Just-in-Time Compiling Ruby Regexps on TruffleRuby

Benoit Daloze and Josef Haider
Presenters

Benoit Daloze
TruffleRuby lead
Twitter: @eregontp
GitHub: @eregon

Josef Haider
TRegex creator and maintainer
GitHub: @djoooooe
TruffleRuby

- A high-performance Ruby implementation
- Uses the GraalVM. JIT Compiler
- Targets full compatibility with CRuby 2.7, including C extensions
- GitHub: oracle/truffleruby, Twitter: @TruffleRuby, website: graalvm.org/ruby
Background: Regexp Engines in TruffleRuby

- CRuby uses Onigmo (Oniguruma), backtracking regexp engine supporting 30 encodings
- TruffleRuby initially used Joni, which is a port of Onigmo to Java by JRuby developers
- Similar performance to Onigmo in CRuby
- TruffleRuby already JIT compiles "small languages of Ruby" like `array.pack("C*")` and "%f % pi", but not yet Regexps
- It would be great if TruffleRuby would also run Regexps faster!
A Wild TRegex Appeared!
• Regular expression engine based on state machines, more specifically "deterministic finite automata" (DFA)
• states have transitions to successor states
• every transition has a set of accepted symbols/characters
Regular Expressions and Finite State Machines

- Regular expressions used to be perfectly representable as state machines, but were extended later
- Basic concepts can still be mapped to state machines directly
- Concatenation: /ab/

Automaton model of /ab/
Regular Expressions and Finite State Machines

- **Disjunction:** \(/ab|ac/\)

(a) NFA model of \(/ab|ac/\)

(b) DFA model of \(/ab|ac/\)
Regular Expressions and Finite State Machines

- Quantifiers: /a\*b+/
Regular Expressions and Finite State Machines

- Capture groups: annotated transitions.

Automaton model of /a (bc | d) /
What is supported?

- Concatenation “ab”
- Disjunction “|”
- Infinite Quantifiers “*”, “+”
- Capture Groups “()”, “(?<name>)”
- Character Classes “[]”, “\p{ }”
- Counted Quantifiers “?”, “{n, m}” (partially)
- Anchors “^”, “$”, “\A”, “\Z”, “\b”, “\B”
- Lookahead Assertions “(?=)”
- Lookbehind Assertions “(?<=)” (partially)
What is not supported yet?

- Back-References "\1, \k<name>" in the Regexp (not in replacement strings: #gsub)
- Negative Lookahead "(?!)"
- Negative Lookbehind "(?<!)"
- Recursive Subexpression Calls "\g<name>" like "(?<sqbr>[\g<sqbr>\*])"
- Possessive Quantifiers "*+", "++", "?+", "{n,m}+"
- Atomic Groups "(?>)
- Conditionals "(?<group>)"
- Absent Expressions "(?~)"
Just-In-Time-Compiling regular expressions

```python
@ExplodeLoop(MERGE_EXPLODE)
def execute(input, index = 0):
    result = -1
    ip = 0
    outer:
        loop do
            current_state = STATES[ip]
            result = index if current_state.final_state?
            return result if index >= input.size
            c = input[index]
            index += 1
            current_state.each_transition do |transition|
                if transition.match?(c)
                    ip = transition.target_ip
                    goto :outer
                end
            end
        end
    return result
    end
end
```
Just-In-Time-Compiling regular expressions

```python
def execute(input, index = 0):  # /a+(b|c)/
    state0:
        return -1 if index >= input.size
        c = input[index]
        index += 1
        if c == 'a' then goto :state1
        else goto :state0
    end
    state1:
        return -1 if index >= input.size
        c = input[index]
        index += 1
        if c == 'a' then goto :state1
        elsif c == 'b' || c == 'c' then goto :state2
        else goto :state0
    end
    state2:
        return index
    end
```

Copyright © 2021, Oracle and/or its affiliates
Performance Results

We use the `benchmark-ips` gem to measure peak performance and compare:

- TruffleRuby+TRegex on GraalVM JVM CE
- TruffleRuby+Joni on GraalVM JVM CE
- CRuby 2.7
Micro-Benchmarks for "abc".match?(Regexp)

Speedup relative to CRuby

- /def/
- /abc/
- /.
- /[a-z]/
- /[0-9]/

Graph showing speedup relative to CRuby 2.7 for different regex patterns using TruffleRuby+Joni and TruffleRuby+TRegex.

Copyright © 2021, Oracle and/or its affiliates
Larger Regexp Benchmarks

- **liquid parse**: `Liquid::Template.new.parse(cart_template)`, so the parsing part of the Liquid template language, and that parser uses Regexps heavily

- **browser sniffer**: from [Shopify/browser_sniffer](https://github.com/Shopify/browser_sniffer), a gem to detect which browser, OS, versions, etc are used from the user agent using Regexps

- **regex redux (no IO)**: a benchmark from the Computer Language Benchmarks Game which reads 50MB of DNA/RNA sequences and transforms them using regexps (gsub!, scan)

- **syslog**: a benchmark parsing a single log line according to the BSD syslog Protocol (RFC 3164)
Larger Regexp Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CRuby 2.7</th>
<th>TruffleRuby+Joni</th>
<th>TruffleRuby+TRegex</th>
</tr>
</thead>
<tbody>
<tr>
<td>liquid parse</td>
<td>3</td>
<td>1.5</td>
<td>1</td>
</tr>
<tr>
<td>browser sniffer</td>
<td>1.5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>regex redux</td>
<td>1</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>syslog</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>

Copyright © 2021, Oracle and/or its affiliates
ReDoS and Catastrophic Backtracking


- TRegex always matches in linear time, no risk of ReDoS with TRegex!

- When falling back to Joni / backtracking, TruffleRuby can emit warnings
  
  `--warn-slow-regex`:
  `file.rb: warning: Regexp /(?!...)/ requires backtracking and might not match in linear time`
Atomic Groups

- Atomic groups cannot be easily supported by finite-state machines regex engines.

- Most usages of atomic groups seem workarounds for excessive backtracking. In that case, it is safe to ignore such groups for TRegex.

- Atomic groups can also be used for semantics (seems rare):
  
  ```ruby
  /"(?>.*)/ =~ "Quote" # => nil
  ```

- Approach: be optimistic and assume atomic groups are used for performance, not for semantics. TruffleRuby has an option to disable this behavior.
Conclusion

- Using finite-state machines for Regexp matching is faster than backtracking and safer

- TruffleRuby and TRegex can compile Ruby Regexp to machine code and inline them together with Ruby code

- On the presented benchmarks, TruffleRuby+TRegex is faster than CRuby by 24x-41x for regexp micro-benchmarks and 2.3x-9x for larger regexp benchmarks

- TruffleRuby can warn when Regexp are at risk of catastrophic backtracking (ReDoS)
Acknowledgments

• Jirka Maršík (@jirkamarsik) for adding support for the many features of Ruby Regexp in TRegex, and most of the integration of TRegex in TruffleRuby

• Duncan MacGregor (@aardvark179) for various optimizations related to Regexp matching (StringScanner, gsub, accessing $~ in the C API, etc)

• Kevin Menard (@nirvdrum) for further optimizations, notably to enable splitting and inlining of regexps