TWO APPLICATIONS OF PARALLEL FINITE STATE MACHINE EXECUTION

Alexandru AGACHE[1]

In this paper we present two results regarding finite state machines (also referred to as FSMs). They involve the concept of parallel finite state machine execution, which proves essential in outlining both findings. The first result concerns FSM equivalence with incomplete knowledge, where we determine the amount of information required about two black box automata that can be run on different inputs. The second leads to a method of testing whether a variable-length code is uniquely decodable.

Keywords: automata theory, finite state machine equivalence, product automaton, unique decodability

1. Preliminaries

In this section we briefly present the most import notations and definitions that are used throughout the paper. While many are widely encountered in automata theory, they are still summarized here for the reader's convenience.

An *alphabet* Σ is a non-empty finite set; its members are called *symbols*. A *word* (or *string*) w over Σ is any finite sequence of symbols from Σ. The *length* of w is equal to the number of symbols in w, and is denoted by $|w|$. Two strings u and v can be *concatenated* (joined into a single sequence with u followed by v) using the $\cdot$ operator. A common shorthand notation for $u \cdot v$ is uv.

It's easy to see that $|uv| = |u| + |v|$. There is one special word, called the *empty string*, which is denoted by e, and has length $|e| = 0$. The emptry string is also the identity element of concatenation: $w \cdot e = e \cdot w = w$, for any word w. The set of all strings of length n over Σ is denoted by Σ^n. It's worth nothing that $\Sigma^0 = \{e\}$ for any alphabet Σ.

The set of all possible strings over Σ is called the *Kleene closure* of Σ, and is denoted by Σ^*. It can be written as $\Sigma^* = \cup_{i \in \mathbb{N}} \Sigma^i$. A *formal language* (or simply *language*) over Σ is any subset of Σ^*. Concatenation is also defined for languages: $L_1 \cdot L_2 = \{uv \,|\, u \in L_1 \text{ and } v \in L_2\}$. Applying the Kleene closure leads to $L^* = \cup_{i \in \mathbb{N}} L^i$.

A *finite state machine* is a mathematical model of computation. We use this term in the restricted sense of *language acceptor*: a finite state machine M computes

[1]PhD Student, Department of Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: alexandru.agache@cs.pub.ro

a function $f : \Sigma^* \to \{0, 1\}$. A word w is accepted by M if and only if $f(w) = 1$. We also say that M accepts the language $L(M) = \{w \in \Sigma^* | f(w) = 1\}$. A *deterministic* FSM is represented by a quintuple $(\Sigma, S, s, \delta, F)$. Σ is the alphabet over which input strings are defined, S is a non-empty set of states, $s \in S$ is the *initial state*, $\delta : S \times \Sigma \to S$ is the *transition function*, and $F \subseteq S$ is the set of *final* (or *accepting*) states. Unless otherwise specified, δ is a total function.

The execution of M for input $w = a_0 \dots a_{n-1}$ ($a_i \in \Sigma$) ends in state $s_n = \delta(s_{n-1}, a_{n-1})$, where $s_0 = s$. If $s_n \in F$ the input is accepted; otherwise the input is rejected. The sequence of states $P_M(w) = s_0 \dots s_n$ is called the *execution path* (or simply *path*) of w in M. Two paths are distinct if they are represented by different sequences. Multiple words can have the same execution path. A state q is *reachable* if at least one input generates a path ending with q. We only consider automata where every state is reachable, because unreachable states can be safely discarded.

The representation of a *non-deterministic* FSM is also possible with the quintuple $(\Sigma, S, s, \delta, F)$, but with one significant difference: the transition function is defined as $\delta : S \times \Sigma \to \mathcal{P}(S)$. The execution of a non-deterministic FSM may lead to multiple paths, so $P_M(w)$ becomes a set for any string w. There is a branch each time multiple states are obtained from the application of δ. The input is accepted if at least one path ends in a final state.

A well known result in automata theory [1] states that for any non-deterministic FSM, there is an equivalent deterministic FSM. Two finite state machines, M_1 and M_2, are *equivalent* if they accept the same language. We denote this by $M_1 \equiv M_2$.

Definition 1.1. *Consider two finite state machines M_1 and M_2, that receive inputs over the same alphabet Σ. A third FSM $M_3 = (\Sigma, S_3, s_3, \delta_3, F_3)$, called the* product automaton*, can be built in the following manner:*

- $S_3 = S_1 \times S_2$
- $s_3 = (s_1, s_2)$
- $\delta_3((p, q), x) = (\delta_1(p, x), \delta_2(q, x))$*, if M_1 and M_2 are deterministic*
- $\delta_3((p, q), x) = \delta_1(p, x) \times \delta_2(q, x)$*, if M_1 and M_2 are non-deterministic*

We say that M_3 executes M_1 and M_2 in parallel: any path in M_3 actually consists two paths side by side, one from M_1, and the other from M_2. The previous definition does not constrain F_3, which can be chosen to fit different purposes. For example, if $F_3 = F_1 \times F_2$, M_3 will accept those words accepted by both M_1 and M_2, and $L(M_3) = L(M_1) \cap L(M_2)$. The existence of the product automaton ultimately shows that a limited number of finite sate machines cannot integrate into a more power construct in terms of computation; we can always find a single FSM that exhibits the same overall behaviour. At the same time, this comes at the cost of a greatly expanded state space, which means the verification of algorithmic properties becomes increasingly difficult.

A *code* is defined as a non-empty set of distinct words $C = \{w_0, w_1, \dots, w_n\}$. If not all words have the same length, C is a *variable-length* code. C is *uniquely decodable* if, for any string that can be written as $x = x_0 \dots x_n$ (with $x_i \in C$), the

representation is unique. This means that, if $x = u_0 \dots u_m$ and $x = v_0 \dots v_n$ (with $u_i, v_i \in C$), then $m = n$ and $u_i = v_i, \forall i$.

2. Introduction

The equivalence of finite-state machines is a classic topic in automata theory, closely related to the notion of minimization. A classic result states that two FSMs M_1 and M_2 are equivalent if, after minimizing them, the states of M_1 can be renamed such that both M_1 and M_2 actually represent the same machine. We look at the equivalence problem from a different angle: if M_1 and M_2 are provided as black boxes, how much information is necessary to determine if they are equivalent?

In this context, black box means a function $f : \Sigma^* \to \{0, 1\}$, that can be computed by a FSM $M = (\Sigma, S, s, \delta, F)$. When M is provided as a black box, the only information available about its quintuple is the input alphabet Σ. Further data can be gathered by feeding different words into M, and observing whether they are accepted or not. In Section 3 we present a necessary and sufficient condition for the equivalence of black box finite state machines.

The problem of deciding if a given code is uniquely decodable or not is also widely known, and has numerous applications. One example [2] is encountered in the field of static verification of computer networks. The goal here is to determine if a network configuration correctly implements a given specification, without any undesirable behaviour. Headers and packets are pieces of data of a given size; every packet begins with one or more headers in succession. Each network protocol has a specific header, which is logically divided into multiple parts, called fields. They have different lengths, measured in bits, and a predefined significance. Network protocol stacks and tunneling lead to packets that have multiple headers.

For symbolic execution, headers can be described as sequences of numbers, representing the lengths of their constituent fields. We can look at natural numbers as distinct symbols (up to the maximum field length) of an alphabet. A header h with m fields can be seen as the code $E(h) = l_0 \dots l_{m-1}$, where l_i is the length of the ith field. Furthermore, the entire sequence of n headers from a single packet p can be represented as $E(p) = E(h_0) \cdot \ldots \cdot E(h_{n-1})$. An important question related to symbolic packets is whether two different protocol combinations can generate packets p_1 and p_2, such that $E(p_1) = E(p_2)$. The answer is yes if and only if the set of all headers is *not* uniquely decodable.

In Section 4, we present an automata-based method of deciding the unique decodability problem. First we discuss the intuition behind our solution, and continue by illustrating a three step algorithm.

3. Equivalence of black box finite state machines

Let M_1 and M_2 be two black box automata. If no other information is provided beforehand, there is no way to determine if $M_1 \equiv M_2$. Our only possibility of interacting with the machines is to observe their behaviour relative to different

string, and we can only provide a finite number of inputs until a decision has to be made. If we find that M_1 and M_2 disagree at some point, they are clearly not equivalent. However, nothing can be said if they agree each time. In this case, M_1 and M_2 can be equivalent, but this is not guaranteed, since for any set of words $X \subset \Sigma^*$ we can build two machines that agree on every $w \in X$, but disagree on other words.

The question turns into how much additional information is needed to determine whether M_1 is equivalent to M_2 . The most extreme case is when both quintuples are known, but we want to find a less stringent requirement. We start by assuming the number of states is know for both M_1 and M_2. For simplicity, we consider that $|S_1| = |S_2| = n$. Any definite result should be easily adapted to the general setting. The aim is to find a limit $k \in \mathbb{N}$, such that M_1 and M_2 are equivalent if they agree on all words of length at most k. Since k is now a specific number, it is possible to run both machines on every desired input in a finite (albeit potentially very large) amount of time.

First, we consider $k = n$. In other words, we try to determine if two finite state machines with n states are equivalent, if they agree on every word of length at most n. The answer is negative once more, because an invalidating example can be found for this assumption.

Example 3.1. *Consider M_1 and M_2 over $\Sigma = \{a\}$, with $S_1 = S_2 = S$, $|S| = n$, $n \geq 3$, $s_1 = s_2 = q_0$, and $F_1 = F_2 = \{q_0, q_{n-1}\}$. The two transition functions, δ_1 and δ_2, are defined as follows:*

- $\delta_1(q_i, a) = \delta_2(q_i, a) = q_{i+1}$, *for* $i = 0..n-2$
- $\delta_1(q_{n-1}, a) = q_0$
- $\delta_2(q_{n-1}, a) = q_{n-1}$

M_1 and M_2 agree on all words of length at most n, but fail to do so for most of the longer inputs. Next, we try to determine if setting the limit higher leads to a satisfactory result. Moreover, we are interested in an expression entirely dependent on the number of states, without additional variables such as the size of Σ. Before going further, we discuss a property that is essential to our later findings.

Lemma 3.1. *Let M be a FSM with n states. If M accepts every input w of lenght at most $|w| \leq n-1$, then M accepts any input.*

Proof. If a FSM accepts every word of length l, then all states which can be reached in l steps from the initial state are accepting states, because every path of length l must lead to an accepting state. Let Q_i be the set of all states reachable in i steps from the initial state, and $Q = Q_0 \cup \ldots \cup Q_{n-1}$.

When M accepts every word of at most $n-1$ symbols, the relation $Q \subseteq F$ must hold. Moreover, there is no state q such that $q \notin Q$, as can be shown using the pigeonhole principle. Any state q which is only reachable in more than $n-1$ steps, has to be at the end of a path that contains $n+1$ distinct states (the first step involves s and at most one other state). But this is impossible, because M has

only n states. Thus, $S \subseteq Q \subseteq F$. Since $F \subseteq S$ (by definition), this means $F = S$, so M will accept every input. □

Lemma 3.1 does not work for a threshold lower than $|S| - 1$. Consider the following example:

Example 3.2. *Let $M = (\Sigma, S, s, \delta, F)$ be a finite state machine, where $\Sigma = \{a\}$, $S = \{s_0, s_1, s_2, s_3\}$, $s = s_0$, $\delta(s_i, a) = s_{i+1(mod\ 4)}$, and $F = \{s_0, s_1, s_2\}$.*

All inputs w such that $|w| \leq |S| - 2$ lead to final states, but M doesn't accept every string ($s_3 \notin F$, but it is reachable). With this in mind, let us return to M_1, M_2, and the question of their equivalence.

Theorem 3.1. *Two finite state machines, $M_1 = (\Sigma, S_1, s_1, \delta_1, F_1)$ and $M_2 = (\Sigma, S_2, s_2, \delta_2, F_2)$, are equivalent if they agree on every word of length at most $k = |S_1| \cdot |S_2| - 1$.*

Proof. Consider the product finite state machine M_3, with $F_3 = F_1 \times F_2$. This particular choice of F_3 means that for any word w, $M_3(w) = M_1(w) \wedge M_2(w)$. In other words, M_3 will only accept w if and only if both M_1 and M_2 accept w. We go further, by enlarging F_3 in the following manner: $F_3 = (F_1 \times F_2) \cup (\overline{F_1} \times \overline{F_2})$, where $\overline{F_i} = S_i \setminus F_i$.

In this case, M_3 also accepts those words which are rejected by both M_1 and M_2, which means M_3 only accepts an input if M_1 and M_2 agree on it. This construct can be used to determine the equivalence of M_1 and M_2. If M_3 accepts every word, then $M_1 \equiv M_2$ (from the previous construction of M_3). The implication in Theorem 3.1 can be proved using Lemma 3.1: M_3 accepts every input if it accepts each word of at most $|S_3| - 1 = |S_1| \cdot |S_2| - 1$ symbols. □

Our equivalence result is both necessary and sufficient. Theorem 3.1 proves the latter, while the former is a trivial implication: if two finite state machines are equivalent, they surely behave in the same way over words of length equal to, or shorter than, a specified threshold. Moreover, the threshold $k = |S_1| \cdot |S_2| - 1$ is minimal: if we can only show that two FSMs agree on words shorter than k, then nothing can be said about their equivalence.

The previous property can be shown to be true by relying on Lemma 3.1 (and further illustrated by Example 3.2). If $k < |S_1| \cdot |S_2| - 1$, then we are no longer certain that the product automaton build in the proof of Theorem 3.1 accepts all possible inputs (as demonstrated in Example 3.2). Every input which is not accepted represents a string that leads to a disagreement between the FSMs under consideration. Thus, any value of k smaller than $|S_1| \cdot |S_2| - 1$ no longer guarantees the equivalence of M_1 and M_2.

To the best of the author's knowledge, Theorem 3.1 is a novel result. It is worth noting that no assumptions were made about Σ, or about anything related to M_1 or M_2, besides knowing the number of states for both of them. We have also shown that $|S_1| \cdot |S_2| - 1$ is the minimum value of k required for the truth of the equivalence result.

4. Deciding unique decodability

Next, we turn our attention toward the problem of codes and unique decodability. A very simple example of a uniquely decodable code is $\{0, 1\}$, as well as any other code with elements that only consist of different symbols. On the other hand, $\{a, ab, aab\}$ is not uniquely decodable: a string like $aaab$ can be represented both as $a \cdot aab$, and as $a \cdot a \cdot ab$.

Let C be the code over Σ which is checked for uniqued decodability, and M a finite state machine that accepts the language $L = C^*$. We want to determine if this FSM can be used to find solution to the unique decodability problem. There is not much to be inferred if M is a deterministic FSM, because every input generates a unique path. We can determine if there is at least one representation of w using codes from C (when w is accepted). The situation changes when M is a non-deterministic finite state machine. A particularly interesting example is the following non-deterministic FSM:

Example 4.1. *Let $M = (\Sigma, S, s, \delta, F)$ be a non-deterministic FSM, where $S = \{s_0\}$, $s = s_0$, $F = \{s_0\}$, and $\delta(s_0, w) = s_0, \forall w \in C$.*

The previous example uses a shorthand method to describe a FSM. First, δ is not total anymore. We assume that any transition which is not explicitly defined leads to an implicit state where any further steps lead to the rejection of the input (a sink state). We call this the *reduced* form of M. It is helpful because the execution of the FSM fails as soon as no transition is available, instead of going for more steps that ultimately reject the input anyway. Second, the domain of δ is now $S \times \Sigma^*$. This allows a more compact representation, by merging multiple states. Once more, if we reach a step where no further progress is possible, the input is implicitly rejected.

The automaton in Example 4.1 has a single explicit state, which is also an accepting state, and each transition starts from, and returns to it. This definition of M is equivalent to exhaustively trying all possible combinations of codes from C (up to $|w|$) to generate the input w. The fact that M is non-deterministic means there can be multiple paths to acceptance. Each such path represents a different sequence of codes, so finding any input that generates at least two different accepting paths means C is not uniquely decodable.

There are two issues with this approach. First, the time complexity is exponential. Second, at each step, we only determine if one particular word can be generated in more than one way. This leads once more to the situation where the search never stops if C actually is uniquely decodable.

Definition 4.1. *Let C be a code, and M' a non-deterministic FSM that accepts the language C^*, and has multiple accepting paths for an input w if and only if w has multiple encodings over C. We say that M* uniquely accepts *C^*.*

If M' uniquely accepts C^*, we build the product automaton $M'' = M' \times M'$, with $F'' = F' \times F'$. A path P'' from M'' is called *non-trivial* if it contains at least one state $(q_i, q_j) \in S''$ with $i \neq j$, where $q_i, q_j \in S'$. The construction of M''

would be redundant for a deterministic FSM, because every path is trivial. Nondeterministic automata allow multiple paths for the same input, so M'' reveals all pairs of execution paths for a given string. It is important to note that M'' can be built using the reduced representation of M' (resulting the reduced form of M''), because any path that leads to a sink state in M' cannot be included in a successful path of M''.

Proposition 4.1. *C is uniquely decodable if and only if M'' has at least one non-trivial accepting path.*

Proof. In particular, by examining M'', we can determine whether *at least one* string can be accepted via two distinct transition sequences. This is equivalent to finding a non-trivial path P'' from s'' to any final state of M''. If P'' exists, it can be split into P'_1 and P'_2, two distinct accepting paths of M', and therefore C is not uniquely decodable. On the other hand, if C is uniquely decodable, then M' has at most one accepting path for any input. In turn, this means all accepting paths of M'' are trivial, because we cannot find two distinct acceping paths in M for the same input. □

The biggest challenge of building M'' is the construction of the transition relation. The shorthand representation in Example 4.1 is not well-suited for describing a product FSM that executes two copies of M, because the presence of transitions which use multiple symbols hinders the application of Definition 1.1.

Example 4.2. *Consider $C = \{u, v, w\}$, and $u = v \cdot w$.*

The input string u can be accepted in two ways: either by $\delta(s_0, u) = s_0$, or via $\delta(\delta(s_0, v), w) = s_0$. However, the product FSM M'' must use the same number of symbols during each step, so δ'' cannot be properly with s_0 as the only explicit state. Additional states are required for the initial FSM to allow the construction from Definition 1.1.

The simplest general solution for building M' that uniquely accepts C^* is to have a separate path for each $w \in C$, one separate transition for each symbol in w, and one separate state for all but the last every symbol in w. We call this the trivial construction of the uniquely accepting FSM in the reduced form. For example, the FSM in Figure 1 uniquely accepts the language C^*, for $C = \{a, ab, aab\}$.

We propose the following three-step procedure to determine if a variable-length code C is uniquely decodable:

(1) Build the reduced form of M', the finite state machine that uniquely accepts C^*, using the trivial construction method.
(2) Build the product automaton $M'' = M' \times M'$, with $F'' = F' \times F'$.
(3) Check whether a non-trivial accepting path exists in M''.

We evaluate the complexity of each phase in reverse order. The search for a non-trivial path is actually done during the construction of M'', which requires at most $O(|C| \cdot |S''|) = O(|C| \cdot |S'|^2)$ steps. The trivial construction method leads to a FSM with $|S'| = \sum_{w \in C} |w| - |C| + 1$. We denote $n = |S'|$. The construction of

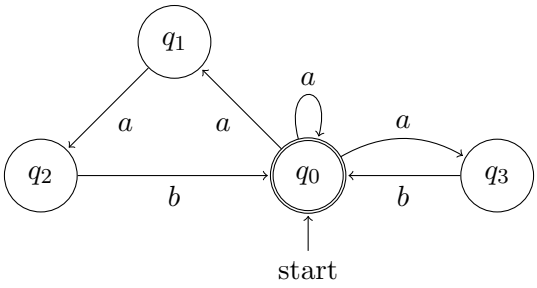

Fig. 1. A FSM that uniquely accepts $\{a, ab, aab\}^*$

M' requires $O(n)$ steps. In conclusion, the complexity of the previous procedure is $O(|C|n^2)$.

Our solution illustrates an important aspect of the deterministic/non-deterministic FSM correspondence. In general, results that concern non-deterministic finite state machines have to account for the exponential state explosion caused by the transformation to a deterministic automaton. We present a problem instance where the asymptotic complexity is conserved. The conversion of M'' to a deterministic FSM is not necessary, because reachability properties carry over between the non-deterministic and deterministic cases.

The overall complexity of the solution can be reduced using instances of M' with smaller number of states. The trivial construction leads to an upper bound, in the sense that we never have to consider a larger FSM. Simple optimizations are, for example, merging common prefixes and suffixes. Figure 2 shows a smaller FSM, which is equivalent to the one in Figure 1. Depending on C, the number of states in S' can be significantly smaller than n. The effort required to find a smaller M' that uniquely accepts C has to be balanced with the complexity of the overall unique decodability problem.

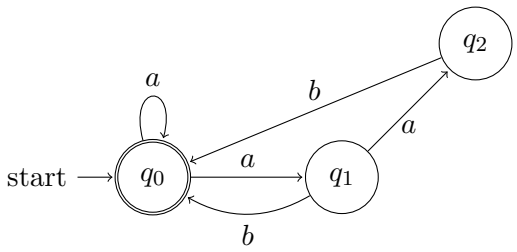

Fig. 2. A smaller FSM that uniquely accepts $\{a, ab, aab\}^*$

5. **Related Work**

The problem of FSM equivalence has been extensively studied [3], and also adapted to modern theoretical constructs and developments, culminating with quantum finite automata [4, 5]. Most current research efforts focus on the situation where

complete information is available, and optimizations aim to increase the applicability at scale [6, 7, 8].

In Section 3, we focus on the equivalence of two finite state machines, without access to full information about their configuration. This can be seen as trying to determine whether two FSMs are equivalent by observing their output on an arbitrary (but finite) number of strings, and leads to Theorem 3.1.

FSM equivalence with incomplete information has been mostly studied in the context of machine learning or language processing [9, 10]. To the author's knowledge, the closest result to the one presented in Section 3 can be found in [11], which gives a somewhat similar property for probabilistic automata, which is further discussed in [12], leading to an equivalence result for 1-way quantum finite automata [13].

The best known algorithm for deciding if a code is uniquely decodable is the Sardinas-Patterson algorithm [14], which can run as fast as $O(|C|n)$ time using specialized data structures. We found our solution easier to implement, and reasonably fast in practice. Our use of automata to discern properties of codes is similar to other approaches [15, 16]. We focus on general variable-length codes, which are useful for static verification of network properties.

6. **Conclusions**

We proved Theorem 3.1, which establishes a necessary and sufficient condition for the equivalence of black box finite state machines, and described an automata-based algorithm that can be used to determine if a variable length code is uniquely decodable.

REFERENCES

[1] *H. R. Lewis and C. H. Papadimitriou*. Elements of the Theory of Computation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd ed., 1997.

[2] *R. Stoenescu, M. Popovici, L. Negreanu, and C. Raiciu*. SymNet: Scalable Symbolic Execution for Modern Networks. In Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference, SIGCOMM '16, pp. 314–327. ACM, New York, NY, USA, 2016.

[3] *J. E. Hopcroft*. Introduction to Automata Theory, Languages, and Computation. Pearson Addison Wesley, 3rd ed., 2007.

[4] *A. Kondacs and J. Watrous*. On the Power of Quantum Finite State Automata. In Proceedings of the 38th Annual Symposium on Foundations of Computer Science, FOCS '97, pp. 66–. IEEE Computer Society, Washington, DC, USA, 1997.

[5] *C. Moore and J. P. Crutchfield*. Quantum Automata and Quantum Grammars. In Theor. Comput. Sci., **vol. 237**, no. 1-2, pp. 275–306, 2000.

[6] *C. A. J. V. Eijk and J. A. G. Jess*. Detection of equivalent state variables in finite state machine verification. In In Proc of the International Workshop on Logic Synthesis, pp. 3–35. 1995.

[7] *S.-Y. Huang, K.-T. Cheng, K.-C. Chen, F. Brewer, and C.-Y. Huang*. AQUILA: An Equivalence Checking System for Large Sequential Designs. In IEEE Trans. Comput., **vol. 49**, no. 5, pp. 443–464, 2000.

[8] *X. Y. Wang and X. R. Ma*. An Equivalence Relation on Extended Finite State Machines. In 2009 International Conference on Computational Intelligence and Software Engineering, pp. 1–7. 2009.

[9] *C. de la Higuera*. Learning Finite State Machines, pp. 1–10. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[10] *B. Balle, J. Castro, and R. Gavald*. Learning probabilistic automata: A study in state distinguishability. In Theoretical Computer Science, **vol. 473**, pp. 46 – 60, 2013.

[11] *A. Paz*. Introduction to Probabilistic Automata (Computer Science and Applied Mathematics). Academic Press, Inc., Orlando, FL, USA, 1971.

[12] *L. Li and D. Qiu*. Determining the equivalence for one-way quantum finite automata. In Theoretical Computer Science, **vol. 403**, no. 1, pp. 42 – 51, 2008.

[13] *A. Brodsky and N. Pippenger*. Characterizations of 1-Way Quantum Finite Automata. In SIAM J. Comput., **vol. 31**, no. 5, pp. 1456–1478, 2002.

[14] *A. Sardinas and C. Patterson*. A necessary sufficient condition for the unique decomposition of coded messages. In IRE Internat. Conv. Rec., , no. 8, p. 104108, 1953.

[15] *A. Kontorovich and A. Trachtenberg*. Deciding Unique Decodability of Bigram Counts via Finite Automata. In J. Comput. Syst. Sci., **vol. 80**, no. 2, pp. 450–456, 2014.

[16] *T. Head and A. Weber*. Deciding Code Related Properties by Means of Finite Transducers, pp. 260–272. Springer New York, New York, NY, 1993.