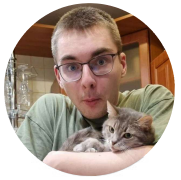

PVS-Studio

Подводные камни регулярных выражений

Уязвимости, катастрофический возврат и ReDoS-атаки

Георгий Тормозов

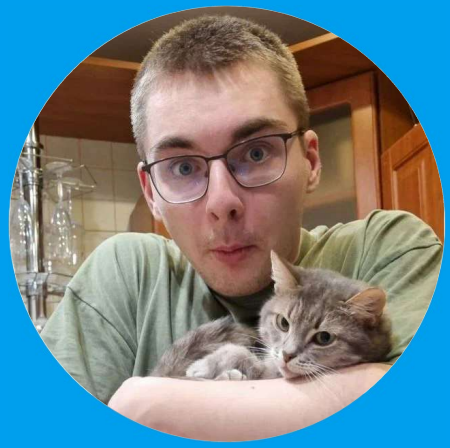
C# разработчик



Георгий Тормозов

C# разработчик

- Занимаюсь разработкой статического анализатора
- Пишу статьи
- Люблю анекдоты



PVS-Studio

План

- Что же такое ReDoS
- ReDoS на реальных проектах
- Как бороться с ReDoS
- Профилактика ReDoS

Что же такое ReDoS

Что же такое ReDoS?

ReDoS – regular expression denial of service

Разновидность DoS атаки, которая осуществляется с помощью уязвимого регулярного выражения

Что же такое ReDoS?

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?

ReDoS – regular expression denial of service

Разновидность DoS атаки, которая осуществляется с помощью уязвимого регулярного выражения

Классификация: **CWE-1333**

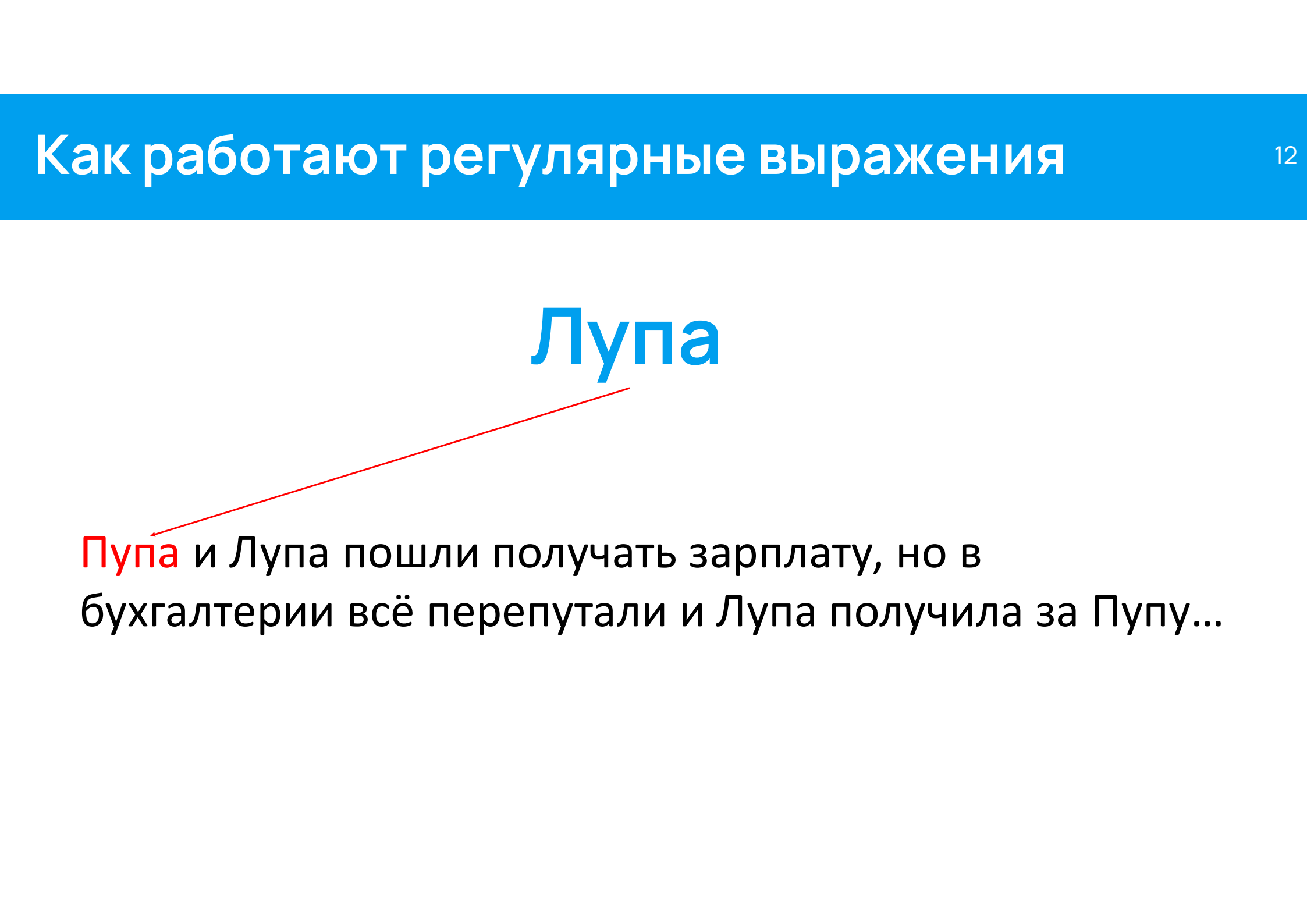
Как работают регулярные выражения

Лупа

Лупа

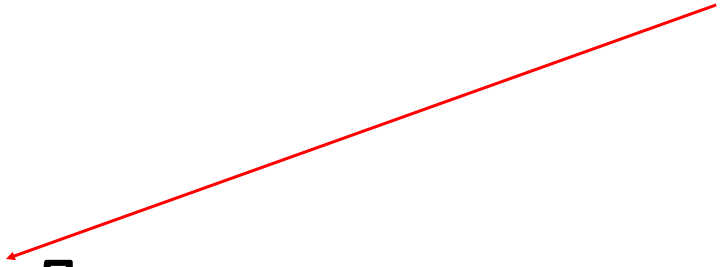
Пупа и Лупа пошли получать зарплату, но в бухгалтерии всё перепутали и Лупа получила за Пупу…

Как работают регулярные выражения



Как работают регулярные выражения

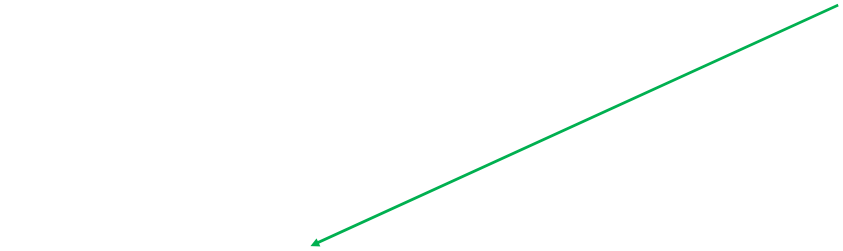
Лупа



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

Как работают регулярные выражения

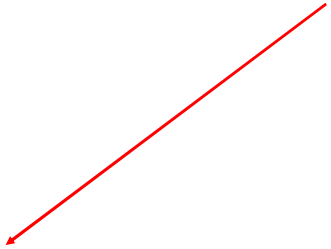
Лупа



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

Как работают регулярные выражения

Лупа



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

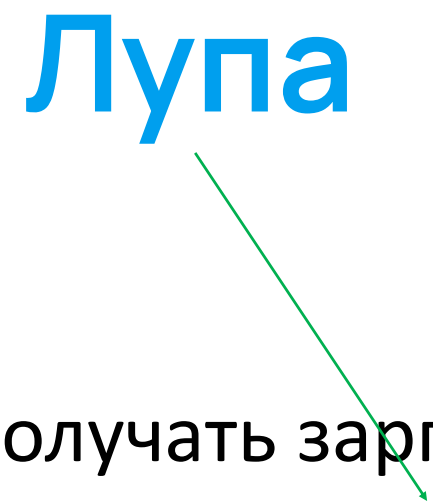
Как работают регулярные выражения

Лупа

...

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

Как работают регулярные выражения



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


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

Как работают регулярные выражения

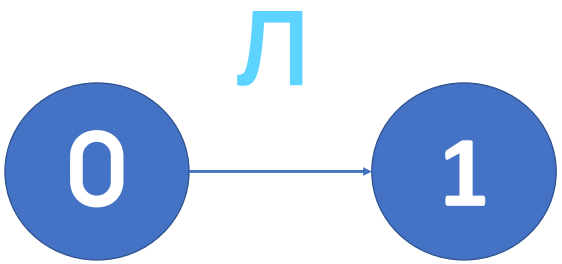
Конечные автоматы

Как работают регулярные выражения

Конечные автоматы

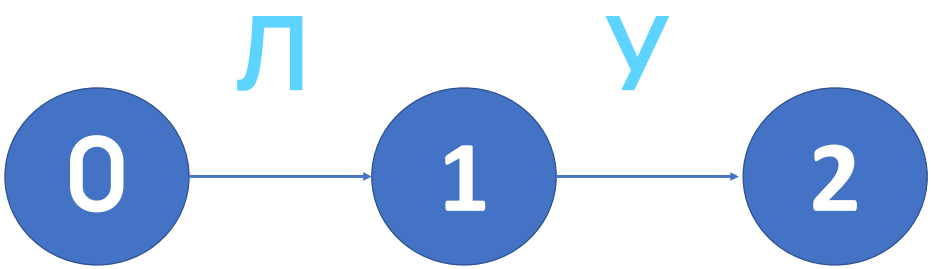
Как работают регулярные выражения

Конечные автоматы



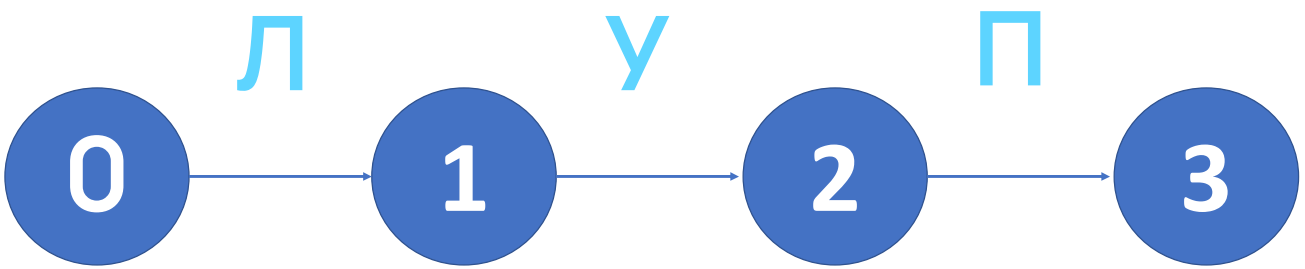
Как работают регулярные выражения

Конечные автоматы



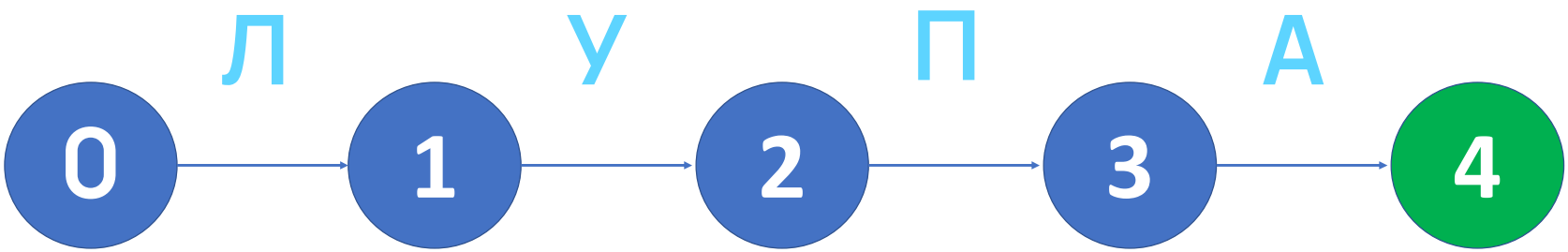
Как работают регулярные выражения

Конечные автоматы



Как работают регулярные выражения

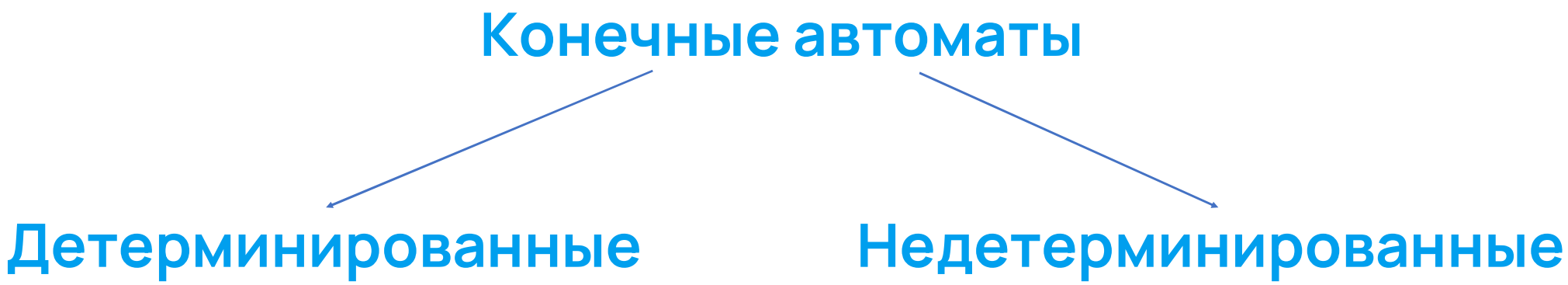
Конечные автоматы



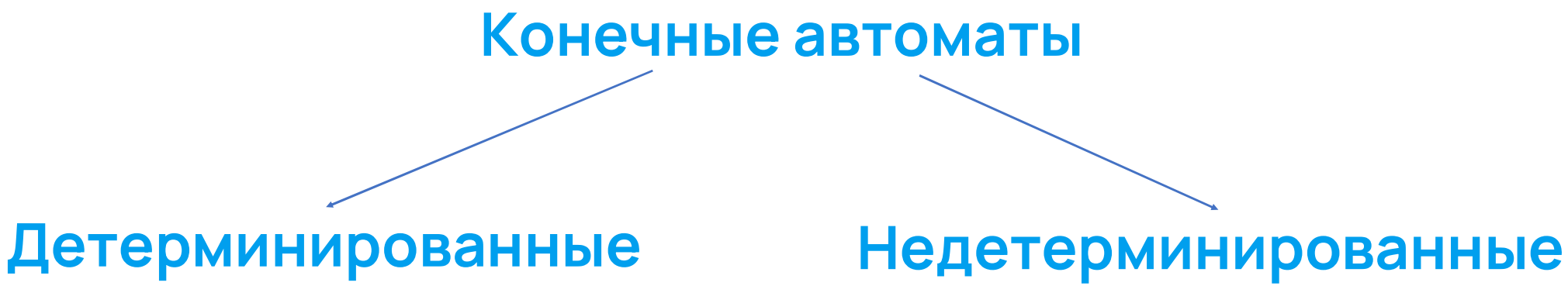
Как работают регулярные выражения

Конечные автоматы

Как работают регулярные выражения



Как работают регулярные выражения



- Быстрее работают в «худших» случаях

Как работают регулярные выражения

Конечные автоматы

Детерминированные

- Быстрее работают в «худших» случаях

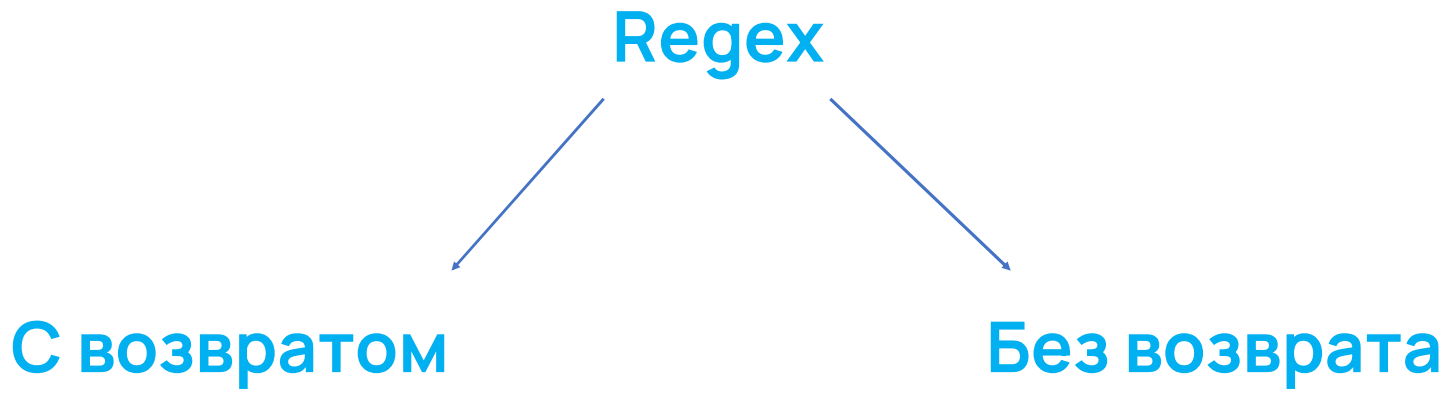
Недетерминированные

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

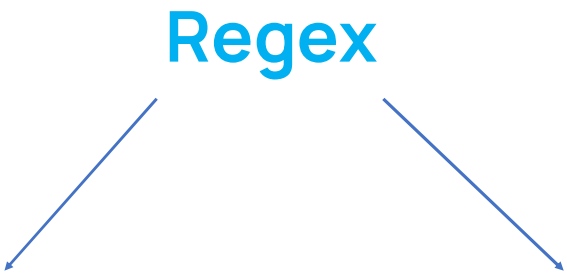
Как работают регулярные выражения

Regex

Как работают регулярные выражения



Как работают регулярные выражения



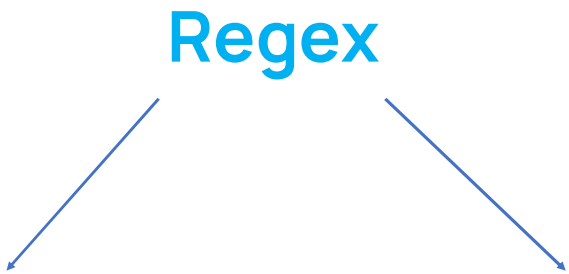
С возвратом

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

Пример: **a+b**

Без возврата

Как работают регулярные выражения



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

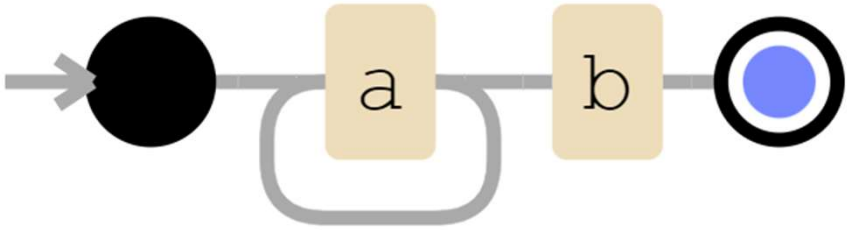
Регулярное выражение: **a+b**

Последовательность для разбора: aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

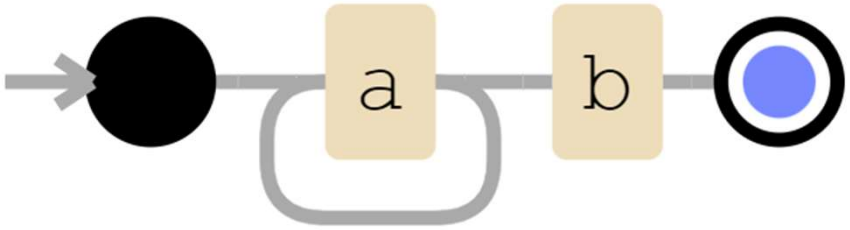
Последовательность для разбора: aaa



Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



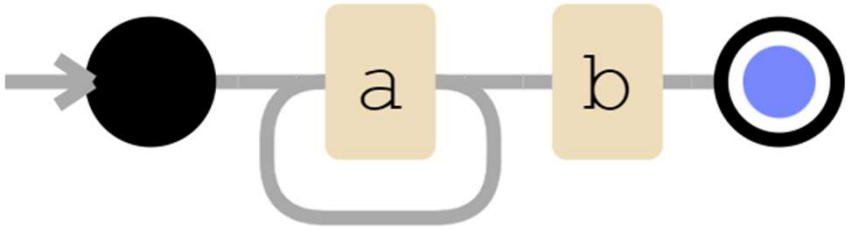

```
Regex regex = new Regex("a+b");

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

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

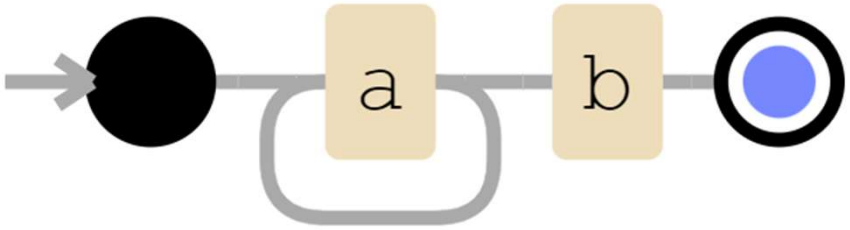


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

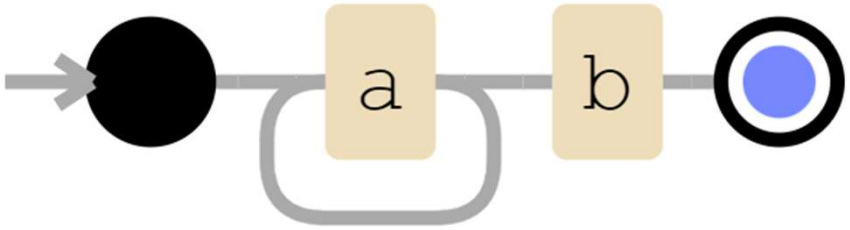


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

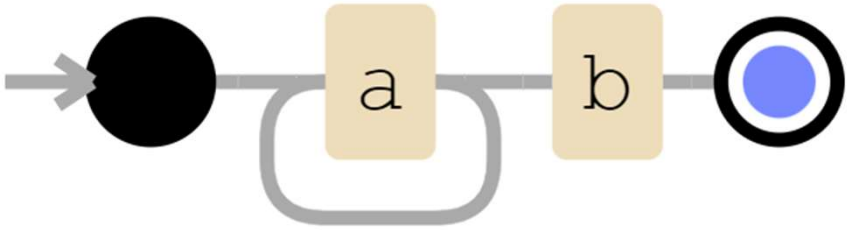


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

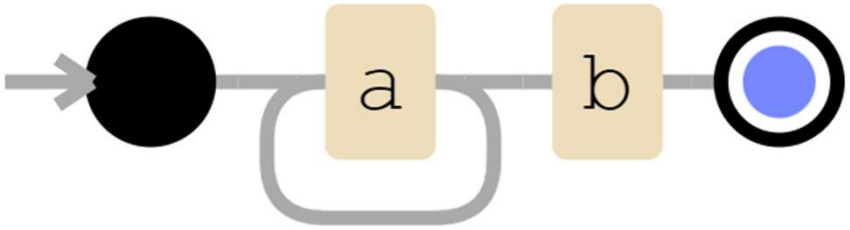


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

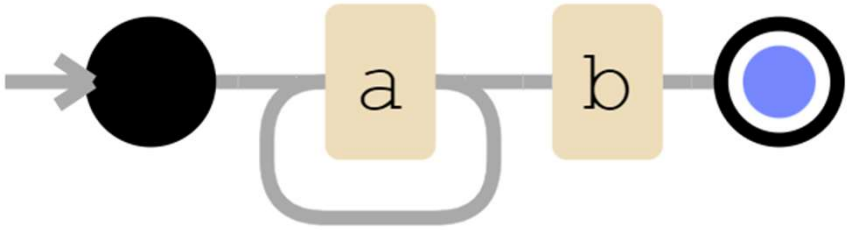


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

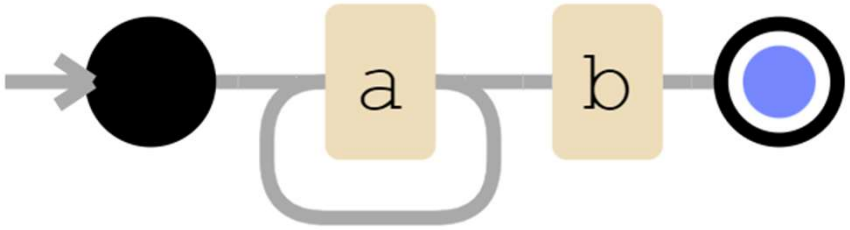


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

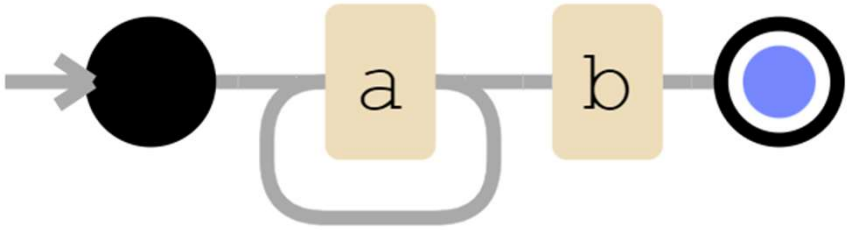


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

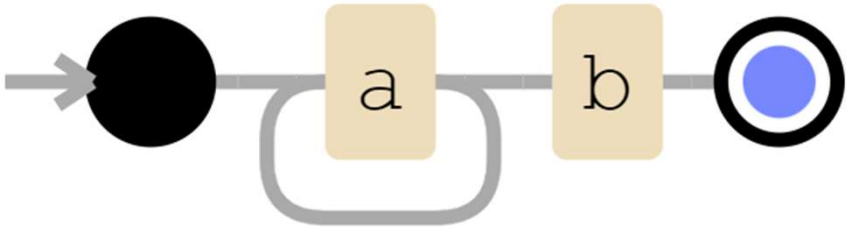


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

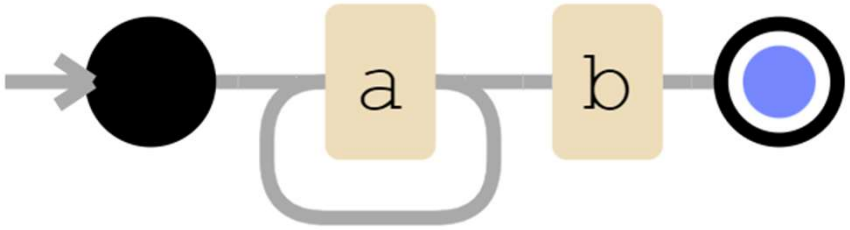


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

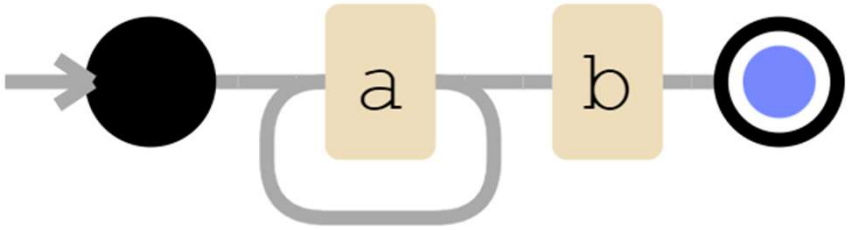


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

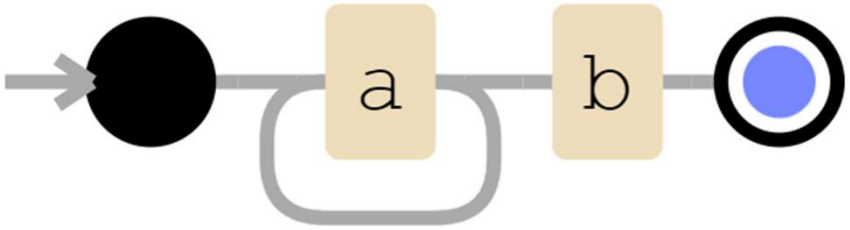


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

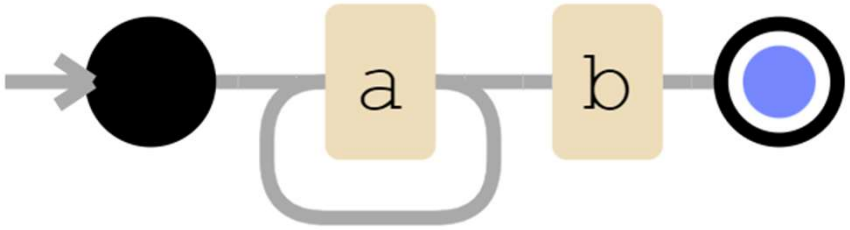


aaa

Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



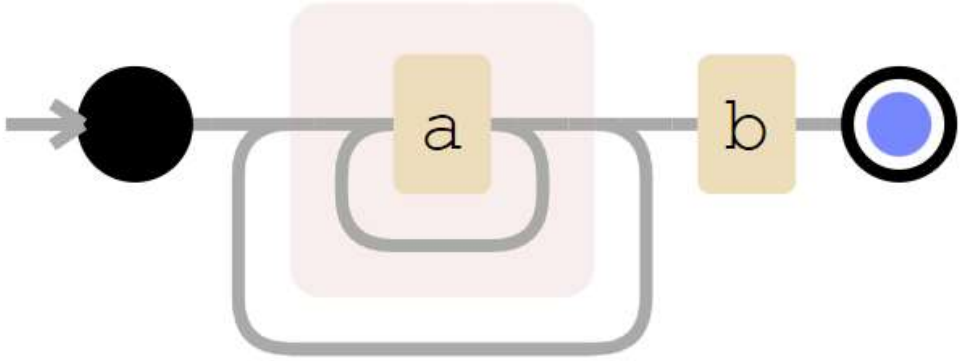
aaa

Рассмотрим пример

Регулярное выражение: **(a+)+b**

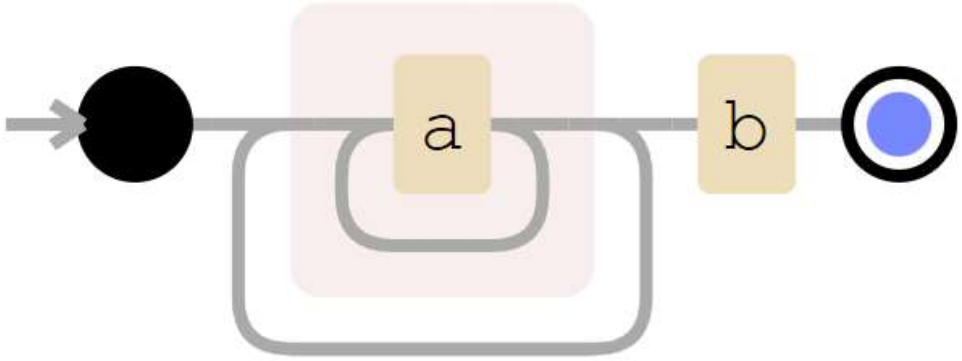
Рассмотрим пример

Регулярное выражение: **(a+)+b**



Рассмотрим пример

Регулярное выражение: **(a+)+b**



Что изменилось?

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

Сравним два регулярных выражения

a+b

Input strings:

```
aaaaaaab
aaaaaaaab
aaaaaaaaab
aaaaaaaaaaab
aaaaaaaaaaaaab
aaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaaab
```

(a+)+b

Input strings:

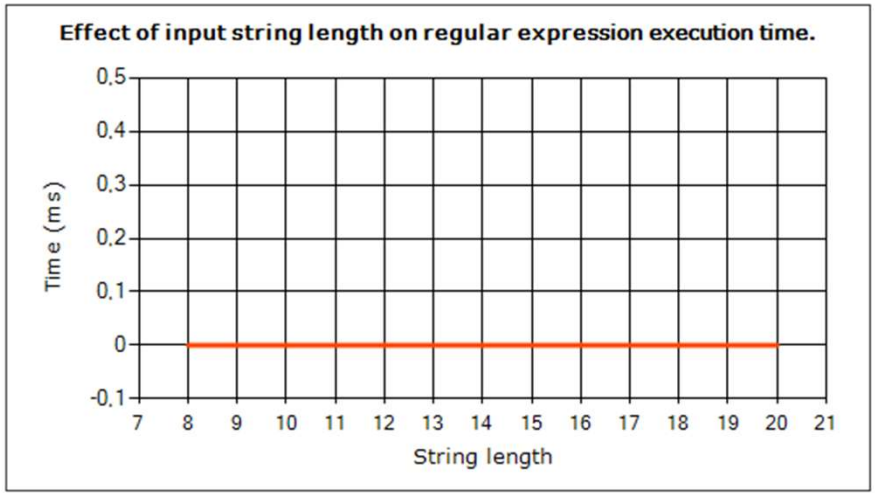
```
aaaaaaab
aaaaaaaab
aaaaaaaaab
aaaaaaaaaaab
aaaaaaaaaaaaab
aaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaaab
```

Сравним два регулярных выражения

a+b

Input strings:

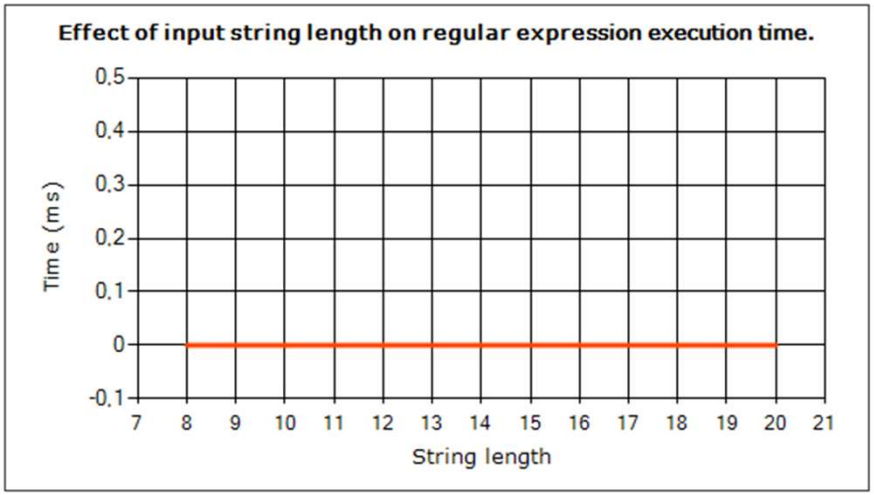
```
aaaaaaab
aaaaaaaab
aaaaaaaaab
aaaaaaaaaab
aaaaaaaaaaaab
aaaaaaaaaaaaaab
aaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaab
```



(a+)+b

Input strings:

```
aaaaaaab
aaaaaaaab
aaaaaaaaab
aaaaaaaaaaab
aaaaaaaaaaaaab
aaaaaaaaaaaaaab
aaaaaaaaaaaaaaab
aaaaaaaaaaaaaaaab
```



Сравним еще раз

a+b

Input strings:

```
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa
```

(a+)+b

Input strings:

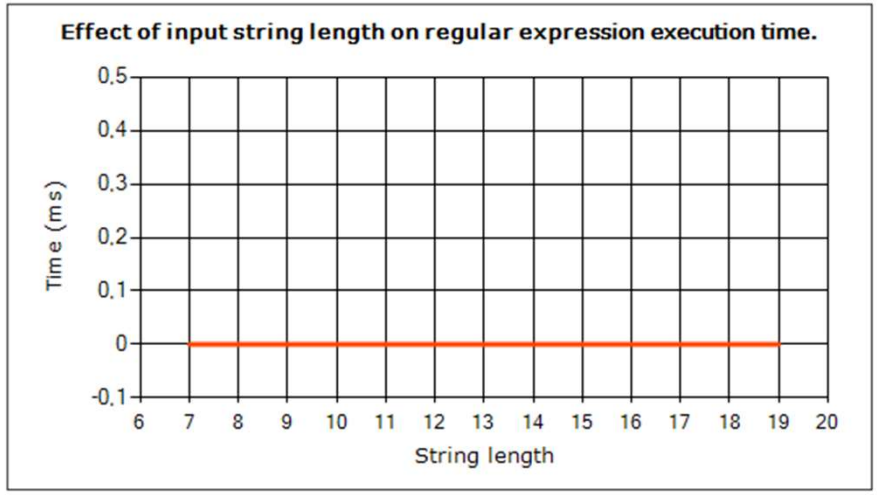
```
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa
```

Сравним еще раз

a+b

Input strings:

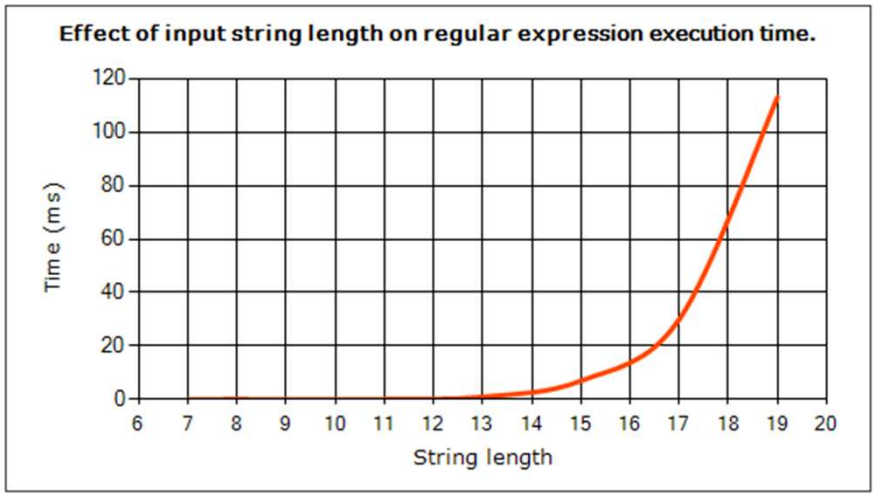
```
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa
```



(a+)+b

Input strings:

```
aaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaa
```



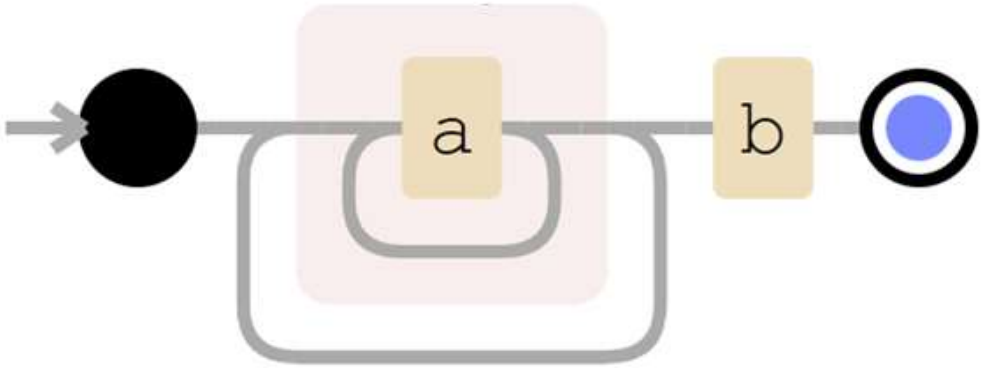
Катастрофический возврат

- 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

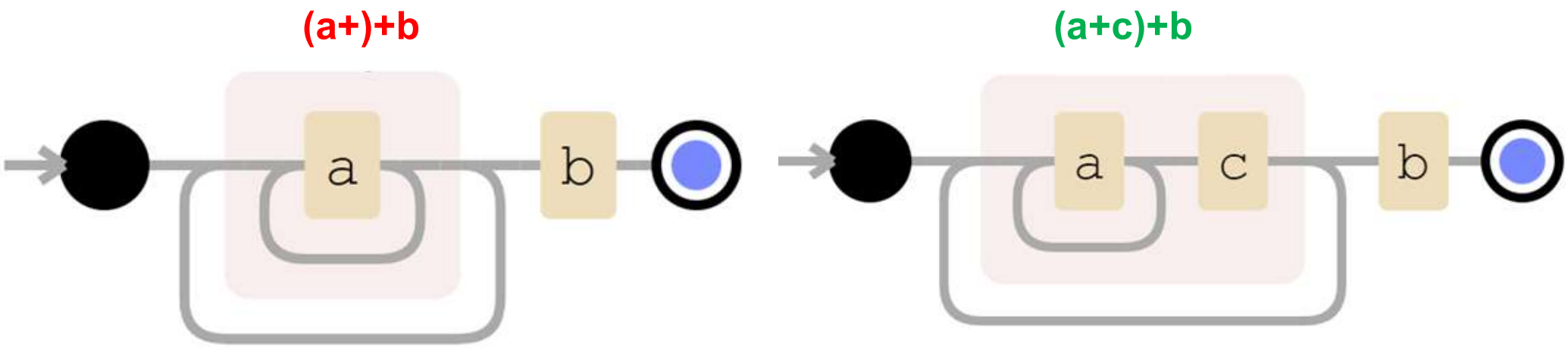
- Существует два подвыражения, при этом одно из них включено в другое и к каждому из них применяется один из следующих кванторов: '*', '+', '*?', '+?', '{...}'
- Существует такая строка, которую можно было бы сопоставить с обоими этими подвыражениями

(a+)+b



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


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

Катастрофический возврат

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

Катастрофический возврат


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

Катастрофический возврат


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

Катастрофический возврат


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

Катастрофический возврат


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

Катастрофический возврат


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

12a

Катастрофический возврат


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

12a

Катастрофический возврат


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

12a

Катастрофический возврат


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

12a

Катастрофический возврат


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

12a

Катастрофический возврат


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

12a

Катастрофический возврат


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

12a

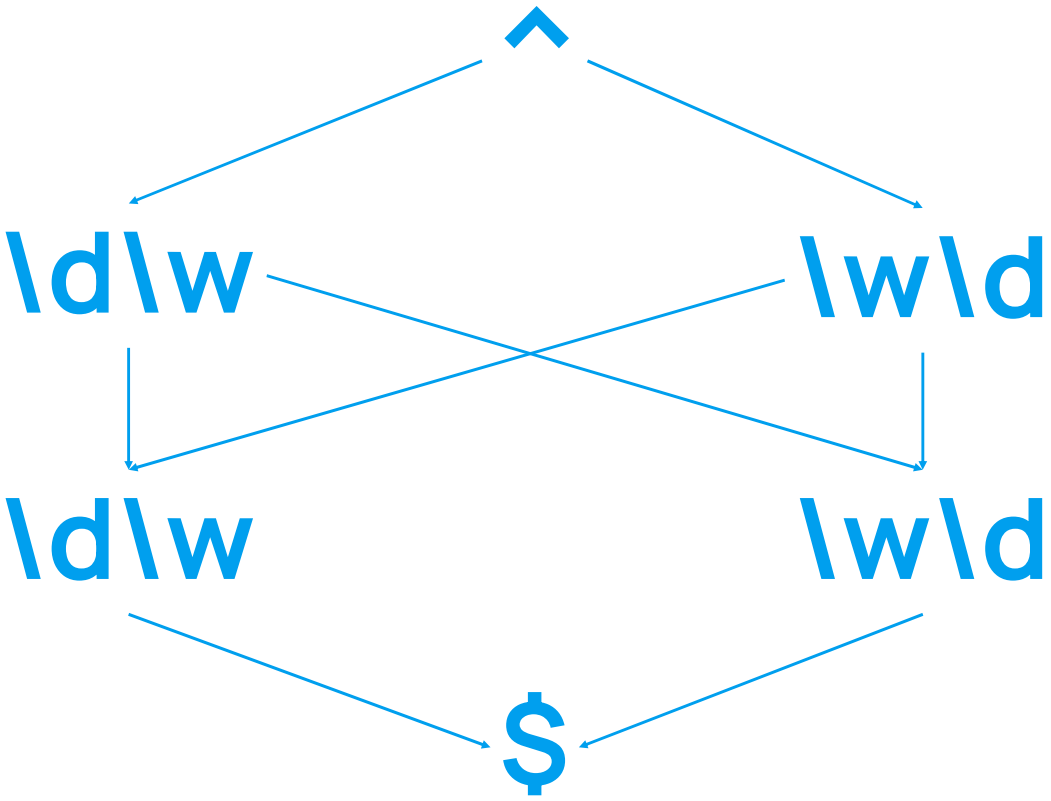
Катастрофический возврат


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

Катастрофический возврат

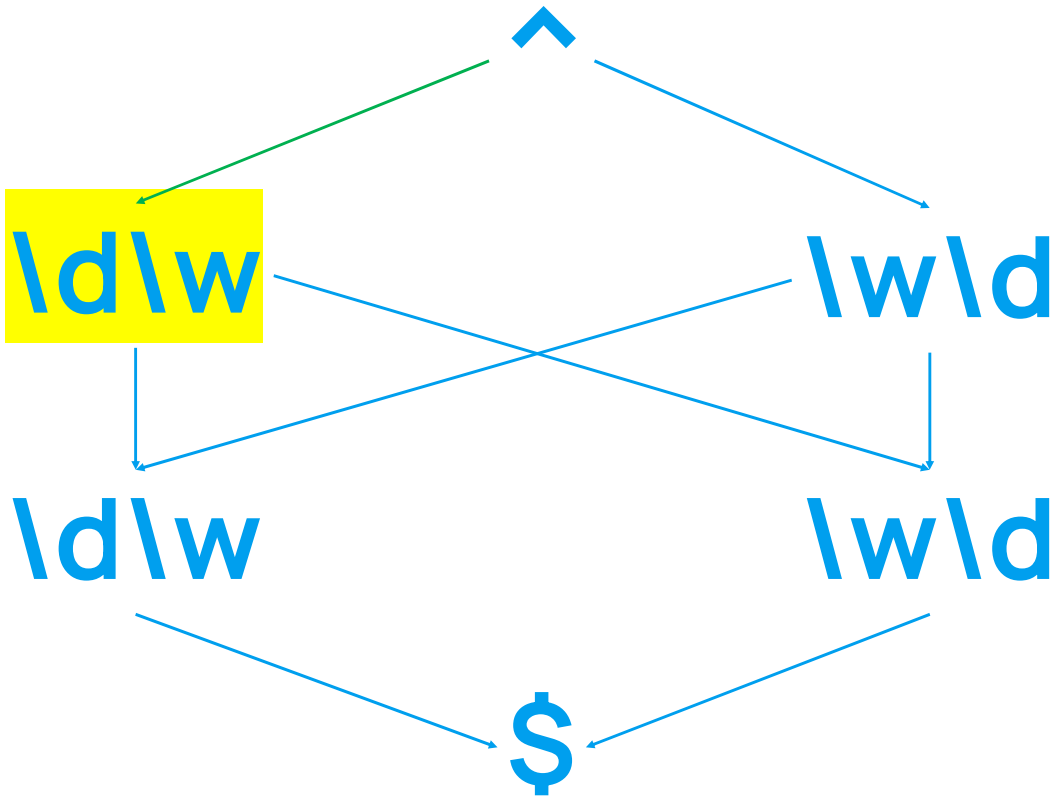

```
^(\d\w|\w\d)(\d\w|\w\d)$
```

Катастрофический возврат



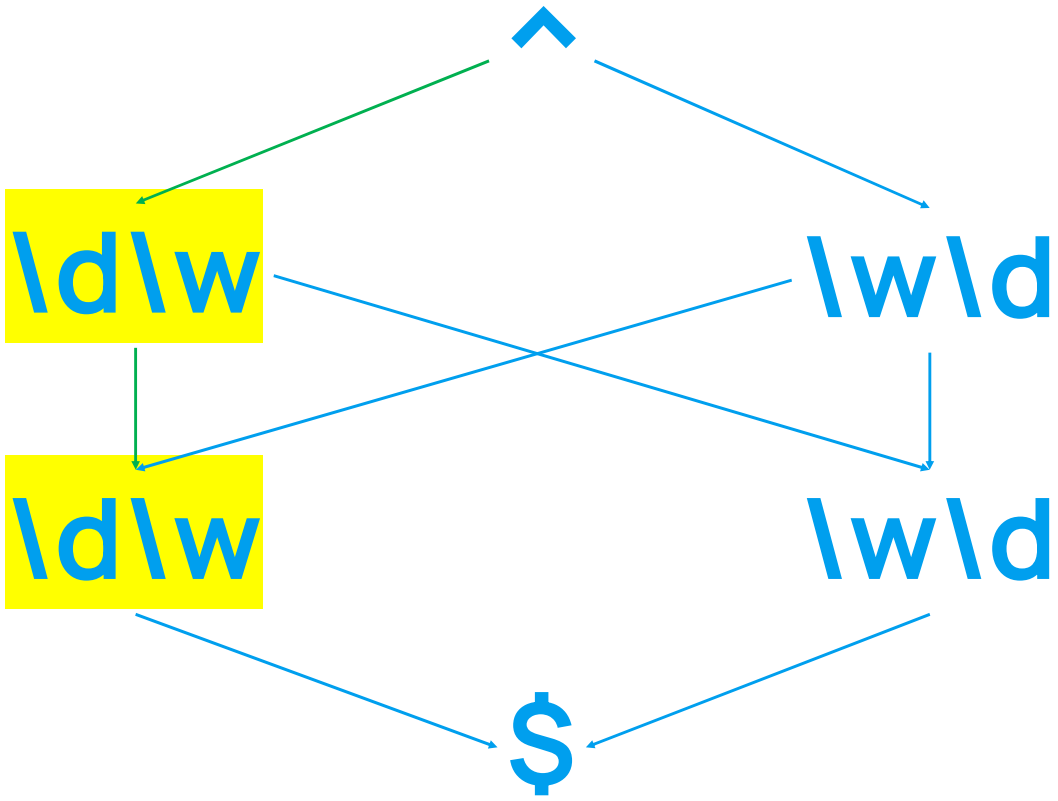
1234a

Катастрофический возврат



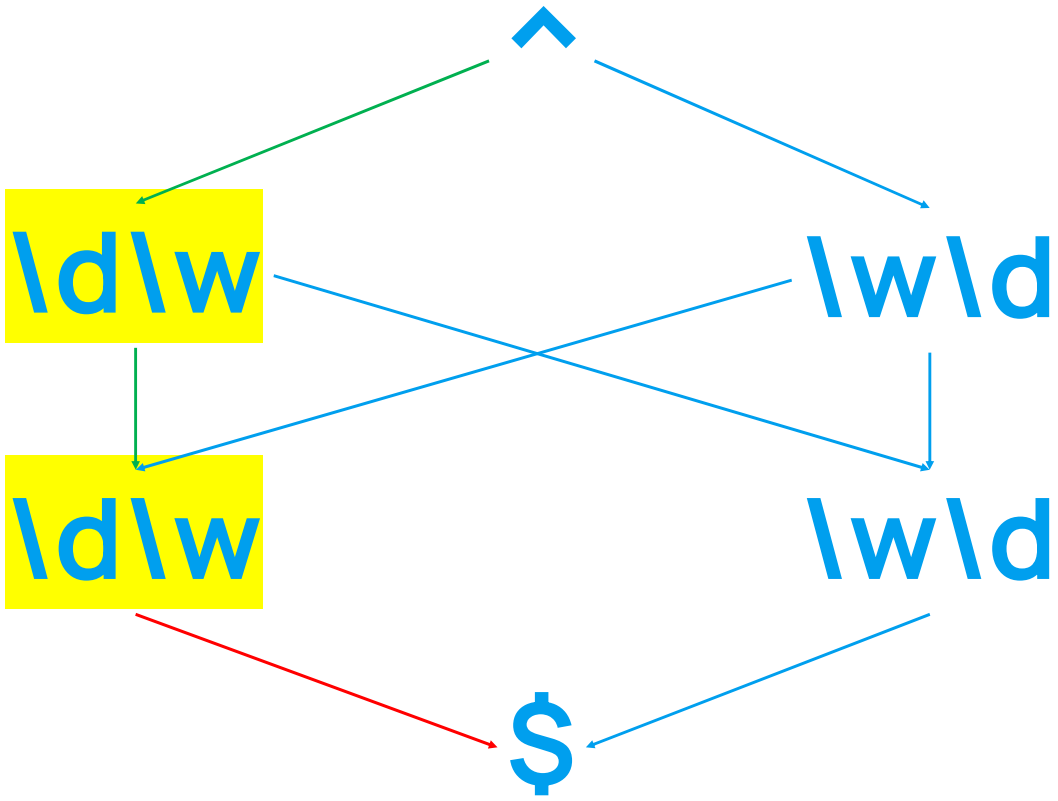
1234a

Катастрофический возврат



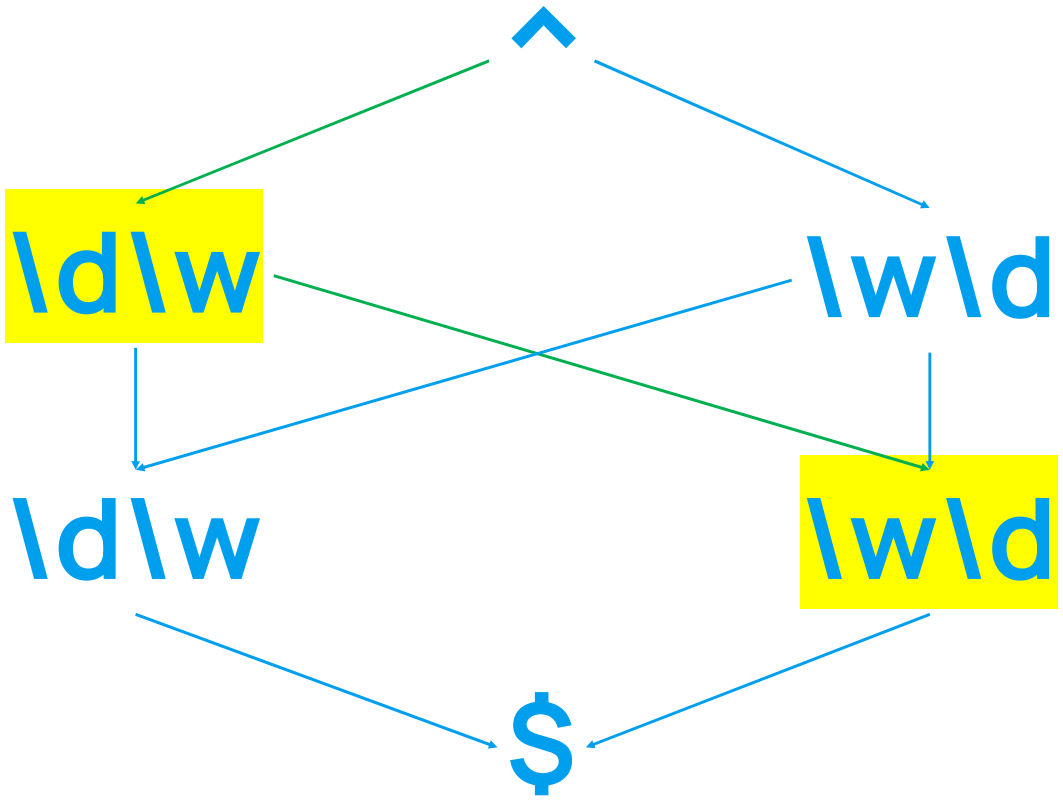
1234a

Катастрофический возврат



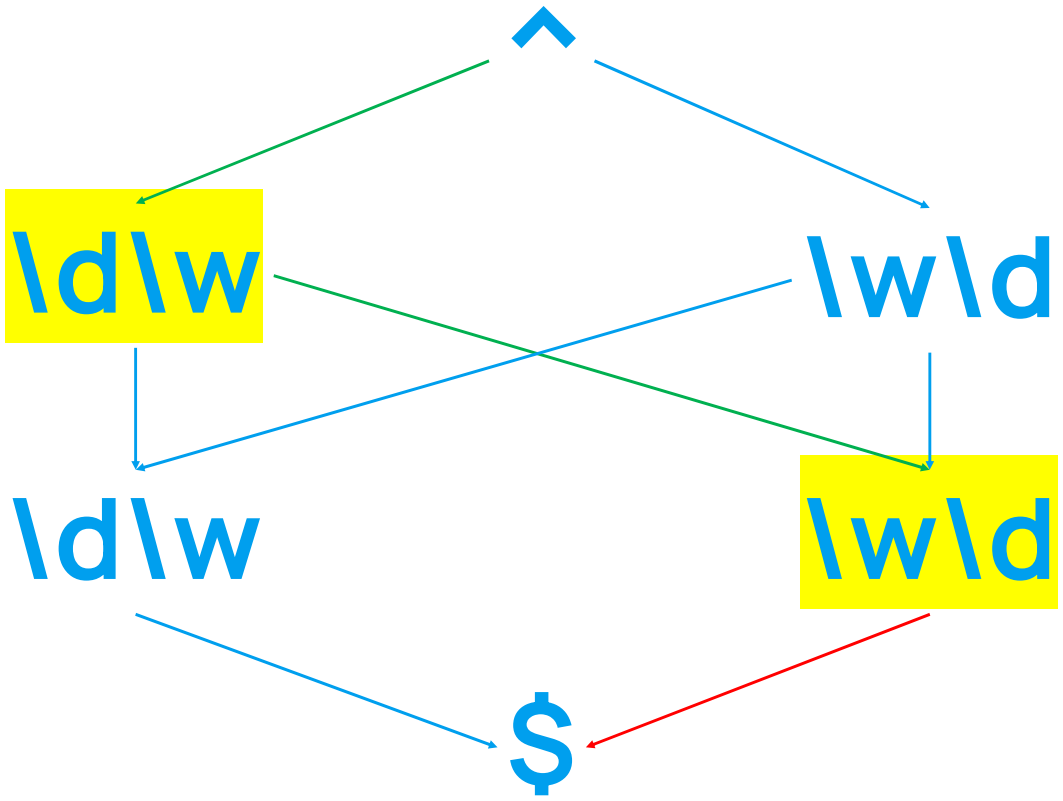
1234a

Катастрофический возврат



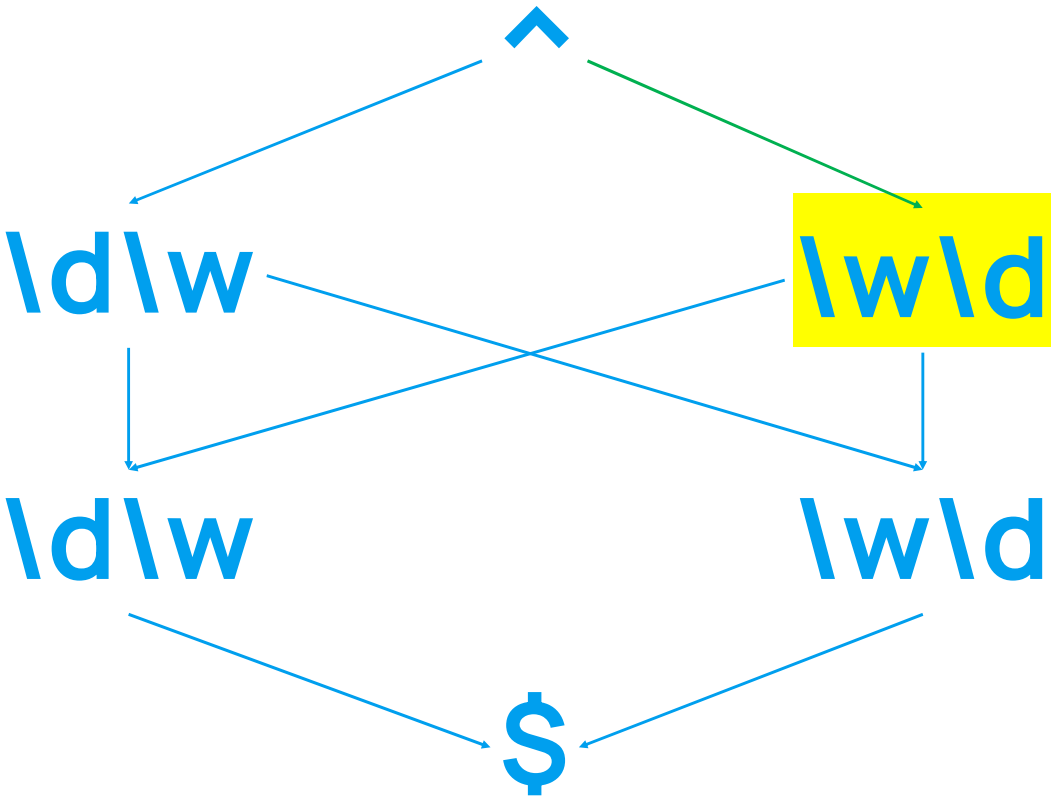
1234a

Катастрофический возврат



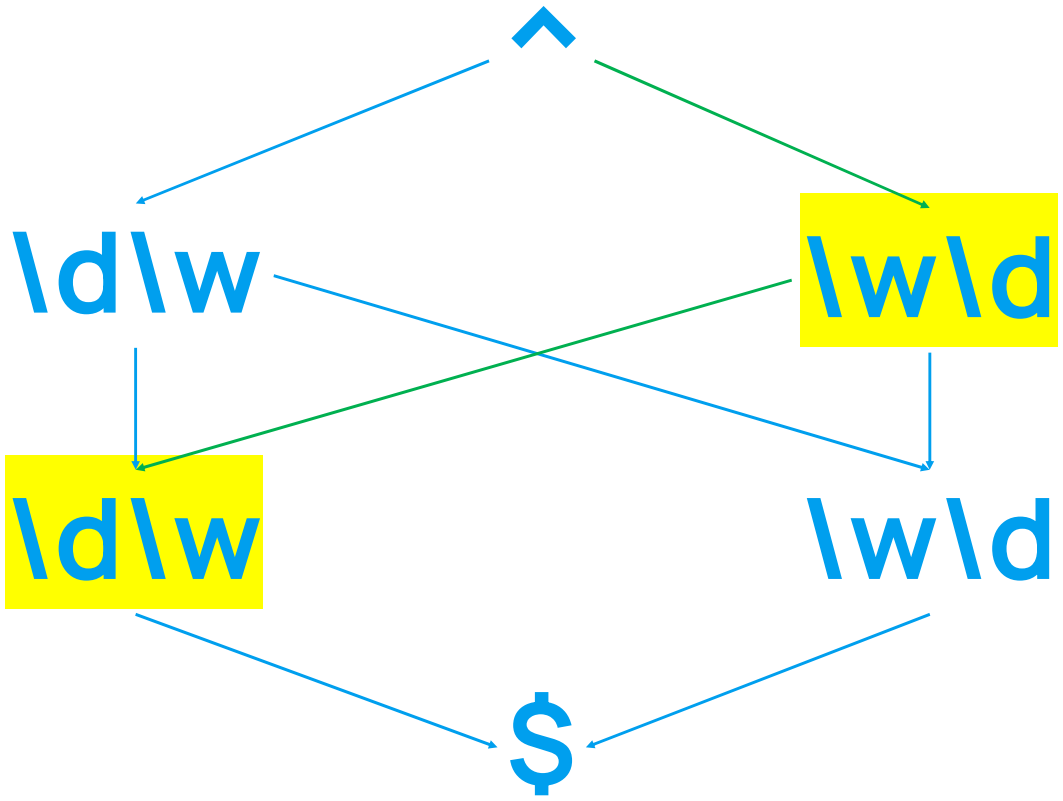
1234a

Катастрофический возврат



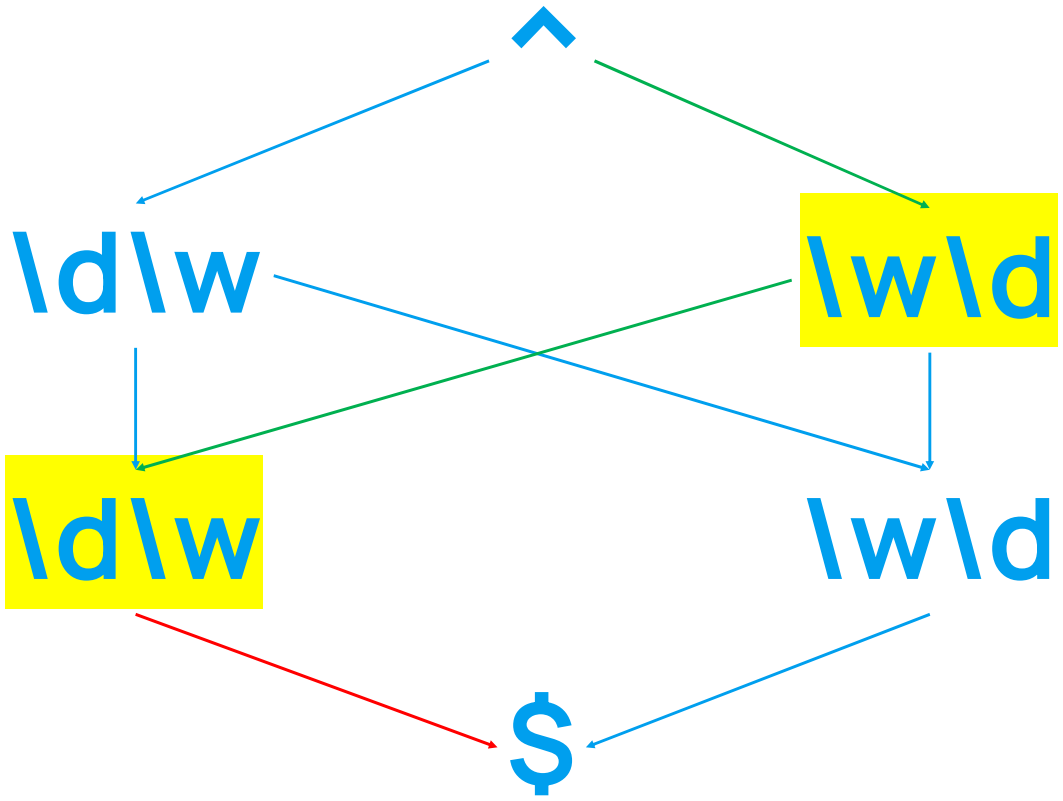
1234a

Катастрофический возврат



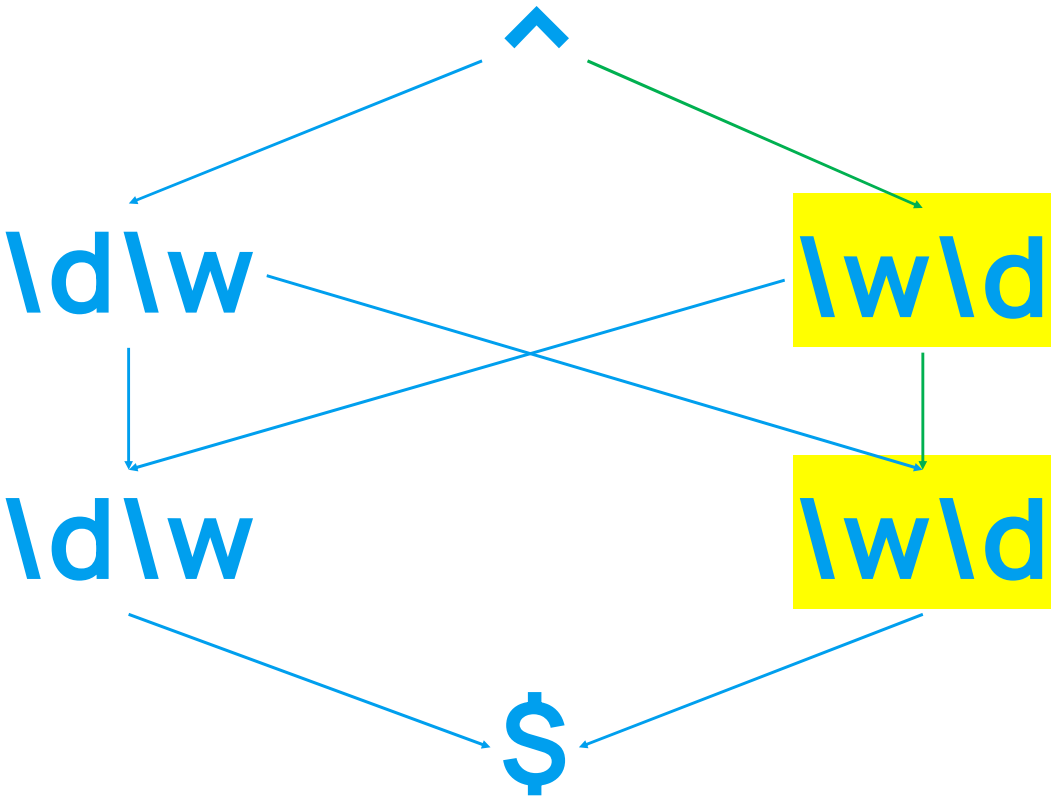
1234a

Катастрофический возврат



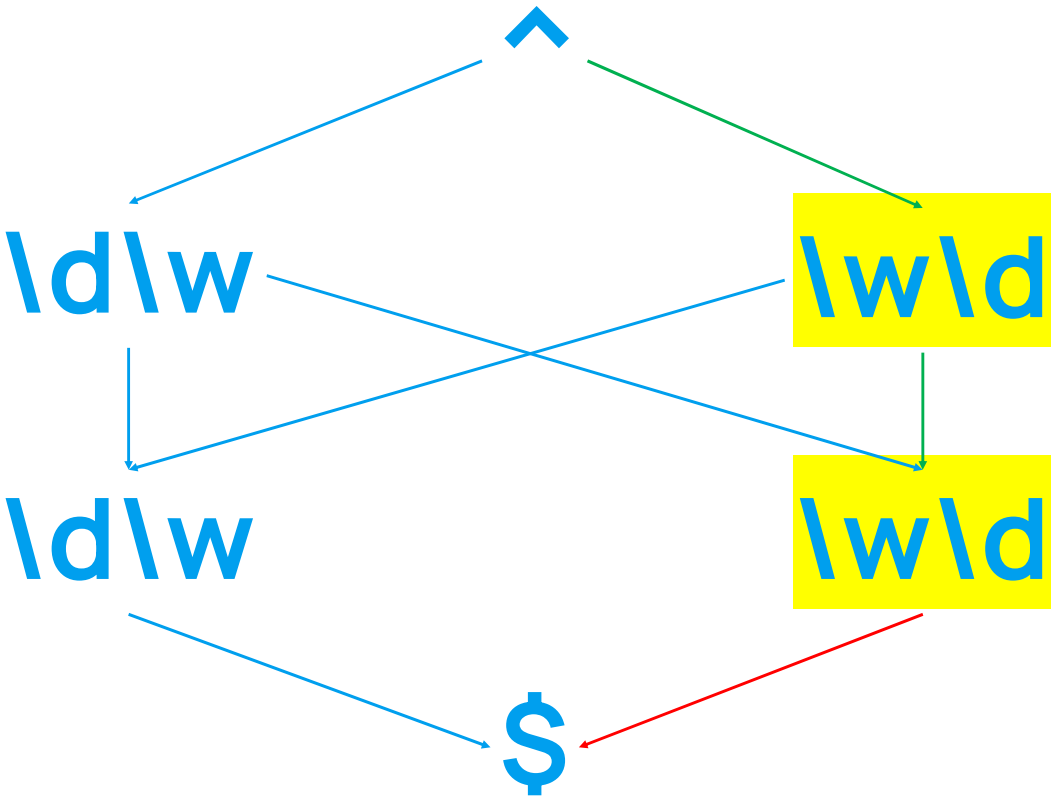
1234a

Катастрофический возврат



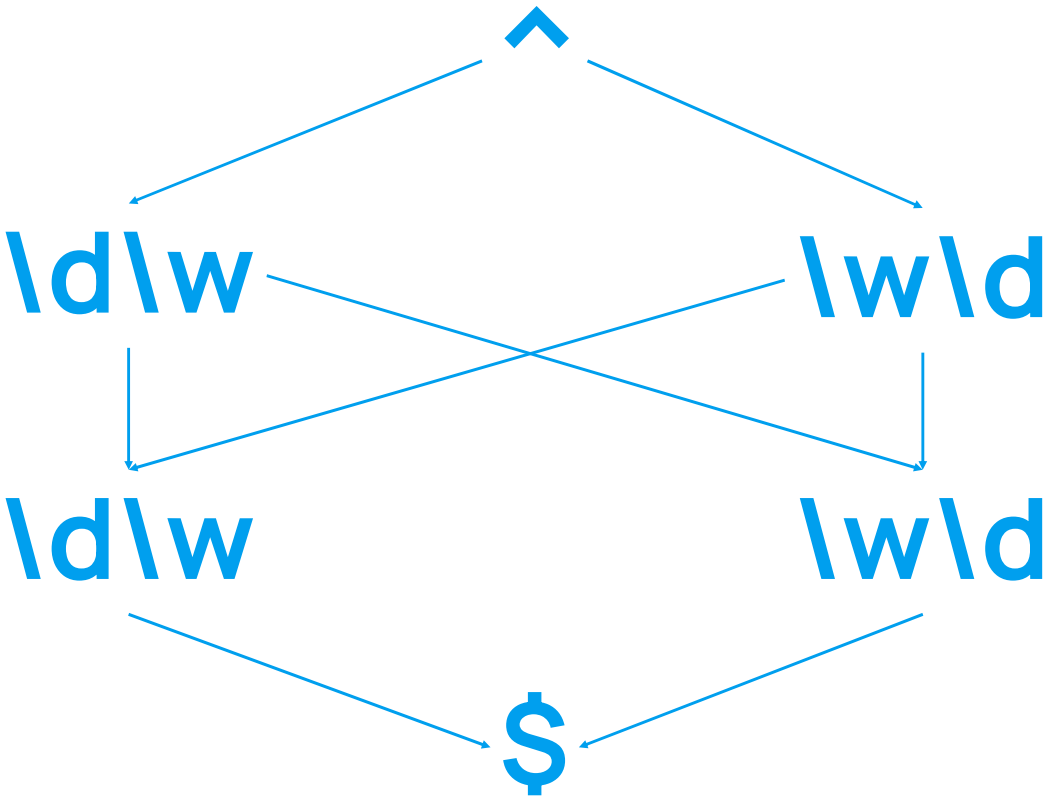
1234a

Катастрофический возврат



1234a

Катастрофический возврат



1234a

Катастрофический возврат


```
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");
}
```

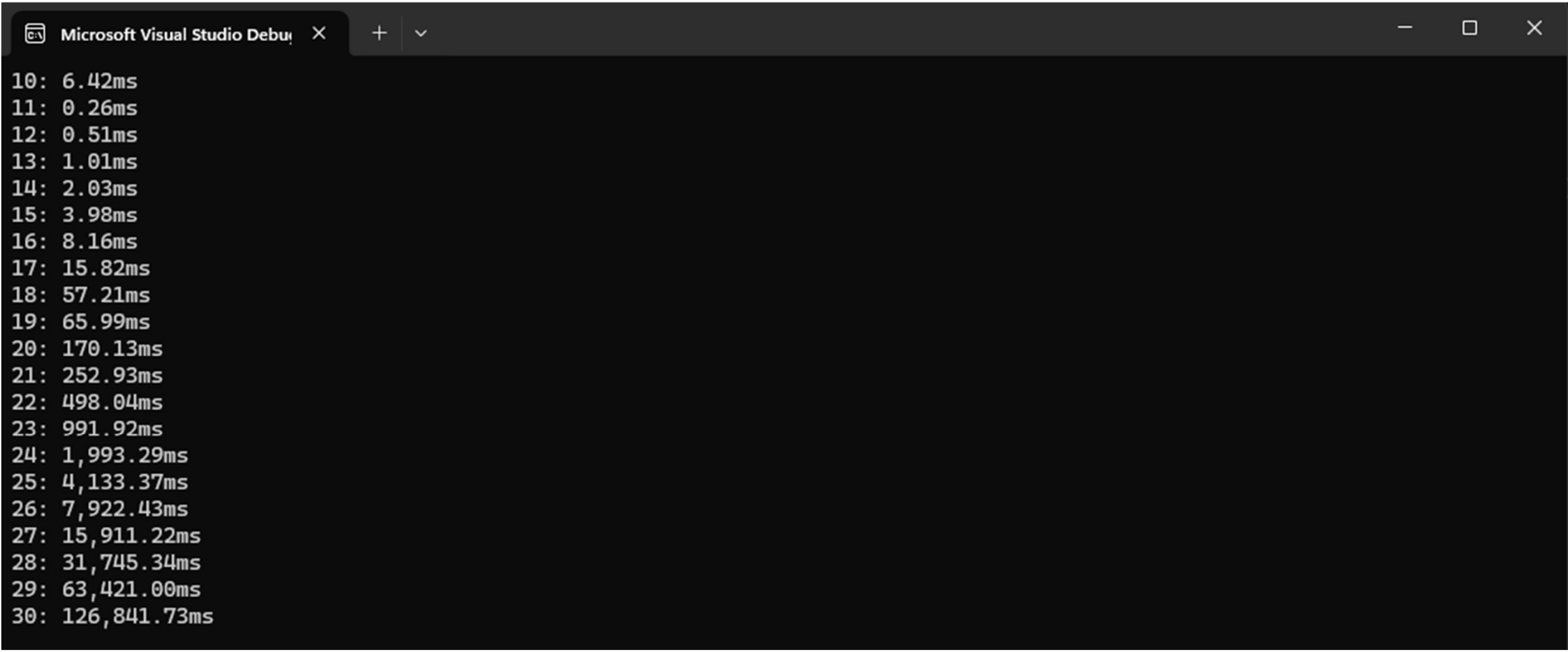
Катастрофический возврат


```
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



Катастрофический возврат


```
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


```
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");
}
```

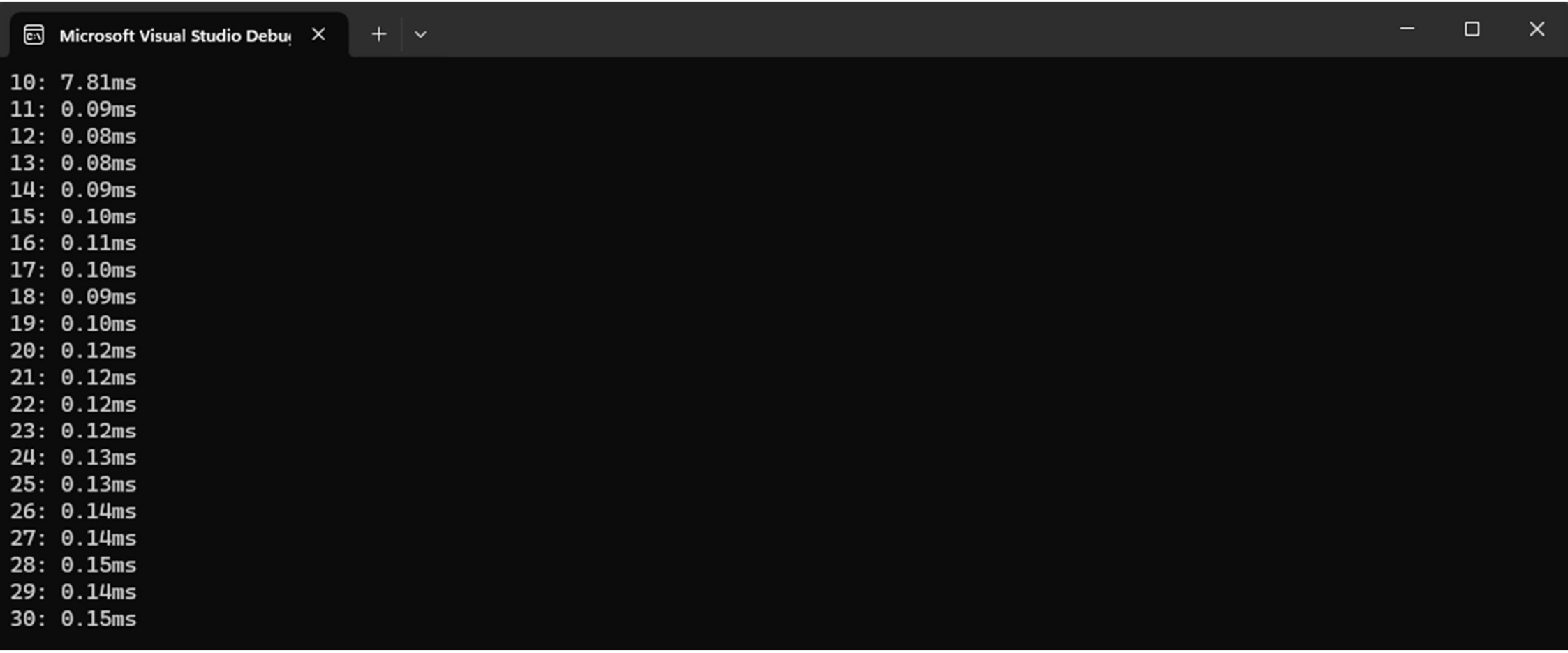
Катастрофический возврат

```
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



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

- 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


```
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


```
string input = "aaaaaaaaaaaaaaaaaaaaaa.";
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}");
```

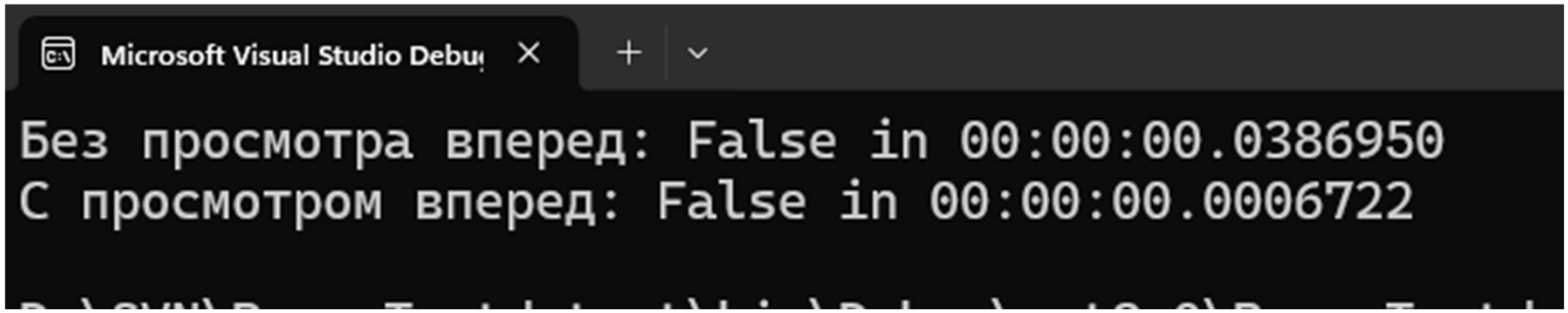
Просмотр вперед/назад


```
string input = "aaaaaaaaaaaaaaaaaaaaa.";
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



Преимущества обратного отслеживания

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

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


```
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


```
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}.");
```

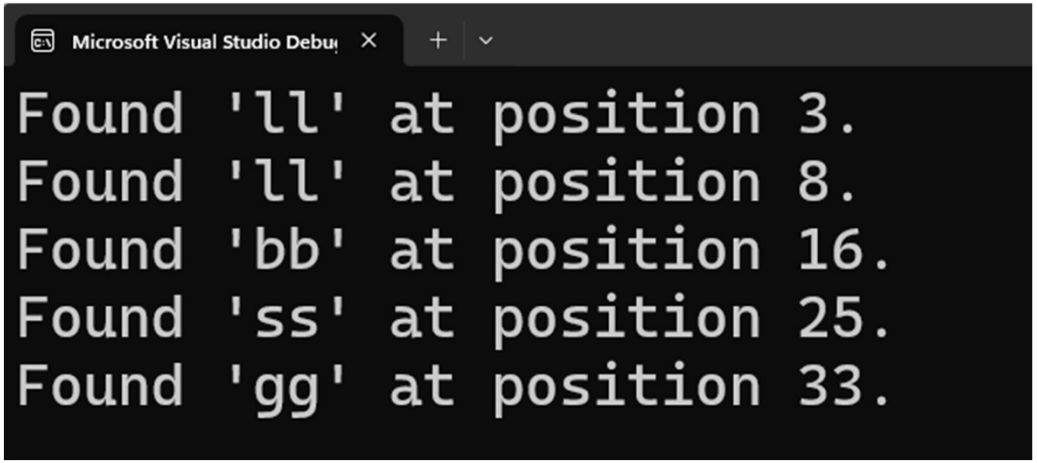
Обратные ссылки


```
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


```
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}.");
```



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 выражения работать более эффективно

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}");
```

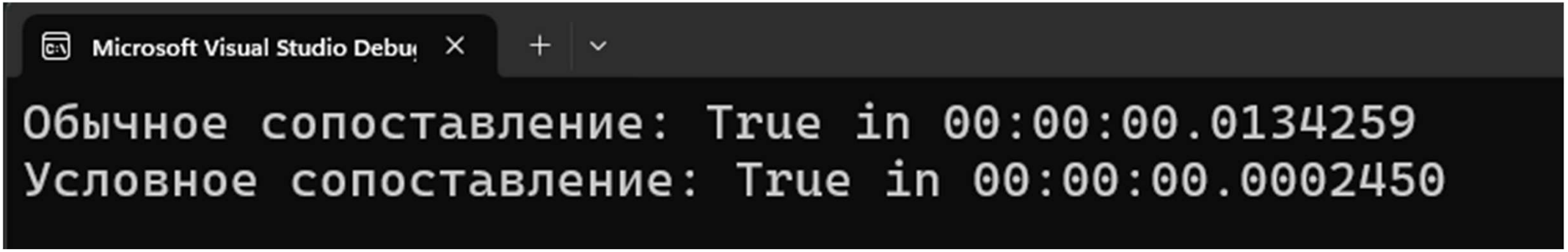
Условное сопоставление с выражением


```
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



Преимущества обратного отслеживания

Преимущества обратного отслеживания

- Позволяет быстрее работать в «хороших» случаях

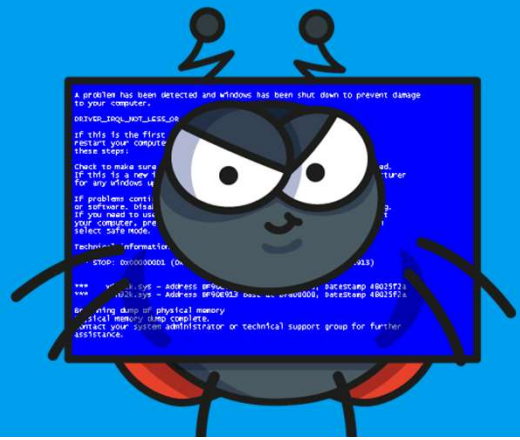
Преимущества обратного отслеживания

- 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

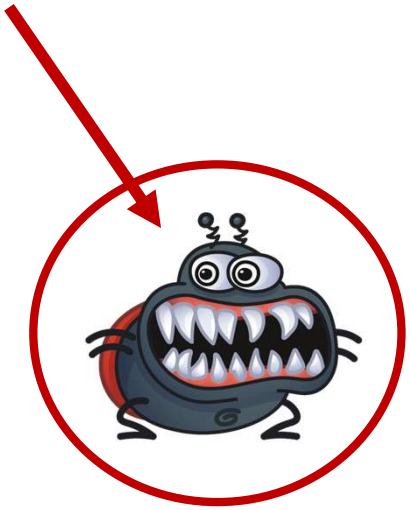


ScreenToGif


```
public static string ReplaceRegexInName(string name)
{
  const string dateTimeFileNameRegEx = @"[?]([ymdhsfzgkt]+[-_ ]*)+[?]";
  if (!Regex.IsMatch(name,
                     dateTimeFileNameRegEx,
                     RegexOptions.IgnoreCase))
    return name;
  var match = Regex.Match(name,
                          dateTimeFileNameRegEx,
                          RegexOptions.IgnoreCase);
  ....
}
```

ScreenToGif


```
public static string ReplaceRegexInName(string name)
{
  const string dateTimeFileNameRegEx = @"[?]([ymdhsfzgkt]+[-_ ]*)+[?]";
  if (!Regex.IsMatch(name,
                     dateTimeFileNameRegEx,
                     RegexOptions.IgnoreCase))
    return name;
  var match = Regex.Match(name,
                          dateTimeFileNameRegEx,
                          RegexOptions.IgnoreCase);
  ....
}
```



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

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

```
[?]([ymdhsfzgkt]+)+[?]
```

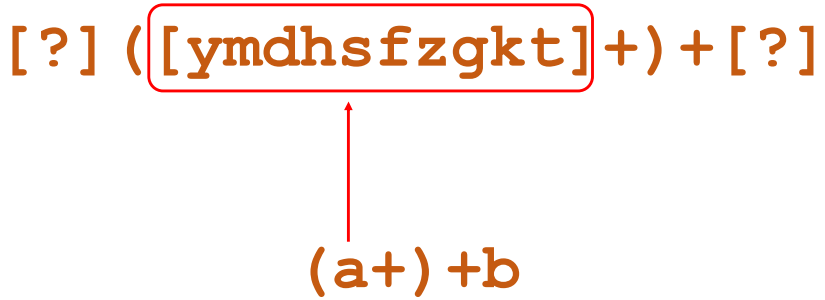
```
[?]([ymdhsfzgkt]+[-_ ]*)+[?]
```

```
[?]([ymdhsfzgkt]+)+[?]
```



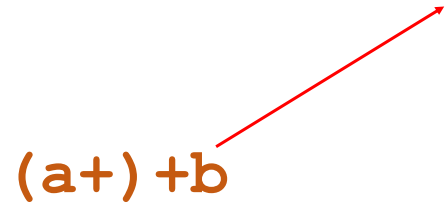
```
(a+)+b
```

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



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

```
[?]([ymdhsfzgkt]+)+[?]
```



ScreenToGif


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

```
[?]([ymdhsfzgkt]+)+[?]
```

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

```
[?]([ymdhsfzgkt]+)+[?]
```


0 вхождений [-_]

ScreenToGif


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

```
[?]([ymdhsfzgkt]+)+[?]
```

0 вхождений [-_]

?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgkt

?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgkt

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

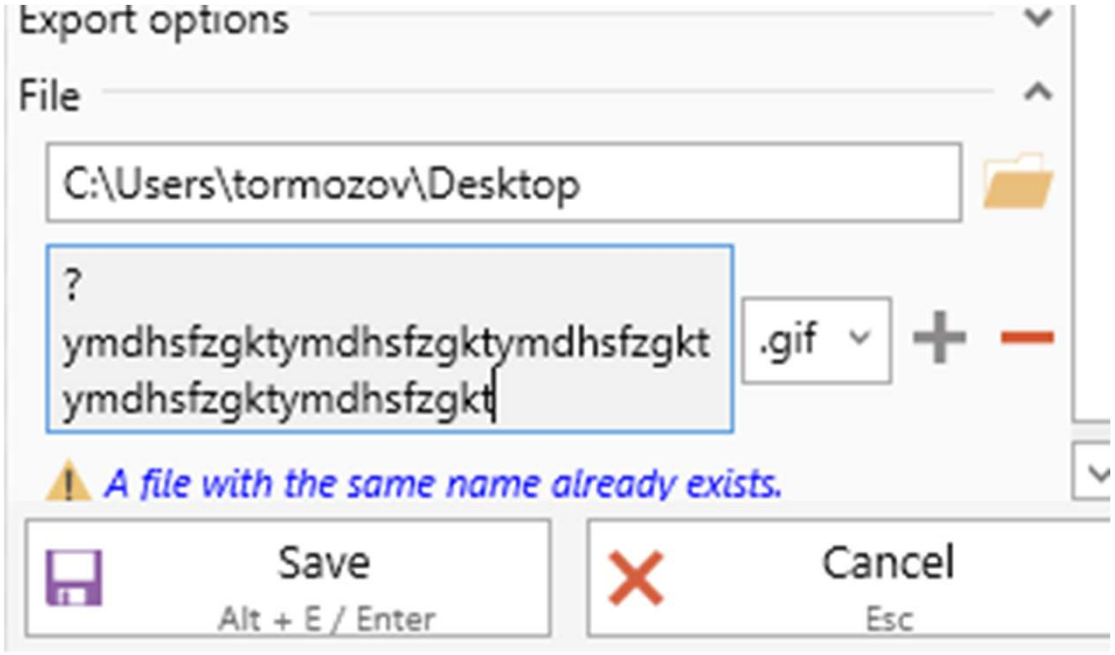
ScreenToGif


```
?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgkt
```

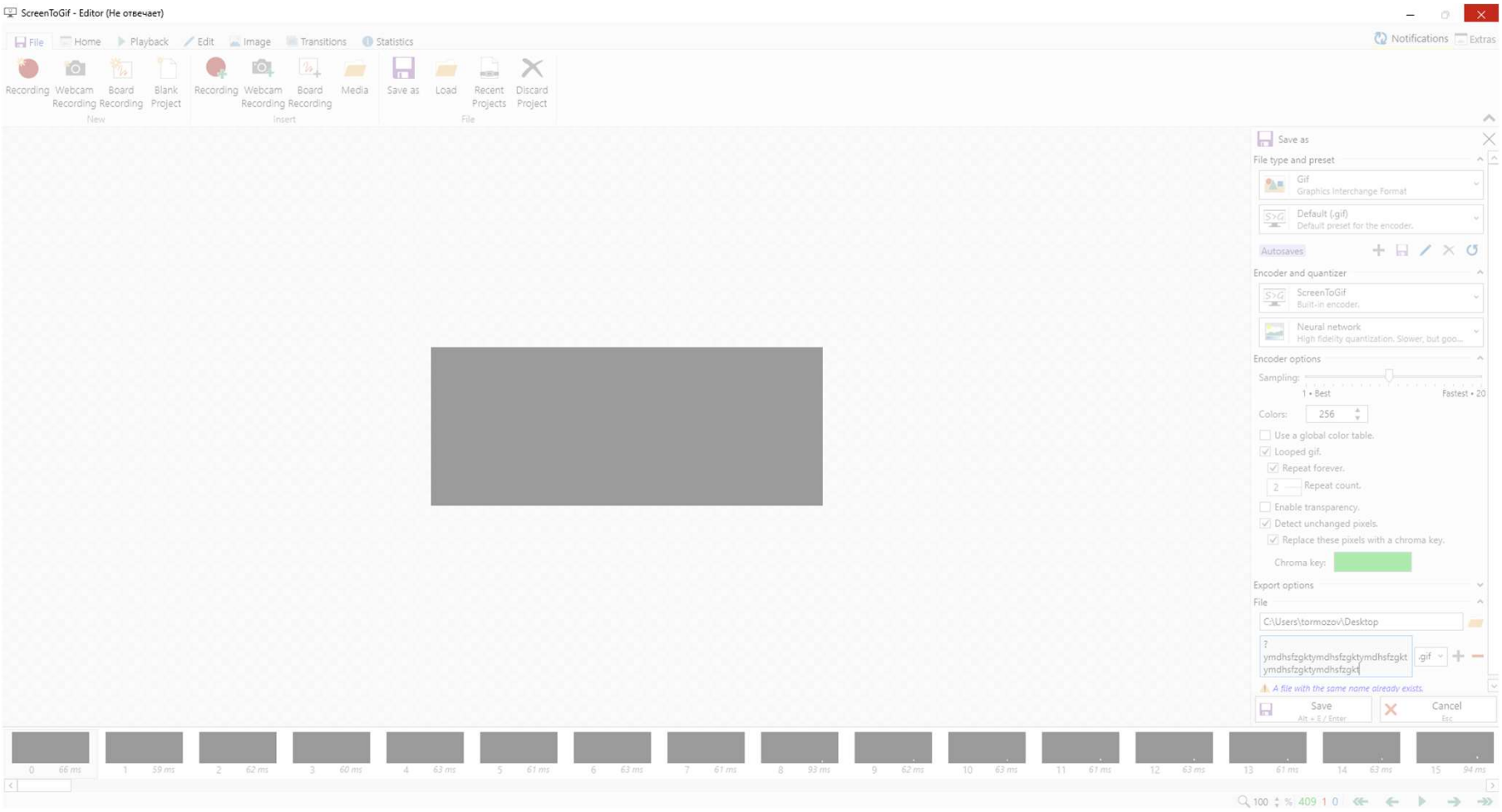
```
[?]([ymdhsfzgkt]+[-_ ]*)+[?]
```

ReDoS

ScreenToGif



ScreenToGif



RestSharp

- CVE-2021-27293

RestSharp

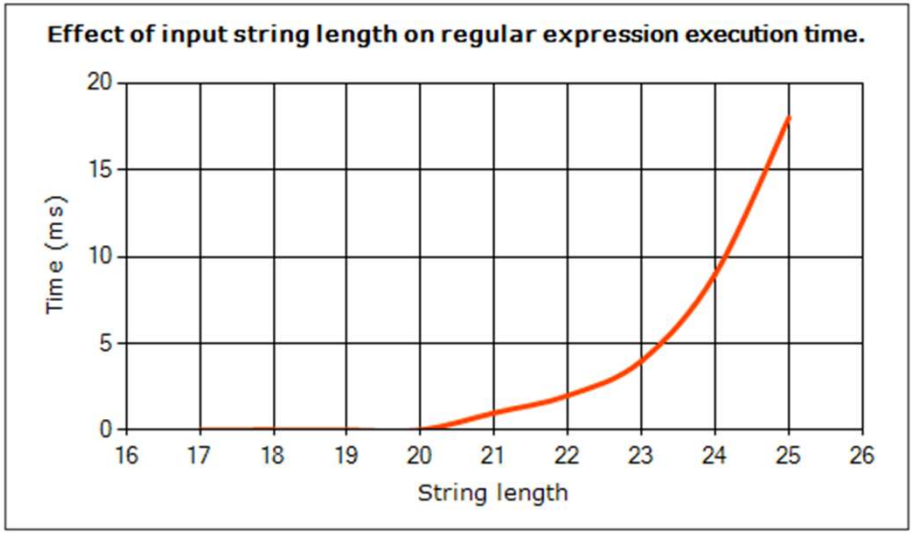
- CVE-2021-27293
- Regex: **"newDate\((-?\d+)*\)"**
- Ищет строку формата:
  newDate(10-01-2007)

RestSharp

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

Input strings:

```
newDate(111111111
newDate(1111111112
newDate(11111111133
newDate(111111111444
newDate(1111111115555
newDate(11111111166666
newDate(111111111777777
newDate(1111111118888888
newDate(11111111199999999
```



System.ComponentModel.DataAnnotations

- CVE-2015-2526

System.ComponentModel.DataAnnotations

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

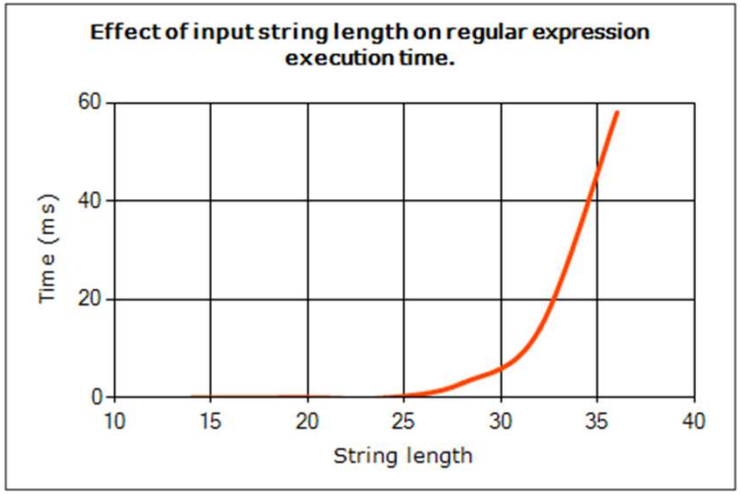
System.ComponentModel.DataAnnotations

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

email validation

Input strings:

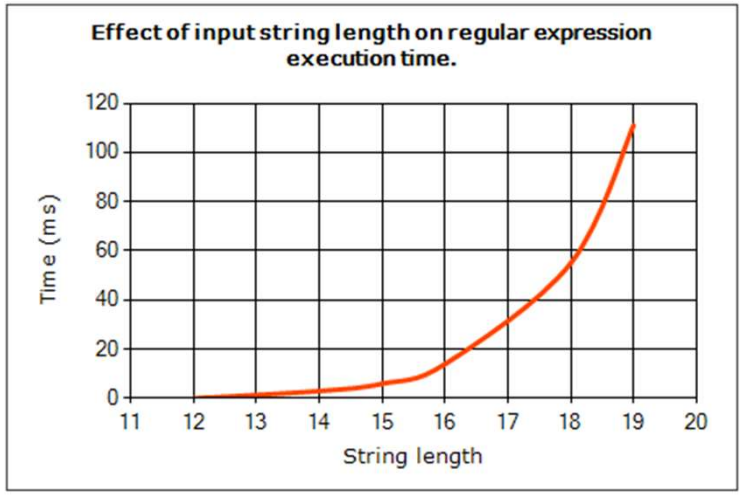
```
t@t.t.t.t.c%20
t@t.t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.t.t.t.t.t.t.c%20
t@t.t.t.t.t.t.t.t.t.t.t.t.t.t.t.c%20
```



phone validation

Input strings:

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



ReDoS вне C# кода

ReDoS вне C# кода

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

ReDoS вне C# кода

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

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

ReDoS вне C# кода

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

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

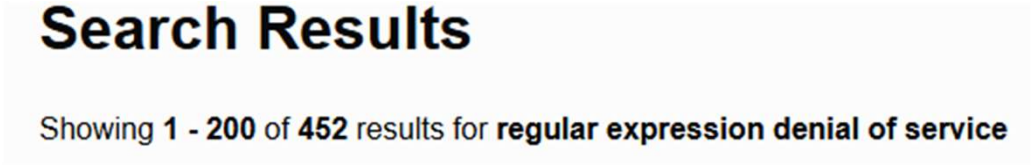
ReDoS вне C# кода

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

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



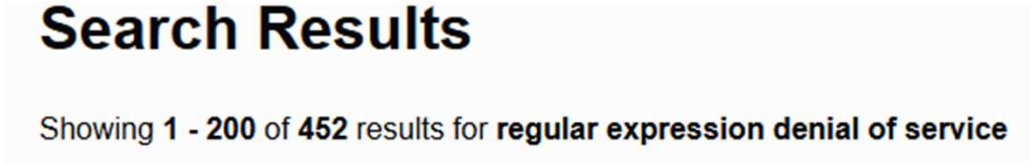
ReDoS вне C# кода

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

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



Из них за:

- **2023: 39 уязвимостей**
- **2024: 44 уязвимостей**

Как бороться с ReDoS

PVS-Studio

Как бороться с ReDoS

- Использовать другое регулярное выражение

Как бороться с ReDoS

- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком

Как бороться с ReDoS

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

Как бороться с ReDoS

- 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


```
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


```
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: aaaaaaaaaaaaaaa!

ReDoS via injection


```
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: aaaaaaaaaaaaaaa!

Результат – ReDoS!

Как бороться с ReDoS

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

Использование timeout

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

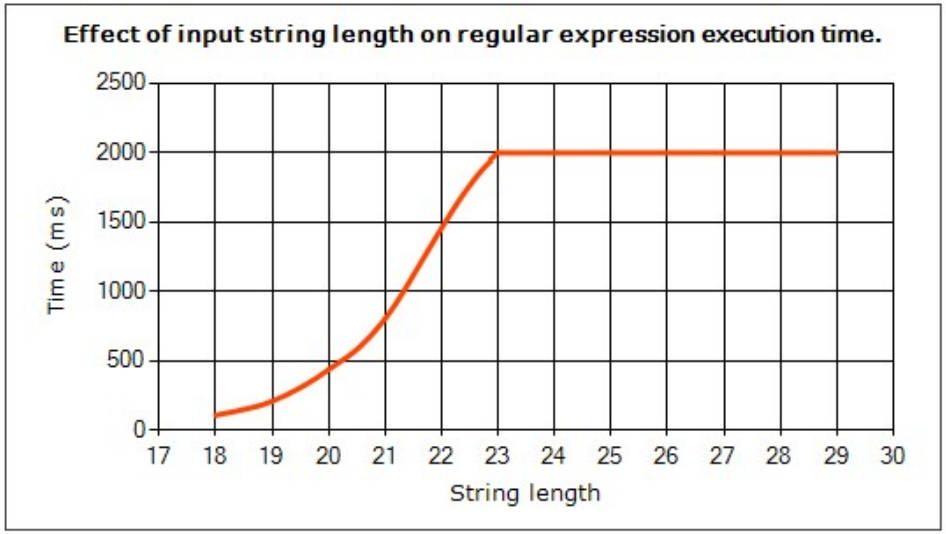
Третий аргумент конструктора регулярных выражений – **matchTimeout**

В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

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

Input strings:

aaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaa



Как бороться с ReDoS

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

Атомарные группы

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

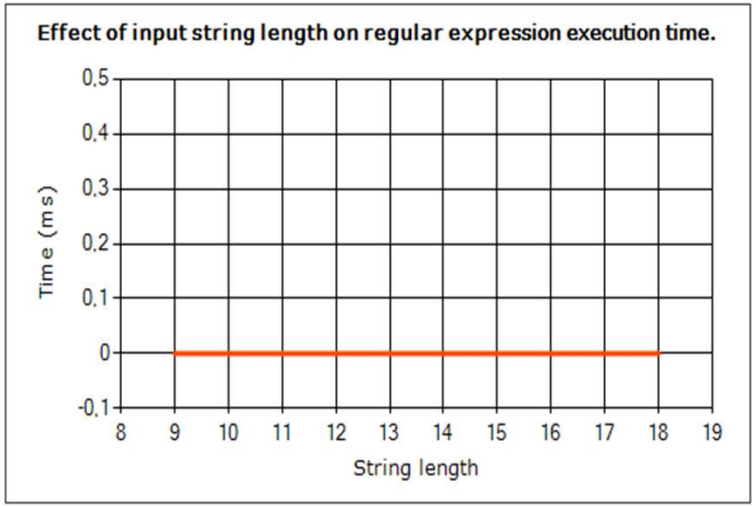
Атомарные группы

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

(?>a*)*b

Input strings:

aaaaaaaaa
aaaaaaaaaa
aaaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaa



Как бороться с ReDoS

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

DFA Regex engine

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

DFA Regex engine

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

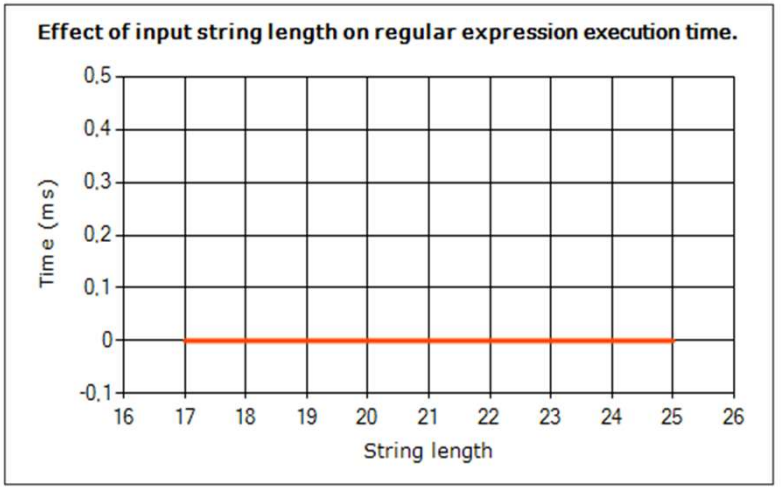
С версии .NET 7 вы можете указать флаг **RegexOptions.NonBacktracking**

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

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

Input strings:

```
newDate(111111111
newDate(1111111112
newDate(11111111133
newDate(111111111444
newDate(1111111115555
newDate(11111111166666
newDate(111111111777777
newDate(1111111118888888
newDate(11111111199999999
```



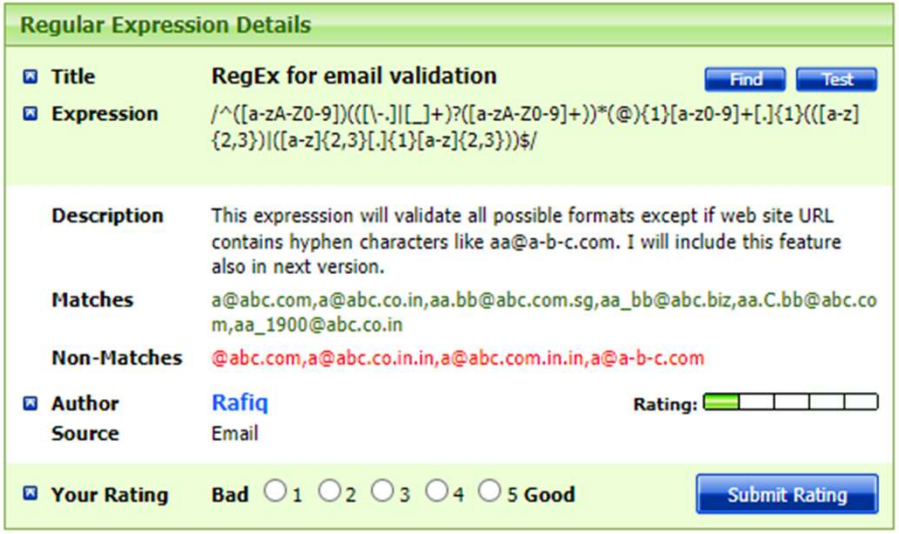
Профилактика ReDoS уязимостей

PVS-Studio

Профилактика ReDoS уязвимостей

- Проверять чужие регулярные выражения

Доверяй, но проверяй



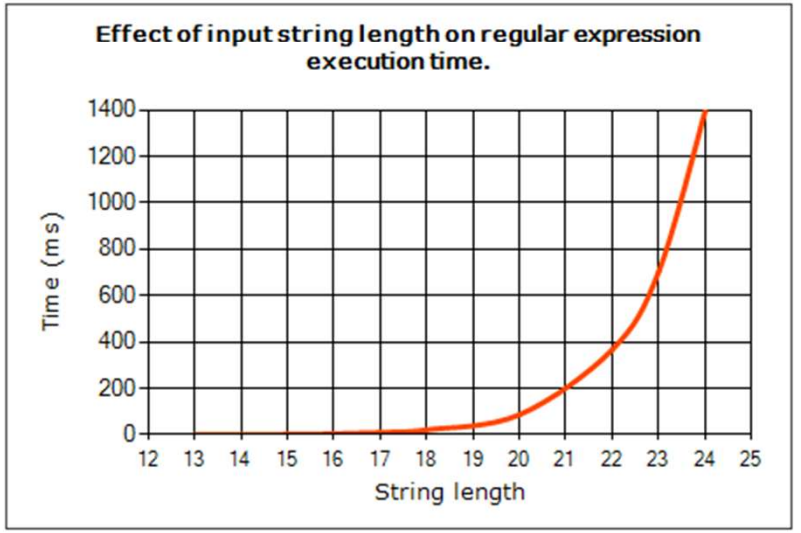
Доверяй, но проверяй

Regular Expression Details

Title	RegEx for email validation (Find) (Test)
Expression	/^([a-zA-Z0-9])(([\-.]|[_]+)?([a-zA-Z0-9]+))*(@){1}[a-z0-9]+[.]{1}(([a-z]{2,3})|([a-z]{2,3}[.]{1}[a-z]{2,3}))$/
Description	This expresssion 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.com,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 Rating:
Source	Email
Your Rating	Bad ○1 ○2 ○3 ○4 ○5 Good (Submit Rating)

Input strings:

```
aaaaaaaaaaaa!
aaaaaaaaaaaaaa!
aaaaaaaaaaaaaaa!
aaaaaaaaaaaaaaaa!
aaaaaaaaaaaaaaaaa!
aaaaaaaaaaaaaaaaaa!
aaaaaaaaaaaaaaaaaaa!
aaaaaaaaaaaaaaaaaaaa!
aaaaaaaaaaaaaaaaaaaaa!
```



Профилактика ReDoS уязвимостей

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

Профилактика ReDoS уязвимостей

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

Профилактика ReDoS уязвимостей

- 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