Simulations of Quantum Finite Automata

Gustaw Lippa, Krzysztof Makieła, and Marcin Kuta(✉)

Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Krakow, Poland
mkuta@agh.edu.pl

Abstract. This paper presents a Python library to simulate different kinds of quantum finite automata on a classical computer. The library also provides tools for language generation and visual representation of simulation results. We have conducted experiments to measure the time complexity of the simulation in a function of the automaton size, alphabet size and word length. Examples of library usage are also provided.

Keywords: Quantum finite automata · Acceptance conditions · Cut-point · Automata simulation

1 Introduction

Finite automata are real models of computers, which have only limited amount of memory. Finite automata are also interesting models themselves, due to their simplicity but at the same time rich structure and interesting properties [7]. Since the introduction of finite automata in 1959 by Rabin and Scott [13], the theme has been quite well recognized and approached from various aspects. Connections with different domains, including algebra and logics, have also been established.

The picture for theory of quantum automata and languages generated by them is less clear, and various important problems remain open [2,11]. The task of quantum finite automata is to recognise quantum languages. Studying these languages is useful in establishing the computational and expressive power of quantum machines in general. However, such devices are not yet available and simulators have to be used instead. Because of that, we developed a library written in Python, running on a classical computer and providing implementation of several types of quantum finite automata.

This paper presents the library for simulating quantum finite automata. The library can help in exploring hypotheses on unknown relations between classes of quantum finite automata and quantum languages by providing evidence about accepting probabilities of particular words and sets of words. The library could also be useful for teaching students courses on finite automata in a quantum context.

2 Related Work

Simulation of classical finite automata is a mature area. A comprehensive survey of simulators of classical finite automata is given in [5] and JFLAP emerges as the most mature and popular tool [14].

On the other hand, there have been many libraries focused on bringing quantum computation onto the classical architectures, perhaps the best known being Q#[1]. The low-level libraries include Quirk[2], with graphical interface available through a web browser, or Quantum++[3], a high performance library written in C++11. The high-level libraries provide similar functionalities but focus more on code expressiveness. They often enable using real life simulators or quantum computers as their back-ends. Examples of such libraries include ProjectQ[4], Qiskit[5] and the aforementioned Q#.

However, we have not found a solution focused solely on quantum finite automata. The available pieces of software were either too low-level, focusing on quantum gates and quantum phenomena in micro-scale, or too abstract, providing interfaces for developing quantum algorithms in general, but without tools dedicated specifically for quantum automata.

3 Quantum Finite Automata

In this work we consider only one-way finite automata, i.e., in each step of a simulation the head reads one symbol from the tape and moves forward. Backward or empty moves of the head are forbidden. The paper defines all automata in a uniform framework, which means that for classical finite automata notation differs slightly from the one adopted widely in literature.

Preliminaries. An input alphabet Σ is a finite set of symbols. The working alphabet Γ equals $\Sigma \cup \{\$\}$, where \$ denotes a special end-marker symbol outside the input alphabet. Set Q is a finite set of states and $q_I \in Q$ is a distinguished state, called the initial state. Each classical state $q \in Q$ has a quantum counterpart $|q\rangle$. A pure quantum state $|\psi\rangle$ of quantum automaton is defined as

$$|\psi\rangle = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \sum_{i=1}^{n} \alpha_i |q_i\rangle \ , \tag{1}$$

where $\alpha_1, \ldots, \alpha_n \in \mathbb{C}$ and $\sum_{i=1}^{n} |\alpha_i|^2 = 1$.

Vector $\mathbf{1}$ denotes column vector consisting of $|Q|$ ones. Matrix $I_{|Q|}$ denotes square $|Q| \times |Q|$ matrix containing ones on a diagonal and zeros elsewhere.

[1] https://docs.microsoft.com/en-us/quantum/language.
[2] https://algassert.com/quirk.
[3] https://github.com/vsoftco/qpp.
[4] https://github.com/ProjectQ-Framework/ProjectQ.
[5] https://qiskit.org.

Definition 1. Nondeterministic Finite Automaton (NFA) [13] is a 5-tuple $\mathcal{A} = (\Sigma, Q, q_I, Q_{\text{acc}}, \{M_\sigma\}_{\sigma\in\Sigma})$, where $Q_{\text{acc}} \subseteq Q$ is a set of accepting states, and for all $\sigma \in \Sigma$ transition matrix M_σ satisfies $M_\sigma \in \{0,1\}^{|Q|\times|Q|}$. If transition matrices M_σ additionally satisfy for all $\sigma \in \Sigma$ condition

$$M_\sigma \mathbf{1} = \mathbf{1} \,, \tag{2}$$

then an automaton is Deterministic Finite Automaton (DFA). Classes of NFAs and DFAs are equivalent as they recognize the same class of languages, i.e., class of regular languages.

Definition 2. Probabilistic Finite Automaton (PFA) [12] is a 5-tuple $\mathcal{A} = (Q, \Sigma, I, F, \{M_\sigma\}_{\sigma\in\Sigma})$, where vector I is a stochastic column vector describing initial distribution of states, i.e., $I \in [0,1]^{|Q|\times 1}$ and $\sum_{i=1}^{|Q|} I_i = 1$. Vector F is a column vector of size $|Q|$ with i-th entry equal 1 if q_i is a accepting state and 0 otherwise. For all $\sigma \in \Sigma$ transition matrix M_σ is Markovian, i.e., its rows define probability distribution. Thus, for all $\sigma \in \Sigma$ we have $M_\sigma \in [0,1]^{|Q|\times|Q|}$ and M_σ satisfies (2).

Definition 3. Measure-Once Quantum Finite Automaton (MO-QFA) [9] is a 5-tuple $\mathcal{A} = (Q, \Sigma, q_I, Q_{\text{acc}}, \{U_\sigma\}_{\sigma\in\Sigma})$, where $Q_{\text{acc}} \subseteq Q$ is a set of accepting states. Transition matrices $\{U_\sigma\}_{\sigma\in\Sigma}$ satisfy $U_\sigma \in \mathbb{C}^{|Q|\times|Q|}$ for all $\sigma \in \Sigma$ and are unitary, i.e., for all $\sigma \in \Sigma$ we have

$$U_\sigma^\dagger U_\sigma = U_\sigma U_\sigma^\dagger = I_{|Q|} \ . \tag{3}$$

The set of accepting states corresponds to a projective operator:

$$P_{\text{acc}} = \sum_{q\in Q_{\text{acc}}} |q\rangle\langle q| \ . \tag{4}$$

Definition 4. Measure-Many Quantum Finite Automaton (MM-QFA) [8] is a 6-tuple $\mathcal{A} = (Q, \Sigma, q_I, Q_{\text{acc}}, Q_{\text{rej}}, \{U_\sigma\}_{\sigma\in\Gamma})$, where $Q_{\text{rej}} \subseteq Q$ is a set of rejecting states, and $U_\sigma \in \mathbb{C}^{|Q|\times|Q|}$ are transition matrices satisfying (3).

The automaton partitions set Q into $Q = Q_{\text{acc}} \cup Q_{\text{rej}} \cup Q_{\text{non}}$, where Q_{non} is a set of nonhalting (neutral) states. Sets $Q_{\text{acc}}, Q_{\text{rej}}$ and Q_{non} should be pairwise disjoint.

In a manner analogous to (4), projective operators P_{rej} and P_{non} are defined as follows:

$$P_{\text{rej}} = \sum_{q\in Q_{\text{rej}}} |q\rangle\langle q| \ , \tag{5}$$

$$P_{\text{non}} = \sum_{q\in Q_{\text{non}}} |q\rangle\langle q| \ . \tag{6}$$

Definition 5. General Quantum Finite Automaton (GQFA) [10] is a 6-tuple $\mathcal{A} = (Q, \Sigma, q_I, Q_{\text{acc}}, Q_{\text{rej}}, \{U_\sigma\}_{\sigma\in\Gamma})$. The model is similar to MM-QFA, but the transition matrices U_σ are more general – they are a composition of a finite sequence of applications of unitary transformations followed by orthogonal measurements. Note, that there is a more general definition of GQFA, provided by Hirvensalo [6], but it is not considered in our work.

For a broader description of these and other types of quantum automata we refer a reader to [3,7,11].

3.1 Language Acceptance Modes

Let $P_{\mathcal{A}}(x)$ denote probability of accepting word $x \in \Sigma^*$ by automaton $\mathcal{A}$ and let λ be a real number such that $\lambda \in [0, 1)$. Given $x = \sigma_1 \dots \sigma_n$, probability $P_{\mathcal{A}}(x)$ is computed as $\|P_{\text{acc}} U_{\sigma_n} \dots U_{\sigma_1} |q_0\rangle\|^2$ for MO-QFA, but for MM-QFA is more complicated.

There exist several modes of language acceptance:

- *with a cut-point* $\lambda \in [0, 1)$, if for all $x \in L$, we have $P_{\mathcal{A}}(x) > \lambda$ and for all $x \notin L$, we have $P_{\mathcal{A}}(x) \leq \lambda$. This mode of acceptance is also called *with an unbounded error.*
- *with an isolated cut-point* $\lambda \in [0, 1)$, if there exists $\varepsilon \geq 0$, such, that for all $x \in L$, we have $P_{\mathcal{A}}(x) \geq \lambda + \varepsilon$ and for all $x \notin L$, we have $P_{\mathcal{A}}(x) \leq \lambda - \varepsilon$.
- *with a bounded error* $\varepsilon \in [0, \frac{1}{2})$, if for all $x \in L$, we have $P_{\mathcal{A}}(x) \geq 1 - \varepsilon$ and for all $x \notin L$, we have $P_{\mathcal{A}}(x) \leq \varepsilon$. This mode of acceptance is equivalent to acceptance with an isolated cut-point, where cut-point $\lambda = \frac{1}{2}$ is isolated with value $\frac{1}{2} - \varepsilon$.
- *with a positive one-sided unbounded error* if for all $x \in L$, we have $P_{\mathcal{A}}(x) > 0$.
- *with a negative one-sided unbounded error* if for all $x \in L$, we have $P_{\mathcal{A}}(x) = 1$.
- *Monte Carlo acceptance* [4], if there exists $\varepsilon \in (0, \frac{1}{2}]$ such, that for all $x \in L$, we have $P_{\mathcal{A}}(x) = 1$ and for all $x \notin L$, we have $P_{\mathcal{A}}(x) \leq \varepsilon$. Such $\mathcal{A}$ is called Monte Carlo *QFA* for L.

Acceptance with a cut-point and acceptance with an isolated cut-point are the most import modes of language acceptance, and are implemented in our library.

4 Library

The simulation library[6] provides its interface through a number of classes, each representing one of the automata models. It also uses modules `LanguageGenerator`, `LanguageChecker` and `Plotter`. Figure 1 presents main components of the simulation library, which offers the following functionality.

Automaton Definition. A new automaton is constructed with an object belonging to a class representing implemented automaton (one of `PFA`, `MO_1QFA`, `MM_1QFA`, `GQFA`) and data defining chosen automaton, such as the alphabet, transition matrices and matrices of projective measurements are passed. A user has to assure unitarity of transition matrices.

[6] https://github.com/gustawlippa/QFA.

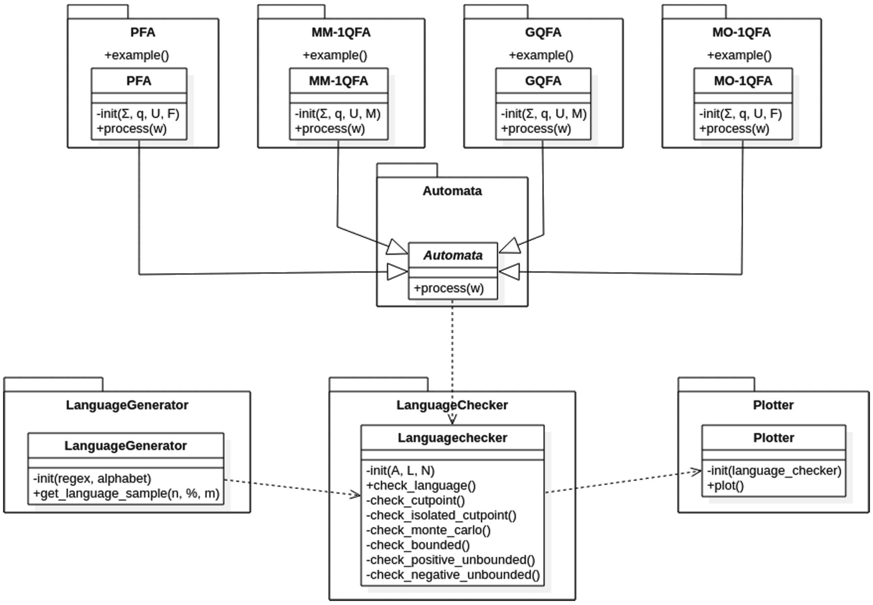


Fig. 1. Model of the quantum finite automata library

Generation of Language Samples. The `LanguageGenerator` module is responsible for this functionality. For finite languages, a user can provide a list of words which entirely define the language. To account for the infinite regular languages, a language can be defined with a regular expression. Generation of samples of stochastic languages is not implemented in the current version of the library.

Automaton Simulation. The `LanguageChecker` module of the library enables simulation of an automaton run on a single word or on a random or user-defined sample of defined language. A simulation is determined by a transition matrix and one simulation of the automaton on a given word is sufficient to obtain all information about word acceptance or rejection. Thus, there is no need to repeat a simulation since transition matrices are stationary. Simulation results are returned with respect to two modes of language acceptance: with a cut-point and with an isolated cut-point.

Results Visualisation. The library has a dedicated `Plotter` module to plot histograms of counts of words accepted and words rejected with a given probability. A cut-point and an isolation interval can also be shown.

5 Simulations and Results

There exist two ways of simulating systems of a probabilistic nature. One way is the *strong* simulation, which requires calculating the exact probability of an outcome. The other way is the *weak* simulation, which is based on a sampling from the output distribution in order to approximate the probability. The latter is the only tractable way of simulating quantum computers because of the complexity of the task. However, our library performs a strong simulation, which is adequate because of the simpler nature of quantum automata.

5.1 Experiments

We have performed a handful of experiments to determine the time complexity of the simulations depending on various parameters.

Automaton size. During our research we concluded that the most important factor in the time complexity of a simulation is the number of automaton states. In a true QFA, the relation between the number of states and computation time would be linear. In our simulator the time complexity is polynomial which agrees with the result proven in [3]. Figure 2 shows simulation time of GQFA with the growing number of its states. The results are quite close to the values predicted theoretically. Simulation time is reported as the arithmetic mean from 5 simulations for each automaton size.

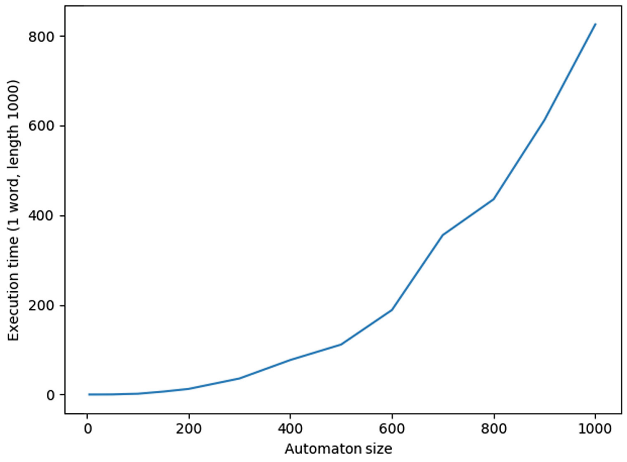
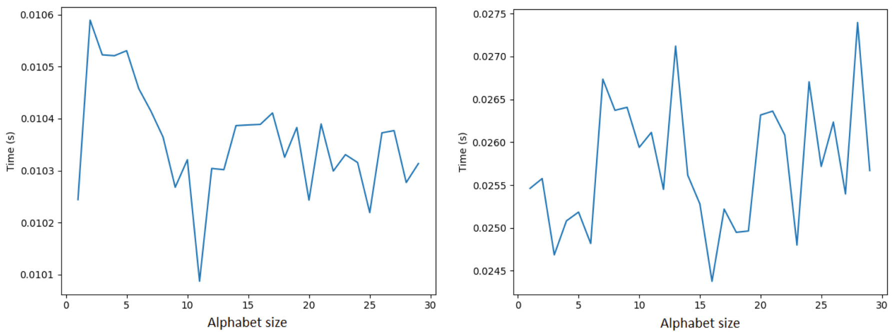


Fig. 2. Simulation time of GQFA in a function of the number of states

Alphabet Size. Figure 3 presents simulation time of GQFA as a function of the size of alphabet, over which automaton is defined. For each alphabet size, simulation time was measured as the arithmetic mean over 500 random words. It should be noted that the scale in ordinate axis (y-axis) does not start from 0. Figure 3 shows that there is no dependence between the size of the alphabet and the time of simulation. This is because the change of the alphabet size influences only the amount of input data (transition matrices and projective measurement matrices) required to define an automaton but does not impact computation time at all.



(a) 20 states (b) 30 states

Fig. 3. Simulation time of GQFA in a function of the alphabet size

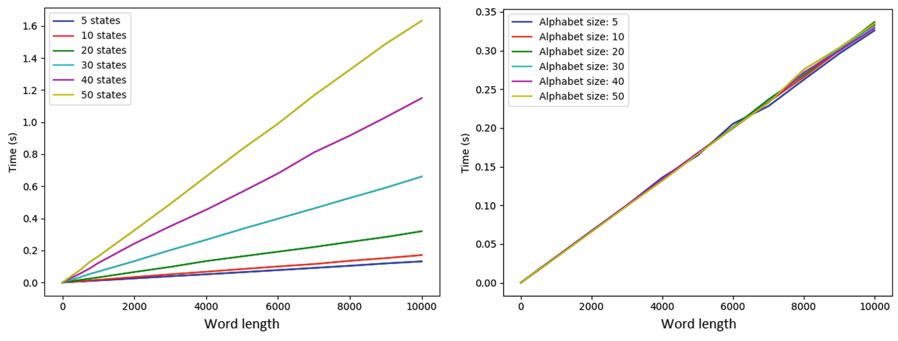
Word Length. The relation between the time of computation and the length of an input word is linear. This is not surprising, because with each letter of the input word, an automaton performs a fixed number of matrix multiplications. The size of these matrices is determined by the number of states of automaton. For each letter a transition function must be applied and a measurement may be performed, depending on the type of an automaton. Figure 4 shows simulation time of GQFA as a function of the length of simulated words. For each word length, 100 random words were simulated and simulation time was taken as the arithmetic mean over these 100 words.

5.2 Usage Example

Listing 1.1 presents exemplary code for defining a MM-QFA automaton. This automaton is well-known in literature and was proposed by Ambainis and Freivalds [1] as a part of the proof that there exists a one-way QFA, which recognizes language a^*b^* with probability $p = 0.68..$, where p is the real root of the equation $p^3 + p = 1$. Figure 5 visualizes obtained acceptance probabilities and their corresponding word counts.

```
import numpy as np
from math import sqrt
from QFA.MM_1QFA import MM_1QFA
from QFA.LanguageGenerator import LanguageGenerator
from QFA.LanguageChecker import LanguageChecker
from QFA.Plotter import Plotter

alphabet = 'ab'

p = 0.682327803828019  # Auxillary variable

# Initial state of automaton
initial_state = np.array([[sqrt(1-p)], [sqrt(p)], [0], [0]])

# Transition matrices
U_a = np.array([[1-p,            sqrt(p*(1-p)), 0, -sqrt(p) ],
                [sqrt(p*(1-p)), p,              0, sqrt(1-p)],
                [0,              0,              1, 0        ],
                [sqrt(p),        -sqrt(1-p),     0, 0        ]])

U_b = np.array([[0, 0, 0, 1],
                [0, 1, 0, 0],
                [0, 0, 1, 0],
                [1, 0, 0, 0]])

U_end = np.array([[0, 0, 0, 1],
                  [0, 0, 1, 0],
                  [0, 1, 0, 0],
                  [1, 0, 0, 0]])

# Accepting and rejecting states are defined with matrices
# representing projective measurements
P_acc = np.array([[0, 0, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 1, 0],
                  [0, 0, 0, 0]])

P_rej = np.array([[0, 0, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 0, 1]])

qfa = MM_1QFA(alphabet, initial_state,
              [U_a, U_b, U_end], P_acc, P_rej)

language_generator = LanguageGenerator('a*b*', alphabet)
language, not_in_language = language_generator.get_language_sample()

language_checker = LanguageChecker(qfa, language, not_in_language)

plotter = Plotter(language_checker)
plotter.plot()
```

Listing 1.1. Code for defining MM-QFA automaton



(a) 5-letter alphabet (b) 20 states of automaton

Fig. 4. Simulation time of GQFA in a function of the length of simulated words

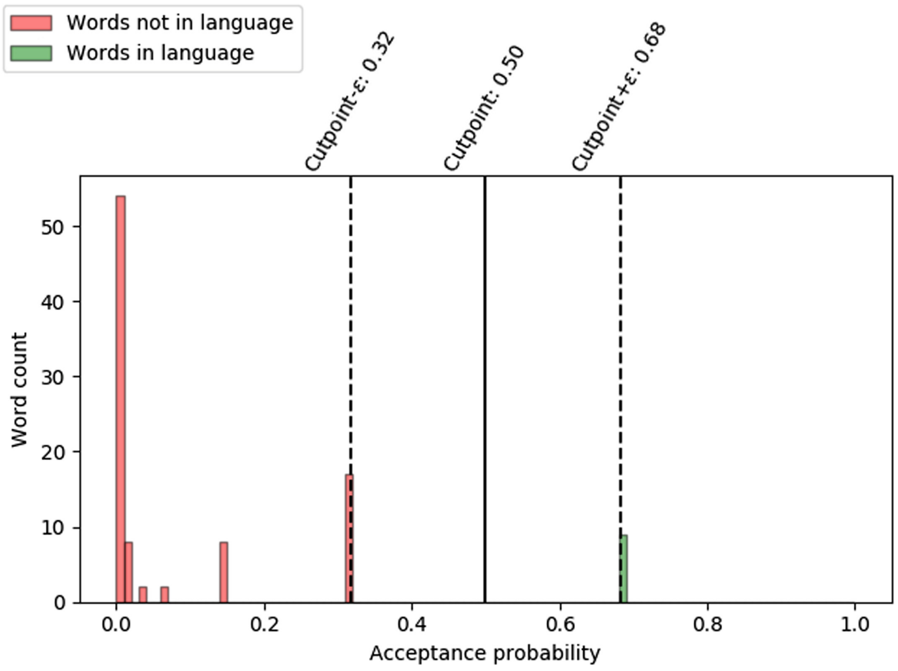


Fig. 5. Visualization of acceptance probabilities and acceptance modes for MM-QFA

6 Conclusions

The library provides a simple API for functionality required to simulate quantum finite automata. We hope the library will encourage research at the intersection of quantum computations and theory of formal languages and automata. We have experimentally shown that the time complexity of simulating a quantum finite automaton is polynomial in relation to the size of the automaton. Nevertheless, we believe that the library may be useful for researchers, lecturers and students as a tool to prove or disprove certain properties of quantum automata and languages in a reasonable time. We have successfully used our library and thus shown that it returns expected results for examples taken from literature.

The scope of the project can be broadened in several directions, e.g. by adding new types of automata.

Acknowledgments. The research presented in this paper was supported by the funds assigned to AGH University of Science and Technology by the Polish Ministry of Science and Higher Education.

References

1. Ambainis, A., Freivalds, R.: 1-way quantum finite automata: strengths, weaknesses and generalizations. In: 39th Annual Symposium on Foundations of Computer Science, FOCS 1998, pp. 332–341 (1998). https://doi.org/10.1109/SFCS.1998.743469
2. Ambainis, A., Kikusts, A., Valdats, M.: On the class of languages recognizableby 1-way quantum finite automata. In: Ferreira, A., Reichel, H. (eds.) 18th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2001, pp. 75–86 (2001)
3. Ambainis, A., Yakaryılmaz, A.: Automata and quantum computing (2015). CoRR abs/1507.01988
4. Bianchi, M.P., Mereghetti, C., Palano, B.: Quantum finite automata: advances on bertoni's ideas. Theor. Comput. Sci. **664**, 39–53 (2017)
5. Chakraborty, P., Saxena, P.C., Katti, C.P.: Fifty years of automata simulation: a review. ACM Inroads **2**(4), 59–70 (2011). https://doi.org/10.1145/2038876.2038893
6. Hirvensalo, M.: Quantum automata with open time evolution. IJNCR **1**(1), 70–85 (2010). https://doi.org/10.4018/jncr.2010010104
7. Hirvensalo, M.: Quantum automata theory – a review. In: Kuich, W., Rahonis, G. (eds.) Algebraic Foundations in Computer Science. LNCS, vol. 7020, pp. 146–167. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24897-9_7
8. Kondacs, A., Watrous, J.: On the power of quantum finite state automata. In: 38th Annual Symposium on Foundations of Computer Science, FOCS 1997, Miami Beach, Florida, USA, 19–22 October 1997, pp. 66–75. IEEE Computer Society (1997)
9. Moore, C., Crutchfield, J.P.: Quantum automata and quantum grammars. Theor. Comput. Sci. **237**(1–2), 275–306 (2000)
10. Nayak, A.: Optimal lower bounds for quantum automata and random access codes. In: 40th Annual Symposium on Foundations of Computer Science, FOCS 1999, pp. 369–377 (1999)
11. Qiu, D., Li, L., Mateus, P., Gruska, J.: Quantum finite automata. In: Wang, J. (ed.) Handbook of Finite State Based Models and Applications, pp. 113–144 (2012)
12. Rabin, M.O.: Probabilistic automata. Inf. Control **6**(3), 230–245 (1963)
13. Rabin, M.O., Scott, D.S.: Finite automata and their decision problems. IBM J. Res. Dev. **3**(2), 114–125 (1959). https://doi.org/10.1147/rd.32.0114
14. Rodger, S.H., Finley, T.W.: JFLAP: An Interactive Formal Languages and Automata Package. Jones and Bartlett Publishers, Sudbury (2006)