Electronic Colloquium on Computational Complexity, Report No. 128 (2005)

The normal form of reversible circuits consisting of CNOT and NOT gates

Miroslava Sotáková*

Abstract

This paper deals with the reversible circuits with n input and output nodes, consisting of the reversible gates FAN-IN=FAN-OUT≤ 2. We define a normal form of such type of circuits and describe a reduction algorithm to transform a circuit in this form. Furthermore we use it for checking whether two circuits are equivalent to each other without evaluating them.

1 Introduction

Reversible computing is computing which uses reversible operations, that means the operations which can be exactly reversed. When this kind of reversibility is performed at the lowest level, in the physical mechanisms of operation of the bit-devices, it avoids dissipating the energy that is associated with the manipulated bits of information. This can help to reduce the energy dissipation of computations.

Reversible computing is also closely connected to the quantum computing which is a hot topic nowadays, mainly because it brings the exponential improvement of the time complexity of several classical algorithms, like factorization.

In the area of reversible computing, Landauer [Lan61] was the first who described the Landauer embedding which is the naive technique to transform irreversible computations into equivalent reversible ones, but his machines do not reversibly get rid of the redundant computation histories. Bennett [Ben73] invented a method to uncompute them and thereby proved that reversible computations could prevent the entropy generation. Fredkin [FT78] introduced conservative logic circuits, and proved they were universal. Toffoli [Tof80] invented the Toffoli gate (also called the controlled-controlled-not gate) which is probably the most convenient universal reversible logic gate. All these works are the headstones of the field of reversible computing.

This paper deals with the reversible circuits with n input and output nodes, consisting of the reversible gates FAN-IN=FAN-OUT≤ 2. Although they are not universal and therefore such circuits are not too powerful, describing their structure can be the first step to towards discovering the structure of general reversible circuits consisting of unniversal ternary gates. More precisely, we search for the formal rules which enable us to recognize the equivalent reversible circuits. It is known that for the first order propositional logic there are finite sets of rewrite rules, which enable us do decide, whether the two given Boolean formulae are equivalent, i.e. their equivalence is a tautology. They use the properties of propositional connectives, like de Morgan rules, associativity, distributivity, idempotency of AND and OR, double negation, properties of the constants 1 and 0, definition of the other propositional connectives and some other inference rules. Using these rules it is possible to rewrite a given formula to the conjunctive normal form. Once a formula is in such form, it suffices to use only one – the resolution inference rule which is refutation-complete over first order logic that means it derives 0 (false symbol) if and only if the

*mirka@brics.dk, Department of Computer Science, Aarhus University

ISSN 1433-8092

given formula is an anti-tautology. If a Boolean formula is given, then we are able do decide, whether the negation of it is an antitautology and thereby find, whether the given formula is a tautology. This process is local, because it always deals with a bounded number of connectives, represented by the gates in the logic circuits.

Is it possible to use the same principle in different calculi? More precisely, we ask, whether there exists a finite rewrite system, which enables us to decide the problem of equivalence of two reversible circuits consisting of unary and binary reversible gates (two such circuits are equivalent if their outputs are equal for the same input). In the third section we introduce the basic technique to rewrite the product of transpositions generating the symmetric group S_n to the unique form. We call this form the *normal representation* of the permutation.

In the fourth section we extend this technique to deal with the circuits containing unary and binary gates. Firstly we focus on the circuits acting on n bits, consisting of *linear* gates. The transformations, which are computable by such circuits are in 1-1 correspondence with the non-singular $n \times n$ matrices over Z_2. This group is generated by elementary linear transformations, i.e. changing two columns and adding a column to another one, written as matrices. The binary gates corresponding to these transformations are called the *linear* gates. We present a finite rewrite system, which allows us to rewrite a linear circuit to its *normal* form which is unique for the empty circuit.

Rewriting an arbitrary circuit to its normal form is the keypoint of the algorithm for checking the equivalence of reversible circuits consisting of linear gates. Only a slight generalization is needed to deal with the circuits consisting of general binary gates.

2 Preliminaries

Syntactic monoids

Let X be an arbitrary set. A *word* over X is a finite string of symbols from X, in which repetition is allowed. The set of words with concatenation and the unit element, which is an empty word is called a *free monoid* over X. It is *generated* by X.

String rewriting system

Let T be a free monoid. A string rewriting system for T is a set R of ordered pairs of elements of T of the form (p, q). Viewed as a set of symmetric relations, there is the least congruence on T, which contains it. Let us denote it by $\tilde{R}$. If T is a free monoid over X, then (X, R) is called a *presentation* of the factor monoid $T/\tilde{R}$ of T. The set of relations, which generate this congruence is called the set of *defining relations* of $T/\tilde{R}$ on T.

The ordered pairs correspond to the *rewrite rules*. Suppose an element u of T has a subword p and (p, q) is a rule of the rewriting system, then we can replace the substring p of u by the word q and obtain a new word v. The word u was rewritten to the word v. u and v represent the same element of A. If u cannot be rewritten using any rule of the rewriting system we say that u is *reduced*.

The equality of two circuits in the free monoid of reversible circuits will denote the bidirectional rewrite rule. For instance, in the fourth section we will associate a monoid of *linear* circuits with n input nodes with the corresponding factor monoid – $GL_n(\mathbb{Z}_2)$ group using the finite set of rewrite rules and the corresponding set of defining relations, which determines the congruence on the set of linear circuits.

Reversible circuits

Definition 2.1 *An n-bit* reversible gate *is a bijective mapping f from the set* $\{0,1\}^n$ *of bit-strings of size n to itself.*

Definition 2.2 Reversible circuit *is a circuit which consists of reversible gates. As in the reversible gate, we need as many outputs as inputs in the reversible circuit.*

Definition 2.3 *A* width *of a reversible circuit is the number of bits in its input.*

3 Defining relations for the symmetric group

Definition 3.1 *The group* S_n *is a finite group of order* $n!$ *whose elements are permutations of the set of integers* $\{1,\dots,n\}$ *and whose group operation is a composition of permutations. A* cycle *of length* k *denoted by a* k*-tuple* $(a_1,\dots,a_k)$ *is a permutation* π*, where* $\pi(a_{i \bmod k}) = a_{(i+1) \bmod k}$*, and* $\pi(b) = b$ *if* b *is not in* $\{a_1,\dots,a_k\}$*. A* transposition *is a cycle of length* 2*.*

Remark: Any element of S_n can be written as a product of disjoint cycles. Any cycle the length k is a product of $(k-1)$ transpositions

(e.g. $(1,2\dots k) = (1,k)(1,k-1)\dots(1,2)$). The S_n group is thereby generated by the set of transpositions.

Proposition 3.2 *Let* X *be the set of transpositions in* $\{1,\dots,n\}$ *and let* $.$ *be the operation of concatenation in the free monoid generated by* X*. Let us denote it by* T*. The* S_n *group is a factor monoid of* T *through the congruence defined by the following relations:*

$$\begin{aligned}
(a\,b)(c\,d) &= (c\,d)(a\,b), \quad \forall a \neq b, c \neq d, \{a,b\} \cap \{c,d\} = \emptyset \\
(a\,b)(a\,c) &= (a\,c)(b\,c), \quad \forall a \neq b, a \neq c, b \neq c \\
(a\,b)(a\,b) &= id, \quad \forall a \neq b.
\end{aligned}$$

Before proving the proposition let us introduce a *normal representation* of a permutation. Proposition 3.2 is a consequence of Proposition 3.4 which describes the reduction of a given product of transpositions to the normal representation of the permutation which it defines.

Definition 3.3 *A* normal representation *of the permutation* π *is a product of transpositions in the form:*

$$\bar{\pi} = \prod_{i=1}^{k} \prod_{j=1}^{l_i - 1} (a_0^i\, a_j^i);$$

$$a_0^1 < a_0^2 < \dots < a_0^k;$$
$$(\forall i \in \{1,2,\dots,k\}) a_0^i = \min\{a_0^i, a_1^i, \dots a_{l_i-1}^i\};$$

$$(\forall i \in \{1,2,\dots,k\})(\forall r,s \in \{0,1,\dots,l_i-1\})\ r \neq s \Rightarrow a_r^i \neq a_s^i;$$
$$(\forall i,j \in \{1,2,\dots,k\})\ i \neq j \Rightarrow \{a_0^i, a_1^i, \dots, a_{l_i-1}^i\} \cap \{a_0^j, a_1^j, \dots, a_{l_j-1}^j\} = \emptyset.$$

In fact π has k cycles, each of them written in the form

$$(a_0\, a_1, \ldots, a_{l-1}) = (a_0\, a_{l-1})(a_0\, a_{l-2}) \ldots (a_0\, a_1), a_0 = \min\{a_0, a_1, \ldots a_{l-1}\}.$$

The cycles are ordered with respect to their least elements.

Remark: Every permutation $\pi \in S_n$, as a product of disjoint cycles, can be written in the normal form, denoted by $\bar{\pi}$.

Notation: If $\pi \in S_n$ is given, then let $\hat{\pi}$ denote some representation of π as a product of transpositions.

Proposition 3.4 *For the representation $\hat{\pi}$ there are representations*

$$\hat{\pi} = \hat{\pi}_0, \hat{\pi}_1, \ldots, \hat{\pi}_r = \bar{\pi},$$

such that for each $i \in \{0, 1, \ldots, r-1\}$ one of the following cases holds for some $\hat{\sigma}$ and $\hat{\varphi}$:

$$\begin{aligned}
\hat{\pi}_i &= \hat{\sigma}(a\, b)(c\, d)\hat{\varphi} & \hat{\pi}_{i+1} &= \hat{\sigma}(c\, d)(a\, b)\hat{\varphi} \\
\hat{\pi}_i &= \hat{\sigma}(a\, b)(a\, c)\hat{\varphi} & \hat{\pi}_{i+1} &= \hat{\sigma}(a\, c)(b\, c)\hat{\varphi} \\
\hat{\pi}_i &= \hat{\sigma}(a\, c)(b\, c)\hat{\varphi} & \hat{\pi}_{i+1} &= \hat{\sigma}(a\, b)(a\, c)\hat{\varphi} \\
\hat{\pi}_i &= \hat{\sigma}(a\, b)(a\, b)\hat{\varphi} & \hat{\pi}_{i+1} &= \hat{\sigma}\hat{\varphi}
\end{aligned}$$

Proof: For $\hat{\pi} = (a_1\, b_1) \ldots (a_m\, b_m)$ let us define

$$\mathrm{dom}\hat{\pi} = \bigcup_{i=1}^{m} \{a_i\, b_i\} \subseteq \{1, \ldots, n\}.$$

Let us proceed by induction according to the number of transpositions in the product.

We will use a *cycle-extraction*, that means we will always pick an element and construct a representation of the cycle which it is contained in. We will do it in such a way that each transposition of this representation will contain the chosen element. Let such an element be called the *leading* element of the cycle representation. We will extract the cycles respectively according to their least elements (which will be the leading elements of their representations) and we will order the corresponding transposition products from the left. At first we will construct the normal representation of the cycle containing the smallest element which is not a fixed point. Normal representations of the other cycles can be constructed analogously.

If there is only one transposition in the product, there is nothing to solve, because it has only one cycle of length 2 and it is already in the normal form. Assume we are able to extract an arbitrary cycle of a product of at most m transpositions by constructing the representation with an arbitrary leading element which it contains. Let us prove that we are able to do it for $(m+1)$ transpositions.

For the given representation $\hat{\pi}$ which is a product of $(m+1)$ transpositions let us define $e = \min \mathrm{dom}\hat{\pi}$. It is possible to transform this representation to the equivalent one containing e in the leftmost transposition, using the rules

$$(y\, z)(e\, x) \to (e\, x)(y\, z),$$
$$(x\, z)(e\, x) \to (e\, z)(x\, z).$$

If there is still a transposition containing e outside the left (connected) block (e.g. $(e\, a)$), it can be joined to the right side of this block by an analogous process. If there is $(e\, a)$ which has at least two occurrences

in this block afterwards, take the rightmost such pair (there is no one behind the left gate of this pair in the left block). Then use the rule

$$(e\,x)(e\,a) \to (e\,a)(x\,a)$$

to bubble the second $(e\,a)$ from the left to the leftmost one and cancel them both. Connected left block of transpositions containing e would be divided to two parts separated by a block of transpositions containing a by this operation. Fortunately each transposition of this block commutes with each transposition from the right part of divided left block (they are always disjoint), hence they can be rearranged to form the new (shorter than before) connected block of transpositions containing e, followed by a block of transpositions containing a. The number of $(e\,a)$-elements is two less than before this operation. This operations can be repeated until the number of elements in the form $(e\,x)$ is at most one for each x in the left block built of transpositions containing e.

Having such a left block it is necessary to get rid of the transpositions which are incident with its domain and do not belong to it. Let us look at the leftmost one (not containing e). Two cases can happen.

1. In the first one, both of the elements of the investigated transposition are involved in the left block. Let us call this transposition $(a\,b)$. In this case, multiplying the left block by $(a\,b)$ from the right will divide the cycle defined by this block to two cycles. Formally, we will use the rule

$$(x\,y)(a\,b) \to (a\,b)(x\,y)$$

to bubble $(a\,b)$ to the first occurrence (WLOG) $(e\,b)$. Then use

$$(e\,b)(a\,b) \to (e\,a)(e\,b).$$

Afterwards bubble $(e\,a)$ to its leftmost occurrence by

$$(e\,x)(e\,a) \to (e\,a)(a\,x)$$

and cancel them both. Finally change the order of the right block of transpositions containing e and a block of transpositions containing a. It can be done, because the transpositions from these blocks commute. In this case no transposition standing to the left or on an original position of $(a\,b)$ is incident with a block containing e in each transposition.

2. In the second case only one of the elements in the investigated transposition is already involved in the left block. The permutation defined by the product of transpositions standing to the left from such transposition, denoted by $(a\,b)$, contains a and b in different cycles. Multiplying by $(a\,b)$ joins these two cycles to the one, which will be defined by the left block. WLOG let a be involved in some $(e\,a)$. Let us look at the sequence of transpositions between $(a\,b)$ (including (a, b)) and the left $(e\,\_)$-block. It contains less then $(m+1)$ transpositions, thus it is possible to extract the cycle containing a (in each transposition of the corresponding transposition product) and place it to the left, just next to the $(e\,\_)$-block.

Then it is possible to move these transpositions commutatively (one by one) to $(e\,a)$ and use

$$(e\,a)(a\,x) \to (e\,x)(e\,a)$$

to make the block containing e in each transposition connected. After repeating this procedure, until we get rid of the transpositions containing a and not e in the right side of the block, it could be easily seen, there is no transposition standing in front of or on the original position of $(a\,b)$, but outside the left block, containing any of the newly added elements of the domain of the left block.

Proof of Proposition 3.2: It is possible to transform each representation of a permutation to the normal one using the rewrite rules given in Proposition 3.4. This representation is uniquely determined

by uniqueness of the cycle-factorization of a permutation. The relations determine a congruence of the free monoid T_n generated by the transpositions on the set of n elements. Each congruence class is represented by a normal representation of a permutation. The mapping which maps the congruence class T_n/R, where R is a congruence generated by the given relations, to the corresponding element of S_n is clearly an isomorphism.

4 Circuits consisting of CNOT, NOT and swap gates

Linear circuits

Let us have a reversible circuit of width n containing one type of logical gates

$$\texttt{CNOT} : (a, b) \to (a, a \oplus b),$$

where a, b are both 0 or 1 and $\oplus$ denotes adding in $\mathbb{Z}_2$.

If we assume the positions of bits in the input to be ordered, then two instances of CNOT-gate can be distinguished. If $a < b$, then

$$\texttt{LCNOT} : (a, b) \to (a \oplus b, b),$$

$$\texttt{RCNOT} : (a, b) \to (a, a \oplus b).$$

Remark that using these two gates it is possible to transpose the operands in the following way:

$$\texttt{RCNOT} \circ \texttt{LCNOT} \circ \texttt{RCNOT} : (a, b) \to (b, a).$$

Thereby we define the swap gate

$$\texttt{C} : (a, b) \to (b, a)$$

which will stand for the previous three-gate composition.

The output of a circuit consisting only of CNOT-gates, acting on

$$(a_1, \ldots, a_n)$$

is

$$\left(\sum_{i=1}^{n} c_i^1 a_i, \sum_{i=1}^{n} c_i^2 a_i, \ldots, \sum_{i=1}^{n} c_i^n a_i \right),$$

where all c_i^j-s are 0 or 1, that means each coordinate is a linear combination of the input bits over $\mathbb{Z}_2$.

Instead such logical circuit β, it is possible to deal with the non-singular matrix $B_{n\times n}$ transforming the input vector to the output one, having the entries c_i^j, i.e. the j-th column of the matrix determines the j-th output bit. Of course, such a non-singular matrix represents infinitely many circuits. The C- and CNOT-gates correspond respectively to changing the order of two columns and adding a column to another one, which are elementary linear transformations. Each non-singular matrix over $\mathbb{Z}_2$ can be created from the identity matrix via these elementary transformations. Hence a C-gate is constructible

from CNOT-gates, then each non-singular $n \times n$ matrix corresponds to a circuit of width n consisting of CNOT-gates. Thus there are exactly

$$(2^n - 1)(2^n - 2)\dots(2^n - 2^{n-1}) = 2^{n(n-1)/2}\prod_{i=1}^{n}(2^i - 1)$$

inequivalent circuits consisting of this type of logical gates, in the sense that for each pair of such circuits there is at least one input, for which the outputs differ. This equivalence is a congruence of the free monoid of circuits and in the case of linear circuits the corresponding factor monoid is isomorphic to the group of non-singular $n \times n$ matrices over $\mathbb{Z}_2$. Therefore let us call these circuits the *linear circuits* and the appropriate transformations (resp. gates) the *linear transformations (gates).*

Remark: $\texttt{LCNOT}(a,b) = \texttt{C} \circ \texttt{RCNOT} \circ \texttt{C}(a,b) = \texttt{RCNOT} \circ \texttt{C} \circ \texttt{RCNOT}(a,b)$. That means that each circuit consisting of CNOT-gates can be rewritten to a circuit which contains only C- and RCNOT-gates.

Notation: In further text C, RCNOT and LCNOT-gates operating on a, b will be denoted by $[a,b]$,$(a,b\rangle$,$\langle a,b)$ respectively.

Theorem 4.1 *In the free monoid of linear circuits of width n over*

$$\{[a_i, a_j], (a_i, a_j\rangle; i \neq j; i, j \in \{1, \dots, n\}\}$$

the following equations give a finite presentation of $GL_n(\mathbb{Z}_2)$ group.

$$\begin{aligned}
[a,b][c,d] &\simeq [c,d][a,b] && (1)\\
[a,b]\,(c,d\rangle &\simeq (c,d\rangle[a,b] && (2)\\
(a,b\rangle[c,d] &\simeq [c,d](a,b\rangle && (3)\\
(a,b\rangle(c,d\rangle &\simeq (c,d\rangle(a,b\rangle && (4)\\
[a,b][a,b] &\simeq (a,b\rangle(a,b\rangle \simeq 1 && (5)\\
[a,b](a,b\rangle[a,b] &\simeq (a,b\rangle[a,b](a,b\rangle && (6)\\
[a,b][b,c] &\simeq [a,c][a,b] && (7)\\
[b,c][a,b] &\simeq [a,c][b,c] && (8)\\
[a,b](b,c\rangle &\simeq (a,c\rangle[a,b] && (9)\\
[b,c](a,b\rangle &\simeq (a,c\rangle[b,c] && (10)\\
(a,b\rangle(b,c\rangle &\simeq (b,c\rangle(a,c\rangle(a,b\rangle && (11)
\end{aligned}$$

Different letters denote different positions in $\{a_1, \dots, a_n\}$. In addition, starting from (5) the bits are ordered (a,b,c) in the input.

Note: The given relations will be used as the bidirectional rewrite rules for the words in the free monoid of linear circuit.

Lemma 4.2 *In the group of transformations generated by* $[\_,\_]$ *and* $(\_,\_\rangle$ *satisfying the relations from the previous theorem the following identities hold for the ordered triple* (a,b,c) *of bits in the input:*

$$\begin{aligned}
[a,c][a,b] &\simeq [b,c][a,c] && (12)\\
[a,c][b,c] &\simeq [a,b][a,c] && (13)
\end{aligned}$$

$$\begin{aligned}
(b,c\rangle[a,c] &\simeq [a,c](a,b\rangle[a,b](a,b\rangle \simeq [a,c]\langle a,b) && (14)\\
(a,b\rangle[a,c] &\simeq [a,c](b,c\rangle[b,c](b,c\rangle \simeq [a,c]\langle b,c) && (15)\\
[a,c](b,c\rangle &\simeq [a,b](a,b\rangle[a,b][a,c] \simeq \langle a,b)[a,c] && (16)\\
[a,c](a,b\rangle &\simeq [b,c](b,c\rangle[b,c][a,c] \simeq \langle b,c)[a,c] && (17)
\end{aligned}$$

$$\begin{aligned}
(a,b\rangle(a,c\rangle &\simeq (a,c\rangle(a,b\rangle && (18)\\
(a,c\rangle(b,c\rangle &\simeq (b,c\rangle(a,c\rangle && (19)
\end{aligned}$$

Proof of the lemma: Thanks to (5) all equations hold if both of their sides are written in inverse order. (12) and (13) are the rules for multiplying transpositions which are implied by (7) and (8), These rules allow us to transform a permutation circuit to the normal form according to the previous section. (14) can be derived in such a way:

$$(b,c\rangle[a,c] \overset{5}{\simeq} [a,c][a,c]\cdot(b,c\rangle[a,c] \overset{7}{\simeq} [a,c]\cdot[a,b][b,c][a,c]\cdot(b,c\rangle[a,c] \overset{9}{\simeq} [a,c][a,b][b,c]\cdot(a,c\rangle[a,b]\cdot[a,c] \overset{10}{\simeq}$$

$[a,c][a,b]\cdot(a,b\rangle[b,c]\cdot[a,b][a,c] \overset{7,5}{\simeq} [a,c]\langle a,b)$.

(18) can be derived in the following way:

$$(a,b\rangle(a,c\rangle \overset{11}{\simeq} (b,c\rangle\cdot(a,b\rangle(b,c\rangle \overset{11}{\simeq} (b,c\rangle\cdot(b,c\rangle(a,c\rangle(a,b\rangle \overset{5}{\simeq} (a,c\rangle(a,b\rangle.$$

Remark: Using these relations allows us to transform each circuit containing `C-` and `RCNOT`-gates to the circuit consisting of the gates acting only on the pairs of neighbouring bits resp. on the pairs containing the rightmost resp. leftmost bit. The generating set of the gates operating on all pairs can be restricted in such a way.

Lemma 4.3 *For a non-singular* $n\times n$ *matrix* B *over* $\mathbb{Z}_2$ *representing a linear circuit there is a rearrangement of its columns, which gives* $\tilde{B}$*, such that* $\tilde{B}$ *arises from* E *(the identity matrix) in the following way:*

For $1\le i\le n$*, in the* i*-th step add some columns to the* $(n+1-i)$*-th column.*

Proof of the lemma: The given matrix is non-singular, i.e. it has non-zero determinant. Computing it via expansion according to the first row gives at least one non-zero element in the first row accompanied with a nonzero determinant of an $(n-1)\times(n-1)$ submatrix, obtained from an original one by deleting the first row and the appropriate column. Let us change the positions of the corresponding column and the first one. Apply this principle inductively, that means that in the i-th iteration choose the i-th column from the $(n+1-i)\times(n+1-i)$ submatrix placed right down analogously and change this column and the originally i-th one. The columns of B will be ordered into $\tilde{B}$ after $(n-1)$ iterations. In this rearrangement it is possible to replace the last i columns of E by the appropriate i columns of $\tilde{B}$ and still obtain non-singular matrix. In i-th step there is precisely one diagonal 1 in each of the first $(n-i)$

columns, thus the determinant of such matrix is equal to the determinant of the $i \times i$ submatrix which is an intersection of last i columns and rows. This one was constructed to have a non-zero determinant. The column vectors stay linearly independent after the replacement of the $(n+1-i)$-th column in the i-th step. Q.E.D.

Previous lemma suggests to define a normal form of a linear circuit.

Definition 4.4 *Let β be a circuit acting on $(a_1, \ldots, a_n)$. Let us call the equivalent circuit*

$$\bar{\beta} = \prod_{j=1}^{m} (a_{\sigma(j)}, a_n\rangle \left(\prod_{i=1}^{k} \left([a_{\pi(i)}, a_n] \prod_{j=1}^{l_i} (a_{\sigma_i(j)}, a_n\rangle \right) \right) \gamma,$$

where

$$k, m \geq 0, l_i > 0,$$
$$1 \leq \pi(i) \leq (n-1), i = 1 \ldots k,$$
$$i \neq j \Rightarrow \pi(i) \neq \pi(j),$$
$$n > \sigma(1) > \sigma(2) > \cdots > \sigma(m) > 0,$$
$$n > \sigma_i(1) > \sigma_i(2) > \cdots > \sigma_i(l_i) > 0$$

and γ is a circuit consisting only of `C`*-gates, corresponding to the normal form of permutation, the* normal form *of β.*

That means, that each `RCNOT`-gate (adding the columns) is operating on the rightmost bit, since then it is possible to add an arbitrary other bit, using one `RCNOT`-gate. At the end, the permutation circuit rearranges the obtained values in the correct order.

Corollary 4.5 *Each linear circuit can be written in a normal form.*

Proof of the corollary: Let the input and output bits of the circuit be $(a_1, \ldots, a_n)$, $(\sum_{i=1}^{n} c_i^1, \sum_{i=1}^{n} c_i^2, \ldots, \sum_{i=1}^{n} c_i^n)$, respectively. Define $B = (c_i^j)_{1 \leq i \leq n}$. Transform B into $\tilde{B}$ according to Lemma 4.3. In the i-th step compute the value defined by the $(n+1-i)$-column on the $(n+1-i)$-th bit, placing this bit to the right and computing the appropriate value by adding the other bits. After computing all the values rearrange the bits in the correct order. Q.E.D.

Remark: The rearrangement of columns from Lemma 4.3 is not uniquely determined, neither is the normal form of linear circuit. In the described case the matrix transformation gives a normal form where the left bits of `C`-gates are ordered from the right to the left. Still two such normal forms can differ from each other in the selected permutation of columns, i.e. in the order of output values, before using the permutation part γ. Moreover in the normal form, constructed via a matrix, the number of `C`-gates behind the first `RCNOT`-gate-block excluding the permutation part is minimal, because the bits are added to the bit only if the corresponding output value is a linear combination of at least two input values. Otherwise the bit stays unchanged during the first part of computation and is placed correctly by γ. Generally, definition of normal form does not force the number of `C`-gates to be minimal and the left bits of them to be ordered. We will present an algorithm to transform the linear circuit into the normal form according to the definition.

Proof of Theorem 4.1: It is necessary to prove that it is possible to verify the equivalence of each pair of two equivalent linear circuits via given relations. Then the equivalence relation on the set of linear circuit is determined by such transformations. The homomorphism between the equivalence classes of linear circuits of width n and $GL_n(\mathbb{Z}_2)$ is an isomorphism, hence the proposed set of relations on the

n-bit input is a presentation of the $GL_n(\mathbb{Z}_2)$ group in the free monoid of linear circuits. Because of reversibility, we only have to prove that for each pair O_1, O_2 of equivalent circuits, the circuit $O_1O_2^{-1}$ is equivalent to the identity via given relations. That would give us

$$O_1 \simeq (O_1O_2^{-1})O_2 \simeq O_2.$$

Let us transform an n-bit linear circuit to the one, having all gates acting on the rightmost bit, using (8) and (10), i.e.

$$\begin{aligned} [i,j] &\simeq [j,n][i,n][j,n] \\ (i,j\rangle &\simeq [j,n](i,n\rangle[j,n] \end{aligned}$$

Then cancel everything what is possible according to (5). If there are still two different C-gates one after the other, let us use (8) and (12) to transform such a pair into the new one having the second gate missing the rightmost bit. Then use (1), (2), (7) or (9) to bubble this gate to the bottom, where it becomes a part of C-circuit γ which can be transformed to define the normal form of permutation exactly as in the proof of Proposition 3.4 according to (7),(8),(12) and (13). Continue with cancelling the pairs of the same consecutive gates until it is no such pair in the circuit any more. Use the commutativity of the RCNOT-gates having the same last bit, according to (18) and (19) to get rid of the multiple occurrences of such gates between two C-gates. Now the circuit is in a form which ends with a permutation circuit and the beginning is built of blocks starting with a C-gate followed by a string of several RCNOT-gates, in which each such gate occurs at most once. In the first block the C-gate is not necessary. The maximal connected block of C-gates at the end of circuit will be called the **tail**, whereas the rest of circuit will be called the **head**. After performing the operationes, which were mentioned above, some C-gates can occur several times in the head, conversely to the normal form. After getting rid of their multiple occurrences we will reach the normal form.

Let $[i,n]$ be a gate which has at least two occurrences in the head-part. Let us take the first and the second ones. The second one is preceded by a block of $(\_,n\rangle$-gates acting on the rightmost bit. Let us call it Y_0. According to (6), (10) and (18) it is possible to transform the fragment $Y_0[i,n]$ into $[i,n]Z_0$, where Z_0 is a block of $(\_,i\rangle$- and $(i,\_)$-gates which add the bits to the i-th bit. In case Y contains $(i,n\rangle$, Z contains $\langle i,n)$ according to (6). The first $[i,n]$-gate from $[i,n]Z$ is preceded either by some $[j,n]$, where $j \neq i$, or by $[i,n]$. In the first case use

$$[j,n][i,n] \simeq [i,n][i,j]$$

Continue with bubbling the second $[i,n]$ through the next preceding block of $(\_,n\rangle$-gates, until it reaches the first occurrence of $[i,n]$-gate, i.e. the second case happens. In the second case cancel both $[i,n]$-gates according to (5).

During the bubbling process of the second $[i,n]$-gate to the first one, the fragment which was placed between two surveyed $[i,n]$-gates in the original circuit is transformed to $Z_m[i,j_{m-1}]Z_{m-1}[i,j_{m-2}]\dots Z_1[i,j_1]Z_0$, where Z_r are the blocks of $\langle i,\_)$- and $(\_,i\rangle$-gates. Let us take Z_0. The gates adding something to the i-th bit commute. If Z_0 contains a $\langle i,n)$-gate, let it be the first one in this block. Bubble all gates of Z_0 through the first gate which stands behind Z_0.

1. If it is $[i,n]$, each $\langle i,k)[i,n]$ becomes $[i,n](k,n\rangle$, and each $(k,i\rangle[i,n]$ becomes $[i,n](k,n\rangle$.

2. Otherwise, if the situation is not $\langle i,n)(i,n\rangle$, then $\langle i,k)(j,n\rangle$, $(k,i\rangle(j,n\rangle$, $\langle i,k)[j,n]$ and $(k,i\rangle[j,n]$ (some of different letters can denote the same bit) are transformed to PQ (preceded by $[j,n]$ in the last two cases), where P is a block of $(\_,n\rangle$-gates and Q is a block of $\langle i,\_)$- and $(\_,i\rangle$-gates. In fact, in

each step the next gate can be bubbled through several $(\_, n\rangle$-gates to reach the block of gates adding something to the i-th bit.

3. $\langle i, n)(i, n\rangle$ can be transformed to $(i, n\rangle[i, n]$. If such a situation happens, bubble the new $[i, n]$-gate to the end of the following block of $\langle i, \_)$- and $(\_, i\rangle$-gates. This action transforms this block to a block of $(\_, n\rangle$-gates.

At the end of such operation Z_0 is transformed either to some U_0 which is a block of $\langle i, \_)$- and $(\_, i\rangle$-gates, preceded by a block of $(\_, n\rangle$-gates, or to the block of $(\_, n\rangle$-gates. In the first case rearrange U_0 to have the $\langle i, n)$-gate at the beginning (if it is contained in it) and continue with bubbling U_0 to the right according to 1, 2 and 3.

Repeat the operation until the block of gates adding something to the i-th bit vanishes or such block reaches the beginning of the tail. In this case add $[i, n][i, n]$ to the front of the block of $\langle i, \_)$- and $(\_, i\rangle$-gates preceding the tail, denoted again by U_0. Move the second $[i, n]$ to the end of U_0 to become a part of the tail. U_0 is finally transformed to the block of $(\_, n\rangle$-gates.

At the end of such operation, $[i, j_1]$ is followed by the gates operating on the rightmost bit only, in each case. Thus it is possible to bubble it to the tail through the whole circuit. After doing this, the `RCNOT`-gates remain to work on the rightmost bit, all $[j_1, n]$-gates are changed to $[i, n]$-gates and all $[i, n]$-gates are changed to $[j_1, n]$-gates. The string $Z_m[i, j_{m-1}] \dots Z_1[i, j_1]Z_0$ has become shorter during this operation. If there was a $[i, j_1]$-gate to bubble, a new circuit's head has two `C`-gates less than the original one (two of $[i, n]$-s were canceled, one $[i, n]$ arose by 2. and 3., and was changed to $[j_1, n]$ afterwards, one $[i, j_1]$-gate was bubbled to the tail). Otherwise it has at least one gate less than before. In each step take the actually last Z-block and do this operation. At most one new `C`-gate, operating on the rightmost bit arises, but one `C`-gate $[i, j_r]$ is bubbled to the tail in each step, except the last one. In the last step, no $[i, j]$-gate is bubbled to the end of the head. Thus the head of the circuit has all gates operating on the rightmost bit and the number of `C`-gates is at least one less than before after $(m + 1)$ iterations.

It is possible to do this operation, if there is at least one `C`-gate with at least two occurrences in the head. Because each linear circuit has an equivalent form, in whose head every admissible `C`-gate occurs at most once, it has to be reached after finitely many iterations. An original circuit can be transformed in its normal form using the given relations. Q.E.D.

Remark: If the given circuit is equivalent to a permutation of the input bits, the normal form is uniquely determined, i.e. the head is empty and the tail is a permutation in its normal representation.

In a normal form of a circuit with a non-empty head, the output on the first bit which the other bits were added to, would be some linear combination of all bits, having at least two 1-s among the coefficients. This value does not change until the end of such circuit's computation, hence the output would be not a pure permutation of bits.

Consequently, the only normal form of a circuit which is equivalent to the identity is the empty circuit.

Remark: For the circuit O the inverse circuit O^{-1} consists of the same gates as O in inverse order, according to (5).

It is possible to verify the equivalence of the two circuits O_1 and O_2 via given relations in the following way. Add the circuit $O_2^{-1}O_2$ which is equivalent to the identity, to the end of the circuit O_1. Transform $O_1O_2^{-1}$ to its normal form according to the presented algorithm, to get an empty circuit. The result of this transformation is the circuit O_2.

All circuits having the same output are equivalent via relations, given by Theorem 4.1. Each equivalence class corresponds to a non-singular matrix whose columns represent the output values of the circuit.

Obviously each circuit has a representing matrix and each matrix corresponds to a circuit (even in the normal form). The mapping which maps the equivalence classes of the circuits to the corresponding matrices is an isomorphism of the groups. The set of relations from Theorem 4.1 is a presentation of the $GL_n(\mathbb{Z}_2)$ group in the free monoid of linear circuits with n input nodes. Q.E.D.

Adding the NOT gates

After discussing linear circuits composed of linear gates, it is natural to focus on all reversible circuits consisting of the gates with at most two inputs and outputs. Surely, the unary gate $\texttt{NOT} : a \to a \oplus 1$ belongs to the set of gates which are not constructible from CNOT-gates.

On the other hand, each reversible binary gate operating on two bits can be viewed as a permutation of four vectors of $\{0,1\}^2$. This can be represented by a 4×4 permutation matrix. An element of $\{0,1\}^2$ can be viewed as a number written in the binary system. For $i \in \{0,1,2,3\}$ let the corresponding element of $\{0,1\}^2$ be denoted by $(i)_2$. The the matrix A corresponding to the permutation π fulfills

$$a_{ij} = 1 \;\Leftrightarrow\; \pi((i)_2) = (j)_2.$$

All elements of S_4 - the permutation group on $\{0,1,2,3\}$, are generated by three transpositions e.g. $(0\,1), (1\,2), (2\,3)$. These correspond to the gates $\neg\texttt{RCNOT} : (a,b) \to (a, a \oplus b \oplus 1)$, C and RCNOT. The first gate can be also constructed using RCNOT and NOT.

That implies that the outputs of circuits acting on

$$(a_1, \ldots, a_n),$$

are in a form:

$$(e_1 \oplus \sum_{i=1}^{n} c_i^1 a_i, e_2 \oplus \sum_{i=1}^{n} c_i^2 a_i, \ldots, e_n \oplus \sum_{i=1}^{n} c_i^n a_i),$$

where c_i^j and e_j are 0 or 1. The outputs in the previously mentioned form correspond uniquely to the $(n+1) \times n$ matrices, whose entries are c_i^j if $i \geq 1$ and e_j in the row labelled by 0. The C-, RCNOT- and NOT-gates correspond to changing columns, adding the column to the right and adding 1 to the 0-th row of the matrix. These are up to the third one, the elementary linear transformations. The additional NOT-gates can be viewed as shifting of the origin of the system of coordinates. Therefore the circuits acting on n bits correspond to *affine* transformations in the n-dimensional vector space over Z_2.

It is possible to construct the circuit which corresponds to the matrix with an arbitrarily chosen 0-th row. Consequently there are

$$2^n 2^{n(n-1)/2} \prod_{i=1}^{n} (2^i - 1) = 2^{n(n+1)/2} \prod_{i=1}^{n} (2^i - 1)$$

inequivalent circuits containing only binary gates.

Let a, b, c be the ordered distinct positions in the input. Denoting $\texttt{NOT}(a)$ as $\tilde{a}$, this operator fulfills the following equations.

$$\tilde{\tilde{a}} \simeq 1 \tag{20}$$
$$\tilde{a}[b,c] \simeq [b,c]\tilde{a} \tag{21}$$
$$\tilde{a}(b,c\rangle \simeq (b,c\rangle\tilde{a} \tag{22}$$
$$\tilde{a}[a,b] \simeq [a,b]\tilde{b} \tag{23}$$
$$\tilde{b}(a,b\rangle \simeq (a,b\rangle\tilde{b} \tag{24}$$
$$\tilde{a}(a,b\rangle \simeq (a,b\rangle\tilde{a}\tilde{b}. \tag{25}$$

According to them, it is possible to separate all the NOT-gates from the others, and place them to the bottom of an arbitrary given circuit.

Theorem 4.6 *The set consisting of the equations 4.1-4.11 together with the equations 4.20-4.25 is a presentation of the group of reversible transformations which are constructible by binary gates.*

Proof: The theorem is a corollary of Theorem 4.1.

Remark: For each circuit consisting of unary and binary reversible gates there is an equivalent one, which size is in $O(n^2)$. The normal form consists of a head of at most n^2 gates, of a tail of at most n gates and eventually of at most n negations.

5 Summary

The main result of this work is contained in the fourth section. We introduce the finite set of rewrite rules which enables us to rewrite a given reversible circuit consisting of CNOT and NOT gates to its normal form containing $O(n^2)$ gates for a circuit with n input nodes. Thereby we can check whether two circuits are equivalent without evaluating them. It would be interesting to generalize this result to ternary circuits, that means find the finite set of rewrite rules which would identify the given circuit with its normal form, if such set exists. Probably it would be useful to use a finite number of auxiliary bits, which does not depend on the number of input nodes of a circuit.

References

[Ben73] C. H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

[FT78] E. Fredkin and T. Toffoli. Design principles for achieving high-performance submicron digital technologies. *MIT Laboratories for Computer Science to DARPA*, 1978.

[Lan61] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5:183–191, 1961.

[Tof80] T. Toffoli. Reversible computing. *Automata, Languages and Programming*, pages 632–634, 1980.

ECCC ISSN 1433-8092
http://www.eccc.uni-trier.de/eccc
ftp://ftp.eccc.uni-trier.de/pub/eccc
ftpmail@ftp.eccc.uni-trier.de, subject 'help eccc'