Transformation Rules for Designing CNOT-based Quantum Circuits

Kazuo Iwama
Sch. of Inform., Kyoto Univ.
QCI, ERATO, JST
iwama@kuis.kyoto-u.ac.jp

Yahiko Kambayashi
Sch. of Inform., Kyoto Univ.
yahiko@i.kyoto-u.ac.jp

Shigeru Yamashita
NTT Communication Science Labs.
QCI, ERATO, JST
ger@cslab.kecl.ntt.co.jp

ABSTRACT

This paper gives a simple but nontrivial set of local transformation rules for *Control-NOT*(*CNOT*)-based combinatorial circuits. It is shown that this rule set is complete, namely, for any two equivalent circuits, S_1 and S_2, there is a sequence of transformations, each of them in the rule set, which changes S_1 to S_2. Our motivation is to use this rule set for developing a design theory for *quantum circuits* whose Boolean logic parts should be implemented by CNOT-based circuits. As a preliminary example, we give a design procedure based on our transformation rules which reduces the cost of CNOT-based circuits.

Categories and Subject Descriptors

B.6.m [**LOGIC DESIGN**]: Miscellaneous

General Terms

Design, Theory

Keywords

Quantum Circuit, CNOT Gate, Local Transformation Rules

1. INTRODUCTION

It is widely considered that logic synthesis is a mature field in our community. However, this is only true for conventional AND-OR-NOT-based circuits or LSI's; new ideas must be needed if we face technology innovations. The main purpose of this paper is to introduce logic synthesis for *quantum Boolean circuits* [12] (QBCs for short). The key difference between conventional circuits and quantum ones is their base-family of logic gates, for the latter of which many people agree that *Control-Not (CNOT)* type logic gates will be a single possibility. Another (even more important) feature of QBCs is its severe restriction against wire-linking between gates: As we can see in a moment, only straight-line, parallel wiring is allowed. Thus, it looks obviously hard to apply conventional logic synthesis techniques for our present purpose.

In spite of such a new target, our basic strategy approaching to it is quite conservative. Namely, our logic synthesis proposed in this paper is based on *local transformations*, which has been constantly popular and successful for conventional circuits [2, 3]. (One can think of, for example, the DeMorgan's law, which has been used most often to make a local simplification of Boolean circuits.) More concretely, we give a set of local transformation rules. The rule set is complete, which means we can transform any QBC into any of its equivalent ones by applying these rules. We also make some concrete suggestions on how to use these transformation rules to simplify QBCs.

It should be noted that quantum algorithms are often described by using QBCs [12]. Designing a "good" QBC is thus plays a key role to the successful implementation of a quantum algorithm. A little surprisingly, however, relatively small attention has been paid for the design methodology of QBCs [1, 9]. [1] shows that any unitary transformation can be broken down into a sequence of basic quantum gates. The method is general, but it does not necessarily generate efficient circuits. [9] gives how to construct QBCs for Boolean functions by using CNOT gates but their resulting circuits are essentially the same as the elementary two-level AND-OR-NOT circuits.

To discuss how to design small QBCs, we should note that QBCs for quantum algorithms can be divided into two parts: One is a quantum specific part, for example, the Walsh-Hadamard transformation for making a quantum superposition and the Quantum Fourier transformation [6]. The other part, which is sometimes called *quantum Boolean oracles* [5], is for calculating (conventional) Boolean functions. To establish an efficient design methodology for QBCs, it is much more important to target the latter part (quantum Boolean oracles) since the structures of the latter part vary depending on each problem, whereas the structure of former part is usually fixed. Our ultimate goal is to develop a design theory for quantum Boolean oracles, and it is a nice start-up to have a set of local transformation rules for CNOT-based QBCs. It should be noted that a similar set of transformation rules for conventional circuits is given in [8].

We also introduce the canonical form, which is another fundamental concept in logic synthesis, for CNOT-based QBCs. Then we can prove the *completeness* of the rule set only by showing that there is a sequence of transformations from any circuit to its canonical form, since each transformation is bidirectional. Accordingly, we can assure that by using our transformation rules, we can modify a given circuit into another with a desirable property (e.g., of small cost) by executing a NP-type search, although the length of the search path might not be always short. The completeness of the rule set is quite non-trivial and theoretically interesting on its own. As mentioned before, however, our final goal is to use our rules for more practical purposes. Actually, we give a preliminary example of a design procedure which simplifies CNOT-based circuits.

2. QUANTUM BOOLEAN CIRCUITS

Although we need virtually no knowledge of quantum computation to understand the main result of this paper, let us start with a minimum introduction of the new technology: In quantum computing, a *quantum bit*, or *qubit* for short, plays an important role. There have been proposed various physical implementation objects for qubits [10], and they can be described as: $|x\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|x\rangle$ means the quantum state of the qubit x, and α and β are

DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.

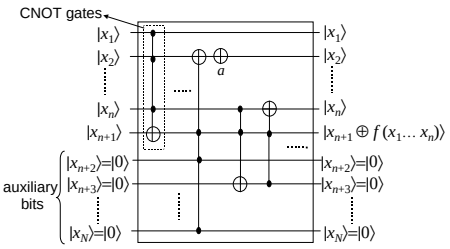

Figure 1: A Quantum Boolean Circuit

complex numbers which satisfy that (1) $|\alpha|^2+|\beta|^2=1$ and (2) if we *measure* the quantum state, we get the states $|0\rangle$ and $|1\rangle$ with the probability $|\alpha|^2$ and $|\beta|^2$, respectively. This means that a qubit can store a *superposition* of the states $|0\rangle$ and $|1\rangle$ unlike a conventional bit. In the general framework of quantum computing, we apply a specific sequence of *unitary* operations to qubits, and get a desired solution by measuring the final quantum states of the qubits. Such a sequence of unitary operations is often described as a *quantum Boolean circuit* (QBC).

A QBC is a quantum system with N qubits, denoted by $|x_1\rangle|x_2\rangle\cdots|x_N\rangle$, as illustrated in Fig. 1 where we apply a specific unitary operation corresponding to each gate one by one to the qubits $|x_1\rangle$ to $|x_N\rangle$ from left to right. As an interaction of their qubits, we can only use CNOT gates whose functionality will be given later. The left-side $|x_1\rangle|x_2\rangle\cdots|x_n\rangle$ are used for the input, and their values should be restored finally. (This restoring is important when the circuit is used for quantum Boolean oracles as we mention later.) The $(n+1)$-st qubit $|x_{n+1}\rangle$, called a *work bit*, is changed into $|x_{n+1}\oplus f(x_1,\cdots,x_n)\rangle$, which is used to obtain the value of the Boolean function f. Furthermore, a circuit can use any finite number of *auxiliary qubits* which are reset to $|0\rangle$ initially and also restored to be $|0\rangle$ finally. More formally:

Definition 1. A *Control-NOT (CNOT) gate* is denoted by $[t,C]$, where t is an integer and C is a finite set of integers ($t\notin C$). $|x_t\rangle$ is called a *target bit* and $|x_k\rangle$ is called a *control bit* if $k\in C$.

This kind of gates is also called n-bit Toffoli gates [11] in *reversible computing*. Also the literature refers to $[t,C]$ with only one control bit as a Control-NOT gate, and to $[t,C]$ with two control bit as *Control-Control-NOT gate*, and so on, but we simply call both of them CNOT gates in this paper. In the figure, we use $\oplus$ for a target bit and $\cdot$ for a control bit. For example, the leftmost CNOT gate in Fig. 1 is given by $[n+1,\{1,2,n\}]$. As its symbol suggest, $[n+1,\{1,2,n\}]$ changes the state of $|x_{n+1}\rangle$ into $|x_{n+1}\oplus x_1\cdot x_2\cdot x_n\rangle$, where $\oplus$ is the conventional XOR. (See Definition 3 for details.)

We denote the set of N-bit basis vectors by $\{0,1\}^N$. The quantum state of N qubits $|x_1\rangle\cdots|x_N\rangle$ is a superposition (a linear combination) of those 2^N basis vectors. However, we often assume in this paper that the state is a single basis vector when describing the behavior of the system. Generalization to superposed states can be done simply by taking a linear combination of the results for each basis vector.

Definition 2. A *quantum Boolean circuit* of size M over qubits $|x_1\rangle,\cdots,|x_N\rangle$ is a sequence of CNOT gates $[t_1,C_1]\cdots[t_i,C_i]\cdots[t_M,C_M]$, where $1\le t_i\le N, C_i\subseteq\{1,\cdots,N\}$. Note that unlike the conventional Boolean circuits we are only allowed to have a sequence of CNOT gates. (We cannot use so-called jump-wires which propagate the result of one gate to another gate placed far away. This is an important difference between our circuit model and the conventional circuit model.)

Definition 3. The state of the circuit after the i-th CNOT gate $[t_i,C_i]$ is denoted by $S_i=|x_1^i\rangle\cdots\cdots|x_N^i\rangle$ and defined as follows:

(1) $S_0=|a_1\rangle|a_2\rangle\cdots|a_N\rangle$, where $|a_1\rangle|a_2\rangle\cdots|a_N\rangle\in\{0,1\}^N$ is an input state.

(2) For $1\le k\le N$ and $1\le i\le M$, $|x_k^i\rangle=|x_k^{i-1}\rangle$ if $k\neq t_i$, and $|x_{t_i}^i\rangle=|x_{t_i}^{i-1}\oplus X_{C_i}^k\rangle$ where $X_{C_i}^k=1\wedge x_{i_1}\wedge\cdots\wedge x_{i_j}$ if $C_i=\{i_1,\cdots,i_j\}$. Here $\wedge$ is the conventional AND and $\oplus$ is the conventional XOR operators.

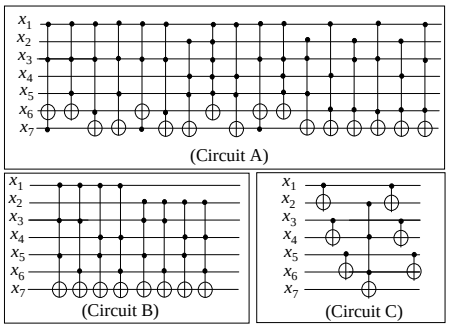

Figure 2: Equivalent Circuits

For example, we can calculate S_2 of Circuit A in Fig. 2 such that $|x_i^2\rangle=|x_i^1\rangle$ for $i\neq 6$, and $|x_6^2\rangle=|x_6^1\oplus x_1^1\cdot x_3^1\cdot x_5^1\rangle$. Note that $X_{C_i}^k$ is 1 when C_i is an empty set, which means that the CNOT gate operates just as a negation (see gate a in Fig. 1). For better exposition, we often say "the state of $|x_1\rangle$ after the i-th gate" in stead of "the state of $|x_1^i\rangle$."

Definition 4. A quantum Boolean circuit is said to be *proper* and to *compute* a Boolean function $f(x_1,\cdots,x_n)$ iff (i) $S_0=|a_1\rangle|a_2\rangle\cdots|a_{n+1}\rangle|0\rangle\cdots|0\rangle$, i.e., all the auxiliary bits are initially cleared, (ii) The final state $S_M=|a_1\rangle|a_2\rangle\cdots|a_{n+1}\oplus f(x_1,\cdots,x_n)\rangle|0\rangle\cdots|0\rangle$, i.e., the $(n+1)$-st state is XORed with the function value and all the others are the same as the input.

Remark. Note that our condition says that all auxiliary qubits must be reset finally. Otherwise, we cannot use the circuit as a Boolean oracle for the following reason: If the auxiliary qubits are not reset, the states of $|x_1\rangle\cdots|x_n\rangle$ may be *entangled* with some states of the auxiliary qubits. In such a case, we can no longer treat the states of $|x_1\rangle\cdots|x_n\rangle$ as one quantum system, and therefore quantum operations for the states of $|x_1\rangle\cdots|x_n\rangle$ might not work as desired. For example, the *f-controlled phase shift* used in the Grover's search algorithm [5] is defined as $|x_1\rangle\cdots|x_n\rangle|x_{n+1}\rangle=(-1)^{f(x_1,\cdots,x_n)}|x_1\rangle\cdots|x_n\rangle|x_{n+1}\rangle$ where $|x_{n+1}\rangle$ is initialized to $\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$. If the circuit is not proper, we can no longer use the circuit as a primitive of the above unitary operation.

Fig. 2 shows three QBCs which are all equivalent, i.e., they compute the same Boolean function $f(x_1,\cdots,x_6)=(x_1\oplus x_2)\cdot(x_3\oplus x_4)\cdot(x_5\oplus x_6)$. Note that x_7 is a work bit in these circuits. As will be shown later, we can always construct a circuit by only using CNOT gates whose target bit is x_{n+1}, i.e., those like Circuit B. However, different types of circuits are of course possible like Circuits A and C, where C is simpler than A and B.

3. TRANSFORMATION RULES

In this section, we introduce six *local transformation rules* which can be applied for a sequence of CNOT gates. Each transformation rule looks like $F\Leftrightarrow G$, where F and G are sequences of CNOT gates. If a (proper) circuit A is written as $A=A_1FA_2$, i.e., it includes a subsequence F, then A changes into $A'=A_1GA_2$ by applying the rule $F\Leftrightarrow G$.

Transformation Rule Set. In the followings, ε means the empty sequence, and we refer to a CNOT gate whose target bit is the i-th bit as $CNOT_i$.

(1) $[t_1,C_1]\cdot[t_1,C_1]\Leftrightarrow\varepsilon$. Namely, two adjacent identical gates cancel.

(2) $[t_1,C_1]\cdot[t_2,C_2]\Leftrightarrow[t_2,C_2]\cdot[t_1,C_1]$, if $t_1\notin C_2$ and $t_2\notin C_1$. The condition means that the two gates are "independent." If there is some influence between two gates, we cannot simply change the order of the two gates. Even in such cases, we can change the order by adding some gates, as we will see in the followings.

(3) $[t_1,C_1]\cdot[t_2,C_2]\Leftrightarrow[t_2,C_2]\cdot[t_1,C_1]\cdot[t_1,C_1\cup C_2-\{t_2\}]$, if $t_1\notin C_2$ and $t_2\in C_1$ (see Fig. 3).

(4) $[t_1,C_1]\cdot[t_2,C_2]\Leftrightarrow[t_2,C_1\cup C_2-\{t_1\}]\cdot[t_2,C_2]\cdot[t_1,C_1]$, if $t_1\in$

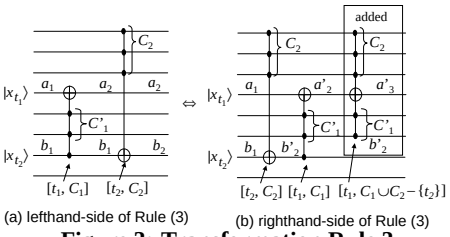

Figure 3: Transformation Rule 3

C_2 and $t_2 \notin C_1$. This rule is the dual of (3), i.e., the relationship between the two gates is opposite.

(5) $[t_1,\{c_1\}]\cdot[t_2,C_2\cup\{c_1\}] \Leftrightarrow [t_1,\{c_1\}]\cdot[t_2,C_2\cup\{t_1\}]$, if $(t_1 > n+1)$ and there is no $CNOT_{t_1}$ before $[t_1,\{c_1\}]$ (see Fig. 4).

(6) $[t,C] \Leftrightarrow \varepsilon$, if there is an integer i such that $i \in C, i > n+1$, and there is no $CNOT_i$ before $[t,C]$.

Lemma 1. Applying any one of Rules (1) to (6) does not change the Boolean function computed by the circuit.

Proof. (1) Rule (1) is obvious since $g\oplus f\oplus f = g$ for any Boolean functions f and g.

(2) Rule (2) is also easy because $[t_1,C_1]$ and $[t_2,C_2]$ do not affect each other if $t_1 \notin C_2$ and $t_2 \notin C_1$.

(3) See Fig. 3 again. We have two important qubits, i.e., $|x_{t_1}\rangle$ and $|x_{t_2}\rangle$ ($t_1 \neq t_2$ by the condition). In Fig. 3 (a), let a_1 and a_2 be the states of $|x_{t_1}\rangle$ before and after the gate $[t_1,C_1]$, respectively. Let a_1, a'_2 and a'_3 be the states of $|x_{t_1}\rangle$ before $[t_2,C_2]$, after $[t_1,C_1]$ and after $[t_1,C_1\cup C_2-\{t_2\}]$, respectively, in Fig. 3 (b). Also, b_1, b_2 and b'_2 are the states of $|x_{t_2}\rangle$ similarly defined. Furthermore let $C'_1 = C_1-\{t_2\}$ (C'_1 and C_2 may not be disjoint). Note that the state whose index is in $C'_1\cup C_2$ does not change throughout this portion of the circuit since there is no target bit index in $C'_1\cup C_2$. Now let us calculate the states a_2, b_2, a'_3 and b'_2. First, both b_2 and b'_2 can be written as $b_1\oplus X_{C_2}$, where X_{C_2} is a product term of all x_j such that $j\in C_2$. Similarly, let $X_{C'_1}$ be a product term of all x_j such that $j\in C'_1$, then we have $a_2 = a_1\oplus X_{C'_1}\cdot b_1$, $a'_2 = a_1\oplus X_{C'_1}\cdot b'_2 = a_1\oplus X_{C'_1}\cdot(b_1\oplus X_{C_2}) = a_1\oplus X_{C'_1}\cdot b_1\oplus X_{C_2}\cdot X_{C'_1}$ and $a'_3 = a'_2\oplus X_{C_2}\cdot X_{C'_1} = a_1\oplus X_{C'_1}\cdot b_1\oplus X_{C_2}\cdot X_{C'_1}\oplus X_{C_2}\cdot X_{C'_1} = a_1\oplus X_{C'_1}\cdot b_1$. Thus $b_2 = b'_2$ and $a_2 = a'_3$. The states of other qubits do not change as mentioned before and hence the transformation does not change the functionality of the circuit.

(4) Similar to the proof for Rule (3).

(5) If there is no $CNOT_{t_1}$ before $[t_1,\{c_1\}]$, $|x_{t_1}\rangle$ remains $|0\rangle$ just before the gate $[t_1,\{c_1\}]$. Therefore, after applying $[t_1,\{c_1\}]$, the state of $|x_{t_1}\rangle$ and that of $|x_{c_1}\rangle$ must be the same, which means that we can change t_1 to c_1 in the index set of the latter gate's control bits.

(6) If there is an integer i which satisfies the condition of the rule, $[t,C]$ has an auxiliary bit $|x_i\rangle$ as its control bit whose current state remains $|0\rangle$. Since one of its control bits is always 0, this CNOT gate is useless. □

4. COMPLETENESS OF THE RULE SET

In this section, we first introduce the canonical form for quantum Boolean circuits. Then it is shown that any circuit can be transformed into its canonical form using the transformation rules. The completeness of the rule set is its immediate consequence, since our transformation rules are bidirectional.

4.1 Canonical Form

Definition 4. A quantum Boolean circuit S is said to be *of the canonical form*, if (1) it has only $CNOT_{n+1}$ gates, namely CNOT gates whose target bit is the work bit, (2) it does not include two or more same $CNOT_{n+1}$ gates, (3) $CNOT_{n+1}$ gates are ordered lexicographically in terms of the indices of their control bits, and (4) no $CNOT_{n+1}$ gates have auxiliary qubits in their control bits, i.e., the circuit S uses no auxiliary qubits at all.

The condition (2) means that the canonical form must not be redundant in terms of Rule (1) in our transformation rule set. The condition (3) means that, for example, $[n+1,\{1,2,3\}]$ should be placed before $[n,\{2,3,4\}]$. Recall that Fig. 2 shows three different circuits computing the same Boolean function, and only Circuit B is of the canonical form. Now consider Boolean formulas of the following form: $a_0\oplus a_1x_1\oplus a_2x_2\oplus\cdots\oplus a_nx_n\oplus a_{1,2}x_1x_2\oplus a_{1,3}x_1x_3\oplus\cdots\oplus a_{n-1,n}x_{n-1}x_n\oplus\cdots\cdots\oplus a_{1,2,\cdots n-1,n}x_1x_2\cdots x_{n-1}x_n$, where no negated literals are allowed and each a_i is 0 or 1. This form is called a *positive polarity Reed-Muller expression*, for which the following lemma is known:

Lemma 2. [4]. Any Boolean function can be expressed by a positive polarity Reed-Muller expression which is unique except for the order of terms.

Lemma 3. Any CNOT-based quantum circuit S has its unique canonical form.

Proof. By definition, S transforms $|x_i\rangle$ to $|x_i\rangle$ for $1\le i\le n$ and $|x_{n+1}\rangle$ to $|x_{n+1}\oplus f(x_1,\cdots,x_n)\rangle$ for some formula f. Compute the positive polarity Reed-Muller expression of f, which naturally corresponds to a sequence of $CNOT_{n+1}$ gates. Since the order of such gates must be lexicographic with respect to their control bits in the canonical form, the resulting circuit is unique by Lemma 2. □

4.2 Transformation Procedure

Theorem 1. Any quantum circuit S can be transformed into its canonical form by using our transformation rules.

Proof. Our procedure, $Main(S)$, for such a transformation is illustrated in Fig. 8. $Main(S)$ first calls $Shift(S,n+1)$ which "shifts" all the $CNOT_{n+1}$ gates in S to the left portion of the circuit (see below for details). After all the $CNOT_{n+1}$ gates are placed in the left portion of the circuit, we let the remaining part be S' and then apply $Shift(S',n)$. Repeat this procedure until $Shift(S',1)$. As one can see later, the *Shift* procedure also includes the lexicographically reordering operation and the operation of deleting redundant gates. Therefore, after $Shift(S',1)$ is finished, all $CNOT_m$ for $1\le m\le n$ have already vanished (see Lemma 4). Thus all we have to do more in line 9-15 of $Main(S)$ is to delete the gates whose target bits are auxiliary qubits.

We first concentrate ourselves on the *Shift* procedure. Let $k = [t_1,C_1]$ and $h = [t_2,C_2]$ be two consecutive gates, where $t_2 = n+1$ and $t_1 \neq n+1$. Then our basic idea is to switch the positions of k and h, i.e., kh into hk. If $t_1\notin C_2$ or $t_2\notin C_1$, then this switch can be done in a single step by using one of Rules (2)-(4). Unfortunately, that is not possible if $t_1\in C_2$ and $t_2\in C_1$; the following algorithm mainly targets this situation where we can no longer depend on a simple repetition of the above switching operations. Now here are details of the procedure $Shift(S,m)$ for $1\le m\le n+1$:

Step 1. Suppose that the entire circuit S includes k CNOT gates whose control bits include the m-th bit. Let these gates be $g_1,g_2,\cdots g_k$, and $g_j = [t_j,C'_j\cup\{m\}]$. What we do first is to remove this m-th bit from the control bits of g_j by using an unused auxiliary bit, say the a_j-th bit. As an illustration, see Fig. 5 for the original circuit S and Fig. 7 for the circuit after Step 1 of $Shift(S,3)$ is applied. In more detail:

Step 1.1. We add two identical gates $[a_j,\{m\}]$, right before g_j by Rule (1). Let these gates be g_{l_j} for the left one and g_{r_j} for the right one.

Step 1.2. We change the order of g_{r_j} and g_j by Rule (2), i.e., to the order of $g_{l_j}g_jg_{r_j}$.

Step 1.3. We apply Rule (5) to g_{l_j} and g_j so that $g_j = [t_j,C'_j\cup\{m\}]$ is changed to $[t_j,C'_j\cup\{a_j\}]$.

Step 2. We move each $g_{l_j}(1\le j\le k)$ to the left as far as possible. Note that we can always do this transformation by applying Rules (2) to (4). We need to add a gate whose target bit is the a_j-th bit $(1\le j\le k)$ when we apply Rule (4). We also move such added gates to the left part of the circuit by Rules (2) to (4). (One might have the concern that this chain-like addition does not stop.

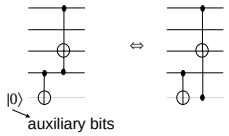

Figure 4: Transformation Rule 5

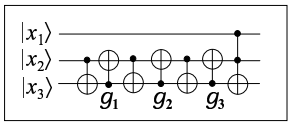

Figure 5: Initial Circuit

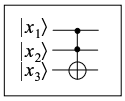

Figure 6: The Final Result

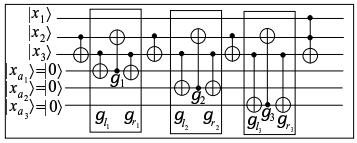

Figure 7: The Circuit after Step 1

```
1   Main(S)
2   {
3       S' = S
4       for (i = n + 1 to 1) {
5           Shift(S', i)
6           C_i = S_L /* the left part of the result of Shift(S', i) which consist
7                   of only CNOT_i gates*/
8           S' = S_R /* the remaining part of the result of Shift(S', i) */
9       }
10      while (gates g that have auxiliary qubits in its control bits exist) {
11          Let g be the left most one of such gates.
12          Move g to the next right position of CNOT_{n+1} by applying one
13              of Rules (2) and (4) as many times as possible.
14          Delete g by applying Rule (6).
15      }
16      Reorder the gates whose target bits are auxiliary qubits lexicographically
17          by applying Rule (2) as many times as possible.
18      Delete redundant pair of gates by applying Rule (1) as many times as possible.
19  }
```

Figure 8: $Main(S)$

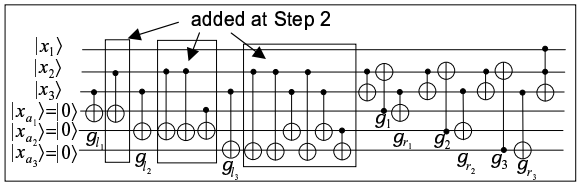

Figure 9: The Circuit after Step 2

This is actually not the case because a $CNOT_{a_j}$ is always added at the lefthand-side of the transformation of Rule (3).) Then we get a circuit where all $CNOT_{a_j}(1 \le j \le k)$ except for $g_{r_j}(1 \le j \le k)$ are placed in the left most part of the circuit. See Fig. 9 for a circuit after Step 2 in our example.

Step 3. We then move each $g_{r_j}(1 \le j \le k)$ to the opposite direction, i.e., to the right as far as possible. Again we can always do this by applying Rules (2) to (4) and exactly the same as before as for the added gates whose target bits are a_j-th $(1 \le j \le k)$. Thus we get the circuit where all $CNOT_{a_j}(1 \le j \le k)$ are placed in the left most part or in the right most part of the circuit.

Step 4. Without loss of generality we can assume that g_{l_1} is placed at the left-end position of the circuit after Step 2. As one can see later, we can overcome several difficulties we encounter when moving $CNOT_m$ gates to the left, by giving a special role to this gate g_{l_1}. Now, for each gate g which has the m-th bit in its control bits and placed in the lefthand-side of the $CNOT_m$ gates, we apply the following transformations: First we move g to the left so that it comes to the next right position of g_{l_1} by using Rule (2) or (3) as many times as possible. Then we apply Rule (5) to now consecutive g_{l_1} and g so that the control bit $|x_m\rangle$ of g_{l_j} is changed to the a_1-th bit. See Fig. 10 for a circuit after Step 4 in our example. At this moment no gates, except for the leftmost gate g_{l_1}, which have the m-th bit in their control bits exist if they are placed in the left side of the $CNOT_m$ gates. Note that the gates added in Step 2 must not have the m-th bit in their control bits, and therefore we can always perform the following Step 5.

Step 5. We move each $CNOT_m$ so that it comes to the next right position of g_{l_1} by applying Rules (2) to (4). If we need to add new $CNOT_m$ gates in the above transformation, we also move these added gates similarly. Now all $CNOT_m$ are placed consecutively, in the next right position of g_{l_1}. The left portion of our example looks like Fig. 11 (a) after this step (recall that $m = 3$).

Step 6. Thus we have almost moved the $CNOT_m$ gates to the left. The only remaining obstacle is g_{l_1}. In this step, we delete redundant $CNOT_m$ gates as follows.

Step 6.1. We reorder those $CNOT_m$ gates lexicographically by applying Rule (2) as many times as possible. Then we have three groups of $CNOT_m$; the first group does not include the a_j-th bit $(1 \le j \le k)$ in the control bits, the second group has only the a_1-th bit (no a_j-th bit $(2 \le j \le k)$) in the control bits, and the third group of the remaining gates has the a_j-th bit $(1 \le j \le k)$ in the control bits. Let these groups be Groups A, B and C, respectively. The three groups of $CNOT_3$ after this step are illustrated in Fig. 11 (b).

Step 6.2. We delete all gates in Group C by Rule (6).

Step 6.3. We apply Rule (1) to delete redundant pairs of gates in Groups A and B. We will claim that all the gates in Group B, and Group A as well if $m \neq n+1$, disappear at this moment in the next Lemma 4. Thus all $CNOT_m$ gates that have the a_1-th bit in their control bits, i.e., those in Groups B or C, have been deleted. Therefore, we can perform the following Step 7.

Step 7. We finally move g_{l_1} which remains in the left most position to the right side of all $CNOT_m$ by applying Rule (3) as many times as possible. We may again need to add $CNOT_{a_1}$, which are moved to the right side as well. Finally we get a circuit where all $CNOT_m$ are placed in the left most part of the circuit. That completes $Shift(S, m)$.

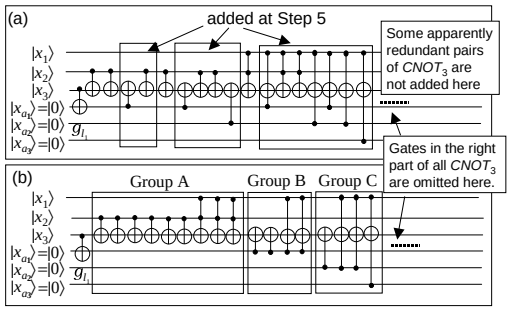

Figure 11: The Circuit in Step 6

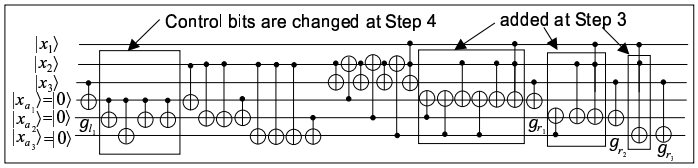

Figure 10: The Circuit after Step 4

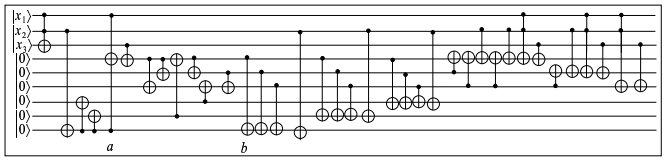

Figure 12: The Circuit after *Shift* Is Finished

Lemma 4. All the gates in Group B must disappear after Step 6.3 of $Shift(S,m)$ if the circuit S is proper. All the gates in Group A must also disappear when $m \neq n+1$.

Proof. Below we consider the state of $|x_m\rangle$. Recall that we have only g_{l_1} in the left part of Group A. Therefore, before Group A the states of $|x_1\rangle$ to $|x_{n+1}\rangle$ remain unchanged, i.e., being equal to their initial states, and only $|x_{a_1}\rangle$ has changed to be equal to $|x_m\rangle$.

Suppose that some gates in Groups A or B have remained after Step 6.3, and let the i-th such gate in Group A be $[m, C_{A_i}]$, and the i-th gate in Group B be $[m, C_{B_i}]$. Furthermore, let $X_{C_{A_i}}$ be a product term of all x_j such that $j \in C_{A_i}$, and $X_{C_{B_i}}$ be a product term for C_{B_i}. Then the value of the state of $|x_m\rangle$ after Group A can be written as $x_m \oplus X_{C_{A_1}} \oplus X_{C_{A_2}} \oplus \cdots = x_m \oplus (X_{C_{A_1}} \oplus X_{C_{A_2}} \oplus \cdots) = x_m \oplus f_A$ because of associative laws for the XOR operator. An important point here is that $X_{C_{A_1}}, X_{C_{A_2}}, \cdots$ are all different since Step 6.3 includes the simplification procedure (by Rule (1)) after reordering the gates into the lexicographical order. This means that f_A is a positive polarity Reed-Muller form. Also note that f_A does not have the literal x_m since each C_{A_i} does not include m. We then can calculate the value of the state of $|x_m\rangle$ after Group B as $x_m \oplus f_A \oplus X_{C_{B_1}} \oplus X_{C_{B_2}} \oplus \cdots$. Recall that all the gates in Group B have the a_1-th bit as their control bits, therefore, all $X_{C_{B_i}}$ in the formula contain the literal x_{a_1} which is equal to x_m because of the leftmost gate g_{l_1}. Accordingly, we can rewrite the formula as $(x_m \oplus f_A \oplus x_m f'_B)$, where f'_B does not have the literal x_m and is a positive polarity Reed-Muller form again. (The XOR has distributive laws, for example, $xyz \oplus x = x(yz \oplus 1)$.)

Since there are no $CNOT_m$ after Group B (recall that Group C's gates have already disappeared) and the circuit is proper, we have

$$|x_m \oplus f_A \oplus x_m f'_B\rangle = \begin{cases} |x_m\rangle & \text{if } 1 \leq m \leq n, \\ |x_m \oplus f\rangle & \text{if } m = n+1, \end{cases}$$

where f has only literals x_1 through x_n. Now we can conclude that $f_A = f'_B = 0$ if $1 \leq m \leq n$, and $f'_B = 0$ if $m = n+1$ for the following reason: The above formulas can be expanded with respect to x_m, i.e., $x_m \oplus f_A \oplus x_m f'_B = (f_A f'_B + \overline{f_A}\ \overline{f'_B})x_m + f_A\overline{x_m}$, and $x_m \oplus f = \overline{f}x_m + f\overline{x_m}$. One can see that these two functions are equivalent iff $\overline{f} = f_A f'_B + \overline{f_A}\ \overline{f'_B}$ and $f = f_A$, which implies $f'_B = 0$. One can also see that $(f_A f'_B + \overline{f_A}\ \overline{f'_B})x_m + f_A\overline{x_m} = x_m$ implies $f_A = f'_B = 0$. Also note that if $g = X_1 \oplus X_2 \oplus \cdots$ is a positive polarity Reed-Muller form, then $g = 0$ iff $X_1 = X_2 = \cdots 0$. Therefore, no product terms are included in f_A or f'_B, i.e., there should be no gates in Group B, and neither in Group A when $m \neq n+1$. □

After we successfully get a circuit where all $CNOT_{n+1}$ are placed in the left most part of the circuit by $Shift(S, n+1)$, $Main$ then calls $Shift(S', n)$, by which all $CNOT_n$ gates are shifted to the left. They are placed next to $CNOT_{n+1}$ gates already shifted. Then we can delete all $CNOT_n$ by Lemma 4. This continues until $Shift(S', 1)$. In our example, after calling $Shift(S', 1)$, we get a circuit in Fig. 12.

After the 8th line in Fig. 8, we have $CNOT_{n+1}$ gates which are already of the canonical form at the left part of the circuit and then $CNOT_i$ gates $(i \geq n+2)$ after them. Now we execute the second half of $Main$, where we delete all the $CNOT_i (i \geq n+2)$ gates, i.e., those whose target bits are auxiliary ones, and finally there remain only $CNOT_{n+1}$ gates in the circuit.

The $CNOT_i$ $(i \geq n+2)$ gates are divided into two groups. The first group, denoted by A, has control bits in auxiliary bits and the second group, B, does not. We first move group A gates to the left until the next positions to $CNOT_{n+1}$ gates. This can be done by switching group A gates with group B gates, which is always possible by using Rules (2)-(4). (Switching two gates both in group A, like gates a and b in Fig. 12, is not so easy.) Then the group A gates can be deleted one by one from the leftmost one by Rule (6). Then we have only group B gates. Recall that when we introduced a $CNOT_i (i \geq n+2)$ gates, we always introduced another $CNOT_i$ which cancels the value of $|x_i\rangle$. Therefore the final values of the auxiliary qubits are all 0. Therefore we can delete the group B gates by reordering and applying Rule (1). (Otherwise, we can imply a contradiction just as we did in the proof of Lemma 4.) In our example, we successfully get to a circuit in Fig. 6 which is the canonical form for the initial circuit. □

4.3 Completeness of the Rule Set

Now our main result is almost immediate:

Theorem 2. Let S_1 and S_2 be any equivalent quantum Boolean circuits. Then there exists a sequence of transformation rules, each in the rule set given in Section 3, which transforms S_1 to S_2.

Proof. Since S_1 and S_2 are equivalent, their canonical forms are the same by Lemma 3. Let this canonical form be S. We can transform S_1 into S by Theorem 1. Let this sequence of transformation rules be r_1. We can also transform S_2 into S by sequence r_2. Since all of our transformation rules are bidirectional, we can get a sequence r'_2 of rules that transforms S to S_2 simply by reversing r_2. Now the sequence r_1 followed by r'_2 transforms S_1 into S_2. □

5. CONSTRUCTION OF EFFICIENT CIRCUITS

In this section, we propose an example of circuit design procedure based on our transformation rules. The design flow consists of the following three steps:

Step 1. We make an arbitrary initial CNOT-based circuit from a given formula (depending on the problem for the quantum algorithm).

Step 2. We transform the initial circuit to its canonical form by our procedure $Main(S)$ in Section 4.

Step 3. A new transformation rule is given in Section 5.2, which can be realized by composing our transformation rules and guarantees a reduction if the cost of the circuit. We just apply this rule as many times as possible.

For Step 1, it should be noted that if we can use auxiliary qubits, we can simulate any type of formula by a CNOT-based circuit whose size is almost the same as the given formula. Here, we show how to simulate CNF formulas by CNOT-based circuits in the following section.

5.1 Circuits for CNF Formulas

For simplicity, we use an example. Suppose that $f = (x_1 + x_3)(\overline{x_1} + x_2 + x_3)$. Then we prepare two auxiliary qubits, $|x_5\rangle$ and $|x_6\rangle$ for the first and the second clause, respectively (see Fig. 13). Then we introduce a gate $g_1 = [5, \{1, 3\}]$. (In general, for the i-th clause, we introduce $g_i = [n+1+i, C_i]$ where C_i includes j iff x_j or $\overline{x_j}$ appears in the clause. Thus we use one auxiliary qubit per clause.) Then, we place NOT_5, right after g_1 where NOT_i is a simpler notation for $[i, \{\}]$. NOT_i works just like a negation gate. Furthermore, we place two NOT_1 gates before and after g_1 since x_1 appears in the positive form. Similarly two NOT_3 gates are inserted. Let G_1 be the set of these gates. Then a similar set, G_2, of gates are introduced for the second clause, but no NOT_1 is inserted since x_1 appears in the negative form in the second clause. Then, we introduce one special gate $p = [4, \{5, 6\}]$ to propagate the product value of the clauses into $|x_4\rangle$. After that we introduce G'_2 and G'_1, which are exactly the same as G_2 and G_1, respectively, in the mirror positions as shown in Fig 13.

Now it is easy to see that the value of $|x_5\rangle$ after G_1 is $\overline{0 \oplus \overline{x_1}\ \overline{x_3}} = (x_1 + x_3)$, which is equal to the first clause, and similarly for $|x_6\rangle$ and G_2. Thus the value of $|x_4\rangle$ after the special gate p simulates the value of the given CNF formula f. G'_1 and G'_2 are used for restoring the states of the auxiliary qubits. Thus our circuits simulate CNF formulas without much increasing the size, which immediately implies that the satisfiability problem for our circuits is NP-complete. Also, our transformation rules can simulate the Resolution system

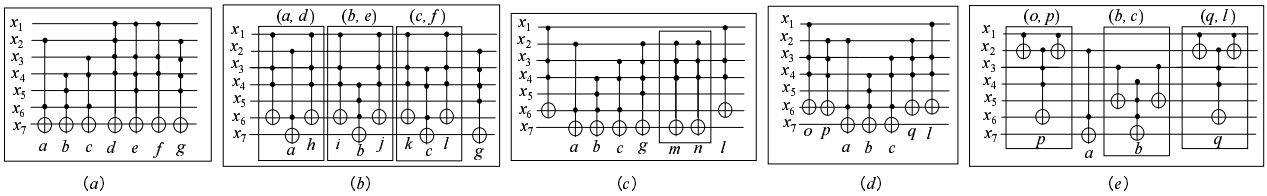

Figure 15: A Minimization Procedure

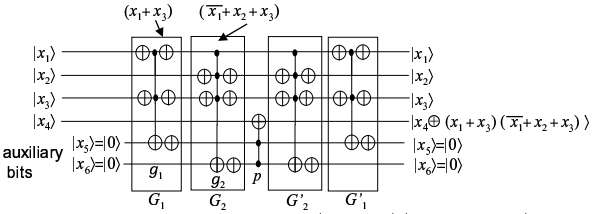

Figure 13: A Circuit for $(x_1+x_3)(\overline{x_1}+x_2+x_3)$

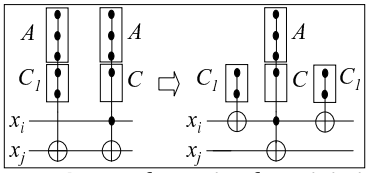

Figure 14: A Transformation for Minimization

to prove the unsatisfiability of CNF formulas with only a polynomial overhead (omitted in this paper).

5.2 Circuit Reduction from the Canonical Form

First we show a useful complex transformation rule which can be realized by applying a sequence of our transformation rules:

$$[j, A \cup C_1] \cdot [j, A \cup C \cup \{i\}] \Rightarrow [i, C_1] \cdot [j, A \cup C \cup \{i\}] \cdot [i, C_1],$$

if $C \subseteq C_1$.

Figure 14 illustrates this transformation. This transformation can be decomposed as follows: First we put two $[i, C_1]$ gates by Rule (1) right after $[j, A \cup C \cup \{i\}]$. Next we move $[j, A \cup C_1]$ and $[j, A \cup C \cup \{i\}]$ to the position between the two added $[i, C_1]$ gates by Rules (2) and (3), where a new $[j, A \cup C_1]$ gate is added. Finally, we remove two $[j, A \cup C_1]$ gates by Rule (1) and get the righthand-side of the transformation. We can generalize the transformation for more than two gates as follows:

$[j, A \cup C_1] \cdot [j, A \cup C_2] \cdots [j, A \cup C_k] \cdot [j, A \cup C \cup \{i\}] \Rightarrow [i, C_1] \cdot [i, C_2] \cdots [i, C_k] \cdot [j, A \cup C \cup \{i\}] \cdot [i, C_1] \cdot [i, C_2] \cdots [i, C_k]$,
if $C \subseteq C_l (1 \leq l \leq k)$.

It is of course hard to guess a reasonable definition of the cost of CNOT gates at this moment. However, as shown later, there is some evidence that the implementation of a CNOT gate with more inputs should be more expensive than with less inputs. If it is true, then the implementation cost of the circuit of the righthand-side of the above transformation is smaller than that of the lefthand-side.

It is easy to see that we can transform Circuit *B* to Circuit *C* in the previous Fig. 2 by applying this transformation many times. Below we present a more complicated and interesting example where we need some heuristics. Suppose we start from the circuit in Fig. 15 (*a*). First we apply the transformation to three pairs of gates, i.e., *a* and *d*, *b* and *e*, and *c* and *f* to get the circuit in Fig. 15 (*b*). Next we can simply delete two pairs of redundant gates, i.e., *h* and *i*, and *j* and *k* by Rule (1). Then we can optimize the circuit further by applying the above transformation to the gates *b* and *c*.

We have another way to optimize the circuit even more in this case: First we swap the order of *l* and *g*, and add two gates *m* and *n* as shown in Fig. 15 (*c*). Then we can apply the transformation to three pairs of gates, i.e., *a* and *m*, *b* and *g*, and *c* and *n* to get the circuit in Fig. 15 (*d*). Next we further apply the transformation to three pairs of gates, i.e., *o* and *p*, *b* and *c*, and *q* and *l* to get the final circuit in Fig. 15 (*e*).

Here we give a comparison between the implementation costs for the circuits in Fig. 15 (*a*) and Fig. 15 (*e*) based on the following cost assumption, where $Cost(n)$ means the cost of a CNOT gate with *n* control bits: (i) $Cost(1) = 1$, (ii) $Cost(2) = 14$, (iii) $Cost(3) = 56$, (iv) $Cost(4) = 140$, and (v) $Cost(m) = 112(m-3)$ when $m \geq 5$. Our assumption is based on [1], which shows that (i) a CNOT gate with two control bits can be decomposed to 14 basic quantum gates with fewer inputs, (ii) a CNOT gate with three and four control bits can be decomposed to four and ten CNOT gates with two control bits, respectively, and (iii) a CNOT gate with $m(\geq 5)$ control bits can be decomposed to $8(m-3)$ CNOT gates with two control bits. By a simple calculation we can see that the cost of the circuit in Fig. 15 (*a*) is 602, whereas the cost for Fig. 15 (*e*) is 188. Thus, we can usually make a circuit smaller if we apply our complex transformation to appropriate portions of the circuit. To find the best portion to which we should apply the transformation is a combinatorial problem similar to finding a best *divisor* in the conventional multilevel logic synthesis [7].

6. CONCLUDING REMARKS

As mentioned earlier, our ultimate goal is to develop a design theory for quantum Boolean circuits. For this purpose: (1) We will need more concrete "guide" or heuristics on how to change a given circuit into a *better* circuit. (2) During the course of this research, we may find "a counter example", i.e., a pair of circuits for which our rule set needs exponentially many steps. The rule set will then need to be modified to cope with such counter examples. (3) A design theory for *sequential* quantum circuits will also be interesting.

7. REFERENCES

[1] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, Nov. 1995.

[2] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang. MIS: A Multiple-Level Logic Optimization System. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6(6):1062–1081, Nov. 1987.

[3] J. Darringer, W. Joyner, L. Berman, and L. Trevillyan. LSS: Logic synthesis through local transformations. *IBM J. Res. and Develop.,*, 25(4):272–280, July 1981.

[4] M. Davio, J.-P. Deshamps, and A. Thayse. *Discrete and Switching Functions*. McGraw Hill International, 1978.

[5] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM Symposium on Theory of Computing*, pages 212–219, 1996.

[6] J. Gruska. *Quantum Computing*. McGraw Hill, 1999.

[7] G. D. Hactel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996.

[8] K. Iwama, K. Hino, H. Kurokawa, and S. Sawada. Random benchmark circuits with controlled attributes. In *Proc. European Design & Test Conference and Exhibition (ED&TC'97)*, pages 90–97, 1997.

[9] J.-S. Lee, Y. Chung, J. Kim, and S. Lee. A Practical Method of Constructing Quantum Combinational Logic Circuits. Technical Report http://arXiv.org/abs/quant-ph/9911053, LANL e-print, 1999.

[10] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[11] T. Toffoli. Reversible computing. Technical Report MIT/LCS/TM-151, MIT LCS, Feb. 1980.

[12] A. Yao. Quantum circuit complexity. In *Proc. 34th Annual IEEE Symposium on Foudations of Computer Science*, pages 352–361, 1993.