Int. J. Appl. Math. Comput. Sci., 2021, Vol. 31, No. 2, 337–351
DOI: 10.34768/amcs-2021-0023

MINIMAL STATE AUTOMATA FOR DETECTING A β GLOBIN GENE MUTATION

DEVI FITRI FERDANIA [a,*], IRAWATI [a], HANNI GARMINIA [a], AKHMALOKA [b], KEMAL AZIEZ RACHMANSYAH [a]

[a] Algebra Research Group
Bandung Institute of Technology
Jl. Ganesha No.10, Jawa Barat, Bandung, Indonesia
e-mail: ferdania.fitri@gmail.com,
fitriferdania@students.itb.ac.id

[b] Biochemistry Research Group
Bandung Institute of Technology
Jl. Ganesha No.10, Jawa Barat, Bandung, Indonesia

Beta-thalassemia is an autosomal recessive blood disorder characterized by abnormalities in the synthesis of β globin. Together with α globin, it is a subunit of globin protein, called hemoglobin, located inside our red blood cells to deliver oxygen from the lungs to all of the tissues throughout our body. Thereby, individuals with β-thalassemia will often feel limp due to a lack of oxygen dissolved in their blood. In this paper, a finite state automaton to detect and classify β globin gene mutations using its DNA sequence is constructed. Finite state automata have a close connection to an algebraic structure, that is, a monoid. Together with the theory of the syntactic monoid, we present a methodology to minimize the number of the internal states of an automaton to have minimal state automata. Therefore, a minimal state automaton can be constructed to detect β globin gene mutation causing the β-thalassemia disease. We have developed a MATLAB program to conduct the appropriate simulations.

Keywords: minimal state automata, syntactic monoid, β-thalassemia, biological sequences.

1. Introduction

Beta globin is a subunit of hemoglobin, a globin protein. Along with α globin, it is located inside the red blood cells with the purpose for delivering oxygen from the lungs to all other tissues throughout the body. β-thalassemia syndromes are a group of hereditary blood disorders characterized by a reduction in beta globin chain synthesis, resulting in reduced hemoglobin in red blood cells (RBCs), which in turn decreases RBC production (Galanello and Origa, 2010). These disorders are caused by a mutation(s) on the β globin gene exon 1. More than 200 different β-thalassemia mutations have now been characterized worldwide, but only approximately 20 are common in South-East Asia (Sunthornwat *et al.*, 2011; Lie-Injo *et al.*, 1989). The World Health Organization (WHO) stated that the carriers of this disorder are roughly more than 250 million people worldwide, which is equal to 4.5% of the global population. Based on this increasing prevalence, proper diagnosis of the mutation and severity classification for β-thalassemia must be provided with the most appropriate treatment (Wahidayat *et al.*, 2018). Thalassemia carrier inspection is very effective for suppressing the number of individuals with the β-thalassemia disease. This inspection involves hematological examination, followed by mutation location detection (Handayani and Onggo, 2014). Methods for classifying and analyzing patterns in nucleotide sequences (contained in DNA) have an important role in the mutation location detection stage.

DNA (deoxyribonucleic acid) is a biomolecule composed of four nucleotides (bases): adenine (A), cytosine (C), guanine (G), and thymine (T). DNA also contains the living organism's genetic code. In recent

*Corresponding author

decades, contemporary molecular biology and computer science have enriched each other for the need to develop solutions for efficient data analysis (Papiez, 2019; Zok *et al.*, 2020). Husna *et al.* (2017) proposed sequencing methods to detect point mutations on the β globin gene, by stretching out a given β globin gene and comparing it to a normal β globin gene. Their method can seek point mutations from a one-by-one nucleotide comparison. However, since the β globin gene has a length of 800–1500 nucleotides, this method is not quite efficient for large scale use. Searls and Murphy (1995) constructed a finite transducer to simulate a mechanism of a single base mutation. It motivated Mazumdar and Raha (2008) to improve and also apply the model on the Fragile X Syndrome disease.

Mehdi and Khan (2016) proposed a method for analyzing DNA patterns using finite automata, Mealy, and Moore machines. Their model can find 3–4 patterns of nucleotides in different machines. Each machine will have a task to find a pattern. But by combining those machines to find several different patterns, it becomes a very complex tool and not that efficient. Lastly, another method to detect point mutations on the β globin gene is greatly explained by Sunthornwat *et al.* (2011). They constructed finite state automata to find a five base pattern, which is assumed to be the optimal size for reliably detecting if a mutation is or is not present in a given β globin gene. However, similar to Mehdi and Khan's (2016) idea, each constructed automaton could only find one single pattern. Despite the shortcomings mentioned before, using automata to detect patterns in genes is still a very good idea because the automaton is one of the simplest models of computation (Straubing and Weil, 2012).

A finite-state automaton is an abstract model of computing devices (Reddy and Dawud, 2015). In formal language theory, finite automata are known as abstract machines that can recognize, accept, and generate a lot of words over a set of languages. In the development of this machine, finite automata play an important role in many fields. Nevertheless, there might be many outputs of finite automata with different internal states constructed for a single problem. From all of those automata, there is an automaton that has a minimal number of internal states. Therefore, it is better to find the automata with the least number of internal states, which can reduce the number of algorithms and solve the problem more efficiently.

Many aspects of finite state automata have strong relationships with pure mathematics such as the algebraic structure, graph theory, logical thinking, algorithm theory, and category theory. During the past few decades, both theories have enriched each other. One of those useful relationships is the construction of a monoid when an automaton is given and vice versa. A monoid is an algebraic structure of a set together with its single associative binary operation and an identity element (Howie, 1976). Finite state automata have many common properties with monoids. Pin (2019) explained the relationship between monoids and automata through a transition monoid. Furthermore, Klima and Polak (2016) developed the concepts of a syntactic monoids structure in a regular language. Straubing and Weil (2012) proposed the relationship between syntactic monoid and automata.

This paper aims to introduce the minimization of finite state automata with an algebraic approach using a syntactic monoid and applying the method to find point mutations in the β globin gene that cause the beta-thalassemia disease. The benefit of the method could be used to create a severity clustering of individuals with β-thalassemia based on the type of mutation found by the minimal state automata. Since it is a novel approach, in this paper, the method is applied to only six specific point mutations to make it easier for us to explain the steps of automata construction.

The paper is organized as follows. The next section will introduce the definition of finite state automata and semigroup theory. Section 3 describes the relationship between monoids and automata. Section 4 presents the methodology to minimize the number of internal states of an automaton by using its relationship to syntactic monoid structure. Section 5 will describe the development of the finite state automaton to detect and classify β globin gene mutations using its biological sequences. Next, with the method described in Section 4, the minimization to produce an efficient algorithm will be done and simulated with the MATLAB program. Finally, a summary of the results and explanation of some open problems for future research will be conveyed in the last section.

2. Definitions

We assume that the reader is familiar with the basic notions of formal language theory and some of the basic notions in algebra. For convenience, we describe some of the important definitions and propositions in automata theory and semigroup theory.

2.1. Finite state automata. Formal language theory has developed a concept about words and languages. A language is a set of words whose letters are taken from some alphabets. As one of the popular types of sequential machines with output that can process a language, a finite state automaton can model a device that can be in one of finitely many states, receive a discrete sequence (called words) of inputs, and then change its state accordingly from one to another.

Definition 1. (*Finite state automata* (*Mazumdar and Raha, 2008*)) An automaton is a quintuple $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F \rangle$, where Q is a non-empty set called the set

of internal states, Σ_k is the input alphabet, δ is a function from $Q \times \Sigma_k \to Q$ called the transition function, a is a given element of Q called the initial state, and F is a given subset of Q called the set of final states of an automaton. Moreover, $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F\rangle$ is called a finite state automaton if the set Q is finite.

For a finite state automaton $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F\rangle$, the function δ is usually defined in a table of transition that shows $\delta(q_i, \sigma_i)$ and visualized by a transition graph, called the Moore graph. Consider the transition function $\delta : Q \times \Sigma_k \to Q$. The domain of δ can be extended from $Q \times \Sigma_k$ to $Q \times \Sigma_k^*$ by the following:

1. $\delta(q, \epsilon) = q, \forall q \in Q$,
2. $\delta(q, x\sigma) = \delta(\delta(q, x), \sigma), \forall q \in Q, \forall \sigma \in \Sigma_k$, $\forall x \in \Sigma_k^*$,

The above conditions occur so that the automata constructed by extending the definition of the transition function domain can maintain their structure. Consider the following proposition.

Proposition 1. (Mazumdar and Raha, 2008) *The transition function δ of Definition 1 is extended from $Q \times \Sigma_k$ to $Q \times \Sigma_k^*$ if and only if the following conditions are satisfied:*

1. $\delta(q, \epsilon) = q, \forall q \in Q$,
2. $\delta(q, \sigma x) = \delta(\delta(q, \sigma), x), \forall q \in Q, \forall \sigma \in \Sigma_k, \forall x \in \Sigma_k^*$.

Proof. Let δ be the extended transition function from $Q \times \Sigma_k$ to $Q \times \Sigma_k^*$. Then $\delta(q, \epsilon) = q$, and $\delta(q, \sigma x) = \delta(\delta(q, x), \sigma), \forall q \in Q, \forall \sigma \in \Sigma_k, \forall x \in \Sigma_k^*$.

Let $x = \sigma_0\sigma_1\sigma_2 \dots \sigma_n$, for $n \in \{0, 1, 2, \dots\}$. Then

$$\begin{aligned}
\delta(q, \sigma x) &= \delta(q, \sigma(\sigma_0\sigma_1\sigma_2 \dots \sigma_n)) \\
&= \delta(\delta(q, (\sigma\sigma_0\sigma_1\sigma_2 \dots \sigma_{n-1})), \sigma_n) \\
&= \delta(\delta(\delta(q, (\sigma\sigma_0\sigma_1\sigma_2 \dots \sigma_{n-2})), \sigma_{n-1}), \sigma_n) \\
&= \dots \\
&= \delta(\delta(\delta(\dots(\delta(q, \sigma), \sigma_0), \dots), \sigma_{n-1}), \sigma_n) \\
&= \delta(\delta(q, \sigma), \sigma\sigma_0\sigma_1\sigma_2 \dots \sigma_n) \\
&= \delta(\delta(q, \sigma), x)
\end{aligned}$$

■

Corollary 1. *The transition function δ is extended from $Q \times \Sigma_k$ to $Q \times \Sigma_k^*$ if and only if the following conditions are satisfied:*

1. $\delta(q, xy) = \delta(\delta(q, x), y), \forall q \in Q, \forall x, y \in \Sigma_k^*$,
2. *If* $\delta(q, x) = \delta(q, y)$ *then* $\delta(q, xz) = \delta(q, yz), \forall q \in Q, \forall x, y, z \in \Sigma_k^*$.

Proof. The steps for proving this corollary are the same as the proof of the previous proposition. ■

Based on the extended automata transition function domain described above, the definition of an automaton by inputting the alphabet in the form of words is clear. However, this expansion can make the constructed automata very large and ineffective. By proving the proposition and corollary above, it is found that the transition from a word $x \in \Sigma_k^*$ can be broken down into a transition of each element Σ_k that constructs x. Consequently, even though the domain of the transition function is changed to be wider on Σ_k^*, the automata are still the same, namely, $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F\rangle$. It does not need to be written as $\langle Q, \Sigma_k^*, \delta, a, F\rangle$.

Definition 2. (*Response function* (*Mazumdar and Raha, 2008*)) The response function of a finite state automaton $\mathcal{A}$, denoted by $rp_{\mathcal{A}}(x)$, is a function from $\Sigma_k^* \longrightarrow Q$, defined by $rp_{\mathcal{A}}(x) = \delta(a, x)$.

There are two types of states that exist in an automaton. That is whether the transition of words brings the state to the final one or not.

Definition 3. (*Accessible state* (*Mazumdar and Raha, 2008*)) A state $q \in Q$ of an automaton $\mathcal{A}$ is called accessible if and only if there exists $x \in \Sigma_k^*$ such that $q = rp_{\mathcal{A}}(x)$.

Then a non-accessible state of an automaton is redundant. By omitting this type of state, a more efficient automaton can be constructed.

Definition 4. (*Connected sub-automaton* (*Mazumdar and Raha, 2008*)) A connected sub-automaton of a given automaton $\mathcal{A}$, denoted by $\mathcal{A}^C$, is defined by $\mathcal{A}^C = \langle Q^C, \Sigma_k, \delta^C, a, F^C\rangle$, where Q^C is the set of all accessible states of Q, i.e., $Q^C = \{q \in Q | \exists x \in \Sigma_k^*$, such that $q = rp_{\mathcal{A}}(x)\}$, δ^C is the restriction of δ, that is, $Q^C \times \Sigma_k^* \longrightarrow Q^C$, and F^C is a set of all accessible final states, i.e., $F^C = F \cap Q^C$. A machine $\mathcal{A}$ is said to be connected if and only if $Q = Q^C$.

Consider $q \in F$. An element of a final state can be regarded as a terminal or an end point of an automaton. If $x \in \Sigma_k^*$ contains n element(s) of Σ_k and $rp_{\mathcal{A}}(x) \in F$, then $n - 1$ transition(s) before the internal state arrives at the final state will respectively give zero output. The transition will give unit output when the last transition reaches the final state.

Definition 5. (*Accepted word* (*Mazumdar and Raha, 2008*)) Let $\mathcal{A}$ be a finite state automaton. A word $x \in \Sigma_k^*$ is said to be accepted by $\mathcal{A}$ if and only if $rp_{\mathcal{A}}(x) \in F$.

Moreover, let L be a language. If all of $x \in L \subseteq \Sigma_k^*$ satisfy $rp_{\mathcal{A}}(x) \in F$, then L is accepted by $\mathcal{A}$.

2.2. Semigroup theory. In algebraic structure theory, a semigroup is a non-empty set M equipped with an associative binary operator. When a semigroup contains an identity element, its structure can be enlarged to another structure, namely, a monoid, or formally as follows.

Definition 6. (*Monoid, Howie, 1976*) A monoid is a pair $(M, *)$, where M is a non-empty set together with an associative operator $*$ on M (that is, $* : M \times M \to M$, where $(a, b) \mapsto a * b$, $\forall a, b \in M$), and M has an identity element ϵ over $*$ (that is, $\epsilon * a = a = a * \epsilon$, $\forall a \in M$).

From a non-empty set, define an equivalence relation (a relation that satisfies transitive, symmetry, and reflexive properties). With a monoid structure, a compatible equivalence relation is possible, as follows.

Definition 7. (*Congruence relation on monoid* (*Howie, 1976*)) Let $(M, *)$ be a monoid. The equivalence relation $\sim$ over M is said to be a congruence relation if, for $x, y \in M$, $x \sim y$ and then $a * x * b \sim a * y * b$ for all $a, b \in M$.

An equivalence relation is just a relation defined on a set that satisfies reflexive, symmetric, and transitive properties. If we have a congruence relation, it is an equivalence relation defined on an algebraic structure (a semigroup, monoid, group, ring, etc.) that preserved the operation. Thus a congruence relation can be seen as an equivalence relation defined on a algebraic structure that is compatible with the structure. If $\sim$ a congruence relation over a monoid $(M, *)$, then every element in M can be grouped into equivalence classes, i.e., $[a] = \{b \in M | a \sim b\}$ for $a \in M$. This congruence relation induces a quotient set $M/\sim = \{[a] | a \in M\}$. Note that $(M/\sim, *)$ forms a monoid structure with the same operation as $(M, *)$, that is,

$$* : M/\sim \times M/\sim \longrightarrow M/\sim,$$
$$[a] * [b] = [a * b].$$

Definition 8. (*Monoid homomorphism* (*Straubing and Wiel, 2012*)) Let $(M, *_1)$ and $(N, *_2)$ be monoids. A monoid homomorphism ψ from M to N is a map between M and N that preserves the monoid operation, that is, for $m_1, m_2 \in M$, we have $\psi(m_1 *_1 m_2) = \psi(m_1) *_2 \psi(m_2)$.

Definition 9. (*Language accepted by a monoid* (*Pin, 2019*)) Let $L \subseteq \Sigma_k^*$ be a language and $\psi : \Sigma_k^* \to M$ be a monoid epimorphism (a surjective homomorphism). Then L is accepted by a monoid M if there exists $P \subseteq M$ such that $L = \psi^{-1}(P)$.

3. Relationship between monoids and automata

Let $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F \rangle$ be a finite state automaton. It will involve two important sets (an internal state and input alphabet set) and a transition function. Consider Σ_k^* to be a set of all possible strings composed of the elements of Σ_k. Define a binary operation $*$ as a concatenation operator on Σ_k^*. For any words $x, y \in \Sigma_k^*$, $x * y$ form a new word xy. A non-commutative free monoid is a non-commutative monoid that has a generating set. The following is an important fact about Σ_k^*.

Proposition 2. (Pal *et al.*, 2016) *The pair* $(\Sigma_k^*, *)$ *is a non-commutative free monoid.*

Proof. It is straightforward (see Pal *et al.*, 2016). ■

The dual form of an automaton in the form of a monoid will be constructed. It must be a structure that can represent the whole activities of an automaton. Therefore, an automaton can be viewed in two different ways, as an automaton itself and as a monoid.

For all $x \in \Sigma_k^*$, consider the following mapping:

$$\tau : \Sigma_k^* \longrightarrow Q^Q,$$
$$x \longmapsto \tau_x,$$

where Q^Q is a set of all possible maps from Q to Q. Thus,

$$\tau_x : Q \longrightarrow Q,$$
$$q \longmapsto \delta(q, x)$$

for all $q \in Q$ and δ is the transition function of $\mathcal{A}$. It is easy to prove that $\mathrm{Im}(\tau) \subseteq Q^Q$, together with a composition operator $\circ$, form a monoid structure. Thus, $(\mathrm{Im}(\tau), \circ)$ can represent all of the automata's characteristics (the transition function, internal state, and final state).

Definition 10. (*Transition monoid* (*Planting, 2013*)) Let $\mathcal{A} = \langle Q, \Sigma_k^*, \delta, a, F \rangle$ be a finite state automaton. Then the transition monoid $M(\mathcal{A})$ over $\mathcal{A}$ is

$$M(\mathcal{A}) = \{\tau_x \in Q^Q | x \in \Sigma_k^*\}.$$

One of the important things about a finite state automaton is the processed object, that is, a language. The relationship between the acceptance of a language on automata and its transition monoid is as follows.

Proposition 3. *Let* $L \subseteq \Sigma_k^*$ *be a language. If an automaton* $\mathcal{A} = \langle Q, \Sigma_k^*, \delta, a, F \rangle$ *accepts* L*, then its transition monoid* $M(\mathcal{A})$ *also accepts* L*.*

Proof. Let L be accepted by a finite state automaton $\mathcal{A}$ and $\tau : \Sigma_k^* \to M(\mathcal{A})$ be the a monoid epimorphism that maps a word in Σ_k^* to a transition monoid $M(\mathcal{A})$. For any word $u \in \Sigma_k^*$, there is a relation on Q, denoted by $\tau(u)$, and defined by $(p, q) \in \tau(u)$ if there exists $p, q \in Q$ such that $\delta(p, u) = q$. Then u is accepted by $\mathcal{A}$ if and only if $(a, q) \in \tau(u)$, where $q \in F$. For convenience, if there exist $p, q \in Q$ such that $\delta(p, u) = q$, then we write $\tau(u)_{(p,q)} = 1$ and zero otherwise. Define

$$P = \{\varphi \in M(\mathcal{A}) | \varphi_{(a,q)} = 1\},$$

$\varphi : Q \to Q$. Note that $L = \tau^{-1}(P)$. Then, conclude that τ accepts L, also $M(\mathcal{A})$ will accept L. ■

Proposition 4. *Let $L \subseteq \Sigma_k^*$ be a language. If a monoid $(M, *)$ accepts L, then an automaton induced by $(M, *)$ also accepts L.*

Proof. Let L be accepted by a finite monoid $(M, *)$. There exists a monoid homomorphism $\psi : \Sigma_k^* \to M$ and $P \subseteq M$ such that $\psi^{-1}(P) = L$. Choose $\mathcal{A} = \langle M, \Sigma_k^*, \delta, 1_M, P\rangle$ with 1_M as the identity element of M and $\forall m \in M$, $\sigma \in \Sigma_k^*$. We must have $\delta(m, \sigma) = m * \psi(\sigma)$. Note that, for any $x \in L$, x is accepted by $\mathcal{A}$ if and only if $\delta(1_M, x) \in P$. Therefore, $\delta(1_M, x) = \psi(x) \in P$, so $u \in \psi^{-1}(P)$, which means $\mathcal{A}$ accepts L. ■

Then a finite state automaton and its transition monoid can be viewed as dual forms of each other's structure. If we have a finite state automaton, then it is possible to construct a monoid structure as a transition monoid and vice versa. The advantages of this relationship are the congruence relation over a monoid, the homomorphism between two monoids, the construction of a monoid quotient over a congruence relation, the isomorphic form of a monoid, and other structures that arise from the algebraic point of view can be observed in automata theory. This concept will make it easier to enlarge finite state automata concepts, such as that of the equivalence relation on automata, or moreover the concept of congruence relation on automata, the concept of the homomorphism between two automata, the concept of two isomorphic automata, etc.

4. Minimal automaton

Let L be a subset of a free monoid M (a monoid that has a generating set). Define the left quotient of L by $v \in M$ as the set

$$v^{-1}L = \{u \in M | vu \in L\},$$

and the right quotient of L by $v \in M$ as the set

$$Lv^{-1} = \{u \in M | uv \in L\}.$$

The left and right quotients above are said to be syntactic quotients. Each induces an equivalence relation on M, called a syntactic relation.

Definition 11. (*Syntactic relation* (*Pin, 2019*)) Let L be a subset of a monoid M. Then, respectively, the right syntactic relation and the left syntactic relation are

$$\sim_L = \{(s,t) \in M \times M | Ls^{-1} = Lt^{-1}\},$$
$$_L\sim = \{(s,t) \in M \times M | s^{-1}L = t^{-1}L\}.$$

From both relations above, define a congruence syntactic relation over a monoid, that is, when the right and left syntactic relations are satisfied.

Definition 12. (*Syntactic congruence* (*Hetzl, 2017*)) Let L be a subset of a free monoid M. Then, the syntactic congruence relation over M, denoted by $\equiv_L$, is the right and left syntactic relation on L, that is, $(s \equiv_L t \iff \forall x, y \in M(xsy \in L \iff xty \in L))$.

Thus, it is possible to define a quotient monoid $M/\equiv_L$, which will be called as syntactic monoid.

Definition 13. (*Syntactic monoid*) Let M be a monoid and L be a language. Define a syntactic relation $\equiv_L$ as in Definition 12. A syntactic monoid $M(L)$ is a quotient monoid $M/\equiv_L$ with operation: for any equivalence class $[s]_L$ and $[t]_L$, it satisfies $[s]_L * [t]_L = [s * t]_L$.

Consider the following proposition, which will be closely related to automata theory.

Proposition 5. *Let L be a language. A syntactic monoid $M(L)$ is the smallest monoid that accepts the language L.*

Proof. Let $M(L)$ accept L. For any monoid N that accepts L, $M(L)$ is a quotient of a submonoid N. Then $M(L)$ is smaller than N. Because it can be applied for any monoid N, then $M(L)$ is the smallest monoid that accepts the language L. ■

From Proposition 5 we conclude that a syntactic monoid can be viewed as the simplest monoid that accepts a language L. Previously, we found that the transition monoid can be seen as a dual form of a finite state automaton. We found that the syntactic monoid is the smallest monoid that accepts L. Thus, the construction of a transition monoid from an automaton based on a syntactic monoid is feasible. It will be the foundation of the minimal state automata concepts.

Let $\mathcal{A} = \langle Q, \Sigma_k^*, \delta, a, F\rangle$ be a connected finite state automaton (see Definition 4) with L being a language accepted by $\mathcal{A}$. In algebra, partition of a non-empty set can be done by collecting the same objects into equivalence classes. Consider the transition monoid $M(\mathcal{A})$. Two functions are said to be the same function if for all $q \in Q$ they satisfy

$$\tau_u(q) = \tau_v(q),$$
$$\delta(q, u) = \delta(q, v).$$

Because $\mathcal{A}$ is a connected automaton, then there exists $y \in \Sigma_k^*$ such that $(\tau_u \circ \tau_y)(q) \in F$ and $(\tau_v \circ \tau_y)(q) \in F$. This can be written as

$$\delta(q, uy) \in F \iff \delta(\delta(q, u), y) \in F \iff uy \in L$$

and

$$\delta(q, vy) \in F \iff \delta(\delta(q, v), y) \in F \iff vy \in L.$$

Then $uy \in L \iff vy \in L$. In the same way, there exists $x \in \Sigma_k^*$ such that $(\tau_x \circ \tau_u)(q) \in F$ and $(\tau_x \circ \tau_v)(q) \in F$. It can be written as

$$\delta(q, xu) \in F \iff \delta(\delta(q, x), u) \in F \iff xu \in L$$

and

$$\delta(q, xv) \in F \iff \delta(\delta(q, x), v) \in F \iff xv \in L.$$

Then $xu \in L \iff xv \in L$. These are, respectively, the right and the left syntactic relation on Σ_k^*. Thus, we have the following.

Proposition 6. *Let $L \subseteq \Sigma_k^*$, $u, v \in \Sigma_k^*$, $\mathcal{A} = \langle Q, \Sigma_k^*, \delta, a, F\rangle$ be a connected finite state automaton, and $M(\mathcal{A})$ be transition monoid of $\mathcal{A}$. Then $\tau_u = \tau_v$ if and only if $\forall x, y \in \Sigma_k^*$ satisfy*

$$xuy \in L \iff xvy \in L.$$

Assume that

$$\tau_u = \tau_v \iff u \equiv_L v$$

From the previous part, it is known that a syntactic monoid $M(\mathcal{A})/\equiv_L$ is the smallest monoid that accepts a language L. Furthermore, observe the automata that correspond to the transition monoid in the form of a syntactic monoid. Let $u, v \in \Sigma_k^*$ satisfy $\tau_u = \tau_v$. Then there exist some $u \equiv_L v$ such that $xuy \in L \iff xvy \in L$. This means u and v are related through the left and right syntactic relation. Because they satisfy the right syntactic relation, for $y \in \Sigma_k^*$ consider

$$vy \in L \iff \delta(a, vy) \in F \iff \delta(\delta(a, v), y) \in F$$

and

$$\delta(\delta(a, u), y) \in F \iff \delta(a, uy) \in F \iff uy \in F.$$

Since $\mathcal{A}$ is a connected automaton, let $p = \delta(a, v)$ and $q = \delta(a, u)$. These two equations above will be equivalent if $\delta(p, y) \in F \iff \delta(q, y) \in F$.

Definition 14. (*Equivalence relation on an internal state*) Let $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F\rangle$ be a connected finite state automaton and $p, q \in Q$. Then $\forall y \in \Sigma_k^*$,

$$p \equiv q \iff (\delta(p, y) \in F \iff \delta(q, y) \in F).$$

Consequently, a finite state automaton that corresponds to a transition monoid $M(\mathcal{A})/\equiv_L$ will have the smallest number of internal states.

Proposition 7. *A syntactic monoid over a language L is a transition monoid of the minimal automaton that accepts L.*

Proof. Let $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F\rangle$ be a connected finite state automaton that accepts L with $M(\mathcal{A})$ be its transition monoid. Choose $\mathcal{A}_{\min} = \langle Q_L, \Sigma_k, \delta_L, a_L, F_L\rangle$, where $Q_L = Q/\equiv$, $\delta_L : Q_L \times \Sigma_k^* \to Q_L$ with $\delta_L([q]_\equiv, x) = [\delta(q, x)]_\equiv$, $a_L = [a]_\equiv$, and $F_L = \{[q]_\equiv | q \in F\}$. Thus $M(\mathcal{A}_{\min})$, the transition monoid of $\mathcal{A}_{\min}$, is $M(\mathcal{A})/\equiv_L$. For any words $x \in L \subseteq \Sigma_k^*$. Note that

$$rp_{\mathcal{A}\min}(x) = \delta_L([a]_\equiv, x) = [\delta(a, x)].$$

Since $\mathcal{A}$ is connected, then $rp_{\mathcal{A}}(x) = \delta(a, x) \in F$. Therefore $rp_{\mathcal{A}\min}(x) \in F_L$. This means that L is accepted by $\mathcal{A}_{\min}$.

For any $\mathcal{B} = \langle Q', \Sigma_k, \delta', a', F'\rangle$, an arbitrary finite state automata accepting L, let $u, v \in \Sigma_k^*$. Since it is already proven that $u \equiv_L v \iff \forall x \in \Sigma_k^*, \delta(a', u) \equiv \delta(a', v)$, the index of $\equiv_L$ (the number of equivalence classes of $\equiv_L$) is no more than $|Q'|$, because the equivalence relation $\equiv$ partitions Q'. As this is satisfied for an arbitrary finite state automaton $\mathcal{B}$, the cardinality of $Q_L = Q/\equiv$ must reach the minimum over another set of internal state from the other finite state automata that accept L. Thus $\mathcal{A}_{\min}$ is a minimal state automaton. ■

A minimal state automaton that corresponds to the transition monoid in the form of syntactic monoid has been constructed. However, another problem arises. How to construct equivalence classes on Q such that we have Q_L? The following lemma will help us to construct an algorithm for the equivalence classes of Q. Let $\equiv_m$ from $\equiv$ defined as a modulo equivalence relation. Define the number of alphabets on a word x by $lg(x)$. For any $p, q \in Q$, $p \equiv_m q$ if and only if for all $v \in \Sigma_k^*$ with $lg(v) \leq m$, $(\delta(p, v) \in F \iff \delta(q, v) \in F)$.

Lemma 1. *Let $p, q \in Q$. Then $p \equiv q$ if and only if $p \equiv_m q$ for $m = |Q| - 2$.*

Proof. For some m, let the equivalence relations $\equiv_m$ and $\equiv_{m+1}$ coincide. Next, claim that $\equiv_m$ and $\equiv$ coincide. Suppose that there are two states $p \not\equiv q$ and they are distinguished only by words with a length greater than m. Choose $w = uv$, where $lg(v) = m + 1$ such that $\delta(p, u) = p'$ and $\delta(q, u) = q'$ is not an equivalent modulo $m + 1$, $p' \not\equiv_{m+1} q'$. Nevertheless, this means that $p' \not\equiv_m q'$, p' and q' being distinguished by a word v' with $lg(v') \leq m$. Thus p and q are distinguished by uv' with $lg(uv') < lg(w)$. The process is then repeated until a word with a length less than $m + 1$ is found and then we have a contradiction. Therefore, the shortest word that distinguishes p, q must be less than or equal to m, thus $\equiv_m$ coincides with $\equiv$.

Let $\equiv_{m+1}$ do not coincide with $\equiv_m$, then $\equiv_{m+1}$ has a larger number of class than $\equiv_m$. From Proposition 7, the number of index $\equiv$ does not exceed $|Q|$, thus, for $m = 0$, $\equiv_0$ has two classes. Therefore, the sequence $\{\equiv_m\}_{m\geq 2}$ will stabilize when m reaches $|Q| - 2$. ■

From Lemma 1, we can partition Q from $\equiv_0$ first, which is an element of Q that needs $v \in \Sigma_k^*$ with $lg(v) = 0$ such that the transition reaches the final state. From $\equiv_0$, we will have two equivalence classes, i.e., the final state and non-final state classes. Then, do the refinement to the class through $\equiv_1$ by collecting the internal state in each previous class, which, if transitioned by one element of Σ_k, the internal state will fall in the same equivalence class. Next, refine the $\equiv_2$ classes and so on until we get the equivalence classes of $\equiv_m$ which can no longer be partitioned by $\equiv_{m+1}$. Then, Q_L is defined clearly and the automaton $\mathcal{A}_{\min} = \langle Q_L, \Sigma_k, \delta_L, a_L, F_L \rangle$ is a minimal state automaton.

5. Detecting and classifying β globin gene mutations

In the genes or chromosomes of a living creature, an alteration in either the structure or the number of genes or chromosomes caused by a chemical or physical agent called a mutagen is possible (Pal *et al.*, 2016). A gene mutation is an alteration in the DNA sequence such that the sequence differs from what can be found in most people. Mutations have various types, which are the recombination, deletion, addition, or substitution of a DNA sequence (or a DNA sub-sequence).

Beta-thalassemia is an autosomal genetic blood disorder characterized by anomalies in the synthesis of the β-globin chain to produce hemoglobin. There are many types of mutations associated with β-thalassemia. However, only a small number of common mutation are usually found in South-East Asia. This paper will focus on three classification types of mutations:

1. *Transcription mutation (missense mutation).*
   The point mutation of this type arises from an error in the transcription process, which occurs when adenine is replaced by guanine on the β globin chain. The mutation causes a permanent change and forms a new DNA strand.

2. *Nonfunctional mRNA (nonsense mutation).*
   In the gene, there is a codon that can a stop protein synthesis, called stop codon. It is possible that the stop one replaces another codon which led to termination of the whole process of protein synthesis before it is completed. This is called a nonfunctional mRNA mutation.

3. *mRNA processing mutation.*
   The base sequence on mRNA consists of 2 types of sequences. One of them is an intron, which transcribes and then excises from mRNA before it is translated into a protein. This mutation makes an intron cut before the transcription process is fulfilled. Splicing the intron can produce untranslated sequences in the transcription process.

Table 1 shows a normal DNA sequence of the β globin gene with an ATG in row 3 as the start codon and an TAA in row 22 as the stop codon. The bold face and underlined letters are the points where mutations usually occurred.

5.1. Method. A biological sequence is a chain of alphabetical characters describing a biological molecular object, such as DNA, RNA, and protein. For DNA, the alphabets used are usually A for adenine, C for cytosine, G for guanine, and T for thymine. Thus, the β globin gene can be shown as a sequence of A, T , G, and C nucleotides. This fact can help the construction of the mathematical model, by choosing $\Sigma_4 = \{A, T, G, C\}$ and Σ_4^* as a set of all possible combinations of nucleotides contained in Σ_4. It is already proven in Proposition 2 that Σ_4^* forms a free monoid. Thus the examined DNA strand will be depicted as a language $L \subseteq \Sigma_4^*$. The application of finite state automata to search for the particular sub-sequences of DNA will be discussed in this section.

Start with the construction of automata to detect a one point mutation. It is done to simplify the explanation of constructing an automaton to search for a particular motif. Furthermore, the main idea of this method is to detect a single point mutation and will be applied to other cases, i.e., for two, six, and more point mutations. Next is the explanation on how to construct an automaton to detect two point mutations. The aim is to help explain the procedure of minimizing an automaton based on the concept built in the previous section. In the same way, it will be applied to other cases, i.e., for six and more point mutations. Consider the steps to construct our finite state automata:

Step 1. Define an optimal size pattern.
Sunthornwat *et al.* (2011) defined an optimal search pattern for mutation as the smallest length pattern of DNA sequences that occurs exactly once in abnormal DNA. Thus, an optimal size pattern can determine whether or not the mutation exists. This particular pattern is very important for the mathematical model because the

Algorithm 1. Minimal automaton for detecting a β globin gene mutation.

Step 1. Define an optimal size pattern.

Step 2. Construct the automata graph to seek the optimal-size pattern.

Step 3. Minimize the constructed automata.

Step 4. Write a new transition table and its graph.

Step 5. Construct the MATLAB code to simulate finite state automata.

Table 1. Normal beta globin gene.

1	CCTAAGCCAG	TGCCAGAAGA	GCCAAGGACA	GGTACGGCTG	TCATCACTTA
51	GACCTCACCC	TGTGGAGCCA	CACCCTAGGG	TTGGCCAATC	TACTCCCAGG
101	AGCAGGGAGG	GCAGGAGCCA	GGGCTGGGC	**ATG** **A**AA AGT	CAG GGC AGA
				1 2 3	4 5 6
149	GCC ATC TAT	TGC TTA CAT	TTG CTT CTG	ACA CAA CTG	TGT TCA CTA
	7 8 9	10 11 12	13 14 15	16 17 18	19 20 21
193	GCA ACC TCA	AAC AGA CAC	CAT GGT GCA	CTG ACT CCT	GAG GAG AAG
	22 23 24	25 26 27	28 29 30	31 32 33	34 35 36
238	TCT GCG GTT	ACT GCC CTG	TGG GGC **A**AG	GTG AAC GTG	GAT GAA GTT
	37 38 39	40 41 42	43 44 45	46 47 48	49 50 51
283	GGT GGT **G**AG	GCC CTG GGC	AG**G** TTG **G**TA	TCA AGG TTA	CAA GAC AGG
	52 53 54	55 56 57	58 59 60	61 62 63	64 65 66
328	TTT AAG GAG	ACC AAT AGA	AAC TGG GCA	TGT GGA GAC	AGA GAA GAC
	67 68 69	70 71 72	73 74 75	76 77 78	79 80 81
373	TCT TGG GTT	TCT GAT AGG	CAC TGT CTC	TCT CTG CCT	ATT GGT CTA
	82 83 84	85 86 87	88 89 90	91 92 93	94 95 96
418	TTT TCC CAC	CCT TTG GCT	CCT GGT GGT	CTA **C**CC AGT	TGG ACC CAG
	97 98 99	100 101 9 102	103 104 105	106 107 108	109 110 111
463	AGG TTC TTT	GAG TCC TTT	TGG GGA TCT	GTC CAC TCC	TGT TGC TGT
	112 113 114	115 116 117	118 119 120	121 122 123	124 125 126
508	TAT GGG CAA	CCC **TAA** GGT	GAA GGC TCA	TGC CAA CAA	AGT GCT CGG
	127 128 129	120 131			
553	TGC CTT TAG	TGA TGG CCT	GGC TCA CCT	GGA CAA CCT	CAA GGG CAC
598	CTT TGC CAC	ACT GAG TGA	GCT GCA CTG	TGA CAA GGT	GCA CGT GGA
643	TCC TGA GAA	CTT CAG GGT	GAG TCT ATG	GGA CCC TTG	ATG TTT TTC
688	CCG GGC TTC	TTT TCT ATG	GTT AAG TTC	ATG TCA TAG	GAA GGG GAG
733	AAG TAA CAG	GGT ACA GTT	TAG AAT GGG	AAA CAG ACG	AAT GAT TGC
778	ATC AG				

sought type of mutation is just a point or base mutation. This makes it very difficult to determine whether or not a mutation occurs in a long chain of the β globin gene if the optimal size pattern was not defined beforehand.

The algorithm to search for the optimal size pattern is initialized from a pattern of three nucleotides. Then, it systematically searches for patterns of increasing length that contain mutation until it terminates

- successfully by finding the pattern that occurs exactly once in an abnormal code and does not occur in the normal code,
- unsuccessfully by searching an abnormal code without finding the search pattern,
- unsuccessfully by reaching a maximum size of the search pattern (usually 6–9 bases).

This paper is focused on the first point, in which the pattern is discovered. Then, the unique pattern will be used to construct the finite state automata. For example, to find an optimal size pattern for a transcription mutation (from adenine to guanine), the process with three bases that contain the point mutation will be initialized; those are ATG, TGA, and GAA. Next, the pattern ATG, TGA, and GAA in the samples that are known to be positive for β-thalassemia will be searched for. Henceforth, the search process for a larger pattern size, that is, four bases, i.e., ATGA, TGAA, CATG, etc., five bases, and six bases, will be performed.

The results show that, for the three and four bases pattern, there are many occurrences of each pattern in both abnormal and normal codes. Therefore, the three and four bases patterns are not suitable to identify and determine the occurrences of a transcription mutation. Meanwhile, for the five bases pattern, there will exist one occurrence of the pattern (ATGAA) if the mutation is present in abnormal genes but not in normal ones. Otherwise, for the other samples, ATGAA will occur in the normal genes. But this only happened for a few samples. With the six bases pattern, on the other hand, the CATGAA pattern occurs only once in abnormal code and does not occur in normal one. Evidently, the six bases pattern is the optimal size pattern for a transcription mutation. Thus, we can conclude that CATGAA can be used to determine the occurrences of a transcription mutation.

As observed by Sunthornwat *et al.* (2011) and with some modifications by the authors, Table 2 shows the optimal size pattern for the common β-thalassemia mutation.

Step 2. Construct the automata graph to search for the optimal size pattern.
Construct the automata graph which can describe the pattern on each transition. The following part will describe how to construct the automata to search for one, two, and six point mutations. The procedure can be applied to other types of point mutation that can cause β-thalassemia.

For seeking the GCTAG patterns, let $\mathcal{A}$ be a finite state automaton that will be created. Let q_0 be the initial state of $\mathcal{A}$. Also, q_0 will be transitioned to the state q_1 when $\mathcal{A}$ comes across 'G' and transitioned back to q_0 (which can actually be considered as no transition) if 'C', 'T', or 'A' is discovered. From q_1, it will be transitioned to the state q_2 when 'C' appears, then transitioned back to q_0 when 'A' or 'T' turns up, and back again to q_1 when 'G' appears. Proceeding in the same way, from q_2, the transition to state q_3 will happen when $\mathcal{A}$ discovers 'T', back to q_0 with 'A', 'C', or 'G'. Therefore, to search for the GCTAG pattern, we need at least six states $\{q_0, q_1, q_2, q_3, q_4, q_5\}$.

Accordingly, from q_0, if the process ended up at q_1 on the next state, this indicates that the automaton has read 'G', if by any chance it went to q_2, the strand has made a 'GC' pattern, and so on. This means it will only end up at q_5 if the pattern the automaton read is exactly 'GCTAG'. Therefore, q_5 is a final state. Finally, define an automaton $\mathcal{A} = \{\{q_0, q_1, q_2, q_3, q_4, q_5\}, \{A, T, G, C\}, \delta_1, q_0, \{q_5\}\}$, where δ_1 is a transition function described in Table 3. Its Moore graph is shown in Fig. 1.

Next, we proceed in the same way for the GTAAG pattern to detect a non-functional mRNA mutation. Let $\mathcal{B}$ be a finite state automaton that will be constructed. Then, define an automaton $\mathcal{B} = \{\{q_6, q_7, q_8, q_9, q_{10}, q_{11}\}, \{A, T, G, C\}, \delta_2, q_6, \{q_{11}\}\}$, where δ_2 is a transition function described in Table 4. Figure 2 shows the Moore graph of $\mathcal{B} = \{\{q_6, q_7, q_8, q_9, q_{10}, q_{11}\}, \{A, T, G, C\}, \delta_2, q_6, \{q_{11}\}\}$.

Combine $\mathcal{A}$ and $\mathcal{B}$ in one finite state automaton in order to make it simpler and run them together. Assume the first automaton, $\mathcal{A}$, starts from q_0 and will be transitioned to q_1 if $\mathcal{A}$ read 'G'. Meanwhile, the same conditions apply to the second automaton, which is $\mathcal{B}$, from q_6; it will be transitioned to q_7 when $\mathcal{B}$ read 'G'. Then we can delete q_6 and q_7, since respectively, q_0 can represent q_6 and q_1 can represent q_7. From q_1, it must be transitioned to q_2 when the automaton read 'C', transitioned to q_8 when the automaton reads 'T', back to q_1 when the automaton read 'G', and back to q_0 when the automaton reads 'A'. Then, repeat the same transition

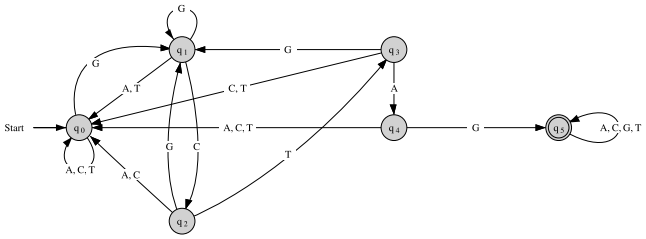


Fig. 1. Automaton $\mathcal{A}$ for seeking the GCTAG pattern.

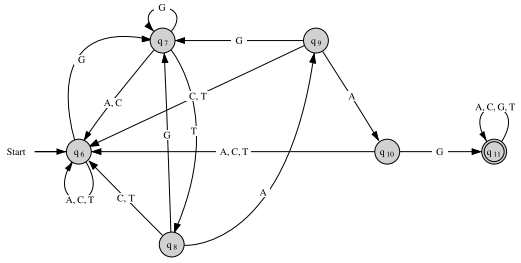


Fig. 2. Automaton $\mathcal{B}$ for seeking the GTAAG pattern.

Table 2. Optimal size patterns.

Types of mutation	Point mutation	Normal pattern	Abnormal pattern
Transcription mutation	(A-G)	CATAAA	CATGAA
Nonfunctional mRNA	(A-T) (G-A) (C-A)	GCAAG GTGAG TACCC	GCTAG GTAAG TAACC
mRNA Processing mutation	(G-C) (G-T)	TGGTA AGGTTG	TGCTA AGTTTG

rules to each state.

For convenience, assume that $q_8 = q_6$, $q_9 = q_7$, $q_{10} = q_8$, $q_{11} = q_9$. Then we have $\mathcal{C} = \{\{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\}, \{A, T, G, C\}, \delta, q_0, \{q_5, q_9\}\}$ as a finite state automaton to detect GCTAG and GTAAG simultaneously. The Moore graph of $\mathcal{C}$ is shown by Fig. 3(b) and its transition in Table 5.

In the same way, generalize the method to construct an automaton to detect six point mutations as mentioned in Table 2, called $\mathcal{D}$. Its transition is shown in Table 6. Because the Moore graph of $\mathcal{D}$ in Fig. 3(a) is too complicated to visualize, the main idea of its transition function is shown in Fig. 3(c).

Step 3. Minimize the constructed automata.
The concept of the minimal automaton has already been constructed in the previous section. In association with the concept of an algebraic structure, namely, a monoid, or more specifically a syntactic monoid, an equivalence

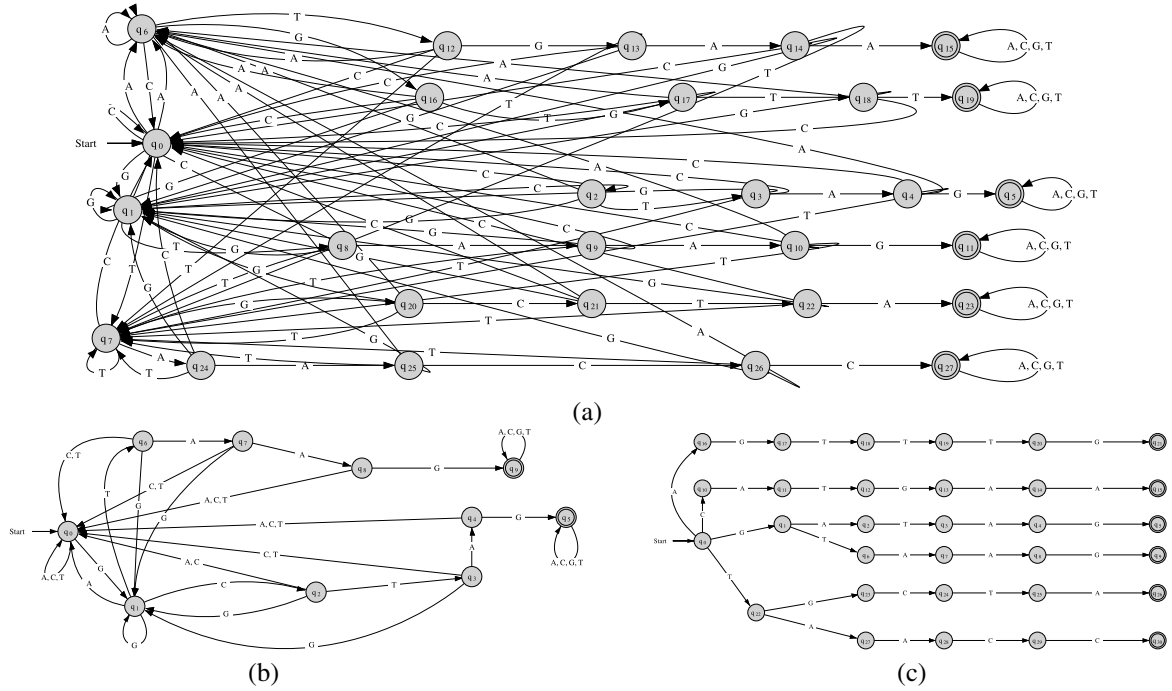


Fig. 3. Automaton $\mathcal{D}$ for seeking six point mutations (a), automaton $\mathcal{C}$ for seeking the GCTAG and GTAAG patterns (b), main transition of $\mathcal{D}$ in a simple visualization graph (c).

Table 3. Transition table of $\mathcal{A}$.

δ	A	C	G	T
q_0	q_0	q_0	q_1	q_0
q_1	q_0	q_2	q_1	q_0
q_2	q_0	q_0	q_1	q_3
q_3	q_4	q_0	q_1	q_0
q_4	q_0	q_0	q_5	q_0
q_5	q_5	q_5	q_5	q_5

Table 4. Transition table of $\mathcal{B}$.

δ	A	C	G	T
q_6	q_6	q_6	q_7	q_6
q_7	q_6	q_6	q_7	q_8
q_8	q_9	q_6	q_7	q_6
q_9	q_{10}	q_6	q_7	q_6
q_{10}	q_6	q_6	q_{11}	q_6
q_{11}	q_{11}	q_{11}	q_{11}	q_{11}

relation on the set of the internal state of an automaton such that we can minimize the number of its internal state has been found, but it still accepts the same language. By an important concept of Lemma 1, the following steps are obtained to minimize the constructed automata.

For convenience and effectiveness of writing, we will describe the steps for minimizing $\mathcal{C}$ (an automaton that detects two patterns) and, in the same way, for minimizing $\mathcal{D}$ (an automaton that detects six patterns). Separation starts from 0-equivalence first. Because the final state in the previously constructed automaton is $\{q_5, q_9\}$, then q_5 and q_9 are combined in one equivalence class, while the other internal states are combined in the different equivalence classes. Then the following will be obtained:

$$P_0 = \{\{q_0, q_1, q_2, q_3, q_4, q_6, q_7, q_8\}, \{q_5, q_9\}\}.$$

Check each equivalence class to re-partition the equivalence class P_0 above. The method to re-partition P_0 is by looking into whether each transition in the same equivalence class falls in internal states with the same class. If the equivalence class is different, and so then separate the internal state from the others.

For each element of the input alphabet, the transitions of q_5 and q_9 always fall in the same class, and so then $\{q_5, q_9\}$ cannot be partitioned any more.

Consider the set $\{q_0, q_1, q_2, q_3, q_4, q_6, q_7, q_8\}$. Because $\delta(q_4, G) = q_5$ and $\delta(q_8, G) = q_9$, a separation of $\{q_4, q_8\}$ into another different class must be done. Note that q_4 and q_8 are equivalent because all of the transitions for the same input alphabets fall in the same equivalence class. Then the following will be obtained:

$$P_1 = \{\{q_0, q_1, q_2, q_3, q_6, q_7\}, \{q_4, q_8\}, \{q_5, q_9\}\}.$$

In the same method, $\{q_0, q_1, q_2, q_3, q_6, q_7\}$ must be re-partitioned. The other equivalence classes do not need to be reviewed because they are already equivalent. Note that it was only the appearance of q_4 and q_8 that needed to be looked into in the transition table

Table 5. Transition table of $\mathcal{C}$.

δ	A	C	G	T
q_0	q_0	q_0	q_1	q_0
q_1	q_0	q_2	q_1	q_6
q_2	q_0	q_0	q_1	q_3
q_3	q_4	q_0	q_1	q_0
q_4	q_0	q_0	q_5	q_0
q_5	q_5	q_5	q_5	q_5
q_6	q_7	q_0	q_1	q_0
q_7	q_8	q_0	q_1	q_0
q_8	q_0	q_0	q_9	q_0
q_9	q_9	q_9	q_9	q_9

of $\{q_0, q_1, q_2, q_3, q_6, q_7\}$. Because $\delta(q_3, A) = q_4$ and $\delta(q_7, A) = q_8$, the separation of q_3 and q_7 into different equivalence class must be done. Because q_3 and q_7 are equivalent, then $\{q_3, q_7\}$ cannot be partitioned anymore. Hence, the following will be obtained:

$$P_2 = \{\{q_0, q_1, q_2, q_6\}, \{q_3, q_7\}, \{q_4, q_8\}, \{q_5, q_9\}\}$$

In the same method, q_2 and q_6 must be separated from $\{q_0, q_1, q_2, q_6\}$. However, note that q_2 and q_6 are not equivalent, because $\delta(q_2, A) = q_0$, while $\delta(q_6, A) = q_7$. Then the following will be obtained:

$$P_3 = \{\{q_0, q_1\}, \{q_2\}, \{q_6\}, \{q_3, q_7\}, \{q_4, q_8\}, \{q_5, q_9\}\}.$$

In the same way,

$$P_4 = \{\{q_0\}, \{q_1\}, \{q_2\}, \{q_6\}, \{q_3, q_7\}, \{q_4, q_8\}, \{q_5, q_9\}\}$$

and

$$P_5 = \{\{q_0\}, \{q_1\}, \{q_2\}, \{q_6\}, \{q_3, q_7\}, \{q_4, q_8\}, \{q_5, q_9\}\} = P_4.$$

Because $P_5 = P_4$, the reduction of internal states of the previously constructed automata is successful, from ten internal states to seven internal states. With the same method, minimization of $\mathcal{D}$ results from 31 internal states to 22 internal states. Based on the previous section, this reduction method will ensure that the results are minimal automata that will still accept the same language.

Step 4. Write a new transition table and the graph of the constructed minimal automata.
For the previous method, a new transition table and new graph for $\mathcal{C}$ are shown, respectively in Table 7 and Fig. 4(a). A new transition table and graph for $\mathcal{D}$ are, respectively, showed in Table 8 and Fig. 4(b).

Step 5. Write MATLAB code to simulate finite state automata.
The MATLAB code can be constructed to simulate the finite state automata obtained above and to summarize the conclusions from the automata. The program contains two functions. The first one is to simulate the automata, while the other one runs through the DNA sequences to find the start the codon ATG, numbering every three nucleotides from start codon to have codon number, and then search for six patterns simultaneously by using the first program. When the automata reach the single final state, the pattern found is then recognized.

5.2. Results and a discussion. There are 15 samples of the β globin gene to examine, consisting of nine samples (samples numbered from 1 to 9) of the abnormal β globin gene (patients are from Indonesia and Thailand) and six samples of the normal β globin gene (sample numbered 10–15). Table 9 shows the results obtained from all 15 samples when processed in the MATLAB program. Based on the results in Table 9, the MATLAB program relying on the constructed automaton has been successfully tested regarding whether normal or abnormal β globin gene chain. Thus, there are no 'false positives' and no 'false negatives' from the automata model, and the mutation on the given β globin gene chain can be detected and classified.

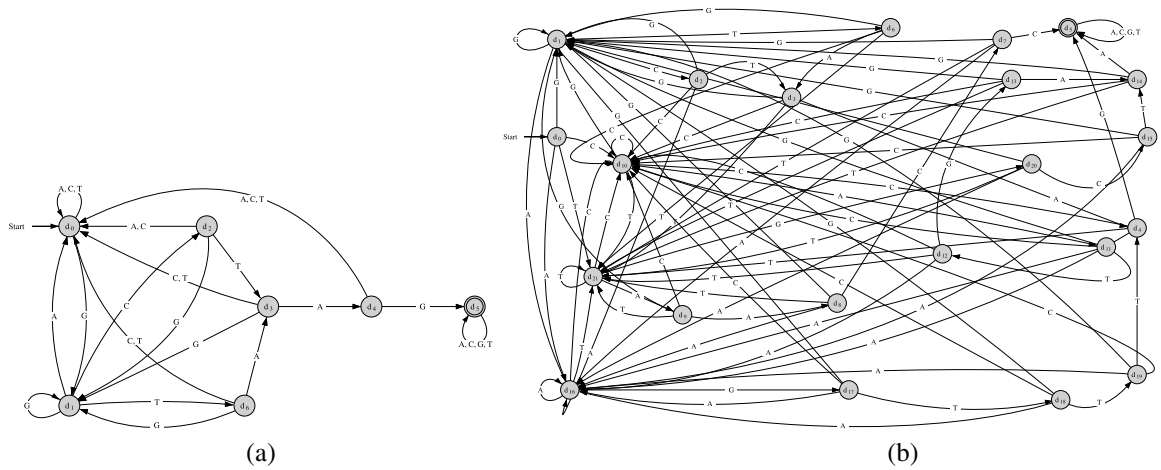
The "Severity class" column in Table 9 is a division of samples into three classes. Samples with no mutation patterns are classified into the normal class, which means the β globin gene may be normal, but further inspection is needed to check for other mutation patterns. The mild class is for samples with a transcription mutation or non-functional mRNA or mRNA processing mutation patterns, or two of them, or all three. Samples in the severe class have all three mutation patterns in them with the possibility to have a deletion or addition mutation because of the feasibility of a shift in the gene. The shift possibility can be seen from the unusual position of the mutations.

Let us consider the time complexity of this method. Suppose that there are m patterns to be found and the length of the DNA sequence is n. The complexity of minimal state automaton is $O(n)$, because it does not need to check for every pattern in each step and there is no operation needed. This is better than in the case of both earlier finite state automata and single pattern-searching algorithms such as regexp in MATLAB or the classical built-in search method. This is because they will search for the pattern from the beginning of a text until the end, then for the second pattern this will start again from the beginning, and so on. So they will have $O(n)$ complexity for each pattern and their complexities are $O(mn)$. For multiple pattern-searching algorithms, e.g., the AC algorithm and the Commentz-Walter algorithm, our method's complexity is on par with them, as those methods' complexities are $O(n)$, too.

The problem with most of the above algorithms,

Table 6. Transition table of $\mathcal{D}$.

δ	A	C	G	T	δ	A	C	G	T
q_0	q_{16}	q_{10}	q_1	q_{22}	q_{16}	q_{16}	q_{10}	q_{17}	q_{22}
q_1	q_{16}	q_2	q_1	q_6	q_{17}	q_{16}	q_{10}	q_1	q_{18}
q_2	q_{11}	q_{10}	q_1	q_3	q_{18}	q_{16}	q_{10}	q_1	q_{19}
q_3	q_4	q_{10}	q_1	q_{22}	q_{19}	q_{16}	q_{10}	q_1	q_{20}
q_4	q_{16}	q_{10}	q_5	q_{22}	q_{20}	q_{16}	q_{10}	q_{21}	q_{22}
q_5	q_5	q_5	q_5	q_5	q_{21}	q_{21}	q_{21}	q_{21}	q_{21}
q_6	q_7	q_{10}	q_1	q_{22}	q_{22}	q_{27}	q_{10}	q_{23}	q_{22}
q_7	q_8	q_{10}	q_1	q_{22}	q_{23}	q_{16}	q_{24}	q_1	q_{22}
q_8	q_{16}	q_{10}	q_9	q_{22}	q_{24}	q_{16}	q_{10}	q_1	q_{25}
q_9	q_9	q_9	q_9	q_9	q_{25}	q_{26}	q_{10}	q_1	q_{22}
q_{10}	q_{11}	q_{10}	q_1	q_{22}	q_{26}	q_{26}	q_{26}	q_{26}	q_{26}
q_{11}	q_{16}	q_{10}	q_{17}	q_{12}	q_{27}	q_{28}	q_{10}	q_1	q_{22}
q_{12}	q_{16}	q_{10}	q_{13}	q_{22}	q_{28}	q_{16}	q_{29}	q_1	q_{22}
q_{13}	q_{14}	q_{10}	q_1	q_{22}	q_{29}	q_{16}	q_{30}	q_1	q_{22}
q_{14}	q_{15}	q_{10}	q_1	q_{22}	q_{30}	q_{30}	q_{30}	q_{30}	q_{30}
q_{15}	q_{15}	q_{15}	q_{15}	q_{15}					



(a) (b)

Fig. 4. Minimal automata for seeking two patterns (a), for seeking six patterns (b).

including minimal state automata, is their pre-processing complexity. It is arguable whether this is a major problem in detecting the β globin gene mutation because the patterns are known beforehand, hence the pre-processing can be done way before the detection.

Lastly, minimal state automata also have fewer internal states compared to the usual finite state automata. This will make the transition table and the Moore graph simpler and more efficient to implement in actual cases.

6. Conclusions and future

The concept of the relationship between automata and algebraic structures in the form of a monoid was constructed. Given an automaton $\mathcal{A} = \langle Q, \Sigma_k, \delta, a, F \rangle$, its dual form is a transition monoid $M(\mathcal{A})$, that is,

$$M(\mathcal{A}) = \{\tau_x \in Q^Q | x \in \Sigma_k^*\},$$

where $\tau : \Sigma_k^* \longrightarrow Q^Q$. The transition monoid will maintain the important characteristics of an automaton, which is the relationship between the set of internal states, the set of the input alphabet, and its transition function. Given an automaton structure, it can be seen from the perspective of a monoid, and vice versa.

Through this relationship, the concept of automata with the least number of internal states, i.e., a minimal state automaton, was also developed. Minimal automata are obtained by reducing the number of internal states through equivalence relations. The relation is induced by a congruence syntactic one. It has already been proven that the syntactic monoid is a transition monoid of a minimal state automaton.

Last, a finite state automaton to detect and classify mutations on β globin gene was constructed. To simulate it, a MATLAB program to test 15 (positive and negative)

Table 7. Transition table of a minimal automaton $\mathcal{C}$ for seeking two point mutations.

δ	A	C	G	T
d_0	d_0	d_0	d_1	d_0
d_1	d_0	d_2	d_1	d_6
d_2	d_0	d_0	d_1	d_3
d_3	d_4	d_0	d_1	d_0
d_4	d_0	d_0	d_5	d_0
d_5	d_5	d_5	d_5	d_5
d_6	d_3	d_0	d_1	d_0

Table 8. Transition table of minimal automata for seeking six point mutations.

δ	A	C	G	T
d_0	d_{16}	d_{10}	d_1	d_{21}
d_1	d_{16}	d_2	d_1	d_6
d_2	d_{16}	d_{10}	d_1	d_3
d_3	d_4	d_{10}	d_1	d_{21}
d_4	d_{16}	d_{10}	d_5	d_{21}
d_5	d_5	d_5	d_5	d_5
d_6	d_3	d_{10}	d_1	d_{21}
d_7	d_{16}	d_5	d_1	d_{21}
d_8	d_{16}	d_7	d_1	d_{21}
d_9	d_8	d_{10}	d_1	d_{21}
d_{10}	d_{11}	d_{10}	d_1	d_{21}
d_{11}	d_{16}	d_{10}	d_1	d_{12}
d_{12}	d_{16}	d_{10}	d_{13}	d_{21}
d_{13}	d_{14}	d_{10}	d_1	d_{21}
d_{14}	d_5	d_{10}	d_1	d_{21}
d_{15}	d_{16}	d_{10}	d_1	d_{14}
d_{16}	d_{16}	d_{10}	d_{17}	d_{21}
d_{17}	d_{16}	d_{10}	d_1	d_{18}
d_{18}	d_{16}	d_{10}	d_1	d_{19}
d_{19}	d_{16}	d_{10}	d_1	d_4
d_{20}	d_{16}	d_{15}	d_1	d_{21}
d_{21}	d_9	d_{10}	d_{20}	d_{21}

β-thalassemia samples was successfully run. Actually, this program is not restricted to the β globin gene cases.

The benefits of this method are, first, the fact that severity clustering of individuals with β-thalassemia can be developed based on the number and type of mutation that has been found. Severity clustering is needed to make a proper diagnosis and to determine the most appropriate treatment. Also, the proposed method comes from an algebraic structure, namely, a monoid. The dual relationship between finite state automata and transition monoids makes it easier to enlarge many concepts in finite state automata from an algebraic point of view. For example, the concept of the homomorphism of two automata, the isomorphism between two automata, the automata quotient, and many others that arise from an algebraic point of view.

For further research, there are many aspects that can be developed. First, the model created in this paper is limited to six types of mutations. Thus, for more comprehensive results on analyzing the β-thalassemia disease, this model can be developed in the same way for other types of mutations that cause β-thalassemia. Second, the manual process of minimalizing the internal states may not be a problem, but an algorithm to automatically do that surely helps, especially in large-scale use. Through the algorithm described in the previous section, a MATLAB program can be made to ease the process of separation of internal states into equivalence classes. Lastly, this paper was written from the perspective of a mathematician, so the aspects of biology were not really a emphasized. For further research, this model can be analyzed from both biological and medical perspectives to obtain better conclusions.

Acknowledgment

This research was fully funded by P3MI ITB (a research and community service program of ITB). Some parts of this research were attended by Elvin J. Moore (see References), especially with regard to sharing some normal and abnormal β globin gene DNA samples from Thailand and the MATLAB algorithm which makes it easy for the authors to develop and enlarge the model.

References

Galanello, R. and Origa, R. (2010). Beta thalassemia, *Orphanet Journal of Rare Diseases* **5**(11): 1–15, DOI: 10.1186/1750-1172-5-11.

Handayani, N.S.N. and Onggo, A.T. (2014). Identifikasi mutasi gen β-globin ekson 1 pada pembawa thalassemia, *Biogenesis Jurnal Ilmiah Biologi* **2**(1): 63–69.

Hetzl, S. (2017). Automata and formal languages, *Technical report*, Vienna University of Technology, Vienna.

Howie, J.M. (1976). *An Introduction to Semigroup Theory*, Academic Press Inc., New York.

Husna, N., Sanka, I., Arif, A.A., Putri, C., Leonard, E., and Nur Handayani, N.S. (2017). Prevalence and distribution of thalassemia trait screening, *Journal of Medical Science* **49**(3): 106–113, DOI: 10.19106/JMedSci004903201702.

Klima, O. and Polak, L. (2016). Syntactic structures of regular languages, *Theoretical Computer Science* **800**: 125–141, DOI: 10.1016/j.tcs.2019.10.020.

Lie-Injo, L., Cai, S., Wahidijat, I., Moeslichan, S., Lim, M., Evangelista, L., Doherty, M. and Kan, Y. (1989). Beta-thalassemia mutations in Indonesia and their linkage to beta-haplotypes, *American Journal of Human Genetics* **45**(6): 971–975.

Mazumdar, D. and Raha, S. (2008). Finite state machine for mutation, *Advanced Modeling and Optimization* **10**(2): 241–265.

Table 9. Simulation results.

Sample no.	Start codon	Simulation results	Base position	Codon position	Severity class
1	base number 130	found nonfunctional mRNA mRNA processing mutation	 261,288,447 302, 306, 741	 44,53,106 57,59,204	mild
2	base number 130	found transcription mutation nonfunctional mRNA mRNA processing mutation	 133 264,450 305,309	 2 45,107 58,60	mild
3	base number 130	found transcription mutation nonfunctional mRNA mRNA processing mutation	 133 263,290,449 304, 308	 2 44,53,106 58,59	mild
4	base number 130	found transcription mutation nonfunctional mRNA mRNA processing mutation	 133 263,290,449 304, 309	 2 44,53,106 58,60	mild
5	base number 131	found transcription mutation nonfunctional mRNA mRNA processing mutation	 134 264,291,450 305, 309	 2 44,54,106 58,60	mild
6	base number 101	found transcription mutation nonfunctional mRNA mRNA processing mutation	 134 264,291,450 302, 309	 11 54,63,116 67,69	severe
7	base number 100	found transcription mutation nonfunctional mRNA mRNA processing mutation	 132 262,289,448 303, 307	 11 56,63,116 68,69	severe
8	base number 129	found transcription mutation nonfunctional mRNA mRNA processing mutation	 131 261 302	 2 44 58	mild
9	base number 101	found nonfunctional mRNA mRNA processing mutation	 261,447 302,306	 53,115 67,69	severe
10	130	not found	–	–	normal
11	130	not found	–	–	normal
12	129	not found	–	–	normal
13	131	not found	–	–	normal
14	130	not found	–	–	normal
15	129	not found	–	–	normal

Mehdi, M. and Khan, A. (2016). DNA pattern analysis using Fa, Mealy, and Moore machines, *International Journal of Computer Science and Information Security* **14**(8): 235–243.

Pal, J., Mazumdar, D. and Raha, S. (2016). An algebra for biological sequences, *International Journal for Computational Biology* **5**(2): 28–40.

Papiez, A., Badie C. and Polanska, J. (2019). Machine learning techniques combined with dose profiles indicate radiation response biomarkers, *International Journal of Applied Mathematics and Computer Science* **29**(1): 169–178, DOI: 10.2478/amcs-2019-0013.

Pin, J.E. (2019). Mathematical foundations of automata, *Technical report*, Paris Diderot University, Paris.

Planting, A. (2013). *From Automata to Monoids and Back Again*, PhD thesis, Radboud University Nijmegen, Nijmegen.

Reddy, P.S. and Dawud, M. (2015). Application of semigroup, *Global Journal of Science Frontier Research* **15**(3): 16–26.

Searls, D.B. and Murphy, K.P. (1995). Automata theoretic models of mutation and alignment, *Proceedings of the International Conference on Intelligent Systems in Molecular Biology, Cambridge, UK*, pp. 341–349.

Straubing, H. and Weil, P. (2012). An introduction to finite automata and their connection to logic, *in* D. D'Souza and P. Shankar (Eds), *Modern Applications of Automata Theory*, World Scientific, Singapore, pp. 3–43.

Sunthornwat, R., Moore, E.J. and Temtanapat, Y. (2011). Detecting and classifying mutations in genetic code with an application to beta-thalassemia, *Science Asia* **37**: 51–61, DOI: 10.2306/scienceasia1513-1874.2011.37.051.

Wahidayat, P.A., Sastroasmoro, S., Fucharoen, S., Setianingsih, I. and Putriasih, S.A. (2018). Applicability of a clinical scoring criteria for disease severity of β-thalassemia/hemoglobin E in Indonesia, *Medical Journal of Indonesia* **27**(1): 26–32, DOI: 10.13181/mji.v27i1.1779.

Zok, T., Badura, J., Swat, S., Figurski, K., Popenda, M. and Antczak, M. (2020). New models and algorithms for RNA pseudoknot order assignment, *International Journal of Applied Mathematics and Computer Science* **30**(2): 315–324, DOI: 10.34768/amcs-2020-0024.

Devi Fitri Ferdania holds a BSc in mathematics from Institut Teknologi Bandung (2020). She is interested in algebra and computer science. Together with Prof. Irawati, she did her first research on the application of algebra in the field of computer science and biology to complete his undergraduate final project. She has also participated in various international competitions, such as the Interdisciplinary Contest in Modeling (ICM, 2020), organized by the Consortium for Mathematics and Its Applications (COMAP), and her team received an honorable mention.

Irawati is a professor at the Algebra Research Group in the Faculty of Mathematics and Natural Sciences at Institut Teknologi Bandung (ITB). In 2006–2014, she was the head of the Algebra Research Group at ITB. Irawati has 39 international publications on various topics in algebra. Her research interests are in ring and module theories.

Hanni Garminia is a lecturer in mathematics. She is a member of the Algebra Research Group in the Faculty of Mathematics and Natural Sciences at Institut Teknologi Bandung (ITB). Garminia has 24 international publications on various topics in algebra. She holds a research interest in module theory.

Akhmaloka is a professor of molecular genetics in Biochemistry Research Group, Faculty of Mathematics and Natural Sciences. Akhmaloka and his group have published more than 80 papers in reputable journals concerning biochemistry and molecular biology of thermophilic microorganisms and thermostable enzymes. His research has been focused on molecular biology of thermostable enzymes since 1995.

Kemal Aziez Rachmansyah obtained his BSc degree in mathematics from Institut Teknologi Bandung in 2020. His undergraduate thesis is about upper bounds for the expansion rate in finite and connected graphs. Mathematical analysis, discrete geometry, and differential geometry are his main research areas. Being in the top 10% of all the participant worldwide, his team won the Meritorious Winner title in the Interdisciplinary Contest in Modeling in 2019.

Received: 3 June 2020
Revised: 2 October 2020
Re-revised: 8 January 2021
Accepted: 7 February 2021