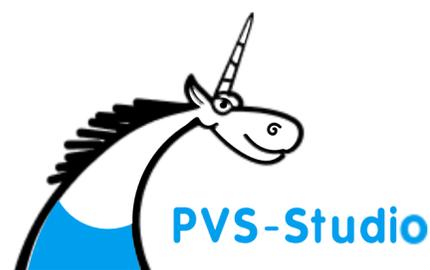


# Подводные камни регулярных выражений

Уязвимости, катастрофический возврат и ReDoS-атаки



**Георгий Тормозов**

C# разработчик



# Георгий Тормозов

C# разработчик

- Занимаюсь разработкой статического анализатора
- Пишу статьи
- Люблю анекдоты



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

# Как работают регулярные выражения

Лупа

## 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...

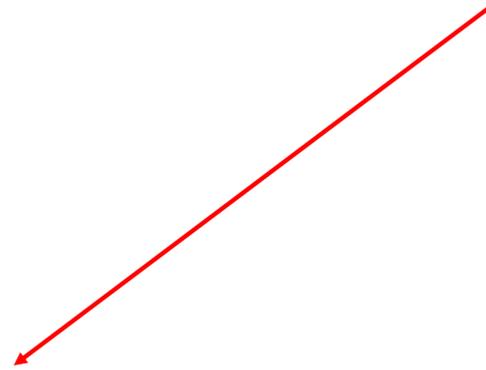
## Лупа

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

## 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...

## 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...

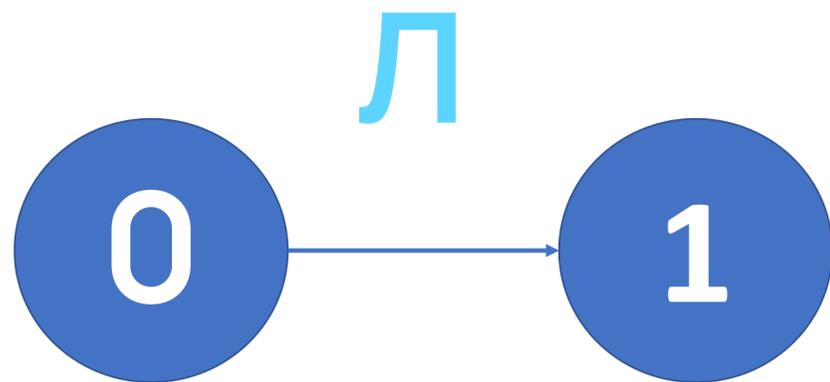
**`^[A-Z0-9._%+@-]+@[A-Z0-9-]+.[A-Z]{2,4}$`**

## Конечные автоматы

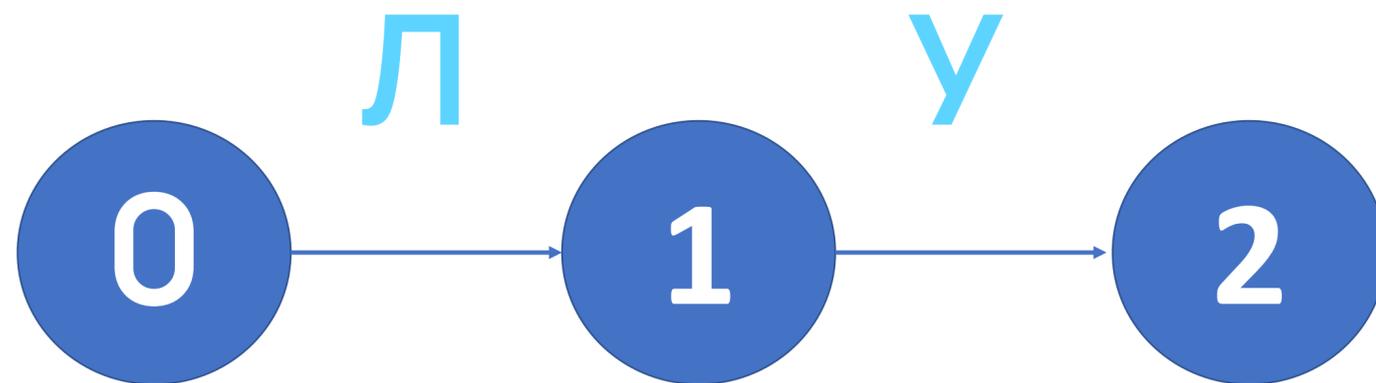
## Конечные автоматы



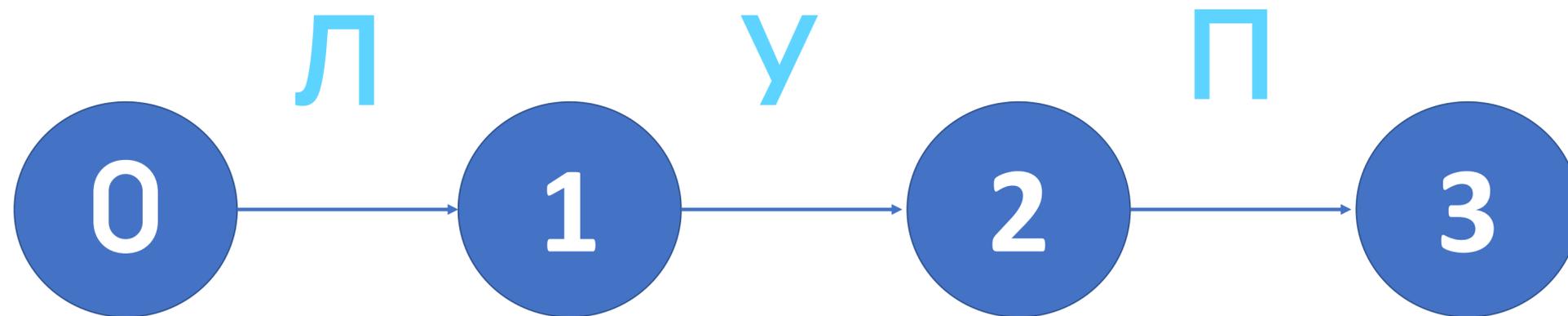
## Конечные автоматы



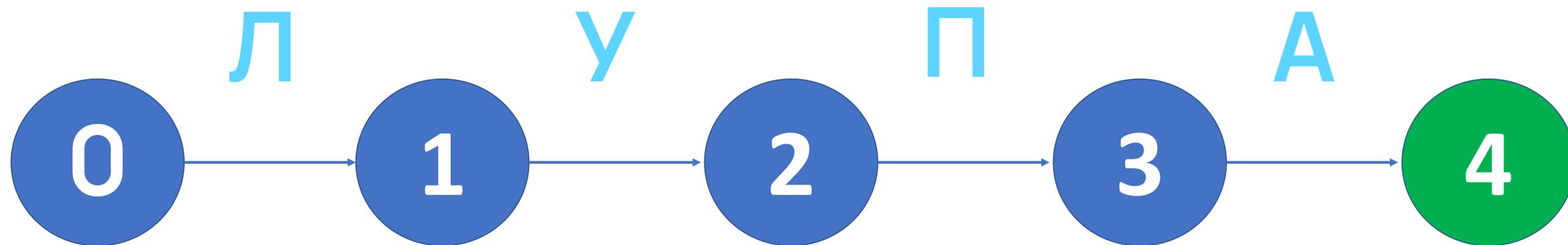
## Конечные автоматы



## Конечные автоматы

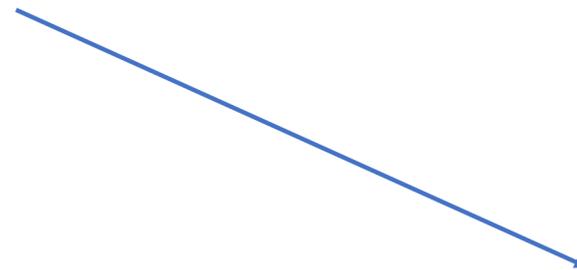
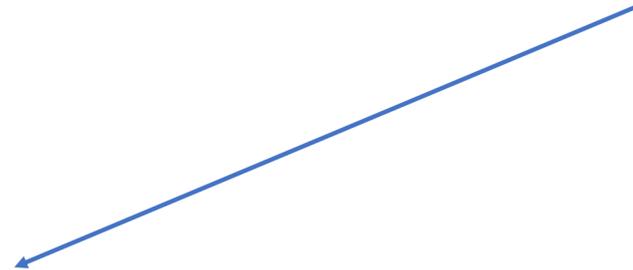


## Конечные автоматы



## Конечные автоматы

**Конечные автоматы**



**Детерминированные**

**Недетерминированные**

## Конечные автоматы

Детерминированные

Недетерминированные

- Быстрее работают в «худших» случаях

## Конечные автоматы

```
graph TD; A[Конечные автоматы] --> B[Детерминированные]; A --> C[Недетерминированные];
```

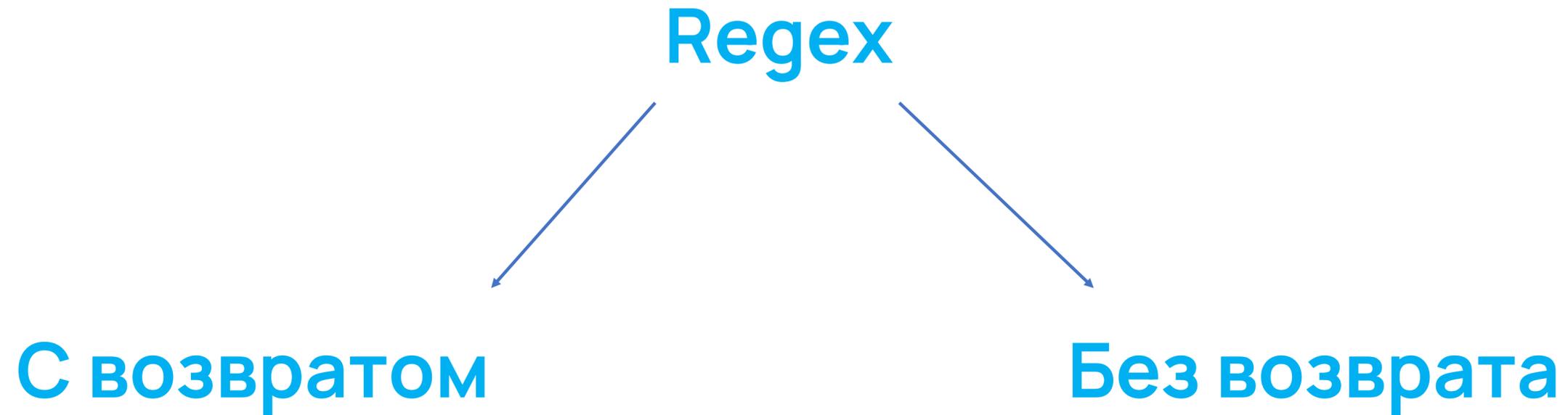
### Детерминированные

- Быстрее работают в «худших» случаях

### Недетерминированные

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

## Regex



Regex

```
graph TD; A[Regex] --> B[С возвратом]; A --> C[Без возврата];
```

С возвратом

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

Пример: **a+b**

Без возврата

## Regex

```
graph TD;
  Regex[Regex] --> WithReturn[С возвратом];
  Regex --> WithoutReturn[Без возврата];
```

### 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

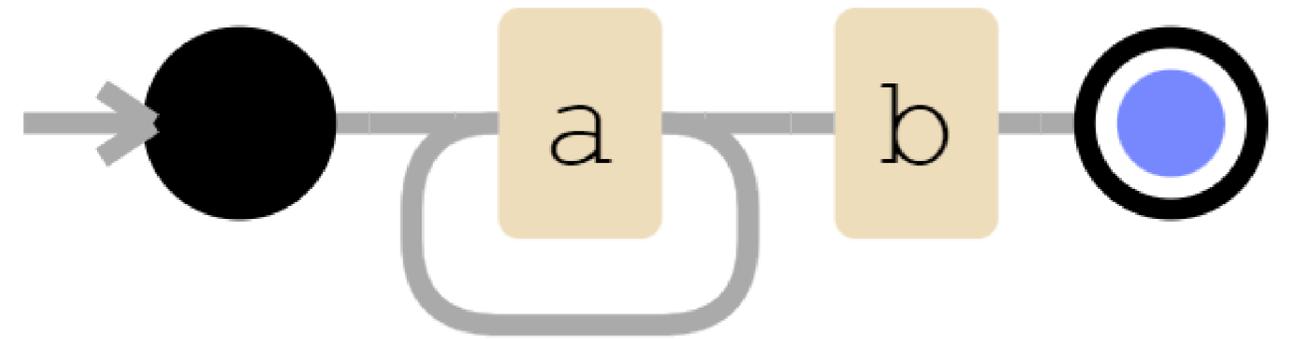
Регулярное выражение: **a+b**

Последовательность для разбора: aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

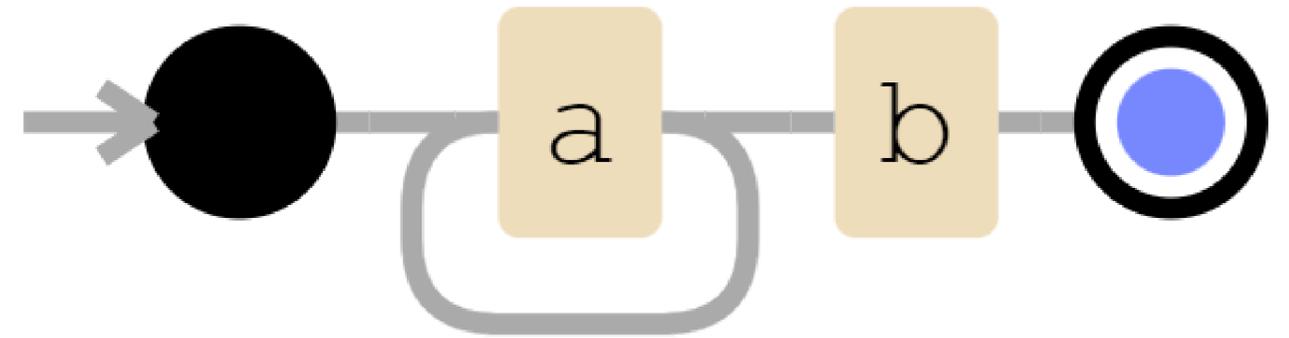
Последовательность для разбора: aaa



# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



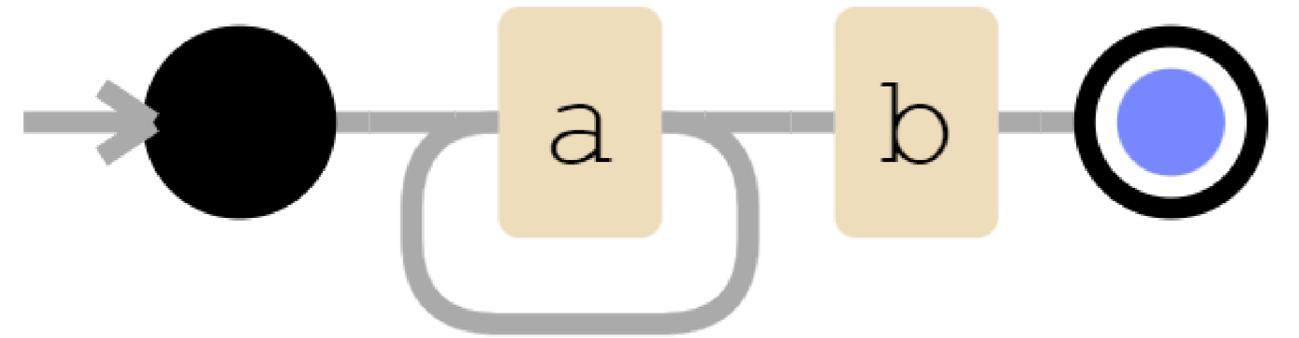
```
Regex regex = new Regex("a+b");
```

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

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

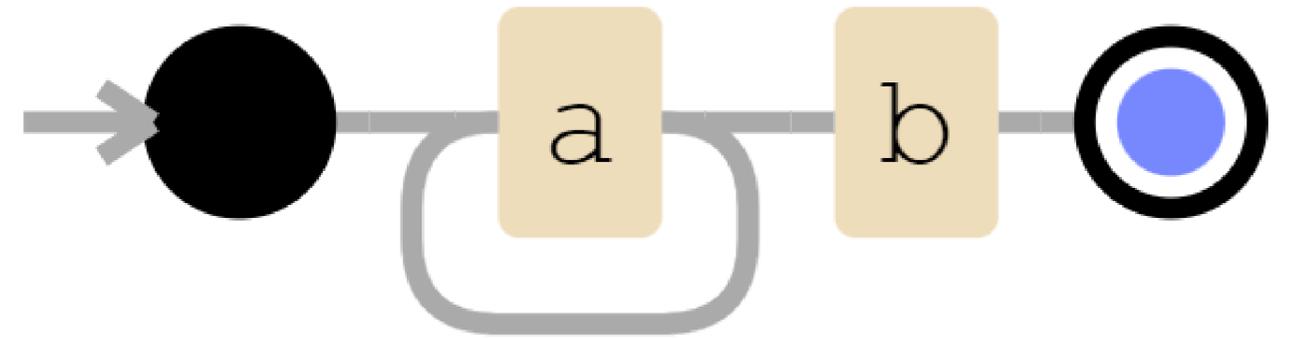


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

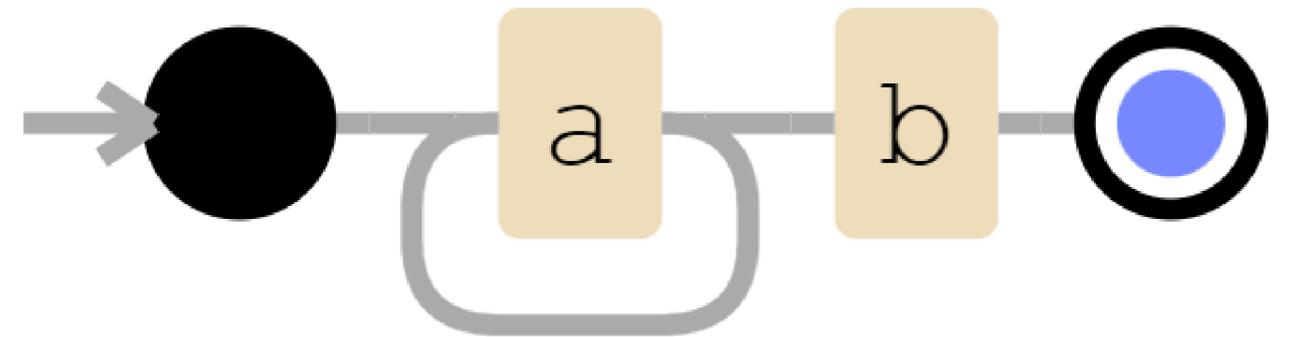


a a a

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

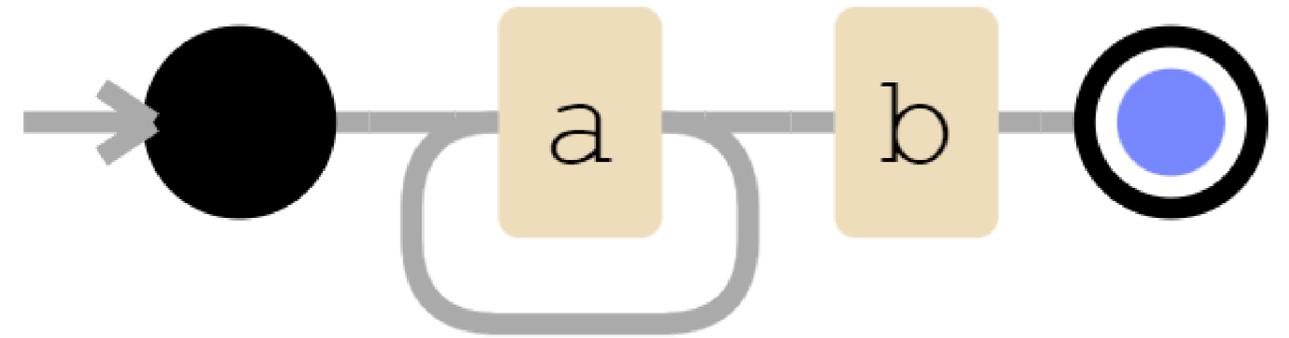


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

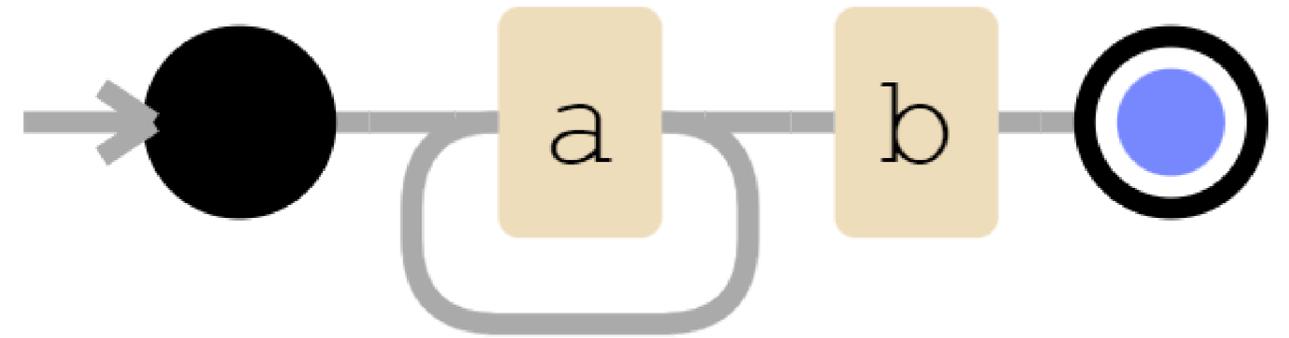


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

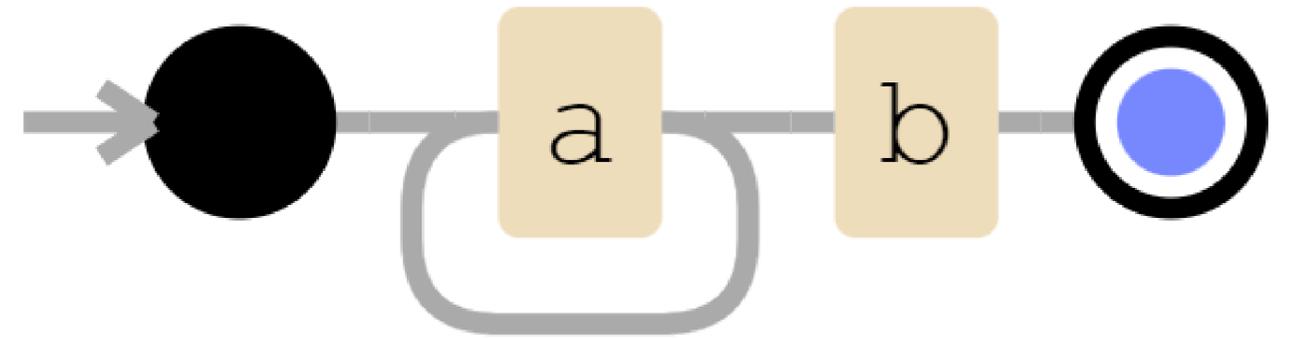


aa a

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

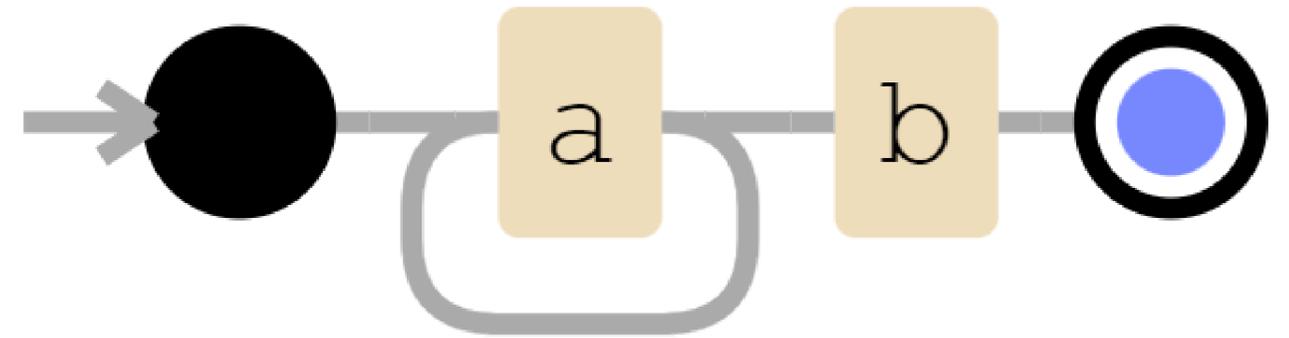


a a a

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

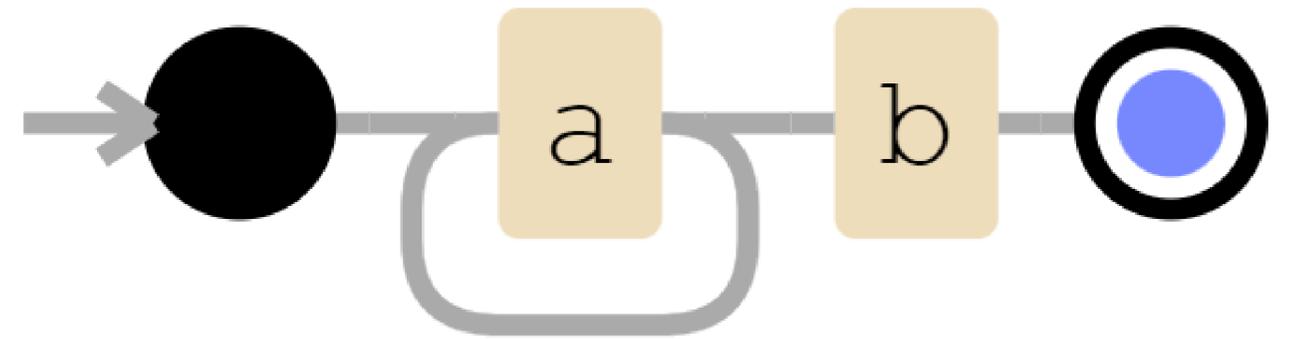


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

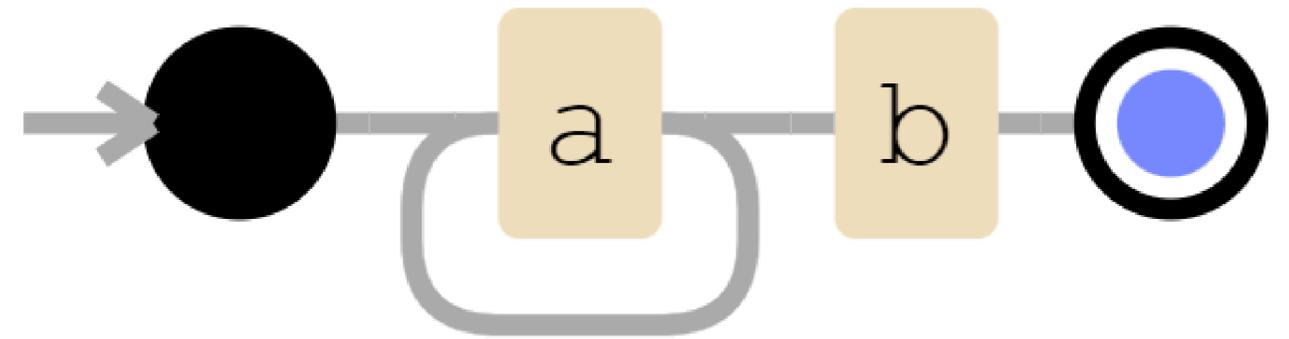


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

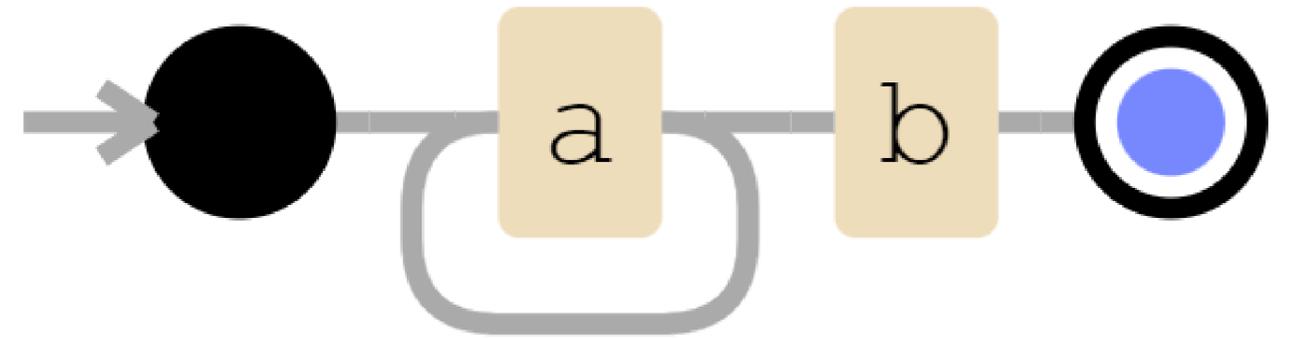


a a a

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

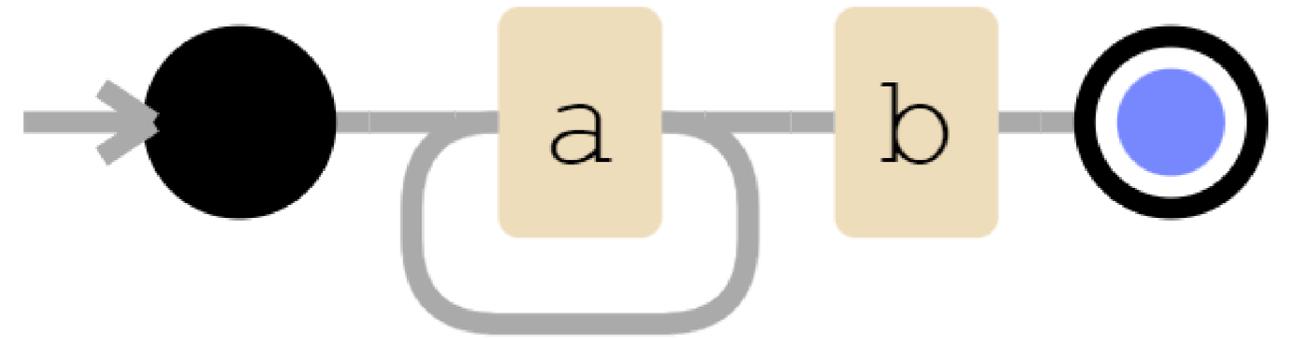


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

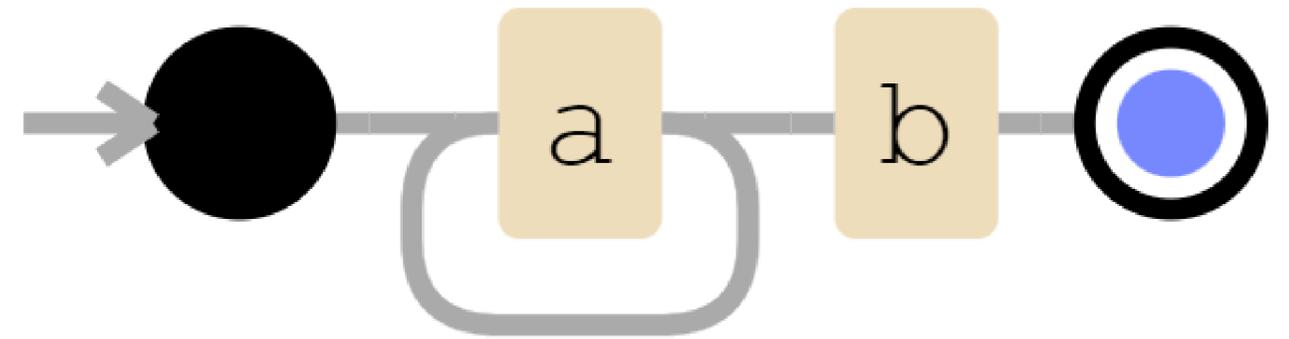


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa

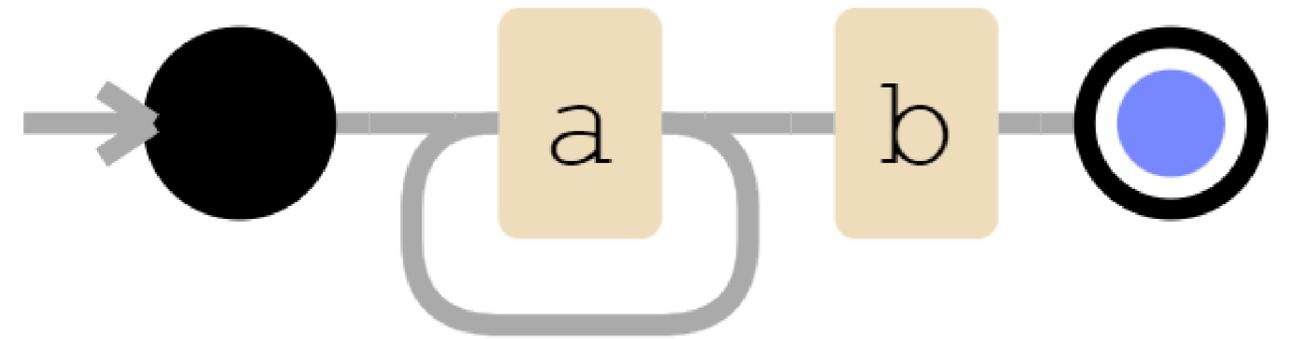


aaa

# Как работают регулярные выражения

Регулярное выражение: **a+b**

Последовательность для разбора: aaa



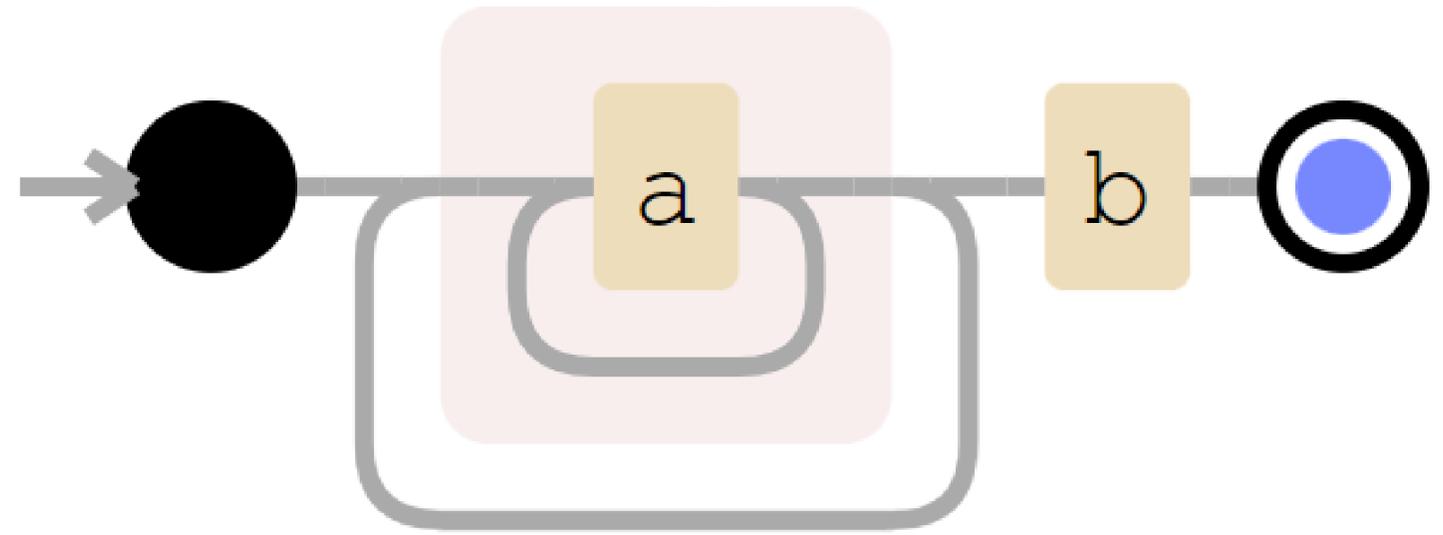
aaa

# Рассмотрим пример

Регулярное выражение:  $(a^+)+b$

# Рассмотрим пример

Регулярное выражение:  $(a^+)+b$





# Сравним два регулярных выражения

$a+b$

Input strings:

```
aaaaaab  
aaaaaaaaab  
aaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaaab  
aaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaaab
```

$(a+)+b$

Input strings:

```
aaaaaab  
aaaaaaaaab  
aaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaaab  
aaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaab  
aaaaaaaaaaaaaaaaab
```

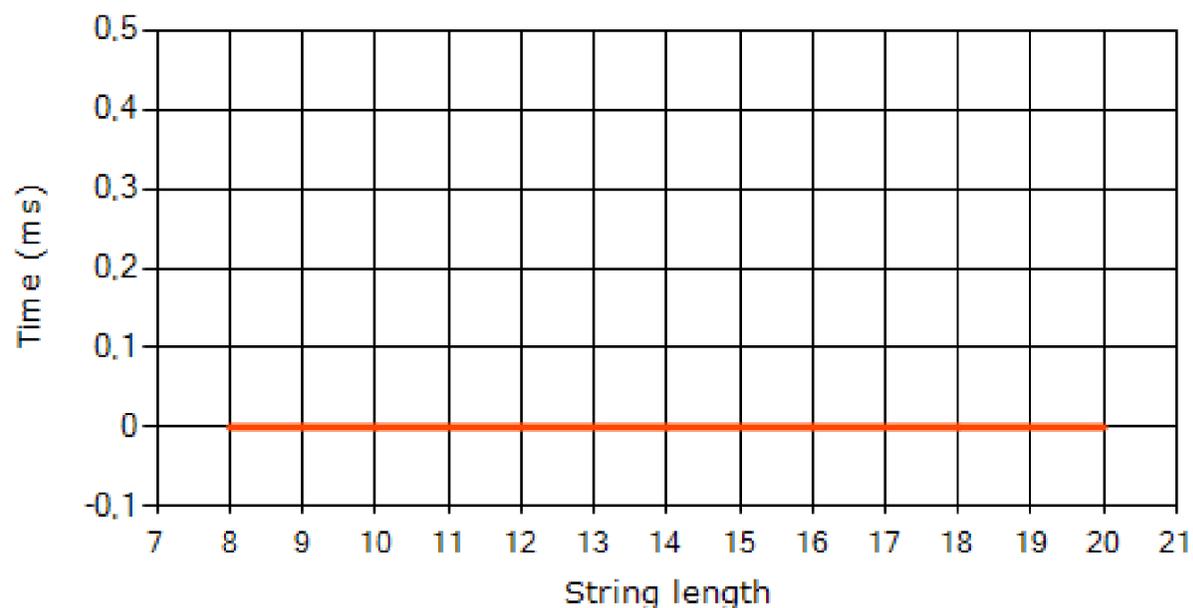
# Сравним два регулярных выражения

**a+b**

Input strings:

```
aaaaaab  
aaaaaaaaab  
aaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaab  
aaaaaaaaaaaaaab  
aaaaaaaaaaaaaab  
aaaaaaaaaaaaaab
```

Effect of input string length on regular expression execution time.

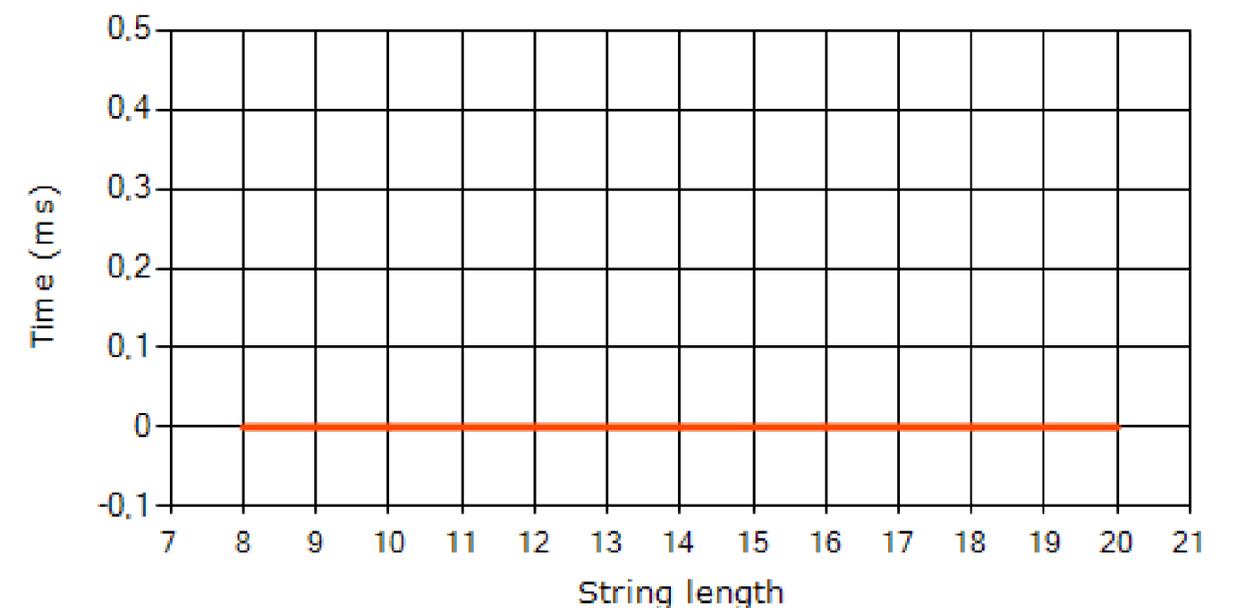


**(a+)+b**

Input strings:

```
aaaaaab  
aaaaaaaaab  
aaaaaaaaaab  
aaaaaaaaaaaab  
aaaaaaaaaaaaab  
aaaaaaaaaaaaaab  
aaaaaaaaaaaaaab  
aaaaaaaaaaaaaab
```

Effect of input string length on regular expression execution time.



# Сравним еще раз

$a+b$

Input strings:

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

$(a^+)+b$

Input strings:

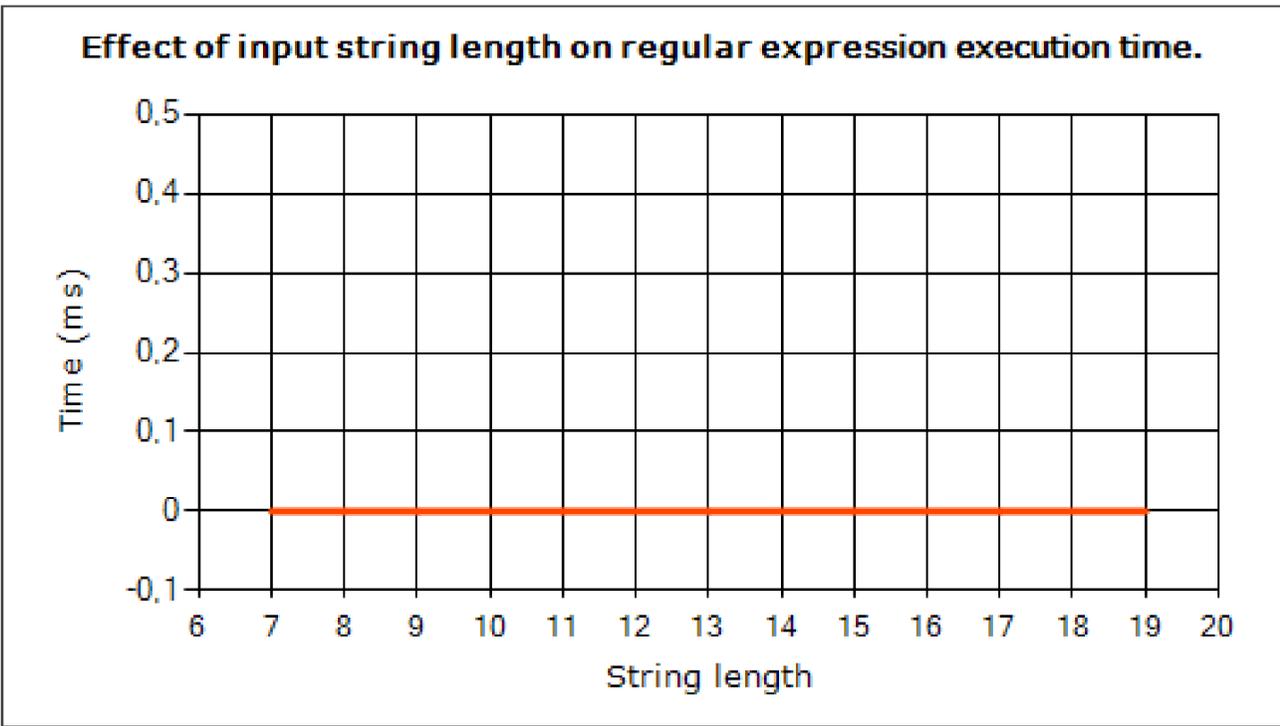
```
aaaaaaa  
aaaaaaaa  
aaaaaaaaa  
aaaaaaaaaaa  
aaaaaaaaaaaa  
aaaaaaaaaaaaa  
aaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaa
```

# Сравним еще раз

**a+b**

Input strings:

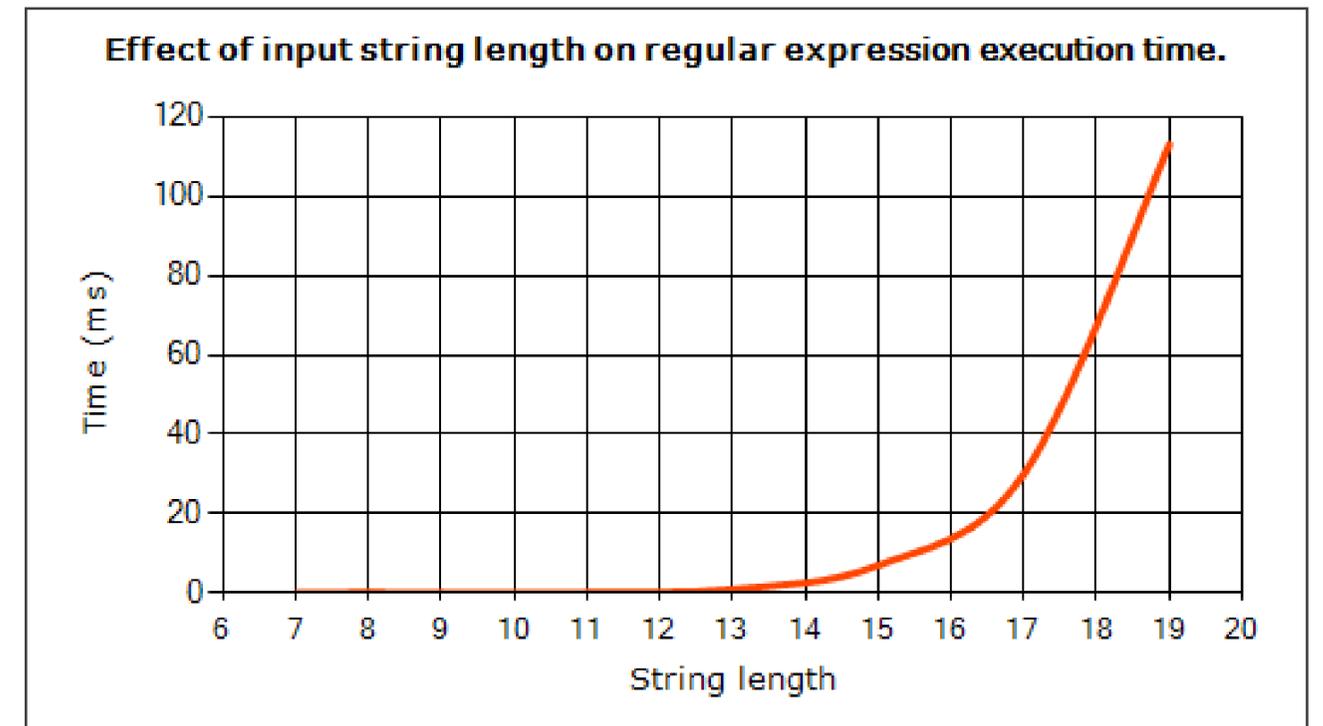
```
aaaaaaaa
aaaaaaaa
aaaaaaaaa
aaaaaaaaaaa
aaaaaaaaaaaa
aaaaaaaaaaaaa
aaaaaaaaaaaaaa
aaaaaaaaaaaaaaa
```



**(a+)+b**

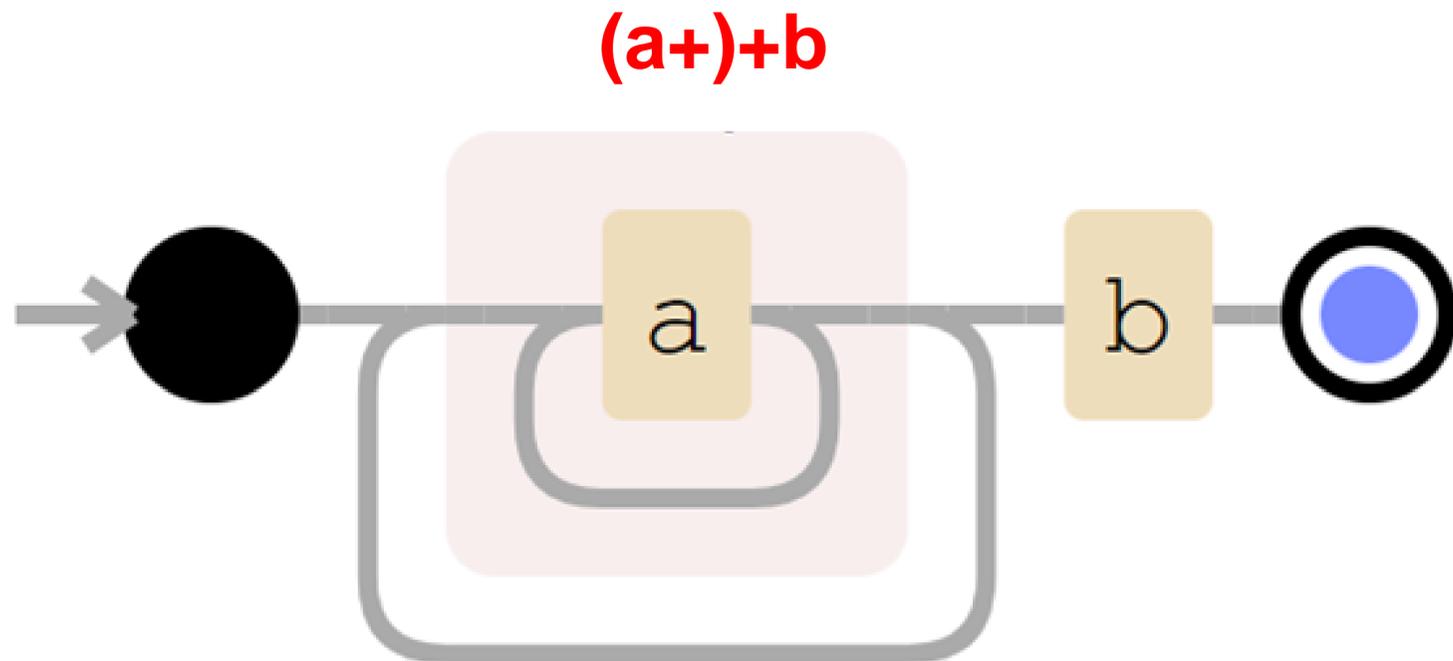
Input strings:

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



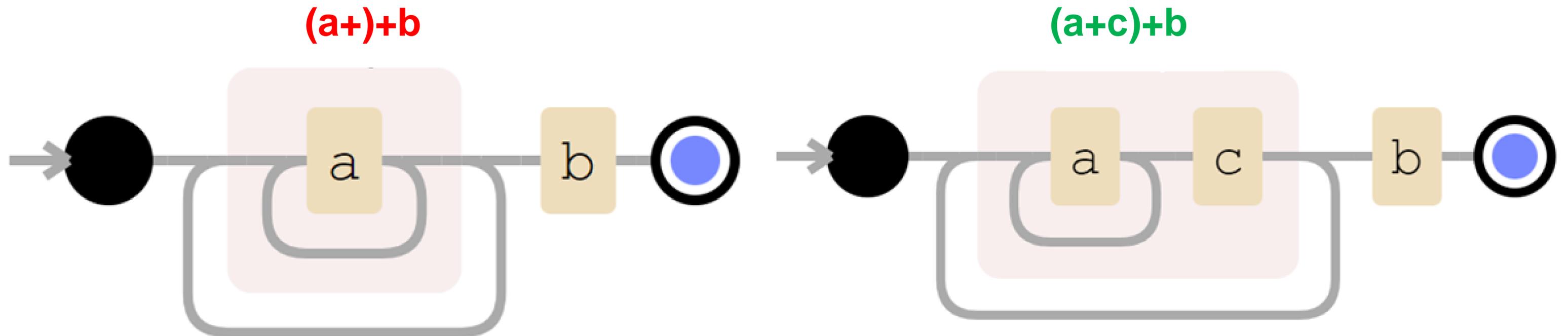
- Существует два подвыражения, при этом одно из них включено в другое и к каждому из них применяется один из следующих кванторов: '\*', '+', '\*?', '+?', '{...}'
- Существует такая строка, которую можно было бы сопоставить с обоими этими подвыражениями

- 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



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

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

$\wedge (ld\wline\wline) \$$

`^(ldlwlwld)$`

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

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

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

12a

^(**ld**lw|lwld)\$

**1**2a

^(**ld|w|w|ld**)\$

**12**a

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

12a

$\wedge (\backslash d \backslash w \backslash w \backslash d) \$$

12a

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

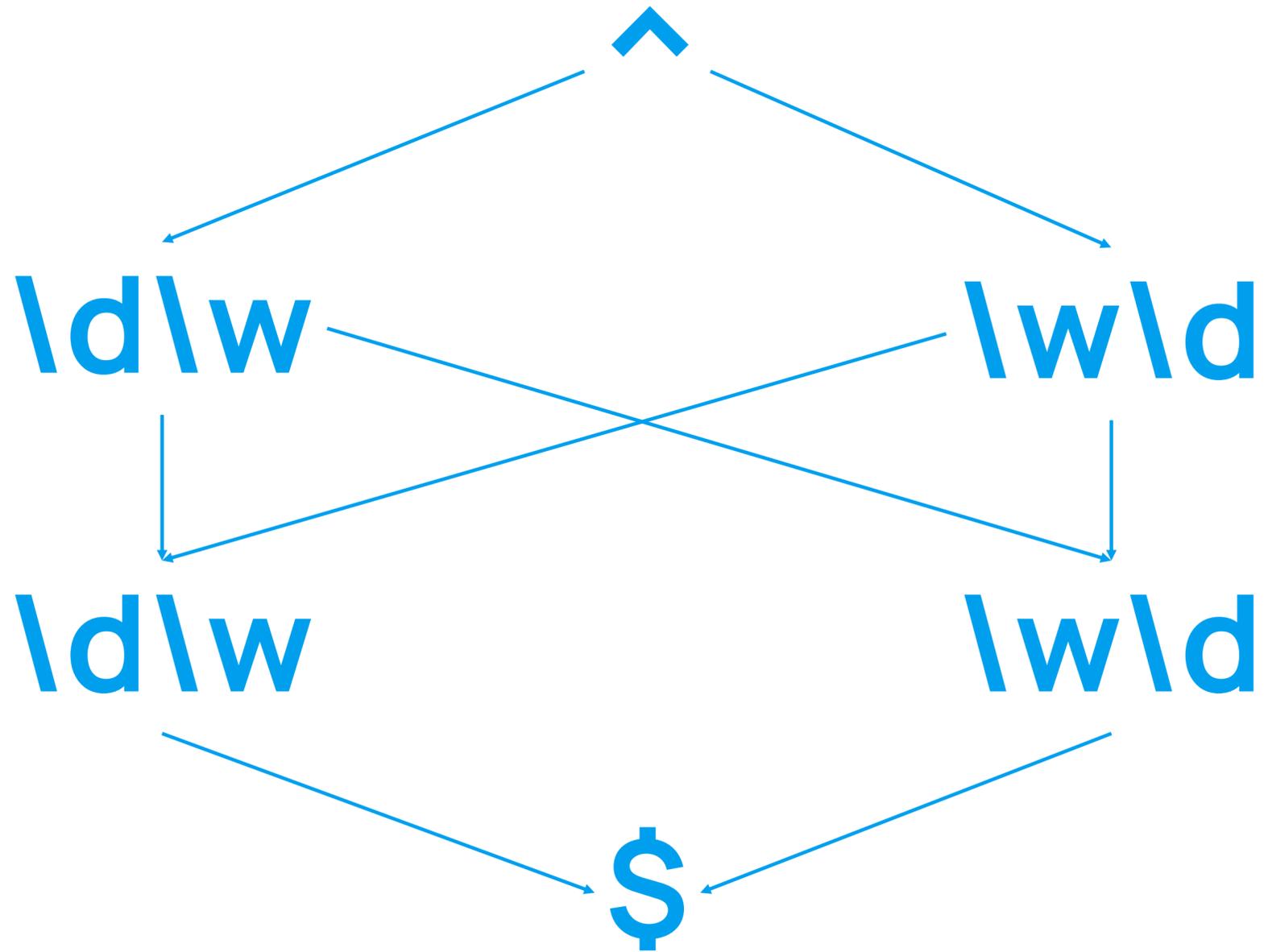
12a

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

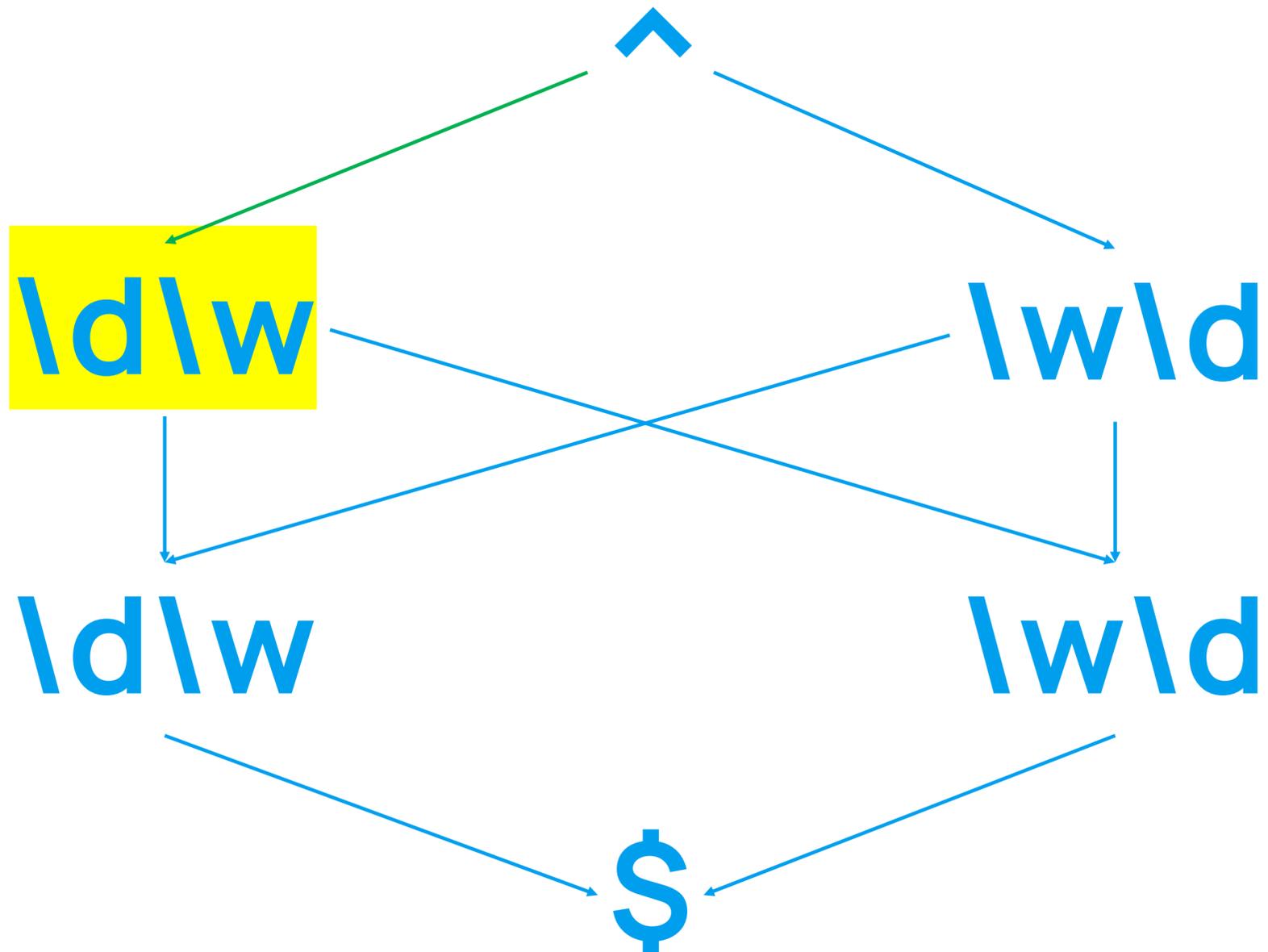
12a

$\wedge (\backslash d \backslash w | \backslash w \backslash d) (\backslash d \backslash w | \backslash w \backslash d) \$$

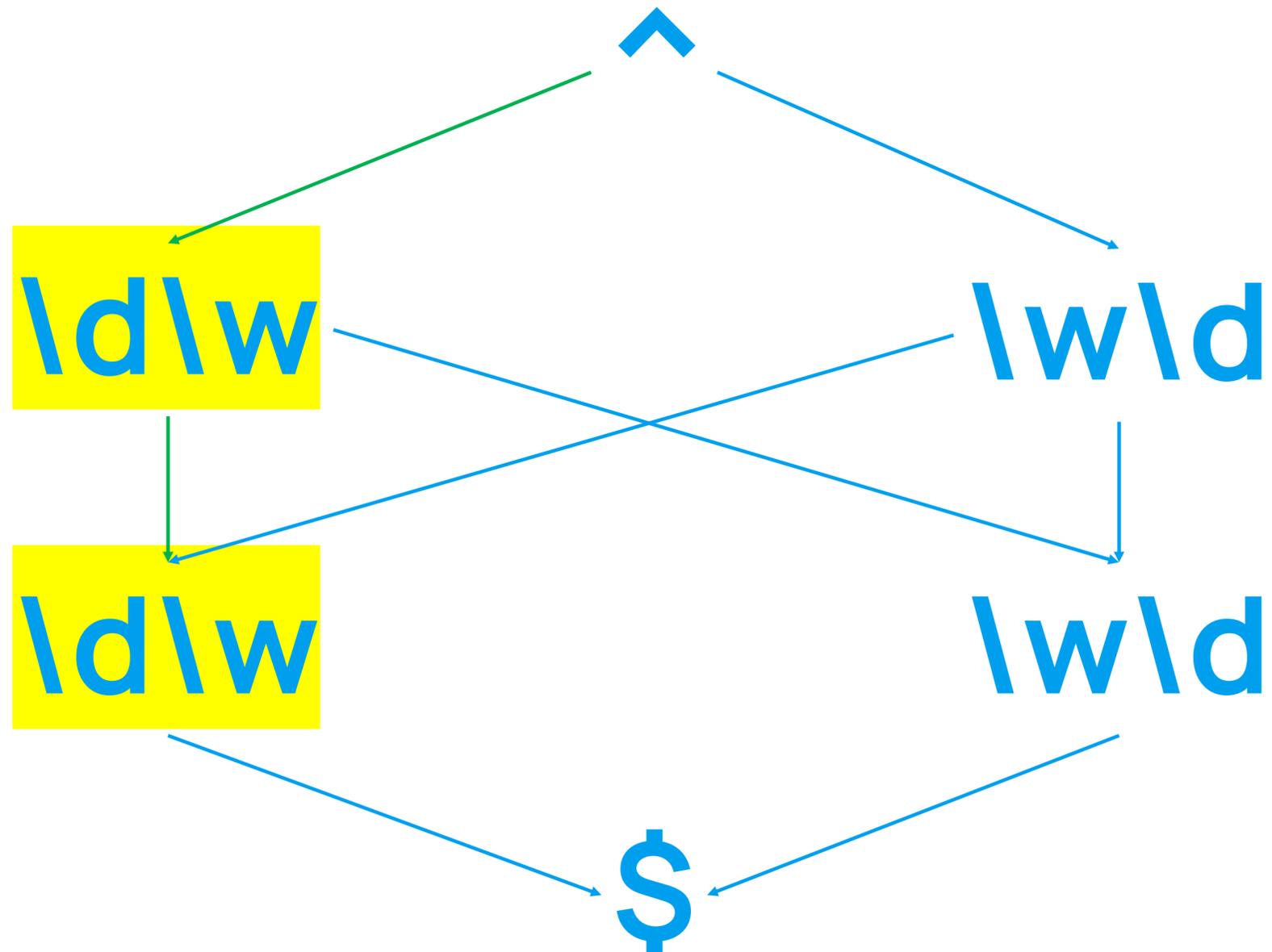
$\wedge (|d|w|w|d) (|d|w|w|d) \$$



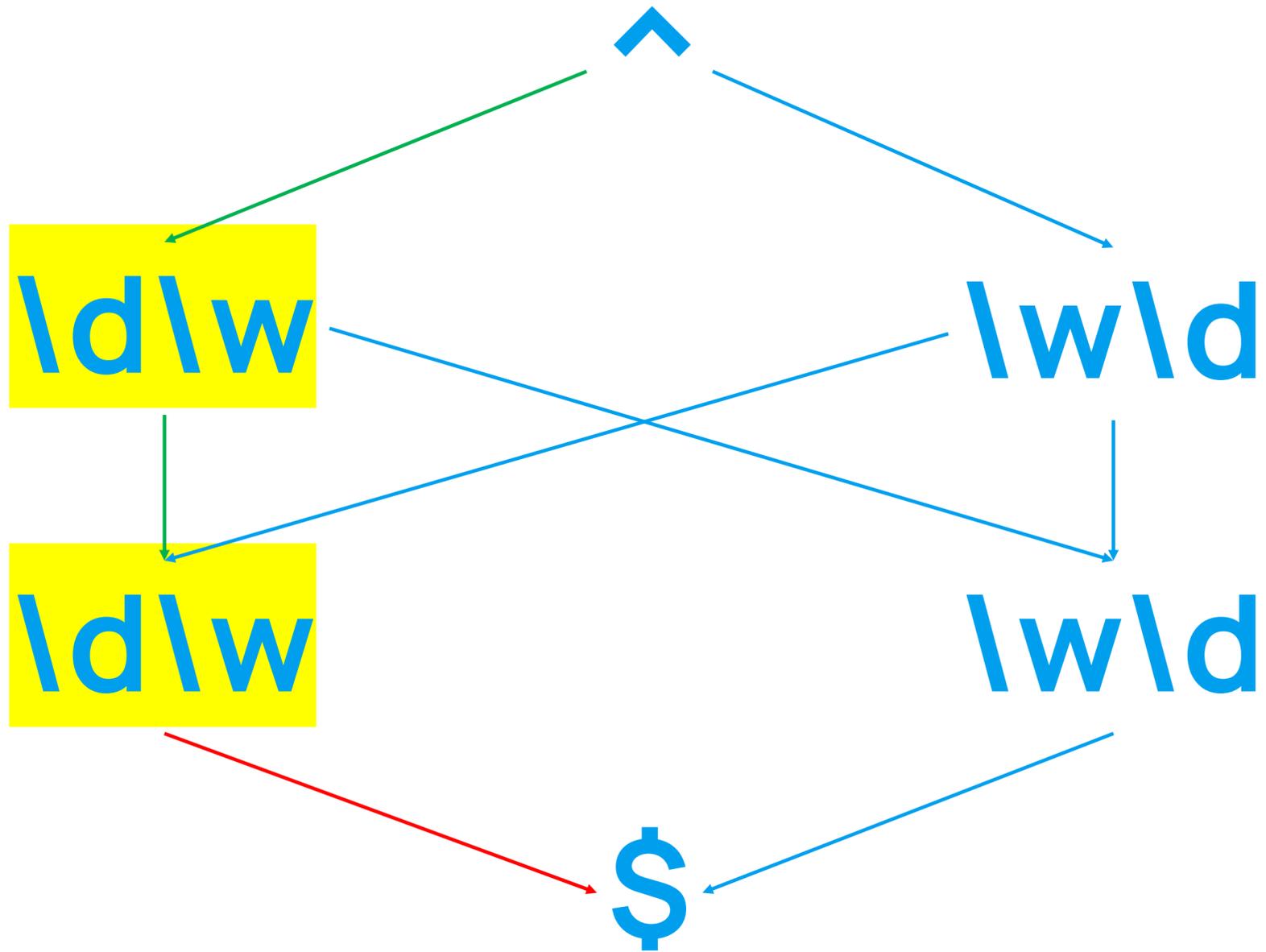
1234a



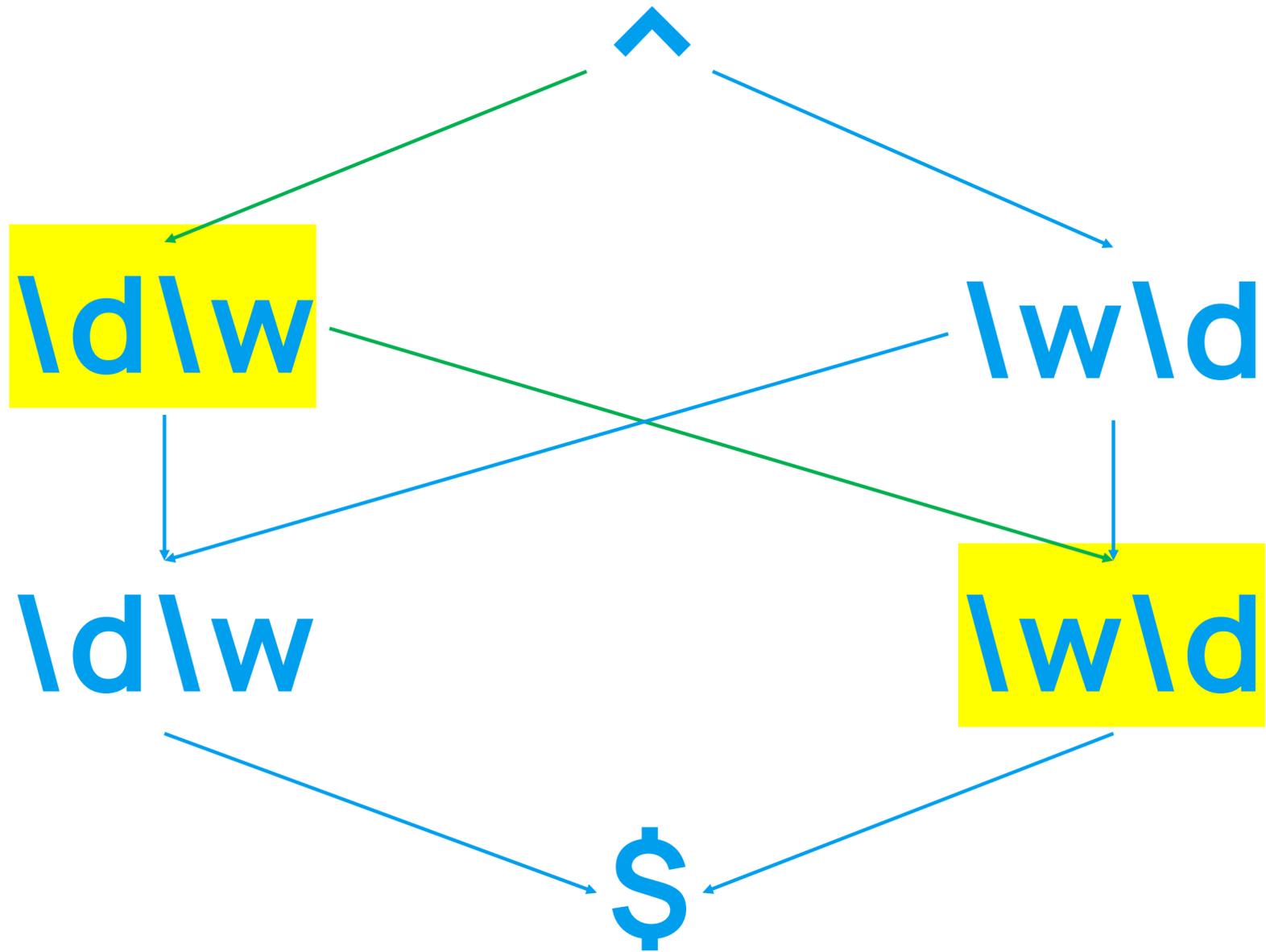
1234a



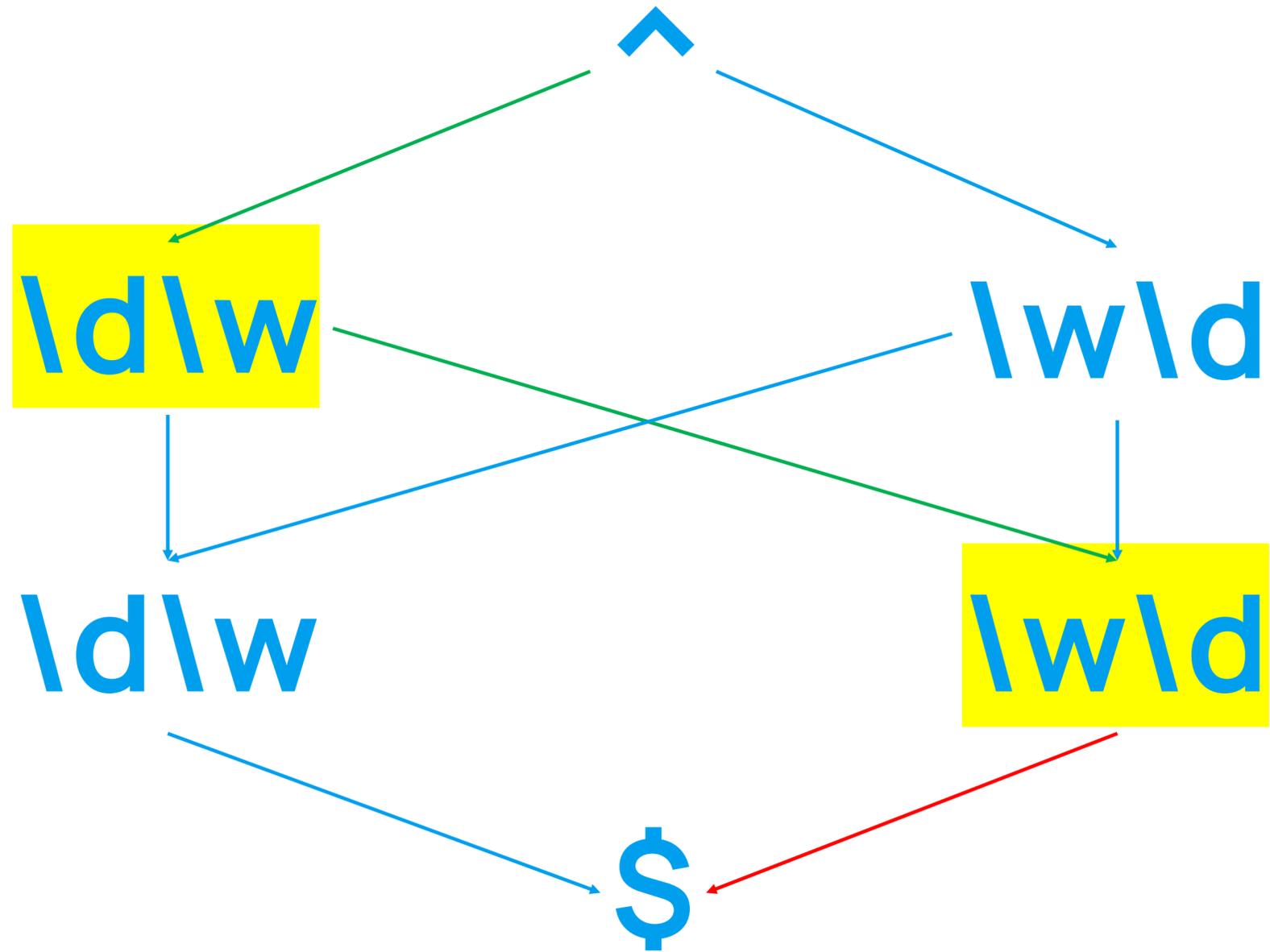
1234a



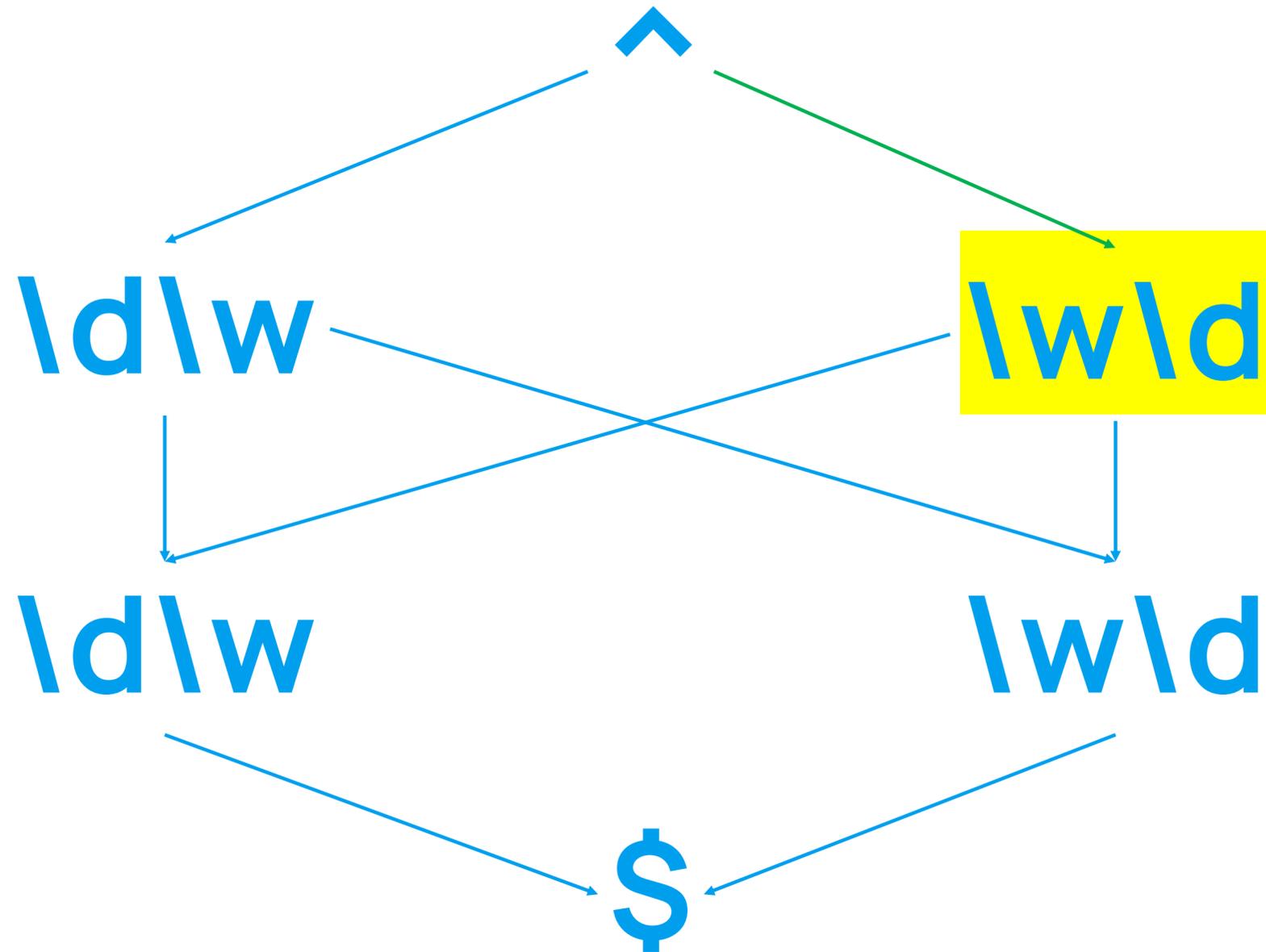
1234a



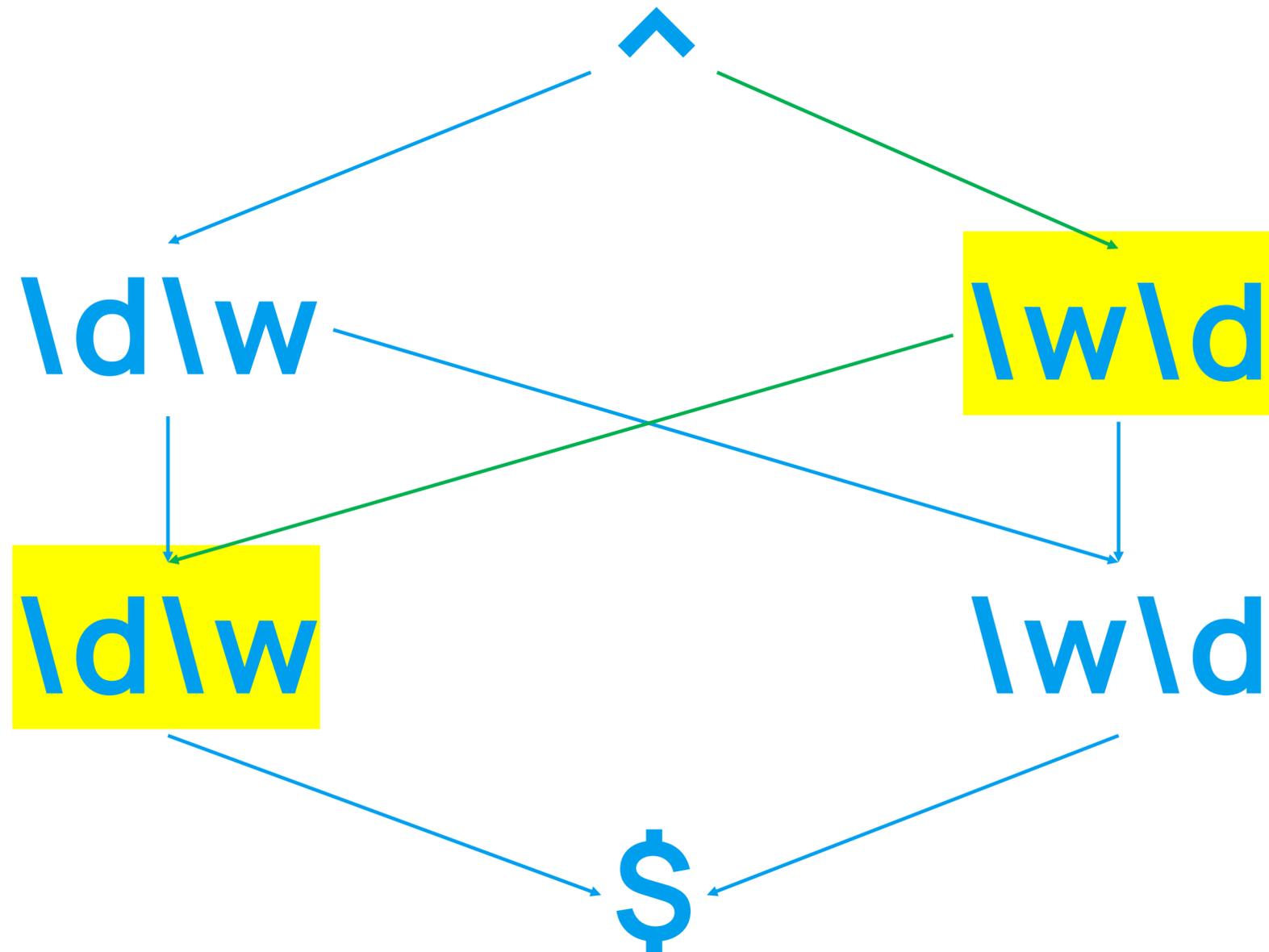
1234a



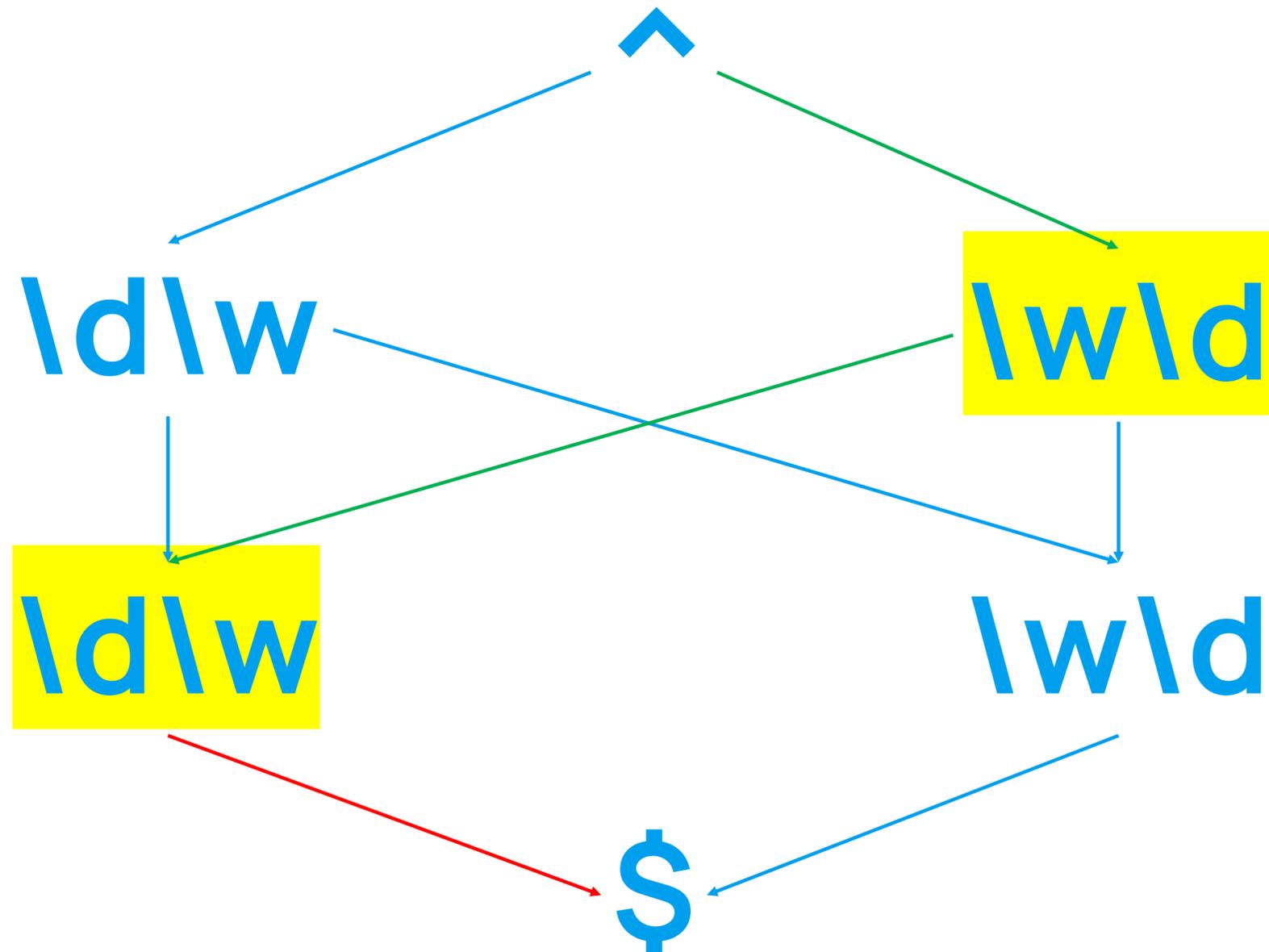
1234a



1234a

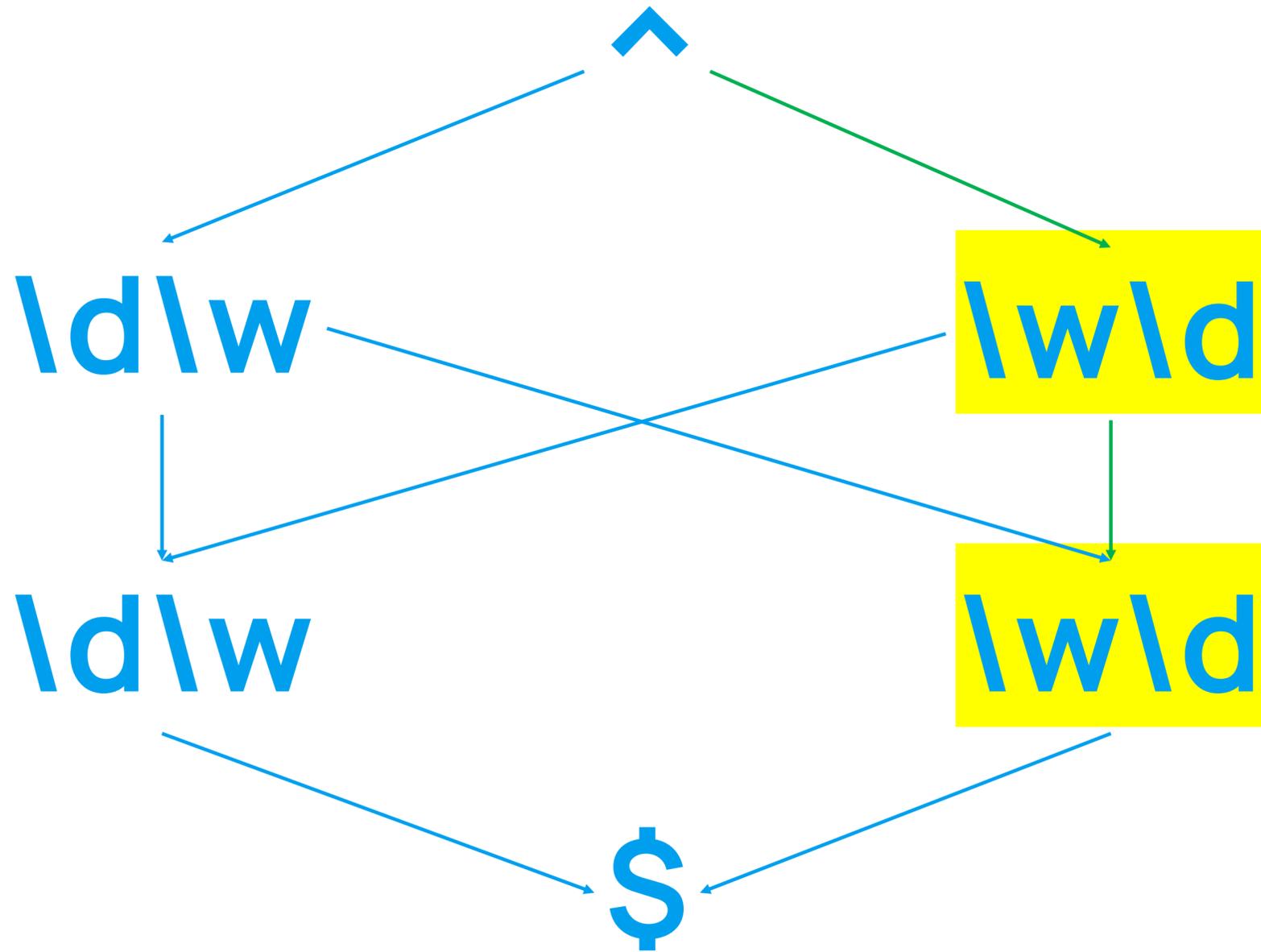


1234a

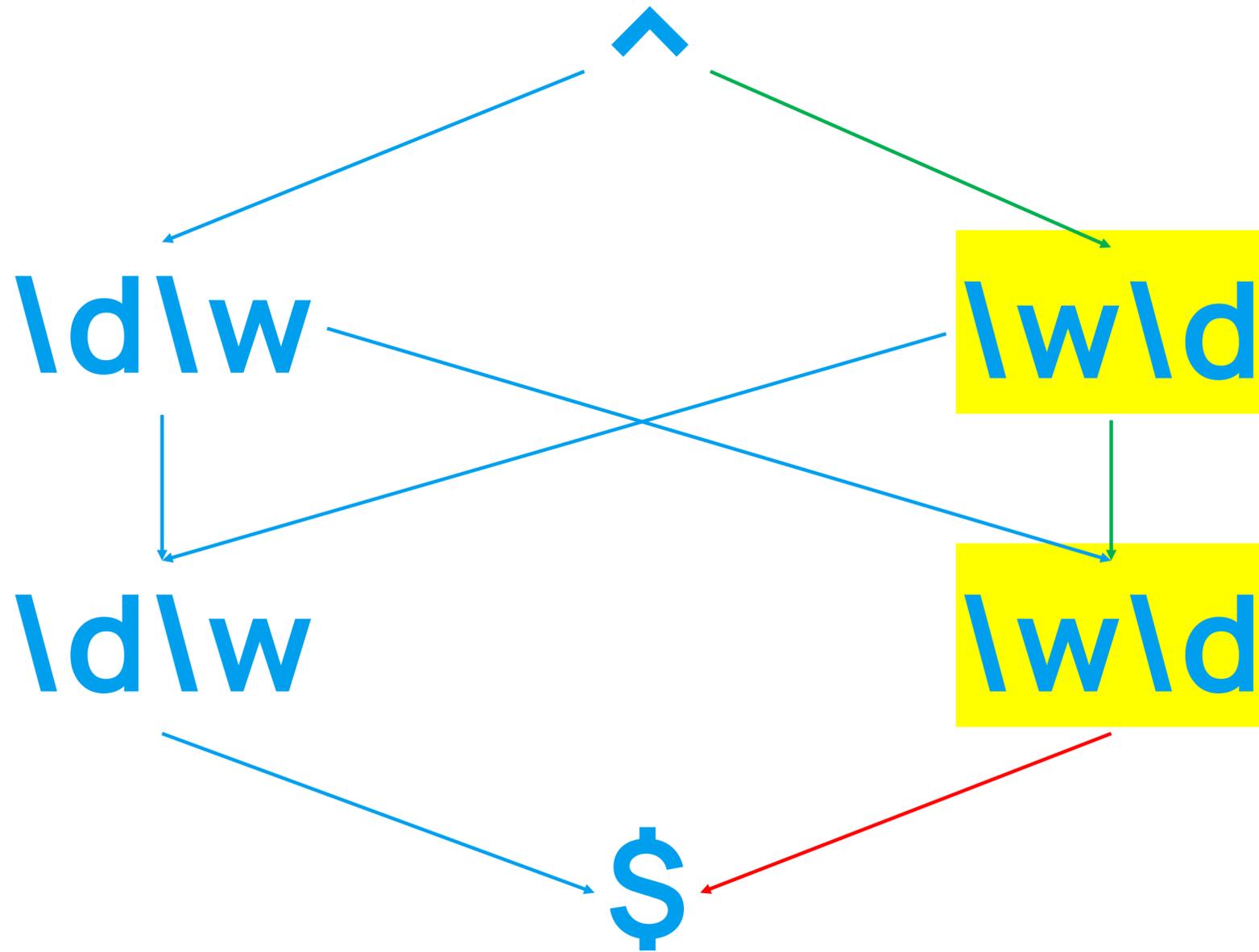


1234a

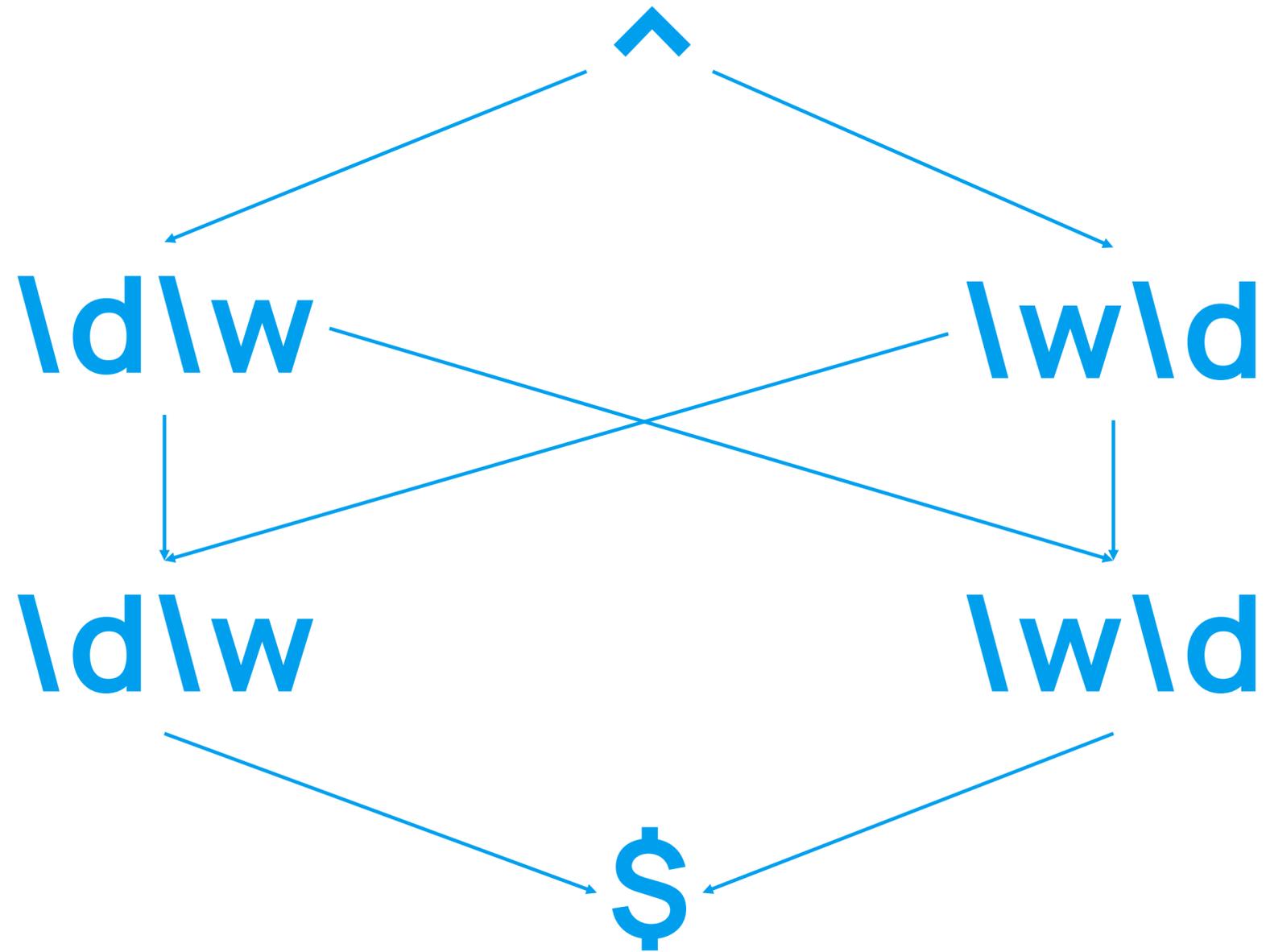
# Катастрофический возврат



1234a



1234a



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

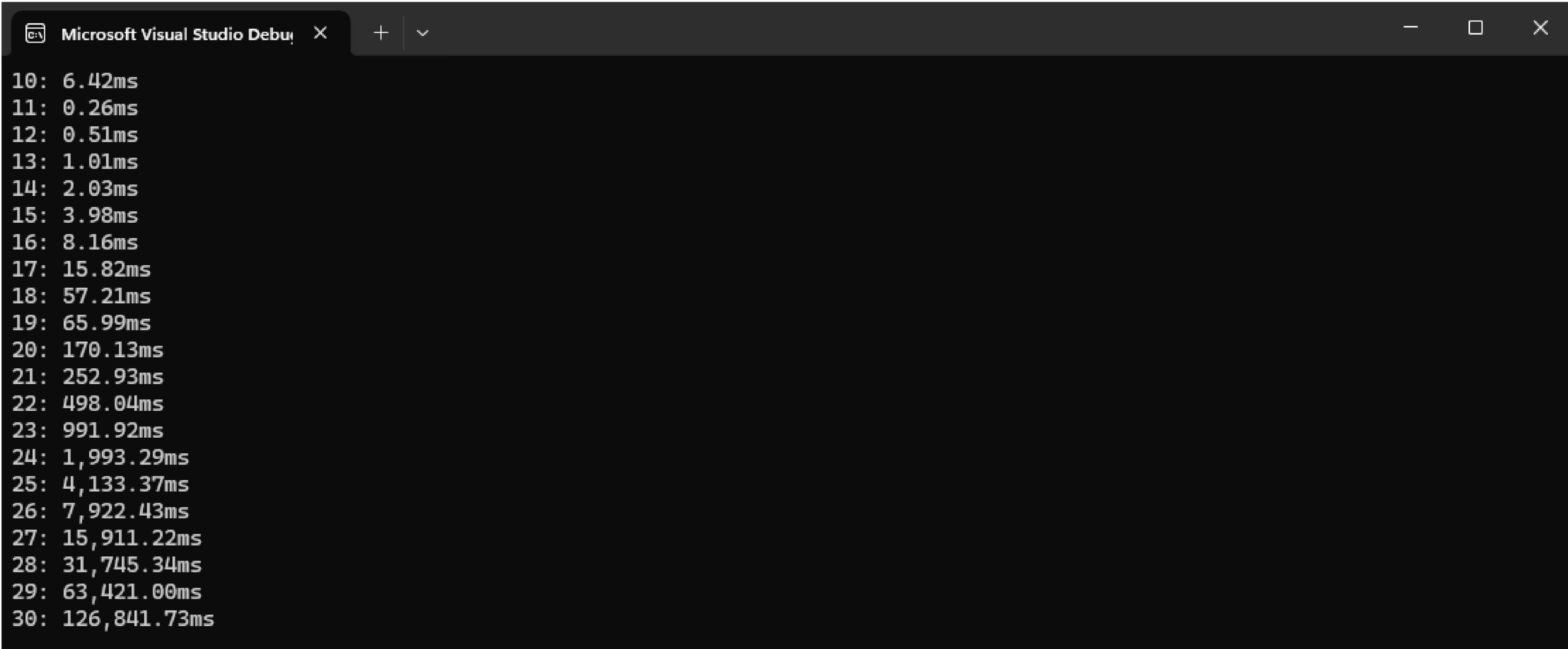
```
using System.Diagnostics;
using System.Text.RegularExpressions;

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

89



```
Microsoft Visual Studio Debug Console
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

```
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");
}
```



A red arrow points downwards from the top right towards the `RegexOptions.NonBacktracking` parameter in the code, which is highlighted with a red rounded rectangle.

# Катастрофический возврат

93

Microsoft Visual Studio Debu

+ v

- □ ×

```
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

- Эффективнее в «хороших» случаях

- Эффективнее в «хороших» случаях
- Проверочные условия с просмотром 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

```
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

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

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

```
Microsoft Visual Studio Debu... X + v
Без просмотра вперед: False in 00:00:00.0386950
С просмотром вперед: False in 00:00:00.0006722
```

- 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

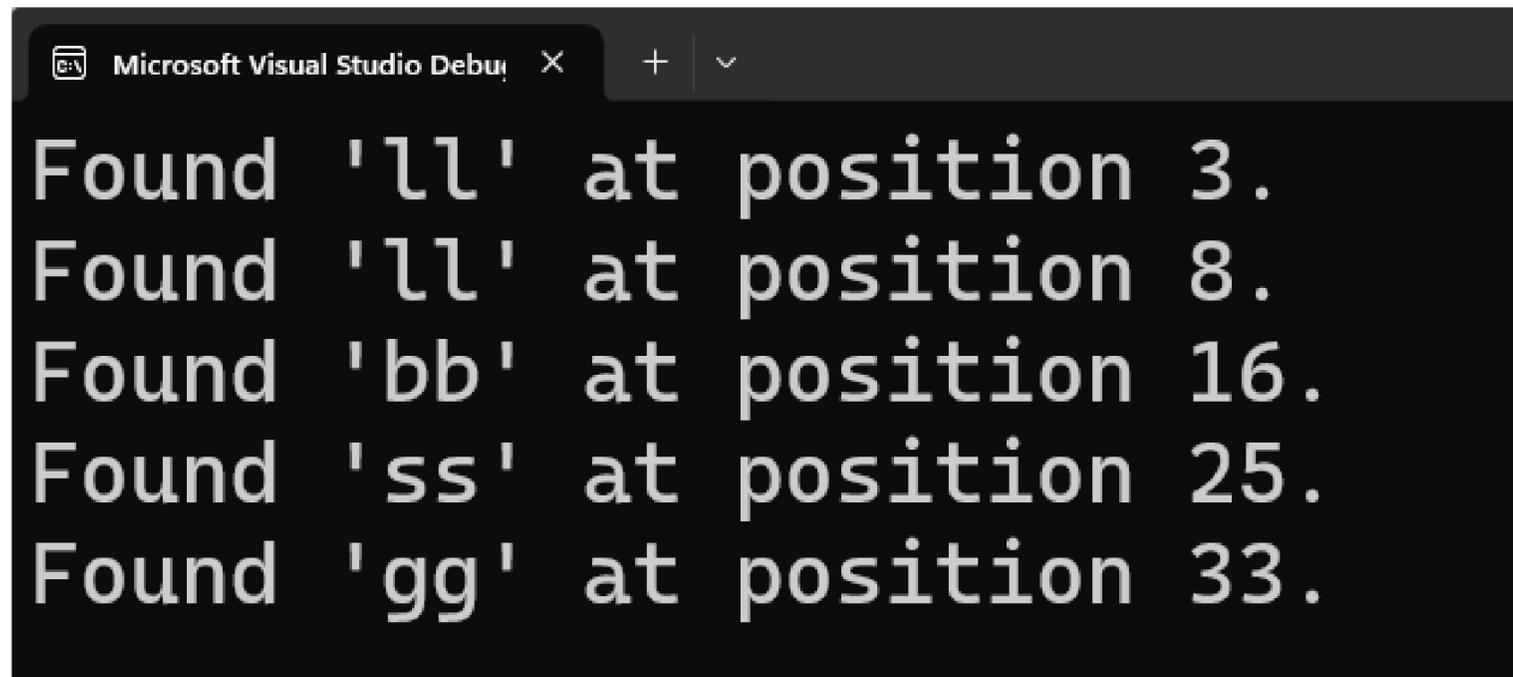
```
string pattern = @"(\w)\1";  
string input = "trellis llama webbing dresser swagger";  
foreach (Match match in Regex.Matches(input, pattern))  
    Console.WriteLine($"Found '{match.Value}' at position {match.Index}.");
```

```
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}.");
```

```
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

```
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}.");
```



```
Microsoft Visual Studio Debug Console  
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.
```

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

```
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

111

```
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}");
```

Microsoft Visual Studio Debug

+

∨

Обычное сопоставление: True in 00:00:00.0134259

Условное сопоставление: True in 00:00:00.0002450

# Преимущества обратного отслеживания

- Позволяет быстрее работать в «хороших» случаях

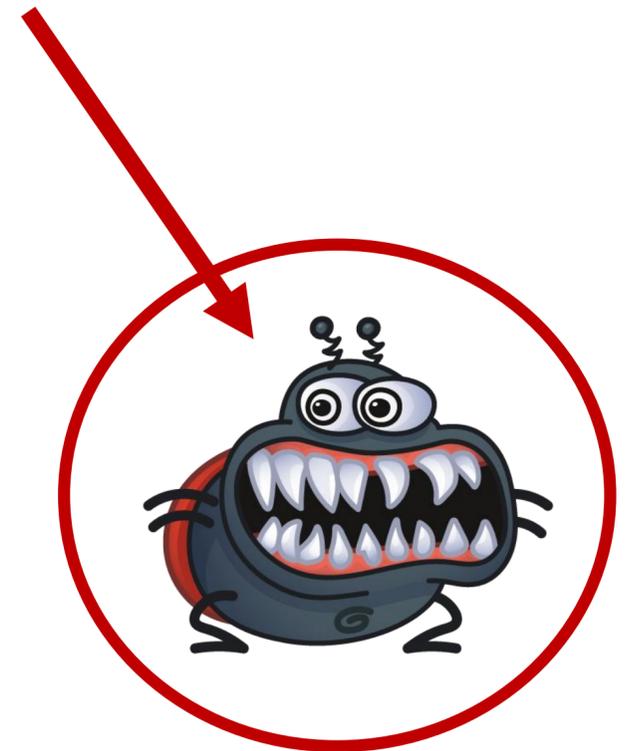
- Позволяет быстрее работать в «хороших» случаях
- Могут быстрее работать в «плохих» случаях

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



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

```
public static string ReplaceRegexInName(string name)
{
    const string dateTimeFileNameRegex = @"(?:([ymdhfszgkt]+[-_ ]*)+[\?])";
    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]+)+[?]

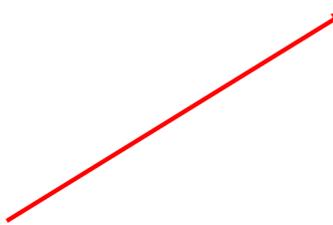
(a+)+b



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

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

(a+)+b



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

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

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

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

0 вхождений [-\_]



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



[?] ([ymdhstzqkt]+)+[?]

0 вхождений [-\_]



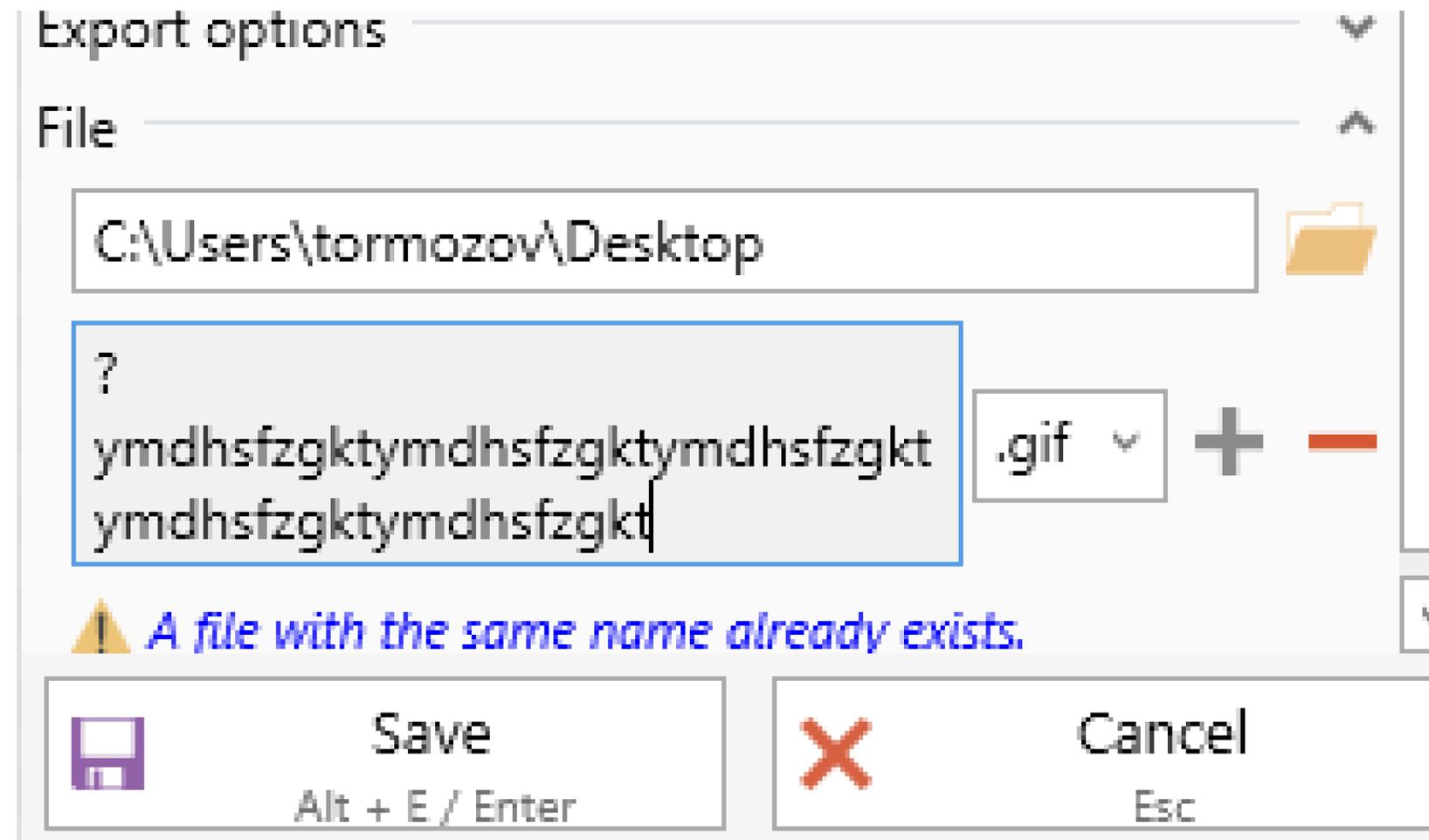
? ymdhsfzgtk ymdhsfzgtk ymdhsfzgtk ymdhsfzgtk ymdhsfzgtk

?ymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgktymdhsfzgk



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







- CVE-2021-27293

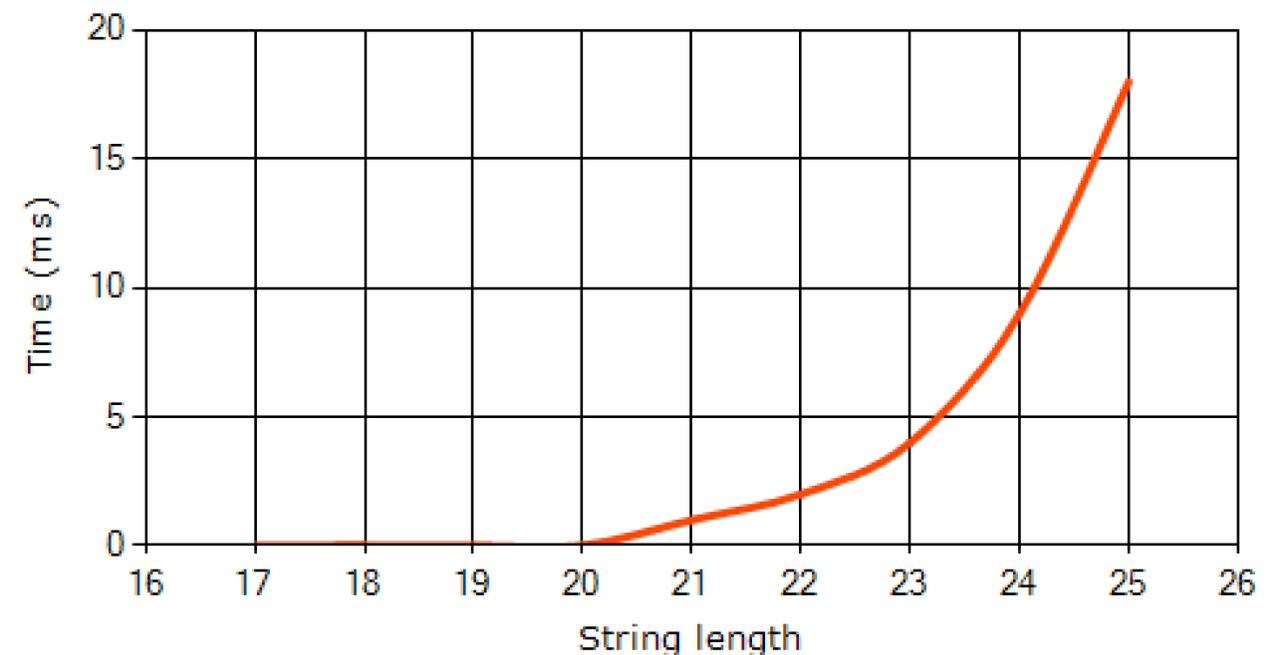
- CVE-2021-27293
- Regex: `"newDate\((-?\d+)*\"`
- Ищете строку формата:  
*newDate(10-01-2007)*

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

Input strings:

```
newDate(1111111111
newDate(1111111112
newDate(11111111133
newDate(111111111444
newDate(1111111115555
newDate(11111111166666
newDate(111111111777777
newDate(1111111118888888
newDate(111111111999999999
```

Effect of input string length on regular expression execution time.



- CVE-2015-2526

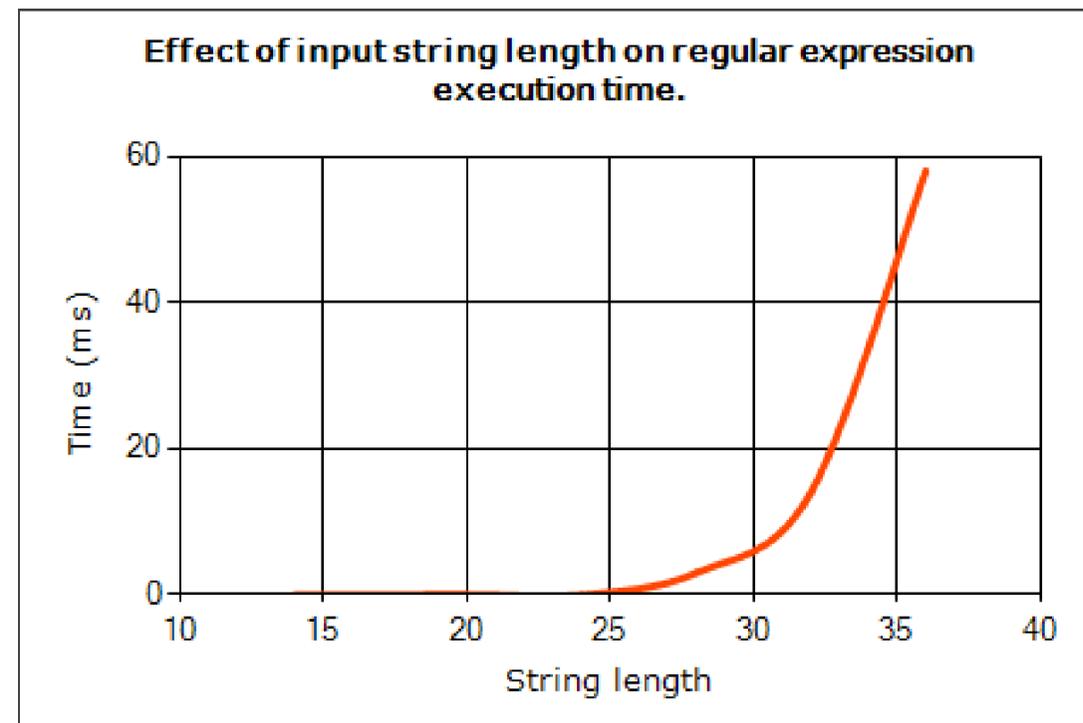
- CVE-2015-2526
- Классы  
EmailAddressAttribute  
и PhoneAttribute
- Попытка валидации  
email и номера  
телефона

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

## email validation

Input strings:

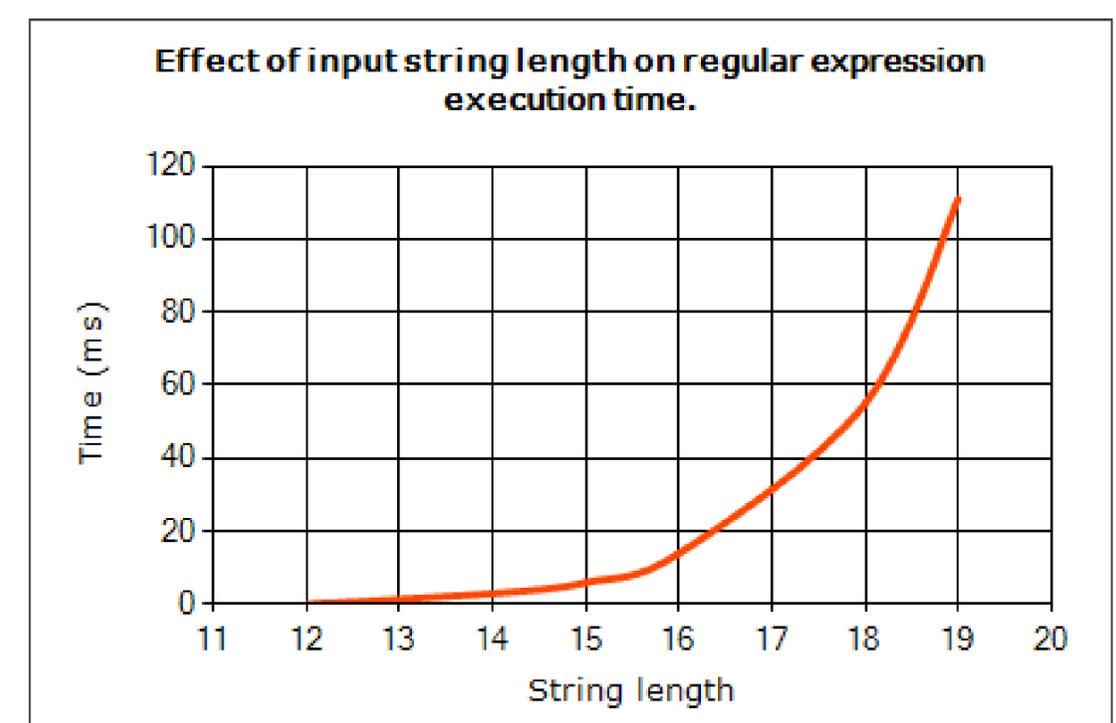
```
t@t.t.t.t.c%20  
t@t.t.t.t.t.t.c%20  
t@t.t.t.t.t.t.t.c%20  
t@t.t.t.t.t.t.t.t.c%20  
t@t.t.t.t.t.t.t.t.t.c%20  
t@t.t.t.t.t.t.t.t.t.t.c%20
```



## phone validation

Input strings:

```
6666666666d  
66666666666666d  
666666666666666d  
6666666666666666d  
66666666666666666d  
666666666666666666d
```





- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

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



- **Cloudflare (2019)**

WAF правило содержало опасное регулярное выражение

- **Stack Overflow (2016)**

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

# Как бороться с ReDoS



- Использовать другое регулярное выражение



- Использовать другое регулярное выражение
- Заменить регулярное выражение рукописным обработчиком



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



- 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



```
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.");
}
```

```
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: аааааааааааааааа!**

```
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.");
}
```

**Результат – ReDoS!**

**Пользовательский ввод:**

**login: (a\*)\*b**

**password: аааааааааааааааааа!**



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



Третий аргумент конструктора регулярных выражений – **matchTimeout**

В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

Третий аргумент конструктора регулярных выражений – **matchTimeout**

В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

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

Третий аргумент конструктора регулярных выражений – **matchTimeout**

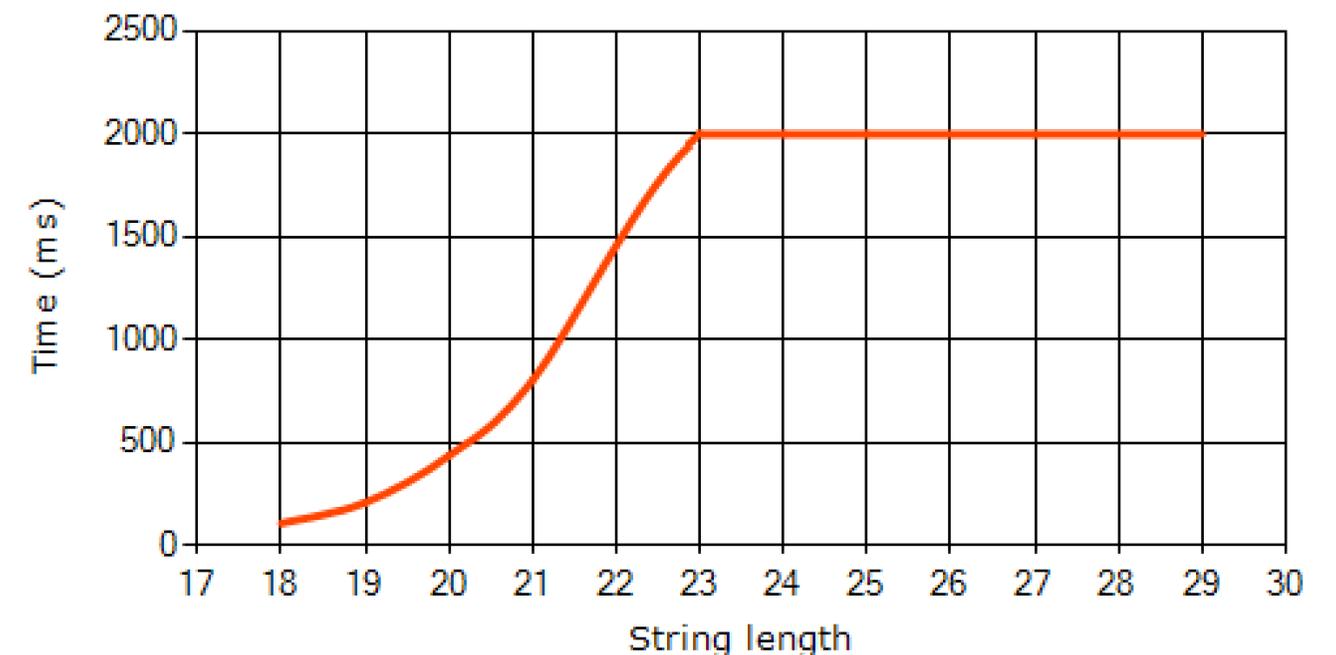
В случае превышения времени разбора выбросит **RegexMatchTimeoutException**

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

Input strings:

```
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa
```

Effect of input string length on regular expression execution time.



- 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



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

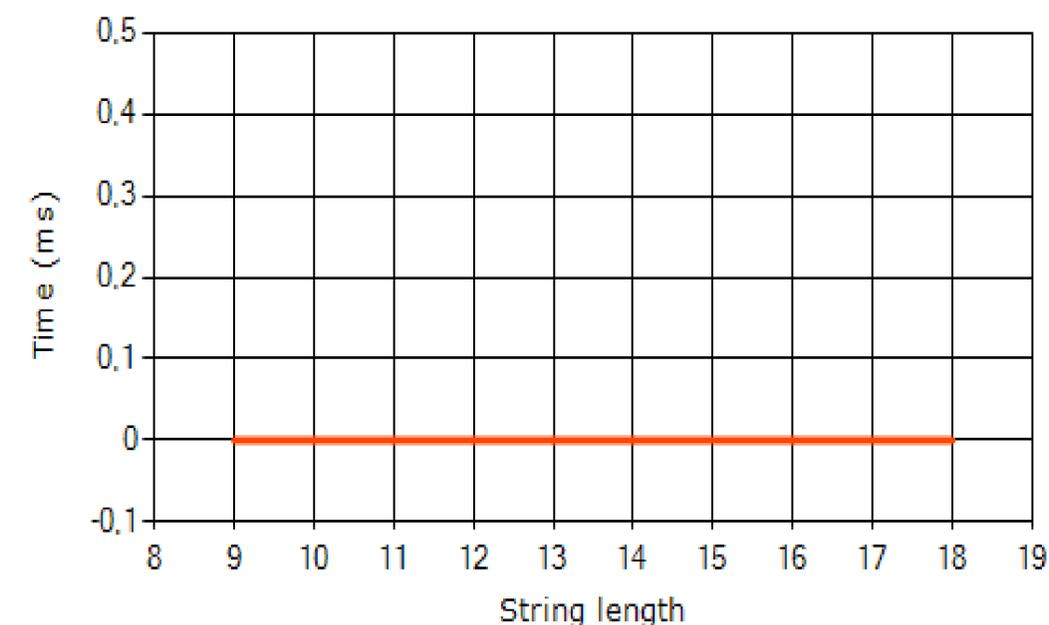
- Синтаксис определения атомарных групп:  
(?>...)
- Для выражения внутри отключается функция обратного отслеживания
- **Внимание!** Использование атомарных групп меняет логику обработки соответствий

(?>a\*)\*b

Input strings:

```
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa  
aaaaaaaaa
```

Effect of input string length on regular expression execution time.



- 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



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

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

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

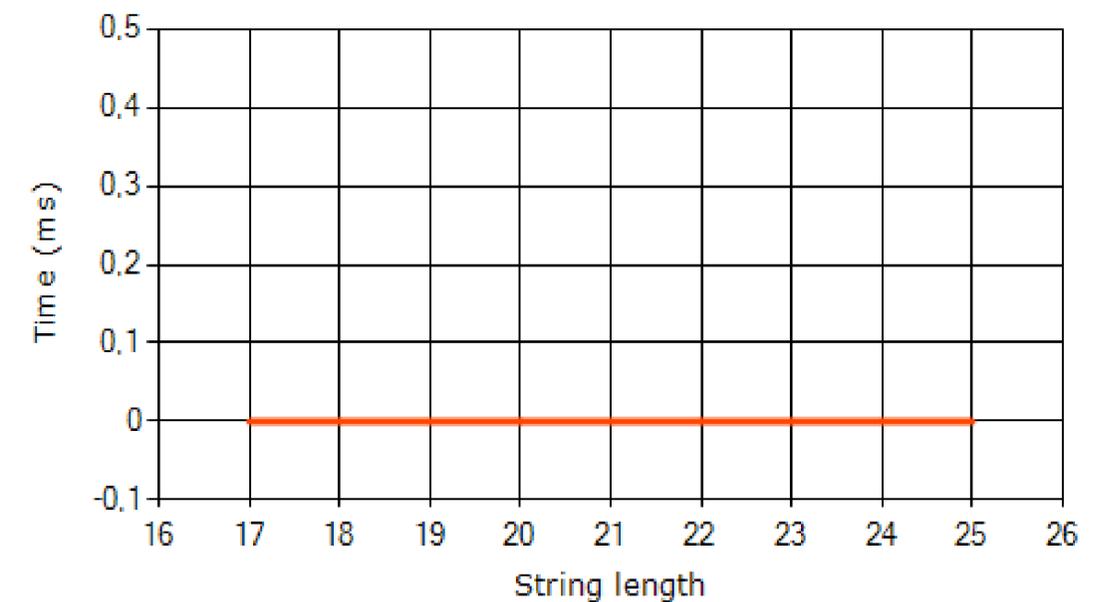
```
Regex regex = new Regex(@"newDate\((-?\d+)*\)",  
                        RegexOptions.NonBacktracking);
```

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

Input strings:

```
newDate(111111111  
newDate(111111112  
newDate(1111111133  
newDate(11111111444  
newDate(111111115555  
newDate(1111111166666  
newDate(11111111777777  
newDate(111111118888888  
newDate(1111111199999999
```

Effect of input string length on regular expression execution time.



# Профилактика ReDoS уязвимостей



- 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	<code>/^([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})\$/</code>
<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.com,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 <b>Rating:</b> <input checked="" type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>
<b>Source</b>	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

**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 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.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

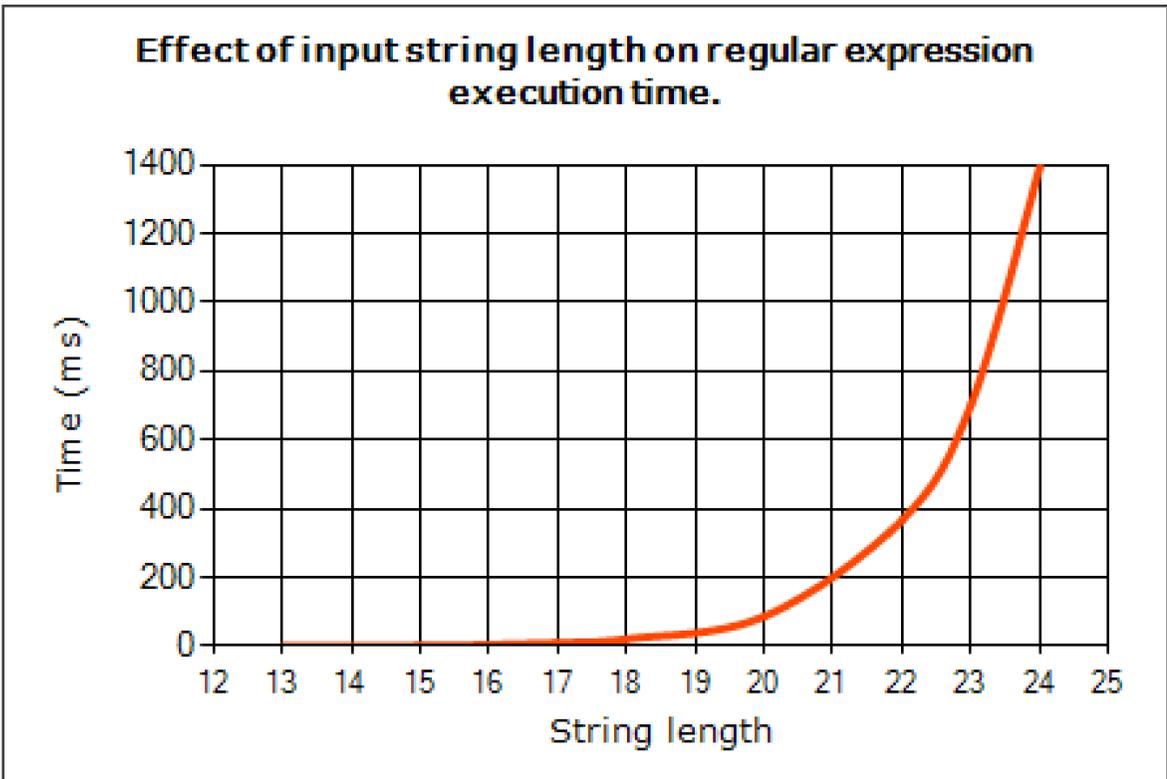
**Author** Rafiq Rating:

**Source** Email

**Your Rating** Bad  1  2  3  4  5 Good Submit Rating

Input strings:

```
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!  
aaaaaaaaaaaaa!
```



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



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



- 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