Bodó Zalán

Reguláris kifejezések kiértékelése

"In theory, there is no difference between theory and practice. In practice, there is."
Jan L.A. van de Snepscheut

Reguláris kifejezések

- ▷ generatív grammatika, nyelvek, automaták – 1950 – 1960
- ▷ Kleene – Kleene algebra, reguláris halmazok
- ▷ reguláris nyelvek ≡ reguláris kifejezések (`regex` v. `regexp`)
- ▷ ⇒ `regexp` ↔ NVA
- ▷ problémák:
  1. (N)DVA → `regexp`
  2. `regexp` → NVA
  3. kiértékelés (!!!)

Kleene algebra

- $\mathcal{K} = (K, +, \cdot, *, 0, 1)$
- axiómák:
  1. $a + (b + c) = (a + b) + c$
  2. $a + b = b + a$
  3. $a + 0 = a$
  4. $a + a = a$
  5. $a(bc) = (ab)c$
  6. $1a = a$
  7. $a1 = a$
  8. $a(b + c) = ab + ac$
  9. $0a = 0$
  10. $a0 = 0$
  11. $\ldots$

- Reg_Σ (a Σ feletti reguláris halmazok családja) egy Kleene algebra
- `regexp` értelmezése:
  (1) ϵ egy `regexp`, $\mathcal{L}(\epsilon) = \epsilon$
  (2) $\forall\, c \in \Sigma :\ c$ `regexp`, $\mathcal{L}(c) = c$
  (3) Ha $E_1,\ E_2$ `regexp`, akkor E_1E_2 $(E_1 \cdot E_2)$ is `regexp`, $\mathcal{L}(E_1E_2) = \mathcal{L}(E_1)\mathcal{L}(E_2)$ (konkatenáció)
  (4) Ha $E_1,\ E_2$ `regexp`, akkor $E_1|E_2$ is `regexp`, $\mathcal{L}(E_1|E_2) = \mathcal{L}(E_1) \cup \mathcal{L}(E_2)$ (egyesítés)
  (5) Ha E_1 `regexp`, E_1* szintúgy `regexp`, $\mathcal{L}(E_1*) = \mathcal{L}(E_1)*$ (Kleene-lezárás)
  (6) Ha E_1 `regexp`, (E_1) szintúgy `regexp`,

$$\mathcal{L}((E_1)) = \mathcal{L}(E_1)$$

(7) $\forall$ `regexp` megépíthető az (1)–(6) szabályok véges számú alkalmazásával

▷ BNF:

```
RegExp ::= RegExp Regexp | RegExp|RegExp |
           RegExp* | (RegExp) | x
```

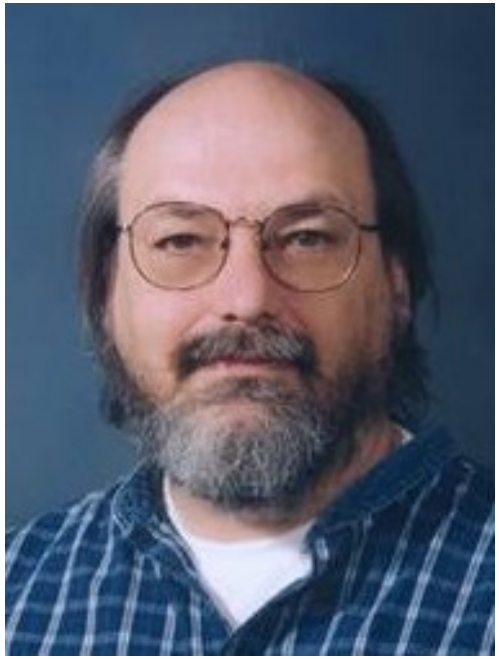
(`x` $\in \Sigma$)

1. (N)DVA $\longrightarrow$ regexp

$\surd$

2. `regexp` → (N)DVA

- Glushkov – 1960
- Thompson – 1968



Ken Thompson

Glushkov automata

- ▷ `regexp` átalakítása ($E \to E'$): helyettesítjük a (Σ-beli) szimbólumokat pozíciójukkal; így E' egy $\{1, \ldots, n\}$ feletti `regexp` lesz
- ▷ a következő halmazokat definiáljuk:
  - $first(E)$ = azon poz. halmaza, melyekkel $\mathcal{L}(E')$ szavai kezdődhetnek
  - $last(E)$ = azon poz. halmaza, melyekkel $\mathcal{L}(E')$ szavai végződhetnek
  - $follow(i, E)$ az i pozíció utáni lehetséges pozíciók halmaza $\mathcal{L}(E')$-ben

- az A automatát a következőképpen szerkesztjük meg:
  - $S = \{0, \ldots, n\}$
  - $Q_0 = \{0\}$
  - $F = \{last(E)\}$
  - $\delta(i, a) = j$, ha $(j \equiv a$ és $j \in follow(i, E))$ vagy $(i = 0$ és $j \in first(E))$

Példa: Legyen $E = (a|b) * (b|\epsilon)a$

$\Rightarrow E' = (1|2) * (3|\epsilon)4$, ahol $1 \equiv a,\ 2 \equiv b,\ 3 \equiv b,\ 4 \equiv a$

köv. old. $\rightarrow$

$first(E) = \{1, 2, 3, 4\}$

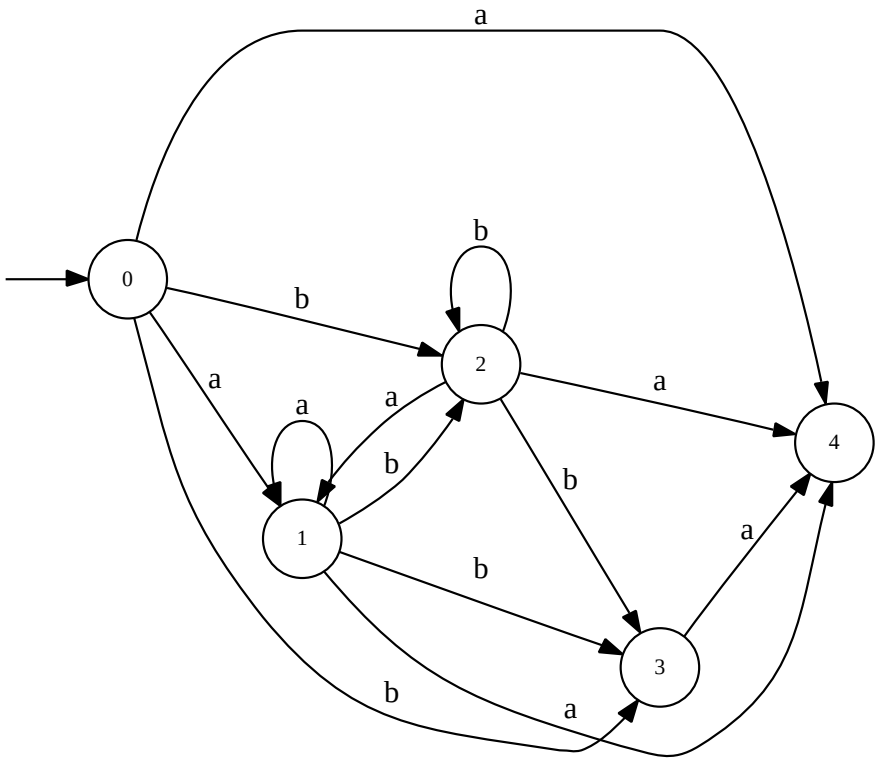
$last(E) = \{4\}$

$follow(1, E) = \{1, 2, 3, 4\}$

$follow(2, E) = \{1, 2, 3, 4\}$

$follow(3, E) = \{4\}$

$follow(4, E) = \emptyset$



Glushkov automata tulajdonságai:

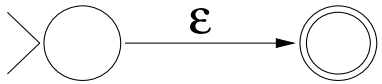
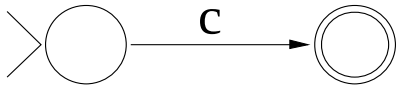
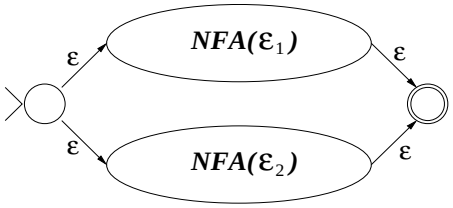
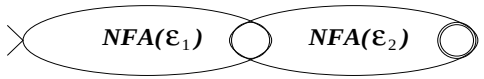
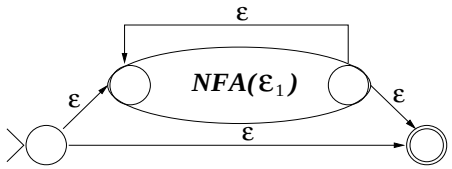
- ▷ egyetlen kezdőállapot, melyből csak kifelé indulnak élek
- ▷ nem tartalmaz ϵ-lépést (λ)
- ▷ egy állapot befutó éleihez rendelt szimbólum ,,állandó”

Thompson automata

- ▷ aritmetikai kifejezések kiértékeléséhez hasonlóan
- ▷ fordított lengyel alakban (RPN) való felírás
- ▷ prioritási sor (bináris operátorok): $(\_, |)$ [v. $(|, \_)$]
- ▷ példa: $a(b|c.a) * b$ $[lp : (\_, |)]$
- ▷ $\Rightarrow a\_(b|c\_.\_a) * \_b \Rightarrow abc.a\_\ \_| * b\_\ \_$
- ▷ RPN-`regexp` $\rightarrow$ NVA: balról jobbra olvassuk a szimbólumokat
- ▷ ha operandus: megépítjük az automatát és berakjuk a verembe (`CREAT`, `PUSH`)
- ▷ ha operátor:

- unáris: kivesszük a legfelső automatát a veremből, elvégezzük a műveletet, majd az így kapott automatát berakjuk a verembe (`POP`, `APPLY`, `PUSH`)
- bináris: kivesszük a *két* legfelső automatát, elvégezzük a műveletet, majd a kapott automatát visszarakjuk a verembe (2×`POP`, `APPLY`, `PUSH`)

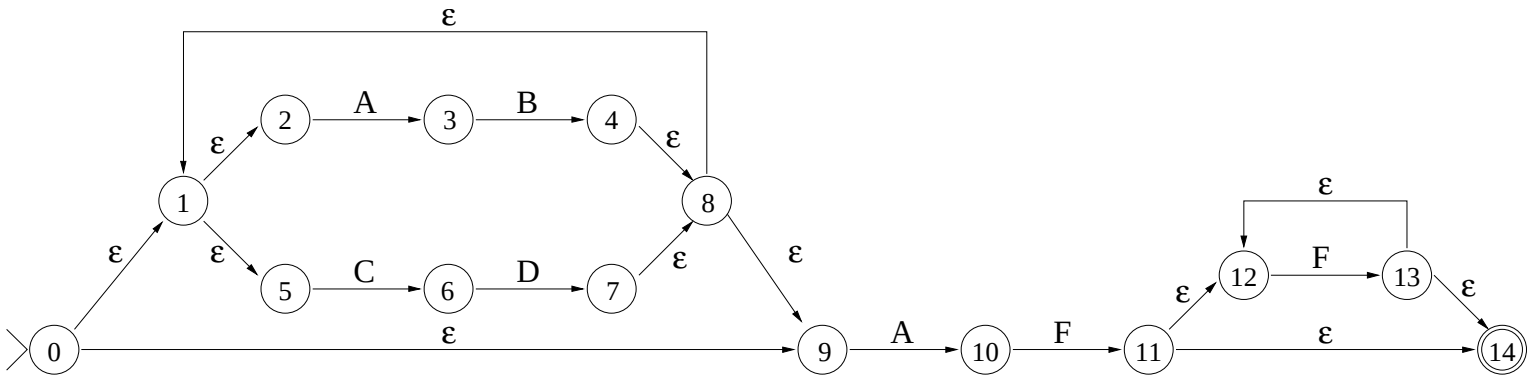
Thompson alap-automatái:

regexp	NVA
ϵ	
$c \in \Sigma$	
$\mathcal{E}_1 \vert \mathcal{E}_2$	
$\mathcal{E}_1\mathcal{E}_2$	
$\mathcal{E}_1*$	

Példa:

$E = (AB|CD) * AFF$

$[lp : (\_, |)]$



Thompson automata tulajdonságai:

- ▷ egyetlen kezdőállapot, melyből csak kifelé indulnak élek
- ▷ egyetlen végállapot, kizárólag befutó élekkel
- ▷ minden állapot maximum 2 bemenő és 2 kimenő éllel rendelkezik

NVA $\rightarrow$ ϵ-mentes DVA

„Subset construction" algoritmus

epsilon-lezárás: $eps(s)$ = azon állapotok halmaza, melyek s-ből csupa ϵ-lépéssel elérhetők $\cup\{s\}$

szomszédos állapotok adott szimbólumra:

$delta(s, c) = \delta(s, c)$ (halmaz, mivel NVA)

Alg:

1. $q'_0 = eps(q_0)$
2. minden új p DVA állapotra és minden lehetséges c input szimbólumra:
   (a) $q = delta(p, c)$
   (b) $q = eps(q)$

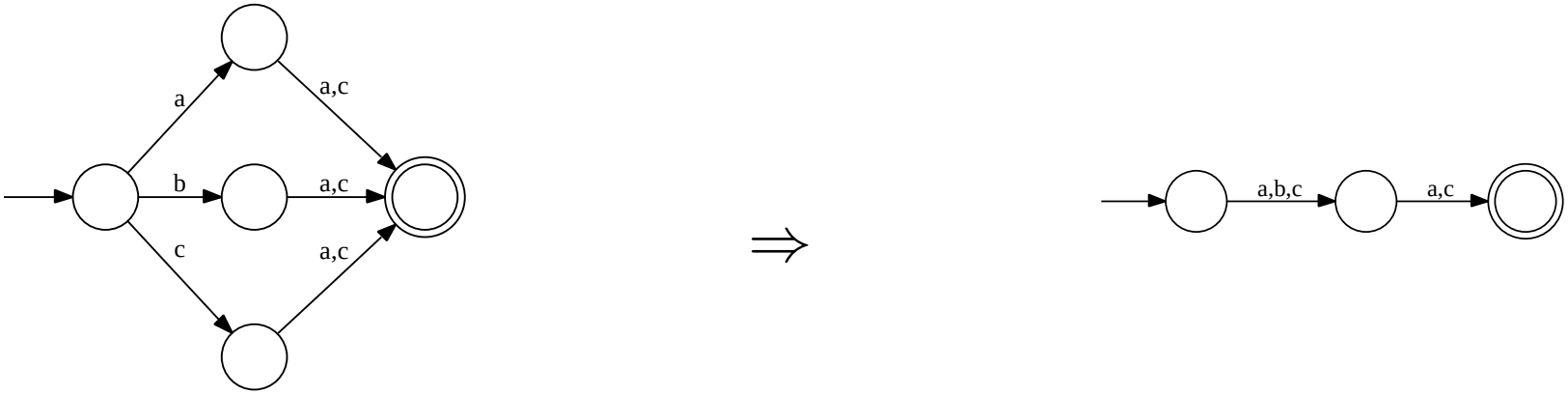
3. Ugrás a 2-es lépésre (STOP: ha a 2. lépésben nem kapunk új állapotot)

4. a DVA végállapotai azon állapotok lesznek, melyekben szerepelnek a NVA végállapotai

+ DVA optimizálás

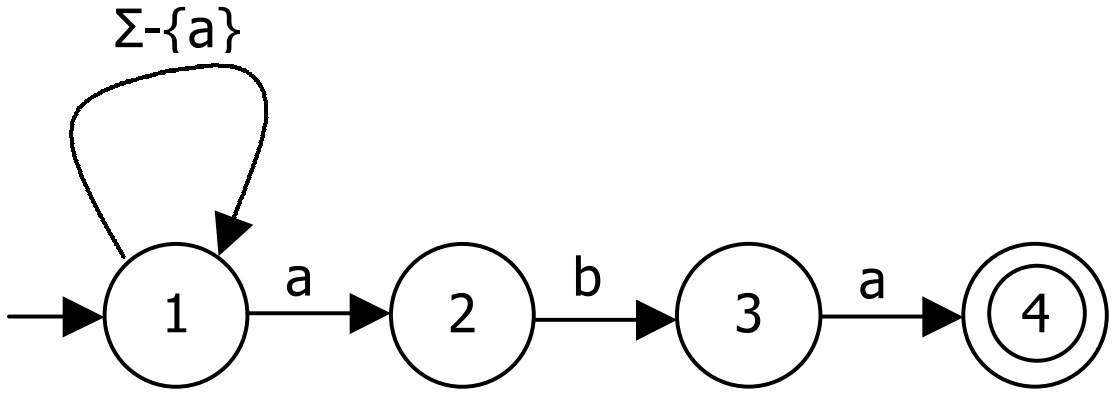
- ▷ nem elérhető állapotok törlése
- ▷ redundáns szerkezetek kiszűrése
- ▷ ...

Példa:



3. Kiértékelés

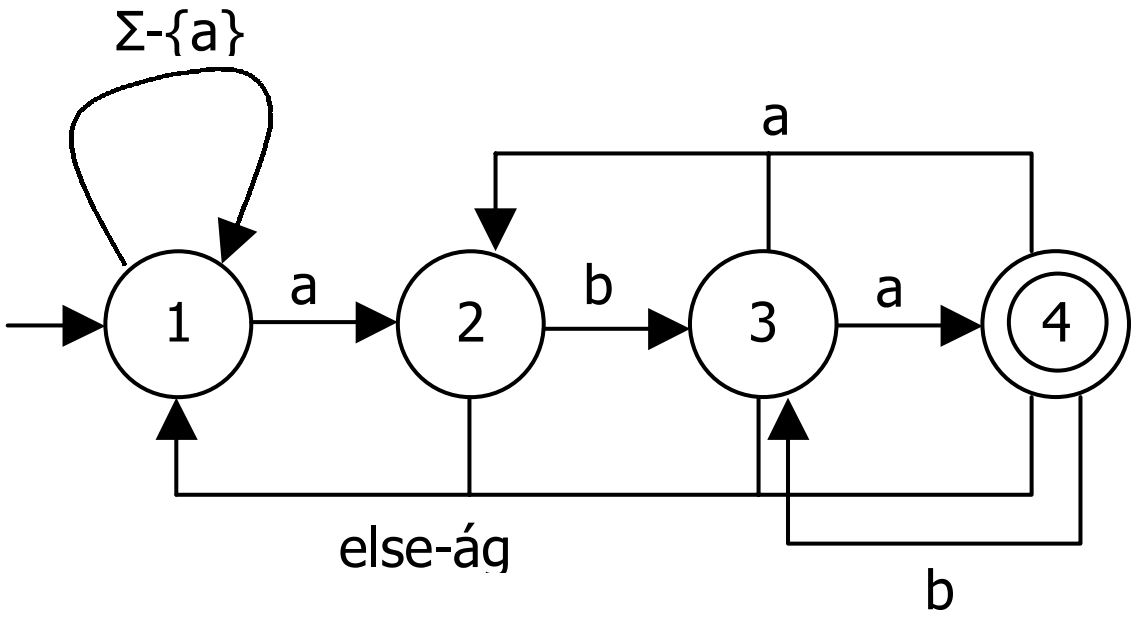
▷ **Példa:** $E = aba$



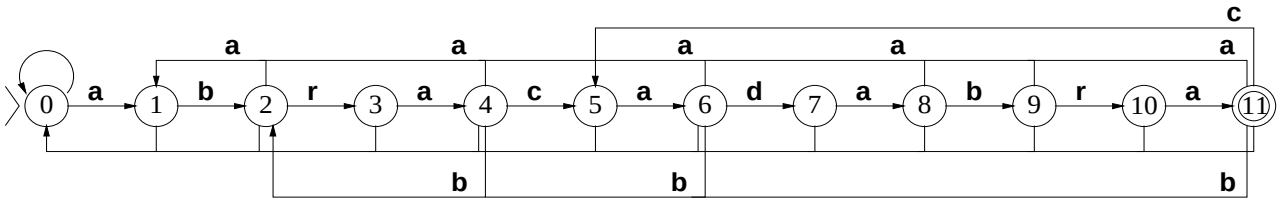
▷ ↑ tulajdonképpen egy NVA (!!!)

▷ egyszerre több aktív állapot

▷ E-nek megfelelő DVA:



- ▷ egyszerre csak egy aktív állapot
- ▷ egy bonyolultabb eset:



„Shift-And” algoritmus

- ▷ Baeza-Yates & Gonnet
- ▷ egyszerű bitműveletek
- ▷ ha $|T| = n$, a kiértékelés $\mathcal{O}(n)$ idejű
- ▷ **Alg:** (egyszerű szövegre)
  - két *bit maszk*: D és B
  - $B[c]_i = 1$ ha $E_i = c$ (bitek helyiértéke: $\leftarrow$)

(0) $D \leftarrow 0$

(1) $D \leftarrow ((D << 1)|1)\&B[c]$

(2) ha D legutolsó bitje 1 $\Rightarrow$ találat

(3) amíg nem a sztring vége, $c \leftarrow$ következő

karakter, és ugrás (1)-re

- ,,$D << 1$” – az előremutató átmeneteket szimulálja (*propagation*)
- ,,$|1$” – kezdeti hurok
- ,,$\&B[c]$” – engedélyezi a következő állapot aktiválását ha az aktuális karakter illeszkedik a minta aktuális pozíciójára
- **Példa:** $E = aba$, $T = ababa$

  $B[a] = 101,\ B[b] = 010$

$D \leftarrow 0$

$a: \ D \leftarrow (0|1)\&101 = 001$

$b: \ D \leftarrow (010|1)\&010 = 010$

$a: \ D \leftarrow (100|1)\&101 = 101 \Rightarrow$ találat! (poz. megjegyzése)

$b: \ D \leftarrow (010|1)\&010 = 010$

$a: \ D \leftarrow (100|1)\&101 = 101 \Rightarrow$ találat!

STOP

,,Shift-Or”

- 0, 1 szerepének felcserélése
- **Alg:**

  (0) $D \leftarrow 0$

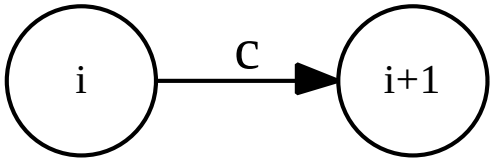
  (1) $D \leftarrow (D << 1)|B[c]$

  (2) . . . ugrás (1)-re

- mivel az eltolásnál az 1. (0.) pozíció értéke mindig 0 lesz, nincs szükség az ,,&1 . . . 10” műveletre (,,|1” megfelelője)
- eggyel kevesebb bitművelet $\Rightarrow$ gyorsabb mint elődje

`Regexp` kiértékelése „Shift-And”-del

- ▷ `regexp` → Thompson automata
- ▷ automata állapotainak átszámozása:



- ▷ minden állapot egy bit a bit maszkban, vagyis állapotok száma $= |B[\ ]| = |D||$
- ▷ $B[c]_i = 1$ ha $\overset{c}{\longrightarrow}$(i)
- ▷ $E[\ ]$ – az aktív állapotok ϵ-lezárása

▷ **Alg:**

(0) $D \leftarrow E[0 \ldots 0]$ (kezdőállapot ϵ-lezárása)

(1) $D \leftarrow (D << 1) \& B[c]$

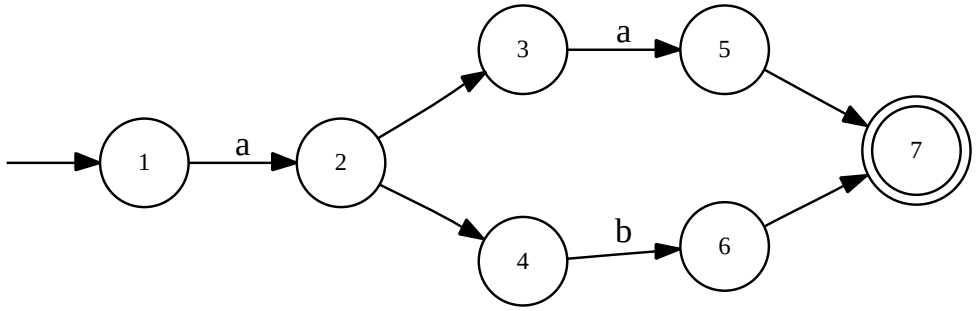
(2) $D \leftarrow E[D]$

(3) ha D legutolsó bitje 1 $\Rightarrow$ találat

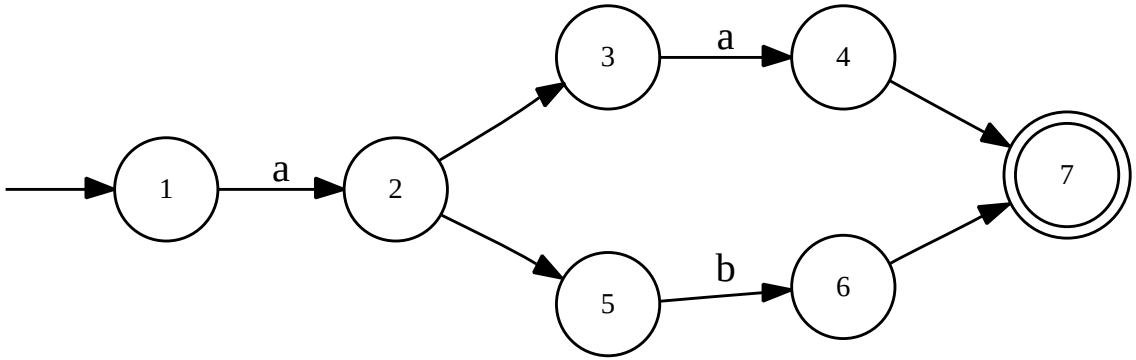
(4) amíg nem a sztring vége, $c \leftarrow$ következő karakter, és ugrás (1)-re

▷ (nincs szükség „|1”-re, mert $E[x \ldots x0] = x \ldots x1$)

Példa: `regexp`$= a(a|b)$, $T = aab$



▷ átszámozás után:



	7	6	5	4	3	2	1
$B[a] =$	0	0	0	1	0	1	0
$B[b] =$	0	1	0	0	0	0	0

$D \leftarrow E[0 \ldots 0] = 0000001$

$a: \ D \leftarrow 0000010 \& 0001010 = 0000010$

$D \leftarrow E[D] = 0001111$

$a: \ D \leftarrow 0011110 \& 0001010 = 0001010$

$D \leftarrow E[D] = 1011111 \Rightarrow$ találat! (poz. megjegyzése)

$b: \ D \leftarrow 0111110 \& 0100000 = 0100000$

$D \leftarrow E[D] = 1100000 \Rightarrow$ találat!

STOP


The End