On the bit-parallel simulation of the nondeterministic Aho-Corasick and suffix automata for a set of patterns

Domenico Cantone, Simone Faro, Emanuele Giaquinta

Università di Catania, Dipartimento di Matematica e Informatica
Viale Andrea Doria 6, I-95125 Catania, Italy

Abstract

In this paper we present a method to simulate, using the bit-parallelism technique, the nondeterministic Aho-Corasick automaton and the nondeterministic suffix automaton induced by the trie and by the Directed Acyclic Word Graph for a set of patterns, respectively. When the prefix redundancy is nonnegligible, this method yields–if compared to the original bit-parallel encoding with no prefix factorization–a representation that requires smaller bit-vectors and, correspondingly, less words. In particular, if we restrict to single-word bit-vectors, more patterns can be packed into a word.

We also present two simple algorithms, based on such a technique, for searching a set $\mathcal{P}$ of patterns in a text T of length n over an alphabet Σ of size σ. Our algorithms, named Log-And and Backward-Log-And, require $\mathcal{O}((m+\sigma)\lceil m/w\rceil)$-space, and work in $\mathcal{O}(n\lceil m/w\rceil)$ and $\mathcal{O}(n\lceil m/w\rceil l_{min})$ worst-case searching time, respectively, where w is the number of bits in a computer word, m is the number of states of the automaton, and l_{min} is the length of the shortest pattern in $\mathcal{P}$.

1. Introduction

Given a set $\mathcal{P}$ of r patterns and a text T of length n, all strings over a common finite alphabet Σ of size σ, the *multiple pattern matching problem* is to determine all the occurrences in T of the patterns in $\mathcal{P}$. In this paper we focus on automata based solutions of such problem and, in particular, on the efficient simulation of the nondeterministic finite automaton (NFA) for the language $\bigcup_{P\in\mathcal{P}} \Sigma^* P$ induced by the *trie* data structure for $\mathcal{P}$ and the nondeterministic automaton for the language $\bigcup_{P\in\mathcal{P}} \mathit{Suff}(P)$ of all the suffixes of the strings in $\mathcal{P}$ induced by the Directed Acyclic Word Graph (DAWG) data structure for $\mathcal{P}$. We shall refer to such two automata as Aho-Corasick NFA and suffix NFA, respectively.

Email addresses: cantone@dmi.unict.it (Domenico Cantone), faro@dmi.unict.it (Simone Faro), giaquinta@dmi.unict.it (Emanuele Giaquinta)

The first linear solution for the multiple pattern matching problem based on finite automata is due to Aho and Corasick in [1]. The Aho-Corasick algorithm uses a deterministic incomplete finite automaton based on the *trie* for the input patterns and on the *failure function*, a generalization of the border function of the Knuth-Morris-Pratt algorithm [9]. The optimal average complexity of the problem is $\mathcal{O}(n \log_\sigma(rl_{min})/l_{min})$ [11], where l_{min} is the length of the shortest pattern in the set $\mathcal{P}$; this bound has been achieved by algorithms based on the suffix automaton induced by the DAWG data structure, namely the Backward-DAWG-Matching (BDM) and Set-Backward-DAWG-Matching (SBDM) algorithms [7, 10]. Later, Baeza-Yates and Gonnet introduced in [3] the bit-parallelism technique to simulate efficiently simple nondeterministic finite automata (NFAs, for short) for the single pattern case. Their Shift-And algorithm is one of the most efficient and elegant simulation of this kind of NFAs. Navarro and Raffinot used this technique to simulate the BDM algorithm; specifically, their algorithm, named Backward-Nondeterministic-DAWG-Matching (BNDM), is based on a bit-parallel encoding of the nondeterministic suffix automaton induced by the DAWG of a single pattern [12].

In the bit-parallel simulation, the automaton current configuration is represented as an array of ℓ bits, where ℓ is the number of states in the automaton. Bits corresponding to active states are set to `1`, whereas bits corresponding to inactive states are set to `0`. Such representation allows one to take advantage of the intrinsic parallelism of the bit operations inside a computer word, thus cutting down the number of operations up to a factor equal to the number of bits in a computer word.

However, to simulate efficiently an NFA with the bit-parallelism technique, the states of the automaton must be mapped into the positions of a bit-vector by a suitable topological ordering of the NFA.[1] There are known bit-parallel simulations for the trie of a single pattern and for the maximal trie of a set of patterns. In the case of a single pattern, the construction of the topological ordering is quite simple, since it is unique [3]. Appropriate topological orderings can be obtained also for the maximal trie of a set of patterns, by interleaving the tries of the single patterns in either a parallel fashion, under the restriction that all the patterns have the same length [14], or in a sequential fashion [12]. The Shift-And and BNDM algorithms can be easily extended to the multiple patterns case by deriving the corresponding automaton from the maximal trie of the set of patterns. The resulting algorithms have a $\mathcal{O}(\sigma\lceil size(\mathcal{P})/w\rceil)$-space complexity and work in $\mathcal{O}(n\lceil size(\mathcal{P})/w\rceil)$ and $\mathcal{O}(n\lceil size(\mathcal{P})/w\rceil l_{min})$ worst-case searching time complexity, respectively, where $size(\mathcal{P}) = \sum_{P\in\mathcal{P}} |P|$ is the sum of the lengths of the strings in $\mathcal{P}$ and w is the size of a computer word.

In both cases, the bit-parallel simulation is based on the following property of the topological ordering π associated to the trie which allows to encode the transitions using a shift of k bits and a bitwise `and`: for each edge (p, q), the

[1] We recall that a *topological ordering* of an NFA is any total ordering $<$ of the set of its states such that $p < q$, for each edge (p, q) of the NFA.

distance $\pi(q) - \pi(p)$ is equal to a constant k. For an in-depth survey on the topic, the reader is referred to [6].

The problem which arises when trying to bit-parallel simulate the Aho-Corasick NFA and the suffix NFA is that, in general, there might be no topological ordering π such that, for each edge (p, q), the distance $\pi(q) - \pi(p)$ is fixed. Cantone and Faro presented in [6] a bit-parallel simulation of the Aho-Corasick NFA that encodes variable length shifts using the carry property of addition and based on a particular topological ordering; however, such topological orderings do not always exist. Their algorithm has a $\mathcal{O}(\sigma\lceil m/w\rceil)$-space and $\mathcal{O}(n\lceil m/w\rceil)$-searching time complexity, where m is the number of nodes in the trie.

As explained above, the current technique used to extend string matching algorithms based on bit-parallelism to the multiple-string matching problem consists, on a conceptual basis, in sequentially concatenating the automata for each pattern. The drawback of this method is that it is not possible to exploit the prefix redundancy in the patterns, a property which can be significant in the case of small alphabets. The trie and the DAWG data structures make it possible to factor common prefixes in the patterns. However, because of the lack of regularity in such structures, it is not possible to devise a simulation of the corresponding automata using the original bit-parallel encoding. In this paper we present a new more general approach to the efficient bit-parallel simulation of the Aho-Corasick NFAs and suffix NFAs. When the prefix redundancy is nonnegligible, this method yields–if compared to the original encoding with no prefix factorization–a representation that requires smaller bit-vectors and, correspondingly, less words. Therefore, if we restrict to single-word bit-vectors, it results that more patterns can be packed into a word. Our construction is based on a result for the Glushkov automaton [13], which however requires exponential space in the number of states in the NFA to encode the transition function. We show that, by exploiting the relation between active states of the NFA and its associated failure function, it is possible to represent the transition function in polynomial space using a similar encoding.

The rest of the paper is organized as follows. In Section 2 we recall some preliminary notions and elementary facts. In Section 3 we present a general technique to simulate NFAs for a set of patterns. Then in Sections 4 and 5 we devise a bit-parallel encoding of the Aho-Corasick NFA and of the suffix NFA, respectively, and describe also two bit-parallel algorithms for the *multiple pattern matching problem* based on such encodings. Finally, we briefly draw our conclusions in Section 6.

2. Basic notions and definitions

A string P of length $|P| = m$ over a given finite alphabet Σ is any sequence of m characters of Σ. For $m = 0$, we obtain the empty string ε. Σ^* is the collection of all finite strings over Σ. We denote by $P[i]$ the $(i+1)$-st character of P, for $0 \leq i < m$. Likewise, the substring of P contained between the $(i+1)$-st and the $(j+1)$-st characters of P is denoted by $P[i\,..\,j]$, for $0 \leq i \leq j < m$.

We also put $P_i =_{\text{Def}} P[0\,..\,i]$, for $0 \leq i < m$, and make the convention that P_{-1} denotes the empty string ε. It is common to identify a string of length 1 with the character occurring in it. For any two strings P and P', we write $P.P'$ to denote the concatenation of P' to P, and $P' \sqsupset P$ to express that P' is a *proper* suffix of P, i.e., $P = P''.P'$ for some nonempty string P''. The notation $P' \sqsupseteq P$ will be used with the obvious meaning. Analogously, $P' \sqsubseteq P$ ($P' \sqsubset P$) expresses that P' is a (proper) prefix of P, i.e., $P = P'.P''$ for some (nonempty) string P''. We say that P' is a factor of P if $P = P''.P'.P'''$, for some strings $P'', P''' \in \Sigma^*$, and we denote by $\mathit{Fact}(P)$ the set of the factors of P. Likewise, we denote by $\mathit{Suff}(P)$ the set of the suffixes of P. We write P^{r} to denote the reverse of the string P, i.e., $P^{\mathsf{r}} = P[m-1]P[m-2]\ldots P[0]$. Given a finite set of patterns $\mathcal{P}$, we put $\mathcal{P}^{\mathsf{r}} =_{\text{Def}} \{P^{\mathsf{r}} \mid P \in \mathcal{P}\}$ and $\mathcal{P}_l =_{\text{Def}} \{P[0\,..\,l-1] \mid P \in \mathcal{P}\}$. Also we put $\mathit{size}(\mathcal{P}) =_{\text{Def}} \sum_{P\in\mathcal{P}} |P|$ and extend the maps $\mathit{Fact}(\cdot)$ and $\mathit{Suff}(\cdot)$ to $\mathcal{P}$ by putting $\mathit{Fact}(\mathcal{P}) =_{\text{Def}} \bigcup_{P\in\mathcal{P}} \mathit{Fact}(P)$ and $\mathit{Suff}(\mathcal{P}) =_{\text{Def}} \bigcup_{P\in\mathcal{P}} \mathit{Suff}(P)$.

We recall the notation of some bitwise infix operators on computer words, namely the bitwise `and` "&", the bitwise `or` "|", the `left shift` "$\ll$" operator (which shifts to the left its first argument by a number of bits equal to its second argument), and the unary bitwise `not` operator "$\sim$". The functions that compute the first and the last bit set to `1` of a word x are $\lfloor \log_2(x \,\&\, (\sim x + 1)) \rfloor$ and $\lfloor \log_2(x) \rfloor$, respectively. [2]

A nondeterministic finite automaton (NFA) with ε-transitions is a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$, where Q is a set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the collection of final states, Σ is an alphabet, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathscr{P}(Q)$ is the transition function ($\mathscr{P}(\cdot)$ is the powerset operator).[3] For each state $q \in Q$, the ε-closure of q, denoted as ECLOSE(q), is the set of states that are reachable from q by following zero or more ε-transitions. ECLOSE can be generalized to a set of states by putting $\text{ECLOSE}(D) = \bigcup_{q\in D} \text{ECLOSE}(q)$. In the case of an NFA without ε-transitions, we have $\text{ECLOSE}(q) = \{q\}$, for any $q \in Q$.

The *extended* transition function $\delta^* : Q \times \Sigma^* \to \mathscr{P}(Q)$ induced by δ is defined recursively by

$$\delta^*(q,u) =_{Def} \begin{cases} \bigcup_{p\in\delta^*(q,v)} \text{ECLOSE}(\delta(p,c)) & \text{if } u = v.c, \text{ for some } v \in \Sigma^* \\ & \quad \text{and } c \in \Sigma, \\ \text{ECLOSE}(q) & \text{otherwise (i.e., if } u = \varepsilon). \end{cases}$$

In particular, when no ε-transition is present, then

$$\delta^*(q,\varepsilon) = \{q\} \quad \text{and} \quad \delta^*(q, v.c) = \bigcup_{p\in\delta^*(q,v)} \delta(p,c)\,.$$

[2] Modern architectures include assembly instructions for this purpose; for example, the *x86* family provides the `bsf` and `bsr` instructions, whereas the *powerpc* architecture provides the `cntlzw` instruction. For a comprehensive list of machine-independent methods for computing the index of the first and last bit set to `1`, see [2].

[3] In the case of NFAs with no ε-transitions, the transition function has the form $\delta : Q \times \Sigma \to \mathscr{P}(Q)$. For the basics on NFAs, the reader is referred to [8].

Both the transition function δ and the extended transition function δ^* can be naturally generalized to handle set of states, by putting $\delta(D, c) =_{Def} \bigcup_{q \in D} \delta(q, c)$ and $\delta^*(D, u) =_{Def} \bigcup_{q \in D} \delta^*(q, u)$, respectively, for $D \subseteq Q$, $c \in \Sigma$, and $u \in \Sigma^*$. The *extended* transition function satisfies the following property:

$$\delta^*(q, u.v) = \delta^*(\delta^*(q, u), v), \text{ for all } u, v \in \Sigma^* . \tag{1}$$

Given a set $\mathcal{P}$ of patterns over a finite alphabet Σ, the *trie* $\mathcal{T}_{\mathcal{P}}$ associated with $\mathcal{P}$ is a rooted directed tree, whose edges are labeled by single characters of Σ, such that

(i) distinct edges out of the same node are labeled by distinct characters,

(ii) all paths in $\mathcal{T}_{\mathcal{P}}$ from the root are labeled by prefixes of the strings in $\mathcal{P}$,

(iii) for each string P in $\mathcal{P}$ there exists a path in $\mathcal{T}_{\mathcal{P}}$ from the root which is labeled by P.

For any node p in the trie $\mathcal{T}_{\mathcal{P}}$, we denote by $lbl(p)$ the string which labels the path from the root of $\mathcal{T}_{\mathcal{P}}$ to p and put $len(p) =_{\text{Def}} |lbl(p)|$. Plainly, the map lbl is injective. Additionally, for any edge (p, q) in $\mathcal{T}_{\mathcal{P}}$, the label of (p, q) is denoted by $lbl(p, q)$.

For a set of patterns $\mathcal{P} = \{P_1, P_2, \ldots, P_r\}$ over an alphabet Σ, the *maximal trie* of $\mathcal{P}$ is the trie $\mathcal{T}_{\mathcal{P}}^{max}$ obtained by merging into a single node the roots of the linear tries $\mathcal{T}_{\mathcal{P}_1}, \mathcal{T}_{\mathcal{P}_2}, \ldots, \mathcal{T}_{\mathcal{P}_r}$ relative to the patterns $P_1, P_2, \ldots, P_r$, respectively. Strictly speaking, the maximal trie is a nondeterministic trie, as property (i) above may not hold at the root.

The directed acyclic word graph (DAWG) for a finite set of patterns $\mathcal{P}$ is a data structure representing the set $Fact(\mathcal{P})$. To describe it precisely, we need the following definitions. Let us denote by $end\text{-}pos(u)$ the set of all positions in $\mathcal{P}$ where an occurrence of u ends, for $u \in \Sigma^*$; more formally, we put

$$end\text{-}pos(u) =_{\text{Def}} \{(P, j) \mid u \sqsupseteq P_j, \text{ with } P \in \mathcal{P} \text{ and } |u| - 1 \leq j < |P|\} .$$

For instance, we have $end\text{-}pos(\varepsilon) = \{(P, j) \mid P \in \mathcal{P} \text{ and } -1 \leq j < |P|\}$, since $\varepsilon \sqsupseteq P_j$, for each $P \in \mathcal{P}$ and $-1 \leq j < |P|$ (we recall that $P_{-1} = \varepsilon$, by convention).

We also define an equivalence relation $R_{\mathcal{P}}$ over Σ^* by putting

$$u \; R_{\mathcal{P}} \; v \iff_{\text{Def}} end\text{-}pos(u) = end\text{-}pos(v) , \tag{2}$$

for $u, v \in \Sigma^*$, and denote by $R_{\mathcal{P}}(u)$ the equivalence class of $R_{\mathcal{P}}$ containing the string u. Also, we put

$$val(R_{\mathcal{P}}(u)) =_{\text{Def}} \text{the longest string in the equivalence class } R_{\mathcal{P}}(u) . \tag{3}$$

Then the DAWG for a finite set $\mathcal{P}$ of patterns is a directed acyclic graph (V, E) with an edge labeling function $lbl()$, where $V = \{R_{\mathcal{P}}(u) \mid u \in Fact(\mathcal{P})\}$, $E = \{(R_{\mathcal{P}}(u), R_{\mathcal{P}}(uc)) \mid u \in \Sigma^*, c \in \Sigma, uc \in Fact(\mathcal{P})\}$, and $lbl(R_{\mathcal{P}}(u), R_{\mathcal{P}}(uc)) = c$, for $u \in \Sigma^*$, $c \in \Sigma$ such that $uc \in Fact(\mathcal{P})$ (cf. [4]).

We define below the Aho-Corasick NFA and the suffix NFA for a set $\mathcal{P}$ of patterns.

2.1. The Aho-Corasick NFA

The Aho-Corasick NFA for a set $\mathcal{P}$ of patterns over an alphabet Σ is induced directly by the trie $\mathcal{T}_{\mathcal{P}}$ for $\mathcal{P}$. More precisely, it is the NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta_A, q_0, F)$, where:

- Q is the set of nodes of $\mathcal{T}_{\mathcal{P}}$ (*the set of states*);
- $q_0 \in Q$ is the root of $\mathcal{T}_{\mathcal{P}}$ (*the initial state*);
- $\delta_A : Q \times \Sigma \to \mathscr{P}(Q)$ is the *transition function*, with

$$\delta_A(q, c) =_{Def} \begin{cases} \{p \in Q \mid lbl(p) = c\} \cup \{q_0\} & \text{if } q = q_0 \\ \{p \in Q \mid lbl(p) = lbl(q).c\} & \text{if } q \neq q_0\,, \end{cases}$$

  for $q \in Q$, $c \in \Sigma$, and where we recall that $\mathscr{P}(\cdot)$ denotes the powerset operator;
- $F =_{\text{Def}} \{q \in Q \mid lbl(q) \in \mathcal{P}\}$ is the set of *final* states.

Plainly we have $|Q| \leq \sum_{P \in \mathcal{P}} |P|$.

We also associate with the NFA $A_{\mathcal{P}}$ a failure function $fail : Q \setminus \{q_0\} \to Q$ such that

- $lbl(fail(q)) \sqsupset lbl(q)$, and
- $len(fail(q)) \geq len(p)$, for each $p \in Q$ such that $lbl(p) \sqsupset lbl(q)$

(in other words, $lbl(fail(q))$ is the longest proper suffix of $lbl(q)$ which is also a prefix of a string in $\mathcal{P}$).

The automaton $A_{\mathcal{P}}$ can be seen as the nondeterministic version of the Aho-Corasick automaton: this is a trie $\mathcal{T}_{\mathcal{P}}$ for a set of patterns $\mathcal{P}$ augmented with *failure* links, which are followed when no transition is possible on a text character (cf. [1]).

An immediate, yet useful, property of the Aho-Corasick NFA, which can be readily proved by induction, is the following

$$q_0 \in \delta_A^*(q_0, u), \text{ for every } u \in \Sigma^*. \tag{4}$$

The Aho-Corasick NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta_A, q_0, F)$ relative to a given set $\mathcal{P}$ of patterns can be used to find the occurrences of the patterns of $\mathcal{P}$ in a given text T, by observing that a pattern $P \in \mathcal{P}$ has an occurrence in T ending at position i, i.e., $P \sqsupseteq T_i$, if and only if $\delta_A^*(q_0, T[0\,..\,i])$ contains a final state $q \in F$ such that $lbl(q) = P$. Thus, to find all the occurrences in T of the patterns of $\mathcal{P}$, it is enough to compute the set $\delta_A^*(q_0, T_i) \cap F$, for $i = 0, 1, \ldots, |T| - 1$. As an immediate consequence of (1) and the definitions of δ_A and δ_A^* on $\mathscr{P}(Q)$, we have $\delta_A^*(q_0, T_i) = \delta_A(\delta_A^*(q_0, T_{i-1}), T[i])$, for $i = 1, 2, \ldots, |T| - 1$. Hence, the problem of computing efficiently the sets $\delta_A^*(q_0, T_i)$ can be reduced to the problem of evaluating efficiently transition actions of the form $\delta_A(D, c)$, for any $c \in \Sigma$ and any *reachable configuration* $D \subseteq Q$ of $A_{\mathcal{P}}$, namely any subset $D \subseteq Q$ such that $D = \delta_A^*(q_0, u)$, for some $u \in \Sigma^*$.

The following property is an immediate consequence of the definition of the failure function.

Lemma 1. *Given the Aho-Corasick NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta_A, q_0, F)$ for a set $\mathcal{P}$ of patterns and its associated failure function $fail : Q \setminus \{q_0\} \to Q$, we have*

$$lbl(p) \sqsupset lbl(q) \to lbl(p) \sqsupseteq lbl(fail(q)),$$

for all $p \in Q$ and $q \in Q \setminus \{q_0\}$. □

2.2. The suffix NFA

The suffix NFA for a finite set $\mathcal{P}$ of patterns over an alphabet Σ is the NFA with ε-transitions $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ induced by the DAWG for $\mathcal{P}$, where

- $Q =_{\text{Def}} \{R_{\mathcal{P}}(u) \mid u \in Fact(\mathcal{P})\}$ is the set of states;[4]
- $q_0 = R_{\mathcal{P}}(\varepsilon)$ is the initial state;
- $\delta_S : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathscr{P}(Q)$ is the transition function defined by:

$$\delta_S(R_{\mathcal{P}}(u), a) =_{\text{Def}} \begin{cases} Q & \text{if } ua = \varepsilon \\ \{R_{\mathcal{P}}(ua)\} & \text{if } ua \in Fact(\mathcal{P}) \setminus \{\varepsilon\} \\ \emptyset & \text{otherwise;} \end{cases}$$

- $F = \{q \in Q \mid val(q) \in Suff(\mathcal{P})\}$ is the set of final states.

It is well-known that the language recognized by the NFA $S_{\mathcal{P}}$ is $Suff(\mathcal{P})$. Additionally, the NFA $S'_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, Q)$, obtained from $S_{\mathcal{P}}$ by considering all states in Q as final, recognizes the language $Fact(\mathcal{P})$ of all factors of strings in $\mathcal{P}$. In other words, for $u \in \Sigma^*$, we have

$$\delta_S(q_0, u) \neq \emptyset \quad \text{if and only if} \quad u \in Fact(\mathcal{P})\,. \tag{5}$$

We also observe that if $size(\mathcal{P}) > 1$, then $|Q| \leq 2\sum_{P \in \mathcal{P}} |P| - 1$ (cf. [4]).

We also define a failure function, $suf : Fact(\mathcal{P}) \setminus \{\varepsilon\} \to Fact(\mathcal{P})$, named *suffix link*, by putting

$$suf(u) =_{\text{Def}} \text{ the longest } v \in Suff(u) \text{ such that } v \not R_{\mathcal{P}} u \tag{6}$$

for $u \in Fact(\mathcal{P}) \setminus \{\varepsilon\}$.

The $suf(\cdot)$ function can be extended to the equivalence classes of $R_{\mathcal{P}}$ not containing ε, and therefore to the set $Q \setminus \{q_0\}$ of states of $S_{\mathcal{P}}$, by putting for all $q \in Q \setminus \{q_0\}$

$$suf(q) =_{\text{Def}} R_{\mathcal{P}}(suf(val(q)))\,.$$

A useful property of the function $suf(\cdot)$ is proved in the following lemma.

Lemma 2. *Given a nondeterministic suffix automaton $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ for a set of patterns $\mathcal{P}$, for all $p, q \in Q \setminus \{q_0\}$ we have*

[4] $R_{\mathcal{P}}$ is the equivalence relation defined by (2).

(a) if $val(p) \sqsupset val(q)$ *then* $val(p) \sqsupseteq val(suf(q)) \sqsupset val(q)$*;*

(b) if $val(p) \sqsupset val(q)$ *then* $suf^{(k)}(q) = p$*, for some* $k \geq 1$
(where $suf^{(0)}(q) =_{Def} q$ *and, recursively,* $suf^{(h+1)}(q) =_{Def} suf(suf^{(h)}(q))$*, for* $h \geq 0$*, provided that* $suf^{(h)}(q) \neq q_0$*).*

PROOF. Let $val(p) \sqsupset val(q)$. From the definition (6) of the function $suf(\cdot)$, we have $val(p) \sqsupseteq suf(val(q)) \sqsupset val(q)$. Since $val(R_{\mathcal{P}}(suf(val(q))))$ is the longest string in the equivalence class of $suf(val(q))$, we have also that $suf(val(q)) \sqsupseteq val(R_{\mathcal{P}}(suf(val(q))))$ and $val(p) \sqsupseteq val(R_{\mathcal{P}}(suf(val(q)))) \sqsupset val(q)$. Thus, (a) follows by observing that $val(suf(q)) = val(R_{\mathcal{P}}(suf(val(q))))$.

Concerning (b), we argue as follows. From (a) we have $val(p) \sqsupseteq val(suf(q))$. If $val(p) = val(suf(q))$, then $suf^{(1)}(q) = suf(q) = p$, and we are done. Otherwise, $val(p) \sqsupset val(suf(q))$. By applying (a) repeatedly, we eventually obtain a sequence

$$val(p) = val(suf^{(k)}(q)) \sqsupset val(suf^{(k-1)}(q)) \sqsupset \cdots \sqsupset val(suf(q)) \sqsupset val(q)\,,$$

for some $k \geq 1$, which implies $suf^{(k)}(q) = p$, thus proving (b). □

Given a set of patterns $\mathcal{P}$ over Σ, the suffix NFA $S_{(\mathcal{P}_{l_{min}})^{\mathrm{r}}} = (Q, \Sigma, \delta_S, q_0, F)$ for $(\mathcal{P}_{l_{min}})^{\mathrm{r}}$ can be used to find the occurrences of the patterns of $\mathcal{P}$ in a text T of length n by observing that a pattern $P \in \mathcal{P}$ of length m has an occurrence in T ending at position $i + m - 1$, i.e., $T[i\,..\,i + m - 1] = P$, if and only if $\delta_S^*(q_0, (T[i\,..\,i+l_{min}-1])^{\mathrm{r}})$ contains a final state $q \in F$ such that $val(q) \sqsupseteq P^{\mathrm{r}}$ and $T[i+l_{min}\,..\,i+m-1] \sqsupseteq P$. Hence, to find all the occurrences of the patterns in $\mathcal{P}$ in T, one can compute $\delta_S^*(q_0, (T[i\,..\,i+l_{min}-1])^{\mathrm{r}}) \cap F$, for $i = 0, 1, \ldots, n-l_{min}$ and then make the appropriate checks for the candidate matches. In practice, algorithms based on this approach can skip windows as follows: for a window of T of size l_{min} beginning at position i, let l be the length of the longest proper suffix of $T[i\,..\,i+l_{min}-1]$ such that $\delta_S^*(q_0, (T[i+l_{min}-l\,..\,i+l_{min}-1])^{\mathrm{r}}) \cap F \neq \emptyset$. Then, the windows at positions $i, i+1, \ldots i+l_{min}-l-1$ can be safely skipped.

3. Bit-parallel simulation of NFAs for the multiple string matching problem

To simulate efficiently the NFAs $A_{\mathcal{P}}$ and $S_{\mathcal{P}}$ with the bit-parallel technique, a suitable representation of the transition function δ is needed, in order that $\delta(D, c)$ can be computed by $\mathcal{O}(\lceil |Q|/w \rceil)$ computer operations, for any reachable configuration $D \subseteq Q$ and character $c \in \Sigma$ (as before, w is the number of bits in a computer word).

Our construction is based on a result for the Glushkov automaton that can be immediately generalized to NFAs like $A_{\mathcal{P}}$ and $S_{\mathcal{P}}$ as follows (cf. [13]).

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA with ε-transitions such that up to the ε-transitions, for each state $q \in Q$, either

(i) all the incoming transitions in q are labeled by the same character, or

(ii) all the incoming transitions in q originate from a unique state.

Let $B(c)$, for $c \in \Sigma$, be the set of states of N with an incoming transition labeled by c, i.e.,

$$B(c) =_{\text{Def}} \{q \in Q \mid q \in \delta(p, c), \text{ for some } p \in Q\}\,.$$

Likewise, let $\mathit{Follow}(q)$, for $q \in Q$, be the set of states reachable from state q with one transition over a character in Σ, i.e.,

$$\mathit{Follow}(q) =_{\text{Def}} \bigcup_{c \in \Sigma} \delta(q, c)\,.$$

Also, let

$$\Phi(D) =_{\text{Def}} \bigcup_{q \in D} \mathit{Follow}(q)\,,$$

for $D \subseteq Q$. Then the following result holds.

Lemma 3 (cf. [13]). *For every $q \in Q$, $D \subseteq Q$, and $c \in \Sigma$, we have*

(a) $\delta(q, c) = \mathit{Follow}(q) \cap B(c)$;

(b) $\delta(D, c) = \Phi(D) \cap B(c)$.

Proof. Concerning (a), we notice that $\delta(q, c) \subseteq \mathit{Follow}(q) \cap B(c)$ holds plainly. To prove the converse inclusion, let $p \in \mathit{Follow}(q) \cap B(c)$. Then $p \in \delta(q, c') \cap \delta(q', c)$, for some $c' \in \Sigma$ and $q' \in Q$. If p satisfies condition (i), then $c' = c$, and therefore $p \in \delta(q, c)$. On the other hand, if p satisfies condition (ii), then $q = q'$ and therefore we have again $p \in \delta(q, c)$.

From (a), we obtain immediately (b), since

$$\begin{aligned}\delta(D, c) &= \bigcup_{q \in D} \delta(q, c) = \bigcup_{q \in D} (\mathit{Follow}(q) \cap B(c)) \\ &= \bigcup_{q \in D} \mathit{Follow}(q) \cap B(c) = \Phi(D) \ \cap \ B(c)\,.\end{aligned}$$ □

Provided that one finds an efficient way of storing and accessing the maps $\Phi(\cdot)$ and $B(\cdot)$, equation (b) of Lemma 3 is particularly suitable for bit-parallelism, as set intersection can be readily implemented by the bitwise `and` operation.

We observe at once that the immediate solution of storing the maps $\Phi(\cdot)$ and $B(\cdot)$ as tables of bit words, respectively indexed by set of states and by characters in Σ, requires $(2^m + \sigma) \cdot m$ bits, which is exponential in the number m of states of $A_{\mathcal{P}}$ or $S_{\mathcal{P}}$ (σ is the size of the alphabet Σ). Thus we have to find a better way to store the map $\Phi(\cdot)$, exploiting the fact that $\Phi(D)$ needs to be evaluated over reachable configurations D of $A_{\mathcal{P}}$ or $S_{\mathcal{P}}$ only.

In Sections 4 and 5 we will show that the map $\Phi(\cdot)$ can be conveniently stored in $\mathcal{O}(m^2)$-space, both in the case of the Aho-Corasick NFA and of the suffix NFA. More specifically, we will show in both cases that each nonempty reachable configuration D can be represented in terms of a unique state, which will be

referred to as $lead(D)$. This will allow us to represent $\Phi(D)$ as $\dot{\Phi}(lead(D))$, where $\dot{\Phi} : Q \to \mathscr{P}(Q)$ is the map such that the q-th bit of $\dot{\Phi}(p)$ is set if and only if there is a transition to state q originating from p or any other state belonging to the reachable configuration uniquely identified by p. Plainly, the map $\dot{\Phi}$ can be stored in $\mathcal{O}(m^2)$-space and allows to rewrite equation (b) of Lemma 3 as

$$\delta(D, c) = \dot{\Phi}(lead(D)) \cap B(c)\,,$$

which in turn translates readily into the bit-parallel assignment

$$D \leftarrow \dot{\Phi}[lead(D)] \;\&\; B[c]\,.$$

4. Bit-parallel simulation of the Aho-Corasick NFA for a set of patterns

In this section we present a bit-parallel encoding of the Aho-Corasick NFA; specifically, based on the idea explained in the previous section, we first show that each reachable configuration of $A_{\mathcal{P}}$ is uniquely identified by a single state. Then, we devise the map $\dot{\Phi}(\cdot)$ by using the relation between reachable configurations of the automaton and the associated failure function, and prove its correctness. Finally, we show that the map $lead(\cdot)$ admits an efficient implementation.

We begin by showing in the following elementary lemma that, for any string u, the configuration of the automaton after reading u consists of all the states whose labels are a suffix of u.

Lemma 4. *Let $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$ be the Aho-Corasick NFA for a finite set $\mathcal{P}$ of patterns over the alphabet Σ, and let $u \in \Sigma^*$. Then $\delta^*(q_0, u) = \{q \in Q \mid lbl(q) \sqsupseteq u\}$.*

Proof. For $u = \varepsilon$, the lemma holds plainly. Thus, let $u = u'.c$, with $u' \in \Sigma^*$ and $c \in \Sigma$. We first show by induction on u that $\delta^*(q_0, u) \subseteq \{q \in Q \mid lbl(q) \sqsupseteq u\}$. Let $p \in \delta^*(q_0, u)$. Since, by (1),

$$\delta^*(q_0, u'.c) = \delta^*(\delta^*(q_0, u'), c) = \delta(\delta^*(q_0, u'), c) = \bigcup_{q \in \delta^*(q_0, u')} \delta(q, c)\,,$$

we have $p \in \delta(\overline{q}, c)$, for some $\overline{q} \in \delta^*(q_0, u')$, so that, by inductive hypothesis, $lbl(\overline{q}) \sqsupseteq u'$, and therefore $lbl(p) = lbl(\overline{q}).c \sqsupseteq u'.c = u$.

To show the converse inclusion relationship, let $p \in Q$ be such that $lbl(p) \sqsupseteq u$. We prove by induction on $lbl(p)$ that $p \in \delta^*(q_0, u)$. In view of (4), we may dismiss at once the case in which $lbl(p) = \varepsilon$, i.e., $p = q_0$, and therefore assume that $lbl(p) = lbl(p').c$, for some $p' \in Q$ and $c \in \Sigma$. Hence $u = u'.c$, for some $u' \in \Sigma^*$ such that $lbl(p') \sqsupseteq u'$, so that, by inductive hypothesis, we have $p' \in \delta^*(q_0, u')$. Thus, by (1), $p \in \delta(p', c) \subseteq \delta(\delta^*(q_0, u'), c) = \delta^*(\delta^*(q_0, u'), c) = \delta^*(q_0, u'.c) = \delta^*(q_0, u)$. □

Given a reachable configuration D, the previous lemma implies that for any two distinct states $p, p' \in D$ we have $|lbl(p)| \neq |lbl(p')|$, since either $lbl(p) \sqsupset lbl(p')$ or $lbl(p') \sqsupset lbl(p)$. Thus there must exist a unique state $\overline{q} \in D$ such that $|lbl(p)| \leq |lbl(\overline{q})|$, for every $p \in D$. Let us denote such a state by $lead(D)$. Then we have:

Corollary 1. *Let $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$ be the Aho-Corasick NFA for a finite set $\mathcal{P}$ of patterns over Σ, and let D be a reachable configuration of $A_{\mathcal{P}}$. Then $D = \{q \in Q \mid lbl(q) \sqsupseteq lbl(lead(D))\}$.*

Proof. Let $u \in \Sigma^*$ be such that $D = \delta^*(q_0, u)$. In view of Lemma 4, it is enough to observe that $lbl(q) \sqsupseteq u$ if and only if $lbl(q) \sqsupseteq lbl(lead(D))$, for every $q \in Q$. □

From the preceding corollary, it follows at once that the reachable configurations of the Aho-Corasick NFA $A_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$, for a set $\mathcal{P}$ of patterns, are in 1-1 correspondence with its states, and therefore their number is $|Q|$.

A convenient way to represent Φ uses the map $\dot{\Phi}_A : Q \to \mathscr{P}(Q)$, recursively defined by

$$\dot{\Phi}_A(q) =_{Def} \begin{cases} Follow(q_0), & \text{if } q = q_0 \\ Follow(q) \cup \dot{\Phi}_A(fail(q)), & \text{if } q \neq q_0\,, \end{cases} \tag{7}$$

as shown in the following lemma.

Lemma 5. *For any reachable configuration D of the Aho-Corasick NFA $A_{\mathcal{P}}$, we have $\Phi(D) = \dot{\Phi}_A(lead(D))$.*

Proof. We proceed by induction on $|lbl(lead(D))|$. If $|lbl(lead(D))| = 0$, then $lead(D) = q_0$ and $D = \{q_0\}$, so that $\Phi(D) = Follow(q_0) = \dot{\Phi}_A(q_0) = \dot{\Phi}_A(lead(D))$. For the inductive step, we have

$$\begin{aligned}
\Phi(D) &= \bigcup_{q \in D} Follow(q) = \bigcup_{\substack{q \in Q \\ lbl(q) \sqsupseteq lbl(lead(D))}} Follow(q) \\
&= Follow(lead(D)) \cup \bigcup_{\substack{q \in Q \\ lbl(q) \sqsupset lbl(lead(D))}} Follow(q) \\
&= Follow(lead(D)) \cup \bigcup_{\substack{q \in Q \\ lbl(q) \sqsupseteq lbl(fail(lead(D)))}} Follow(q) \\
&= Follow(lead(D)) \cup \Phi(\{q \in Q \mid lbl(q) \sqsupseteq lbl(fail(lead(D)))\}) \\
&= Follow(lead(D)) \cup \dot{\Phi}_A(fail(lead(D)) = \dot{\Phi}_A(lead(D))).
\end{aligned}$$ □

Plainly, the map $\dot{\Phi}_A(\cdot)$ requires only $|Q|^2$ bits. Additionally, the map $lead(\cdot)$ can be computed very efficiently at run-time, provided that the states of $A_{\mathcal{P}}$ are ordered in such a way that a state p precedes a state q whenever $|lbl(p)| < |lbl(q)|$ (say, by a breadth-first visit of $A_{\mathcal{P}}$ from q_0). Indeed, in such a case, if we assume that D is encoded as a bit mask, then $lead(D)$ is the index of the highest bit of D set to `1`, and therefore is equal to $\lfloor \log_2 D \rfloor$.

```
Log-And (T, P = {P_1, P_2, ..., P_r})
  /* Preprocessing */
 1.  Let A_P = (Q, Σ, δ, q_0, F) be the Aho-Corasick NFA relative to the set of
     patterns P and let the maps Follow(), lbl(), and fail() be defined as before,
     relative to A_P. We also assume that Q = {0, 1, ..., ℓ−1}, where ℓ = |Q|, and
     that if |lbl(p)| < |lbl(q)| then p < q, for any p, q ∈ Q.
 2.  L ← 0^ℓ
 3.  for c ∈ Σ do B[c] ← 0^{ℓ−1}1
 4.  for p ← 0 to ℓ − 1 do
 5.      Φ̇_A[p] ← 0^{ℓ−1}1
 6.      for q ∈ Follow(p) \ {0} do
 7.          H ← 0^{ℓ−1}1 ≪ q
 8.          c ← lbl(p, q)
 9.          B[c] ← B[c] | H
10.          if q ∈ F then L ← L | H
11.          Φ̇_A[p] ← Φ̇_A[p] | H
12.      if p ≠ 0 then
13.          Φ̇_A[p] ← Φ̇_A[p] | Φ̇_A[fail(p)]
  /* Searching */
14.  D ← 0^{ℓ−1}1
15.  for j ← 0 to |T| − 1 do
16.      lead ← ⌊log_2(D)⌋
17.      D ← Φ̇_A[lead] & B[T[j]]
18.      if D & L ≠ 0^ℓ then Output(j)
```

Figure 1: The Log-And algorithm for the multiple string matching problem.

4.1. The Log-And *algorithm*

Based on the previous considerations, we present an efficient bit-parallel algorithm, which we call Log-And, for solving the multiple string matching problem.

In the Log-And algorithm, reported in Figure 1, the sets D, B and the map $\dot{\Phi}_A$ are encoded as bit tables.

As opposed to the Shift-And algorithm, bit 0 is reserved for the initial state, so that $lead(D)$ is never computed for an empty set (0 value) as the initial state is always active.

In the preprocessing phase, the Log-And algorithm iterates over the nodes of $A_{\mathcal{P}}$, which are assumed to be sorted by a breadth-first search; for each node, the corresponding Φ mask is computed using (7) and the B masks associated to the labels of its outgoing edges are augmented accordingly. The algorithm precomputes also a *final state* bit mask, L, where a bit is set to 1 if and only if it corresponds to a final state of the automaton. The maps $Follow(\cdot)$, $lbl(\cdot)$ and $fail(\cdot)$ can be constructed using the algorithm introduced in [1], while the loop iterating over the states of the automaton in breadth-first order can be easily implemented using a queue.

Then, during the searching phase, the Log-And algorithm scans the text T, character by character, using the following basic transition, based on Lemma 3(b),

$$D \leftarrow \dot{\Phi}_A[\lfloor \log_2(D) \rfloor] \;\&\; B[c]\,.$$

The resulting algorithm has $\mathcal{O}((m+\sigma)\lceil m/w \rceil)$-space and $\mathcal{O}(n\lceil m/w \rceil)$-searching time complexity, where $n = |T|$, m is the number of nodes of $A_{\mathcal{P}}$, σ is the alphabet size, and w is the word size in bits. When $m \in \mathcal{O}(w)$, the Log-And algorithm turns out to have a $\mathcal{O}(m + \sigma)$-space and $\mathcal{O}(n)$-searching time complexity.

If one is interested also in retrieving the patterns that match (if any) at each text position, it is convenient to precompute a table which maps each final state of $A_{\mathcal{P}}$ to the corresponding pattern index. Then, in the searching phase, for each position j, the algorithm iterates over the bits of $(D \ \& \ L)$ by computing the index of the highest bit set and querying the corresponding pattern number. The whole sequence is repeated, after having cleared the highest bit, until there are no more bits set.

5. Bit-parallel simulation of the suffix NFA for a set of patterns

In this section we devise a bit-parallel encoding of the suffix automaton induced by the DAWG data structure for a set of patterns. We observe that the maximal trie of a set $\mathcal{P}$ of patterns can also be turned into an automaton that recognizes the language $\mathit{Suff}(\mathcal{P})$, by adding an ε-transition from the initial state to all remaining states. The size of the DAWG data structure can vary between the number $|Q_{A_{\mathcal{P}}}|$ of states of the Aho-Corasick automaton for $\mathcal{P}$ and $2 \cdot \mathit{size}(\mathcal{P}) - 1$ (cf. [5]). Thus, although the DAWG allows to factor prefix redundancy in the patterns, it is not always preferable to the maximal trie, whose size is $\mathit{size}(\mathcal{P})$. However, it turns out that the average size of the DAWG is close to $|Q_{A_{\mathcal{P}}}|$ which, depending on the degree of prefix redundancy in $\mathcal{P}$ may be much smaller than $\mathit{size}(\mathcal{P})$.

Let $S_{\mathcal{P}}$ be the suffix NFA for a set $\mathcal{P}$ of patterns over an alphabet Σ. We devise a bit-parallel encoding of this automaton much along the lines of what has been done for the $A_{\mathcal{P}}$ automaton.

The following lemma is the analogous of Lemma 4 for the present context of suffix NFAs. It shows that, for any string u, the configuration of the automaton after reading u consists of all the states such that u is a suffix of the corresponding labels. For the sake of completeness, we include its proof, though, up to few adaptations, it follows closely the proof of Lemma 4.

Lemma 6. *Let $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ be the suffix NFA for a finite set $\mathcal{P}$ of patterns, and let $u \in \Sigma^*$. Then $\delta_S^*(q_0, u) = \{q \in Q \mid u \sqsupseteq \mathit{val}(q)\}$.*

Proof. For $u = \varepsilon$, the lemma holds plainly. Thus, let $u = u'.c$, with $u' \in \Sigma^*$ and $c \in \Sigma$. We first show by induction on u that $\delta_S^*(q_0, u) \subseteq \{q \in Q \mid u \sqsupseteq \mathit{val}(q)\}$. Thus, let $p \in \delta_S^*(q_0, u)$. By (1), we have $\delta_S^*(q_0, u'.c) = \delta_S^*(\delta_S^*(q_0, u'), c) = \delta_S(\delta_S^*(q_0, u'), c) = \bigcup_{q \in \delta_S^*(q_0, u')} \delta_S(q, c)$. Hence, $p \in \delta_S(\overline{q}, c)$, for some $\overline{q} \in \delta_S^*(q_0, u')$, so that, by inductive hypothesis, $u' \sqsupseteq \mathit{val}(\overline{q})$, and therefore $u = u'.c \sqsupseteq \mathit{val}(\overline{q}).c = \mathit{val}(p)$.

To show the converse inclusion relationship, let $p \in Q$ be such that $u \sqsupseteq \mathit{val}(p)$. We prove by induction on $\mathit{val}(p)$ that $p \in \delta_S^*(q_0, u)$. If $\mathit{val}(q) = \varepsilon$, then

$p = q_0$ and $u = \varepsilon$, so that $p \in \delta_S^*(q_0, u)$ holds trivially. Let us then assume that $val(p) = val(p').c$, for some $p' \in Q$ and $c \in \Sigma$. Hence $u = u'.c$, for some $u' \in \Sigma^*$ such that $u' \sqsupseteq val(p')$, so that, by inductive hypothesis, we have $p' \in \delta_S^*(q_0, u')$. Thus, by (1), $p \in \delta_S(p', c) \subseteq \delta_S(\delta_S^*(q_0, u'), c) = \delta_S^*(\delta_S^*(q_0, u'), c) = \delta_S^*(q_0, u'.c) = \delta_S^*(q_0, u)$. □

The following lemma and corollary illustrate some useful properties concerning a nonempty reachable configuration $D = \delta_S^*(q_0, u)$ of $S_{\mathcal{P}}$ for a string u and relative equivalence class $R_{\mathcal{P}}(u)$ defined by (2). In particular, it will follow from them that there is a 1-1 correspondence between nonempty reachable configurations of the automaton and the equivalence classes of $R_{\mathcal{P}}$.

Lemma 7. *Let $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ be the suffix NFA for a set of patterns $\mathcal{P}$. Then, for any string $u \in \Sigma^*$, the following implications hold:*

(a) if $q \in \delta_S^*(q_0, u)$, then $val(R_{\mathcal{P}}(u)) \sqsupseteq val(q)$;

(b) if $\delta_S^*(q_0, u) \neq \emptyset$, then $R_{\mathcal{P}}(u) \in \delta_S^*(q_0, u)$;

(c) if $\delta_S^*(q_0, u) = \delta_S^*(q_0, v) \neq \emptyset$, then $u \; R_{\mathcal{P}} \; v$, for $v \in \Sigma^*$.

Proof. Concerning (a), let $q \in \delta_S^*(q_0, u)$. From (5), it follows that $u \in Fact(\mathcal{P})$, so that $val(R_{\mathcal{P}}(u))$ is defined. Then by lemma 6 we have that $u \sqsupseteq val(q)$, which in turn implies that $end\text{-}pos(val(q)) \subseteq end\text{-}pos(u) = end\text{-}pos(val(R_{\mathcal{P}}(u)))$. Hence, $val(R_{\mathcal{P}}(u)) \sqsupseteq val(q)$.

Concerning (b), from the very definitions of $R_{\mathcal{P}}$ and $val(\cdot)$ (see (2) and (3)), we have that $u \sqsupseteq val(R_{\mathcal{P}}(u))$ which, by Lemma 6, implies that $R_{\mathcal{P}}(u) \in \delta_S^*(q_0, u)$.

Finally, concerning (c), let $\delta_S^*(q_0, u) = \delta_S^*(q_0, v) \neq \emptyset$. Then (b) yields $R_{\mathcal{P}}(u) \in \delta_S^*(q_0, v)$ and $R_{\mathcal{P}}(v) \in \delta_S^*(q_0, u)$ which, again by Lemma 6, imply $val(R_{\mathcal{P}}(v)) \sqsupseteq val(R_{\mathcal{P}}(u))$ and $val(R_{\mathcal{P}}(u)) \sqsupseteq val(R_{\mathcal{P}}(v))$, respectively. Hence, $val(R_{\mathcal{P}}(u)) = val(R_{\mathcal{P}}(v))$ so that $u \; R_{\mathcal{P}} \; v$. □

Given a nonempty reachable configuration D for a string u, the previous lemma implies that the set

$$\{R_{\mathcal{P}}(u) \mid \delta_S^*(q_0, u) = D, \text{ for } u \in Fact(\mathcal{P})\}$$

has exactly one element. Therefore the following definition is well-given

$$lead(D) \; =_{\text{Def}} R_{\mathcal{P}}(u)\,,$$

for any $u \in Fact(\mathcal{P})$ such that $\delta_S^*(q_0, u) = D$.

Corollary 2. *Let $S_{\mathcal{P}} = (Q, \Sigma, \delta, q_0, F)$ be the suffix NFA for a set of patterns $\mathcal{P}$, and let D be a nonempty reachable configuration of $S_{\mathcal{P}}$. Then $D = \{q \in Q \mid val(lead(D)) \sqsupseteq val(q)\}$.*

Proof. Let $u \in Fact(\mathcal{P})$ such that $\delta_S^*(q_0, u) = D$ and let $q \in D$. From Lemma 7(a) we have that $val(lead(D)) = val(R_{\mathcal{P}}(u)) \sqsupseteq val(q)$. Conversely, if $val(R_{\mathcal{P}}(u)) \sqsupseteq val(q)$, then $u \sqsupseteq val(q)$, so that, by Lemma 6, $q \in D$. □

From the preceding corollary, it follows at once that the nonempty reachable configurations of a suffix NFA $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ for a set $\mathcal{P}$ of patterns are in 1-1 correspondence with its states, and therefore their number is $|Q|$.

For $q \in Q$, let

$$rsuf(q) \ =_{\text{Def}} suf^{-1}[\{q\}] = \{p \in Q \mid suf(p) = q\}$$

be the set of states whose suffix link is q, where $suf(\cdot)$ is the map defined in (6).

We will show that a reachable configuration of $S_{\mathcal{P}}$ can be represented in terms of the maps $lead(\cdot)$ and $rsuf(\cdot)$.

Lemma 8. *Let D be a nonempty reachable configuration of the suffix NFA $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ for a set $\mathcal{P}$ of patterns. Then*

$$D = \{lead(D)\} \cup \bigcup_{p \in rsuf(lead(D))} \{q \in Q \mid val(p) \sqsupseteq val(q)\}\,.$$

Proof. From Corollary 2 we have

$$\begin{aligned} D &= \{q \in Q \mid val(lead(D)) \sqsupseteq val(q)\} \\ &= \{lead(D)\} \cup \{q \in Q \mid val(lead(D)) \sqsupset val(q)\}\,. \end{aligned}$$

Then to prove the lemma it is enough to show that

$$\{q \in Q \mid val(lead(D)) \sqsupset val(q)\} = \bigcup_{p \in rsuf(lead(D))} \{q \in Q \mid val(p) \sqsupseteq val(q)\}\,.$$

Let $q' \in Q$ be such that $val(lead(D)) \sqsupset val(q')$. By Lemma 2(b), there exists $k \geq 1$ such that $suf^{(k)}(q') = lead(D)$. Let $p' = suf^{(k-1)}(q')$. Plainly, $suf(p') = lead(D)$, so that $p' \in rsuf(lead(D))$. Additionally, $val(p') = val(suf^{(k-1)}(q')) \sqsupseteq val(q')$.

Hence,

$$q' \in \{q \in Q \mid val(p') \sqsupseteq val(q)\} \subseteq \bigcup_{p \in rsuf(lead(D))} \{q \in Q \mid val(p) \sqsupseteq val(q)\}$$

so that

$$\{q \in Q \mid val(lead(D)) \sqsupset val(q)\} \subseteq \bigcup_{p \in rsuf(lead(D))} \{q \in Q \mid val(p) \sqsupseteq val(q)\}\,.$$

To prove the converse relationship, let

$$q' \in \bigcup_{p \in rsuf(lead(D))} \{q \in Q \mid val(p) \sqsupseteq val(q)\}$$

and let $p' \in rsuf(lead(D))$ such that $val(p') \sqsupseteq val(q')$. Then $val(lead(D)) = val(suf(p')) \sqsupset val(p) \sqsupseteq val(q')$, since $suf(p') = lead(D)$. Hence $q' \in \{q \in Q \mid val(lead(D)) \sqsupset val(q)\}$ proving

$$\bigcup_{p \in rsuf(lead(D))} \{q \in Q \mid val(p) \sqsupseteq val(q)\} \subseteq \{q \in Q \mid val(lead(D)) \sqsupset val(q)\}$$

and in turn completing the proof of the lemma. □

A convenient way to represent the map $\Phi(\cdot)$ makes use of the following map $\dot{\Phi}_S : Q \to \mathscr{P}(Q)$, defined by

$$\dot{\Phi}_S(q) =_{Def} \begin{cases} \mathit{Follow}(q), & \text{if } \mathit{rsuf}(q) = \emptyset \\ \mathit{Follow}(q) \cup \bigcup_{p \in \mathit{rsuf}(q)} \dot{\Phi}_S(p), & \text{if } \mathit{rsuf}(q) \neq \emptyset\,, \end{cases} \tag{8}$$

as proved in the following lemma.

Lemma 9. *For any nonempty reachable configuration D of the suffix NFA $S_{\mathcal{P}} = (Q, \Sigma, \delta_S, q_0, F)$ for a set $\mathcal{P}$ of patterns, we have*

$$\Phi(D) = \dot{\Phi}_S(\mathit{lead}(D))\,.$$

Proof. To begin with, let us put $D_p =_{\text{Def}} \{q \in Q \mid \mathit{val}(p) \sqsupseteq \mathit{val}(q)\}$, for $p \in \mathit{rsuf}(\mathit{lead}(D))$, so that the decomposition of D provided by the preceding lemma can be rewritten in a more compact way as

$$D = \{\mathit{lead}(D)\} \cup \bigcup_{p \in \mathit{rsuf}(\mathit{lead}(D))} D_p\,. \tag{9}$$

Additionally, we observe that

$$\mathit{lead}(D_p) = p\,, \tag{10}$$

for each $p \in \mathit{rsuf}(\mathit{lead}(D))$. Indeed, by Lemma 6,

$$\delta_S^*(q_0, \mathit{val}(p)) = \{q \in Q \mid \mathit{val}(p) \sqsupseteq \mathit{val}(p)\} = D_p\,,$$

so that $\mathit{lead}(D_p) = R_{\mathcal{P}}(\mathit{val}(p)) = p$.

We are now ready to prove the lemma.

We proceed by induction on $\mathit{height}(\mathit{lead}(D))$, where

$$\mathit{height}(q) =_{\text{Def}} \text{ length of the longest chain of suffix link ending at } q\,.$$

If $\mathit{height}(\mathit{lead}(D)) = 0$ then $D = \{\mathit{lead}(D)\}$ and $\mathit{rsuf}(\mathit{lead}(D)) = \emptyset$. For the inductive step, in view of (9) and (10) above and of the fact that

$$\mathit{height}(\mathit{lead}(D_p)) < \mathit{height}(\mathit{lead}(D))$$

for $p \in \mathit{rsuf}(\mathit{lead}(D))$, we have

$$\begin{aligned} \Phi(D) &= \Phi(\{\mathit{lead}(D)\}) \cup \bigcup_{p \in \mathit{rsuf}(\mathit{lead}(D))} \Phi(D_p) \\ &= \mathit{Follow}(\mathit{lead}(D)) \cup \bigcup_{p \in \mathit{rsuf}(\mathit{lead}(D))} \dot{\Phi}_S(\mathit{lead}(D_p)) \\ &= \mathit{Follow}(\mathit{lead}(D)) \cup \bigcup_{p \in \mathit{rsuf}(\mathit{lead}(D))} \dot{\Phi}_S(p) \\ &= \dot{\Phi}_S(\mathit{lead}(D))\,, \end{aligned}$$

completing the proof of the lemma. □

As for the Aho-Corasick NFA, the map $\dot{\Phi}_S$ requires only $|Q|^2$ bits and the map $lead(\cdot)$ can be computed very efficiently at run-time, provided that the states of $S_{\mathcal{P}}$ are ordered in such a way that a state p precedes a state q whenever $|val(p)| < |val(q)|$ (say, by a breadth-first search from q_0). Indeed, in such a case, if we assume that D is encoded as a bit mask, then $lead(D)$ is the index of the lowest bit of D set to 1, and therefore is equal to $\lfloor \log_2(D \,\&\, (\sim D + 1)) \rfloor$.

5.1. The Backward-Log-And *algorithm*

In this section we present the Backward-Log-And algorithm, a BNDM-like bit-parallel algorithm based on the suffix NFA, for the multiple string matching problem. In the Backward-Log-And algorithm, whose pseudocode is reported in Figure 2, the sets D, B and the map $\dot{\Phi}_S(\cdot)$ are encoded as bit tables. There is no need to reserve bit 0 for the initial state, as the simulation stops when there are no more active states. For simplicity, in the pseudocode it is assumed that all patterns have the same length l.

During the preprocessing phase, the Backward-Log-And algorithm iterates over the states of the suffix NFA $S_{(\mathcal{P}_{l_{min}})^r}$, which are assumed to be sorted by a breadth-first search; for each state, the corresponding masks B and L are computed as in the Log-And algorithm, while the mask Φ is computed using (8). The maps $Follow(\cdot)$, $val(\cdot)$ and $suf(\cdot)$ can be constructed using the algorithm introduced in [4], while the loop iterating over the states of the automaton in breadth-first order can be easily implemented using a queue.

Then, during the searching phase, the Backward-Log-And algorithm scans the text T, character by character, using the following transition based on Lemma 3(b),

$$D \leftarrow \dot{\Phi}_A[\lfloor \log_2(D \,\&\, (\sim D + 1)) \rfloor] \,\&\, B[c] \,.$$

The resulting algorithm has $\mathcal{O}((m+\sigma)\lceil m/w \rceil)$-space and $\mathcal{O}(n\lceil m/w \rceil l_{min})$-searching time complexity, where $n = |T|$, l_{min} is the length of the shortest pattern, m is the number of nodes of $S_{(\mathcal{P}_{l_{min}})^r}$, σ is the alphabet size and w is the word size in bits. When $m \in \mathcal{O}(w)$, the Backward-Log-And algorithm turns out to have a $\mathcal{O}(m+\sigma)$-space and $\mathcal{O}(nl_{min})$-searching time complexity.

6. Conclusions

We have presented a method to simulate, using the bit-parallelism technique, the nondeterministic Aho-Corasick automaton induced by the trie for a set of patterns and the nondeterministic suffix automaton induced by the DAWG for a set of patterns. Our construction, based on a previous result for the Glushkov automaton, achieves polynomial (in the number of nodes of the automata) space complexity by exploiting the relation between active states of the NFAs and their associated failure functions. By using these automata it is possible to remove prefix redundancy in the pattern set, which allows to spare bits in the bit-parallel representation of the data structure.

We plan to investigate whether the approach presented here could be extended also to classes of characters and q-grams, among others.

```
Backward-Log-And (T, P = {P_1, P_2, ..., P_r})
  /* Preprocessing */
 1.  Let S_{(P_{l_min})^r} = (Q, Σ, δ, q_0, F) be the suffix NFA relative to the set of pat-
     terns (P_{l_min})^r and let the maps Follow(), val(), and suf() be defined as
     before, relative to S_{(P_{l_min})^r}. We also assume that Q = {0, 1, ..., ℓ − 1},
     where ℓ = |Q|, and that if |val(p)| < |val(q)| then p < q, for any p, q ∈ Q.
 2.  L ← 0^ℓ
 3.  for c ∈ Σ do B[c] ← 0^ℓ
 4.  for p ← ℓ − 1 to 0 do
 5.      for q ∈ Follow(p) do
 6.          H ← 0^{ℓ−1}1 ≪ q
 7.          c ← lbl(p, q)
 8.          B[c] ← B[c] | H
 9.          if q ∈ F then L ← L | H
10.          Φ̇_S[p] ← Φ̇_S[p] | H
11.      if p ≠ 0 then
12.          Φ̇_S[suf(p)] ← Φ̇_S[suf(p)] | Φ̇_S[p]
  /* Searching */
13.  j ← l − 1
14.  while j < n do
15.      k ← 0, last ← 0
16.      D ← 1^ℓ
17.      while D ≠ 0^ℓ do
18.          lead ← ⌊log_2(D & (∼ D + 1))⌋
19.          D ← Φ̇_S[lead] & B[T[j − k]]
20.          if D & L ≠ 0^ℓ then
21.              if k < l then
22.                  last ← k
23.              else Output(j)
24.          k ← k + 1
25.      j ← j + l − last
```

Figure 2: The Backward-Log-And algorithm for the multiple string matching problem.

Acknowledgements

We thank the anonymous reviewers for their helpful comments.

References

[1] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: an aid to bibliographic search. *Commun. ACM*, 18(6):333–340, 1975.

[2] Jörg Arndt. *Matters Computational*. Springer, 2011. `http://www.jjj.de/fxt/`.

[3] Ricardo Baeza-Yates and Gaston H. Gonnet. A new approach to text searching. *Commun. ACM*, 35(10):74–82, 1992.

[4] A. Blumer, J. Blumer, D. Haussler, R. McConnell, and A. Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *J. ACM*, 34(3):578–595, 1987.

[5] Anselm Blumer, Andrzej Ehrenfeucht, and David Haussler. Average sizes of suffix trees and dawgs. *Discrete Appl. Math.*, 24(1-3):37 – 45, 1989.

[6] Domenico Cantone and Simone Faro. A space efficient bit-parallel algorithm for the multiple string matching problem. *Int. J. Found. Comput. Sci.*, 17(6):1235–1252, 2006.

[7] M. Crochemore and W. Rytter. *Text algorithms*. Oxford University Press, 1994.

[8] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2001.

[9] Donald E. Knuth, James H. Morris, Jr, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(1):323–350, 1977.

[10] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences*. Cambridge University Press, 2002.

[11] Gonzalo Navarro and Kimmo Fredriksson. Average complexity of exact and approximate multiple string matching. *Theor. Comput. Sci.*, 321(2-3):283–290, 2004.

[12] Gonzalo Navarro and Mathieu Raffinot. Fast and flexible string matching by combining bit-parallelism and suffix automata. *J. Exp. Algorithmics*, 5:4, 2000.

[13] Gonzalo Navarro and Mathieu Raffinot. New techniques for regular expression searching. *Algorithmica*, 41(2):89–116, 2005.

[14] Sun Wu and Udi Manber. Fast text searching: allowing errors. *Commun. ACM*, 35(10):83–91, 1992.