

PVO30 Textual Information Systems

Petr Sojka

Faculty of Informatics
Masaryk University, Brno

Spring 2013

Outline (week two)

- ① Watson
- ② Exact search methods I (without pattern preprocessing) – completion.
- ③ Exact search methods II (with pattern preprocessing, left to right): KMP (animation), Rabin-Karp, AC.
- ④ Search with an automaton.

Outline (week two)

- ① Watson
- ② Exact search methods I (without pattern preprocessing) – completion.
- ③ Exact search methods II (with pattern preprocessing, left to right): KMP (animation), Rabin-Karp, AC.
- ④ Search with an automaton.

Outline (week two)

- ① Watson
- ② Exact search methods I (without pattern preprocessing) – completion.
- ③ Exact search methods II (with pattern preprocessing, left to right): KMP (animation), Rabin-Karp, AC.
- ④ Search with an automaton.

Evaluation of questionnaire

- ① Yes: syllabus suits expectations; positively is awaited dissect of Google; indexing and search; examples.
- ② No: too much theory, deep digestion of algorithms.
- ③ Examples.
- ④ This year: further enrichment of information retrieval part (Google), textual (mathematical) digital libraries and languages enhancements of TIS (on the example of Watson).

Evaluation of questionnaire

- ① Yes: syllabus suits expectations; positively is awaited dissect of Google; indexing and search; examples.
- ② No: too much theory, deep digestion of algorithms.
- ③ Examples.
- ④ This year: further enrichment of information retrieval part (Google), textual (mathematical) digital libraries and languages enhancements of TIS (on the example of Watson).

Evaluation of questionnaire

- ① Yes: syllabus suits expectations; positively is awaited dissect of Google; indexing and search; examples.
- ② No: too much theory, deep digestion of algorithms.
- ③ Examples.
- ④ This year: further enrichment of information retrieval part (Google), textual (mathematical) digital libraries and languages enhancements of TIS (on the example of Watson).

II – Exact search with query preprocessing

Karp-Rabin search algorithm

Motivation

- ① Search in text editor (Vim, Emacs), in the source code of a web page.
- ② Data search (biological molecules approximated as sequences of nucleotides or amino acids).
- ③ Literature/abstracts search—recherche, corpus linguistics.

The size of available data doubles every 18 months (Moore's law) → higher effectiveness of algorithms needed.

Left-to-right direct search methods

During the preprocessing, structure of the query pattern(s) is examined and, on that basis, the search engine is built (on-the-fly).

Definition: *exact* (vs. *fuzzy (proximitní)*) search aims at exact match (localization of searched pattern(s)).

Definition: *left-to-right (LR, sousměrné)* (vs. *right-to-left (RL, protisměrné)*) search compares query pattern to the text from left to right (vs. right to left).

Left-to-right direct search methods

During the preprocessing, structure of the query pattern(s) is examined and, on that basis, the search engine is built (on-the-fly).

Definition: **exact** (vs. **fuzzy (proximitní)**) search aims at exact match (localization of searched pattern(s)).

Definition: **left-to-right (LR, sousměrné)** (vs. **right-to-left (RL, protisměrné)**) search compares query pattern to the text from left to right (vs. right to left).

Left-to-right direct search methods

During the preprocessing, structure of the query pattern(s) is examined and, on that basis, the search engine is built (on-the-fly).

Definition: **exact** (vs. **fuzzy (proximitní)**) search aims at exact match (localization of searched pattern(s)).

Definition: **left-to-right (LR, sousměrné)** (vs. **right-to-left (RL, protisměrné)**) search compares query pattern to the text from left to right (vs. right to left).

Left-to-right methods

- ① 1 query pattern (vzorek):
 - Shift-Or algorithm.
 - Karp-Rabin algorithm, (KR, 1987).
 - Knuth-Morris-Pratt algorithm, (KMP, designed (MP) in 1970, published 1977).
- ② n patterns: Aho-Corasick algorithm, (AC, 1975).
- ③ ω patterns: construction of a search engine (finite automaton) for the search of a potentially infinite set of patterns (given as regular expression).

Left-to-right methods

- ① 1 query pattern (vzorek):
 - Shift-Or algorithm.
 - Karp-Rabin algorithm, (KR, 1987).
 - Knuth-Morris-Pratt algorithm, (KMP, designed (MP) in 1970, published 1977).
- ② n patterns: Aho-Corasick algorithm, (AC, 1975).
- ③ ω patterns: construction of a search engine (finite automaton) for the search of a potentially infinite set of patterns (given as regular expression).

Left-to-right methods

- ① 1 query pattern (vzorek):
 - Shift-Or algorithm.
 - Karp-Rabin algorithm, (KR, 1987).
 - Knuth-Morris-Pratt algorithm, (KMP, designed (MP) in 1970, published 1977).
- ② n patterns: Aho-Corasick algorithm, (AC, 1975).
- ③ ω patterns: construction of a search engine (finite automaton) for the search of a potentially infinite set of patterns (given as regular expression).

Shift-Or algorithm

- Pattern $v_1v_2 \dots v_m$ over an alphabet $\Sigma = a_1, \dots, a_c$.
- Incidence matrix X ($m \times c$), $X_{ij} = \begin{cases} 0 & \text{if } v_i = a_j \\ 1 & \text{otherwise.} \end{cases}$
- Let matrix column X corresponding to a_j is named A_j .
- At the beginning, we put unitary vector/column into R . In every algorithm, step R moves down by one line/position, top-most position is filled by zero and one character a_j is read from input. Resulted R is combined with A_j by binary disjunction:
 $R := \text{SHIFT}(R) \text{ OR } A_j$.
- Algorithm stops successfully when 0 appears at the bottom-most position in R .

Shift-Or algorithm

- Pattern $v_1v_2 \dots v_m$ over an alphabet $\Sigma = a_1, \dots, a_c$.
- Incidence matrix X ($m \times c$), $X_{ij} = \begin{cases} 0 & \text{if } v_i = a_j \\ 1 & \text{otherwise.} \end{cases}$
- Let matrix column X corresponding to a_j is named A_j .
- At the beginning, we put unitary vector/column into R . In every algorithm, step R moves down by one line/position, top-most position is filled by zero and one character a_j is read from input. Resulted R is combined with A_j by binary disjunction:
 $R := \text{SHIFT}(R) \text{ OR } A_j$.
- Algorithm stops successfully when 0 appears at the bottom-most position in R .

Shift-Or algorithm

- Pattern $v_1v_2 \dots v_m$ over an alphabet $\Sigma = a_1, \dots, a_c$.
- Incidence matrix X ($m \times c$), $X_{ij} = \begin{cases} 0 & \text{if } v_i = a_j \\ 1 & \text{otherwise.} \end{cases}$
- Let matrix column X corresponding to a_j is named A_j .
- At the beginning, we put unitary vector/column into R . In every algorithm, step R moves down by one line/position, top-most position is filled by zero and one character a_j is read from input. Resulted R is combined with A_j by binary disjunction:
 $R := \text{SHIFT}(R) \text{ OR } A_j$.
- Algorithm stops successfully when 0 appears at the bottom-most position in R .

Shift-Or algorithm

- Pattern $v_1v_2 \dots v_m$ over an alphabet $\Sigma = a_1, \dots, a_c$.
- Incidence matrix X ($m \times c$), $X_{ij} = \begin{cases} 0 & \text{if } v_i = a_j \\ 1 & \text{otherwise.} \end{cases}$
- Let matrix column X corresponding to a_j is named A_j .
- At the beginning, we put unitary vector/column into R . In every algorithm, step R moves down by one line/position, top-most position is filled by zero and one character a_j is read from input. Resulted R is combined with A_j by binary disjunction:
 $R := \text{SHIFT}(R) \text{ OR } A_j$.
- Algorithm stops successfully when 0 appears at the bottom-most position in R .

Shift-Or algorithm

- Pattern $v_1v_2 \dots v_m$ over an alphabet $\Sigma = a_1, \dots, a_c$.
- Incidence matrix X ($m \times c$), $X_{ij} = \begin{cases} 0 & \text{if } v_i = a_j \\ 1 & \text{otherwise.} \end{cases}$
- Let matrix column X corresponding to a_j is named A_j .
- At the beginning, we put unitary vector/column into R . In every algorithm, step R moves down by one line/position, top-most position is filled by zero and one character a_j is read from input. Resulted R is combined with A_j by binary disjunction:
 $R := \text{SHIFT}(R) \text{ OR } A_j$.
- Algorithm stops successfully when 0 appears at the bottom-most position in R .

Shift-Or algorithm (cont.) – example

Example: $V = \text{vzorek}$ over $\Sigma = \{e, k, o, r, v, z\}$.
Cf. [POK, page 31–32].

Karp-Rabin search

Quite different approach: usage of hash function. Instead of matching of pattern with text on every position, we check the match only when pattern ‘looks similar’ as searched text substring. For similarity, a hash function is used. It has to be

- ☞ efficiently computable,
- ☞ and it should be good at separating different strings (close to perfect hashing).

KR search is quadratic at the worst case, but on average $O(T + V)$.

Karp-Rabin search

Quite different approach: usage of hash function. Instead of matching of pattern with text on every position, we check the match only when pattern ‘looks similar’ as searched text substring. For similarity, a hash function is used. It has to be

- ☞ efficiently computable,
- ☞ and it should be good at separating different strings (close to perfect hashing).

KR search is quadratic at the worst case, but on average $O(T + V)$.

Karp-Rabin search

Quite different approach: usage of hash function. Instead of matching of pattern with text on every position, we check the match only when pattern ‘looks similar’ as searched text substring. For similarity, a hash function is used. It has to be

- ☞ efficiently computable,
- ☞ and it should be good at separating different strings (close to perfect hashing).

KR search is quadratic at the worst case, but on average $O(T + V)$.

Karp-Rabin search (cont.)—implementation

```
#define REHASH(a, b, h) (((h-a*d)<<1+b)
void KR(char *y, char *x, int n, int m) {
int hy, hx, d, i;
/* preprocessing: computation of  $d = 2^{m-1}$  */
d=1; for (i=1; i<m; i++) d<<=1;
hx=hy=0;
for (i=0; i<m; i++)
  { hx=((hx<<1)+x[i]); hy=((hy<<1)+y[i]); }
/* search */
for (i=m; i<=n; i++) {
  if (hy==hx) && strncmp(y+i-m,x,m)==0) OUTPUT(i-m);
  hy=REHASH(y[i-m], y[i], hy);
} }
```

Karp-Rabin search (cont.)—example

Example: ([HCS, Ch. 6]) $V = ing$, $T = string$ matching.

Preprocessing: $hash = 105 \times 2^2 + 110 \times 2 + 103 = 743$.

Search:

T=	s	t	r	i	n	g	
hash=			806	797	776	743	678
	m	a	t	c	h	i	n
	585	443	746	719	766	709	736
							743

Part I

Exact search of one pattern

(K)MP

Search engine (finite automaton)

Construction of the KMP engine

Morris-Pratt algorithm (MP)

Idea: Inefficiency of naïve search are caused by the fact that in the case of mismatch the pattern is shifted by only one position to the right and checking starts from the beginning. This does not use the information that was gained by the inspection of text position that failed. The idea is to shift as much as possible so that we do not have to go back in searched text.

The main part of the (K)MP algorithm

```

var text: array[1..T] of char; pattern: array[1..V] of char;
i, j: integer; found: boolean;
i := 1;                                     ▷ text index
j := 1;                                     ▷ pattern index
while (i ≤ T) and (j ≤ V) do
    while (j > 0) and (text[i] ≠ pattern[j]) do
        j := h[j];
    end while
    i := i + 1; j := j + 1
end while
found := j > V;                             ▷ if found, it is on the position i - V

```

Analysis of (K)MP

- ▶ $O(T)$ complexity plus complexity of preprocessing (creation of the array h).
- ▶ Animation of tracing of the main part of KMP.

Analysis of (K)MP

- ☞ $O(T)$ complexity plus complexity of preprocessing (creation of the array h).
- ☞ Animation of tracing of the main part of KMP.

Knuth-Morris-Pratt algorithm

- ☞ h is used when prefix of pattern $v_1v_2 \dots v_{j-1}$ matches with substring of text $t_{i-j+1}t_{i-j+2} \dots t_{i-1}$ and $v_j \neq t_i$.
- ☞ May I shift by more than 1? By j ? How to compute h ?
- ☞ $h(j)$ the biggest $k < j$ such that $v_1v_2 \dots v_{k-1}$ is suffix of $v_1v_2 \dots v_{j-1}$, e.g. $v_1v_2 \dots v_{k-1} = v_{j-k+1}v_{j-k+2} \dots v_{j-1}$ and $v_j \neq v_k$.
- ☞ KMP: backward transitions for so long, so that $j = 0$ (prefix of pattern is not contained in the searched text) or $t_i = v_j$ ($v_1v_2 \dots v_j = t_{i-j+1}t_{i-j+2} \dots t_{i-1}t_i$).
- ☞ Animation Lecroq, also [POK, page 27], also see [MAR] for detailed description.

Knuth-Morris-Pratt algorithm

- ☞ h is used when prefix of pattern $v_1v_2 \dots v_{j-1}$ matches with substring of text $t_{i-j+1}t_{i-j+2} \dots t_{i-1}$ and $v_j \neq t_i$.
- ☞ May I shift by more than 1? By j ? How to compute h ?
- ☞ $h(j)$ the biggest $k < j$ such that $v_1v_2 \dots v_{k-1}$ is suffix of $v_1v_2 \dots v_{j-1}$, e.g. $v_1v_2 \dots v_{k-1} = v_{j-k+1}v_{j-k+2} \dots v_{j-1}$ and $v_j \neq v_k$.
- ☞ KMP: backward transitions for so long, so that $j = 0$ (prefix of pattern is not contained in the searched text) or $t_i = v_j$ ($v_1v_2 \dots v_j = t_{i-j+1}t_{i-j+2} \dots t_{i-1}t_i$).
- ☞ Animation Lecroq, also [POK, page 27], also see [MAR] for detailed description.

Knuth-Morris-Pratt algorithm

- ☞ h is used when prefix of pattern $v_1v_2 \dots v_{j-1}$ matches with substring of text $t_{i-j+1}t_{i-j+2} \dots t_{i-1}$ and $v_j \neq t_i$.
- ☞ May I shift by more than 1? By j ? How to compute h ?
- ☞ $h(j)$ the biggest $k < j$ such that $v_1v_2 \dots v_{k-1}$ is suffix of $v_1v_2 \dots v_{j-1}$, e.g. $v_1v_2 \dots v_{k-1} = v_{j-k+1}v_{j-k+2} \dots v_{j-1}$ and $v_j \neq v_k$.
- ☞ KMP: backward transitions for so long, so that $j = 0$ (prefix of pattern is not contained in the searched text) or $t_i = v_j$ ($v_1v_2 \dots v_j = t_{i-j+1}t_{i-j+2} \dots t_{i-1}t_i$).
- ☞ Animation Lecroq, also [POK, page 27], also see [MAR] for detailed description.

Knuth-Morris-Pratt algorithm

- ☞ h is used when prefix of pattern $v_1v_2 \dots v_{j-1}$ matches with substring of text $t_{i-j+1}t_{i-j+2} \dots t_{i-1}$ and $v_j \neq t_i$.
- ☞ May I shift by more than 1? By j ? How to compute h ?
- ☞ $h(j)$ the biggest $k < j$ such that $v_1v_2 \dots v_{k-1}$ is suffix of $v_1v_2 \dots v_{j-1}$, e.g. $v_1v_2 \dots v_{k-1} = v_{j-k+1}v_{j-k+2} \dots v_{j-1}$ and $v_j \neq v_k$.
- ☞ KMP: backward transitions for so long, so that $j = O$ (prefix of pattern is not contained in the searched text) or $t_i = v_j$ ($v_1v_2 \dots v_j = t_{i-j+1}t_{i-j+2} \dots t_{i-1}t_i$).
- ☞ Animation Lecroq, also [POK, page 27], also see [MAR] for detailed description.

Knuth-Morris-Pratt algorithm

- ☞ h is used when prefix of pattern $v_1v_2 \dots v_{j-1}$ matches with substring of text $t_{i-j+1}t_{i-j+2} \dots t_{i-1}$ and $v_j \neq t_i$.
- ☞ May I shift by more than 1? By j ? How to compute h ?
- ☞ $h(j)$ the biggest $k < j$ such that $v_1v_2 \dots v_{k-1}$ is suffix of $v_1v_2 \dots v_{j-1}$, e.g. $v_1v_2 \dots v_{k-1} = v_{j-k+1}v_{j-k+2} \dots v_{j-1}$ and $v_j \neq v_k$.
- ☞ KMP: backward transitions for so long, so that $j = O$ (prefix of pattern is not contained in the searched text) or $t_i = v_j$ ($v_1v_2 \dots v_j = t_{i-j+1}t_{i-j+2} \dots t_{i-1}t_i$).
- ☞ Animation Lecroq, also [POK, page 27], also see [MAR] for detailed description.

Construction of h for KMP

```
i:=1; j:=0; h[1]:=0;
while (i<V) do
  begin while (j>0) and (v[i]<>v[j]) do j:=h[j];
    i:=i+1; j:=j+1;
    if (i<=V) and (v[i]=v[j])
      then h[i]:=h[j] else h[i]:=j (*MP*)
    end;
```

Complexity of h computation, e.g. preprocessing, is $O(V)$, thus in total $O(T + V)$.

Example: h for *ababa*. KMP vs. MP.

Universal search algorithm,

that uses transition table g derived from the searched pattern,
(g relates to the transition function δ of FA):

```

var i,T:integer; found: boolean;
text: array[1..T] of char; state,q0: TSTATE;
g:array[1..maxstate,1..maxsymb] of TSTATE;
F: set of TSTATE;...
begin
  found:= FALSE; state:= q0; i:=0;
  while (i <= T) and not found do
    begin
      i:=i+1; state:= g[state,text[i]];
      found:= state in F;
    end;
  end;
end;

```

How to transform pattern into g ?

Search engine (SE) for left-to-right search

☞ **SE for left-to-right search** $A = (Q, T, g, h, q_0, F)$

- Q is a finite set of states.
- T is a finite input alphabet.
- $g: Q \times T \rightarrow Q \cup \{\text{fail}\}$ is a forward state-transition function.
- $h: (Q - q_0) \rightarrow Q$ is a backward state-transition function.
- q_0 is an initial state.
- F is a set of final states.

☞ **A depth of the state** $q: d(q) \in \mathbb{N}_0$ is a length of the shortest forward sequence of the state transitions from q_0 to q .

Search engine (SE) for left-to-right search

☞ **SE for left-to-right search** $A = (Q, T, g, h, q_0, F)$

- Q is a finite set of states.
- T is a finite input alphabet.
- $g: Q \times T \rightarrow Q \cup \{\text{fail}\}$ is a forward state-transition function.
- $h: (Q - q_0) \rightarrow Q$ is a backward state-transition function.
- q_0 is an initial state.
- F is a set of final states.

☞ **A depth of the state** $q: d(q) \in N_0$ is a length of the shortest forward sequence of the state transitions from q_0 to q .

Search engine (cont.)

☞ Characteristics g, h :

- $g(q_0, a) \neq \text{fail}$ for $\forall a \in T$ (there is no backward transition in the initial state).
- If $h(q) = p$, then $d(p) < d(q)$ (the number of the backward transitions is restricted from the top by a multiple of the maximum depth of the state c and the sum of the forward transitions V). So the speed of searching is linear in relation to V .

Search engine (cont.)

☞ Characteristics g, h :

- $g(q_0, a) \neq \text{fail}$ for $\forall a \in T$ (there is no backward transition in the initial state).
- If $h(q) = p$, then $d(p) < d(q)$ (the number of the backward transitions is restricted from the top by a multiple of the maximum depth of the state c and the sum of the forward transitions V). So the speed of searching is linear in relation to V .

SE configuration, transition

- ☞ **SE configuration** (q, w) , $q \in Q$, $w \in T^*$ the not yet searched part of the text.
- ☞ **An initial configuration of SE** (q_0, w) , w is the entire searched text.
- ☞ **An accepting configuration of SE** (q, w) , $q \in F$, w is the not yet searched text, the found pattern is immediately before w .
- ☞ **SE transition**: relation $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
 - $g(q, a) = p$, then $(q, aw) \vdash (p, w)$ **forward transition** for $\forall w \in T^*$.
 - $h(q) = p$, then $(q, w) \vdash (p, w)$ **backward transition** for $\forall w \in T^*$.

SE configuration, transition

- ☞ **SE configuration** (q, w) , $q \in Q$, $w \in T^*$ the not yet searched part of the text.
- ☞ **An initial configuration of SE** (q_0, w) , w is the entire searched text.
- ☞ **An accepting configuration of SE** (q, w) , $q \in F$, w is the not yet searched text, the found pattern is immediately before w .
- ☞ **SE transition**: relation $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
 - $g(q, a) = p$, then $(q, aw) \vdash (p, w)$ **forward transition** for $\forall w \in T^*$.
 - $h(q) = p$, then $(q, w) \vdash (p, w)$ **backward transition** for $\forall w \in T^*$.

SE configuration, transition

- ☞ **SE configuration** (q, w) , $q \in Q$, $w \in T^*$ the not yet searched part of the text.
- ☞ **An initial configuration of SE** (q_0, w) , w is the entire searched text.
- ☞ **An accepting configuration of SE** (q, w) , $q \in F$, w is the not yet searched text, the found pattern is immediately before w .
- ☞ **SE transition**: relation $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
 - $g(q, a) = p$, then $(q, aw) \vdash (p, w)$ **forward transition** for $\forall w \in T^*$.
 - $h(q) = p$, then $(q, w) \vdash (p, w)$ **backward transition** for $\forall w \in T^*$.

SE configuration, transition

- ☞ **SE configuration** (q, w) , $q \in Q$, $w \in T^*$ the not yet searched part of the text.
- ☞ **An initial configuration of SE** (q_0, w) , w is the entire searched text.
- ☞ **An accepting configuration of SE** (q, w) , $q \in F$, w is the not yet searched text, the found pattern is immediately before w .
- ☞ **SE transition**: relation $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
 - $g(q, a) = p$, then $(q, aw) \vdash (p, w)$ **forward transition** for $\forall w \in T^*$.
 - $h(q) = p$, then $(q, w) \vdash (p, w)$ **backward transition** for $\forall w \in T^*$.

SE configuration, transition

- ☞ **SE configuration** (q, w) , $q \in Q$, $w \in T^*$ the not yet searched part of the text.
- ☞ **An initial configuration of SE** (q_0, w) , w is the entire searched text.
- ☞ **An accepting configuration of SE** (q, w) , $q \in F$, w is the not yet searched text, the found pattern is immediately before w .
- ☞ **SE transition**: relation $\vdash \subseteq (Q \times T^*) \times (Q \times T^*)$:
 - $g(q, a) = p$, then $(q, aw) \vdash (p, w)$ **forward transition** for $\forall w \in T^*$.
 - $h(q) = p$, then $(q, w) \vdash (p, w)$ **backward transition** for $\forall w \in T^*$.

Searching with SE

During the forward transition, a single input symbol is read and the engine switches to the next state p . However, if $g(q, a) = \underline{\text{fail}}$, the backward transition is executed without reading an input symbol. $S = O(T)$ (we measure the number of SE transitions).

Construction of the KMP SE for pattern $v_1v_2 \dots v_V$

- ① An initial state q_0 .
- ② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1v_2 \dots v_jv_{j+1}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.
- ⑥ The backward state-transition function h is defined on the page 17 by the below mentioned algorithm.

Construction of the KMP SE for pattern $v_1v_2 \dots v_V$

- ① An initial state q_0 .
- ② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1v_2 \dots v_jv_{j+1}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.
- ⑥ The backward state-transition function h is defined on the page 17 by the below mentioned algorithm.

Construction of the KMP SE for pattern $v_1v_2 \dots v_V$

- ① An initial state q_0 .
- ② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1v_2 \dots v_jv_{j+1}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.
- ⑥ The backward state-transition function h is defined on the page 17 by the below mentioned algorithm.

Construction of the KMP SE for pattern $v_1v_2 \dots v_V$

- ① An initial state q_0 .
- ② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1v_2 \dots v_jv_{j+1}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.
- ④ $g(q, a) = \text{fail}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.
- ⑥ The backward state-transition function h is defined on the page 17 by the below mentioned algorithm.

Construction of the KMP SE for pattern $v_1v_2 \dots v_V$

- ① An initial state q_0 .
- ② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1v_2 \dots v_jv_{j+1}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.
- ④ $g(q, a) = \text{fail}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.
- ⑥ The backward state-transition function h is defined on the page 17 by the below mentioned algorithm.

Construction of the KMP SE for pattern $v_1v_2 \dots v_V$

- ① An initial state q_0 .
- ② $g(q, v_{j+1}) = q'$, where q' is equivalent to the prefix $v_1v_2 \dots v_jv_{j+1}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous step.
- ④ $g(q, a) = \text{fail}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.
- ⑥ The backward state-transition function h is defined on the page 17 by the below mentioned algorithm.

Outline (week two)

- ① Summary of the previous lecture, searching with SE.
- ② Left-to-right search of n patterns algorithms. (AC, NFA \rightarrow DFA.)
- ③ Left-to-right search of infinite patterns algorithms.
- ④ Regular expressions (RE).
- ⑤ Direct construction of (N)FA for given RE.

Outline (week two)

- ① Summary of the previous lecture, searching with SE.
- ② Left-to-right search of n patterns algorithms. (AC, NFA \rightarrow DFA.)
- ③ Left-to-right search of infinite patterns algorithms.
- ④ Regular expressions (RE).
- ⑤ Direct construction of (N)FA for given RE.

Outline (week two)

- ① Summary of the previous lecture, searching with SE.
- ② Left-to-right search of n patterns algorithms. (AC, NFA \rightarrow DFA.)
- ③ Left-to-right search of infinite patterns algorithms.
- ④ Regular expressions (RE).
- ⑤ Direct construction of (N)FA for given RE.

Outline (week two)

- ① Summary of the previous lecture, searching with SE.
- ② Left-to-right search of n patterns algorithms. (AC, NFA \rightarrow DFA.)
- ③ Left-to-right search of infinite patterns algorithms.
- ④ Regular expressions (RE).
- ⑤ Direct construction of (N)FA for given RE.

Outline (week two)

- ① Summary of the previous lecture, searching with SE.
- ② Left-to-right search of n patterns algorithms. (AC, NFA \rightarrow DFA.)
- ③ Left-to-right search of infinite patterns algorithms.
- ④ Regular expressions (RE).
- ⑤ Direct construction of (N)FA for given RE.

Part II

Search of a finite set of patterns

Search of n patterns

Aho-Corasick algorithm

Finite automata for searching

Search of a set of patterns

SE for left-to-right search of a set of patterns $p = \{v^1, v^2, \dots, v^P\}$.

Instead of repeated search of text for every pattern, there is only “one” pass (FA).

Common SE algorithm

```
var text: array[1..T] of char;  
    i: integer; found: boolean; state: tstate;  
    g: array[1..maxstate,1..maxsymbol] of tstate;  
    h: array[1..maxstate] of tstate; F: set of tstate;  
found:=false; state:=q0; i:=0;  
while (i<=T) and not found do  
begin i:=i+1;  
    while g[state,text[i]]=fail do state:=h[state];  
    state:=g[state,text[i]]; found:=state in F  
end
```


Common SE algorithm (cont.)

- Construction of the state-transition functions h, g ?
- How about for P patterns? The main idea?
- Aho, Corasick, 1975 (AC search engine).

Common SE algorithm (cont.)

- Construction of the state-transition functions h, g ?
- How about for P patterns? The main idea?
- Aho, Corasick, 1975 (AC search engine).

Common SE algorithm (cont.)

- Construction of the state-transition functions h, g ?
- How about for P patterns? The main idea?
- Aho, Corasick, 1975 (AC search engine).

Aho-Corasick algorithm I

Construction of g for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

- ① An initial state q_0 .
- ② $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \dots b_{j+1}$ of the pattern v^i , for $\forall i \in \{1, \dots, P\}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous steps.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.

An example: $p = \{he, ehe, her\}$ over $T = \{h, e, r, s, x\}$, where x is anything else than $\{h, e, r, s\}$.

Aho-Corasick algorithm I

Construction of g for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

- ① An initial state q_0 .
- ② $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1b_2 \dots b_{j+1}$ of the pattern v^i , for $\forall i \in \{1, \dots, P\}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous steps.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.

An example: $p = \{he, she, her\}$ over $T = \{h, e, r, s, x\}$, where x is anything else than $\{h, e, r, s\}$.

Aho-Corasick algorithm I

Construction of g for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

- ① An initial state q_0 .
- ② $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \dots b_{j+1}$ of the pattern v^i , for $\forall i \in \{1, \dots, P\}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous steps.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.

An example: $p = \{he, she, her\}$ over $T = \{h, e, r, s, x\}$, where x is anything else than $\{h, e, r, s\}$.

Aho-Corasick algorithm I

Construction of g for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

- ① An initial state q_0 .
- ② $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \dots b_{j+1}$ of the pattern v^i , for $\forall i \in \{1, \dots, P\}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous steps.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.

An example: $p = \{he, she, her\}$ over $T = \{h, e, r, s, x\}$, where x is anything else than $\{h, e, r, s\}$.

Aho-Corasick algorithm I

Construction of g for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

- ① An initial state q_0 .
- ② $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \dots b_{j+1}$ of the pattern v^i , for $\forall i \in \{1, \dots, P\}$.
- ③ For q_0 , we define $g(q_0, a) = q_0$ for $\forall a$, for which $g(q_0, a)$ has not been defined in the previous steps.
- ④ $g(q, a) = \underline{\text{fail}}$ for $\forall q$ and a , for which $g(q, a)$ has not been defined in the previous steps.
- ⑤ A state that corresponds to the complete pattern is the final one.

An example: $p = \{he, she, her\}$ over $T = \{h, e, r, s, x\}$, where x is anything else than $\{h, e, r, s\}$.

The failure function h (AC II)

Construction of h for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

At first, we define the failure function f inductively relative to the depth of the states this way:

- ① For $\forall q$ of the depth 1, $f(q) = q_0$.
- ② Let us assume that f is defined for each state of the depth d and lesser. The variable q_D denotes the state of the depth d and $g(q_D, a) = q'$. Then we compute $f(q')$ as follows:

```
q := f(q_D);  
while g(q, a) = fail do q := f(q);  
f(q') := g(q, a).
```

The failure function h (AC II)

Construction of h for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

At first, we define the failure function f inductively relative to the depth of the states this way:

- ① For $\forall q$ of the depth 1, $f(q) = q_0$.
- ② Let us assume that f is defined for each state of the depth d and lesser. The variable q_D denotes the state of the depth d and $g(q_D, a) = q'$. Then we compute $f(q')$ as follows:

$q := f(q_D);$

while $g(q, a) = \underline{\text{fail}}$ **do** $q := f(q);$

$f(q') := g(q, a).$

The failure function h (AC II)

Construction of h for AC SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

At first, we define the failure function f inductively relative to the depth of the states this way:

- ① For $\forall q$ of the depth 1, $f(q) = q_0$.
- ② Let us assume that f is defined for each state of the depth d and lesser. The variable q_D denotes the state of the depth d and $g(q_D, a) = q'$. Then we compute $f(q')$ as follows:

$q := f(q_D);$

while $g(q, a) = \underline{\text{fail}}$ **do** $q := f(q);$

$f(q') := g(q, a).$

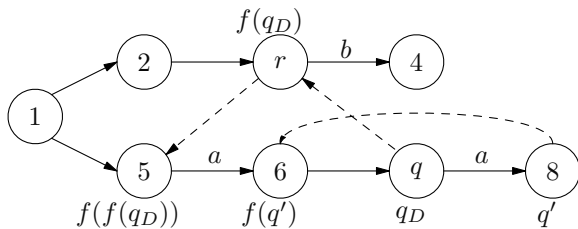
The failure function h (AC II, cont.)

- The cycle terminates, since $g(q_0, a) \neq \text{fail}$.
- If the states q, r represent prefixes u, v of some of the patterns from p , then $f(q) = r \Leftrightarrow v$ is the longest proper suffix u .

The failure function h (AC II, cont.)

- The cycle terminates, since $g(q_0, a) \neq \text{fail}$.
- If the states q, r represent prefixes u, v of some of the patterns from p , then $f(q) = r \Leftrightarrow v$ is the longest proper suffix u .

The failure function h (AC III)



Construction of h for AC SE for a set of patterns

$p = \{v^1, v^2, \dots, v^P\}$ (cont.)

- We could use f as the backward state-transition function h , however, redundant backward transitions would be performed.
- We define function h inductively relative to the depth of the states this way:
 - For \forall state q of the depth 1, $h(q) = q_0$.
 - Let us assume that h is defined for each state of the depth d and lesser. Let the depth q be $d + 1$. If the set of letters, for which is in a state $f(q)$ the value of the function g different from fail, is the subset of the set of letters, for which is the value of the function g in a state q different from fail, then $h(q) := h(f(q))$, otherwise $h(q) := f(q)$.

Construction of h for AC SE for a set of patterns

$p = \{v^1, v^2, \dots, v^P\}$ (cont.)

- We could use f as the backward state-transition function h , however, redundant backward transitions would be performed.
- We define function h inductively relative to the depth of the states this way:
 - For \forall state q of the depth 1, $h(q) = q_0$.
 - Let us assume that h is defined for each state of the depth d and lesser. Let the depth q be $d + 1$. If the set of letters, for which is in a state $f(q)$ the value of the function g different from fail, is the subset of the set of letters, for which is the value of the function g in a state q different from fail, then $h(q) := h(f(q))$, otherwise $h(q) := f(q)$.

Construction of h for AC SE for a set of patterns

$p = \{v^1, v^2, \dots, v^P\}$ (cont.)

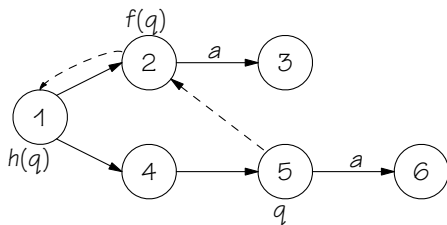
- We could use f as the backward state-transition function h , however, redundant backward transitions would be performed.
- We define function h inductively relative to the depth of the states this way:
 - For \forall state q of the depth 1, $h(q) = q_0$.
 - Let us assume that h is defined for each state of the depth d and lesser. Let the depth q be $d + 1$. If the set of letters, for which is in a state $f(q)$ the value of the function g different from fail, is the subset of the set of letters, for which is the value of the function g in a state q different from fail, then $h(q) := h(f(q))$, otherwise $h(q) := f(q)$.

Construction of h for AC SE for a set of patterns

$p = \{v^1, v^2, \dots, v^P\}$ (cont.)

- We could use f as the backward state-transition function h , however, redundant backward transitions would be performed.
- We define function h inductively relative to the depth of the states this way:
 - For \forall state q of the depth 1, $h(q) = q_0$.
 - Let us assume that h is defined for each state of the depth d and lesser. Let the depth q be $d + 1$. If the set of letters, for which is in a state $f(q)$ the value of the function g different from fail, is the subset of the set of letters, for which is the value of the function g in a state q different from fail, then $h(q) := h(f(q))$, otherwise $h(q) := f(q)$.

Construction of h for AC SE (cont.)



Finite automata for searching

Deterministic finite automaton (DFA) $M=(K,T,\delta,q_0,F)$

- ① K is a finite set of inner states.
- ② T is a finite input alphabet.
- ③ δ is a projection from $K \times T$ to K .
- ④ $q_0 \in K$ is an initial state.
- ⑤ $F \subseteq K$ is a set of final states.

Finite automata for searching

Deterministic finite automaton (DFA) $M=(K,T,\delta,q_0,F)$

- ① K is a finite set of inner states.
- ② T is a finite input alphabet.
- ③ δ is a projection from $K \times T$ to K .
- ④ $q_0 \in K$ is an initial state.
- ⑤ $F \subseteq K$ is a set of final states.

Finite automata for searching

Deterministic finite automaton (DFA) $M=(K,T,\delta,q_0,F)$

- ① K is a finite set of inner states.
- ② T is a finite input alphabet.
- ③ δ is a projection from $K \times T$ to K .
- ④ $q_0 \in K$ is an initial state.
- ⑤ $F \subseteq K$ is a set of final states.

Finite automata for searching

Deterministic finite automaton (DFA) $M=(K,T,\delta,q_0,F)$

- ① K is a finite set of inner states.
- ② T is a finite input alphabet.
- ③ δ is a projection from $K \times T$ to K .
- ④ $q_0 \in K$ is an initial state.
- ⑤ $F \subseteq K$ is a set of final states.

Finite automata for searching

Deterministic finite automaton (DFA) $M=(K,T,\delta,q_0,F)$

- ① K is a finite set of inner states.
- ② T is a finite input alphabet.
- ③ δ is a projection from $K \times T$ to K .
- ④ $q_0 \in K$ is an initial state.
- ⑤ $F \subseteq K$ is a set of final states.

Finite automata for searching

- ① **Completely specified automaton** if δ is defined for every pair $(q, a) \in K \times T$, otherwise **incompletely specified automaton**.
- ② **Configuration M** is a pair (q, w) , where $q \in K$, $w \in T^*$ is the not yet searched part of the text.
- ③ **An initial configuration M** is (q_0, w) , where w is the entire text to be searched.
- ④ **An accepting configuration M** is (q, w) , where $q \in F$ and $w \in T^*$.

Finite automata for searching

- ① **Completely specified automaton** if δ is defined for every pair $(q, a) \in K \times T$, otherwise **incompletely specified automaton**.
- ② **Configuration M** is a pair (q, w) , where $q \in K$, $w \in T^*$ is the not yet searched part of the text.
- ③ **An initial configuration M** is (q_0, w) , where w is the entire text to be searched.
- ④ **An accepting configuration M** is (q, w) , where $q \in F$ and $w \in T^*$.

Finite automata for searching

- ① **Completely specified automaton** if δ is defined for every pair $(q, a) \in K \times T$, otherwise **incompletely specified automaton**.
- ② **Configuration M** is a pair (q, w) , where $q \in K$, $w \in T^*$ is the not yet searched part of the text.
- ③ **An initial configuration M** is (q_0, w) , where w is the entire text to be searched.
- ④ **An accepting configuration M** is (q, w) , where $q \in F$ and $w \in T^*$.

Finite automata for searching

- ① **Completely specified automaton** if δ is defined for every pair $(q, a) \in K \times T$, otherwise **incompletely specified automaton**.
- ② **Configuration M** is a pair (q, w) , where $q \in K$, $w \in T^*$ is the not yet searched part of the text.
- ③ **An initial configuration M** is (q_0, w) , where w is the entire text to be searched.
- ④ **An accepting configuration M** is (q, w) , where $q \in F$ and $w \in T^*$.

Searching with FA M

During the transition, a single input symbol is read and the engine switches to the next state p .

- ☞ **Transition M** : is defined by a state and an input symbol; relation $\vdash \subseteq (K \times T^*) \times (K \times T^*)$; if $\delta(q, a) = p$, then $(q, aw) \vdash (p, w)$ for every $\forall w \in T^*$.
- ☞ **The k th power, transitive** or more precisely **transitive reflexive closure** of the relation \vdash : $\vdash^k, \vdash^+, \vdash^*$.
- ☞ $L(M) = \{w \in T^* : (q_0, w) \vdash^* (q, w') \text{ for some } q \in F, w' \in T^*\}$ **the language accepted by FA M** .
- ☞ time complexity $O(T)$ (we measure the number of transitions of FA M).

Searching with FA M

During the transition, a single input symbol is read and the engine switches to the next state p .

- ☞ **Transition M** : is defined by a state and an input symbol; relation $\vdash \subseteq (K \times T^*) \times (K \times T^*)$; if $\delta(q, a) = p$, then $(q, aw) \vdash (p, w)$ for every $\forall w \in T^*$.
- ☞ **The k th power, transitive** or more precisely **transitive reflexive closure** of the relation \vdash : $\vdash^k, \vdash^+, \vdash^*$.
- ☞ $L(M) = \{w \in T^* : (q_0, w) \vdash^* (q, w') \text{ for some } q \in F, w' \in T^*\}$ **the language accepted by FA M** .
- ☞ time complexity $O(T)$ (we measure the number of transitions of FA M).

Searching with FA M

During the transition, a single input symbol is read and the engine switches to the next state p .

- ☞ **Transition M** : is defined by a state and an input symbol; relation $\vdash \subseteq (K \times T^*) \times (K \times T^*)$; if $\delta(q, a) = p$, then $(q, aw) \vdash (p, w)$ for every $\forall w \in T^*$.
- ☞ **The k th power, transitive** or more precisely **transitive reflexive closure** of the relation \vdash : $\vdash^k, \vdash^+, \vdash^*$.
- ☞ $L(M) = \{w \in T^* : (q_0, w) \vdash^* (q, w') \text{ for some } q \in F, w' \in T^*\}$ **the language accepted by FA M** .
- ☞ time complexity $O(T)$ (we measure the number of transitions of FA M).

Searching with FA M

During the transition, a single input symbol is read and the engine switches to the next state p .

- ☞ **Transition M** : is defined by a state and an input symbol; relation $\vdash \subseteq (K \times T^*) \times (K \times T^*)$; if $\delta(q, a) = p$, then $(q, aw) \vdash (p, w)$ for every $\forall w \in T^*$.
- ☞ **The k th power, transitive** or more precisely **transitive reflexive closure** of the relation \vdash : $\vdash^k, \vdash^+, \vdash^*$.
- ☞ $L(M) = \{w \in T^* : (q_0, w) \vdash^* (q, w') \text{ for some } q \in F, w' \in T^*\}$ **the language accepted by FA M** .
- ☞ time complexity $O(T)$ (we measure the number of transitions of FA M).

Nondeterministic FA

Definition: **Nondeterministic finite automaton** (NFA) is $M = (K, T, \delta, q_0, F)$, where K, T, q_0, F are the same as those in the deterministic version of FA, but $\delta : K \times T \rightarrow 2^K$ $\delta(q, a)$ is now **a set** of states.

Definition: $\vdash \in (K \times T^*) \times (K \times T^*)$ **transition**: if $p \in \delta(q, a)$, then $(q, aw) \vdash (p, w)$ for $\forall w \in T^*$.

Definition: a final state, $L(M)$ analogically as in DFA.

Nondeterministic FA

Definition: **Nondeterministic finite automaton** (NFA) is $M = (K, T, \delta, q_0, F)$, where K, T, q_0, F are the same as those in the deterministic version of FA, but $\delta : K \times T \rightarrow 2^K$ $\delta(q, a)$ is now **a set** of states.

Definition: $\vdash \in (K \times T^*) \times (K \times T^*)$ **transition**: if $p \in \delta(q, a)$, then $(q, aw) \vdash (p, w)$ for $\forall w \in T^*$.

Definition: a final state, $L(M)$ analogically as in DFA.

Nondeterministic FA

Definition: **Nondeterministic finite automaton** (NFA) is $M = (K, T, \delta, q_0, F)$, where K, T, q_0, F are the same as those in the deterministic version of FA, but $\delta : K \times T \rightarrow 2^K$ $\delta(q, a)$ is now **a set** of states.

Definition: $\vdash \in (K \times T^*) \times (K \times T^*)$ **transition**: if $p \in \delta(q, a)$, then $(q, aw) \vdash (p, w)$ for $\forall w \in T^*$.

Definition: a final state, $L(M)$ analogically as in DFA.

Construction of SE (DFA) from NFA

Theorem: for every nondeterministic finite automaton $M=(K,T,\delta,q_0,F)$, we can build deterministic finite automaton $M'=(K',T,\delta',q'_0,F')$ such that $L(M) = L(M')$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):

Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.

Output: deterministic FA.

- ① $K' = \{\{q_0\}\}$, state $\{q_0\}$ in unmarked.
- ② If there are in K' all the states marked, continue to the step 4.
- ③ We choose from K' unmarked state q' :
 - $\delta'(q', a) = \bigcup \{\delta(p, a)\}$ for $\forall p \in q'$ and $a \in T$;
 - $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
 - we mark q' and continue to the step 2.
- ④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):

Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.

Output: deterministic FA.

- ① $K' = \{\{q_0\}\}$, state $\{q_0\}$ in unmarked.
- ② If there are in K' all the states marked, continue to the step 4.
- ③ We choose from K' unmarked state q' :
 - $\delta'(q', a) = \bigcup \{\delta(p, a)\}$ for $\forall p \in q'$ and $a \in T$;
 - $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
 - we mark q' and continue to the step 2.
- ④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):

Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.

Output: deterministic FA.

- ① $K' = \{\{q_0\}\}$, state $\{q_0\}$ in unmarked.
- ② If there are in K' all the states marked, continue to the step 4.
- ③ We choose from K' unmarked state q' :
 - $\delta'(q', a) = \bigcup \{\delta(p, a)\}$ for, $\forall p \in q'$ and $a \in T$;
 - $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
 - we mark q' and continue to the step 2.
- ④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):

Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.

Output: deterministic FA.

- ① $K' = \{\{q_0\}\}$, state $\{q_0\}$ in unmarked.
- ② If there are in K' all the states marked, continue to the step 4.
- ③ We choose from K' unmarked state q' :
 - $\delta'(q', a) = \bigcup \{\delta(p, a)\}$ for $\forall p \in q'$ and $a \in T$;
 - $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
 - we mark q' and continue to the step 2.
- ④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):

Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.

Output: deterministic FA.

- ① $K' = \{\{q_0\}\}$, state $\{q_0\}$ in unmarked.
- ② If there are in K' all the states marked, continue to the step 4.
- ③ We choose from K' unmarked state q' :
 - $\delta'(q', a) = \bigcup \{\delta(p, a)\}$ for $\forall p \in q'$ and $a \in T$;
 - $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
 - we mark q' and continue to the step 2.
- ④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of SE (DFA) from NFA (cont.)

A constructive proof (of the algorithm):

Input: nondeterministic FA $M = (K, T, \delta, q_0, F)$.

Output: deterministic FA.

- ① $K' = \{\{q_0\}\}$, state $\{q_0\}$ in unmarked.
- ② If there are in K' all the states marked, continue to the step 4.
- ③ We choose from K' unmarked state q' :
 - $\delta'(q', a) = \bigcup \{\delta(p, a)\}$ for $\forall p \in q'$ and $a \in T$;
 - $K' = K' \cup \delta'(q', a)$ for $\forall a \in T$;
 - we mark q' and continue to the step 2.
- ④ $q'_0 = \{q_0\}$; $F' = \{q' \in K' : q' \cap F \neq \emptyset\}$.

Construction of g for SE

Construction g' for SE for a set of patterns $p = \{v^1, v^2, \dots, v^P\}$

① We create NFA M :

- An initial state q_0 .
- For $\forall a \in T$, we define $g(q_0, a) = q_0$.
- For $\forall i \in \{1, \dots, P\}$, we define $g(q, b_{j+1}) = q'$, where q' is equivalent to the prefix $b_1 b_2 \dots b_{j+1}$ of the pattern v^i .
- The state corresponding to the entire pattern is the final one.

② ...and its corresponding DFA M' with g' .

Part III

Search for an infinite set of patterns

Left-to-right methods

Regular expression (RE)

Definition: **Regular expression E over the alphabet A :**

- ① ε, \mathbf{O} are RE and for $\forall a \in A$ is a RE.
- ② If x, y are RE over A , then:
 - $(x + y)$ is RE (union);
 - $(x.y)$ is RE (concatenation);
 - $(x)^*$ is RE (iteration).

A convention about priority of regular operations:

union $<$ concatenation $<$ iteration.

Definition: Thereafter, we consider as a **(generalized) regular expression** even those terms that do not contain, with regard to this convention, the unnecessary parentheses.

Value of RE

$$\textcircled{1} \quad h(\mathbf{0}) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$$

$$\textcircled{2} \quad \bullet \quad h(x + y) = h(x) \cup h(y)$$

$$\bullet \quad h(x.y) = h(x).h(y)$$

$$\bullet \quad h(x^*) = (h(x))^*$$

$$\textcircled{3} \quad h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$$

$\textcircled{4}$ The value of RE is a regular language (RL).

$\textcircled{5}$ Every RL can be represented as RE.

$\textcircled{6}$ For $\forall \text{ RE } Y \exists \text{ FA } M: h(Y) = L(M)$.

Value of RE

$$\textcircled{1} \quad h(\mathbf{0}) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$$

$$\textcircled{2} \quad \bullet \quad h(x + y) = h(x) \cup h(y)$$

$$\bullet \quad h(x.y) = h(x).h(y)$$

$$\bullet \quad h(x^*) = (h(x))^*$$

$$\Rightarrow h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$$

\Rightarrow The value of RE is a regular language (RL).

\Rightarrow Every RL can be represented as RE.

\Rightarrow For \forall RE \exists FA M : $h(Y) = L(M)$.

Value of RE

$$\textcircled{1} \quad h(\emptyset) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$$

$$\textcircled{2} \quad \bullet \quad h(x + y) = h(x) \cup h(y)$$

$$\bullet \quad h(x.y) = h(x).h(y)$$

$$\bullet \quad h(x^*) = (h(x))^*$$

$$\Rightarrow h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$$

\Rightarrow The value of RE is a regular language (RL).

\Rightarrow Every RL can be represented as RE.

\Rightarrow For \forall RE \exists FA M : $h(V) = L(M)$.

Value of RE

$$\textcircled{1} \quad h(\mathbf{0}) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$$

$$\textcircled{2} \quad \bullet \quad h(x + y) = h(x) \cup h(y)$$

$$\bullet \quad h(x.y) = h(x).h(y)$$

$$\bullet \quad h(x^*) = (h(x))^*$$

$$\Rightarrow h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$$

\Rightarrow The value of RE is a regular language (RL).

\Rightarrow Every RL can be represented as RE.

\Rightarrow For \forall RE $V \exists$ FA $M: h(V) = L(M)$.

Value of RE

$$\textcircled{1} \quad h(\emptyset) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$$

$$\textcircled{2} \quad \bullet \quad h(x + y) = h(x) \cup h(y)$$

$$\bullet \quad h(x.y) = h(x).h(y)$$

$$\bullet \quad h(x^*) = (h(x))^*$$

$$\Rightarrow h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$$

\Rightarrow The value of RE is a regular language (RL).

\Rightarrow Every RL can be represented as RE.

\Rightarrow For $\forall \text{ RE } V \exists \text{ FA } M: h(V) = L(M)$.

Value of RE

$$\textcircled{1} \quad h(\emptyset) = \emptyset, h(\varepsilon) = \{\varepsilon\}, h(a) = \{a\}$$

$$\textcircled{2} \quad \bullet \quad h(x + y) = h(x) \cup h(y)$$

$$\bullet \quad h(x.y) = h(x).h(y)$$

$$\bullet \quad h(x^*) = (h(x))^*$$

$$\Rightarrow h(x^*) = \varepsilon \cup x \cup x.x \cup x.x.x \cup \dots$$

\Rightarrow The value of RE is a regular language (RL).

\Rightarrow Every RL can be represented as RE.

\Rightarrow For $\forall \text{ RE } V \exists \text{ FA } M: h(V) = L(M)$.

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (x + \varepsilon)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (x + \varepsilon)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (x + \varepsilon)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (x + \varepsilon)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (x + \varepsilon)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (x + \varepsilon)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $0.x = 0$ inverse element for concatenation

A9: $x + 0 = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $\mathbf{O}.x = \mathbf{O}$ inverse element for concatenation

A9: $x + \mathbf{O} = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $\mathbf{O}.x = \mathbf{O}$ inverse element for concatenation

A9: $x + \mathbf{O} = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $\mathbf{0}.x = \mathbf{0}$ inverse element for concatenation

A9: $x + \mathbf{0} = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$

Axiomatization of RE (Salomaa 1966)

A1: $x + (y + z) = (x + y) + z = x + y + z$ associativity of union

A2: $x.(y.z) = (x.y).z = x.y.z$ associativity of concatenation

A3: $x + y = y + x$ commutativity of union

A4: $(x + y).z = x.z + y.z$ right distributivity

A5: $x.(y + z) = x.y + x.z$ left distributivity

A6: $x + x = x$ idempotence of union

A7: $\varepsilon.x = x$ identity element for concatenation

A8: $\mathbf{O}.x = \mathbf{O}$ inverse element for concatenation

A9: $x + \mathbf{O} = x$ identity element for union

A10: $x^* = \varepsilon + x^*x$

A11: $x^* = (\varepsilon + x)^*$