Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

TEXT SEARCHING ALGORITHMS

VOLUME II:
BACKWARD STRING MATCHING

Bořivoj Melichar

March 2006

Preface

There are two basic principles of pattern matching:

1. Forward pattern matching.
2. Backward pattern matching.

We covered the forward pattern matching in Volume I in great detail. In this Volume we will show the methods of backward pattern matching.

We use, in this Volume, some principles, notions and algorithms presented in Volume I. Let us list important notions used in this Volume:

- finite automata, their properties and operations with them,
- d-subsets created during determinisation of nondeterministic automata,
- depth of state of acyclic finite automaton,
- prefix, suffix, factor, and factor oracle automata, their properties and construction,
- backbone of suffix or factor automaton,
- terminal state of prefix, suffix, factor, and factor oracle automata,
- border of a string and its computation,
- repetition in a string, repetition table and its construction.

Prague, March 2006 Bořivoj Melichar

Contents

7 Exact backward matching of one pattern

The exact backward pattern matching approach of one pattern is discussed in this Chapter. The text and the pattern are matched in the backward direction. It means that the comparison of symbols is performed from right to left. Surprisingly, backward and forward pattern matching are not symetrical for pattern. The main difference of backward pattern matching with respect to forward pattern matching and the main advantage of backward pattern matching is that in can be faster. This property follows from the fact that some symbols in the text need not be inspected during matching.

7.1 Elementary algorithm

The elementary algorithm compares all symbols of the pattern with symbols of the text. The principle of this approach is shown in Fig. 7.1. This algorithm performs the exact backward matching of one pattern. When the

```
var TEXT : array[1..N] of char;
    PATTERN : array[1..M] of char;
    I, J: integer;
    ...
    begin
     I := 0;
     J := M;
     while I ≤ N − M do
      begin
       while (J > 0) and (TEXT[I + J] = PATTERN[J]) do J := J − 1;
       if J = 0 then
       begin
        output (I + 1);
        J := M;
       end;
(∗)    I := I + 1; {length of shift=1}
      end;
    end;
```

Figure 7.1: The elementary algorithm for backward exact pattern matching of one pattern

pattern is found then the value of variable I is the position just before the first symbol of the occurrence of the pattern in the text. The pattern is then "shifted" one position to the right. Meaning of variables in the elementary algorithm is shown in Fig. 7.2.

We will use the number of comparisons of symbols (see expression $TEXT[I + J] = PATTERN[J]$) as the measure of the time complexity of the algorithm. The maximal number of symbol comparisons is $NC = n * m$,

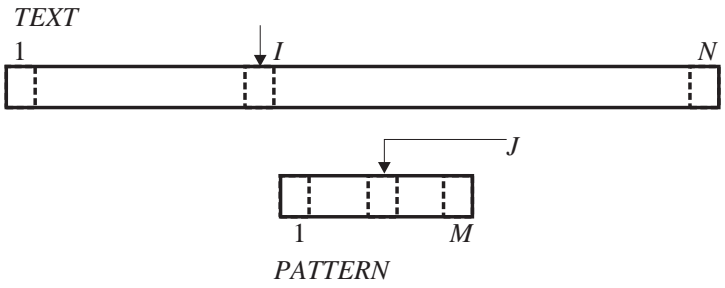

Figure 7.2: Meaning of variables in the program from Fig. 7.1

where n is the length of text and m is the length of the pattern. The time complexity is $\mathcal{O}(n*m)$. The maximal number of comparisons NC is reached for text $T = a^{n-m}ba^{m-1}$ and for pattern $P = ba^{m-1}$. The elementary algorithm has no extra space requirements.

The use of the elementary algorithm for matching of a finite set of patterns is also possible. In this case, the algorithm is used for each pattern separately. The time complexity is $\mathcal{O}(n * \sum_{i=1}^{p} m_i)$, where p is the number of patterns in the set and m_i is the length of i–th pattern, $i = 1, 2, \ldots, p$. The next variant of the elementary algorithm is for the backward approximate pattern matching using Hamming distance. It is shown in Fig. 7.3. The time complexity is again $\mathcal{O}(n * m)$. The maximal number of comparisons NC is reached for text $T = a^{n-m}b^k a^{m-k}$ and for pattern $b^k a^{m-k}$. This algorithm has low extra space requirements. The only additional integer variables ($NERR$,K) are used.

7.2 Backward pattern matching automata for one pattern

In the elementary algorithms (see Fig. 7.1 and 7.3), the length of shift of the pattern is always equal to one. The shift can be greater if we take into account the periodicity inside the pattern. The simplest method how to enlarge the shift is a heuristics called "bad character shift". This heuristics is used in the Boyer–Moore–Horspool (BMH) algorithm. The BMH algorithm uses the distance of symbols from the end of pattern. Longer shifts can be achieved using longer parts of the pattern than one symbol. We present four principles, how to enlarge the shift:

1. To look for the occurence of the rightmost inspected symbol of the text in the pattern (see Section 7.2.1).
2. To look for a repeated suffix of the pattern (see Section 7.2.2).
3. To look for a prefix of the pattern (see Section 7.2.3).
4. To look for an antifactor of the pattern (see Section 7.2.5).

These principles are depicted in Figs. 7.4 - 7.7.

```
var TEXT : array[1..N] of char;
    PATTERN : array[1..M] of char;
    I, J, K, NERR, SHIFT: integer;
    ...
    K := {number of errors allowed};
    begin
     SHIFT := 1;
     I := 0;
     while I ≤ N − M do
      begin
       J := M;
       NERR := 0;
       while NERR ≤ K and (J > 0) do
        begin
         if (TEXT[I + J] ≠ PATTERN[J]) then NERR := NERR + 1;
         J := J − 1;
        end;
        if J = 0 then output (I + 1);
       I := I + SHIFT;
      end;
    end;
```

Figure 7.3: The elementary algorithm for the backward approximate matching of one pattern using the Hamming distance

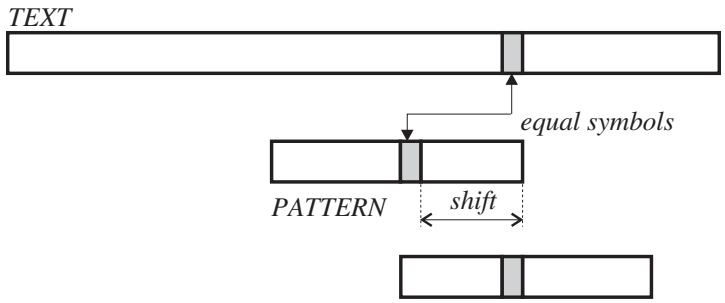

Figure 7.4: Principle of looking for the rightmost repetition of the last inpected symbol in the text

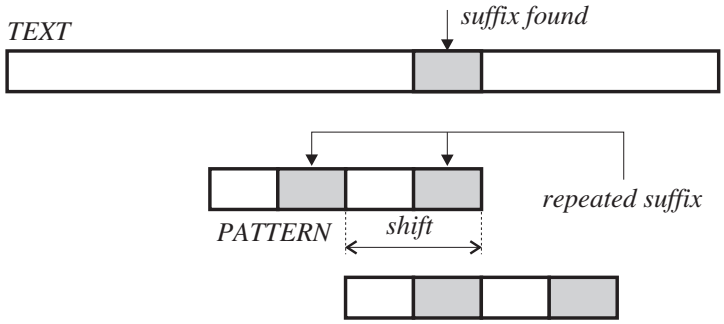

Figure 7.5: Principle of looking for a repeated suffix of the pattern

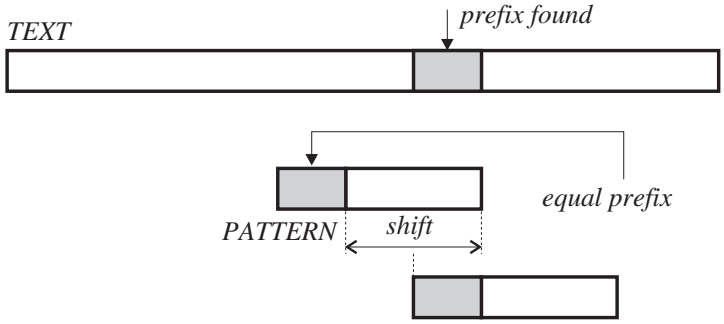

Figure 7.6: Principle of looking for a prefix of the pattern

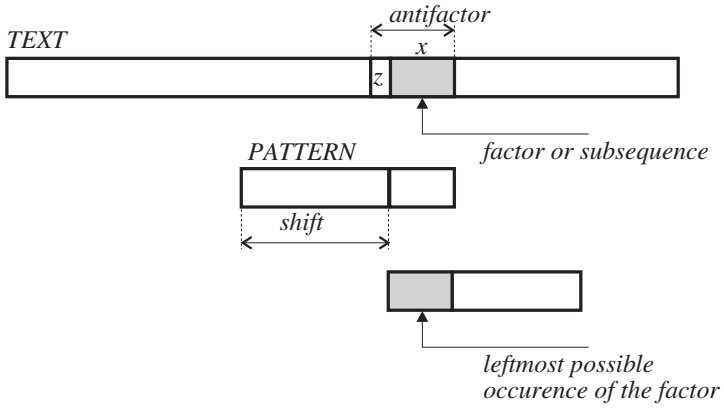

Figure 7.7: Principle of looking for an antifactor of the pattern

The length of shift is shown for all cases. Moreover we present classical Boyer–Moore algorithm which is a combination of several approaches (see Section 7.2.4). Suffix automata, factor automata and factor oracle automata for reversed patterns can serve as models for all these principles. All these automata are described in Volume. I. The length of shift is central notion in all algorithms presented here. It is important, in this context, computation of shift when the pattern is found. In some cases it is the special shift and it is called *matchshift*. The matchshift is computed in this way:

$matschshift := m - length(Border(P))$,

where P is the pattern having length m,

$Border(P)$ is the longest border of P.

The matchshift is shown in Fig. 7.8.

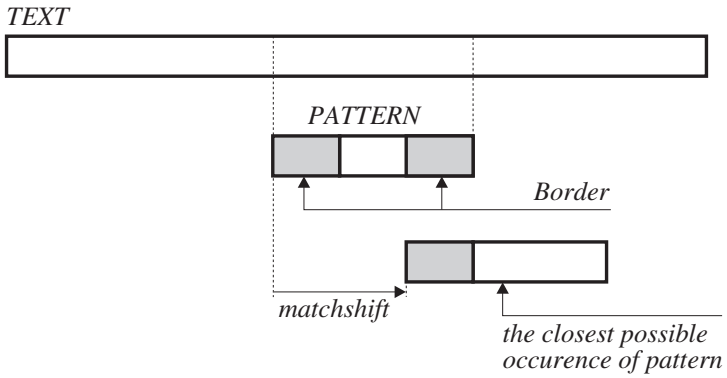

Figure 7.8: Principle of matchshift

The matchshift ensures that the closest occurence of the pattern after its match can be found.

7.2.1 *BMH* algorithm

BMH (Boyer-Moore-Horspool) algorithm compares symbols of pattern and symbols of text from right to left starting from the last symbol of the pattern. As soon as a mismatch occurs then the shift is performed. The shift is given by the distance of occurence of the rightmost inspected symbol in the text from the right end of the pattern. This distance must be greater than zero. For symbols absent in the pattern the shift is equal to the length of the pattern. To explain it consistently with the approach used below a factor automaton is used as the base for the computation of "bad character shift" (*BCS*) table.

Algorithm 7.1
Computation of *BCS* table.
Input: Pattern $P = p_1p_2 \dots p_m,\ p_1, p_2, \dots, p_m \in A$.

Output: BCS table.
Method:

1. Construct a nondeterministic factor automaton for the reversed pattern P^R.
2. Construct the first row of the transition table of the deterministic factor automaton.
3. Select for each symbol $a \in A$ the state q having the shortest distance *dist(q)* from the end of the pattern greater than 0.
   Set $BCS(a) = dist(q) - 1$.
4. If some symbol $b \in A$ does not occur in the pattern, set $BCS(b) = m$.

□

Example 7.2
Let pattern be $P = abcab$ and alphabet $A = \{a, b, c, d\}$. Let us compute the BCS table.

1. The nondeterministic factor automaton for the reversed pattern $P^R = bacba$ has the transition diagram depicted in Fig. 7.9

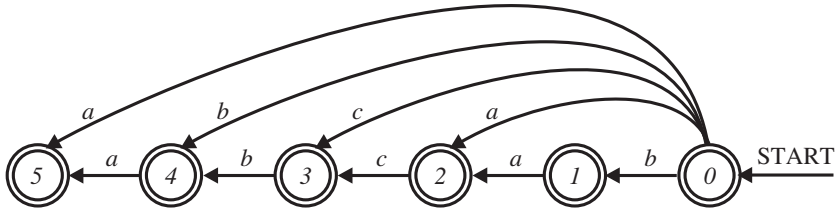

Figure 7.9: Transition diagram of the nondeterministic factor automaton for pattern $P^R = bacba$ from Example 7.2

2. As we use the distances from the end of the pattern as labels of states then $dist(q) = q$. The first row of the transition table of the deterministic factor automaton has the following form:

δ	a	b	c	d
0	25	14	3	

3. The BCS table has the following form:

	a	b	c	d
BCS	1	3	2	5

Note, that $BCS(d) = 5$, because symbol d does not occur in the pattern (see step 4.).

□

Now we can make the change in the elementary algorithm (see Fig. 7.1) in order increase the shift:

Replace statement

(∗) $I := I + 1;$

by statement

$I := I + BCS[TEXT[I + M]];$

It means that the shift depends on the rightmost symbol in text aligned to the rightmost symbol of the pattern. If pattern is found then the shift is given by $BCS[TEXT[I + M]]$. The next Example shows the backward pattern matching using the BCS table.

Example 7.3
Let pattern be $P = abcab$ (see Example 7.2) and text $T = bcbababcabab$. The backward pattern matching performs following steps:

$b\,c\,\underline{b}\,a\,b\,a\,b\,c\,a\,b\,a\,b$	*mismatch* $(c \neq b)$
$a\,b\,\underline{c}\,a\,b$	$BCS[b] = 3$
$b\,c\,b\,a\,b\,a\,b\,\underline{c}\,a\,b\,a\,b$	*mismatch* $(b \neq c)$
$a\,b\,c\,a\,\underline{b}$	$BCS[c] = 2$
$b\,c\,b\,a\,b\,a\,b\,c\,a\,b\,a\,b$	*match*
$a\,b\,c\,a\,b$	$BCS[b] = 3$

The matching ends because the shift is behind the end of the text. Fig. 7.10 shows the visualisation of the behaviour of the BMH algorithm for pattern $P = abcab$. Note that some symbols of the text are not inspected. □

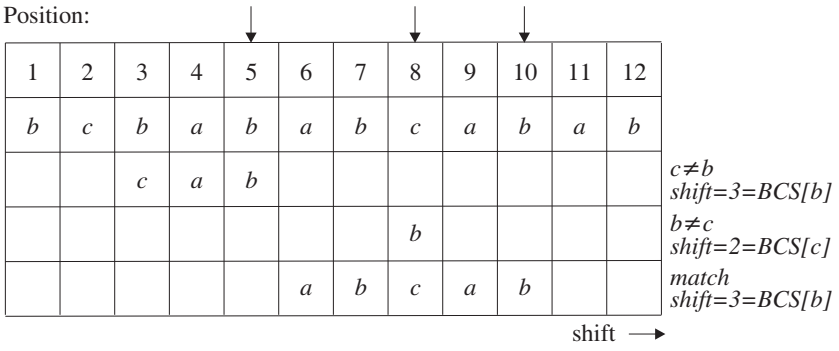

Figure 7.10: Behaviour of BMH algorithm for pattern $P = abcab$ from Example 7.3

7.2.2 Looking for repeated suffixes of the pattern

The basic tool for this approach, looking for repeated suffixes of the pattern, is a backbone of the factor automaton for the reversed pattern. Such automaton is able to identify repeated suffixes of the pattern. This heuristics is called "good suffix shift" (GSS). Let us show this principle using an example.

Example 7.4
Let pattern be $P = baaba$. The reversed pattern is $P^R = abaab$. We construct factor automaton M_1 for this reversed pattern. Transition diagram of nondeterministic factor automaton M_1 is depicted in Fig. 7.11. All its

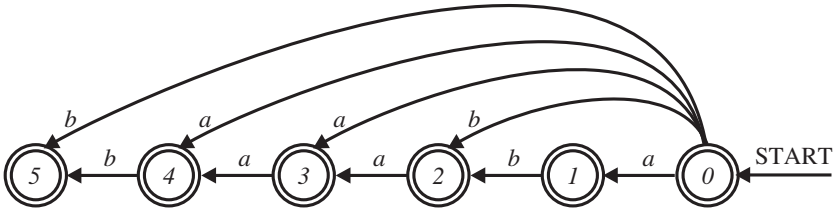

Figure 7.11: Transition diagram of nondeterministic factor automaton M_1 for reversed pattern $P^R = abaab$ from Example 7.4

states are final states. The next step is construction of equivalent deterministic factor automaton M_2. During this construction we save created d–subsets. Transition tables of both nondeterministic factor automaton M_1 and its deterministic equivalent M_2 are shown in Table 7.1. Transition di-

	a	b
0	$1, 3, 4$	$2, 5$
1		2
2	3	
3	4	
4		5
5		

a) transition table of M_1

	a	b
0	134	25
134	4	25
25	3	
3	4	
4		5
5		

b) transition table of M_2

Table 7.1: Transition tables of factor automata M_1 and M_2 from Example 7.4

agram of deterministic factor automaton M_2 is depicted in Fig. 7.12. To

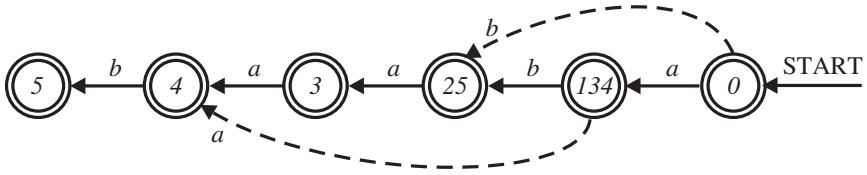

Figure 7.12: Transition diagram of deterministic factor automaton M_2 for $P^R = abaab$ from Example 7.4

obtain backbone of factor automaton M_2, we remove transitions from state 0 to state 25 for input symbol b and from state 134 to state 4 for input

symbol a. Now we construct the suffix repetition table. It has the following form:

d-subset	Suffix	Repetitions
134	a	$(1, F), (3, G), (4, S)$
25	ba	$(2, F), (5, G)$

□

Backbone of factor automaton M_2 starting from the initial state 0 is reading text from right to left. As soon as a mismatch occurs, the automaton is starting to read the text from the position which is given by the previous position and the shift. The length of shift is the distance between the found suffix and its closest repetition (see Fig. 7.5). If the repetition of the suffix does not exist, then the shift is given by $m - \mathit{length}(\mathit{Border}(P))$, where m is the length of the pattern, and $\mathit{Border}(P)$ is the longest border of P. The shift in the case of the mismatch in the initial state is equal to 1. The table of shifts from Example 7.4 for states in which the mismatch occurs is shown in Table 7.2.

State	Suffix	GSS	J
0	ε	1	5
134	a	2	4
25	ba	3	3
3	aba	3	2
4	$aaba$	3	1
5	$baaba$	3	0

Table 7.2: Table of shifts (good suffix shifts, GSS) for pattern $P = baaba$ from Example 7.4

The elementary algorithm can be used for this type of the backward pattern matching (see Fig. 7.1). The only required change is replacement of statement

$(*)$ $I := I + 1;$
by statement
$I := I + GSS[J];$

where GSS is the "good suffix shift". The good suffix shift depends on the distance between end of the pattern and the rightmost repetition of the found suffix. Variable J expresses a position in the pattern. For each position it is possible to identify the rightmost longest repeating suffix and the distance of its end from the end of the pattern. In Table 7.2 the possible values of J are included.

Example 7.5
Let text T be $T = ababbababab$ and pattern $P = baaba$. The suffix automaton from Example 7.4 performs following sequence of moves:

$TEXT: a\,b\,a\,b\,b\,a\,b\,a\,a\,b\,a$	*state,action*	J
$b\,a\,a\,b\,a$	0,*mismatch,shift* $= 1$	5
$b\,a\,a\,b\,a$	0,*match*	5
	134,*match*	4
	25,*mismatch,shift* $= 3$	3
$b\,a\,a\,b\,a$	0,*match*	5
	134,*mismatch,shift* $= 1$	4
$b\,a\,a\,b\,a$	0,*mismatch,shift* $= 1$	5
$b\,a\,a\,b\,a$	0,*match*	5
	134,*match*	4
	25,*match*	3
	3,*match*	2
	4,*match*	1
	5,*pattern found,shift* $= 3$	0

In this situation the pattern matching ends, because the shift is behind the text. The behaviour of the suffix automaton using *GSS* shifts is visualised in Fig. 7.13. Backbone of the factor automaton M_2 starts allways

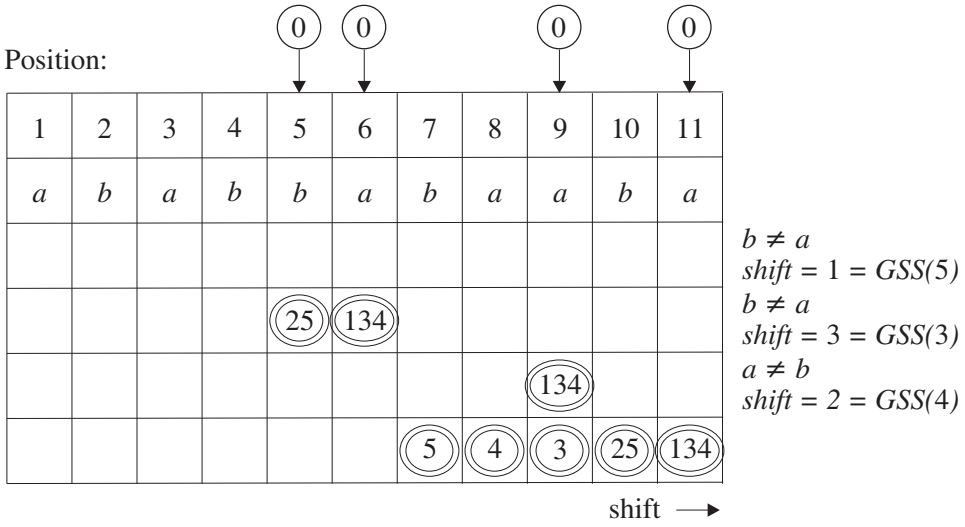

Figure 7.13: Behaviour of the suffix automaton for reversed pattern $P^R = baaba$ using *GSS* table from Example 7.5

in state 0. The matching starts at position 5. Because in the initial state mismatch occurs, the shift is equal to 1 and the next starting position is position 6.

7.2.3 Looking for prefixes of the pattern

We present two methods based on the principle of looking for prefixes of the pattern:

- backward *DAWG* matching (*BDM*),
- reduced good prefix shift (*RGPS*).

7.2.3.1 Backward *DAWG* matching (*BDM*) The basic tool for the approach is a suffix automaton for the reversed pattern. This automaton is able to identify all suffixes of the reversed pattern, e.g. it is able to identify all prefixes of the pattern while reading the text backwards. As soon as the mismatch occurs, pattern is shifted and the automaton starts to read the text from the position which is given by the previous position and the shift. The heuristics is called "good prefix shift" (*GPS*). Let us show this principle using an example.

Example 7.6
Let pattern P be $P = abaab$ over alphabet $A = \{a, b, c\}$. The reversed pattern is $P^R = baaba$. We construct nondeterministic suffix automaton S_N for this reversed pattern. Transition diagram of nondeterministic suffix automaton S_N is depicted in Fig. 7.14. The next step is the construc-

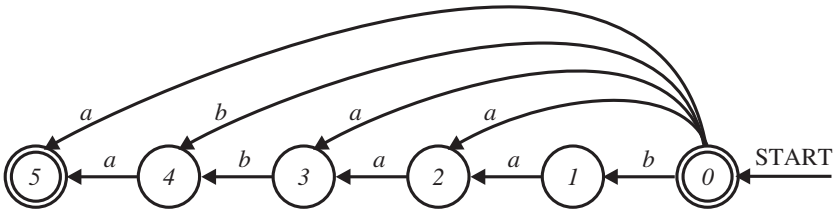

Figure 7.14: Transition diagram of nondeterministic suffix automaton S_N for $P^R = baaba$ from Example 7.6

tion of deterministic suffix automaton S_D. During this construction we save created d–subsets. Transition table of this automaton is shown in Table 7.3. Transition diagram of deterministic suffix automaton S_D is depicted in Fig. 7.15. This automaton is able to identify following set of prefixes of P:
$Pref(P) = \{\varepsilon, a, ab, aba, abaa, abaab\}$. □

The suffix automaton is therefore used for the identification of the longest prefix of the pattern while reading the text backwards. As soon as the mismatch occurs, the automaton starts to read the text from the position which is given by the previous position and the shift. The length of the shift is the difference between the length of the pattern and the length *lprefix* of the longest found prefix: *shift* := m – *lprefix*. Let us note that

	a	b	c
0	235	14	
14	25		
25	3		
235	3	4	
3		4	
4	5		
5			

Table 7.3: Transition table of deterministic suffix automaton S_D for reversed pattern $P^R = baaba$ from Example 7.6

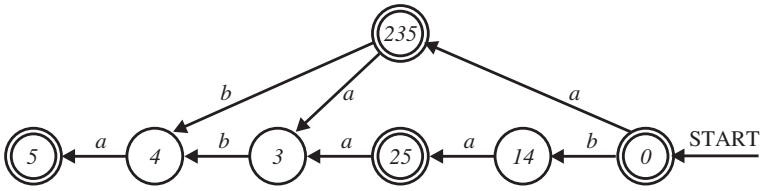

Figure 7.15: Transition diagram of deterministic suffix automaton S_D for $P^R = baaba$ from Example 7.6

lprefix is not the number of symbols read, but can be less or equal to this number. Fig. 7.16 shows the behaviour of the suffix automaton for the reversed pattern. The arrows (↓) show the position after each shift. In position 20 the pattern is found. The previous prefix has length 4 and it can be seen that this already found prefix is not necessary to match it again. This is the source of an optimisation.

The *BDM* pattern matching algorithm shown in Fig. 7.17 uses the suffix automaton for the reversed pattern, length m of the pattern and *matchshift* as parameters. The *matchshift* is the shift performed after reporting the match and it is equal to the length of the pattern minus the length of the Border (the longest border) of the pattern. Moreover, it uses variable *lprefix* which is the length of the longest prefix found since the recent shift.

The configuration of Algorithm *BDM* is fourtuple
(q,I,J,*lprefix*),
where q is a state of automaton M. The initial configuration is $(q_0, 0, m, 0)$.

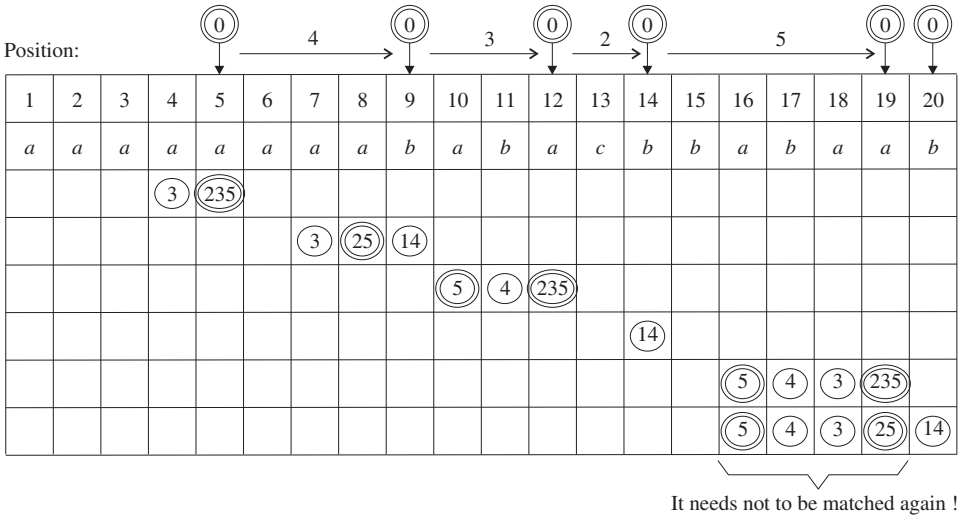

Figure 7.16: Behaviour of suffix automaton S_D for reversed pattern $P^R = baaba$ from Example 7.6

Example 7.7
Let us use the suffix automaton for reversed pattern $P^R = baaba$ from Example 7.6 and let us show the matching in text:

$T =$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	*b*	*a*	*a*	*a*	*b*	*a*	*b*	*a*	*a*	*b*	*b*	*a*	*b*

BDM algorithm performs the following sequence of steps:

$$
\begin{aligned}
(0,0,5,0) &\overset{a}{\vdash} (235,0,4,1)\\
&\overset{a}{\vdash} (3,0,3,1)\, shift\ 4 = m - lprefix = 4\\
&\vdash (0,4,5,0)\\
&\overset{a}{\vdash} (235,4,4,1)\\
&\overset{b}{\vdash} (4,4,3,1)\\
&\overset{a}{\vdash} (5,4,2,3)\, shift = m - lprefix = 2\\
&\vdash (0,6,5,0)\\
&\overset{b}{\vdash} (14,6,4,0)\\
&\overset{a}{\vdash} (25,6,3,2)\\
&\overset{a}{\vdash} (3,6,2,2)\\
&\overset{b}{\vdash} (4,6,1,2)
\end{aligned}
$$

$\overset{a}{\vdash} (5,6,0,5) match\ 7, shift\ = matchshift = 3$
$\vdash (0,9,5,0)$
$\overset{b}{\vdash} (14,9,4,0)$
$\overset{a}{\vdash} (25,9,3,2) shift = m - lprefix = 3$
$\vdash$ stop $\square$

BDM algorithm performs the comparison of 12 symbols. The behaviour of the algorithm is visualised in Fig. 7.19.
BDM algorithm is somehow ineffective, because it is matching symbols of prefixes already found. It is possible to improve it and obtain more effective algorithm. The modification consists in the introduction of an additional variable *ncomp* having the value equal to the length of shift and decrementing its value by one after every comparison. If *ncomp* is zero, then the pattern is found.

The configuration of *BDM* algorithm with optimisation is fivetuple (*q,I,J,lprefix,ncomp*).

The initial configuration is $(q_0, 0, m, 0, m)$.

Example 7.8
BDM algorithm with optimisation performs for the pattern and text from Example 7.7 the following sequence of steps:

(0,0,5,0,5) $\overset{a}{\vdash} (235,0,4,1,4)$
$\overset{a}{\vdash} (3,0,3,1,3) shift = m - lprefix = 4$
$\vdash (0,4,5,0,4)$
$\overset{a}{\vdash} (235,4,4,1,3)$
$\overset{b}{\vdash} (4,4,3,1,2)$
$\overset{a}{\vdash} (5,4,2,3,1) shift = m - lprefix = 2$
$\vdash (0,6,5,0,2)$
$\overset{b}{\vdash} (14,6,4,0,1)$
$\overset{a}{\vdash} (25,6,3,2,0) match, shift = matchshift = 3$
$\vdash (0,9,5,0,3)$
$\overset{b}{\vdash} (14,9,4,0,2)$
$\overset{a}{\vdash} (25,9,3,2,1) shift = m - lprefix = 3$
$\vdash$ stop $\square$

Algorithm *BDM* with optimisation performs 9 comparisons only. Behaviour of *BDM* algorithm with optimisation is visualised in Fig. 7.20.

```
const M = {length of pattern};
      MATCHSHIFT = {length of shift when pattern is found};
var TEXT : array [1..N] of char;
    I,J : integer;
    STATE : TSTATE;
    δ : array[1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F : set of TSTATE;
    LPREFIX: integer;
    SHIFT: integer;
    ...
begin
    LPREFIX := 0;
    STATE := q0;
    I := 0;
    J := M;
    while (I ≤ N-M) do
      begin
        if δ [STATE,TEXT[I+J]] = fail
          then
            begin
              if LPREFIX = M then
                begin
                  output(I + 1);
                  SHIFT := MATCHSHIFT;
                end
              else
                SHIFT := M - LPREFIX;
                LPREFIX := 0;
                STATE := q0;
                I := I + SHIFT;
                J := M;
            end;
          else
            begin
              STATE := δ[STATE,TEXT[I + J]];
              J := J - 1;
              if STATE ∈ F then LPREFIX := M - J;
            end;
      end;
end;
```

Figure 7.17: *BDM* algorithm

```
const M = {length of pattern};
      MATCHSHIFT = {length of shift when pattern is found};
var TEXT : array [1..N] of char;
    I,J,LPREFIX,SHIFT,NCOMP : integer;
    STATE : TSTATE;
    δ : array[1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F : set of TSTATE;
    ...
begin
    LPREFIX := 0;
    STATE := q0;
    NCOMP := M;
    I := 0;
    J := M;
    while (I≤N-M) do
       begin
         if δ [STATE,TEXT[I+J]]= fail or NCOMP = 0
           then
             begin
               if LPREFIX = M or NCOMP = 0 then
                 begin
                   output(I + 1);
                   SHIFT := MATCHSHIFT;
                 end
               else
                 SHIFT := M - LPREFIX;
                 NCOMP := SHIFT;
                 LPREFIX := 0;
                 STATE := q0;
                 I := I + SHIFT;
                 J := M;
             end;
           else
             begin
               STATE := δ[STATE,TEXT[I + J]];
               J := J - 1;
               if STATE ∈ F then LPREFIX := M - J;
               NCOMP := NCOMP - 1;
             end;
       end;
end;
```

Figure 7.18: *BDM* algorithm with optimisation

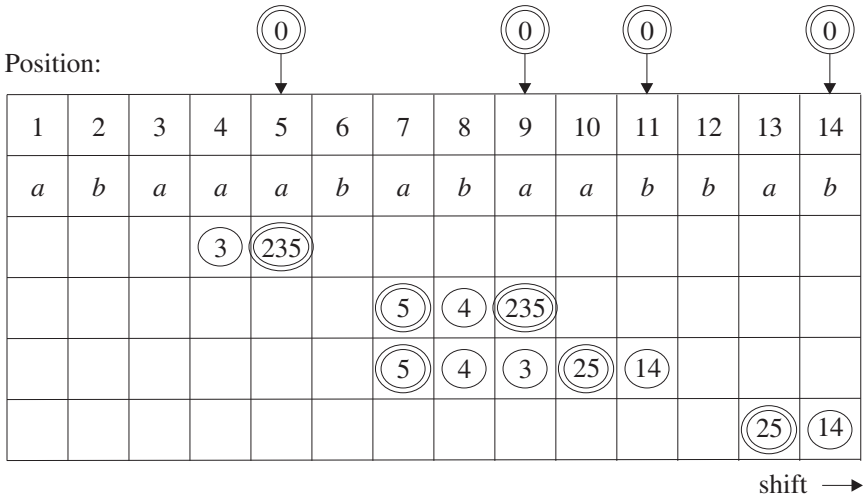

Figure 7.19: Behaviour of the suffix automaton for reversed pattern $P^R = baaba$ from Example 7.7

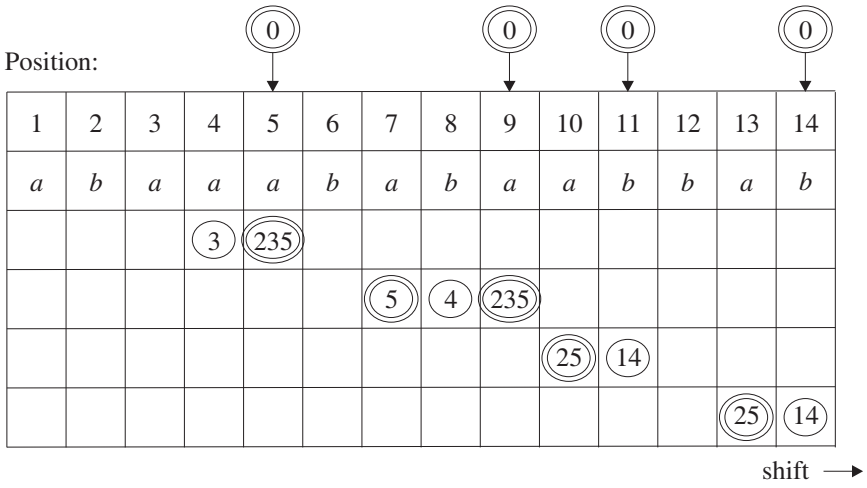

Figure 7.20: Behaviour of the suffix automaton for reversed pattern $P^R = baaba$ from Example 7.8 using *BDM* algorithm with the optimisation

7.2.3.2 Reduced backward *DAWG* matching (*RGPS*) The second method, looking for prefixes which are suffixes of the pattern, is also based on the use of a suffix automaton for the reversed pattern. Finding prefixes of the pattern which are also suffixes of the pattern is done by a reduced suffix automaton containing only its backbone. It leads to the heuristics called "reduced good prefix shift" (*RGPS*). But the *RGPS* shift is not "safe". It means that some occurrences of the pattern can be missed. This is why this approach cannot be used alone but only in a combination with other approaches. An example of such combination is the Boyer–Moore algorithm presented in the next section. Let us show this principle using an example.

Example 7.9
Let pattern P be $P = babab$ over alphabet $A = \{a, b, c\}$. The reversed pattern is $P^R = babab = P$. We construct a suffix automaton for the reversed pattern. Transition diagram of the nondeterministic suffix automaton for reversed pattern $P^R = babab$ is depicted in Fig. 7.21. Transition tables of both

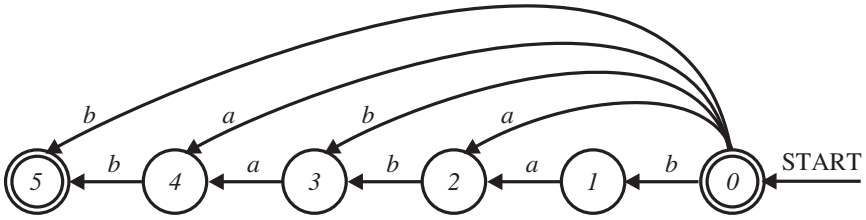

Figure 7.21: Transition diagram of the nondeterministic suffix automaton for reversed pattern $P^R = babab$ from Example 7.9

nondeterministic and deterministic suffix automata are shown Table 7.4.

	a	b	c
0	$2, 4$	$1, 3, 5$	
1	2		
2		3	
3	4		
4		5	
5			

	a	b	c
0	24	135	
135	24		
24		35	
35	4		
4		5	
5			

Table 7.4: Transition tables of the nondeterministic and deterministic suffix automata for reversed pattern $P^R = babab$ from Example 7.9

Transition diagram of the deterministic suffix automaton is depicted in Fig. 7.22. The part of this automaton which is not the backbone is drawn by dashed line. A table containing information on the repeated prefix for each suffix is shown in Table 7.5. □

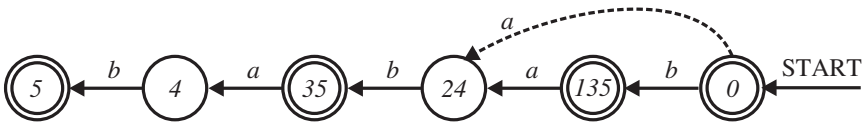

Figure 7.22: Transition diagram of the deterministic suffix automaton for reversed pattern $P^R = babab$ from Example 7.9

$d-$subset	Suffix	Repeated prefix
0	ε	ε
135	b	b
24	ab	b
35	bab	bab
4	$abab$	bab
5	$babab$	bab

Table 7.5: Correspondence between suffixes and repeated prefixes of reversed pattern $P^R = babab$ from Example 7.9

Algorithm 7.10
Computation of reduced good prefix shift.
Input: Backbone $B = (Q, A, \delta, q_0, F)$ of the suffix automaton M for pattern P having length m.
Output: Reduced good prefix shift table for pattern P.
Method: Inspect backbone B of the suffix automaton starting in state q_0 and continuing to next state up to the terminal state. Two situations occur:

1. The inspected state q is a final state. The value of $RGPS(q) = m - depth(q)$.
2. The inspected state q is not a final state. The value of $RGPS = m - depth(p)$, where p is the state having the closest depth less than $depth(q)$. □

Now we can construct a table of shifts (restricted good prefix shifts, $RGPS$). The $RGPS$ table is shown in Table 7.6. The elementary algorithm can be used for this type of the backward pattern matching (see Fig. 7.1). The only required change is replacement of statement

(*) $I := I + 1;$
by statement
$I := I + RGPS[J];$

where $RGPS$ is the "reduced good prefix shift" table. In Table 7.6 the possible values of J are included.

State	Suffix	$RGPS$	J
0	ε	5	5
135	b	4	4
24	ab	4	3
35	bab	2	2
4	$abab$	2	1
5	$babab$	2	0

Table 7.6: Table of shifts (Reduced Good Prefix Shifts, $RGPS$) for reversed pattern $P^R = babab$ from Example 7.9 □

Example 7.11
Let T text be $T = bbbababab$ and pattern $P = babab$. The reduced suffix automaton from Example 7.9 performs the following moves:

Text: $bbaababab$	State	Action	J	
$babab$	0	$match$	5	
	135	$match$	4	
	24	$mismatch, shift = 4$	3	$RGPS(3) = 4$
$babab$	0	$match$	5	
	135	$match$	4	
	24	$match$	3	
	35	$match$	2	
	4	$match$	1	
	5	$patternfound, shift = 2$	0	$RGPS(0) = 2$

In this situation the pattern matching ends, because the shift is behind the end of the text. The behaviour of the suffix automaton using $RGPS$ shifts is visualised in Fig. 7.23 □

Example 7.12
This example demonstrates, how one occurrence of the pattern is missed. Let text T be $T = bbbbabababab$ and pattern $P = babab$. The reduced suffix automaton from Example 7.9 performs the following moves:

Text: $bbbbabababab$	State	Action	J	
$babab$	0	$mismatch, shift = 5$	5	$RGPS(5) = 5$
$babab$	0	$match$	5	
	135	$match$	4	
	24	$match$	3	
	35	$match$	2	
	4	$match$	1	
	5	$patternfound, shift = 2$	0	$RGPS(0) = 2$

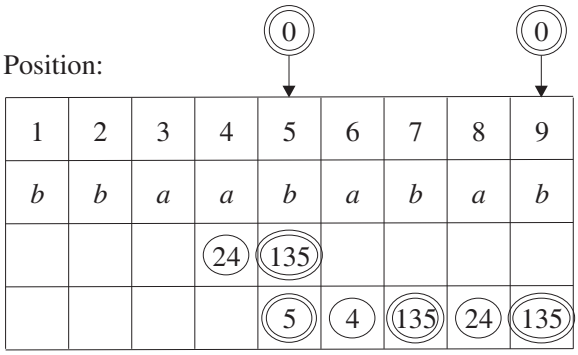

Figure 7.23: Behaviour of the suffix automaton for reversed pattern $P^R = babab$ using $RGPS$ table from Example 7.11

We can see that the first occurrence of the pattern at position 4 is not found. □

7.2.4 Boyer–Moore algorithm

The classical Boyer–Moore (BM) algorithm [BM77] uses:
bad character shift (BCS, see Section 7.2.1),
good suffix shift (GSS, see Section 7.2.2), and
reduced good prefix shift ($RGPS$, see Section 7.2.3.2) heuristics.

The basic principle of BM algorithm is based on the selection of the longer shift from BCS and minimum of GSS and $RGPS$ shifts. Let us show the principle of the BM algorithm using an example.

Example 7.13
Let pattern P be $P = ababba$ over alphabet $A = \{a, b, c\}$. We compute BCS, GSS, and $RGPS$ for pattern P. The nondeterministic factor automaton for reversed pattern $P^R = abbaba$ has transition diagram depicted in Fig. 7.24. Transition tables of both nondeterministic and deterministic fac-

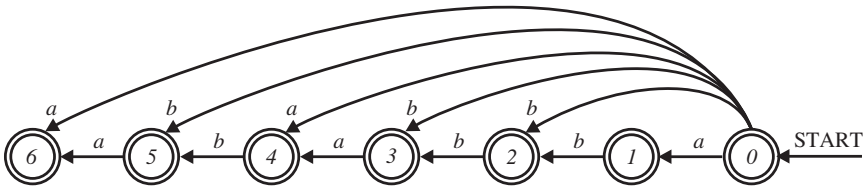

Figure 7.24: Transition diagram of the nondeterministic factor automaton for the reversed pattern $P^R = abbaba$ from Example 7.13

tor automata are shown in Table 7.7. The transition diagram of the deterministic factor automaton is depicted in Fig. 7.25. The tables of BCS, GSS, and $RGPS$ are shown in Table 7.8. □

	a	b	c
0	$1, 4, 6$	$2, 3, 5$	
1		2	
2		3	
3	4		
4		5	
5	6		
6			

	a	b	c
0	146	235	
146		25	
25	6	3	
235	46	3	
3	4		
4		5	
5	6		
6			
46		5	

Table 7.7: Transition tables of the nondeterministic and deterministic factor automata for reversed pattern $P^R = abbaba$ from Example 7.13

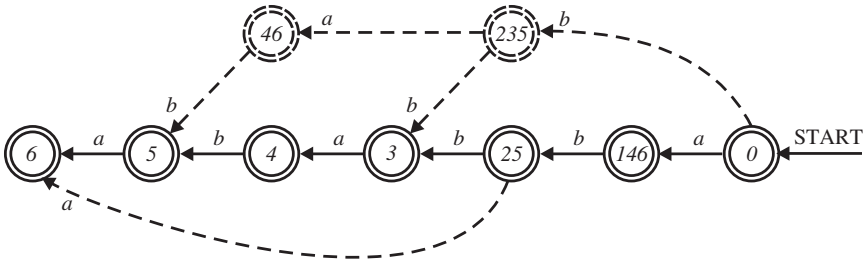

Figure 7.25: Transition diagram of the deterministic factor automaton for the reversed pattern $P^R = abbaba$ from Example 7.13

The elementary algorithm (see Fig. 7.1) can be used for this type of backward pattern matching. The only required change is replacement of statement

$(*)$ $I := I + 1;$
by statement
$I := I + min(RGPS[J], max(BCS[TEXT[I + J]], GSS[J]));$

where $RGPS$ is the "reduced good prefix shift", BCS is the "bad character shift", and GSS is the "good suffix shift".

Example 7.14
Let text T be $T = aaabbababba$ and pattern $P = ababba$ (from Example 7.13). The pattern matching algorithm performs the following sequence of moves:

Symbol	*a*	*b*	*c*
BCS	3	1	6

State	*d-subsets*	*Suffix*	*GSS*	*RGPS*	*J*
0	0	ε	1	6	6
1	146	*a*	3	5	5
2	25	*ba*	3	5	4
3	3	*bba*	5	5	3
4	4	*abba*	5	5	2
5	5	*babba*	5	5	1
6	6	*ababba*	5	5	0

Table 7.8: *BCS*, *GSS*, and *RGPS* tables for reversed pattern $P^R = abbaba$ from Example 7.13

Text: *aacbbababba*	State	Action	*J*	
ababba	0	*match*	6	
	146	*match*	5	
	25	*match*	4	
	3	$mismatch, shift = 5$	3	$(*)$
ababba	0	*match*	6	
	146	*match*	5	
	25	*match*	4	
	3	*match*	3	
	4	*match*	2	
	5	*match*	1	
	6	$patternfound, shift = 5$	0	$(**)$

Shifts were computed as follows:

(*) $J = 3$, $TEXT[I+M] = a$:
$min(RGPS[J], max(BCS[TEXT[I+M]], GSS[J]))$
$= min(5, max(3, 5)) = 5$

(**) $J = 0$, $TEXT[I+M] = a$:
$min(RGPS[J], max(BCS[TEXT[I+m]], GSS[J]))$
$= min(5, max(3, 5)) = 5$

7.2.5 Looking for antifactors of the pattern

The method described in this section is based on the use of factor automaton or factor oracle automaton for the recognition of antifactors for the reversed pattern. An antifactor of the string w is any string which is not a factor of w. The principle of this approach is shown in Fig. 7.7. Let us suppose, that x is a factor of the pattern. If during the reading of symbol z we find out, that zx is an antifactor, then we can safely shift the pattern behind symbol z. We will call this shift "antifactor shift" (AFS). This method is namely effective if a factor oracle automaton is used. We start with the presentation of this

principle using a factor automaton which is able to recognise exactly factors and antifactors as well. A factor oracle automaton recognises antifactors reliable, but factors tentatively, only.

7.2.5.1 Backward factor matching (*BFM*) We present backward factor matching (*BFM*). Let us use of a factor automaton for looking for antifactors in the next Example.

Example 7.15
Let pattern P be $P = abaab$ over alphabet $A = \{a, b, c\}$. The reversed pattern is $P^R = baaba$. We construct the factor automaton for this reversed pattern. Transition diagram of nondeterministic factor automaton M_1 for reversed pattern P^R is depicted in Fig. 7.26. The next step is the con-

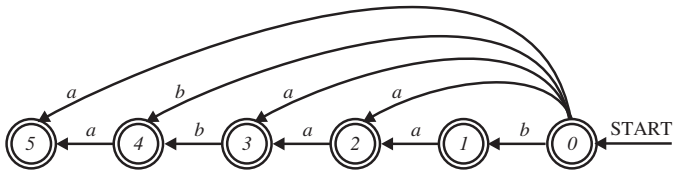

Figure 7.26: Transition diagram of nondeterministic factor automaton M_1 for $P^R = baaba$ from Example 7.15

struction of deterministic factor automaton M_2. Transition table and the transition diagram of it is shown in Fig. 7.27. The factor automaton is used

	a	b	c
0	235	14	
14	25		
25	3		
235	3	4	
3		4	
4	5		
5			

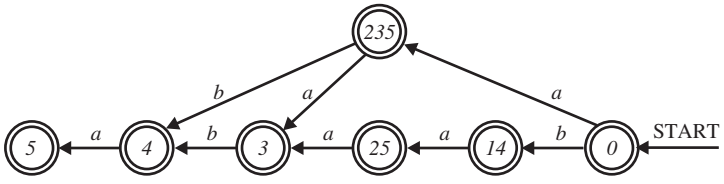

Figure 7.27: Transition table and transition diagram of deterministic factor automaton M_2 for reversed pattern $P^R = baaba$ from the Example 7.15

for the identification of the longest factor of the pattern while reading text

backwards. As soon as a mismatch occurs reading symbol z, an antifactor of the pattern is recognized. The pattern is then shifted behind symbol z which does not belong to the factor of the pattern. If means that the antifactor shift is:

$$AFS = m - lfactor,$$

where m is length of pattern,
lfactor is length of recognised factor.

If the pattern is found, then the shift is given by the difference between the length of the pattern and the length of the Border (the longest border) of the pattern (*matchshift*). Fig. 7.28 shows the visualisation of the behaviour of the factor automaton for reversed pattern $P^R = baaba$. The last column contains the length of antifactor shift (AFS). The pattern matching algorithm (*BFM*, see Fig. 7.29) uses the factor automaton for reversed pattern, the length m of the pattern and *matchshift* as parameters. The length of $Border(abaab) = ab$ is equal to 2. Therefore $matchshift = m - 2 = 5 - 2 = 3$. □

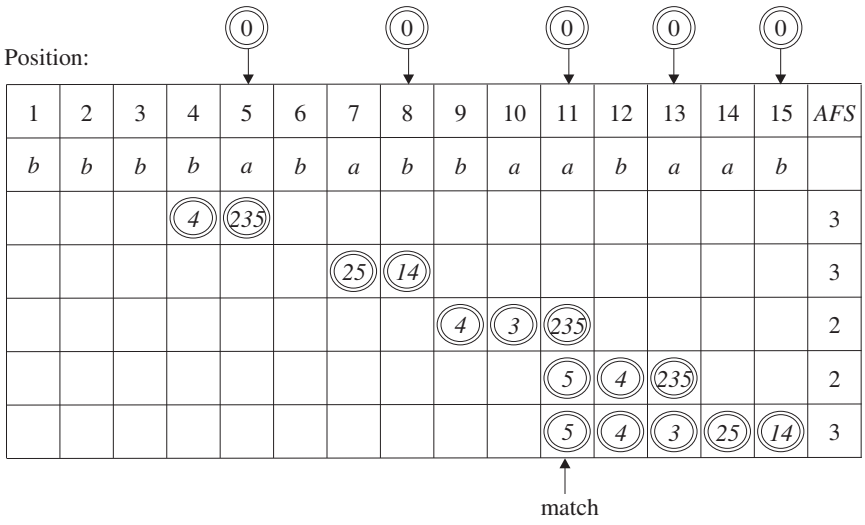

Figure 7.28: Behaviour of the factor automaton for reversed pattern $P^R = baaba$ from Example 7.15

Configuration of the *BFM* algorithm is a quadruple
(*state,I,J,lfactor*).
The initial configuration is $(q_0, 0, m, 0)$.

```
const M = {length of pattern};
      MATCHSHIFT = {length of shift when pattern is found};
var TEXT : array [1..N] of char;
    I,J : integer;
    STATE : TSTATE;
    δ : array[1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F : set of TSTATE;
    LFACTOR: integer;
    SHIFT: integer;
    ...
begin
    LFACTOR := 0;
    STATE := q0;
    I := 0;
    J := M;
    while (I ≤ N-M) do
      begin
        if δ [STATE,TEXT[I+J]] = fail
          then
            begin
              if LFACTOR = M then
                begin
                  output(I + 1);
                  SHIFT := MATCHSHIFT;
                end
              else
                SHIFT := M - LFACTOR;
                LFACTOR := 0;
                STATE := q0;
                I := I + SHIFT;
                J := M;
            end;
          else
            begin
              STATE := δ[STATE,TEXT[I + J]];
              J := J - 1;
              LFACTOR := LFACTOR + 1;
            end;
      end;
end;
```

Figure 7.29: *BFM* and *BOM* algorithms

Example 7.16
Let us use the factor automaton for pattern $P^R = baaba$ from Example 7.15 and show the pattern matching in text:

$T =$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
b	b	b	b	a	b	a	b	b	a	a	b	a	a	b

BFM algorithm performs the following sequence of steps:

$$
\begin{aligned}
(0,0,5,0) &\overset{a}{\vdash} (235,0,4,1)\\
&\overset{b}{\vdash} (4,0,3,2)shift\ 3\\
&\vdash (0,3,5,0)\\
&\overset{b}{\vdash} (14,3,4,1)\\
&\overset{a}{\vdash} (25,3,3,2)shift\ 3\\
&\vdash (0,6,5,0)\\
&\overset{a}{\vdash} (235,6,4,1)\\
&\overset{a}{\vdash} (3,6,3,2)\\
&\overset{a}{\vdash} (4,6,2,3)shift\ 2\\
&\vdash (0,8,5,0)\\
&\overset{a}{\vdash} (235,8,4,1)\\
&\overset{b}{\vdash} (4,8,3,2)\\
&\overset{a}{\vdash} (5,8,2,3)shift\ 2\\
&\vdash (0,10,5,0)\\
&\overset{b}{\vdash} (14,10,4,1)\\
&\overset{a}{\vdash} (25,10,3,2)\\
&\overset{a}{\vdash} (3,10,2,3)\\
&\overset{b}{\vdash} (4,10,1,4)\\
&\overset{a}{\vdash} (5,10,0,5)match\ 11, shift\ 3\\
&\vdash stop
\end{aligned}
$$

□

7.2.5.2 Backward oracle matching (*BOM*) For the looking for antifactors the factor oracle automaton can be used as well. The method using this approach is called also Backward Oracle Matching (*BOM*). The basic tool for the looking for antifactors of the pattern is a factor oracle automaton for the reversed pattern. This automaton is accepting the set of all factors of the pattern and moreover some of its subsequences. Let us show the use of a factor oracle automaton for looking for antifactors in the next example.

Example 7.17
Let pattern be $P = abaab$ over alphabet $A = \{a, b, c\}$. The reversed pattern is $P^R = baaba$. We construct the factor oracle automaton for this reversed

pattern. Transition diagram of nondeterministic factor automaton $M_1(P^R)$ is depicted in Fig. 7.26. The next step is construction of deterministic factor oracle automaton $M_2(P^R)$. Transition table and transition diagram of it is shown in Fig. 7.30. Factor oracle automaton $M_2(P^R)$ accepts the set

	a	b	c
0	235	14	
14	25		
235	3	4	
3		4	
4	5		
5			

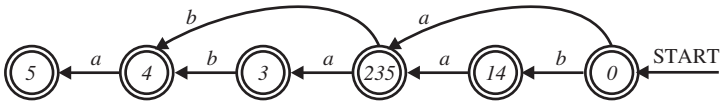

Figure 7.30: Transition table and the transition diagram of factor oracle automaton $M_2(P^R)$ for reversed pattern $P^R = baaba$ from the Example 7.17

$L(Oracle(baaba)) =$
$= \{\varepsilon, a, b, aa, ab, ba, aab, aba, baa, aaba, baab, baaba, bab, baba\}$
$= Fact(baaba) \cup \{bab, baba\},$
where bab and $baba$ are subsequences of string $baaba$.
If a factor oracle automaton for a text T accepts string $x,\ x \in A^*$, and does not accepts string $xz, z \in A$, then the string xz is not a factor of text T and therefore xz is an antifactor. Visualisation of the behaviour of factor oracle automaton $M(P^R)$ is shown in Fig. 7.31. The pattern matching algorithm BFM looking for antifactors can be used in connection with factor oracle automaton as well. If we take factor oracle automaton $M(P^R)$ as an input of algorithm BFM we obtain algorithm called BOM and the algorithm performs for reversed pattern $P^R = baaba$ and for the text from Example 7.16 the following sequence of steps:

$$
\begin{aligned}
(0,0,5,0) &\overset{a}{\vdash} (235, 0, 4, 1) \\
&\overset{b}{\vdash} (4, 0, 3, 2) shift\ 3 \\
&\vdash (0, 3, 5, 0) \\
&\overset{b}{\vdash} (14, 3, 4, 1) \\
&\overset{a}{\vdash} (235, 3, 3, 2) \\
&\overset{b}{\vdash} (4, 3, 2, 3)
\end{aligned}
$$

$\overset{a}{\vdash} (5, 3, 1, 4) shift\ 1$
$\vdash (0, 4, 5, 0)$
$\overset{b}{\vdash} (14, 4, 4, 1) shift\ 4$
$\vdash (0, 8, 5, 0)$
$\overset{a}{\vdash} (235, 8, 4, 1)$
$\overset{b}{\vdash} (4, 8, 3, 2)$
$\overset{a}{\vdash} (5, 8, 2, 3) shift\ 2$
$\vdash (0, 10, 5, 0)$
$\overset{b}{\vdash} (14, 10, 4, 1)$
$\overset{a}{\vdash} (235, 10, 3, 2)$
$\overset{a}{\vdash} (3, 10, 2, 3)$
$\overset{b}{\vdash} (4, 10, 1, 4)$
$\overset{a}{\vdash} (5, 10, 0, 5) match\ 11, shift\ 3$
$\vdash stop$

□

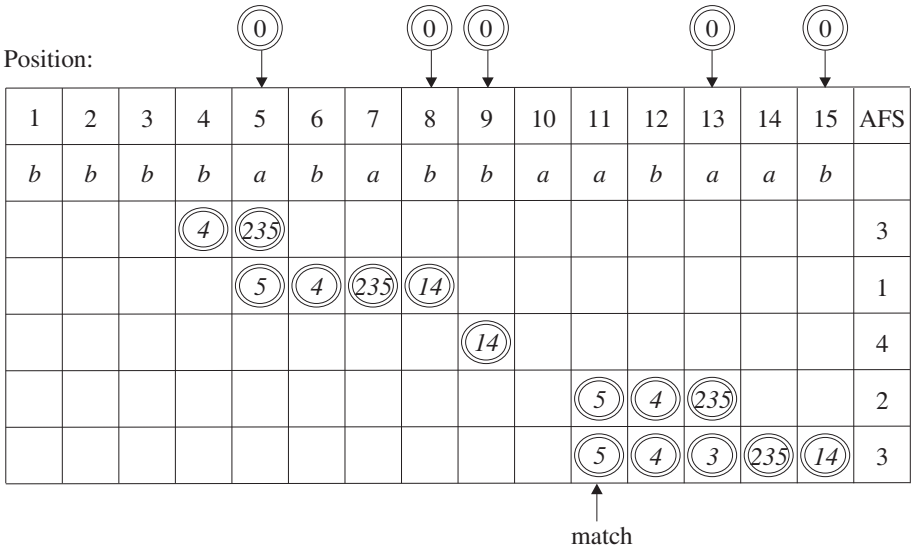

Figure 7.31: Behaviour of factor oracle automaton $M_2(P^R)$ for reversed pattern $P^R = baaba$ from Example 7.17

8 Exact backward matching of a finite set of patterns

The idea of the use finite automata as a base for the backward pattern matching of one pattern can be used for the backward pattern matching of a finite set of patterns. First we show the basic model of the multi–backward pattern matching. Further we show how to use finite automata approach for Multi*BMH*, Multi*BDM*, looking for repeated suffixes, Commentz–Walter, and looking for antifactors of a finite set of patterns.

8.1 Model of the multibackward pattern matching

The base for the model of the multibackward pattern matching is a deterministic finite automaton accepting set of reversed patterns. This automaton is extended in order to identify situations when the next transition is not possible for some state and some input symbol. Let us call this automaton multibackward pattern matching automaton (*MBPM* automaton).

Algorithm 8.1
Construction of a finite automaton for multibackward pattern matching.
Input: Finite set of patterns $S = \{p_1, p_2, \ldots, p_{|S|}\}, p_1, p_2, \ldots, p_{|S|} \in A^+$.
Output: Model of multibackward pattern matching – deterministic finite automaton $M = (Q, A, \delta, q_0, F)$.
Method:

1. Create nondeterministic finite automaton $M_1 = (Q_1, A, \delta_1, q_{01}, F_1)$ accepting set of reversed patterns $S^R = \{p_1^R, p_2^R, \ldots, p_{|S|}^R\}$.
2. Create deterministic finite automaton $M_2 = (Q_2, A, \delta_2, q_{02}, F_2)$ equivalent to automaton M_1.
3. Create finite automaton $M = (Q, A, \delta, q_0, F)$ using automaton M_2 where
   $Q = Q_2$,
   $q_0 = q_{02}$,
   $F = F_2$,
   $\delta(q, a) = \delta_2(q, a)$ if $\delta_2(q, a)$ is defined,
   $\delta(q, a) = fail$ if $\delta_2(q, a)$ is undefined. □

Example 8.2
Let S be set of patterns $S = \{cba, aba, cb\}$. Set of reversed patterns is $S^R = \{abc, aba, bc\}$. We construct finite automaton M accepting set S^R with transition function δ having some values equal to *fail*.

1. $M_1 = (\{q_0, q_{11}, q_{12}, q_{13}, q_{21}, q_{22}, q_{23}, q_{31}, q_{32}\}, \{a, b, c\}, \delta_1, q_0, \{q_{13}, q_{23}, q_{32}\})$. The transition table for δ_1 is shown in Table 8.1. Transition diagram of the automaton M_1 is depicted in Fig. 8.1.

δ_1	a	b	c
q_0	q_{11}, q_{21}	q_{31}	
q_{11}		q_{12}	
q_{12}			q_{13}
q_{13}			
q_{21}		q_{22}	
q_{22}	q_{23}		
q_{23}			
q_{31}			q_{32}
q_{32}			

δ_2	a	b	c
q_0	$q_{11}q_{21}$	q_{31}	
$q_{11}q_{21}$		$q_{12}q_{22}$	
$q_{12}q_{22}$	q_{23}		q_{13}
q_{13}			
q_{23}			
q_{31}			q_{32}
q_{32}			

Table 8.1: Transition tables of automata M_1 and M_2 from Example 8.2

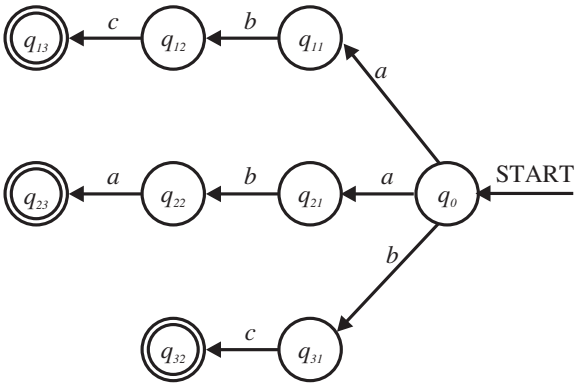

Figure 8.1: Transition diagram of the nondeterministic finite automaton M_1 from Example 8.2

2. $M_2 = (\{q_0, q_{11}q_{21}, q_{12}q_{22}, q_{13}, q_{23}, q_{31}, q_{32}\}, \{a, b, c\}, \delta_2, \{q_{13}, q_{23}, q_{32}\})$. The transition table for δ_2 is shown in Table 8.1. Transition diagram of automaton M_2 is depicted in Fig. 8.2.
3. $M = (\{q_0, q_{11}q_{21}, q_{12}q_{22}, q_{13}, q_{23}, q_{31}, q_{32}\}, \{a, b, c\}, \delta, \{q_{13}, q_{23}, q_{32}\})$. Transition function δ is shown in Table 8.2. □

Algorithm for exact backward matching of the set of patterns *MBPM* is shown in Fig. 8.3. The *MBPM* searching algorithm starts reading the text at position *lmin* which is length of the shortest pattern. It is matching the text from right to left. As soon as the value of $\delta(q, a)$ is *fail* then the automaton is shifted one position to the right and reads the text again starting from the initial state. We can see, that the shift has always the length equal to one like in the elementary algorithm for backward matching of one pattern. However the shift can be greater taking into account repetitions inside set of patterns.

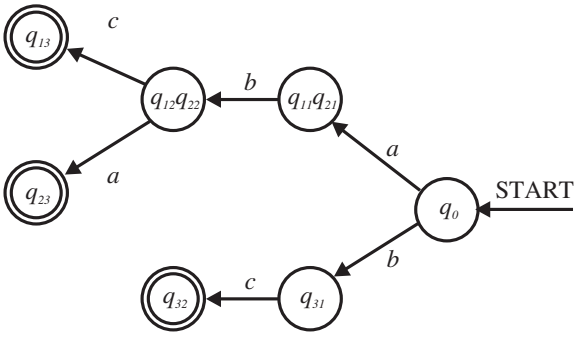

Figure 8.2: Transition diagram of the deterministic finite automaton M_2 from Example 8.2

δ	a	b	c
q_0	$q_{11}q_{21}$	q_{31}	$fail$
$q_{11}q_{21}$	$fail$	$q_{12}q_{22}$	$fail$
$q_{12}q_{22}$	q_{23}	$fail$	q_{13}
q_{13}	$fail$	$fail$	$fail$
q_{23}	$fail$	$fail$	$fail$
q_{31}	$fail$	$fail$	q_{32}
q_{32}	$fail$	$fail$	$fail$

Table 8.2: Transition table of the resulting finite automaton M from Example 8.2

8.2 Backward matching automata for a finite set of patterns

8.2.1 Multi*BMH* algorithm

We can extend the principle of *BMH* algorithm (see Section 7.2.1) to a finite set of patterns. The length of shift in the Multi*BMH* algorithm is given for each symbol of the alphabet by the distance of the rightmost occurrence of the rightmost inspected symbol in the text in some of respective pattern. This distance must be greater than zero. Moreover, the length of shift is limited by the length of the shortest pattern which is denoted *lmin*. If some symbol does not occur in any element of the set of patterns then the shift is equal to *lmin*. As for *BMH* algorithm, we use for computation of multi bad character shift (*MBCS*) table a factor automaton for a finite set of patterns (see Volume I, Chapter 3).

```
const LMIN = {length of the shortest pattern};
var TEXT : array [1..N] of char;
    I,J : integer;
    STATE : TSTATE;
    δ : array[1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F : set of TSTATE;
...
begin
    STATE := q0;
    I := 0;
    J := LMIN; {starting point of matching}
    while (I ≤ N - LMIN) do
       begin
          if δ[STATE,TEXT[I + J]] = fail
             then
                begin
(*)                I := I + 1; {length of shift = 1}
                   STATE := q0;
                   J := LMIN;
                end
             else
                begin
                   STATE := δ[STATE,TEXT[I + J]];
                   J := J - 1;
                end;
          if STATE in F then output (I + 1)
       end;
end
```

Figure 8.3: Algorithm *MBPM* (multibackward pattern matching) for exact backward pattern matching of a finite set of patterns

Algorithm 8.3
Computation of *MBCS* table.
Input: Finite set of patterns $S = \{p_1, p_2, \ldots, p_{|S|}\}, p_1, p_2, \ldots, p_{|S|} \in A^+$.
Output: *MBCS* table.
Method:

1. Construct nondeterministic factor automaton $M_N = (Q, A, \delta, q_0, F)$ for set of reversed patterns $S^R = \{p_1^R, p_2^R, \ldots, p_{|S|}^R\}$.
2. Construct the first row of transition table of deterministic factor automaton M_D equivalent to M_N.
3. Select for each symbol $a \in A$ the state having the shortest distance $dist(q)$ from the end of some of patterns greater than 0.
   Set $MBCS(a) = min(dist(q) - 1, lmin)$.
4. If some symbol $b \in A$ does not occur in any pattern, set $MBCS(b) = lmin$. □

Example 8.4
Let the set of patterns be $S = \{cbaba, bba, abbb\}$ over alphabet $A = \{a, b, c, d\}$. We will compute the *MBCS* table. Set $lmin = |bba| = 3$.

1. Transition diagram of the nondeterministic factor automaton M_N for set of reversed patterns $S^R = \{ababc, abb, bbba\}$ has the transition diagram depicted in Fig. 8.4.

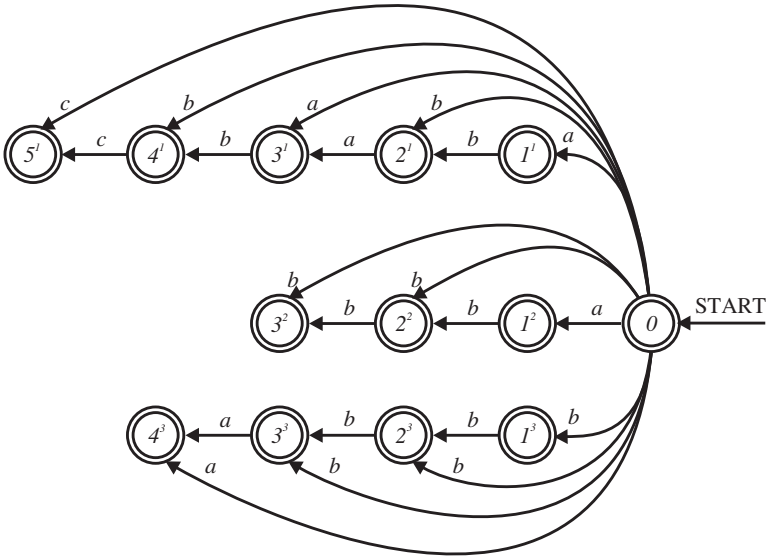

Figure 8.4: Transition diagram of the nondeterministic factor automaton M_N for set of pattern $S^R = \{ababc, abb, bbba\}$ from Example 8.4

2. As we use the distances from the end of pattern as part of the labels of states (i^j, i is the distance from the end of element p_j), then $dist(q^j) = q$. The first row of transition table of the deterministic factor automaton M_D for set of pattern S has the following form:

δ	a	b	c	d
0	$1^1 3^1 1^2 4^3$	$2^1 4^1 2^2 3^2 1^3 2^3 3^3$	5^1	

3. The *MBCS* table has the following form:

	a	b	c	d
MBCS	2	1	3	3

The shifts for symbols c and d are limited to *lmin* (see points 3. and 4. of Algorithm 8.3). □

The algorithm of pattern matching uses the transition table defining mapping δ of the deterministic finite automaton M_D accepting the reversed patterns from the set S^R. We can adapt the multi backward pattern matching algorithm (*MBPM*, see Fig. 8.3) by replacement of statement

(∗) $I := I + 1;$
by statement
$I := I + MBCS[TEXT[I + LMIN]];$

where *MBCS* is multi bad character shift table.

Example 8.5
Let set of patterns be $S = \{cbaba, bba, abbb\}$ (see Example 8.4) and let text $T = cbabbbacbaba$. Transition diagram of the deterministic finite automaton M_D accepting reversed set of patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ is depicted in Fig. 8.5. □

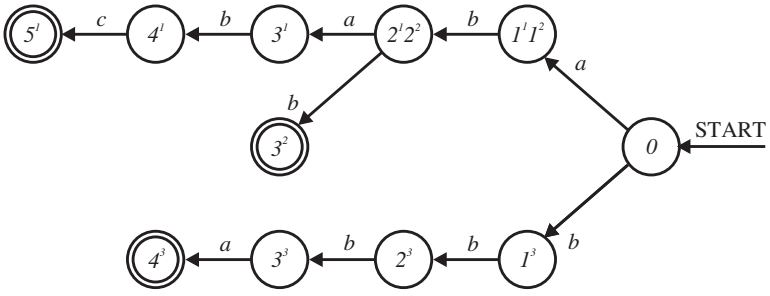

Figure 8.5: Transition diagram of the finite automaton M_D accepting set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ from Example 8.5

Transition table of the automaton M_D is shown in Table 8.3. The empty items have value *fail*. Behaviour of the Multi*BMH* algorithm is visualised in Fig. 8.6.

δ	a	b	c	d
0	$1^1 1^2$	1^3		
$1^1 1^2$		$2^1 2^2$		
1^3		2^3		
$2^1 2^2$	3^1	3^2		
2^3		3^3		
3^1		4^1		
3^2				
3^3	4^3			
4^1			5^1	
4^3				
5^1				

Table 8.3: Transition table of the automaton M_D accepting set of reversed patterns $S^R = \{ababc, abb, bbba\}$ from Example 8.5

8.2.2 Looking for repeated suffixes of a finite set of patterns

The basic tool for this approach, looking for repeated suffixes of a finite set of patterns, is a backbone of the factor automaton for a set of reversed patterns (compare Section 7.2.2). This automaton identifies repeated suffixes of a set of patterns. The heuristics of using such information we call multi good suffix shift (*MGSS*). Let us show this principle using an example.

Example 8.6
Let set of patterns be $S = \{cbaba, bba, abbb\}$ over alphabet $A = \{a, b, c, d\}$. We compute *MGSS* table. Set $lmin = |bba| = 3$. We construct nondeterministic factor automaton M_N for set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$. Its transition diagram is depicted in Fig. 8.7.

The next step is to construct equivalent deterministic factor automaton M_D. We save d-subsets during this construction. Transition tables of both nondeterministic factor automaton M_N and its deterministic equivalent M_D are shown in Table 8.4. Transition diagram of the deterministic factor automaton M_D is depicted in Fig. 8.8.

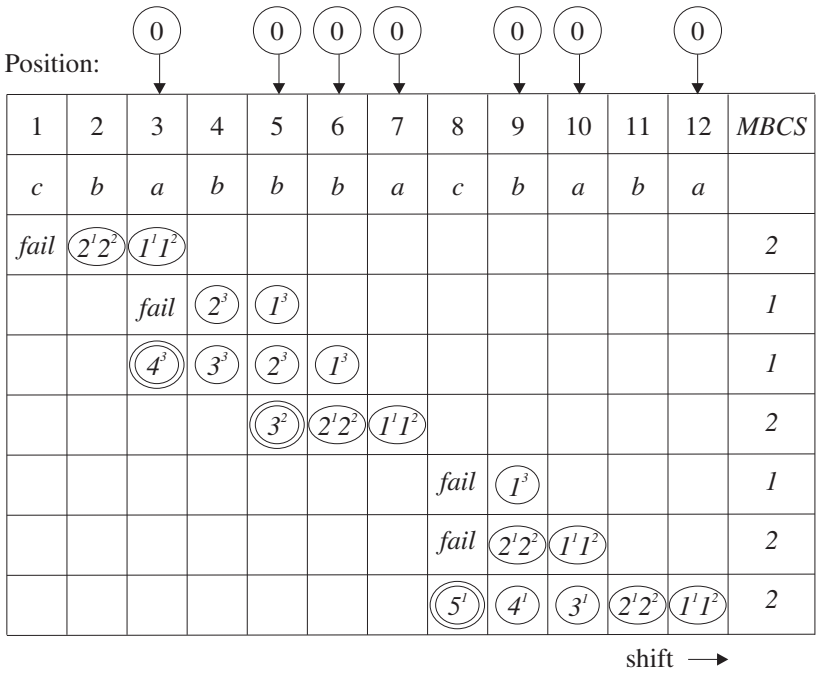

Figure 8.6: Behaviour of the Multi*BMH* algorithm for the set of patterns $S = \{cbaba, bba, abbb\}$ from Example 8.5

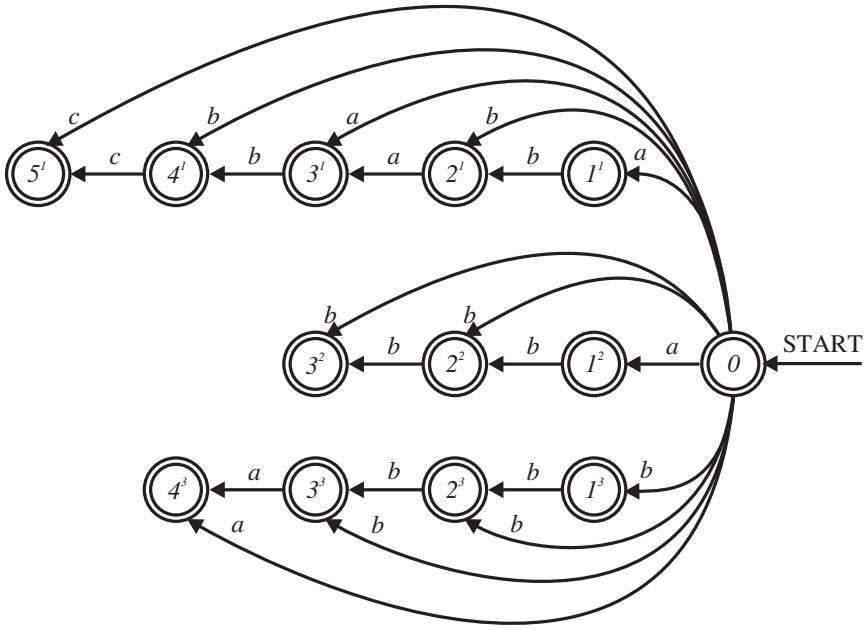

Figure 8.7: Transition diagram of the nondeterministic factor automaton M_N for set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ from Example 8.6

	a	b	c	d
0	$1^1, 3^1, 1^2, 4^3$	$2^1, 4^1, 2^2, 3^2, 1^3, 2^3, 3^3$	5^1	
1^1		2^1		
2^1	3^1			
3^1		4^1		
4^1			5^1	
5^1				
1^2		2^2		
2^2		3^2		
3^2				
1^3		2^3		
2^3		3^3		
3^3	4^3			
4^3				

a) Transition table of M_N

	a	b	c	d
0	$1^1 3^1 1^2 4^3$	$2^1 4^1 2^2 3^2 1^3 2^3 3^3$	5^1	
$1^1 3^1 1^2 4^3$		$2^1 4^1 2^2$		
$2^1 4^1 2^2 3^2 1^3 2^3 3^3$	$3^1 4^3$	$3^2 2^3 3^3$	5^1	
$2^1 4^1 2^2$	3^1	3^2	5^1	
$3^1 4^3$		4^1		
$3^2 2^3 3^3$	4^3	3^3		
3^1		4^1		
3^2				
3^3	4^3			
4^1			5^1	
4^3				
5^1				

b) Transition table of M_D

Table 8.4: Transition tables of factor automata M_N and M_D from Example 8.6

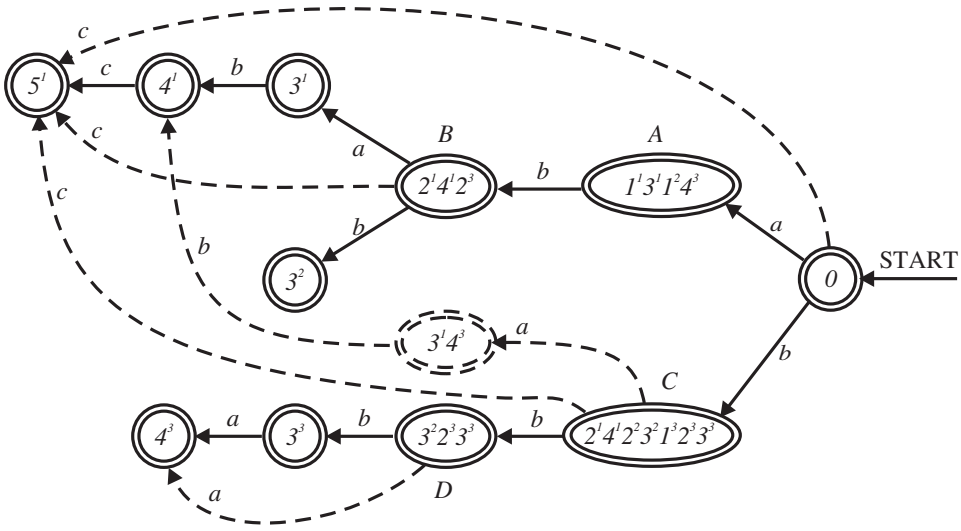

Figure 8.8: Transition diagram of the deterministic factor automaton M_D for set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ from Example 8.6

To obtain the backbone of this factor automaton we remove state $3^1 4^3$ and the transitions drawn by dashed lines. Let us remind that the backbone of a factor automaton can be constructed by intersection of a prefix automaton and a factor automaton (see Volume I, Chapter 3). Now we construct the suffix repetition table. It is shown in Table 8.5.

d–subset	Suffix	Repetitions of suffixes
$1^1 3^1 1^2 4^3$	a	$(1^1, F), (1^2, F), (3^1, G), (4^3, G)$
$2^1 4^1 2^3$	ba	$(2^1, F), (2^3, F), (4^1, S)$
$2^1 4^1 2^2 3^2 1^3 2^3 3^3$	b	$(1^3, F), (2^1, S), (4^1, G), (2^2, S), (3^2, G), (2^3, S), (3^3, G)$
$3^2 2^3 3^3$	bb	$(2^3, F), (3^2, O), (3^3, O)$

Table 8.5: The suffix repetition table from Example 8.6

The factor automaton is able to find the longest suffix matching the respective part of text. As soon as the mismatch occurs, the automaton starts to read the text from the position given by the previous position and the shift. The length of shift is the minimum from the distance between the found suffix and its closest repetition and *lmin*. If the repetition of the suffix does not exist, then the shift is *lmin* – *length*(*mBorder*(S)), where *mBorder* is the longest border of S. In the case of the mismatch the shift in the initial state is equal to 1. Table of shifts for states in which the mismatch occurs is shown in Table 8.6. □

State	Suffix	$MGSS$	
0	ε	1	
A	a	2	
B	ba	2	
3^1	aba	1	
4^1	$baba$	1	
5^1	$cbaba$	1	$match$
3^2	bba	1	$match$
C	b	1	
D	bb	1	
3^3	bbb	1	
4^3	$abbb$	1	$match$

Table 8.6: Table of "multi good suffix shifts" ($MGSS$) from Example 8.6

Algorithm *MBPM* (see Fig. 8.3) can be used for this type of the backward pattern matching. The only required change is replacement of statement

$(*)$ $I := I + 1;$
by statement
$I := I + MGSS(STATE);$

where *MGSS* is the multi good suffix shift table.

Example 8.7
Let set of patterns be $S = \{cbaba, bba, abbb\}$ (see Example 8.6) and let text be $T_1 = cbabbbacbaba$. Transition diagram of deterministic factor automaton M_D for set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ is depicted in Fig. 8.8. Its backbone is drawn by solid lines. This backbone is used by modified algorithm *MBPM*. The modification consists of replacing of simple shifts by *MGSS* shifts. Behaviour of the modified *MBPM* algorithm for text $T_2 = abccbababbba$ is visualised in Fig. 8.9. Behaviour of this algorithm for text T_1 is visualised in Fig. 8.10. □

8.2.3 Looking for prefixes of the finite set of patterns

We present two methods based on the principle of looking for prefixes of the finite set of patterns. The first one is called also Multi Backward *DAWG* Matching (Multi*BDM*, compare Section 7.2.3.1). The second method is looking for the prefixes which are suffixes of some element of the set of patterns. Let us call this method reduced multi good prefix shift method (*RMGPS*). Let us start with the first method.

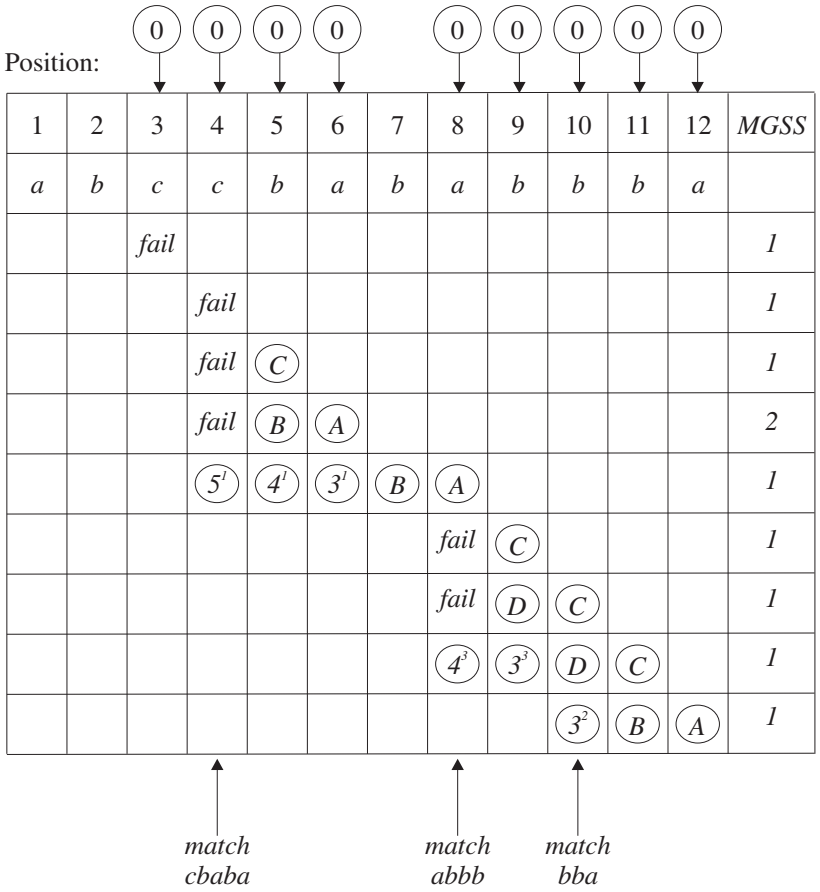

Figure 8.9: Behaviour of modified *MBPM* algorithm for set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ from Example 8.7

8.2.3.1 Multi backward *DAWG* matching The basic tool for the approach, looking for prefixes of the finite set of patterns, is a suffix automaton for the set of reversed patterns (see Volume I, Section 3.7). This automaton identifies all suffixes of the set of reversed patterns, e.g. it is able to identify all prefixes of the set of patterns while reading the text from right to left. As soon as a mismatch occurs, set of patterns is shifted and the automaton is starting to read the text from the position which is given by the previous position and the shift. The heuristics is called multi good prefix shift (*MGPS*). Let us show this principle using an example.

Example 8.8
Let us have set of patterns $S = \{baa, aba, bab\}$ over alphabet $A = \{a, b, c\}$. We construct the suffix automaton for the set $S^R = \{baa^R, aba^R, bab^R\}$ of the reversed patterns. Transition diagram of the nondeterministic suffix automaton with ε–transitions is depicted in Fig. 8.11. The next step is

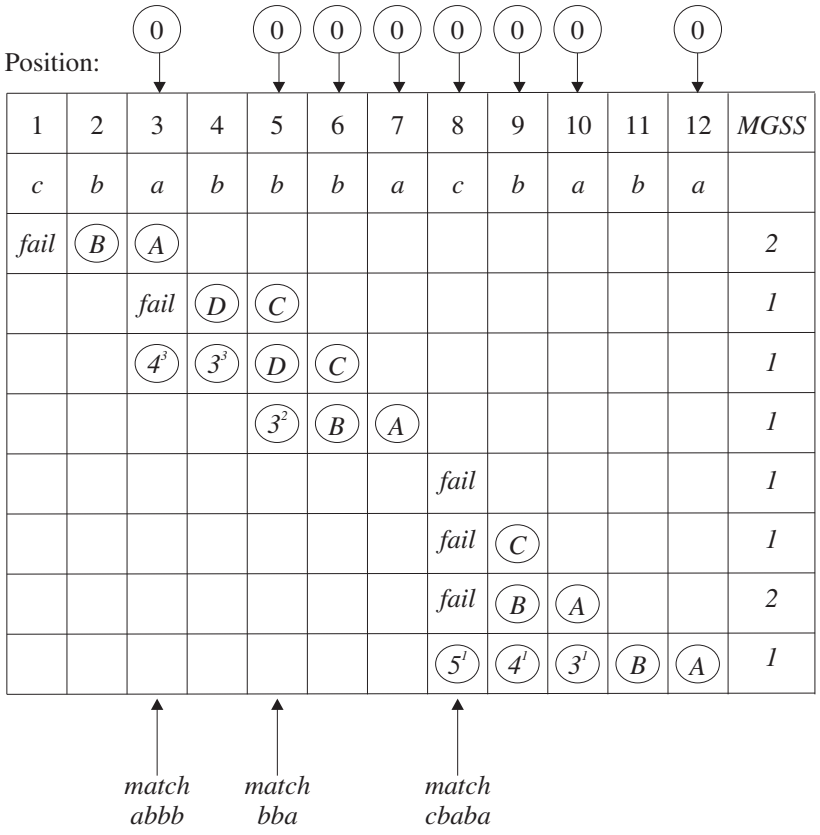

Figure 8.10: Behaviour of modified *MBPM* algorithm for the set patterns $S = \{cbaba, bba, abbb\}$ from Example 8.7

removal of ε–transitions. Transition diagram of this automaton is depicted in Fig. 8.12. The last step is construction of the deterministic suffix automaton. We save the d-subsets during the determinisation. Transition table of this automaton is shown in Table 8.7.

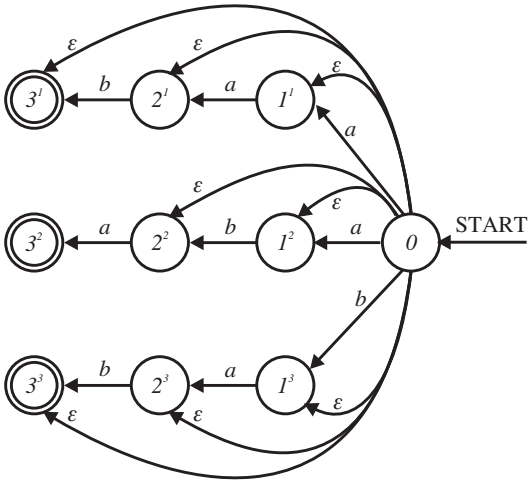

Figure 8.11: Transition diagram of the nondeterministic suffix automaton with ε–transitions for $S^R = \{baa^R, aba^R, bab^R\}$ from Example 8.8

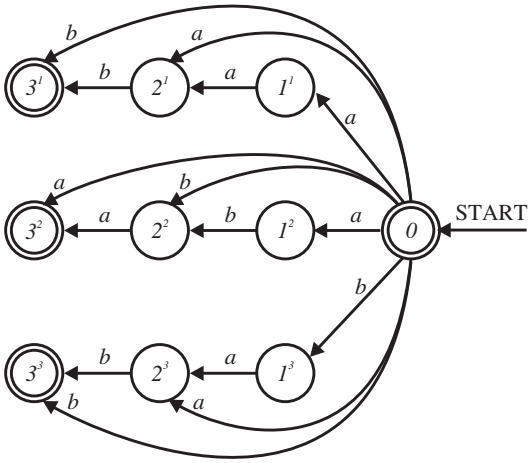

Figure 8.12: Transition diagram of the nondeterministic suffix automaton after removal of ε–transitions from Example 8.8

	a	b	c
0	$1^1, 2^1, 1^2, 3^2, 2^3$	$3^1, 2^2, 1^3, 3^3$	
$1^1, 2^1, 1^2, 3^2, 2^3$	2^1	$3^1, 2^2, 3^3$	
$3^1, 2^2, 1^3, 3^3$	$3^2, 2^3$		
2^1		3^1	
$3^1, 2^2, 3^3$	3^2		
$3^2, 2^3$	3^3		
3^1			
3^2			
3^3			

Table 8.7: Transition table of the deterministic suffix automaton from Example 8.8

Transition diagram of the deterministic suffix automaton is depicted in Fig. 8.13.

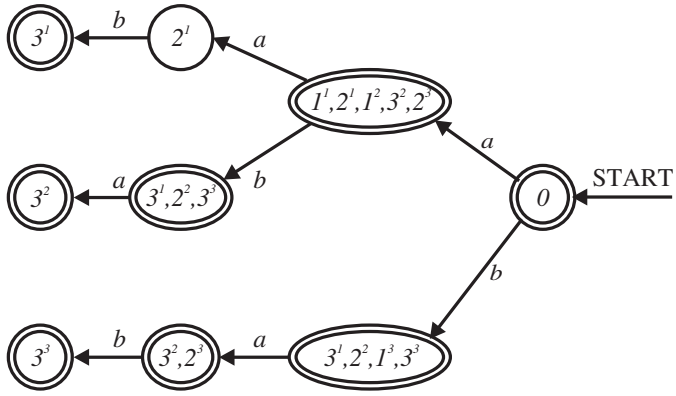

Figure 8.13: Transition diagram of the deterministic suffix automaton from Example 8.8

This automaton is able to identify the following set of prefixes of strings from the set $S = \{baa, aba, bab\}$:

$Pref(S) = \{\varepsilon, a, b, ba, ab, baa, aba, bab\}$. □

```
const M = {length of pattern};
      MATCHSHIFT = {length of shift when pattern is found};
var TEXT : array [1..N] of char;
    I,J : integer;
    STATE : TSTATE;
    δ : array[1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F : set of TSTATE;
    LFACTOR: integer;
    SHIFT: integer;
    ...
begin
    LFACTOR := 0;
    STATE := q0;
    I := 0;
    J := M;
    while (I ≤ N-M) do
      begin
        if δ [STATE,TEXT[I+J]] = fail
          then
            begin
              if LFACTOR = M then
                begin
                  output(I + 1);
                  SHIFT := MATCHSHIFT;
                end
              else
                SHIFT := M - LFACTOR;
                LFACTOR := 0;
                STATE := q0;
                I := I + SHIFT;
                J := M;
            end;
          else
            begin
              STATE := δ[STATE,TEXT[I + J]];
              J := J - 1;
              LFACTOR := LFACTOR + 1;
            end;
      end;
end;
```

Figure 8.14: *MBFM* and *MBOM* algorithms

d–subset	Prefix	Repetitions of prefixes
$1^1 2^1 1^2 3^2 2^3$	a	$(1^1, F), (1^2, F), (2^1, S), (3^2, G), (2^3, S)$
$3^1 2^2 3^3$	ba	$(2^2, F), (3^1, O), (3^3, O)$
$3^1 2^2 1^3 3^3$	b	$(1^3, F), (2^2, S), (3^1, G), (3^3, G)$
$3^2 2^3$	ab	$(2^3, F), (3^2, O)$

Table 8.8: Prefix repetition table for the set of reversed patterns $S^R = \{aab, aba, bab\}$ from Example 8.9

State	Longest prefix	*MGPS*
$1^1 2^1 1^2 3^2 2^3$	a	2
2^1	ba	2
$3^1 2^2 3^3$	ba	1
3^2	ba	1
$3^1 2^2 1^3 3^3$	b	1
$3^2 2^3$	ab	1
3^3	ab	1

Table 8.9: *MGPS* table from Example 8.9

The suffix automaton is therefore used for the identification of the longest prefix of some string from the set of patterns S. As soon as the mismatch occurs, the automaton is starting to read the text from the position which is given by the previous position and the shift. This heuristic is called *multi good prefix shift* (*MGPS*). The length of the shift is given by the distance of found prefix from the initial state of *MBPM* automaton limited by *lmin*. This is the base for computing *MGPS* (multi good prefix shift) table. To construct *MGPS* table, we construct prefix repetition table.

Example 8.9
Let us have set of patterns $S = \{baa, aba, bab\}$ (see Example 8.8). The prefix repetition table is shown in Table 8.8. The *MGPS* table is shown in Table 8.9.

Fig. **??** shows the visualisation of the behaviour of *MBDM* algorithm using the suffix automaton for set of reversed patterns $S^R = \{aab, aba, bab\}$. The arrows (↓) show the position after each shift. In position 5 the pattern *baa* is found, in position 7 the pattern *aba* is found, and in position 8 the pattern *bab* is found. □

8.2.3.2 Reduced multibackward *DAWG* matching The second method, looking for prefixes which are suffixes of some elements of the set

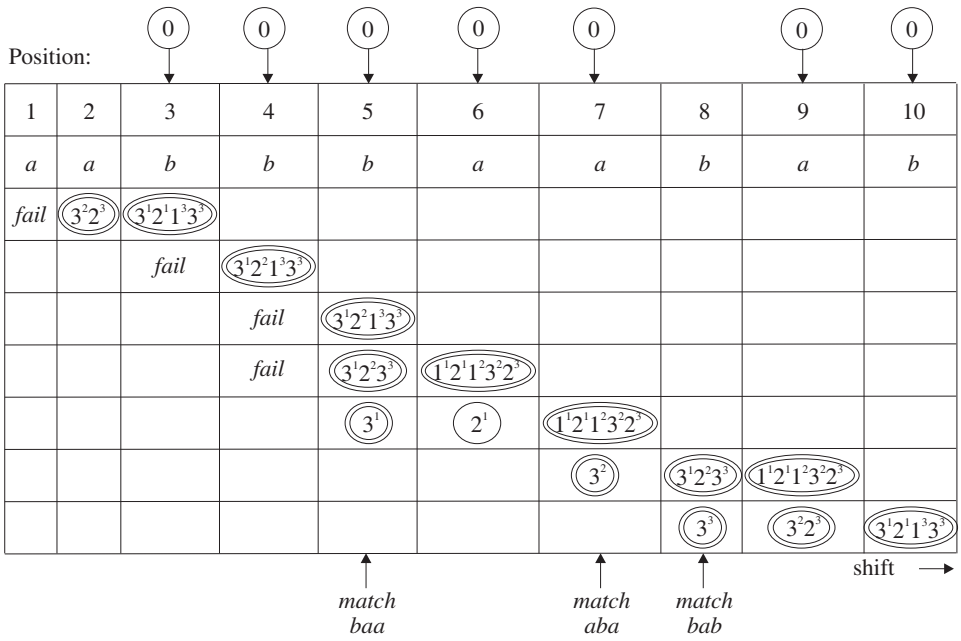

Figure 8.15: Behaviour of the suffix automaton for the reversed set of patterns $S^R = \{baa^R, aba^R, bab^R\}$ from Example 8.9

of pattern, is based also on the use of a suffix automaton for the set of reversed patterns. Finding prefixes which are also suffixes of some patterns is done by the backbone of the suffix automaton. It leads to the heuristic called "reduced multi good prefix shift" (*RMGPS*). The problem of *RMGPS* shift is that it is not "safe" similarly to *RGPS* shift (see Section 7.2.3.2). It means that some occurrences of the pattern can be missed. This is why this approach cannot be used alone but only in a combination with other approaches. An example of such combination is the Commentz–Walter algorithm presented in the next Section. Let us show this principle using an example.

Example 8.10
Let set of patterns be $S = \{babb, caba\}$ over alphabet $A = \{a, b, c\}$. The set of reversed patterns is $S^R = \{bbab, abac\}$. Transition diagram of the nondeterministic suffix automaton for the set of reversed pattern S^R is depicted in Fig. 8.16. Its transition table is shown in Table 8.10. Transition table of the equivalent deterministic suffix automaton is shown in Table 8.11. Transition diagram of this deterministic suffix automaton is shown in Fig. 8.17. State $3^1 3^2$ and transitions which are out of the backbone are drawn by dashed lines.

	a	b	c
0	$1^1, 3^1, 3^2$	$2^1, 1^2, 2^2, 4^2$	4^1
1^1		2^1	
2^1	3^1		
3^1			4^1
4^1			
1^2		2^2	
2^2	3^2		
3^2		4^2	
4^2			

Table 8.10: Transition table of the nondeterministic suffix automaton from Example 8.10

	a	b	c
0	$1^1 3^1 3^2$	$2^1 1^2 2^2 4^2$	4^1
$1^1 3^1 3^2$		$2^1 4^2$	4^1
$2^1 1^2 2^2 4^2$	$3^1 3^2$	2^2	
$2^1 4^2$	3^1		
2^2	3^2		
3^1			4^1
$3^1 3^2$		4^2	4^1
3^2		4^2	
4^1			
4^2			

Table 8.11: Transition table of the deterministic suffix automaton from Example 8.10

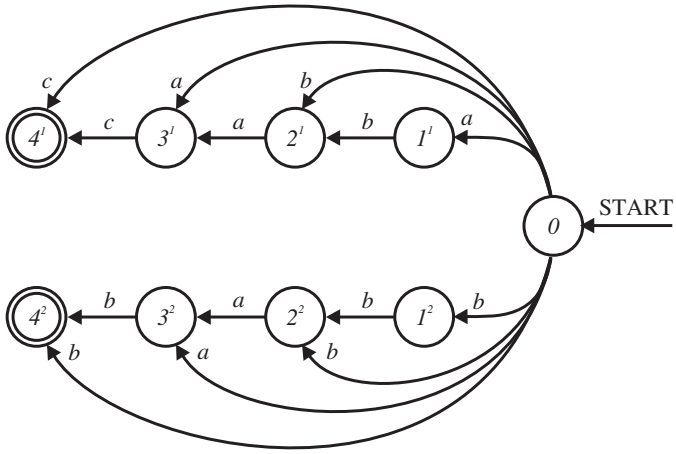

Figure 8.16: Transition diagram of the nondeterministic suffix automaton from Example 8.10

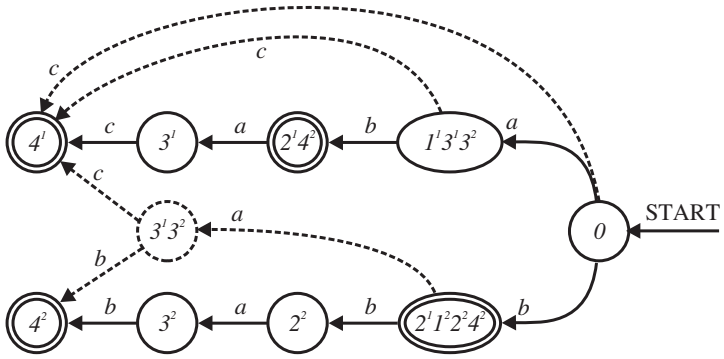

Figure 8.17: Transition diagram of the deterministic suffix automaton and its backbone from Example 8.10

8.2.4 Commentz–Walter algorithm

Commentz–Walter (CW) algorithm [CW79] is the classical algorithm devoted to the backward pattern matching of a finite set of patterns. It uses:
multi bad character shift ($MBCS$, see Section 8.2.1),
multi good suffix shift ($MGSS$, see Section 8.2.2), and
reduced multi good prefix shift ($RMGPS$, see Section 8.2.3.2).

The basic principle of CW algorithm is based on the selection of the longer shift from $MBCS$ and $MGSS$ shifts. Let us show the principle of CW algorithm using an example.

Example 8.11
Let the set of patterns be $S = \{cbaba, bba, abbb\}$ over the alphabet $A = \{a, b, c, d\}$ (see Examples 8.4 and 8.6). The $MBCS$ table is shown in Example 8.4. The $MGSS$ table is shown in the Table 8.6 (see Example 8.6). The finite automaton for multibackward pattern matching is depicted in Fig. 8.9. □

The multi backward searching algorithm ($MBPM$, see Fig. 8.3) uses the transition table defining the mapping δ of the finite automaton accepting the set of reversed patterns from the set S^R (see the transition table of the automaton M_2 shown in Table 8.4). The adaptation of the multi backward searching algorithm ($MBPM$) for CW algorithm requires only replacement of statement

$(*)$ $I := I + 1;$
by statement
$I := I + min(LMIN, min(RMGPS[I],$
$max(MBCS[TEXT[I + J]], MGSS[J])));$

where $RMGPS$ is reduced multi good prefix shift, $MBCS$ is multi bad character shift and $MGSS$ is multi good suffix shift.

Example 8.12
Let the set of patterns be $S = \{cbaba, bba, abbb\}$ over the alphabet $A = \{a, b, c, d\}$ (see Example 8.11). The behaviour of CW algorithm for the text $T = abdbbacbababbb$ is visualised in the Fig. 8.18.

8.2.5 Looking for antifactors of a finite set of patterns

The principle based on the looking for antifactors of one string (see Section 7.2.5) can be extended for a set of strings. The use of *multi antifactor shift* ($MAFS$) is described in this section. We will use both factor automaton and factor oracle automaton for a set of patterns in order to compute $MAFS$.

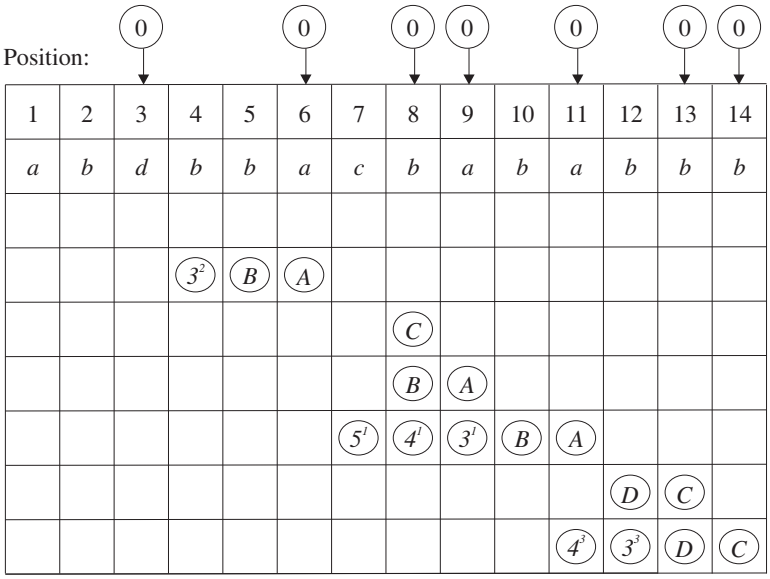

Figure 8.18: Behaviour of *MBPM* algorithm adapted for *CW* Algorithm for the set of reversed patterns $S^R = \{cbaba^R, bba^R, abbb^R\}$ from Example 8.12

8.2.5.1 Multi backward factor matching We present *multi backward factor matching* (*MBFM*). Let us show the use of a factor automaton for looking for antifactors of a set of patterns in the next Example.

Example 8.13
Let set of patterns be $S = \{abba, cabc\}$ over alphabet $A = \{a, b, c\}$. The set of reversed paterns $S^R = \{abba^R, cabc^R\} = \{abba, cbac\}$. We construct the factor automaton for this set of reversed patterns. Transition diagram of the nondeterministic factor automaton M_1 for set of reversed patterns S^R is depicted in Fig. 8.19. Transition table of the nondeterministic factor automaton M_1 for set $S^R = \{abba, cbac\}$ is shown in Table 8.12. Transition table of the deterministic factor automaton M_2 is shown in Table 8.13 and its transition diagram is depicted in Fig. 8.20.

The factor automaton is used for the identification of the longest factor of the set of patterns while reading text backwards. As soon as a mismatch occurs reading symbol z, an antifactor of the set of patterns is recognised. The set of patterns is then shifted to the right behind symbol z which does not belong to the factor of the set of patterns. This shift is, however, limited by *lmin* which is the length of the shorterst element of the set. If the pattern is found, then the shift is given by *matchshift*. This is a parametr of the pattern matching algorithm (*MBFM*), see Fig. 8.21.

	a	b	c
0	$1^1, 4^1, 3^2$	$2^1, 3^1, 2^2$	$1^2, 4^2$
1^1		2^1	
2^1		3^1	
3^1	4^1		
4^1			
1^2		2^2	
2^2	3^2		
3^2			4^2
4^2			

Table 8.12: Transition table of the nondeterministic factor automaton M_1 for set $S^R = \{abba, cbac\}$ from Example 8.13

	a	b	c
0	$1^1 4^1 3^2$	$2^1 3^1 2^2$	$1^2 4^2$
$1^1 4^1 3^2$		2^1	4^2
2^1		3^1	
$2^1 3^1 2^2$	$4^1 3^2$	3^1	
3^1	4^1		
4^1			
$4^1 3^2$			4^2
$1^2 4^2$		2^2	
2^2	3^2		
3^2			4^2
4^2			

Table 8.13: Transition table of the deterministic factor automaton M_2 for set $S^R = \{abba, cbac\}$ from Example 8.13

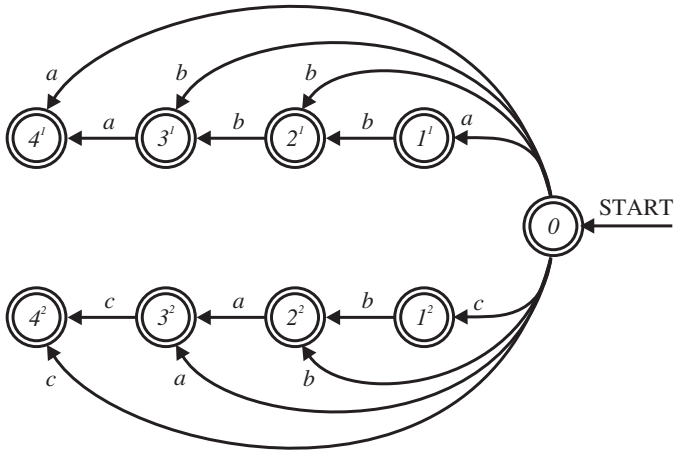

Figure 8.19: Transition diagram of the nondeterministic factor automaton M_1 for set $S^R = \{abba, cbac\}$ from Example 8.13

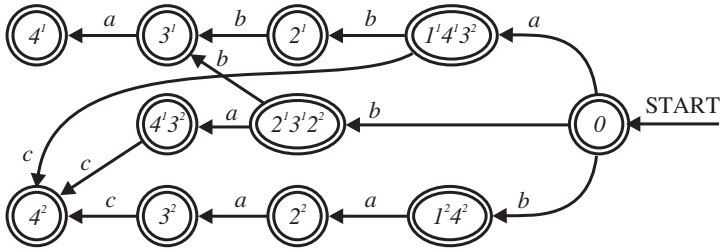

Figure 8.20: Transition diagram of the deterministic factor automaton M_2 for set $S^R = \{abba, cbaa\}$ from Example 8.13

```
const M = {length of pattern};
      MATCHSHIFT = {length of shift when pattern is found};
var TEXT : array [1..N] of char;
    I,J : integer;
    STATE : TSTATE;
    δ : array[1..MAXSTATE,1..MAXSYMBOL] of TSTATE;
    F : set of TSTATE;
    LFACTOR: integer;
    SHIFT: integer;
    ...
begin
    LFACTOR := 0;
    STATE := q0;
    I := 0;
    J := M;
    while (I ≤ N-M) do
      begin
        if δ [STATE,TEXT[I+J]] = fail
          then
            begin
              if LFACTOR = M then
                begin
                  output(I + 1);
                  SHIFT := MATCHSHIFT;
                end
              else
                SHIFT := M - LFACTOR;
                LFACTOR := 0;
                STATE := q0;
                I := I + SHIFT;
                J := M;
            end;
          else
            begin
              STATE := δ[STATE,TEXT[I + J]];
              J := J - 1;
              LFACTOR := LFACTOR + 1;
            end;
      end;
end;
```

Figure 8.21: *MBFM* and *MBOM* algorithms

Configuration of the *MBFM* algorithm is a quadruple:
(*state,I,J,lfactor*).
The initial configuration is $(q_0, 0, lmin, 0)$.

Example 8.14
Let us use the factor automaton for the set of patterns $S^R = \{abba, cbac\}$ from Example 8.13 and show the pattern matching in text:

$T =$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
c	a	a	b	b	a	a	b	b	a	a	b	c	c

MBFM algorithm performs the following sequence of steps:

$$
\begin{aligned}
(0,0,4,0) &\overset{b}{\vdash} (2^1 3^1 2^2, 0, 3, 1)\\
&\overset{a}{\vdash} (4^1 3^2, 0, 2, 2) \textit{shift } 2\\
&\vdash (0, 2, 4, 0)\\
&\overset{a}{\vdash} (1^1 4^1 3^2, 2, 3, 1)\\
&\overset{b}{\vdash} (2^1, 2, 2, 2)\\
&\overset{b}{\vdash} (3^1, 2, 1, 3)\\
&\overset{a}{\vdash} (4^1, 2, 0, 4) \textit{match, matchshift } = 3\\
&\vdash (0, 5, 4, 0)\\
&\overset{b}{\vdash} (2^1 3^1 2^2, 5, 3, 1)\\
&\overset{b}{\vdash} (3^1, 5, 2, 2)\\
&\overset{a}{\vdash} (4^1, 5, 1, 3) \textit{shift } 1\\
&\vdash (0, 6, 4, 0)\\
&\overset{a}{\vdash} (1^1 4^1 3^2, 6, 3, 1)\\
&\overset{b}{\vdash} (2^1, 6, 2, 2)\\
&\overset{b}{\vdash} (3^1, 6, 1, 3)\\
&\overset{a}{\vdash} (4^1, 6, 0, 4) \textit{match, matchshift } = 3\\
&\vdash (0, 9, 4, 0)\\
&\overset{c}{\vdash} (1^2 4^2, 9, 3, 1)\\
&\overset{b}{\vdash} (2^2, 9, 2, 2)\\
&\overset{a}{\vdash} (3^2, 9, 1, 3) \textit{match, matchshift } = 3\\
&\vdash (0, 12, 4, 0)
\end{aligned}
$$

At this point the searching is finished, because the shift is behind the text.

8.2.5.2 Multi backward oracle matching The factor oracle automaton for the looking for antifactors in a set of strings can be used as well. The method using this approach is called also Multi Backward Oracle Matching (*MBOM*). The basic tool for the looking for antifactors of the set of patterns is a factors oracle automaton for the reversed set of patterns. This automaton is accepting the set of all factors of set of reversed patterns and moreover some of it subsequences. Let us show the use of a factor oracle automaton for looking for antifactors in the next example.

Example 8.15
Let set of patterns be $S = \{abba, cabc\}$ over alphabet $A = \{a, b, c\}$ as in Example 8.13. Transition diagram of the nondeterministic factor automaton for the set $S^R = \{abba, cbac\}$ $M_1(S^R)$ is depicted in Fig. 8.19. Deterministic factor automaton $M_2(S^R)$ has transition diagram shown in Fig. 8.20. There is possible to obtain a factor oracle automaton by merging some states of the factor automaton. We can use three approaches:

1. The first approach is merging states $\{2^1 3^1 2^2, 2^1\}$ and $\{4^1 3^2, 3^2\}$. The resulting factor oracle automaton M_{O1} has transition diagram depicted in Fig. 8.22. The language accepted by factor oracle automaton M_{O1} is $L_1(Oracle(\{abba, cbac\})) = L(M_{O1}) = Fact(\{abba, cbac\}) \cup \{aba, abac\}$.

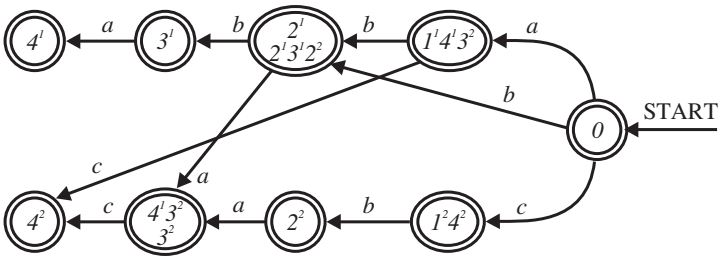

Figure 8.22: Transition diagram of factor oracle automaton M_{O1} for the set of patterns $S^R = \{abba, cbac\}$ from Example 8.14 obtained by merging states $\{2^1 3^1 2^2, 2^1\}$ and $\{4^1 3^2, 3^2\}$

2. The second approach is merging states $\{2^1 3^1 2^2, 2^2\}$ and $\{4^1 3^2, 3^2\}$. The resulting factor oracle automaton M_{O2} has transition diagram depicted in Fig. 8.23. The language accepted by factor oracle automaton M_{O2} is

   $L_2(Oracle(\{abba, cbac\})) = L(M_{O2}) = Fact(\{abba, cbac\}) \cup \{cbb, cbba\}$.

3. The third approach is merging states $\{2^1 3^1 2^2, 2^1, 2^2\}$ and $\{4^1 3^2, 3^2\}$. The resulting factor oracle automaton M_{O3} has transition diagram depicted in Fig. 8.24. The language accepted by factor oracle automaton M_{O3} is

$$L_3(Oracle(\{abba, cbac\})) = L(M_{O3}) = L(M_{O1}) \cup L(M_{O2}) =$$
$$Fact(\{abba, cbac\}) \cup \{cbb, cbba, aba, abac\}. \qquad \square$$

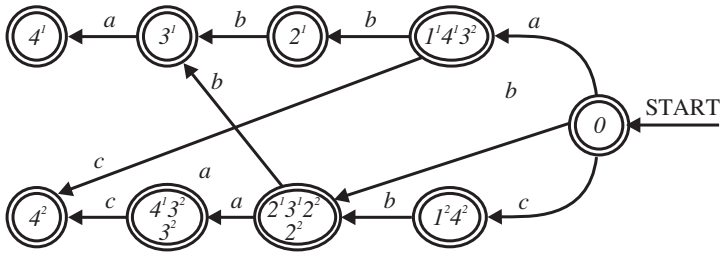

Figure 8.23: Transition diagram of factor oracle automaton M_{O2} for the set of reversed patterns $S^R = \{abba, cbac\}$ from Example 8.14 obtained by merging states $\{2^1 3^1 2^2, 2^2\}$ and $\{4^1 3^2, 3^2\}$

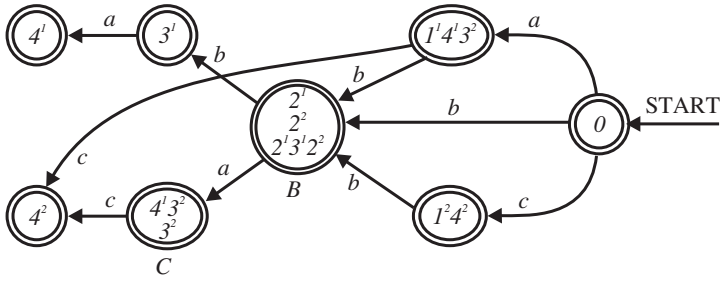

Figure 8.24: Transition diagram of factor oracle automaton M_{O3} for the set of reversed patterns $S^R = \{abba, cbac\}$ from Example 8.14 obtained by merging states $\{2^1 3^1 2^2, 2^1, 2^2\}$ and $\{4^1 3^2, 3^2\}$

Let us remember, that the configuration of *MBOM* algorithm is a quadruple:
$(state, I, J, lfactor)$.

The initial configuration is
$(q_0, 0, lmin, 0)$.

Example 8.16
Let use use the factor oracle automaton for the set of patterns $S^R = \{abba, cbac\}$ from Example 8.15 and show the pattern matching for the same text as in Example 8.13.

$T =$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
c	*a*	*a*	*b*	*b*	*a*	*a*	*b*	*b*	*a*	*a*	*b*	*c*	*c*

MBOM algorithm performs the following sequence of steps using the factor oracle automaton M_{O3} having transition diagram depicted in Fig. 8.24.

$$
\begin{aligned}
(0,0,4,0) &\overset{b}{\vdash} (B,0,3,1)\\
&\overset{a}{\vdash} (C,0,2,2)\mathit{shift}\ 2\\
&\vdash (0,2,4,0)\\
&\overset{a}{\vdash} (1^1 4^1 3^2,2,3,1)\\
&\overset{b}{\vdash} (B,2,2,2)\\
&\overset{b}{\vdash} (3^1,2,1,3)\\
&\overset{a}{\vdash} (4^1,2,0,4)\mathit{match},\ \mathit{matchshift}\ = 3\\
&\vdash (0,5,4,0)\\
&\overset{b}{\vdash} (B,5,3,1)\\
&\overset{b}{\vdash} (3^1,5,2,2)\\
&\overset{a}{\vdash} (4^1,5,1,3)\mathit{shift}\ 1\\
&\vdash (0,6,4,0)\\
&\overset{a}{\vdash} (1^1 4^1 3^2,6,3,1)\\
&\overset{b}{\vdash} (B,6,2,2)\\
&\overset{b}{\vdash} (3^1,6,1,3)\\
&\overset{a}{\vdash} (4^1,6,0,4)\mathit{match},\ \mathit{matchshift}\ = 3\\
&\vdash (0,9,4,0)\\
&\overset{C}{\vdash} (1^2 4^2,9,3,1)\\
&\overset{b}{\vdash} (B,9,2,2)\\
&\overset{a}{\vdash} (C,9,1,3)\mathit{shift}\ 2\\
&\vdash (0,11,4,0)
\end{aligned}
$$

At this point the searching is finished, because the shift is behind the text. □

References