Wyrażenia regularne i regexpy

dr inż. Marcin Ciura

Wydział Informatyki, Akademia Górniczo-Hutnicza

Motywujące przykłady

„W” czy „we”? (1)

we Francji, we Wrocławiu, we foyer
we dwoje, we troje, we czworo
we mnie
we Lwowie, we łbie, we łzach, we mgle, we śnie

„W” czy „we”? (2)

Należy używać „we” przed:

```
^([fvw]([bcdfghjklłmnprstwzż]|o[iy])
|dwoj|dwó|troj|tró|czwor|czwó|mnie$
|lwow|łb|łz|mg[lł]|sn|śn)
```

Rząd liczebników (1)

1 grat	2–4 graty	0,5–9 gratów
11 gratów	12–14 gratów	10,15–19 gratów
21 gratów	22–24 graty	20,25–29 gratów
91 gratów	92–94 graty	90,95–99 gratów

101 gratów

3141592 graty

31415926 gratów

Rząd liczebników (2)

```
^0*1$ grat
^([0-9]*[02-9])?[2-4]$ graty
^([0-9]*[05-9]|1[0-9])$ gratów
```

Wołacz imion

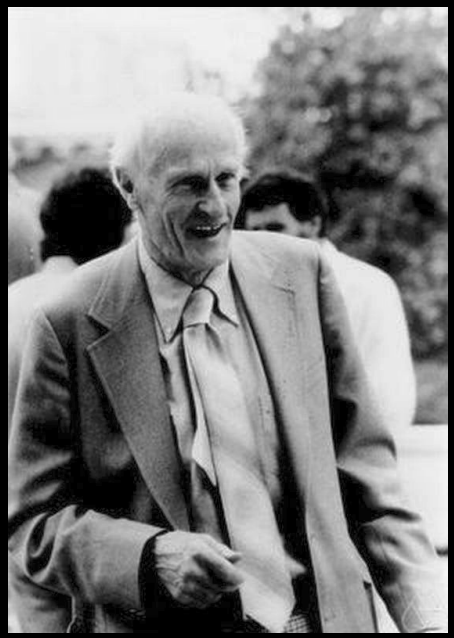
[cnsz]ia$	-len(a)	+u
ja$	-len(a)	+u
a$	-len(a)	+o
ek$	-len(ek)	+ku
eł$	-len(eł)	+le
er$	-len(er)	+rze
ł$	-len(ł)	+le
ś$	-len(ś)	+siu
niec$	-len(niec)	+ńcu
[crs]z$	-0	+u
[bfmnpswxz]$	-0	+ie
[cghjkl]$	-0	+u
d$	-0	+zie
r$	-0	+ze
t$	-len(t)	+cie

Plan na dziś: 105 slajdów, w tym 12 zagadek

- Motywujące przykłady
- Wyrażenia regularne
- Automaty skończone
- Regexpy

Wyrażenia regularne

Stephen Cole Kleene (5.1.1909–25.1.1994)



Amerykański matematyk. Wspinał się po górach, działał na rzecz ochrony przyrody. W 1951 roku, badając sieci neuronowe, wynalazł wyrażenia regularne. Jego nazwisko nosi gwiazdka Kleena’a.

Wyrażenia regularne a regex(p)y

- Wyrażenia regularne: pojęcie matematyczne
- *Regex, regexp*: implementacja wyrażeń regularnych w języku programowania, z rozszerzeniami, które ułatwiają pracę programistom (*syntactic sugar* = *lukier składniowy*)

Wyrażenia regularne: definicje (1)

- *Alfabet*: skończony zbiór *znaków*,
  np. {a, b}, ASCII, Unicode…
- *Napis* (*string*): skończony ciąg znaków,
  np. Ala␣ma␣kota

Wyrażenia regularne: definicje (2)

- *Wzorzec* pasuje do pewnego zbioru napisów
- *Wyrażenia regularne*: popularny rodzaj wzorców

Elementarne wyrażenia regularne

- zbiór pusty $\emptyset$ pasuje do pustego zbioru napisów: $\emptyset$
- pusty napis ε pasuje do jednoelementowego zbioru, złożonego z pustego napisu, czyli napisu o długości 0 znaków: $\{\varepsilon\}$
- konkretny znak a pasuje do jednoelementowego zbioru, złożonego z napisu a o długości 1 znaku: $\{a\}$

Złożone wyrażenia regularne: konkatenacja

Jeśli R i S są wyrażeniami regularnymi, to:

- wyrażenie regularne RS (konkatenacja) pasuje do zbioru, złożonego z napisów, będących konkatenacją napisów pasujących do R i pasujących do S,

  np. jeśli R pasuje do `{a, bb}`, a S do `{cc, d}`, to RS pasuje do `{acc, ad, bbcc, bbd}`

Złożone wyrażenia regularne: alternatywa

Jeśli R i S są wyrażeniami regularnymi, to:

- wyrażenie regularne $R|S$ (alternatywa) pasuje do sumy zbiorów napisów pasujących do R i pasujących do S,

  np. jeśli R pasuje do `{a, bb}`, a S do `{bb, c}`, to $R|S$ pasuje do `{a, bb, c}`

Złożone wyrażenia regularne: gwiazdka Kleene’a

Jeśli R jest wyrażeniem regularnym, to:

- wyrażenie regularne R^* (gwiazdka Kleene’a) pasuje do zbioru napisów, powstałych przez wielokrotną ($n \geq 0$) konkatenację dowolnych napisów pasujących do R,

  np. jeśli R pasuje do `{a, bc}`, to R^* pasuje do

  ```
  {ε,
    a, bc,
    aa, abc, bca, bcbc,
    aaa, aabc, abca, abcbc, bcaa, bcabc, bcbca, bcbcbc,
    ...
  }
  ```

Złożone wyrażenia regularne: nawiasy

Jeśli R jest wyrażeniem regularnym, to:

- wyrażenie regularne (R) oznacza to samo, co R

Złożone wyrażenia regularne: kolejność działań

Żeby uniknąć mnogości nawiasów, przyjmujemy następującą kolejność działań:

- najpierw stosujemy gwiazdkę Kleene’a
- potem konkatenujemy
- na końcu stosujemy alternatywę

Zagadka 1

1. `ab*` = `a(b*)` czy `(ab)*`?
2. `a|b*` = `a|(b*)` czy `(a|b)*`?
3. `ab|cd` = `(ab)|(cd)` czy `a(b|c)d`?

Zagadka 1: rozwiązanie

1. `ab*` = `a(b*)`
2. `a|b*` = `a|(b*)`
3. `ab|cd` = `(ab)|(cd)`

Zagadka 2

Do których napisów pasuje wyrażenie regularne `(b*(a|ε)b)*`

1. `ε`
2. `a`
3. `b`
4. `c`
5. `ab`
6. `aa`
7. `bb`
8. `ba`
9. `bbbbb`
10. `bbbba`
11. `abbbb`
12. `aaabb`
13. `bbabb`
14. `baabb`
15. `ababab`

Zagadka 2: rozwiązanie

Do których napisów pasuje wyrażenie regularne `(b*(a|ε)b)*`

1. `ε`
2. `a`
3. `b`
4. `c`
5. `ab`
6. `aa`
7. `bb`
8. `ba`
9. `bbbbb`
10. `bbbba`
11. `abbbb`
12. `aaabb`
13. `bbabb`
14. `baabb`
15. `ababab`

Automaty skończone

Automaty skończone: nieformalna definicja (1)

Automat skończony można sobie wyobrazić jako graf skierowany, w którym:

- wierzchołki nazywamy *stanami automatu*
- krawędzie nazywamy *przejściami między stanami*

Automaty skończone: nieformalna definicja (2)

Cechy automatu skończonego (Moore'a):

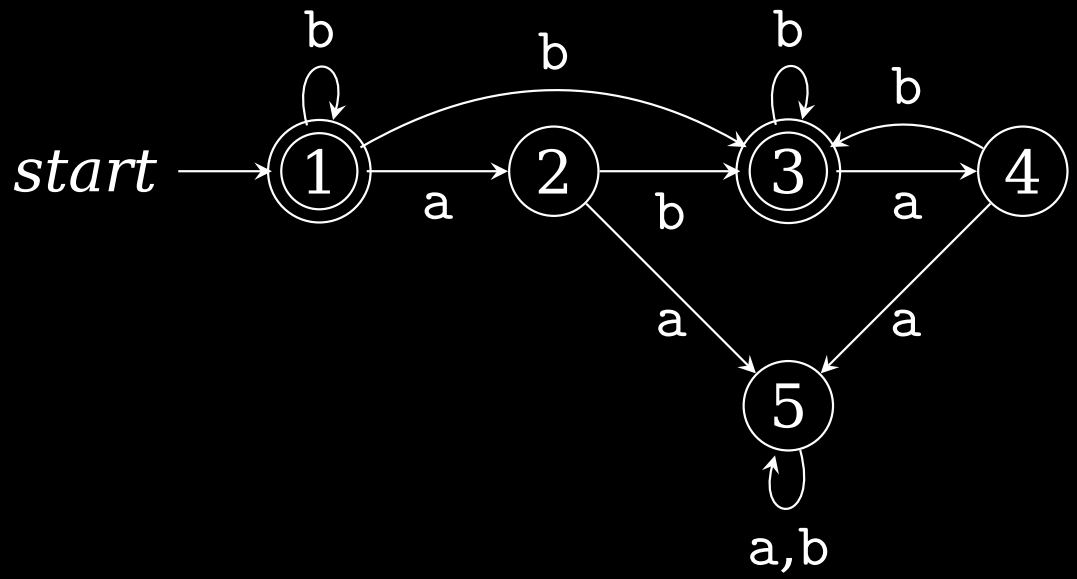
- wyróżniony jest jeden *stan początkowy*
- wyróżnione są *stany końcowe*
  (oznaczane na diagramach podwójnym kółkiem)
- każde przejście między stanami jest oznaczone co najmniej jednym symbolem alfabetu
- z każdego stanu wychodzą przejścia oznaczone wszystkimi symbolami alfabetu

Niedeterministyczne automaty skończone

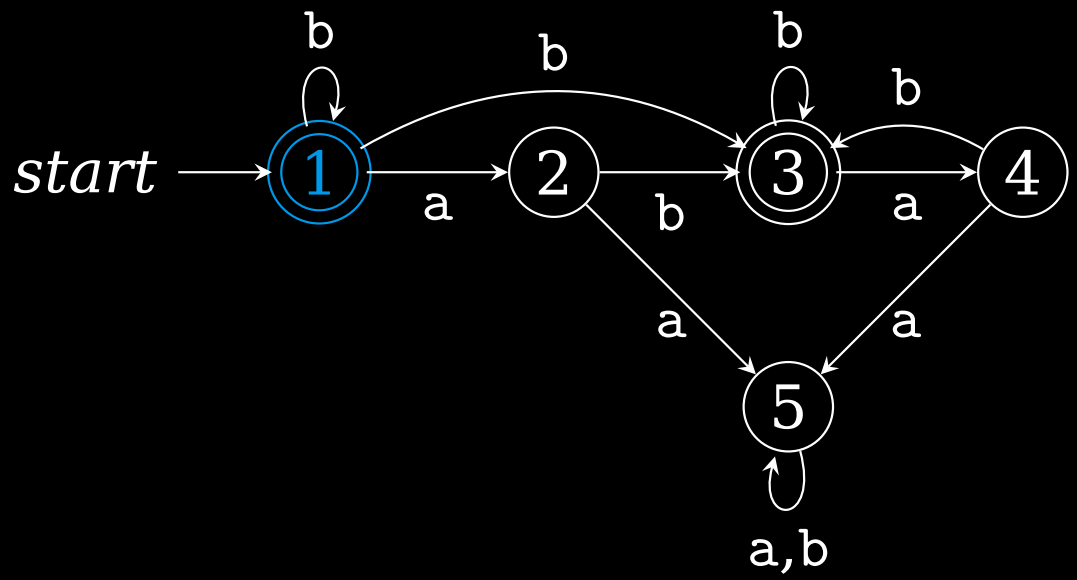
NFA (*Nondeterministic Finite Automaton*)

Ze stanów automatu niedeterministycznego może wychodzić więcej niż jedno przejście oznaczone tym samym symbolem alfabetu

NFA: przykład

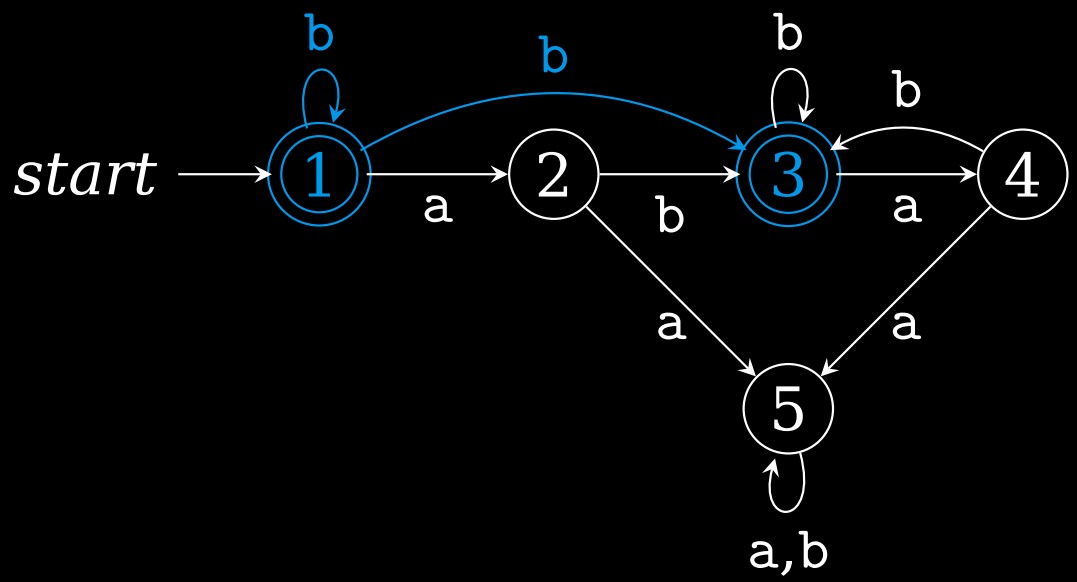


NFA: rozpoznawanie napisu (1)



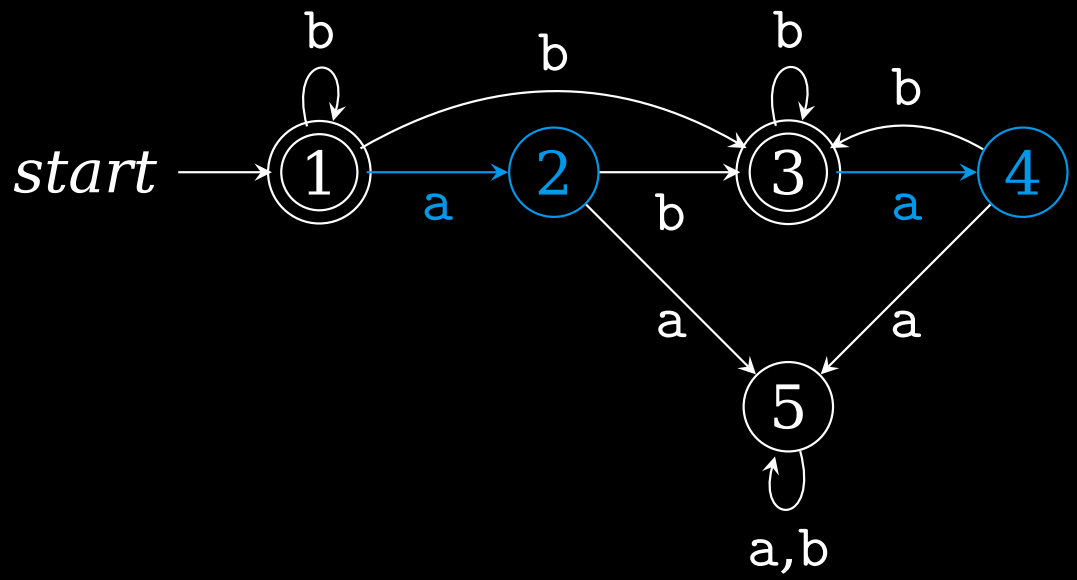
- babb

NFA: rozpoznawanie napisu (2)



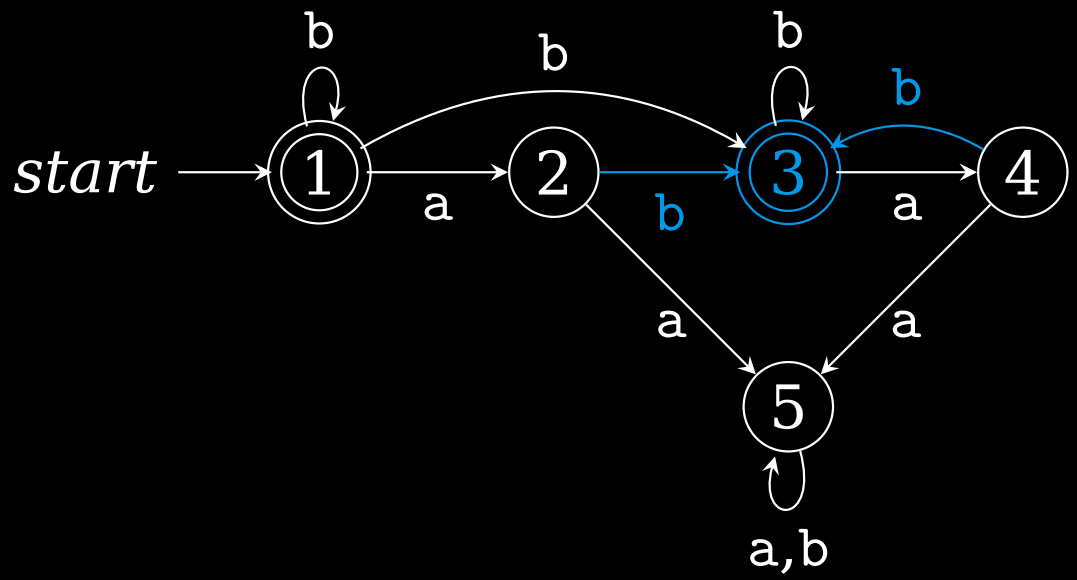
b · abb

NFA: rozpoznawanie napisu (3)



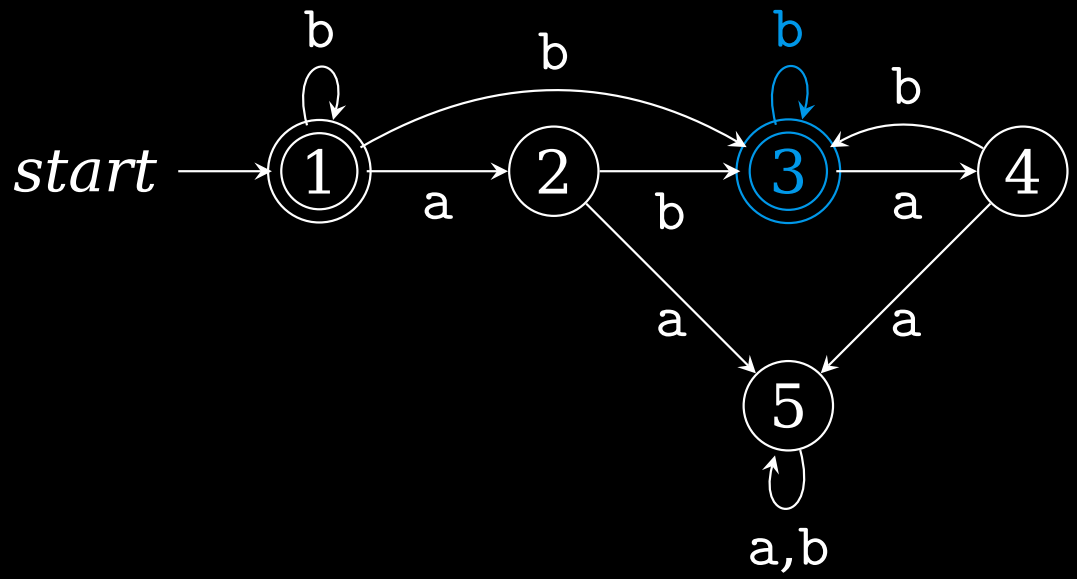
ba·bb

NFA: rozpoznawanie napisu (4)



bab·b

NFA: rozpoznawanie napisu (5)



babb·

Sukces!

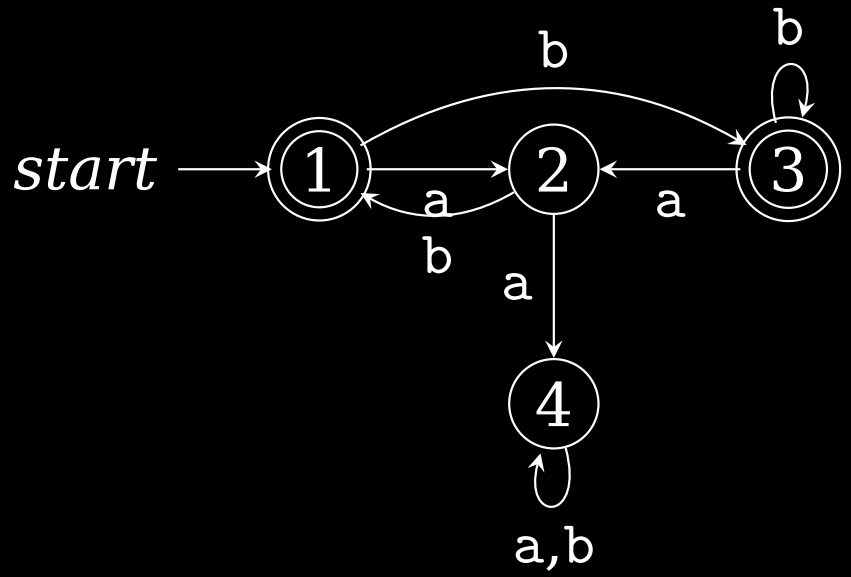
Automat zatrzymał się w stanie końcowym

Deterministyczne automaty skończone

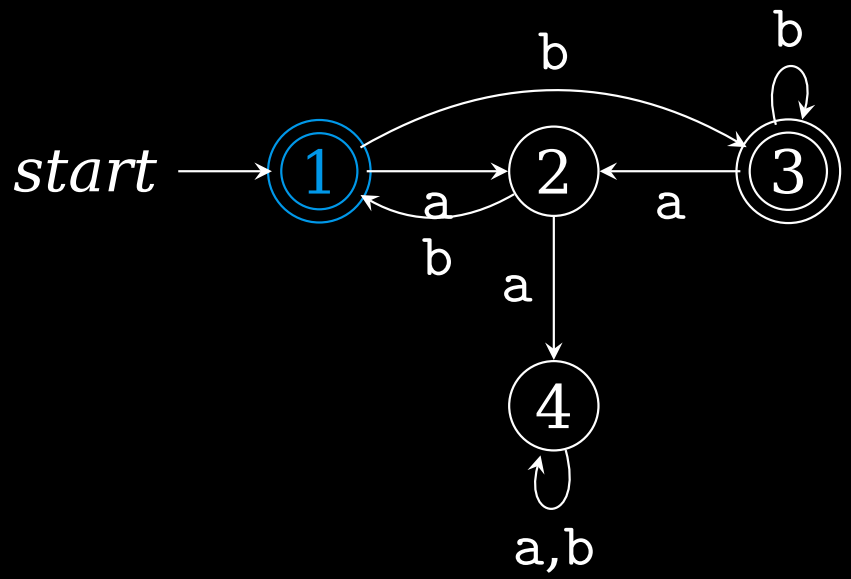
DFA (*Deterministic Finite Automaton*)

Z każdego stanu automatu deterministycznego wychodzi po jednym przejściu oznaczonym każdym z symboli alfabetu

DFA: przykład

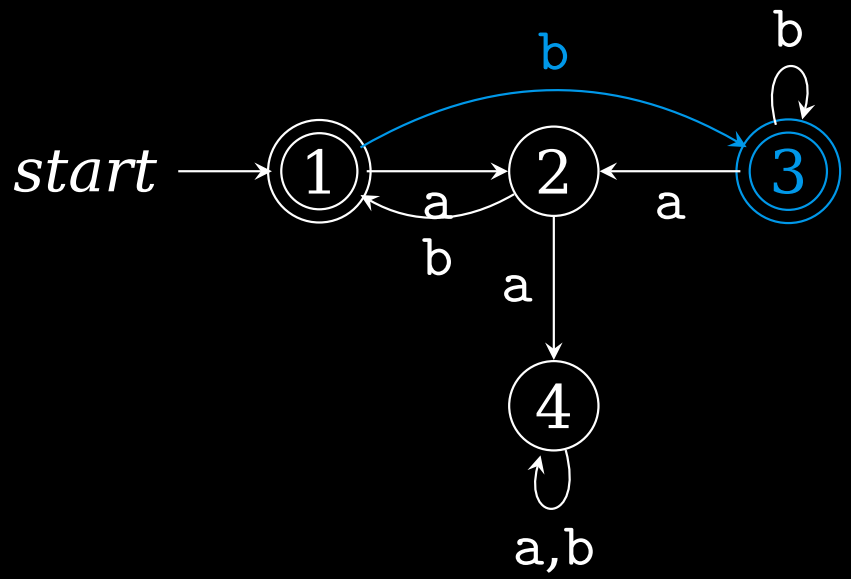


DFA: rozpoznawanie napisu (1)



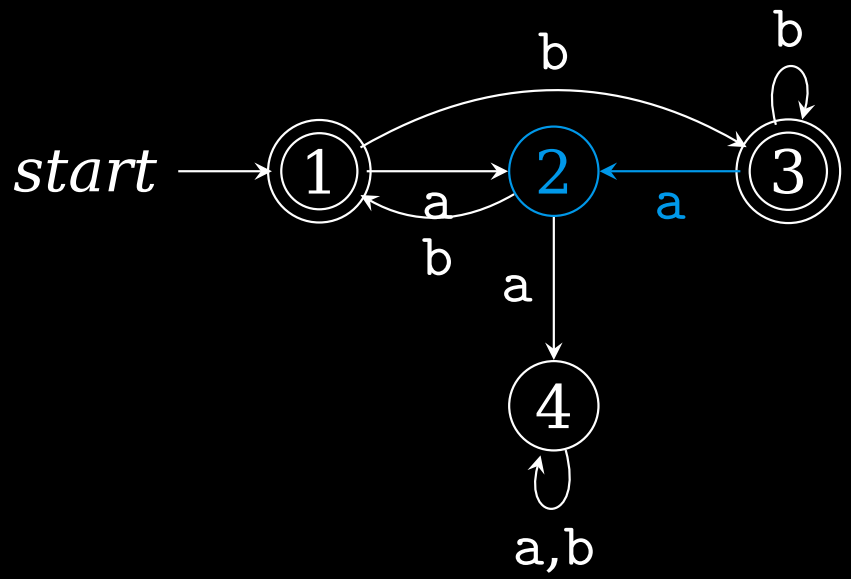
- babb

DFA: rozpoznawanie napisu (2)



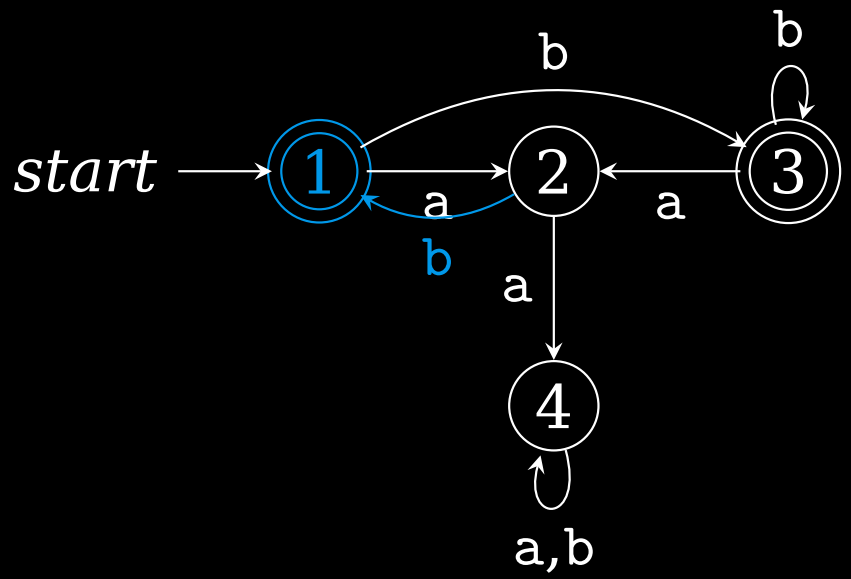
b·abb

DFA: rozpoznawanie napisu (3)



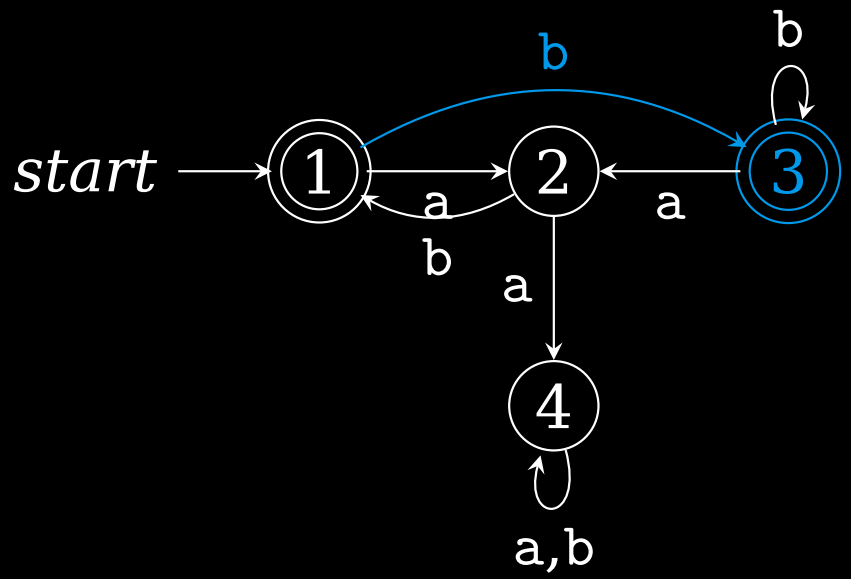
ba·bb

DFA: rozpoznawanie napisu (4)



bab·b

DFA: rozpoznawanie napisu (5)

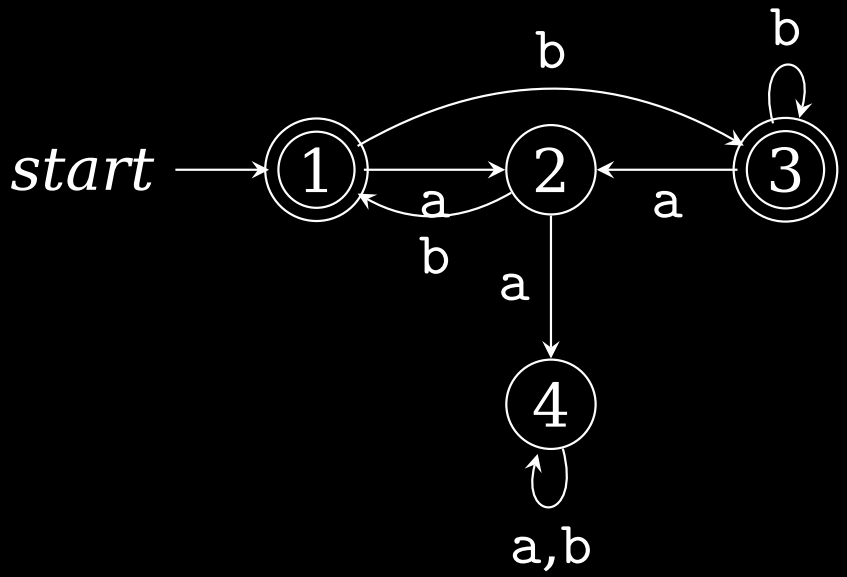


`babb`.

Sukces!

Automat zatrzymał się w stanie końcowym

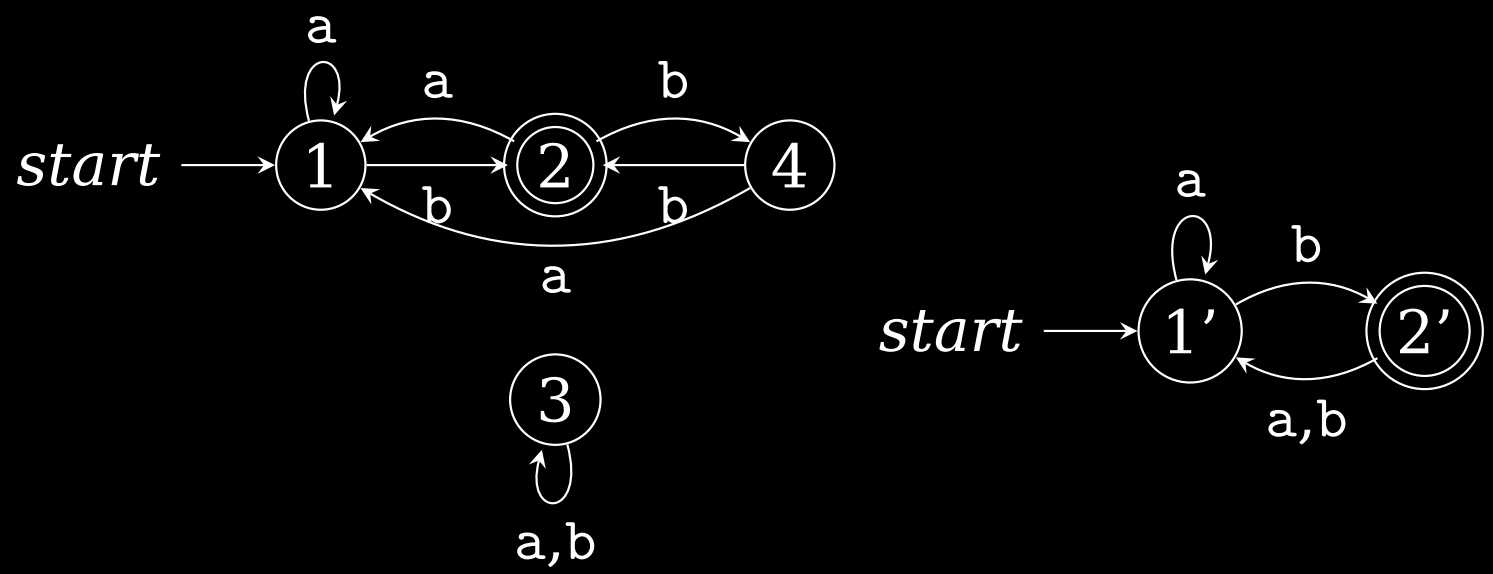
DFA: postać tablicowa



stan	koniec?	a	b
1	tak	2	3
2	nie	4	1
3	tak	2	3
4	nie	4	4

Równoważne automaty skończone

Automaty skończone, które rozpoznają
te same zbiory napisów



Minimalny DFA

Minimalny DFA nie ma zbędnych stanów, czyli:

- stanów, które można połączyć w jeden stan bez zmiany zbioru napisów rozpoznawanych przez DFA
- stanów, do których nie można dotrzeć ze stanu początkowego

Dopasowywanie wyrażenia regularnego do napisu

- Bezpośrednio interpretować wyrażenie regularne
- Zbudować NFA, odpowiadający wyrażeniu regularnemu (np. algorytmem Thompsona), po czym:
  - przechodzić przez NFA, pamiętając bieżący *zbiór stanów*
  - przechodzić przez NFA z nawrotami (*backtracking*)
  - zbudować DFA równoważny temu NFA
- Od razu zbudować DFA, odpowiadający danemu wyrażeniu regularnemu, korzystając z:
  - algorytmu Myhilla-Nerode'a
  - algorytmu DeRemera
  - pochodnych Brzozowskiego

Pochodne Brzozowskiego

Janusz Antoni Brzozowski (10.5.1935–24.10.2019)

Polsko-kanadyjski informatyk. Urodził się w Warszawie.
W 1964 roku wynalazł pochodną Brzozowskiego.

Pochodna Brzozowskiego zbioru napisów

Dane:

- zbiór napisów $\mathcal{L}$
- symbol a

Określanie *pochodnej Brzozowskiego* $\partial_a \mathcal{L}$:

- znaleźć w $\mathcal{L}$ napisy, zaczynające się od symbolu a
- odciąć pierwszy symbol a
  od każdego ze znalezionych napisów

Zagadka 3

$$\partial_{\mathtt{w}}\{\mathtt{się}, \mathtt{i}, \mathtt{w}, \mathtt{wszystko}, \mathtt{więc}\} = ?$$
$$\partial_{\mathtt{n}}\{\varepsilon, \mathtt{nie}, \mathtt{na}, \mathtt{o}\} = ?$$
$$\partial_{\mathtt{z}}\{\varepsilon, \mathtt{z}, \mathtt{za}, \mathtt{abrakadabra}\} = ?$$

Zagadka 3: rozwiązanie

$$\partial_{\mathtt{w}}\{\mathtt{się}, \mathtt{i}, \mathtt{w}, \mathtt{wszystko}, \mathtt{więc}\} = \{\varepsilon, \mathtt{szystko}, \mathtt{ięc}\}$$
$$\partial_{\mathtt{n}}\{\varepsilon, \mathtt{nie}, \mathtt{na}, \mathtt{o}\} = \{\mathtt{ie}, \mathtt{a}\}$$
$$\partial_{\mathtt{z}}\{\varepsilon, \mathtt{z}, \mathtt{za}, \mathtt{abrakadabra}\} = \{\varepsilon, \mathtt{a}\}$$

Pochodna Brzozowskiego wyrażeń regularnych (1)

$\partial_a \emptyset = ?$

Pochodna Brzozowskiego wyrażeń regularnych (2)

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = ?$

Pochodna Brzozowskiego wyrażeń regularnych (3)

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = \emptyset$

$\partial_a a = ?$

Pochodna Brzozowskiego wyrażeń regularnych (4)

$$\partial_a \emptyset = \emptyset$$
$$\partial_a \varepsilon = \emptyset$$
$$\partial_a a = \varepsilon$$
$$\partial_a b = ?$$

Pochodna Brzozowskiego wyrażeń regularnych (5)

$$\partial_a \emptyset = \emptyset$$
$$\partial_a \varepsilon = \emptyset$$
$$\partial_a a = \varepsilon$$
$$\partial_a b = \emptyset$$
$$\partial_a aR = ?$$

Pochodna Brzozowskiego wyrażeń regularnych (6)

$$\partial_a \emptyset = \emptyset$$
$$\partial_a \varepsilon = \emptyset$$
$$\partial_a a = \varepsilon$$
$$\partial_a b = \emptyset$$
$$\partial_a aR = R$$
$$\partial_a bR = ?$$

Pochodna Brzozowskiego wyrażeń regularnych (7)

$$\partial_a \emptyset = \emptyset$$
$$\partial_a \varepsilon = \emptyset$$
$$\partial_a a = \varepsilon$$
$$\partial_a b = \emptyset$$
$$\partial_a aR = R$$
$$\partial_a bR = \emptyset$$
$$\partial_a (R|S) = ?$$

Pochodna Brzozowskiego wyrażeń regularnych (8)

$$\partial_a \emptyset = \emptyset$$
$$\partial_a \varepsilon = \emptyset$$
$$\partial_a a = \varepsilon$$
$$\partial_a b = \emptyset$$
$$\partial_a aR = R$$
$$\partial_a bR = \emptyset$$
$$\partial_a (R|S) = \partial_a (R) | \partial_a (S)$$
$$\partial_a R^* = ?$$

Pochodna Brzozowskiego wyrażeń regularnych (9)

$$\partial_a \emptyset = \emptyset$$
$$\partial_a \varepsilon = \emptyset$$
$$\partial_a a = \varepsilon$$
$$\partial_a b = \emptyset$$
$$\partial_a aR = R$$
$$\partial_a bR = \emptyset$$
$$\partial_a (R|S) = \partial_a (R) | \partial_a (S)$$
$$\partial_a R^* = (\partial_a R) R^*$$

Zagadka 4

$\partial_{\texttt{w}}\texttt{się|i|nie|w(}\varepsilon\texttt{|szystko|ięc)} = ?$

$[\partial_a R^* = (\partial_a R)R^*]$

$\partial_{\texttt{x}}\texttt{x}^* = ?$

$\partial_{\texttt{x}}((\texttt{xy})^*) = ?$

$\partial_{\texttt{x}}((\texttt{x|y})^*) = ?$

$\partial_{\texttt{x}}((\texttt{x}^*\texttt{|y})^*) = ?$

Zagadka 4: rozwiązanie

$$\partial_{\mathtt{w}}\mathtt{się|i|nie|w(\varepsilon|szystko|ięc)} = \mathtt{\varepsilon|szystko|ięc}$$

$$[\partial_a R^* = (\partial_a R)R^*]$$

$$\partial_{\mathtt{x}}\mathtt{x}^* = \mathtt{x}^*$$

$$\partial_{\mathtt{x}}((\mathtt{xy})^*) = \partial_{\mathtt{x}}(\mathtt{xy})(\mathtt{xy})^* = \mathtt{y(xy)}^*$$

$$\partial_{\mathtt{x}}((\mathtt{x|y})^*) = \partial_{\mathtt{x}}(\mathtt{x|y})(\mathtt{x|y})^* = (\partial_{\mathtt{x}}\mathtt{x}|\partial_{\mathtt{x}}\mathtt{y})(\mathtt{x|y})^* = \mathtt{(x|y)}^*$$

$$\partial_{\mathtt{x}}((\mathtt{x}^*|\mathtt{y})^*) = \partial_{\mathtt{x}}(\mathtt{x}^*|\mathtt{y})(\mathtt{x}^*|\mathtt{y})^* = \mathtt{x}^*\mathtt{(x}^*|\mathtt{y)}^*$$

Algorytm oparty na pochodnych Brzozowskiego

Dane: alfabet Σ i wyrażenie regularne R

V := $\emptyset$;
E := $\emptyset$;
W := $\{R\}$;
while $W \neq \emptyset$ **do**
    wybierz dowolny stan $w \in W$;
    $W := W \setminus \{w\}$;
    foreach $s \in \Sigma$ **do**
        $d := \partial_s w$;
        if $d \notin V$ **then**
            $W := W \cup \{d\}$;
            $V := V \cup \{d\}$;
    $E := E \cup \{(w, d, s)\}$;

Wyniki: V: zbiór stanów DFA, E: zbiór przejść DFA

Algorytm Brzozowskiego: przykład (1)

1: `(b*(a|ε)b)*`

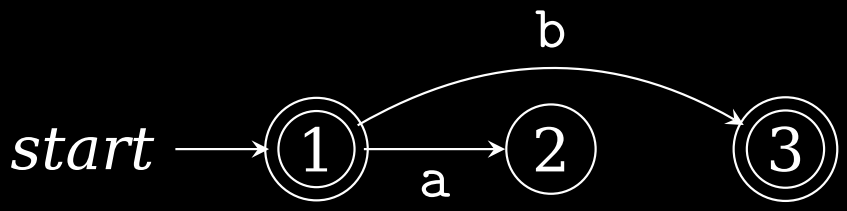
Algorytm Brzozowskiego: przykład (2)

1: `(b*(a|ε)b)*`

2: `b(b*(a|ε)b)*`

∂_a`(b*(a|ε)b)*` = `b(b*(a|ε)b)*`

Algorytm Brzozowskiego: przykład (3)



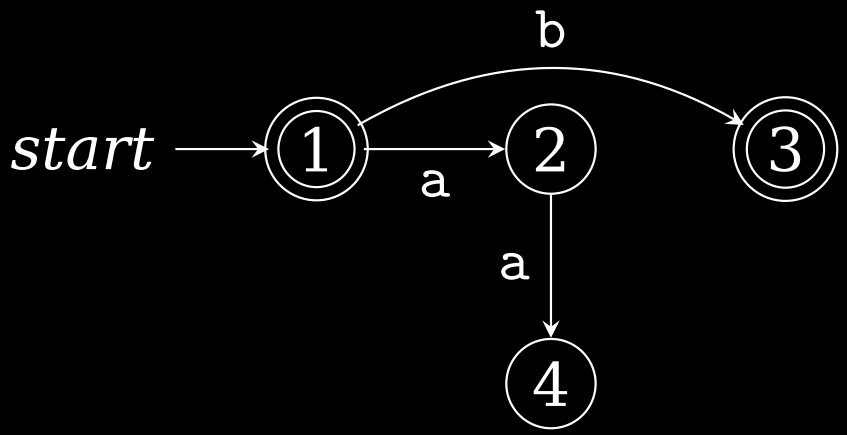
1: `(b*(a|ε)b)*`

2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

$$\partial_b(\texttt{b*(a|ε)b)*} = \texttt{b*(b*(a|ε)b)*}$$

Algorytm Brzozowskiego: przykład (4)



1: `(b*(a|ε)b)*`

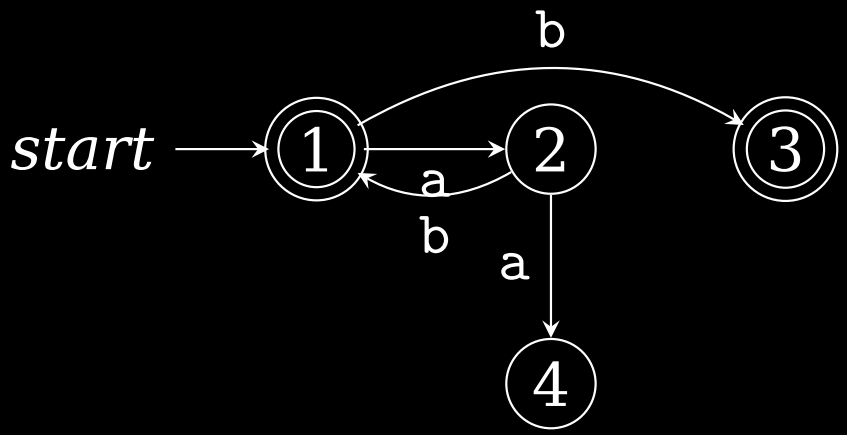
2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

4: Ø

∂_a`b(b*(a|ε)b)*` = Ø

Algorytm Brzozowskiego: przykład (5)



1: `(b*(a|ε)b)*`

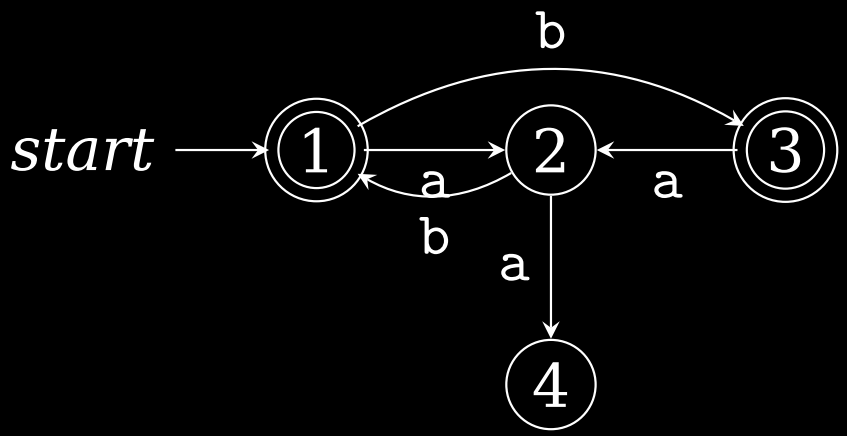
2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

4: Ø

∂_b`b(b*(a|ε)b)*` = `(b*(a|ε)b)*`

Algorytm Brzozowskiego: przykład (6)



1: `(b*(a|ε)b)*`

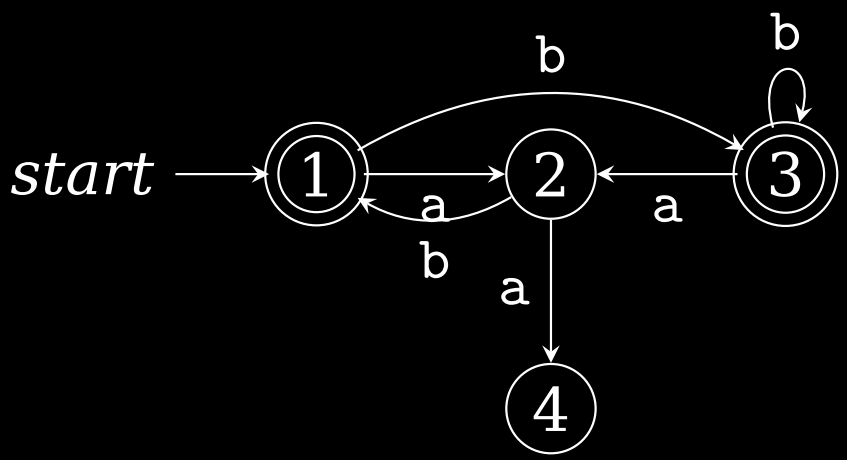
2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

4: Ø

$$\partial_a \texttt{b*(b*(a|ε)b)*} = \texttt{b(b*(a|ε)b)*}$$

Algorytm Brzozowskiego: przykład (7)



1: `(b*(a|ε)b)*`

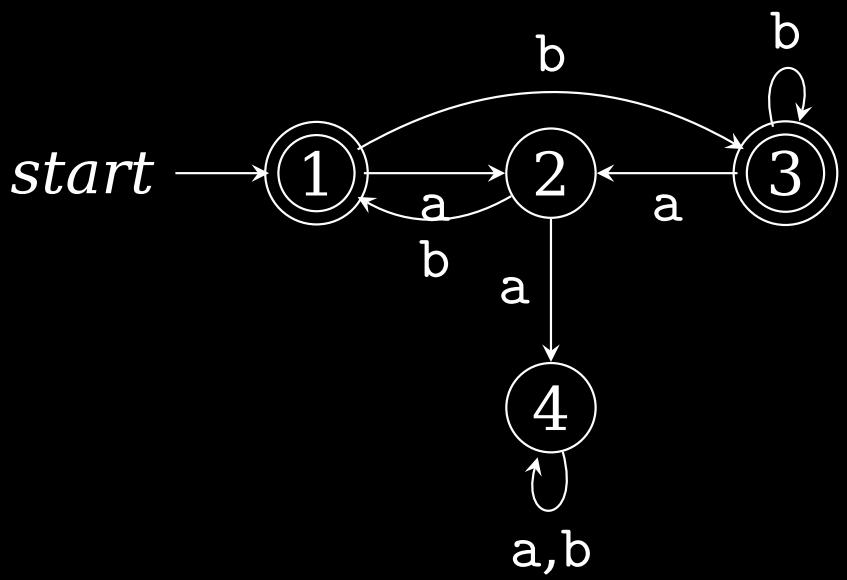
2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

4: Ø

$$\partial_b \texttt{b*(b*(a|ε)b)*} = \texttt{b*(b*(a|ε)b)*}$$

Algorytm Brzozowskiego: przykład (8)



$$\partial_a \emptyset = \partial_b \emptyset = \emptyset$$

1: `(b*(a|ε)b)*`

2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

4: $\emptyset$

Regexpy

Regexpy

Implementacja wyrażeń regularnych z dodatkiem:

- dopasowywania zbiorów znaków
- metaznaków oznaczających początek i koniec napisu oraz dowolny znak
- dopasowywania (pod-)wyrażeń występujących co najwyżej jeden raz, co najmniej jeden raz, określoną liczbę razy
- dopasowywania napisu, który dopasował się do podwyrażenia (*backreference*)
- innych udogodnień, często różnych w różnych implementacjach

Regexpy: działania na napisach

- Dopasowywanie
- Wyszukiwanie
- Zamiana podnapisów
- Podział napisu na części

Regexpy w Pythonie

```
import re

r1 = re.compile('kot|pies')
r1.match(text)     # Dopasowanie
r1.search(text)    # Wyszukiwanie
r1.findall(text)   # Wyszukiwanie
r1.sub('królik', text)  # Zamiana podnapisów

r2 = re.compile(',|;')
r2.split(text)     # Podział napisu na części
```

Regexpy: zbiory znaków `[...]` (1)

Między metaznakami `[` oraz `]` mieszczą się regexpy, pasujące do zbiorów znaków. Mogą to być:

- indywidualne znaki, np. `[ąę]` = `(ą|ę)`
- przedziały znaków, np. `[A-D]` = `(A|B|C|D)`
- zanegowane zbiory znaków, np. `[^x-z]` pasuje do każdego pojedynczego znaku oprócz `x`, `y` i `z`

Regexpy: zbiory znaków `[...]` (2)

Między metaznakami `[` oraz `]`:

- do prawdziwego znaku `^` pasuje:
  - albo `\^`
  - albo znak `^`, jeśli nie występuje zaraz po `[`
- do prawdziwego znaku `]` pasuje:
  - albo `\]`
  - albo znak `]` zaraz po `[`
- do prawdziwego znaku `-` pasuje:
  - albo `\-`
  - albo znak `-` zaraz po `[`
  - albo znak `-` zaraz przed `]`

Regexpy: zbiory znaków `[...]` (3)

Przykład:

```
[A-ZĄĆĘŁŃÓŚŹŻa-ząćęłńóśźż]
```

Regexpy: metaznaki `^` i `$`

Metaznak `^` pasuje do początku napisu, a w trybie `re.MULTILINE` też do początku każdego wiersza napisu

Metaznak `$` pasuje do końca napisu, a w trybie `re.MULTILINE` też do końca każdego wiersza napisu

Prawdziwe znaki strzałki w górę i dolara: `\^` i `\$`

Częsty błąd: `^kot|pies$` = `(^kot)|(pies$)`
Poprawnie: `^(kot|pies)$`

Zagadka 5

```
import re

r1 = re.compile('dom')
r1.search('wiadomo') is not None

r2 = re.compile('^dom')
r2.search('domownik') is not None

r3 = re.compile('dom$')
r3.search('świadom') is not None

r4 = re.compile('^dom$')
r4.search('domek') is not None
```

Zagadka 5: rozwiązanie

```
import re

r1 = re.compile('dom')
r1.search('wiadomo') is not None
True
r2 = re.compile('^dom')
r2.search('domownik') is not None
True
r3 = re.compile('dom$')
r3.search('świadom') is not None
True
r4 = re.compile('^dom$')
r4.search('domek') is not None
False
```

Zagadka 6

```
import re

r = re.compile('dom')
r.match('wiadomo') is not None

r.match('domownik') is not None

r.match('świadom') is not None

r.match('domek') is not None
```

Zagadka 6: rozwiązanie

```
import re
r = re.compile('dom')
r.match('wiadomo') is not None
False
r.match('domownik') is not None
True
r.match('świadom') is not None
False
r.match('domek') is not None
True
```

Regexpy: metaznak `.`

Metaznak `.` pasuje do dowolnego znaku oprócz znaku nowego wiersza

Prawdziwy znak kropki: `\.` lub `[.]`

Zagadka 7

```
import re

r1 = re.compile('^d.m$')
r1.findall(słownik, re.MULTILINE)


r2 = re.compile('^K.*ów$')
r2.findall(słownik, re.MULTILINE)
```

Zagadka 7: rozwiązanie

```
import re

r1 = re.compile('^d.m$')
r1.findall(słownik, re.MULTILINE)
['dam', 'dem', 'dom', 'dum', 'dym']

r2 = re.compile('^K.*ów$')
r2.findall(słownik, re.MULTILINE)
['Kraków', 'Knurów', 'Kryspinów', 'Krzeszów',
 'Kijów', 'Kiszyniów',...
```

Regexpy: powtórzenia podregexpów

- Znak zapytania oznacza, że poprzedni regexp może wystąpić albo 0 razy albo 1 raz:
  - $R? = R|\varepsilon$
- Znak plus oznacza, że poprzedni regexp może wystąpić co najmniej 1 raz:
  - $R+ = R|R*$
- Nawiasy klamrowe otaczają konkretne liczby powtórzeń poprzedniego regexpa:
  - $R\{3\} = RRR$
  - $R\{3,5\} = RRR|RRRR|RRRRR$
  - $R\{2,\} = RRR*$
  - $R\{,3\} = \varepsilon|R|RR|RRR$

Dygresja 1: „syndrom pochylonych wykałaczek” (1)

Leaning Toothpick Syndrome

```
import re

root_dir = re.compile('^[A-Za-z]:\\\\$')
```

Dygresja 1: „syndrom pochylonych wykałaczek” (2)

Leaning Toothpick Syndrome

```
import re
root_dir = re.compile('^[A-Za-z]:\\\\$')
root_dir = re.compile(r'^[A-Za-z]:\\$')
```

Regexpy: grupy (1)

(Pod-)regexp pomiędzy nawiasami okrągłymi `(...)` nazywa się *grupą*

Zagadka 7

Do czego pasuje taki regexp?

```
^(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]
 (25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]
 (25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]
 (25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$
```

Regexpy: grupy (2)

```
import re

r = re.compile(
  '^(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]'
  '(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]'
  '(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]'
  '(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$')
m = r.match('127.0.0.1')
m.group(1), m.group(2), m.group(3), m.group(4)
```

Regexpy: grupy (3)

```
import re

r = re.compile(
  '^(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]'
  '(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]'
  '(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])[.]'
  '(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9]?[0-9])$')
m = r.match('127.0.0.1')
m.group(1), m.group(2), m.group(3), m.group(4)
('127', '0', '0', '1')
```

Regexpy: *backreference*

Regexpy `\1`…`\99` pasują do tego samego, do czego pasuje grupa o odpowiednim numerze kolejnym

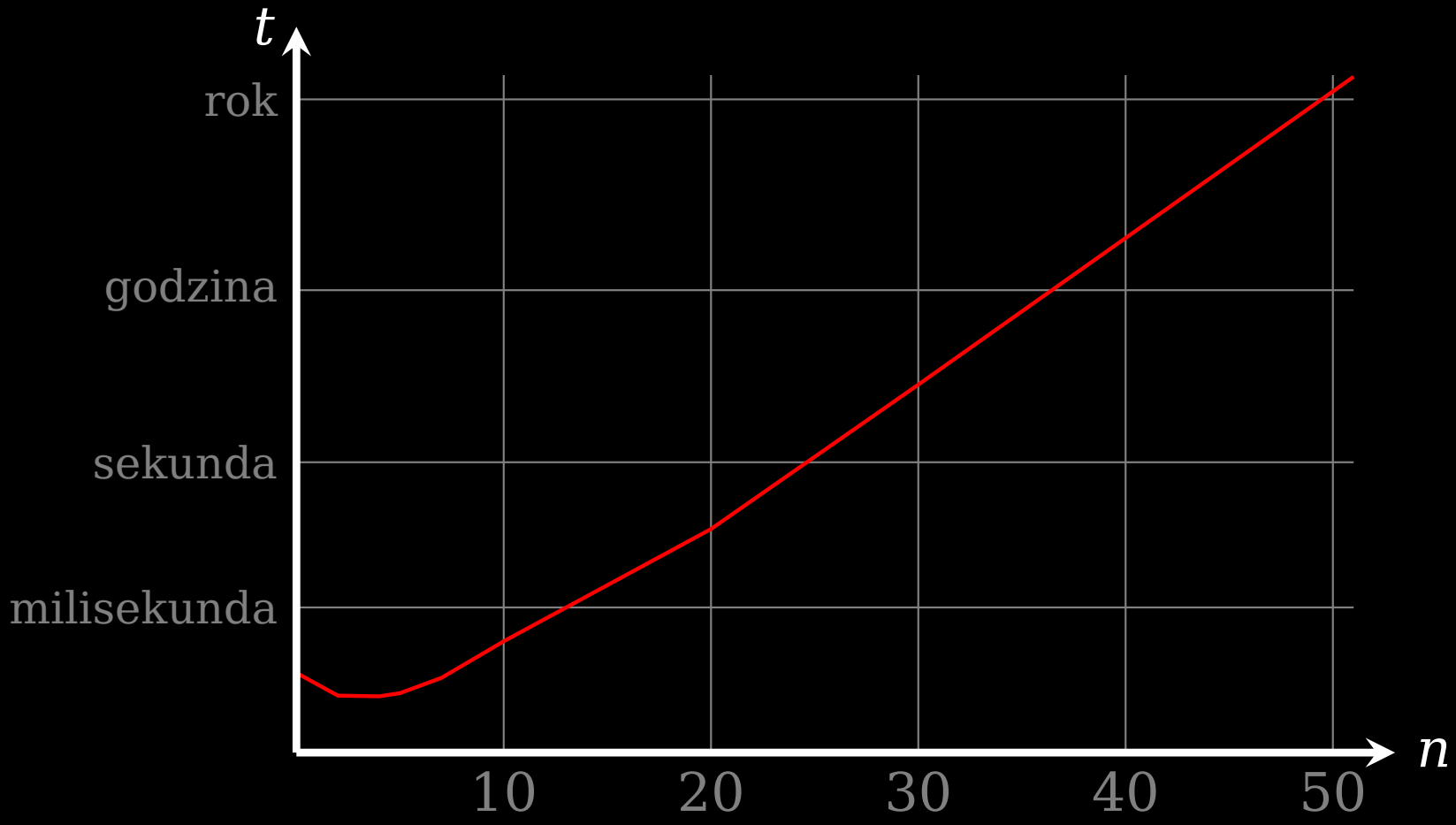
Zagadka 8

```
import re

r = re.compile(r'^(.+)\1$')
r.findall(słownik, RE.MULTILINE)
```

Zagadka 8: rozwiązanie

```
import re

r = re.compile(r'^(.+)\1$')
r.findall(słownik, RE.MULTILINE)
['baba', 'berber', 'bobo', 'dada', 'dodo',
 'dowodowo', 'dudu', 'dziadzia', 'dzidzi', 'gogo',
 'jaja', 'jojo', 'kankan', 'kuku', 'kuskus', 'lala',
 'lulu', 'mama', 'niania', 'ojoj', 'papa', 'psipsi',
 'siusiu', 'tata', 'toto']
```

Dygresja 2: nieliniowy wzrost liczby nawrotów (1)




```
r = re.compile(f'(a?){{{n}}}a{{{n}}}')
r.match(n * 'a')
```

Dygresja 2: blokada usług przez regexpy (1)

DoS = Denial of Service

Gdy użytkownicy mogą:

- wprowadzać dowolne regexpy
- wprowadzać dowolne napisy, które prowadzą do nieliniowej liczby nawrotów regexpu użytego w programie

Dotyczy tylko regexpów korzystających z NFA!

Dygresja 2: blokada usług przez regexpy (2)

W 150 najpopularniejszych javowych aplikacjach na GitHubie, które zawierały jakiekolwiek regexpy, znaleziono:

2868 regexpów

522 regexpy z nieliniową liczbą nawrotów, w tym

37 regexpów z wykładniczą liczbą nawrotów

27 z tych aplikacji udało się na 41 sposobów zablokować na ponad 10 minut

(V. Wüstholz *i inni,* Static Detection of DoS Vulnerabilities in Programs that use Regular Expressions, w: TACAS'17)

Zagadka 9

```
^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$
```

Zagadka 9: rozwiązanie

```
^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$
```

pasuje do liczb w notacji naukowej

Zagadka 10

```
^M{,3}(C[MD]|D?C{,3})(X[CL]|L?X{,3})(I[XV]|V?I{,3})$
```

Zagadka 10: znaleźć bug

```
^M{,3}(C[MD]|D?C{,3})(X[CL]|L?X{,3})(I[XV]|V?I{,3})$
```

pasuje do liczb rzymskich

Zagadka 10: rozwiązanie

```
^M{,3}(C[MD]|D?C{,3})(X[CL]|L?X{,3})(I[XV]|V?I{,3})$
```

pasuje do liczb rzymskich,
ale też do pustego napisu

Zagadka 11

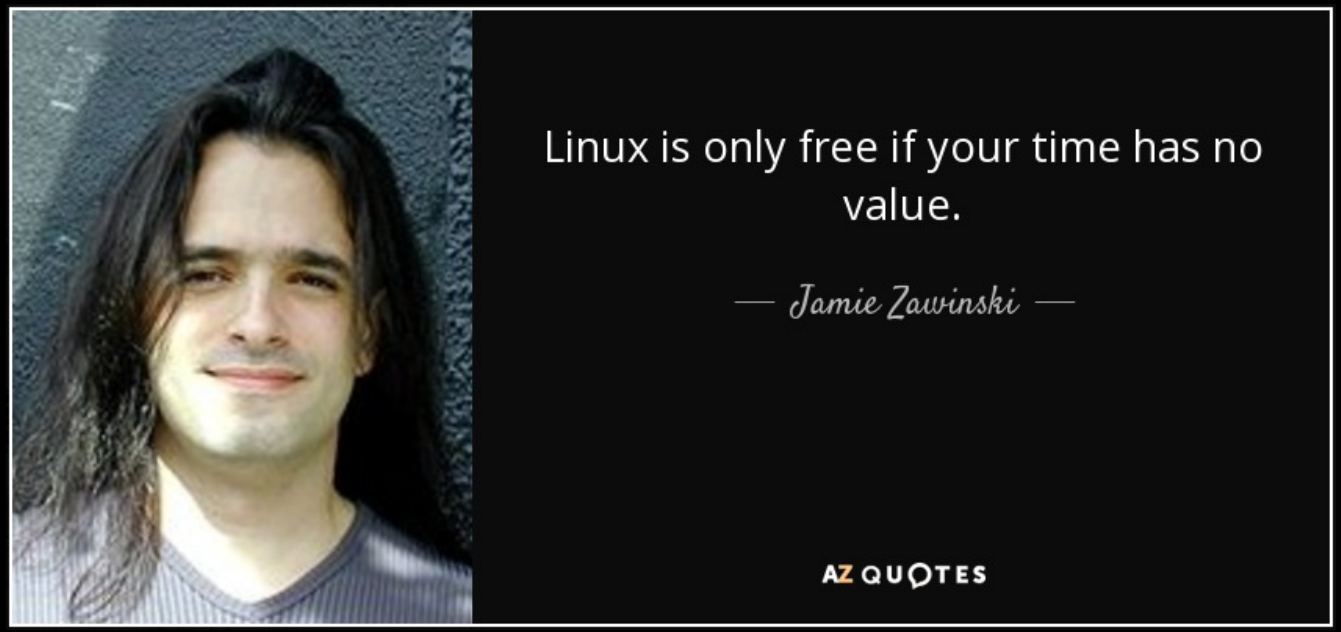
```
^[0-9]{4}-([0][0-9]|1[0-2])-([0-2][0-9]|3[01])$
```

Zagadka 11: rozwiązanie

```
^[0-9]{4}-([0][0-9]|1[0-2])-([0-2][0-9]|3[01])$
```

pasuje do dat w formacie ISO 8601

Jamie Zawinski (3.11.1968–)



Amerykański programista. Twórca przeglądarki Netscape Navigator. Właściciel klubu nocnego DNA Lounge w San Francisco. Autor wielu trafnych powiedzeń.

Netscape Navigator → Mozilla → Firefox

Tylko nie przedobrzyć

Some people, when confronted with a problem, think “I know, I’ll use regular expressions.” Now they have two problems.

Jamie Zawinski (1997)

Dokładniejszy regexp pasujący do dat ISO 8601

```
^((((19|20)(([02468][048])|([13579][26]))-02-29))|
((20[0-9][0-9])|(19[0-9][0-9]))-
((((0[1-9])|(1[0-2]))-((0[1-9])|(1[0-9])|(2[0-8])))|
((((0[13578])|(1[02]))-31)|(((0[1,3-9])|
(1[0-2]))-(29|30)))))$
```

Zagadka 12

```
^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+[A-Za-z]{2,}$
```

Zagadka 12: rozwiązanie

```
^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+[A-Za-z]{2,}$
```

pasuje do adresów email

Dokładniejszy regexp pasujący do adresów email

```
^(?:[a-z0-9!#$%&'{}*+/=?^_`{|}~-]+
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*
|"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]
|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")
@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)+
[a-z0-9](?:[a-z0-9-]*[a-z0-9])?
|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}
(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|
[a-z0-9-]*[a-z0-9]:
(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]
|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])$
```

Silniki regexpów (NFA, DFA)

- C: *third-party libraries* PCRE (*Perl Compatible Regular Expressions*), *wrapper* wokół RE2
- C++11: `std::basic_regex<char>`, `std::basic_regex<wchar_t>`, *third-party library* RE2
- Go: `import "regexp"`
- Java: `import java.util.regex.Matcher;` `import java.util.regex.Pattern;`
- Javascript: `let r = /[Rr]egexp?/;`
- Python: `import re`

Regexpy: zalety i wady

+ zwięzłość: jeden regexp może zastąpić wiele wierszy kodu
+ uniwersalność: podstawowe konstrukcje są przenośne między jęsykami
– nieczytelność bardziej złożonych regexpów
– niebezpieczeństwo nieliniowej liczby nawrotów, jeśli silnik regexpów używa NFA

Podsumowanie

Wyrażenia regularne

- konkatenacja, alternatywa, gwiazdka Kleena’a
- automaty skończone: NFA, DFA, minimalny DFA
- algorytm budowania DFA oparty na pochodnych Brzozowskiego

Regexpy

- metaznaki `[...]`, `^`, `$`, `.`
- metaznaki `?`, `+`, `{...}`
- metody w Pythonie: `.match()`, `.search()`, `.sub()`
- dostęp do grup w Pythonie: `.group()`

Do zobaczenia
na następnym wykładzie