



# Подводные камни регулярных выражений

Уязвимости, катастрофический возврат и  
ReDoS-атаки



Георгий Тормозов  
C# разработчик



# Георгий Тормозов

C# разработчик

- Занимаюсь разработкой статического анализатора
- Пишу статьи
- Люблю анекдоты



PVS-Studio

# План

4

Что же такое ReDoS

ReDoS на реальных проектах

Как бороться с ReDoS

Профилактика ReDoS



# Что же такое ReDoS



# Что же такое ReDoS?

6

ReDoS – regular expression denial of service

Разновидность DoS атаки, которая осуществляется с помощью уязвимого регулярного выражения

# Что же такое ReDoS?

7

ReDoS – regular expression denial of service

Разновидность DoS атаки, которая осуществляется с помощью уязвимого регулярного выражения

Последствия ReDoS:

- Отказ в обслуживании
- Замедление приложения

# Что же такое ReDoS?

8

ReDoS – regular expression denial of service

Разновидность DoS атаки, которая осуществляется с помощью уязвимого регулярного выражения

Классификация: **CWE-1333**

# Как работают регулярные выражения

# Как работают регулярные выражения

10

## Лупа

# Как работают регулярные выражения

11

## Лупа

Пупа и Лупа пошли получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу...

## Лупа

Пупа и Лупа пошли получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу...

## Лупа

Пупа **и** Лупа пошли получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу...

# Как работают регулярные выражения

14

## Лупа

Пупа и **Лупа** пошли получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу...

# Как работают регулярные выражения

15

## Лупа

Пупа и **Лупа пошли** получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу...

# Как работают регулярные выражения

16

## Лупа

■ ■ ■

Пупа и **Лупа** пошли получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу...

# Как работают регулярные выражения

17

## Лупа

Пупа и **Лупа** пошли получать зарплату, но в  
бухгалтерии всё перепутали и **Лупа** получила за Пупу...

## Лупа

Пупа и **Лупа** пошли получать зарплату, но в бухгалтерии всё перепутали и **Лупа** получила за Пупу...

# Как работают регулярные выражения

19

`^[A-Z0-9._%+-]+@[A-Z0-9-]+.[A-Z]{2,4}$`

# Как работают регулярные выражения

20

## Конечные автоматы

# Как работают регулярные выражения

21

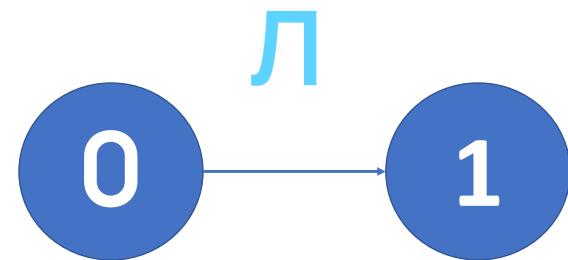
## Конечные автоматы



# Как работают регулярные выражения

22

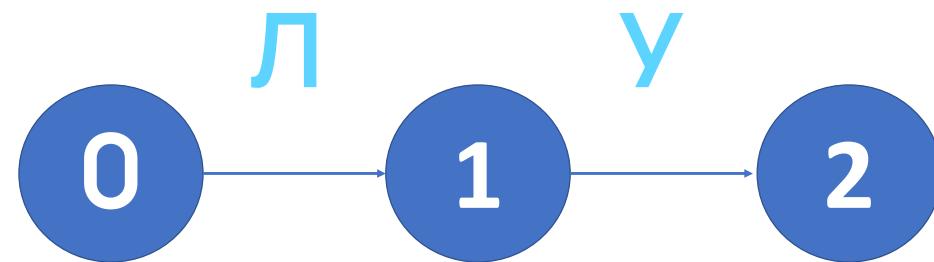
## Конечные автоматы



# Как работают регулярные выражения

23

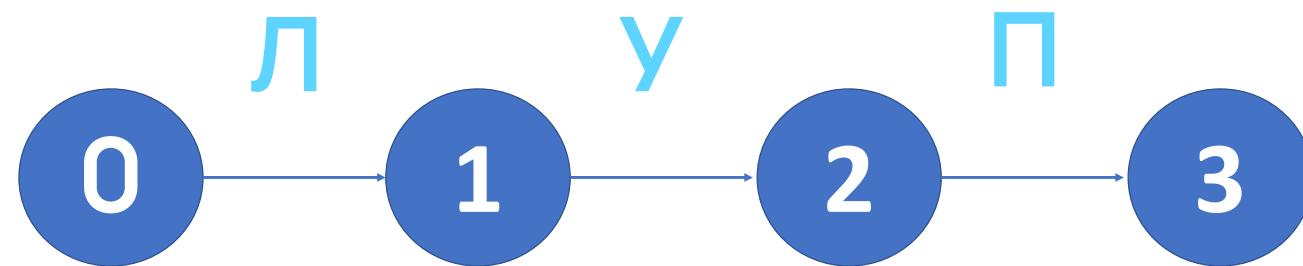
## Конечные автоматы



# Как работают регулярные выражения

24

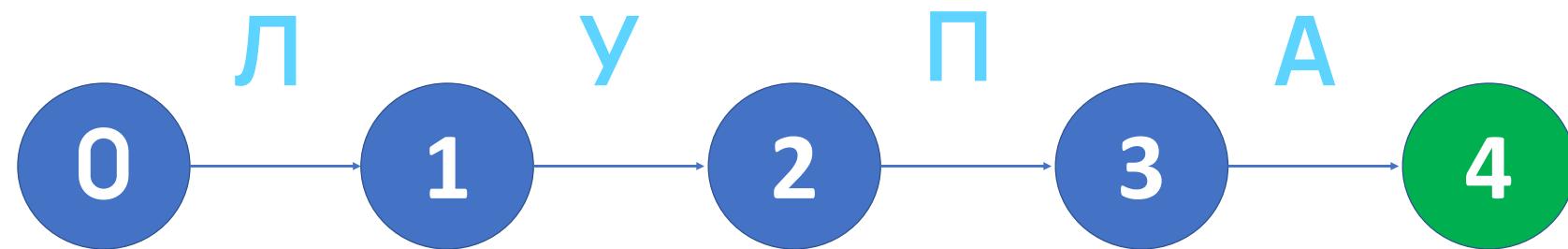
## Конечные автоматы



# Как работают регулярные выражения

25

## Конечные автоматы



# Как работают регулярные выражения

26

## Конечные автоматы

# Как работают регулярные выражения

27

Конечные автоматы

Детерминированные

Недетерминированные

# Как работают регулярные выражения

28

## Конечные автоматы

### Детерминированные

### Недетерминированные

- Быстрее работают в «худших» случаях

# Как работают регулярные выражения

29

## Конечные автоматы

### Детерминированные

- Быстрее работают в «худших» случаях

### Недетерминированные

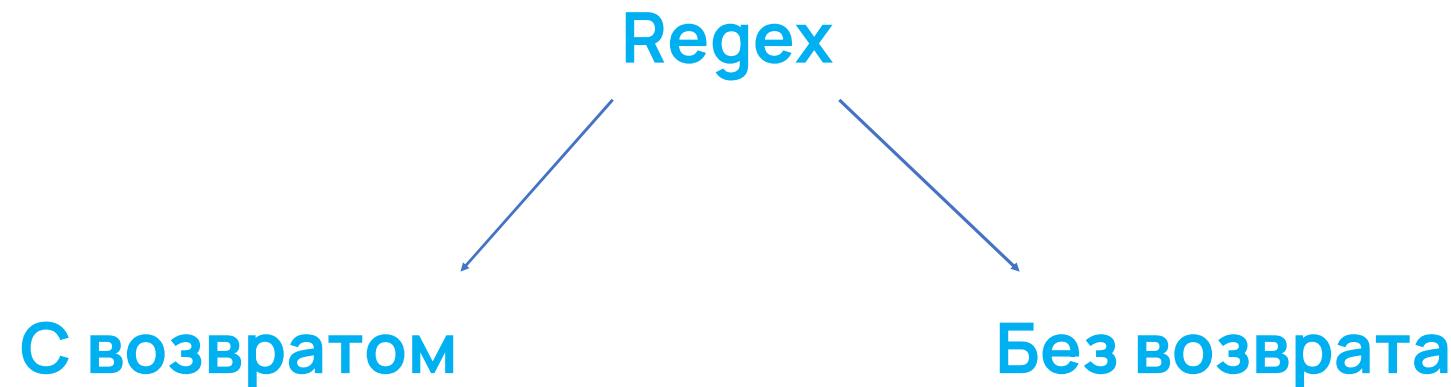
- Быстрее работают в «лучших» случаях
- Граф быстрее строится
- Доступна функция возврата

# Как работают регулярные выражения

30

Regex

# Как работают регулярные выражения



# Как работают регулярные выражения

Regex

С возвратом

Без возврата

Содержит переменные  
кванторы +, \*, ?, {n,}, {n, m}

Пример: **a+b**

# Как работают регулярные выражения

33

## Regex

### С возвратом

Содержит переменные  
кванторы +, \*, ?, {n,}, {n, m}

Пример: **a+b**

### Без возврата

Не содержит переменные  
кванторы или конструкторы  
изменения

Пример: **a{2}b**

# Как работают регулярные выражения

34

Регулярное выражение: **a+b**

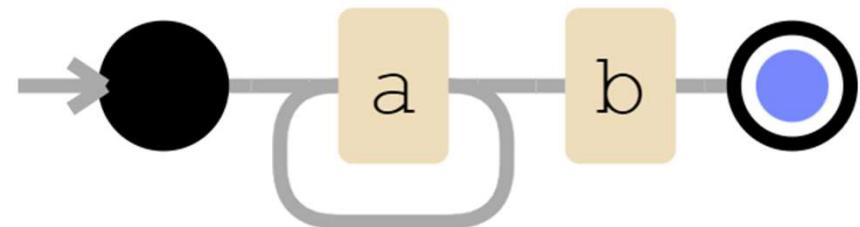
Последовательность для разбора: aaa

# Как работают регулярные выражения

35

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

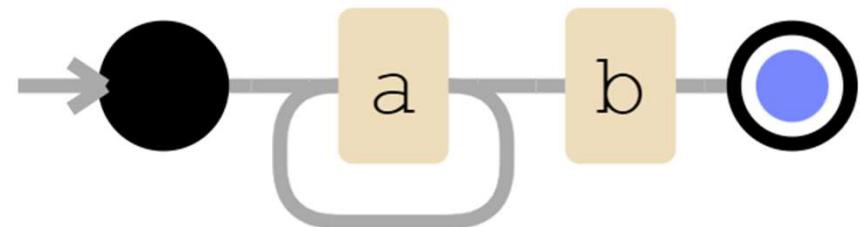


# Как работают регулярные выражения

36

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



```
Regex regex = new Regex("a+b");
```

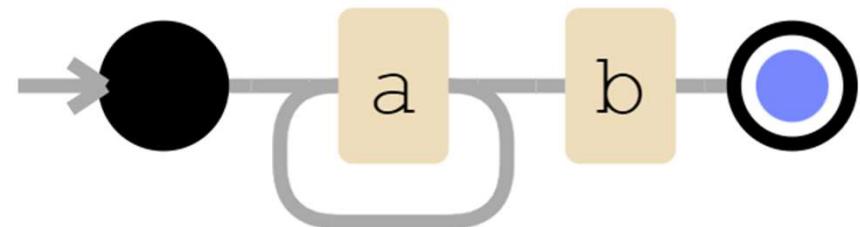
```
Match match = regex.Match("aaa");
```

# Как работают регулярные выражения

37

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



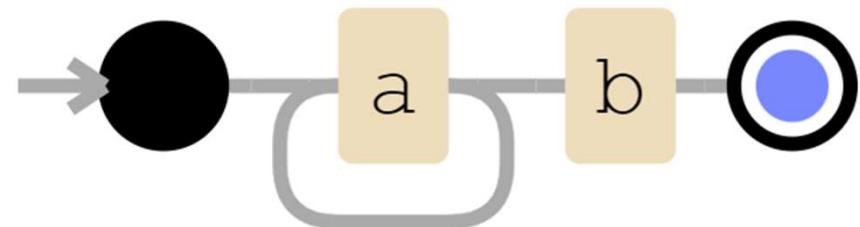
aaa

# Как работают регулярные выражения

38

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



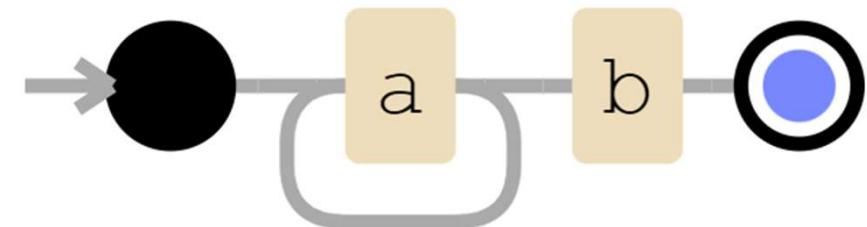
aaa

# Как работают регулярные выражения

39

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



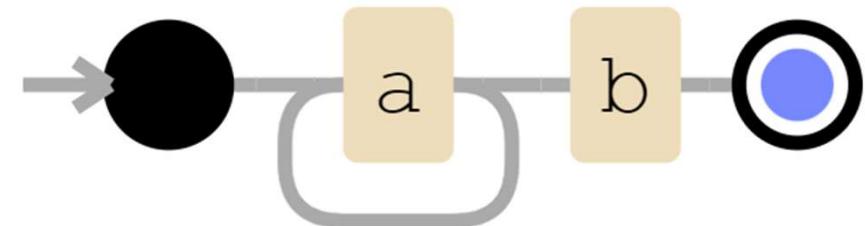
aaa

# Как работают регулярные выражения

40

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



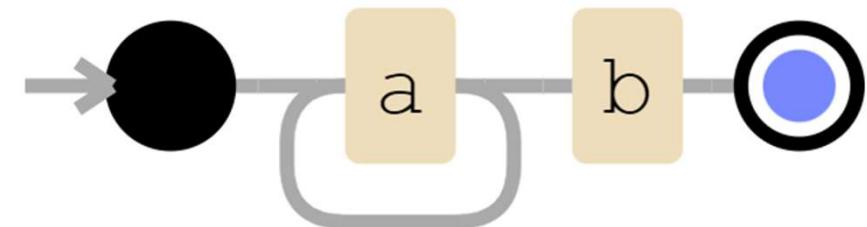
aaa

# Как работают регулярные выражения

41

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



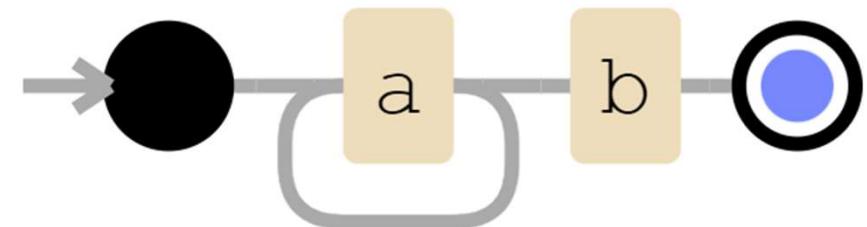
aaa

# Как работают регулярные выражения

42

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



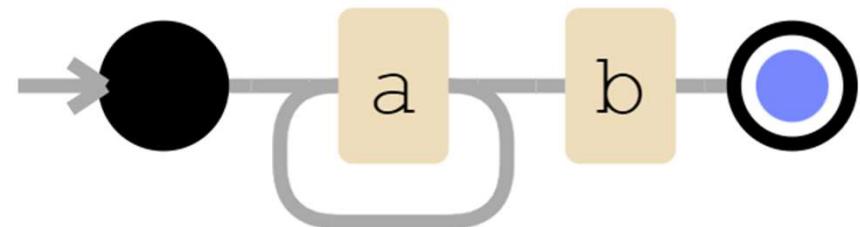
aaa

# Как работают регулярные выражения

43

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



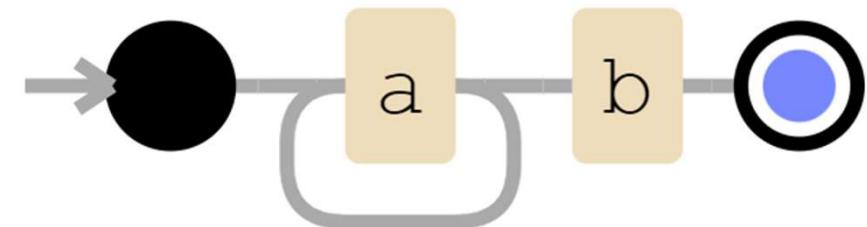
aaa

# Как работают регулярные выражения

44

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



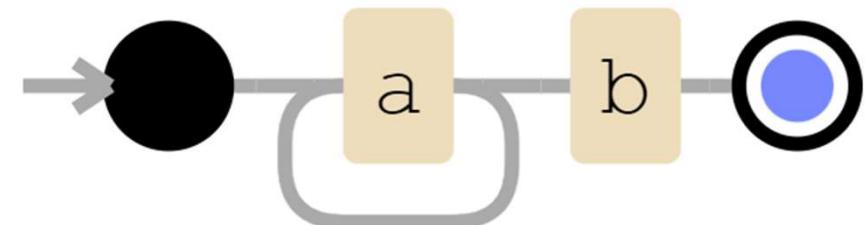
aaa

# Как работают регулярные выражения

45

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



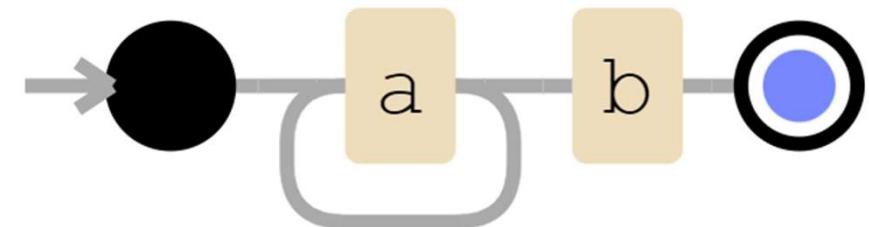
aaa

# Как работают регулярные выражения

46

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



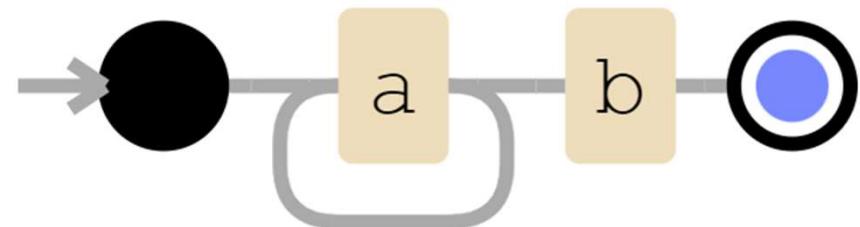
aaa

# Как работают регулярные выражения

47

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



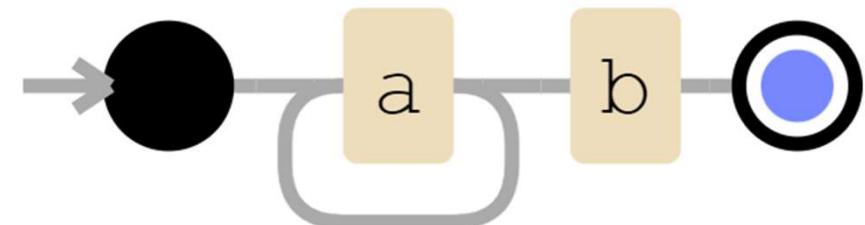
aaa

# Как работают регулярные выражения

48

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



aaa

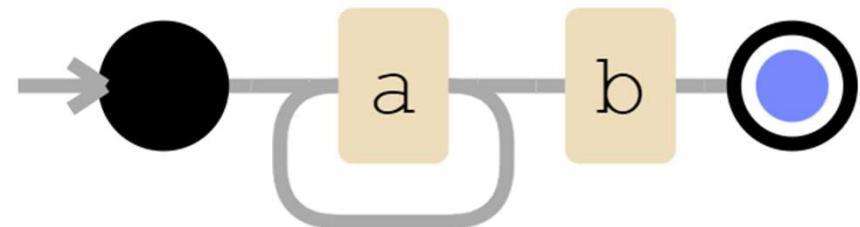
The string "aaa" is displayed in large black font. The third 'a' character is highlighted with a thick yellow vertical bar, indicating the current position of the regex engine during the search for the pattern "a+b".

# Как работают регулярные выражения

49

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



aaa

# Рассмотрим пример

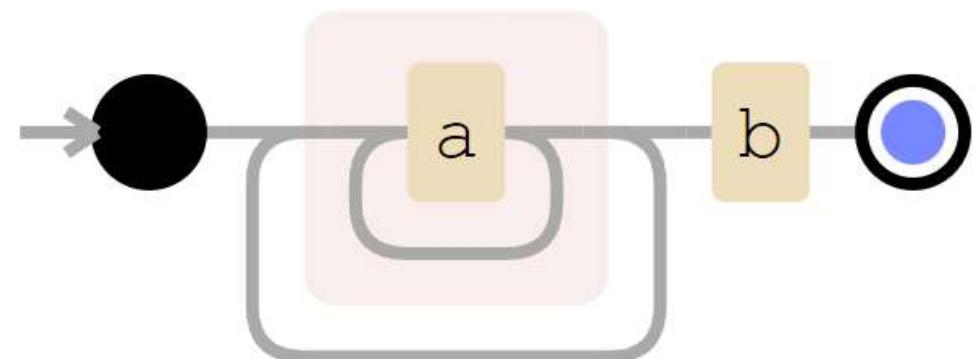
50

Регулярное выражение: **(a<sup>+</sup>)<sup>+</sup>b**

# Рассмотрим пример

51

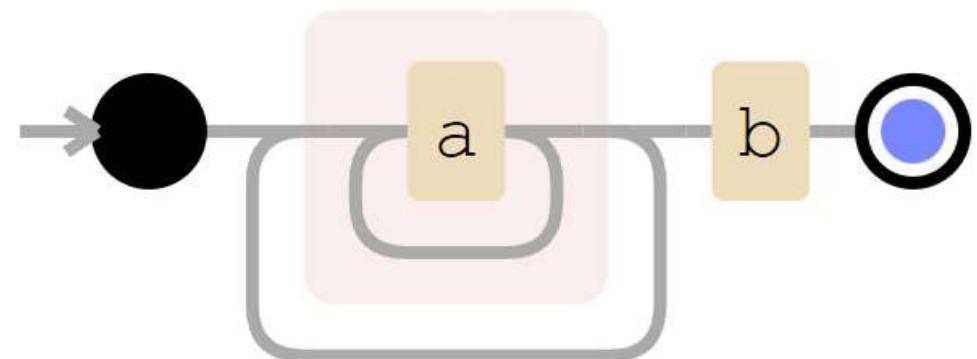
Регулярное выражение:  $(a^+)^+b$



# Рассмотрим пример

52

Регулярное выражение:  $(a^+)^+b$



Что изменилось?

- На графике появилась дополнительная петля
- Количество возможных вариантов сопоставления увеличилось

# Сравним два регулярных выражения

53

$a+b$

Input strings:

aaaaaaab  
aaaaaaaaab  
aaaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaab  
aaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab

$(a+)^+b$

Input strings:

aaaaaaab  
aaaaaaaaab  
aaaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaab  
aaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab

# Сравним два регулярных выражения

54

$a+b$

Input strings:

aaaaaaab  
aaaaaaaaab  
aaaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaab  
aaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab

Effect of input string length on regular expression execution time.

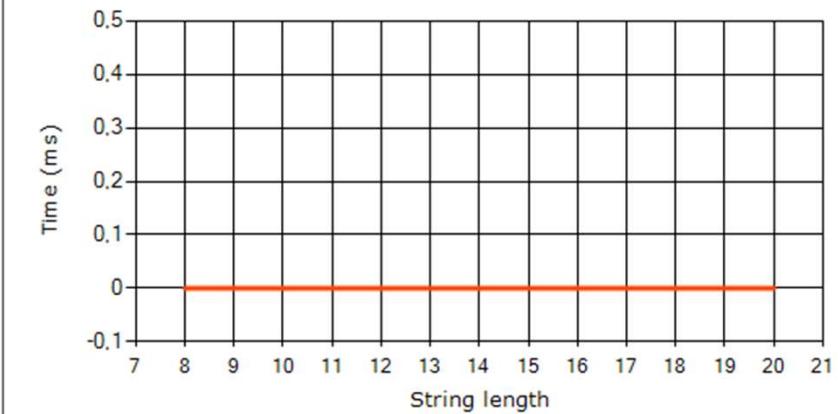


$(a^+)^+b$

Input strings:

aaaaaaab  
aaaaaaaaab  
aaaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaab  
aaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab

Effect of input string length on regular expression execution time.



# Сравним еще раз

55

$a+b$

Input strings:

```
aaaaaaa  
aaaaaaaa  
aaaaaaaaa  
aaaaaaaaaa  
aaaaaaaaaaa  
aaaaaaaaaaaa  
aaaaaaaaaaaaa  
aaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa
```

$(a+)^+b$

Input strings:

```
aaaaaaa  
aaaaaaaa  
aaaaaaaaa  
aaaaaaaaaa  
aaaaaaaaaaa  
aaaaaaaaaaaa  
aaaaaaaaaaaaa  
aaaaaaaaaaaaaa
```

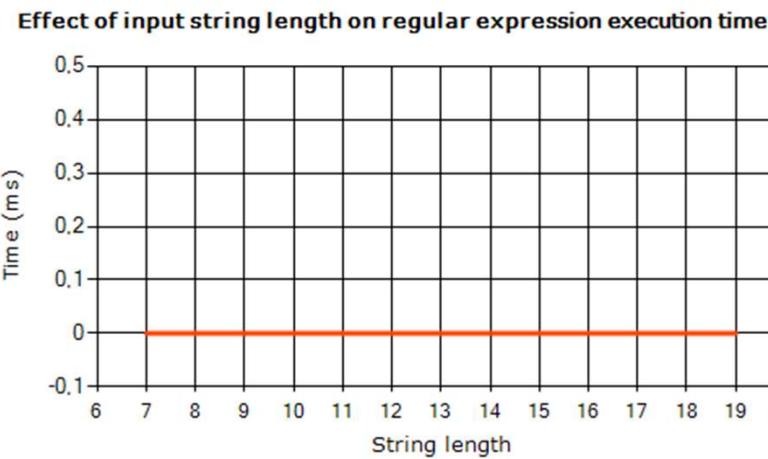
# Сравним еще раз

56

$a+b$

Input strings:

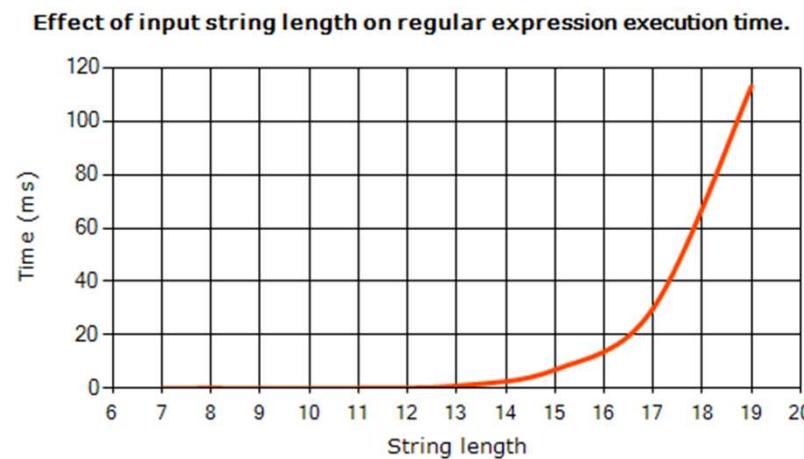
```
aaaaaaaa  
aaaaaaaaa  
aaaaaaaaaa  
aaaaaaaaaaa  
aaaaaaaaaaaa  
aaaaaaaaaaaaa  
aaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa
```



$(a^+)+b$

Input strings:

```
aaaaaaaa  
aaaaaaaaa  
aaaaaaaaaa  
aaaaaaaaaaa  
aaaaaaaaaaaa  
aaaaaaaaaaaaa  
aaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa
```



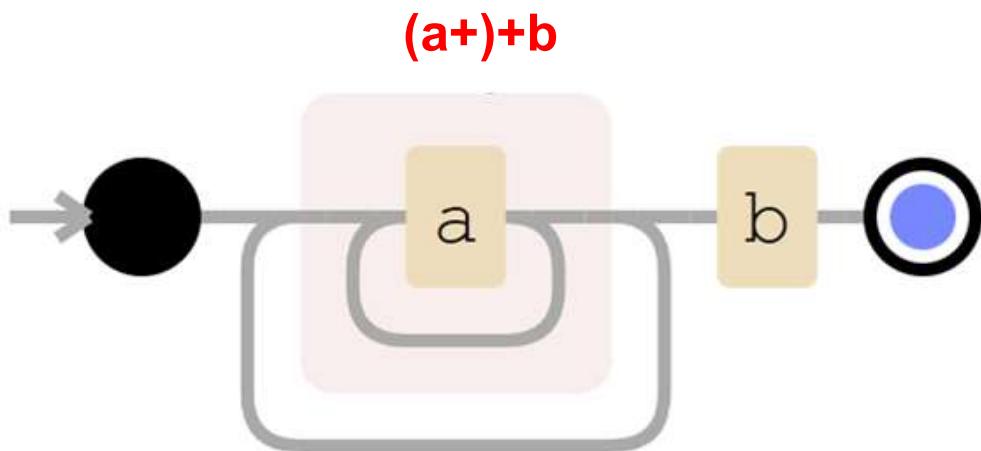
# Катастрофический возврат

- Существует два подвыражения, при этом одно из них включено в другое и к каждому из них применяется один из следующих кванторов: '\*', '+', '\*?', '+?', '{...}'
- Существует такая строка, которую можно было бы сопоставить с обоими этими подвыражениями

# Катастрофический возврат

58

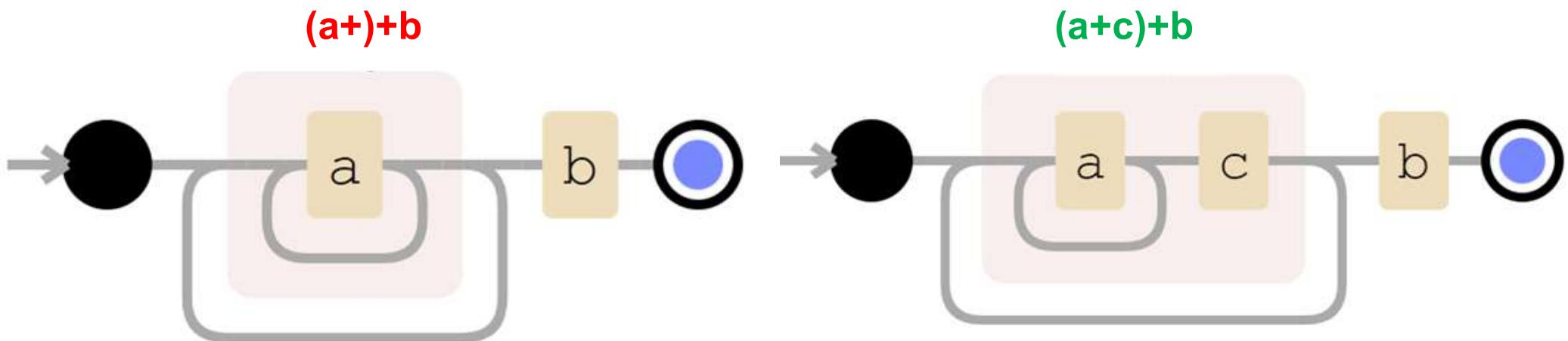
- Существует два подвыражения, при этом одно из них включено в другое и к каждому из них применяется один из следующих кванторов: '\*', '+', '\*?', '+?', '{...}'
- Существует такая строка, которую можно было бы сопоставить с обоими этими подвыражениями



# Катастрофический возврат

59

- Существует два подвыражения, при этом одно из них включено в другое и к каждому из них применяется один из следующих кванторов: '\*', '+', '\*?', '+?', '{...}'
- Существует такая строка, которую можно было бы сопоставить с обоими этими подвыражениями



# Катастрофический возврат

60

$\wedge (\text{d}\text{w}\text{l}\text{w}\text{d})\$$

# Катастрофический возврат

61



(\d\w|\w\d)\$

# Катастрофический возврат

62

`^\d\w|\w\d)$`

# Катастрофический возврат

63

`^(\\d\\w|\\w\\d)$`

# Катастрофический возврат

64

`^(\\d\\w|\\w\\d)$`

# Катастрофический возврат

65

$\wedge (\text{d}\text{w}\text{l}\text{w}\text{d})\$$

# Катастрофический возврат

66

`^(\\d\\w|\\w\\d)$`

12а

# Катастрофический возврат

67

`^\d\w|\w\d)$`

12a

# Катастрофический возврат

68

`^\d\w|\w\d$`

12a

# Катастрофический возврат

69

`^(\\d\\w|\\w\\d)$`

12а

# Катастрофический возврат

70

`^(\\d\\w|\\w\\d)$`

12а

# Катастрофический возврат

71

`^(\\d\\w|\\w\\d)\\$`

12a

# Катастрофический возврат

72

`^(\\d\\w|\\w\\d)$`

12a

# Катастрофический возврат

73

`^(\\d\\w|\\w\\d)(\\d\\w|\\w\\d)$`

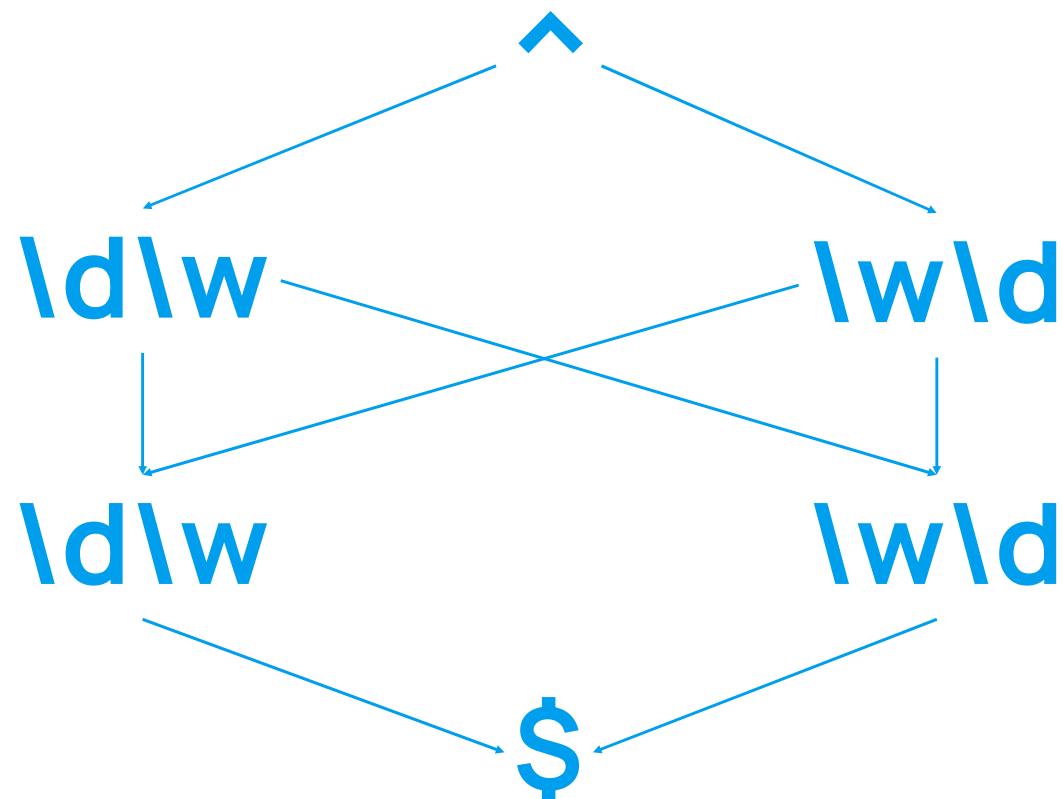
# Катастрофический возврат

74

`^(\\d\\w|\\w\\d)(\\d\\w|\\w\\d)$`

# Катастрофический возврат

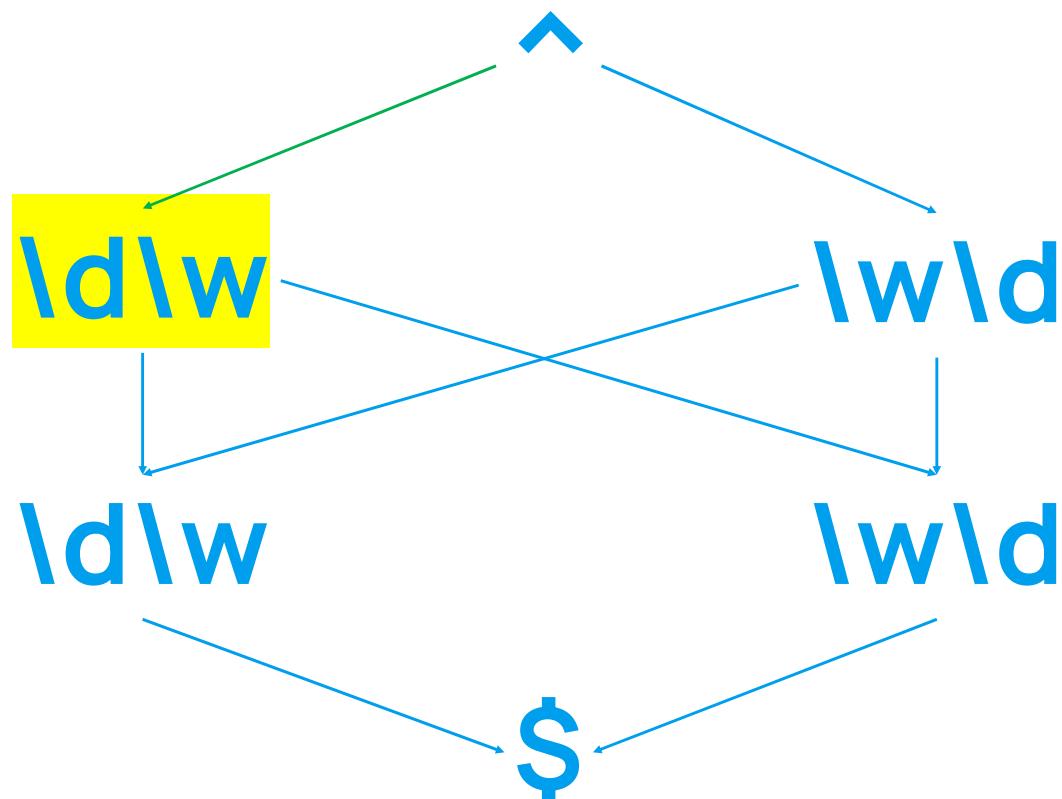
75



1234a

# Катастрофический возврат

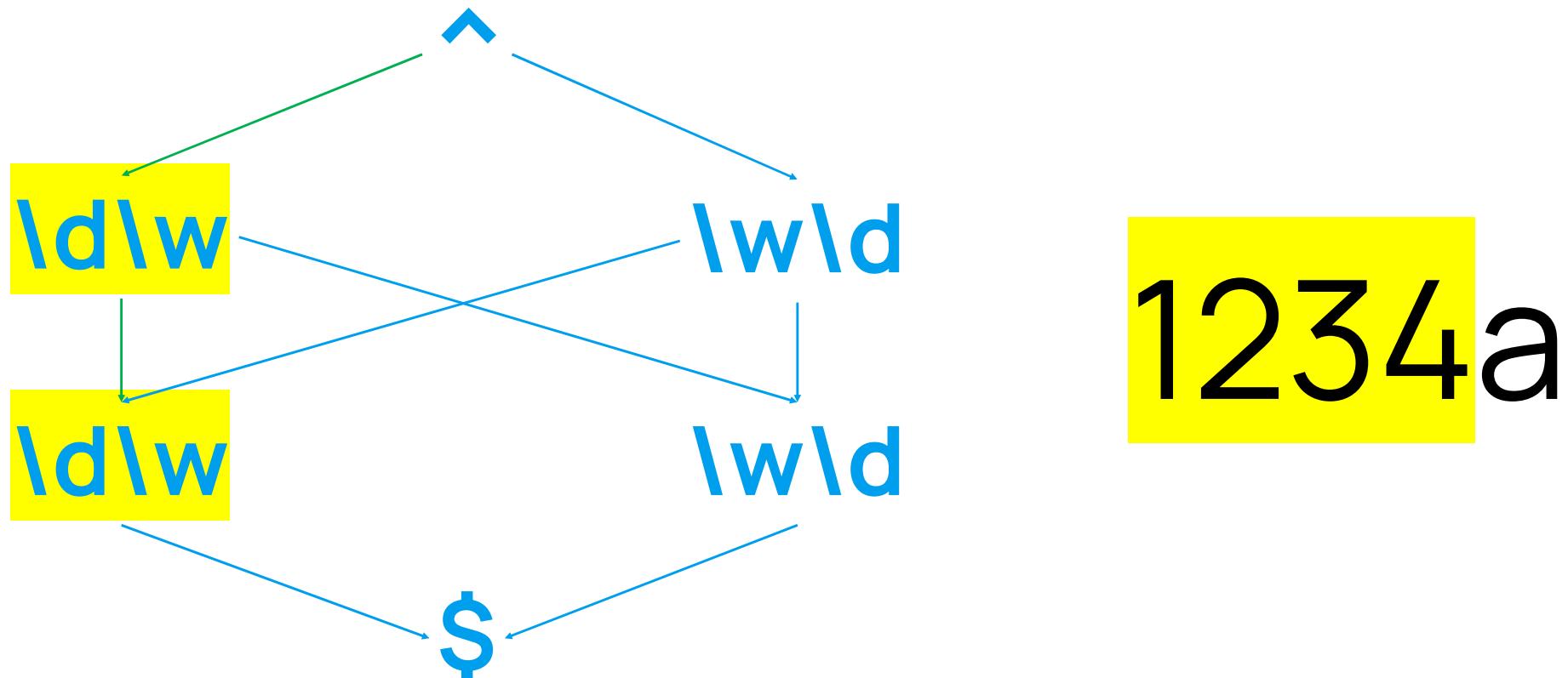
76



1234a

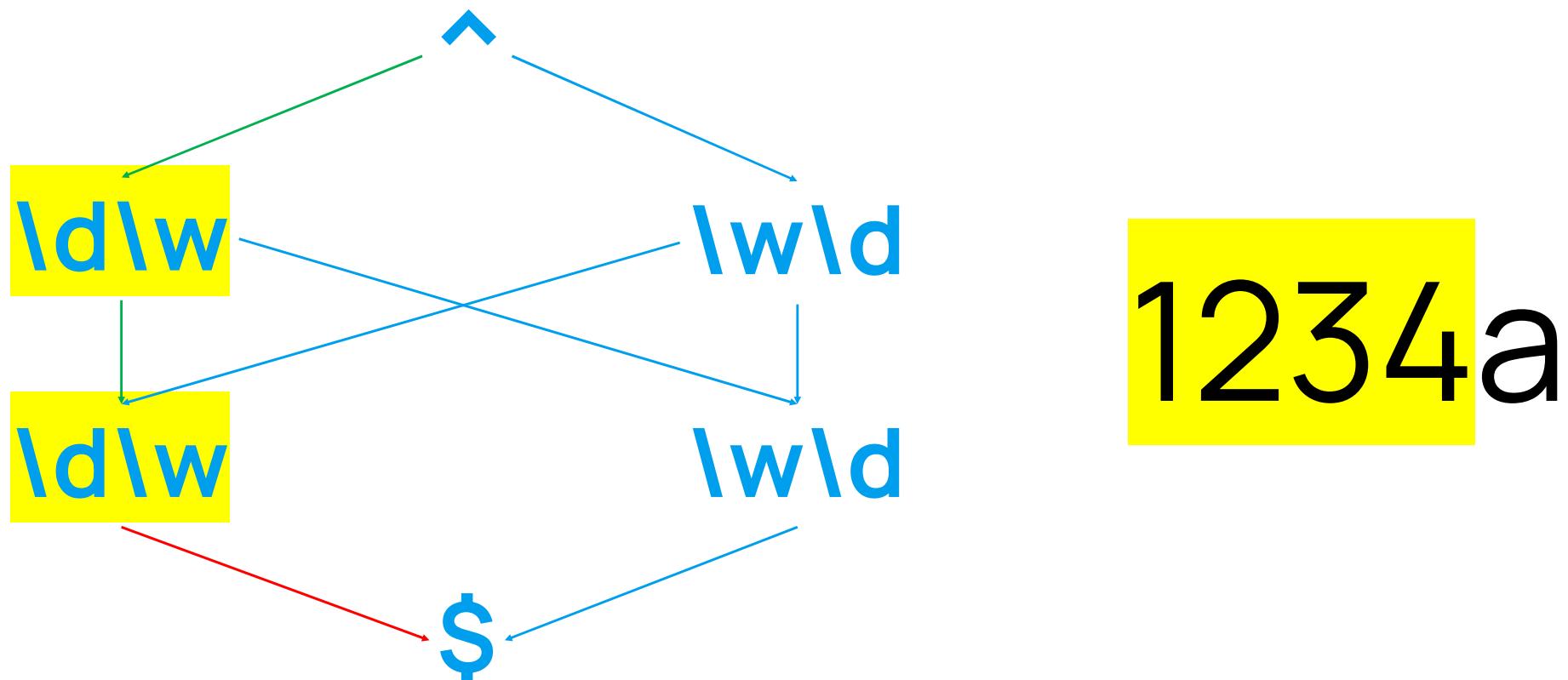
# Катастрофический возврат

77



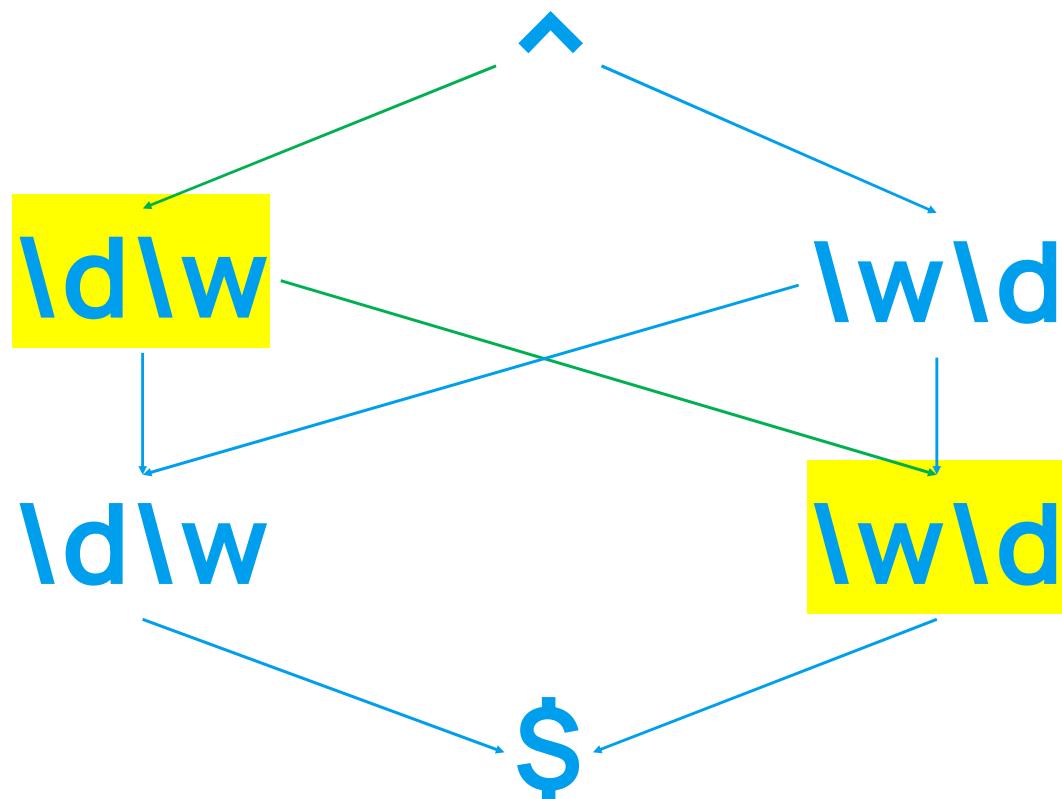
# Катастрофический возврат

78



# Катастрофический возврат

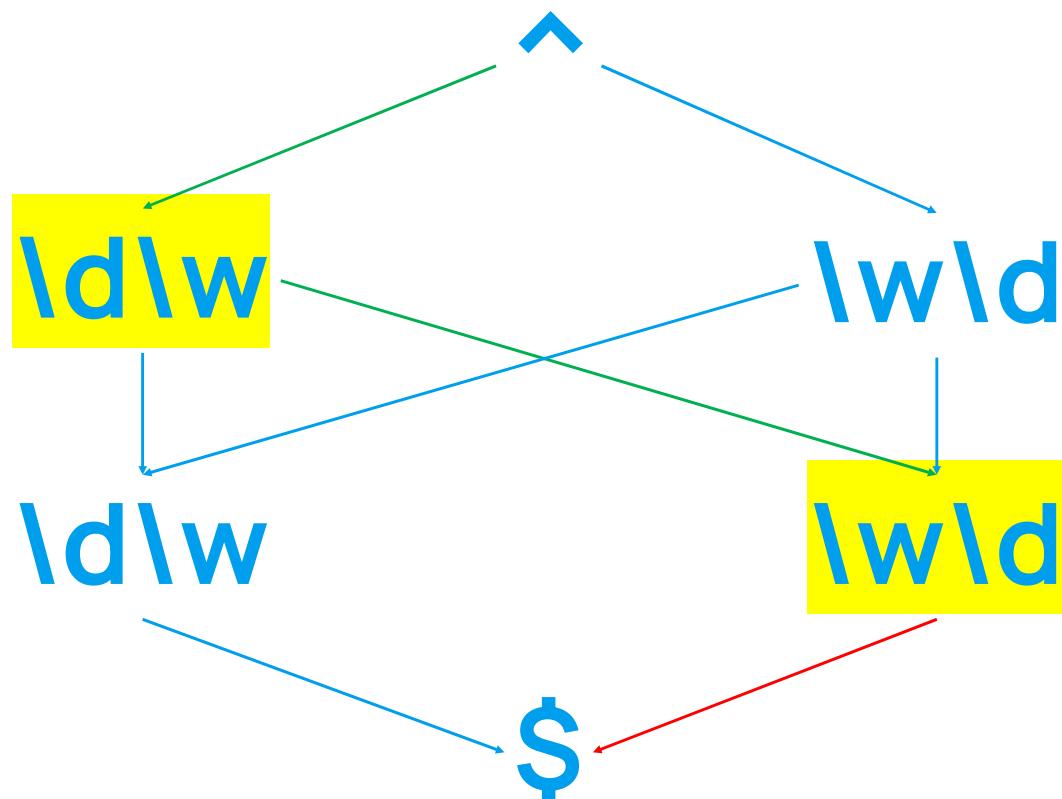
79



1234a

# Катастрофический возврат

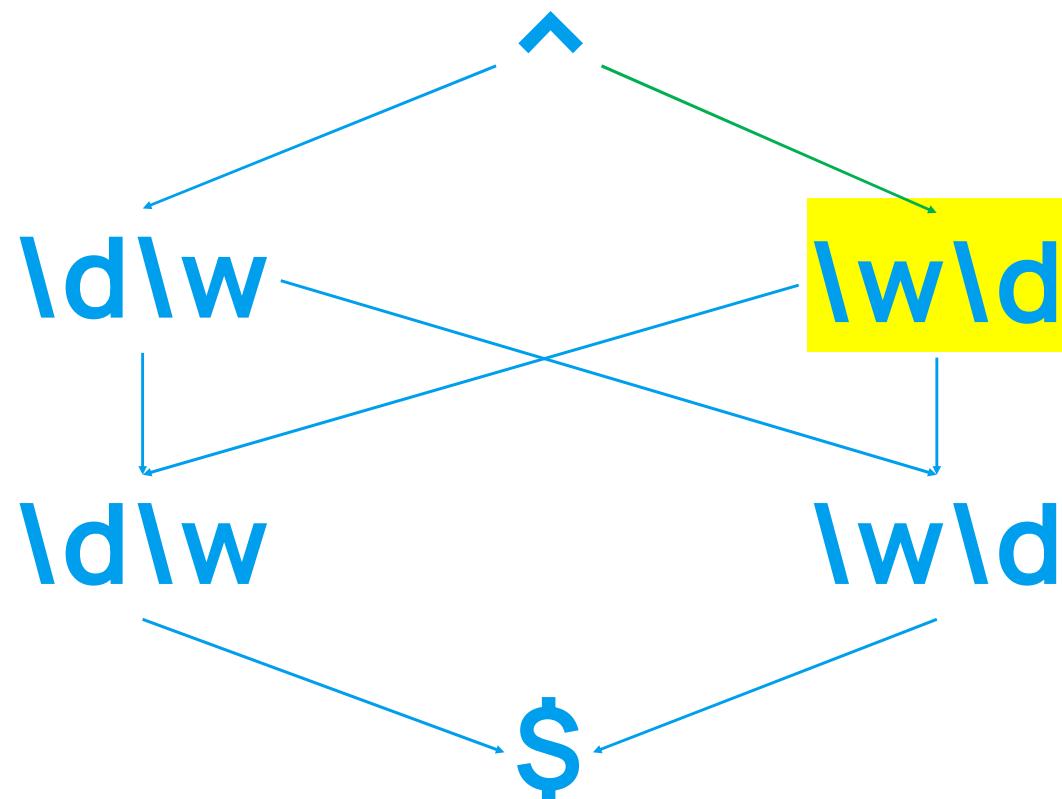
80



1234a

# Катастрофический возврат

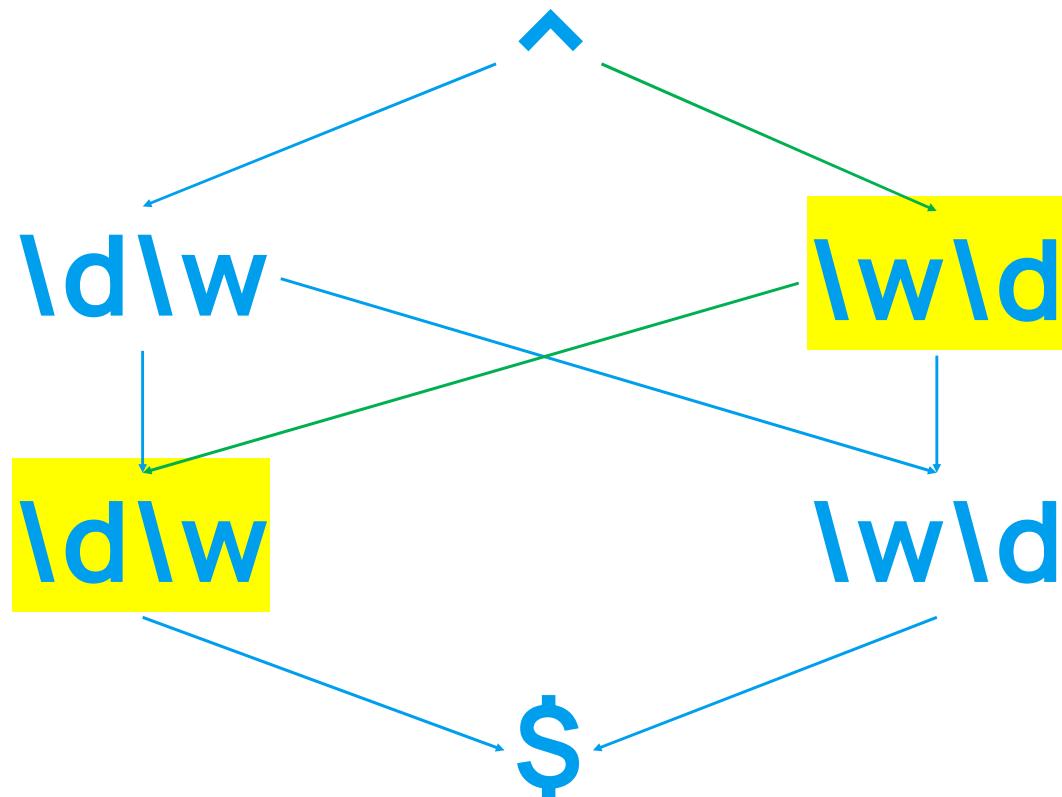
81



1234a

# Катастрофический возврат

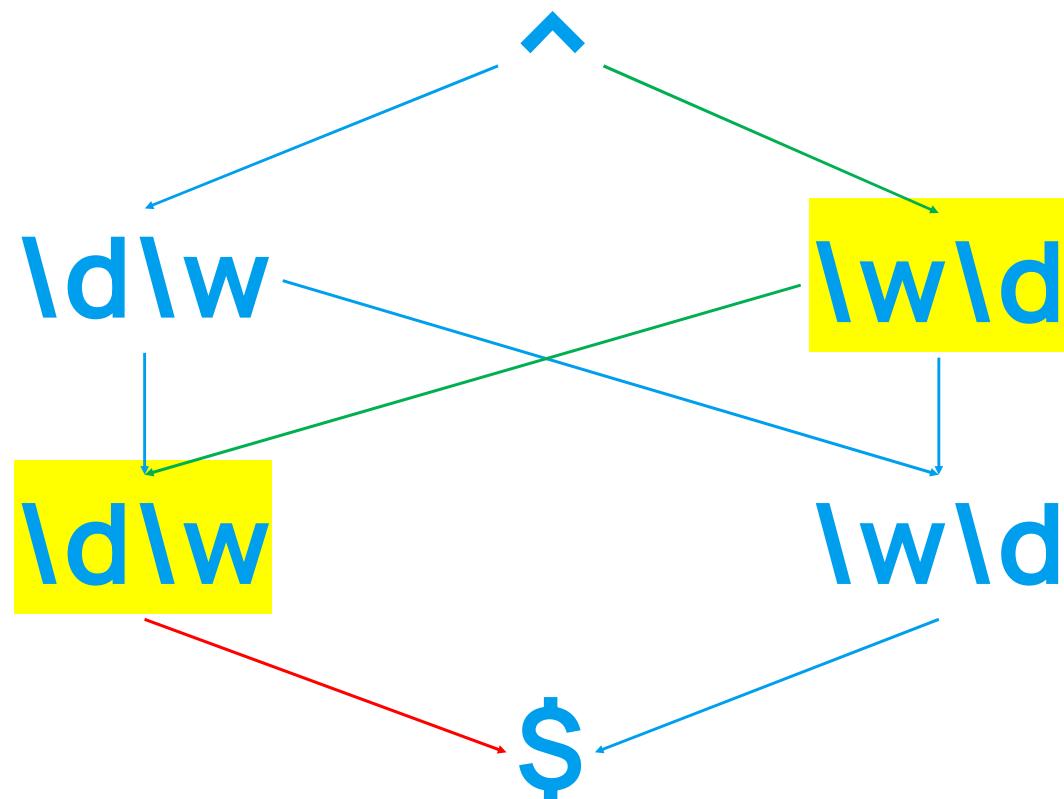
82



1234a

# Катастрофический возврат

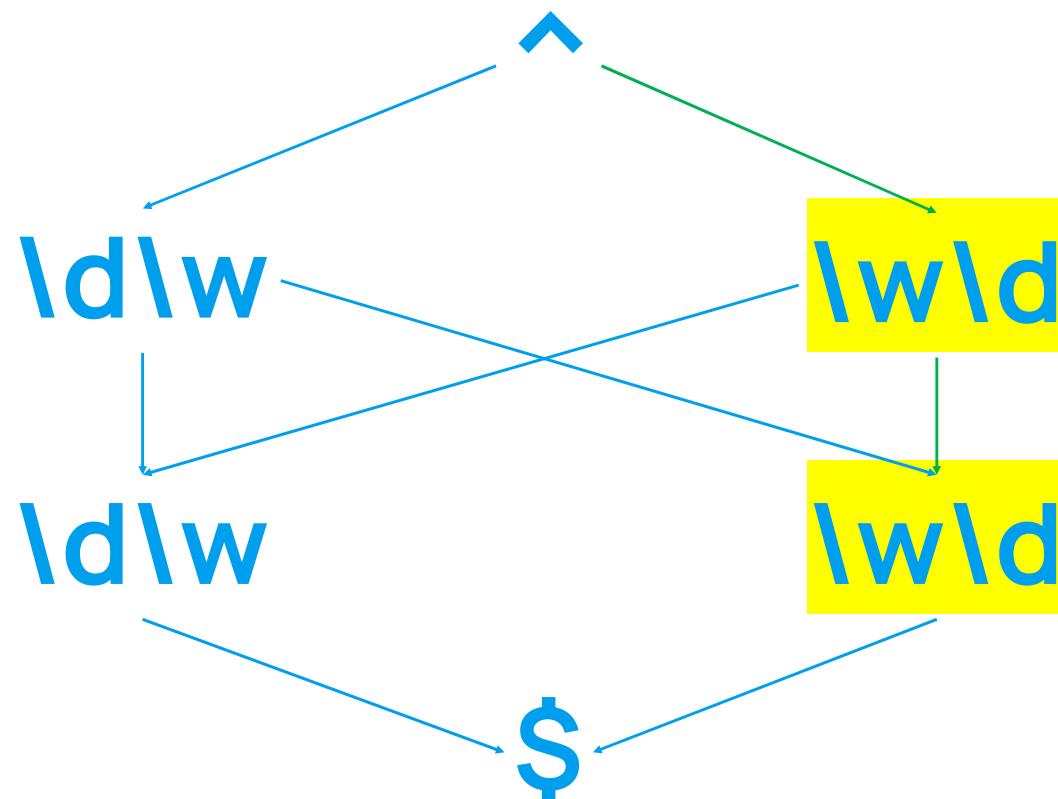
83



1234a

# Катастрофический возврат

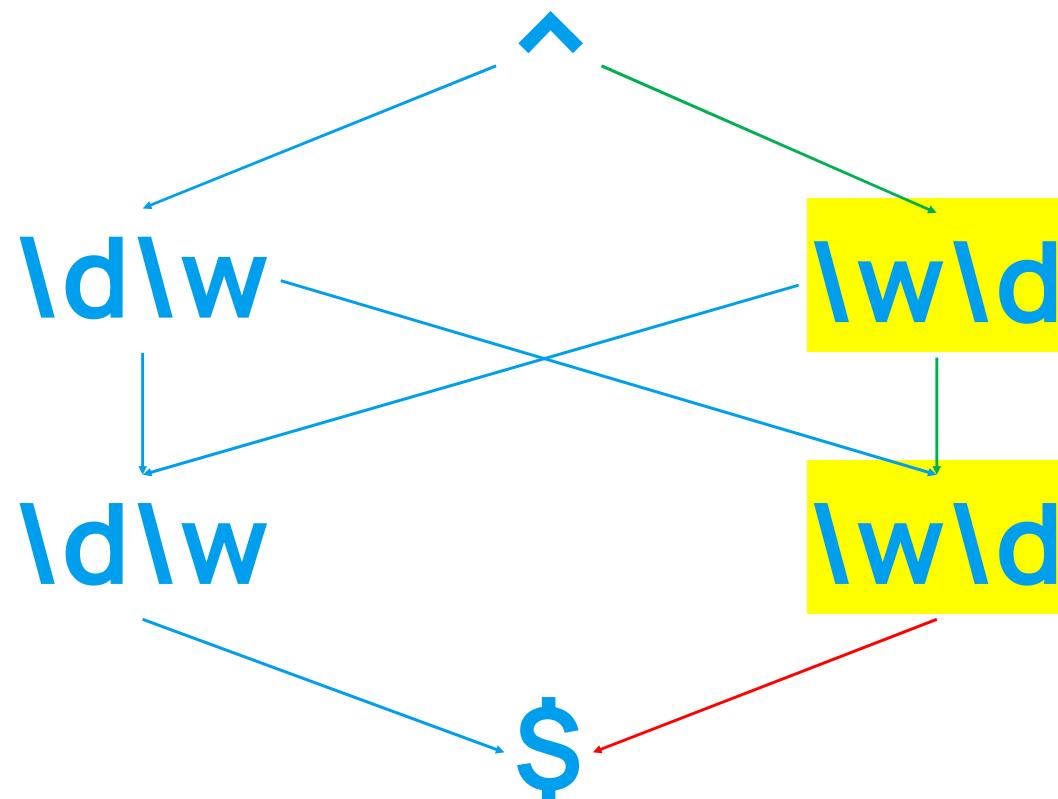
84



1234a

# Катастрофический возврат

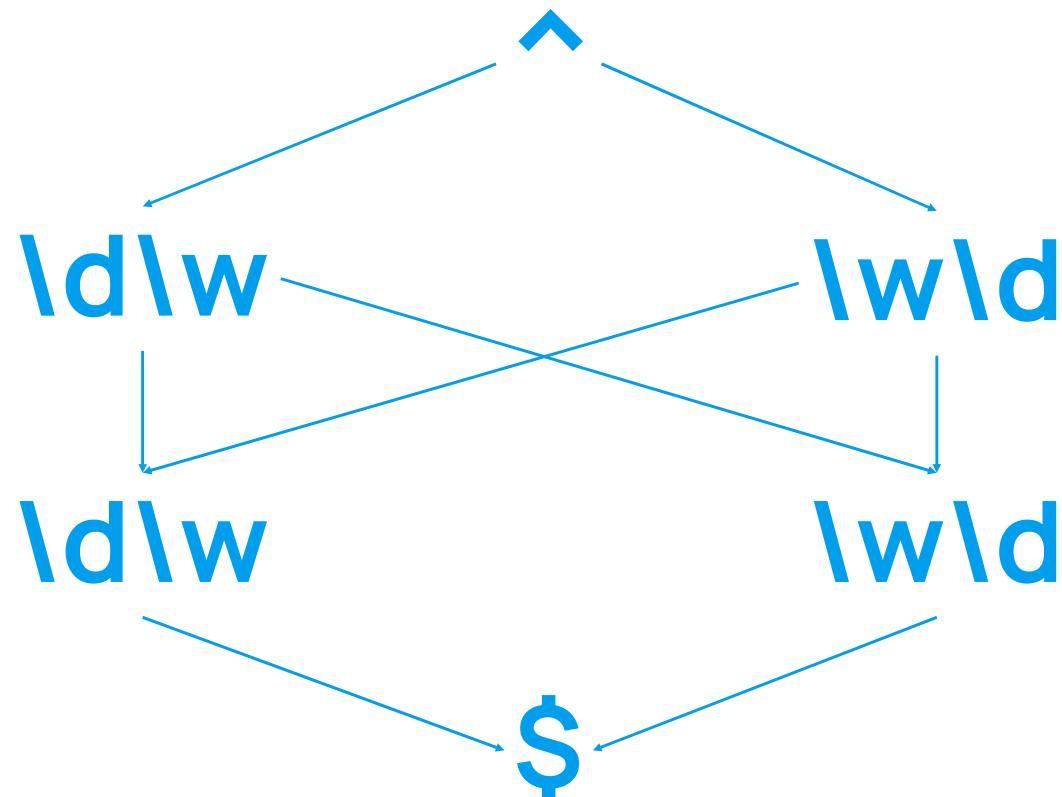
85



1234a

# Катастрофический возврат

86



1234a

# Катастрофический возврат

87

```
using System.Diagnostics;
using System.Text.RegularExpressions;

var sw = new Stopwatch();
for (int i = 10; i <= 30; i++)
{
    var r = new Regex($@"^(\w\d|\d\w){{{{i}}}}$");
    string input = new string('1', (i * 2) + 1);

    sw.Restart();
    r.IsMatch(input);
    sw.Stop();
    Console.WriteLine($"{i}: {sw.Elapsed.TotalMilliseconds:N}ms");
}
```

# Катастрофический возврат

88

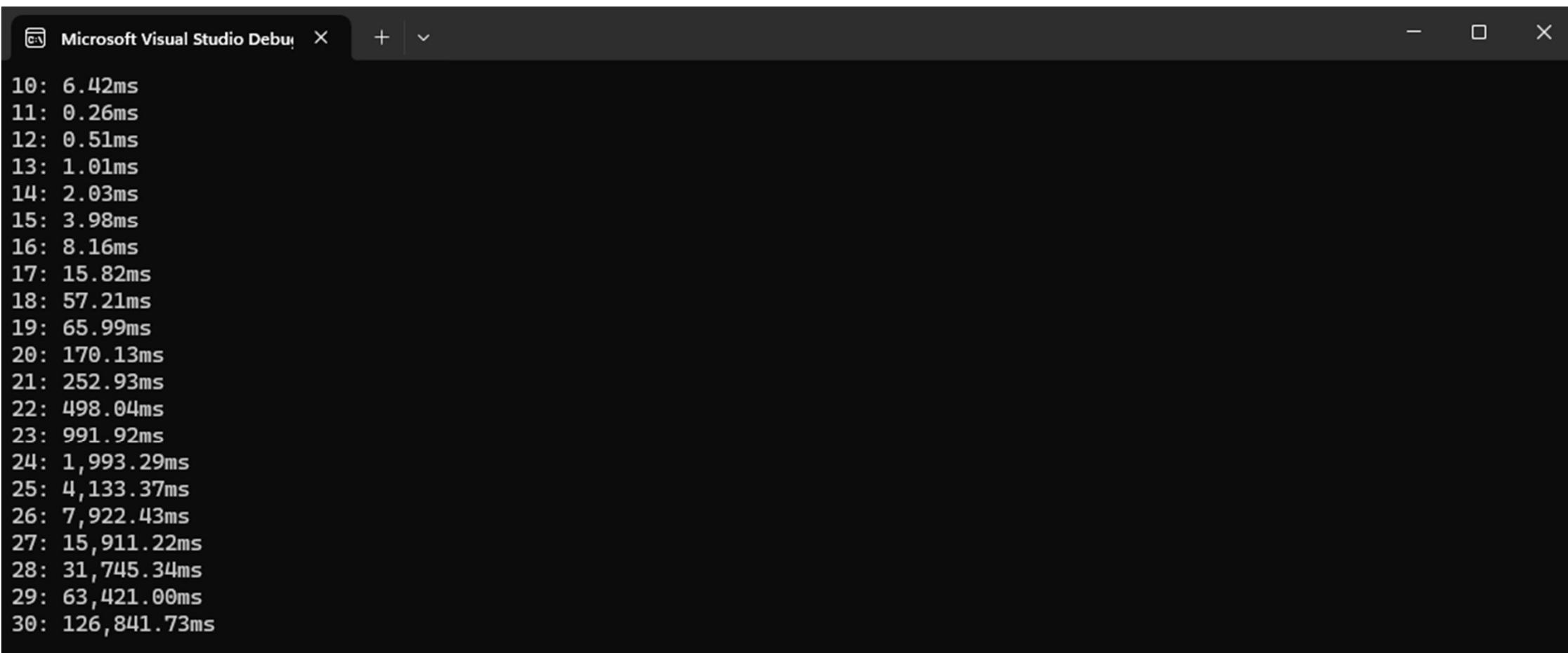
```
using System.Diagnostics;
using System.Text.RegularExpressions;

var sw = new Stopwatch();
for (int i = 10; i <= 30; i++)
{
    var r = new Regex($@"^(\w\d|\d\w) {{{i}}} }$");
    string input = new string('1', (i * 2) + 1);

    sw.Restart();
    r.IsMatch(input);
    sw.Stop();
    Console.WriteLine($"{i}: {sw.Elapsed.TotalMilliseconds:N}ms");
}
```

# Катастрофический возврат

89



The screenshot shows a Microsoft Visual Studio Debug window. The title bar reads "Microsoft Visual Studio Debug". The main area displays a list of 30 entries, each consisting of a number followed by a colon and a time value in milliseconds (ms). The values increase exponentially from entry 10 to entry 30, illustrating a catastrophic return value.

Index	Time (ms)
10	6.42ms
11	0.26ms
12	0.51ms
13	1.01ms
14	2.03ms
15	3.98ms
16	8.16ms
17	15.82ms
18	57.21ms
19	65.99ms
20	170.13ms
21	252.93ms
22	498.04ms
23	991.92ms
24	1,993.29ms
25	4,133.37ms
26	7,922.43ms
27	15,911.22ms
28	31,745.34ms
29	63,421.00ms
30	126,841.73ms

# Катастрофический возврат

90

```
using System.Diagnostics;
using System.Text.RegularExpressions;

var sw = new Stopwatch();
for (int i = 10; i <= 30; i++)
{
    var r = new Regex($@"^(\w\d|\d\w){{{{i}}}}$",
        RegexOptions.NonBacktracking);
    string input = new string('1', (i * 2) + 1);

    sw.Restart();
    r.IsMatch(input);
    sw.Stop();
    Console.WriteLine($"{i}: {sw.Elapsed.TotalMilliseconds:N}ms");
}
```

# Катастрофический возврат

91

```
using System.Diagnostics;
using System.Text.RegularExpressions;

var sw = new Stopwatch();
for (int i = 10; i <= 30; i++)
{
    var r = new Regex($@"^(\w\d|\d\w){{{i}}}\$",
        RegexOptions.NonBacktracking);
    string input = new string('1', (i * 2) + 1);

    sw.Restart();
    r.IsMatch(input);
    sw.Stop();
    Console.WriteLine($"{i}: {sw.Elapsed.TotalMilliseconds:N}ms");
}
```

# Катастрофический возврат

92

```
using System.Diagnostics;
using System.Text.RegularExpressions;

var sw = new Stopwatch();
for (int i = 10; i <= 30; i++)
{
    var r = new Regex($@"^(\w\d|\d\w){{{i}}}$", RegexOptions.NonBacktracking);
    string input = new string('1', (i * 2) + 1);

    sw.Restart();
    r.IsMatch(input);
    sw.Stop();
    Console.WriteLine($"{i}: {sw.Elapsed.TotalMilliseconds:N}ms");
}
```



# Катастрофический возврат

93

```
Microsoft Visual Studio Debug X + | - | X  
10: 7.81ms  
11: 0.09ms  
12: 0.08ms  
13: 0.08ms  
14: 0.09ms  
15: 0.10ms  
16: 0.11ms  
17: 0.10ms  
18: 0.09ms  
19: 0.10ms  
20: 0.12ms  
21: 0.12ms  
22: 0.12ms  
23: 0.12ms  
24: 0.13ms  
25: 0.13ms  
26: 0.14ms  
27: 0.14ms  
28: 0.15ms  
29: 0.14ms  
30: 0.15ms
```

# Преимущества обратного отслеживания

# Преимущества обратного отслеживания

- Эффективнее в «хороших» случаях

# Преимущества обратного отслеживания

- Эффективнее в «хороших» случаях
- Проверочные условия с просмотром вперед/назад

# Просмотр вперед/назад

97

- Синтаксис:  
**(?= выражение)** – положительный просмотр вперед и  
**(?! выражение)** – отрицательный просмотр вперед
- Символы, непосредственно следующие за текущим, (не) должны соответствовать **выражению**.
- Позволяет регулярному выражению работать эффективнее

# Просмотр вперед/назад

98

```
string input = "aaaaaaaaaaaaaaaaaaaaaaa.";  
bool result;  
Stopwatch sw;  
  
string pattern = @"^(([A-Z]\w*)+\.)*[A-Z]\w*$";  
sw = Stopwatch.StartNew();  
result = Regex.IsMatch(input, pattern, RegexOptions.NonBacktracking);  
sw.Stop();  
Console.WriteLine($"Без просмотра вперед: {result} in {sw.Elapsed}");  
  
string aheadPattern = @"^(?=([A-Z])\w+\.).*[A-Z]\w*$";  
sw = Stopwatch.StartNew();  
result = Regex.IsMatch(input, aheadPattern);  
sw.Stop();  
Console.WriteLine($"С просмотром вперед: {result} in {sw.Elapsed}");
```

# Просмотр вперед/назад

99

```
string input = "aaaaaaaaaaaaaaaaaaaaaaa.";  
bool result;  
Stopwatch sw;  
  
string pattern = @"^((([A-Z]\w*)+\.)*[A-Z]\w*$/";  
sw = Stopwatch.StartNew();  
result = Regex.IsMatch(input, pattern, RegexOptions.NonBacktracking);  
sw.Stop();  
Console.WriteLine($"Без просмотра вперед: {result} in {sw.Elapsed}");  
  
string aheadPattern = @"^(?=([A-Z])\w+\.).*[A-Z]\w*$/";  
sw = Stopwatch.StartNew();  
result = Regex.IsMatch(input, aheadPattern);  
sw.Stop();  
Console.WriteLine($"С просмотром вперед: {result} in {sw.Elapsed}");
```

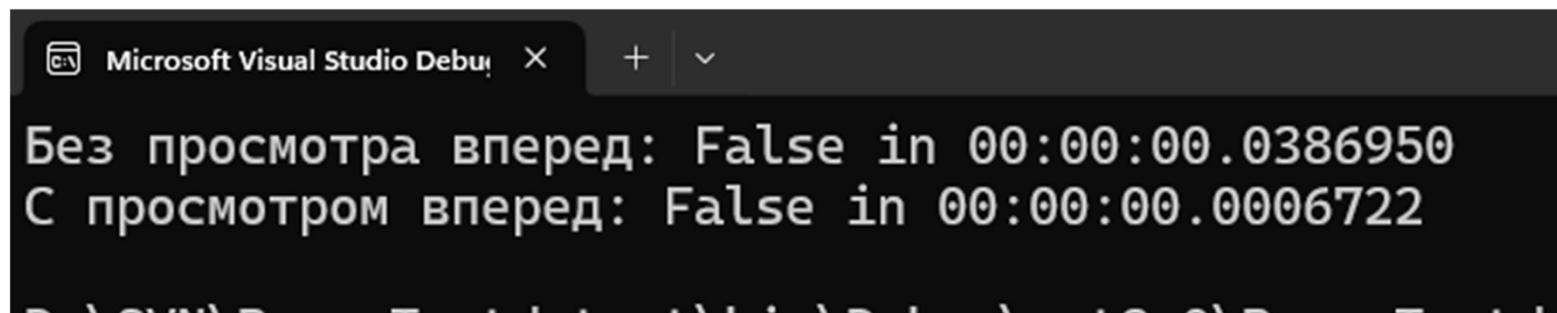
# Просмотр вперед/назад

100

```
string input = "aaaaaaaaaaaaaaaaaaaaaaa.";  
bool result;  
Stopwatch sw;  
  
string pattern = @"^(([A-Z]\w*)+\.)*[A-Z]\w*$";  
sw = Stopwatch.StartNew();  
result = Regex.IsMatch(input, pattern, RegexOptions.NonBacktracking);  
sw.Stop();  
Console.WriteLine($"Без просмотра вперед: {result} in {sw.Elapsed}");  
  
string aheadPattern = @"^((?=([A-Z])\w+)\.)*[A-Z]\w*$";  
sw = Stopwatch.StartNew();  
result = Regex.IsMatch(input, aheadPattern);  
sw.Stop();  
Console.WriteLine($"С просмотром вперед: {result} in {sw.Elapsed}");
```

# Просмотр вперед/назад

101



Microsoft Visual Studio Debug

Без просмотра вперед: False in 00:00:00.0386950  
С просмотром вперед: False in 00:00:00.0006722

# Преимущества обратного отслеживания

- Эффективнее в «хороших» случаях
- Проверочные условия с просмотром вперед/назад
- Обратные ссылки

- Синтаксис: `\число`

Где `число` – порядковый номер захватываемой группы в регулярном выражении

- Предоставляют удобный способ идентификации повторяющегося символа или подстроки в строке

# Обратные ссылки

104

```
string pattern = @"\w\1";
string input = "trellis llama webbing dresser swagger";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine($"Found '{match.Value}' at position {match.Index}.");
```

# Обратные ссылки

105

```
string pattern = @"(\w)\1";
string input = "trellis llama webbing dresser swagger";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine($"Found '{match.Value}' at position {match.Index}.");
```

# Обратные ссылки

106

```
string pattern = @"(\w)\1";
string input = "trellis llama webbing dresser swagger";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine($"Found '{match.Value}' at position {match.Index}.");
```

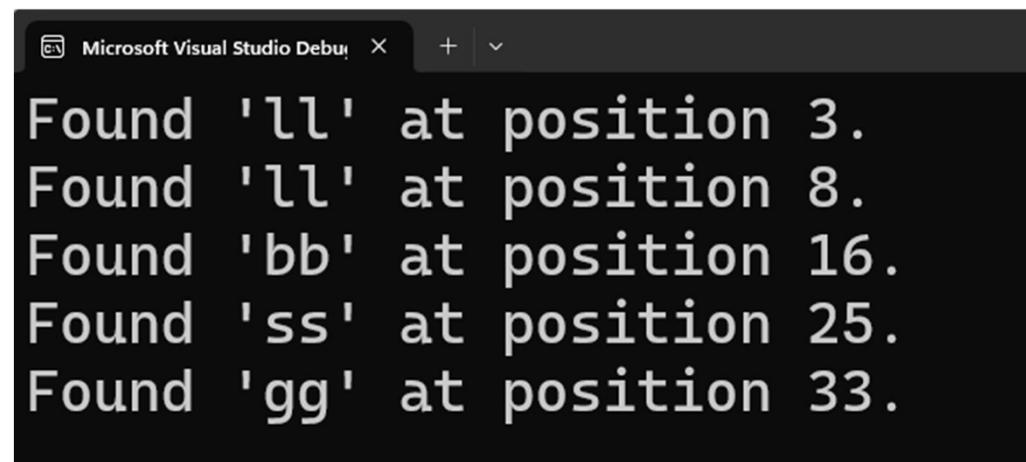


Первая захватываемая группа

# Обратные ссылки

107

```
string pattern = @"\w\1";
string input = "trellis llama webbing dresser swagger";
foreach (Match match in Regex.Matches(input, pattern))
    Console.WriteLine($"Found '{match.Value}' at position {match.Index}.");
```



The screenshot shows the Microsoft Visual Studio Debug console window. The title bar reads "Microsoft Visual Studio Debug". The console output displays five lines of text, each indicating a found match with its value and index:

```
Found 'll' at position 3.
Found 'll' at position 8.
Found 'bb' at position 16.
Found 'ss' at position 25.
Found 'gg' at position 33.
```

# Преимущества обратного отслеживания

- Эффективнее в «хороших» случаях
- Проверочные условия с просмотром вперед/назад
- Обратные ссылки
- Условное сопоставление с выражением

# Условное сопоставление с выражением

- Синтаксис:  
**(?( выражение) да) или (?( выражение) да\нет)**
- Выбирает шаблон для регулярного выражения в зависимости от того, возможно ли сопоставить изначальное **выражение**
- Позволяет регулярному выражению работать более эффективно

# Условное сопоставление с выражением

110

```
Stopwatch sw;
string input = "01-9999999 020-333333 777-88-9999";
bool result;

string pattern = @"\d{2}-\d{7}|\d{3}-\d{2}-\d{4}";
sw = Stopwatch.StartNew();
result = Regex.IsMatch(input, pattern);
sw.Stop();
Console.WriteLine($"Обычное сопоставление: {result} in {sw.Elapsed}");

string behindPattern = @"\b(?:(\d{2}-)\d{2}-\d{7}|\d{3}-\d{2}-\d{4})\b";
sw = Stopwatch.StartNew();
result = Regex.IsMatch(input, behindPattern);
sw.Stop();
Console.WriteLine($"Условное сопоставление: {result} in {sw.Elapsed}");
```

# Условное сопоставление с выражением

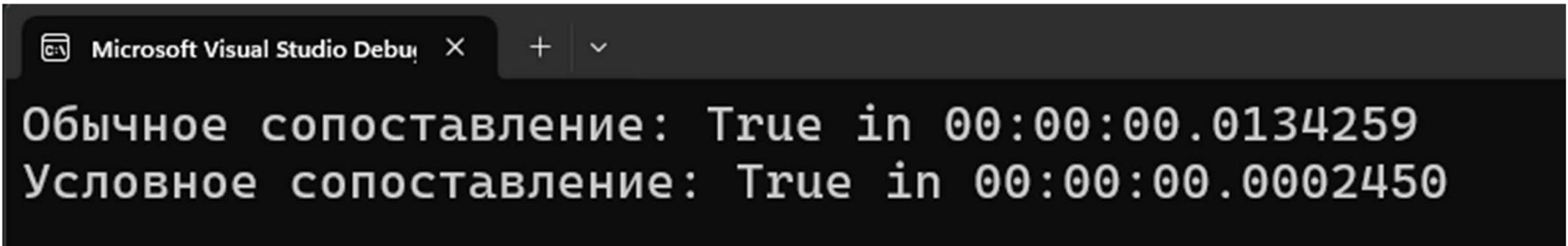
```
Stopwatch sw;
string input = "01-9999999 020-333333 777-88-9999";
bool result;

string pattern = @"\"d{2}-\"d{7} | \"d{3}-\"d{2}-\"d{4}"";
sw = Stopwatch.StartNew();
result = Regex.IsMatch(input, pattern);
sw.Stop();
Console.WriteLine($"Обычное сопоставление: {result} in {sw.Elapsed}");

string behindPattern = @"\b(?(\\"d{2}-)\"d{2}-\"d{7} | \"d{3}-\"d{2}-\"d{4})\b";
sw = Stopwatch.StartNew();
result = Regex.IsMatch(input, behindPattern);
sw.Stop();
Console.WriteLine($"Условное сопоставление: {result} in {sw.Elapsed}");
```

# Условное сопоставление с выражением

112



Microsoft Visual Studio Debug X + ▾

Обычное сопоставление: True in 00:00:00.0134259  
Условное сопоставление: True in 00:00:00.0002450

# Преимущества обратного отслеживания

113

# Преимущества обратного отслеживания

- Позволяет быстрее работать в «хороших» случаях

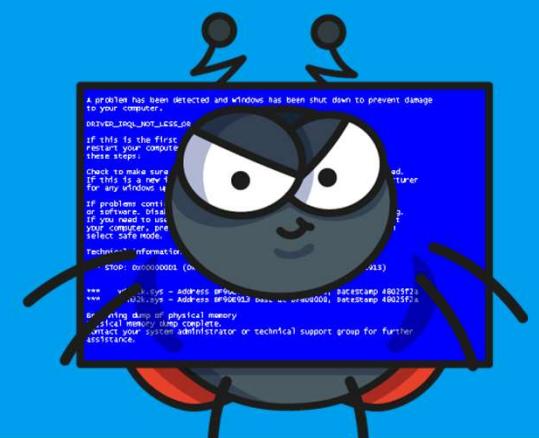
# Преимущества обратного отслеживания

- Позволяет быстрее работать в «хороших» случаях
- Могут быстрее работать в «плохих» случаях

# Преимущества обратного отслеживания

- Позволяет быстрее работать в «хороших» случаях
- Могут быстрее работать в «плохих» случаях
- Предоставляет более мощный функционал

# Уязвимости на реальных проектах



# ScreenToGif

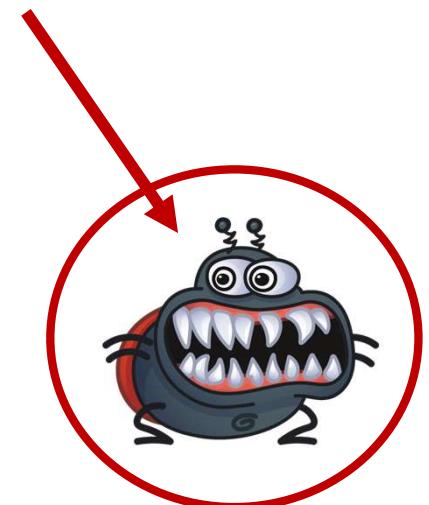
118

```
public static string ReplaceRegexInName(string name)
{
    const string dateTimeFileNameRegEx = @"[?]( [y whole month] [m whole month] [d whole day] [h hour] [m minute] [s second] [f fraction] [z zone] [- dash] [_ underscore] *)+[?]";
    if (!Regex.IsMatch(name,
        dateTimeFileNameRegEx,
        RegexOptions.IgnoreCase))
        return name;
    var match = Regex.Match(name,
        dateTimeFileNameRegEx,
        RegexOptions.IgnoreCase);
    ....
}
```

# ScreenToGif

119

```
public static string ReplaceRegexInName(string name)
{
    const string dateTimeFileNameRegEx = @"[?]( [y whole month] [m whole month] [d whole day] [h hour] [m minute] [s second] [f fraction] [z zone] [- dash] [_ underscore] *)+[?]";
    if (!Regex.IsMatch(name,
        dateTimeFileNameRegEx,
        RegexOptions.IgnoreCase))
        return name;
    var match = Regex.Match(name,
        dateTimeFileNameRegEx,
        RegexOptions.IgnoreCase);
    ...
}
```



# ScreenToGif

120

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

# ScreenToGif

121

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

[?] ([yndhsfzgkt]+)+[?]

# ScreenToGif

122

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

[?] ([yndhsfzgkt]+)+[?]

(a+) + b

# ScreenToGif

123

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

[?] ([yndhsfzgkt]+)+[?]

(a+) + b  
↓

# ScreenToGif

124

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

[?] ([yndhsfzgkt]+)+[?]

(a+)+b

# ScreenToGif

125

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

[?] ([yndhsfzgkt]+)+[?]

# ScreenToGif

126

[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

[?] ([yndhsfzgkt]+)+[?]

0 вхождений [-\_ ]

[?] ([ymdh~~s~~fzgkt]+[-\_ ]\*)+[?]



[?] ([ymdh~~s~~fzgkt]+)+[?]

0 вхождений [-\_ ]



# ScreenToGif

128

?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgkt

# ScreenToGif

129

?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgk



[?] ([yndhsfzgkt]+[-\_ ]\*)+[?]

?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgkt



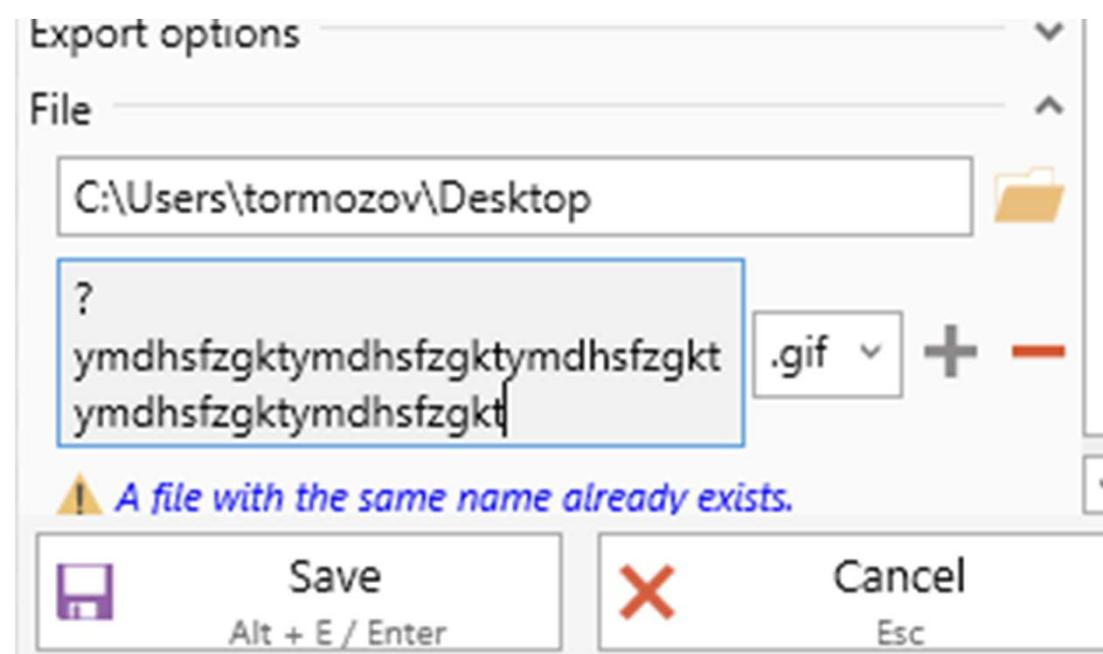
[?] ([ymdhsfzgkt]+[-\_ ]\*)+[?]



**ReDoS**

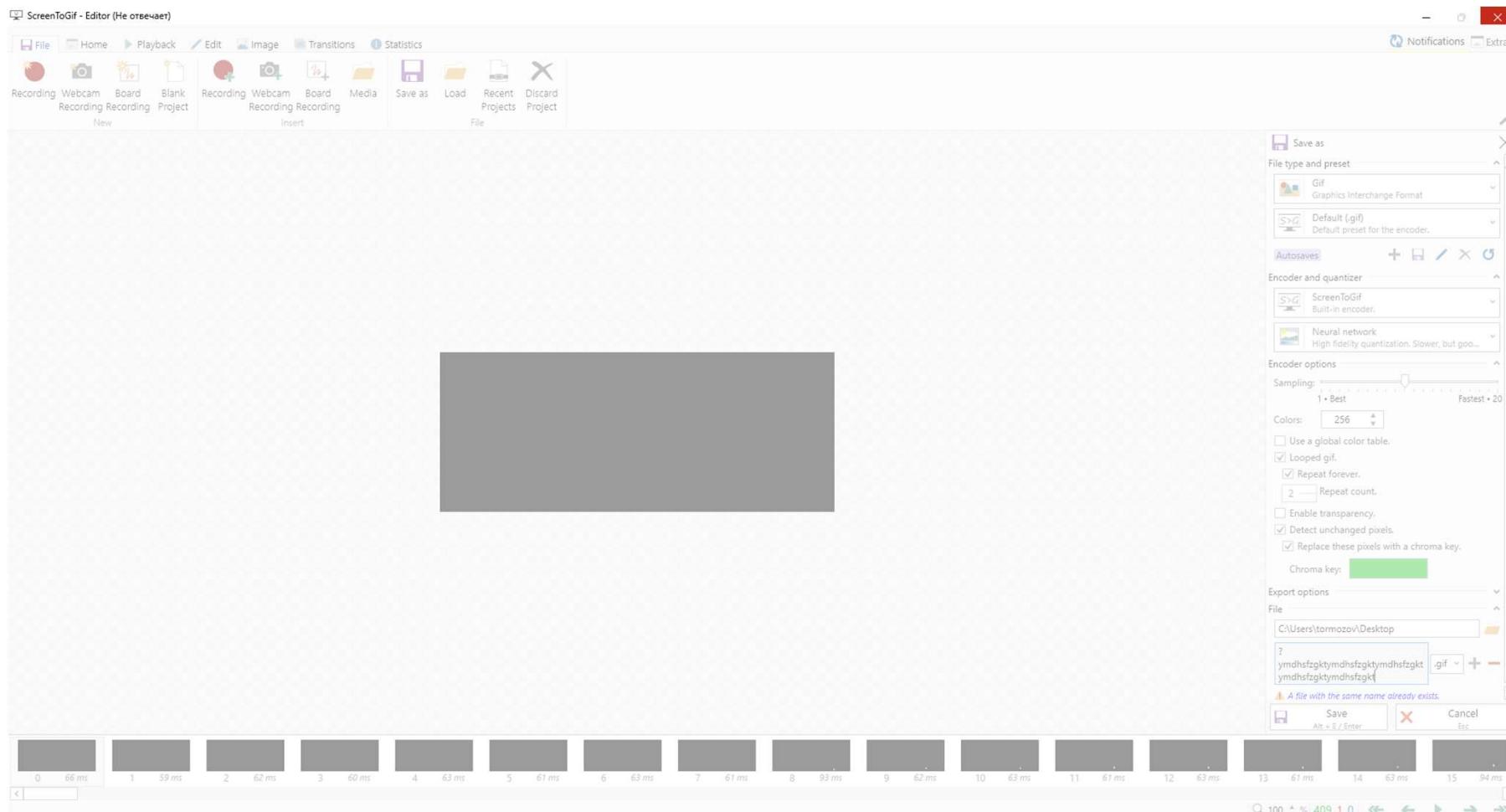
# ScreenToGif

131



# ScreenToGif

132



- CVE-2021-27293

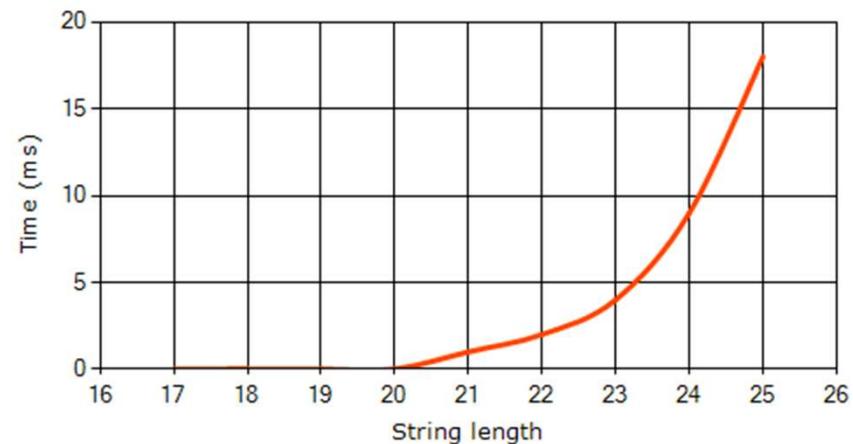
- CVE-2021-27293
- Regex: "**newDate\((-?\d+)\*\)**"
- Ищет строку формата:  
*newDate(10-01-2007)*

- CVE-2021-27293
- Regex: "**newDate\((-?\d+)\*\)**"
- Ищет строку формата:  
*newDate(10-01-2007)*

**Input strings:**

```
newDate(1111111111
newDate(11111111112
newDate(111111111133
newDate(1111111111444
newDate(11111111115555
newDate(111111111166666
newDate(1111111111777777
newDate(11111111118888888
newDate(1111111111999999999
```

**Effect of input string length on regular expression execution time.**



# System.ComponentModel.DataAnnotations

136

- CVE-2015-2526

# System.ComponentModel.DataAnnotations

137

- CVE-2015-2526
- Классы  
EmailAddressAttribute  
и PhoneAttribute
- Попытка валидации  
email и номера  
телефона

# System.ComponentModel.DataAnnotations

138

- CVE-2015-2526
- Классы  
EmailAddressAttribute  
и PhoneAttribute
- Попытка валидации  
email и номера  
телефона

## email validation

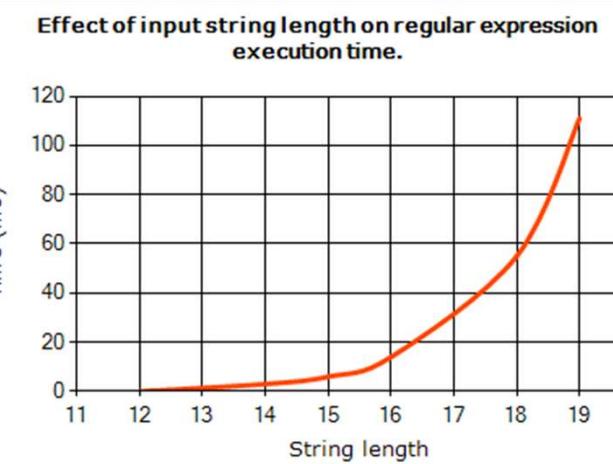
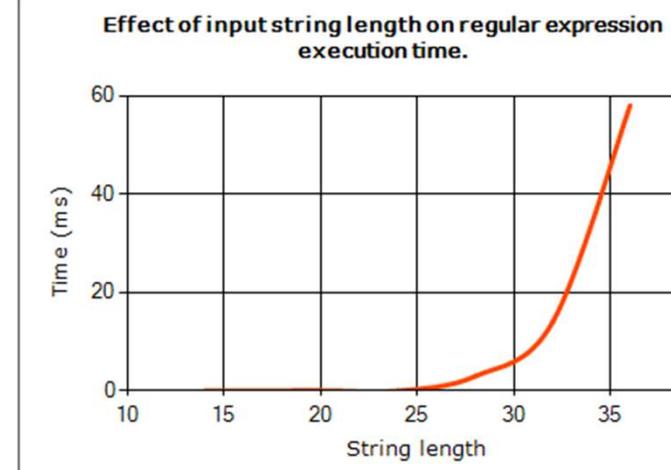
Input strings:

```
t@t.t.c%20
t@t.t.t.t.c%20
t@t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.t.t.c%20
```

## phone validation

Input strings:

```
66666666666d
6666666666666d
66666666666666d
666666666666666d
6666666666666666d
66666666666666666d
```



# ReDoS вне C# кода

139

# ReDoS вне C# кода

140

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

# ReDoS вне C# кода

141

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

(?:(?:\"|'|\\]|\\}|\\\\|\\d|(?::nan|infinity|true|false|null|undefined|symbol|math)|\\`|\\-|\\+)+[])\*;?((?:\\s|-|~|!|{}|\\||\\||\\+)\*.\*(?:.\*=.\*)))



# ReDoS вне C# кода

142

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

DOS в результате обработки опасным регулярным выражением  
вопроса, содержащего 20 000 символов пробела

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

DOS в результате обработки опасным регулярным выражением  
вопроса, содержащего 20 000 символов пробела

## Search Results

Showing 1 - 200 of 452 results for **regular expression denial of service**

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

DOS в результате обработки опасным регулярным выражением  
вопроса, содержащего 20 000 символов пробела

## Search Results

Showing 1 - 200 of 452 results for **regular expression denial of service**

Из них за:

- 2023: 39 уязвимостей
- 2024: 44 уязвимостей

# Как бороться с ReDoS



# Как бороться с ReDoS

146

- Использовать другое регулярное выражение



# Как бороться с ReDoS

147

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком



# Как бороться с ReDoS

148

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком
- Разбивать регулярное выражение



# Как бороться с ReDoS

149

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком
- Разбивать регулярное выражение
- Быть внимательным к сгенерированным в runtime и основанном на пользовательском вводе регулярном выражении



# ReDoS via injection

150

```
String login = loginTB.Text;
String password = passwordTB.Text;
Regex testPassword = new Regex(login);
Match match = testPassword.Match(password);
if (match.Success)
{
    MessageBox.Show("Don't include login in password.");
}
else
{
    MessageBox.Show("Good password.");
}
```

# ReDoS via injection

151

```
String login = loginTB.Text;
String password = passwordTB.Text;
Regex testPassword = new Regex(login);
Match match = testPassword.Match(password);
if (match.Success)
{
    MessageBox.Show("Don't include login in password.");
}
else
{
    MessageBox.Show("Good password.");
}
```

**Пользовательский ввод:**

login: (a\*)<sup>\*</sup>b

password:aaaaaaaaaaaaaa!

# ReDoS via injection

152

```
String login = loginTB.Text;
String password = passwordTB.Text;
Regex testPassword = new Regex(login);
Match match = testPassword.Match(password);
if (match.Success)
{
    MessageBox.Show("Don't include login in password.");
}
else
{
    MessageBox.Show("Good password.");
}
```

Пользовательский ввод:  
login: (a\*)\*b  
password:aaaaaaaaaaaaaa!

Результат – ReDoS!



# Как бороться с ReDoS

153

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком
- Разбивать регулярное выражение
- Быть внимательным к сгенерированным в runtime и основанном на пользовательском вводе регулярном выражении
- Использовать timeout



# Использование timeout

154

Третий аргумент конструктора регулярных выражений – **matchTimeout**

В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

# Использование timeout

Третий аргумент конструктора регулярных выражений – **matchTimeout**

В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

```
Regex regex = new Regex("(a*)*b",
    RegexOptions.None,
    TimeSpan.FromSeconds(2));
```

# Использование timeout

156

Третий аргумент конструктора регулярных выражений – **matchTimeout**

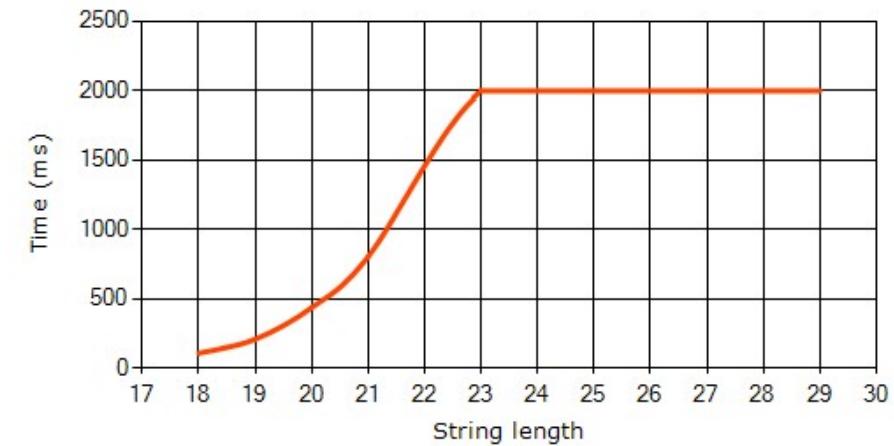
В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

```
Regex regex = new Regex("(a*)*b",
    RegexOptions.None,
    TimeSpan.FromSeconds(2));
```

**Input strings:**

```
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
```

Effect of input string length on regular expression execution time.



# Как бороться с ReDoS

157

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком
- Разбивать регулярное выражение
- Быть внимательным к сгенерированным в runtime и основанном на пользовательском вводе регулярном выражении
- Использовать timeout
- Использовать атомарные группы



# Атомарные группы

158

- Синтаксис определения атомарных групп:  
**(?>...)**
- Для выражения внутри отключается функция обратного отслеживания
- **Внимание!** Использование атомарных групп меняет логику обработки соответствий

# Атомарные группы

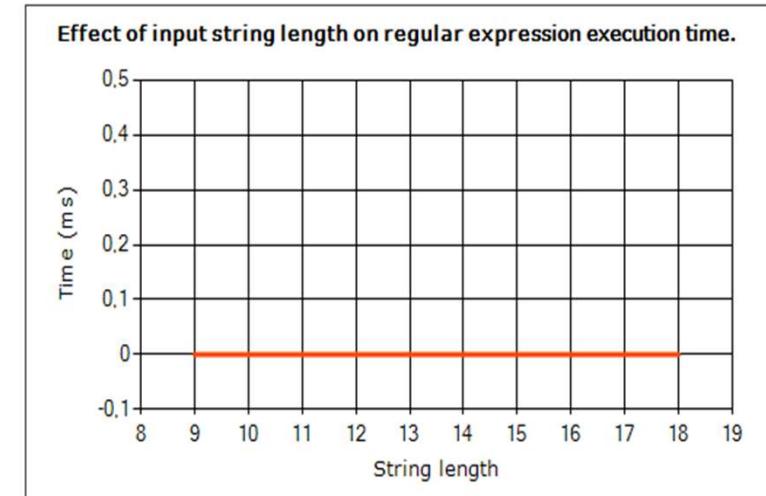
159

- Синтаксис определения атомарных групп:  
 $(?>...)$
- Для выражения внутри отключается функция обратного отслеживания
- Внимание!** Использование атомарных групп меняет логику обработки соответствий

Input strings:

$(?>a^*)^*b$

```
aaaaaaaaaa
aaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
```



# Как бороться с ReDoS

160

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком
- Разбивать регулярное выражение
- Быть внимательным к сгенерированным в runtime и основанном на пользовательском вводе регулярном выражении
- Использовать timeout
- Использовать атомарные группы
- Использовать движок Regex на основе DFA



# DFA Regex engine

161

С версии .NET 7 вы можете указать  
флаг `RegexOptions.NonBacktracking`

# DFA Regex engine

162

С версии .NET 7 вы можете указать флаг `RegexOptions.NonBacktracking`

```
Regex regex = new Regex(@"newDate\((-?\d+)*\")",
    RegexOptions.NonBacktracking);
```

В таком случае для разбора будет использован основанный на DFA движок регулярных выражений

# DFA Regex engine

163

С версии .NET 7 вы можете указать флаг `RegexOptions.NonBacktracking`

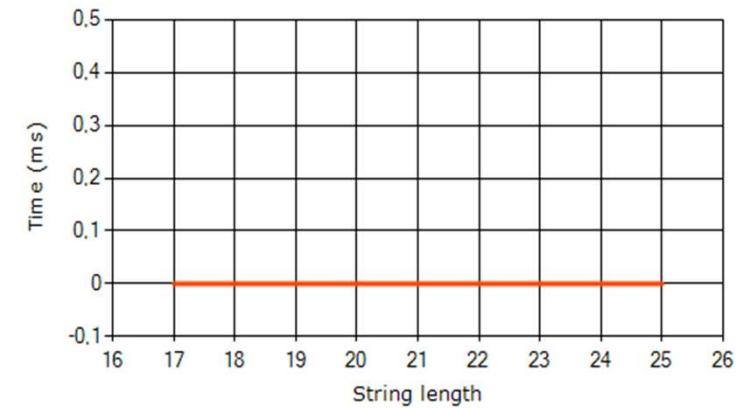
```
Regex regex = new Regex(@"newDate\((-?\d+)*\")",
    RegexOptions.NonBacktracking);
```

В таком случае для разбора будет использован основанный на DFA движок регулярных выражений

Input strings:

```
newDate(1111111111
newDate(11111111112
newDate(111111111133
newDate(1111111111444
newDate(11111111115555
newDate(111111111166666
newDate(1111111111777777
newDate(11111111118888888
newDate(1111111111999999999
```

Effect of input string length on regular expression execution time.



# Профилактика ReDoS уязвимостей



# Профилактика ReDoS уязвимостей

165

- Проверять чужие регулярные выражения



# Доверяй, но проверяй

Regular Expression Details

<input checked="" type="checkbox"/> Title	RegEx for email validation	<input type="button" value="Find"/>	<input type="button" value="Test"/>
<input checked="" type="checkbox"/> Expression	/^([a-zA-Z0-9])(([\\-\\.] [_]+)?([a-zA-Z0-9]+))*(@){1}[a-zA-Z]+[.]{1}(1){1}(([a-zA-Z]{2,3}) ([a-zA-Z]{2,3}[.]{1}[a-zA-Z]{2,3}))\$/		
<b>Description</b>	This expression will validate all possible formats except if web site URL contains hyphen characters like aa@a-b-c.com. I will include this feature also in next version.		
<b>Matches</b>	a@abc.com,a@abc.co.in,aa.bb@abc.com.sg,aa_bb@abc.biz,aa.C.bb@abc.co m,aa_1900@abc.co.in		
<b>Non-Matches</b>	@abc.com,a@abc.co.in.in,a@abc.com.in.in,a@a-b-c.com		
<input checked="" type="checkbox"/> Author	Rafiq	Rating:	<div style="width: 100px; height: 10px; background-color: green;"></div>
Source	Email		
<input checked="" type="checkbox"/> Your Rating	Bad	<input type="radio"/> 1 <input type="radio"/> 2 <input type="radio"/> 3 <input type="radio"/> 4 <input type="radio"/> 5 Good	<input type="button" value="Submit Rating"/>

# Доверяй, но проверяй

167

Regular Expression Details

Title RegEx for email validation

Expression `/^([a-zA-Z0-9])(([\\-\\.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-zA-Z]+[.]{1}{1}(([a-zA-Z]{2,3})|([a-zA-Z]{2,3}[.]{1}[a-zA-Z]{2,3}))$/`

**Description** This expression will validate all possible formats except if web site URL contains hyphen characters like aa@a-b-c.com. I will include this feature also in next version.

**Matches** a@abc.com,a@abc.co.in,aa.bb@abc.com.sg,aa\_bb@abc.biz,aa.C.bb@abc.co.m,aa\_1900@abc.co.in

**Non-Matches** @abc.com,a@abc.co.in.in,a@abc.com.in.in,a@a-b-c.com

Author Rafiq "/>

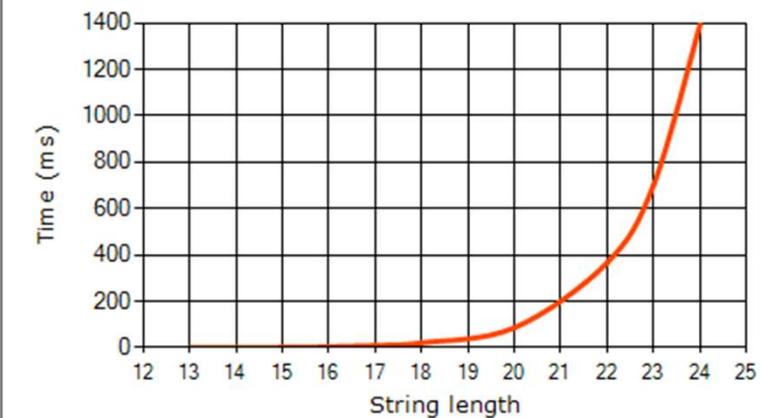
Source Email

Your Rating Bad  1  2  3  4  5 Good

**Input strings:**

aaaaaaaaaaaa!  
aaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!  
aaaaaaaaaaaaaaa!

Effect of input string length on regular expression execution time.



# Профилактика ReDoS уязвимостей

168

- Проверять чужие регулярные выражения
- Использовать утилиты для проверки регулярных выражений



# Профилактика ReDoS уязвимостей

169

- Проверять чужие регулярные выражения
- Использовать утилиты для проверки регулярных выражений
- Обновлять уязвимые зависимости



# Профилактика ReDoS уязвимостей

170

- Проверять чужие регулярные выражения
- Использовать утилиты для проверки регулярных выражений
- Обновлять уязвимые зависимости
- Использовать SAST решения для анализа кода



# PVS-Studio

## Статический анализатор кода (SAST)

- Статический анализ и SAST
- Языки C, C++, C#, Java
- Поддержка стандартов безопасности и защищенности
- Интегрируется в большинство популярных IDE и CI/CD
- B2B, 17 лет на рынке



Сделай свой проект чистым  
и безопасным вместе  
с PVS-Studio



Самые интересные  
и актуальные новости  
нашего C# анализатора  
и не только



# Q/A