Wyrażenia regularne

dr inż. Marcin Ciura
mgc@agh.edu.pl

Wydział Informatyki, Akademia Górniczo-Hutnicza

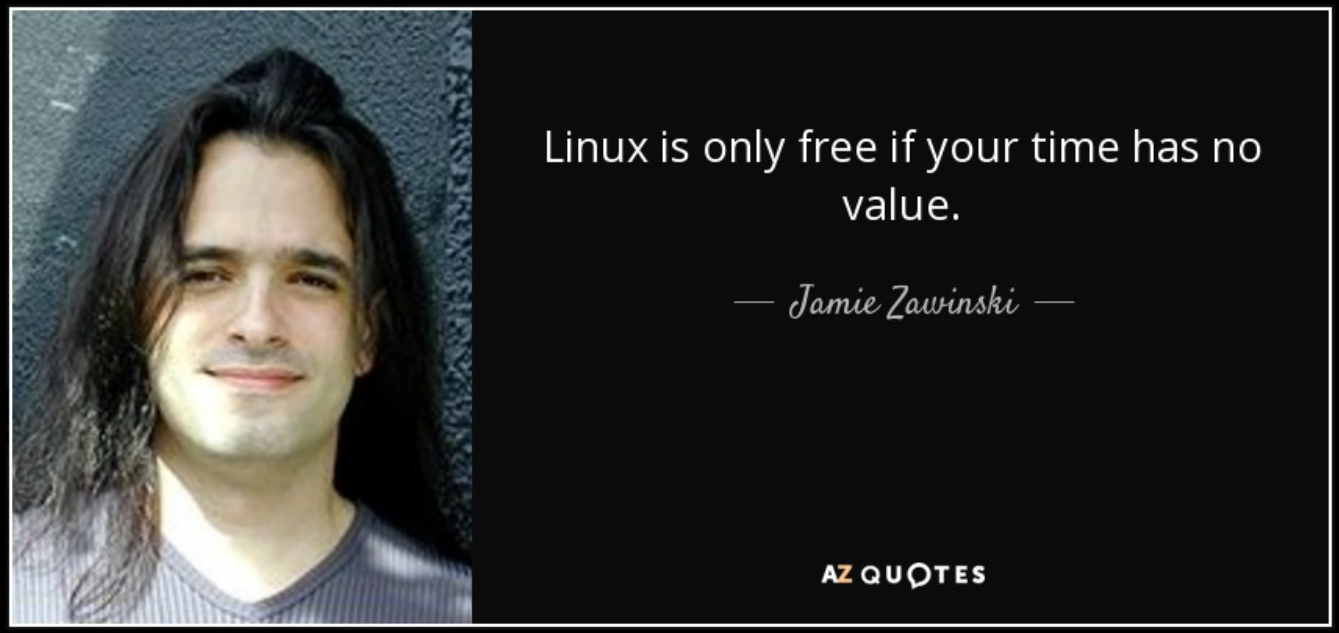
Historia przeglądarki Firefox

Firefox

Mozilla

Netscape Navigator

Jamie Zawinski (3.11.1968–) `jwz`



Amerykański programista. Twórca przeglądarki Netscape Navigator. Właściciel klubu nocnego DNA Lounge, który wiele razy zdobywał tytuł „Best Dance Club” w San Francisco.

Jamie Zawinski (3.11.1968–) `jwz`

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

Jamie Zawinski (1997)

Plan na dziś

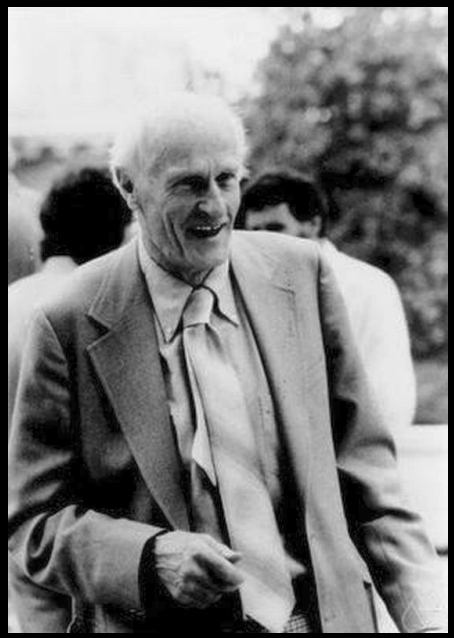
- Wyrażenia regularne
- Automaty skończone
- Regexpy
- Regexpy w Go i nie tylko

Wyrażenia regularne

Wyrażenia regularne a regex(p)y

- Wyrażenia regularne to pojęcie matematyczne
- Regexpy, znane też jako regexy, to implementacja wyrażeń regularnych w danym języku programowania wraz z dodatkami, które ułatwiają pracę programistom

Stephen Cole Kleene (5.1.1909–25.1.1994)



Amerykański matematyk, znany z pewnej gwiazdki. Stworzył teorię obliczalności. W 1951 roku, kiedy badał sieci neuronowe, wynalazł wyrażenia regularne. Wspinał się po górach, działał na rzecz ochrony przyrody.

Definicje

- Alfabet: skończony zbiór symboli czyli znaków,
  na przykład {a, b}, ASCII, Unicode...
- Łańcuch znaków, krócej łańcuch: to samo, co ciąg znaków,
  na przykład `Ala␣ma␣kota`
- Długość łańcucha: liczba znaków w tym łańcuchu. Oznaczam długość łańcucha, otaczając ten łańcuch kreskami pionowymi,
  na przykład $|\texttt{Ala␣ma␣kota}| = 11$
- Łańcuch pusty: łańcuch o długości 0 znaków. Oznaczam łańcuch pusty grecką literą epsilon: ε

Definicje

- Wyrażenie regularne to zgodny z pewnymi regułami łańcuch,
  który opisuje pewien zbiór łańcuchów
  Zbiór łańcuchów, który jest opisany przez pewne wyrażenie regularne,
  pasuje do tego wyrażenia regularnego
  Każdy łańcuch z tego zbioru też pasuje do tego wyrażenia regularnego

Elementarne wyrażenia regularne

- Do symbolu zbioru pustego Ø pasuje pusty zbiór łańcuchów: Ø
- Do symbolu pustego łańcucha ε pasuje zbiór {ε}, który zawiera tylko łańcuch pusty
- Do wyrażenia regularnego złożonego z jednego znaku pasuje zbiór, który zawiera tylko jeden łańcuch o długości 1 złożony tylko z tego znaku,
  na przykład do wyrażenia regularnego a pasuje zbiór łańcuchów {a}

Złożone wyrażenia regularne

- nawiasy
- konkatenacja wyrażeń regularnych
- alternatywa wyrażeń regularnych
- gwiazdka Kleene'a

Złożone wyrażenia regularne: nawiasy

Jeśli R jest wyrażeniem regularnym, to:

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

Złożone wyrażenia regularne: konkatenacja

Jeśli R i S są wyrażeniami regularnymi, to:

- Do wyrażenia regularnego RS, czyli do konkatenacji wyrażeń regularnych R i S, pasuje zbiór takich łańcuchów, które powstają, gdy łączę dowolny łańcuch pasujący do R z dowolnym łańcuchem pasującym do S

Złożone wyrażenia regularne: konkatenacja

Przykłady konkatenacji wyrażeń regularnych:

- Jeśli do R pasuje zbiór łańcuchów {d},
  a do S pasuje zbiór łańcuchów {o},
  to do RS pasuje zbiór łańcuchów {do}
- Jeśli do R pasuje zbiór łańcuchów {do, od},
  a do S pasuje zbiór łańcuchów {dać, pisać},
  to do RS pasuje zbiór łańcuchów {dodać, dopisać, oddać, odpisać}

Złożone wyrażenia regularne: alternatywa

Jeśli R i S są wyrażeniami regularnymi, to:

- Do wyrażenia regularnego $R|S$, czyli do alternatywy wyrażeń regularnych R i S, pasuje suma 2 zbiorów: zbioru takich łańcuchów, które pasują do R i zbioru takich łańcuchów, które pasują do S

Złożone wyrażenia regularne: alternatywa

Przykłady alternatywy wyrażeń regularnych:

- Jeśli do R pasuje zbiór łańcuchów {do},
  a do S pasuje zbiór łańcuchów {od},
  to do $R|S$ pasuje zbiór łańcuchów {do, od}
- Jeśli do R pasuje zbiór łańcuchów {cegła, dach},
  a do S pasuje zbiór łańcuchów {cecha, cegła},
  to do $R|S$ pasuje zbiór łańcuchów {cecha, cegła, dach}

Złożone wyrażenia regularne: gwiazdka Kleene'a

Jeśli R jest wyrażeniem regularnym, to:

- Do wyrażenia regularnego R^*, czyli do domknięcia Kleene'a wyrażenia regularnego R) pasuje zbiór takich łańcuchów, które powstają, gdy łączę 0 lub więcej łańcuchów pasujących do R
  Domknięcie Kleene'a nazywa się również gwiazdką Kleene'a

Złożone wyrażenia regularne: gwiazdka Kleene'a

Przykłady użycia gwiazdki Kleene'a:

- Jeśli do R pasuje zbiór łańcuchów {x}, to do R^* pasuje zbiór łańcuchów {ε, x, xx, xxx, xxxx, xxxxx, xxxxxx, xxxxxxx, ...}
- Jeśli do R pasuje zbiór łańcuchów {fa, sol}, to do R^* pasuje zbiór łańcuchów

  {ε,
  fa, sol,
  fafa, fasol, solfa, solsol,
  fafafa, fafasol, fasolfa, fasolsol,
  solfafa, solfasol, solsolfa, solsolsol,
  ...
  }

Złożone wyrażenia regularne: kolejność działań

Wykonuję działania w takiej kolejności:

- najpierw wykonuję działania w nawiasach
- potem stosuję gwiazdkę Kleene'a
- potem konkatenuję wyrażenia regularne
- potem buduję alternatywy wyrażeń regularnych

Zagadka

Które z 2 wyrażeń regularnych po prawej stronie znaku = jest równoważne wyrażeniu regularnemu po lewej stronie znaku =?

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: rozwiązanie

Które z 2 wyrażeń regularnych po prawej stronie znaku = jest równoważne wyrażeniu regularnemu po lewej stronie znaku =?

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

Zagadka

Które łańcuchy pasują do wyrażenia regularnego `(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: rozwiązanie

Które łańcuchy pasują do wyrażenia regularnego `(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

Każdy automat skończony (po angielsku: finite automaton, liczba mnoga: finite automata) może wczytywać kolejne symbole i zmieniać swój stan w zależności od tych symboli. Mówimy, że automat skończony przechodzi ze stanu do innego stanu

Liczba stanów automatu skończonego jest skończona

Automaty skończone

Każdy automat skończony można przedstawić jako taki graf skierowany, w którym:

- wierzchołki odpowiadają stanom automatu
- krawędzie odpowiadają przejściom między stanami
- każda krawędź jest oznaczona co najmniej jednym symbolem alfabetu

Automaty skończone

Zanim automat skończony wczyta jakikolwiek symbol, jest w stanie początkowym. Potem ten automat wczytuje kolejne symbole i przechodzi do kolejnych stanów.

Każdy stan, do którego przechodzi automat skończony, zależy tylko od bieżącego stanu tego automatu i od ostatniego wczytanego symbolu. Stan, do którego przechodzi automat skończony z bieżącego stanu, nie zależy od tych symboli, które ten automat wczytał, zanim przeszedł do bieżącego stanu

Kiedy automat skończony przeszedł do pewnego stanu końcowego, mówimy, że ten automat skończony rozpoznał wczytany ciąg symboli. Automat skończony może mieć wiele stanów końcowych. Stany końcowe oznacza się na diagramach kółkami z pogrubionym lub podwójnym obwodem

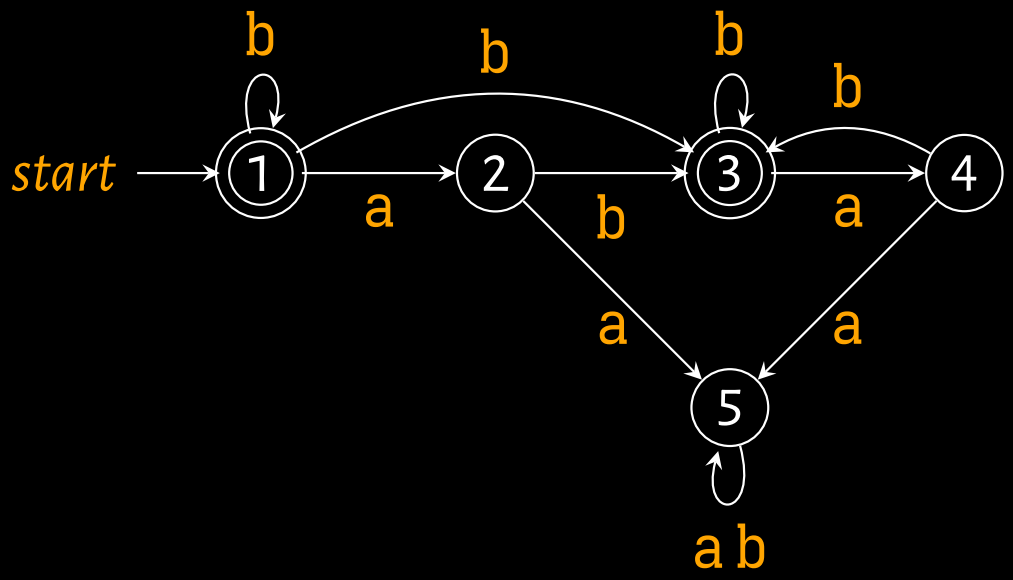
Niedeterministyczne automaty skończone

NFA (niedeterministyczny automat skończony, po angielsku: Nondeterministic Finite Automaton)

Gdy NFA znajduje się w pewnym stanie i wczytuje pewien symbol, może przejść do jednego z wielu różnych stanów

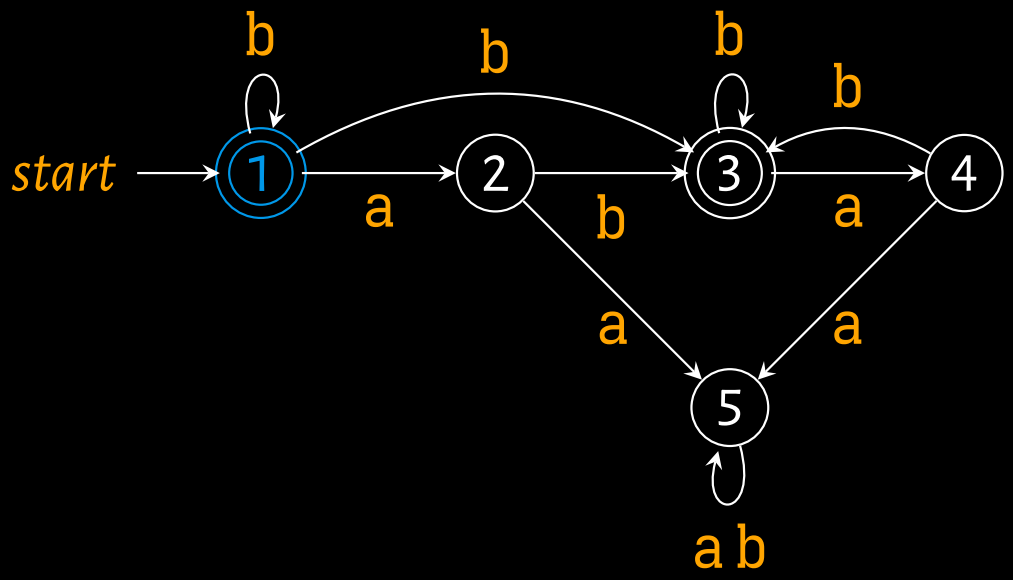
Z 1 wierzchołka grafu, który odpowiada danemu NFA, może wychodzić więcej niż 1 krawędź oznaczona tym samym symbolem

Przykład NFA



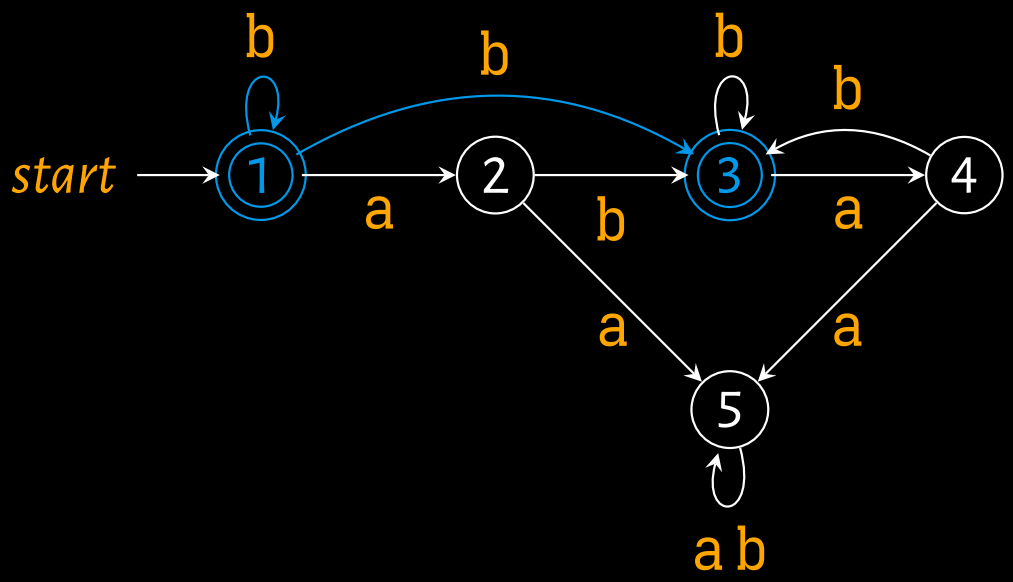
- babb

Jak NFA rozpoznaje ciąg znaków babb?



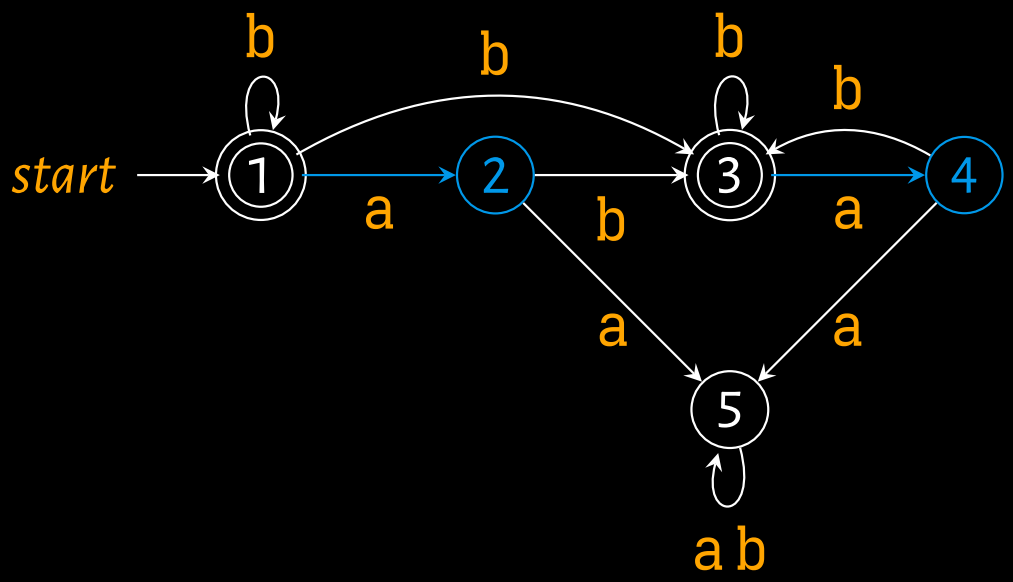
- babb

Jak NFA rozpoznaje ciąg znaków babb?



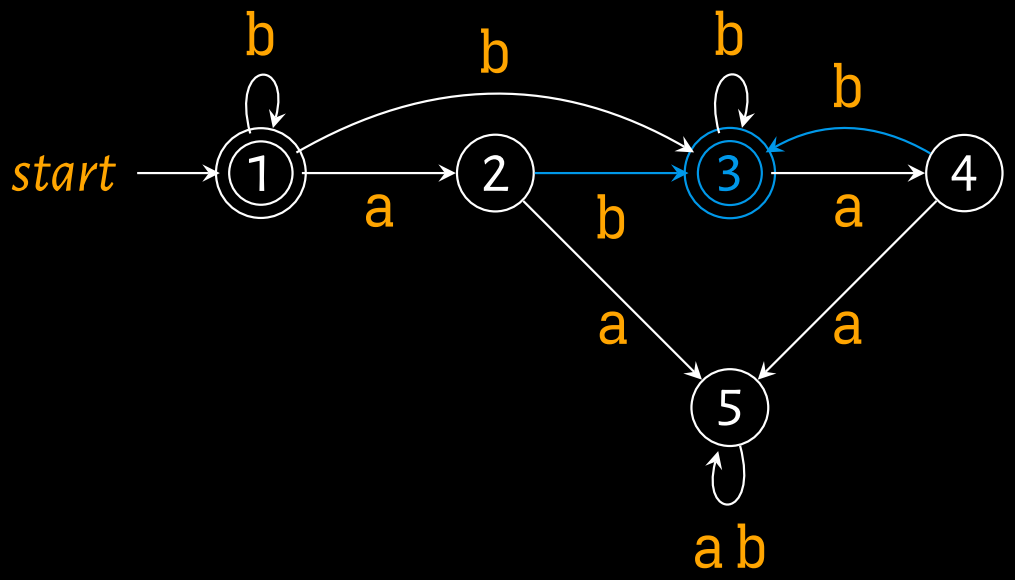
$b \cdot abb$

Jak NFA rozpoznaje ciąg znaków babb?



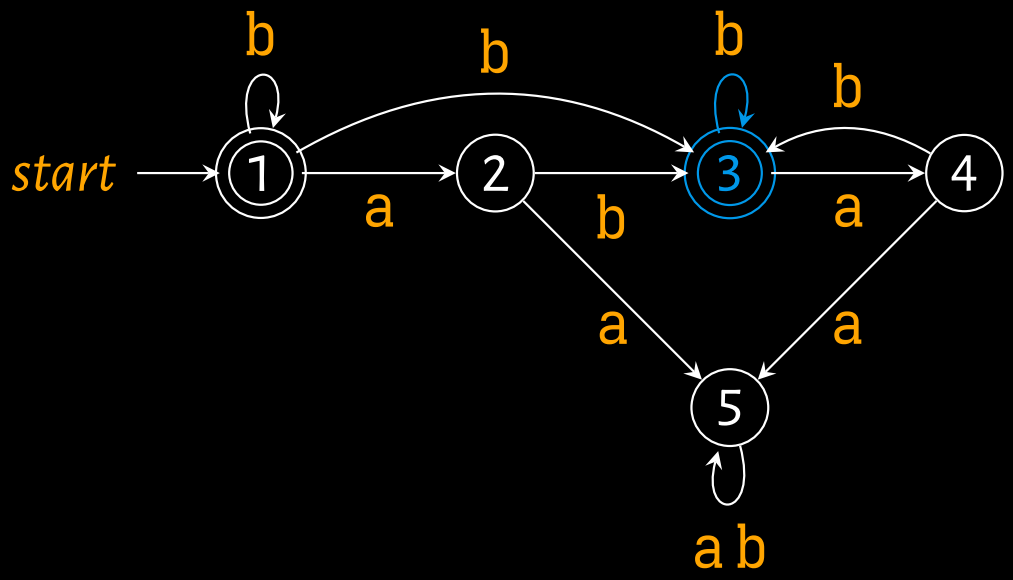
$ba \cdot bb$

Jak NFA rozpoznaje ciąg znaków babb?



bab · b

Jak NFA rozpoznaje ciąg znaków babb?



babb·

Sukces!

Automat zatrzymał się w stanie końcowym

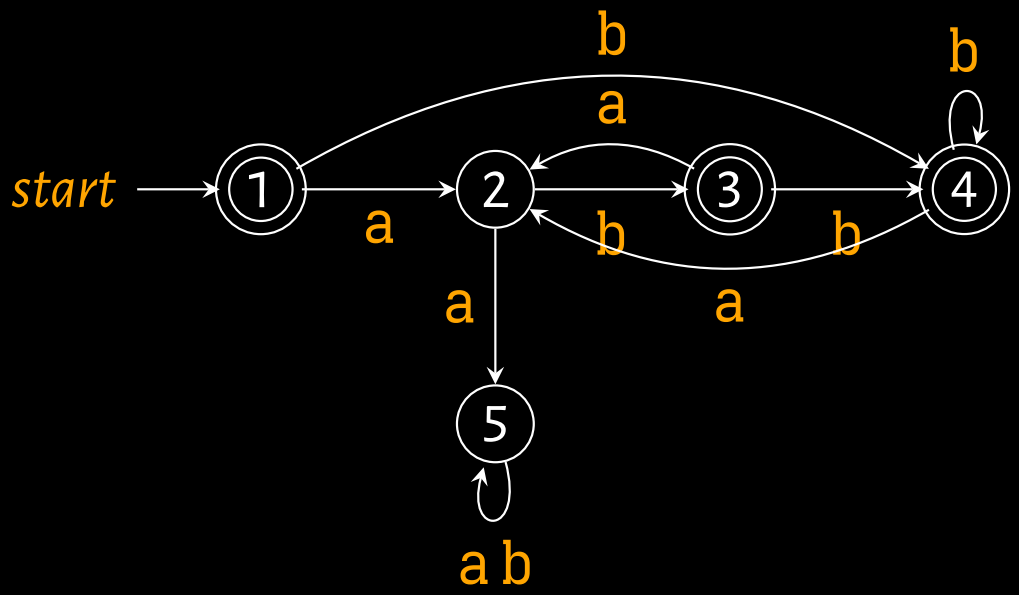
Deterministyczne automaty skończone

DFA (deterministyczny automat skończony, po angielsku: Deterministic Finite Automaton)

Gdy DFA znajduje się w pewnym stanie i wczytuje pewien symbol, zawsze przechodzi do określonego stanu

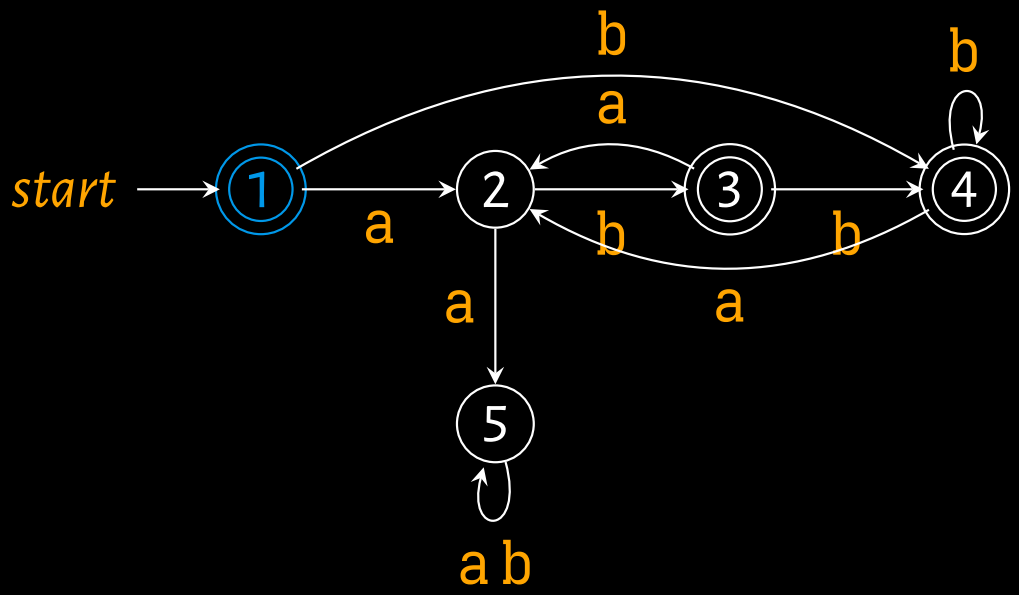
Z 1 wierzchołka grafu, który odpowiada danemu DFA, wychodzi dokładnie 1 krawędź oznaczona danym symbolem

Przykład DFA



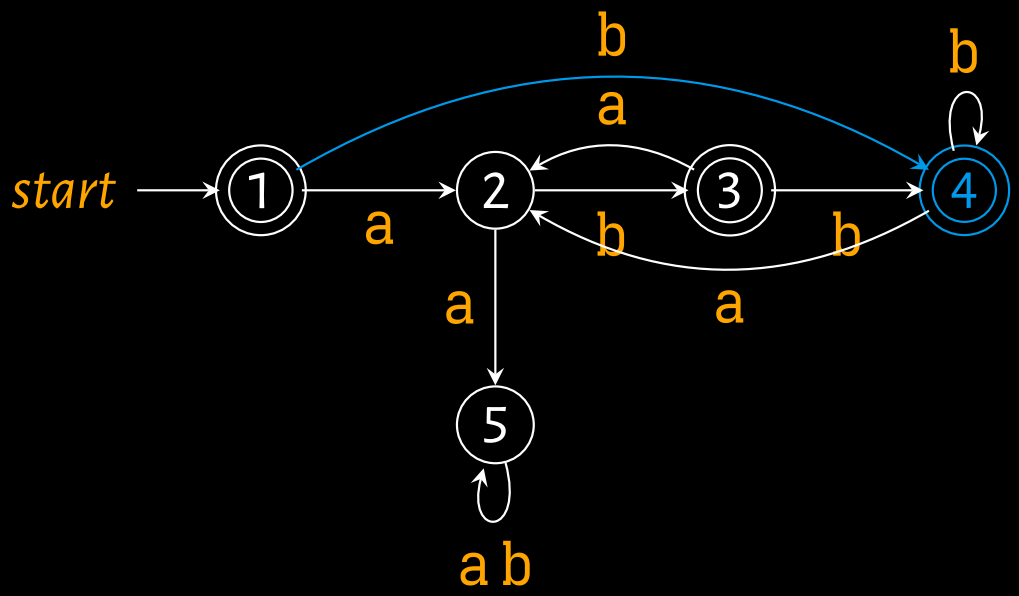
- babb

Jak DFA rozpoznaje ciąg znaków babb?



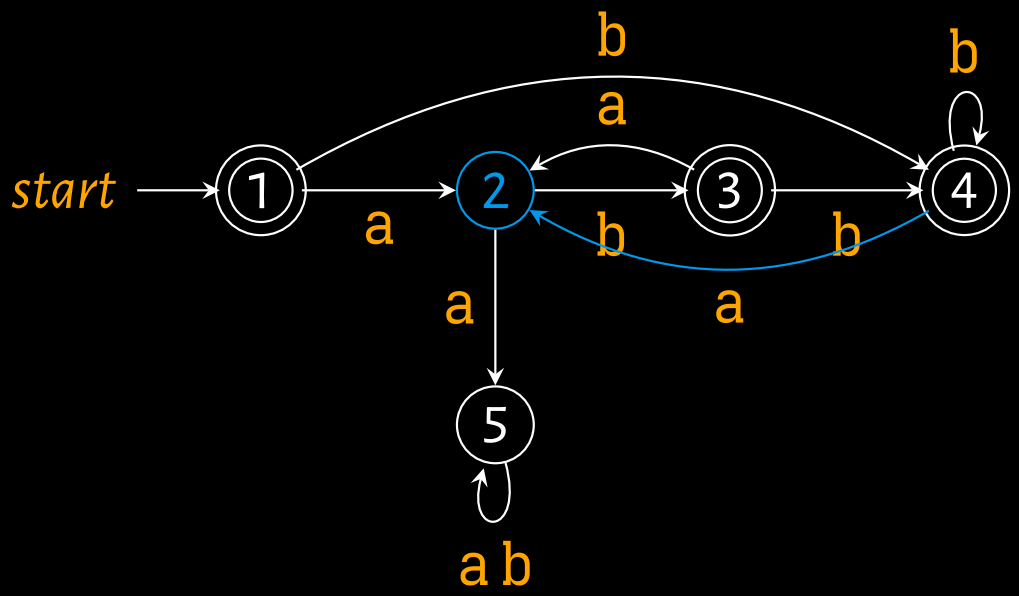
- babb

Jak DFA rozpoznaje ciąg znaków babb?



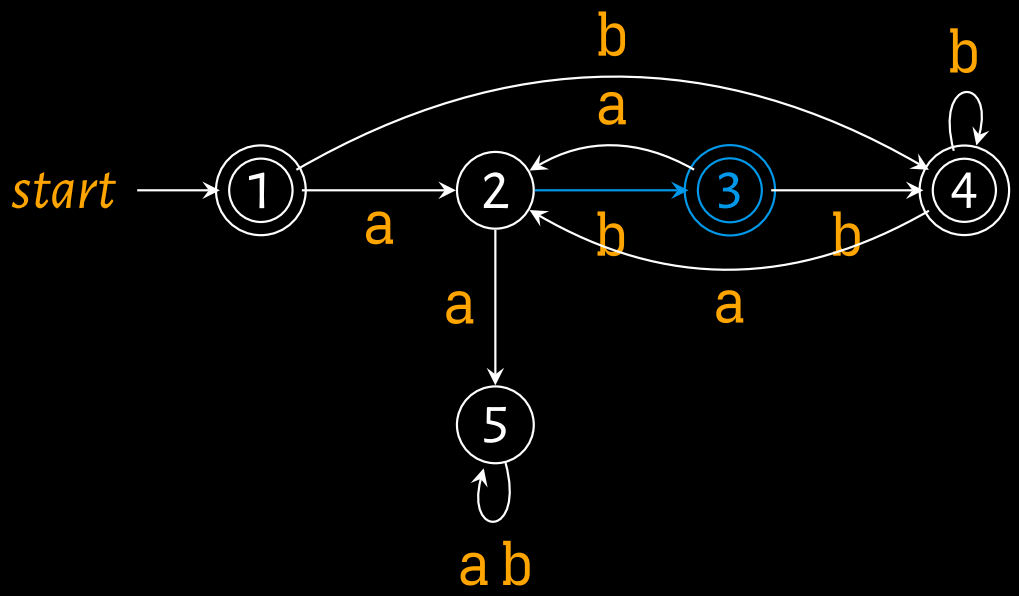
b · abb

Jak DFA rozpoznaje ciąg znaków babb?



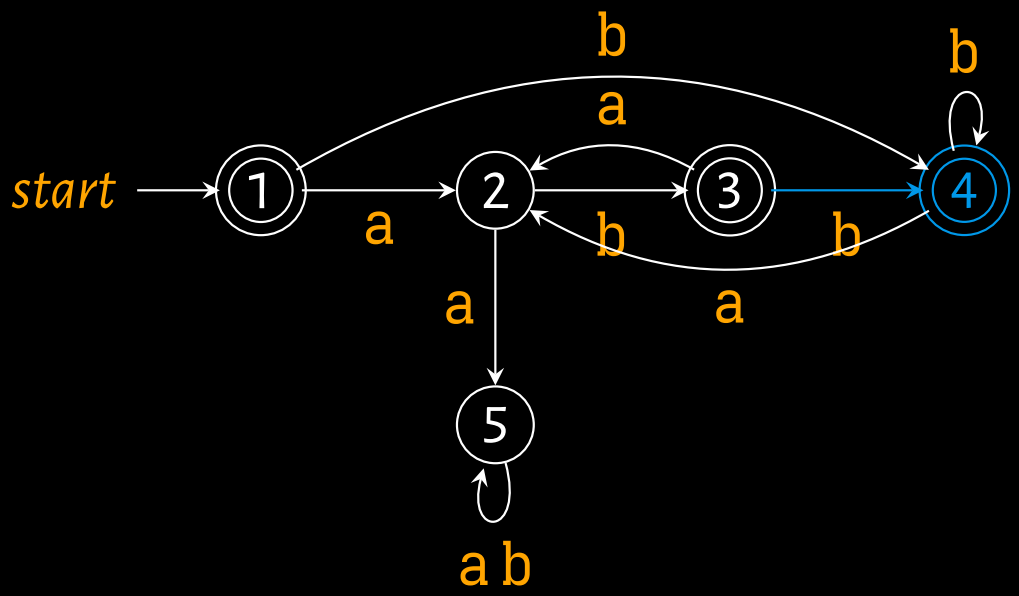
ba · bb

Jak DFA rozpoznaje ciąg znaków babb?



bab · b

Jak DFA rozpoznaje ciąg znaków babb?

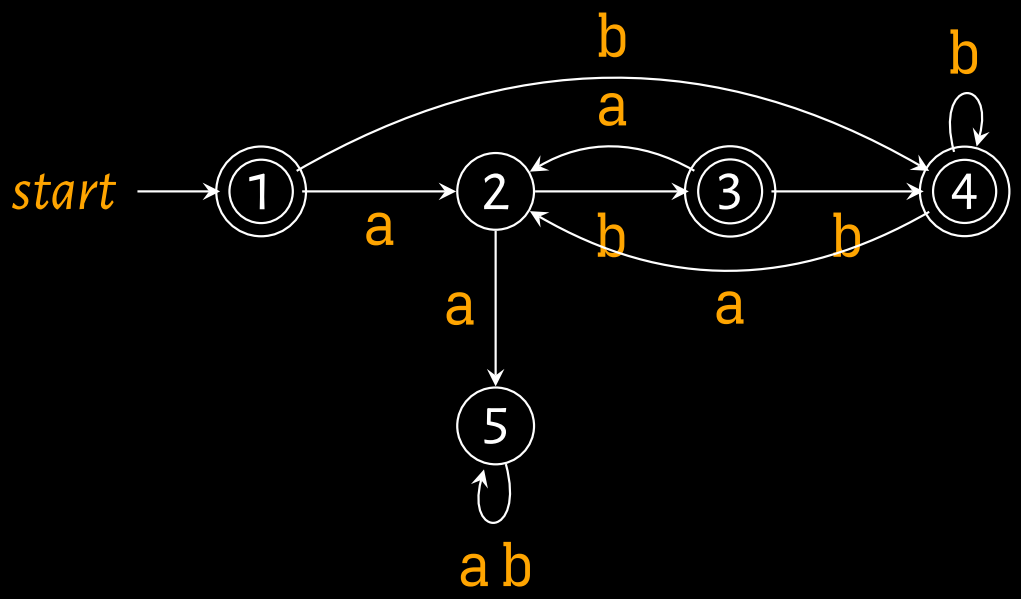


babb·

Sukces!

Automat zatrzymał się w stanie końcowym

DFA: tabela przejść i wyjść

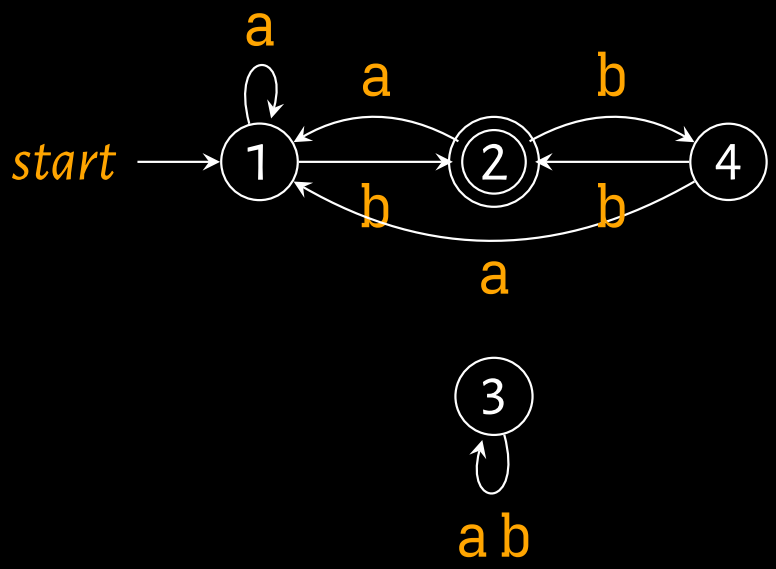


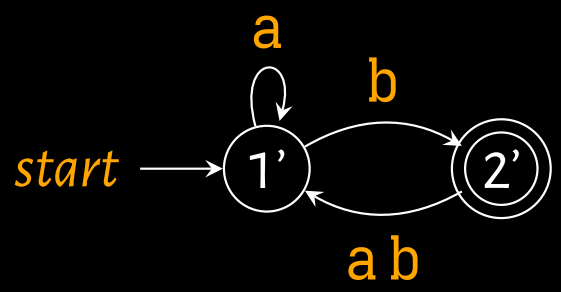
stan	końcowy?	a	b
1	true	2	4
2	false	3	5
3	true	2	4
4	true	2	4
5	false	5	5

Równoważne automaty skończone

Mówimy, że 2 automaty skończone są równoważne, jeśli rozpoznają te same zbiory łańcuchów

Przykład 2 równoważnych automatów skończonych:





Minimalny DFA

W minimalnym DFA nie ma żadnych zbędnych stanów, czyli:

- takich stanów, które można połączyć w jeden stan bez zmiany zbioru łańcuchów, które rozpoznaje ten DFA
- takich stanów, do których nie można dotrzeć ze stanu początkowego

Jak dopasować łańcuch do wyrażenia regularnego?

Aby dopasować łańcuch do danego wyrażenia regularnego, można:

- Bezpośrednio interpretować to wyrażenie regularne
- Zbudować NFA, który odpowiada temu wyrażeniu regularnemu (na przykład algorytmem Thompsona), po czym:
  - przechodzić przez ten NFA, pamiętając bieżący zbiór stanów
  - przechodzić przez ten NFA z nawrotami (po angielsku:backtracking)
  - zbudować DFA równoważny temu NFA
- Od razu zbudować DFA, który odpowiada temu wyrażeniu regularnemu, korzystając z:
  - algorytmu Myhilla-Nerode'a
  - algorytmu DeRemera
  - pochodnych Brzozowskiego

Pochodne Brzozowskiego

Janusz Brzozowski (10.5.1935–24.10.2019)

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

Pochodna Brzozowskiego zbioru łańcuchów

Dane:

- zbiór łańcuchów L
- symbol a

Szukane:

- $\partial_a L$, pochodna Brzozowskiego zbioru łańcuchów L względem symbolu a

Tak obliczam pochodną Brzozowskiego $\partial_a L$:

- znajduję w zbiorze L wszystkie takie łańcuchy, które zaczynają się od symbolu a
- odcinam pierwszy symbol a od każdego ze znalezionych łańcuchów

Zagadka

Jakimi zbiorami łańcuchów są pochodne Brzozowskiego tych zbiorów łańcuchów względem danych symboli?

$\partial_{\texttt{w}}\{\texttt{się}, \texttt{i}, \texttt{w}, \texttt{wszystko}, \texttt{więc}\} = ?$

$\partial_{\texttt{n}}\{\varepsilon, \texttt{nie}, \texttt{na}, \texttt{o}\} = ?$

$\partial_{\texttt{z}}\{\varepsilon, \texttt{z}, \texttt{za}, \texttt{abrakadabra}\} = ?$

Zagadka: rozwiązanie

Jakimi zbiorami łańcuchów są pochodne Brzozowskiego tych zbiorów łańcuchów względem danych symboli?

$\partial_{\texttt{w}}\{\texttt{się}, \texttt{i}, \texttt{w}, \texttt{wszystko}, \texttt{więc}\} = \{\varepsilon, \texttt{szystko}, \texttt{ięc}\}$

$\partial_{\texttt{n}}\{\varepsilon, \texttt{nie}, \texttt{na}, \texttt{o}\} = \{\texttt{ie}, \texttt{a}\}$

$\partial_{\texttt{z}}\{\varepsilon, \texttt{z}, \texttt{za}, \texttt{abrakadabra}\} = \{\varepsilon, \texttt{a}\}$

Pochodna Brzozowskiego wyrażeń regularnych

Dane jest wyrażenie regularne R, do którego pasuje pewien zbiór łańcuchów L

Pochodna Brzozowskiego wyrażenia regularnego R względem symbolu a to takie wyrażenie regularne, do którego pasuje pochodna Brzozowskiego zbioru łańcuchów L względem symbolu a, czyli $\partial_a L$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$$\partial_a \emptyset = ?$$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = ?$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = \emptyset$

$\partial_a a = ?$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = \emptyset$

$\partial_a a = \varepsilon$

$\partial_a b = ?$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = \emptyset$

$\partial_a a = \varepsilon$

$\partial_a b = \emptyset$

$\partial_a aR = ?$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = \emptyset$

$\partial_a a = \varepsilon$

$\partial_a b = \emptyset$

$\partial_a aR = R$

$\partial_a bR = ?$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$\partial_a \emptyset = \emptyset$

$\partial_a \varepsilon = \emptyset$

$\partial_a a = \varepsilon$

$\partial_a b = \emptyset$

$\partial_a aR = R$

$\partial_a bR = \emptyset$

$\partial_a (R|S) = ?$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$$\partial_a \emptyset = \emptyset$$

$$\partial_a \varepsilon = \emptyset$$

$$\partial_a a = \varepsilon$$

$$\partial_a b = \emptyset$$

$$\partial_a aR = R$$

$$\partial_a bR = \emptyset$$

$$\partial_a (R|S) = \partial_a(R)|\partial_a(S)$$

$$\partial_a R^* = ?$$

Pochodna Brzozowskiego wyrażeń regularnych

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem symbolu a?

$$\partial_a \emptyset = \emptyset$$

$$\partial_a \varepsilon = \emptyset$$

$$\partial_a a = \varepsilon$$

$$\partial_a b = \emptyset$$

$$\partial_a aR = R$$

$$\partial_a bR = \emptyset$$

$$\partial_a (R \mid S) = \partial_a (R) \mid \partial_a (S)$$

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

Zagadka

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem danych symboli?

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

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

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

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

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

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

Zagadka: rozwiązanie

Jakimi wyrażeniami regularnymi są pochodne Brzozowskiego tych wyrażeń regularnych względem danych symboli?

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

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

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

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

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

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

Algorytm 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)*`

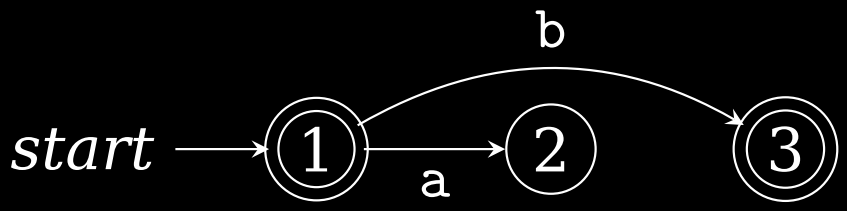
Algorytm Brzozowskiego: przykład (2)

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)



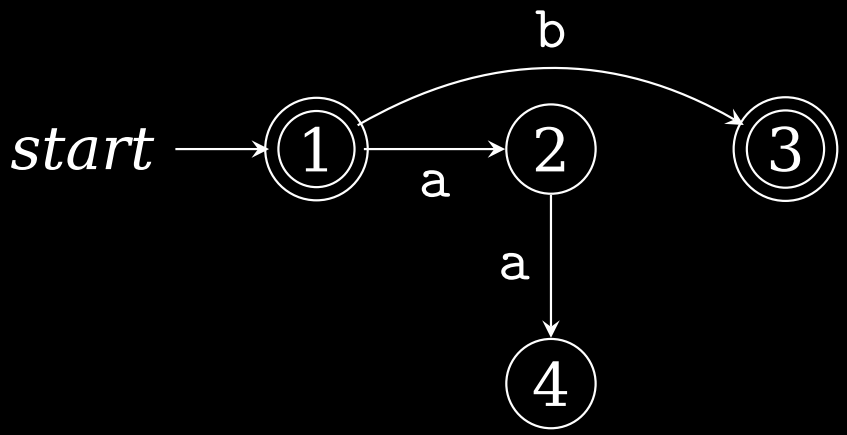
1: `(b*(a|ε)b)*`

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)*`

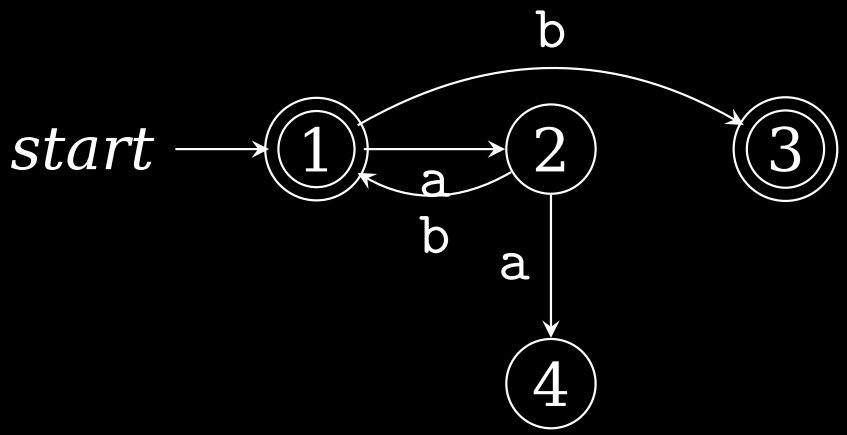
2: `b(b*(a|ε)b)*`

3: `b*(b*(a|ε)b)*`

4: Ø

∂_a`b(b*(a|ε)b)*` = Ø

Algorytm Brzozowskiego: przykład (5)



1: `(b*(a|ε)b)*`

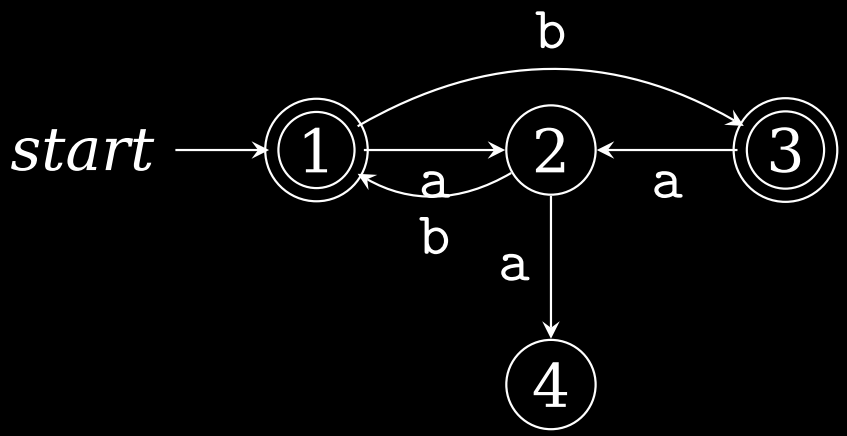
2: `b(b*(a|ε)b)*`

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)*`

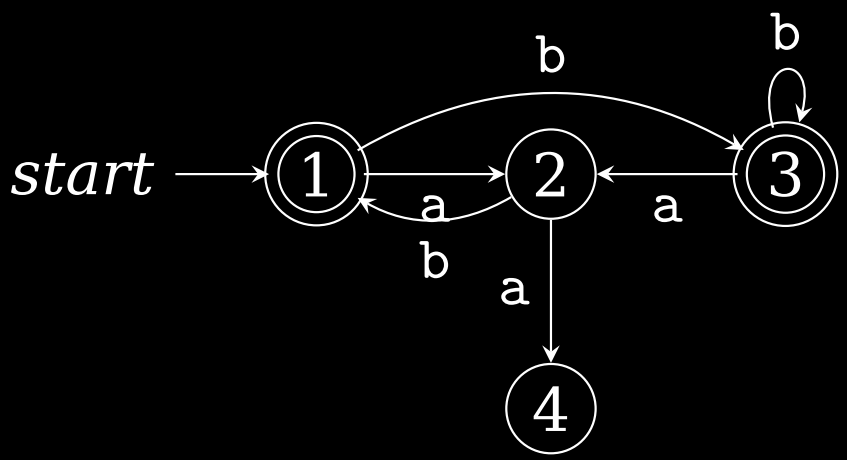
2: `b(b*(a|ε)b)*`

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)*`

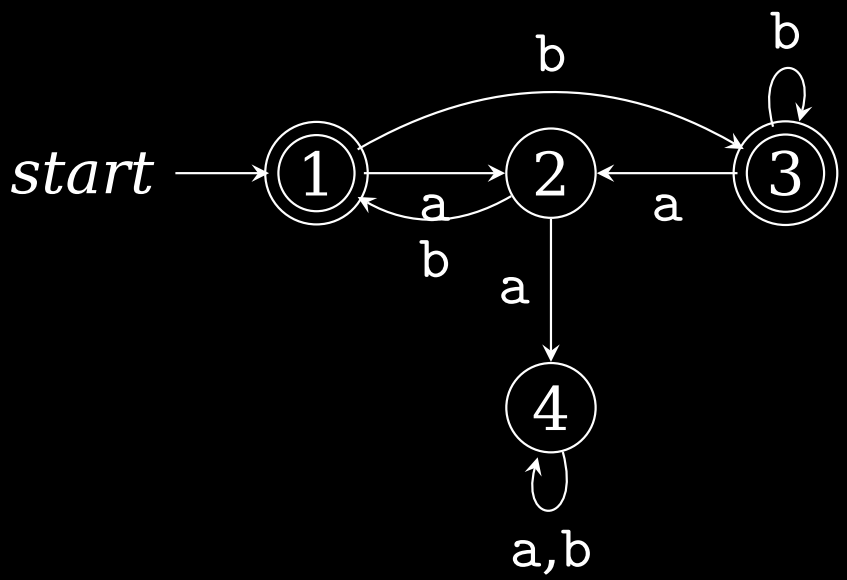
2: `b(b*(a|ε)b)*`

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

Regexpy, znane też jako regexy, to implementacja wyrażeń regularnych w danym języku programowania wraz z dodatkami, które ułatwiają pracę programistom

Tę część wykładu opracowałem na podstawie dokumentacji pakietu `regexp/syntax`

Dokumentacja tego pakietu znajduje się pod adresem `https://pkg.go.dev/regexp/syntax` i na dysku lokalnym

Gdy wydaję polecenie `go doc regexp/syntax`, widzę ją na ekranie

Składnia regexpów

- Jak rozpoznawać początek i koniec tekstu?
- Jak dopasować 1 znak do regexpa?
- Jak składać regexpy?
- Jak powtarzać regexpy?
- Jak grupować regexpy?

Składnia regexpów: jak rozpoznawać początek i koniec tekstu?

Początek tekstu pasuje do znaku daszka `^`

Koniec tekstu pasuje do znaku dolara `$`

Regexp `^kot|pies$` jest równoważny regexpowi `(^kot)|(pies$)`
Taki regexp, do którego pasują tylko łańcuchy `kot` i `pies`, to `^(kot|pies)$`

Znaki daszka i dolara pasują do regexpów `\^` i `\$`

Składnia regexpów: jak dopasować 1 znak do wyrażenia regularnego?

`.`	dowolny znak
`[xyz]`	klasa znaków
`[^xyz]`	zanegowana klasa znaków
`\d`	klasa znaków w stylu Perla
`\D`	zanegowana klasa znaków w stylu Perla
`\pN`	klasa znaków Unicode (1-literowa nazwa)
`\p{Greek}`	klasa znaków Unicode (dłuższa nazwa)
`\PN`	zanegowana klasa znaków Unicode'u (1-literowa nazwa)
`\P{Greek}`	zanegowana klasa znaków Unicode'u (dłuższa nazwa)

Składnia regexpów: jak dopasować 1 znak do wyrażenia regularnego?

Wiele znaków wewnątrz regexpów oznacza te same znaki tekstu,
na przykład do regexpa a pasuje znak a

Takie znaki tekstu, które mają specjalne znaczenie, gdy występują wewnątrz regexpów, trzeba wewnątrz regexpów poprzedzać znakiem odwrotnego ukośnika \
Na przykład znak * pasuje do regexpa *

„Syndrom pochylonych wykałaczek”

Po angielsku: Leaning Toothpick Syndrome

Gdy do regexpa ma pasować znak odwrotnego ukośnika \ w tekście, wewnątrz regexpa trzeba podwoić ten znak

W wielu językach programowania znaki odwrotnego ukośnika wewnątrz stałych łańcuchowych trzeba znowu podwajać

„Syndrom pochylonych wykałaczek”

Na przykład łańcuch `C:\` pasuje do regexpa zapisanego w tej stałej łańcuchowej:

```
"^[A-Za-z]:\\\\$"
```

W Go mogę używać stałych łańcuchowych otoczonych znakami odwrotnego apostrofu `` ` ``. Wewnątrz takich stałych łańcuchowych odwrotny ukośnik nie jest znakiem specjalnym. Dlatego wewnątrz stałych otoczonych znakami odwrotnego apostrofu nie podwajam znaku odwrotnego ukośnika `\`:

```
`^[A-Za-z]:\\$`
```

Składnia regexpów: jak dopasować dowolny znak do wyrażenia regularnego?

Do znaku kropki `.` pasuje dowolny znak oprócz znaku nowego wiersza

Znak kropki pasuje do regexpa `\.` lub `[.]`

Dygresja: grep

`grep` (global regular expression print) to program, który wyszukuje w plikach takie wiersze, które pasują do danego wyrażenia regularnego

Składnia regexpów: jak dopasować dowolny znak do wyrażenia regularnego? – przykład

Gdy wydałem polecenie `grep ^d.m$ slowa.txt`, widzę taki wynik:

```
dam
dom
dum
dym
```

Składnia regexpów: klasy znaków

Nawiasy kwadratowe `[` i `]` otaczają klasy znaków

Wewnątrz klasy znaków:

- mogę wymieniać pojedyncze znaki,
  na przykład regexp `[ąę]` jest równoważny regexpowi `(ą|ę)`
- mogę definiować przedziały znaków, podając 2 znaki połączone znakiem minusa `-`,
  na przykład regexp `[A-D]` jest równoważny regexpowi `(A|B|C|D)`
- takie znaki, które mają specjalne znaczenie poza klasami znaków, oznaczają zwykłe znaki,
  na przykład regexp `[(.*)]` jest równoważny regexpowi `\(|\.|\*|\)`

Składnia regexpów: klasy znaków – przykład

Do regexpa `[A-ZĄĆĘŁŃÓŚŹŻa-ząćęłńóśźż]` pasują pojedyncze litery alfabetu polskiego wraz z literami `q`, `v` i `x`

Składnia regexpów: klasy znaków – przykład

Zwykle używam przyimka „w”

Używam przyimka „we” przed takimi wyrazami, które zaczynają się dwiema spółgłoskami, z których pierwsza wymawiana spółgłoska to „f” lub „w”, na przykład:

we Francji, we Wrocławiu, we foyer

Ponadto używam przyimka „we” przed tymi wyrazami:

we dwoje, we troje, we czworo
we mnie
we Lwowie, we łbie, we łbach, we łzach, we mgle, we mgłach, we śnie

Składnia regexpów: klasy znaków – przykład

Używam przyimka „we” przed takimi wyrazami, które pasują do tego regexpa:

```
^([fvw]([bcdfghjkllłmnprstwzż]|o[iy])
|dwoje$|troje$|czworo$|mnie$
|lwow|łb|łz|mg[lł]|śn)
```

Składnia regexpów: zanegowane klasy znaków

Wewnątrz regexpa mogę definiować zanegowane klasy znaków.
Zanegowane klasy znaków oznaczają takie znaki, które nie należą do pewnego zbioru znaków

Pierwszy znak zanegowanej klasy znaków to znak daszka ^

Składnia regexpów: zanegowane klasy znaków

Przykłady:

Do regexpa `[^()]` pasuje dowolny znak oprócz znaku nawiasu otwierającego `(` i znaku nawiasu zamykającego `)`

Do regexpa `[^A-Za-z]` pasuje dowolny znak oprócz liter alfabetu angielskiego

Składnia regexpów: znaki specjalne `^]-` wewnątrz klas znaków

Znak daszka `^` pasuje:

- do części regexpa `[\^...]`
- do części regexpa `[...^...]`

Składnia regexpów: znaki specjalne `^]-` wewnątrz klas znaków

Znak nawiasu kwadratowego zamykającego `]` pasuje:

- do części regexpa `[...\]...]`
- do części regexpa `[]...]`

Składnia regexpów: znaki specjalne `^]-` wewnątrz klas znaków

Znak minusa `-` pasuje:

- do części regexpa `[...\-...]`
- do części regexpa `[-...]`
- do części regexpa `[...-]`

Składnia regexpów: klasy znaków w stylu Perla

Klasy znaków w stylu Perla oznaczają zbiory znaków ASCII:

- `\d` cyfra (== `[0-9]`)
- `\D` nie cyfra (== `[^0-9]`)
- `\s` biały znak (== `[\t\n\f\r ]`)
- `\S` nie biały znak (== `[^\t\n\f\r ]`)
- `\w` znak identyfikatorów z czasów ASCII (== `[0-9A-Za-z_]`)
- `\W` nie znak identyfikatorów z czasów ASCII (== `[^0-9A-Za-z_]`)

Składnia regexpów: klasy znaków Unicode'u

Jednoliterowe nazwy klas znaków Unicode oznaczają kategorie znaków

- `\pC` znak sterujący lub formatujący, np. `\t CHARACTER TABULATION`
- `\PC` dowolny znak oprócz znaków sterujących i formatujących
- `\pL` znak litery dowolnego systemu pisma
- `\PL` dowolny znak oprócz liter
- `\pM` znak akcentu itp., np. ◌́ `COMBINING ACUTE ACCENT`
- `\PM` dowolny znak oprócz znaków z kategorii `M`
- `\pN` cyfra dowolnego systemu pisma
- `\PN` dowolny znak oprócz cyfr
- `\pP` znak przestankowy, np. `? QUESTION MARK`
- `\PP` dowolny znak oprócz znaków przestankowych
- `\pS` symbol, np. symbol waluty, symbol matematyczny
- `\PS` dowolny znak, który nie jest symbolem według Unicode'u
- `\pZ` tak zwany znak podziału, np. ␣ `SPACE`
- `\PZ` dowolny znak, który nie jest znakiem podziału

Składnia regexpów: klasy znaków Unicode'u

2-literowe nazwy klas znaków Unicode'u oznaczają bardziej szczegółowe kategorie znaków niż 1-literowe nazwy klasy

Przykłady:

`\p{Ll}`	mała litera dowolnego systemu pisma
`\P{Ll}`	dowolny znak oprócz małych liter
`\p{Lu}`	duża litera dowolnego systemu pisma
`\P{Lu}`	dowolny znak oprócz dużych liter

Gdy wydaję polecenie `go doc unicode.Categories`, widzę pełną listę kategorii znaków

Składnia regexpów: klasy znaków Unicode'u

Wieloliterowe nazwy klas znaków Unicode'u oznaczają systemy pisma

Przykłady:

`\p{Latin}`	znak pisma łacińskiego
`\P{Latin}`	dowolny znak spoza pisma łacińskiego
`\p{Cyrillic}`	znak cyrylicy
`\P{Cyrillic}`	dowolny znak spoza cyrylicy
`\p{Han}`	znak pisma chińskiego
`\P{Han}`	dowolny znak oprócz znaków pisma chińskiego

Gdy wydaję polecenie `go doc unicode.Scripts`, widzę pełną listę systemów pisma

Składnia regexpów: jak składać regexpy?

Mogę konkatenować regexpy i budować alternatywy regexpów

`xy`	`x`, a po nim `y`
`x|y`	`x` lub `y` (preferuj `x`)

Składnia regexpów: jak powtarzać regexpy?

Tak mogę powtarzać regexpy:

`x*`	0 lub więcej powtórzeń `x`, preferuj więcej
`x+`	1 lub więcej powtórzeń `x`, preferuj więcej
`x?`	0 lub 1 powtórzeń `x`, preferuj 1
`x{n,m}`	n lub n+1 lub ... lub m powtórzeń `x`, preferuj więcej
`x{n,}`	n lub więcej powtórzeń `x`, preferuj więcej
`x{n}`	dokładnie n powtórzeń `x`
`x*?`	0 lub więcej powtórzeń `x`, preferuj mniej
`x+?`	1 lub więcej powtórzeń `x`, preferuj mniej
`x??`	0 lub 1 powtórzeń `x`, preferuj 0
`x{n,m}?`	n lub n+1 lub ... lub m powtórzeń `x`, preferuj mniej
`x{n,}?`	n lub więcej powtórzeń `x`, preferuj mniej

Zagadka

Jakie łańcuchy pasują do tego regexpa?

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

Zagadka: rozwiązanie i nowa zagadka

Do tego regexpa pasują liczby rzymskie, na przykład `MMXXIV`, ale nie tylko

Znajdź bug :-)

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

Zagadka: rozwiązanie

Do tego regexpa pasują liczby rzymskie, na przykład `MMXXIV`, ale pasuje do niego również łańcuch pusty:

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

Składnia regexpów: jak grupować regexpy?

Gdy tekst pasuje do regexpa, mogę pobrać te części tekstu, które pasują do części regexpa otoczonych nawiasami okrągłymi (i)

Takie części tekstu to poddopasowania

Poddopasowanie z numerem 0 odpowiada całemu tekstowi

Poddopasowania z numerami 1,... odpowiadają kolejnym częściom regexpa otoczonym nawiasami okrągłymi (i). Takie części regexpa to grupy

Zagadka

Jakie łańcuchy pasują do tego regexpa?

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

Zagadka: rozwiązanie

Do tego regexpa pasują daty w formacie ISO 8601:

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

Poddopasowanie 0 to cała data, np. `1791-05-03`

Poddopasowanie 1 to rok, np. `1791`

Poddopasowanie 2 to miesiąc, np. `05`

Poddopasowanie 3 to dzień, np. `03`

Zagadka: rozwiązanie

Do tego regexpa pasują daty w formacie ISO 8601:

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

Pasują do niego również niepoprawne daty, np. `2024-02-31`

Jamie Zawinski (3.11.1968–) `jwz`

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

Jamie Zawinski (1997)

Now they have two problems

Ten regexp opisuje tylko poprawne daty w formacie 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

Jakie łańcuchy pasują do tego regexpa?

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

Zagadka: rozwiązanie

Do tego regexpa pasują adresy email:

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

Now they have two problems

Ten regexp opisuje poprawne adresy 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])?
|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)){3}
(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|
[a-z0-9-]*[a-z0-9]:
(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]
|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])$
```

Regexpy w różnych językach programowania

- Go – `import "regexp"` (DFA)
- C – zewnętrzne biblioteki: PCRE (Perl-Compatible Regular Expressions) (NFA), biblioteka opakowująca RE2 (DFA)
- C++11 – `std::basic_regex<char>`, `std::basic_regex<wchar_t>` (NFA), zewnętrzna biblioteka RE2 (DFA)
- Java – `import java.util.regex.Matcher;` `import java.util.regex.Pattern;` (NFA)
- Javascript – `let r = /[Rr]egexp?/;` (NFA)
- Python – `import re` (NFA)

Różnice między regexpami opartymi na DFA a regexpami opartymi na NFA

Regexpy w Go są oparte na DFA

W regexpach opartych na NFA można korzystać z odwołań wstecznych (po angielsku: backreferences)

Takie programy, które korzystają z regexpów opartych na NFA i przyjmują regexpy lub łańcuchy od użytkowników, można zablokować (blokada usług, po angielsku: DoS czyli Denial of Service)

Regexpy oparte na NFA: odwołania wsteczne

Do odwołań wstecznych \1...\99 pasuje ten sam łańcuch, który pasuje do pewnej grupy o odpowiednim numerze kolejnym. Ta grupa musi poprzedzać odwołanie wsteczne

W Go nie ma odwołań wstecznych

Regexpy oparte na NFA: odwołania wsteczne – przykład w Pythonie

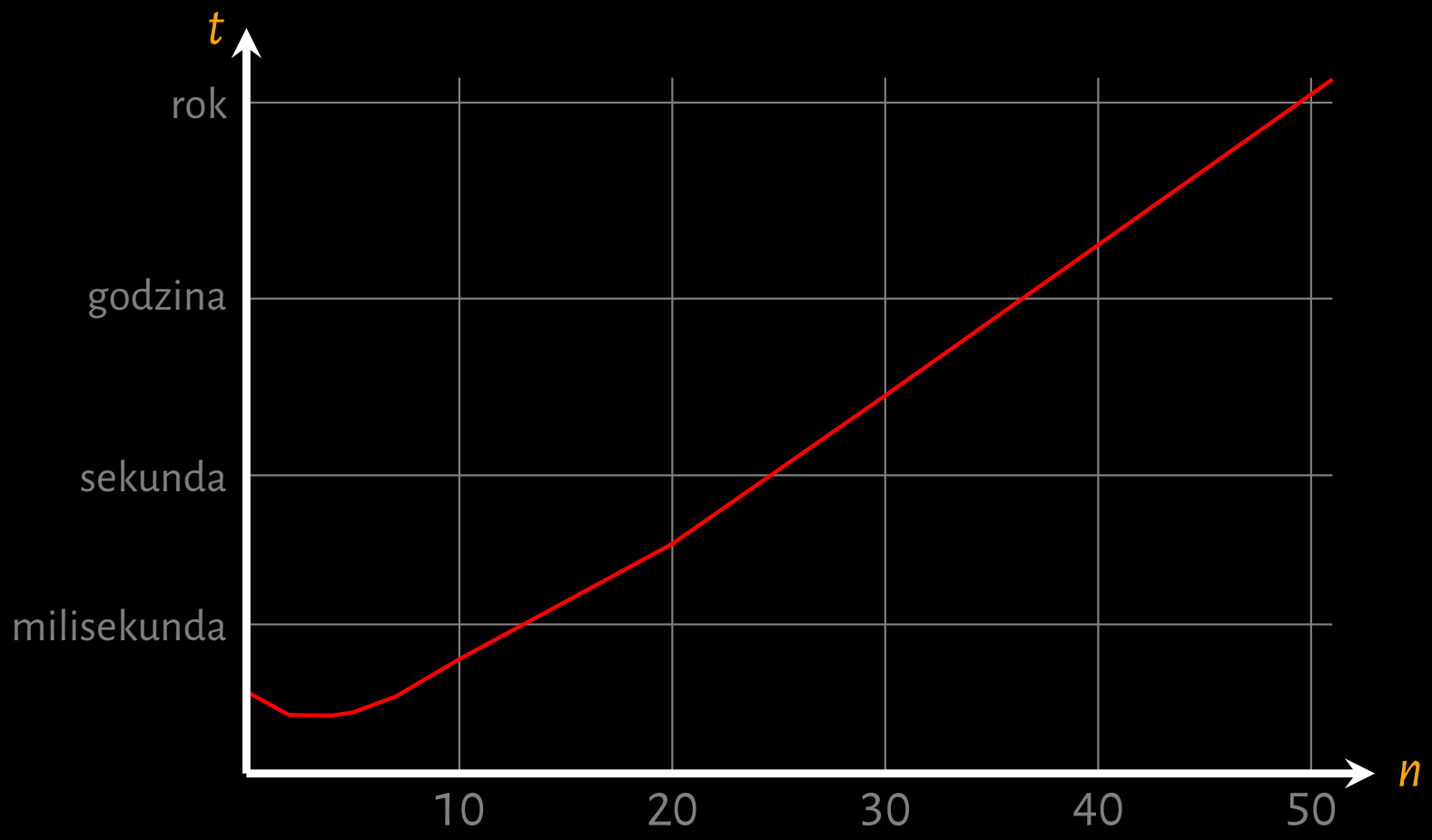
```
import re

r = re.compile(r'^(.+)\1$', re.MULTILINE)

with open('slowa.txt') as f:
  słowa = f.read()
print([x.group(0) for x in r.finditer(słowa)])
```

```
['baba', 'berber', 'bobo', 'dada', 'dodo',
'dowodowo', 'dudu', 'dziadzia', 'dzidzi', 'gogo',
'jaja', 'jojo', 'kankan', 'kuku', 'kuskus', 'lala',
'lulu', 'mama', 'niania', 'ojoj', 'papa', 'psipsi',
'rowerowe', 'siusiu', 'tata', 'toto']
```

Regexpy oparte na NFA: nieliniowa złożoność czasowa dopasowywania regexpa




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

Regexpy oparte na NFA: blokada usług

Gdy program korzysta z regexpów opartych na NFA, a jego użytkownicy mogą:

- wprowadzać dowolne regexpy
- wprowadzać dowolne łańcuchy znaków, które powodują, że dopasowywanie regexpa ma nieliniową złożoność czasową

to działanie tego programu można zablokować

Regexpy oparte na NFA: blokada usług

Badacze znaleźli w tych 150 najpopularniejszych javowych aplikacjach internetowych na GitHubie, które zawierały jakiekolwiek regexpy:

2868 regexpów

522 regexpy z nieliniowym czasem dopasowywania, w tym

37 regexpów z wykładniczym czasem dopasowywania

Badacze zablokowali na ponad 10 minut działanie 27 z tych aplikacji

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

Regexpy: zalety i wady

+ zwięzłe: jeden regexp może zastępować wiele wierszy kodu
+ przenośne: podstawowe konstrukcje są takie same w różnych językach programowania
− bardziej złożone regexpy mogą być nieczytelne
− gdy silnik regexpów używa NFA, czas dopasowania regexpa do łańcucha znaków może rosnąć szybciej niż liniowo w stosunku do długości tego łańcucha

Regexpy w Go

Wybrane funkcje pakietu `regexp`

Tę część wykładu opracowałem na podstawie dokumentacji pakietu `regexp`

Gdy wydaję polecenie `go doc regexp`, widzę ją na ekranie

Wybrane funkcje pakietu regexp

```
func MustCompile(expr string) *Regexp
func (re *Regexp) FindAllString(s string, n int) []string
func (re *Regexp) FindStringSubmatch(s string) []string
func (re *Regexp) ReplaceAllString(src, repl string) string
func (re *Regexp) Split(s string, n int) []string
```

Funkcja `MustCompile`

```
func MustCompile(expr string) *Regexp
```

`MustCompile` kompiluje łańcuch `expr`. Jeśli kompilacja się powiodła, zwraca wskaźnik do obiektu typu `Regexp`. Jeśli kompilacja się nie powiodła, wywołuje funkcję wbudowaną `panic`

Dzięki `MustCompile` mogę bezpiecznie inicjalizować takie zmienne globalne, które zawierają skompilowane regexpy

Funkcja `FindAllString`

```
func (re *Regexp) FindAllString(s string, n int) []string
```

`FindAllString` zwraca wycinek. Ten wycinek zawiera kolejne części łańcucha `s`, które pasują do regexpa `re`. Jeśli `n >= 0`, to `FindAllString` zwraca co najwyżej `n` takich części łańcucha `s`. Jeśli `n < 0`, to `FindAllString` zwraca wszystkie takie części

Funkcja FindAllString

```go
package main

import (
    "fmt"
    "regexp"
)

var Word = regexp.MustCompile(`\pL+`)

func main() {
    text := "Wydział Wiertnictwa, Nafty i Gazu"
    fmt.Printf("%#v", Word.FindAllString(text, -1))
}
```

```
[]string{"Wydział", "Wiertnictwa", "Nafty", "i", "Gazu"}
```

Funkcja `FindStringSubmatch`

```
func (re *Regexp) FindStringSubmatch(s string) []string
```

`FindStringSubmatch` zwraca wycinek. Ten wycinek zawiera pierwszą taką część łańcucha `s`, która pasuje do regexpa `re`, a następnie kolejne poddopasowania łańcucha `s`, które pasują do kolejnych grup regexpa `re`

Funkcja FindStringSubmatch – przykład

```
var IP = regexp.MustCompile(
    "^(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])$")

func main() {
    fmt.Printf("%#v", IP.FindStringSubmatch("127.0.0.1"))
}
```

```
[]string{"127.0.0.1", "127", "0", "0", "1"}
```

Funkcja `ReplaceAllString`

```
func (re *Regexp) ReplaceAllString(src, repl string) string
```

`ReplaceAllString` zwraca kopię łańcucha `src`. Te części tej kopii, które pasują do regexpa `re`, zastępuje łańcuchem `repl`. Wewnątrz łańcucha `repl` podstawia zamiast szablonów `${1}`, `${2}`,... odpowiednie poddopasowania regexpa `re`

Funkcja ReplaceAllString

```
var Animal = regexp.MustCompile("([Kk]ot|[Pp]ies)")

func main() {
    text := "Kot i pies leżą na kanapie"
    fmt.Printf("%#v", Animal.ReplaceAllString(text, "${1}ek"))
}
```

```
"Kotek i piesek leżą na kanapie"
```

Funkcja `Split`

```
func (re *Regexp) Split(s string, n int) []string
```

`Split` dzieli łańcuch `s` na części. Te części są rozdzielone takimi podłańcuchami, które pasują do regexpa `re`. Jeśli `n >= 0`, to `Split` zwraca co najwyżej `n` takich części łańcucha `s`. Jeśli `n < 0`, to `Split` zwraca wszystkie takie części

Funkcja Split – przykład

```
var Comma = regexp.MustCompile(`, +\pL+[^iy]( |$)`)

func main() {
    text := "Wydział Informatyki, Elektroniki " +
        "i Telekomunikacji, Biuro Dziekana"
    fmt.Printf("%#v", Comma.Split(text, -1)[0])
}
```

```
"Wydział Informatyki, Elektroniki i Telekomunikacji"
```

Podsumowanie

Wyrażenia regularne

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

Regexpy

- Jak rozpoznawać początek i koniec tekstu?
- Jak dopasować 1 znak do regexpa?
- Jak składać regexpy?
- Jak powtarzać regexpy?
- Jak grupować regexpy?

Regexpy w Go

```
func MustCompile(expr string) *Regexp
func (re *Regexp) FindAllString(s string, n int) []string
func (re *Regexp) FindStringSubmatch(s string) []string
func (re *Regexp) ReplaceAllString(src, repl string) string
func (re *Regexp) Split(s string, n int) []string
```

Pomysły, uwagi, pytania, sugestie

Proszę wysyłać podpisane pomysły, uwagi, pytania, sugestie na temat wykładów lub laboratoriów na adres mgc@agh.edu.pl

lub wpisywać anonimowe pomysły, uwagi, pytania, sugestie pod adresem https://tiny.cc/algorytmy-tekstowe

Do zobaczenia
na następnym wykładzie

Jego tematem będzie wyszukiwanie wzorca w tekście

Źródła zdjęć

https://en.wikipedia.org/wiki/Stephen_Cole_Kleene
https://en.wikipedia.org/wiki/Janusz_Brzozowski_(computer_scientist)
https://www.azquotes.com/author/16151-Jamie_Zawinski