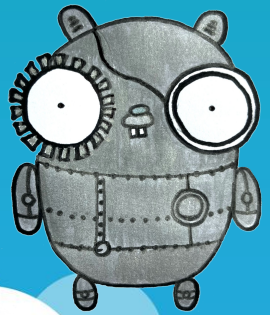
FOSDEM Go Devroom, February 2022

Fun with Finite Automata

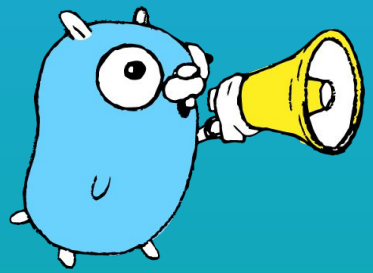


Bryan Boreham

Grafana Labs

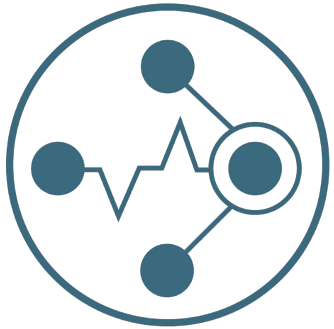
@bboreham

Overview



- Who am I?
- Why am I talking about this?
- Automa-what?
- How do regexps work?
- Where did I speed them up?

I work on

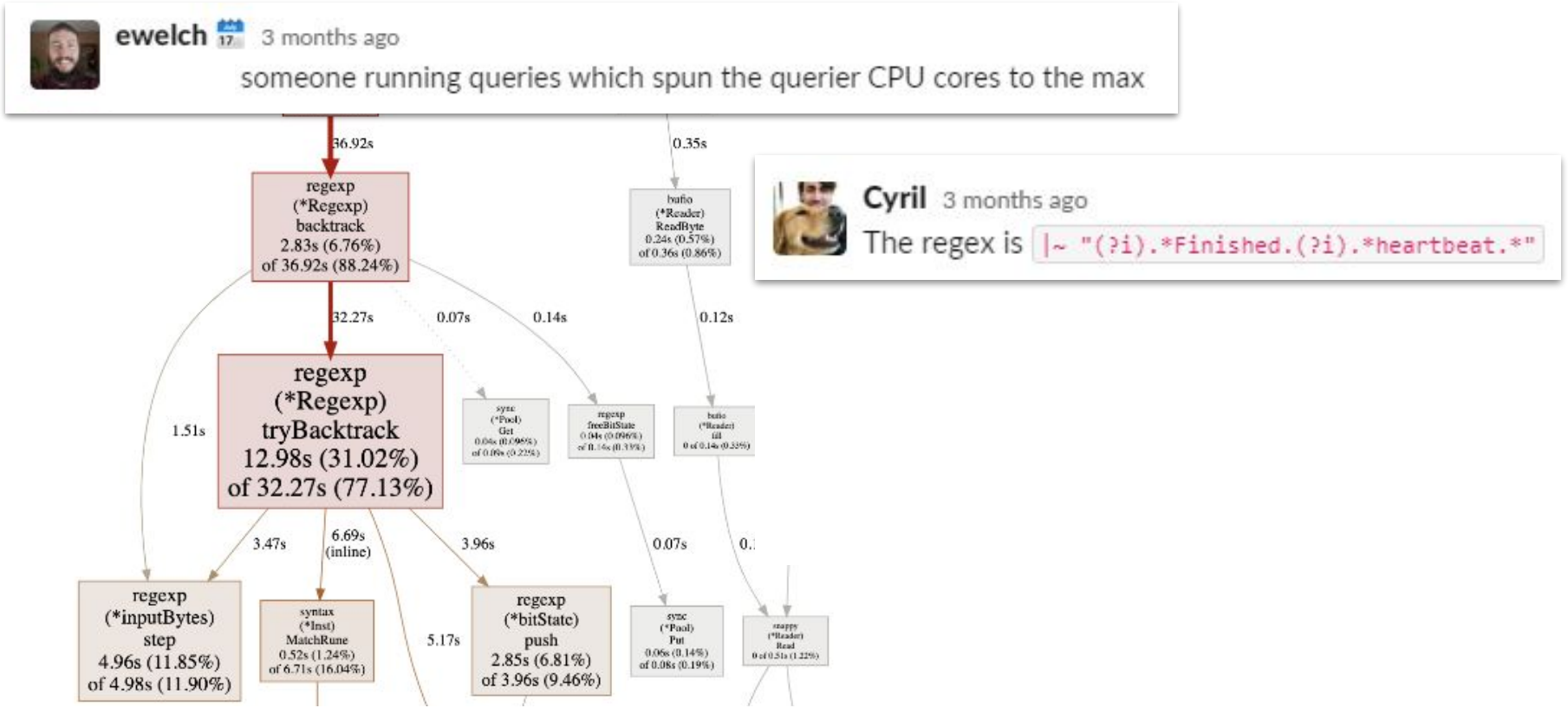


cortex

loki

Tempo

This is where the story begins...



"A **Regular Expression**, or Regexp, is used to express a search pattern in text" - *Wikipedia*

Example:

```
qui.*fox
```

Matches:

```
"The quick brown fox jumps
    over the lazy dog"
```

Basic syntax

```
.          any character
[xyz]      one of a set
xy         x followed by y
x|y        x or y
x*         zero or more x
x+         one or more x
x?         zero or one x
^          beginning of text
$          end of text
(re)       capturing group
(?i)       case-insensitive
```

Regular expressions are built out of simple blocks.

A lot of simple blocks...

kinds of single-character expressions	examples
any character, possibly including newline	`.`
character class	`[xyz]`
negated character class	`[^xyz]`
Perl character class (link)	`\d`
negated Perl character class	`\D`
ASCII character class (link)	`[[:alpha:]]`
negated ASCII character class	`[[:^alpha:]]`
Unicode character class (one-letter name)	`\pN`
Unicode character class	`\p{Greek}`
negated Unicode character class (1-letter)	`\PN`
negated Unicode character class	`\P{Greek}`

	Repetitions
`x*`	zero or more `x`, prefer more
`x+`	one or more `x`, prefer more
`x?`	zero or one `x`, prefer one
`x{n,m}`	`n` or `n` +1 or ... or `m` `x`, prefer more
`x{n,}`	`n` or more `x`, prefer more
`x{n}`	exactly `n` `x`
`x*?`	zero or more `x`, prefer fewer
`x+?`	one or more `x`, prefer fewer
`x??`	zero or one `x`, prefer zero
`x{n,m}?`	`n` or `n` +1 or ... or `m` `x`, prefer fewer
`x{n,}?`	`n` or more `x`, prefer fewer
`x{n}?`	exactly `n` `x`

	Character classes
`\d`	digits (≡ `[0-9]`)
`\D`	not digits (≡ `[^0-9]`)
`\s`	whitespace (≡ `[\t\n\f\r ]`)
`\S`	not whitespace (≡ `[^\t\n\f\r ]`)
`\w`	word characters (≡ `[0-9A-Za-z_]`)
`\W`	not word characters (≡ `[^0-9A-Za-z_]`)

	Composites
`xy`	`x` followed by `y`
`x|y`	`x` or `y` (prefer `x`)

	Empty strings
`^`	at beginning of text or line (`m`=true)
`$`	at end of text (like `\z` not `\Z`) or line (`m`=true)
`\A`	at beginning of text
`\b`	at ASCII word boundary (`\w` on one side and `\W`,
`\B`	not at ASCII word boundary
`\z`	at end of text

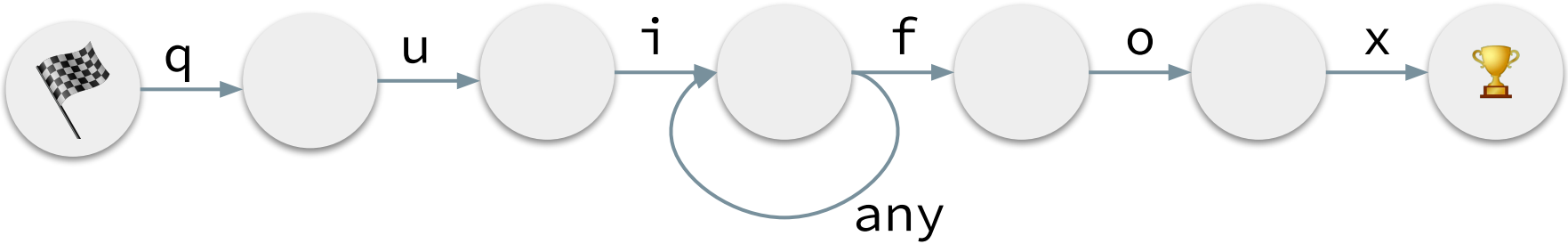
	Grouping
`(re)`	numbered capturing group (submatch)
`(?P<name>re)`	named & numbered capturing group (submatch)
`(?:re)`	non-capturing group
`(?flags)`	set flags within current group; non-capturing
`(?flags:re)`	set flags during re; non-capturing

	Flags
`i`	case-insensitive (default false)
`m`	multi-line mode: `^` and `$` match begin/end line (default false)
`s`	let `.` match `\n` (default false)
`U`	ungreedy: swap meaning of `x*` and `x*?`, `x+` and `x+?`, etc

https://github.com/google/re2/wiki/Syntax

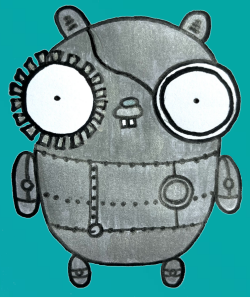
Regexp as state transitions

```
qui.*fox
```



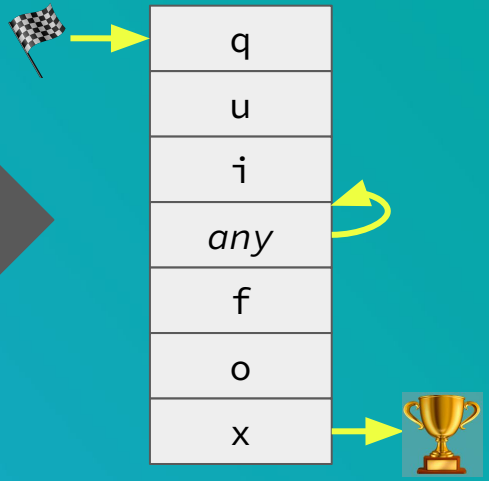
Nondeterministic
Finite
Automaton

Regexp Compilation in Go



qui.*fox

compile



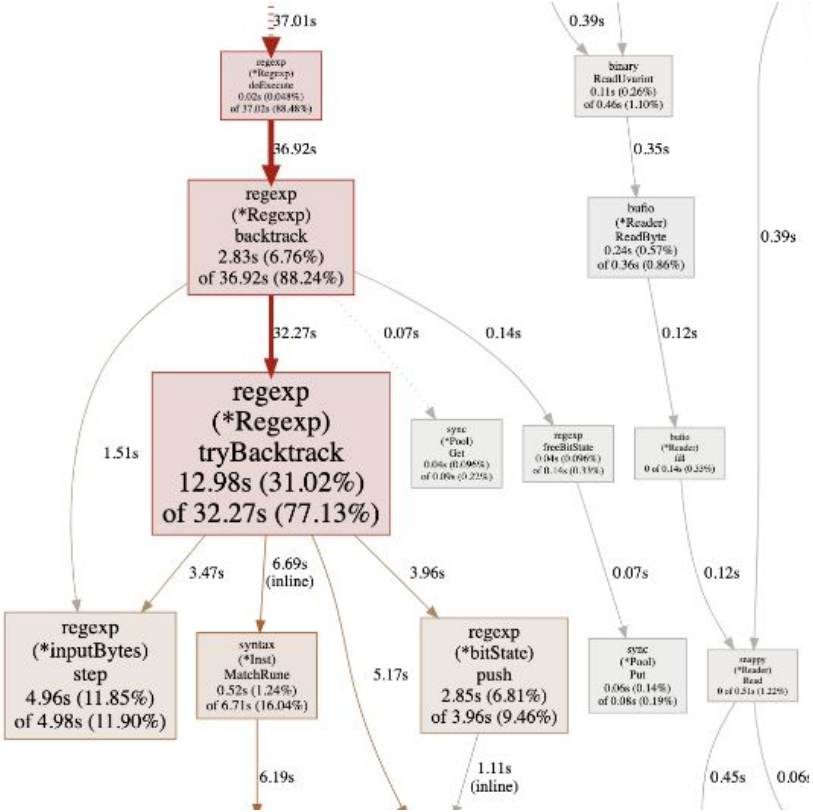
Inside the code

This is the data structure for a compiled regexp:

```
type Prog struct {
    Inst   []Inst
    Start  int
    NumCap int
}

type Inst struct {
    Op   InstOp
    Out  uint32
    Arg  uint32
    Rune []rune
}
```

```
type InstOp uint8

    InstAlt
    InstAltMatch
    InstCapture
    InstEmptyWidth
    InstMatch
    InstFail
    InstNop
    InstRune
    InstRune1
    InstRuneAny
    InstRuneAnyNotNL
```

Back to the profile



Benchmark

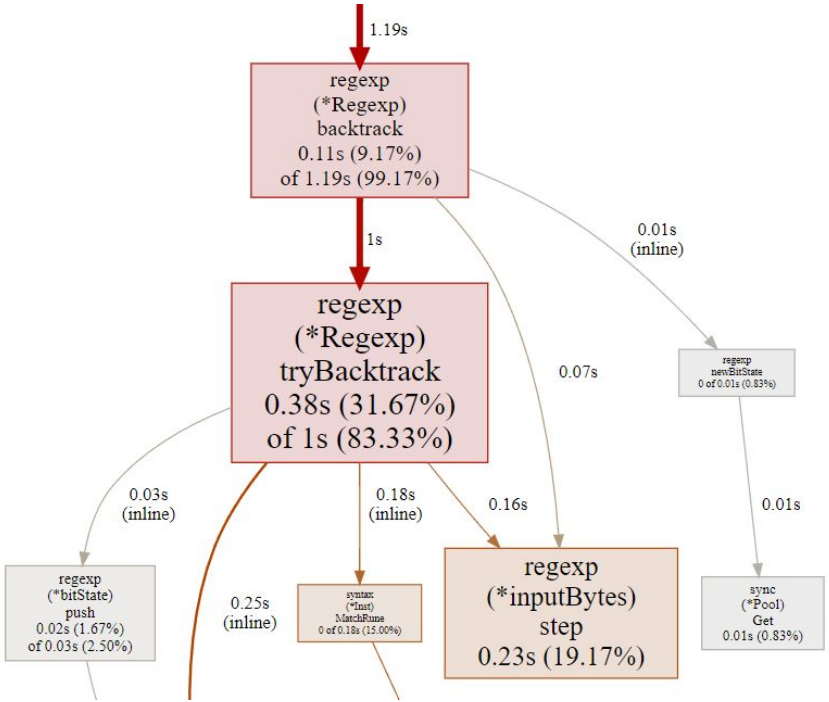
```
func BenchmarkRegexp(b *testing.B) {
    b.StopTimer()
    x := bytes.Repeat([]byte("x"), 50)
    re := MustCompile("(?i).*foo.*bar.*")
    b.StartTimer()
    for i := 0; i < b.N; i++ {
        re.Match(x)
    }
}

go test -run xxx -bench Regexp -cpuprofile cprof
```

Benchmark results

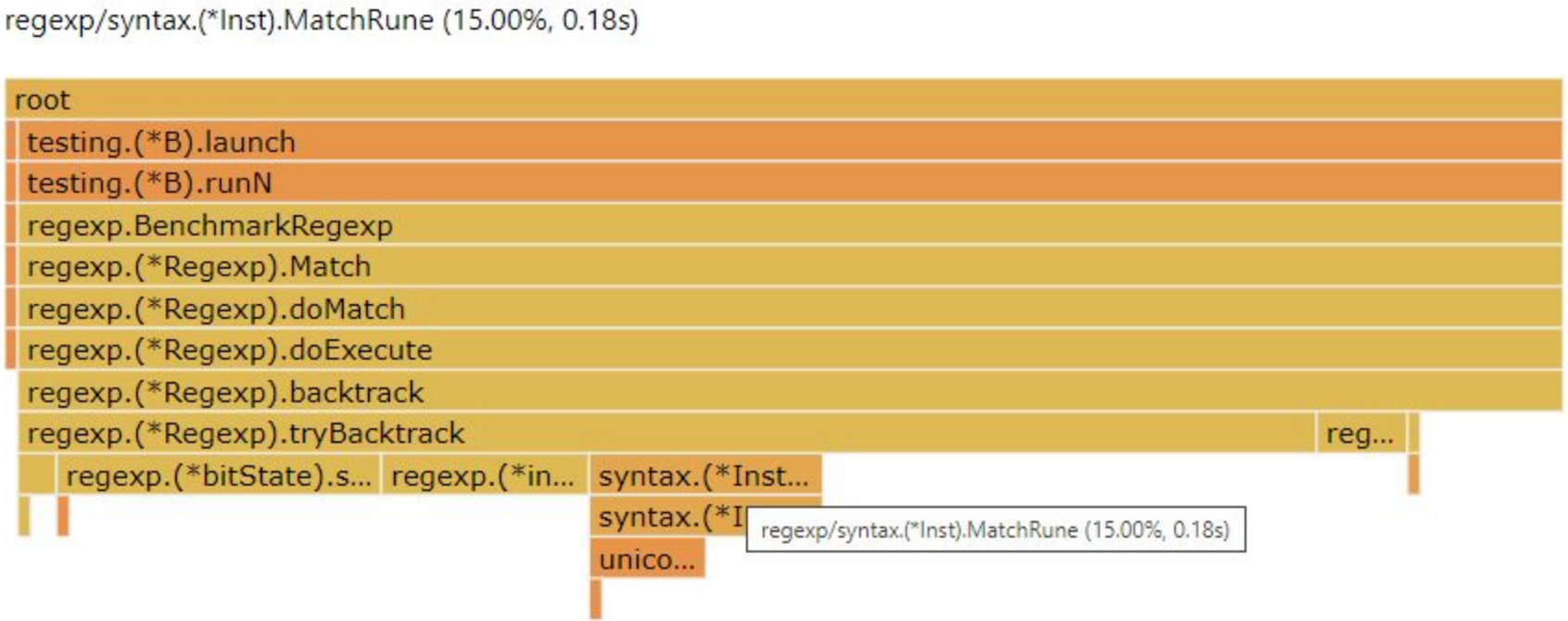
```
$ go test -run xxx -bench Regexp -cpuprofile cprof
goos: linux
goarch: amd64
pkg: github.com/grafana/regexp
cpu: Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
BenchmarkRegexp-4          283477              4123 ns/op
PASS
ok      regexp      1.415s
```

Profile of the benchmark

```
go tool pprof -http :8000 cprof
```



Profile of the benchmark (flame graph)



Benchmark

```
func BenchmarkRegexp(b *testing.B) {
   b.StopTimer()
   x := bytes.Repeat([]byte("x"), 50)
   re := MustCompile("(?i).*foo.*bar.*")
   b.StartTimer()
   for i := 0; i < b.N; i++ {
      re.Match(x)
   }
}
```

Benchmark regexp

```
                                "(?i).*foo.*bar.*"


4123 ns/op
```

Remove case-insensitive

```
".*foo.*bar.*"
```

```
3413 ns/op
```

Remove opening/closing wildcard

```
"foo.*bar"
```

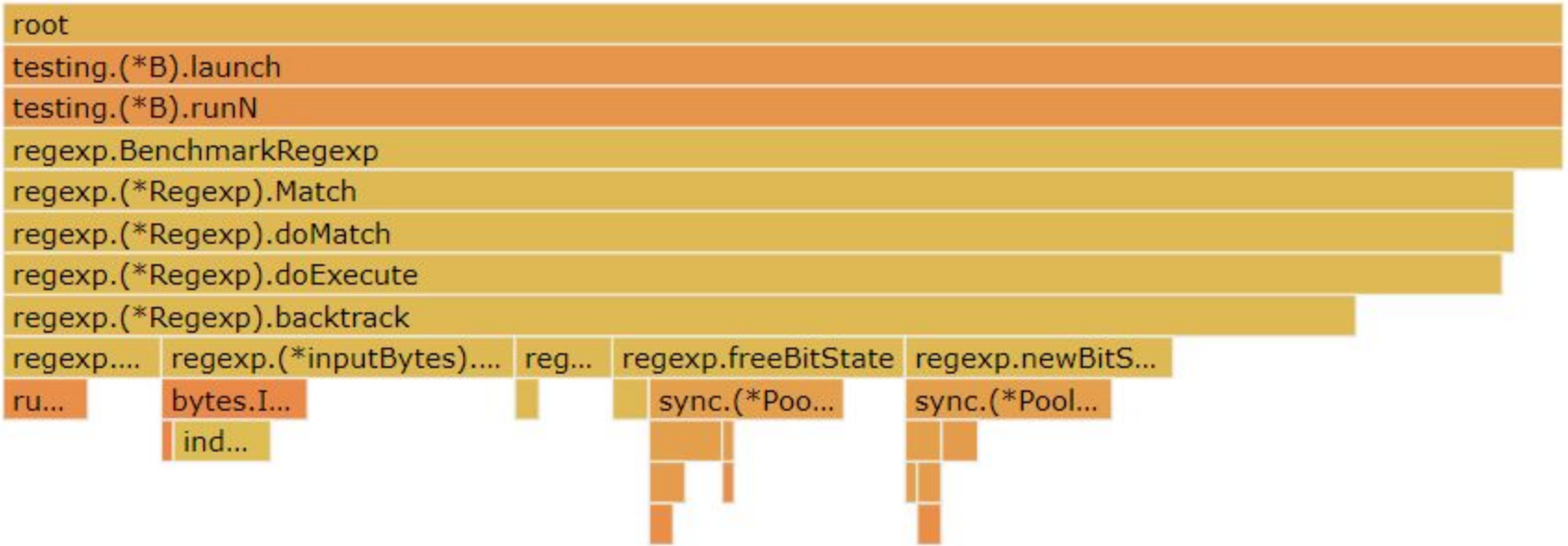
```
91.25 ns/op
```

Remove all wildcards

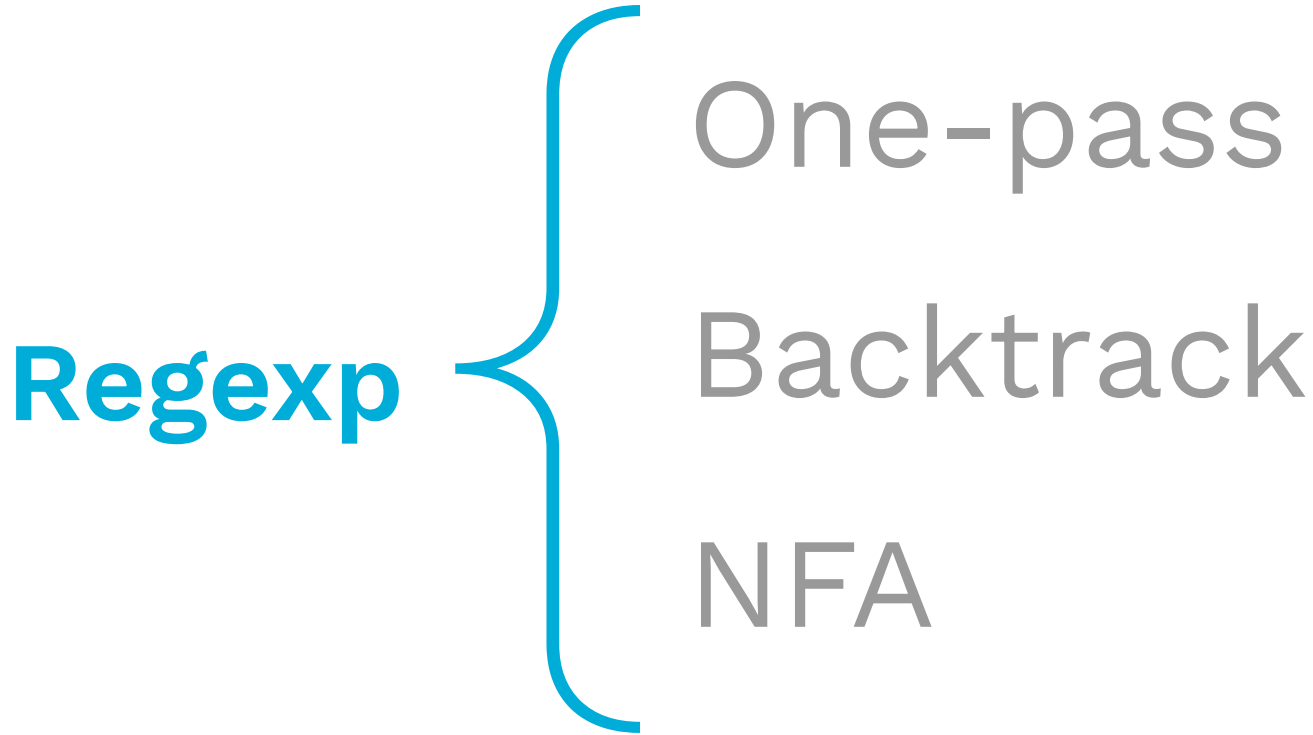
```
"foo"
```

```
90.85 ns/op
```

Profile of benchmark "foo"



Go has three regexp matchers

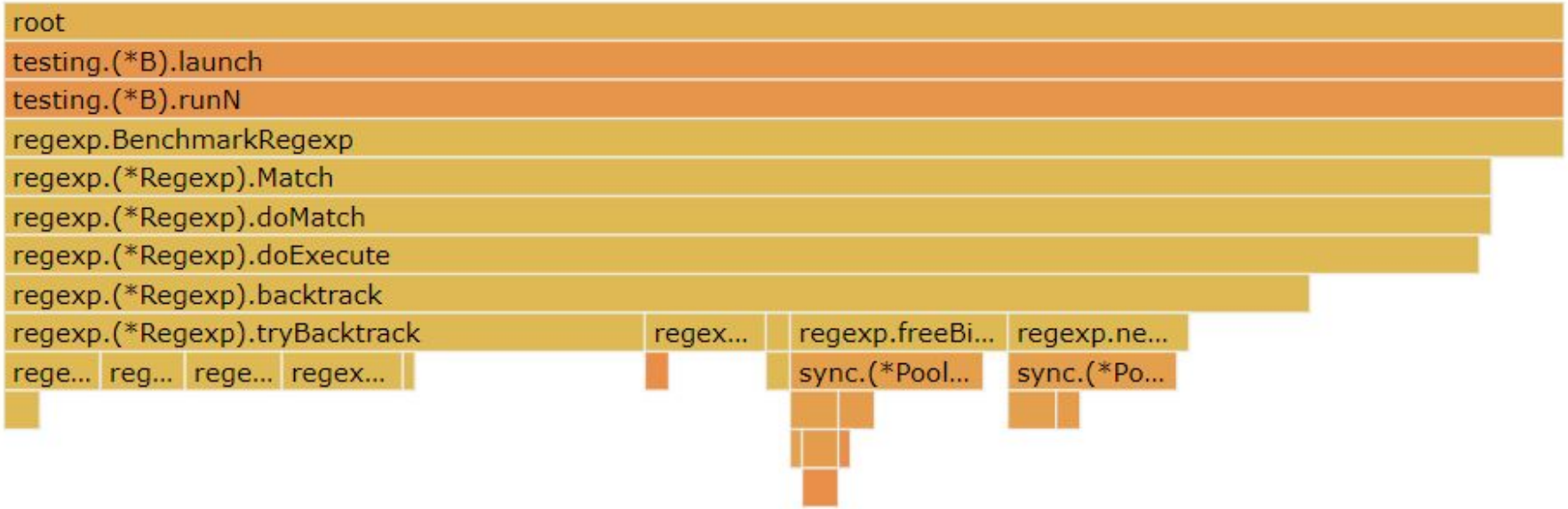


Try to force one-pass

```
"^foo"
```

```
115.1 ns/op
```

Profile of benchmark "^foo"



golang / go Public

Watch 3.5k | Fork 14k

Code | Issues 5k+ | Pull requests 225 | Discussions | Actions | Projects

regexp: allow patterns with no alternates to be one-pass #48748

Open **bboreham** wants to merge 3 commits into `golang:master` from `bboreham:more-onepass`

Conversation 8 | Commits 3 | Checks 0 | Files changed 2

bboreham commented on 2 Oct 2021 • edited — Contributor

Check whether a regex has any 'alt' instructions before rejecting it as one-pass. Previously `^abc` would run the backtrack matcher.

I tried to make the comment match what the code does now.

Updates #21463

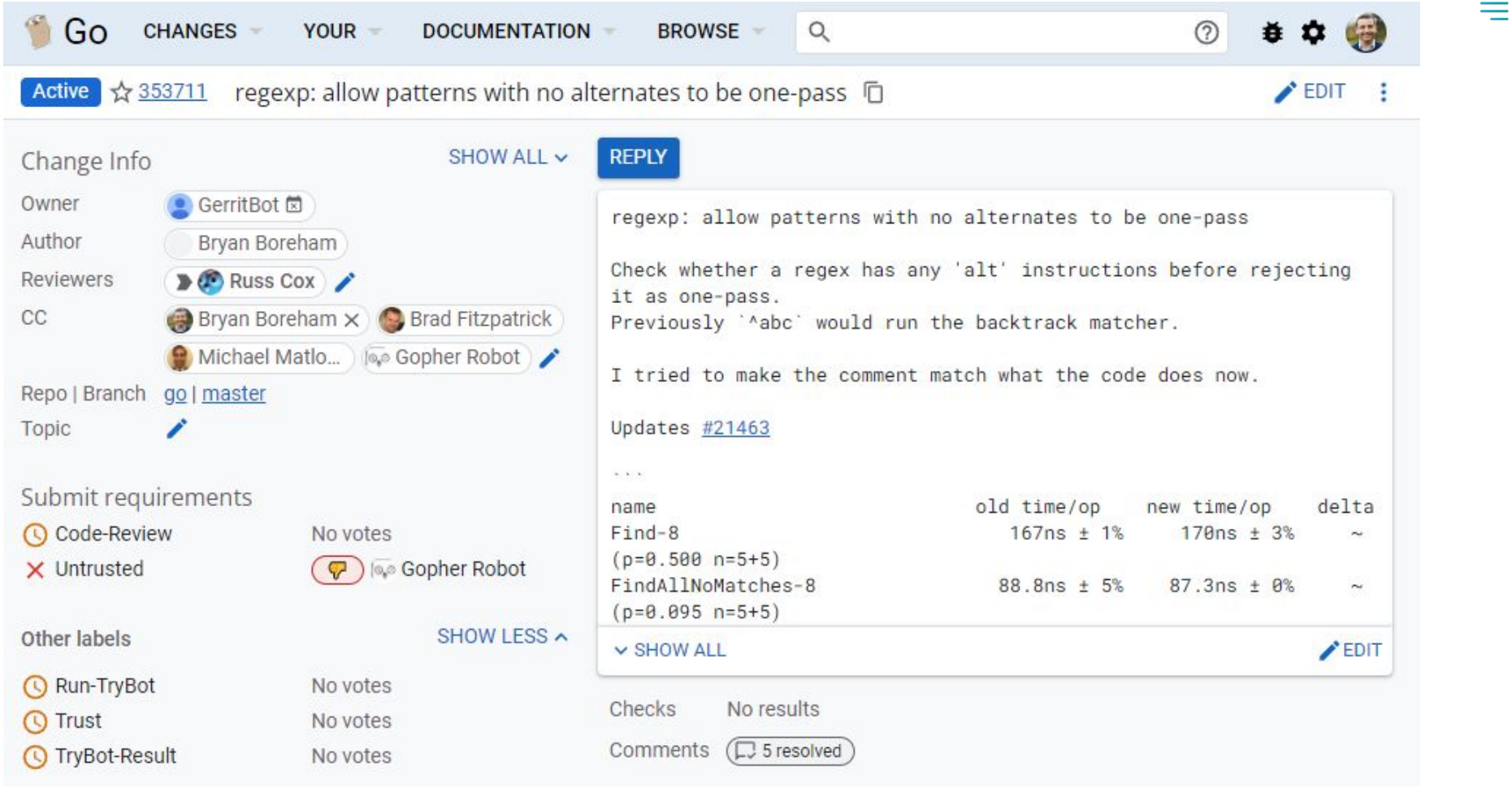
Reviewers

No reviews

Still in progress? Cc

Assignees

No one assigned


Go
CHANGES
YOUR
DOCUMENTATION
BROWSE
Active
353711
regexp: allow patterns with no alternates to be one-pass
EDIT
Change Info
SHOW ALL
REPLY
Owner
GerritBot
Author
Bryan Boreham
Reviewers
Russ Cox
CC
Bryan Boreham
Brad Fitzpatrick
Michael Matlo...
Gopher Robot
Repo | Branch
go | master
Topic
Submit requirements
Code-Review
No votes
Untrusted
Gopher Robot
Other labels
SHOW LESS
Run-TryBot
No votes
Trust
No votes
TryBot-Result
No votes
regexp: allow patterns with no alternates to be one-pass
Check whether a regex has any 'alt' instructions before rejecting it as one-pass.
Previously `^abc` would run the backtrack matcher.
I tried to make the comment match what the code does now.
Updates #21463
...
name old time/op new time/op delta
Find-8 167ns ± 1% 170ns ± 3% ~
(p=0.500 n=5+5)
FindAllNoMatches-8 88.8ns ± 5% 87.3ns ± 0% ~
(p=0.095 n=5+5)
SHOW ALL
EDIT
Checks
No results
Comments
5 resolved

```
469  469  »       // onepass regexp is anchored
470  470  »       if prog.Inst[prog.Start].Op != syntax.InstEmptyWidth ||
471  471  »       »       syntax.EmptyOp(prog.Inst[prog.Start].Arg)&syntax.EmptyBeginText != syntax.EmptyBegin
     Text {
472  472  »       »       return nil
473  473  »       }
474       »       // every instruction leading to InstMatch must be EmptyEndText
     474  »       hasAlt := false
     475  »       for _, inst := range prog.Inst {
     476  »       »       if inst.Op == syntax.InstAlt || inst.Op == syntax.InstAltMatch {
     477  »       »       »       hasAlt = true
     478  »       »       »       break
     479  »       »       }
     480  »       }
     481  »       // If we have alternates, every instruction leading to InstMatch must be EmptyEndText.
     482  »       // Also, any match on empty text must be $.
475  483  »       for _, inst := range prog.Inst {
476  484  »       »       opOut := prog.Inst[inst.Out].Op
477  485  »       »       switch inst.Op {
478  486  »       »       default:
479       »       »       »       if opOut == syntax.InstMatch {
     487  »       »       »       if opOut == syntax.InstMatch && hasAlt {
480  488  »       »       »       »       return nil
481  489  »       »       »       }
482  490  »       »       case syntax.InstAlt, syntax.InstAltMatch:
```

Now this case is much faster

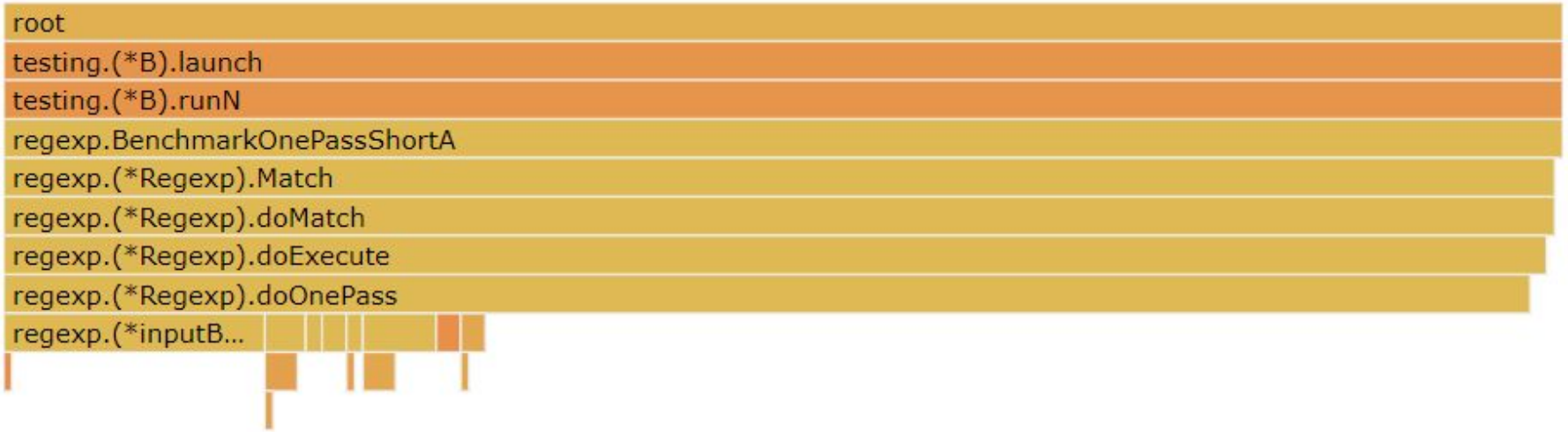
```
"^foo"
```

```
78.09 ns/op
```

Different Benchmark

```
func BenchmarkOnePassShortA(b *testing.B) {
    b.StopTimer()
    x := []byte("abcddddddeeeededd")
    re := MustCompile("^.bc(d|e)*$")
    b.StartTimer()
    for i := 0; i < b.N; i++ {
        re.Match(x)
    }
}

730.6 ns/op
```

Profile of `BenchmarkOnePassShortA`



Source view

```
regexp.(*Regexp).doOnePass in regexp/exec.go
  440ms    720ms (flat, cum) 98.63% of Total
     .        .    398: func (re *Regexp) doOnePass …

     .        .    444:   for {
  170ms    170ms    445:     inst = re.onepass.Inst[pc]
   40ms     40ms    446:     pc = int(inst.Out)
   40ms     40ms    447:     switch inst.Op {
     .        .    448:     default:

```

Simple change

355789 regexp: avoid copying each instruction executed ☑ src/regexp/exec.go File 3 of 3 Prev

Base browse → Patchset 1 browse DOWNLOAD SHOW BLAME EDIT Diff view:

```
440  440  »       »       r1, width1 = i.step(pos + width)
441  441  »       »       flag = i.context(pos)
442  442  »       »       pc = int(re.prefixEnd)
443  443  »       }
444  444  »       for {
445       »       »       inst = re.onepass.Inst[pc]
     445  »       »       inst = &re.onepass.Inst[pc]
446  446  »       »       pc = int(inst.Out)
44                                                  {
```

730.6 -> 633.0 ns/op

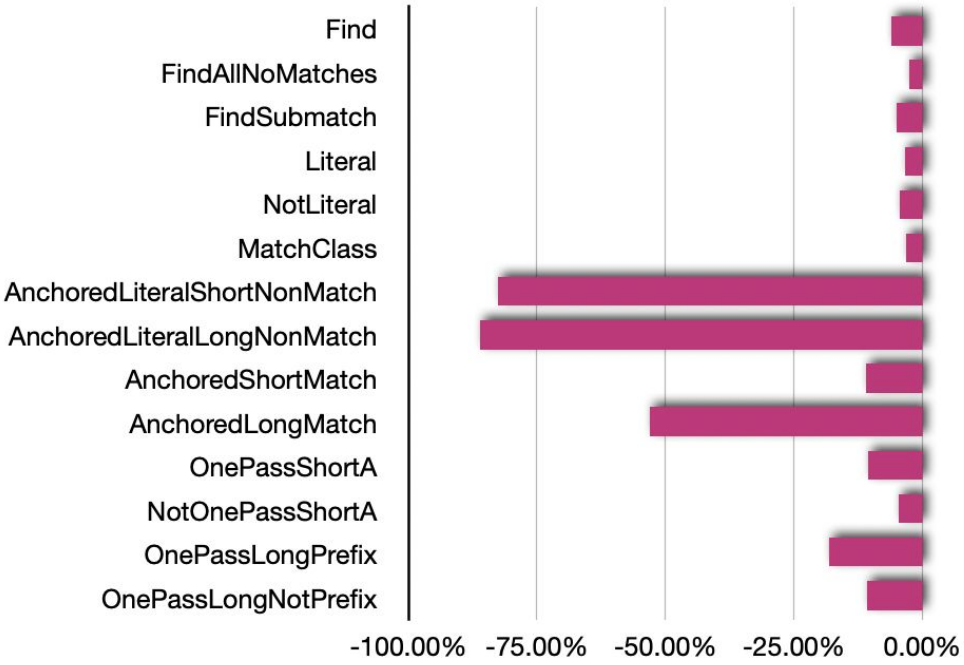
Brad Fitzpatrick

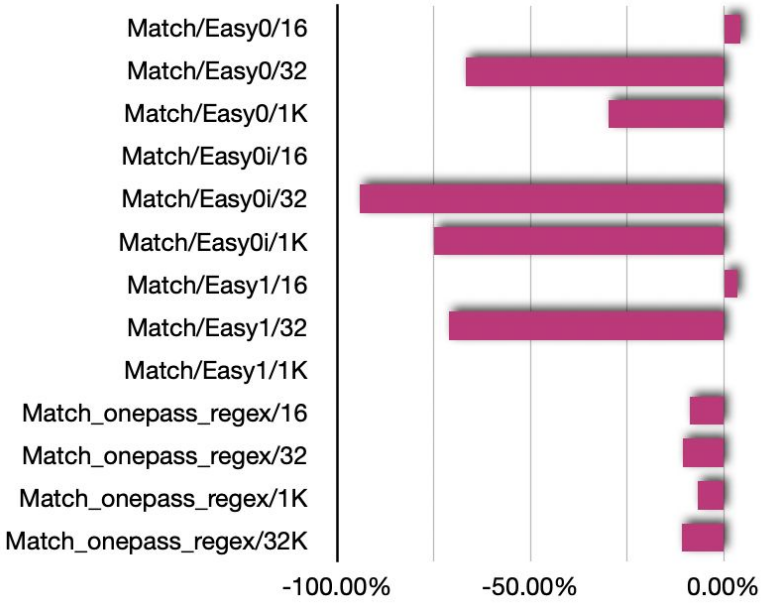
Wow, nice find!

List of change lists

- 353711: allow patterns with no alternates to be one-pass
- 354909: speed up onepass prefix check
- 355789: avoid copying each instruction executed
- 358756: handle prefix string with fold-case
- 377294: allow prefix string anchored at beginning

Overall Score





Links

https://github.com/grafana/regexp/tree/speedup

https://github.com/google/re2/wiki/Syntax

GopherBot automaton  by Mary Pollard.
Other gophers and the original Gopher design by Renee French.