Deterministic finite automata (DFAs) are constructed for various purposes in computational biology. Little attention, however, has been given to the efficient construction of minimal DFAs. In this article, we define simple non-deterministic finite automata (NFAs) and prove that the standard subset construction transforms NFAs of this type into minimal DFAs. Furthermore, we show how simple NFAs can be constructed from two types of patterns popular in bioinformatics, namely (sets of) generalized strings and (generalized) strings with a Hamming neighborhood.

1 Introduction

Deterministic and non-deterministic finite automata belong to the curriculum of every theoretical computer scientist. It is well known that, given a non-deterministic finite automaton (NFA), we can construct a deterministic finite automaton (DFA) recognizing the same language by employing the classical subset construction; each state in the resulting DFA corresponds to a set of NFA states. The details can be found in many textbooks on the topic, for example in [3, 6, 18]. If \( Q \) is an NFA’s finite state space, then there are \( 2^{|Q|} \) subsets and hence the same number of DFA states. In most cases, many of these states turn out to be inaccessible from the start state and can be discarded. In practice, we can use a construction scheme that only generates the accessible states by performing a breadth-first search on the state space [11]. For each DFA, there exists a unique (up to isomorphism) minimal DFA that accepts the same language [6]. Following the subset construction, we may thus want to minimize the resulting DFA, for example by using Hopcroft’s algorithm [2, 5].

In computational biology, the processing of sequences plays a prominent role. Sequences of nucleotides (DNA or RNA) and amino acids (proteins) are key players in the
biology of cells. Recurring elements in such sequences, called patterns or motifs, can often be associated with biological function \[4, 16\]. Three important problem fields in connection with motifs are those of \textit{motif search} \[11\], \textit{motif statistics} \[15, 14, 12, 8, 9\] and \textit{motif discovery} \[20, 7, 17, 10\]. Not surprisingly, in many algorithms in these fields, motifs are transformed into deterministic automata recognizing all possible instances of the motif. Motivated by this observation, we explore the construction of minimal DFAs for two common motif classes, namely (sets of) generalized strings and consensus strings with a Hamming neighborhood. Ultimately, the goal is to find algorithms whose runtime depends linearly on the number of states of the minimal DFA (which would be optimal). Although automata theory has been subject to extensive research for decades, not much attention has been given to this particular topic. Recently in 2008, van Glabbeek and Ploeger \[21\] addressed the problem of determinization and integrated minimization. In Section 3.1, we discuss the connections between their work and this article.

Our contributions We identify a class of NFAs that directly result in minimal DFAs when subjected to the classical subset construction. Although the concept is quite simple and seemingly restrictive, we show that it is strong enough to cover many patterns found in computational biology. To this end, we give construction schemes to transform (sets of) generalized strings and consensus strings into NFAs which exhibit this property.

The article is organized as follows. First, we establish notation by briefly re-stating textbook definitions of automata in Section 2. Then, in Section 3, we introduce the concept of \textit{simple NFA} and show that applying the subset construction to a simple NFA directly yields a minimal DFA. The theory is put to work in Sections 4 and 5, where we discuss the construction of minimal DFAs from generalized strings and consensus strings, respectively.

2 Notation and Basic Definitions

Let \( \Sigma \) be a finite alphabet and let \( \Sigma^k \) be the set of all \textit{strings} of length \( k \). Then, the set of all finite strings \( \bigcup_{i=0}^{\infty} \Sigma^i \) is denoted \( \Sigma^* \) and \( \bigcup_{i=1}^{\infty} \Sigma^i \) is denoted \( \Sigma^+ \). For a string \( s \in \Sigma^* \), its length is written \( |s| \), and \( s_1s_2 \) denotes the concatenation of \( s_1 \) and \( s_2 \). The only string \( \varepsilon \in \Sigma^* \) such that \( |\varepsilon| = 0 \) is called \textit{empty string}. By \( s[i] \), we refer to the \( i \)-th character of \( s \), i.e. \( s = s[1]s[2] \ldots s[|s|] \). Furthermore, \( s[i, j] := s[i]s[i + 1] \ldots s[j] \) refers to a substring of \( s \). If \( i > j \), we define \( s[i, j] := \varepsilon \). Prefixes and suffixes of \( s \) are written \( s[..i] := s[1, i] \) and \( s[i..] := s[i, |s|] \), respectively.

We can extend the notion of a string in a natural way by allowing a \textit{generalized string} to be a sequence of sets of characters:

\textbf{Definition 1} (Generalized string). Given an alphabet \( \Sigma \), we call the set \( \mathcal{G}_\Sigma := 2^\Sigma \setminus \{\emptyset\} \) \textit{generalized alphabet} over \( \Sigma \) and a string over \( \mathcal{G}_\Sigma \) \textit{generalized string}. By \( \mathcal{G}_\Sigma^k \) and \( \mathcal{G}_\Sigma^* \), we refer to the set of all generalized strings of length \( k \) and the set of all generalized strings of finite length, respectively. We say a string \( s \in \Sigma^* \) \textit{matches} the generalized string \( g \in \mathcal{G}_\Sigma^* \), written \( s \prec g \), if \( |s| = |g| \) and \( s[i] \in g[i] \) for \( 1 \leq i \leq |g| \). 

2
We write $G$ instead of $G_Σ$ if the used alphabet is clear from the context. Note that every string $s \in Σ$ can be translated into the generalized string $\{s[1]\}\{s[2]\} \ldots \{s[|s|]\}$. In this sense, strings can be seen as special cases of generalized strings. Let us now proceed to the classical definitions of automata.

**Definition 2** (Deterministic finite automaton (DFA)). A deterministic finite automaton is a tuple $(Q, Σ, δ, q_α, F)$, where $Q$ is a finite set of states, $Σ$ is a finite alphabet, $δ : Q \times Σ → Q$ is a transition function, $q_α \in Q$ is the start state, and $F \subset Q$ is the set of accepting states.

**Definition 3** (Non-deterministic finite automaton (NFA)). A non-deterministic finite automaton is a tuple $(Q, Σ, Δ, Q_α, F)$, where $Q$, $Σ$ and $F$ are defined as for the DFA above, $Δ : Q \times Σ → 2^Q$ is the non-deterministic transition function and $Q_α \subset Q$ is a set of start states.

Note that using a set $Q_α$ instead of only one start state is a notational convenience rather than a conceptual change: we can always transform the automaton to have only one start state by adding the start state $q_α$ and defining its outgoing transitions by $(q_α, σ) \mapsto \bigcup q' \in Q_α Δ(q', σ)$.

Another convenience is the extension of a DFA’s transition function to strings (instead of single characters):

$$\hat{δ} : Q × Σ^* → Q \quad (q, s) \mapsto \begin{cases} q & \text{if } s = ε, \\ \hat{δ}(δ(q, s[1]), s[2..]) & \text{otherwise.} \end{cases}$$

Analogously, the transition function $Δ$ of an NFA can be extended to $\hat{Δ}$. Furthermore, we define $L(q) := \{ s \in Σ^* | \hat{Δ}(q, s) \cap F \neq \emptyset \}$ and call it language of state $q$. The language of a set of states $Q'$ is defined as $L(Q') := \bigcup_{q' \in Q'} L(q')$. Following [1], we call a state $q \in Q$ accessible, if there exist a string $s \in Σ^*$ and a start state $q_α \in Q_α$ such that $\hat{Δ}(q_α, s) = q$. A state $q \in Q$ is called coaccessible if there exist a string $s \in Σ^*$ and an accepting state $q_f \in F$ such that $\hat{Δ}(q, s) = q_f$. Equivalently, $q \in Q$ is coaccessible if $L(q) \cap F \neq \emptyset$. If all states of an automaton are accessible and coaccessible, it is called trim.

Let us briefly review the classical textbook construction of a DFA recognizing the same language as a given NFA.

**Lemma 1** (Subset Construction; Rabin and Scott, [13]). Let $M = (Q, Σ, Δ, Q_α, F)$ be an NFA. Then $(2^Q, Σ, Δ, Q_α, \{Q' \in 2^Q | Q' \cap F \neq \emptyset\})$, with $δ : (Q', σ) → \bigcup_{q' \in Q'} Δ(q', σ)$, is a DFA that recognizes the same language as $M$.

**Proof.** Omitted. See [13] or [6].

As mentioned above, some DFA states may be inaccessible. These states can be removed from the DFA’s state space. To ease notation, we write SUBSETCONSTRUCTION($M$) to denote the DFA resulting from the subset construction and subsequent removal of inaccessible states.
3 Simple NFAs

Recall that our goal is to identify a class of NFAs for which the subset construction yields a minimal DFA; where a DFA is called minimal if there does not exist a DFA with fewer states that recognizes the same language. To this end, we define \textit{simple NFAs}.

\textbf{Definition 4 (Simple non-deterministic finite automaton).} Let an NFA \( M = (Q, \Sigma, \Delta, q_\alpha, F) \) be given. \( M \) is called \textit{simple} if all states are accessible and the languages \( L(q) \) of all states \( q \in Q \) are non-empty and pairwise disjoint.

Therefore, an automaton is simple if and only if it is trim and the languages of all states are pairwise disjoint. Note that an automaton can easily be made trim: If there is a state \( q \) that is not coaccessible, that is, \( L(q) \) is empty, we can safely remove \( q \) from \( Q \) without changing the recognized language. Likewise, all inaccessible states can be removed without changing the recognized language.

\textbf{Theorem 1 (Minimality of DFA constructed from simple NFA).} Let \( M_n = (Q, \Sigma, \Delta, Q_\alpha, F) \) be a simple NFA. Then, the DFA

\[ M_d = (Q \subset 2^Q, \Sigma, \delta, Q_\alpha, F) = \text{SubsetConstruction}(M_n) \]

is minimal.

Before we are able to prove this, we need an auxiliary lemma and the notion of equivalent states in a DFA. We define two states \( p \) and \( q \) of a DFA \( (Q', \Sigma', \delta', q'_\alpha, F') \) to be equivalent if \( \hat{\delta}'(p, s) \in F' \iff \hat{\delta}'(q, s) \in F' \) for all \( s \in \Sigma^* \).

\textbf{Lemma 2.} A DFA is minimal if and only if its states are pairwise non-equivalent.

\textit{Proof.} See Chapters 13 and 15 in \cite{6}.

\textit{Proof of Theorem 1.} Let \( Q', Q'' \in Q \) be two distinct DFA states. By Lemma 2, we have to show that \( Q' \) and \( Q'' \) are not equivalent, or more formally

\[ L(Q') = \bigcup_{q' \in Q'} L(q') \neq \bigcup_{q'' \in Q''} L(q'') = L(Q''). \quad (1) \]

Without loss of generality, assume that \( Q' \setminus Q'' \neq \emptyset \) and let \( q \in Q' \setminus Q'' \). By Definition 4, \( L(q) \cap L(q'') = \emptyset \) for all \( q'' \in Q'' \) and thus \( L(q) \setminus L(Q'') = \emptyset \). But, by choice of \( q \), \( L(q) \subset L(Q') \) and, by Definition 4, \( L(q) \neq \emptyset \). Hence, it follows that \( L(Q') \neq L(Q'') \).

\subsection*{3.1 An Alternative Proof}

We give an alternative proof of Theorem 1 by means of the theory developed in \cite{21}. There, van Glabbeek and Ploeger consider five different variants of the classical subset construction. Each variant is characterized by an operation \( f : 2^Q \to 2^Q \), where \( Q \) is the state space of an NFA. When a new DFA state is produced in the course of the
subset construction, it is subjected to the operation $f$ before being added to the final automaton. In one variant, they define $f$ to be the closure operation $	ext{close}_{\subseteq}: Q' \mapsto \{ q \in Q \mid L(q) \subseteq L(Q') \}$

and show that the subset construction endowed with this operation directly produces minimal DFAs. Theorem 1 now follows from the definition of simple NFAs: As all sets $L(q)$ for $q \in Q$ are pairwise disjoint, $\text{close}_{\subseteq}(Q') = Q'$ for each $Q' \subseteq Q$ and, thus, the classical subset construction yields a minimal DFA.

Note that the language inclusion problem required to be solved for the $\text{close}_{\subseteq}$-operation is in general hard to compute. According to [21], it is PSPACE-complete.

3.2 Self-Transitions of Start States

In most practical settings like pattern search or pattern statistics, we are given a certain type of pattern and need to construct an automaton that accepts all strings with a suffix matching this pattern, rather than an automaton that accepts only the strings that match the pattern. For instance, if our pattern is the single string $\text{ABC}$ and we want to find all occurrences of $\text{ABC}$ in a long text, we need to build an automaton recognizing all strings whose last three letters are $\text{ABC}$. For NFAs, we can easily obtain such an automaton once we have constructed an NFA accepting all strings that match our pattern. All we need to do is to modify the transition function by adding self-transitions to all start states

$$\Delta_{\circledast}: \Delta \leftarrow \begin{cases} \{q\} \cup \Delta(q, \sigma) & \text{if } q \in Q_\alpha, \\
\Delta(q, \sigma) & \text{otherwise}. \end{cases}$$

Throughout this article, the subscript “$\circledast$” refers to this modification of a transition function. The next Lemma characterizes those simple NFAs that remain simple under this modification.

**Lemma 3.** Let $M = (Q, \Sigma, \Delta, Q_\alpha, F)$ be a simple NFA. The modified automaton $M_{\circledast} := (Q, \Sigma, \Delta_{\circledast}, Q_\alpha, F)$ is simple if and only if, in $M$, no start state can be reached from any other state. That means there do not exist $\sigma \in \Sigma, q_\alpha \in Q_\alpha,$ and $q \in Q$ with $q_\alpha \neq q$ such that $q_\alpha \in \Delta(q, \sigma)$.

**Proof.** In this proof, we use the notation $L_{\circledast}(q)$ to refer to the language of the state $q$ with respect to the modified NFA $(Q, \Sigma, \Delta_{\circledast}, Q_\alpha, F)$.

“A$\Rightarrow$”: Suppose $(Q, \Sigma, \Delta_{\circledast}, Q_\alpha, F)$ is simple and there exist $\sigma \in \Sigma, q_\alpha \in Q_\alpha,$ and $q \in Q$ with $q_\alpha \neq q$ such that $q_\alpha \in \Delta(q, \sigma)$. Thus, $\sigma s \in L(q)$ for all $s \in L(q_\alpha)$. Because of the added self-transition, we also have $\sigma s \in L_{\circledast}(q_\alpha)$ and, thus, $L_{\circledast}(q_\alpha)$ and $L_{\circledast}(q)$ are not disjoint, contradicting the assumption that $M_{\circledast}$ is simple.

“A$\Leftarrow$”: Now, we assume that there do not exist any $\sigma \in \Sigma, q_\alpha \in Q_\alpha,$ and $q \in Q$ with $q_\alpha \neq q$ such that $q_\alpha \in \Delta(q, \sigma)$. The properties that all states are accessible and coaccessible cannot get lost by adding the additional self-transitions. Therefore, we only
need to verify that \( \mathcal{L}_\circ(q) \) and \( \mathcal{L}_\circ(q') \) are disjoint for all distinct \( q, q' \in Q \). For the sake of contradiction, we assume there exist distinct \( q, q' \in Q \) violating this condition. We choose \( s \in \mathcal{L}_\circ(q) \cap \mathcal{L}_\circ(q') \) such that \( s \in \mathcal{L}_\circ(q) \setminus \mathcal{L}(q) \); if that is not possible, it becomes possible after swapping \( q \) and \( q' \), because \( \mathcal{L}(p) \subseteq \mathcal{L}_\circ(p) \) for all \( p \in Q \) and \( \mathcal{L}(q) \cap \mathcal{L}(q') = \emptyset \). We have to distinguish two cases:

**Case 1** (\( s \in \mathcal{L}(q') \)): By our assumption, there does not exist a state in \( Q \setminus Q_\alpha \) from which a start state can be reached. This means that the transition function remains unchanged for all states reachable from any state in \( Q \setminus Q_\alpha \), which implies that \( \mathcal{L}(p) = \mathcal{L}_\circ(p) \) for all \( p \in Q \setminus Q_\alpha \). Therefore, \( q \) must be a start state. We chose \( s \) to lie in \( \mathcal{L}_\circ(q) \setminus \mathcal{L}(q) \), which implies that there exists a \( k \in \mathbb{N} \) such that \( s[k..] \in \mathcal{L}(q) \). Since all \( \mathcal{L}(p) \) for \( p \in Q \) are disjoint, it follows that \( s[k..] \notin \mathcal{L}(p) \) for all \( p \in Q \setminus \{q\} \). As \( s \in \mathcal{L}(q') \), we thus conclude that \( \Delta(q', s[.k - 1]) = q' \), which contradicts the assumption that we cannot reach a start state from any other state than itself.

**Case 2** (\( s \notin \mathcal{L}(q') \)): By the same argument as in the last case, we conclude that \( q \) and \( q' \) must be start states. Again, this implies the existence of \( k, k' \in \mathbb{N} \) such that \( s[k..] \in \mathcal{L}(q) \) and \( s[k'..] \in \mathcal{L}(q') \). If \( k = k' \), then \( s[k..] \in \mathcal{L}(q) \cap \mathcal{L}(q') \neq \emptyset \), contradicting the simplicity of \( M \). We assume, without loss of generality, that \( k < k' \). Since \( s[k'..] \in \mathcal{L}(q') \) and \( s[k'..] \notin \mathcal{L}(p) \) for all \( p \in Q \setminus \{q\} \), we conclude that \( \Delta(q, s[k, k' - 1]) = q' \), again contradicting the assumption that we cannot reach a start state from any other state than itself.

\[ \square \]

### 4 Application to Generalized Strings

In the next two sections, we show that generalized strings and sets of generalized strings admit the construction of simple NFAs. Obviously, a single string is a special case of a set of strings. To aid understandability, we nonetheless start with the easier case of one single string.

#### 4.1 Single Generalized Strings

For a generalized string \( g \), an NFA recognizing all strings that match \( g \) can easily be constructed by connecting the state set \( Q = \{0, \ldots, |g|\} \) with the transition function

\[
\Delta : (q, \sigma) \mapsto \begin{cases} 
\{q + 1\} & \text{if } q < |g| \text{ and } \sigma \in g[q + 1], \\
\emptyset & \text{otherwise}.
\end{cases}
\]

Setting \( Q_\alpha = \{0\} \) and \( F = \{|g|\} \) completes the construction of our NFA \((Q, \Sigma, \Delta, Q_\alpha, F)\). For brevity, we write \( \text{NFA}(g) \) to denote the automaton created from a generalized string \( g \) using the above construction.

**Lemma 4.** Let \( g \) be a generalized string. Then \( M_g := \text{NFA}(g) \) is a simple NFA.

**Proof.** Clearly, all states \( i \in Q \) are accessible and coaccessible. \( M_g \) admits only transitions from a state \( i \) to its successor state \( i + 1 \); only the last state in this chain is an
Figure 1: Example of a simple NFA (with self-transition added to the start state) constructed from the generalized string \{A\}{A,B}\{A,C\} over the alphabet \(\Sigma = \{A,B,C\}\). The accepting state is represented by two concentric circles.

accepting state. Thus, for each state \(i \in Q\), the lengths of all accepted strings \(s \in L(i)\) equal \(|g| - i\). Hence, for two different states \(i\) and \(j\), accepted strings have different lengths. Thus, all \(L(i)\) must be pairwise disjoint (for \(i \in Q\)).

As discussed in Section 3.2, we often need to add a self-transition to the start state. This modification is defined formally in Equation (2). We write \(\mathsf{NFA}_\gamma(g)\) to refer to the resulting automaton. See Figure 1 for an example. Combining Theorem 1, Lemma 4, and Lemma 3, we arrive at the following corollary:

**Corollary 1.** Let \(g\) be a generalized string and \(M_g := \mathsf{NFA}_\gamma(g)\) the corresponding NFA. Then, \(\mathsf{SubsetConstruction}(M_g)\) is a minimal DFA.

#### 4.2 Sets of Generalized Strings

In this section, we generalize the above results to finite sets of generalized strings of equal length. Speaking formally, we assume a length \(\ell\) and \(G \subseteq G^\ell\) to be given and seek to construct a simple NFA that recognizes all strings that have a suffix matching a \(g \in G\). As above, we first construct an automaton that recognizes all strings matching a \(g \in G\) and, in a second step, add self-transitions to the start states \(Q_\alpha\).

The automaton we build is organized level-wise with \(\ell + 1\) levels. Transitions are only possible between states in adjacent levels and only in one direction (which we choose to call *downwards*). The bottom level contains just one state which is the single accepting state; all states in the top level are start states. As before for a single generalized string, two states \(q'\) and \(q''\) in different levels are obviously "language-disjoint", meaning that \(L(q') \cap L(q'') = \emptyset\). But here, we possibly need more than one state in a level, which entails the problem of ensuring language-disjointness for states in the same level. We achieve this by using a state space induced by a special parent-child relation between states in adjacent levels. Before we formally construct state space and automaton, the impatient reader may have a look at the example in Figure 2.

Let us begin with the formal specification of a suitable state space \(Q\). We choose \(Q\) to be a special subset of \(\bar{Q} := 2^G \times \{0, \ldots, \ell\}\) with the following semantics in mind: to be in state \(q = (H, k)\) means that the last \(k\) characters read match the first \(k\) positions of a \(g \in H\). For the definition of \(Q\), we need the function \(\mathsf{Parent}: \bar{Q} \times \Sigma \to \bar{Q} \cup \{\bot\}\).
given by

\[
\text{Parent} : \left( (H, k), \sigma \right) \mapsto \begin{cases} 
\{ h \in H \mid \sigma \in h[k] \}, & k > 0, \\
\bot, & \text{otherwise} \end{cases} \quad (3)
\]

We say that \text{Parent}(q, \sigma) is a parent of \( q \) under the character \( \sigma \). The special symbol \( \bot \) is used to indicate that a state is in the top level and therefore does not have any parents. The \text{Parent} mapping induces a hierarchy of \( \ell + 1 \) levels of states:

\[
Q_\ell := \{(G, \ell)\},
\]

\[
Q_i := \left\{ (H, i), H \in 2^G \setminus \{\emptyset\} \mid \exists q \in Q_{i+1}, \sigma \in \Sigma : \text{Parent}(q, \sigma) = (H, i) \right\},
\]

for \( 0 \leq i < \ell \). Finally, we write our state space as

\[
Q := Q_0 \cup \ldots \cup Q_\ell.
\]
The Parent mapping also induces a transition function $\Delta$:

$$\Delta : ((H, k), \sigma) \mapsto \begin{cases} \{ q \in Q_{k+1} | \text{PARENT}(q, \sigma) = (H, k) \} & \text{if } k < \ell, \\
\emptyset & \text{otherwise}. \end{cases} \quad (7)$$

To complete the construction, we set $Q_\alpha := Q_0$ and $F := Q_\ell = \{(G, \ell)\}$ and obtain $\text{NFA}(G) := (Q, \Sigma, \Delta, Q_\alpha, F)$. The next lemma states that an NFA constructed in this way accepts exactly the language given by $G$.

**Lemma 5.** Let a length $\ell \in \mathbb{N}$, a set of generalized strings $G \subset \mathcal{G}^\ell$, and $(Q, \Sigma, \Delta, Q_\alpha, F) = \text{NFA}(G)$ be given. Then,

$$\exists q \in Q_\alpha : \hat{\Delta}(q, s) \cap F \neq \emptyset \iff \exists g \in G : s \prec g,$$

for all $s \in \Sigma^*$. 

**Proof.** We start with the forward direction “$\Rightarrow$”. If $s \in \Sigma^*$ is accepted by $\text{NFA}(G)$, then there exists a sequence of states $q_0, \ldots, q_\ell$ such that $q_0 \in Q_\alpha, q_\ell \in F$, and $q_i \in \Delta(q_{i-1}, s[i])$ for $0 < i \leq |s|$. It follows from Equation $7$ that $q_{\ell-1} = \text{PARENT}(q_i, s[i])$. Hence, Equation $5$ implies that $H_0 \subseteq \ldots \subseteq H_\ell$, where $(H_i, k_i) := q_i$. Furthermore, by Equation $5$, $H_0$ is non-empty. Inductively applying $5$ now yields that $s \prec h$ for all $h \in H_0$, which proves the forward direction.

Let us prove the backward direction “$\Leftarrow$”. Let $g \in G$, such that $s \prec g$. Consider the sequence of states $q_0', \ldots, q_\ell'$ with $(H_0', k_0') := q_0'$ given by $q_0' := (G, \ell)$ and $q_i' := \text{PARENT}(q_i', s[i])$ for $0 < i \leq |s|$. From $s \prec g$ and Equation $6$ it follows that $g \in H_i'$ for $0 \leq i \leq |s|$. Thus, each $H_i'$ is non-empty and by Equations $4$ and $5$ we get $q_i' \in Q_i$ for $0 \leq i \leq |s|$, implying that $q_0' \in Q_0 = Q_\alpha$ is a start state. From Equation $7$ we conclude that $\hat{\Delta}(q_0', s) = q_\ell'$ which proves the claim as $q_\ell' \in Q_\ell = F$. 

In analogy to Lemma $4$, we verify that $\text{NFA}(G)$ is indeed a simple NFA.

**Lemma 6.** Let $\ell \in \mathbb{N}$ and $G \subset \mathcal{G}^\ell$. Then, $M_G := \text{NFA}(G)$ is a simple NFA.

**Proof.** The level-wise construction directly implies that all states are accessible and coaccessible, i.e. $L(q)$ is non-empty for all $q \in Q$. States with empty $L(q)$ cannot be generated by Equation $5$.

It remains to be shown that for all distinct $p, q \in Q$ the sets $L(p)$ and $L(q)$ are disjoint. By construction, this is clearly true if $p$ and $q$ are in different levels. Hence, it suffices to show that

$$L(p) \cap L(q) = \emptyset$$

for all $Q_i$ with $0 \leq i \leq \ell$. We prove this by induction on $i$. First, note that for $i = \ell$, Condition $8$ is fulfilled as $|Q_\ell| = 1$. Assume that $8$ holds for $i > 0$. For the sake of contradiction, we further assume there exist distinct $p, q \in Q_{i-1}$, such that $L(p) \cap L(q) \neq \emptyset$. Let $s \in L(p) \cap L(q)$; it follows that $\hat{\Delta}(p, s) \in F$. There must exist a state $r \in Q_i$ such that $\hat{\Delta}(r, s[2..]) \in F$. As, by our induction hypothesis, Condition $8$ holds for $i$, we conclude that the state $r$ is unique. It follows from $7$ that $r \in \Delta(p, s[1])$ and $r \in \Delta(q, s[1])$. Applying the definition of $\Delta$, we get $p = \text{PARENT}(r, s[1]) = q$ and, thus, $p = q$. 

9
In Section 4.1, we added an initial self-transition to the constructed NFA in order to accept not only the given generalized string, but all strings whose suffix matches the generalized string. We thereby obtained an automaton that finds all occurrences of the generalized string in a given text. Now we repeat this step by transforming $\text{NFA}(G)$ using Equation (2). Again, we refer to the resulting modified automaton by $\text{NFA}_\supset(G)$. Note that for $|G| = 1$ we obtain the same automaton as constructed in Section 4.1.

Combining Theorem 1, Lemma 6, and Lemma 3 yields the following corollary:

**Corollary 2.** Let $\ell \in \mathbb{N}$, $G \subset G^\ell$, and $M_G := \text{NFA}_\supset(G)$.

Then, the result of $\text{SUBSETCONSTRUCTION}(M_G)$ is a minimal DFA.

### 4.2.1 Algorithm and Runtime

The construction scheme formalized in Equations (4) and (5) can directly be translated into an algorithm:

1. Initialize transition map $\Delta$ to be empty.
2. Initialize the bottom level $Q_\ell$ to contain its only state $(G, \ell)$.
3. For $k$ from $\ell - 1$ down to 0, build level $Q_k$:
   a) Initialize level $Q_k$ to be empty.
   b) For each node $(H', k + 1) \in Q_{k+1}$ and each $\sigma \in \Sigma$
      i. Compute the set $H := \{h \in H' \mid \sigma \in h[k+1]\}$.
      ii. If $H \neq \emptyset$ and $(H, k) \notin Q_k$, add $(H, k)$ to $Q_k$.
      iii. Add transition $((H, k), \sigma) \mapsto (H', k + 1)$ to $\Delta$.
4. Add self-transitions to all $q \in Q_0$.

In Loop 3, we build $\ell$ levels. Each level contains at most $2^{|G|}$ states and thus the body of Loop 3 is executed $O(2^{|G|} \cdot |\Sigma|)$ times for each level, where Step 3(b) takes $O(|G|)$ time and the other steps can be performed in constant time. All in all, the algorithm takes $O(2^{|G|} \cdot \ell \cdot |\Sigma| \cdot |G|)$ time.

The construction of a minimal DFA from a set of generalized strings thus takes $O(2^{|G|} \cdot \ell \cdot |\Sigma| \cdot |G| + m)$ time, where $m$ is the number of states in the minimal DFA.

### 5 Application to Consensus Strings with a Hamming Neighborhood

Another type of motif commonly used in computational biology is a consensus string along with a distance threshold. Here, we assume that a (generalized) string $s$ and a distance threshold $d_{max}$ are given and want to compute the minimal DFA that recognizes
Figure 3: Example of a simple NFA over the alphabet $\Sigma = \{A, B, C, D\}$ recognizing the consensus ADC and all strings within a Hamming distance of two or less. Characters with bars stand for the inverse, e.g. $\overline{A}$ stands for $B$, $C$, or $D$. The accepting state is represented by two concentric circles.

all strings with a Hamming distance to $s$ of at most $d_{\text{max}}$, where the Hamming distance between a string $s$ and a generalized string $g$ of same length is defined as

$$d(s, g) := \left| \{ i \in \{1, \ldots, |s|\} \mid s[i] \notin g[i] \} \right|.$$  

In this section, we construct a simple NFA recognizing a generalized string and its Hamming neighborhood. The construction is similar to the one given in [11]. Interestingly, the resulting NFA turns out to be simple.

The basic idea for the construction is to use a two-dimensional grid as a state space, where we advance into one dimension whenever a valid character has been read and into the other dimension for each mismatch. Figure 3 illustrates an NFA built in this way. Formally the state space is defined by

$$Q := \left\{ (e, k) \in \{0, \ldots, d_{\text{max}}\} \times \{0, \ldots, |g|\} \mid |g| - k - e \geq 0 \right\}.$$  

(9)
with the following semantics: state \((e, k)\) accepts all strings of length \(|g| - k\) that match the respective suffix of \(g\) with exactly \(e\) errors. The condition \(|g| - k - e \geq 0\) states that the number of errors \(e\) cannot be larger than \(|g| - k\), which is the number of characters left. We define the transition function to obey this semantics:

\[
\Delta : (e, k) \times \sigma \mapsto \begin{cases} 
z(e, k + 1) & \text{if } \sigma \in g[k + 1], \\
z(e - 1, k + 1) & \text{otherwise}, \end{cases}
\]

(10)

where the function \(z : \mathbb{Z} \times \mathbb{Z} \rightarrow 2^Q\) returns the empty set whenever we “fall off the grid”. More precisely,

\[
z : (e, k) \mapsto \begin{cases} \{(e, k)\} & \text{if } (e, k) \in Q, \\
\emptyset & \text{otherwise}. \end{cases}
\]

(11)

As before, the topmost level constitutes the start states, i.e. \(Q_\alpha := \{(e, k) \in Q \mid k = 0\}\), and the bottommost level contains only the single accepting state, i.e. \(F := \{(0, |g|)\}\). We write \(\text{NFA}(g, d_{\text{max}}) := (Q, \Sigma, \Delta, Q_\alpha, F)\) to denote the NFA constructed in this way. Again, we use the notation \(\text{NFA}(g, d_{\text{max}}) := (Q, \Sigma, \Delta_c, Q_\alpha, F)\) to refer to the automaton with self-transitions added to the start states. Note that for \(d_{\text{max}} = 0\), the resulting automaton is isomorphic to the one constructed from a single generalized string in Section 4.1.

In order to prove that the construction is correct and produces simple NFAs, we use the following Lemma on the state’s languages.

**Lemma 7.** Let \(g \in G_\Sigma\), \(d_{\text{max}} \in \mathbb{N}_0\) and \(M = \text{NFA}(g, d_{\text{max}}) = (Q, \Sigma, \Delta, Q_\alpha, F)\). Then, the language of state \((e, k)\) is characterized by

\[
\mathcal{L}((e, k)) = \left\{ s \in \Sigma^{|g| - k} \mid d(s, g[k + 1..]) = e \right\},
\]

for all \((e, k) \in Q\).

**Proof.** We start with the direction “\(\subseteq\)”. By construction of \(\Delta\) and \(F\), we have \(\mathcal{L}((e, k)) \subseteq \Sigma^{|g| - k}\). Let \(s \in \mathcal{L}((e, k))\), then \(\tilde{\Delta}((e, k), s) = (0, |g|)\). That means, in the course of \(|s|\) state transitions the first component of the state changes from \(e\) to 0. As we see from Equation (10), the only change possible in the first component is a decrease by 1, which happens if and only if the read character is a mismatch. Thus, it follows that \(d(s, g[k + 1..]) = e\).

Now we prove the backward direction “\(\supseteq\)”. Let \(s \in \Sigma^{|g| - k}\) and \(d(s, g[k + 1..]) = e\). That means there are exactly \(e\) indices \(a_1, \ldots, a_e\) such that \(s[a_i] \notin g[k + a_i]\) for \(1 \leq i \leq e\). Provided that all states exist and thus the \(z\) function never returns \(\emptyset\), we apply the first case of (10) exactly \(|s| - e\) times and the second case exactly \(e\) times, ending in state \((0, |g|)\) as claimed. The only thing left to verify is that \(z\) indeed never returns \(\emptyset\). Note that, by (10), the term \(|g| - k - e\) cannot increase. Since it reaches zero after \(|s|\) steps, it cannot have been smaller than zero at any time. Hence, by Equation (10), all intermediate states exist and, thus, the first case of Equation (11) is applied for all state transitions. \(\square\)
Using this lemma, the construction’s correctness is easily verified:

**Lemma 8.** Let \( g \in G_{\Sigma}^* \), \( d_{\text{max}} \in \mathbb{N}_0 \) and \( M = \text{NFA}(g, d_{\text{max}}) = (Q, \Sigma, \Delta, Q_\alpha, F) \). Then, \( M \) accepts exactly the strings \( \{ s \in \Sigma^{|g|} \mid d(s, g) \leq d_{\text{max}} \} \).

**Proof.** By definition, \( M \) accepts the strings \( \mathcal{L}(Q_\alpha) \). By construction of \( Q_\alpha \) and Lemma 7, we obtain
\[
\mathcal{L}(Q_\alpha) = \bigcup_{e=0}^{\min(d_{\text{max}}, |g|)} \mathcal{L}((e, 0)) = \bigcup_{e=0}^{\min(d_{\text{max}}, |g|)} \{ s \in \Sigma^{|g|} \mid d(s, g) = e \}
\]

**Lemma 9.** Let \( g \in G_{\Sigma}^* \), \( d_{\text{max}} \in \mathbb{N}_0 \). Then, \( \text{NFA}(g, d_{\text{max}}) = (Q, \Sigma, \Delta, Q_\alpha, F) \), is a simple NFA.

**Proof.** By construction, all states are accessible and coaccessible. The disjointness of \( \mathcal{L}(q) \) and \( \mathcal{L}(q') \) for distinct \( q, q' \in Q \) follows immediately from Lemma 7.

In analogy to Sections 4.1 and 4.2, we can now add self-transitions to the start states to obtain \( \text{NFA}_\bowtie(g, d_{\text{max}}) \). Note that, again, the conditions of Lemma 3 are satisfied, allowing us to apply Theorem 1.

**Corollary 3.** Let \( g \in G_{\Sigma}^* \), \( d_{\text{max}} \in \mathbb{N}_0 \), and \( M = \text{NFA}_\bowtie(g, d_{\text{max}}) \). Then, the result of \( \text{SubsetConstruction}(M) \) is a minimal DFA.

The state space of \( \text{NFA}_\bowtie(g, d_{\text{max}}) \) has a size of \( O(|g| \cdot d_{\text{max}}) \). Deriving a construction algorithm that uses \( O(1) \) time per state is straightforward. We can, therefore, construct the minimal DFA from a generalized string \( g \) and the distance threshold \( d_{\text{max}} \) in time \( O(|g| \cdot d_{\text{max}} + m) \), where \( m \) is the size of the minimal DFA.

### 6 Conclusions

We introduced the concept of simple NFAs. These automata have a useful property: when subjected to the standard subset construction, they result in minimal DFAs. Motivated by a background in bioinformatics, we turned our attention to pattern classes found in this field. We gave an algorithm to construct a simple NFA from a set \( G \) of generalized strings of equal length \( \ell \) in time \( O(2^{|G|} \cdot \ell \cdot |\Sigma| \cdot |G|) \). Interestingly, this result suggests that the difficulty in dealing with sets of generalized strings stems from the size of the set rather than from the length of the strings. For motifs given in the form of a single (generalized) string \( g \) along with a Hamming neighborhood bounded by a distance threshold \( d_{\text{max}} \), we presented an algorithm that constructs a simple NFA in time \( O(|g| \cdot d_{\text{max}}) \). A third important class of motifs are position weight matrices (PWMs) with a score threshold \[19\]. Such a motif could be transformed into a set of generalized strings, which in turn could be handled by the presented algorithm. Nonetheless, a
more direct method to construct a simple NFA from a PWM is desirable and should be subject of future research.

In this article, we demonstrated that, for the considered pattern classes, a minimal DFA can be constructed directly, that is, without the intermediate step of a non-minimal DFA. A question we did not address, regards the size of the constructed minimal automata. In practice, we might still be faced with an exponential blow-up in the number of states. Thus, on the practical side, this study should be complemented by experiments measuring automata sizes and runtimes for typical motifs in future work.

7 Acknowledgments

I wish to thank Sven Rahmann for giving valuable comments on an earlier version of this manuscript, Wim Martens for pointing me to the paper by van Glabbeek and Ploeger [21], Marcel Martin and Chris Schwiegelshohn for proof-reading, and Hoi-Ming Wong, who assisted in finding a graph layout with few edge crossings for Figure 2.

References

[1] J. Berstel and D. Perrin. Theory of Codes. Academic Press, 1985.
[2] J. E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi and A. Paz, editors, The theory of machines and computations, pages 189–196. Academic Press, New York, 1971.
[3] J. E. Hopcroft and J. D. Ullman. Introduction to automata theory, languages, and computation. Addison-Wesley, 1979.
[4] N. Hulo, A. Bairoch, V. Bulliard, L. Cerutti, E. De Castro, P. Langendijk-Genevaux, M. Pagni, and C. Sigrist. The PROSITE database. Nucleic Acids Research, 34(S1):D227–230, 2006.
[5] T. Knuutila. Re-describing an algorithm by Hopcroft. Theoretical Computer Science, 250(1-2):333–363, January 2001.
[6] D. C. Kozen. Automata and Computability (Undergraduate Texts in Computer Science). Springer, December 1999.
[7] N. Li and M. Tompa. Analysis of computational approaches for motif discovery. Algorithms for Molecular Biology, 1(1):8, 2006.
[8] M. Lladser, M. D. Betterton, and R. Knight. Multiple pattern matching: A Markov chain approach. Journal of Mathematical Biology, 56(1-2):51–92, 2008.
[9] T. Marschall and S. Rahmann. Probabilistic arithmetic automata and their application to pattern matching statistics. In P. Ferragina and G. M. Landau, editors,
T. Marschall and S. Rahmann. Efficient exact motif discovery. Bioinformatics (Proceedings of ISMB), 25:i356–364, 2009.

G. Navarro and M. Raffinot. Flexible Pattern Matching in Strings. Cambridge University Press, June 2002.

P. Nicodème, B. Salvy, and P. Flajolet. Motif statistics. Theoretical Computer Science, 287:593–617, 2002.

M. O. Rabin and D. Scott. Finite automata and their decision problems. IBM Journal of Research and Development, 3(2):114–125, 1959.

M. Régnier. A unified approach to word occurrence probabilities. Discrete Applied Mathematics, 104:259–280, 2000.

G. Reinert, S. Schbath, and M. S. Waterman. Probabilistic and statistical properties of words: An overview. Journal of Computational Biology, 7(1-2):1–46, 2000.

A. Sandelin, W. Alkema, P. G. Engström, W. W. Wasserman, and B. Lenhard. JASPAR: an open access database for eukaryotic transcription factor binding profiles. Nucleic Acids Research, 32(1):Database Issue, 2004.

G. Sandve and F. Drabløs. A survey of motif discovery methods in an integrated framework. Biology Direct, 1(1):11, 2006.

M. Sipser. Introduction to the Theory of Computation, Second Edition. Course Technology, February 2005.

R. Staden. Computer methods to locate signals in nucleic acid sequences. Nucleic Acids Research, 12(1):505–519, 1984.

M. Tompa, N. Li, T. Bailey, G. Church, B. De Moor, E. Eskin, A. Favorov, M. Frith, Y. Fu, W. Kent, V. Makeev, A. Mironov, W. Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu. Assessing computational tools for the discovery of transcription factor binding sites. Nature Biotechnology, 23:137–144, 2005.

R. van Glabbeek and B. Ploeger. Five determinisation algorithms. Proceedings of the 13th Conference on Implementation and Applications of Automata (CIAA), LNCS 5148, pages 161–170, 2008.