Backward Pattern Matching Automaton

Jan Antoš and Bořivoj Melichar

Department of Computer Science & Engineering
Faculty of Electrical Engineering
Czech Technical University
Karlovo nám. 13, 121 35 Prague 2

e-mail: {antosj,melichar}@fel.cvut.cz

Abstract. We present a new algorithm to solve a large number of backward pattern matching problems. This algorithm is specified by the theory of finite automata. The algorithm is based on the utilization of a formal tool called "Backward Pattern Matching Automaton", which we specify in this paper. Introduction of such a tool presents a formal base to the world of backward pattern matching.

Keywords: backward pattern matching, string matching, sequence matching, approximate pattern matching, subpattern matching, don't care symbols, finite automata, formal tool

1 Introduction

Pattern matching (string and sequence matching) is an essential part of many applications. This discipline has been intensively studied since the beginning of the seventies and many pattern matching problems have been discovered and extensively studied. A number of new algorithms was presented. Yet these algorithms lack a common theory and are often hard to understand, evaluate and proof. One reason for such a diversity is the nonexistence of a uniform formalism needed for the specification of the problems themselves.

In 1996 a formalism was found, which allows principles of matching algorithms to be formally specified. This formalism is based on a finding, that all one-dimensional matching algorithms are sequential problems and thus can be solved by the use of finite automata [MH97a].

At the same time a classification of matching problems was presented. This classification is not (and cannot be) complete, but it classifies 192 different pattern matching problems in a six-dimensional space [MH97][1]. Together with the new formalism it resulted in an interesting fact: Having a finite automaton to describe the pattern matching problem of one string in a text, all the other 191 problems can be solved by simple operations applied to this one automaton [MH97]. Only a forward matching technique was explored in [MH97] leaving the question open, if similar operations

[1] The number of problems was further increased to 336 in the following years by the addition of approximate matching over well-ordered alphabets.

can be defined to solve all the above mentioned pattern matching problems using the backward pattern matching algorithm.

The motivation of this paper is to present a formal specification of a backward pattern matching automaton which will be used as a model in a general backward pattern matching algorithm. The algorithm itself is simple and general and is the same for any backward pattern matching problem. The only part that is changed is the model of the problem which is fed to the algorithm input. This paper specifies the algorithm and the model and shows the construction of the model for a selected problem. It is important to mention that models for other 336 problems (as well as any future ones) can be obtained from the already defined models by simple operations over the finite automata.

2 Basic Definitions

This paper uses common notions from graph and finite automata theory. Only notions not commonly used, or notions that are specific to this paper are mentioned in this section.

Definition 2.1 (Complement of symbol). *Given an alphabet A and a symbol $a \in A$, the* complement *of a according to A is the set of symbols $\overline{a} = \{s : \ s \in A, s \neq a\}$.*

Definition 2.2 (Proper prefix). *w is a* proper prefix *of P when $w \in pref(P) \ \wedge (wv \in P, v \in A^+)$ which can also be expressed as $w \in pref(P \setminus \{w\})$.*

Definition 2.3 (Move of FA). *A* move *of a finite automaton is such a change of configuration of the finite automaton, that exactly one symbol has been read from the automaton input.*

Remark. *Note the difference between a* move *and a* transition. *While a move is a change of configuration resulting from reading a symbol a transition is a relation $\vdash_M \subset (Q \times A^*) \times (Q \times A^*)$ defined as $(q, aw) \vdash_M (p, w)$ where $p \in \delta(q, a)$, $a \in A \cup \{\varepsilon\}$, $w \in A^*$, $p, q \in Q$. Because an automaton can contain ε-transitions, one move can look for example like: $(q_1, aw) \vdash (q_2, aw) \vdash (q_3, w)$ given $a \in A$, $w \in A^*$, $q_2 \in \delta(q_1, \varepsilon)$, $q_3 \in \delta(q_2, a)$.*

Definition 2.4 (Collection). *A* Ccollection *is a set, that can contain duplicates. We will use symbols [and] to mark the collection.*

Definition 2.5 (Reversed string). *Let us have string $u \in A^*$, $u = a_1 a_2 \ldots a_n$, $a_i \in A$. Then string $v = u^R$, where $v \in A^*$ and $v = a_n a_{n-1} \ldots a_1$, $a_i \in A$ is called* reversed string. *All reversed strings from a set of strings $W \subset A^*$ will be denoted by W^R.*

Remark. *A particular* substring *of a string s, where the substring starts at position i of the string s and ends at positions j (inclusive), will be denoted as $s_i \ldots s_j$.*

3 Problem Specification

3.1 Brief Introduction to Backward Pattern Matching

Backward pattern matching can greatly speed up the pattern matching process because it is capable of skipping parts of the text. Thus we can achieve time complexity

lower then $O(n)$. The main point of backward pattern matching is that the pattern is compared from the right to left. Several techniques exist, this paper is going to explore the BDM method [CR94]. The prefix of the pattern is searched for in the text. When the longest prefix is found, the position in the text is shifted accordingly. The algorithm is therefore skipping parts of the text, where no match can occur. This principle is visualized in Figure 1.

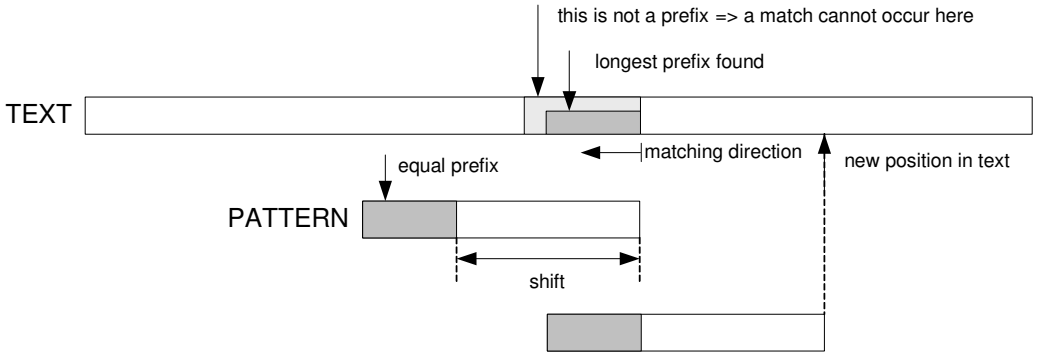

Figure 1: The backward pattern matching principle followed in this paper

3.2 Classification of Pattern Matching Problems

The classification of pattern matching problems has been described in [MH97]. This subsection presents a brief extract of the main ideas. See [MH97] for full details.

Pattern matching problems for a finite size alphabet can be classified according to several criteria. We will use six criteria for classification leading to a six-dimensional space in which one point corresponds to a particular pattern matching problem. Let us make a list of all dimensions including possible values in each dimension:

1. Nature of the pattern: string, sequence.
2. Integrity of the pattern: full pattern, subpattern.
3. Number of patterns: one, finite number, infinite number.
4. The way of matching: exact, approximate matching with Hamming distance (R-matching), approximate matching with Levenshtein distance (DIR-matching), approximate matching with generalized Levenshtein distance (DIRT-matching), approximate with Δ-matching, approximate with Γ-matching, approximate with $max(\Delta, \Gamma)$-matching.
5. Importance of symbols in pattern: take care of all symbols, don't care of some symbols.
6. Number of instances of pattern: one, finite sequence.

If we count the number of possible pattern matching problems, we obtain

$$N = 2 \cdot 2 \cdot 3 \cdot 7 \cdot 2 \cdot 2 = 336.$$

In order to make references to a particular pattern matching problem easy, we will use abbreviations for all the problems. These abbreviations are summarized in Table 1. Using this method, we can, for example, refer to exact string matching of one string as the SFOECO problem.

Instead of a single pattern matching problem we will use the notion of a family of pattern matching problems. In this case we will use symbol ? instead of a particular letter. For example SFO??? is the family of all the problems concerning one full string matching.

We will denote a pattern matching problem by symbol Θ. A pattern matching problem can be then written, for example, as $\Theta = SFOECO$ or $\Theta = SFO???$.

Dimension	1	2	3	4	5	6
	S Q	F S	O F I	E R D T Δ Γ M	C D	O C

Table 1: Abbreviations of dimension values

Remark. *The input to the pattern matching algorithm is text T and pattern set P. Because it is often difficult to define the set P, we will sometimes specify the input to the algorithm using the base pattern set P_Θ and the pattern matching problem Θ. Let us show how these relate to each other on a few examples:*

$\Theta = SFOECO, P_\Theta = \{banana\} \quad \Rightarrow \quad P = \{banana\},$
$\Theta = SSOECO, P_\Theta = \{banana\} \quad \Rightarrow \quad P = \{w : w \in fact(banana)\}.$

To further simplify the notation and to make the text more readable we will use an abbreviation of the above statements. For example:

$P = \{banana\}_{SFORCO} \quad \Rightarrow \quad P = \{w : w \in A^*, D_H(w, banana) \leq k\},$

where D_H is the Hamming distance and k is the maximum distance still considered to be a match.

4 Range of Problems Solved by This Paper

In this paper we will present a general algorithm which is capable of solving all the above mentioned problems. This algorithm should also solve any future problems. The algorithm uses the *Backward Pattern Matching Automaton (BPMA)* which is used as a model of a particular pattern matching problem. Some of these BPMA are presented here as examples. In the case of new pattern matching problems defined in the future, the only task is to define the appropriate BPMA. A very important part of this paper is to show, that these BPMA can be derived from the simplest BPMA for the SFOECO problem (exact pattern matching of one string) only by the simple operations performed over the finite automaton.

5 The Solution

The motivation is to design a simple algorithm which can be applied to a vast range of problems. Such an algorithm has to be independent of the actual problem we are trying to solve. We thus separate the pattern matching into two phases.

Phase One is the "construction phase". The input to Phase One is the type of problem specified by Θ and the set of patterns P that we want to match. The output of Phase One is the model M of the problem Θ applied to the base set of patterns P_Θ. Model M has the form of an attributed nondeterministic finite automaton. Construction of this model is different for different problems, but it has a common base: the basic pattern matching model is constructed first and then automaton operations are applied to it and the final model is derived.

Phase Two is the "matching phase". The input to Phase Two is the model M (the model of the pattern matching problem constructed in Phase one) and the text T. The output of Phase Two is the set of occurrences of patterns $p \in P$ in text T. The automaton M is repeatedly used in the matching phase and attributes of its states and transitions are evaluated. Phase Two is thus completely independent of the problem Θ.

6 Backward Pattern Matching Automaton

Each pattern matching problem can be described using its model in the form of an attributed nondeterministic finite automaton (Definition 6.1). This model is then used in the pattern matching phase.

Definition 6.1 (Attributed Nondeterministic Finite Automaton). Attributed Nondeterministic Finite Automaton (ANFA) M *is five-tuple* $M = (NFA, R, \gamma_q, \gamma_\delta, G)$ *where*

NFA	*is nondeterministic finite automaton* $NFA = (Q, A, \delta, q_0, F)$,
R	*is a finite set of attributes. Every attribute has a domain* $H(r)$ *specifying possible values of attribute* r. $R = R_q \cup R_\delta$,
γ_q	*is a mapping* $Q \times R_q \to H(r) \cup \emptyset$ *where* $r \in R_q$,
γ_δ	*is a mapping* $Q \times Q \times A \times R_\delta \to H(r) \cup \emptyset$ *where* $r \in R_\delta$,
G	*is a finite set of semantic rules of the following form:* $p.r \leftarrow g(q.r_1, \ldots, q.r_n, p.r_1, \ldots, p.r_m, t_{q,p,a}.r_1, \ldots, t_{q,p,a}.r_k)$ *where* $q.r$ *denotes a* $\gamma_q(q, r)$ *and reads as "value of attribute* r *of state* q", $t_{q,p,a}.r$ *denotes* $\gamma_\delta(q, p, a, r)$ *and reads as "value of attribute* r *of transition* $\delta(q, a) \ni p$ ", *and where* $m, n, k \in \mathbb{N}$. *At places, where no confusion arises, we will use* $t.r$ *instead of* $t_{q,p,a}.r$.

In this paper we are going to define ANFA common to the S????O pattern matching problems. We are going to call this automaton BPMA (Backward Pattern Matching Automaton). If more pattern matching problems are to be solved, there might be a need to extend its set of attributes R and/or its set of semantic rules G.

Definition 6.2 (Backward Pattern Matching Automaton). *A* Backward Pattern Matching Automaton (BPMA) M *is an attributed nondeterministic finite automaton* $M = (NFA, R, \gamma_q, \gamma_\delta, G)$ *where*

$NFA = (Q, A, \delta, q_0, F)$,
$L(NFA) = pref(P^R)$, P *is set of patterns,*
$R = R_q \cup R_\delta$,
$R_q = \{tc, plc, pf\}$,
$H(tc) = \mathbb{N}$,
$H(plc) = \mathbb{N}$,
$H(pf) = \{\textsc{True}, \textsc{False}\}$,
$R_\delta = \{ptf\}$,
$H(ptf) = \{\textsc{True}, \textsc{False}\}$,
$G = \{p.tc \leftarrow q.tc + 1$,
$p.pf \leftarrow$ ***if*** $t.ptf = \textsc{True}$ ***then*** $\textsc{True}$ ***else*** $q.pf$,
$p.plc \leftarrow$ ***if*** $q \in F \ \wedge \ p.pf = \textsc{True}$ ***then*** $p.tc$ ***else*** $q.plc$,
$t.ptf$ *is precomputed for all transitions,*
$q_0.tc \leftarrow 0$,
$q_0.pf \leftarrow \textsc{False}$,
$q_0.plc \leftarrow 0$ $\}$ *for* $q, p, t : \ \delta(q, a) \ni p, \ t \sim t_{q,p,a}$.

Let's have a BPMA and

$$(q_0, w^R z^R) \vdash^* (q, z^R), \ q \in Q, \ w, z \in A^*,$$

then we can explain the meaning of BPMA state attributes as follows:

Attribute *tc* is the acronym for *Transition Counter*. This attribute stores the number of automaton moves. This number equals the number of symbols read from the automaton input to reach the current configuration:

$$q.tc = |w|.$$

Attribute *plc* is the acronym for *Prefix Length Counter*. This attribute stores the length of some proper prefix found from the last shift operation. Because the automaton is nondeterministic and several options for $(q_0, w^R z^R) \vdash (q, z^R)$ are possible, the value of $q.plc$ does not have to be the actual longest proper prefix of w, so in the final count, we have to evaluate all of the $q.plc, q \in F$ to find the plc_{max}. The fact that $plc_{max} = |pref(P \backslash \{w\})|_{max}$ has to be assured by the way the model is built. Then we can state that:

$$q.plc \in \{|v| : v, u \in A^*, \ vu = w, \ v \in pref(P \backslash \{v\})\},$$

$$plc_{max} = max\{q.plc : q \in F\}.$$

Attribute *pf* is the acronym for *Prefix Flag*. Attribute *pf* of a state q has value $\textsc{True}$ if from a current automaton configuration every future final configuration reached indicates that a proper prefix of some pattern $p \in P$ has been found. If the value of any final state is $\textsc{False}$ it indicates, that an occurrence of a pattern has been found:

$$q.pf = \textsc{True} \quad \Rightarrow \quad \forall u \in A^*, \ \forall q_f \in F, \ \delta(q, u^R) \ni q_f : uw \in pref(P \backslash \{uw\}),$$

$$q.pf = \textsc{False} \ \wedge \ q \in F \quad \Rightarrow \quad w \in P.$$

The meaning of the BPMA transition attributes can be explained as follows:

Attribute *ptf* is the acronym for *Prefix Transition Flag*. Attribute *ptf* of transition $\delta(q, a) = q',\ q, q' \in Q,\ a \in A^*$ has value TRUE if by an associated move the automaton will move to such a configuration, that any final state reached from there will mean, that we have found a proper prefix of some pattern $p \in P$:

$$t_{q,q',a}.ptf = \text{TRUE} \quad \Rightarrow$$
$$\forall u, v \in A^*,\ \forall q_f \in F,\ \delta(q_0, v^R) \ni q,\ \delta(q', u^R) \ni q_f : uav \in pref(P \backslash \{uav\}).$$

Note, that while some string $w \in P$ can also be a proper prefix $w \in pref(P \backslash \{w\})$, the automaton mentioned above is inherently nondeterministic: both of the following situations can happen at the same time:

$$q_f \in \delta(q_0, w^R) \ \wedge\ q_f.pf = \text{TRUE}$$
$$q_f \in \delta(q_0, w^R) \ \wedge\ q_f.pf = \text{FALSE}.$$

This behavior is wanted in this case because we want to detect both situations simultaneously. We need to know that a pattern occurrence has been found and also we need to know that an occurrence of a proper prefix has been found, so we can compute the appropriate shift function.

See the following sections for examples of backward pattern matching automata.

7 The Algorithm

7.1 Definition of the Algorithm

Phase Two has as input model M of pattern matching problem Θ and the text T in which we want to perform the actual pattern matching. Phase Two performs the matching itself. It consists of the specific backward pattern matching algorithm. This algorithm is simple and unified – the algorithm is the same for all the pattern matching problems defined in 3.2 and possibly for future ones.

The backward pattern matching algorithm is described in Algorithm 1. This algorithm uses a nondeterministic pattern matching model M and therefore it has to simulate its deterministic behavior. Future work is to construct a deterministic pattern matching model and to simplify the backward pattern matching algorithm.

Also notice, that instead of a set of states, the algorithm uses a collection of states. This is required to allow the processing of one state with different attributes – this situation can happen when the automaton has two transitions for the same symbol going from state q to state p and for one transition $t_{q,p,a}.ptf = \text{TRUE}$ and for the second $t_{q,p,a}.ptf = \text{FALSE}$.

Algorithm 1:	AUTOMATON-BASED BACKWARD PATTERN MATCHING ALGORITHM
Input:	Model M in the form of Backward Pattern Matching, Automaton $M = (NFA, R, \gamma_q, \gamma_\delta, G)$, text T.
Output:	Set of numbers, each number represents a position in text T where pattern $p \in P$ occurs.
Method:	

1 $\quad position \leftarrow |P|_{min}$

```
2     offset ← 0
3     plc_max ← 0
4     Q' ← [q_0] (see Definition 2.4)
5     while position ≤ |T| do
6         Q'' ← [q: q ∈ δ(q', T_position−offset), q' ∈ Q']
7         if Q'' ≠ ∅ then
8             for ∀q ∈ Q'' do
9                 if q ∈ F ∧ q.pf = False then
10                    output(position − offset)
11                end if
12                if q ∈ F ∧ q.pf = True then
13                    plc_max ← max{plc_max, q.plc}
14                end if
15            end for
16            Q' ← Q''
17            increment offset
18        else
19            shift ← max{1, |P|_min − plc_max}
20            position ← position + shift
21            offset ← 0
22            plc_max ← 0
23            Q' ← [q_0]
24        end if
25    end while
```

The main idea of the algorithm is as follows:

1. The algorithm computes the initial position.

2. The algorithm utilizes the BPMA automaton in order to decide, if there is some pattern ending at this position, i.e. if

$$\exists x \in \mathbb{N} : T_x \dots T_{position} \ \in \ P.$$

   This event occurs if

$$\exists q_f \in F, \ \exists w \in A^* : \delta(q_0, w^R) \ni q_f \ \wedge \ q_f.pf = \text{False}.$$

   In this case, the value of x is output.

3. Simultaneously with Step 2, the algorithm also has to decide what the longest proper prefix ending at this position is, i.e. it computes $|w|_{max}$ where

$$w \in pref(P \backslash \{w\}) \ \wedge \ w = T_{position-|w|} \dots T_{position}.$$

   This $|w|_{max}$ (named plc_{max} in the algorithm) equals the following expression in BMPA:

$$|w|_{max} = max\{q.plc : q \in F\}.$$

4. When the length plc_{max} of the longest proper prefix is known, the algorithm can attempt to compute the longest safe shift. A safe shift means how much it can advance the position in the text in order not to skip any occurrence of any pattern. The trivial safe shift is 1. It is easy to see, that the longest safe shift can never be longer than the shortest pattern $p \in P$ which is $|P|_{min}$. Since we know, that there is a potential of a pattern occurring at position $position - plc_{max}$, and we know $|P|_{min}$, the shift of $|P|_{min} - plc_{max}$ will be safe. So, summarized, shift can be

$$shift = max\{1,\ |P|_{min} - plc_{max}\}.$$

   Note at this point of time, why we are using the proper prefixes in contrary to traditional prefixes. If we have found string w and $w \in pref(P)$ but $w \notin pref(P \backslash \{w\})$ then $w \in P$. The value of shift would always be 1, which is inefficient in most cases, since there is no possibility of finding another pattern starting at the position $position - |w|$ or $position - |w| + 1$.

   The longest safe shift can be longer than the one mentioned in previous paragraphs. The idea of a longer safe shift is to select the shortest pattern that can start with the prefix (or prefixes) ending at the current position (w in step 3). In most cases this number can be higher than $|P|_{min}$. Let us compute P' based on that finding:

$$P' = \{p;\ p \in P, w \in pref(p)\}.$$

   The longest safe shift is then

$$shift = max\{1,\ min\{|P|_{min},\ |P'|_{min} - plc_{max}\}\}.$$

   This optimization is not employed in the current algorithm. It should be included in future works.

5. The algorithm advances its position by the $shift$ value:

$$position \leftarrow position + shift$$

   and the algorithm repeats steps 2 through 5 of this explanation until the end of the text is reached.

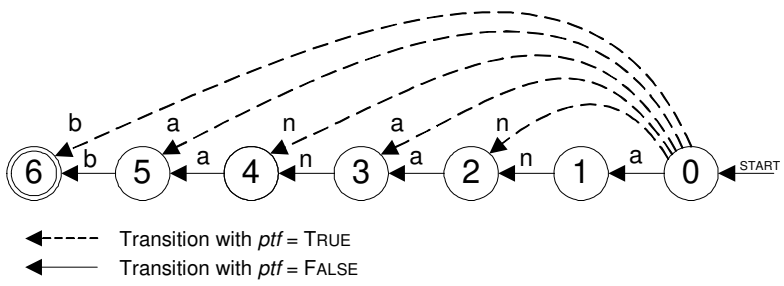

Figure 2: Transition diagram of BPMA which is a model of pattern matching problem $\Theta = SFOECO$ and pattern set $P_\Theta = \{banana\}$

7.2 Example

Let us demonstrate Algorithm 1 on a simple example. A more advanced example is given in Section 8.

Let us have a pattern matching problem $\Theta = SFOECO$, pattern set $P = \{banana\}$ and text $T = banabbababnananbananaba$. The model M of this problem is the nondeterministic pattern matching automaton given by the transition diagram specified in Figure 2. The algorithm steps are shown in Figure 3.

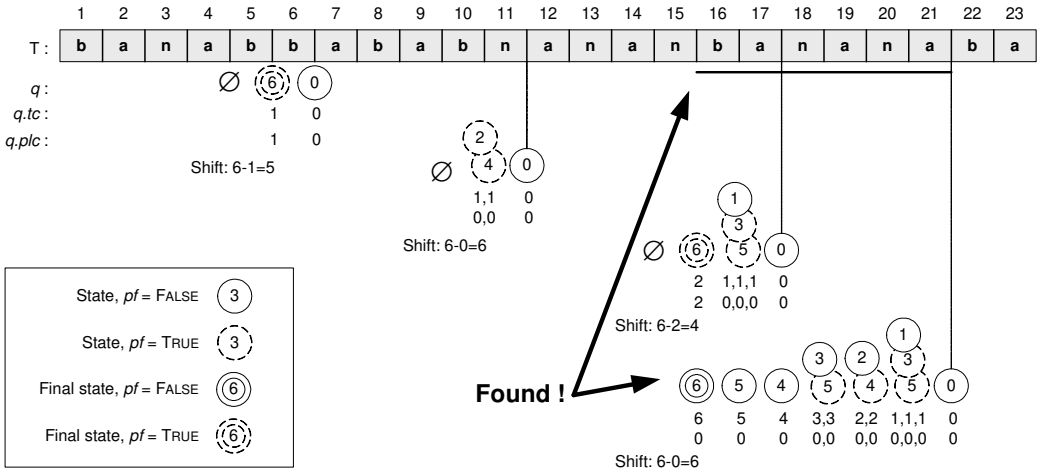

Figure 3: Steps taken during the pattern matching of $P_{SFOECO} = \{banana\}$ in text $T = banababababnananbananaba$

8 BPMA Construction

The construction Phase (i.e. Phase One) is dependent on the Backward Pattern Matching Problem solved. The output of Phase One is the uniform model of the problem. This model is in the form of a BPMA. Phase Two then uses this model to perform the actual pattern matching.

We will demonstrate the construction of such a BPMA on a selected example: Let us have a problem $\Theta = SFORCO$ (i.e. approximate R-matching of one pattern). R-matching means approximate matching where the operation "replace" is allowed. This kind of approximate matching was first explored by Hamming in [HAM50].

Let us have a base pattern set $P_\Theta = \{banana\}$ and $k = 1$. We can express P as

$$P = \{w : D_H(p, w) \leq k,\ p \in P_\Theta, w \in A^*\},$$

where k denotes the maximum distance between two patterns that we consider being equal (and thus representing a match in the text).

We first construct the *base nondeterministic finite automaton* which accepts the language $L = P^R$. We can build this automaton incrementally using the given 6D classification as an advantage: we can start with the base SFOECO problem first and then add the complexity dimension by dimension. In our case there will be only one more step necessary and it is to change the choice of value in the 4^{th} dimension: SFOECO $\rightarrow$ SFORCO.

We first build the base automaton M_1 for SFOECO problem: $P_{SFOECO} = \{banana\}$. The language accepted by this automaton is $L(M_1) = \{ananab\}$.We will use Algorithm 2. The result of this algorithm is given in Figure 4.

Algorithm 2: CONSTRUCTION OF SFOECO BASE NFA
Input: Pattern p, $|p| = m$, $p \in A^*$.
Output: Deterministic finite automaton M.
Method:

1 $Q \leftarrow \{q_0, q_1, \ldots, q_m\}$
2 $\delta(q_i, p_{m-i}) \leftarrow \{q_{i+1}\}$ for all $i = 0, 1, \ldots, m-1$
3 $F \leftarrow \{q_m\}$
4 $M \leftarrow (Q, A, \delta, q_0, F)$

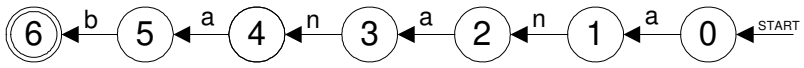

Figure 4: Transition diagram of automaton M_1, which is base NFA for $\Theta = SFOECO$ and $P_\Theta = \{banana\}$

The next step is to construct the base automaton M_2 for our chosen SFORCO problem. We use the already built automaton M_1 and modify it to recognize the language $L(M_2) = \{w : D_H(ananab, w) \leq 1,\ w \in A^*\}$. This is done employing Algorithm 3. The result is shown in Figure 5.

Algorithm 3: CONSTRUCTION OF SF?R?O BASE NFA FROM SF?E?O BASE NFA
Input: Nondeterministic finite automaton $M_{SF?E?O} = (Q, A, \delta, q_0, F)$.
Output: Nondeterministic finite automaton $M_{SF?R?O}$.
Method:

1 $Q' \leftarrow \emptyset,\ F' \leftarrow \emptyset$
2 **for** $\forall l \in \langle 0, k\rangle$ **do**
3 $\quad Q' \leftarrow Q' \cup \{q_{l,i} : q_i \in Q\}$
4 $\quad \delta'(q_{l,i}, a) \leftarrow \delta'(q_{l,i}, a) \cup \{q_{l,j} : q_j \in \delta(q_i, a)\}$ for all $a \in A,\ q_i \in Q$
5 $\quad F' \leftarrow F' \cup \{q_{l,i} : q_i \in F\}$
6 **end for**
7
8 **for** $\forall l \in \langle 0, k-1\rangle$ **do**
9 $\quad \delta'(q_{l,i}, \overline{a}) \leftarrow \delta'(q_{l,i}, \overline{a}) \cup \{q_{l+1,j} : q_j \in \delta(q_i, a)\}$ for all $a \in A,\ q_i \in Q$
10 **end for**
11
12 $M_{SF?R?O} \leftarrow (Q', A, \delta', q_{0,0}, F')$

After this step we are ready to build the BPMA itself. We can use a general Algorithm 4. This algorithm constructs the BPMA from the given base NFA, where the base NFA can represent any of the problems solvable by the BPMA itself. The resulting automaton M is given in Figure 6.

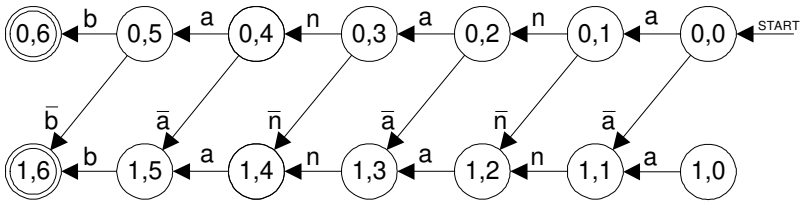

Figure 5: Transition diagram of base NFA for $\Theta = SFORCO$, $k = 1$ and $P_\Theta = \{banana\}$

Algorithm 4: Construction of BPMA from given NFA
Input: Nondeterministic finite automaton $M_{NFA} = (Q, A, \delta, q_0, F)$.
Output: Backward pattern matching automaton M_{BPMA}.
Method:

1 **if** $\exists q \in Q, \exists a \in A : q_0 \in \delta(q, a)$ **then**
2 $Q' \leftarrow Q \cup \{q_{00}\},\ q'_0 \leftarrow q_{00}$
3 $\delta'(q, a) \leftarrow \delta(q, a)$ for all $q \in Q, a \in A$
4 $\delta'(q_{00}, a) \leftarrow \delta(q_0, a)$ for all $a \in A$
5 **else**
6 $Q' \leftarrow Q,\ q'_0 \leftarrow q_0$
7 $\delta'(q, a) \leftarrow \delta(q, a)$ for all $q \in Q, a \in A$
8 **end if**
9 $M'_{NFA} \leftarrow (Q', A, \delta', q'_0, F)$
10
11 $\delta''(q', a) \leftarrow \delta'(q', a)$ for all $q' \in Q', a \in A$
12 $\delta''(q'_0, a) \leftarrow \bigcup_{q' \in Q' \setminus \{q'_0\}} \delta'(q', a)$ for all $a \in A$
13 $M''_{NFA} \leftarrow (Q', A, \delta'', q'_0, F)$
14
15 $t_{q,q',a}.ptf \leftarrow$ True for all $a \in A,\ q' \in \delta(q, a),\ q \in Q$
16 $t_{q'_0,q',a}.ptf \leftarrow$ True for all $a \in A,\ q' \in \delta'(q'_0, a)$
17 $t_{q'_0,q',a}.ptf \leftarrow$ False for all $a \in A,\ q' \in \bigcup_{q' \in Q' \setminus \{q'_0\}} \delta'(q', a)$
18 $M_{BPMA} \leftarrow (M''_{NFA}, R, \gamma_q, \gamma_\delta, G)$

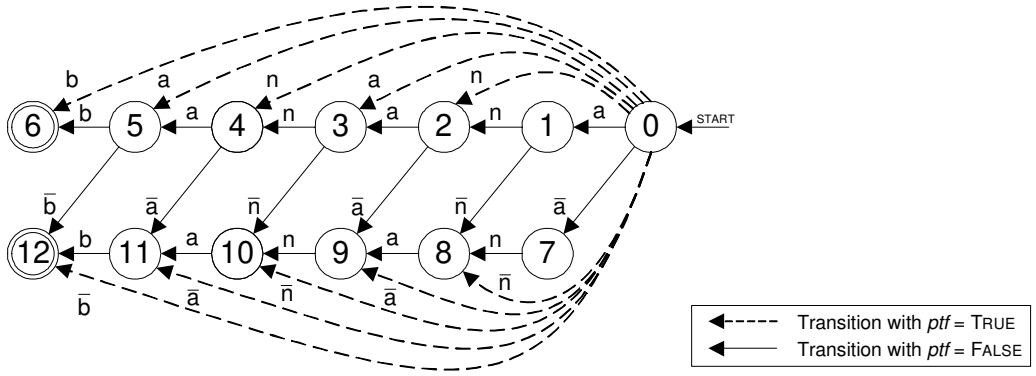

Figure 6: Transition diagram of model M of pattern matching problem $\Theta = SFORCO$ and pattern set $P_\Theta = \{banana\}$

We can now feed the resulting automaton M to Phase Two to perform the actual pattern matching. Let us take text $T =$*is it banana or ananas?* and run the Algorithm 1. The visualization of this process is presented in Figure 7.

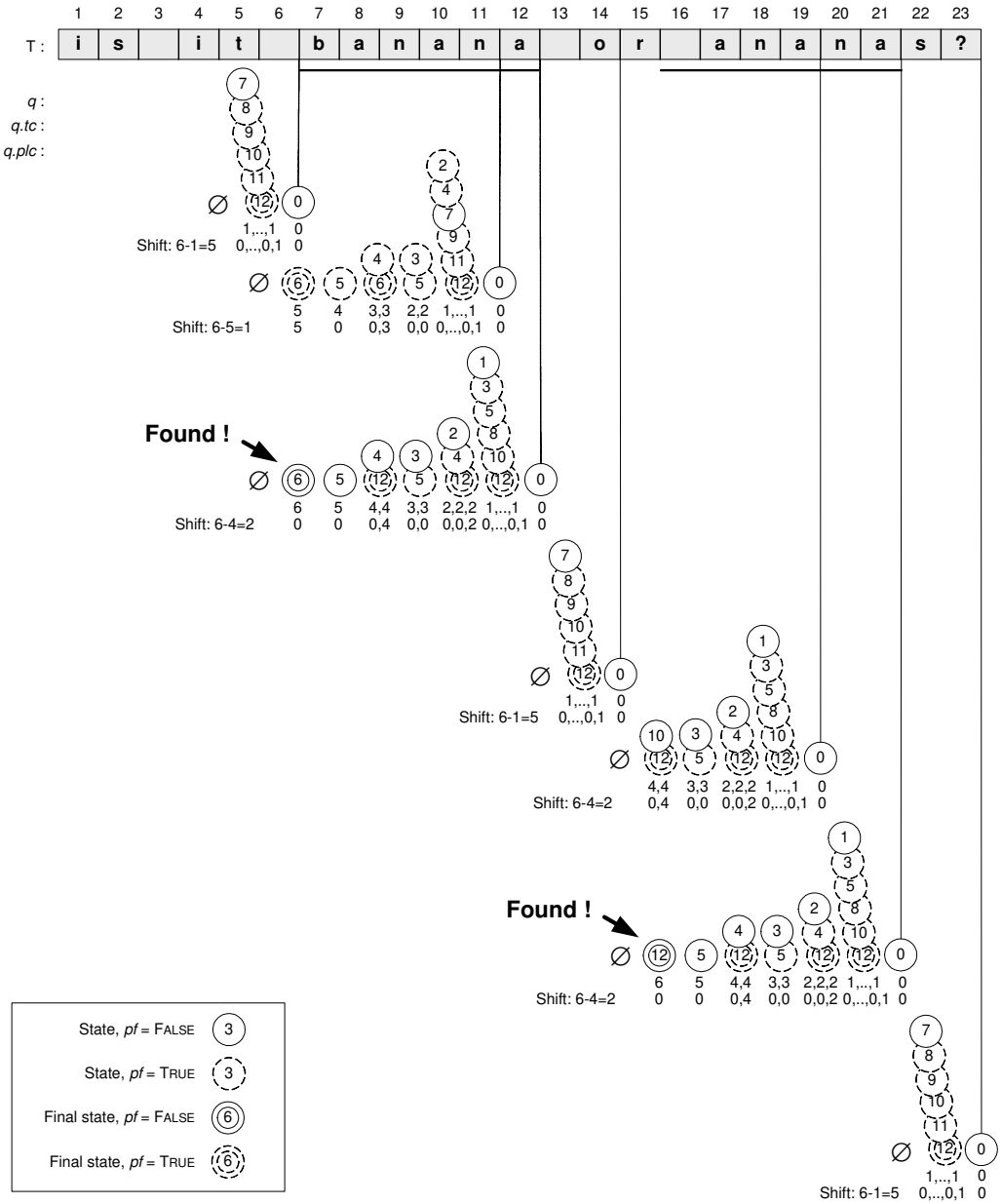

Figure 7: Pattern matching of $P = \{banana\}_{SFORCO}$ in text $T =$ *is it banana or ananas?* from the example

9 Future Work

The presented algorithm posses some drawbacks, that have to be solved in future work:

1. The finite automaton used to specify the pattern matching algorithm is nondeterministic and an equivalent deterministic automaton cannot be constructed by any known algorithm, because the automaton is attributed. To resolve this issue, a new algorithm constructing the equivalent attributed deterministic automaton has to be invented.

2. The longest safe shift computed by the current algorithm is not optimal. This shift can be further optimized by the observation mentioned at the end of Section 7.1 (Step 4).

3. The pattern matching algorithm presented in this report has the upper bound of its time complexity set higher than $O(n)$, where n is the length of text. The upper bound can be theoretically lowered to $O(n)$ but this optimization is yet to be found.

References

[BM77] R. S. Boyer, J. S. Moore: *A fast string searching algorithm.* C. ACM, Vol. 20, No. 10, pp. 762-772, October 1977.

[MH97] B. Melichar, J. Holub: *6D Classification of Pattern Matching Problems.* Proceedings of the Prague Stringology Club Workshop '97, July 1997, pp. 24-32.

[MH97a] B. Melichar, J. Holub: *Pattern Matching and Finite Automata.* In Proceedings of Summer School of Information Systems and Their Applications 1998, Ruprechtov, Czech Republic, September 1998, pp. 154-183.

[HAM50] R. W. Hamming: *Error-detecting and error-correcting codes.* Bell System Technical Journal 29:2, 1950, pp. 147-160.

[CR94] M. Crochemore, W. Rytter: *Text Algorithms.* Oxford University Press, 1994.