4

Evolutionary Logic Synthesis of Quantum Finite State Machines for Sequence Detection

Martin Lukac[1] and Marek Perkowski[2]
[1]*Intelligent Integrated Systems Laboratory GSIS, Tohoku University,*
[2]*Department of Electrical and Computer Engineering, Portland State University*
[1]*Japan*
[2]*USA*

1. Introduction

Quantum Finite State Machines (QFSM) are a well known model of computation that was originally formalized by Watrous [Wat95a, Wat95b, Wat97], Kondacs [KW97] and more generally Quantum Turing Machines (QTM) have been described by Bernstein [BV97]. In particular the 2-way QFSM have been shown to be more powerful than classical FSM [KW97]. Thus the interest in quantum computational models of automata and machines is not only theoretical but has also possible applications realization of future quantum computer and robotics controllers.

In this chapter we present the evolutionary approach to the synthesis of QFSM's specified by a quantum circuits. This approach was originally proposed by [LP09] and is possible on yet only theoretical basis. In particular this approach requires a selective qubit-initialization in a quantum register. In contrast the current methodology and approaches to practical Quantum Computation, the current practical realization of quantum computation always starts with the initialization of the whole quantum register and terminates by the measurement of either all of the qubits or by the measurement of a given subset of qubits. Moreover in general there is no reuse of any element of the quantum register.

In this text we analyze in details what type of QFSM can be successfully synthesized.

The evolutionary approach will evaluate the results based on both the correctness and the cost of the evolved machines. Multiple parameters such as type of error evaluation, synthesis constraints and evolutionary operators will be discussed when evaluating to the obtained results.

In particular we show how to synthesize QFSMs as sequence detectors and illustrate their functionality both in the quantum world and in the classical (observable) world. The application of the synthesized quantum devices is illustrated by the analysis of recognized sequences.

Finally, we provide analytic method for the used evolutionary approach and we describe the experimental protocol, and its heuristic improvements. We also discuss the results. In addition, we investigate the following aspects of the Evolutionary Quantum Logic Synthesis:

- Quantum probabilistic FSM and Reversible FSM.
- Hardware acceleration for the Fitness evaluation using CBLAS [cbl] and using CUBLAS [cud] (CUDA[cud] implemented Basic Linear Algebra Subprograms (BLAS)[cbl] subroutines).

Source: New Achievements in Evolutionary Computation, Book edited by: Peter Korosec,
ISBN 978-953-307-053-7, pp. 318, February 2010, INTECH, Croatia, downloaded from SCIYO.COM

2. Background in quantum computing

In Quantum Computing the information is represented by a Quantum Bit also called qubit. The wave equation is used to represent a qubit or a set of them. Equation 1 shows a general form in the Dirac notation.

$$\begin{aligned} |\phi\rangle &= e^{i\rho} cos\theta + e^{i(\rho+\psi)} sin\theta \\ &= e^{i\rho}(cos\theta + e^{i\psi} sin\theta) \end{aligned} \tag{1}$$

In Dirac notation $|\cdot\rangle$ represents a column vector, also called a *ket*. The *bra* element denoted $\langle\cdot|$ stands for hermitian conjugate. In this manner a bra-ket $\langle\cdot|\cdot\rangle$ represents the inner, dot-vector product while $|\cdot\rangle\langle\cdot|$ represents the outer vector product. The general equation (1), $e^{i\rho}cos\theta|0\rangle + e^{i(\rho+\psi)}sin\theta|1\rangle$ can be written as $\alpha|0\rangle+\beta|1\rangle$ with $|\alpha|^2+|\beta|^2 = 1$ and $|\alpha|^2$is the probability of observing the state $|0\rangle$ while $|\beta|^2$ is the probability of observing $|1\rangle$.

In general, to describe basis states of a Quantum System, the Dirac notation is preferred to the vector-based Heisenberg notation. However, Heisenberg notation can be more practical to represent the exponential growth of the quantum register. Let two orthonormal quantum states be represented in the vector (Heisenberg) notation eq. 2.

$$\begin{aligned} |\uparrow\rangle = |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \\ |\downarrow\rangle = |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned} \tag{2}$$

Different states in this vector notation are then multiplications of all possible states of the system, and for a two-qubit system we obtain (using the Kronecker product[Gru99, Gra81, NC00]) the states represented in eq. 3:

$$\begin{aligned} |00\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad |10\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ |01\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad |11\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \end{aligned} \tag{3}$$

The Kronecker product exponentially increases the dimension of the space for matrices as well:

$$I \otimes X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4}$$

This tensor product operation for a parallel connection of to wires is shown in Figure 1.

Assume that qubit a (with possible states $|0\rangle$ and $|1\rangle$) is represented by $|\Psi_a\rangle = \alpha_a|0\rangle + \beta_a|1\rangle$ and qubit b is represented by $|\Psi_b\rangle = \alpha_b|0\rangle + \beta_b|1\rangle$. Each of them is represented by the

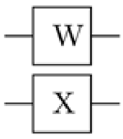


Fig. 1. Circuit representing the $W \otimes X$ operation

superposition of their basis states, but put together the characteristic wave function of their combined states will be:

$$\begin{aligned}|\Psi_a\Psi_b\rangle &= \alpha_a\alpha_b|00\rangle + \alpha_a\beta_b|01\rangle \\ &+ \beta_a\alpha_b|10\rangle + \beta_a\beta_b|11\rangle\end{aligned} \tag{5}$$

with α_a and β_b being the complex amplitudes of states of each EP respectively. As shown before, the calculations of the composed state used the Kronecker multiplication operator. Hence comes the possibility to create quantum memories with extremely large capacities and the requirement for efficient methods to calculate such large matrices.

Quantum Computation uses a set of Quantum properties. These are the measurement, the superposition and the entanglement. First, however, the principles of multi-qubit system must be introduced.

2.1 Multi-Qubit System

To illustrate the superposition let's have a look at a more complicated system with two quantum particles a and b represented by $|\psi_a\rangle = \alpha_0|0\rangle + \beta_a|1\rangle$ and $|\psi_b\rangle = \alpha_b|0\rangle + \beta_b|1\rangle$ respectively. For such a system the problem space increases exponentially and is represented using the Kronecker product [Gru99].

$$|\psi_a\rangle \otimes |\psi_b\rangle = \begin{bmatrix}\alpha_0\\ \beta_0\end{bmatrix} \otimes \begin{bmatrix}\alpha_1\\ \beta_1\end{bmatrix} = \begin{bmatrix}\alpha_0\alpha_1\\ \alpha_0\beta_1\\ \beta_0\alpha_1\\ \beta_0\beta_1\end{bmatrix} \tag{6}$$

Thus the resulting system is represented by $|\psi_a\psi_b\rangle = \alpha_a\alpha_b|00\rangle + \alpha_a\beta_b|01\rangle + \beta_a\alpha_b|10\rangle + \beta_a\beta_b|11\rangle$ (5) where the double coefficients obey the unity (completeness) rule and each of their powers represents the probability to measure the corresponding state. The superposition means that the quantum system is or can be in any or all the states at the same time. This superposition gives the massive parallel computational power to quantum computing.

2.2 Entanglement and projective measurements

Assume the above two-particle vector (two-qubit quantum system) is transformed using the quantum circuit from Figure 2.

This circuit executes first a Hadamard transform on the top qubit and then a Controlled-Not operation with the bottom qubit as the target. Depending on the initial state of the quantum register the output will be either $|\psi_a\psi_b\rangle = \alpha_a\alpha_b|00\rangle \pm |\beta_a\beta_b|11\rangle$ or $|\psi_a\psi_b\rangle = \alpha_a\beta_b|01\rangle \pm |\beta_a\alpha_b|10\rangle$. Thus it is not possible to estimate with 100% probability the initial state of the quantum register.

Let $|ab\rangle = |00\rangle$ at level a (Figure 2). The first step is to apply the [H] gate on the qubit-a and the resulting state at level b of the circuit is

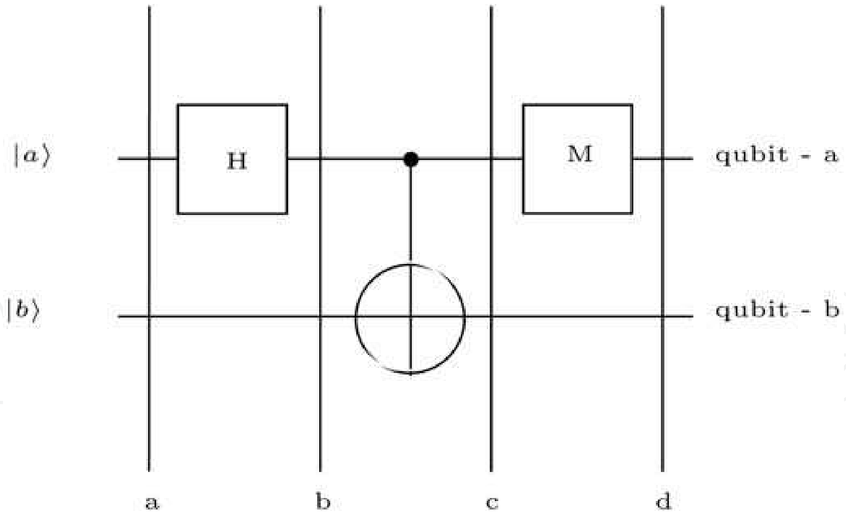


Fig. 2. EPR producing circuit

$$\begin{aligned} |ab\rangle &\rightarrow (H \otimes W)|ab\rangle \\ &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)|0\rangle \\ &= \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{bmatrix} \end{aligned} \tag{7}$$

Next the application of the CNOT gate results in:

$$|\psi_a\psi_b\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{8}$$

For an output 0 (on the qubit-a), the projective measurement of the first (topmost) qubit (qubit-a on Figure 2) on this stage would collapse the global state (with a single measurement) to the state $|00\rangle$:

$$|ab\rangle \rightarrow \frac{M_0|ab\rangle}{\sqrt{\langle ab|M_0^\dagger M_0|ab\rangle}} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^T = |00\rangle \tag{9}$$

with

$$M_0|ab\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{10}$$

and

$$\begin{aligned}\sqrt{\langle ab|M_0^{\dagger}M_0|ab\rangle} &= \sqrt{\frac{1}{2}}\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\begin{bmatrix}1\\0\\0\\1\end{bmatrix}\\ &= \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\\0\end{bmatrix}\\ &= \frac{1}{\sqrt{2}}\end{aligned} \tag{11}$$

Similarly, the probability of measuring output on the qubit-a in state $|0\rangle$ is:

$$\begin{aligned}p(0) &= \begin{bmatrix}\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}}\end{bmatrix}\left(\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}+\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\right)\begin{bmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{bmatrix}\\ &= \begin{bmatrix}\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{bmatrix}\\ &= \frac{1}{2}\end{aligned} \tag{12}$$

If one would look to the output of the measurement on the second qubit (qubit-b), the probability for obtaining $|0\rangle$ or $|1\rangle$ is in this case the following:

$$\begin{aligned}p(0) &= \begin{bmatrix}\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}}\end{bmatrix}\left(\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}+\begin{bmatrix}0&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}\right)\begin{bmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{bmatrix}\\ &= \begin{bmatrix}\frac{1}{\sqrt{2}} & 0 & 0 & \frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&0&0&0\\0&0&1&0\\0&0&0&0\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}}\\0\\0\\\frac{1}{\sqrt{2}}\end{bmatrix}\\ &= p(1) = \frac{1}{2}\end{aligned} \tag{13}$$

Thus the expectation values for measuring both values 0 or 1 on each qubit independently are $\frac{1}{2}$.

If however one looks on the second and non-measured qubit (if the qubit-a is measured, it is the qubit-b, and vice versa) and calculates the output probabilities, the output is contradictory to the expectations given by standard probabilistic distribution such as a coin toss $q = 1 - p$. To see this let's start in the state

$$\begin{bmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{14}$$

and measure the qubit-a and obtain a result. In this case assume the result of the measurement is given by:

$$|\Psi\rangle \rightarrow \frac{M_0|\Psi\rangle}{\sqrt{\langle\Psi|M_0^\dagger M_0|\Psi\rangle}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{15}$$

Then measuring the second qubit (qubit-b) will not affect the system because the measurement of the qubit-a has collapsed the whole system into a single basis state:

$$|\Psi\rangle \xrightarrow{M} |00\rangle \tag{16}$$

The probability for obtaining a $|1\rangle$ on the qubit-b is thus 0 and the measurement on qubit-b (after having measured qubit-a) has no effect on the system at all. The states of qubits are thus correlated. This non-locality paradox was first described by Einstein-Podolsky-Rosen work[EPR35] and is known as the EPR paradox. This particular phenomenon is one of the most powerful in quantum mechanics and quantum computing, as it allows together with superposition the speedup of finding solutions to certain types of problems. Finally, it can be noted that mathematically, the entangled state is such that it cannot be factored into simpler terms. For example, the state $\frac{(|00\rangle+|01\rangle)}{\sqrt{2}} \rightarrow \frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)|0\rangle$ and thus it can be factored. However, the states as those introduced in eq. 15 cannot be transformed in such a manner and are thus entangled; physically implying that they are related through measurement or observation. □

2.3 Single-Qubit quantum gates

We are now concerned with matrix representation of operators. The first class of important quantum operators are the one-qubit operators realized in the quantum circuit as the one-qubit (quantum) gates. Some of their matrix representations can be seen in equation 17.

$$\begin{array}{lll} a)\ \ X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} & b)\ \ Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} & c)\ \ Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \\ d)\ \ H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} & e)\ \ V = \frac{(1+i)}{2}\begin{bmatrix} 1 & -i \\ -i & 1 \end{bmatrix} & f)\ \ Phase = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \end{array} \tag{17}$$

Each matrix of an Operator has its inputs from the top (from left to right) and the outputs on the side (from top to bottom). Thus taking a state $|\psi\rangle$ (eq.18) and an unitary operator H (eq. 19)

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{18}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{19}$$

the result of computation is represented in equation 20.

$$H|\Psi\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{\alpha+\beta}{\sqrt{2}} \\ \frac{\alpha-\beta}{\sqrt{2}} \end{bmatrix} \tag{20}$$

$$U = \begin{array}{cc} & \begin{array}{cccc} 00 & 01 & 10 & 11 \\ \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ \begin{array}{c} 00 \leftarrow \\ 01 \leftarrow \\ 10 \leftarrow \\ 11 \leftarrow \end{array} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array} \tag{21}$$

Equation 21 shows the inputs (input minterms) on the top of the matrix and the output minterms on the left side. Thus for an input $|10\rangle$ (from the top) the output is $|11\rangle$ (from the side).

2.4 Multi-Qubit quantum gates

The second class of quantum gates includes the Controlled-U gates. Schematic representation of such gates can be seen in Figure 3. Gates in Figure 3a – Figure 3c represent the general structures for single-control-qubit single-qubit gate, two-control-qubit single-qubit gate, single-control-qubit two-qubit gate and two-control-qubit two-qubit gate respectively. The reason for calling these gates *Controlled* is the fact that they are based on two operations: first there is one or more control bits and second there is a unitary transformation similar to matrices from equation 17 that is controlled. For instance the Feynman gate is a Controlled-NOT gate and has two input qubits a and b as can be seen in

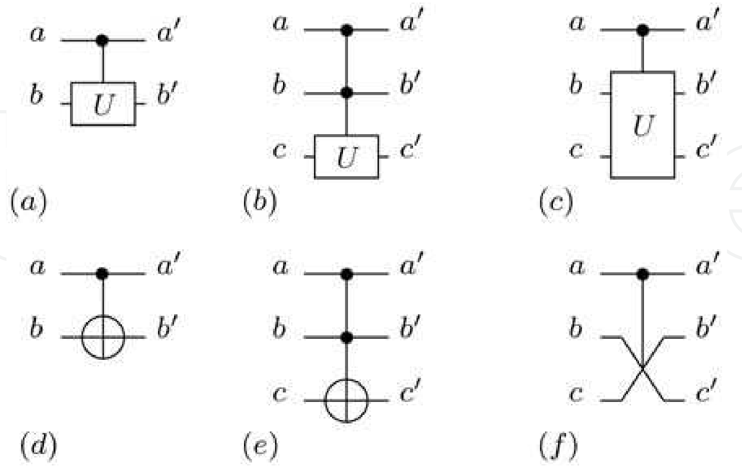


Fig. 3. Schematic representation of Controlled-U gates: a) general structure of single-qubit controlled-U gate (control qubit a, target qubit, b) two-qubit controlled, single-qubit operation, c) single-qubit controlled, two-qubit target quantum gate, d) Feynman (CNOT), e) Toffoli (CCNOT), f) Fredkin. a, b, c are input qubits and a′, b′ and c′ are respective outputs.

Figure 3. Its unitary matrix with input and output minters is shown in eq. (21). Thus qubits controlling the gate are called the control qubits and the qubits on which the unitary transform is applied to are called the target qubits.

Figures 3d - Figure 3f represent special cases where the controlled unitary operator is Not, Not and Swap, respectively. The respective unitary matrices are in equations 21, 22a and 22b.

Equation 21 shows that if the input state is for instance $|00\rangle$ (from the top) the output is given by $U|00\rangle = p_{00}|00\rangle = 1 \times |00\rangle$. Similarly for all other possible input /output combinations.

$$
(a)\quad \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1} \\
0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{0}
\end{bmatrix}
\qquad
(b)\quad \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{0} & \mathbf{1} & 0 \\
0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{0} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\qquad (22)
$$

The *Controlled-U* gate means that while the controlled qubit a is equal to 0 the qubits on output of both wires are the same as they were before entering the gate (a′ = a, b′ = b). Now if qubit a equals to 1, the result is a′ = a and b′ = ¬b according to matrix in equation (17.a). It can be easily verified that the CCNOT (Toffoli) gate is just a Feynman gate with one more control qubit and the Fredkin gate is a controlled swap as shown on Figure 3.

A closer look at equations (21 and 22) gives more explanation about what is described in eq. 21: CNOT, eq. 22*a* : Toffoli and eq. 22*b* : Fredkin gates. For instance, equation 21 shows that while the system is in states $|00\rangle$ and $|01\rangle$ the output of the circuit is a copy of the input. For the inputs $|10\rangle$ and $|11\rangle$ the second output is inverted and it can be seen that the right-lower corner of the matrix is the NOT gate. Similarly in the other two Controlled gates the NOT gate matrix can be found.

2.5 NMR-based quantum logic gates

The NMR (Nuclear Magnetic Resonance) technology approach to quantum computing [Moo65, PW02, DKK03] is the most advanced quantum realization technology used so far, mainly because it was used to implement the Shor algorithm [Sho94] with 7 qubits [NC00]. Yet other technologies such as Ion trap [DiV95], Josephson Junction [DiV95] or cavity QED [BZ00] are being used. The NMR quantum computing has been reviewed in details in [PW02, DKK03] and for this paper it is important that it was so far the NMR computer that allowed the most advanced algorithm (7 qubit logic operation) to be practically realized and analyzed in details. Thus it is based on this technology that the constraints of the synthesis are going to be established for the cost and function evaluation. Some prior work on synthesis has been also already published [LLK^{+}06] and few simple cost functions have been established.

For the NMR-constrained logic synthesis the conditions are:

- Single qubit operations: rotations R_x, R_y, R_z for various degrees of rotation θ. With each unitary rotation (R_x, R_y, R_z) represented in equation 23

$$\begin{aligned} R_x(\theta) &= e^{-i\theta X/2} = cos\frac{\theta}{2}I - isin\frac{\theta}{2}X = \begin{pmatrix} cos(\frac{\theta}{2}) & -isin(\frac{\theta}{2}) \\ -isin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{pmatrix} \\ R_y(\theta) &= e^{-i\theta Y/2} = cos\frac{\theta}{2}I - isin\frac{\theta}{2}Y = \begin{pmatrix} cos(\frac{\theta}{2}) & -sin(\frac{\theta}{2}) \\ sin(\frac{\theta}{2}) & cos(\frac{\theta}{2}) \end{pmatrix} \\ R_z(\theta) &= e^{-i\theta Z/2} = cos\frac{\theta}{2}I - isin\frac{\theta}{2}Z = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix} \end{aligned} \tag{23}$$

- Two-qubit operation; depending on approach the Interaction operator is used as J_{zz} or J_{xy} for various rotations θ

Thus a quantum circuit realized in NMR will be exclusively built from single qubit rotations about three axes x,y,z and from the two-neighbor-qubit operation of interaction allowing to realize such primitives as CNOT or SWAP gates. Examples of gates realized using NMR quantum primitives are shown in Figure 5 to Figure 8.

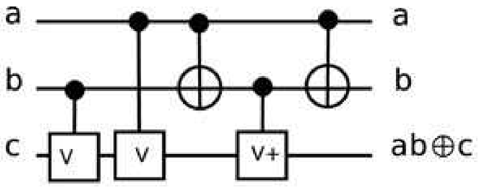


Fig. 4. Structure of the Toffoli gate

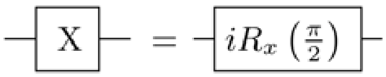


Fig. 5. Single pulse Logic gate – NOT

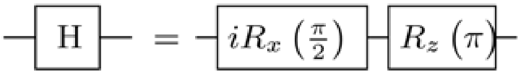


Fig. 6. Two-pulses logic gate – Hadamard

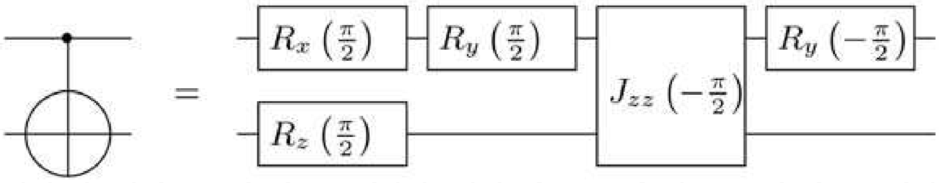

=
$R_x\left(\frac{\pi}{2}\right)$
$R_y\left(\frac{\pi}{2}\right)$
$J_{zz}\left(-\frac{\pi}{2}\right)$
$R_y\left(-\frac{\pi}{2}\right)$
$R_z\left(\frac{\pi}{2}\right)$

Fig. 7. Detailed Realization of Feynman Gate with five EM pulses.

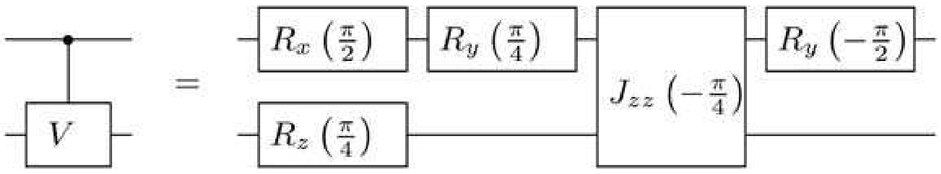

V
=
$R_x\left(\frac{\pi}{2}\right)$
$R_y\left(\frac{\pi}{4}\right)$
$J_{zz}\left(-\frac{\pi}{4}\right)$
$R_y\left(-\frac{\pi}{2}\right)$
$R_z\left(\frac{\pi}{4}\right)$

Fig. 8. Five-pulses logic gate - Controlled-V

Also, the synthesis using the NMR computing model using EM pulses, is common to other technologies such as Ion Trap [CZ95, PW02] or Josephson Junction [BZ00]. Thus the cost model used here can be applied to synthesize circuits in various technologies, all of these technologies having the possibility to express the implemented logic as a sequence of EM pulses.

3. Quantum finite state machines

The paradigms of quantum circuits from Section 2 are applied in this paper to the synthesis of computational models such as QFSM as defined in [LPK09]. This section briefly introduces the knowledge about Quantum computational models and their properties as well as specifies the types of devices that are going to be synthesized. We describe the 1-way Quantum Finite State Machines (FSM) from both the theoretical (computational) point of view as well as from the engineering (circuit) point of view. Most of the work in this area is still on the theoretical level but the proofs of concept quantum devices [Dun98, SKT04, MC06, RCHCX+08, YCS09] allow to speculate that such models will be useful for quantum logical devices that will appear in close future.

3.1 1-way quantum finite automata

Quantum Finite State Machines (QFSM) are a natural extension of classical (probabilistic) FSM's. Two main types of QFSM are well known: One-way QFSM (1QFSM) [AF98, MC00] and two-way QFSM (2QFSM)[AW02, KW97]. As will be illustrated and explained the 1QFSM, can accept sequentially classical input, quantize it, process it and measures its quantum memory after each operation (Figure 9). In this work the focus is on the synthesis of the 1QFSM from Figure 9(b). From now on the general designation of QFSM will refer to 1QFSM in this work. Other type of described QFSMs will be specifically named.

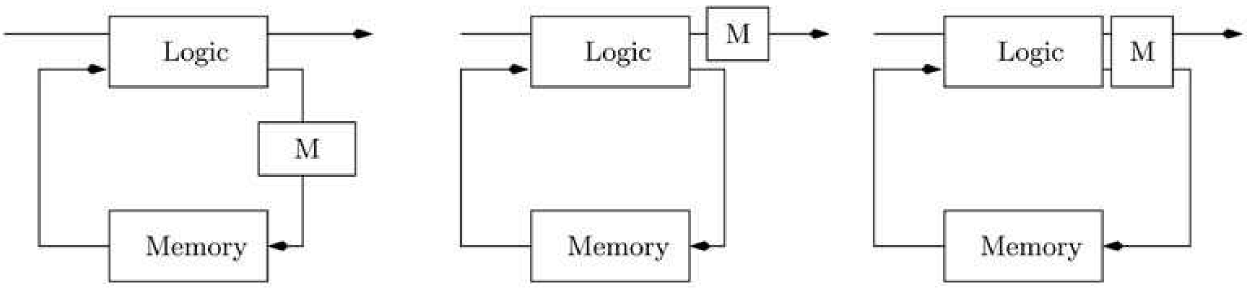


Fig. 9. Schematic representation of a 1QFSM; (a) after each computation step the machine state is measured, (b) after each computation step the output is measured, (c) after each computational step the machine state and the output state are measured.

In contrast to that, the 2QFSM is designed to operate on quantum input data (allowing to put the reading head in superposition with the input tape, and requiring all the input data to be present at once for the maximum efficiency) and the measurement is done only at the end of a whole process.

Definition 3.1

Quantum State Machine - a QFSM is a tuple $\Gamma = \{Q, \Lambda, q_0, Q_{ac}, Q_{rjI}, \delta\}$, where Q is a finite set of states, σ is the input alphabet, δ is the transition function. The states $q_0 \in Q'$, $Q_{ac} \subset Q$ and $Q_{rj} \subset Q$ are the initial states, the set of accepting states and the set of rejected states, respectively. □

The QFSM machine action maps the set of machine states and the set of input symbols into the set of complex machine next states. The computation of such machine is required to be done using unitary operators and is performed on the basis set B^q using unitary operators U_θ, $\theta \in \Theta$. In particular the QFSM uses a set of Unitary Operators corresponding to the input of input characters on the input tape. Thus for a given string to be processed and prior to the whole process termination (string either accepted or rejected), the overall processing can be represented as:

$$MU_{\theta_n}MU_{\theta_n-1}MU_{\theta_n-2}\dots MU_{\theta_3}MU_{\theta_2}MU_{\theta_1}|q_0\rangle \tag{24}$$

with MU_{θ_n} being the application of the U_{θ_n} operator to the current state and creating the configuration $U_{\theta_n}|q\rangle$ followed by the measurement of the current state M (projecting the state into G).

The 1QFSM was proven to be less powerful or equally powerful to its classical counterpart 1FSM [Gru99, KW97] in that it can recognize the same classes of regular languages as the classical FSM can recognize.

The above described 1QFSM is also called the measure-many quantum finite automaton [KW97]. A model called measure-once quantum finite automata was also introduced and studied by Moore [MC00]. The measure-many 1QFSM is similar to the concepts of the 2QFSM. For comparison we illustrate the main differences between the 1QFSM and 2QFSM below.

Example 3.1.1 1QFSM

Let $Q = \{|q_0\rangle, |q_1\rangle\}$be two possible states (including the accepting and rejecting states) of a single-qubit machine M and with transition functions specified by the transitions defined in eq. 25 corresponding to the state diagram in Figure 10a.

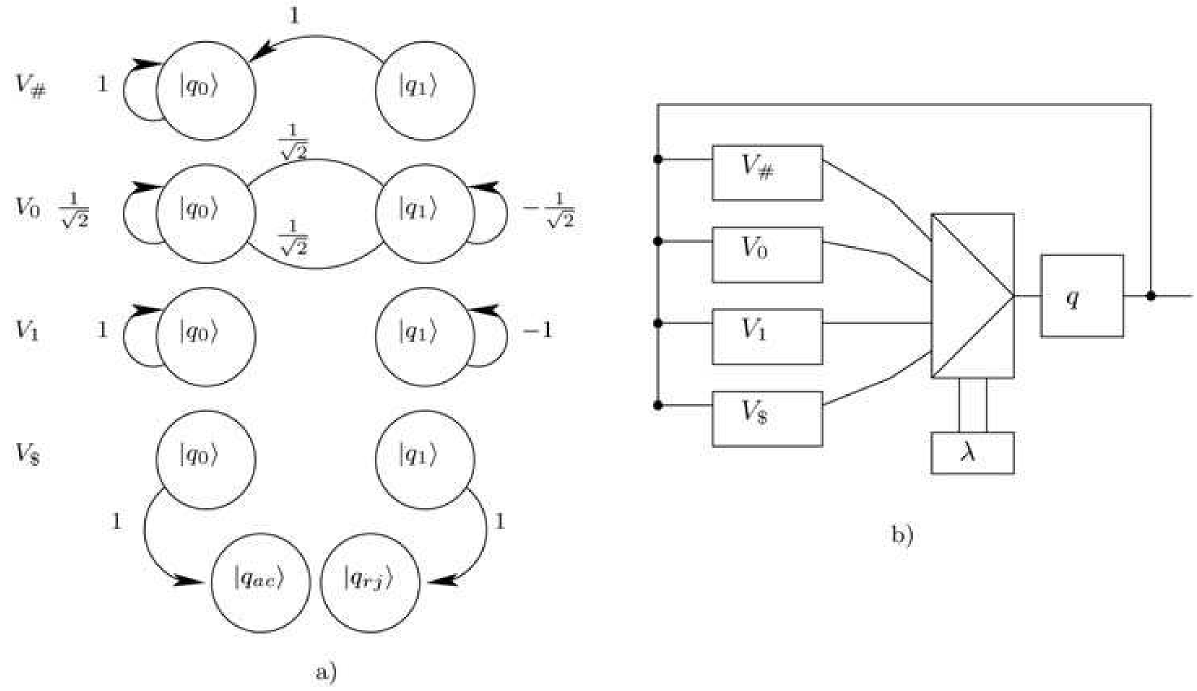


Fig. 10. (a) State transition diagram for the 1QFSM defined by the transition function 25, (b) the representation of the QFSM using quantum multiplexers. Observe two control outputs $|q\rangle$ specifying the machine action/states and the input symbols selecting the appropriate unitary transform V_λ for $\lambda \subset \{\#, \$, 0, 1\}$.

$$\begin{array}{ll} & V_0|q_0\rangle = \frac{1}{\sqrt{2}}|q_0\rangle + \frac{1}{\sqrt{2}}|q_1\rangle \\ V_{\#}|q_i\rangle = |q_0\rangle & \\ V_{\$}|q_0\rangle = |q_{ac}\rangle & V_0|q_1\rangle = \frac{1}{\sqrt{2}}|q_0\rangle - \frac{1}{\sqrt{2}}|q_1\rangle \\ V_{\$}|q_1\rangle = |q_{rj}\rangle & V_1|q_0\rangle = |q_0\rangle \\ & V_1|q_1\rangle = -|q_1\rangle \end{array} \tag{25}$$

The machine M, specified in eq. 25 represents a state machine that uses the H gate when the input is 0 ($V_0 = H$) and the Pauli-Z rotation gate when the input is 1 ($V_1 = Z$). Observe that machine M would have different behavior for measure-once and measure-many implementation. In the measure-many case, the machine generates a quantum coin-flip while receiving input 0 and while receiving input 1 the Pauli-Z rotation is applied. Observe in the measure-once case, that for example for the string input θ = ″010″ the many-measure machine will implement a NOT using [H][Z][H]. □

Note that in this approach to QFSM each input symbol λ∈{#, $, 0, 1} is represented by a unitary transform that can be seen as shown in Figure 10. No measurement is done here on |q⟩ while the sequence of quantum operators is applied to this state. The 2QFSM operates on a similar principle as the 1QFSM model but with the main difference being the application of the measurement. This is schematically shown in Figure 11 for the completeness of explanation.

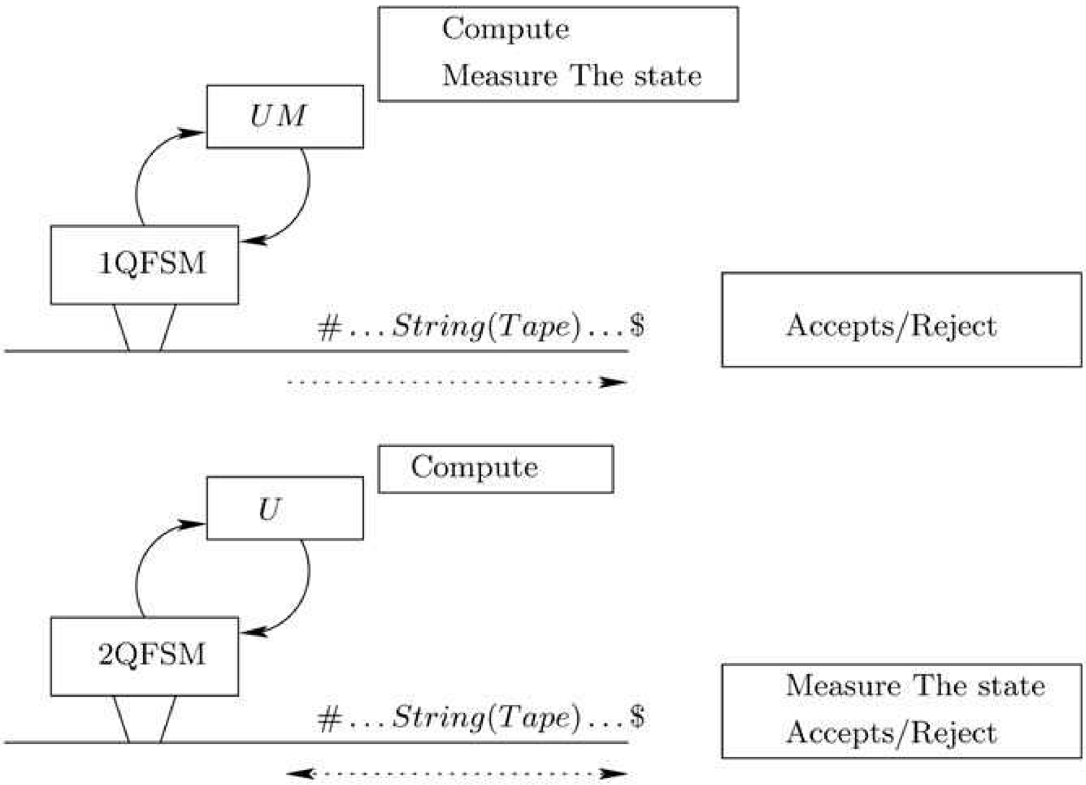


Fig. 11. Schematics representing the difference between the 1QFSM and 2QFSM. On the top, the 1QFSM - for each input character read from left to right from the tape, a unitary transform U is applied on the state and the state is measured. On the bottom, the 2QFSM moves on the input tape left and right, the unitary transform U is applied on the state and only once the computation is terminated the final state is observed/measured.

3.2 Quantum logic synthesis of sequence detectors

The problem to synthesize the QFSM is to find the simplest quantum circuit for a given set of input-output sequences thus letting the state assignment problem for this machine be directly solved by our synthesis algorithm. This direct synthesis approach can be applied to binary, multiple-valued and fuzzy quantum machines with no principle differences - only fitness functions are modified in an evolutionary algorithm [LPG+03, LP05].

Let us assume that there exists a sequential oracle that represents for instance Nature, robot control or robot's environment. In our example this oracle is specified by a state diagram in Figure 12a. This oracle can represent partial knowledge and a deterministic or probabilistic machine of any kind. Assume that there is a clearing signal (denoted by an arrow in Figure 12a) to set the oracle into its initial state. By giving initial signals and input sequences and observing output sequences the observer can create a behavior tree from Figure 12b.

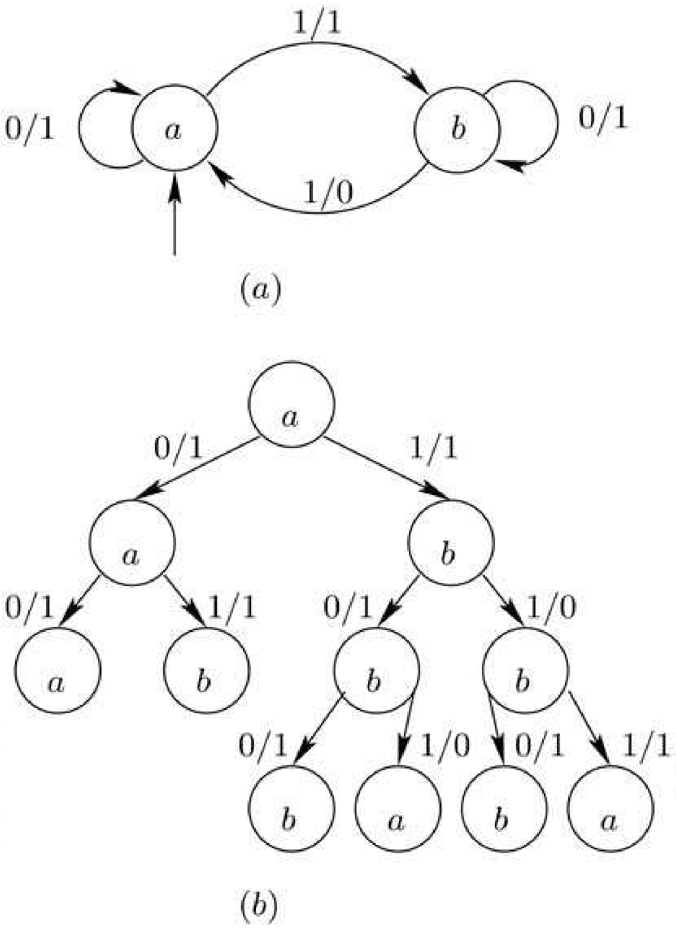


Fig. 12. Example of a deterministic oracle and its diagnostic tree.

As in general this oracle is never fully known, we perform experiments with it to determine some of its input-output behaviors. Assume that the oracle from Figure 12a is represented by the sequences from the experiments. These input-output sequences are shown in eq. 26 with $|iqo\rangle$ represents the input qubit, the state qubit and the output qubit respectively. Observe that the diagnostic tree form Figure 12(b) shows the state with {*a*, *b*} and the inputs and the outputs as 0 and 1.

$$\begin{array}{c} |iq0\rangle \rightarrow |iqo\rangle \\ 000 \rightarrow 011 \\ 001 \rightarrow 011 \\ 100 \rightarrow 111 \\ 101 \rightarrow 110 \\ 110 \rightarrow 101 \\ 111 \rightarrow 101 \end{array} \tag{26}$$

As the full knowledge of the oracle is in general impossible - the oracle is approximated by sets of input-output sequences and the more such sequences that we create - the more accurate characterization of the oracle as a QFSM can be created.

The overall procedure for the detection of a sequence of length j can be summarized as follows:

1. Initialize all qubits of the quantum register to the initial desired state,
2. repeat j times:
    a. Initialize the input qubit to a desired state and set the output qubit to $|0\rangle$
    b. Apply the quantum operator on the quantum register of the QFSM
    c. Measure the output qubit and observe the result

Using the procedure describe above one can synthesize quantum circuits for oracles being well known universal quantum gates such as Fredkin. The input-output sequences found from this oracle are next used to synthesize the QFSM from Figure 13a. Figure 13b shows the state-diagram of the machine.

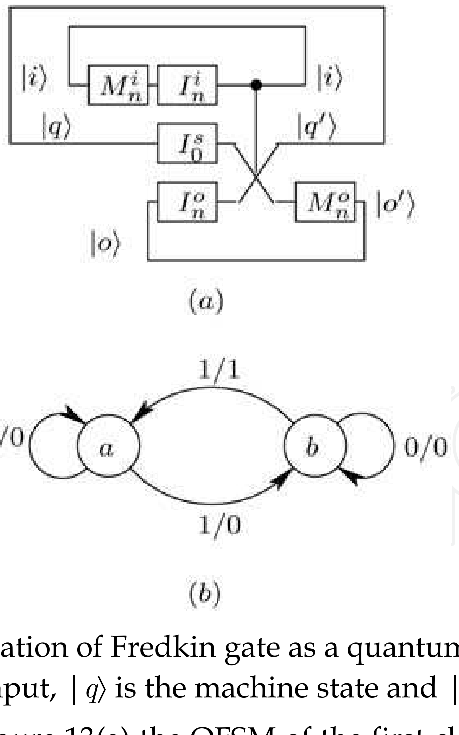


Fig. 13. Example of implementation of Fredkin gate as a quantum FSM of first class. Observe the notation where $|i\rangle$ is the input, $|q\rangle$ is the machine state and $|o\rangle$ is the machine output.

We will call the machine in Figure 13(a) the QFSM of the first class. This is because both the output and the input qubits are initialized after each computation. Observe that it is represented with feedback lines as in Figure 9 with input and output being initialized for each input and the state initialized only once - at the beginning of the computation. The interested reader can read more on this representation in [LP09], however it is important to understand that the feedback lines are shown here only as the equivalent notation to the classical FSM as in Figure 9. The circuit-based approach to QFSM does not require this notation as this ″loop″ is represented by the fact that the quantum qubit preserves its state [LP09].

A set of input-output sequences defining partially the "Fredkin QFSM" is represented in eq. 27.

$$\begin{array}{c} |iq0\rangle \rightarrow |iqo\rangle \\ 000 \rightarrow 000 \\ 001 \rightarrow 000 \\ 100 \rightarrow 100 \\ 101 \rightarrow 100 \\ 110 \rightarrow 101 \\ 111 \rightarrow 101 \end{array} \qquad (27)$$

A class two QFSM has in turn the initialization I_n applied only to the input qubit. This way the generated sequence is now expressed not only as a function $f(i,q)\oplus|0\rangle$ but rather as $f(i,q)\oplus|o\rangle$. This means that now the output is directly dependent also on the previous output state. This QFSM of the second class is shown in Figure 14. The difference between the QFSM of the first and of the second class can be seen on the output qubit $|o\rangle$ where in the case of the QFSM of the first class the initialization I_n^o means the initialization of the output at each computation step while the class two QFSM uses I_0^o initializes the output only once, at the beginning of the computation.

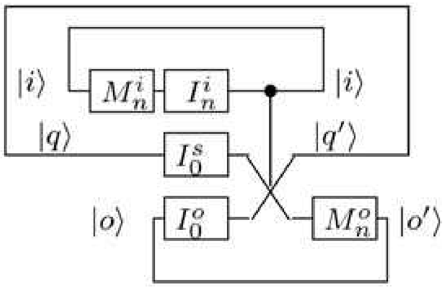


Fig. 14. Example of implementation of Fredkin gate as a quantum FSM of second class where the output is initialized only once and the measurement is done either after each input or only completely at the end.

For instance, a class two QFSM constructed from a "Fredkin oracle" differs from the class by different possible state transition. This is shown in Table 1. The first column represent the current state of the quantum register build from the input, state and output qubits $|iqo\rangle$. The second column shows the state transitions of the class one QFSM. Observe that as the output qubit is always being initialized to $|0\rangle$ only four possible initial states exists (see eq. 27). The third column representing the state transitions of the class two QFSM and as can be seen in this case the state transition function is the full "Fredkin oracle" function.

Moreover, the difference between the first and the second class of these QFSM's has also deeper implications. Observe that the QFSM presented in this paper, if implemented without the measurement on the output and the input qubit (the measurement is executed only after l computational steps) the QFSM becomes the well-known two-way QFSM

PS	NS_{one}	NS_{two}
$\vert iqo\rangle$	$\vert iqo\rangle$	$\vert iqo\rangle$
000	000	000
001	—	001
010	010	010
011	—	011
100	100	100
101	—	110
110	101	101
111	—	111

Table 1. Comparison of the state transition between the class one and class two QFSMs

[KW97] because the machine can be in superposition with the input and the output. This is equivalent to stating that the reading head of a QFSM is in superposition with the input tape as required for the time-quadratic recognition of the $\{a^n b^n\}$ language [KW97].

Observe that to represent the 1-way and the 2-way QFSM in the circuit notation the main difference is in the missing measurement operations between the application of the different *CU* (Controlled-U) operations. This is represented in Figures 15 and 16 for 1-way and the 2-way QFSMs, respectively.

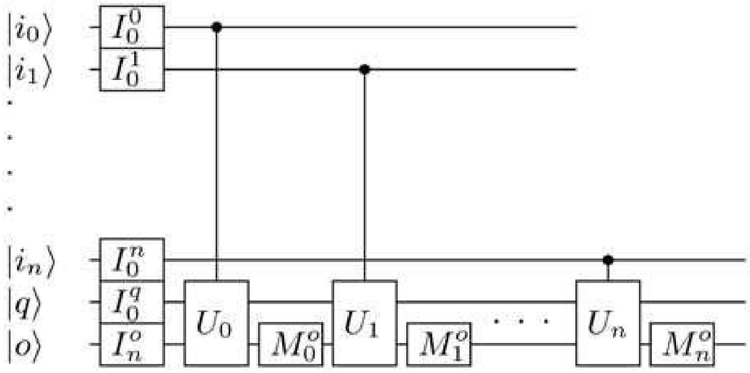


Fig. 15. Example of circuit implementing 1-way QFSM.

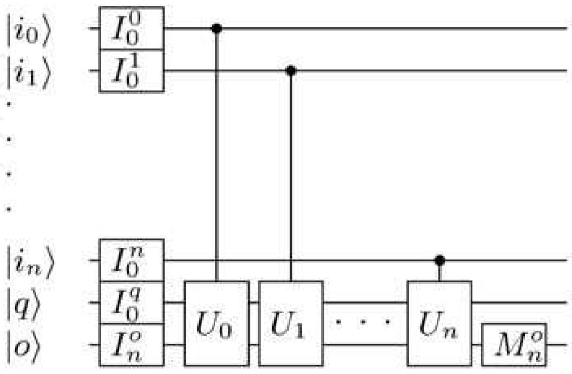


Fig. 16. Example of circuit implementing 2-way QFSM.

An interesting example of QFSM is a machine with quantum controls signals. For instance a circuit with the input qubit in the superposition generating the EPR quantum state [NC00] is shown in Figure 17.

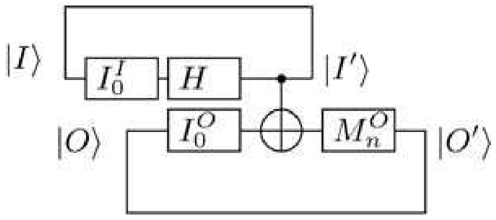


Fig. 17. Example of the EPR circuit used as a QFSM.

Observe the behavior of this QFSM as both class one and class two machine given in Table 2. In this case the distinction between the class one and class two machines is negligible because any measurement of the system collapses the whole system as the result of the entanglement present in it.

PS	NS_{one}	NS_{two}
$\|iqo\rangle$	$\|iqo\rangle$	$\|iqo\rangle$
00	00+11	00+11
01	–	01+10
11	–	00+11
10	01+10	01+10

Table 2. Comparison of the state transition between the class one and class two EPR circuit QFSM

Figure 17 shows that because of the entanglement this machine has two distinct possible recognizable sequences. When the machine uses exclusively the output qubit initialized to $|0\rangle$ the possible initial states are only $|00\rangle$ and $|10\rangle$ because the measurement of the output state resulting in $|11\rangle \xrightarrow{I_n^O} |10\rangle$ and $|01\rangle \xrightarrow{I_n^O} |00\rangle$.

4. Evolutionary algorithms and quantum logic synthesis

In general the evolutionary problem solving can be split into two main categories; not separated by the methods that each of the trends are using but rather by the problem representation and by the type of problem solved. On one hand, there is the Genetic Algorithm (GA) [Gol89, GKD89] and Evolutionary strategies (ES) [Bey01, Sch95] that in general represents the information by strings of characters/integers/floats and in general attempts to solve combinatorial problems. On the other hand the design of algorithms as well as state machines was traditionally done by the Genetic Programming (GP) [Koz94, KBA99] and the Evolutionary Programming (EP) [FOW66, ES03].

Each of this approaches has its particular advantages and each of them has been already more or less successfully applied to the Quantum Logic synthesis. In the EQLS field the main body of research was done using the Genetic Programming (GP) for the synthesis of either quantum algorithms and programs [WG98, Spe04, Lei04, MCS04] or some specific types of quantum circuits[WG98, Rub01, SBS05, SBS08, LB04, MCS05]. While the GP approach has been quite active area of research the Genetic Algorithm approach is less popular and recently only [LP08, YI00] were using a Genetic Algorithm for the synthesis of quantum circuits. However, it was shown in [LP09] that it is also possible to synthesize quantum finite state machines specified as quantum circuit using a GA. The difference between the popularity of the usage between the GP and the GA for EQLS is mainly due to fact that the problem space of quantum computing is not well known and is extremely large. Thus synthesizing quantum algorithms or circuits using the circuit approach (as in GA) can be much harder than using a rule-based or a program based approach (as in GP). Thus one could conclude that the GP approach deals only with the required information (programming, logic rules, relations) while the GA circuit based approach synthesize the overall unitary operator without any regards to the structure of the required information itself.

5. Genetic algorithm

A Genetic algorithm is a set of directed random processes that make probabilistic decisions - simulated evolution. Table 3 shows the general structure of a GA algorithm used in this work and this section follows this structure with the focus on the information encoding in the individuals and on the evaluation of the designed QFSMs that are created by the GA.

```
$t \leftarrow 0$;
initialize(P(t));                              /* initial population */
while (not termination-condition) do
evaluate(P(t));                                /* evaluate fitness */
$t \leftarrow t + 1$;
$Q_s(t) \leftarrow select(P(t \leftarrow 1))$;        /* selection operator */
$Q_r(t) \leftarrow recombine(Qs(t))$;             /* crossover operator */
$P(t) \leftarrow mutate(Q_r(t))$;                 /* mutation operator */
end while
```

Table 3. Structure of a Genetic Algorithm

5.1 Encoding/Representation

For quantum logic synthesis the representation that we use is based on the encoding introduced in [LPMP02]. This representation allows to describe any Quantum or Reversible circuit [LPG+03, LP02]. All individuals in the GA are strings of ordered characters (each character representing a quantum gate) partitioned into parallel Blocks (Figure 18). Each block has as many inputs and outputs as the width of the quantum array (five in the case

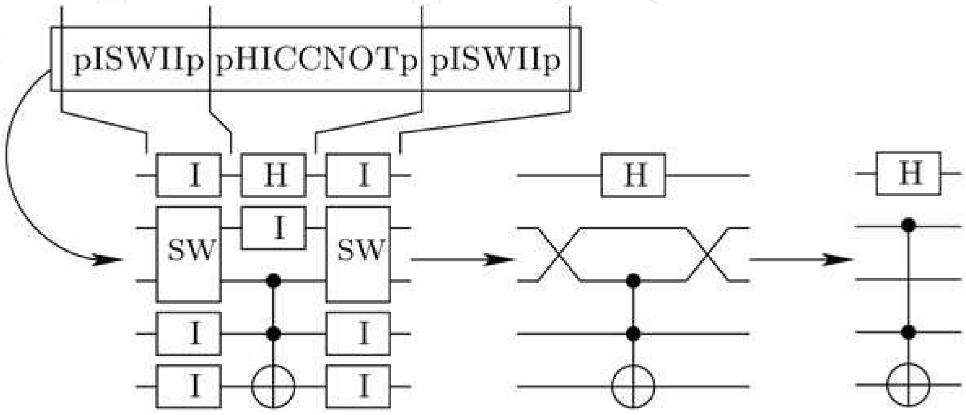


Fig. 18. Transformation of a QC from the chromosome (on the top) encoded string, to a final quantum circuit notation representation of this QC (on the right). Here *SW* is a Swap gate, H is a Hadamard gate and I is a Identity. In the middle there is one *CCNOT* (Toffoli) gate.

of Figure 18). The chromosome of each individual is a string of characters with two types of tags. First a group of characters is used to represent the set of possible gates that can be used in the individual string representation. Second, a single character ′p′ is used as a separator between parallel blocks of quantum gates. An example of a chromosome can be seen in Figure 18. In this encoding each space (empty wire or a gate) is represented by a character with appropriate decoding shown. Our problem-specific encoding was applied to allow the construction of as simple genetic operators as possible. The advantage of these strings is that they allow encoding of an arbitrary QL or RL circuit without any additional parameters. Several such parameters were used in previous research [LPG+03, LP05] and using them made the genetic algorithm more complicated. Please note that only the possibility to move gate characters, remove and add them to the chromosome consequently make it possible to construct an arbitrary circuit and also to modify this circuit in order to optimize it.

5.2 Initialization steps of GA

The GA requires an input file (c.f. Pseudo-Code 28 and Pseudo-Code 29) which specifies all input parameters and required settings.

$$\begin{aligned}
&20:\text{Measurement}:1\\
&21:\text{Measured qubits indexes:}0\\
&22:16\\
&22:0:(1,0)\\
&23:1:(1,0)\\
&\quad\vdots\\
&38:0:(1,0)
\end{aligned} \tag{28}$$

However, for the clarity of explanation we focus only on particular settings required for the synthesis of the QFSM. The lines (20-38) shows the to search for a QFSM recognizing a sequence, first the measurement is required (line 20), the index of the output qubit is given (line 21) and finally the desired input sequence is given. This is done by both specifying the input value (here 0 or 1) and the probability of detection (here 1). Observe that the probabilities are specified as complex numbers with only the real component defined, e.g. (1,0). The use of complex coefficients for these observation probabilities is due to the fact that as in our previous work [LP05, Luk09] it allows to specify don't cares. For instance the coefficient (0,1) represents a logical don't care.

The GA has several other settings, (common to most of GA methods) but also requires to specify circuit specific parameters. The initial circuits are created with a random size within the interval specified by a maximal (t_{max}) and minimal number of segments (t_{min}) in each individual (chromosome). Thus the size of the chromosome is not limited during the lifetime of an individual to a precise value, rather each individual has a dynamically changing genome within the bounds defined by the above variables. The presented GA is a subclass of the Messy GA [GKD89].

Another important parameter is related to the cost of the implemented Quantum Circuit. Each evolutionary run has specified the minimal cost *MinCost* that represents the known minimum for the target function or device. If such minimal value is not known, a small value is used so that it always underestimates a possible minimal cost of the implementation. This circuit cost value is used in the cost function described in Section 5.4.1.

$$
\begin{aligned}
&44 :1\\
&45 :1\\
&46 :\text{wire}\\
&47 :(1,0)(0,0)\\
&48 :(0,0)(1,0)\\
&\quad\vdots\\
&64 :2\\
&65 :1\\
&66 :\text{Controlled\_V}\\
&66 :(1,0)(0,0)(0,0)(0,0)\\
&67 :(0,0)(1,0)(0,0)(0,0)\\
&68 :(0,0)(0,0)(0.5,0.5)(0.5,-0.5)\\
&69 :(0,0)(0,0)(0.5,-0.5)(0.5,0.5)
\end{aligned}
\tag{29}
$$

The input specifications also include the elementary quantum gates to be used as components, like the single qubit H, X, Y, Z or V gates and two qubit operations such as *CNOT* or *CV*, which are the building blocks of the quantum circuits to be found. The quantum gates are represented as quantum unitary (and Hermitian) matrices with the cost specified for each gate. This is shown in eq. 29, where for each input gate the number of wires and its cost is given as well. For instance, lines 66 to 69 in eq. 29 shows the unitary matrix of the *CV* gate[BBC+95], line 64 shows the number of qubits of this gate and the line 65 shows its cost.

Observe that each unitary matrix is specified by complex coefficients with real and imaginary component. For instance (1, 0) represents the real state while (0.5, 0.5) represents a complex state with coefficient $\frac{1+i}{2}$.

In the presented experiments various sets of Quantum gates have been used but only the most succesful runs are presented. In particular only circuits with the most common gates are shown. These gates include single-qubit gates such as Pauli rotations, the V and V† gates, two-qubit gates such as CNOT, CV and CV† and three-qubit macros such as Toffoli gates.

5.3 Evaluation of synthesis errors in sequential quantum circuits

In order to properly evaluate a a QFSM for sequence detection the measurement operation must by applied on several occasions during the detection procedure. As was explained in the section 2, a quantum system must be measured in order for the information to be obtainable and readable in the macro world. Moreover, the measurement is a vital operation if one desires to reuse a quantum state. Recently, it was proven that a unknown quantum state cannot be completely erased [PB99] but is also easily understandable by observing the nature of the quantum computing.

The simplest explanation of the impossibility of completely erase an unknown state is due to the fact that there is no such a reversible quantum operation that would bring any quantum state to let's say the $|0\rangle$ state. This is because every reversible operation is a permutation (even when it contains complex coefficients) and any operation that would achieve such a state reduction is inherently non reversible and by default non-quantum. An example of such non-reversible operation is shown in eq. 30.

$$\Xi|\psi\rangle \rightarrow |0\rangle \Leftrightarrow \Xi = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \tag{30}$$

Thus measuring a quantum state allows to determine its observable and consequently allows to apply a Unitary transformation that would generate the desired state. The model of computation for Quantum Finite State Machines proposed in [LPK09] is used here as model. Figure 19 shows steps of evaluation of a sequential Quantum Circuit. Observe that this QFSM has one qubit $|q\rangle$ for state, one qubit $|i\rangle$ for input and one qubit $|o\rangle$ for output. From the classical point of view this can be seen as an instance of a Mealy finite state machine.

The synthesis process generates a unitary transformation matrix U, that during the evaluation is applied to the sequence of quantum states. Observe that both the input qubit and the output qubit must be measured in order to preserve a valid quantum state qubit $|q\rangle$ as well as allow to properly restore both the input and the output qubit. After each iteration of the computation (application of the U operator) the output qubit is set to $|0\rangle$ while the input qubit is set to either $|0\rangle$ or $|1\rangle$ depending on the the user specifications from the input file.

Equation 31 shows the first and the last step of the QFSM evaluation for the detection of the input sequence starting with s = {10011001110001}. Note that the detection requires that for all the input values but the last one the output qubit is set to $|0\rangle$ and is set to $|1\rangle$ for the last character.

$$\begin{aligned}
|000\rangle &\xrightarrow{I_1^1} |010\rangle \\
|010\rangle &\xrightarrow{U} |\psi\rangle \\
|\psi\rangle &\xrightarrow{M^0} |\rho\rangle \otimes |o\rangle \\
|\rho\rangle \otimes |o\rangle &\rightarrow p^0(0) \\
&\xrightarrow{M^1} |q\rangle \otimes |i\rangle \otimes |o\rangle \\
&\vdots \\
|q00\rangle &\xrightarrow{I_1^1} |010\rangle \\
|q10\rangle &\xrightarrow{U} |\psi\rangle \\
|\psi\rangle &\xrightarrow{M^1} |\rho\rangle \otimes |o\rangle \\
|\rho\rangle \otimes |o\rangle &\rightarrow p^0(1) \\
&\xrightarrow{M^1} |q\rangle \otimes |i\rangle \otimes |o\rangle
\end{aligned} \tag{31}$$

At the end of each evaluation sequence, the state of the output qubit and of the input qubit is determined by the measurement and can be reset with desired Unitary transformation to either $|0\rangle$ or to $|1\rangle$. The machine state qubit $|q\rangle$ is known only at the beginning of each evaluation sequence. This means that the state of the qubit can be in superposition or an orthonormal state. This also means that the machine state can be a multi qubit state that can become entangled between the various state qubits.

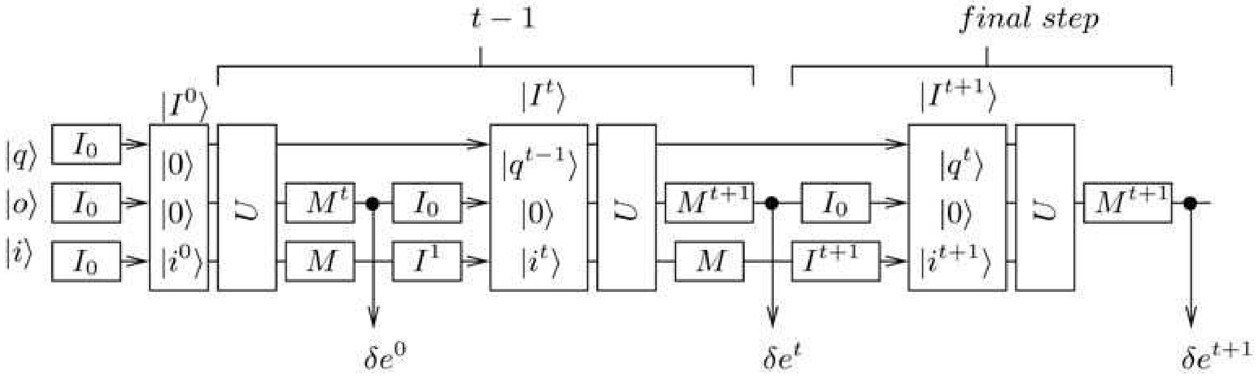


Fig. 19. Schematic representation of the process of evaluation of a QC as a Measure-Many (one-way)QFSM in this work.

Finally, the evaluation process is recored as a set of probabilities denoted as $p^0(0)$ and $p^0(1)$. They represent the probability of observation of the desired output 0 or 1 during the sequence detection. In particular, in this case the overall correctness of the detection can be written as:

$$\begin{aligned} p(s) =& p^0(0) \cdot p^0(0) \cdot p^0(0) \cdot \ldots \cdot p^0(1) \\ =& \left[\prod^{n-2} p^0(0)\right] \cdot p^0(1) \end{aligned} \tag{32}$$

To evaluate the error of the detector either the eq. 32 was used as a direct measure (it represents the correctness of the detection with respect to the selected observables), or a more standard calculation was used. The eq. 33 shows the standard RMS error computation. Both of these error evaluations are compared in the experimental section of this work.

$$p(s) = \frac{1}{n}\left(\left[\sum_{j=0}^{n-2}(1-p_j^0(0))^2\right] + (1-p_{n-1}^0(1))^2\right) \tag{33}$$

5.4 Fitness functions of the GA

During the search for the QFSM's a parameterized fitness function was used. This was done in order to allow the minimization for both the error and the cost of the synthesized Quantum circuit. This "weighted function" methodology was based on our previous experience in the evolutionary quantum Logic synthesis [LPG+ 03, LP05, LP09].

The parameterization allows to select the weight with which the error of the circuit and the cost of the circuit modifies the overall fitness value. The choice of this weight is left on the user that can decide what criteria of evaluation is more important. However, we experimentally determined some optimal settings that allowed correct circuits with minimal cost to be synthesized.

5.4.1 The cost function

The cost function is based on a parameter known as the *minimum cost* that is provided by the user and that permits to estimate a normalization constant. This means that the cost function acts as a bonus inversely proportional to the size of the circuit to the fitness function for a given estimated and unreachable minimum. In this work the *cost function* is defined by

$$G(c) = exp^{-\frac{(MinCost-Cost)^2}{\frac{Cost^2}{2}}} \tag{34}$$

where *Mincost* is the parameter given by the user and *Cost*, given by $\sum_{j=1}^{k} c_j$, is the sum of costs of all gates in the evolved circuit. Equation 34 was experimentally determined to be sensitive enough to influence both circuits far and close to the optimal cost.

5.4.2 The weighted fitness function

The weighted fitness functions used is shown in eq. 35 and an alternative version is in eq. 36. Both equations calculate the fitness value using the fitness function and the cost function together. In this case, the error of the circuit (QFSM) is calculated with respect to the overall probability of detecting the desired sequence as specified by eq. 32.

Each component of these weighted functions can be adjusted by the values of parameters α and β.

$$f_3 = \alpha(1-e) + \beta G(c) \tag{35}$$

$$f_4 = \alpha\left(\frac{1}{e+1}\right) + \beta G(c) \tag{36}$$

The reasons for these various fitness functions are the following:

- to allow different selection pressures during the individual selection process,
- by calibrating the cost to always underestimate the minimal possible size of the desired circuit it is possible to further manipulate the selection process.
- the parameterization allows in the extreme cases to completely eliminate the cost component and thus also includes fitness functions solely based on the correctness of the circuit.

For instance the fitness function 35 is not equal to one, unless both the cost of the circuit and the error are minimal. Thus a GA using such a weighted function has more freedom for searching a solution, because the fitness function is now optimizing the circuit for two parameters. Similarly in the case of the fitness function 36 which decreases the value of the fitness of longer circuits, therefore preferring the shorter ones. Thus individuals with different circuit properties will have equal fitness value.

5.5 Other evolutionary settings

For the clarity and the focus of this paper we present the rest of the settings in the Table 4. Only the final parameters are shown and in particular only those that were used during the runs that generated the presented results. To sum it up, the SUS[Bak87] selection method was used with n = 4 individuals. The mutation operator was used both on the level of individual quantum gates but also on the level of the parallel blocks. The crossover was a two parent, two point recombination process that preserves the width of the quantum circuit by selecting cut points only between the parallel blocks.

selection	Stochastic Universal Sampling with n = 4 individuals
mutation	Gate level, Segment level
crossover	Two point, two parent
population size	50,90,120
generations	500,900
elitism	yes

Table 4. Parameters of the GA used during the experiments.

5.6 CUDA acceleration

The CUDA framework was developed by NVIDIA for the growing usage of the GPU for computing tasks. The acceleration implemented in the GA is restricted only to the matrix calculation. Figure 20 shows where the CUDA acceleration is used.

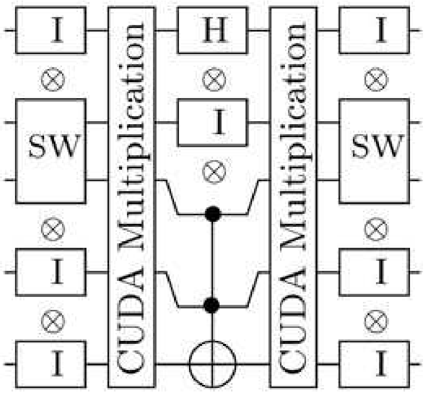


Fig. 20. Schema representing the usage of the CUDA accelerator in the computation of a Quantum Circuit Matrix Representation.

The reason for using the accelerated matrix multiplication only during the matrix multiplication and not for the Kronecker matrix product is the fact that the Kronecker product is less computationally expensive as it requires only $2^n \times 2^n$ multiplications while the matrix product requires $2^n \times 2^n$ multiplications and 2^n additions. Moreover, in order to maximize the CUDA usage it is more optimal to use multiplication on matrices of the same dimensions without requiring to re-parameterize the CUDA device. This is the case in the matrix multiplication between each parallel block in a Quantum Circuit.

6. Experiments and discussion

The experiments carried in this section confirms the possibility to design classical sequence detectors using the 1-way (measure many) circuit-based QFSM's model. The experimentation was done over a set of random sequences of various length. Each sequence was being tested for circuits with different number of state qubits. This was done in order to observe the role of embedding of the non-reversible sequence into a larger, reversible unitary matrix.

The general function that the QFSM generates on the output is described by the eq. 37

$$o(\lambda) = \begin{cases} 1 & ifj < length(s) \\ 0 & o.w. \end{cases} \tag{37}$$

with λ being a symbol read from the input and j is the index of the λ symbol in the sequence. Thus the minimal condition for each sequence to be detected properly is that the amount of the states is large enough to embed all the zero output to one half of the truth table. this is a required consequence because the QFSM must have both the output function and the state transition function reversible.

The experimentation was done for 5 randomly generated binary sequences with 7, 10, 15, 20 and 35 binary digits each. The sequences are shown in eq. 38

$$\begin{aligned} s_7 &= \{\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ \} \\ s_{10} &= \{\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ \} \\ s_{15} &= \{\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ \} \\ s_{20} &= \{\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ \} \\ s_{25} &= \{\ 1\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ \} \end{aligned} \tag{38}$$

Each sequence was synthesized using a circuit with 3,4,5 and 6 qubits. The six qubit circuit is large enough to embed even the largest sequence so as a reversible function is synthesizable. Figures 21, 24 and 27 shows some examples of obtained circuits for each of the sequences.

Figure 21 is an example of Quantum Circuit that was used to detect the s_7 sequence and does not use any probabilistic states. For the sake of understanding let us analyze the sequence detection procedure using this circuit. The desired sequence is $s_7 = \{\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ \}$ thus the set of input states is given by $\{|0^{\otimes^r}\rangle|00\rangle, |\phi 10\rangle, |\phi 00\rangle, |\phi 10\rangle, |\phi 10\rangle, |\phi 10\rangle, |\phi 10\rangle\}$, with $|\phi\rangle$ being the unmeasured component of the automata state. Naturally there are cases where it is going to be determined by the measurement but for the clarity it is left in symbolic form and thus allowing it to be in superposed or entangled state.

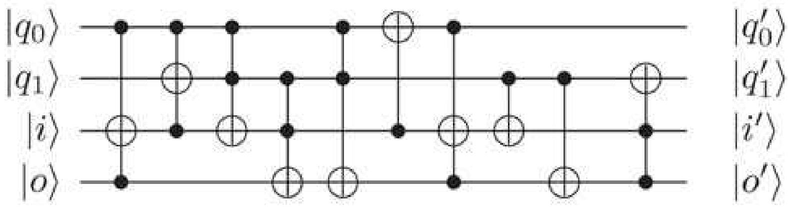
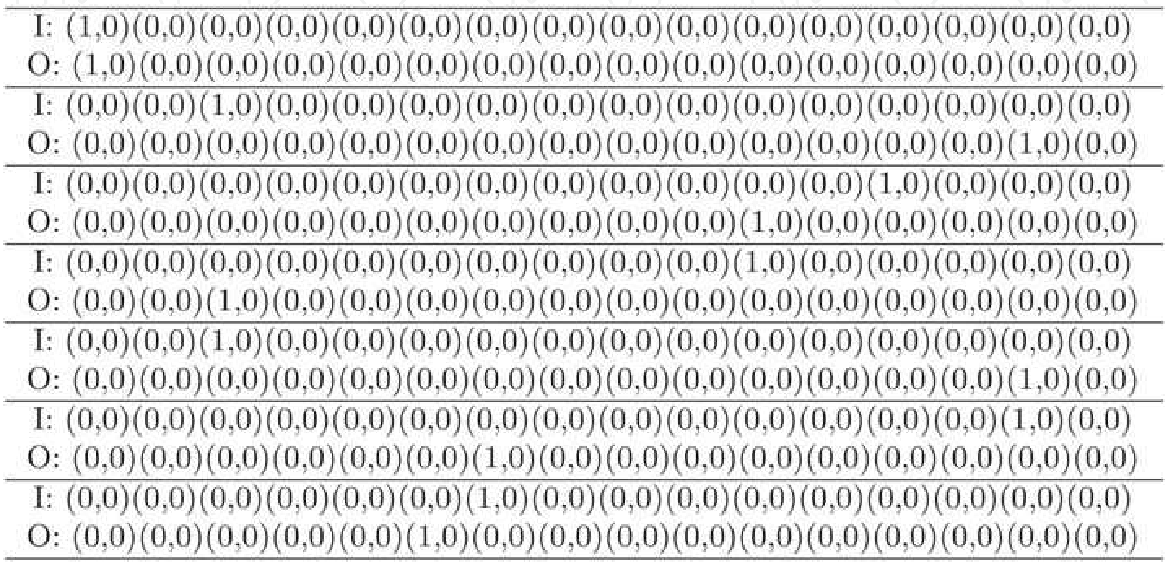


Fig. 21. s_7-sequence exact detector

The size of the QFSM is four qubits with two topmost qubits $|q_0\rangle$, $|q_1\rangle$ are the state qubits, $|i\rangle$ is the input qubit and $|o\rangle$ is the output qubit (Figure 21). Table 22 represents the consecutive states as obtained during the QFSM procedure described in this work (Section 5.3). In particular this QFSM shows that it recognize the given sequence without the use of any probabilistic or superposed states.

This can also be seen on the circuit matrix that can easily be build from the given sequence of gates. For clarity the state transition is also shown in the form of a equation (eq. 39).



I:	(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O:	(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
I:	(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)
I:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)
O:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)
I:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O:	(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
I:	(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)
I:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)
O:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
I:	(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O:	(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)

Fig. 22. Four qubits s_7-sequence detector with deterministic input and output states

$$\begin{aligned}
s_0 &\Rightarrow 0: && |0000\rangle \xrightarrow{U} |0000\rangle \\
s_1 &\Rightarrow 1: && |0000\rangle \rightarrow |0010\rangle \xrightarrow{U} |1110\rangle \\
s_2 &\Rightarrow 0: && |1110\rangle \rightarrow |1100\rangle \xrightarrow{U} |1010\rangle \\
s_3 &\Rightarrow 1: && |1010\rangle \xrightarrow{U} |0010\rangle \\
s_4 &\Rightarrow 1: && |0010\rangle \xrightarrow{U} |1110\rangle \\
s_5 &\Rightarrow 1: && |1110\rangle \xrightarrow{U} |0110\rangle \\
s_6 &\Rightarrow 1: && |0110\rangle \xrightarrow{U} |0101\rangle
\end{aligned} \tag{39}$$

Observe that two different steps can be clearly distinguished in eq. 39; first a standard step that acts directly on a previously generated machine state such as in steps s_0, s_3, s_4, s_5 and s_6, second a step that requires explicit modification of the previous machine state, in particular a state that requires an initialization of the output and/or the input qubit, such as shown in steps s_1 to s_2. Observe that this particular sequence detector does not requires - for this sequence – any re-initialization of the input qubit as a result of previous step; the input qubit is not modified by the automaton. Also observe that despite the fact that this circuit can generate quantum states, these states are not generated during the sequence s_7. This can be seen on Figure 23.

The states in a circle represent natural states as would be obtained by the cycle of the reversible function, while the states in the hexagons represents forced states that are obtained after modifying the input qubit. The Figure 23 also represents the forced states with one dotted arrow incoming and one outgoing dashed arrows. The arrow incoming to the forced state is indexed with a binary number representing the required input change so that the forced state is obtained. The outgoing arrow represents that the forced state is then used as a normal natural state; i.e. a Unitary transform is applied to it and the result is computed. For instance the s1 character recognition, starts with the machine in the state |0000⟩, which is forced to |0010⟩ and then the Unitary transformation is applied and yields |1110⟩ state. The whole sequence detection can be in this manner analysed from eq. 39 and Figure 23.

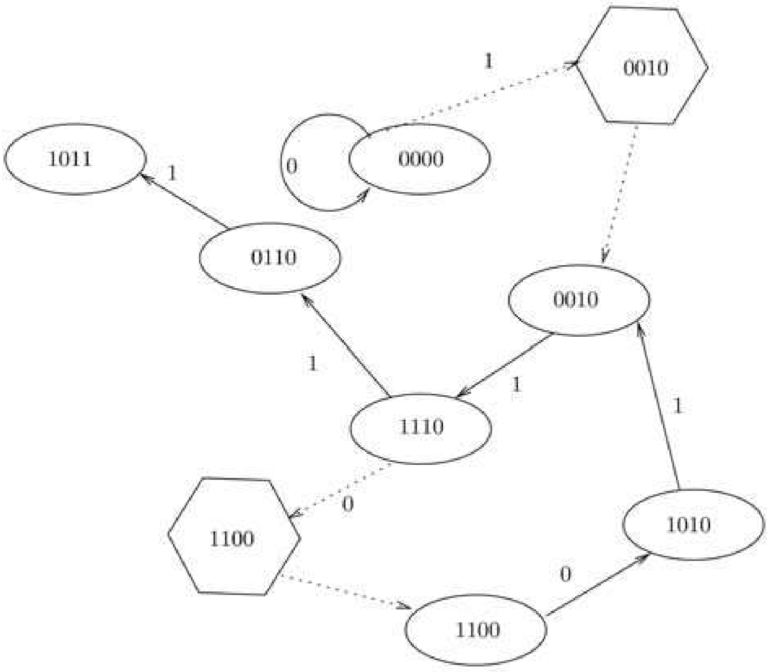


Fig. 23. The cycle of a Reversible Circuit used as a detector for the s_7 sequence

A more interesting example is shown in Figure 24. The displayed circuit also recognizes the same sequence s_7 but in this case the automaton uses probabilistic and superposed quantum states. This can be seen in Table 25; this table has every row split in half so that it fits in size. For more details eq. 40 shows step by step the process of recognition performed by this automaton. Observe that as the result of the last step the output of the circuit is |o⟩ = |1 ⟩ thus confirming the correct sequence has been detected.

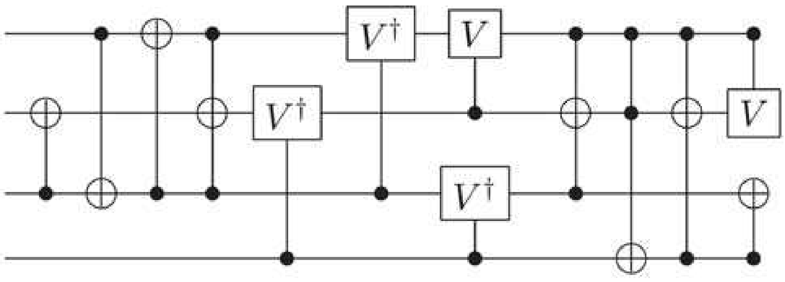
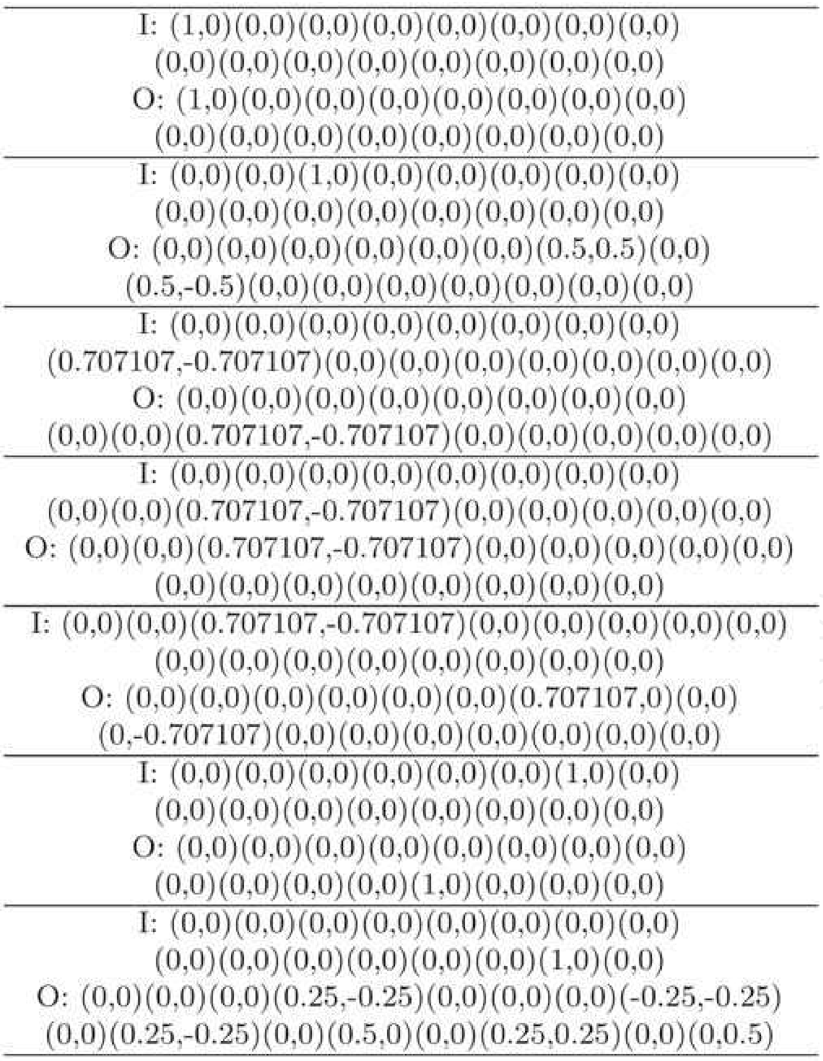


Fig. 24. s_7-sequence detector with probabilistic and superposed states

$$
\begin{aligned}
s_0 \Rightarrow 0: &\quad |0000\rangle \xrightarrow{U} |0000\rangle \\
s_1 \Rightarrow 1: &\quad |0000\rangle \rightarrow |0010\rangle \xrightarrow{U} \frac{1+i}{2}|0110\rangle + \frac{1-i}{2}|1000\rangle \\
s_2 \Rightarrow 0: &\quad \frac{1+i}{2}|0110\rangle + \frac{1-i}{2}|1000\rangle \rightarrow \frac{1-i}{\sqrt{2}}|1000\rangle \xrightarrow{U} \frac{1-i}{\sqrt{2}}|1010\rangle \\
s_3 \Rightarrow 1: &\quad \frac{1-i}{\sqrt{2}}|1010\rangle \xrightarrow{U} \frac{1-i}{\sqrt{2}}|0010\rangle \\
s_4 \Rightarrow 1: &\quad \frac{1-i}{\sqrt{2}}|0010\rangle \xrightarrow{U} \frac{i}{\sqrt{2}}|0110\rangle + \frac{-i}{\sqrt{2}}|1000\rangle \\
s_5 \Rightarrow 1: &\quad \frac{i}{\sqrt{2}}|0110\rangle + \frac{-i}{\sqrt{2}}|1000\rangle \rightarrow |0110\rangle \xrightarrow{U} |1100\rangle \\
s_6 \Rightarrow 1: &\quad |1100\rangle \rightarrow |1110\rangle \xrightarrow{U} \begin{aligned}[t] &\frac{1-i}{4}|0011\rangle + \frac{-1-i}{4}|0111\rangle + \frac{1-i}{4}|1001\rangle \\ &+ \frac{1}{2}|1011\rangle + \frac{1+i}{4}|1101\rangle + \frac{i}{2}|1111\rangle \end{aligned}
\end{aligned}
\tag{40}
$$



```
I: (1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O: (1,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)

I: (0,0)(0,0)(1,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0.5,0.5)(0,0)
(0.5,-0.5)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)

I: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0.707107,-0.707107)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0.707107,-0.707107)(0,0)(0,0)(0,0)(0,0)(0,0)

I: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0.707107,-0.707107)(0,0)(0,0)(0,0)(0,0)(0,0)
O: (0,0)(0,0)(0.707107,-0.707107)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)

I: (0,0)(0,0)(0.707107,-0.707107)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0.707107,0)(0,0)
(0,-0.707107)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)

I: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
O: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)(0,0)(0,0)

I: (0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)
(0,0)(0,0)(0,0)(0,0)(0,0)(0,0)(1,0)(0,0)
O: (0,0)(0,0)(0,0)(0.25,-0.25)(0,0)(0,0)(0,0)(-0.25,-0.25)
(0,0)(0.25,-0.25)(0,0)(0.5,0)(0,0)(0.25,0.25)(0,0)(0,0.5)
```

Fig. 25. Four qubits s_7-sequence detector with probabilistic input and output states

6.1 Sequence detection

The detection of a given sequence by the here formulated QFSM's can be analyzed starting from Reversible Circuits. Assume, the initial state is 0000 for the rest of this discussion. As example take the reversible (deterministic) detector such as given in figure 21. It is obvious that the power of properly detecting a given sequence; i.e. to generate a sequence $0 \times n + 1 \times 1$ is proportional to the cycle of this detector given by the reversible circuit for a fixed n and to the fact of having the cycle connected to a finish sequence either by a forced change of input or by a natural evolution.

To see this, just assume that the given circuit is specified by the following permutation cycle (0, 4, 8, 12)(2, 6, 10, 14)(1, 3, 5, 7, 9, 11, 13, 15). Clearly, the first cycle (0, 4, 8, 12) represents the states containing the 0 as input and 0 as output, the (2, 6, 10, 14) cycle represents the states having 1 for input and 0 as output and the last cycle represents all outputs having the output bit set to 1. The longest possible sequence this automaton can detect (without using the force states transitions) is of length 0 becausethe detecting cycle is disjoint from both the cycles identifying 0's and 1's.

For illustration assume the Reversible Circuit specifying the automaton be described by (0,6,4,2) (8,12,10,14,1) (3,5,7,9,11,13,15) permutation cycles. This automaton will not detect successfully any sequence if starting from the initial state 0000. This is shown in figure 26. Observe that no matter the input change of any of the state of the cycle (0,6,4,2) will always lead back to a state from the same cycle. Thus such a machine cannot generate a 1 on the output when starting in the state $|0000\rangle$.

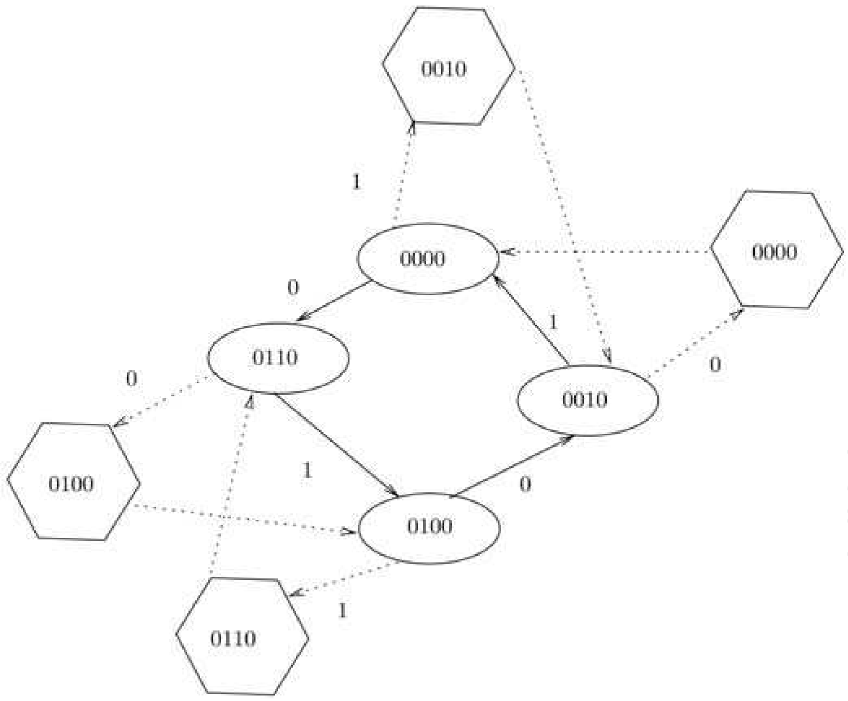
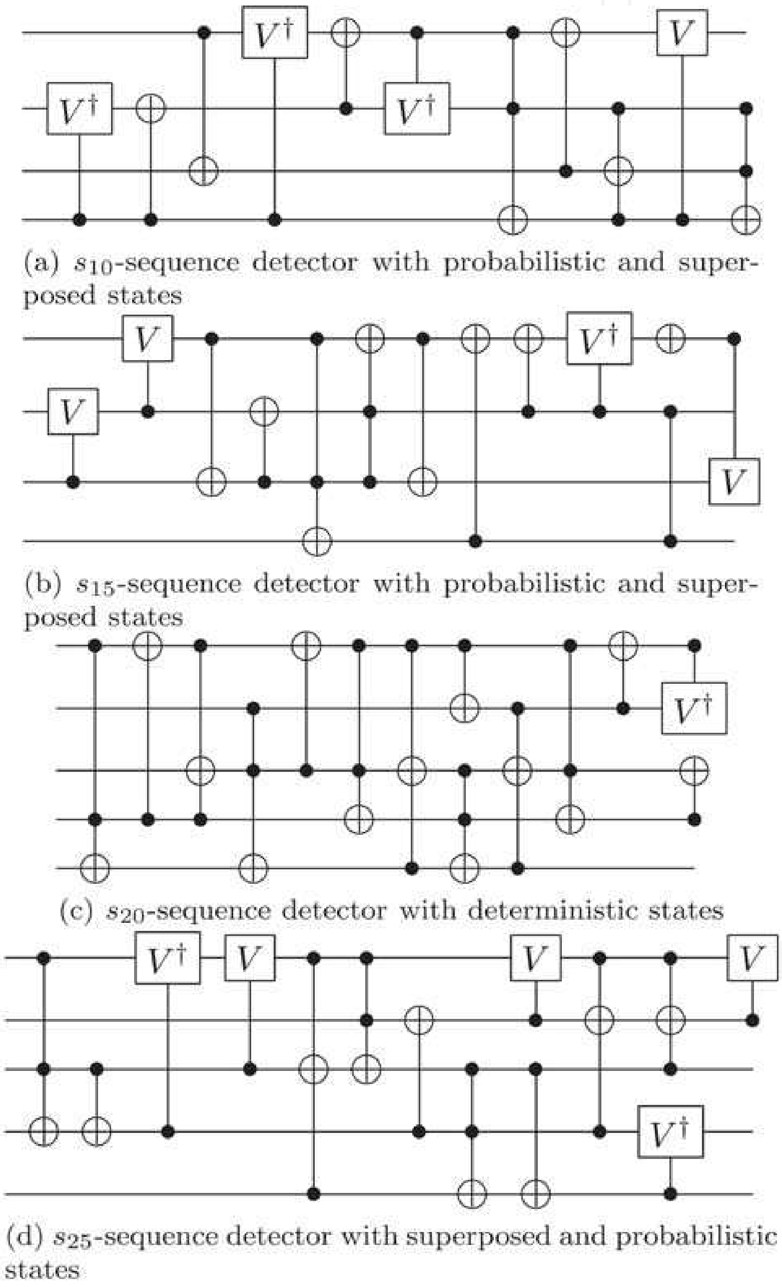


Fig. 26. The cycle of a Reversible Circuit used as a detector

The Figure 26 shows that in order to have a successful detector, at least one natural transition $|\phi\rangle \xrightarrow{U} |\phi'\rangle$ or a forced transition $|\rho\rangle\,|i\rangle\,|o\rangle \rightarrow |\rho\rangle\,|\bar{i}\rangle\,|o\rangle$ must lead to a cycle that allows to generate an output with value 1.

Now consider an Reversible Circuit defined by the permutations given by (0, 4, 8, 12, 3) (2, 6, 10, 14, 1) (5, 7, 9, 11, 13, 15). Such automaton now can detect any sequence that contains at least four consecutive 0’s or four consecutive 1’s. To maximize the length of a given sequence it is possible to allow the automaton to modify also its input qubit. In that case, as also seen in the presented protocol in the Section 5.3, the maximal complexity of the detected sequence is still equal to the sequence of maximum four 0’s and four 1’s.

The important cycles used for detection of the above specified Reversible circuit are shown in Figure 28. Observe that only two out of three cycles are shown as the last cycle contains all minterms that have 1 as output and thus can only be used as the end of sequence indicator. Also the cycles are shown only up to a final state. Thus for instance the state $|0011\rangle$ is not connected back to $|0000\rangle$ because once such state is attained the detection is terminated. Such specified detector will detect any sequence that terminates with four 1’s or four 0’s.



(a) s_{10}-sequence detector with probabilistic and superposed states

(b) s_{15}-sequence detector with probabilistic and superposed states

(c) s_{20}-sequence detector with deterministic states

(d) s_{25}-sequence detector with superposed and probabilistic states

Fig. 27. Quantum circuits detecting the sequences s_{10} to s_{25}

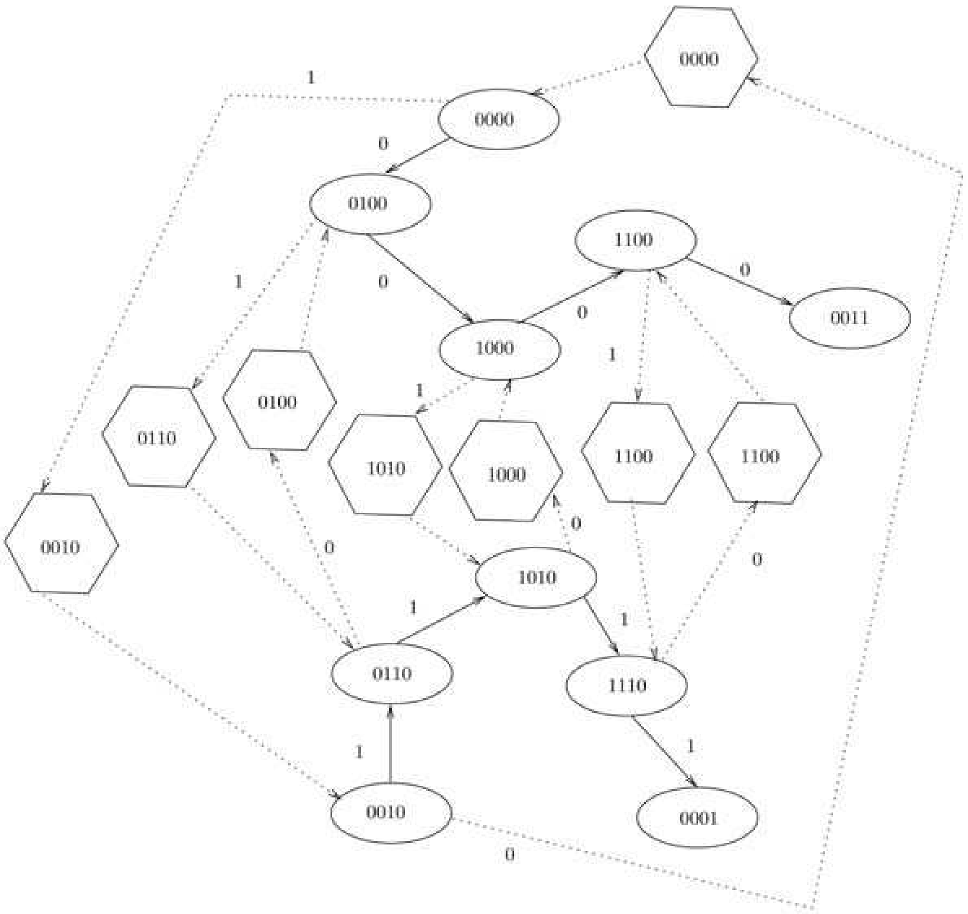


Fig. 28. The Reversible Circuit specified by the cycles (0,4,8,12,3) (2,6,10,14,1) (5,7,9,11,13,15) used as a detector

Finally observe that for a given sequence it is possible to design a detector that is either specified by only natural state transitions - as specified by the cycles of a reversible quantum function or by combining the cycle with forced transitions. The former method will always generate larger circuits while the later will allow more compact designs. However in the framework of the presented Quantum detectors these approaches are equivalent from the point of view of implementation. That is, at the begining of each computation cycle one need to know exactly the input quantum state. Thus the main adavantage in designing detectors with only natural state transitions resides in the fact that no initialization of the input qubit is required because it is set at the output of the previous computational cycle.

To close this discussion about the detectors it is possible to synthesize detectors using both purely Reversible or Quantum Unitary matrix. The size of the required circuit is dependent on the amount of continuous 0′s or 1′s however it is not restricted by it. It is straight forward to imagine such sequence detector that will have only smaller cycles and still will detect similar sequence. This is because if the unitary transform modifies the input qubit, smaller cycles can be combined to detect these particular sequences. For instance Figure 29 shows a portion of a detector specified by a Reversible circuit. This detector will detect among others the sequences terminating with three 0's or two 1's. Recall that only natural transitions are used for the detection procedure. Thus for instance in figure 29 $|1110\rangle$ changes to state $|1100\rangle$ when the input is changed from 1 to 0 and the consequent application of the Unitary matrix on this state generates an 1 on output. This is the final state, and it indicates that at least three 0 have been successfully detected before attaining it. The interested reader is encouraged to read more about reversible and quantum sequence detector in [LP09, LPK09]. Table 5 shows the minimum number of qubits that have been experimentally obtained in order to properly detect the sequences studied here. The sequence s_7 has a sequence of four 1's and a set of individual 1 and thus cannot be detected by less than circuit with 4 qubits.

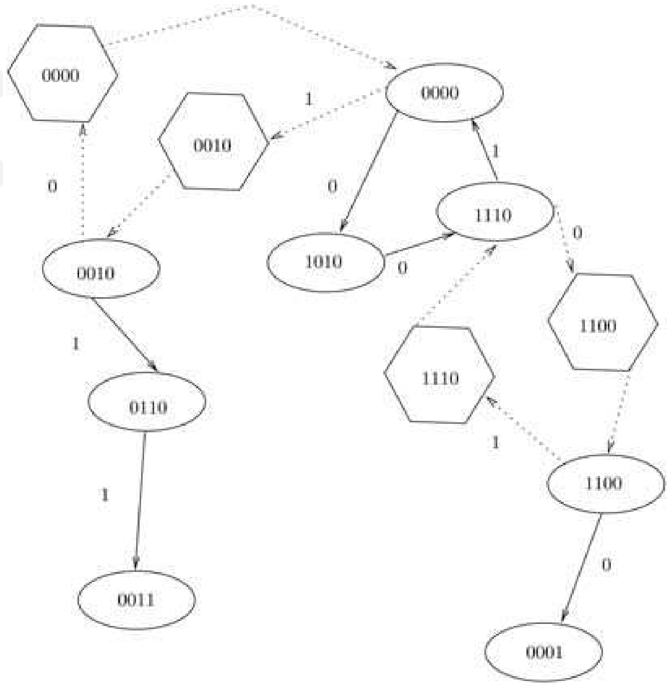


Fig. 29. The Reversible Circuit detecting sequences ending with four 0's or four 1's.

s_7	4
s_{10}	4
s_{15}	4
s_{20}	5
s_{25}	5

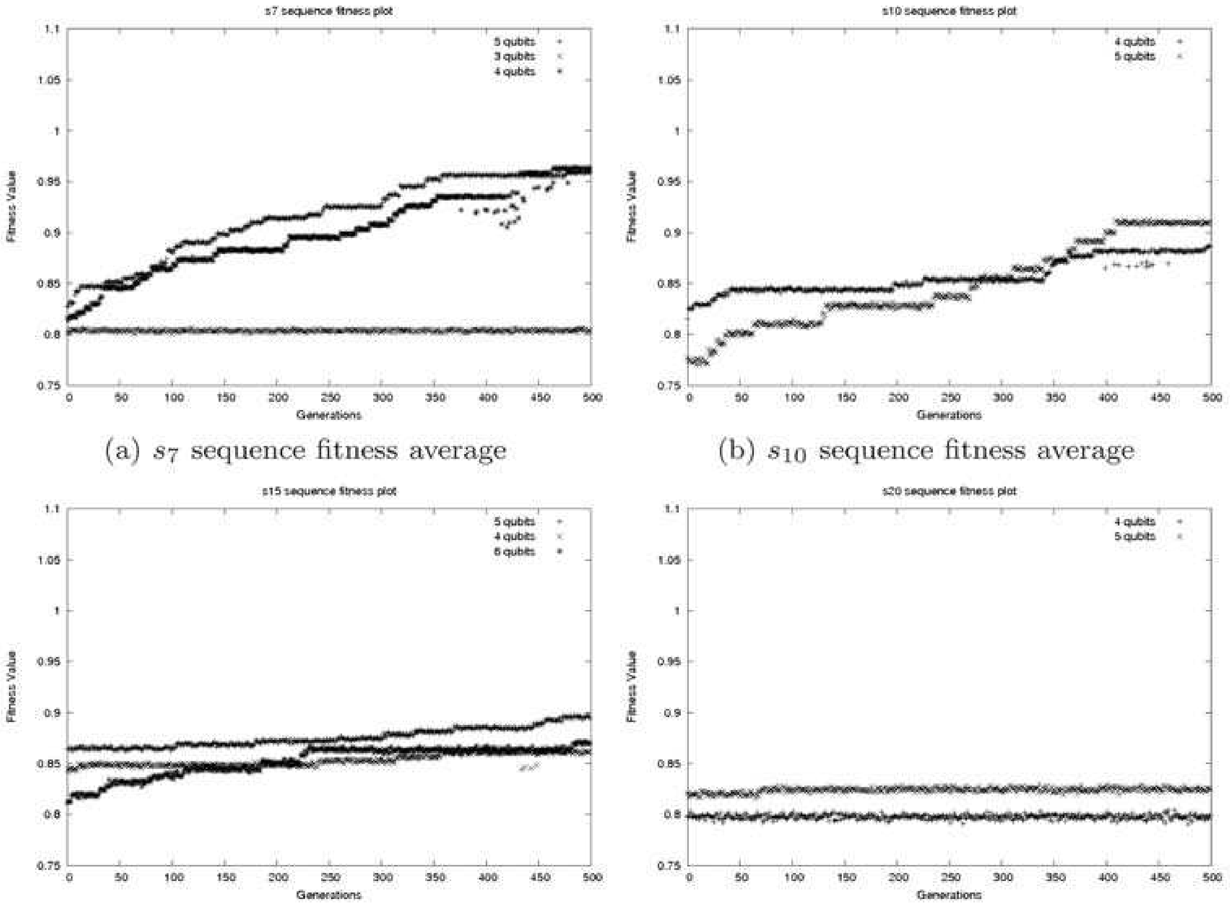
Table 5. Minimum number of qubits required to detect a sequence: experimental observations

The sequence s_{10} has two cycles at least: one with a sequence of two 1's followed by a 0 and a sequence of three 0's. It is also not possible to construct such detector on 3 qubits because the number of states required is at least 4 for both sequence and not counting the individual 0's and 1's. Similarly other sequences can be analyzed.

The Genetic Algorithm was run for multiple sizes for each sequence starting with three qubits. The search was terminated when a circuit satisfying the constraints was found and multiple searches were performed at the minimal width. Figure 30 shows the average of the Fitness value for the s_7, s_{10} and s_{15} sequences. The drawings show each curve over 500 generation cycles required for the detection of each of the sequences after which the maximum generation is attained. Each curve is an average of 15 runs and the most interesting feature is that similarly to quantum function logic synthesis the algorithm finds a circuit that is very close to the complete solution and then stagnates before finding a solution. This problem is related to both the difficulty of synthesis as well as the fact that

Quantum circuit are specified by Unitary matrices in which the error is of symmetric nature. Such error can be seen as a step function where with each step a pair of coefficient in the Unitary matrix is corrected.

Also observe how the fitness value stagnates with larger sequences with the presented qubits despite the fact that a solution was found for each sequence for the presented number of qubits. Interestingly, observe that the sequence s_7 to s_{20} are from the same class as they have been identified by detectors of similar size. This goes back to the discussion above about the limits of a Quantum and Reversible circuit to recognize a particular class of sequences.



(a) s_7 sequence fitness average

(b) s_{10} sequence fitness average

(c) s_{15} sequence fitness average

(d) s_{20} sequence fitness average

Fig. 30. Figures capturing the fitness average for four sequence detectors

7. Conclusion

In this paper we presented a methodology and we showed some experimental results confirming that our approach is possible in simulated environment. Also because all simulated elements of the presented experiments are based on existing Quantum operations, the simulated detectors are Quantum-realizable.

It is well known that the state assignment problem is a NP-complete problem [Esc93] and the finding a minimal State Assignment has been solved only for particular subsets of FSM's [LPD95] or using Quantum computing [ANdMM08]. This problem is here naturally solved (without addressing it). The setup of this experimental approach automatically generates a state assignment such that when the detection is successful the state assignment is as well.

This is a natural consequence of both the fact that the machine is reversible and the fact that the sequence is successfully identified.

The presented algorithm proved successful in the design of Quantum detectors. Despite the sequences were randomly generated the proposed approach was possible due to the hardware accelerated computational approach. For more details about this approach the reader can consult [LM09].

The synthesis of quantum detectors has not been completely explored and remains still an open issue mainly because Quantum computing implementation is not a well established approach. Each technology provides different possibilities and has different limitations. In some cases specification using Quantum circuits is the most appropriate in others Hamiltonians must be used. Thus one of the main remaining tasks is to completely describe Quantum detectors and formally define their issues related with implementation and define classes of circuits more approapriate for different technologies.

8. References

[AF98] A. Ambainis and R. Freivalds. 1-way quantum finite automata: strengths, weaknesses and generalizations. pages 332–341, Nov 1998.

[ANdMM08] M.P.M. Araujo, N. Nedjah, and L. de Macedo Mourelle. Quantum-inspired evolutionary state assignment for synchronous finite state machines. Journal of Universal Computer Science, 14(15):2532–2548, 2008.

[AW02] A. Ambainis and J. Watrous. Two-way finite automata with quantum and classical states. Theoretical Computer Science, 287(1):299–311, 2002.

[Bak87] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *In Proceedings of the Second International Conference on Genetic Algorithms and their Application*, pages 14–21, 1987.

[BBC^{+}95] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and Weinfurter H. Elementary gates for quantum computation. *The American Physical Society*, 5:3457–3467, 1995.

[Bey01] H.G. Beyer. *The Theory of Evolution Strategies*. Springer, 2001.

[BV97] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal of computing*, pages 1411–1473, 1997.

[BZ00] A. Blais and A. M. Zagoskin. Operation of universal gates in a solid state quantum computer based on clean josephson junctions between d-wave superconductors. *Phys. Rev. A*, 61, 2000.

[cbl] *GSL CBLAS*. http://www.gnu.org/software/gsl/manual/html node/GSLCBLAS-Library.html.

[cud] *NVIDIA CUDA*. http://www.nvidia.com/object/cuda learn.html.

[CZ95] J.I. Cirac and P. Zoller. Quantum computation with cold trapped ions. *Physical Review letters*, 74(20):4091, 1995.

[DiV95] P. DiVincenzo. Two-bit gate for quantum computation. *Physical Review A*, 50:1015, 1995.

[DKK03] L.M. Duan, A. Kuzmich, and H.J. Kimble. Cavity QED and quantum-information processing with 'hot' trapped atoms. *Physical Review A*, 67, 2003.

[Dun98] M. R. Dunlavey. Simulation of finite state machines in a quantum computer, 1998.

[EPR35] A. Einstein, B. Podolsky, and N. Rosen. Can quantummechanical description of physical reality be considered complete? *Phys. Rev.*, 47(10):777–780, May 1935.

[ES03] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

[Esc93] B. Eschermann. State assignment for hardwired vlsi control units. *ACM Comput. Surv.*, 25(4):415–436, 1993.

[FOW66] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.

[GKD89] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 3:493– 530, 1989.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, MA, 1989.

[Gra81] A. Graham. *Kronecker Products and Matrix Calculus With Applications*. Ellis Horwood Limited, Chichester, U.K., 1981.

[Gru99] J. Gruska. *Quantum computing*. Osborne/McGraw-Hill,U.S., 1999.

[KBA99] J.R. Koza, F.H. Bennett, and D. Andre. *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, California: Morgan Kaufmann Publishers, 1999.

[Koz94] J.R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[KW97] A. Kondacs and J.Watrous. On the power of quantum finite state automata. In *IEEE Symposium on Foundations of Computer Science*, pages 66–75, 1997.

[LB04] A. Leier and W. Banzhaf. Comparison of selection strategies for evolutionary quantum circuit design. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 557–568, 2004.

[Lei04] A. Leier. *Evolution of Quantum Algorithms Using Genetic Programming*. PhD thesis, University of Dortmund, 2004.

[LLK+06] S. Lee, S-J. Lee, T. Kim, J-S. Lee, J. Biamonte, and M. Perkowski. The cost of quantum gate primitives. *Journal of Multiple Valued Logic and Soft Computing*, 12(5/6):561–574, 2006.

[LM09] Miller M. Perkowski M. Lukac M., Kameyama M. Evolutionary quantum logic synthesis: Representation vs. micro-parallelism - submitted, 2009.

[LP02] M. Lukac and M. Perkowski. Evolving quantum circuit using genetic algorithm. In *Proceedings of the 2002 NASA/DoD Conference on Evolvable hardware*, pages 177–185, 2002.

[LP05] M. Lukac and M. Perkowski. Combining evolutionary and exhaustive search to find the least expensive quantum circuits. In *Proceedings of ULSI symposium*, 2005.

[LP08] M. Lukac and M. Perkowski. Inductive learning of quantum behaviors. *Facta Universitatis, special issue on Binary and Multiple-Valued Switching Theory and Circuit Design,* 2008.

[LP09] M. Lukac and M. Perkowski. Quantum finite state machines as sequential quantum circuits. In *Proceedings of ISMVL*, 2009.

[LPD95] Shihming Liu, Massoud Pedram, and Alvin M. Despain. A fast state assignment procedure for large fsms. In *DAC '95: Proceedings of the 32nd ACM/IEEE conference on Design automation*, pages 327–332, New York, NY, USA, 1995. ACM.

[LPG+03] M. Lukac, M. Perkowski, H. Goi, M. Pivtoraiko, C. H. Yu, K. Chung, H. Jee, B.-G. Kim, and Y.-D. Kim. Evolutionary approach to quantum reversible circuit synthesis. *Artif. Intell. Review.*, 20(3-4):361–417, 2003.

[LPK09] M. Lukac, M. Perkowski, and M Kameyama. Sequential quantum devices: A circuit-based approach, 2009.

[LPMP02] M. Lukac, M. Pivtoraiko, A. Mishchenko, and M. Perkowski. Automated synthesis of generalized reversible cascades using genetic algorithms. In *Proceedings of Fifth Intern. Workshop on Boolean Problems*, pages 33–45, 2002.

[Luk09] M. Lukac. *Quantum Logic Synthesis and Inductive Machine Learning, Ph.D. dissertation*. PhD thesis, 2009.

[MC00] C. Moore and J.P. Crutchfield. Quantum automata and quantum grammars. *Theoretical Computer Science*, 237:275–306, 2000.

[MC06] Jialin Mi and Chunhong Chen. Finite state machine implementation with single-electron tunneling technology. In *ISVLSI '06: Proceedings of the IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, page 237, Washington, DC, USA, 2006. IEEE Computer Society.

[MCS04] P. Massey, J.A. Clark, and S. Stepney. Evolving quantum circuits and programs through genetic programming. In *Proceedings of the Genetic and Evolutionary Computation conference (GECCO)*, pages 569–580, 2004.

[MCS05] P. Massey, J.A. Clark, and S. Stepney. Evolving of a humancompetitive quantum fourier transform algorithm using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation conference (GECCO)*, pages 1657–1664, 2005.

[Moo65] G.E. Moore. Cramming more components onto integrated circuits. In *Electronics, April 19*, 1965.

[NC00] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[PB99] Arun Kumar Pati and Samuel L. Braunstein. Impossibility of deleting an unknown quantum state, 1999.

[PW02] J. Pachos and H. Walther. Quantum computation with trapped ions in an optical cavity. *Physical Review Letters*, 89(18), 2002.

[RCHCX$^+$08] Y. Rong-Can, L. Hong-Cai, L. Xiu, H. Zhi-Ping, and X. Hong. Implementing a universal quantum cloning machine via adiabatic evolution in ion-trap system. *Communications in Theoretical Physics*, 49(1):80–82, 2008.

[Rub01] B.I.P. Rubinstein. Evolving quantum circuits using genetic programming. In *Congress on Evolutionary Computation (CEC2001)*, pages 114–121, 2001.

[SBS05] R. Stadelhofer,W. Banzhaf, and D. Suter. Quantum and classical parallelism in parity algorithms for ensemble quantum computers. *Physical Review A*, 71, 2005.

[SBS08] R. Stadelhofer, W. Banzhaf, and D. Suter. Evolving blackbox quantum algorithms using genetic programming. *Artif. Intell. Eng. Des. Anal. Manuf.*, 22:285–297, 2008.

[Sch95] H.P Schwefel. *Evolution and Optimum Seeking*. New York, Wiley & Sons, 1995.

[Sho94] P.W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35nd Annual Symposium on Foundations of Computer Science (Shafi Goldwasser, ed.)*, pages 124– 134. IEEE Computer Society Press, 1994.

[SKT04] A. Sakai, Y. Kamakura, and K. Taniguchi. Quantum lattice-gas automata simulation of electronic wave propagation in nanostructures. pages 241–242, Oct. 2004.

[Spe04] L. Spector. *Automatic Quantum Computer Programming: A Genetic Programming Approach*. Kluwer Academic Publishers, 2004.

[Wat95a] J. Watrous. On one-dimensional quantum cellular automata. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 528–537, 1995.

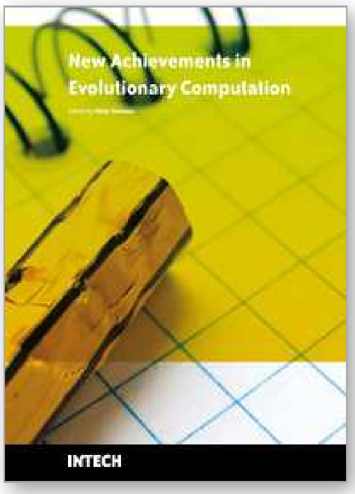
[Wat95b] J. Watrous. On one-dimensional quantum cellular automata. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 528–532, 1995.

[Wat97] J. Watrous. On the power of 2-way quantum finite state automata. Technical Report CS-TR-1997-1350, 1997.

[WG98] C. Williams and A. Gray. Automated design of quantum circuits. In *in Proceedings of QCQC 1998*, pages 113–125, 1998.

[YCS09] A. YakaryIlmaz and A.C. Cem Say. Efficient probability amplification in two-way quantum finite automata. *Theoretical Computer Science*, 410(20):1932 – 1941, 2009. Quantum and Probabilistic Automata.

[YI00] T. Yabuki and H. Iba. Genetic algorithms for quantum circuit design, evolving a simpler teleportation circuit. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 421–425, 2000.



New Achievements in Evolutionary Computation
Edited by Peter Korosec

ISBN 978-953-307-053-7
Hard cover, 318 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

Evolutionary computation has been widely used in computer science for decades. Even though it started as far back as the 1960s with simulated evolution, the subject is still evolving. During this time, new metaheuristic optimization approaches, like evolutionary algorithms, genetic algorithms, swarm intelligence, etc., were being developed and new fields of usage in artificial intelligence, machine learning, combinatorial and numerical optimization, etc., were being explored. However, even with so much work done, novel research into new techniques and new areas of usage is far from over. This book presents some new theoretical as well as practical aspects of evolutionary computation. This book will be of great value to undergraduates, graduate students, researchers in computer science, and anyone else with an interest in learning about the latest developments in evolutionary computation.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821