Fredkin/Toffoli Templates for Reversible Logic Synthesis

Dmitri Maslov
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada
cetrau@mail.ru

Gerhard W. Dueck
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada
gdueck@unb.ca

D. Michael Miller
Dept. of Computer Science
University of Victoria
Victoria, BC, V8W 3P6
Canada
mmiller@csr.uvic.ca

ABSTRACT

Reversible logic has applications in quantum computing, low power CMOS, nanotechnology, optical computing, and DNA computing. The most common reversible gates are the Toffoli gate and the Fredkin gate. Our synthesis algorithm first finds a cascade of Toffoli and Fredkin gates with no backtracking and minimal look-ahead. Next we apply transformations that reduce the size of the circuit. Transformations are accomplished via template matching. The basis for a template is a network with m gates that realizes the identity function. If a sequence in the network to be synthesized matches more than half of a template, then a transformation that reduces the gate count can be applied. In this paper we show that Toffoli and Fredkin gates behave in a similar manner. Therefore, some gates in the templates may not need to be specified—they can match a Toffoli or a Fredkin gate. We formalize this by introducing the box gate. All templates with less than six gates are enumerated and classified. We synthesize all three input, three output reversible functions and compare our results to those obtained previously.

1. INTRODUCTION

Reversible logic is an emerging research area. The synthesis of reversible circuits differs significantly from synthesis using traditional irreversible gates. Two restrictions are added for reversible networks, namely fan-outs and feedbacks are not allowed. The only possible structure for a reversible network is a cascade of reversible gates. The most frequently used gates are the Toffoli gate [9] and the Fredkin gate [3]. The Toffoli gate inverts a single bit if the AND of a set of control lines is 1. That is, it performs a controlled NOT. The Fredkin gate interchanges two bits if the AND of a set of control lines is 1. In other words, a controlled swap. The formal definitions are given in Section 2.

Only a few synthesis methods have been proposed for reversible logic. Suggested methods include: using Toffoli gates to implement an ESOP (EXOR sum-of-products) [7], exhaustive enumeration [8], heuristic methods that iteratively make the function simpler (simplicity is measured by the Hamming distance [1] or by spectral means [5]), and transformation based synthesis [4], among others. Some methods use excessive search time, others are not guaranteed to converge, and some require many additional outputs (garbage). We follow the two-step approach suggested in [6] and further investigated in [2]. The first paper [6] describes templates with Toffoli gates only. The second [2] introduces some templates with Fredkin gates, however, they are restricted to three inputs. First a network for the given function is found. The algorithm for this step is guaranteed to converge. In fact, the algorithm is very fast. Improvements on a naive algorithm are described in [6] (as they apply to Toffoli networks). The second step consists of applying transformations that reduce the number of gates. In this paper we describe and classify the templates used for such transformations in detail.

2. DEFINITIONS

In this work we consider cascades of generalized Toffoli [9] and generalized Fredkin [3] gates defined as follows.

DEFINITION 1. *For the set of domain variables $\{x_1, x_2, ..., x_n\}$ the* **generalized Toffoli gate** *has the form $TOF(C, T)$, where $C = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}, T = \{x_j\}$ and $C \cap T = \emptyset$. It maps the Boolean pattern $(x_1^0, x_2^0, ..., x_n^0)$ to $(x_1^0, x_2^0, ..., x_{j-1}^0, x_j^0 \oplus x_{i_1}^0 x_{i_2}^0 ... x_{i_k}^0, x_{j+1}^0, ..., x_n^0)$.*

DEFINITION 2. *For the set of domain variables $\{x_1, x_2, ..., x_n\}$* **generalized Fredkin gate** *has the form $FRE(C, T)$, where $C = \{x_{i_1}, x_{i_2}, ..., x_{i_k}\}, T = \{x_j, x_l\}$ and $C \cap T = \emptyset$. It maps the Boolean pattern $(x_1^0, x_2^0, ..., x_n^0)$ to $\{x_1^0, x_2^0, ..., x_{j-1}^0, x_l^0, x_{j+1}^0, ..., x_{l-1}^0, x_j^0, x_{l+1}^0, ..., x_n^0)$ iff $x_{i_1}^0 x_{i_2}^0 ... x_{i_k}^0 = 1$. In other words, the generalized Fredkin gate interchanges bits x_j and x_l if and only if corresponding product equals 1.*

For both gate types, C will be called the **control** set and T will be called the **target** set. The number of elements of the set of controls C defines the **width** of the gate. The set of generalized Toffoli and generalized Fredkin gates will be called the **Fredkin-Toffoli family**. For the control set $C = \{x_3, x_4, ..., x_{k+2}\}$ the pictorial representation of gate $TOF(C, x_2)$ is shown in Fig. 1a and the pictorial representation of gate $FRE(C, x_1 + x_2)$ (where "+" stays for set union operation) is shown in Fig. 1b.

Toffoli and Fredkin gates are closely related. In fact, they can be written as one general gate $G(S, B)$. In section 4 we will see how useful it is to unite these two gates together.

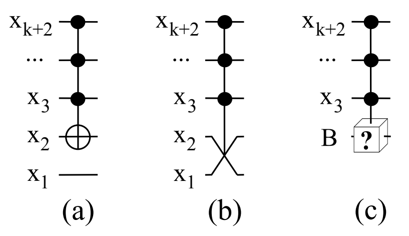


Figure 1: Gates Toffoli, Fredkin and Box.

The uniform way of writing Toffoli and Fredkin gates is captured in the definition of a **box gate** $G(S, B)$, which for $|B| = 1$ is the $TOF(S, B)$ gate and for $|B| = 2$ is $FRE(S, B)$. Such a way of writing the gates is needed when we consider a general gate from the Fredkin-Toffoli family and do not want to specify which gate it is. So, if the size of the set B is not specified, it can be either 1 or 2. The gate shown in Fig. 1c is $G(C, B)$ where the set B is not specified. If a box gate is found in a network the following rules help to decide which gate (Toffoli or Fredkin) it represents and the way the network looks after assigning EXOR or SWAP to the box.

- If the EXOR operation is assigned to a box symbol, all the boxes on this line become EXORs and nothing else changes.
- If the SWAP operation is assigned to a box symbol, all the boxes on this line are changed to SWAPs. SWAPs require two lines to be properly written, therefore, add one line so that the two lines with the newly built SWAP have controls everywhere the line with box had, and only there.
- In a correctly built circuit a box symbol will never be on the same line with EXOR or SWAP symbols.

Further, if the setting for the box (Toffoli or Fredkin) is not specified, the box can be either one, assigned accordingly to the above rules.

3. THE ALGORITHM

In this section we consider the synthesis of completely specified reversible functions realized with gates from the Fredkin-Toffoli family. Before we describe the algorithm, we have to agree on the function representation. For our method it is best to think of the function to be realized as given by a truth table, which has input patterns on the left and output patterns on the right side. The input patterns are arranged in lexicographical order.

The basic algorithm starts with an empty circuit (the identity function). At every step of the synthesis algorithm we add some gates from the Fredkin-Toffoli family to the end of the circuit. Since the reversible cascade can be built from either end. The basic algorithm starts from the output.

Basic algorithm.

Step 0. *Idea: take the narrowest gates and arrange them in a cascade so that they bring the first output pattern to the first input pattern.*

The first pattern in the truth table is the lowest on the order sequence $(0, 0, ..., 0)$ (assume Boolean n-tuples are arranged in lexicographical order), let the first pattern in output be $(b_1, b_2, ..., b_n)$. To bring it to the form $(0, 0, ..., 0)$, use gates $TOF(\emptyset, x_i)$ for every $i : b_i \neq 0$. After adding the gates to the cascade, update the output part of the table such that the pattern $(b_1, b_2, ..., b_n)$ is transformed to the desired form $(0, 0, ..., 0)$.

Step S, $1 \leq S \leq 2^n - 2$**.** *Idea: without influencing the patterns of the lower order that were put at their desired places in the previous steps of the algorithm, use the least number of the narrowest gates to bring the output pattern to the form of correspondent input pattern.*

The input pattern of the table, $(a_1, a_2, ..., a_n)$ is the binary representation of natural number (S+1). The pattern in the last update of the output part is any pattern $(b_1, b_2, ..., b_n)$ of higher order. If the order is the same, the patterns are equal, and no action is required. The order of $(b_1, b_2, ..., b_n)$ cannot be less than the order of $(a_1, a_2, ..., a_n)$ since all such patterns were put to their places during the previous steps of the algorithm.

For pattern $(b_1, b_2, ..., b_n)$ to be transformed to the form $(a_1, a_2, ..., a_n)$ note that each application of a Toffoli gate is capable of flipping one bit of pattern $(b_1, b_2, ..., b_n)$ either from value 1 to value 0 or vice versa, and each Fredkin gate is capable of permuting a pair of unequal Boolean values. Now, a problem can be formulated as follows: using the two operations "flip" and "swap" bring a Boolean pattern $(b_1, b_2, ..., b_n) \succ (a_1, a_2, ..., a_n)$ to the form $(a_1, a_2, ..., a_n)$ so the all intermediate Boolean patterns are greater then $(a_1, a_2, ..., a_n)$. The controls for the corresponding gates will be assigned later. The problem's solution is as follows.

- If the number of ones in $(b_1, b_2, ..., b_n)$ is less than the number of ones in $(a_1, a_2, ..., a_n)$ apply "swaps" that improve 2 bit positions and flip the remaining incorrect bits. Use "swaps" so that the order of each intermediate pattern $(x_1, x_2, ..., x_n)$ is less than the order of $(a_1, a_2, ..., a_n)$ and the set of controls defined as minimal subset of unit values of $(x_1, x_2, ..., x_n)$ such that this subset forms a Boolean pattern of an order higher than $(a_1, a_2, ..., a_n)$ is minimal. This can be easily done if "swaps" are done on the low order bits first. Note, that in this case initial pattern $(b_1, b_2, ..., b_n)$ was greater than $(a_1, a_2, ..., a_n)$, so the most significant binary digit of $(b_1, b_2, ..., b_n)$ equal one was greater than the most significant one digit of $(a_1, a_2, ..., a_n)$. Thus, it will be taken as the control (when a control is needed) for all corresponding Fredkin and Toffoli gates except the last Toffoli gate, for which the control will consist of all unit digits of $(a_1, a_2, ..., a_n)$.
- If the number of ones in $(b_1, b_2, ..., b_n)$ is equal to the number of ones in $(a_1, a_2, ..., a_n)$, it is possible to transform one pattern into the other using "swap" operation only. Controls are determined by the procedure described in the above case.
- If the number of ones in $(b_1, b_2, ..., b_n)$ is greater then the number of ones in $(a_1, a_2, ..., a_n)$, apply "swaps" starting from the end of the pattern $(b_1, b_2, ..., b_n)$ and then apply necessary Toffoli gates. All the necessary controls can be found using the procedure from the first case.

Step $2^n - 1$**.** When all the $2^n - 1$ of previous patterns are in places, the last patterns will automatically be correct.

Motivation. Given a target technology, it usually happens that the narrower the gate, the less costly it is, thus we try to use the narrowest gates. Although choosing the narrowest gates at each step may lead to larger initial circuits which might not be simplified enough by the template tool. Another possibility is to chose the control set such that the remaining function is as simple as possible. To measure "simplicity" we use the Hamming distance as a heuristics. It also happens that the template simplification tool is sensitive to the width of gates, therefore by taking narrow gates we prepare the circuit for better template reduction.

Bidirectional modification. The basic algorithm worked from the output to input by adding the gates in one direction starting from the end of desired cascade and ending at its beginning. What if we were able to understand what happens if during the procedure when a gate is added to the beginning of cascade? Then we would be able to construct the network from the two ends simultaneously by growing the number of gates from the two sides. The idea of the method is the same—by applying the gates to match input with output part of the truth table to each other by assuring that at each step of calculation we put at least one pattern at its place. It makes sense that such a bidirectional algorithm on average will converge faster.

- Toffoli gate application. Without loss of generality apply gate $TOF(C, x_{k+1})$, $C = \{x_1, x_2, ..., x_k\}$ with the controls on first k variables and target on the $(k+1)$ variable (all other generalized Toffoli gates have permuted set of controls and, maybe, a different target). Then, in the input part of the truth table the patterns $(1, 1, ..., 1, x^0_{k+1}, x^0_{k+2}, ..., x^0_n)$ will be interchanges with patterns $(1, 1, ..., 1, \bar{x}^0_{k+1}, x^0_{k+2}, ..., x^0_n)$. Which, in terms of our understanding is the same as permuting the output patterns in front of the $(1, 1, ..., 1, x^0_{k+1}, x^0_{k+2}, ..., x^0_n)$ and $(1, 1, ..., 1, \bar{x}^0_{k+1}, x^0_{k+2}, ..., x^0_n)$ input patterns without changing the input part. This understanding is easier to visualize, since the matched patterns do not get mixed up in the middle of the truth table which makes it hard to track everything. For the program realization, the input part may be changed.

- Fredkin gate application. Apply a Fredkin gate $FRE(C; x_{k+1}, x_{k+2})$, $C = \{x_1, x_2, ..., x_k\}$. This results in the following change of all patterns in the input part of the table: $(1, 1, ..., 1, x^0_{k+1}, x^0_{k+2}, x^0_{k+3}, ..., x^0_n)$ is interchanged with $(1, 1, ..., 1, x^0_{k+2}, x^0_{k+1}, x^0_{k+3}, ..., x^0_n)$. If we want to keep the conventional form of input part of the truth table when patterns are arranged lexicographically, this operation is the same as interchanging the output patterns of the truth table which stay in front of input patterns $(1, 1, ..., 1, x^0_{k+1}, x^0_{k+2}, x^0_{k+3}, ..., x^0_n)$ and $(1, 1, ..., 1, x^0_{k+2}, x^0_{k+1}, x^0_{k+3}, ..., x^0_n)$.

The presented algorithm is an improved version of the algorithm proposed in [6]. The algorithm in [6] uses Toffoli gates only, and its basic structure was simple: for input pattern $(a_1, a_2, ..., a_n)$ use the Toffoli gates to bring the output pattern $(b_1, b_2, ..., b_n)$ to the form $(a_1 \vee b_1, a_2 \vee b_2, ..., a_n \vee b_n)$ (increase the order) using controls $x_i : b_i = 1$. And then use the Toffoli gates with controls $x_i : a_i = 1$ to bring $(a_1 \vee b_1, a_2 \vee b_2, ..., a_n \vee b_n)$ to the desired form $(a_1, a_2, ..., a_n)$. For such an algorithm it was easy to construct the worst case function. Particularly, for $n = 3$ such a function was constructed (it is unique) and called 3_17.*pla*. The cost of realizing this function was 17.

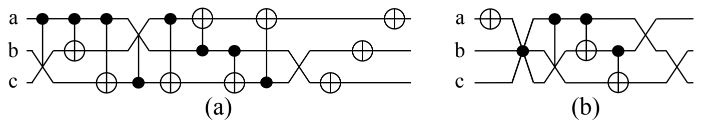


Figure 2: Circuits for 3_17.*pla*.

The following two tables illustrate the work of the algorithm synthesizing the circuits for 3_17.*pla*. Table 1 refers to the basic approach (Toffoli gates are denoted by T and Fredkin gates by F), and Table 2 illustrates the bidirectional algorithm. Each column in the tables shows the change of the output part of the truth table as the gates from the previous steps are applied. The arrows in Table 2 indicate the direction of gate assignment (beginning of the cascade $\rightarrow$, or the end of the cascade $\leftarrow$). The corresponding circuits are shown in the Fig. 2.

4. TEMPLATE SIMPLIFICATION TOOL

Let a **size** m **template** be a sequence of m gates (a circuit) which realizes identity function. The template size m should also be independent of the templates of smaller size, e.g. for a given template size m no application of any set of templates of smaller size can decrease the number of its elements. For a template $G_0\ G_1 ...\ G_{m-1}$ its **application** is one of the two operations:

1. Forward application. A sequence of gates in the network which matches the sequence $G_i\ G_{(i+1)\ mod\ m} \cdots G_{(i+k-1)\ mod\ m}$ in the template $G_0\ G_1 ...\ G_{m-1}$ is replaced with the sequence $G_{(i-1)\ mod\ m}\ G_{(i-2)\ mod\ m} \cdots G_{(i+k)\ mod\ m}$ without changing the network's output, where $k \in M, k \geq \frac{m}{2}$.

2. Backward application. A sequence of gates in the network which matches the sequence $G_i\ G_{(i-1)\ mod\ m} \cdots G_{(i-k+1)\ mod\ m}$ is replaced with the sequence $G_{(i+1)\ mod\ m}\ G_{(i+2)\ mod\ m} \ ...\ G_{(i-k)\ mod\ m}$ without changing the network output, where $k \in M, k \geq \frac{m}{2}$.

Thus, each application of a template size m for parameter $k > \frac{n}{2}$ leads to a reduction in the number of gates.

Definition 3. *A* **class of templates size n** *is a circuit $G(S_1, B_1)\ G(S_2, B_2)\ ...G(S_n, B_n)$ with a set of logical conditions on sets $S_1, B_1, S_2, B_2, ..., S_n, B_n$. When a class is mentioned, it may be written as $G_{i_1}\ G_{i_2} ...\ G_{i_n}$, where $i_k = i_j$ iff $S_k = S_j$ and $B_k = B_j$.*

This approach to defining a class is useful for short classification and computer implementation, but it is difficult to visualize. Thus, we introduce the following notation (definition):

Definition 4. *A class can be written as a set of* **disjoint** *formulas, i.e. formulas $G_1(S_1, B_1)\ G_2(S_2, B_2)\ ...G_m(S_m, B_m)$, where:*

- *according to the number of elements in B_i G_i is written as $TOF(|B_i| = 1)$ or $FRE(|B_i| = 2)$;*
- *S_i is written as a union of sets (C) and single variables (t): $S_i = C_{i_1} + C_{i_2} + ... + C_{i_k} + t_{i_1} + t_{i_2} + ... + t_{i_j}$;*
- *if $|B_i| = 1$, it is written as single variable, t_j; if $|B_i| = 1$ it is written as union $t_j + t_k$;*

In	Out	S0	S1	S2	S3	S4	S5
000	111	**000**	**000**	**000**	**000**	**000**	**000**
001	001	110	**001**	**001**	**001**	**001**	**001**
010	100	011	111	**010**	**010**	**010**	**010**
011	011	100	100	100	**011**	**011**	**011**
100	000	111	011	110	111	**100**	**100**
101	010	101	110	011	101	110	**101**
110	110	001	010	111	110	101	**110**
111	101	010	101	101	100	111	**111**
apply gates:		$T(a)$ $T(b)$ $T(c)$	$F(;b,c)$ $T(c;a)$	$T(b;c)$ $T(b;a)$	$T(a;c)$ $F(c;a,b)$	$T(a;c)$ $T(a;b)$	$F(a;b,c)$

Table 1: An example of the basic synthesis algorithm.

In	Out	S0	S1	S2	S3	S4	S5
000	111	**000**	**000**	**000**	**000**	**000**	**000**
001	001	010	**001**	**001**	**001**	**001**	**001**
010	100	110	101	**010**	**010**	**010**	**010**
011	011	101	110	111	**011**	**011**	**011**
100	000	111	111	110	110	**100**	**100**
101	010	001	010	100	100	110	**101**
110	110	100	100	011	111	101	**110**
111	101	011	011	101	101	111	**111**
apply gates:		$\rightarrow TOF(;a)$	$\leftarrow FRE(;b,c)$	$\leftarrow FRE(;a,b)$ $\leftarrow TOF(b;c)$	$\rightarrow FRE(b;a,c)$	$\leftarrow TOF(a;b)$	$\rightarrow FRE(a;b,c)$

Table 2: Bidirectional synthesis of $3\_17.pla$.

- *all the sets are disjoint: $C_i \cap C_j = \emptyset, C_j \cap t_k = \emptyset, t_k \cap t_l = \emptyset$.*

In order to classify the templates, we need to discuss the box notation in more detail. If the box is found in a network, there are certain rules for changing the network when the operation of EXOR or SWAP is assigned to the box.

- If the assignment was EXOR, then the box is substituted with the EXOR symbol. If the line with the box assigned EXOR contains other box symbols, they are all substituted with EXOR.
- If the assignment was SWAP, the line with the box becomes the two lines, where the symbol SWAP is put. Every occurrence of a control on the line with this box is substituted with two controls and every occurrence of the box symbol is substituted with SWAP. EXOR symbol cannot appear in this line, since by the first item, had it be there, all the boxes would be substituted with EXOR, thus SWAP substitution will be incorrect initially.

Further, if a box symbol in a circuit is not specified, it can be either EXOR or SWAP which are substituted into the circuit by the above rules.

m=1. There are no templates of size 1, since every gate changes at least two input patterns.

m=2. There is one class of templates size 2 the **duplication deletion rule**, AA, which is defined as $G(S_1, B_1)\ G(S_1, B_1)$. This class is a generalization of the duplication deletion rule [6] and it is true for any two gates which perform a self-inverse transformation. In disjoint notations this class can be written as two formulas, one for two Toffoli gates and one for two Fredkin gates: $TOF(C_1, t_1)\ TOF(C_1, t_1)$ and $FRE(C_1, t_1 + t_2)\ FRE(C_1, t_1 + t_2)$ shown in Fig. 3.

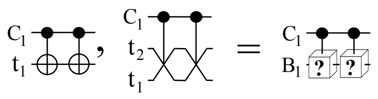


Figure 3: Class AA.

m=3. There are no templates of size 3.
m=4. There are several classes of size 4 templates.

- A very important class is the **passing rule**, a class ABAB (analogy of the passing rule from [6]) $G(S_1, B_1)\ G(S_2, B_2)\ G(S_1, B_1)\ G(S_2, B_2)$ with conditions: $(S_2 \cap B_1 = \emptyset, S_1 \cap B_2 = \emptyset, B_1 = B_2)$ OR $(S_2 \cap B_1 = \emptyset, S_1 \cap B_2 = \emptyset, B_1 \cap B_2 = \emptyset)$ OR $(|B_1| = 2, B_1 \subseteq S_2)$. There can be a shorter but less formal condition: the template $G(S_1, B_1)\ G(S_2, B_2)\ G(S_1, B_1)\ G(S_2, B_2)$ exists if for the first (if there are two with this property) line containing a control (dot) and a BOX, the BOX is SWAP, and sets B_1, B_2 either disjoint or equal. All the cases are shown in Fig. 4. The first part of the OR condition covers the first picture, the second OR condition describes the second. The third and fourth pictures illustrate the case when the third condition holds.

  There is a regular procedure for finding all the templates of the form ABAB. Since ABAB is the identity, the circuit produced by the sequence of gates AB must be a self-inverse permutation. The search of the templates of the form ABAB, thus, becomes equivalent to the search of self-inverse permutations that can be realized by two different gates.

- The following sets of templates can be treated as one,

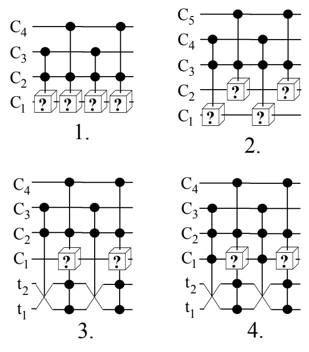


Figure 4: Class ABAB.

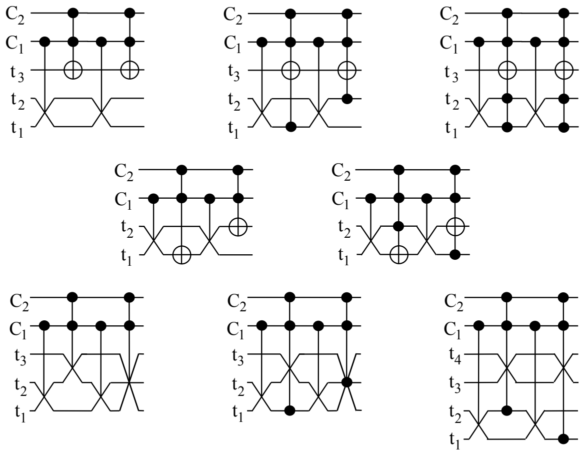


Figure 5: A group of semi passes.

two or even three classes, depending on one's view of the templates. The sets are:

– Semi passing rule: a group FAFB of gates $FRE(S_1, B_1)$ $G(S_2, B_2)$ $FRE(S_1, B_1)$ $G(S_3, B_3)$ with conditions $S_1 \subseteq S_2$, $B_2 \nsubseteq S_1$, and gate $G(S_3, B_3)$ is the gate $G(S_2, B_2)$ with controls and targets permuted according to the swap operation defined by the 2-bit set B_1. This description clarifies the name of the group - if the template is applied for parameter $k = 2$, the change of the network that we see can be described by: gates $FRE(S_1, B_1)$ and $G(S_2, B_2)$ are interchanged, but gate $G(S_2, B_2)$ may be slightly changed. The above group of gates has a non-empty intersection with the passing rule class. For example, the second template in Fig. 4 where the first box is Fredkin and the second is Toffoli and the set C_4 is empty, is a template of a semi-passing group. The new templates added by this group are shown in Fig. 5. Note, some of the semi-passes leave the gate $G(S_2, B_2)$ unchanged. Also, if we take the set of all semi-passing group templates and subtract the set of all templates of the passing rule group, the resulting set will have the semi-passing group templates where the second gate always changes.

– A group can be treated as the **definition of the**

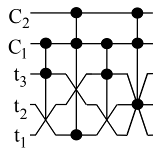


Figure 6: Link group.

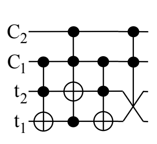
Fredkin gate in terms of Toffoli gates, TTTF, $TOF(S_1, B_1)$ $TOF(S_2, B_2)$ $TOF(S_1, B_1)$ $FRE(S_3, B_3)$ with conditions $B_1 \subseteq S_2, B_2 \subseteq S_1$, $B_1 \neq B_2$, $(S_1 \setminus B_1) \subseteq (S_2 \setminus B_2)$, $S_3 = S_2 \setminus (B_1 \cup B_2), B_3 = B_1 \cup B_2$. Pictorial representation of this class can be found in Fig. 7.

– And a **link** between the semi passing rule and Fredkin definition groups, group of templates FFFF, $FRE(S_1, B_1)$ $FRE(S_2, B_2)$ $FRE(S_1, B_1)$ $FRE(S_3, B_3)$ with conditions: $|S_1 \cap S_2| = 1$, $S_1 \cap B_2 \neq \emptyset$, $S_1 \cap B_2 \neq \emptyset$, $(S_1 \cup B_1) \subseteq (S_2 \cup B_2)$ and $FRE(S_3, B_3)$ is the gate $FRE(S_1, B_1)$ with the controls and targets permuted by the swap defined by the set B_1 (Fig. 6). This group is not a part of the semi passing rule, since, for instance, condition $S_1 \in S_2$ does not hold, but essentially it is doing the same thing. For a certain configuration of the first two gates, it allows passing of one gate through the other by permuting the elements of one gate. From the other point of view, this group is similar to the Fredkin definition group in the sense that if we cut out line t_2 and each occurrence of half of the SWAP (which requires two lines, so the half is one line) change with EXOR symbol, it results in getting the Fredkin definition group.

Given these three groups, the classification is not unique. We suggest unifying the semi passing group with the link group under the name of **semi passing class** and leaving the Fredkin definition group as a separate **Fredkin definition class**. If the reader is not comfortable with this classification, other classifications are possible with the only condition of semi passing and Fredkin definition group can be in one class only if the link group is a part of it.

m=5. Amazingly, there is only one class of templates of size 5. Class $ATATB$, $G(S_1, B_1)$ $TOF(S_2, B_2)$ $G(S_1, B_1)$ $TOF(S_2, B_2)$ $G(S_3, B_3)$ has conditions $B_2 \subseteq S_1$, $B_1 \nsubseteq S_2$, $S_3 = (S_1 \cup S_2) \setminus B_2$, $B_3 = B_1$. Although this is the largest class we have, and one would expect to see less applications of larger classes, since it is harder to match them then to match smaller classes, in practice this class is the most useful. The pictorial representation of this class is shown in Fig. 7.

m=6. Using the idea of regular search for the ABAB type templates, it was possible to find and generalize a template of size 6 of the form ABCABC, where ABC is a self-inverse permutation. Using this idea, the template $FTTFTT$ of the form $FRE(S_1, B_1)$ $TOF(S_2, B_2)$ $TOF(S_3, B_3)$ $FRE(S_1, B_1)$ $TOF(S_2, B_2)$ $TOF(S_3, B_3)$ with conditions $B_2 \subseteq B_1$, $S_2 \cap B_1 = \emptyset$, $B_3 \subseteq B_1$, $B_2 \neq B_1$, $S_3 \cap B_1 = \emptyset$, $(S_2 \triangle S_3) \subseteq S_1$. This class is illustrated in Fig. 8. The program which searches for the self dual functions of size three has found



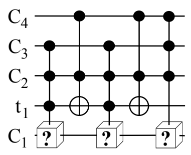


Figure 7: Fredkin definition group and class $ATATB$.

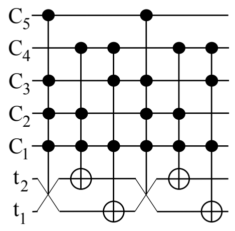


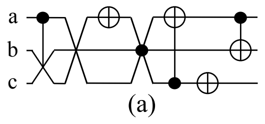
Figure 8: Class $FTTFTT$.

only those functions that are described by the presented template or circuits which can be simplified by other templates. Thus, we conclude that we built all the size 6 templates of the form ABCABC.

Applying the templates to the circuits shown in Fig. 2 results in the two circuits shown in Fig. 9. The circuit of size 12 (Fig. 2a) resulted in a circuit of size 7 (Fig. 9a). The second circuit (Fig. 9b) was reduced to size 6 from the original 7 (Fig. 2b).

5. RESULTS

We have written a program that synthesizes functions using the bidirectional algorithm and then, applies the template tool as a primary circuit simplification procedure. We ran our program exhaustively for all reversible functions with 3 variables and compared the results of our algorithm to the results of optimal synthesis. Table 3 shows how many functions with 3 inputs can be realized with $k = 0..10$ gates in optimal synthesis with the model gates NOT, CNOT, Toffoli, SWAP and Fredkin, (calculated in [2]), optimal synthesis with the model gates NOT, CNOT and Toffoli (calculated in [8]), optimal synthesis with the model gates NOT, CNOT, Toffoli and SWAP (calculated in [8]), our previous results of the heuristic synthesis presented in [2] and for the presented algorithm realization. WA shows the weighted average of the circuit size of a three variable reversible function. Note, our algorithm produces the circuits which on average are 105.9% of the optimal size, which in comparison with the previous realization (111.5%) is almost twice as close to the optimal. The 105.9% difference from the optimal circuit allows us to say that our algorithm produces near optimal circuits for reversible functions of a small number of variables. Also note, that even the heuristic synthesis of Toffoli/Fredkin networks (Algorithm column) produces a better weighted average then the synthesis of Toffoli networks (NCT column) only. The program can be used to synthesize functions with $n > 3$. The algorithm does not impose any limits on the number of variables. However, since the truth table must be stored, functions with more than 16 varialbes may require too much space and time.



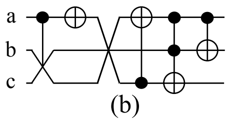


Figure 9: Circuits for 3_17.*pla* after simplification.

Size	Optimal	NCT	NCTS	CCECE	Algorithm
0	1	1	1	1	1
1	18	12	15	18	18
2	184	102	134	175	184
3	1318	625	844	1105	1290
4	6474	2780	3752	4437	5680
5	17695	8921	11194	10595	13209
6	14134	17049	17531	13606	13914
7	496	10253	6817	8419	5503
8	0	577	32	1877	512
9	0	0	0	86	9
10	0	0	0	1	0
WA:	5.134	5.866	5.629	5.724	5.437

Table 3: Number of reversible functions using a specified number of gates for $n = 3$.

6. REFERENCES

[1] G. W. Dueck and D. Maslov. Reversible function synthesis with minimum garbage outputs. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, pages 154–161, March 2003.

[2] G. W. Dueck, D. Maslov, and D. M. Miller. Transformation-based synthesis of networks of toffoli/fredkin gates. In *IEEE Canadian Conference on Electrical and Computer Engineering*, Montreal, Canada, May 2003.

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

[4] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing cnot-based quantum circuits. In *Proceedings of the Design Automation Conference*, New Orleans, Louisiana, USA, June 10-14 2002.

[5] D. M. Miller and G. W. Dueck. Spectral techniques for reversible logic synthesis. In *6th International Symposium on Representations and Methodology of Future Computing Technologies*, March 2003.

[6] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Proceedings of the Design Automation Conference*, June 2003.

[7] A. Mishchenko and M. Perkowski. Logic synthesis of reversible wave cascades. In *International Workshop on Logic Synthesis*, June 2002.

[8] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Reversible logic circuit synthesis. In *ICCAD*, pages 125–132, San Jose, California, USA, Nov 10-14 2002.

[9] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.