460-4065: Teoretická informatika (TI)

prof. RNDr Petr Jančar, CSc.

katedra informatiky FEI VŠB-TUO
www.cs.vsb.cz/jancar

Základní informace o kursu

http://www.cs.vsb.cz/jancar/TEORET-INF/teoret-inf.htm

Návaznost na kurs Úvod do teoretické informatiky z bakalářského studia (prohloubení a rozšíření vybraných partií teorie jazyků a automatů, vyčíslitelnosti, výpočetní složitosti, algoritmů, ...
pravděpodobnostní algoritmy, aproximační algoritmy, ...).

Základním pracovním textem je
P. Jančar: Teoretická informatika, VŠB-TU, Ostrava, 2007 (2010);
pdf-soubor na webu (rozsah 336 stran).

(Nejen) teoretická informatika se pochopitelně opírá o logiku.
M. Duží: Logika pro informatiky, VŠB-TU Ostrava 2012 (pdf na webu).

Na webu také informace o průběhu kursu po jednotlivých týdnech (mj. zadání cvičení předem).

Zápočet a zkouška

- Zápočtová písemka (45-minutová, v určeném týdnu ke konci semestru). Celkově bude možné získat 21 bodů. Nutnou podmínkou k získání zápočtu je zisk alespoň 7 bodů.
  Jeden pokus, který v zásadě není možné opakovat.
  (Další info na webu, včetně možnosti $x \geq 11$, zápis $7 + (x-11)/2$.)
- Referát (další nutná podmínka k získání zápočtu). Do termínu uvedeného na webu bude každému přiděleno zadání emailem. Prověření podkladů a skutečného porozumění proběhne v určeném termínu ke konci semestru. Za obhájený referát lze získat 5-10 bodů. (Nebo 1-5 bodů, další info na webu.)
- Aktivita na cvičení, 0-7 bodů. (Konkrétní informace cvičící.)

Zápočet: min. $7 + 1 + 0 = 8$ bodů, max. $21 + 10 + 7 = 38$ bodů.

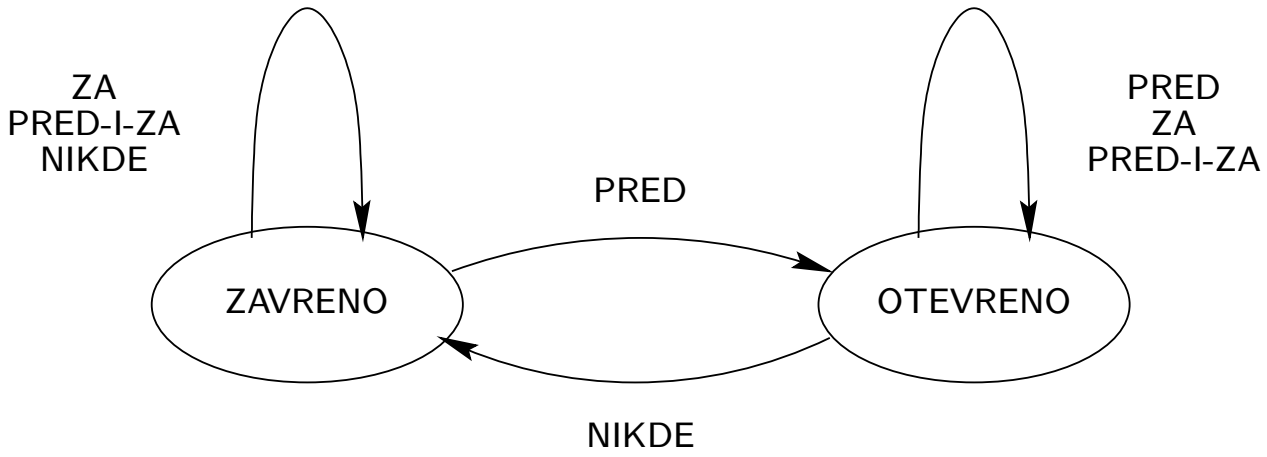
Zkouška: písemná (90-minutová), podle potřeby doplněná ústní částí; max. 62 bodů; dvě (tématické) části po 31 bodech. V každé části nutno získat alespoň 11 bodů a celkově alespoň 25 bodů.
Ke zkoušce je možné jít jen po splnění požadavků k zápočtu.

Cvičení, konzultace

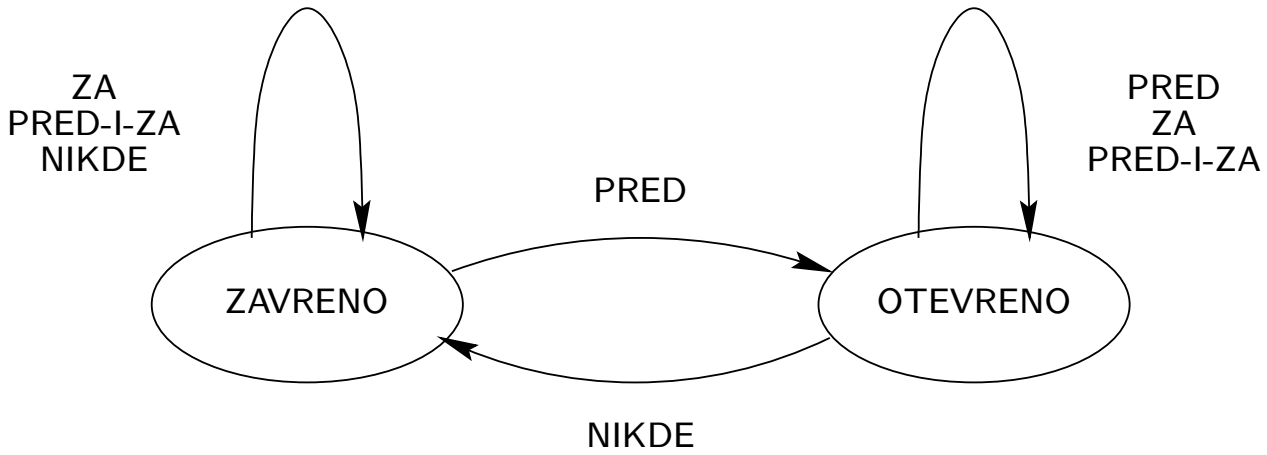
Cvičení ... předpokládá se předběžná příprava a aktivita !

Konzultace ... primárně v konzultačních hodinách vyučujících (avizujte emailem nebo se domluvte po cvičení či přednášce).

Konečný automat (jako model systému)

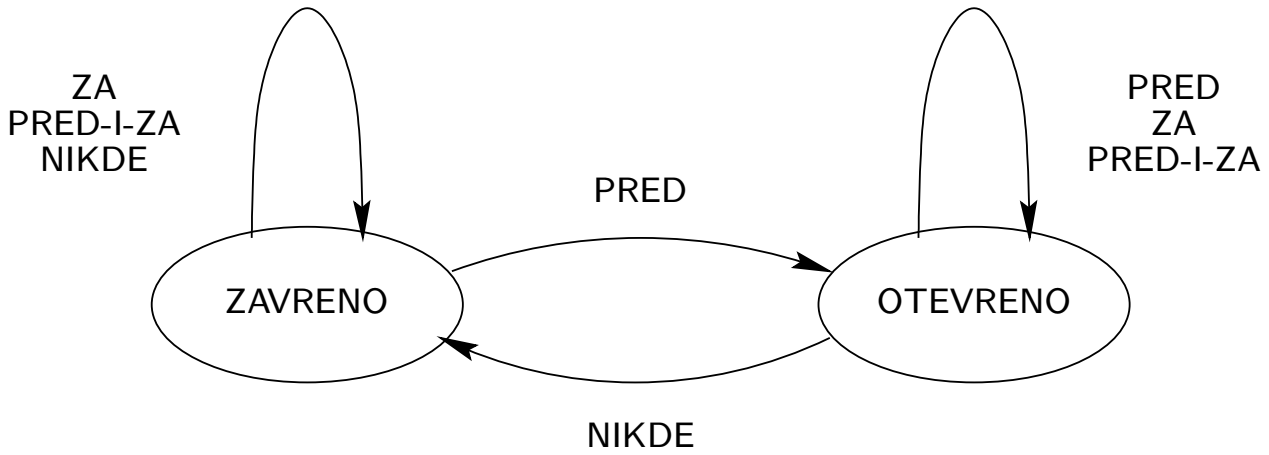


Konečný automat (jako model systému)



	PRED	ZA	PR-I-ZA	NIKDE
ZAV	OTEV	ZAV	ZAV	ZAV
OTEV	OTEV	OTEV	OTEV	ZAV

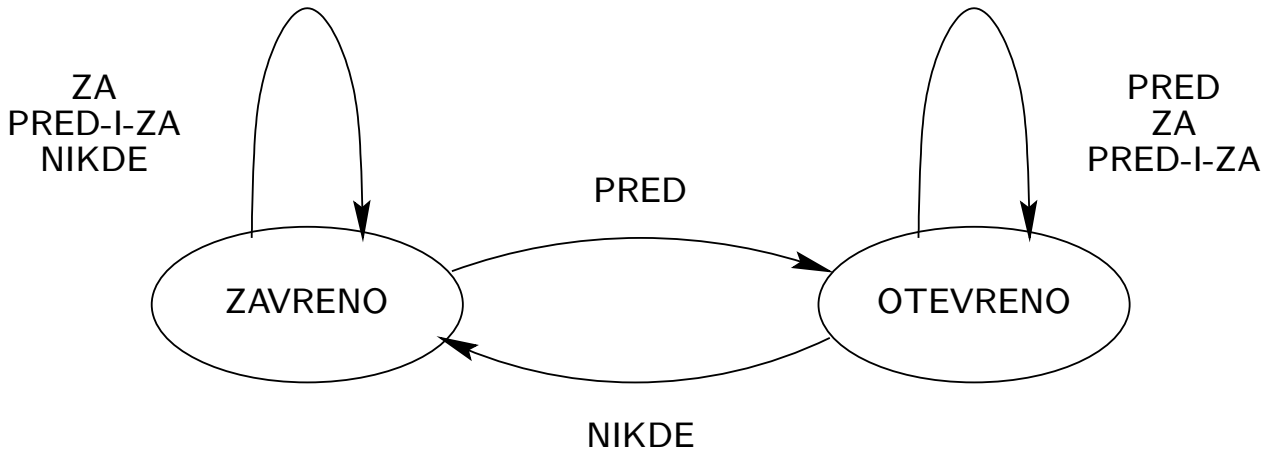
Konečný automat (jako model systému)



	PRED	ZA	PR-I-ZA	NIKDE
ZAV	OTEV	ZAV	ZAV	ZAV
OTEV	OTEV	OTEV	OTEV	ZAV

$A = (Q, \Sigma, \delta)$

Konečný automat (jako model systému)



	PRED	ZA	PR-I-ZA	NIKDE
ZAV	OTEV	ZAV	ZAV	ZAV
OTEV	OTEV	OTEV	OTEV	ZAV

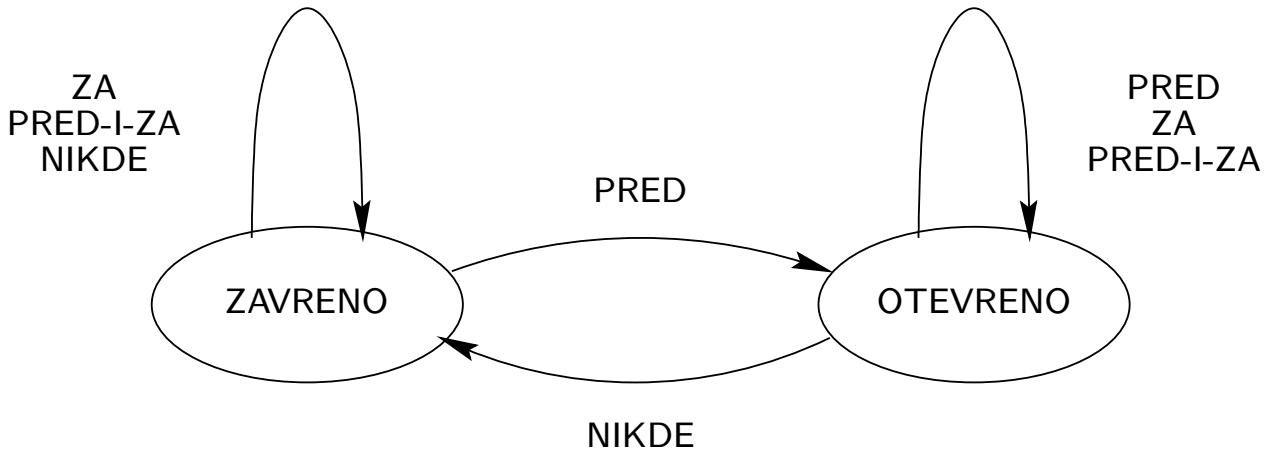
$A = (Q, \Sigma, \delta)$

Q ... (konečná) množina stavů

Σ ... (konečná) abeceda (množina *písmen*, *akcí*, *symbolů*, ...)

$\delta : Q \times \Sigma \to Q$ ... přechodová funkce

Konečný automat (jako model systému)



	PRED	ZA	PR-I-ZA	NIKDE
ZAV	OTEV	ZAV	ZAV	ZAV
OTEV	OTEV	OTEV	OTEV	ZAV

$A = (Q, \Sigma, \delta)$

Q ... (konečná) množina stavů

Σ ... (konečná) abeceda (množina *písmen, akcí, symbolů, ...*)

$\delta : Q \times \Sigma \to Q$ ... přechodová funkce

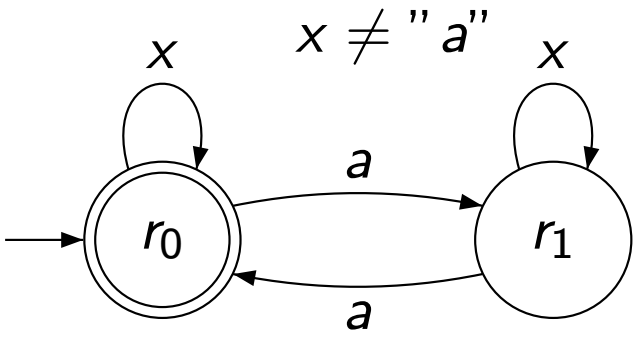
$Q = \{\text{ZAV, OTEV}\}$, $\Sigma = \{\text{PRED, ZA, PR-I-ZA, NIKDE}\}$,

$\delta = \{((\text{ZAV}, \text{PRED}), \text{OTEV}), \ldots\}$

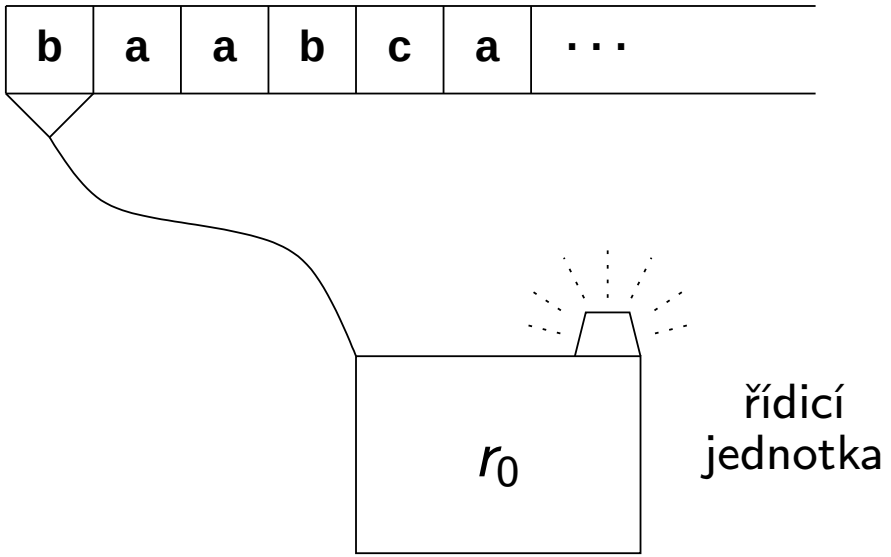
Co dělá tento program (algoritmus)?

```
#include <stdio.h>
int main(int argc, char **argv) {
    bool even_a = true;
    while(true) {
        int c = getchar();
        switch(c) {
            case 'a': even_a = !even_a; break;
            case EOF:
            case '\n':
                if (even_a) { printf("Yes\n"); }
                    else { printf("No\n"); }
                return 0;
        }
    }
}
```

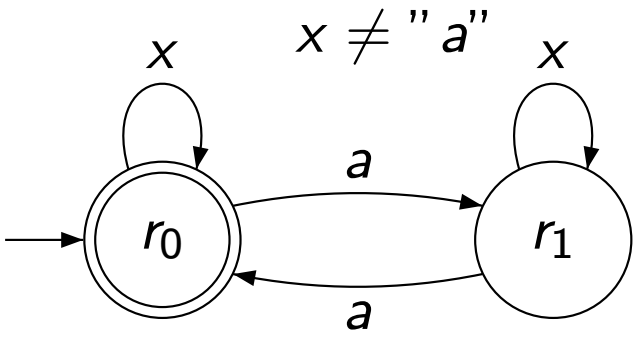
Podstata předchozího algoritmu ... jistý konečný automat



$r_0 = even$ $r_1 = odd$

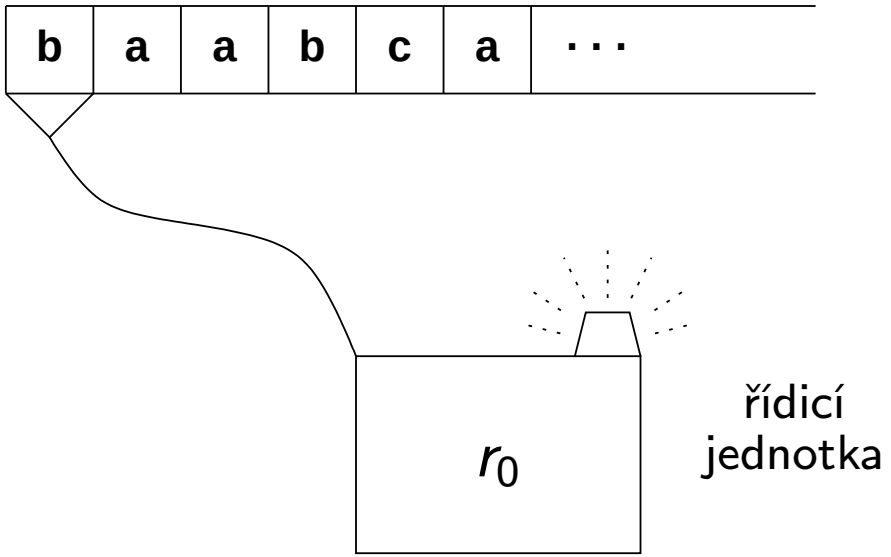


Podstata předchozího algoritmu ... jistý konečný automat

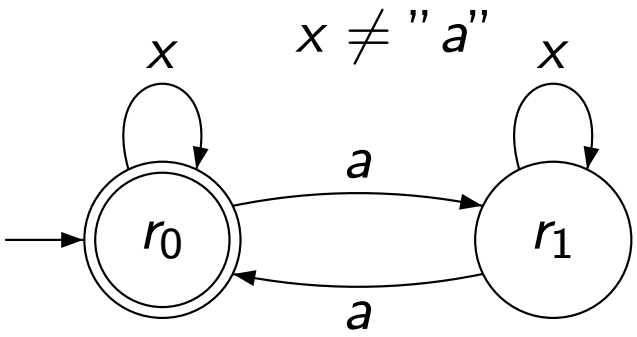


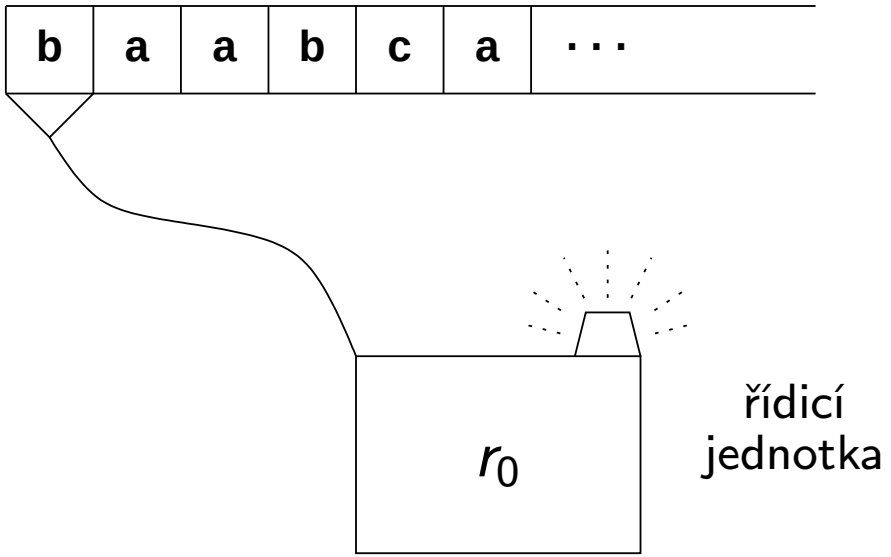
$r_0 = even$ $\qquad$ $r_1 = odd$

	A	a	b	c	$\ldots$
$\rightarrow$	(r_0)	r_1	r_0		
	r_1	r_0	r_1		



Podstata předchozího algoritmu ... jistý konečný automat



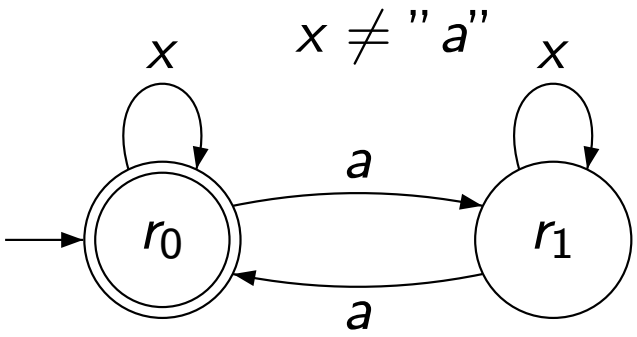


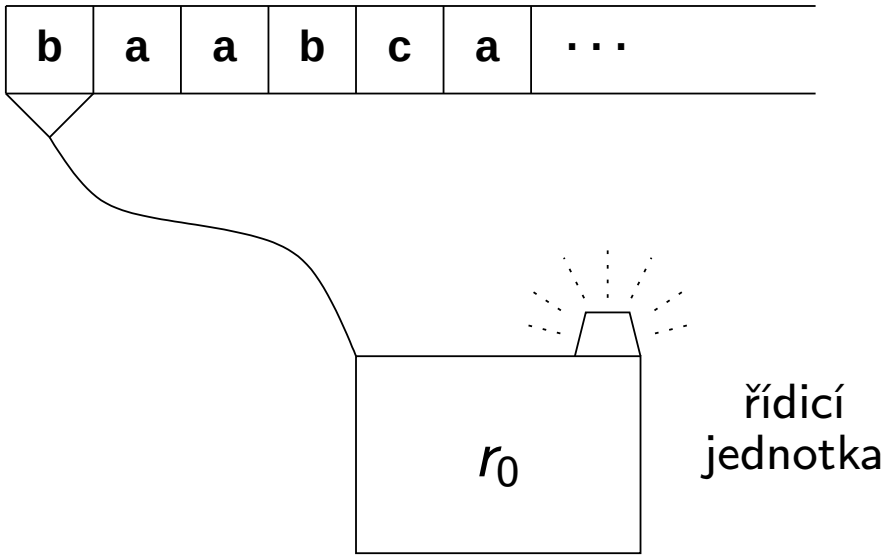
$r_0 = even$ $\quad$ $r_1 = odd$

	A	a	b	c	$\ldots$
$\rightarrow$	r_0	r_1	r_0		
	r_1	r_0	r_1		

$A = (Q, \Sigma, \delta, q_0, F) = (\{r_0, r_1\}, \{a, b, c, \ldots\}, \delta, r_0, \{r_0\})$, kde
$\delta = \{((r_0, a), r_1), ((r_0, b), r_0), \ldots\}$;
q_0 je počáteční stav a $F \subseteq Q$ je množina přijímajících stavů.

Podstata předchozího algoritmu ... jistý konečný automat





$r_0 = even$ $\quad$ $r_1 = odd$

	A	a	b	c	$\ldots$
$\rightarrow$	r_0	r_1	r_0		
	r_1	r_0	r_1		

$A = (Q, \Sigma, \delta, q_0, F) = (\{r_0, r_1\}, \{a, b, c, \ldots\}, \delta, r_0, \{r_0\})$, kde
$\delta = \{((r_0, a), r_1), ((r_0, b), r_0), \ldots\}$;
q_0 je počáteční stav a $F \subseteq Q$ je množina přijímajících stavů.

Tedy např. $\delta(r_0, a) = r_1$ ...

píšeme také $r_0 \xrightarrow{a}_A r_1$ či jen $r_0 \xrightarrow{a} r_1$ (když A je dán kontextem).

```
procedure XY (var F: file)
const maxstate = 1
type state = 0 .. maxstate
type alphabet = (a,b)
const A: array [ state , alphabet ] of state = [[1,0],[0,1]]
const AccSt: set of state = [0]
var q: state; ch: char
begin
  q:=0
  while true do
    read( ch, F )
    if EOF then (if q in AccSt then return 1 else return 0)
    q := A[ q, ch ]
  endwhile
end
```

```
procedure XY (var F: file)
const maxstate = 1
type state = 0 .. maxstate
type alphabet = (a,b)
const A: array [ state , alphabet ] of state = [[1,0],[0,1]]
const AccSt: set of state = [0]
var q: state; ch: char
begin
  q:=0
  while true do
    read( ch, F )
    if EOF then (if q in AccSt then return 1 else return 0)
    q := A[ q, ch ]
  endwhile
end
```

Procedura interpretuje (provádí, simuluje) předchozí automat.

```
procedure XY (var F: file)
const maxstate = 1
type state = 0 .. maxstate
type alphabet = (a,b)
const A: array [ state , alphabet ] of state = [[1,0],[0,1]]
const AccSt: set of state = [0]
var q: state; ch: char
begin
  q:=0
  while true do
    read( ch, F )
    if EOF then (if q in AccSt then return 1 else return 0)
    q := A[ q, ch ]
  endwhile
end
```

Procedura interpretuje (provádí, simuluje) předchozí automat.
Snadno lze modifikovat tak, že i automat je vstupním parametrem.

Příklad (výpočetního) problému

Příklad (výpočetního) problému

Vstup (neboli *instance* problému):
abeceda Σ, řetězec $t \in \Sigma^*$, vzorek $p \in \Sigma^*$.
(Např.: t je obsah 100 GB databáze s genetickou informací,
p je jistý vzorek (řetězec) délky 100.)

Příklad (výpočetního) problému

Vstup (neboli *instance* problému):
abeceda Σ, řetězec $t \in \Sigma^*$, vzorek $p \in \Sigma^*$.
(Např.: t je obsah 100 GB databáze s genetickou informací,
p je jistý vzorek (řetězec) délky 100.)

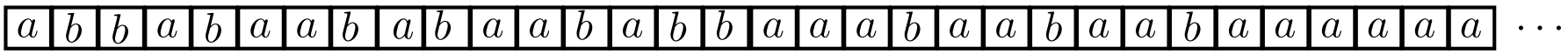
Výstup: (vhodná) reprezentace pozic výskytů vzorku p v řetězci t.
(Výskyty vzorku se mohou překrývat.)

Příklad (výpočetního) problému

Vstup (neboli *instance* problému):
abeceda Σ, řetězec $t \in \Sigma^*$, vzorek $p \in \Sigma^*$.
(Např.: t je obsah 100 GB databáze s genetickou informací,
p je jistý vzorek (řetězec) délky 100.)

Výstup: (vhodná) reprezentace pozic výskytů vzorku p v řetězci t.
(Výskyty vzorku se mohou překrývat.)

Pro jednoduchost se zaměříme na (pod)problém, kde máme fixně
$\Sigma = \{a, b\}$ a $p = abaaba$. Příklad vstupu:

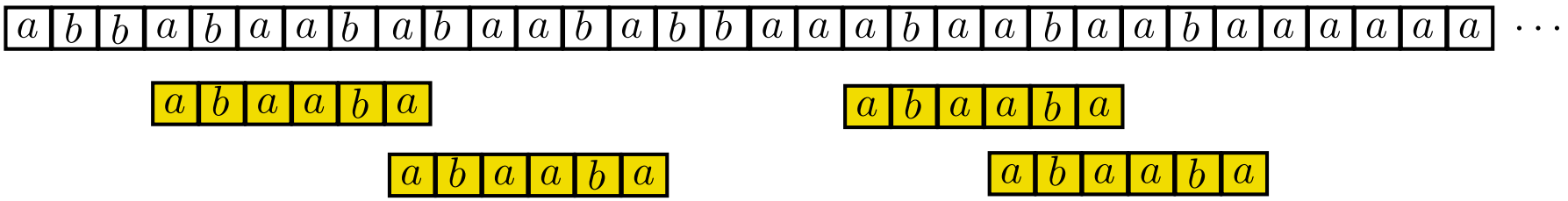


Příklad (výpočetního) problému

Vstup (neboli *instance* problému):
abeceda Σ, řetězec $t \in \Sigma^*$, vzorek $p \in \Sigma^*$.
(Např.: t je obsah 100 GB databáze s genetickou informací,
p je jistý vzorek (řetězec) délky 100.)

Výstup: (vhodná) reprezentace pozic výskytů vzorku p v řetězci t.
(Výskyty vzorku se mohou překrývat.)

Pro jednoduchost se zaměříme na (pod)problém, kde máme fixně
$\Sigma = \{a, b\}$ a $p = abaaba$. Příklad:

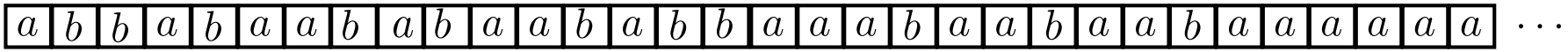


Přímočarý algoritmus (ještě jednou v C-čku ...)

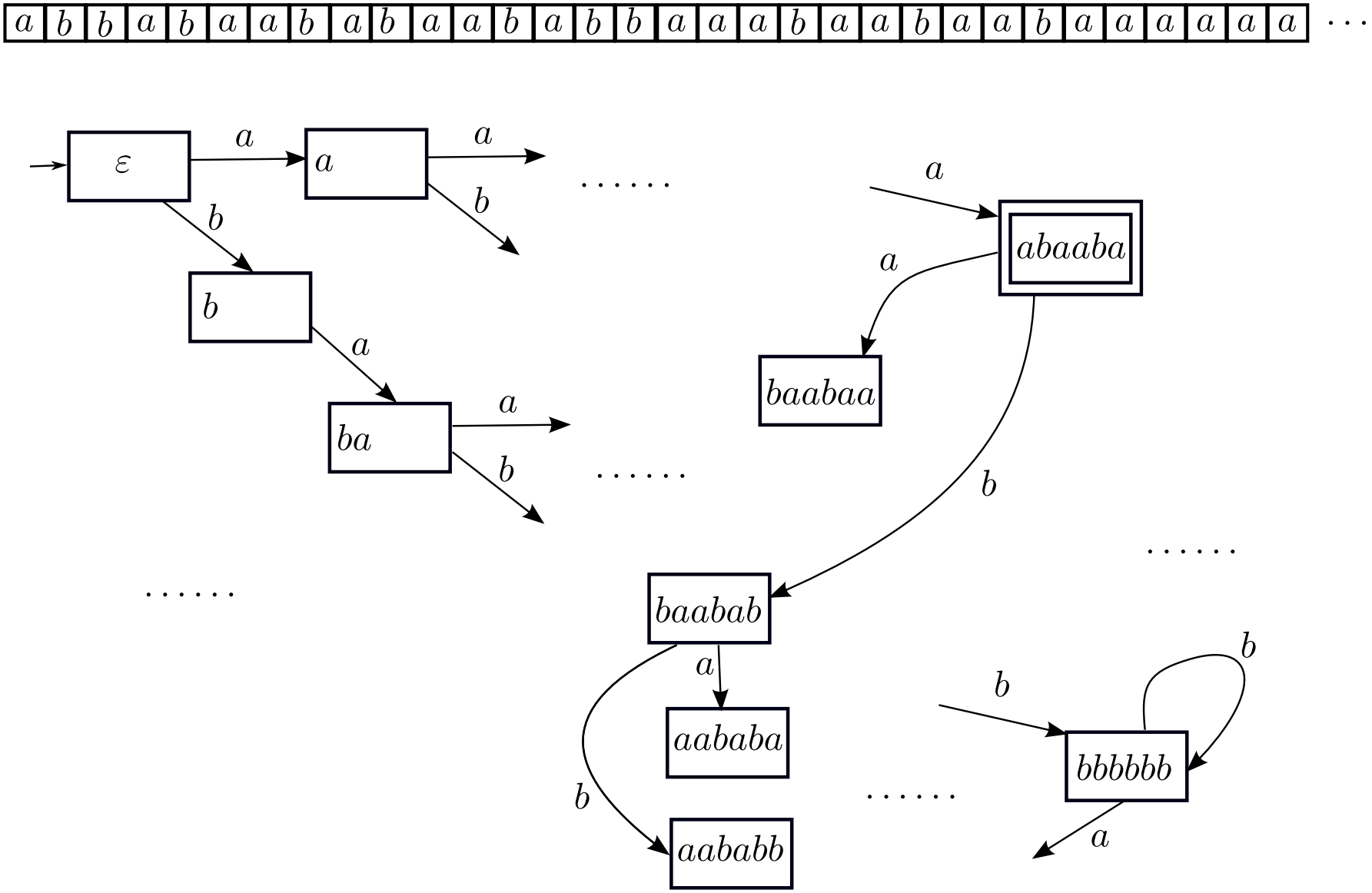
```
int main(int argc, char **argv)
{
  char tail[6] = { 0,0,0,0,0,0 };
  while(true) { // Infinite loop
    int c = getchar();
    if (c == '\n') { return 0; }
    tail[0] = tail[1]; tail[1] = tail[2];
    tail[2] = tail[3]; tail[3] = tail[4];
    tail[4] = tail[5]; tail[5] = c;

    if (tail[0] == 'a' && tail[1] == 'b' &&
        tail[2] == 'a' && tail[3] == 'a' &&
        tail[4] == 'b' && tail[5] == 'a')
    { zpracuj ...;
    }
  }
}
```

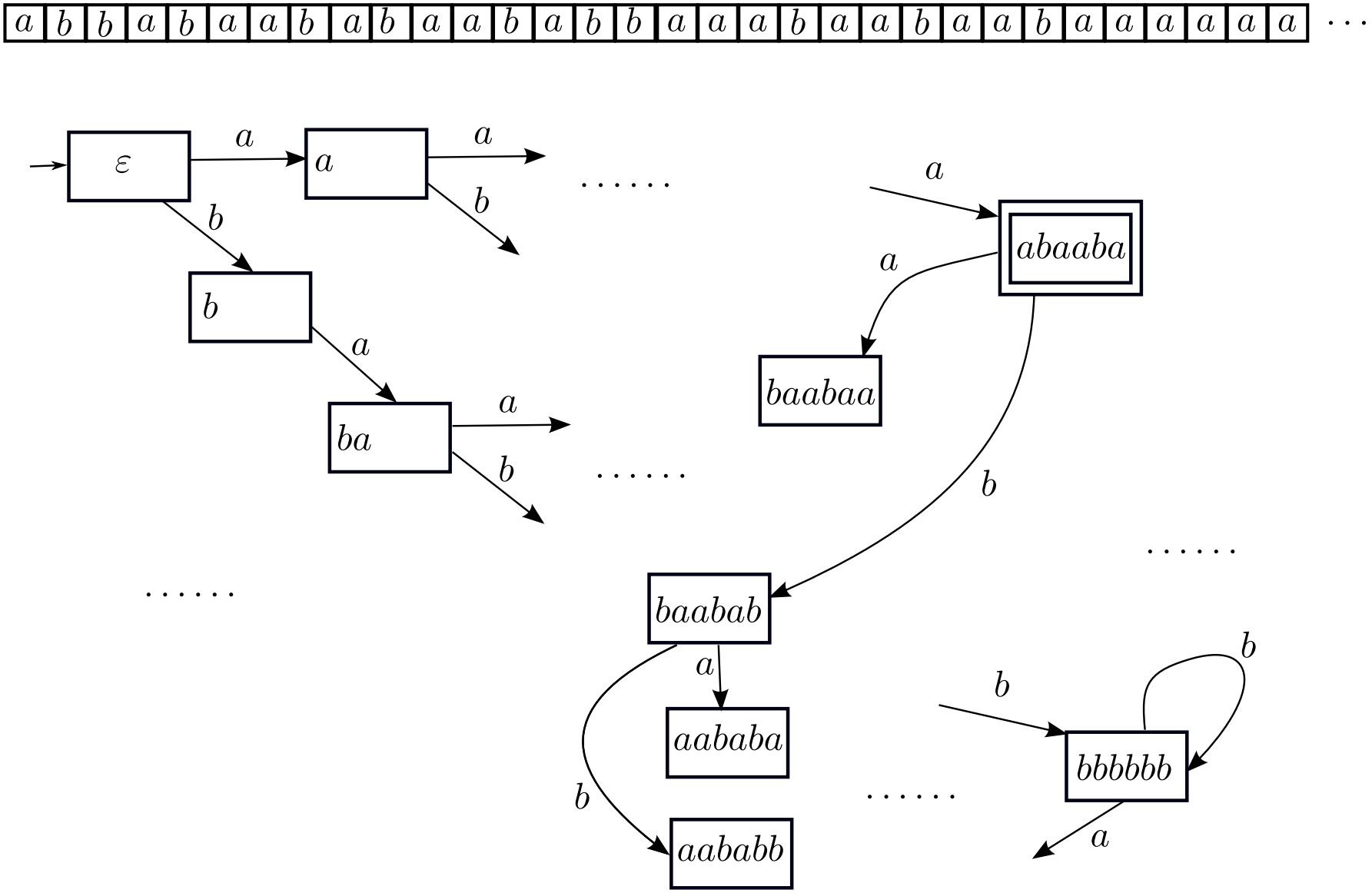
Je podstatou předchozího algoritmu konečný automat?



Je podstatou předchozího algoritmu konečný automat?



Je podstatou předchozího algoritmu konečný automat?



Dá se říci ano, ale je “trochu” velký . . . co když délka vzorku je např. 100 ?

Složitost (uvedeného) algoritmu

Algoritmus $\mathcal{A}$ má (časovou) složitost $T_{\mathcal{A}}$ ... co to je?

Složitost (uvedeného) algoritmu

Algoritmus $\mathcal{A}$ má (časovou) složitost $T_{\mathcal{A}}$ ... co to je?
$T_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ ... co znamená např. $T_{\mathcal{A}}(35) = 2800$?

Složitost (uvedeného) algoritmu

Algoritmus $\mathcal{A}$ má (časovou) složitost $T_{\mathcal{A}}$ ... co to je?
$T_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ ... co znamená např. $T_{\mathcal{A}}(35) = 2800$?

horní odhady
$T_{\mathcal{A}} \in O(f)$ (či $T_{\mathcal{A}}(n) \in O(f(n))$, někdy se píše i $T_{\mathcal{A}} = O(f)$ apod.)
Např. $T_{\mathcal{A}}(n) \in O(n^2)$ ...

Složitost (uvedeného) algoritmu

Algoritmus $\mathcal{A}$ má (časovou) složitost $T_{\mathcal{A}}$ ... co to je?
$T_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ ... co znamená např. $T_{\mathcal{A}}(35) = 2800$?

horní odhady
$T_{\mathcal{A}} \in O(f)$ (či $T_{\mathcal{A}}(n) \in O(f(n))$, někdy se píše i $T_{\mathcal{A}} = O(f)$ apod.)
Např. $T_{\mathcal{A}}(n) \in O(n^2)$ ...

$f(n) \in O(g(n)) \Leftrightarrow_{\mathsf{df}} \forall n : f(n) \leq g(n)$ ... nebo je to jinak? Ano, jinak!

Složitost (uvedeného) algoritmu

Algoritmus $\mathcal{A}$ má (časovou) složitost $T_{\mathcal{A}}$ ... co to je?
$T_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ ... co znamená např. $T_{\mathcal{A}}(35) = 2800$?

horní odhady
$T_{\mathcal{A}} \in O(f)$ (či $T_{\mathcal{A}}(n) \in O(f(n))$, někdy se píše i $T_{\mathcal{A}} = O(f)$ apod.)
Např. $T_{\mathcal{A}}(n) \in O(n^2)$ ...

$f(n) \in O(g(n)) \Leftrightarrow_{\text{df}} \forall n : f(n) \leq g(n)$ ... nebo je to jinak? Ano, jinak!

Někdy je vhodné použít více parametrů, jako např. u našeho algoritmu:

vstup: vzorek p (délky m), "text" t (délky n);
vezmi prázdný buffer délky m, nastav čtení t na začátek;
opakuj dokud nepřečteš celé t:

přesuň symbol z t do bufferu (a posuň čtecí hlavu);
když je v bufferu p, ohlaš jako další výskyt vzorku

Složitost (uvedeného) algoritmu

Algoritmus $\mathcal{A}$ má (časovou) složitost $T_{\mathcal{A}}$ ... co to je?
$T_{\mathcal{A}} : \mathbb{N} \to \mathbb{N}$ ... co znamená např. $T_{\mathcal{A}}(35) = 2800$?

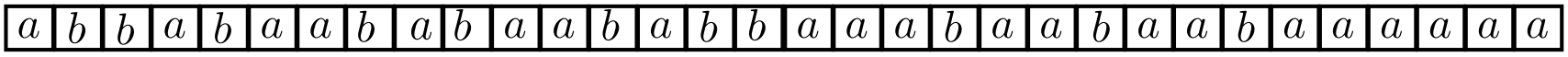
horní odhady
$T_{\mathcal{A}} \in O(f)$ (či $T_{\mathcal{A}}(n) \in O(f(n))$, někdy se píše i $T_{\mathcal{A}} = O(f)$ apod.)
Např. $T_{\mathcal{A}}(n) \in O(n^2)$ ...

$f(n) \in O(g(n)) \Leftrightarrow_{\mathsf{df}} \forall n : f(n) \leq g(n)$ ... nebo je to jinak? Ano, jinak!

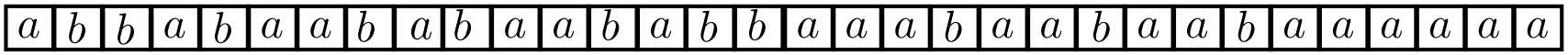
Někdy je vhodné použít více parametrů, jako např. u našeho algoritmu:

> *vstup: vzorek p (délky m), "text" t (délky n);*
> *vezmi prázdný buffer délky m, nastav čtení t na začátek;*
> *opakuj dokud nepřečteš celé t:*
>
> > *přesuň symbol z t do bufferu (a posuň čtecí hlavu);*
> > *když je v bufferu p, ohlaš jako další výskyt vzorku*

Složitost algoritmu je v $O(mn)$ (tělo vnějšího cyklus se provede n-krát, v každém běhu se provede "skrytý" vnitřní cyklus s max. m "běhy" ...).

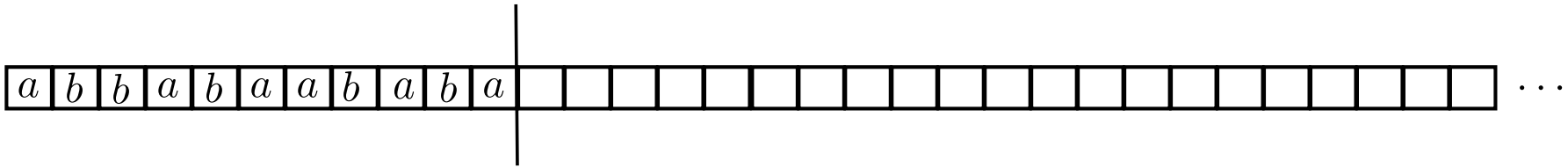


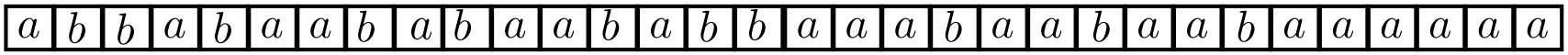
Pro velká m, n (např. $m = 100$, $n = 10^{11}$) je jistě vhodné se pokusit o lepší algoritmus, než je ten se složitostí $O(mn)$ (přesněji $\Theta(mn)$).



Pro velká m, n (např. $m = 100$, $n = 10^{11}$) je jistě vhodné se pokusit o lepší algoritmus, než je ten se složitostí $O(mn)$ (přesněji $\Theta(mn)$).

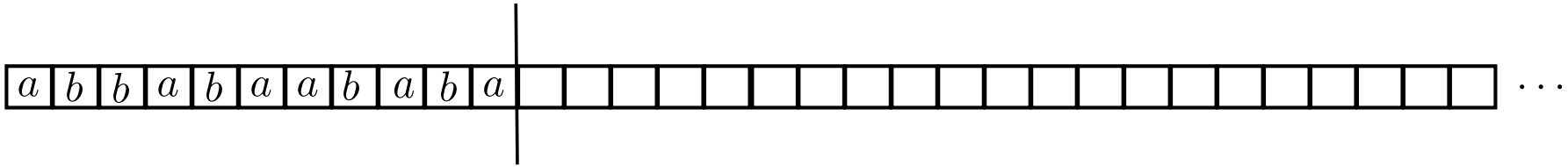
Klíčová idea (otázka): jakou informaci si stačí pamatovat z (dosud) přečteného úseku?



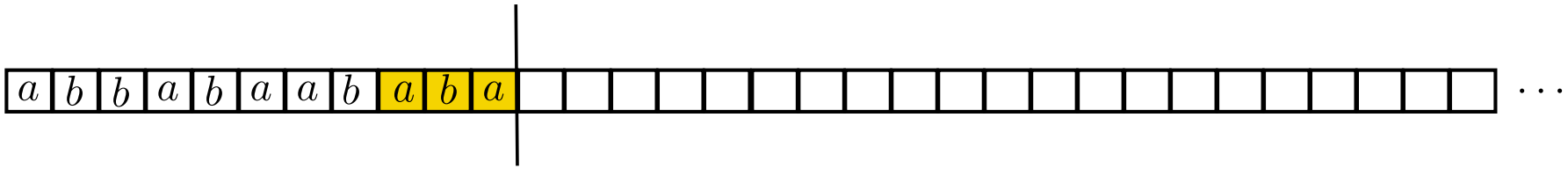


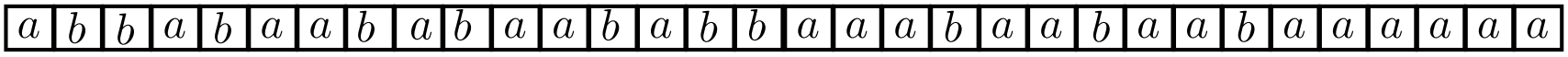
Pro velká m, n (např. $m = 100$, $n = 10^{11}$) je jistě vhodné se pokusit o lepší algoritmus, než je ten se složitostí $O(mn)$ (přesněji $\Theta(mn)$).

Klíčová idea (otázka): jakou informaci si stačí pamatovat z (dosud) přečteného úseku?



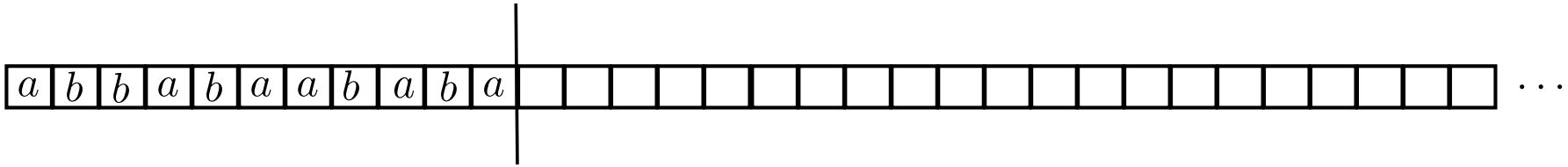
Lze nahlédnout, že si stačí pamatovat nejdelší sufix dosud přečteného, který je prefixem vzorku, tedy de facto číslo v rozmezí $0 \ldots m$.
V našem konkrétním příkladu (kde $p = abaaba$):



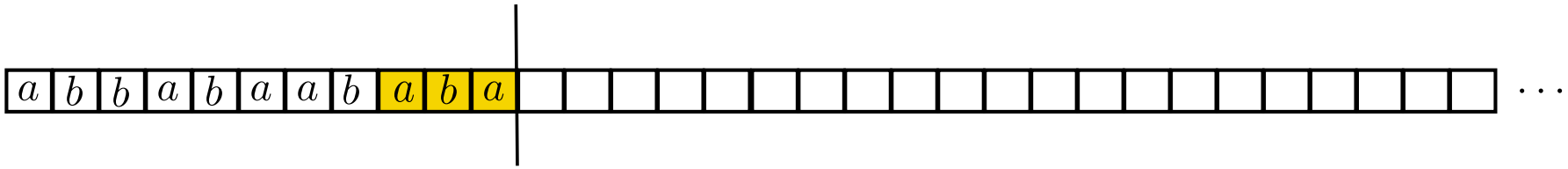


Pro velká m, n (např. $m = 100$, $n = 10^{11}$) je jistě vhodné se pokusit o lepší algoritmus, než je ten se složitostí $O(mn)$ (přesněji $\Theta(mn)$).

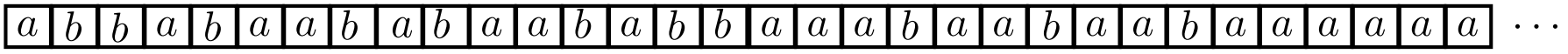
Klíčová idea (otázka): jakou informaci si stačí pamatovat z (dosud) přečteného úseku?



Lze nahlédnout, že si stačí pamatovat nejdelší sufix dosud přečteného, který je prefixem vzorku, tedy de facto číslo v rozmezí $0 \ldots m$.
V našem konkrétním příkladu (kde $p = abaaba$):

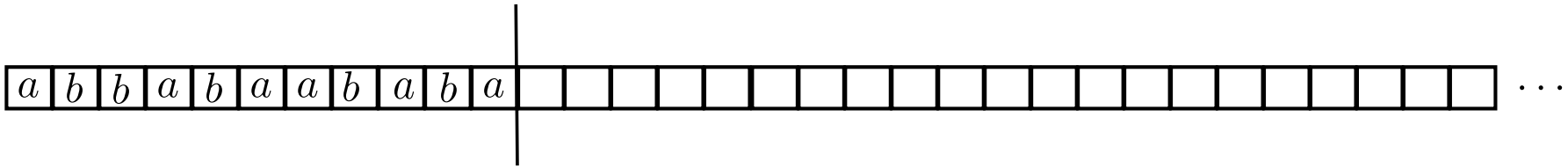


Teď se ukazuje, že algoritmus lze založit na kon. automatu s $m+1$ stavy.
Ale jak, a jak rychle, automat k danému vzorku sestrojíme?

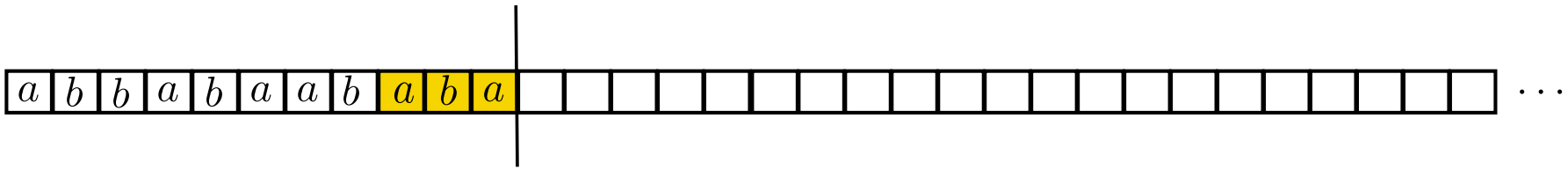


Pro velká m, n (např. $m = 100$, $n = 10^{11}$) je jistě vhodné se pokusit o lepší algoritmus, než je ten se složitostí $O(mn)$ (přesněji $\Theta(mn)$).

Klíčová idea (otázka): jakou informaci si stačí pamatovat z (dosud) přečteného úseku?



Lze nahlédnout, že si stačí pamatovat nejdelší sufix dosud přečteného, který je prefixem vzorku, tedy de facto číslo v rozmezí $0 \ldots m$.
V našem konkrétním příkladu (kde $p = abaaba$):



Teď se ukazuje, že algoritmus lze založit na kon. automatu s $m+1$ stavy.
Ale jak, a jak rychle, automat k danému vzorku sestrojíme?
Půjde to v čase $O(m)$, takže celkově složitost srazíme na $O(m+n)$ ...

Konstrukce automatu (Knuth, Morris, Pratt)

K danému vzorku p : array$[1..m]$ of char (pro $m \geq 1$) konstruujeme dvourozměrné pole *Next* (což je onen automat):

$Next[0, p[1]] := 1;\ \forall x \in \Sigma \setminus \{p[1]\} : Next[0, x] := 0;$
$Sec[1] := 0;$
for $i := 1$ *to* $m-1$ *do*
$\quad Next[i, p[i+1]] := i+1;$
$\quad \forall x \in \Sigma \setminus \{p[i+1]\} : Next[i, x] := Next[Sec[i], x];$
$\quad Sec[i+1] := Next[Sec[i], p[i+1]];$

$\forall x \in \Sigma : Next[m, x] := Next[Sec[m]; x];$

Sec je pomocné jednorozměrné pole; $Sec[i]$ je "sekundární možnost", tedy délka nejdelšího vlastního sufixu řetězce $p[1..i]$, který je rovněž prefixem vzorku p.

Konstrukce automatu (Knuth, Morris, Pratt)

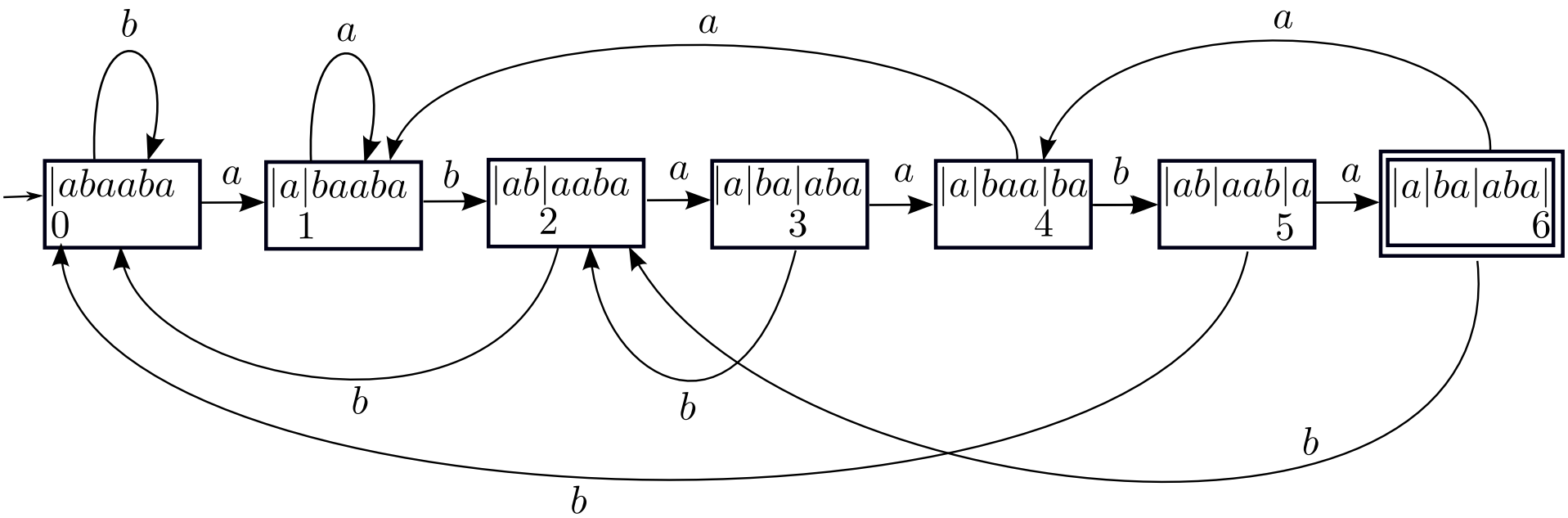
K danému vzorku p : array[1..m] of char (pro $m \geq 1$) konstruujeme dvourozměrné pole *Next* (což je onen automat):

$Next[0, p[1]] := 1;\ \forall x \in \Sigma \setminus \{p[1]\} : Next[0, x] := 0;$
$Sec[1] := 0;$
for $i := 1$ *to* $m-1$ *do*
$\quad Next[i, p[i+1]] := i+1;$
$\quad \forall x \in \Sigma \setminus \{p[i+1]\} : Next[i, x] := Next[Sec[i], x];$
$\quad Sec[i+1] := Next[Sec[i], p[i+1]];$

$\forall x \in \Sigma : Next[m, x] := Next[Sec[m]; x];$

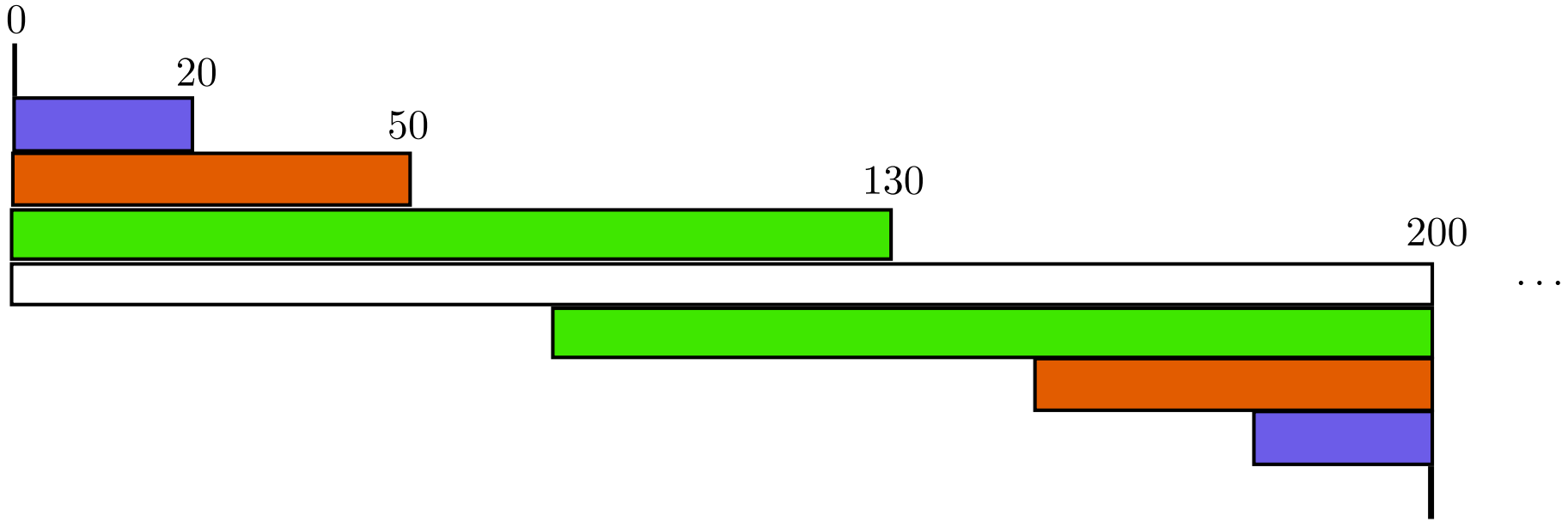
Sec je pomocné jednorozměrné pole; $Sec[i]$ je "sekundární možnost", tedy délka nejdelšího vlastního sufixu řetězce $p[1..i]$, který je rovněž prefixem vzorku p.
Pozn.: zmíněná složitost $O(m)$ zde platí pro fixní abecedu Σ; v obecném případě se vyhneme explicitní konstrukci přechodů pro všechna $x \in \Sigma$.

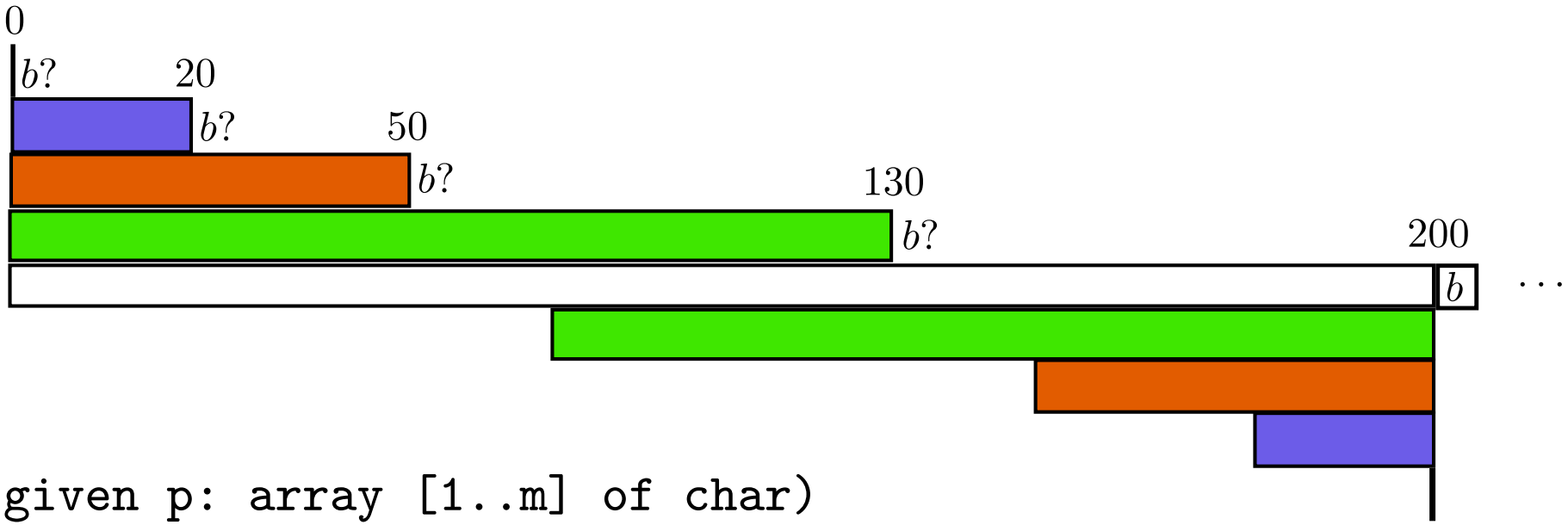
Sestrojený automat



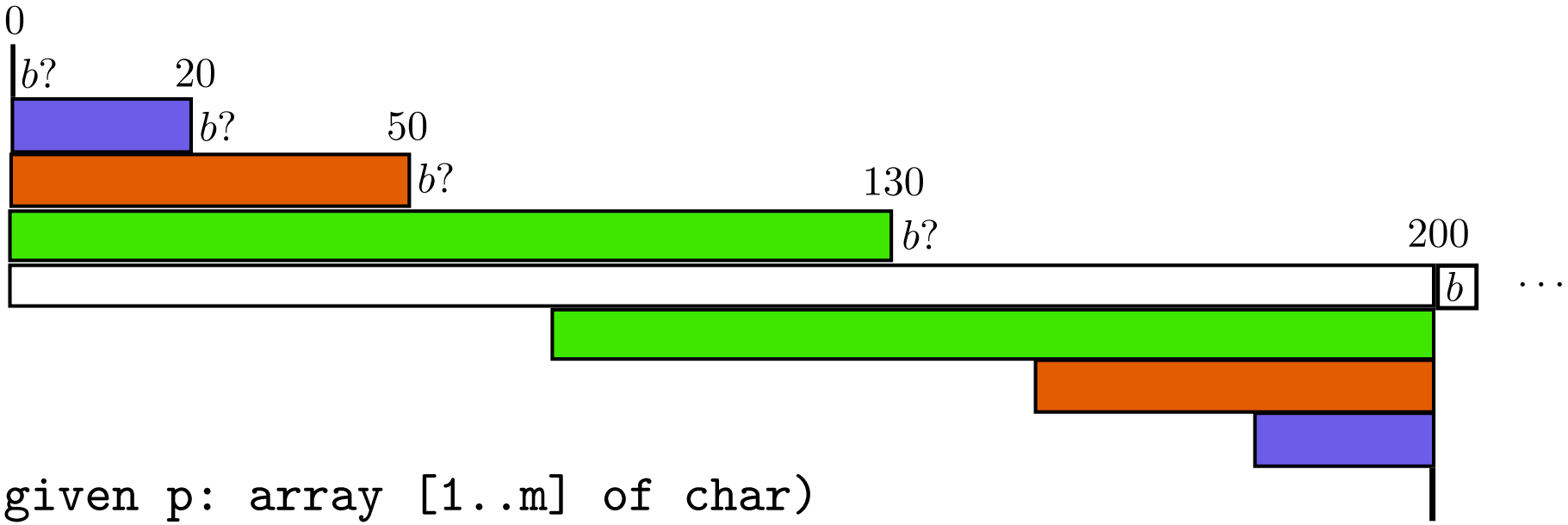
K poznámce o složitosti $O(m)$ konstrukce automatu

Klíčem je funkce *Sec*, kde
Sec(i) je délka nejdelšího vlastního sufixu $p(1)p(2)\ldots p(i)$
(tedy prefixu vzorku délky i),
který je rovněž prefixem vzorku.
(V našem příkladu bylo např. $Sec(5) = 2$, $Sec(2) = 0$, atd.)
Na obrázku níže je znázorněna jiná situace, kde $Sec(200) = 130$,
$Sec(130) = 50$, $Sec(50) = 20$, $Sec(20) = 0$.






```
(given p: array [1..m] of char)
Sec[0]:=-1; Sec[1]:=0;
for i:=2 to m do
  x:=p[i]; j:=Sec[i-1]; end:=false;
  while (j>=0 and (not end)) do
    if x = p[j+1] then (Sec[i]:=j+1; end:=true)
                  else j:=Sec[j] ;
  if j<0 then Sec[i]:=0
```




```
(given p: array [1..m] of char)
Sec[0]:=-1; Sec[1]:=0;
for i:=2 to m do
  x:=p[i]; j:=Sec[i-1]; end:=false;
  while (j>=0 and (not end)) do
    if x = p[j+1] then (Sec[i]:=j+1; end:=true)
                  else j:=Sec[j] ;
  if j<0 then Sec[i]:=0
```

Hrubý odhad dá $O(m^2)$ (jeden cyklus vnořen do druhého), ale detailnější analýza (amortizované složitosti) ukáže $O(m)$ (rozdíl $i - Sec(i)$ nemůže klesnout, tedy celkový počet operací vnitřního cyklu je $O(m)$).

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Induktivní definice

1. $q \xrightarrow{\varepsilon} q$ (axiom),
2. $(\delta(q, a) = q' \wedge q' \xrightarrow{u} q'') \Longrightarrow q \xrightarrow{au} q''$ (odvozovací pravidlo).

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Induktivní definice

1. $q \xrightarrow{\varepsilon} q$ (axiom),
2. $(\delta(q, a) = q' \wedge q' \xrightarrow{u} q'') \Longrightarrow q \xrightarrow{au} q''$ (odvozovací pravidlo).

Zkratka: pro $R \subseteq Q$, píšeme stručněji $q \xrightarrow{u} R$ místo

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Induktivní definice

1. $q \xrightarrow{\varepsilon} q$ (axiom),
2. $(\delta(q, a) = q' \wedge q' \xrightarrow{u} q'') \Longrightarrow q \xrightarrow{au} q''$ (odvozovací pravidlo).

Zkratka: pro $R \subseteq Q$, píšeme stručněji $q \xrightarrow{u} R$ místo $\exists r \in R : q \xrightarrow{u} r$.

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Induktivní definice

1. $q \xrightarrow{\varepsilon} q$ (axiom),
2. $(\delta(q, a) = q' \wedge q' \xrightarrow{u} q'') \Longrightarrow q \xrightarrow{au} q''$ (odvozovací pravidlo).

Zkratka: pro $R \subseteq Q$, píšeme stručněji $q \xrightarrow{u} R$ místo $\exists r \in R : q \xrightarrow{u} r$.

Slovo $w \in \Sigma^*$ je *přijímáno* automatem $A \quad \Leftrightarrow_{\text{df}} \quad q_0 \xrightarrow{w} F$.

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Induktivní definice

1. $q \xrightarrow{\varepsilon} q$ (axiom),
2. $(\delta(q, a) = q' \wedge q' \xrightarrow{u} q'') \Longrightarrow q \xrightarrow{au} q''$ (odvozovací pravidlo).

Zkratka: pro $R \subseteq Q$, píšeme stručněji $q \xrightarrow{u} R$ místo $\exists r \in R : q \xrightarrow{u} r$.

Slovo $w \in \Sigma^*$ je *přijímáno* automatem $A \quad \Leftrightarrow_{\text{df}} \quad q_0 \xrightarrow{w} F$.
Jazykem rozpoznávaným (přijímaným) automatem A je jazyk
$L(A) = \{w \in \Sigma^* \mid \text{ slovo } w \text{ je přijímáno } A\} = \{w \mid q_0 \xrightarrow{w} F\}$.

Slova a jazyky přijímané konečnými automaty

Mějme zadán KA $A = (Q, \Sigma, \delta, q_0, F)$.
Zavádíme značení

$$q \xrightarrow{w} q'$$

pro ternární relaci, podmnožinu $Q \times \Sigma^* \times Q$:

Induktivní definice

1. $q \xrightarrow{\varepsilon} q$ (axiom),
2. $(\delta(q, a) = q' \wedge q' \xrightarrow{u} q'') \Longrightarrow q \xrightarrow{au} q''$ (odvozovací pravidlo).

Zkratka: pro $R \subseteq Q$, píšeme stručněji $q \xrightarrow{u} R$ místo $\exists r \in R : q \xrightarrow{u} r$.

Slovo $w \in \Sigma^*$ je *přijímáno* automatem $A \quad \Leftrightarrow_{\text{df}} \quad q_0 \xrightarrow{w} F$.
Jazykem rozpoznávaným (přijímaným) automatem A je jazyk
$L(A) = \{w \in \Sigma^* \mid \text{ slovo } w \text{ je přijímáno } A\} = \{w \mid q_0 \xrightarrow{w} F\}$.

Jazyk L je *regulární* $\quad \Leftrightarrow_{df} \quad$ existuje KA A tž. $L(A) = L$.

Příklad jazyka v abecedě $\{0, 1\}$

$L = \{\, w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi a
w neobsahuje podřetězec 101 $\}$

kde $bn(w)$ znamená číslo, pro něž je w jeho binárním zápisem (např. $bn(0101) = 5$); klademe $bn(\varepsilon) = 0$.

Příklad jazyka v abecedě $\{0, 1\}$

$L = \{\, w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi a
w neobsahuje podřetězec 101 $\}$

kde $bn(w)$ znamená číslo, pro něž je w jeho binárním zápisem (např. $bn(0101) = 5$); klademe $bn(\varepsilon) = 0$.

Jedná se o regulární jazyk?

Příklad jazyka v abecedě $\{0, 1\}$

$L = \{\, w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi a
w neobsahuje podřetězec 101 $\}$

kde $bn(w)$ znamená číslo, pro něž je w jeho binárním zápisem (např. $bn(0101) = 5$); klademe $bn(\varepsilon) = 0$.

Jedná se o regulární jazyk?

Všimněme si, že

$$L = L_1 \cap \overline{L_2}$$

kde
$L_1 = \{\, w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi $\}$
$L_2 = \{w \in \{0,1\}^* \mid w$ obsahuje podřetězec 101$\}$

Příklad jazyka v abecedě $\{0, 1\}$

$L = \{ w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi a
w neobsahuje podřetězec 101 $\}$

kde $bn(w)$ znamená číslo, pro něž je w jeho binárním zápisem (např. $bn(0101) = 5$); klademe $bn(\varepsilon) = 0$.

Jedná se o regulární jazyk?

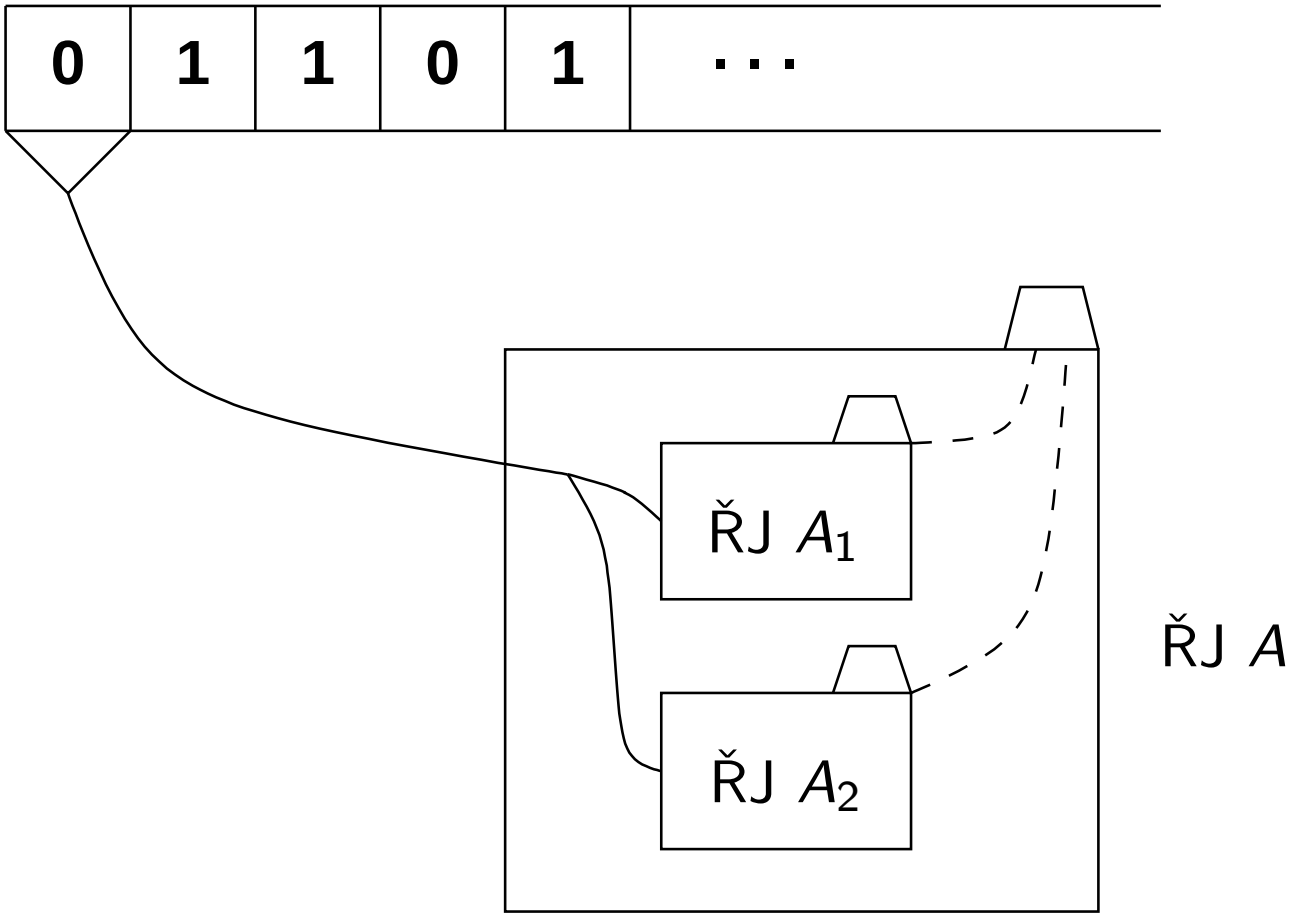
Všimněme si, že

$$L = L_1 \cap \overline{L_2}$$

kde
$L_1 = \{ w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi $\}$
$L_2 = \{w \in \{0,1\}^* \mid w$ obsahuje podřetězec 101$\}$

Nabízí se modulární přístup.

KA A pro jazyk $L(A_1) \cap L(A_2)$ (nebo $L(A_1) \cup L(A_2)$)



Automat A, pro nějž platí $L(A) = L$

$L = \{ w \in \{0,1\}^* \mid bn(w)$ je dělitelné třemi a
w neobsahuje podřetězec 101$\}$

A_1	0	1
$\leftrightarrow q_1$	q_1	q_2
q_2	q_3	q_1
q_3	q_2	q_3

A_2	0	1
$\leftrightarrow r_1$	r_1	r_2
$\leftarrow r_2$	r_3	r_2
$\leftarrow r_3$	r_1	r_4
r_4	r_4	r_4

A	0	1
$\leftrightarrow(q_1, r_1)$	(q_1, r_1)	(q_2, r_2)
$\leftarrow(q_1, r_2)$	(q_1, r_3)	(q_2, r_2)
$\leftarrow(q_1, r_3)$	(q_1, r_1)	(q_2, r_4)
(q_1, r_4)	(q_1, r_4)	(q_2, r_4)
(q_2, r_1)	(q_3, r_1)	(q_1, r_2)
(q_2, r_2)	(q_3, r_3)	(q_1, r_2)
(q_2, r_3)	(q_3, r_1)	(q_1, r_4)
(q_2, r_4)	(q_3, r_4)	(q_1, r_4)
(q_3, r_1)	(q_2, r_1)	(q_3, r_2)
(q_3, r_2)	(q_2, r_3)	(q_3, r_2)
(q_3, r_3)	(q_2, r_1)	(q_3, r_4)
(q_3, r_4)	(q_2, r_4)	(q_3, r_4)

Izomorfní úprava automatu

A	0	1
$\leftrightarrow(q_1, r_1)$	(q_1, r_1)	(q_2, r_2)
$\leftarrow(q_1, r_2)$	(q_1, r_3)	(q_2, r_2)
$\leftarrow(q_1, r_3)$	(q_1, r_1)	(q_2, r_4)
(q_1, r_4)	(q_1, r_4)	(q_2, r_4)
(q_2, r_1)	(q_3, r_1)	(q_1, r_2)
(q_2, r_2)	(q_3, r_3)	(q_1, r_2)
(q_2, r_3)	(q_3, r_1)	(q_1, r_4)
(q_2, r_4)	(q_3, r_4)	(q_1, r_4)
(q_3, r_1)	(q_2, r_1)	(q_3, r_2)
(q_3, r_2)	(q_2, r_3)	(q_3, r_2)
(q_3, r_3)	(q_2, r_1)	(q_3, r_4)
(q_3, r_4)	(q_2, r_4)	(q_3, r_4)

A'	0	1	
$\leftrightarrow s_1$	s_1	s_2	$s_1 = (q_1, r_1)$
s_2	s_3	s_4	$s_2 = (q_2, r_2)$
s_3	s_5	s_6	$s_3 = (q_3, r_3)$
$\leftarrow s_4$	s_7	s_2	$s_4 = (q_1, r_2)$
s_5	s_8	s_4	$s_5 = (q_2, r_1)$
s_6	s_9	s_6	$s_6 = (q_3, r_4)$
$\leftarrow s_7$	s_1	s_9	$s_7 = (q_1, r_3)$
s_8	s_5	s_{10}	$s_8 = (q_3, r_1)$
s_9	s_6	s_{11}	$s_9 = (q_2, r_4)$
s_{10}	s_{12}	s_{10}	$s_{10} = (q_3, r_2)$
s_{11}	s_{11}	s_9	$s_{11} = (q_1, r_4)$
s_{12}	s_8	s_{11}	$s_{12} = (q_2, r_3)$

Charakterizace regulárních jazyků

Levý kvocient jazyka L podle slova u: $\quad u\backslash L = \{v \mid uv \in L\}$.

Věta. Jazyk $L \subseteq \Sigma^*$ je regulární právě tehdy, když je množina kvocientů $\{w\backslash L \mid w \in \Sigma^*\}$ konečná.

(... čili jazyk je regulární právě tehdy, když si při čtení slova [zleva doprava] stačí [pro finální rozhodnutí o přijetí/nepřijetí] z dosud přečteného prefixu pamatovat omezenou informaci, omezenou nezávisle na délce slova ...)

Důkaz. ...

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w (“délka w v a-čkách”),

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w ("délka w v a-čkách"),

1. $\{w \in \{a, b\}^*; |w|_a \bmod 2 = 0\}$

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w ("délka w v a-čkách"),

1. $\{w \in \{a, b\}^*; |w|_a \bmod 2 = 0\}$
2. $\{w \in \{a, b\}^* \mid w$ začíná nebo končí dvojicí stejných písmen $\}$

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w ("délka w v a-čkách"),

1. $\{w \in \{a,b\}^*; |w|_a \bmod 2 = 0\}$
2. $\{w \in \{a,b\}^* \mid w$ začíná nebo končí dvojicí stejných písmen $\}$
3. $\{w \in \{a,b\}^*; |w|_a < |w|_b\}$

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w ("délka w v a-čkách"),

1. $\{w \in \{a,b\}^*; |w|_a \bmod 2 = 0\}$
2. $\{w \in \{a,b\}^* \mid w$ začíná nebo končí dvojicí stejných písmen $\}$
3. $\{w \in \{a,b\}^*; |w|_a < |w|_b\}$
4. $\{w \in \{a,b,c\}^* \mid$ jestliže w neobsahuje podřetězec abc, pak končí řetězcem $bca\}$

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w ("délka w v a-čkách"),

1. $\{w \in \{a,b\}^*; |w|_a \bmod 2 = 0\}$
2. $\{w \in \{a,b\}^* \mid w$ začíná nebo končí dvojicí stejných písmen $\}$
3. $\{w \in \{a,b\}^*; |w|_a < |w|_b\}$
4. $\{w \in \{a,b,c\}^* \mid$ jestliže w neobsahuje podřetězec abc, pak končí řetězcem $bca\}$
5. $\{w \in \{a,b\}^*; |w|_a > |w|_b$ nebo w končí $baa\}$

Schválně, poznáte, které jazyky jsou regulární?

pro $w \in \Sigma^*$,
$|w|$ označuje délku slova w,
$|w|_a$ označuje počet výskytů symbolu a ve w ("délka w v a-čkách"),

1. $\{w \in \{a,b\}^*; |w|_a \bmod 2 = 0\}$
2. $\{w \in \{a,b\}^* \mid w$ začíná nebo končí dvojicí stejných písmen $\}$
3. $\{w \in \{a,b\}^*; |w|_a < |w|_b\}$
4. $\{w \in \{a,b,c\}^* \mid$ jestliže w neobsahuje podřetězec abc, pak končí řetězcem $bca\}$
5. $\{w \in \{a,b\}^*; |w|_a > |w|_b$ nebo w končí $baa\}$
6. $\{w \in \{a,b\}^*; |w|_a > |w|_b$ nebo $|w|_b \geq 2\}$