UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Extending the PCRE Library with Static Backtracking Based Just-in-Time Compilation Support

Zoltán Herczeg
19/02/14

Outline

- Motivation
- PCRE-JIT compiler
- Static backtracking
- Results
  - Raw backtracking performance
  - PCRE-JIT engine

Motivation

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

History of Regular Expressions

- Kleene: regular sets in 1950s
- Thompson: Deterministic Finite Automaton (DFA) in 1960s
  - A pattern was constructed from character sets, and few operators: star, vertical bar
- Purpose of 'real' regular expressions
  - Defining a pattern which matches a set of words
  - /c|ab*/ matches c, a, ab, abb, abbb, …

Regular Expressions Today

- Henry Spencer: backtracking engine in 1980s (open source)
  - Nondeterministic Finite Automaton (NFA)
- Improved later by PERL developers
  - Command based pattern matching language
  - Backward compatible
  - Closer to scripting languages
- Context sensitive decisions
  - Reduces complexity

Matches only when fox is outside of <> brackets:

PERL: /<((?:[^<>]*+(?:<(?1)|))++>?)(*SKIP)(*F)|fox/

Matches: <<fox>fox>fox

No match: <fox<<>fox>fox>

No match: <<fox>fox

Multibyte character sequences:

PERL: /a(?>\r\n|\r|\n){2}b/

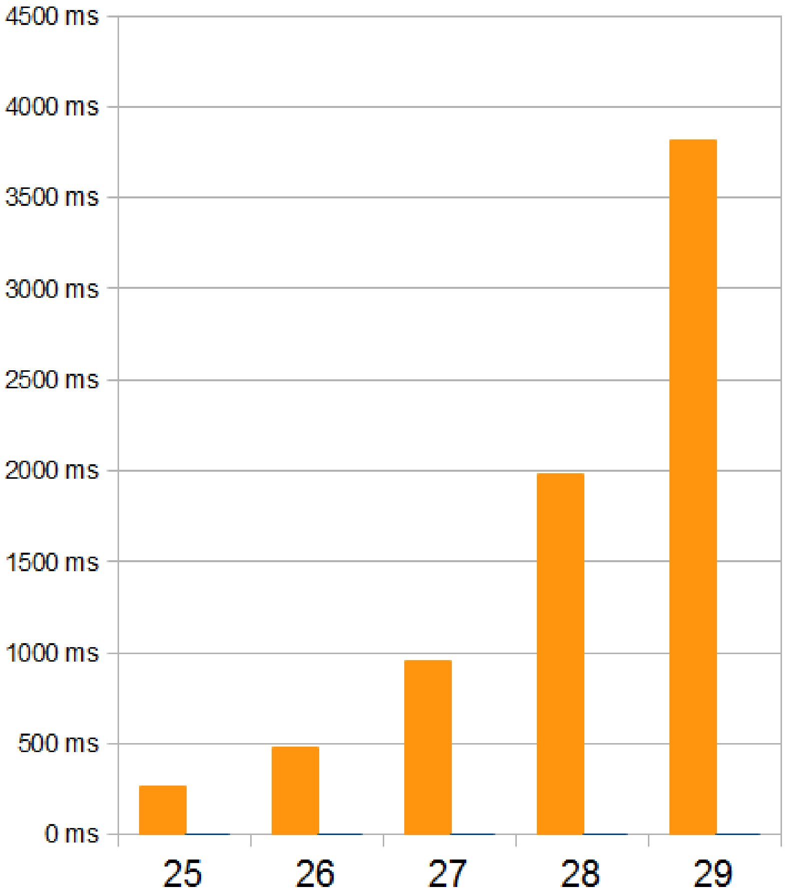
DFA: /a(?:\r(?:\n(?:\r\n|\r|\n)|\r\n)|\n\r\n?)b/

Matches: a\r\r\nb

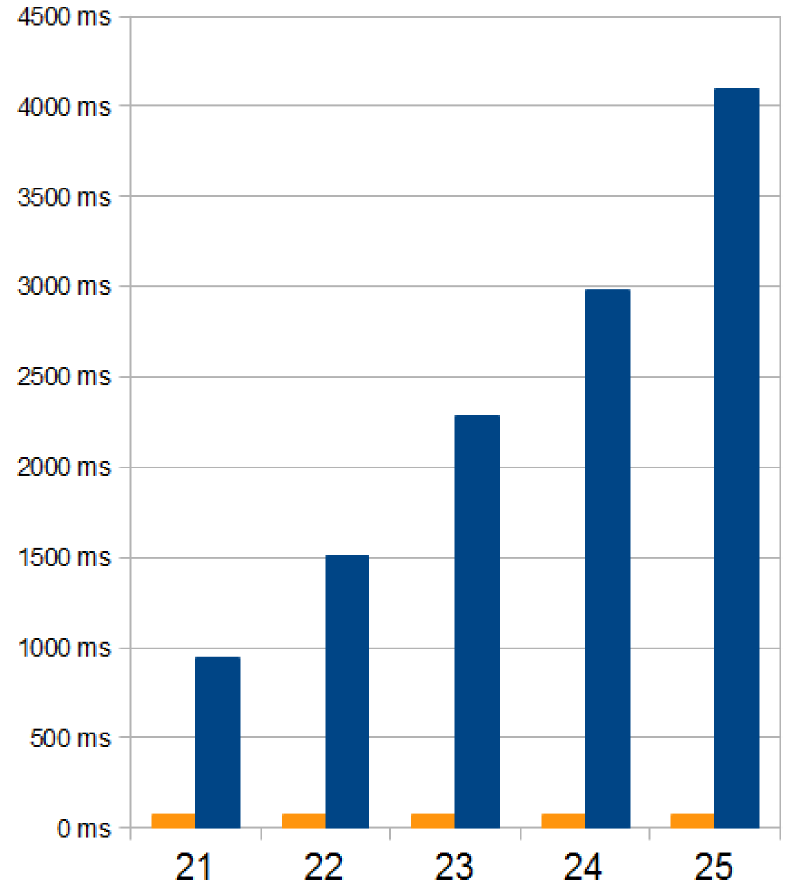
Matches: a\n\rb

No match: a\r\nb

/(x|x){n}y/



/a[^b]{n}abc/



PCRE RE2

Summary

- New approach is needed to accelerate these “regular expressions”
  - DFA is not enough anymore
- Command based languages can be efficiently accelerated by Just-in-time (JIT) compilation
  - 2009 Irregexp
  - 2009 YARR (Yet Another Regex Runtime)
  - 2011 PCRE-JIT

PCRE-JIT compiler

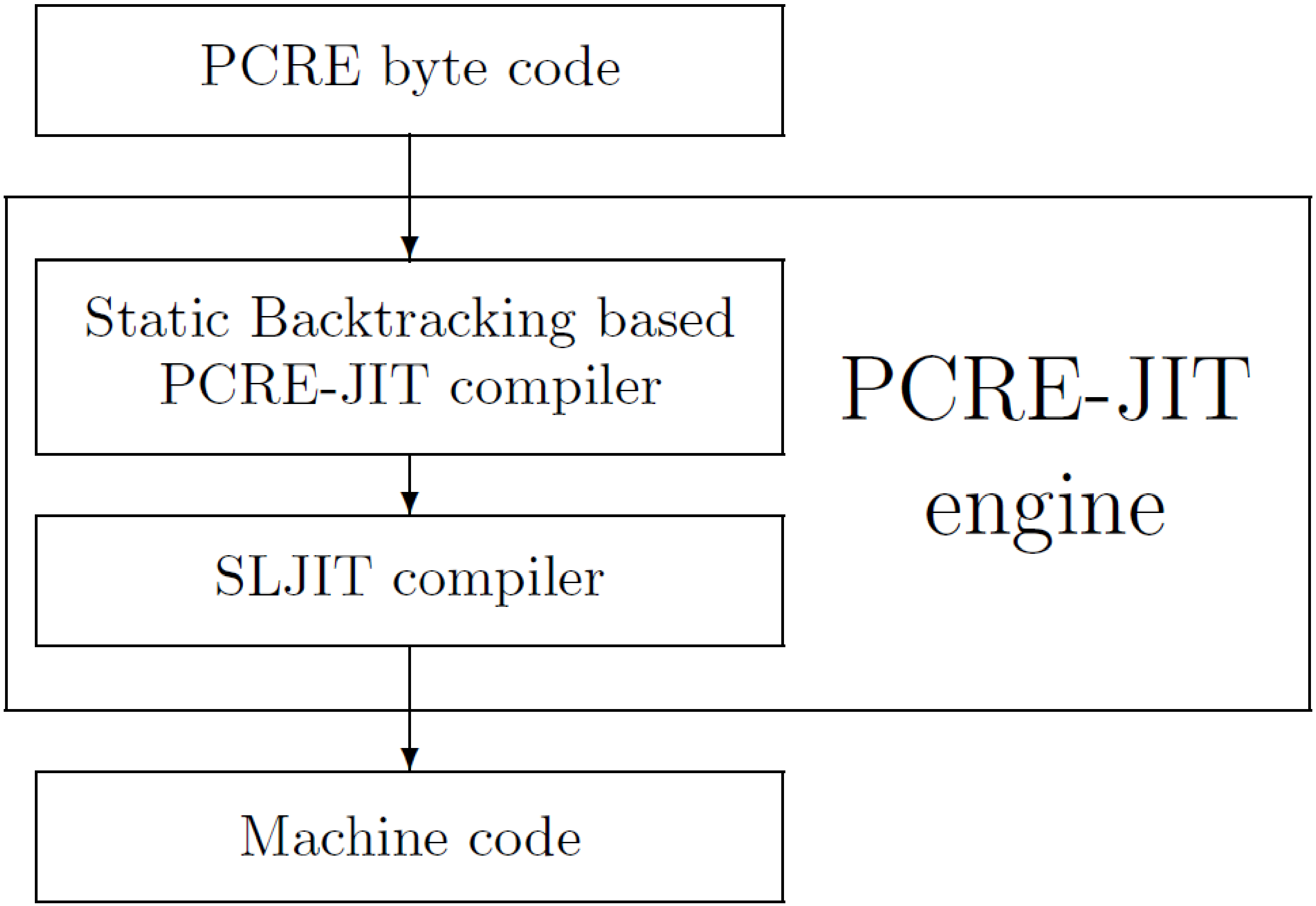
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

PCRE Overview

- Perl Compatible Regular Expressions
  - Standalone C library, which supports PERL style patterns
- Started by Philip Hazel in 1997
- PCRE-JIT supports all of its features
  - Comparison of regular expression engines in Wikipedia: Only PCRE supports all 19 categories

http://en.wikipedia.org/wiki/Comparison_of_regular_expression_engines

Main Components of PCRE-JIT



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

SLJIT Compiler

- Low-level assembly like language
  - Translated to machine code
- Platform independent
  - LIR is designed by uniting the common features of CPUs
- Supported architectures
  - x86-32/64, ARMv5/7/Thumb2, PowerPC-32/64, MIPS-32, SPARC-32
  - ARM-64 is close to done

Evolution of Backtracking Algorithms

Interpreting (?:B)+ Pattern

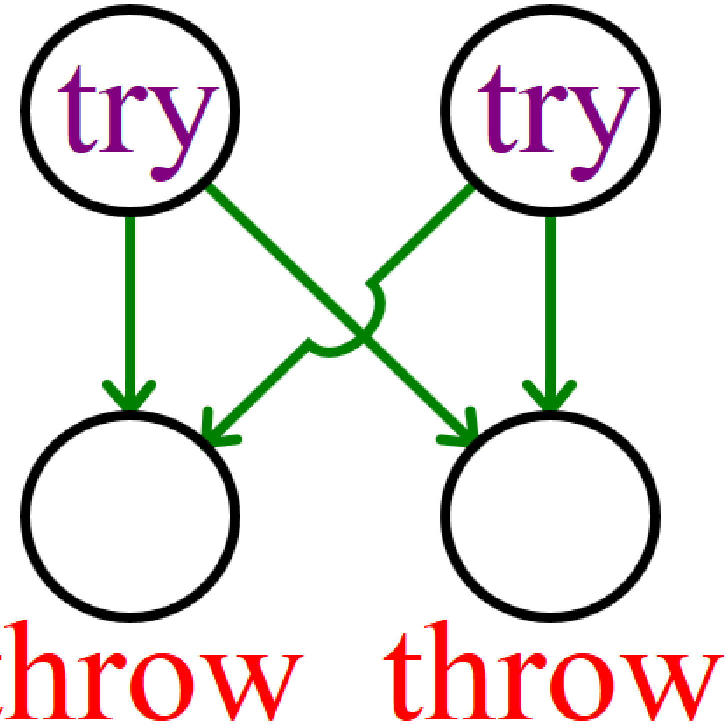
```
boolean recursive_match() {
  // Infinite loop for tail merging.
  while (TRUE) {
    switch (*current_opcode) {
    // ...

    case GREEDY_PLUS_CLOSE_BRACKET:
      if (recursive_match() == SUCCESS)
        return SUCCESS
      // Tail merge.
      current_opcode += opcode_size(
                GREEDY_PLUS_CLOSE_BRACKET)
      // Back to the while loop.
      break;

    // ...
    }
  }
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

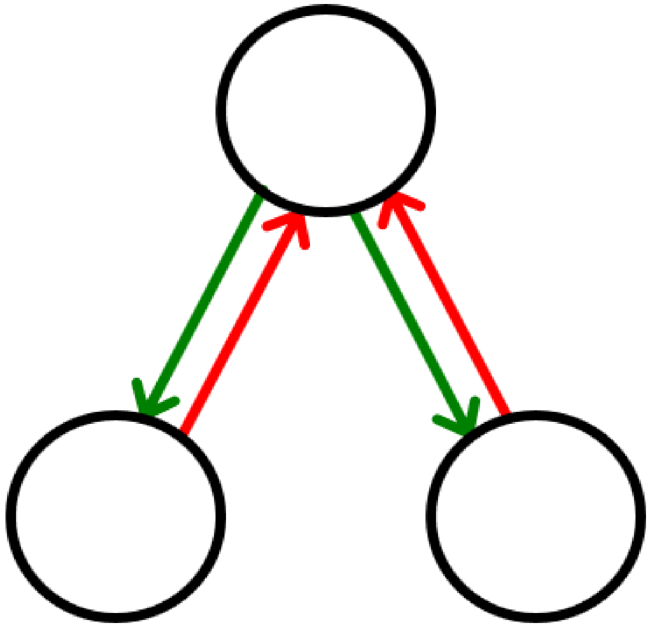
Static Backtracking

- JIT compilers replaced recursive function calls to try-catch blocks
  - Unknown destination when backtracking
- Our new approach: **Static Backtracking**
- Generating code from Abstract Syntax Tree (AST)
  - Each node has exactly one parent node, and it is known at compile time

Nondeterministic Finite Automaton



Abstract Syntax Tree



Continue matching: green lines
Perform backtracking: red lines / texts

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

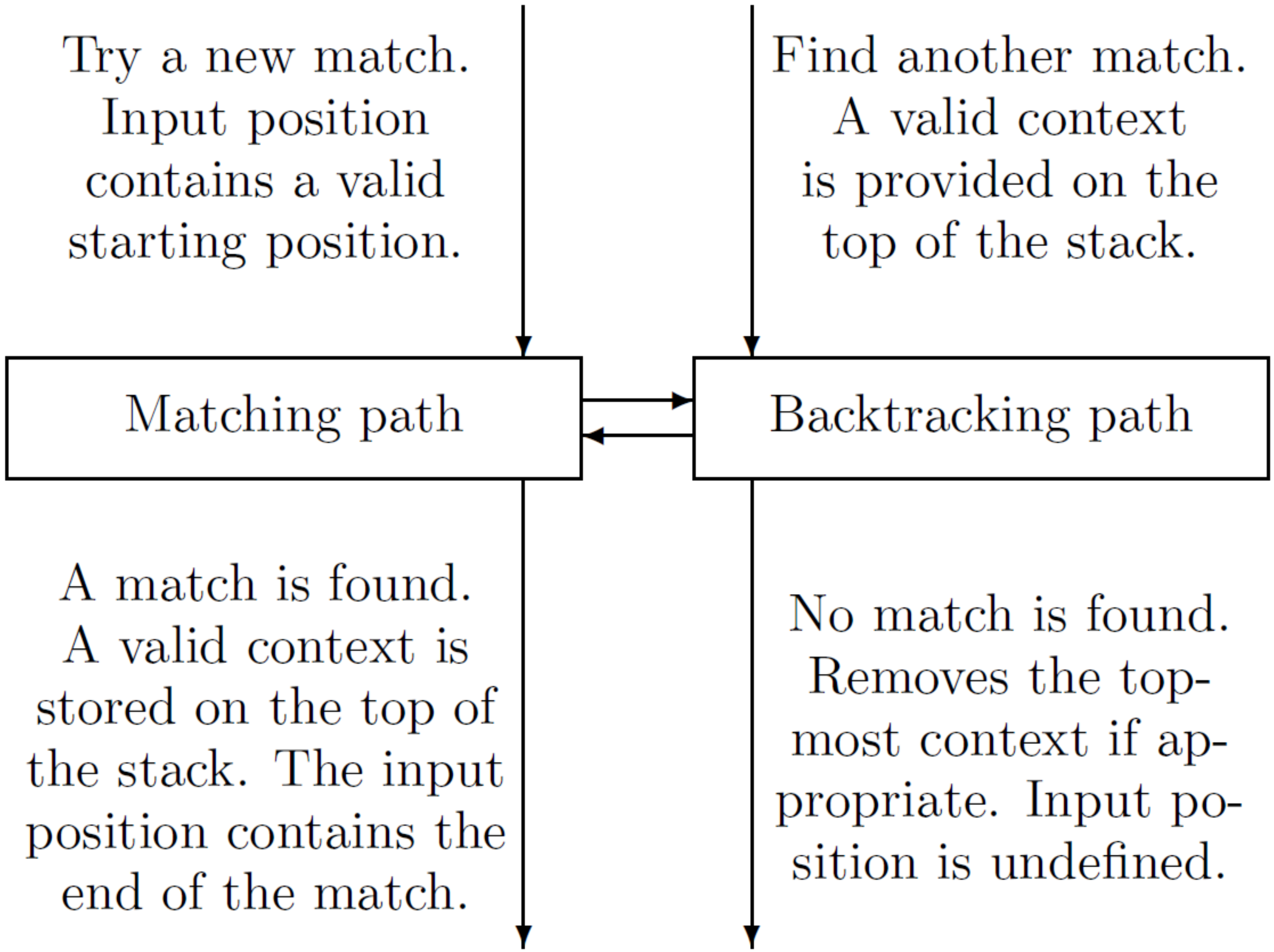
Static Backtracking (2)

- Not all NFAs are supported
  - Enough for PERL compatible patterns: they are always compiled to an AST first
- Context dependent backtracking
- No need to set-up catch handlers
- Direct, conditional jumps
  - Indirect jumps are usually not conditional: two jumps are needed for checking a condition and backtrack on fail

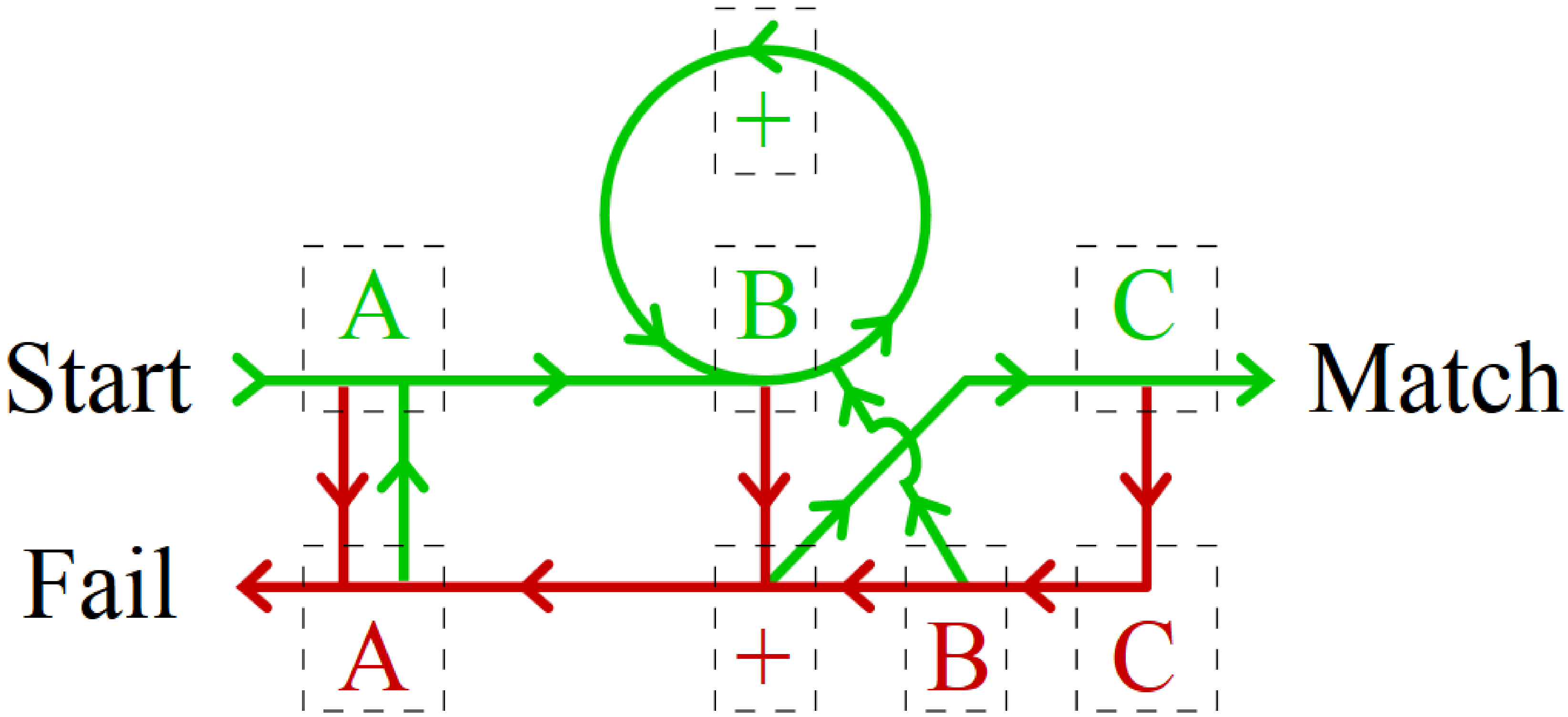
Overview of Building Blocks

- Two code paths are generated from each AST node
  - Matching path: action
  - Backtracking path: reverse action
- These two code paths form a single function
  - With two entry, two exit points
  - Optimization: these points can represent passing or returning a boolean value

Basic Building Block



Control Flow of /A(?:B)+C/



Matching path color: green

Backtracking path color: red

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Results

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Raw Backtracking Performance

- Measured on pathological cases
  - Exponential runtime, large amount of backt.
- Compared to Irregexp (in V8 JavaScript engine)
  - Traditional (indirect jump) based backtracking
  - Highly optimized engine
- These patterns cannot be optimized by backtracking elimination techniques
  - E.g: expected character check

Pathological Cases

	Pattern	Input	Irregexp (ms)	Irregexp ratio	PCRE JIT (ms)	PCRE JIT ratio
1	$/a?^{26}c\vert/$	$a^{26}bc$	777.0	2.20	$\underline{352.5}$	$\underline{1.00}$
2	$/a+^{14}c\vert/$	$a^{26}b^{14}c$	221.6	1.21	$\underline{182.8}$	$\underline{1.00}$
3	$/(?:a+)+c\vert/$	$a^{23}bc$	101.2	1.44	$\underline{70.5}$	$\underline{1.00}$
4	$/(?:(?:a)+)+c\vert/$	$a^{23}bc$	102.8	1.20	$\underline{85.9}$	$\underline{1.00}$
5	$/(?:(?:aa)+)+c\vert/$	$a^{46}bc$	99.5	1.18	$\underline{84.2}$	$\underline{1.00}$
6	$/((a)+)+c\vert/$	$a^{23}bc$	220.1	1.70	$\underline{129.2}$	$\underline{1.00}$
7	$/(?:(?:(?:aa)+)+)+c\vert/$	$a^{30}bc$	217.7	2.14	$\underline{101.7}$	$\underline{1.00}$
8	$/(((aa)+)+)+c\vert/$	$a^{28}bc$	364.6	2.16	$\underline{168.6}$	$\underline{1.00}$
9	$/(?:a\vert a)^{26}c\vert/$	$a^{26}bc$	$\underline{110.5}$	$\underline{1.00}$	400.7	3.63
10	$/(?:aa\vert a)^{24}c\vert/$	$a^{48}bc$	194.2	1.93	$\underline{100.4}$	$\underline{1.00}$
11	$/(aa\vert a)^{24}c\vert/$	$a^{48}bc$	179.4	1.15	$\underline{155.6}$	$\underline{1.00}$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Overall Speedup

- Benchmark set provided by SNORT Intrusion Detection System (IDS)
  - >1000 HTTP content filtering patterns
- Compared to PCRE Interp. , and Irregexp
  - All engine optimizations are enabled (including backtracking eliminations)
- Backtracking is not a rare event
  - About 46% of matching attempts are failed in the PCRE interpreter

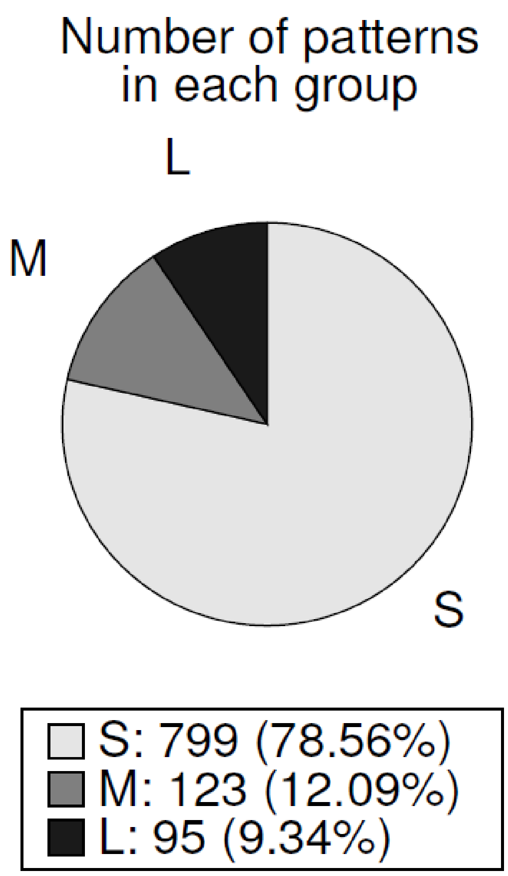
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

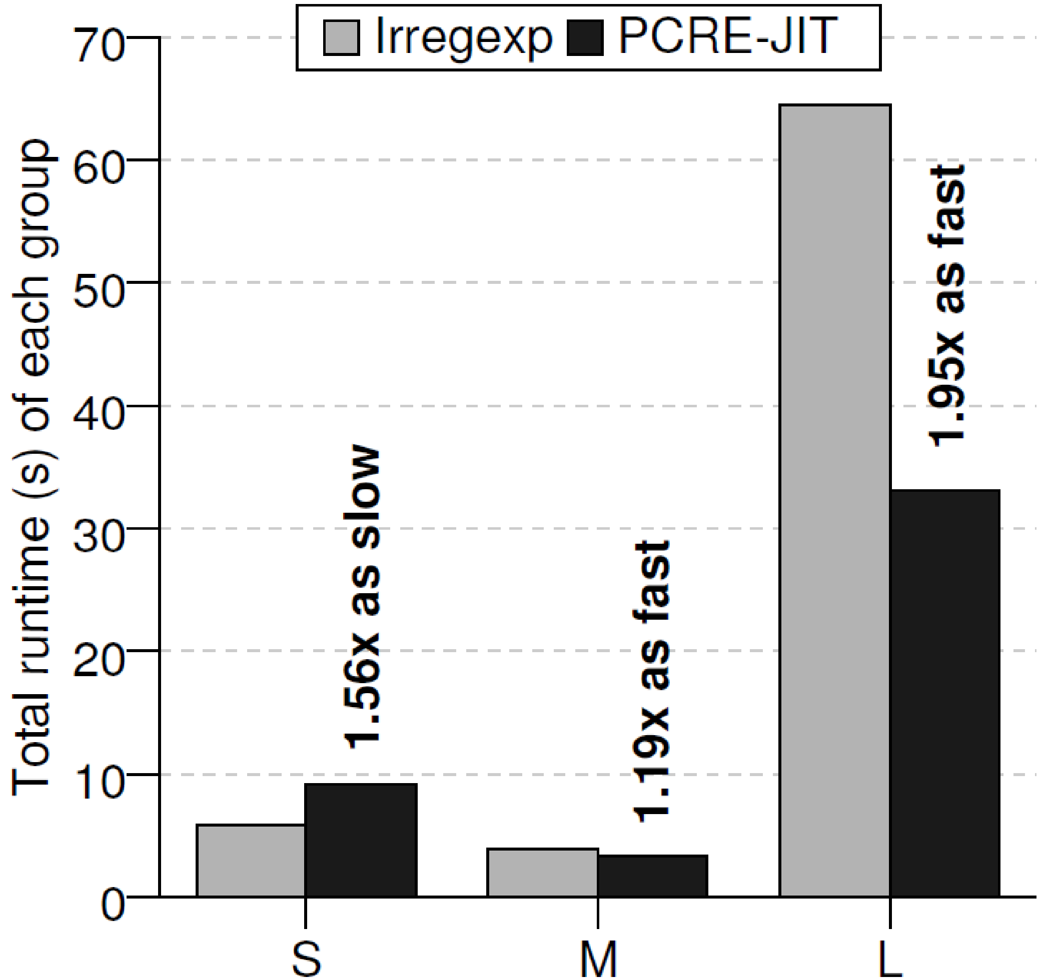
PCRE-JIT vs PCRE Interpreter

Target CPU	Average speedup (x as fast)	< 3.0 x as fast		$3.0 - \infty$ x as fast	
		% of patterns	% of total runtime	% of patterns	% of total runtime
x86/32	6.84	84.99%	3.66%	15.01%	96.34%
x86/64	5.55	82.92%	3.80%	17.08%	96.20%
ARM-V7/32	6.76	82.24%	3.71%	17.76%	96.29%
ARM-THUMB2/32	7.24	84.59%	3.78%	15.41%	96.22%
PowerPC/32	5.48	88.13%	6.47%	11.87%	93.53%
PowerPC/64	5.55	88.42%	6.38%	11.58%	93.62%
SPARC/32	5.85	83.61%	4.08%	16.39%	95.92%
MIPS/32	7.59	81.35%	3.27%	18.65%	96.73%
Average	6.36	84.53%	4.39%	15.47%	95.61%
Std. dev.	0.79	2.42%	1.19%	2.42%	1.19%

Overall: PCRE-JIT is 1.63x faster than Irregexp

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

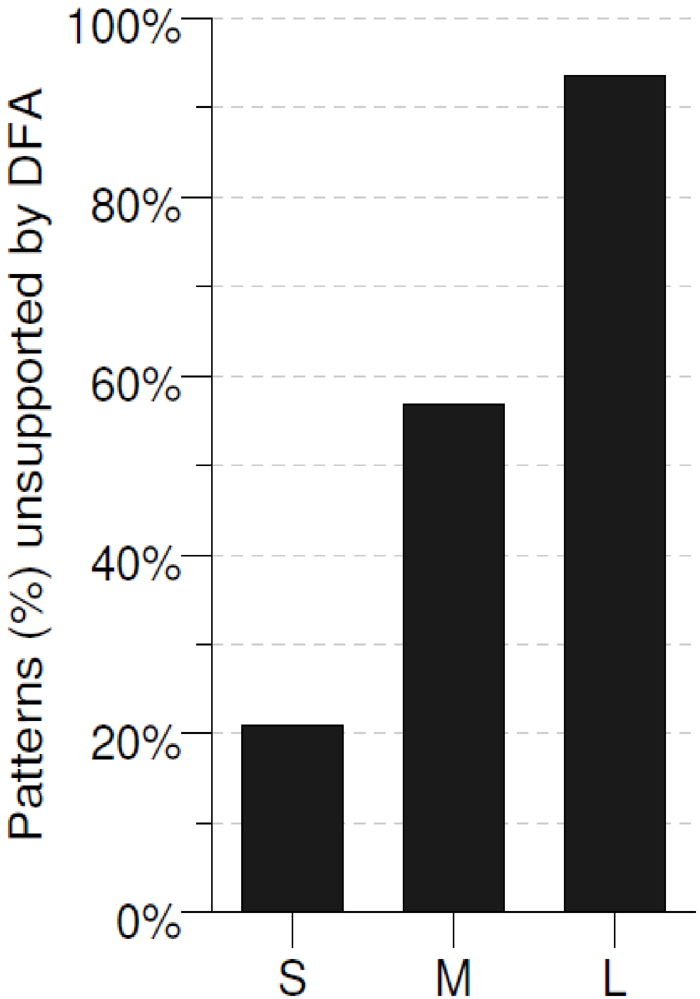




S: Patterns with short runtime (0-20 ms)
M: Patterns with medium runtime (20-200 ms)
L: Patterns with long runtime (200-2000ms)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Patterns(%) Unsupported by DFA



- Most patterns in group L are not supported by DFA based engines!
- DFA is not a viable option to accelerate these long running patterns

Conclusion and Future Work

Conclusion

- Regular expressions are changed
  - Better to optimize them using JIT compiling
- Traditional NFA based backtracking can be improved by AST based backtracking
  - Context dependent backtracking
- Results (SNORT benchmark set)
  - 6.36x faster than interpreter
  - 1.63x faster than Irregexp

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Future Work

- Further improve the JIT compiler
  - Integrating optimizations used by other engines
  - A simplified Boyer-Moore string search was already implemented
- Support more CPUs (in SLJIT)
  - MIPS-64, SPARC-64

PCRE-JIT Is Production Ready

- NGINX Web Server
- Qt Framework (since 5.0)
- Suricata Intrusion Detection System
- Apache ModSecurity Firewall
- GNU grep (with -P option)
- HAProxy Load Balancer
- Sigil E-book Editor

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
UNIVERSITY OF SZEGED
Department of Software Engineering

Thanks for listening

Questions?