appendix 13, we will prove that if we allow loose generation, then the only change to Theorem 3 is that every $\mathcal{C}+$ NOTNOT class collapses with the corresponding $\mathcal{C}+$ NOT class.

[8]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.

[9]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.

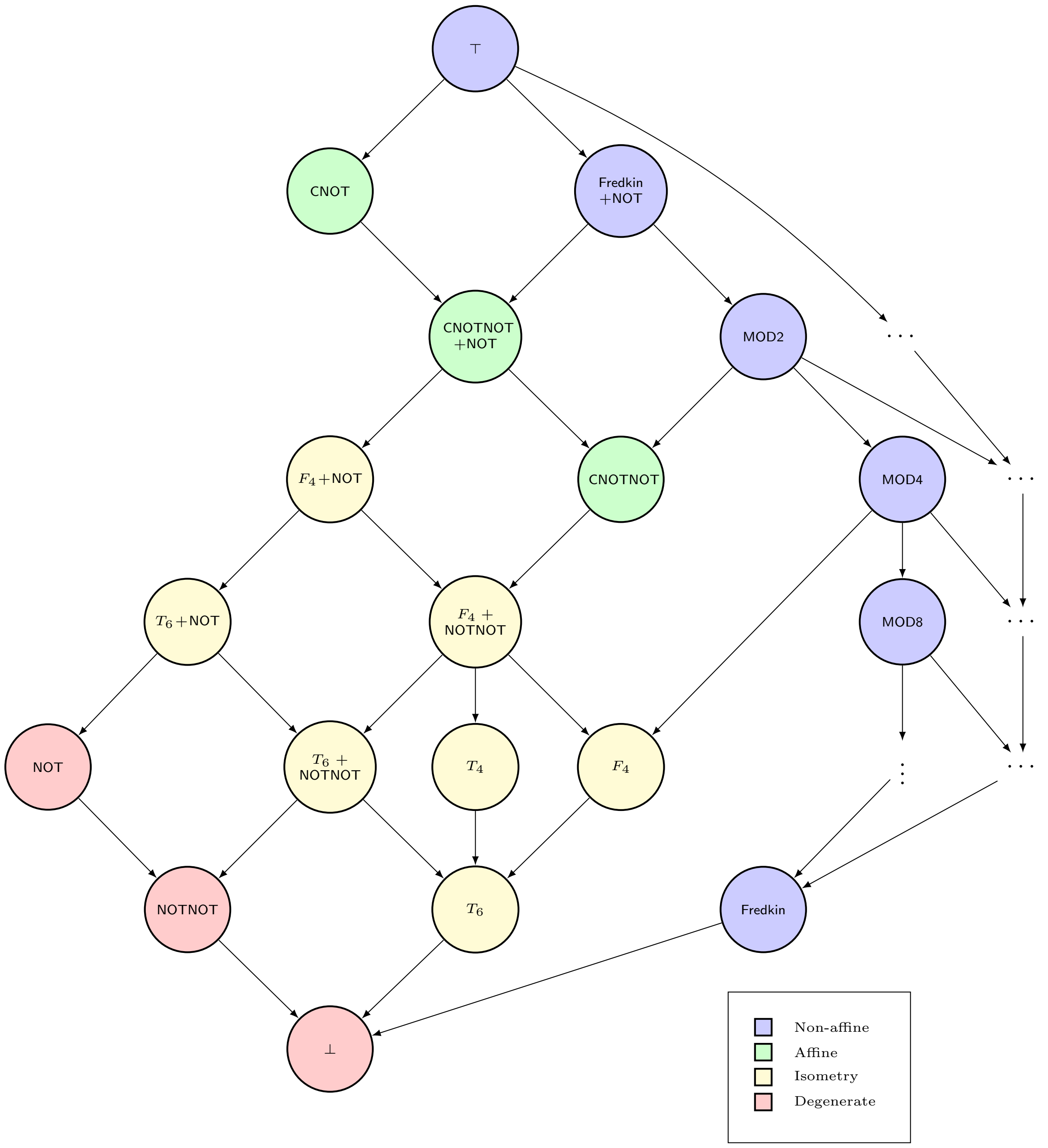

Figure 3: The inclusion lattice of reversible gate classes

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 $\mathcal{C} = \langle G, \text{NOT} \rangle$ or $\mathcal{C} = \langle G, \text{NOTNOT} \rangle$, 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, $G \otimes \text{NOT}$ or $G \otimes \text{NOTNOT}$. ■

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 $F^*(x_1 \ldots x_n) := \overline{F(\overline{x_1 \ldots x_n})}$. The dual can be thought of as F with the roles of 0 and 1 interchanged: for example, $\text{Toffoli}^*(xyz)$ 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 [12], 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 $\operatorname{poly}(n)$, which is negligible compared to the input size of $n2^n$. $\blacksquare$

We have implemented the algorithm described in Theorem 7, and Java code is available for download [24].

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 $2^n \operatorname{poly}(n)$ gates and $O(1)$ ancilla bits.*[10]

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 [25, 6], we will take care in Section 7 to ensure that each non-affine circuit construction uses at most $2^n \operatorname{poly}(n)$ 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).

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. $\blacksquare$

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

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 [28]) *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 $\operatorname{polylog} T$ 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;[11] 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.*

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

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

[11] This was proven by Lloyd [19], as well as by Kerntopf et al. [13] and De Vos and Storme [29]; we include a proof for completeness in Section 8.2.

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 $\langle\mathrm{CNOTNOT}\rangle$, 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

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

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

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

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 $\alpha(x_1)\ldots\alpha(x_n)$, 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.[12]

[12]The upshot of the Galois connection for clones [12] 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 begs 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.

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, and occupies the majority of the paper. The organization 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 Section 8, we complete the classification of all non-affine gate sets. In Section 8.1, we show that the only classes that contain a Fredkin gate are $\langle \text{Fredkin} \rangle$ itself, $\langle \text{Fredkin}, \text{NOTNOT} \rangle$, $\langle \text{Fredkin}, \text{NOT} \rangle$, $\langle \text{C}_k \rangle$ for $k \geq 3$, and $\langle \text{Toffoli} \rangle$. Next, in Section 8.3, we show that every nontrivial conservative gate generates Fredkin. Then, in Section 8.4, we build on the result of Section 8.4 to show that every non-affine gate set generates Fredkin.
- In Section 9, we complete the classification of all affine gate sets. For simplicity, we start with *linear* gate sets only. In Section 9.1, we 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, in Section 9.2, we show that every non-orthogonal linear gate generates CNOTNOT. Then, in Section 9.3, we show that every non-parity-preserving linear gate generates CNOT. Since CNOT generates all linear transformations, completes the classification of linear gate sets. Finally, in Section 9.4, we "put back the affine part," showing that it can lead to only 8 additional classes besides the linear classes $\langle \varnothing \rangle$, $\langle \text{T}_6 \rangle$, $\langle \text{T}_4 \rangle$, $\langle \text{CNOTNOT} \rangle$, 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.
- $\langle \text{F}_4, \text{NOT} \rangle = \langle \text{T}_4, \text{NOT} \rangle$ and $\langle \text{F}_4, \text{NOTNOT} \rangle = \langle \text{T}_4, \text{NOTNOT} \rangle$ are orthogonal in their linear part (isometries).
- $\langle \text{T}_6, \text{NOT} \rangle$ and $\langle \text{T}_6, \text{NOTNOT} \rangle$ are orthogonal and mod-4-preserving in their linear part.

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 $a_1 \ldots a_k$, 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 $|\mathrm{NOT}(x)| \equiv |x| + 1 \pmod 2$ 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

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

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

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

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

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

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 $k = 2$. ■

In Appendix 15, we provide an alternative proof of Theorem 12, using linear algebra. The alternative proof is longer, but perhaps less mysterious.

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:

$$z^{G(x)\cdot G(y)} \equiv z^{x \cdot y} \left(\operatorname{mod}\left(z^k - 1\right)\right)$$

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:

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

The summation on the right simplifies as follows.

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

Similarly,

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

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

$$(1+z)^{|G(x)|}\, 2^{n-|G(x)|} \equiv (1+z)^{|x|}\, 2^{n-|x|} \left(\operatorname{mod}\left(z^k - 1\right)\right),$$
$$0 \equiv (1+z)^{|x|} 2^{n-|G(x)|} \left(2^{|G(x)|-|x|} - (1+z)^{|G(x)|-|x|}\right) \left(\operatorname{mod}\left(z^k - 1\right)\right),$$

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

$$G(x \oplus y) \equiv G(x) \oplus G(y) \pmod{2}.$$

■

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

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

■

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

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

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

In Appendix 15, we provide an alternative proof of Theorem 19, one that does not rely on Theorem 13.

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.

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 [28, 25]) Toffoli *generates all reversible transformations on n bits, using only 2 ancilla bits.*[13]

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.

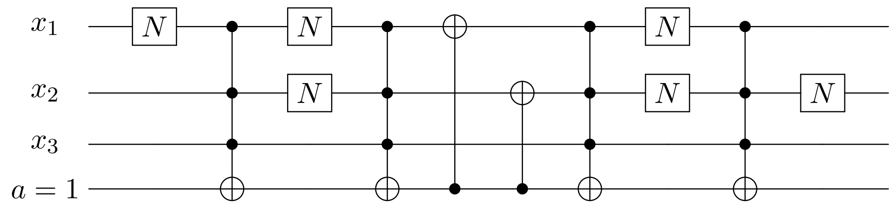

Figure 4: Generating the transposition $\sigma_{011,101}$

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.

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

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

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 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.[14]

Carefully inspecting the above proof shows that $O\left(n^2 2^n\right)$ gates and 3 ancilla bits suffice to generate any transformation. Notice the main reason we need two of the three ancillas is to apply the NOT gate while the ancilla a is active. Case analysis shows that any circuit constructible from NOT, CNOT, and Toffoli is equivalent to a circuit of NOT gates followed by a circuit of CNOT and Toffoli gates. For example, see Figure 5. This at most triples the size of the circuit. Therefore, we can construct a circuit that uses only two ancilla bits: apply the recursive construction, push the NOT gates to the front, and use two ancilla bits to generate the NOT gates. The recursive construction itself uses one ancilla bit, plus one more to implement CNOT. ■

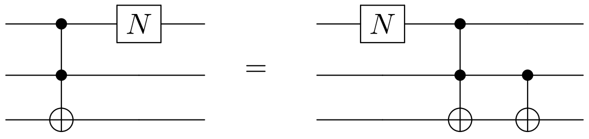

Figure 5: Example of equivalent Toffoli circuit with NOT gates pushed to the front

The particular construction above was inspired by a result of Ben-Or and Cleve [6], in which they compute algebraic formulas in a straight-line computation model using a constant number of registers. We note that Toffoli [28] proved a version of Theorem 23, but with $O(n)$ ancilla bits rather than $O(1)$. More recently, Shende et al. [25] gave a slightly more complicated construction which uses only 1 ancilla bit, and 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.

[14]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 $T(n) = O\left(n^2\right)$.

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 $y = y_1 \dots y_n$ and $z = z_1 \dots z_n$ 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 three dual-rail registers) to $a\overline{a} = 01$.
2. Apply y-CSWAP $(x_1 \dots x_n, a, \overline{a})$.
3. Apply z-CSWAP $(x_1 \dots x_n, a, \overline{a})$.
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 $(x_1 \dots x_n, a, \overline{a})$.
6. Apply y-CSWAP $(x_1 \dots x_n, a, \overline{a})$.

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 $(x_1, \dots, x_n, y, z)$ as follows:

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

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 $(x_1 \dots x_n, y, z)$, 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 6 holds (i.e., we can remove the NOT gates).

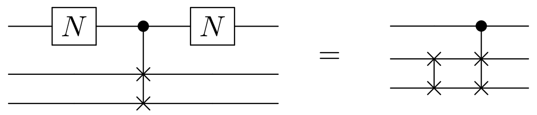
Figure 6: Removing NOT gates from the Fredkin circuit

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.

In [10], 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 $x = 0$.[15]
3. Apply C_k to $y_1 \ldots y_k$.
4. Repeat step 2.

■

Then we have the following.

Theorem 27 Fredkin $+ \mathrm{CC}_k$ *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 $+ \mathrm{CC}_1 =$ 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.

[15] 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.

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:

$$\begin{aligned} &\mathrm{C}_k\left(x, y, a_1 \ldots a_{k-2}\right), \\ &\mathrm{C}_k\left(x, z, a_1 \ldots a_{k-2}\right), \\ &\mathrm{C}_k\left(x, y, a_1 \ldots a_{k-2}\right). \end{aligned}$$

■

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 on* n *bits, using only* 6 *ancilla bits.*

Proof. Let F be any parity-flipping transformation on n bits. Then $F \otimes \mathrm{NOT}$ is an $(n+1)$-bit parity-preserving transformation. So by Theorem 27, we can implement $F \otimes \mathrm{NOT}$ using Fredkin + CC_2 (and we have CC_2 by Proposition 30). We can then apply a NOT gate to the $(n+1)^{st}$ bit to get F alone. ■

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 $x = x_1 \ldots x_n$, 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 $b = b_1 \ldots b_n$ 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 $F(x) = y_1 \ldots y_n$ to $x = x_1 \ldots x_n$. 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 The Non-Affine Part

Our goal, in this section, is to prove that there are no non-affine classes besides the ones listed in Theorem 3: namely, the conservative transformations, the parity-respecting transformations, the mod-k-preserving transformations for $k \geq 2$, and all transformations.

We will divide our analysis into two parts. We first show, in Section 8.1, that once a Fredkin gate is available, matters become fairly simple. At that point, the only possibilities are $\langle \mathrm{Fredkin} \rangle$, $\langle \mathrm{Fredkin}, \mathrm{NOTNOT} \rangle$, $\langle \mathrm{Fredkin}, \mathrm{NOT} \rangle$, $\langle \mathrm{C}_k \rangle$ for $k \geq 3$, and $\langle \mathrm{Toffoli} \rangle$. Then, in Sections 8.3 and 8.4, we prove the harder result that *every non-affine gate generates* Fredkin. This, in turn, is broken into three pieces:

- In Section 8.2, we reprove a result of Lloyd [19], showing that every non-affine gate is capable of universal computation with garbage.
- In Section 8.3, we show that every nontrivial conservative gate generates Fredkin (using the result of Section 8.2 as one ingredient).
- In Section 8.4, we build on the result of Section 8.3, to show that every non-affine gate generates Fredkin. This requires our first use of lattices, and also draws on some of the results about inner products and modularity obstructions from Section 6.

Summarizing the results of this section, we will obtain the following.

Theorem 39 *Every non-affine gate set generates one of the following classes:* $\langle \text{Fredkin} \rangle$*,* $\langle \text{C}_k \rangle$ *for some* $k \geq 3$*,* $\langle \text{Fredkin}, \text{NOTNOT} \rangle$*,* $\langle \text{Fredkin}, \text{NOT} \rangle$*, or* $\langle \text{Toffoli} \rangle$*.*

8.1 Above Fredkin

Our goal, in this section, is to classify all reversible gate classes containing Fredkin. We already know from Theorem 25 that Fredkin generates all conservative transformations. We will prove a substantial generalization of that result. First, however, we need a proposition that will also be used later in the paper. Given a reversible transformation G, let

$$W(G) := \{|G(x)| - |x| : x \in \{0,1\}^n\}$$

be the set of possible changes that G can cause to the Hamming weight of its input.

Proposition 40 *Let* G *be any non-conservative gate. Then for all integers* q*, there exists a* t *such that* $q \cdot k(G) \in W(G^{\otimes t})$*.*

Proof. Let γ be the gcd of the elements in $W(G)$. Then clearly G is mod-γ-respecting. By Proposition 1, this means that γ must divide $k(G)$.[16]

Now by reversibility, $W(G)$ has both positive and negative elements. But this means that we can find any integer multiple of γ in *some* set of the form

$$W(G^{\otimes t}) = \{w_1 + \cdots + w_m : w_1, \ldots, w_m \in W(G)\}.$$

Therefore we can find any integer multiple of $k(G)$ in some $W(G^{\otimes t})$ as well. ■

We can now characterize all reversible gate sets that contain Fredkin.

Theorem 41 *Let* G *be any gate. Then* Fredkin $+G$ *generates all mod-*$k(G)$*-preserving transformations (including in the cases* $k(G) = 1$*, in which case* Fredkin $+G$ *generates all transformations, and* $k(G) = \infty$*, in which case* Fredkin $+G$ *generates all conservative transformations).*

Proof. Let $k = k(G)$. If $k = \infty$ then we are done by Theorem 25, so assume k is finite. We will assume without loss of generality that G is mod-k-preserving. By Theorem 12, the only other possibility is that G is parity-flipping, but in that case we can simply repeat everything below with $G \otimes G$, which is parity-preserving and satisfies $k(G \otimes G) = 2$, rather than with G itself.

[16]Indeed, by using Theorem 12, one can show that $\gamma = k(G)$, except in the special case that G is parity-flipping, where we have $\gamma = 1$ and $k(G) = 2$.

By Theorem 27, it suffices to use Fredkin $+G$ to generate the CC_k gate. Let H be the gate $G \otimes G^{-1}$, followed by a swap of the two input registers. Observe that H^2 is the identity. Also, by Proposition 2,

$$k(H) = \gcd\left(k(G), k\left(G^{-1}\right)\right) = k.$$

So by Proposition 40, there exists a positive integer t, as well as inputs $y = y_1 \dots y_n$ and $z = z_1 \dots z_n$ such that $z = H^{\otimes t}(y)$ (and $y = H^{\otimes t}(z)$, since $\left(H^{\otimes t}\right)^2 = I$), and $|z| = |y| + k$.

We can assume without loss of generality that y has the form $0^a 1^b$—i.e., that its bits are in sorted order. We would like to sort the bits of z as well. Notice that, since $|z| > |y|$, there is some $i \in [n]$ such that $y_i = 0$ and $z_i = 1$. So we can easily design a circuit U of Fredkin gates, controlled by bit i, which reorders the bits of z so that

$$z' := U(z) = 0^{a-k} 1^{b+k}$$

whereas $U(y) = y$.

Observe that $H^{\otimes t}$ has a large number of fixed points: we have $H(u, G(u)) = (u, G(u))$ for any u; hence any string of the form $u_1, G(u_1), \dots, u_t, G(u_t)$ is a fixed point of $H^{\otimes t}$. Call one of these fixed points w, and let $w' := U(w)$.

We now consider a circuit R that applies U^{-1}, followed by $H^{\otimes t}$, followed by U. This R satisfies the following identities:

$$\begin{aligned}
R(y) &= U\left(H^{\otimes t}\left(U^{-1}(y)\right)\right) = U\left(H^{\otimes t}(y)\right) = U(z) = z'. \\
R(z') &= U\left(H^{\otimes t}\left(U^{-1}(z')\right)\right) = U\left(H^{\otimes t}(z)\right) = U(y) = y. \\
R(w') &= U\left(H^{\otimes t}\left(U^{-1}(w')\right)\right) = U\left(H^{\otimes t}(w)\right) = U(w) = w'.
\end{aligned}$$

Using R, we now construct $\mathrm{CC}_k(x_1 \dots x_k, c)$. Let A and B be two n-bit registers, initialized to $A := w'$ and $B := 0^{a-k} x_1 \dots x_k 1^b$. Also, let $q\overline{q}$ be two ancilla bits in dual-rail representation, initialized to $q\overline{q} = 01$. Then to apply CC_k, we do the following:

1. Swap q with $\overline{q}$ if and only if $x_1 = \cdots = x_k$ and $c = 1$.
2. Swap A with B if and only if $q = 1$.
3. Apply R to the A register.
4. Swap A with B if and only if $q = 1$.
5. Swap q with $\overline{q}$ if and only if $x_1 = \cdots = x_k$ and $c = 1$.

Here each conditional swap is implemented using Fredkin gates; recall from Theorem 25 that Fredkin generates every conservative transformation.

It is not hard to check that the above sequence maps $x_1 \dots x_k = 0^k$ to 1^k and $x_1 \dots x_k = 1^k$ to 0^k if $c = 1$, otherwise it maps the inputs to themselves. Furthermore, the ancilla bits are returned to their original states in all cases, since w' is a fixed point of R. Therefore we have implemented CC_k. ■

Theorem 41 has the following corollary.

Corollary 42 *Let S be any non-conservative gate set. Then* Fredkin $+S$ *generates one of the following classes:* $\langle \text{Fredkin}, \text{NOTNOT} \rangle$, $\langle \text{Fredkin}, \text{NOT} \rangle$, $\langle \mathrm{C}_k \rangle$ *for some* $k \geq 3$, *or* $\langle \text{Toffoli} \rangle$.

Proof. We know from Proposition 2 that S generates a single gate G such that $k(G) = k(S)$. If $k(S) \geq 3$, then Theorem 41 implies that Fredkin $+G$ generates all $k(S)$-preserving transformations, which equals $\left\langle \mathrm{C}_{k(S)} \right\rangle$ by Corollary 29. If $k(S) = 2$ and S is parity-preserving, then Theorem 41 implies that Fredkin $+G$ generates all parity-preserving transformations, which equals $\langle \text{Fredkin}, \text{NOTNOT} \rangle$ by Proposition 30. If $k(S) = 1$, then Theorem 41 implies that Fredkin $+G$ generates all transformations, which equals $\langle \text{Toffoli} \rangle$ by Theorem 23.

By Theorem 12, the one remaining case is that $k(S) = 2$ and some $G \in S$ is parity-flipping. By Theorem 41, certainly Fredkin $+G$ at least generates all parity-preserving transformations. Furthermore, let F be any parity-flipping transformation. Then $F \otimes G^{-1}$ is parity-preserving. So we can use Fredkin $+G$ to implement $F \otimes G^{-1}$, then compose with G itself to get F. Therefore we generate all parity-flipping transformations, which equals $\langle \text{Fredkin}, \text{NOT} \rangle$ by Theorem 31. ■

8.2 Computing with Garbage

For completeness, in this section we reprove some lemmas first shown by Seth Lloyd [19] in an unpublished 1992 technical report,[17] and later rediscovered by Kerntopf et al. [13] and De Vos and Storme [29]. We will use these lemmas to show the power of non-affine gates.

Recall the notion of *generating with garbage* from Section 2.3.

Lemma 43 ([19, 29]) *Every nontrivial reversible gate G generates* NOT *with garbage.*

Proof. Let $G(x_1 \ldots x_n) = y_1 \ldots y_n$ be nontrivial, and let $y_i = f_i(x_1 \ldots x_n)$. Then it suffices to show that at least one f_i is a non-monotone Boolean function. For if f_i is non-monotone, then by definition, there exist two inputs $x, x' \in \{0,1\}^n$, which are identical except that $x_j = 1$ and $x'_j = 0$ at some bit j, such that $f_i(x) = 0$ and $f_i(x') = 1$. But then, if we set the other $n-1$ bits consistent with x and x', we have $y_i = \text{NOT}(x_j)$.

Thus, suppose by contradiction that every f_i is monotone. Then reversibility clearly implies that $G(0^n) = 0^n$, and that the set of strings of Hamming weight 1 is mapped to itself: that is, there exists a permutation σ such that $G(e_j) = e_{\sigma(j)}$ for all j. Furthermore, by monotonicity, for all $j \neq k$ we have $G(e_j \oplus e_k) \geq e_{\sigma(j)} \oplus e_{\sigma(k)}$. But then reversibility implies that $G(e_j \oplus e_k)$ can only be $e_{\sigma(j)} \oplus e_{\sigma(k)}$ itself, and so on inductively, so that we obtain $G(x_1 \ldots x_n) = x_{\sigma^{-1}(1)} \ldots x_{\sigma^{-1}(n)}$ for all $x \in \{0,1\}^n$. But this means that G is trivial, contradiction. ■

Proposition 44 (folklore) *For all $n \geq 3$, every non-affine Boolean function on n bits has a non-affine subfunction on $n-1$ bits.*

Proof. Let $f : \{0,1\}^n \to \{0,1\}$ be non-affine, and let f_0 and f_1 be the $(n-1)$-bit subfunctions obtained by restricting f's first input bit to 0 or 1 respectively. If either f_0 or f_1 is itself non-affine, then we are done. Otherwise, we have $f_0(x) = (a_0 \cdot x) \oplus b_0$ and $f_1(x) = (a_1 \cdot x) \oplus b_1$, for some $a_0, a_1 \in \{0,1\}^{n-1}$ and $b_0, b_1 \in \{0,1\}$. Notice that f is non-affine if and only if $a_0 \neq a_1$. So there is some bit where a_0 and a_1 are unequal. If we now remove any of the *other* rightmost $n-1$ input bits (which must exist since $n-1 \geq 2$) from f, then we are left with a non-affine function on $n-1$ bits. ■

Lemma 45 ([19, 29]) *Every non-affine reversible gate G generates the* 2*-bit* AND *gate with garbage.*

[17]Prompted by the present work, Lloyd has recently posted his 1992 report to the arXiv.

Proof. Certainly every non-affine gate is nontrivial, so we know from Lemma 43 that G generates NOT with garbage. For this reason, it suffices to show that G can generate *some* non-affine 2-bit gate with garbage (since all such gates are equivalent to AND under negating inputs and outputs). Let $G(x_1 \ldots x_n) = y_1 \ldots y_n$, and let $y_i = f_i(x_1 \ldots x_n)$. Then some particular f_i must be a non-affine Boolean function. So it suffices to show that, by restricting $n-2$ of f_i's input bits, we can get a non-affine function on 2 bits. But this follows by inductively applying Proposition 44. ■

By using Lemma 45, it is possible to prove directly that the only classes that contain a CNOT gate are $\langle \text{CNOT} \rangle$ (i.e., all affine transformations) and $\langle \text{Toffoli} \rangle$ (i.e., all transformations)—or in other words, that if G is any non-affine gate, then $\langle \text{CNOT}, G \rangle = \langle \text{Toffoli} \rangle$. However, we will skip this result, since it is subsumed by our later results.

Recall that COPY is the 2-bit partial gate that maps 00 to 00 and 10 to 11.

Lemma 46 ([19, 13]) *Every non-degenerate reversible gate G generates* COPY *with garbage.*

Proof. Certainly every non-degenerate gate is nontrivial, so we know from Lemma 43 that G generates NOT with garbage. So it suffices to show that there is some pair of inputs $x, x' \in \{0,1\}^n$, which differ only at a single coordinate i, such that $G(x)$ and $G(x')$ have Hamming distance at least 2. For then if we set $x_i := z$, and regard the remaining $n-1$ coordinates of x as ancillas, we will find at least two copies of z or $\overline{z}$ in $G(x)$, which we can convert to at least two copies of z using NOT gates. Also, if all of the ancilla bits that receive a copy of z were initially 1, then we can use a NOT gate to reduce to the case where one of them was initially 0.

Thus, suppose by contradiction that $G(x)$ and $G(x')$ are neighbors on the Hamming cube whenever x and x' are neighbors. Then starting from 0^n and $G(0^n)$, we find that every $G(e_i)$ must be a neighbor of $G(0^n)$, every $G(e_i \oplus e_j)$ must be a neighbor of $G(e_i)$ and $G(e_j)$, and so on, so that G is just a rotation and reflection of $\{0,1\}^n$. But that means G is degenerate, contradiction. ■

8.3 Conservative Generates Fredkin

In this section, we prove the following theorem.

Theorem 47 *Let G be any nontrivial conservative gate. Then G generates* Fredkin.

The proof will be slightly more complicated than necessary, but we will then reuse parts of it in Section 8.4, when we show that every non-affine, *non*-conservative gate generates Fredkin.

Given a gate Q, let us call Q *strong quasi-Fredkin* if there exist control strings a, b, c, d such that

$$Q(a, 01) = (a, 01), \tag{3}$$
$$Q(b, 01) = (b, 10), \tag{4}$$
$$Q(c, 00) = (c, 00), \tag{5}$$
$$Q(d, 11) = (d, 11). \tag{6}$$

Lemma 48 *Let G be any nontrivial n-bit conservative gate. Then G generates a strong quasi-Fredkin gate.*

Proof. By conservativity, G maps unit vectors to unit vectors, say $G(e_i) = e_{\pi(i)}$ for some permutation π. But since G is nontrivial, there is some input $x \in \{0,1\}^n$ such that $x_i = 1$, but the corresponding bit $\pi(i)$ in $G(x)$ is 0. By conservativity, there must also be some bit j such that $x_j = 0$, but bit $\pi(j)$ of $G(x)$ is 1. Now permute the inputs to make bit j and bit i the last two bits, permute the outputs to make bits $\pi(j)$ and $\pi(i)$ the last two bits, and permute either inputs or outputs to make x match $G(x)$ on the first $n-2$ bits. After these permutations are performed, x has the form $w01$ for some $w \in \{0,1\}^{n-2}$. So

$$\begin{aligned} G\left(0^{n-2}, 01\right) &= \left(0^{n-2}, 01\right), \\ G(w, 01) &= (w, 10), \\ G\left(0^{n-2}, 00\right) &= \left(0^{n-2}, 00\right), \\ G\left(1^{n-2}, 11\right) &= \left(11^{n-2}, 11\right), \end{aligned}$$

where the last two lines again follow from conservativity. Hence G (after these permutations) satisfies the definition of a strong quasi-Fredkin gate. ■

Next, call a gate C a *catalyzer* if, for every $x \in \{0,1\}^{2n}$ with Hamming weight n, there exists a "program string" $p(x)$ such that

$$C\left(p(x), 0^n 1^n\right) = (p(x), x).$$

In other words, a catalyzer can be used to transform $0^n 1^n$ into any target string x of Hamming weight n. Here x can be encoded in any manner of our choice into the auxiliary program string $p(x)$, as long as $p(x)$ is left unchanged by the transformation. The catalyzer itself cannot depend on x.

Lemma 49 *Let Q be a strong quasi-Fredkin gate. Then Q generates a catalyzer.*

Proof. Let $z := 0^n 1^n$ be the string that we wish to transform. For all $i \in \{1, \ldots, n\}$ and $j \in \{n+1, \ldots, 2n\}$, let s_{ij} denote the operation that swaps the i^{th} and j^{th} bit of z. Then consider the following list of "candidate swaps":

$$s_{1,n+1}, \ldots, s_{1,2n}, \quad s_{2,n+1}, \ldots, s_{2,2n}, \quad \ldots, \quad s_{n,n+1}, \ldots, s_{n,2n}.$$

Suppose we go through the list in order from left to right, and for each swap in the list, get to choose whether to apply it or not. It is not hard to see that, by making these choices, we can map $0^n 1^n$ to any x such that $|x| = n$, by pairing off the first 0 bit that should be 1 with the first 1 bit that should be 0, the second 0 bit that should be 1 with the second 1 bit that should be 0, and so on, and choosing to swap those pairs of bits and not any other pairs.

Now, let the program string $p(x)$ be divided into n^2 registers $r_1, \ldots, r_{n^2}$, each of the same size. Suppose that, rather than applying (or not applying) the t^{th} swap s_{ij} in the list, we instead apply the gate F, with r_t as the control string, and z_i and z_j as the target bits. Then we claim that we can map z to x as well. If the t^{th} candidate swap is supposed to occur, then we set $r_t := b$. If the t^{th} candidate swap is not supposed to occur, then we set r_t to either a, c, or d, depending on whether $z_i z_j$ equals 01, 00, or 11 at step t of the swapping process. Note that, because we know x when designing $p(x)$, we know exactly what $z_i z_j$ is going to be at each time step. Also, $z_i z_j$ will never equal 10, because of the order in which we perform the swaps: we swap each 0 bit z_i that needs to be swapped with the first 1 bit z_j that we can. After we have performed the swap, $z_i = 1$ will then only be compared against other 1 bits, never against 0 bits. ■

Finally:

Lemma 50 *Let G be any non-affine gate, and let C be any catalyzer. Then $G + C$ generates* Fredkin.

Proof. We will actually show how to generate *any* conservative transformation $F : \{0,1\}^n \rightarrow \{0,1\}^n$.

Since G is non-affine, Lemmas 43, 45, and 46 together imply that we can use G to compute any Boolean function, albeit possibly with input-dependent garbage.

Let $x \in \{0,1\}^n$. Then by assumption, C maps $0^n 1^n$ to $F(x)\overline{F(x)}$ using the program string $p(F(x)\overline{F(x)})$. Now, starting with x and ancillas $0^n 1^n$, we can clearly use G to produce

$$x, \operatorname{gar}(x), p(F(x)\overline{F(x)}), 0^n 1^n,$$

for some garbage $\operatorname{gar}(x)$. We can then apply C to get

$$x, \operatorname{gar}(x), p(F(x)\overline{F(x)}), F(x), \overline{F(x)}.$$

Uncomputing $p(F(x)\overline{F(x)})$ yields

$$x, F(x), \overline{F(x)}.$$

Notice that since F is conservative, we have $\left|x, \overline{F(x)}\right| = n$. Therefore, there exists some program string $p(x, \overline{F(x)})$ that can be used as input to C^{-1} to map $x, \overline{F(x)}$ to $0^n 1^n$. Again, we can generate this program string using the fact that G is non-affine:

$$x, F(x), \overline{F(x)}, \operatorname{gar}(F(x)), p(x, \overline{F(x)}).$$

Applying C^{-1} and then uncomputing, we get

$$F(x), 0^n 1^n$$

which completes the proof. ■

By Lemma 18, every nontrivial conservative gate is also non-affine. Therefore, combining Lemmas 48, 49, and 50 completes the proof of Theorem 47, that every nontrivial conservative gate generates Fredkin.

8.4 Non-Conservative Generates Fredkin

Building on our work in Section 8.3, in this section we handle the *non*-conservative case, proving the following theorem.

Theorem 51 *Every non-affine, non-conservative gate generates* Fredkin.

Thus, let G be a non-affine, non-conservative gate. Starting from G, we will perform a sequence of transformations to produce gates that are "gradually closer" to Fredkin. Some of these transformations might look a bit mysterious, but they will culminate in a strong quasi-Fredkin gate, which we already know from Lemmas 49 and 50 is enough to generate a Fredkin gate (since G is also non-affine).

The first step is to create a non-affine gate with two particular inputs as fixed points.

Lemma 52 *Let G be any non-affine gate on n bits. Then G generates a non-affine gate H on n^2 bits that acts as the identity on the inputs 0^{n^2} and 1^{n^2}.*

Proof. We construct H as follows:

1. Apply $G^{\otimes n}$ to n^2 input bits. Let G_i be the i^{th} gate in this tensor product.
2. For all $i \in [n-1]$, swap the i^{th} output bit of G_i with the i^{th} output bit of G_n.
3. Apply $\left(G^{-1}\right)^{\otimes n}$.

It is easy to see that H maps 0^{n^2} to 0^{n^2} and 1^{n^2} to 1^{n^2}. (Indeed, H maps *every* input that consists of an n-bit string repeated n times to itself.) To see that H is also non-affine, first notice that G^{-1} is non-affine. But we can cause any input $x = x_1 \ldots x_n$ that we like to be fed into the final copy of G^{-1}, by encoding that input "diagonally," with each G_i producing x_i as its i^{th} output bit. Therefore H is non-affine. ■

As a remark, with all the later transformations we perform, we will want to maintain the property that the all-0 and all-1 inputs are fixed points. Fortunately, this will not be hard to arrange.

Let H be the output of Lemma 52. If H is conservative (i.e., $k(H) = \infty$), then H already generates Fredkin by Theorem 47, so we are done. Thus, we will assume in what follows that $k(H)$ is finite. We will further assume that H is mod-$k(H)$-preserving. By Theorem 12, the only gates H that are *not* mod-$k(H)$-preserving are the parity-flipping gates—but if H is parity-flipping, then $H \otimes H$ is parity-preserving, and we can simply repeat the whole construction with $H \otimes H$ in place of H.

Now we want to show that we can use H to decrease the inner product between a pair of inputs by exactly 1 mod m, for any m we like.

Lemma 53 *Let H be any non-conservative, nonlinear gate. Then for all $m \geq 2$, there is a positive integer t, and inputs x, y, such that*

$$H^{\otimes t}(x) \cdot H^{\otimes t}(y) - x \cdot y \equiv -1 \pmod{m}.$$

Proof. Let $m = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_s^{\alpha_s}$ where each p_i is a distinct prime. By Corollary 15, we know that for each p_i, there is some pair of inputs x_i, y_i such that

$$H(x_i) \cdot H(y_i) \not\equiv x_i \cdot y_i \pmod{p_i}.$$

In other words, letting

$$\gamma_i := H(x_i) \cdot H(y_i) - x_i \cdot y_i,$$

we have $\gamma_i \not\equiv 0 \pmod{p_i}$ for all $i \in \{1, \ldots, s\}$. Our goal is to find an (x, y) such that

$$H^{\otimes t}(x) \cdot H^{\otimes t}(y) - x \cdot y \equiv -1 \pmod{m}.$$

To do so, it suffices to find nonnegative integers $d_1, \ldots, d_s$ that solve the equation

$$\sum_{i=1}^{s} d_i \gamma_i \equiv -1 \pmod{m}. \qquad (7)$$

Here d_i represents the number of times the pair (x_i, y_i) occurs in (x, y). By construction, no p_i divides γ_i, and since the p_i's are distinct primes, they have no common factor. This implies that $\gcd(\gamma_1, \ldots, \gamma_s, m) = 1$. So by the Chinese Remainder Theorem, a solution to (7) exists. ∎

Note also that, if H maps the all-0 and all-1 strings to themselves, then $H^{\otimes t}$ does so as well.

To proceed further, it will be helpful to introduce some terminology. Suppose that we have two strings $x = x_1 \ldots x_n$ and $y = y_1 \ldots y_n$. For each i, the pair $x_i y_i$ has one of four possible values: 00, 01, 10, or 11. Let the *type* of (x, y) be an ordered triple $(a, b, c) \in \mathbb{Z}^3$, which simply records the number of occurrences in (x, y) of each of the three pairs 01, 10, and 11. (It will be convenient not to keep track of 00 pairs, since they don't contribute to the Hamming weight of either x or y.) Clearly, by applying swaps, we can convert between any pairs (x, y) and (x', y') of the same type, provided that x, y, x', y' all have the same length n.

Now suppose that, by repeatedly applying a gate H, we can convert some input pair (x, y) of type (a, b, c) into some pair (x', y') of type (a', b', c'). Then we say that H *generates the slope*

$$\left(a' - a, b' - b, c' - c\right).$$

Note that, if H generates the slope (p, q, r), then by inverting the transformation, we can also generate the slope $(-p, -q, -r)$. Also, if H generates the slope (p, q, r) by acting on the input pair (x, y), and the slope (p', q', r') by acting on (x', y'), then it generates the slope $(p + p', q + q', r + r')$ by acting on (xx', yy'). For these reasons, the achievable slopes form a 3-dimensional *lattice*—that is, a subset of $\mathbb{Z}^3$ closed under integer linear combinations—which we can denote $\mathcal{L}(H)$.

What we really want is for the lattice $\mathcal{L}(H)$ to contain a particular point: $(1, 1, -1)$. Once we have shown this, we will be well on our way to generating a strong quasi-Fredkin gate. We first need a general fact about slopes.

Lemma 54 *Let H map the all-0 input to itself. Then $\mathcal{L}(H)$ contains the points $(k(H), 0, 0)$, $(0, k(H), 0)$, and $(0, 0, k(H))$.*

Proof. Recall from Proposition 40 that there exists a t, and an input w, such that $\left|H^{\otimes t}(w)\right| = |w| + k(H)$. Thus, to generate the slope $(k(H), 0, 0)$, we simply need to do the following:

- Choose an input pair (x, y) with sufficiently many $x_i y_i$ pairs of the forms 10 and 00.
- Apply $H^{\otimes t}$ to a subset of bits on which x equals w, and y equals the all-0 string.

Doing this will increase the number of 10 pairs by $k(H)$, while not affecting the number of 01 or 11 pairs.

To generate the slope $(0, k(H), 0)$, we do exactly the same thing, except that we reverse the roles of x and y.

Finally, to generate the slope $(0, 0, k(H))$, we choose an input pair (x, y) with sufficiently many $x_i y_i$ pairs of the forms 11 and 00, and then use the same procedure to increase the number of 11 pairs by $k(H)$. ∎

We can now prove that $(1, 1, -1)$ is indeed in our lattice.

Lemma 55 *Let H be a mod-$k(H)$-preserving gate that maps the all-0 input to itself, and suppose there exist inputs x, y such that*

$$H(x) \cdot H(y) - x \cdot y \equiv -1 \pmod{k(H)}.$$

Then $(1, 1, -1) \in \mathcal{L}(H)$.

Proof. The assumption implies directly that H generates a slope of the form $(p, q, -1 + rk(H))$, for some integers p, q, r. Thus, Lemma 54 implies that H also generates a slope of the form $(p, q, -1)$, via some gate $G \in \langle H \rangle$ acting on inputs (x, y). Now, since H is mod-$k(H)$-preserving, we have $|G(x)| \equiv |x| \pmod{k(H)}$ and $|G(y)| \equiv |y| \pmod{k(H)}$. But this implies that $p \equiv 1 \pmod{k(H)}$ and $q \equiv 1 \pmod{k(H)}$. So, again using Lemma 54, we can generate the slope $(1, 1, -1)$. ■

Combining Lemmas 52, 53, and 55, we can summarize our progress so far as follows.

Corollary 56 *Let G be any non-affine, non-conservative gate. Then either G generates* Fredkin*, or else it generates a gate H that maps the all-*0 *and all-*1 *inputs to themselves, and that also satisfies* $(1, 1, -1) \in \mathcal{L}(H)$.

We now explain the importance of the lattice point $(1, 1, -1)$. Given a gate Q, let us call Q *weak quasi-Fredkin* if there exist strings a and b such that

$$Q(a, 01) = (a, 01),$$
$$Q(b, 01) = (b, 10).$$

Then:

Lemma 57 *A gate H generates a weak quasi-Fredkin gate if and only if* $(1, 1, -1) \in \mathcal{L}(H)$.

Proof. If H generates a weak quasi-Fredkin gate Q, then applying Q to the input pair $(a, 01)$ and $(b, 01)$ directly generates the slope $(1, 1, -1)$. For the converse direction, if H generates the slope $(1, 1, -1)$, then by definition there exists a gate $Q \in \langle H \rangle$, and inputs x, y, such that $|Q(x)| = |x|$ and $|Q(y)| = |y|$, while

$$Q(x) \cdot Q(y) = x \cdot y - 1.$$

In other words, applying Q decreases by one the number of 1 bits on which x and y agree, while leaving their Hamming weights the same. But in that case, by permuting input and output bits, we can easily put Q into the form of a weak quasi-Fredkin gate. ■

Next, recall the definition of a strong quasi-Fredkin gate from Section 8.3. Then combining Corollary 56 with Lemma 57, we have the following.

Corollary 58 *Let G be any non-affine, non-conservative gate. Then either G generates* Fredkin*, or else it generates a strong quasi-Fredkin gate.*

Proof. Combining Corollary 56 with Lemma 57, we find that either G generates Fredkin, or else it generates a weak quasi-Fredkin gate that maps the all-0 and all-1 strings to themselves. But such a gate *is* a strong quasi-Fredkin gate, since we can let c be the all-0 string and d be the all-1 string. ■

Combining Corollary 58 with Lemmas 49 and 50 now completes the proof of Theorem 51: that every non-affine, non-conservative gate generates Fredkin. However, since every non-affine, conservative gate generates Fredkin by Theorem 47, we get the following even broader corollary.

Corollary 59 *Every non-affine gate generates* Fredkin.

Finally, combined with Corollary 42, Corollary 59 completes the proof of Theorem 39, that every non-affine gate set generates either $\langle \text{Fredkin} \rangle$, $\langle \text{Fredkin}, \text{NOTNOT} \rangle$, $\langle \text{Fredkin}, \text{NOT} \rangle$, $\langle \text{C}_k \rangle$ for some $k \geq 3$, or $\langle \text{Toffoli} \rangle$.

9 The Affine Part

Having completed the classification of the non-affine classes, in this section we turn our attention to proving that there are no affine classes besides the ones listed in Theorem 3: namely, the trivial, T_6, T_4, F_4, CNOTNOT, and CNOT classes, as well as various extensions of them by NOTNOT and NOT gates.

To make the problem manageable, we start by restricting attention to the linear parts of affine transformations (i.e., if a transformation has the form $G(x) = Ax \oplus b$, we ignore the additive constant b). We show that the only possibilities for the linear part are: the identity, all mod-4-preserving orthogonal transformations, all orthogonal transformations, all parity-preserving linear transformations, or all linear transformations. This result, in turn, is broken into several pieces:

- In Section 9.1, we show that any mod-4-preserving orthogonal gate generates *all* mod-4-preserving orthogonal transformations, and that any non-mod-4-preserving orthogonal gate generates all orthogonal transformations.
- In Section 9.2, we show that every non-orthogonal, parity-preserving linear gate generates CNOTNOT. This again requires "slope theory" and the analysis of a 3-dimensional lattice. It also draws on the results of Section 6.3, which tell us that it suffices to restrict attention to the case $k(G) = 2$.
- In Section 9.3, we show that every non-parity-preserving linear gate generates CNOT. In this case we are lucky that we only need to analyze a 1-dimensional lattice (i.e., an ideal in $\mathbb{Z}$)

Finally, in Section 9.4, we complete the classification by showing that including the affine parts can yield only the following additional possibilities: NOTNOT, NOT, F_4, $\mathrm{F}_4 +$ NOTNOT, $\mathrm{F}_4 +$ NOT, $\mathrm{T}_6 +$ NOTNOT, $\mathrm{T}_6 +$ NOT, or CNOTNOT + NOT. Summarizing, the results of this section will imply the following.

Theorem 60 *Any set of affine gates generates one of the following* 13 *classes:* $\langle\varnothing\rangle$, $\langle\mathrm{NOTNOT}\rangle$, $\langle\mathrm{NOT}\rangle$, $\langle\mathrm{T}_6\rangle$, $\langle\mathrm{T}_6,\mathrm{NOTNOT}\rangle$, $\langle\mathrm{T}_6,\mathrm{NOT}\rangle$, $\langle\mathrm{T}_4\rangle$, $\langle\mathrm{F}_4\rangle$, $\langle\mathrm{T}_4,\mathrm{NOTNOT}\rangle$, $\langle\mathrm{T}_4,\mathrm{NOT}\rangle$, $\langle\mathrm{CNOTNOT}\rangle$, $\langle\mathrm{CNOTNOT},\mathrm{NOT}\rangle$*, or* $\langle\mathrm{CNOT}\rangle$*.*

Together with Theorem 39, this will then complete the proof of Theorem 3.

9.1 The T and F Swamplands

In this section, we wish to characterize the orthogonal classes. We first need a lemma.

Lemma 61 T_{4k+2} *generates* T_6*, and* T_{4k} *generates* T_4*, for all* $k \geq 1$*.*

Proof. We first describe how to simulate $\mathrm{T}_6(x_1 \ldots x_6)$, using three applications of T_{4k+2}. Let $b_x := x_1 \oplus \cdots \oplus x_6$. Also, let a be a string of ancilla bits, initialized to 0^{2k-2}. Then:

1. Apply T_{4k+2} to the string $0^{4k-4}x_1 \ldots x_6$. This yields $b_x^{4k-4}, x_1 \oplus b_x, \ldots, x_6 \oplus b_x$.

2. Swap out $2k-2$ of the b_x bits with the ancilla string $a = 0^{2k-2}$, and apply T_{4k+2} again. This yields
$$\mathrm{T}_{4k+2}\left(0^{2k-2}, b_x^{2k-2}, x_1 \oplus b_x, \ldots, x_6 \oplus b_x\right) = \left(b_x^{2k-2}, 0^{2k-2}, x_1 \ldots x_6\right),$$
since the number of 'b_x' entries is even.

3. Swap the $2k-2$ bits that are now 0 with $a = b_x^{2k-2}$, and apply T_{4k+2} a third time. This returns a to 0^{2k-2}, and yields
$$\mathrm{T}_{4k+2}\left(b_x^{4k-4}, x_1 \ldots x_6\right) = \mathrm{T}_{4k+2}\left(0^{4k-4}, x_1 \oplus b_x, \ldots x_6 \oplus b_x\right).$$

Thus, we have successfully applied T_6 to $x_1 \ldots x_6$. The same sequence of steps can be used to simulate $\mathrm{T}_4\left(x_1 \ldots x_4\right)$ using three applications of T_{4k}. ■

We can now show that there is only one nontrivial orthogonal class that is also mod-4-preserving: namely, $\langle \mathrm{T}_6 \rangle$.

Theorem 62 *Any nontrivial mod-4-preserving linear gate G generates* T_6.

Proof. Let $G(x) = Ax$, for some $A \in \mathbb{F}_2^{n \times n}$. Then recall from Corollary 21 that A is orthogonal. By Lemma 16, this implies that $A^{-1} = A^T$, so G can also generate A^T.

Let B be the $(n+1) \times (n+1)$ matrix that acts as the identity on the first bit, and as A on bits 2 through $n+1$. Observe that B^T acts as the identity on the first bit, and as A^T on bits 2 through $n+1$. Also, since A preserves Hamming weight mod 4, so do A^T, B, and B^T. By Corollary 22, this implies that each of $\left(B^T\right)$'s column vectors must have Hamming weight 1 mod 4. Furthermore, since A is nontrivial, there must be some column of B^T with Hamming weight $4k+1$, for some $k \geq 1$. Then by swapping rows and columns, we can get B^T into the form
$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & & -v_1- & \\ \vdots & \vdots & & \vdots & \\ 0 & 1 & & -v_{4k+1}- & \\ 0 & 0 & & -v_{4k+2}- & \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & & -v_n- & \end{pmatrix},$$
where $v_1, \ldots, v_n$ are row vectors each of length $n-1$. Let δ_{ij} equal 1 if $i = j$ or 0 otherwise. Then note that by orthogonality,
$$v_i \cdot v_j = \begin{cases} \overline{\delta_{ij}} & \text{if } i, j \leq 4k+1, \\ \delta_{ij} & \text{otherwise.} \end{cases}$$

Now let C^T be the matrix obtained by swapping the first two columns of B^T. Then we claim that $C^T B$ yields a T_{4k+2} transformation. Since T_{4k+2} generates T_6 by Lemma 61, we will be done after we have shown this.

We have

$$C^T B = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & & —v_1— & \\ \vdots & \vdots & & \vdots & \\ 1 & 0 & & —v_{4k+1}— & \\ 0 & 0 & & —v_{4k+2}— & \\ \vdots & \vdots & & \vdots & \\ 0 & 0 & & —v_n— & \end{pmatrix} \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & | & & | & | & & | \\ \vdots & v_1^T & \cdots & v_{4k+1}^T & v_{4k+2}^T & \cdots & v_n^T \\ 0 & | & & | & | & & | \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & \ddots & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

One can check that the above transformation is actually T_{4k+2} on the first $4k+2$ bits, and the identity on the rest. ■

Likewise, there is only one orthogonal class that is *not* mod-4-preserving: namely, $\langle \mathrm{T}_4 \rangle$.

Theorem 63 *Let G be any nontrivial orthogonal gate that does not preserve Hamming weight mod 4. Then G generates* T_4.

Proof. We use essentially the same construction as in Theorem 62. The only change is that Corollary 22 now tells us that there must be a column of B^T with Hamming weight $4k+3$ for some $k \geq 1$, so we use that in place of the column with Hamming weight $4k+1$. This leads to an $(n+1) \times (n+1)$ matrix $C^T B$, which acts as T_{4k+4} on the first $4k+4$ bits and as the identity on the rest. But T_{4k+4} generates T_4 by Lemma 61, so we are done. ■

9.2 Non-Orthogonal Linear Generates CNOTNOT

In classifying all linear gate sets, our next goal is to show that "there is nothing between orthogonal and parity-preserving." In other words:

Theorem 64 *Let G be any non-orthogonal, parity-preserving linear gate. Then G generates* CNOTNOT *(or equivalently, all parity-preserving linear transformations).*

The main idea of the proof is as follows. Let CPD, or *Copying with a Parity Dumpster*, be the following partial reversible gate:

$$\begin{aligned} \mathrm{CPD}\,(000) &= 000, \\ \mathrm{CPD}\,(001) &= 001, \\ \mathrm{CPD}\,(100) &= 111, \\ \mathrm{CPD}\,(101) &= 110. \end{aligned}$$

In other words, CPD maps $x0y$ to $x, x, x \oplus y$—copying x, but also XORing x into the y "dumpster" in order to preserve the total parity. Notice that CPD is consistent with CNOTNOT; indeed, it is simply the restriction of CNOTNOT to inputs whose second bit is 0. Notice also that, whenever we have a 3-bit string of the form xxy, we can apply CPD in reverse to get $x, 0, x \oplus y$.

Then we will first observe that CPD generates CNOTNOT. We will then apply the theory of types and slopes, which already made an appearance in Section 8.4, to show that any non-orthogonal linear gate generates CPD: in essence, that there are no modularity or other obstructions to generating it.

Lemma 65 *Let G be any gate that generates* CPD*. Then G generates* CNOTNOT *(or equivalently, all parity-preserving linear transformations).*

Proof. Let $F : \{0,1\}^n \to \{0,1\}^n$ be any reversible, parity-preserving linear transformation. Then we can generate the following sequence of states:

$$\begin{aligned}
x &\to x, \operatorname{gar}(x), F(x) \\
&\to x, \operatorname{gar}(x), F(x), F(x), |x| \,(\operatorname{mod} 2) \\
&\to x, F(x), |x| \,(\operatorname{mod} 2) \\
&\to x, F(x), \operatorname{gar}(F(x)), x, |x| \,(\operatorname{mod} 2) \\
&\to F(x), \operatorname{gar}(F(x)), x, |x| + |F(x)| \,(\operatorname{mod} 2) \\
&= F(x), \operatorname{gar}(F(x)), x, 0 \,(\operatorname{mod} 2) \\
&\to F(x),
\end{aligned}$$

for some garbage strings $\operatorname{gar}(x)$ and $\operatorname{gar}(F(x))$. Here the first line computes $F(x)$ from x; the second line applies CPD to copy $F(x)$ (using a single "dumpster" bit for each bit of $F(x)$); the third line uncomputes $F(x)$; the fourth line computes a second copy of x from $F(x)$; the fifth line applies CPD in reverse to erase one of the copies of x (reusing same dumpster bit from before); and the sixth line uncomputes x. Also, $|x| + |F(x)| \equiv 0 \,(\operatorname{mod} 2)$ follows because F is parity-preserving. ■

So, given a non-orthogonal, parity-preserving linear gate G, we now need to show how to implement CPD.

For the rest of this section, we will consider a situation where we are given an n-bit string, with the initial state $xy0^{n-2}$ (where x and y are two arbitrary bits), and then we apply a sequence of $\mathbb{F}_2$ linear transformations to the string. Here we do *not* assume that ancilla bits initialized to 1 are available, though ancilla bits initialized to 0 are fine. As a result, at every time step, every bit in our string will be either x, y, $x \oplus y$, or 0. Because we are studying only the linear case here, not the affine case, we do not need to worry about the possibilities $x \oplus 1$, $y \oplus 1$, etc., which would considerably complicate matters. (We will handle the affine case in Section 9.4.)

By analogy to Section 8.4, let us define the *type* of a string $z(x,y) \in \{0,1\}^n$ to be (a,b,c), if z contains a copies of x and b copies of y and c copies of $x \oplus y$. Since any string of type (a,b,c) can be transformed into any other string of type (a,b,c) using bit-swaps, the type of z is its only relevant property. As before, if by repeatedly applying a linear gate G, we can map some string of type (a,b,c) into some string of type (a',b',c'), then we say that G *generates the slope*

$$(a' - a, b' - b, c' - c).$$

Again, if G generates the slope (p,q,r), then G^{-1} generates the slope $(-p,-q,-r)$. Also, if G generates the slope (p,q,r) using the string z, and the slope (p',q',r') using the string z', then it generates the slope $(p+p',q+q',r+r')$ using the string zz'. For these reasons, the set of achievable slopes forms a 3-dimensional lattice, which we denote $\mathcal{L}(G) \subseteq \mathbb{Z}^3$. Moreover, this is a lattice with a strong symmetry property:

Proposition 66 *$\mathcal{L}(G)$ is symmetric under all 6 permutations of the 3 coordinates.*

Proof. Clearly we can interchange the roles of x and y. However, we can also, e.g., define $x' := x$ and $y' := x \oplus y$, in which case $x' \oplus y' = y$. In the triple $(x, y, x \oplus y)$, each element is the XOR of the other two. ■

Just like before, our entire question will boil down to whether or not the lattice $\mathcal{L}(G)$ contains a certain point. In this case, the point is $(1,-1,1)$. The importance of the $(1,-1,1)$ point comes from the following lemma.

Lemma 67 *Let G be any linear gate. Then G generates* CPD*, if and only if $(1,-1,1) \in \mathcal{L}(G)$.*

Proof. If G generates CPD, then it maps $x0y$, which has type $(1,1,0)$, to $x, x, x \oplus y$, which has type $(2,0,1)$. This amounts to generating the slope $(1,-1,1)$.

Conversely, suppose $(1,-1,1) \in \mathcal{L}(G)$. Then there is some gate $H \in \langle G \rangle$, and some string of the form $z = x^a y^b (x \oplus y)^c$, such that

$$H(z) = x^{a+1} y^{b-1} (x \oplus y)^{c+1}.$$

But the very fact that G generates such an H implies that G is non-degenerate, and if G is non-degenerate, then Lemma 46 implies that, starting from $xy0^{n-2}$, we can use G to increase the numbers of x, y, and $x \oplus y$ simultaneously without bound. That is, there is some $Q \in \langle G \rangle$ such that (omitting the 0 bits)

$$Q(xy) = x^{a'} y^{b'} (x \oplus y)^{c'},$$

where $a' > a$ and $b' > b$ and $c' > c$. So then the procedure to implement CPD is to apply Q, then H, then Q^{-1}. ■

Thus, our goal now is to show that, if G is any non-orthogonal, parity-preserving linear gate, then $(1,-1,1) \in \mathcal{L}(G)$. Observe that, if $k(G) = 4$, then Corollary 21 implies that G is orthogonal, contrary to assumption. By Theorem 19, this means that the only remaining possibility is $k(G) = 2$. This has the following consequence for the lattice $\mathcal{L}(G)$.

Proposition 68 *If G is a linear gate with $k(G) \leq 2$, then $\mathcal{L}(G)$ contains all even points (i.e., all (p,q,r) such that $p \equiv q \equiv r \equiv 0 \pmod 2$).*

Proof. By Proposition 40, we must be able to use G to map 10^{n-1} to 1110^{n-3}. Since 0^n is mapped to itself by any linear transformation, this implies that G can map $x0^{n-1}$ to $xxx0^{n-3}$, which means that it generates the slope $(2,0,0)$. So $(2,0,0) \in \mathcal{L}(G)$. By Proposition 66, then, $\mathcal{L}(G)$ also contains the points $(0,2,0)$ and $(0,0,2)$. But these three generate all the even points. ■

Proposition 68 has the following immediate corollary.

Corollary 69 *Let G be a linear gate with $k(G) \leq 2$, and suppose $\mathcal{L}(G)$ contains any point (p,q,r) such that $p \equiv q \equiv r \equiv 1 \pmod 2$. Then $\mathcal{L}(G)$ contains $(1,-1,1)$.*

Thus, it remains only to prove the following lemma.

Lemma 70 *Let G be any parity-preserving, non-orthogonal linear gate. Then $\mathcal{L}(G)$ contains a point (p,q,r) such that $p \equiv q \equiv r \equiv 1 \pmod 2$.*

Proof. In the proof of Theorem 64, this is the first place where we use the linearity of G in an essential way—i.e., not just to deduce that $k(G) \in \{2,4\}$, or to avoid dealing with bits of the form $x \oplus 1$, $y \oplus 1$, etc. It is also the first place where we use the non-orthogonality of G, other than to rule out the possibility that $k(G) = 4$; and the first place where we use that G is parity-preserving.

Let us view G as an $n \times n$ matrix over $\mathbb{F}_2$. Then the fact that G is parity-preserving means that every column of G has odd Hamming weight. Also, the fact that G is non-orthogonal means that it must have two columns with an odd inner product. Assume without loss of generality that these are the first and second columns. Let the first two columns of G consist of:

$$\begin{gathered} a \text{ rows of the form } 1,0, \\ b \text{ rows of the form } 1,1, \\ c \text{ rows of the form } 0,1, \\ d \text{ rows of the form } 0,0, \end{gathered}$$

where a,b,c,d are nonnegative integers summing to n. Then from the above, we have that $a+b$ and $b+c$ and b are all odd, from which it follows that a and c are even.

Now consider applying G to the input $xy0^{n-2}$. The result will contain:

$$\begin{gathered} a \text{ copies of } x, \\ c \text{ copies of } y, \\ b \text{ copies of } x \oplus y. \end{gathered}$$

This means that we've mapped a string of type $(1,1,0)$ to a string of type (a,c,b), thereby generating the slope $(a-1,c-1,b)$. But this is the desired odd point in $\mathcal{L}(G)$. ■

Combining Lemma 65, Lemma 67, Corollary 69, and Lemma 70 now completes the proof of Theorem 64.

9.3 Non-Parity-Preserving Linear Generates CNOT

To complete the classification of linear gate sets, our final task is to prove the following theorem.

Theorem 71 *Let G be any non-parity-preserving linear gate. Then G generates* CNOT *(or equivalently, all linear transformations).*

Recall that COPY is the partial gate that maps $x0$ to xx. We will first show how to use G to generate COPY, and then use COPY to generate CNOT.

Note that since G is linear, it cannot be parity-flipping. So since G is non-parity-preserving, it is also non-parity-respecting, and $k(G)$ must be finite and odd. But by Theorem 19, this means that $k(G) = 1$: in other words, G is non-mod-respecting.

Let z be an n-bit string that consists entirely of copies of x and 0. Let the *type* of z be the number of copies of x. Clearly we can map any z to any other z of the same type using swaps, so the type of z is its only relevant property. Also, we say that a gate G *generates the slope* p, if by applying G repeatedly, we can map some input z of type a to some input z' of type $a+p$. Note that if G generates the slope p, then by reversibility, it also generates the slope $-p$. Also, if G generates the slope p by mapping z to z', and the slope q by mapping w to w', then it generates the slope $p+q$ by mapping zw to $z'w'$. For these reasons, the set of achievable slopes forms an ideal in $\mathbb{Z}$ (i.e., a 1-dimensional lattice), which we can denote $\mathcal{L}(G)$. The question of whether G generates COPY can then be rephrased as the question of whether $\mathcal{L}(G)$ contains 1—or equivalently, of whether $\mathcal{L}(G) = \mathbb{Z}$.

Lemma 72 *A linear gate G generates* COPY *if and only if* $1 \in \mathcal{L}(G)$.

Proof. If G generates COPY, then clearly $1 \in \mathcal{L}(G)$. For the converse direction, suppose $1 \in \mathcal{L}(G)$. Then G can be used to map an input of type a to an input of type $a+1$, for some a. Hence G can also be used to map inputs of type b to inputs of type $b+1$, for all $b \geq a$. This also implies that G is non-degenerate, so by Lemma 46, it can be used to increase the number of copies of x without bound. So to copy a bit x, we first apply some gate $H \in \langle G \rangle$ to map x to x^b for some $b \geq a$, then map x^b to x^{b+1}, and finally apply H^{-1} to map x^{b+1} to x^2. ■

Now, the question of whether $1 \in \mathcal{L}(G)$ is easily answered.

Lemma 73 *Let G be any non-mod-respecting linear gate. Then $\mathcal{L}(G) = \mathbb{Z}$.*

Proof. This follows almost immediately from Proposition 40, together with the fact that $k(G) = 1$. We simply need to observe that, if $x = 1$, then the number of copies of x corresponds to the Hamming weight. ■

Finally, we show that COPY suffices for CNOT.

Lemma 74 *Let G be any linear gate that generates* COPY*. Then G generates* CNOT.

Proof. We will actually prove that G generates *any* linear transformation F. Observe that, if G generates COPY, then it must be non-degenerate. Therefore, by copying bits whenever needed, and using G to do computation on them, clearly we can map the input x to a string of the form

$$x, \operatorname{gar}(x), F(x),$$

for some garbage string $\operatorname{gar}(x)$. Since G generates COPY, we can then make one copy of $F(x)$, mapping the above to

$$x, \operatorname{gar}(x), F(x), F(x).$$

Next we can uncompute the computation of F to get

$$x, F(x).$$

By reversibility, we can then map the above to

$$x, F(x), \operatorname{gar}(F(x)), x.$$

By inverting COPY, we can then implement $xx \to x$, to map the above to

$$F(x), \operatorname{gar}(F(x)), x.$$

Finally, we can uncompute the computation of x to get $F(x)$ alone. ■

Combining Lemmas 72, 73, and 74 now completes the proof of Theorem 71. Then combining Theorems 32, 33, 36, 37, 62, 63, 64, and 71, we can summarize our progress on the linear case as follows.

Corollary 75 *Every set of linear gates generates either* $\langle\varnothing\rangle$, $\langle \mathrm{T}_6\rangle$, $\langle \mathrm{T}_4\rangle$, $\langle\mathrm{CNOTNOT}\rangle$, *or* $\langle\mathrm{CNOT}\rangle$.

9.4 Adding Back the NOTs

Now that we have completed the classification of the *linear* gate classes, the final step that remains is to take care of the affine parts. We first give some useful lemmas for manipulating affine gates.

Lemma 76 $\mathrm{NOT}^{\otimes k}$ *generates* NOTNOT *for all* $k \geq 1$, *as well as* NOT *if* k *is odd.*

Proof. To implement $\mathrm{NOTNOT}(x,y)$, apply $\mathrm{NOT}^{\otimes k}$ to $x, a_1 \ldots a_{k-1}$ and then to $y, a_1 \ldots a_{k-1}$. To implement $\mathrm{NOT}(x)$, let $\ell := \frac{k-1}{2}$. Apply $\mathrm{NOT}^{\otimes k}$ to $x, a_1 \ldots a_\ell, b_1 \ldots b_\ell$, then $x, a_1 \ldots a_\ell, c_1 \ldots c_\ell$, then $x, b_1 \ldots b_\ell, c_1 \ldots c_\ell$. ■

More generally:

Lemma 77 *Let* G *be any gate of the form* $\mathrm{NOT} \otimes H$. *Then* G *generates* NOTNOT.

Proof. To implement $\mathrm{NOTNOT}(x,y)$, first apply G to x, a where a is some ancilla string; then apply G^{-1} to y, a. ■

Also:

Lemma 78 *Let* $G(x) = Ax \oplus b$ *be an affine gate. Then* $G + \mathrm{NOTNOT}$ *generates* A *itself.*

Proof. First we use $G^{\otimes 2}$ to map $x, 0^n$ to $Ax \oplus b, b$; then we use NOTNOT gates to map $Ax \oplus b, b$ to $Ax, 0^n$. ■

By combining Lemmas 77 and 78, we obtain the following.

Corollary 79 (Cruft Removal) *Let* $G(x) = Ax \oplus b$ *be an* n*-bit affine gate. Suppose* A *applies a linear transformation* A' *to the first* m *bits of* x, *and acts as the identity on the remaining* $n-m$ *bits. Then* G *generates an* m*-bit gate of the form* $H(x) = A'x \oplus c$.

Proof. If $b_i = 0$ for all $i > m$, then we are done. Otherwise, we can use Lemma 77 to generate NOTNOT, and then Lemma 78 to generate $H(x) = A'x$. ■

Lemma 80 *Let* S *be any class of parity-preserving linear or affine gates. Then there are no classes between* $\langle S\rangle$ *and* $\langle S + \mathrm{NOT}\rangle$ *other than* $\langle S + \mathrm{NOTNOT}\rangle$.

Proof. Let G be a transformation that is generated by $S + \mathrm{NOT}$ but not by S. Then we need to show how to generate NOT or NOTNOT themselves using $S + G$.

We claim that G acts as

$$G(x) = V(x) \oplus b,$$

where $V(x)$ is some parity-preserving affine transformation generated by S, and b is some nonzero string. First, V must be generated by S because, given any circuit for G over the set $S + \mathrm{NOT}$, we can always push the NOT gates to the end; this leaves us with a circuit for the "S part" of G. (This is the one place where we use that S is affine.) Also, b must be nonzero because otherwise, G would already be generated by S.

Given x, suppose we first apply V^{-1} (which must be generated by S), then apply G. This yields

$$G\left(V^{-1}\left(x\right)\right) = V\left(V^{-1}\left(x\right)\right) \oplus b = x \oplus b,$$

which is equivalent to $\mathrm{NOT}^{\otimes k}$ for some nonzero k. By Lemma 76, this generates NOTNOT. If $|b|$ is always even, then since V is parity-preserving, clearly we remain within $\langle S + \mathrm{NOTNOT}\rangle$. If, on the other hand, $|b|$ is ever odd, then again by Lemma 76, we can generate NOT. ■

We can finally complete the proof of Theorem 60, characterizing the possible affine classes.

Proof of Theorem 60. If we restrict ourselves to the linear part of the class, then we know from Corollary 75 that the only possibilities are $\langle\mathrm{CNOT}\rangle$, $\langle\mathrm{CNOTNOT}\rangle$, $\langle\mathrm{T}_4\rangle$, $\langle\mathrm{T}_6\rangle$, and $\langle\varnothing\rangle$ (i.e., the trivial class). We will handle these possibilities one by one.

Linear part is $\langle\mathrm{CNOT}\rangle$. Since CNOT can already generate all affine transformations (by Theorem 32), using an ancilla bit initialized to 1, we have $\langle S\rangle \subseteq \langle\mathrm{CNOT}\rangle$. For the other direction, Corollary 79 implies that S must generate a gate of the form $\mathrm{CNOT}\left(x\right) \oplus b$, for some $b \in \{0,1\}^2$. However, it is not hard to see that all such gates can generate CNOT itself.

Linear part is $\langle\mathrm{CNOTNOT}\rangle$. Here we clearly have $\langle S\rangle \subseteq \langle\mathrm{CNOTNOT}, \mathrm{NOT}\rangle$. Meanwhile, Corollary 79 again implies that S generates a gate of the form $G\left(x\right) = \mathrm{CNOTNOT}\left(x\right) \oplus b$, for some $b \in \{0,1\}^3$. Suppose the first bit of b is 1; this is the bit that corresponds to the control of the CNOTNOT. Then $G\left(G\left(x\right)\right)$ generates NOTNOT, so by Lemma 78, we can generate CNOTNOT. If, on the other hand, the first bit of b is 0, then G generates NOT or NOTNOT directly, so we can again use Lemma 78 to generate CNOTNOT. Therefore $\langle S\rangle$ lies somewhere between $\langle\mathrm{CNOTNOT}\rangle$ and $\langle\mathrm{CNOTNOT}, \mathrm{NOT}\rangle$. But since CNOTNOT already generates NOTNOT, Lemma 80 says that the only possibilities are $\langle\mathrm{CNOTNOT}\rangle$ and $\langle\mathrm{CNOTNOT}, \mathrm{NOT}\rangle$.

Linear part is $\langle\mathrm{T}_4\rangle$. In this case $\langle S\rangle \subseteq \langle\mathrm{T}_4, \mathrm{NOT}\rangle$. Again Corollary 79 implies that S generates a gate of the form $G\left(x\right) = \mathrm{T}_4\left(x\right) \oplus b$, for some $b \in \{0,1\}^4$. If $b = 1111$, then S generates F_4. So $\langle S\rangle$ lies somewhere between $\langle\mathrm{F}_4\rangle$ and $\langle\mathrm{F}_4, \mathrm{NOT}\rangle = \langle\mathrm{T}_4, \mathrm{NOT}\rangle$, but then Lemma 80 ensures that $\langle\mathrm{F}_4\rangle$, $\langle\mathrm{F}_4, \mathrm{NOTNOT}\rangle = \langle\mathrm{T}_4, \mathrm{NOTNOT}\rangle$, and $\langle\mathrm{T}_4, \mathrm{NOT}\rangle$ are the only possibilities. Likewise, if $b = 0000$, then S generates T_4, so $\langle\mathrm{T}_4\rangle$, $\langle\mathrm{T}_4, \mathrm{NOTNOT}\rangle$, and $\langle\mathrm{T}_4, \mathrm{NOT}\rangle$ are the only possibilities.

Next suppose $|b|$ is odd. Then $G\left(G\left(x\right)\right) = \mathrm{NOT}^{\otimes 4}\left(x\right)$, which generates NOTNOT by Lemma 76. So by Lemma 78, we generate T_4 as well. Thus we have at least $\langle\mathrm{T}_4, \mathrm{NOTNOT}\rangle$. But since G itself is parity-flipping, $\langle S\rangle$ is not parity-preserving, leaving $\langle\mathrm{T}_4, \mathrm{NOT}\rangle$ as the only possibility by Lemma 80. Finally suppose $|b| = 2$: without loss of generality, $b = 1100$. Let Q be an operation that swaps the first two bits of x with the last two bits. Then $G\left(Q\left(G\left(x\right)\right)\right)$ is equivalent to $\mathrm{NOT}^{\otimes 4}\left(x\right)$ up to swaps, so again we have at least $\langle\mathrm{T}_4, \mathrm{NOTNOT}\rangle$, leaving $\langle\mathrm{T}_4, \mathrm{NOTNOT}\rangle$ and $\langle\mathrm{T}_4, \mathrm{NOT}\rangle$ as the only possibilities.

Linear part is $\langle\mathrm{T}_6\rangle$. In this case $\langle S\rangle \subseteq \langle\mathrm{T}_6, \mathrm{NOT}\rangle$. Again, Corollary 79 implies that S generates $G(x) = \mathrm{T}_6\left(x\right) \oplus b$ for some $b \in \{0,1\}^6$. If $b = 000000$, then S generates T_6, so $\langle\mathrm{T}_6\rangle$, $\langle\mathrm{T}_6, \mathrm{NOTNOT}\rangle$, and $\langle\mathrm{T}_6, \mathrm{NOT}\rangle$ are the only possibilities by Lemma 80. If $|b|$ is odd, then $G\left(G\left(x\right)\right) = \mathrm{NOT}^{\otimes 6}\left(x\right)$. By Lemma 76, this means that S generates NOTNOT, so by Lemma 78, it generates T_6 as well. But G is parity-flipping, leaving $\langle\mathrm{T}_6, \mathrm{NOT}\rangle$ as the only possibility by Lemma 80. If $|b|$ is 2 or 4, then by an appropriate choice of swap operation Q, we can cause $G\left(Q\left(G\left(x\right)\right)\right)$ to generate NOTNOT, so again $\langle\mathrm{T}_6, \mathrm{NOTNOT}\rangle$ and $\langle\mathrm{T}_6, \mathrm{NOT}\rangle$ are the only possibilities.

Finally, if $b = 111111$, then $G(x) = \mathrm{F}_6(x)$. In this case we start with the operation

$$\mathrm{F}_6(x00000) = 1\overline{x}\overline{x}\overline{x}\overline{x}\overline{x}$$

Using three of the $\overline{x}$ outputs and three fresh 0 ancilla bits, we then perform

$$\mathrm{F}_6(\overline{x}\overline{x}\overline{x}000) = 111xxx$$

Next, bringing the xxx outputs together with the remaining $\overline{x}\overline{x}$ outputs and one fresh 0 ancilla bit, we apply

$$\mathrm{F}_6(xxx\overline{x}\overline{x}0) = 11100\overline{x}$$

In summary, we have performed a $\mathrm{NOT}(x)$ operation with some garbage still around. However, if we repeat this entire procedure 6 times, then the Hamming weight of the garbage will be a multiple of 6. We can remove all this of garbage using the F_6 gate. Therefore, we have created a $\mathrm{NOT}^{\otimes 6}$ gate, which generates NOTNOT by Lemma 76. So again we can generate T_6, leaving $\langle \mathrm{T}_6, \mathrm{NOTNOT}\rangle$ and $\langle \mathrm{T}_6, \mathrm{NOT}\rangle$ as the only possibilities by Lemma 78.

Linear part is $\langle \varnothing \rangle$. In this case $\langle S\rangle \subseteq \langle \mathrm{NOT}\rangle$, so Lemma 80 implies that the only possibilities are $\langle \varnothing \rangle$, $\langle \mathrm{NOTNOT}\rangle$, and $\langle \mathrm{NOT}\rangle$. ■

10 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—which would build directly on our results here—would be to classify all possible quantum gate sets within the *stabilizer group*, which is a quantum generalization of the group of affine classical reversible transformations. Since the stabilizer group is discrete, here at least there is no need for representation theory, Lie algebras, or any notion of approximation, but the problem still seems complicated. A different step in the direction we want, which *would* involve Lie algebras, would be to classify all sets of 1- and 2-qubit gates. A third 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 [9], which classifies qubit Hamiltonians in terms of the complexity of approximating ground state energies, might be relevant. Yet a fourth 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 [18]) 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? 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?

11 Acknowledgments

At the very beginning of this project, Emil Jeřábek [12] 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 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] 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.

[5] 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.

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

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

[8] 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.

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

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

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

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

[13] 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.

[14] 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`.

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

[16] D. E. Knuth. *The Art of Computer Programming, Volume 1, 2nd edition.* Addison-Wesley, 1969.

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

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

[19] 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.

[20] J. MacWilliams. Orthogonal matrices over finite fields. *American Mathematical Monthly*, 76(2):152–164, 1969.

[21] 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.

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

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

[24] L. Schaefer. Reversible Gate Classifier, 2015. `https://github.com/lrschaeffer/Gate-Classifier`.

[25] 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.

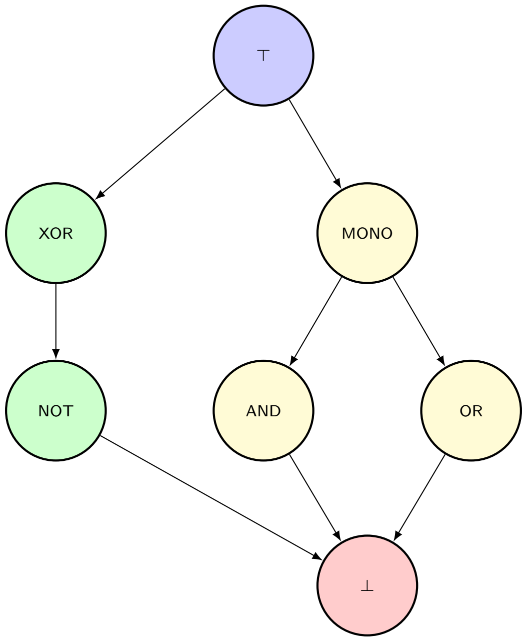

Figure 7: "Post's Lattice Lite"

[26] 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.

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

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

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

12 Appendix: Post's Lattice with Free Constants

For completeness, in this appendix we prove a 'quick-and-dirty' version of Post's 1941 classification theorem [22], for sets of ordinary (non-reversible) Boolean logic gates.

Theorem 81 (Post's Lattice Lite) *Assume the constant functions $f = 0$ and $f = 1$, as well as the identity function $f(x) = x$, are available for free. Then the only Boolean clones (i.e., classes of Boolean functions $f : \{0,1\}^n \to \{0,1\}$ closed under composition and addition of dummy variables) are the following:*

1. *The trivial class (containing the constant and identity functions).*
2. *The* AND *class.*

3. *The* OR *class.*

4. *The class of monotone functions (generated by* $\{\text{AND}, \text{OR}\}$*).*

5. *The* NOT *class.*

6. *The class of affine functions (generated by* $\{\text{XOR}, \text{NOT}\}$*).*

7. *The class of all Boolean functions (generated by* $\{\text{AND}, \text{NOT}\}$*).*

Proof. We take it as known that $\{\text{AND}, \text{OR}\}$ generates all monotone functions, $\{\text{XOR}, \text{NOT}\}$ generates all affine functions, and $\{G, \text{NOT}\}$ generates all functions, for any 2-bit non-affine gate G.

Let $\mathcal{C}$ be a Boolean clone that contains the constant 0 and 1 functions. Then $\mathcal{C}$ is closed under restrictions (e.g., if $f(x, y) \in \mathcal{C}$, then $f(0, y)$ and $f(x, 1)$ are also in $\mathcal{C}$), and that is the crucial fact we exploit.

First suppose $\mathcal{C}$ contains a non-monotone gate. Then certainly we can construct a NOT gate by restricting inputs. If, in addition, $\mathcal{C}$ contains a non-affine gate, then by Proposition 44, we can construct a 2-bit non-affine gate by restricting inputs: AND, OR, NAND, NOR, IMPLIES, or NOT (IMPLIES). Together with the NOT gate, this puts us in class 7. If, on the other hand, $\mathcal{C}$ contains only affine gates, then as long as one of those gates depends on at least two input bits, by restricting inputs we can construct a 2-bit non-degenerate affine gate: XOR or NOT (XOR). Together with the NOT gate, this puts us in class 6. If, on the other hand, every gate depends on only 1 input bit, then we are in class 5.

Next suppose $\mathcal{C}$ contains only monotone gates. Clearly the only *affine* monotone gates are trivial. Thus, as long as one of the gates is nontrivial, it is non-affine, so Proposition 44 again implies that we can construct a non-affine 2-bit monotone gate by restricting inputs: AND or OR. If we can construct only AND gates, then we are in class 2; if only OR gates, then we are in class 3; if both, then we are in class 4. If, on the other hand, every gate is trivial, then we are in class 1. ■

The simplicity of Theorem 81 underscores how much more complicated it is to understand reversible gates than non-reversible gates, when we impose a similar rule in both cases (i.e., that 0 and 1 constant or ancilla bits are available for free).

13 Appendix: The Classification Theorem with Loose Ancillas

Theorem 82 *Under the loose ancilla rule, the only change to Theorem 3 is that every* $\mathcal{C} +$ NOTNOT *class collapses with the corresponding* $\mathcal{C} +$ NOT *class.*

Proof. That this collapse happens is clear: under the loose ancilla rule, we can always simulate a NOT gate by applying a NOTNOT gate to the desired bit, as well as to a "dummy" ancilla bit that will never be used for any other purpose.

To see that no other collapses happen, we must show that the remaining classes are distinct. Under the usual ancilla rule, the classes are distinct because for any pair of classes we can find an invariant satisfied by one, but not the other, to separate the two. We would like to do the same for loose ancilla classes, but invariants under the usual rule need not, *a priori*, be invariants under the loose ancilla rule. More concretely, as we have seen, a gate set that preserves parity under the usual rule need no longer preserve it under the loose ancilla rule. However, we claim that all the *other* invariants are also loose ancilla invariants.

Suppose $G(x,a) = (H(x), b)$ is a transformation generated under the loose ancilla rule, where a and b are constants, so that under the loose ancilla rule, we have also generated H. We would like to show that any invariant of G must also hold for H, so let us consider the invariants one by one.

- If G is mod-k-respecting then
$$|G(x)| - |x| = |H(x)| - |x| + |a| - |b|,$$
is constant modulo k, and hence $|H(x)| - |x|$ is constant modulo k, so $H(x)$ is mod-k-respecting. For $k \geq 3$, mod-k-respecting is equivalent to mod-k-preserving by Theorem 12. When $k = 2$, we have already seen that NOTNOT collapses with NOT.

- If G is conservative then $0 = |G(x)| - |x| = |H(x)| - |x| + |a| - |b|$ as above. If we average over all x and appeal to reversibility, then we see that $|a| - |b|$ must be 0, and hence H is conservative.

- If G is affine then
$$G\begin{pmatrix} x \\ a \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}\begin{pmatrix} x \\ a \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} H(x) \\ b \end{pmatrix},$$
so clearly $H(x) = M_{11}x + M_{12}a + c_1$ is affine as well. Since $M_{21}x + M_{22}a + c_2 = b$ for all x, we must have $M_{21} = 0$. But this means that if the columns of
$$\begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix},$$
the linear part of G, have weight 2, weight 4, or are orthogonal, then the same is true of columns of
$$\begin{pmatrix} M_{11} \\ 0 \end{pmatrix},$$
and hence the columns of M_{11} itself. In short, if the linear part of G has any of the properties we are interested in, then so does the linear part of H.

- If G is orthogonal then $c_1 = 0$ and $c_2 = 0$. Recall that $M_{21} = 0$, and since a matrix of the form
$$\begin{pmatrix} A & B \\ 0 & C \end{pmatrix}$$
has an inverse of the same form, and the inverse of an orthogonal matrix is its transpose, we see that $M_{12} = 0$. It follows that $H(x) = M_{11}x + M_{12}a + c_1$ is actually just $H(x) = M_{11}x$ when G is orthogonal, therefore H is orthogonal because M_{11} is orthogonal.

■

14 Appendix: Number of Gates Generating Each Class

In this appendix, we count how many n-bit gates belong to each of the classes of Theorem 3. Let us write $\langle G \rangle_n$ for the set of n-bit gates generated by G, and $\# \langle G \rangle_n$ for the number of n-bit gates generated by G. Then Theorem 83 gives the exact number of gates in each class, while Theorem 87 gives the asymptotics.

Theorem 83 *Let $n \geq 1$ be an integer.*

- *The total number of gates is*
$$\# \langle \text{Toffoli} \rangle_n = (2^n)!$$
and the non-affine classes break down as follows. For $k \geq 3$,
$$\begin{aligned}
\# \langle \text{Fredkin}, \text{NOT} \rangle_n &= 2 \left(2^{n-1}! \right)^2 \\
\# \langle \text{Fredkin}, \text{NOTNOT} \rangle_n &= \left(2^{n-1}! \right)^2 \\
\# \langle \text{C}_k \rangle_n &= \prod_{i=0}^{k-1} \left(\sum_{j-i \equiv 0 (\operatorname{mod} k)} \binom{n}{j} \right)! \\
\# \langle \text{Fredkin} \rangle_n &= \prod_{i=0}^{n} \left(\binom{n}{i}! \right)
\end{aligned}$$

- *The total number of affine gates is*
$$\# \langle \text{CNOT} \rangle_n = 2^{n(n+1)/2} \prod_{i=1}^{n} \left(2^i - 1 \right).$$

- *The numbers of parity-preserving and parity-respecting gates are:*
$$\begin{aligned}
\# \langle \text{CNOTNOT} \rangle_n &= 2^{n(n+1)/2-1} \prod_{i=1}^{n-1} \left(2^i - 1 \right) \\
\# \langle \text{CNOTNOT}, \text{NOT} \rangle_n &= 2^{n(n+1)/2} \prod_{i=1}^{n-1} \left(2^i - 1 \right)
\end{aligned}$$

- *The numbers of gates in $\langle \varnothing \rangle$, $\langle \text{T}_6 \rangle$, and $\langle \text{T}_4 \rangle$ are:*
$$\# \langle \varnothing \rangle_n = n!$$
$$\# \langle \text{T}_4 \rangle_n = \begin{cases} 2^{m^2} \prod_{i=1}^{m-1} \left(2^{2i} - 1 \right), & \text{if } n = 2m \\ 2^{m^2} \prod_{i=1}^{m} \left(2^{2i} - 1 \right), & \text{if } n = 2m+1 \end{cases}$$
$$\# \langle \text{T}_6 \rangle_n = \begin{cases} 1 & \text{if } n = 1 \\ 2^{4m^2+1} \prod_{i=1}^{2m} \left(2^{2i} - 1 \right) & \text{if } n = 4m+2 \\ 2^{4m^2+2m+1} \left(2^{2m+1} + (-1)^m \right) \prod_{i=1}^{2m} \left(2^{2i} - 1 \right) & \text{if } n = 4m+3 \\ 2^{4m^2-2m+1} \left(2^{2m-1} - (-1)^m \right) \prod_{i=1}^{2m-2} \left(2^{2i} - 1 \right) & \text{if } n = 4m \geq 4 \\ 2^{4m^2-2m+1} \left(2^{2m} - (-1)^m \right) \prod_{i=1}^{2m-1} \left(2^{2i} - 1 \right) & \text{if } n = 4m+1 \geq 5 \end{cases}$$

Furthermore,

$$\# \langle \mathrm{F}_4 \rangle_n = \# \langle \mathrm{T}_4 \rangle_n .$$

- *For any linear class $\langle G \rangle$ we have*

$$\# \langle G, \mathrm{NOT} \rangle_n = \# \langle G \rangle_n 2^n$$
$$\# \langle G, \mathrm{NOTNOT} \rangle_n = \# \langle G \rangle_n 2^{n-1}$$

Let us count each class in turn. To start, note that an n-bit reversible gate is, by definition, a permutation of $\{0,1\}^n$, so there are $(2^n)!$ gates in total.

Parity-preserving gates map even-weight strings to even-weight strings, and odd-weight strings to odd-weight strings. It follows that there are $\left((2^{n-1})!\right)^2$ parity-preserving gates. Clearly there are exactly twice as many parity-respecting gates, since we can append a NOT gate to any parity-preserving gate to get a parity-flipping gate, and vice versa.

The mod-k-preserving gates (for $k \geq 3$) also decompose into a product of permutations, one for each Hamming weight class modulo k. This leads to the formula

$$\# \langle \mathrm{C}_k \rangle_n = \prod_{i=0}^{k-1} \left(\sum_{j-i \equiv 0 (\operatorname{mod} k)} \binom{n}{j} \right)!$$

Likewise, for conservative gates, we have

$$\# \langle \mathrm{Fredkin} \rangle_n = \prod_{i=0}^{n} \left(\binom{n}{i}! \right).$$

The linear part of an affine gate is an $n \times n$ invertible matrix A. The number of such matrices is well-known to be

$$\prod_{i=0}^{n-1} \left(2^n - 2^i\right) = 2^{n(n-1)/2} \prod_{i=1}^{n} \left(2^i - 1\right).$$

There are an additional 2^n choices for the affine part, so

$$\# \langle \mathrm{CNOT} \rangle_n = 2^{n(n+1)/2} \prod_{i=1}^{n} \left(2^i - 1\right).$$

A parity-preserving affine transformation is an affine transformation on the $(n-1)$-dimensional subspace of even Hamming-weight vectors, extended to the entire space by defining the transformation on any odd-weight vector. There are $2^{n(n-1)/2} \prod_{i=1}^{n-1} \left(2^i - 1\right)$ affine transformations on $n-1$ dimensions and 2^{n-1} choices of odd-weight vector, so there are

$$\# \langle \mathrm{CNOTNOT} \rangle_n = 2^{n(n+1)/2-1} \prod_{i=1}^{n-1} \left(2^i - 1\right)$$

parity-preserving affine transformations, and twice as many parity-respecting affine transformations.

We refer to MacWilliams [20] for the formula (below) for the number of orthogonal $n \times n$ matrices.

$$\# \langle \mathrm{T}_4 \rangle_n = \begin{cases} 2^{m^2} \prod_{i=1}^{m-1} \left(2^{2i} - 1\right), & \text{if } n = 2m, \\ 2^{m^2} \prod_{i=1}^{m} \left(2^{2i} - 1\right), & \text{if } n = 2m+1. \end{cases}$$

We now turn our attention to counting $\langle \mathrm{T}_6 \rangle_n$, which is more involved. The approach will be similar to that of MacWilliams [20]. It will help to consider $\langle \mathrm{T}_4 \rangle_n$ and $\langle \mathrm{T}_6 \rangle_n$ as groups. Indeed, $\langle \mathrm{T}_4 \rangle_n$ is just the orthogonal group $\mathrm{O}(n)$ over $\mathbb{F}_2$, and $\langle \mathrm{T}_6 \rangle_n$ is a proper subgroup.

The idea is to find a unique representative for each of the cosets of $\langle \mathrm{T}_6 \rangle_n$ in $\langle \mathrm{T}_4 \rangle_n$. Since we know $\# \langle \mathrm{T}_4 \rangle_n$ by [20], dividing by the number of unique representatives will give us $\# \langle \mathrm{T}_6 \rangle_n$ as desired.

Recall that by Lemma 16, the Hamming weight of each column vector of an orthogonal matrix is either $1 \bmod 4$ or $3 \bmod 4$. If $A \in \langle \mathrm{T}_4 \rangle_n$ is an orthogonal matrix with column vectors $a_1, \ldots, a_n$, then the *characteristic vector* $c(A)$ is an n-dimensional vector whose i^{th} entry, $c_i(A)$, is defined as follows:

$$c_i(A) = \begin{cases} 1 & \text{if } |a_i| \equiv 3 \pmod 4 \\ 0 & \text{if } |a_i| \equiv 1 \pmod 4 \end{cases}.$$

The following lemma shows that these characteristic vectors can be used as a representatives for the cosets of $\langle \mathrm{T}_6 \rangle_n$.

Lemma 84 *Two orthogonal transformations, $A, B \in \langle \mathrm{T}_4 \rangle_n$, are in the same coset of $\langle \mathrm{T}_6 \rangle_n$ if and only if $c(A) = c(B)$.*

Proof. Note that A and B are in the same coset if and only if $T := BA^{-1} = BA^T$ is in $\langle \mathrm{T}_6 \rangle_n$. We know that $T \in \langle \mathrm{T}_4 \rangle_n$, and that $T(a_i) = b_i$ for all i. Since $a_1, \ldots, a_n$ is an orthogonal basis, Theorem 37 says that $T \in \langle \mathrm{T}_6 \rangle_n$ if and only if T is mod-4-preserving. By Theorem 20, this holds if and only if $|a_i| \equiv |b_i| \pmod 4$ for all i, or equivalently, $c(A) = c(B)$. ■

Lemma 84 shows that it suffices to count the number of possible characteristic vectors. Perhaps surprisingly, not every characteristic vector is achievable; the following lemma shows exactly which ones are.

Lemma 85 *If $A \in \langle \mathrm{T}_4 \rangle_n$, then $|c(A)| \equiv 0 \pmod 4$. Furthermore, for every characteristic vector c such that $|c| \equiv 0 \pmod 4$, there exists a matrix $A \in \langle \mathrm{T}_4 \rangle_n$ such that $c(A) = c$.*

Proof. Let $A \in \langle \mathrm{T}_4 \rangle_n$ with column vectors $a_1, \ldots, a_n$. Of course, A might not preserve Hamming weight mod 4. The main idea of the proof is to promote A to an affine function $f(x) = Ax \oplus b$ that *does* preserve Hamming weight mod 4. We know that such a function exists because we can decompose A into a circuit of T_4 gates by Theorem 36. Replacing each such gate with F_4 will yield a circuit of the desired form that preserves Hamming weight mod 4.

Recall from Theorem 20 that if f preserves Hamming weight mod 4, then $|a_i| + 2(a_i \cdot b) \equiv 1 \pmod 4$. Expanding out this condition we get

$$a_i \cdot b \equiv \begin{cases} 1 \pmod 2 & \text{if } |a_i| \equiv 3 \pmod 4 \\ 0 \pmod 2 & \text{if } |a_i| \equiv 1 \pmod 4 \end{cases},$$

which is equivalent to the condition $A^T b = c(A)$. Therefore,

$$|b| = |Ac(a)| = \left|\sum_{i=1}^{n} a_i c_i(A)\right| \equiv \sum_{i=1}^{n} c_i(A)|a_i| + 2\sum_{i<j} c_i(A) c_j(A)(a_i \cdot a_j) \equiv \sum_{i=1}^{n} c_i(A)|a_i| \equiv 3|c(a)| \pmod 4,$$

which implies that $|b| \equiv |c(A)| \pmod 4$. But we know by Theorem 20 that $|b| \equiv 0 \pmod 4$. So $|c(A)| \equiv 0 \pmod 4$, which completes the first part of the lemma.

We now need to show that any characteristic vector of Hamming weight divisible by 4 is realized by some matrix $A \in \langle \mathrm{T}_4 \rangle_n$. Notice that $c(\mathrm{T}_4) = (1,1,1,1)$. Therefore, by taking an appropriate tensor product of T_4 gates and permuting the rows and columns, we can achieve any characteristic vector of Hamming weight divisible by 4. ■

Corollary 86 $\# \langle \mathrm{F}_4 \rangle_n = \# \langle \mathrm{T}_4 \rangle_n$.

Proof. The condition $A^T b = c(A)$ in the proof of Lemma 85 implies that there is a unique vector $b = Ac(A)$ such that $f(x) = A(x) \oplus b$ is mod-4-preserving. ■

Combining Lemmas 84 and 85, we find that the number of representatives for the cosets of $\langle \mathrm{T}_6 \rangle_n$ in $\langle \mathrm{T}_4 \rangle_n$ equals the number of n-bit strings with Hamming weight 4. An explicit formula for this quantity is given by Knuth [16, p. 70]. This now completes the proof of Theorem 83.

Table 1 gives the number of generators of each class for $3 \le n \le 7$.

	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$
$\langle$Toffoli$\rangle$	37,980	20,919,528,228,864	2.6313×10^{35}	1.2689×10^{89}	3.8562×10^{215}
$\langle$Fredkin, NOT$\rangle$	480	1,625,691,648	4.3776×10^{26}	6.9238×10^{70}	1.6100×10^{178}
$\langle$Fredkin, NOTNOT$\rangle$	450	1,624,862,256	4.3776×10^{26}	6.9238×10^{70}	1.6100×10^{178}
$\langle \mathrm{C}_3 \rangle$	36	9,953,280	5.7818×10^{21}	2.9340×10^{60}	5.1283×10^{156}
$\langle \mathrm{C}_4 \rangle$	0	414,696	6.6368×10^{18}	5.1015×10^{53}	1.2863×10^{142}
$\langle \mathrm{C}_5 \rangle$	0	0	1.8962×10^{17}	1.0352×10^{50}	1.1760×10^{133}
$\langle \mathrm{C}_6 \rangle$	0	0	0	2.1567×10^{48}	4.4602×10^{128}
$\langle \mathrm{C}_7 \rangle$	0	0	0	0	7.0797×10^{126}
$\langle$Fredkin$\rangle$	30	414,696	1.8962×10^{17}	2.1567×10^{48}	7.0797×10^{126}
$\langle$CNOT$\rangle$	1152	301,056	309,657,600	1,269,678,735,360	20,807,658,944,593,920
$\langle$CNOTNOT, NOT$\rangle$	72	10,368	5,149,440	10,238,607,360	82,569,982,279,680
$\langle$CNOTNOT$\rangle$	72	10,368	5,149,440	10,238,607,360	82,569,982,279,680
$\langle \mathrm{T}_4$, NOT$\rangle$	0	192	9600	691,200	90,316,800
$\langle \mathrm{T}_4$, NOTNOT$\rangle$	0	144	8400	648,000	87,494,400
$\langle \mathrm{T}_6$, NOT$\rangle$	0	0	0	23,040	2,257,920
$\langle \mathrm{T}_6$, NOTNOT$\rangle$	0	0	0	22,320	2,222,640
$\langle \mathrm{T}_4 \rangle, \langle \mathrm{F}_4 \rangle$	0	24	600	21,600	1,411,200
$\langle \mathrm{T}_6 \rangle$	0	0	0	720	35,280
$\langle$NOT$\rangle$	24	192	1920	23,040	322,560
$\langle$NOTNOT$\rangle$	18	168	1800	22,320	317,520
$\langle \varnothing \rangle$	6	24	120	720	5040

Table 1: Number of n-bit generators for each reversible gate class.

Theorem 87 *The asymptotic size of each reversible gate class is as follows.*

$$\begin{aligned}
\log_2 \# \langle \text{Toffoli} \rangle_n &= n2^n - \frac{2^n}{\ln 2} + \frac{n}{2} + \frac{1}{2}\log_2 2\pi + O(2^{-n}) \\
\log_2 \# \langle \text{Fredkin}, \text{NOTNOT} \rangle_n &= n2^n - \frac{2^n}{\ln 2} - 2^n + n\log_2 n + \log_2 \pi + O(2^{-n}) \\
\log_2 \# \langle \text{Fredkin}, \text{NOT} \rangle_n &= \log_2 \# \langle \text{Fredkin}, \text{NOTNOT} \rangle_n + 1 \\
\log_2 \# \langle \text{C}_k \rangle_n &= n2^n - \frac{2^n}{\ln 2} - 2^n \log_2 k + o(2^n) \\
\log_2 \# \langle \text{Fredkin} \rangle_n &= n2^n - \frac{2^n}{\ln 2} - 2^n \log_2 \frac{\pi e\sqrt{n}}{2} + o(2^n) \\
\log_2 \# \langle \text{CNOT} \rangle_n &= n\left(n+1\right) - \alpha + O(2^{-n}) \\
\log_2 \# \langle \text{CNOTNOT}, \text{NOT} \rangle_n &= n\left(n-1\right) - \alpha + O(2^{-n}) \\
\log_2 \# \langle \text{CNOTNOT} \rangle_n &= \log_2 \# \langle \text{CNOTNOT}, \text{NOT} \rangle_n - 1 \\
\log_2 \# \langle \varnothing \rangle_n &= n\log_2 n - \frac{n}{\ln 2} + \frac{1}{2}\log_2 2\pi + O\!\left(\frac{1}{n}\right) \\
\log_2 \# \langle \text{T}_4 \rangle_n &= \frac{n(n-1)}{2} - \beta + O(2^{-n}) \\
\log_2 \# \langle \text{T}_6 \rangle_n &= \frac{n^2 - 3n + 4}{2} - \beta + O\!\left(2^{-n/2}\right),
\end{aligned}$$

where

$$\begin{aligned}
\alpha &= -\sum_{i=1}^{\infty} \log_2(1 - 2^{-i}) \approx 1.7919, \\
\beta &= -\sum_{i=1}^{\infty} \log_2(1 - 2^{-2i}) \approx 0.53839.
\end{aligned}$$

Recall that $\# \langle \text{F}_4 \rangle_n = \# \langle \text{T}_4 \rangle_n$*. The asymptotics of the remaining affine classes follow from the rules*

$$\begin{aligned}
\log_2 \# \langle G, \text{NOT} \rangle_n &= n + \log_2 \# \langle G \rangle_n\,, \\
\log_2 \# \langle G, \text{NOTNOT} \rangle_n &= n - 1 + \log_2 \# \langle G \rangle_n\,,
\end{aligned}$$

where $\langle G \rangle$ *is a linear class.*

Proof. Most of these results follow directly from Theorem 83 with liberal use of well-known logarithm properties, especially Stirling's approximation:

$$\log_2(m!) = m\log_2 m - \frac{m}{\ln 2} + \frac{1}{2}\log_2 2\pi m + O\left(\frac{1}{m}\right).$$

For the affine classes, we use the fact that

$$\begin{aligned}
\sum_{i=1}^{m} \log_2(2^i - 1) &= \frac{m(m+1)}{2} + \sum_{i=1}^{m} \log_2(1 - 2^{-i}) \\
&= \frac{m(m+1)}{2} - \alpha + O(2^{-m})
\end{aligned}$$

where $\alpha = -\sum_{i=1}^{\infty} \log_2 \left(1-2^{-i}\right)$. Note that $\alpha = -\log_2 (1/2;1/2)_\infty$ where $(1/2;1/2)_\infty$ is the *q-Pochhammer symbol*. Similarly, $\beta := -\sum_{i=1}^{\infty} \log_2 \left(1-2^{-2i}\right) = -\log_2 (1/4;1/4)_\infty$ differs from the m^{th} partial sum by $O(2^{-2m})$.

It turns out that the even and odd cases of $\# \langle \mathrm{T}_4 \rangle$ have the same asymptotic behavior, and similarly for the four cases of $\# \langle \mathrm{T}_6 \rangle$.

However, there are two special cases that require extra care: $\langle \mathrm{C}_k \rangle$ (for $k \geq 3$) and $\langle \mathrm{Fredkin} \rangle$. Recall that

$$\# \langle \mathrm{C}_k \rangle_n = \prod_{i=0}^{k-1} a_i!.$$

where we define $a_i = \sum_{j \equiv i (\mathrm{mod}\, k)} \binom{n}{j}$. Clearly $a_i = \frac{2^n}{k}(1+o(1))$. Then Stirling's approximation gives

$$\begin{aligned}
\log_2 \# \langle \mathrm{C}_k \rangle_n &= \sum_{i=0}^{k-1} \left(a_i \log_2 a_i - \frac{a_i}{\ln 2} + o(a_i) \right) \\
&= \sum_{i=0}^{k-1} \left(a_i \log_2 \frac{2^n}{k} + a_i \log_2 (1+o(1)) - \frac{a_i}{\ln 2} + o(a_i) \right) \\
&= n2^n - \frac{2^n}{\ln 2} - 2^n \log_2 k + o(2^n).
\end{aligned}$$

For $\langle \mathrm{Fredkin} \rangle$, we use the fact that if x is a uniformly-random n-bit string, then the entropy of $|x|$ is

$$\frac{1}{2} \log_2 \frac{\pi e n}{2} + O\left(\frac{1}{n}\right) = -\sum_{i=0}^{n} 2^{-n} \binom{n}{i} \log_2 \left(2^{-n} \binom{n}{i} \right).$$

One can show this by approximating the binomial with a Gaussian distribution. Rearranging gives us

$$\sum_{i=0}^{n} \binom{n}{i} \log_2 \binom{n}{i} = n2^n - 2^n \log_2 \frac{\pi e \sqrt{n}}{2} - O\left(\frac{2^n}{n}\right).$$

Now we can apply Stirling's approximation to $\# \langle \mathrm{Fredkin} \rangle_n$, as calculated in Theorem 83:

$$\begin{aligned}
\log_2 \# \langle \mathrm{Fredkin} \rangle_n &= \sum_{i=0}^{n} \left[\binom{n}{i} \log_2 \binom{n}{i} - \binom{n}{i} + o\left(\binom{n}{i} \right) \right] \\
&= n2^n - \frac{2^n}{\ln 2} - 2^n \log_2 \frac{\pi e \sqrt{n}}{2} - o(2^n).
\end{aligned}$$

■

One can clearly see "the pervasiveness of universality" in Table 1: within almost every class, the gates that are universal for that class quickly come to dominate the gates that are not universal for that class in number. Theorem 87 lets us make that observation rigorous.

Corollary 88 *Let $\mathcal{C}$ be any reversible gate class, and let G be an n-bit gate chosen uniformly at random from $\mathcal{C}$. Then*

$$\Pr[G \text{ generates } \mathcal{C}] = 1 - O\left(2^{-n}\right),$$

*unless $\mathcal{C}$ is one of the "*NOT *classes" ($\langle$*Fredkin, NOT*$\rangle$, $\langle$*F$_4$, NOT*$\rangle$, $\langle$*T$_6$, NOT*$\rangle$, or $\langle$*NOT*$\rangle$), in which case*

$$\Pr\left[G \textit{ generates } \mathcal{C}\right] = \frac{1}{2} - O\left(2^{-n}\right).$$

15 Appendix: Alternate Proofs of Theorems 12 and 19

Alternate Proof of Theorem 12. Suppose $j \not\equiv 0 \,(\operatorname{mod} k)$, and let q be j's order mod k (that is, the least positive i such that $ij \equiv 0 \,(\operatorname{mod} k)$). We first show that q must be a power of 2. For $i \in \{0, \ldots, q-1\}$, let S_i be the set of all $x \in \{0,1\}^n$ whose Hamming weight satisfies $|x| \equiv ij \,(\operatorname{mod} k)$. Let q be the number of distinct S_i's. Now, since the gate G maps everything in S_i to $S_{(i+1) \operatorname{mod} q}$, we have

$$|S_0| = \cdots = |S_{q-1}| = \frac{2^n}{q}.$$

But the above must be an integer.

Observe that, if there existed a G such that $|G(x)| \equiv |x| + j \,(\operatorname{mod} k)$, where j's order mod k was any positive power of 2 (say 2^p), then the iterated map $G^{2^{p-1}}$ would satisfy

$$\left|G^{2^{p-1}}(x)\right| \equiv |x| + \frac{k}{2} \,(\operatorname{mod} k),$$

and so would have order *exactly* 2 mod k. For that reason, it suffices to rule out, for all $k \geq 2$ and all n, the possibility of a reversible transformation G that satisfies

$$|G(x)| \equiv |x| + k \,(\operatorname{mod} 2k)$$

for all $x \in \{0,1\}^n$.

To do the above, it is necessary and sufficient to show that there is a "cardinality obstruction" to any G of the required form. In other words, for all $j \in \{0, \ldots, 2k-1\}$, let

$$A_{n,j} := \{x \in \{0,1\}^n : |x| \equiv j \,(\operatorname{mod} 2k)\}$$

be the set of n-bit strings of Hamming weight j mod $2k$. Then the problem boils down to showing that for all $k \geq 2$ and n, there exists a $j < k$ such that $|A_{n,j}| \neq |A_{n,j+k}|$—and therefore, that no mapping from $A_{n,j}$ to $A_{n,j+k}$ (or vice versa) can be reversible.

This, in turn, can be interpreted as a statement about binomial coefficients: for all $k \geq 2$ and all n, there exists a j such that

$$\sum_{i=j,j+2k,j+4k,\ldots} \binom{n}{i} \neq \sum_{i=j,j+2k,j+4k,\ldots} \binom{n}{i+k}.$$

A nice way to prove the above statement is by using what we call the *wraparound Pascal's triangle of width* $2k$: that is, Pascal's triangle with a periodic boundary condition. This is simply an iterative map on row vectors $(a_0, \ldots, a_{2k-1}) \in \mathbb{Z}^{2k}$, obtained by starting from the row $(1, 0, \ldots, 0)$, then repeatedly applying the update rule $a_i' := a_i + a_{(i-1) \operatorname{mod} 2k}$ for all i. So for example, when

$2k = 4$ we obtain

$$\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 2 & 1 & 0 \\ 1 & 3 & 3 & 1 \\ 2 & 4 & 6 & 4 \\ 6 & 6 & 10 & 10 \\ 16 & 12 & 16 & 20 \\ \vdots & \vdots & \vdots & \vdots \end{array}$$

It is not hard to see that the i^{th} entry of the n^{th} row of the above "triangle," encodes $|A_{n,i}|$: that is, the number of n-bit strings whose Hamming weights are congruent to i mod $2k$.

So the problem reduces to showing that, when $k \geq 2$, no row of the wraparound Pascal's triangle of width $2k$ can have the form

$$(a_0, \ldots, a_{k-1}, a_0, \ldots, a_{k-1}).$$

That is, no row can consist of the same list of k numbers repeated twice. (Note we *can* get rows that satisfy $a_i = a_{i+k}$ for *specific* values of i: to illustrate, in the width-4 case above, we have $a_1 = a_3 = 4$ in the fifth row, and $a_0 = a_2 = 16$ in the seventh row. But we need to show that no row can satisfy $a_i = a_{i+k}$ for all $i \in \{0, \ldots, k-1\}$ simultaneously.) We prove this as follows.

Notice that the update rule that defines the wraparound Pascal's triangle, namely $a'_i := a_i + a_{(i-1) \bmod 2k}$, is just a linear transformation on $\mathbb{R}^{2k}$, corresponding to a $2k \times 2k$ band-diagonal matrix M. For example, when $k = 2$ we have

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

Notice further that $\operatorname{rank}(M) = 2k - 1$. The image of M is a $(2k-1)$-dimensional subspace $P \leq \mathbb{R}^{2k}$ (the "parity-respecting subspace"), which is defined by the linear equation

$$a_0 + a_2 + \cdots + a_{2k-2} = a_1 + a_3 + \cdots + a_{2k-1}.$$

Thus, M acts invertibly, as long we restrict to vectors in P.

Next, let $D \leq \mathbb{R}^{2k}$ (the "duplicate subspace") be the k-dimensional subspace defined by the k linear equations

$$a_0 = a_k, \ldots, a_{k-1} = a_{2k-1}.$$

Then let $S = P \cap D$ be the $(k-1)$-dimensional intersection of the parity-respecting and duplicate subspaces.

Observe that S is an invariant subspace of M: that is, if $x \in S$, then $Mx \in S$. But now, using the fact that M acts invertibly within P, this means that the converse also holds: namely, if $x \in P \setminus S$, then $Mx \in P \setminus S$. In other words: as we generate more and more rows of the wraparound Pascal's triangle, if we're not *already* in S by the second row (i.e., after the first time we've applied M), then we're never going to get into S.

Now, the first row of the wraparound Pascal's triangle is $(1, 0, \ldots, 0)$, and the second row is $(1, 1, 0, \ldots, 0)$. This second row is not in S unless $k = 1$. ■

Alternate Proof of Theorem 19. We will actually prove a stronger result, that if G is any nontrivial *affine* gate that preserves Hamming weight mod k, then either $k = 2$ or $k = 4$. We have $G(x) = Ax \oplus b$, where A is an $n \times n$ invertible matrix over $\mathbb{F}_2$, and $b \in \mathbb{F}_2^n$. Since G is nontrivial, Lemma 18 implies that at least one of A's column vectors $v_1, \ldots, v_n$ must have Hamming weight at least 2; assume without loss of generality that v_1 is such a column. Notice that $|G(0^n)| \equiv |b| \equiv 0 \pmod{k}$, while

$$|G(e_1)| \equiv |v_1 \oplus b| \equiv 1 \pmod{k}$$

Clearly $|G(e_1)| \equiv |e_1| \equiv 1 \pmod{k}$. Let y be an n-bit string whose first bit is 0. Then by Lemma 17, we have

$$\begin{aligned} 1 + |y| &\equiv |e_1 \oplus y| \\ &\equiv |G(e_1 \oplus y)| \\ &\equiv |G(e_1) \oplus G(y) \oplus b| \\ &\equiv |Ae_1 \oplus b \oplus b \oplus G(y)| \\ &\equiv |v_1 \oplus G(y)| \\ &\equiv |v_1| + |G(y)| - 2(v_1 \cdot G(y)) \\ &\equiv |v_1| + |y| - 2(v_1 \cdot G(y)) \pmod{k}. \end{aligned}$$

Thus

$$2(v_1 \cdot G(y)) \equiv |v_1| - 1 \pmod{k}.$$

Note that the above equation must hold for all 2^{n-1} possible y's that start with 0. Such y's, of course, account for half of all n-bit strings. So we deduce that

$$\Pr_{x \in \{0,1\}^n} [2(v_1 \cdot x) \equiv |v_1| - 1 \pmod{k}] \geq \frac{1}{2}.$$

Equivalently, if we let S be the set of all $x \in \{0,1\}^{|v_1|}$ such that $2|x| \equiv |v_1| - 1 \pmod{k}$, then we find that

$$\Pr_{x \in \{0,1\}^{|v_1|}} [x \in S] \geq \frac{1}{2}, \tag{8}$$

or $|S| \geq 2^{|v_1|-1}$. But we will prove this impossible.

First suppose k is even. Then for the inequality (8) to have any chance of being satisfied, $|v_1|$ needs to be odd, so assume it is. Then S equals the set of all $x \in \{0,1\}^{|v_1|}$ such that

$$|x| \equiv 0 \left(\operatorname{mod} \frac{k}{2}\right). \tag{9}$$

If $k = 2$, then $|x| \equiv 0 \pmod{1}$ holds for all x, while if $k = 4$, then $|x| \equiv 0 \pmod{2}$ holds whenever $|x|$ is even. In either case, (8) is satisfied. On the other hand, suppose $k \geq 6$. Then we claim that (8) cannot hold: in other words, that $|S| < 2^{|v_1|-1}$. To prove this, let

$$S' := \{x \oplus e_1 : x \in S\}$$

contain, for each $x \in S$, the string x' obtained by flipping the first bit of x. Then clearly $|S| = |S'|$, and S and S' are disjoint (since no two elements of S are neighbors in the Hamming cube). So it suffices to show that $S \cup S'$ still does not cover all of $\{0,1\}^{|v_1|}$. Since $\frac{k}{2} \geq 3$, observe that S' can contain at most one string of Hamming weight 1, namely $x' = 10\cdots 0$ (the neighbor of $x = 0^{|v_1|}$). But since $|v_1| \geq 2$, there are other strings of Hamming weight 1, not included in S'. Hence $S \cup S' \neq \{0,1\}^{|v_1|}$.

Next suppose $k \geq 3$ is odd. Then first, we claim that we cannot have $|v_1| = 2$. For suppose we did. Then $|b \oplus v_1|$ would be either $|b|$, or $|b| - 2$, or $|b| + 2$. But this contradicts the facts that $|b| \equiv 0 \pmod{k}$, while $|b \oplus v_1| \equiv 1 \pmod{k}$. Since $|v_1| \neq 1$, this means that $|v_1| \geq 3$. But in that case, we can use a similar argument as before to show that (8) cannot hold, and that $|S| < 2^{|v_1|-1}$. Letting S' be as above, we again have that $|S| = |S'|$, and that S and S' are disjoint. And we will again show that $S \cup S'$ fails to cover all of $\{0,1\}^{|v_1|}$. Notice that, since the Hamming weights of the S elements are separated by $k \geq 3$, every S' element that is "below" an S element must start with 0, and every S' element that is "above" an S element must start with 1. Also, since $|v_1| \geq 3$, there must be some $x' \in S'$ with a Hamming weight that is neither maximal nor minimal (that is, neither $|v_1|$ nor 0). But since the first bit of x' has a fixed value, not all strings of Hamming weight $|x'|$ can belong to S'. Hence $S \cup S' \neq \{0,1\}^{|v_1|}$, and $|S| < 2^{|v_1|-1}$. ■