On computer algebra application to simulation of quantum computation

Vladimir P. Gerdt [1)], Robert Kragler [2)], Alexander N. Prokopenya [3)]

[1)] Joint Institute for Nuclear Research
141980 Dubna, Russia
gerdt@jinr.ru

[2)] University of Applied Sciences
D-88241 Weingarten, Germany
kragler@hs-weingarten.de

[3)] Brest State Technical University
Moskowskaya 267, 224017 Brest, Belarus
prokopenya@brest.by

Abstract. In the present paper which is an extended version of paper [1] we consider a *Mathematica*-based package for simulation of quantum circuits. It provides a user-friendly graphical interface to specify a quantum circuit, to draw it, and to construct the unitary matrix for quantum computation defined by the circuit. The matrix is computed by means of the linear algebra tools built-in *Mathematica*. For circuits composed from the Toffoli and Hadamard gates the package can also output the corresponding multivariate polynomial system over $\mathbb{F}_2$ whose number of solutions in $\mathbb{F}_2$ determines the circuit matrix. Thereby the matrix can also be constructed by applying to the polynomial system the Gröbner basis technique based on the corresponding functions built-in Mathematica. We illustrate the package and the method used by a number of examples.

1 Introduction

Quantum computations is a topic of great interest for the last two decades. One reason for this is a potential ability of a quantum computer to do certain computational task much more efficiently than can be done by any classical computer [2, 3]. Two the most famous examples of such calculations are Shor's algorithm [4] for efficient factorization of large integers and Grover's algorithm [5] of element search in an unsorted list. Nevertheless, despite of considerable efforts in the quantum computing community, the number of such efficient quantum algorithms which have been discovered still remains rather small. By this reason a search for other problems which may be efficiently solved with a quantum computer and developing the corresponding quantum algorithms, as well as the physical question of the feasibility of building a quantum computer, is a very topical direction of present-day investigations.

Since realistic quantum computers have not yet been built, it is worthwhile to simulate quantum computation on a classical computer, and there is quite a number of such simulators (see, for example, [6, 7]). Among two equivalent models of quantum computation – quantum Turing machine and the circuit model – the last one is more convenient both for simulation and application [2].

The circuit model of computation was introduced first for a classical computer that can be considered as an electrical circuit made up of wires and logical gates. The wires are used to carry information around the circuit, while the logic gates perform manipulations of the information, converting it from one form to another. Note that inside a classical computer any information is encoded into a sequence of bits which are the elementary units of information. And any complex logic operation can be represented as an ordered sequence of some elementary logic gates which act on single bits or pairs of bits. Using special notation for these gates, one can easily visualize a circuit and clearly show a structure of computations. Thus, the circuit model turned out to be very convenient and realistic for many applications and is widely used in computer science.

In the present paper we use the computer algebra system *Mathematica* [8] for simulation of quantum computation and develop a package which provides a user-friendly graphical interface to specify a quantum circuit, to draw the circuit specified, and to construct a unitary $2^n \times 2^n$ matrix U defined by the circuit with n qubits. In section 2 we discuss general structure of an arbitrary quantum circuit and introduce the basic logical gates commonly used in the quantum circuit model. In section 3 we describe an algorithm for the quantum circuits generation with *Mathematica*. And in section 4 we develop an algorithm for computing the unitary matrix defined by the circuit and implement it with *Mathematica*. In section 5 circuits of special type are considered which are composed from the Hadamard and Toffoli gates only. If a circuit is of this type our Mathematica package can generate the system of polynomial equations over $\mathbb{F}_2$ whose solution space in $\mathbb{F}_2$ uniquely determines the circuit matrix [9]. We show in section 6 how the built-in *Mathematica* Gröbner bases module can be used to to construct the circuit matrix in terms of the polynomial system. Throughout the paper we illustrate the methods used and the package by simple examples.

2 Structure and basic elements of quantum circuits

The circuit model is easily transferred to quantum computations by means of creating quantum analogues for the basic components of a classical computer [2]. Quantum information is represented as a sequence of quantum bits or *qubits* which are the elementary units of quantum information. A qubit is a two-level quantum system that can be prepared, manipulated and measured in a controlled way. The state of a qubit is denoted as $|a\rangle$ corresponding to standard Dirac notation for quantum

mechanical states. Two possible states for a qubit are usually denoted as $|0\rangle$ and $|1\rangle$, which correspond to the states 0 and 1 for a classical bit. But in contrast to classical bits, qubit as a quantum system may exist not only in one of the states $|0\rangle$ or $|1\rangle$ but also in the state $|a\rangle$ being a superposition of these states

$$|a\rangle = \alpha|0\rangle + \beta|1\rangle \ , \tag{1}$$

where α and β are complex numbers constrained by the normalization condition $|\alpha|^2 + |\beta|^2 = 1$. Thus, the state of a qubit is represented by the vector (1) in the two-dimensional complex vector space, where the special states $|0\rangle$ and $|1\rangle$ form an orthonormal basis and are known as computational basis states [2].

A set of n qubits forms a quantum memory register, where the input data and any intermediate results of computations are held. It is shown on diagrams as a column of states of the form $|a_j\rangle$ $(j = 1, 2, ..., n)$ from which quantum wires start. Although a quantum circuit doesn't contain any wires as such, the term "wires" is merely used to show evolution of qubits acted on by various quantum gates. General structure of any quantum circuit can be readily understood from Fig. 1, where a very simple quantum circuit containing two qubits and two quantum gates is depicted.

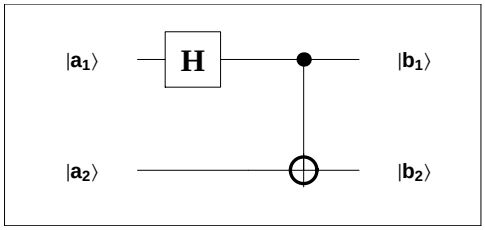

Figure 1: A simple quantum circuit

The circuit is to be read from left-to-right. It means that a column of two qubits $|a_1\rangle$ and $|a_2\rangle$ in the left-hand side of the diagram in Fig. 1 corresponds to the initial state of quantum register. Then it is successively acted on by two quantum gates and its final state is shown on the right-hand side of the diagram as the column of qubits $|b_1\rangle$ and $|b_2\rangle$. Note that a quantum circuit containing more qubits and quantum gates can be built in a similar way.

Hadamard gate	Pauli – X	Pauli – Y	Pauli – Z	Phase	π / 8 – gate
H	X	Y	Z	S	T
$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$

Figure 2: Single-qubit gates

As in the case of classical computation, there are two groups of the elementary quantum gates which perform manipulation of quantum information. The first group

consist of the single-qubit gates. Such gates have only one input and one output wires and are depicted by some capital letter placed into a square. Following to [2], we'll use here only six non-trivial single-qubit quantum gates which are shown in Fig. 2 together with their matrix representation with respect to the computational basis states. Using this set of single-qubit matrices one can construct any operation on a single qubit. Note that the matrix of any single-qubit gate shows how the gate acts on the states of the computational basis, for example,

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \ , \ H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \ . \tag{2}$$

The second group of elementary quantum gates consists of the gates acting on two and more qubits, i.e., the multi-qubit gates. Let us consider first two-qubit quantum gates. Remember that the state of one qubit is represented by the vector (1) in the two-dimensional complex vector space with the basis vectors $|0\rangle$ and $|1\rangle$. Hence, the state of the system of two independent qubits may be determined as a direct (tensor) product of two two-dimensional spaces associated with each qubit. Such a space has four basis vectors

$$\begin{aligned} |0\rangle_1 \otimes |0\rangle_2 \equiv |00\rangle \ , \ |0\rangle_1 \otimes |1\rangle_2 \equiv |01\rangle \ , \\ |1\rangle_1 \otimes |0\rangle_2 \equiv |10\rangle \ , \ |1\rangle_1 \otimes |1\rangle_2 \equiv |11\rangle \ , \end{aligned} \tag{3}$$

where the symbol $\otimes$ denotes a direct product of basis states associated with the first ($|0\rangle_1, |1\rangle_1$) and the second ($|0\rangle_2, |1\rangle_2$) qubits. Thus, the system of two qubits has a four-dimensional space of states with computational basis states (3), an arbitrary state of such a system may be represented as superposition of the form

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle \ , \tag{4}$$

where coefficients $\alpha, \beta, \gamma, \delta$ are complex numbers constrained by the normalization condition $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. Hence, any two-qubit quantum gate can be represented as a 4×4 matrix in the computational basis states (3).

One of the useful two-qubit gates is a controlled-NOT or CNOT gate, its graphical and matrix representation is shown in Fig. 3. This gate has two input qubits, known as the control qubit ($|a_1\rangle$ with black dot on the corresponding wire) and the target qubit ($|a_2\rangle$ marked with the sign $\oplus$). It flips the state of the target qubit if the control qubit is in the state $|1\rangle$ and does nothing if the control qubit is in the state $|0\rangle$. In other words, if the control qubit is in the state $|1\rangle$ the Pauli-X gate is applied to the target qubit. Other controlled gates shown in Fig. 3 act similarly as the CNOT gate: if the control qubit is in the state $|1\rangle$ then the corresponding Z, S or T gate is applied to the target qubit, otherwise the target qubit is left alone. Another example of the two-qubit gate is the SWAP gate (see Fig. 3) which interchanges the states of two input qubits.

The set of six single-qubit gates shown in Fig. 2 together with the CNOT-gate is a universal set of gates [2]. It means that any quantum computation can be

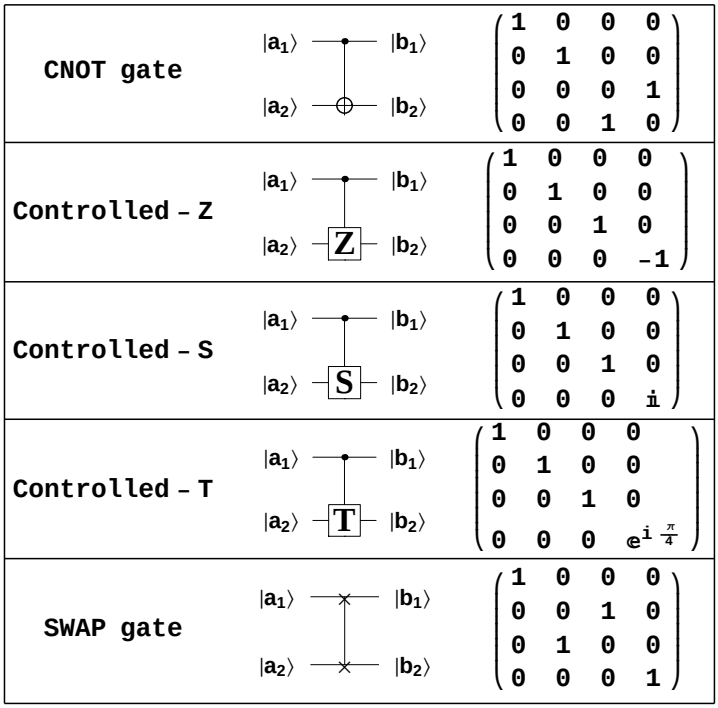

Figure 3: Controlled two-qubit gates and swap gate

decomposed in terms of this set gates. In practice, however, there may be more specialized quantum gates that would enable us to construct more compact circuits for specific computations, some of them are shown in Fig. 3. We'll use also the Toffoli gate (Fig. 4) or controlled-controlled-gate [2]. It can be considered as a generalization of CNOT-gate and has three input qubits and three output qubits: two of them are the control qubits and one is the target qubit. And it flips the state of the target qubit only if both control qubits are in the state $|1\rangle$. Note that other two-qubit gates in Fig. 3 may be generalized to the three-qubit gates in a similar way.

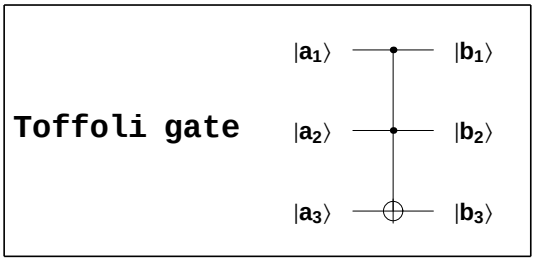

Figure 4: Toffoli gate

3 Building of quantum circuits

Before describing an algorithm for generating quantum circuits let us note that any quantum circuit may be represented as a rectangular table. The number of rows in the table is equal to the number of qubits in the circuit, while the number of columns depends on the number of quantum gates and their arrangement. Consider, for example, a quantum circuit containing three qubits and six quantum gates (see diagram in the left-hand side of Fig. 5). Drawing dashed lines, we can separate neighboring rows and columns in the table in such a way that each of its cell would contain some elementary gate (wires without any gates can be considered as the identity quantum gates, i.e., the gates making identical transformation of the corresponding qubits).

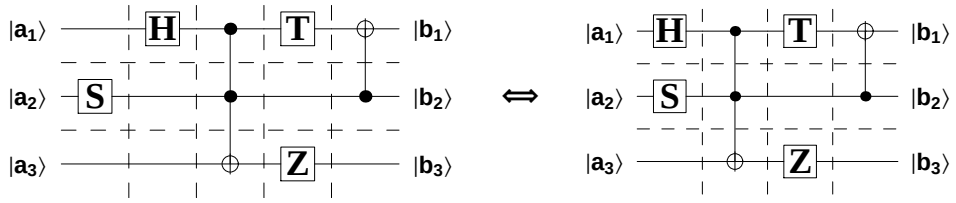

Figure 5: Table representation of the quantum circuit

Obviously, we can shift the S-gate to the second column in the table without any disturbance of the circuit. Then the first column will contain only three identity gates and it can be removed from the table. Hence, two diagrams in Fig. 5 represent the same circuit but the table in the right-hand side is smaller and simpler. In general, drawing the circuit, we'll adhere the following convention: *each column in the table can contain either one multi-qubit gate or only single-qubit gates and there are not any neighboring columns containing only single-qubit gates acting on different qubits*. Note, that according to this convention, we can not shift the Pauli-Z gate in Fig. 5 to the last column because it would turn out to be in the same column together with multi-qubit CNOT gate.

Thinking of any quantum circuit as a table of elementary quantum gates, we can define a matrix whose elements are some symbols, denoting the gates, and this matrix will contain all information on the circuit. An example of such matrix is shown in Fig. 6, where C and X correspond to the control and target qubits, respectively (the symbols corresponding to the single-qubit gates are obvious). Obviously, this matrix completely determines the structure of the quantum circuit.

In our *Mathematica* package we just use such matrix representation for quantum circuits and define the function $circuit[mat\_?MatrixQ]$ which generates the quantum circuit, corresponding to any given matrix mat. It is quite cumbersome and we do not describe it here. Of course, before evaluating this function, we should choose a set of symbols, denoting different quantum gates, and define the functions, determining the corresponding graphical objects. One possible function, generating the

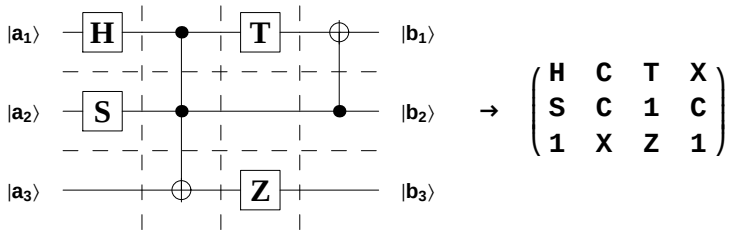

Figure 6: Matrix representation of the quantum circuit

Hadamard gate, is shown in Fig. 7 (the corresponding expression is written in *Mathematica* code). Here the integers yy, xx correspond to the row and column numbers,

```
hadamardGate[yy_, xx_] := Block[ {x, y},
        x = 0.1 (xx - 1); y = 0.1 (1 - yy);
  { Line[{{x, y}, {x + 0.025, y}}],
    Line[{{x + 0.075, y}, {x + 0.1, y}}],
    Line[{{x + 0.025, y - 0.025}, {x + 0.025, y + 0.025},
          {x + 0.075, y + 0.025}, {x + 0.075, y - 0.025},
          {x + 0.025, y - 0.025}}], Text[
     StyleForm["H", FontFamily → "Times", FontSize → 18,
            FontWeight → "Bold"], {x + 0.05, y - 0.003}]} ]
```

Figure 7: Function generating the Hadamard gate

respectively, and determine the position of the Hadamard gate in the matrix *mat*. Note that other single-qubit gates are defined in a similar way.

Then we define a function *matrixGenerating* which generates a cell, containing two *Mathematica* expressions: the matrix *mat* with given number of rows and columns all of its elements are equal to 1 and function *circuit*[*mat*] (Fig. 8).

$$\texttt{mat} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}\texttt{; circuit[mat]}$$

Figure 8: Function for generating the quantum circuits

Now, in order to generate a quantum circuit, it is sufficient to change units in the matrix *mat* by symbols, corresponding to constituent quantum gates, and evaluate the cell. If the matrix *mat* in Fig. 8 has been changed in such a way that it would coincide with the matrix given in Fig. 6, for example, then, as a result, we obtain the circuit shown in Fig. 6 as a result. It should be noted that, using the functions *matrixGenerating* and *circuit*, we can generate any quantum circuit.

4 Computing the circuit matrix

A system of n qubits has 2^n basis states of the form $|a_1 a_2 \dots a_n\rangle$, where $a_j = 0, 1$ ($j = 1, \dots, n$). They are obtained as direct product of basis states ($|0\rangle_j, |1\rangle_j$) associated with all n qubits [2]. In the case of $n = 2$ the corresponding basis states have been written in (3). Hence, the unitary matrix U defined by the quantum circuit with n qubits may be represented as a $2^n \times 2^n$ matrix with respect to these basis states.

As the circuit is read from left-to-right and we use the matrix *mat* to represent the circuit, then the matrix U can be written as the following product

$$U = U_m U_{m-1} \dots U_1 \ , \tag{5}$$

where U_j ($j = 1, 2, \dots, m$) is the $2^n \times 2^n$ matrix defined by the quantum gates being in the jth column of the matrix *mat* and m is a number of columns.

If the column contains only single-qubit gates then its matrix U_j may be constructed as a direct product of all 2×2 matrices corresponding the gates available in this column. For example, the first column of the matrix in Fig. 6 contains Hadamard (H), phase (S) and identity gates (matrices of H and S gates are shown in Fig. 2 while the matrix of the identity gate is just the 2×2 unity one). Then the matrix U_1 defined by this column is determined by the following *Mathematica* command

$$Fold[BlockMatrix[Outer[Times, \#1, \#2]]\&, \{\{1\}\}, \{matH, matS, matI\}]$$

where $matH$, $matS$, $matI$ are the matrices of the corresponding gates.

```
gateCN[n_, kn_, kc_?ListQ] :=
 Block[ {basisOld, basisNew, u0, rules},
  basisOld =
   Table[IntegerDigits[j, 2, n], {j, 0, 2^n - 1}];
  basisNew = Map[ReplacePart[#,
      Mod[Apply[Times, #[[kc]]] + #[[kn]], 2],
      kn] &, basisOld, 1];
  rules = Table[{Position[basisNew, basisOld[[j]] ][[
       1, 1]], j} → 1, {j, 2^n}];
  u0 = SparseArray[rules, {2^n, 2^n}] ;
  u0  ]
```

Figure 9: Function for computing the matrix of CNOT and Toffoli gates

Matrices defined by the multi-qubit gates can not be, in general, found as direct product of some 2×2 matrices and we need to compute them separately. For example, the function $gateCN[n\_\ ,\ kn\_\ ,\ kc\_?ListQ]$ shown in Fig. 9 computes the $2^n \times 2^n$ matrix U_j corresponding to CNOT and Toffili gates. It has three arguments: integer n, the number of qubits in the circuit; number kn and the list of numbers

```
gateCN[3, 3, {1, 2}] →
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Figure 10: Matrix corresponding to the Toffoli gate

kc determine position of the target qubit and the controlled qubits, respectively. To compute the matrix corresponding to the Toffoli gate we have to evaluate the function $gateCN[3, 3, \{1, 2\}]$ (see Fig. 10). The matrix defined by the last column of *mat* shown in Fig. 6 is computed by the command $gateCN[3, 1, \{2\}]$. Similarly, one can compute matrices corresponding to other controlled gates.

$$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & i\, e^{\frac{i\pi}{4}} & 0 & 0 & 0 & -i\, e^{\frac{i\pi}{4}} \\ 0 & 0 & -i\, e^{\frac{i\pi}{4}} & 0 & 0 & 0 & i\, e^{\frac{i\pi}{4}} & 0 \\ e^{\frac{i\pi}{4}} & 0 & 0 & 0 & -e^{\frac{i\pi}{4}} & 0 & 0 & 0 \\ 0 & -e^{\frac{i\pi}{4}} & 0 & 0 & 0 & e^{\frac{i\pi}{4}} & 0 & 0 \\ 0 & 0 & i & 0 & 0 & 0 & i & 0 \\ 0 & 0 & 0 & -i & 0 & 0 & 0 & -i \end{pmatrix}$$

Figure 11: Matrix defined by the circuit of Fig. 6

As soon as all matrices defined by the columns of the primary matrix *mat* are computed, we can evaluate the matrix U defined by the corresponding quantum circuit. This task is fulfilled by the function $matrixU[mat\_?MatrixQ]$, whose argument is just the matrix *mat* which we use to define a quantum circuit. For the circuit shown in Fig. 6 the corresponding matrix U given in Fig. 11.

In order to demonstrate that with the package developed we can easily simulate any quantum circuit, let us consider the circuit for a 3 qubit quantum Fourier transformation [2]. To generate this circuit and compute its unitary matrix U we have to evaluate the following *Mathematica* commands. First of all, we generate a skeleton 3×7 matrix with the function *matrixGenerating* (see Fig. 12).

```
mat =
```

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

```
; circuit[mat]
```

Figure 12: A skeleton 3×7 matrix

Then we replace some units in the matrix *mat* according to the algorithm of quantum

Fourier transformation [2] (see Fig. 13) and evaluate the function *circuit*[*mat*]. As a result, the corresponding quantum circuit is obtained as output.

$$\texttt{mat} = \begin{pmatrix} \texttt{H} & \texttt{S} & \texttt{T} & \texttt{1} & \texttt{1} & \texttt{1} & \texttt{SW} \\ \texttt{1} & \texttt{C} & \texttt{1} & \texttt{H} & \texttt{S} & \texttt{1} & \texttt{1} \\ \texttt{1} & \texttt{1} & \texttt{C} & \texttt{1} & \texttt{C} & \texttt{H} & \texttt{SW} \end{pmatrix} \texttt{; circuit[mat]}$$

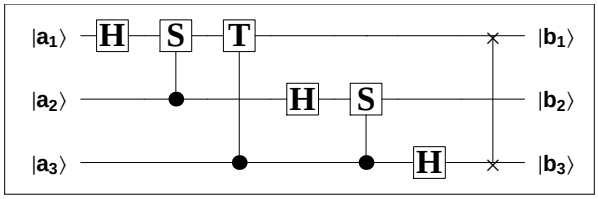

Figure 13: Circuit for 3 qubit quantum Fourier transform

At last, we evaluate the function *matrixU*[*mat*] and obtain the matrix U (see Fig. 14) defined by the circuit of Fig. 13 (cf. [2]).

$$\frac{1}{2\sqrt{2}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & e^{\frac{i\pi}{4}} & i & i\,e^{\frac{i\pi}{4}} & -1 & -e^{\frac{i\pi}{4}} & -i & -i\,e^{\frac{i\pi}{4}} \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & i\,e^{\frac{i\pi}{4}} & -i & e^{\frac{i\pi}{4}} & -1 & -i\,e^{\frac{i\pi}{4}} & i & -e^{\frac{i\pi}{4}} \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -e^{\frac{i\pi}{4}} & i & -i\,e^{\frac{i\pi}{4}} & -1 & e^{\frac{i\pi}{4}} & -i & i\,e^{\frac{i\pi}{4}} \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & -i\,e^{\frac{i\pi}{4}} & -i & -e^{\frac{i\pi}{4}} & -1 & i\,e^{\frac{i\pi}{4}} & i & e^{\frac{i\pi}{4}} \end{pmatrix}$$

Figure 14: Matrix defined by the circuit of Fig. 13

5 Polynomial equations describing circuits built from Hadamard and Toffoli gates

If a circuit contains only the Toffoli and Hadamard gates, then one can construct its circuit matrix by the alternative method [9, 10] that does not use the above considered method based on the straightforward linear algebra. Instead, the alternative method exploits algebra of multivariate polynomials associated with the circuit. With all this going on, it should be noted that the Toffoli and Hadamard gates form a universal gate set [11], and there is famous Soloway-Kitaev algorithm and implementing it software [2, 12, 13] for conversion of circuits composed from other gates.

To construct the system of multivariate polynomials one can apply the quantum-mechanical Feynman's sum-over-paths approach to a quantum circuit [9]. This

means for every path any quantum gate in the circuit under consideration acts as its classical counterpart. In so doing, the classical gate for the quantum Haramard gate outputs the path variable $x \in \mathbb{F}_2$ [9] irrespective of the input. Its value determines one of the two possible paths of computation. Thereby, the classical Hadamard gate acts as

$$a_1 \mapsto x\,, \qquad a_i, x \in \mathbb{F}_2\,,$$

whereas the classical Toffoli gate acts as

$$(a_1, a_2, a_3) \mapsto (a_1, a_2, a_3 \oplus a_1 a_2)$$

where $\oplus$ denotes addition modulo 2.

Let us now consider example [9] of the 3 qubit circuit built of the Hadamard and Toffoli. Again we generate, first, the 3×4 matrix corresponding to the example by means of the function *matrixGenerating* as shown in Fig. 15:

$$\texttt{mat} = \begin{pmatrix} \texttt{H} & \texttt{C} & \texttt{H} & \texttt{N} \\ \texttt{H} & \texttt{C} & \texttt{1} & \texttt{C} \\ \texttt{1} & \texttt{N} & \texttt{H} & \texttt{C} \end{pmatrix} \texttt{; circuit[mat]}$$

Figure 15: 3×4 matrix generating the example from [9]

It outputs the circuit of Fig.16

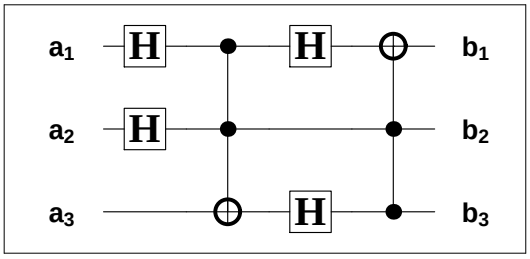

Figure 16: The circuit example from [9]

Its circuit matrix computed by the function $matrixU[mat]$ as described in of section 4 is given by

Turning back to Feynman's sum-over-paths approach, a classical path is defined by a sequence of classical bit strings $\mathbf{a}, \mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_m} = \mathbf{b}$ produced from action of the classical gates. Each set of values of the path variables x_i gives a sequence of classical bit strings which is called an admissible classical path. All path variables and, thus, all admissible classical paths for Fig.16 are explicitly shown in Fig.18. The corresponding sequence of classical bit strings is $\mathbf{a} = \{a_1, a_2, a_3\}$, $\mathbf{a_1} = \{x_1, x_2, a_3\}$, $\mathbf{a_2} = \{x_1, x_2, a_3 \oplus x_1 x_2\}$, $\mathbf{a_3} = \{x_3, x_2, x_4\}$, $\mathbf{a_4} = \{x_3 \oplus x_2 x_4, x_2, x_4\} = \mathbf{b}$.

Each admissible classical path is provided with a phase which is determined by the Hadamard gates applied [9]. The phase is changed only when the input and

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \end{pmatrix}$$

Figure 17: Matrix for circuit of Fig.16 computed by linear algebra

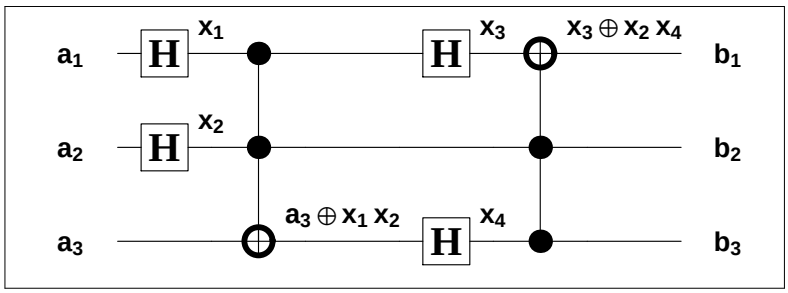

Figure 18: The admissible paths for the circuit of Fig.16

output of the Hadamard gate are simultaneously equal to 1. Thereby, this gives the formula

$$\varphi(\mathbf{x}) = \sum_{\text{Hadamard gates}} input \bullet output \tag{1}$$

with the sum evaluated in $\mathbb{F}_2$. As to Toffoli gates, they do not change the phase. In example of Fig. 16 the phase of the path $\mathbf{x}$ reads

$$\varphi(\mathbf{x}) = a_1x_1 \oplus a_2x_2 \oplus x_1x_3 \oplus a_3x_4 \oplus x_1x_2x_4\,.$$

According to the Feynman's sum-over-paths method the matrix element of a quantum circuit is the sum over all the allowed paths from the classical state $\mathbf{a}$ to $\mathbf{b}$

$$\langle \mathbf{b}|\, U_f\, |\mathbf{a}\rangle = \frac{1}{\sqrt{2^h}} \sum_{\mathbf{x}:\mathbf{b}(\mathbf{x})\mathbf{b}} (-1)^{\varphi(\mathbf{x})}\,,$$

where h is the number of Hadamard gates. Apparently, the terms in the sum have the same absolute value but may vary in sign.

Let N_0 be the number of positive terms in the sum and N_1 be the number of negative terms:

$$N_0 = |\{\ \mathbf{x} \mid \mathbf{b}(\mathbf{x}) = \mathbf{b} \quad \textbf{and} \quad \varphi(\mathbf{x}) = 0\ \}|\,, \tag{2}$$
$$N_1 = |\{\ \mathbf{x} \mid \mathbf{b}(\mathbf{x}) = \mathbf{b} \quad \textbf{and} \quad \varphi(\mathbf{x}) = 1\ \}|\ . \tag{3}$$

Hence, N_0 and N_1 count the number of solutions for the indicated systems of $n+1$ polynomials in h variables over $\mathbb{F}_2$. Then the matrix element may be written as the difference

$$\langle \mathbf{b}|\, U_f\, |\mathbf{a}\rangle = \frac{1}{\sqrt{2^h}}\,(N_0 - N_1)\ . \tag{4}$$

In our *Mathematica* package there is function $polynomials[mat\_?MatrixQ]$ which constructs and outputs the set of polynomials over $\mathbb{F}_2$ which follows from the bit string of the form $\mathbf{b}(\mathbf{x}) = \mathbf{b}$ that relates the output bits with the path variables. Here we denoted by $\mathbf{b}(\mathbf{x})$ the last bit string $\mathbf{a_m}$ in the admissible path set which depends polynomially on the path variables $\mathbf{x} = \{x_1, \ldots, x_h\}$. Since in constructing of the circuit matrix we have to count the number of solutions for polynomial systems (2) and (3) in $\mathbb{F}_2$, and input and output bit variables a_i, b_i also take values in $\mathbb{F}_2$, function $polynomials[mat\_?MatrixQ]$ outputs the polynomials in the form $\mathbf{b}(\mathbf{x}) + \mathbf{b} = 0$ and adds the phase polynomial (1) to the system.

For the circuit of Fig.16 function $polynomials[mat\_?MatrixQ]$ outputs

$$x_3 \oplus x_2\ \ x_4 \oplus b_1$$
$$x_2 \oplus b_2$$
$$x_4 \oplus b_3$$
$$a_1\ x_1 \oplus a_2\ x_2 \oplus x_1\ x_3 \oplus a_3\ x_4 \oplus x_1\ x_2\ x_4$$

Figure 19: Polynomial system for the circuit of Fig.16

The upper tree polynomial in Fig.19 are those generated by the output bit string relating the input and output qubit values for admissible paths coded in terms of the variables $\{x_1, x_2, x_3, x_4\}$. The bottom polynomial is the phase polynomial defined by formula (1).

6 Solving circuit polynomial system

To count the number of solutions in $\mathbb{F}_2$ for the polynomial systems (2) and (3) in order to apply formula (4) we rewrite them into the form

$$F_0 = \{\mathbf{b}(\mathbf{x}) + \mathbf{b}, \phi(\mathbf{x})\}\,, \tag{5}$$
$$F_1 = \{\mathbf{b}(\mathbf{x}) + \mathbf{b}, \phi(\mathbf{x}) + 1\}\,. \tag{6}$$

Here F_0 denotes the output of the function $polynomials[mat\_?MatrixQ]$ in our *Mathematica* package. it is convenient to transform the system into the canonical

Gröbner basis form [14]. The Gröbner basis method invented in [15] is the most universal algorithmic tool for investigation and solving multivariate polynomial systems.

To compute N_0 and N_1 one can convert F_0 and F_1 into an appropriate triangular form [16] providing elimination of the path variables $x_1, ..., x_h$. One of such triangular forms is the pure lexicographical Gröbner basis that can be computed by means of *Mathematica* which has a built-in module for computing polynomial Gröbner bases.

For the system of polynomials F_0 in (5) shown in Fig.19 the lexicographical Gröbner basis for the ordering on the variables $x_1 \succ x_2 \succ x_3 \succ x_4$ is given by

$$G_0 : \begin{cases} g_1 = a_1x_1 \oplus b_1x_1 \oplus a_2b_2 \oplus a_3b_3 \,, \\ g_2 = x_2 \oplus b_2 \,, \\ g_3 = x_3 \oplus b_1 \oplus b_2b_3 \,, \\ g_3 = x_4 \oplus b_3 \,, \end{cases} \tag{7}$$

The Gröbner basis (7) can easily be obtained with *Mathematica*. To do this it is sufficient to define the polynomial set (5) as *Mathematica* polynomial list by the command

```
F0 = {x3 + x2 * x4 + b1, x2 + b2, x4 + b3,
  a1 * x1 + a2 * x2 + x1 * x3 + a3 * x4 + x1 * x2 * x4}
```

Figure 20: Input *Mathematica* form for F_0 in (5)

and invoke the *Mathematica* function *GroebnerBasis* with the arguments specified as follows

```
GB0 = GroebnerBasis[F0, {x1, x2, x3, x4},
  MonomialOrder → Lexicographic, Modulus → 2]
```

Figure 21: *Mathematica* command for computation of (7)

The last option in Fig.21 says to the function that coefficient field is $\mathbb{F}_2$. As a result *Mathematica* will output the Gröbner basis (7)

```
{b3 + x4, b1 + b2 b3 + x3, b2 + x2, a2 b2 + a3 b3 + a1 x1 + b1 x1}
```

Figure 22: The *Mathematica* output for the command of Fig.21

Similarly, for the system F_1 in (6) the Gröbner basis is

$$G_1 : \begin{cases} g_1 = a_1x_1 \oplus b_1x_1 \oplus a_2b_2 \oplus a_3b_3 \oplus 1 \,, \\ g_2 = x_2 \oplus b_2 \,, \\ g_3 = \ x_3 \oplus b_1 \oplus b_2b_3 \,, \\ g_3 = x_4 \oplus b_3 \,. \end{cases} \tag{8}$$

The lexicographical Gröbner bases (7) and (8) immediately yield the following conditions on the parameters:

$$G_0: \quad a_1 \oplus b_1 = a_2 b_2 \oplus a_3 b_3 = 0\,, \tag{9}$$

$$G_1: \quad a_1 \oplus b_1 = 0\,,\ a_2 b_2 \oplus a_3 b_3 = 1\,. \tag{10}$$

It is immediately follows that if conditions (9) are satisfied then the polynomial system G_0 (resp. F_0) has two common roots in $\mathbb{F}_2$ and G_1 (resp. F_1) has no common roots, and, vise-versa, if conditions (10) are satisfied then G_0 has no roots and G_1 has two roots. In all other cases there is one root of G_0 and one root of G_1.

In that way, the 8×8 matrix for the circuit of 16 is easily determined by the formulae (4) where the numbers N_0 and N_1 are defined from systems (7) and (8). As a result, the matrix of Fig.17 is obtained.

A $n-$qubit circuit with $h-$Hadamard gates the polynomial systems (5) and (6) contains $n+1$ polynomials in $h-$variables $\mathbf{x} = \{x_1, x_2, \ldots, x_h\}$ and $2n-$parameters $\mathbf{a} = \{a_1, a_2, \ldots, a_n\}$, $\mathbf{b} = \{b_1, b_2, \ldots, b_n\}$. These parameters determine the values of the input and output qubits, respectively. To apply formula (4) for computing the circuit matrix by the Gröbner bases method one needs to take into account that both variables and parameters are elements in the finite field $\mathbb{F}_2$. By this reason, generally, to increase efficiency of computation with the use of *Mathematica* function *GroebnerBasis* (Fig.21) one should add to every of the systems (5) and (6) the binomials of the form

$$x_i^2 + x_1 \quad (i = 1, \ldots, h)\,. \tag{11}$$

and also take into account the restrictions

$$a_j^2 + a_j = 0, \quad b_j^2 + b_j = 0 \quad (j = 1, \ldots, n)\,.$$

Due to the last restrictions all the intermediate polynomials arising at the Gröbner basis construction by Buchberger's algorithm [14, 15] admit substantial simplification.

It turns out that if one uses another algorithmic approach to constructiion of Gröbner bases called involutive (see [17] and references therein), then this makes possible to avoid handling extra polynomials (11). In doing so, one can work with variables directly as with elements in $\mathbb{F}_2$. The first implementation in C++ of an involutive algorithm for computation of Gröbner basis over $\mathbb{F}_2$ with polynomial variables from $\mathbb{F}_2$ described in [18]. After proper optimization it is planned to incorporate that C++ code into the open source software GINV [19] which is a C++ module of Python and oriented to computation of Gröbner bases for polynomial ideals and modules by involutive methods.

It should be noted that solving systems of multivariate polynomial equations variables over $\mathbb{F}_2$ whose variables take values in $\mathbb{F}_2$ is also of interest in cryptoanalysis. One of the attacks of a HFE (Hidden Fields Equations) public key cryptosystem is based on construction of a Gröbner basis for multivariate polynomial system over

finite fields [20]. In particular, quadratic n polynomials in n with $n \geq 80$, variables over field $\mathbb{F}_2$ was recommended as a public key, and $n = 80$ was suggested as the first challenge. In paper [21] his challenge was broken by the Gröbner basis computation by means of the C program implementing author's algorithm [22]. This remarkable computational result gives a hope that construction of the circuit matrices by means of polynomial systems (5) and (6) may be computationally superior to the linear algebra based method (sect.4) for circuits with $n \gg h$ where n and h are, as above, the numbers of qubits and Hadamard gates.

7 Acknowledgements

The contribution of one of the authors (V.P.G.) was partially supported by grant 07-01-00660 from the Russian Foundation for Basic Research and by grant 5362.2006.2 from the Ministry of Education and Science of the Russian Federation.

References

[1] *Gerdt V.P., Kragler R. and Prokopenya A.N.* On Simulation of Quantum Circuits with *Mathematica*. In: "Computer Algebra Systems in Teaching and Research / CASCR 2007", University of Podlasie, Scieldce, Poland, 2007, pp.135-144.

[2] *Nielsen M. and Chuang I.* Quantum Computation and Quantum Information. Cambridge University Press (2000)

[3] *Williams C.P. and Clearwater S.H.* Exploations in Quantum Computing. Springer-Verlag, New York, Berlin, Heidelberg (1997)

[4] *Shor P.W.* Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comp. **26**(5) (1997) 1484–1509

[5] *Grover L.K.* Quantum mechanics helps in searching for a needle in a haystack. Phys. Rev. Lett. **79** (1997) 325–328

[6] *De Raedt H. and Michielsen K.* Computational Methods for Simulating Quantum Computers. In: Handbook of Theoretical and Computational Nanotechnology, Vol.3, M.Rieth and W.Schommers (Eds.), Forschungszentrum Karlsruhe (2006). arXiv:quant-ph/0406210

[7] *Julia-Diaz B., Burdis J.M. and Tabakin F.* QDENSITY – A Mathematica quantum computer simulation. Elsevier Science (2005) http://www.pitt.edu/∼tabakin/QDENSITY

[8] *Wolfram S.* The Mathematica book. Wolfram Media/Cambridge University Press (1999)

[9] *Dawson C.M., Nielsen M.A. et al.* Quantum computing and polynomial equations over the finite field Z_2. arXiv:quant-ph/0408129.

[10] *Gerdt V.P. and Severyanov V.M.* An Algorithm for Constructing Polynomial Systems Whose Solution Space Characterizes Quantum Circuits. In: "Quantum Informatics 2005", Yu.I.Ozhigov (Ed.), SPIE Proceedings, Vol. 6264, 2006

[11] *Aharonov D.* A Simple Proof that Toffoli and Hadamard Gates are Quantum Universal. arXiv:quant-ph/0301040

[12] *Dawson C.M. and Nielsen M.A.* The Solovay-Kitaev Algorithm. arXiv: quant-ph/0505030

[13] *Nagy A.B.* On an implementation of the Solovay-Kitaev algorithm. arXiv:quant-ph/0606077

[14] *Buchberger B. and Winkler F.* (eds.) Gröbner Bases and Applications. Cambridge University Press, 1998

[15] *Buchberger B.* An Algorithm for Finding a Basis for the Residue Class Ring of a Zero-Dimensional Polynomial Ideal. PhD Thesis, University of Innsbruck, 1965 (in German)

[16] *Wang D.* Elimination Methods. Springer-Verlag, Berlin, 1999.

[17] *Gerdt V.P.* Involutive Algorithms for Computing Gröbner Bases. In: "Computational commutative and non-commutative algebraic geometry", IOS Press, Amsterdam, 2005, pp. 199–225. arXiv:math.AC/0501111

[18] *Gerdt V.P. and Zinin M.V.* On computation of Gröbner bases over $\mathbb{F}_2$. Submitted to the International Workshop on Computer Algebra and Differential Equations (Turku, Finnland, February 20-23, 2007).

[19] `http://invo.jinr.ru`

[20] *Patarin J.* Hidden fields equations (HFE) and isomorphisms of polynomials (IP): Two new families of asymmetric algorithms. In: EUROCRYPT'96, volume 1070 of LNCS 1070, Springer-Verlag, 1996, pp.33-48, 1996

[21] *Faugère J.C. and Joux A.* Algebraic cryptanalysis of Hidden Field Equations (HFE) Using Gröbner Bases. LNCS 2729, Springer-Verlag, 2003, pp. 44–60

[22] *Faugère J.C.* A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In: Proceedings of Issac'2002, ACM Press, New York (2002), pp.75–83