Письма в ЭЧАЯ. 2007. Т. 4, № 2(138). С. 225–230

A C# PACKAGE FOR ASSEMBLING QUANTUM CIRCUITS AND GENERATING ASSOCIATED POLYNOMIAL SETS

V. P. Gerdt[1], *V. M. Severyanov*[2]

Joint Institute for Nuclear Research, Dubna

Recently it has been shown that elements of the unitary matrix determined by a quantum circuit can be computed by counting the number of common roots in the finite field $\mathbb{Z}_2$ for a certain set of multivariate polynomials over $\mathbb{Z}_2$. In this paper we present a C# package that allows a user to assemble a quantum circuit and to generate the multivariate polynomial set associated with the circuit. The generated polynomial system can further be converted to the canonical triangular involutive basis form which is appropriate for counting the number of common roots of the polynomials.

Недавно было показано, что элементы унитарной матрицы, определяемой квантовой схемой, могут быть вычислены путем подсчета числа общих корней в конечном поле $\mathbb{Z}_2$ для некоторой системы полиномов от многих переменных над $\mathbb{Z}_2$. В данной статье мы представляем программный пакет, написанный на языке C#, который позволяет пользователю компоновать квантовую схему и строить систему полиномов от многих переменных, ассоциированную с этой схемой. В дальнейшем порожденная система полиномов может быть преобразована к каноническому треугольному виду с помощью инволютивных базисов, удобному для подсчета числа общих корней полиномов.

INTRODUCTION

In [1] it was shown that elements of the unitary matrix determined by a quantum circuit can be computed by counting the number of common roots in the finite field $\mathbb{Z}_2$ for a certain set of multivariate polynomials over $\mathbb{Z}_2$. Given a quantum circuit, the polynomial set is uniquely constructed. In this talk we present a C# package called QuPol (Quantum Polynomials) that allows a user to assemble a quantum circuit and to generate the multivariate polynomial set associated with the circuit.

The generated polynomial system can further be converted into the canonical Gröbner basis form for the lexicographical monomial order. Gröbner bases form the most universal algorithmic tool of modern computer algebra to investigate and to solve the systems of polynomial equations [2]. Construction of the lexicographical Gröbner basis substantially alleviates the problem of the root finding for polynomial systems. To construct such a Gröbner basis one can use efficient involutive algorithms developed in [3]. Our QuPol

[1]E-mail: gerdt@jinr.ru
[2]E-mail: severyan@jinr.ru

package, together with a Gröbner basis software, provides a tool for simulation of quantum circuits. We illustrate this tool by the example taken from [1].

Our program has a user-friendly graphical interface and a built-in base of the elementary gates, i.e. quantum gates and wires. The user can easily assemble a quantum circuit from those elements.

1. QUANTUM CIRCUITS

To compute a reversable Boolean vector-function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2^n$, one applies the appropriate unitary transformation U_f to an input state $|\mathbf{a}\rangle$ composed of some number of qubits:

$$|\mathbf{b}\rangle = U_f \, |\mathbf{a}\rangle \,, \quad |\mathbf{a}\rangle \,, |\mathbf{b}\rangle \in \mathbb{C}^{2 \otimes n}. \tag{1}$$

The output state $|\mathbf{b}\rangle$ is not the searchable result of the computation until somebody observes it. After that, the output state becomes classical and can be used anywhere.

Some elementary unitary transformations are called quantum gates. A quantum gate acts only on a few qubits, on the rest it acts as the identity. Someone assembles a quantum circuit appropriately aligning quantum gates. The unitary transformation of the circuit is the composition of its elementary unitary transformations:

$$U_f = U_m U_{m-1} \cdots U_2 U_1. \tag{2}$$

Quantum gate basis is a set of universal quantum gates, i.e. any unitary transformation can be presented as a composition of the gates of the basis. As in the classical case, there are several sets of universal quantum gates. For our work, it is convenient to choose the universal gate basis consisting of Hadamard and Toffoli gates.

Hadamard gate turns a computational state into an equally weighted superposition:

$$\begin{aligned} H : |0\rangle &\mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \\ H : |1\rangle &\mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned} \tag{3}$$

The resulting superpositions for $|0\rangle$ and $|1\rangle$ differ in a phase factor.

Toffoli gate is a three-qubit gate. Inputs x and y control the behavior of bit z: only when x and y simultaneously equal to one bit z is inverted, else, Toffoli gate is a three-fold tensor product of the identity one-bit gates:

$$(x, y, z) \mapsto (x, y, z \oplus xy). \tag{4}$$

Action of a quantum circuit can be described by a square matrix with the matrix element $\langle \mathbf{b}| \, U_f \, |\mathbf{a}\rangle$ which is a probability amplitude for transition from an initial quantum state $|\mathbf{a}\rangle$ to the final quantum state $|\mathbf{b}\rangle$. The matrix element is decomposed in accordance with the gate decomposition of the circuit unitary transformation (2) and is calculated with account of all the intermediate states $\mathbf{a}_i$, $i = 1, 2, \ldots, m-1$:

$$\langle \mathbf{b}| \, U_f \, |\mathbf{a}\rangle = \sum_{\mathbf{a}_i} \langle \mathbf{b}| \, U_m \, |\mathbf{a}_{m-1}\rangle \cdots \langle \mathbf{a}_1| \, U_1 \, |\mathbf{a}\rangle \,. \tag{5}$$

2. SUM-OVER-PATHS AND QUANTUM CIRCUITS

To apply the famous Feynman's sum-over-paths approach to calculate the matrix element of a quantum circuit, we replace every quantum gate of the circuit under consideration by its classical counterpart. The main problem here is to select the corresponding classical gate for quantum Hadamard gate because for any input value, 0 or 1, it gives the equal probability 0 or 1. We denote the output of classical Hadamard gate by the path variable x – its value determines one of the two possible paths of computation.

In Fig. 1 is shown an example of quantum circuit (taken from [1]) and its classical correspondence. Path variables x_i comprise path variable $\mathbf{x} = (x_1, x_2, x_3, x_4)^T \in \mathbb{Z}_2^4$.

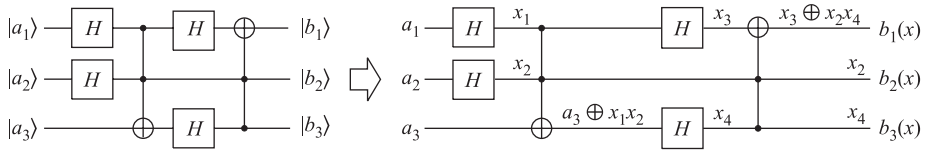

Fig. 1. From quantum to classical circuit

A classical path is a sequence of classical bit strings $a, a_1, a_2, \ldots, a_m = b$ obtained after each classical gate has been applied. For each selection of values for the path variables x_i we have a sequence of classical bit strings which is called admissible classical path. An admissible classical path has a phase which is determined by Hadamard gates. The phase changes only when the input and the output of Hadamard gate equal to 1 simultaneously and this coincides with multiplication modulo 2 of input and output values:

$$\varphi(\mathbf{x}) = \sum_{\text{Hadamardgates}} \text{input} \bullet \text{output}. \tag{6}$$

Toffoli gates do not change the phase. For our circuit the phase of the path is:

$$\varphi(\mathbf{x}) = a_1x_1 \oplus a_2x_2 \oplus x_1x_3 \oplus x_4(a_3 \oplus x_1x_2). \tag{7}$$

The matrix element of a quantum circuit is the sum of all the allowed paths from $\mathbf{a}$ to $\mathbf{b}$:

$$\langle \mathbf{b} | U_f | \mathbf{a} \rangle = \frac{1}{\sqrt{2^h}} \sum_{\mathbf{x}:\mathbf{b}(\mathbf{x})=\mathbf{b}} (-1)^{\varphi(\mathbf{x})}, \tag{8}$$

where h – the number of Hadamard gates. All terms in the sum have the same absolute value but vary in sign.

Let $N_0 = |\{x|b(x) = b \quad \& \quad \varphi(x) = 0\}|$ be the number of positive terms in the sum and $N_1 = |\{x|b(x) = b \quad \& \quad \varphi(x) = 1\}|$ the number of negative terms. This equations count solutions to a system of $n + 1$ polynomials in h variables over $\mathbb{Z}_2$. Then the matrix element may be written in this convenient form:

$$\langle \mathbf{b} | U_f | \mathbf{a} \rangle = \frac{1}{\sqrt{2^h}} (N_0 - N_1). \tag{9}$$

3. ASSEMBLING CIRCUIT

For assembling arbitrary quantum circuits composed from Hadamard and Toffoli gates, we suggest to use the set of elementary gates shown in Fig. 2 and to represent a circuit as a rectangular table (Fig. 3, *a*), each cell of which contains an elementary gate, so that the output for each row is determined by the composition of the row elementary gates. To assemble a circuit, we define an empty table of the required size. In this case, the output and the phase are not fixed. Then, we place the required elementary gates in appropriate cells and construct the circuit polynomials (Fig. 3, *b*).

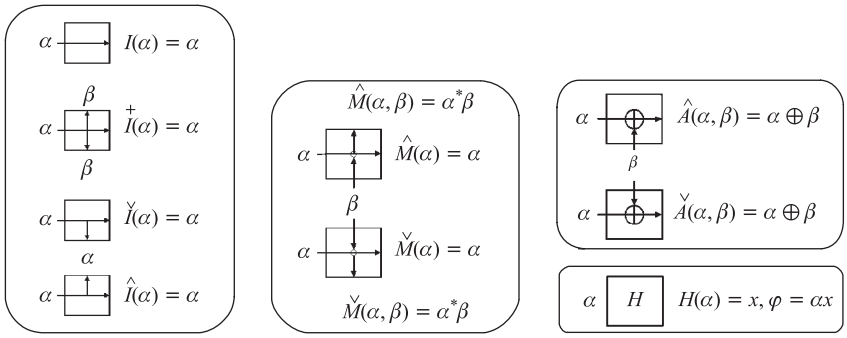

Fig. 2. Elementary gates

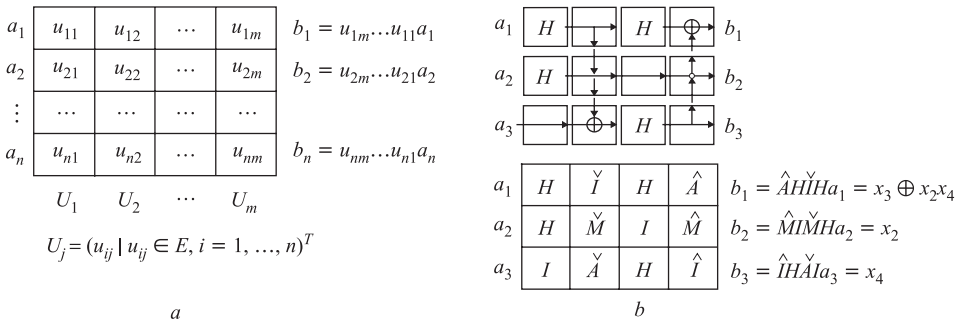

Fig. 3. Elementary decomposition (*a*) and assembling of a circuit (*b*)

A circuit is represented in the program as two 2d-arrays: one for the elementary gates and another for their polynomials. The phase polynomial is separately represented. The construction of the circuit polynomials:

```
for each Column in Table of Gates
    for each Gate in Column {
        construct Gate Polynomial;
        if Gate id Hadamard
            reconstruct Phase Polynomial; }
```

The method for constructing a gate polynomial is recursive because of the need to go up or down for some gates.

Written by us the C# name space **Polynomial_Modulo_2** can be used independently of our program. It contains classes needed for handling polynomials over the finite field $\mathbb{Z}_2$. Class Polynomial is a list of monomials, class Monomial is a list of letters, class Letter is a letter with its index and power.

4. HANDLING QUANTUM POLYNOMIALS

A system generated by the program is a finite set $F \subset R$ of polynomials in the ring

$$R := \mathbb{Z}_2[a_i, b_j][x_1, \ldots, x_h], \quad a_i, b_j \in \mathbb{Z}_2, \ \ i, j = 1, \ldots, n \tag{10}$$

in h variables and $2n$ binary coefficients. One has to count the number of roots N_0 and N_1 in $\mathbb{Z}_2$ of the polynomial sets

$$F_0 = \{f, \ldots, f_k, \varphi\}, \quad F_1 = \{f, \ldots, f_k, \varphi + 1\}. \tag{11}$$

Then, the circuit matrix is as in Eq. (9). To count the number of roots we convert F_0 and F_1 into the triangular form by computing the lexicographical Gröbner basis by means of the Buchberger algorithm or by involutive algorithm published in [3]. For the exapmle circuit in Fig. 1 we have the following polynomial system:

$$\begin{aligned} f_1 &= x_2x_4 + x_3 + b_1, \\ f_2 &= x_2 + b_2, \\ f_3 &= x_4 + b_3, \\ \varphi &= x_1x_2 + x_1x_3 + a_1x_1 + a_2x_2 + a_3x_4. \end{aligned} \tag{12}$$

The lexicographical Gröbner basis with the ordering $x_1 \succ x_2 \succ x_3 \succ x_4$ of variables and for F_0 and F_1 is as follows:

$$\begin{aligned} g_1 &= (a_1 + b_1)x_1 + a_2b_2 + a_3b_3\ (+1), \\ g_2 &= x_2 + b_2, \\ g_3 &= x_3 + b_1 + b_2b_3, \\ g_3 &= x_4 + b_3. \end{aligned} \tag{13}$$

From the lexicographical Gröbner bases with this ordering of variables we have the following conditions on the parameters:

$$\begin{aligned} a_1 + b_1 = 0 \ \ \& \ \ a_2b_2 + a_3b_3 = 0, \\ a_1 + b_1 = 0 \ \ \& \ \ a_2b_2 + a_3b_3 = 1. \end{aligned} \tag{14}$$

Under these conditions we have 2 (0) roots of F_0 (F_1) and 0 (2) roots of F_0 (F_1), in all other cases, we have 1 root of F_0 and F_1. Some matrix elements:

$$\langle 000|\, U\, |000\rangle = \frac{1}{2}, \quad \langle 001|\, U\, |011\rangle = -\frac{1}{2}, \quad \langle 000|\, U\, |111\rangle = 0. \tag{15}$$

CONCLUSIONS

The first version of a program tool for assembling arbitrary quantum circuits and for constructing quantum polynomial systems has been designed. There is the algorithmic Gröbner basis approach to convert the system of quantum polynomials into a triangular form which is useful for computing the number of solutions. The number of solutions uniquely determines the circuit matrix. Thus, the above software and algorithmic methods provide a tool for simulating quantum circuits.

REFERENCES

1. *Dawson C. M. et al.* Quantum Computing and Polynomial Equations over the Finite Field Z_2. quant-ph/0408129. 2004.

2. Gröbner Bases and Applications / Eds. Buchberger B., Winkler F. Cambridge Univ. Press, 1998.

3. *Gerdt V. P.* Involutive Algorithms for Computing Gröbner Bases // Proc. of the NATO Advanced Research Workshop «Computational Commutative and Non-Commutative Algebraic Geometry», Chishinau, June 6–11, 2004. IOS Press, 2005.

4. Microsoft Visual C# .net Standard. Version 2003.