Three Quantum Algorithms to Solve 3-SAT

Alberto Leporati, Sara Felloni

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
{leporati,sara.felloni}@disco.unimib.it

Summary. We propose three quantum algorithms to solve the 3-SAT NP-complete decision problem. The first algorithm builds, for any instance ϕ of 3-SAT, a quantum Fredkin circuit that computes a superposition of all classical evaluations of ϕ in a given output line. Similarly, the second and third algorithms compute the same superposition on a given register of a quantum register machine, and as the energy of a given membrane in a quantum P system, respectively.

Assuming that a specific non–unitary operator, built using the well known creation and annihilation operators, can be realized as a quantum gate, as an instruction of the quantum register machine, and as a rule of the quantum P system, respectively, we show how to decide whether ϕ is a positive instance of 3-SAT. The construction relies also upon the assumption that an external observer is able to distinguish, as the result of a measurement, between a null and a non–null vector.

1 Introduction

Membrane systems (also called P systems) have been introduced in [31] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. Usually, the result of a computation is the multiset of objects contained into an *output membrane* or emitted from the skin of the system.

In [13] a variant of P systems has been introduced, in which a non–negative integer value is assigned to each membrane. Such value can be conveniently interpreted as the *energy* of the membrane. In such P systems, rules are assigned to the membranes rather than to the regions of the system. Every rule has the form $(in_i : \alpha, \Delta e, \beta)$ or $(out_i : \alpha, \Delta e, \beta)$, where i is the number of the membrane in a

one-to-one labeling, α and β are symbols of the alphabet and Δe is a (possibly negative) integer number. The rule $(in_i : \alpha, \Delta e, \beta)$ is interpreted as follows: if a copy of α is in the region immediately surrounding membrane i, then this object crosses membrane i, is transformed to β, and modifies the energy of membrane i from the current value e_i to the new value $e_i + \Delta e$. Similarly, the rule $(out_i : \alpha, \Delta e, \beta)$ is interpreted as follows: if a copy of α is in the region surrounded by membrane i, then this object crosses membrane i, is transformed to β, and modifies the energy of membrane i from the current value e_i to the new value $e_i + \Delta e$. Both kind of rules can be applied only if $e_i + \Delta e$ is non–negative. Since these rules transform one copy of an object to (one copy of) another object, in [13] they are referred to as *unit* rules. For conciseness, in what follows we will refer to P systems with unit rules and energy assigned to membranes as *UREM P systems*. An important observation is that in [13] the rules of UREM P systems are applied in a *sequential* way: at each computation step, *one* rule is selected from the pool of currently active rules, and it is applied. In [13] it has been proved that if we assign some local (that is, affecting every single membrane) priorities to the rules then UREM P systems are Turing complete. Instead, if we omit the priorities then we do not get systems with universal computational power.

In [21] a *quantum* version of UREM P systems has been introduced, and it has been shown that such model of computation is able to compute every partial recursive function (that is, it reaches the computational power of Turing machines) without the need to assign any priority between the rules of the system. In quantum UREM P systems, the rules $(in_i : \alpha, \Delta e, \beta)$ and $(out_i : \alpha, \Delta e, \beta)$ are realized through (not necessarily unitary) linear operators, which can be expressed as an appropriate composition of creation and annihilation operators. The operators which correspond to the rules have the form $|\beta\rangle \langle\alpha| \otimes O$, where O is a linear operator which modifies the energy associated with the membrane.

In [21] also Quantum Register Machines (QRMs, for short) have been introduced. It has been shown that they are able to simulate any classical (deterministic) register machine, and hence they are (at least) Turing complete.

In this paper we show that, under the assumption that an external observer is able to distinguish between a null vector and a non–null vector, the NP-complete problem 3-SAT can be solved using quantum circuits, quantum register machines and quantum UREM P systems. The solutions are presented in the so-called *semi–uniform* setting, which means that for every instance ϕ of 3-SAT a specific computation device (circuit, register machine or P system) that solves it is built. Even if it is not formally proved, it will be apparent that the proposed constructions can be performed in polynomial time by a classical deterministic Turing machine (whose output is a "reasonable" encoding of the machine, in the sense given in [18]).

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For a systematic introduction, we refer the reader to [32]. The latest information about P systems can be found in [35].

Let us note that this is by no means the first time that energy is considered in P systems: we recall in particular [1, 12, 34, 17, 23, 24, 22].

For a nice introduction to Quantum Computing see [27, 4]. A comprehensive set of achievements in this field is contained in [20]. This is not the first paper in which quantum computers are proposed to solve NP-complete problems; in particular, see [28]. However, in the solution proposed in [28] the probability to observe the correct answer at the end of the computation may decrease exponentially with the number of variables contained into the instance of 3-SAT. To solve this problem, in [29, 30] it has been proposed to use chaotic systems which are able to amplify such probability. However, a drawback of this approach is that the use of chaotic systems brings the computational power of the machinery used to solve 3-SAT beyond the power of Turing machines.

The paper is organized as follows. In Section 2 some preliminaries are given: in particular, we recall some basic notions of Quantum Computing (Subsection 2.1) and the formulation of 3-SAT (Subsection 2.2). In Section 3 we define quantum Fredkin circuits, and we show first how to associate to any instance ϕ of 3-SAT a quantum Fredkin circuit, and then how to extract from it the solution of the problem. In Section 4 we recall Quantum Register Machines (QRMs), and we show how to solve any instance of 3-SAT through an appropriately crafted QRM. In Section 5 we recall quantum UREM P systems, and we show how to solve 3-SAT also with this kind of computational device. The conclusions, as well as some directions for future research, are given in Section 6.

2 Preliminaries

2.1 Quantum Computers

From an abstract point of view, a quantum computer can be considered as made up of interacting parts. The elementary units (memory cells) that compose these parts are two–levels quantum systems called *qubits*. A qubit is typically implemented using the energy levels of a two–levels atom, or the two spin states of a spin–$\frac{1}{2}$ atomic nucleus, or a polarization photon. The mathematical description — independent of the practical realization — of a single qubit is based on the two–dimensional complex Hilbert space $\mathbb{C}^2$. The boolean truth values 0 and 1 are represented in this framework by the unit vectors of the canonical orthonormal basis, called the *computational basis* of $\mathbb{C}^2$:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad\qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Qubits are thus the quantum extension of the classical notion of bit, but whereas bits can only take two different values, 0 and 1, qubits are not confined to their two base (also *pure*) states, $|0\rangle$ and $|1\rangle$, but can also exist in states which are coherent superpositions such as $\psi = c_0 |0\rangle + c_1 |1\rangle$, where c_0 and c_1 are complex

numbers satisfying the condition $|c_0|^2 + |c_1|^2 = 1$. Performing a measurement of the state alters it. Specifically, performing a measurement on a qubit in the above superposition will return 0 with probability $|c_0|^2$ and 1 with probability $|c_1|^2$; the state of the qubit after the measurement (*post–measurement state*) will be $|0\rangle$ or $|1\rangle$, depending on the outcome.

A *quantum register* of size n (also called an *n–register*) is mathematically described by the Hilbert space $\otimes^n\mathbb{C}^2 = \underbrace{\mathbb{C}^2 \otimes \ldots \otimes \mathbb{C}^2}_{n \text{ times}}$, representing a set of n qubits labeled by the index $i \in \{1, \ldots, n\}$. An *n–configuration* (also *pattern*) is a vector $|x_1\rangle \otimes \ldots \otimes |x_n\rangle \in \otimes^n\mathbb{C}^2$, usually written as $|x_1, \ldots, x_n\rangle$, considered as a quantum realization of the boolean tuple $(x_1, \ldots, x_n)$. Let us recall that the dimension of $\otimes^n\mathbb{C}^2$ is 2^n and that $\{|x_1, \ldots, x_n\rangle : x_i \in \{0, 1\}\}$ is an orthonormal basis of this space called the *n*–register *computational basis*.

Computations are usually performed as follows. Each qubit of a given *n*–register is prepared in some particular pure state ($|0\rangle$ or $|1\rangle$) in order to realize the required *n*–configuration $|x_1, \ldots, x_n\rangle$, quantum realization of an input boolean tuple of length n. Then, a linear operator $G : \otimes^n\mathbb{C}^2 \to \otimes^n\mathbb{C}^2$ is applied to the *n*–register. The application of G has the effect of transforming the *n*–configuration $|x_1, \ldots, x_n\rangle$ into a new *n*–configuration $G(|x_1, \ldots, x_n\rangle) = |y_1, \ldots, y_n\rangle$, which is the quantum realization of the output tuple of the computer. We interpret such modification as a computation step performed by the quantum computer. The action of the operator G on a superposition $\Phi = \sum c^{i_1 \ldots i_n} |x_{i_1}, \ldots, x_{i_n}\rangle$, expressed as a linear combination of the elements of the *n*–register basis, is obtained by linearity: $G(\Phi) = \sum c^{i_1 \ldots i_n} G(|x_{i_1}, \ldots, x_{i_n}\rangle)$. We recall that linear operators which act on *n*–registers can be represented as order 2^n square matrices of complex entries. Usually (but not in this paper) such operators, as well as the corresponding matrices, are required to be unitary. In particular, this implies that the implemented operations are logically reversible (an operation is *logically reversible* if its inputs can always be deduced from its outputs).

All these notions can be easily extended to quantum systems which have $d > 2$ pure states. In this setting, the *d*–valued versions of qubits are usually called *qudits* [19]. As it happens with qubits, a qudit is typically implemented using the energy levels of an atom or a nuclear spin. The mathematical description — independent of the practical realization — of a single qudit is based on the *d*–dimensional complex Hilbert space $\mathbb{C}^d$. In particular, the pure states $|0\rangle, \left|\frac{1}{d-1}\right\rangle, \left|\frac{2}{d-1}\right\rangle, \ldots, \left|\frac{d-2}{d-1}\right\rangle, |1\rangle$ are represented by the unit vectors of the canonical orthonormal basis, called the *computational basis* of $\mathbb{C}^d$:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \left|\frac{1}{d-1}\right\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \cdots, \quad \left|\frac{d-2}{d-1}\right\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

As before, a *quantum register* of size n can be defined as a collection of n qudits. It is mathematically described by the Hilbert space $\otimes^n\mathbb{C}^d$. An *n–configuration* is now a vector $|x_1\rangle \otimes \ldots \otimes |x_n\rangle \in \otimes^n\mathbb{C}^d$, simply written as $|x_1, \ldots, x_n\rangle$, for x_i running on $L_d = \left\{0, \frac{1}{d-1}, \frac{2}{d-1}, \ldots, \frac{d-2}{d-1}, 1\right\}$. An n–configuration can be viewed as the quantum realization of the "classical" tuple $(x_1, \ldots, x_n) \in L_d^n$. The dimension of $\otimes^n\mathbb{C}^d$ is d^n and the set $\{|x_1, \ldots, x_n\rangle : x_i \in L_d\}$ of all n–configurations is an orthonormal basis of this space, called the *n–register computational basis*. Notice that the set L_d can also be interpreted as a set of truth values, where 0 denotes falsity, 1 denotes truth and the other elements indicate different degrees of indefiniteness.

Let us now consider the set $\mathcal{E}_d = \left\{\varepsilon_0, \varepsilon_{\frac{1}{d-1}}, \varepsilon_{\frac{2}{d-1}}, \ldots, \varepsilon_{\frac{d-2}{d-1}}, \varepsilon_1\right\} \subseteq \mathbb{R}$ of real values; we can think to such quantities as energy values. To each element $v \in L_d$ (and hence to each object $|v\rangle \in A$) we associate the energy level ε_v; moreover, let us assume that the values of $\mathcal{E}_d$ are all positive, equispaced, and ordered according to the corresponding objects: $0 < \varepsilon_0 < \varepsilon_{\frac{1}{d-1}} < \cdots < \varepsilon_{\frac{d-2}{d-1}} < \varepsilon_1$. If we denote by $\Delta\varepsilon$ the gap between two adjacent energy levels then the following linear relation holds:

$$\varepsilon_k = \varepsilon_0 + \Delta\varepsilon\,(d-1)\,k \qquad\qquad \forall\, k \in L_d \tag{1}$$

Notice that it is not required that $\varepsilon_0 = \Delta\varepsilon$. As explained in [22], the values ε_k can be thought of as the energy eigenvalues of the infinite dimensional quantum harmonic oscillator truncated at the $(d-1)$-th excited level.

To modify the state of a qudit we can use creation and annihilation operators on the Hilbert space $\mathbb{C}^d$, which are defined respectively as:

$$a^\dagger = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & \sqrt{2} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sqrt{d-1} & 0 \end{bmatrix} \qquad a = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{d-1} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

It is easily verified that the action of $a^\dagger$ on the vectors of the canonical orthonormal basis of $\mathbb{C}^d$ is the following:

$$a^\dagger \left|\frac{k}{d-1}\right\rangle = \sqrt{k+1}\left|\frac{k+1}{d-1}\right\rangle \qquad \text{for } k \in \{0, 1, \ldots, d-2\}$$
$$a^\dagger |1\rangle = \mathbf{0}$$

whereas the action of a is:

$$a \left|\frac{k}{d-1}\right\rangle = \sqrt{k}\left|\frac{k-1}{d-1}\right\rangle \qquad \text{for } k \in \{1, 2, \ldots, d-1\}$$
$$a\,|0\rangle = \mathbf{0}$$

The collection of all linear operators on $\mathbb{C}^d$ is a d^2–dimensional linear space whose canonical basis is:

$$\{E_{x,y} = |y\rangle \langle x| \ : \ x, y \in L_d\}$$

Since $E_{x,y} |x\rangle = |y\rangle$ and $E_{x,y} |z\rangle = \mathbf{0}$ for every $z \in L_d$ such that $z \neq x$, this operator transforms the unit vector $|x\rangle$ into the unit vector $|y\rangle$, collapsing all the other vectors of the canonical orthonormal basis of $\mathbb{C}^d$ to the null vector. Each of the operators $E_{x,y}$ can be expressed, using the whole algebraic structure of the associative algebra of operators, as a suitable composition of creation and annihilation operators, as explained in [22].

2.2 The 3-SAT problem

A *boolean* variable is a variable which can assume one of two possible *truth* values: TRUE and FALSE. As usually done in the literature, we will denote TRUE by 1 and FALSE by 0. A *literal* is either a directed or a negated boolean variable. A *clause* is a disjunction of literals. A 3-*clause* is a disjunction of exactly three literals. Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables, an *assignment* is a mapping $a : X \to \{0, 1\}$ that associates to each variable a truth value. The number of all possible assignments to the variables of X is 2^n. We say that an assignment *satisfies* the clause C if, assigned the truth values to all the variables which occur in C and evaluated the formula which represents C, the result is 1.

The 3-SAT decision problem is defined as follows.

Problem 1. NAME: 3-SAT.

- INSTANCE: a set $C = \{c_1, c_2, \ldots, c_m\}$ of 3-clauses, built on a finite set $\{x_1, x_2, \ldots, x_n\}$ of variables;
- QUESTION: is there an assignment of the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C? □

Notice that the number m of possible 3-clauses is polynomially bounded with respect to n: in fact, since each clause contains exactly three literals, we can have at most $(2n)^3 = 8n^3$ clauses.

In what follows we will equivalently say that an instance of 3-SAT is a boolean formula ϕ_n, built on n free variables and expressed in conjunctive normal form, with each clause containing exactly three literals. The formula ϕ_n is thus the conjunction of the above clauses.

It is well known [18] that 3-SAT is an NP-complete problem.

3 Solving 3-SAT with Quantum Circuits

3.1 Quantum Circuits

A *Fredkin gate* is a three–input/three–output boolean gate, whose input/ output map FG : $\{0, 1\}^3 \to \{0, 1\}^3$ associates any input triple (x_1, x_2, x_3) with its corresponding output triple (y_1, y_2, y_3) as follows:

$$\begin{aligned} y_1 &= x_1 \\ y_2 &= (\neg x_1 \wedge x_2) \vee (x_1 \wedge x_3) \\ y_3 &= (x_1 \wedge x_2) \vee (\neg x_1 \wedge x_3) \end{aligned} \tag{2}$$

The Fredkin gate is (logically) *reversible*, since it computes a bijective map on $\{0,1\}^3$. A useful point of view is that the Fredkin gate behaves as a *conditional switch*: that is, $\mathrm{FG}(1, x_2, x_3) = (1, x_3, x_2)$ and $\mathrm{FG}(0, x_2, x_3) = (0, x_2, x_3)$ for every $x_2, x_3 \in \{0,1\}$. Hence, x_1 can be considered as a control input whose value determines whether the input values x_2 and x_3 have to be exchanged or not. The Fredkin gate is also *functionally complete* for boolean logic: by fixing $x_3 = 0$ we get $y_3 = x_1 \wedge x_2$, whereas by fixing $x_2 = 1$ and $x_3 = 0$ we get $y_2 = \neg x_1$.

Putting together Fredkin gates we can build *Fredkin circuits*, that is, acyclic and connected directed graphs made up of *layers* of Fredkin gates. For a precise and formal definition of circuits see, for example, [37]. Figure 1 depicts an example of Fredkin circuit having three gates arranged in two layers. Evaluating a Fredkin

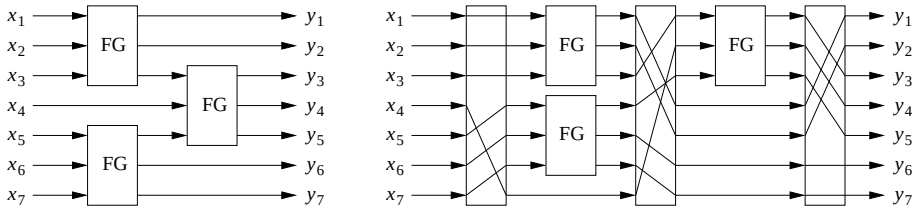

Fig. 1. A Fredkin circuit (on the left) and its normalized version

circuit in topological order (i.e. layer by layer, starting from the layer directly connected to the input lines) we can define the boolean function computed by the circuit as the composition of the functions computed by each layer of Fredkin gates. In evaluating the resources used by a Fredkin circuit to compute a boolean function we consider the *size* and the *depth* of the circuit, respectively defined as the number of gates and the number of layers of the circuit.

Since the Fredkin gate is functionally complete, for any boolean function $f : \{0,1\} \to \{0,1\}$ there exists a Fredkin circuit that computes it in some prefixed output line.

A *reversible* n–input Fredkin circuit is a Fredkin circuit FC_n which computes a bijective map $f_{FC_n} : \{0,1\}^n \to \{0,1\}^n$. In a reversible Fredkin circuit the FanOut function, defined as $\textsc{FanOut}(x) = (x, x)$ for all $x \in \{0,1\}$, is explicitly computed with a gate. Fortunately, the Fredkin gate can also be used for this purpose, since $\textsc{fg}(x, 0, 1) = (x, x, \neg x)$ for $x \in \{0,1\}$. Compare this situation with usual (non reversible) circuits, where the FanOut function is simply implemented by splitting wires. It should be apparent that for any boolean function $f : \{0,1\}^n \to \{0,1\}$ there also exists a m–input reversible Fredkin circuit FC_m (with $m \geq n$) that computes it in some prefixed output line. Without loss of generality, we can assume

that the value of f always appears in the first output line of FC_m. Observe that, in order to compute f through a reversible Fredkin circuit, we could need more that n input/output lines: the additional $m-n$ lines are usually called *ancillae* in the literature.

A quantum version of the Fredkin gate can be represented with the following order 8 ($= 2^3$, where 3 is the number of input and output lines) unitary matrix:

$$U_{\mathrm{FG}} = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

In fact it is easily verified that, for all $x_1, x_2, x_3 \in \{0,1\}$, $U_{\mathrm{FG}} \, |x_1, x_2, x_3\rangle = |y_1, y_2, y_3\rangle$, where $(y_1, y_2, y_3) = \mathrm{FG}(x_1, x_2, x_3)$.

We can also associate an order 2^n unitary matrix to any reversible n–input Fredkin circuit FC_n, as follows. Each layer of FC_n is composed by some Fredkin gates, acting in parallel, and some wires which are not affected by any gate. Since the state of such wires remains unaltered during the computation performed by the layer, we can think that the *identity operator*, or *identity gate*, is applied to them. The unitary matrix associated with the identity gate which acts on a single wire is:

$$\mathrm{ID}_1 = \begin{bmatrix} 1&0 \\ 0&1 \end{bmatrix}$$

It is immediately seen that the unitary matrix associated with an n–input/n–output identity gate is the order 2^n identity matrix ID_n, which can also be expressed as the n–fold tensor product of ID_1:

$$\mathrm{ID}_n = \otimes^n \mathrm{ID}_1$$

Similarly, it is easily seen that the unitary matrix associated with a given layer is obtained by computing the tensor product of the matrices which correspond to the Fredkin gates and to the identity gates which occur from the top to the bottom of the layer. For example, the unitary matrix associated with the first and the second layers of the circuit depicted in the left side of Figure 1 are:

$$U_{\mathrm{FG}} \otimes \mathrm{ID}_1 \otimes U_{\mathrm{FG}} \qquad \text{and} \qquad \mathrm{ID}_2 \otimes U_{\mathrm{FG}} \otimes \mathrm{ID}_2,$$

respectively.

Observe also that we may need to send the output values of a given layer of a Fredkin circuit to *any* input line of the next layer. In other words, we may need to interleave the layers of Fredkin circuits with appropriate fixed permutations, as shown in the right side of Figure 1. If desired, we can also move the Fredkin gates

to any position into the layer (in Figure 1 we can see *normalized* layers, as named in [24, 25], where the Fredkin gates are all moved as upward as possible). Since a fixed permutation $\pi_n : \{0,1\}^n \to \{0,1\}^n$ is a bijection on $\{0,1\}^n$, and hence it is reversible, it can always be represented by an order 2^n unitary matrix U_{π_n}. Indeed U_{π_n} is a *permutation matrix*, having a single 1 in every row and in every column. Let us note in passing that also U_{FG}, the unitary matrix associated with the Fredkin gate, is a permutation matrix. A practical problem which may occur during the construction of the unitary matrices which correspond to fixed permutations is that these matrices may be very large. Indeed, their size grows exponentially with respect to the number n of elements to be permuted. A possible solution is to decompose the permutation into smaller permutations which involve only a small number of not-too-far elements, thus transforming the fixed permutation layer to a permutation (sub)circuit.

Let ℓ be the number of layers of FC_n. We can finally build the unitary matrix U_{FC_n} which corresponds to the entire Fredkin circuit FC_n as the product of the matrices $U_{L_1}, U_{L_2}, \ldots, U_{L_\ell}$ associated with the layers of FC_n, as follows:

$$U_{FC_n} = U_{L_\ell} \cdot \ldots \cdot U_{L_2} \cdot U_{L_1}$$

Now let us show that every matrix U_{FC_n}, which can be obtained as we have just described, can also be represented as a formula in the associative algebra of all linear operators on $\otimes^n \mathbb{C}^2$. The formula consists of the product of the formulas which represent the matrices $U_{L_\ell}, \ldots, U_{L_2}, U_{L_1}$ associated with the layers $L_1, L_2, \ldots, L_\ell$ of FC_n. On its turn, these formulas are obtained as the tensor product of the formulas which represent Fredkin gates and identities. Let us assume that the elementary components which can be used to build the formula which corresponds to U_{FC_n} are the identity (ID_1), the creation ($a^\dagger$) and the annihilation (a) operators on $\mathbb{C}^2$. Moreover, besides the usual ($\cdot$) and the tensor ($\otimes$) products we will also use sums ($+$), which allow us to build linear combinations of operators. As told above, the identity ID_n which acts on n wires can be simply obtained as $\otimes^n \mathrm{ID}_1$. As for the Fredkin gate, we can express it as follows:

$$\begin{aligned} |0\rangle\langle 0| \otimes \mathrm{ID}_1 + |1\rangle\langle 1| \otimes \big(|00\rangle\langle 00| + |11\rangle\langle 11| + |01\rangle\langle 10| + |10\rangle\langle 01| \big) = \\ = aa^\dagger \otimes \mathrm{ID}_1 \otimes \mathrm{ID}_1 + a^\dagger a \otimes \big(aa^\dagger \otimes aa^\dagger + a^\dagger a \otimes a^\dagger a + a \otimes a^\dagger + a^\dagger \otimes a \big) \end{aligned}$$

Hence we can conclude that, given a boolean formula ϕ_n on n free variables, there exists a corresponding formula ψ_m (with $m \geq n$) that describes the structure of the reversible Fredkin circuit FC_m which computes the value of ϕ_n in its first output line, built using only the operators a, $a^\dagger$ and ID_1, and the connectives $+$, $\cdot$ and $\otimes$.

3.2 Solving 3-SAT with Quantum Circuits

Let ϕ_n be an instance of 3-SAT with n free variables. As told above, there exists a reversible Fredkin circuit FC_m (with $m \geq n$) that computes ϕ_n in its first output line. Let U_{FC_m} be the unitary matrix which corresponds to FC_m. Moreover, let:

$$H_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

be the unitary matrix which corresponds to the Hadamard gate, whose effect on the base state $|0\rangle$ of a single qubit is:

$$H_1 |0\rangle = \frac{1}{\sqrt{2}} \big(|0\rangle + |1\rangle \big)$$

A well known technique in Quantum Computing is to use the m–fold tensor product of H_1:

$$H_m = \otimes^m H_1 = \frac{1}{\sqrt{2^m}} \otimes^m \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

whose effect on the base state $|0 \cdots 0\rangle$ of the computational basis of $\otimes^m \mathbb{C}^2$ is:

$$H_m |0 \cdots 0\rangle = \otimes^m H_1 |0\rangle = \otimes^m \frac{1}{\sqrt{2}} \big(|0\rangle + |1\rangle \big) =$$
$$= \frac{1}{\sqrt{2^m}} \sum_{x_1,\ldots,x_m \in \{0,1\}} |x_1, \ldots, x_m\rangle$$

to create a *uniform* superposition (that is, a linear combination whose coefficients are all the same) of all the base states of the computational basis of $\otimes^m \mathbb{C}^2$. It is also well known that if we apply the linear operator represented by U_{FC_m} to such superposition we obtain as a result a linear combination of all possible "classical" results in the output lines. In particular, in the first output line of FC_m we will obtain one of two possible results:

- $|0\rangle$, if ϕ_n is not satisfiable;
- a linear combination $\alpha_0 |0\rangle + \alpha_1 |1\rangle$, with $\alpha_1 \neq 0$, if ϕ_n is satisfiable. The quantity $|\alpha_1|$ will be directly proportional to the number of assignments which satisfy ϕ_n, and thus it could be *exponentially small* with respect to α_0 (we recall that $|\alpha_0|^2 + |\alpha_1|^2 = 1$).

Now, the problem is that if we measure the state of the first output line then it collapses to a classical state in a random way, and the probability to observe the post–measurement state $|i\rangle$, with $i \in \{0, 1\}$, is $|\alpha_i|^2$. This means that, even if ϕ_n is satisfiable, in the worst case we should make an exponential number of computations and successive measurements to obtain a $|1\rangle$ in the first output line of FC_m. This problem also affected the solution of SAT through quantum circuits exposed in [28]. To solve this problem, it has been subsequently proposed to amplify $|\alpha_1|$ (and thus the probability to observe $|1\rangle$) by feeding a "chaotic machine" with the output generated by the quantum circuit [29]. A drawback of such solution is that it puts ourselves beyond the computational power of Turing machines, because the chaotic system used in [29] has super–Turing capabilities. Here we note that if we are able to build a gate whose linear operator O is represented by the following (non–unitary) matrix:

$$2^n \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} = 2^n a^\dagger a = 2^n |1\rangle \langle 1| \tag{3}$$

then also the following "selection" operator can be built:

$$O^{(m)} = O \otimes \left(\otimes^{m-1} \mathrm{ID}_1 \right)$$

which applies O to the value of the first output line of the circuit, and the identity operator to the other lines.

Hence the global operator which describes the computation performed by FC_m on all possible classical inputs, and the subsequent query on the first output line, is:

$$O^{(m)} \cdot U_{FC_m} \cdot H_m |0 \cdots 0\rangle$$

If we observe the resulting vector, we have two possible outcomes:

- the *null* vector $\mathbf{0}$, if ϕ_n is not satisfiable. This is due to the fact that:

$$O|0\rangle = 2^n |1\rangle \langle 1|0\rangle = \mathbf{0}$$

- a *non–null* vector if ϕ_n is satisfiable, since

$$\begin{aligned} O\big(\alpha_0 |0\rangle + \alpha_1 |1\rangle\big) &= \alpha_0 2^n |1\rangle \langle 1|0\rangle + \alpha_1 2^n |1\rangle \langle 1|1\rangle \\ &= \mathbf{0} + \alpha_1 2^n |1\rangle = \alpha_1 2^n |1\rangle \end{aligned}$$

  Let us note here that the coefficient 2^n has been chosen so that the length of the resulting vector is not too small; this should help to distinguish it from the null vector.

We can thus conclude that if both the following conditions hold:

1. it is possible to apply the operator $2^n |1\rangle \langle 1|$ to the first output line of the quantum version of the Fredkin circuit FC_m,
2. an external observer is able to distinguish, as the result of a measurement, between a null and a non–null vector,

then we have a quantum computational device which is able to solve the NP-complete problem 3-SAT. Let us note that this computational device is built in a semi–uniform way: the structure (topology) of the quantum circuit FC_m depends upon the instance ϕ_n of 3-SAT we want to solve.

4 Solving 3-SAT with QRMs

4.1 Quantum Register Machines

A (classical, deterministic) *n–register machine* is a construct $M = (n, P, l_0, l_h)$, where n is the number of registers, P is a finite set of instructions injectively labeled with a given set $lab(M)$, l_0 is the label of the first instruction to be executed, and l_h is the label of the last instruction of P. Registers contain non–negative integer values. Without loss of generality, we can assume $lab(M) = \{1, 2, \ldots, m\}$, $l_0 = 1$ and $l_h = m$. The instructions of P have the following forms:

- $j : (INC(r), k)$, with $j, k \in lab(M)$
  This instruction increments the value contained in register r, and then jumps to instruction k.
- $j : (DEC(r), k, l)$, with $j, k, l \in lab(M)$
  If the value contained in register r is positive then decrement it and jump to instruction k. If the value of r is zero then jump to instruction l (without altering the contents of the register).
- $m : Halt$
  Stop the machine. Note that this instruction can only be assigned to the final label m.

Register machines provide a simple universal computational model. Indeed, the results proved in [14] (based on the results established in [26]) as well as in [15] and [16] immediately lead to the following proposition.

Proposition 1. *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$–register machine M computing f in such a way that, when starting with $(n_1, \ldots, n_\alpha) \in \mathbb{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label l_h with registers 1 to β containing r_1 to r_β, and all other registers being empty; if the final label cannot be reached, then $f(n_1, \ldots, n_\alpha)$ remains undefined.*

A *quantum n–register machine* is defined exactly as in the classical case, as a four–tuple $M = (n, P, l_0, l_h)$. Each register of the machine is an infinite dimensional quantum harmonic oscillator, capable to assume the base states $|\varepsilon_0\rangle, |\varepsilon_1\rangle, |\varepsilon_2\rangle, \ldots$, corresponding to its energy levels. The program counter of the machine is instead realized through a quantum system capable to assume m different base states, from the set $\{|x\rangle : x \in L_m\}$. For simplicity, the instructions of P are denoted in the usual way:

$$j : (INC(i), k) \qquad \text{and} \qquad j : (DEC(i), k, l)$$

This time, however, these instructions are appropriate linear operators acting on the Hilbert space whose vectors describe the (global) state of M. Precisely, the instruction $j : (INC(r), k)$ is defined as the operator

$$O^{INC}_{j,r,k} = |p_k\rangle \langle p_j| \otimes \left(\otimes^{r-1}\mathbb{I}\right) \otimes a^\dagger \otimes \left(\otimes^{n-r}\mathbb{I}\right)$$

with $\mathbb{I}$ the identity operator on $\mathcal{H}$, whereas the instruction $j : (DEC(r), k, l)$ is defined as the operator

$$\begin{aligned} O^{DEC}_{j,r,k,l} = {} & |p_l\rangle \langle p_j| \otimes \left(\otimes^{r-1}\mathbb{I}\right) \otimes |\varepsilon_0\rangle \langle \varepsilon_0| \otimes \left(\otimes^{n-r}\mathbb{I}\right) + \\ & |p_k\rangle \langle p_j| \otimes \left(\otimes^{r-1}\mathbb{I}\right) \otimes a \otimes \left(\otimes^{n-r}\mathbb{I}\right) \end{aligned}$$

Hence the program P can be formally defined as the sum O_P of all these operators:

$$O_P = \sum_{j,r,k} O^{INC}_{j,r,k} + \sum_{j,r,k,l} O^{DEC}_{j,r,k,l}$$

Thus O_P is the global operator which describes a computation step of M. The Halt instruction is simply executed by doing nothing when the program counter assumes the value $|p_m\rangle$. For such value, O_P would produce the null vector as a result; however, in the following we will add a term to O_P that allows us to extract the solution of the problem from a prefixed register when the program counter assumes the value $|p_m\rangle$.

A *configuration* of M is given by the value of the program counter and the values contained in the registers. From a mathematical point of view, a configuration of M is a vector of the Hilbert space $\mathbb{C}^m \otimes (\otimes^n \mathcal{H})$, where $\mathcal{H}$ is the Hilbert space associated with every quantum harmonic oscillator. A transition between two configurations is obtained by executing one instruction of P (the one pointed at by the program counter), that is, by applying the operator O_P to the current configuration of M.

As shown in [21], QRMs can simulate any (classical, deterministic) register machine, and thus they are computationally complete.

4.2 Solving 3-SAT with Quantum Register Machines

Let ϕ_n be an instance of 3-SAT containing n free variables. We will first show how to evaluate ϕ_n with a classical register machine; then, we will use the same trick we have adopted with quantum circuits: we will initialize the input registers with a superposition of all possible assignments, we will compute the corresponding superposition of output values into an output register, and finally we will apply the linear operator $2^n |1\rangle \langle 1|$ to the output register to check whether ϕ_n is a positive instance of 3-SAT.

The register machine that we use to evaluate ϕ_n is composed by $n+1$ registers. The first n registers correspond (in a one-to-one manner) to the free variables of ϕ_n, while the last register is used to compute the output value. The *structure* of the program used to evaluate ϕ_n is the following:

$\phi = 0$
`if` $C_1 = 0$ `then goto` end
`if` $C_2 = 0$ `then goto` end
$\vdots$
`if` $C_m = 0$ `then goto` end
$\phi = 1$
end:

where ϕ denotes the output register, and $C_1, C_2, \ldots, C_m$ are the clauses of ϕ_n. Let $X_{i,j}$, with $j \in \{1, 2, 3\}$, be the literals (directed or negated variables) which occur in the clause C_i (hence $C_i = X_{i,1} \vee X_{i,2} \vee X_{i,3}$). We can thus write the above structure of the program, at a finer grain, as follows:

$\phi = 0$
`if` $X_{1,1} = 1$ `then goto` end_1

$$
\begin{array}{rl}
 & \texttt{if } X_{1,2} = 1 \texttt{ then goto } \mathrm{end}_1 \\
 & \texttt{if } X_{1,3} = 1 \texttt{ then goto } \mathrm{end}_1 \\
 & \texttt{goto end} \\
\mathrm{end}_1\texttt{:} & \texttt{if } X_{2,1} = 1 \texttt{ then goto } \mathrm{end}_2 \\
 & \texttt{if } X_{2,2} = 1 \texttt{ then goto } \mathrm{end}_2 \\
 & \texttt{if } X_{2,3} = 1 \texttt{ then goto } \mathrm{end}_2 \\
 & \texttt{goto end} \\
\mathrm{end}_2\texttt{:} & \cdots\cdots \\
 & \vdots \\
\mathrm{end}_{m-1}\texttt{:} & \texttt{if } X_{m,1} = 1 \texttt{ then goto end} \\
 & \texttt{if } X_{m,2} = 1 \texttt{ then goto end} \\
 & \texttt{if } X_{m,3} = 1 \texttt{ then goto end} \\
 & \phi = 1 \\
\texttt{end:} &
\end{array}
\tag{4}
$$

In the above structure it is assumed that each literal $X_{i,j}$, with $1 \leq i \leq m$ and $j \in \{1, 2, 3\}$, is substituted with the corresponding variable which occurs in it; moreover, if the variable occurs negated into the literal then the comparison must be done with 0 instead of 1:

$$\texttt{if } X_{i,j} = 0 \texttt{ then goto } \mathrm{end}_i$$

Since the free variables of ϕ_n are bijectively associated with the first n registers of the machine, in order to evaluate ϕ_n we need a method to check whether a given register contains 0 (or 1) without destroying its value. Let us assume that, when the program counter of the machine reaches the value k, we have to execute the following instruction:

$$k\texttt{: if } X_{i,j} = 1 \texttt{ then goto } \mathrm{end}_i$$

We translate such instruction as follows (where, instead of $X_{i,j}$, we specify the register which corresponds to the variable indicated in $X_{i,j}$):

$$
\begin{array}{rl}
k\texttt{:} & DEC(X_{i,j}), k+1, k+2 \\
k+1\texttt{:} & INC(X_{i,j}), \mathrm{end}_i
\end{array}
$$

The instruction:

$$k\texttt{: if } X_{i,j} = 0 \texttt{ then goto } \mathrm{end}_i$$

is instead translated as follows:

$$
\begin{array}{rl}
k\texttt{:} & DEC(X_{i,j}), k+1, \mathrm{end}_i \\
k+1\texttt{:} & INC(X_{i,j}), k+2
\end{array}
$$

Notice that the only difference with the above sequence of instructions is in the position of "end_i" and "$k+2$". Moreover, the structure of the program is always the same. As a consequence, given an instance ϕ_n of 3-SAT, the program P of a register machine which evaluates ϕ_n can be obtained in a very straightforward (mechanical) way. For example, if:

$$\phi_4 = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee x_4 \vee x_3)$$

then the following program P can be immediately obtained (here we assume that the output register ϕ has already been initialized with 0):

```
 1: DEC(1), 2, 3          // if x_1 = 1 then goto end_1
 2: INC(1), end_1
 3: DEC(2), 4, 5          // if x_2 = 1 then goto end_1
 4: INC(2), end_1
 5: DEC(3), 6, end_1      // if x_3 = 0 then goto end_1
 6: INC(3), end           //              else  goto end
 7: DEC(1), 8, 9          // if x_1 = 1 then goto end
 8: INC(1), end
 9: DEC(4), 10, 11        // if x_4 = 1 then goto end
10: INC(4), end
11: DEC(3), 12, 13        // if x_3 = 1 then goto end
12: INC(3), end
13: INC(φ), 14            // φ = 1
14: HALT
```

where $\text{end}_1 = 7$, $\text{end} = 14$ and $\phi = 5$.

On a *classical* register machine, this program computes the value of ϕ_n for a given assignment to its variables $x_1, x_2, \ldots, x_n$. On a *quantum* register machine we can initialize the registers with the following state:

$$\otimes^{n-1} H_1 \left|0\right\rangle \otimes \left|0\right\rangle$$

which sets the output register ϕ to 0 and the registers corresponding to $x_1, x_2, \ldots, x_n$ to a superposition of all possible assignments. Then, we apply the global operator O_P which corresponds to the program P until the program counter reaches the value $|p_{\text{end}}\rangle$, thus computing in the output register a superposition of all classical results. The operator O_P is built as described above, with the only difference that now it contains also the term:

$$\left|p_{\text{end}}\right\rangle \left\langle p_{\text{end}}\right| \otimes \text{ID}_n \otimes 2^n \left|1\right\rangle \left\langle 1\right|$$

which extracts the result from the output register when the program counter assumes the value $|p_{\text{end}}\rangle$. The number of times we have to apply O_P is equal to the length of P, that is, $2 \cdot 3m + 2 = 6m + 2$: two instructions for each literal in every clause, plus 2 final instructions.

Now, if ϕ_n is not satisfiable then the contents of the output register is $|0\rangle$, and when the program counter reaches the value $|p_{\text{end}}\rangle$ the operator O_P transforms it to the null vector. On the other end, if ϕ_n is satisfiable then the contents of the output register will be a superposition $\alpha_0 \left|0\right\rangle + \alpha_1 \left|1\right\rangle$, with $|\alpha_0|^2 + |\alpha_1|^2 = 1$ and $\alpha_1 \neq 0$. By applying the operator O_P we obtain (here $|\psi_n\rangle$ denotes the state of the n input registers):

$$
\begin{aligned}
&O_P\big(\,|p_{\text{end}}\rangle \otimes |\psi_n\rangle \otimes (\alpha_0\,|0\rangle + \alpha_1\,|1\rangle)\big) = \\
&\quad = \big(\,|p_{\text{end}}\rangle\,\langle p_{\text{end}}| \otimes \text{ID}_n \otimes 2^n\,|1\rangle\,\langle 1|\,\big)\cdot \\
&\qquad\qquad \cdot\big(\,|p_{\text{end}}\rangle \otimes |\psi_n\rangle \otimes (\alpha_0\,|0\rangle + \alpha_1\,|1\rangle)\big) = \\
&\quad = |p_{\text{end}}\rangle\,\langle p_{\text{end}}|p_{\text{end}}\rangle \otimes \text{ID}_n\,|\psi_n\rangle \otimes 2^n\,|1\rangle\,\langle 1|\,(\alpha_0\,|0\rangle + \alpha_1\,|1\rangle) = \\
&\quad = |p_{\text{end}}\rangle \otimes |\psi_n\rangle \otimes (2^n\alpha_0\,|1\rangle\,\langle 1|0\rangle + 2^n\alpha_1\,|1\rangle\,\langle 1|1\rangle) = \\
&\quad = |p_{\text{end}}\rangle \otimes |\psi_n\rangle \otimes (\mathbf{0} + 2^n\alpha_1\,|1\rangle) = \\
&\quad = |p_{\text{end}}\rangle \otimes |\psi_n\rangle \otimes 2^n\alpha_1\,|1\rangle
\end{aligned}
$$

that is, a non–null vector.

We can thus conclude that if an external observer is able to distinguish between a null and a non–null vector, then we have a quantum algorithm that allows to solve 3-SAT on QRMs. Just like the solution proposed for quantum Fredkin circuits, this algorithm works in a *semi–uniform* setting: in particular, the program P executed by the QRM depends upon the instance ϕ_n of 3-SAT we want to solve.

5 Solving 3-SAT with Quantum UREM P Systems

5.1 Quantum UREM P Systems

A UREM P system [13] of degree $d+1$ is a construct Π of the form:

$$\Pi = (A, \mu, e_0, \ldots, e_d, w_0, \ldots, w_d, R_0, \ldots, R_d)$$

where:

- A is an alphabet of *objects*;
- μ is a *membrane structure*, with the membranes labelled by numbers $0, \ldots, d$ in a one-to-one manner;
- $e_0, \ldots, e_d$ are the initial energy values assigned to the membranes $0, \ldots, d$. In what follows we assume that $e_0, \ldots, e_d$ are non–negative integers;
- $w_0, \ldots, w_d$ are multisets over A associated with the regions $0, \ldots, d$ of μ;
- $R_0, \ldots, R_d$ are finite sets of *unit rules* associated with the membranes $0, \ldots, d$. Each rule has the form $(\alpha : a, \Delta e, b)$, where $\alpha \in \{in, out\}$, $a, b \in A$, and $|\Delta e|$ is the amount of energy that — for $\Delta e \geq 0$ — is added to or — for $\Delta e < 0$ — is subtracted from e_i (the energy assigned to membrane i) by the application of the rule.

By writing $(\alpha_i : a, \Delta e, b)$ instead of $(\alpha : a, \Delta e, b) \in R_i$, we can specify only one set of rules R with

$$R = \{(\alpha_i : a, \Delta e, b) \;:\; (\alpha : a, \Delta e, b) \in R_i, 0 \leq i \leq d\}$$

The *initial configuration* of Π consists of $e_0, \ldots, e_d$ and $w_0, \ldots, w_d$. The transition from a configuration to another one is performed by non-deterministically

choosing one rule from some R_i and applying it (observe that here we consider a *sequential* model of applying the rules instead of choosing rules in a maximally parallel way, as it is often required in P systems). Applying $(in_i : a, \Delta e, b)$ means that an object a (being in the membrane immediately outside of i) is changed into b while entering membrane i, thereby changing the energy value e_i of membrane i by Δe. On the other hand, the application of a rule $(out_i : a, \Delta e, b)$ changes object a into b while leaving membrane i, and changes the energy value e_i by Δe. The rules can be applied only if the amount e_i of energy assigned to membrane i fulfills the requirement $e_i + \Delta e \geq 0$. Moreover, we use some sort of local priorities: if there are two or more applicable rules in membrane i, then one of the rules with $\max |\Delta e|$ has to be used.

A sequence of transitions is called a *computation*; it is *successful* if and only if it halts. The *result* of a successful computation is considered to be the distribution of energies among the membranes (a non–halting computation does not produce a result). If we consider the energy distribution of the membrane structure as the input to be analysed, we obtain a model for accepting sets of (vectors of) non–negative integers.

The following result, proved in [13], establishes computational completeness for this model of P systems.

Proposition 2. *Every partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ can be computed by a UREM P system with (at most)* $\max\{\alpha, \beta\} + 3$ *membranes.*

On the other hand, by omitting the priority feature we do not get systems with universal computational power. Precisely, in [13] it is proved that UREM P systems without priorities and with an arbitrary number of membranes characterize the family $PsMAT^\lambda$ of Parikh sets generated by context–free matrix grammars (without occurrence checking and with λ-rules).

In *quantum* UREM P systems, all the elements of the model (multisets, the membrane hierarchy, configurations, and computations) are defined just like the corresponding elements of the classical P system, but for objects and rules. The objects of A are represented as pure states of a quantum system. If the alphabet contains $d \geq 2$ elements, then without loss of generality we can put $A = \left\{ |0\rangle, \left|\frac{1}{d-1}\right\rangle, \left|\frac{2}{d-1}\right\rangle, \ldots, \left|\frac{d-2}{d-1}\right\rangle, |1\rangle \right\}$, that is, $A = \{|a\rangle \ : \ a \in L_d\}$. As stated above, the quantum system will also be able to assume as a state any superposition of the kind:

$$c_0 |0\rangle + c_{\frac{1}{d-1}} \left|\frac{1}{d-1}\right\rangle + \ldots + c_{\frac{d-2}{d-1}} \left|\frac{d-2}{d-1}\right\rangle + c_1 |1\rangle$$

with $c_0, c_{\frac{1}{d-1}}, \ldots, c_{\frac{d-2}{d-1}}, c_1 \in \mathbb{C}$ such that $\sum_{i=0}^{d-1} \left| c_{\frac{i}{d-1}} \right|^2 = 1$. A multiset is simply a collection of quantum systems, each in its own state.

In order to represent the energy values assigned to membranes we should use quantum systems which can exist in an infinite (countable) number of states. Hence we should assume that every membrane of the quantum P system has an associated

infinite dimensional quantum harmonic oscillator whose state represents the energy value assigned to the membrane. To modify the state of such harmonic oscillator we should use the infinite dimensional version of the creation ($a^\dagger$) and annihilation (a) operators described above, which are commonly used in quantum mechanics. The actions of $a^\dagger$ and a on the state of an infinite dimensional harmonic oscillator are analogous to the actions on the states of truncated harmonic oscillators; the only difference is that in the former case there is no state with maximum energy, and hence the creation operator never produces the null vector. However, as we will see, in this paper we do not require to store an unlimited amount of energy into the harmonic oscillators; on the contrary, we will need to put them only in the base states $|\varepsilon_0\rangle$ and $|\varepsilon_1\rangle$, as well as in superpositions of such states.

As in the classical case, rules are associated to membranes rather than to the regions enclosed by them. Each rule of R_i is an operator of the form

$$|y\rangle \langle x| \otimes O, \qquad \text{with } x, y \in L_d \tag{5}$$

where O is a linear operator which can be expressed by an appropriate composition of operators $a^\dagger$ and a. The part $|y\rangle \langle x|$ is the *guard* of the rule: it makes the rule "active" (that is, the rule produces an effect) if and only if a quantum system in the basis state $|x\rangle$ is present. The semantics of rule (5) is the following: If an object in state $|x\rangle$ is present in the region immediately outside membrane i, then the state of the object is changed to $|y\rangle$ and the operator O is applied to the state of the harmonic oscillator associated with the membrane. Notice that the application of O can result in the null vector, so that the rule has no effect even if its guard is satisfied; this fact is equivalent to the condition $e_i + \Delta e \geq 0$ on the energy of membrane i required in the classical case. Differently from the classical case, no local priorities are assigned to the rules. If two or more rules are associated to membrane i, then they are summed. This means that, indeed, we can think to each membrane as having only one rule with many guards. When an object is present, the inactive parts of the rule (those for which the guard is not satisfied) produce the null vector as a result. If the region in which the object occurs contains two or more membranes, then all their rules are applied to the object. Observe that the object which activates the rules never crosses the membranes. This means that the objects specified in the initial configuration can change their state but never move to a different region. Notwithstanding, transmission of information between different membranes is possible, since different objects may modify in different ways the energy state of the harmonic oscillators associated with the membranes.

The application of one or more rules determines a *transition* between two configurations. A *halting configuration* is a configuration in which no rule can be applied. A sequence of transitions is a *computation*. A computation is *successful* if and only if it *halts*, that is, reaches a halting configuration. The *result* of a successful computation is considered to be the distribution of energies among the membranes in the halting configuration. A non–halting computation does not produce a result. Just like in the classical case, if we consider the energy distribution

of the membrane structure as the input to be analyzed, we obtain a model for accepting sets of (vectors of) non–negative integers.

In [21] it has been shown that quantum UREM P systems are able to simulate any QRM, and hence they are (at least) Turing complete.

5.2 Solving 3-SAT with Quantum UREM P Systems

Let ϕ_n be an instance of 3-SAT containing n free variables. The structure and the initial configuration of the P system are shown in Figure 2. As we have done with

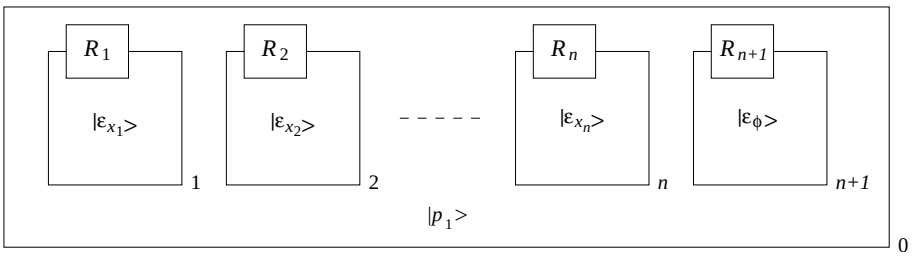

Fig. 2. Structure and initial configuration of the quantum UREM P system used to solve 3-SAT

quantum circuits and with quantum register machines, let us start by showing how to evaluate ϕ_n for a given assignment of truth values to its variables $x_1, \ldots, x_n$. The input values are set as the energies $|\varepsilon_{x_i}\rangle$ of the harmonic oscillators associated with the membranes from 1 to n. The energy (eventually) associated with the skin membrane is not used. The $(n+1)$-th membrane, whose harmonic oscillator will contain the output at the end of the computation, is initialized with $|\varepsilon_0\rangle$. The alphabet A consists of all the possible values which can be assumed by the program counter. In the initial configuration the P system contains only one object $|p_1\rangle$, corresponding to the initial value of the program counter, in the region enclosed by the skin membrane (see Figure 2).

The evaluation of ϕ_n could be performed by simulating the QRM obtained from ϕ_n as explained in the previous section. However, we can obtain a slightly more efficient P system as follows. We start from the program structure (4), which can be obtained from ϕ_n in a straightforward way. Now, let us suppose we must execute the following instruction:

$$k\text{: } \texttt{if } X_{i,j} = 1 \texttt{ then goto } \text{end}_i$$

As told above, this instruction is performed as follows in a register machine:

$$\begin{aligned} k\text{: } & DEC(X_{i,j}), k+1, k+2 \\ k+1\text{: } & INC(X_{i,j}), \text{end}_i \end{aligned}$$

If we had to simulate these two instructions using a quantum UREM P system, we should use the following sum of rules:

$$\underbrace{\left(|p_{\mathrm{end}_i}\rangle \langle p_{k+1}| \otimes a^\dagger\right)}_{k+1\colon INC(X_{i,j}),\ \mathrm{end}_i} + \underbrace{\left(|p_{k+2}\rangle \langle p_k| \otimes |\varepsilon_0\rangle \langle \varepsilon_0| + |p_{k+1}\rangle \langle p_k| \otimes a\right)}_{k\colon DEC(X_{i,j}),\, k+1,\, k+2} \in R_\ell$$

where $\ell = \langle i, j\rangle$ is the index of the variable (in the set $\{x_1, x_2, \ldots, x_n\}$) which occurs in literal $X_{i,j}$. As we can see, this operator produces the vector $|p_{k+2}\rangle \otimes |\varepsilon_0\rangle$ if the harmonic oscillator of membrane ℓ is in state $|\varepsilon_0\rangle$; otherwise, it produces the vector $|p_{\mathrm{end}_i}\rangle \otimes |\varepsilon_1\rangle$. Hence we can simplify the above expression as follows:

$$\begin{aligned} |p_{\mathrm{end}_i}\rangle \langle p_k| \otimes |\varepsilon_1\rangle \langle \varepsilon_1| + |p_{k+2}\rangle \langle p_k| \otimes |\varepsilon_0\rangle \langle \varepsilon_0| &= \\ = |p_{\mathrm{end}_i}\rangle \langle p_k| \otimes a^\dagger a + |p_{k+2}\rangle \langle p_k| \otimes a a^\dagger & \end{aligned}$$

We denote this operator by $O^{(1)}_{i,j,k}$. Analogously, if the instruction to be executed is:

$$k\colon \texttt{if } X_{i,j} = 0 \texttt{ then goto } \mathrm{end}_i$$

we use the operator

$$O^{(0)}_{i,j,k} = |p_{\mathrm{end}_i}\rangle \langle p_k| \otimes a a^\dagger + |p_{k+2}\rangle \langle p_k| \otimes a^\dagger a \in R_\ell$$

which produces the vector $|p_{k+2}\rangle \otimes |\varepsilon_1\rangle$ if the harmonic oscillator of membrane ℓ is in state $|\varepsilon_1\rangle$, otherwise it produces the vector $|p_{\mathrm{end}_i}\rangle \otimes |\varepsilon_0\rangle$.

Since the value $|p_{k+1}\rangle$ is no longer used, we can "compact" the program by redefining the operators $O^{(0)}_{i,j,k}$ and $O^{(1)}_{i,j,k}$ respectively as:

$$\begin{aligned} O^{(0)}_{i,j,k} &= |p_{\mathrm{end}_i}\rangle \langle p_k| \otimes a a^\dagger + |p_{k+1}\rangle \langle p_k| \otimes a^\dagger a \\ O^{(1)}_{i,j,k} &= |p_{\mathrm{end}_i}\rangle \langle p_k| \otimes a^\dagger a + |p_{k+1}\rangle \langle p_k| \otimes a a^\dagger \end{aligned}$$

The "`goto` end" instructions in (4) can be executed as if they were `if` statements whose condition is the negation of the condition given in the previous `if`. Hence the two instructions:

$$\begin{aligned} &7\colon \texttt{if } X_{2,3} = 1 \texttt{ then goto } \mathrm{end}_2 \\ &8\colon \texttt{goto end} \end{aligned}$$

can be thought of as:

$$\begin{aligned} &7\colon \texttt{if } X_{2,3} = 1 \texttt{ then goto } \mathrm{end}_2 \\ &8\colon \texttt{if } X_{2,3} = 0 \texttt{ then goto end} \end{aligned}$$

which are realized by the operators $O^{(1)}_{2,3,7}$ and $O^{(0)}_{2,3,8}$ (to be added to membrane $\langle 2, 3\rangle$). The last instruction ($\phi = 1$) of the program can be implemented as follows:

$$|p_{\mathrm{end}}\rangle \langle p_{\mathrm{end}-1}| \otimes a^\dagger$$

to be added to membrane $n+1$.

For each membrane $i \in \{1, 2, \ldots, n\}$, the set of rules R_i is obtained by summing all the operators which concern variable x_i.

Note that the formulation given in terms of quantum P systems is simpler than the one obtained with QRMs. As usual, if we consider a single assignment to the variables of ϕ_n then at the end of the computation we will obtain the result of the evaluation of ϕ_n as the energy of the output membrane. Instead, if we initialize the harmonic oscillators of the n input membranes with a uniform superposition of all possible classical assignments to $x_1, x_2, \ldots, x_n$, then at the end of the computation the harmonic oscillator of membrane $n+1$ will be in one of the following states:

- $|0\rangle$, if ϕ_n is not satisfiable;
- a superposition $\alpha_0 |0\rangle + \alpha_1 |1\rangle$, with $|\alpha_0|^2 + |\alpha_1|^2 = 1$ and $\alpha_1 \neq 0$, if ϕ_n is satisfiable.

Once again, we add the rule:

$$|p_{\text{end}}\rangle \langle p_{\text{end}}| \otimes 2^n |1\rangle \langle 1| \in R_{n+1}$$

to membrane $n+1$ to extract the result.

We have thus obtained a quantum membrane algorithm which solves 3-SAT. As with the solutions proposed for quantum circuits and QRMs, also this algorithm works in the *semi–uniform* setting: in fact, it is immediately verified that the rules of the system depend upon the instance ϕ_n of 3-SAT to be solved.

6 Conclusions and Directions for Future Research

In this paper we have proposed three quantum algorithms that solve (in the semi–uniform setting) the 3-SAT NP-complete decision problem. Their construction relies upon the assumption that an external observer is able to distinguish, as the result of a measurement, between a null and a non–null vector.

The first algorithm builds, for any instance ϕ_n of 3-SAT, a quantum Fredkin circuit that computes a superposition of all classical evaluations of ϕ_n in the first output line. Similarly, the second and third algorithms compute the same superposition on a given register of a quantum register machine, and as the energy of a given membrane in a quantum P system, respectively. Assuming that a given non–unitary operator, which can be expressed using the well known creation and annihilation operators, can be realized as a quantum gate, as an instruction of the quantum register machine, and as a rule of the quantum P system, respectively, we can apply the operator to the result of the above computation in order to extract the solution of 3-SAT for the instance ϕ_n given in input.

One possible direction for future research is to study the computational properties of quantum P systems which contain and process entangled objects. Another line of research is to study the limits of the computational power of quantum P

systems by attacking harder than NP-complete problems. In particular, we conjecture that EXP-complete problems can be solved in polynomial time with quantum P systems.

References

1. G. Alford: Membrane systems with heat control. In *Pre–Proceedings of the Workshop on Membrane Computing (WMC-CdeA2002)*, Curtea de Argeş, Romania, August 2002.
2. A. Alhazov, R. Freund, A. Leporati, M. Oswald, C. Zandron: (Tissue) P systems with unit rules and energy assigned to membranes. To appear in *Fundamenta Informaticae*.
3. A. Barenco, D. Deutsch, A. Ekert, R. Jozsa. Conditional quantum control and logic gates. *Physical Review Letters*, 74 (1995), 4083–4086.
4. G. Benenti, G. Casati, G. Strini: *Principles of Quantum Computation and Information – Volume I: Basic Concepts*. World Scientific, 2004.
5. P. Benioff: Quantum mechanical Hamiltonian models of discrete processes. *Journal of Mathematical Physics*, 22 (1981), 495–507.
6. P. Benioff. Quantum mechanical Hamiltonian models of computers. *Annals of the New York Academy of Science*, 480 (1986), 475–486.
7. D. Deutsch: Quantum theory, the Church–Turing principle, and the universal quantum computer. *Proceedings of the Royal Society of London*, A 400 (1985), 97–117.
8. R.P. Feynman: Simulating physics with computers. *International Journal of Theoretical Physics*, 21, 6–7 (1982), 467–488.
9. R.P. Feynman: Quantum mechanical computers. *Optics News*, 11 (1985), 11–20.
10. E. Fredkin, T. Toffoli: Conservative logic. *International Journal of Theoretical Physics*, 21, 3-4 (1982), 219–253.
11. R. Freund: Sequential P systems. *Romanian Journal of Information Science and Technology*, 4, 1–2 (2001), 77–88.
12. R. Freund: Energy-controlled P systems. In *Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August 2002* (Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron, eds.), LNCS 2597, Springer, 2003, 247–260.
13. R. Freund, A. Leporati, M. Oswald, C. Zandron: Sequential P systems with unit rules and energy assigned to membranes. In *Proceedings of Machines, Computations and Universality, MCU 2004*, Saint–Petersburg, Russia, September 21–24, 2004, LNCS 3354, Springer, 2005, 200–210.
14. R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, 49, 1–3 (2002), 81–102.
15. R. Freund, Gh. Păun: On the number of non-terminals in graph-controlled, programmed, and matrix grammars. In *Proc. Conf. Universal Machines and Computations* (M. Margenstern, Y. Rogozhin, eds.), Chişinău (2001), LNCS 2055, Springer, 2001, 214–225.
16. R. Freund, Gh. Păun: From regulated rewriting to computing with membranes: Collapsing hierarchies. *Theoretical Computer Science*, 312 (2004), 143–188.
17. P. Frisco: The conformon–P system: a molecular and cell biology–inspired computability model. *Theoretical Computer Science*, 312 (2004), 295–319.

18. M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the Theory on NP–Completeness*. W.H. Freeman and Company, 1979.
19. D. Gottesman: Fault-tolerant quantum computation with higher-dimensional systems. *Chaos, Solitons, and Fractals*, 10 (1999), 1749–1758.
20. J. Gruska: *Quantum Computing*. McGraw–Hill, 1999.
21. A. Leporati, G. Mauri, C. Zandron: Quantum sequential P systems with unit rules and energy assigned to membranes. In *Membrane Computing: 6^{th} International Workshop, WMC 2005* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Vienna, Austria, July 18–21, 2005, LNCS 3850, Springer, 2006, 310–325.
22. A. Leporati, D. Pescini, C. Zandron: Quantum energy–based P systems. In *Proceedings of the First Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma de Mallorca, Spain, November 8–10, 2004, 145–167.
23. A. Leporati, C. Zandron, G. Mauri: Simulating the Fredkin gate with energy–based P systems. *Journal of Universal Computer Science*, 10, 5 (2004), 600–619. A preliminary version is contained in [33], 292–308.
24. A. Leporati, C. Zandron, G. Mauri: Universal families of reversible P systems. In *Proceedings of Machines, Computations and Universality, MCU 2004*, Saint–Petersburg, Russia, September 21–24, 2004, LNCS 3354, Springer, 2005, 257–268.
25. A. Leporati, C. Zandron, G. Mauri: Reversible P systems to simulate Fredkin circuits. To appear in *Fundamenta Informaticae*.
26. M.L. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
27. M.A. Nielsen, I.L. Chuang: *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.
28. M. Ohya, N. Masuda: NP problem in quantum algorithm. *Open Systems & Information Dynamics*, 7, 1 (2000), 33–39. A preliminary version appears in `http://arxiv.org/abs/quant-ph/9809075`.
29. M. Ohya, I.V. Volovich: Quantum computing, NP-complete problems and chaotic dynamics. In *Quantum Information* (T. Hita, K. Saito, eds.), World Scientific, 2000, 161–171. A preliminary version appears in `http://arxiv.org/abs/quant-ph/9912100`.
30. M. Ohya, I.V. Volovich: Quantum computing and the chaotic amplifier. *Journal of Optics B: Quantum and Semiclassical Optics*, 5 (2003), S639–S642.
31. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143. See also Turku Centre for Computer Science – TUCS Report No. 208, 1998.
32. Gh. Păun: *Membrane Computing. An Introduction*. Springer-, Berlin, 2002.
33. Gh. Păun, A. Riscos Nuñez, A. Romero Jiménez, F. Sancho Caparrini, eds.: *Second Brainstorming Week on Membrane Computing*, Seville, Spain, February 2–7, 2004. Department of Computer Sciences and Artificial Intelligence, University of Seville TR 01/2004.
34. Gh. Păun, Y. Suzuki, H. Tanaka: P systems with energy accounting. *International Journal Computer Math.*, 78, 3 (2001), 343–364.
35. The P Systems Web Page: `http://psystems.disco.unimib.it/`
36. T. Toffoli: *Reversible Computing*, MIT/LCS Technical Report 151, February 1980.
37. H. Vollmer: *Introduction to Circuit Complexity: A Uniform Approach*. Springer, 1999.