Exploring Regular Expression Comprehension

Carl Chapman*, Peipei Wang, Kathryn T. Stolee

Sandia National Laboratories Albuquerque*, North Carolina State University

carlallenchapman@gmail.com, pwang7@ncsu.edu, ktstolee@ncsu.edu

Nov 1st, 2017

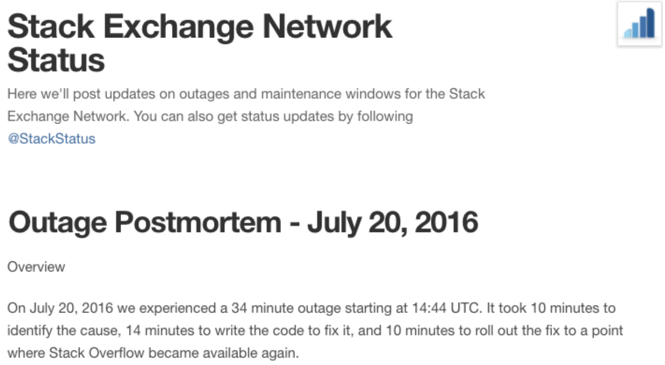
Why should we use regular expressions?

- A succinct way to express pattern matching.
- Less code and flexible.

Why should we NOT use regular expressions?

- Hard to write the correct regular expression.
- Complicated to understand.
- Difficult to test and debug.

Example of Bad Regex



Stack Exchange Network Status

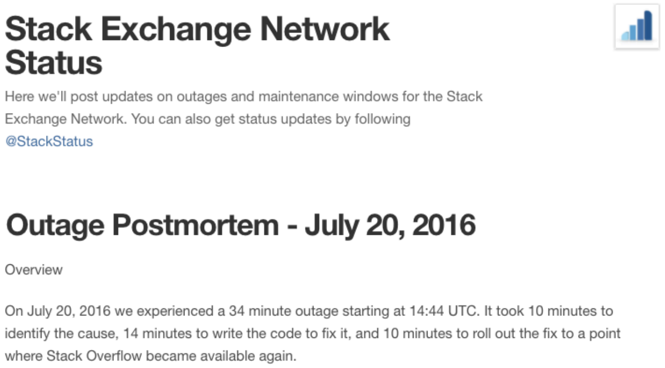
Here we'll post updates on outages and maintenance windows for the Stack Exchange Network. You can also get status updates by following @StackStatus

Outage Postmortem - July 20, 2016

Overview

On July 20, 2016 we experienced a 34 minute outage starting at 14:44 UTC. It took 10 minutes to identify the cause, 14 minutes to write the code to fix it, and 10 minutes to roll out the fix to a point where Stack Overflow became available again.

Example of Bad Regex



Stack Exchange Network Status

Here we'll post updates on outages and maintenance windows for the Stack Exchange Network. You can also get status updates by following @StackStatus

Outage Postmortem - July 20, 2016

Overview

On July 20, 2016 we experienced a 34 minute outage starting at 14:44 UTC. It took 10 minutes to identify the cause, 14 minutes to write the code to fix it, and 10 minutes to roll out the fix to a point where Stack Overflow became available again.

Regex

```
^[\s\u200c]+|[\s\u200c]+$
```

State of the Art

- Tools for visual debugging (e.g., Regex101, Regexr)
- Tools for graphical regular expression (e.g., Rex, Brics)
- Tools for automatic generation of regex and strings(e.g., Rex, ReLIE)

Running Example

Which regular expression should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`

Running Example

Which regular expression should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`

Difference: How to express Double-Bounded repetition of digits?

Running Example

Which regular expression should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`

Difference: How to express Double-Bounded repetition of digits?

- A: repetition bounds using {}
- B: digits can appear or not appear using ?
- C: explicit repetitions using OR

Regex Representation

Regex representation: syntactic expression

- matching a digit (Custom Character Class): `[0123456789]`, `(0|1|2|3|4|5|6|7|8|9)`, `[0-9]`, `[\u30-\u39]`, `\d`, ...
- matching at least one digit (Lower-Bounded): `[0-9]+`, `[0-9][0-9]*`, `[0-9]{1,}`, `[0-9][0-9]{0,}`, `\d+`, ...
- matching at most three digits and at least one digit (Double-Bounded): `[1-9][0-9]{0,2}`, `[1-9][0-9]?[0-9]?`, `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`, `[1-9]\d{0,2}`, ...

Regex Representation

Regex representation: syntactic expression

- matching a digit (Custom Character Class): `[0123456789]`, `(0|1|2|3|4|5|6|7|8|9)`, `[0-9]`, `[\u30-\u39]`, `\d`, . . .
- matching at least one digit (Lower-Bounded): `[0-9]+`, `[0-9][0-9]*`, `[0-9]{1,}`, `[0-9][0-9]{0,}`, `\d+`, . . .
- matching at most three digits and at least one digit (Double-Bounded): `[1-9][0-9]{0,2}`, `[1-9][0-9]?[0-9]?`, `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`, `[1-9]\d{0,2}`, . . .

Regex Representation

Regex representation: syntactic expression

- matching a digit (Custom Character Class): [0123456789], (0|1|2|3|4|5|6|7|8|9), [0-9], [\u30-\u39], \d, ...
- matching at least one digit (Lower-Bounded): [0-9]+, [0-9][0-9]*, [0-9]{1,}, [0-9][0-9]{0,}, \d+, ...
- matching at most three digits and at least one digit (Double-Bounded): [1-9][0-9]{0,2}, [1-9][0-9]?[0-9]?, [1-9]|[1-9][0-9]|[1-9][0-9][0-9], [1-9]\d{0,2}, ...

Regex Representation

Regex representation: syntactic expression

- matching a digit (Custom Character Class): `[0123456789]`, `(0|1|2|3|4|5|6|7|8|9)`, `[0-9]`, `[\u30-\u39]`, `\d`, ...
- matching at least one digit (Lower-Bounded): `[0-9]+`, `[0-9][0-9]*`, `[0-9]{1,}`, `[0-9][0-9]{0,}`, `\d+`, ...
- matching at most three digits and at least one digit (Double-Bounded): `[1-9][0-9]{0,2}`, `[1-9][0-9]?[0-9]?`, `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`, `[1-9]\d{0,2}`, ...

Research Goals

Explore regex comprehension

1. Which regex representations are most understandable? (understandability study)
2. Which regex representations are used most frequently? (community study)
3. Which regex representations should we use? (desirability analysis)

Regex Comparison Prerequisite

Equivalence class: a group of **behaviorally equivalent** regexes

Regex Comparison Prerequisite

Equivalence class: a group of **behaviorally equivalent** regexes

- Match the same set of character strings

Regex Comparison Prerequisite

Equivalence class: a group of **behaviorally equivalent** regexes

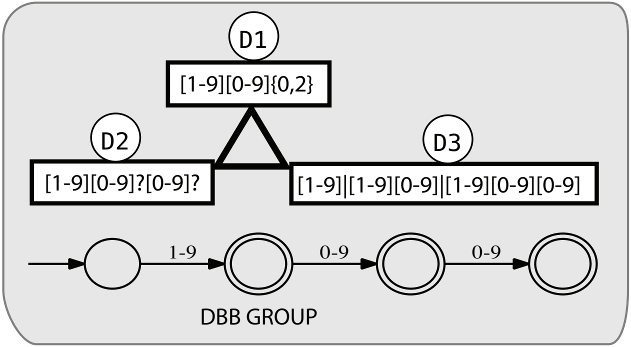
- Match the same set of character strings
- Different regex representations

Regex Comparison Prerequisite

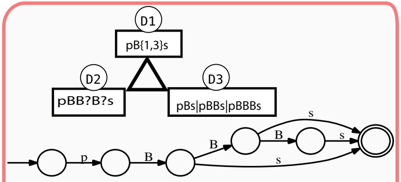
Equivalence class: a group of **behaviorally equivalent** regexes

- Match the same set of character strings
- Different regex representations
- Equivalent DFAs (Deterministic Finite Automaton)

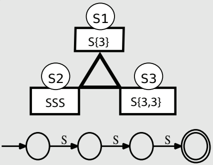
Double-Bounded Group of Equivalence Classes



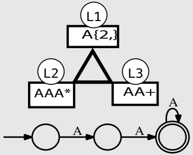
Five Equivalence Classes & 18 Regex Representations



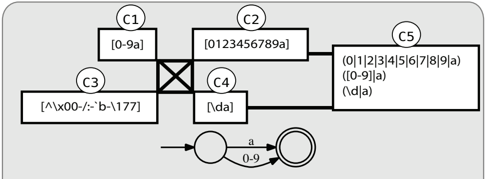
DBB GROUP
(using the abstract `pB{1,3}s' where B is any pattern),
p and s are any (possibly empty) prefix, suffix



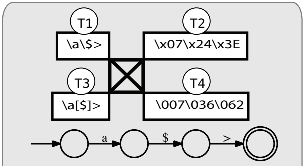
SNG GROUP
(using the abstract `S{3}'
where S is any pattern)



LWB GROUP
(using the abstract `A{2,}'
where A is any pattern)



CCC GROUP
(using the concrete example of `[0-9a]' and assuming an ASCII charset)



LIT GROUP
(using the concrete example `\a\$>'
and assuming an ASCII charset)

Understandability Study

RQ1

Which representations are most understandable?

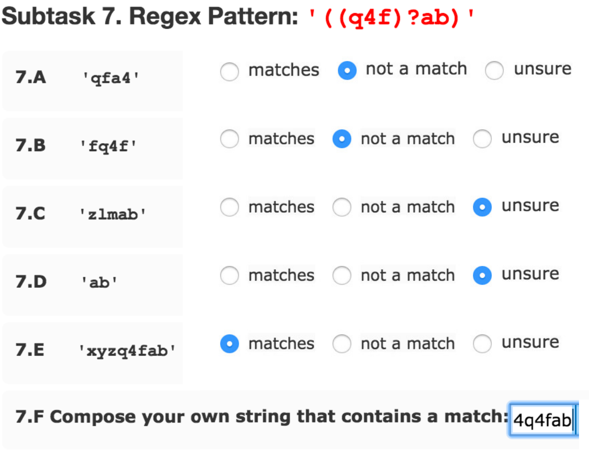
Understandability Study

RQ1

Which representations are most understandable?

- 180 Amazon‘s Mechanical Turk (MTurk) participants
- 60 regular expressions
- 26 equivalence groups (18 of two members, 8 of three members)
- 41 pairs of equivalent regexes

Study Example



Subtask 7. Regex Pattern: `'((q4f)?ab)'`

7.A	`'qfa4'`	○ matches	● not a match	○ unsure
7.B	`'fq4f'`	○ matches	● not a match	○ unsure
7.C	`'zlmab'`	○ matches	○ not a match	● unsure
7.D	`'ab'`	○ matches	○ not a match	● unsure
7.E	`'xyzq4fab'`	● matches	○ not a match	○ unsure

7.F Compose your own string that contains a match: 4q4fab

Comprehension Metrics

1. Matching
2. Composition

Comprehension Metrics

1. **Matching**
2. Composition

String	‘RR*’	Oracle	P1	P2	P3	P4
1						
2						
3						
4						
5						

✓ = match, ✗= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	‘RR*’	Oracle	P1	P2	P3	P4
1	“ARROW”	✓				
2	“qRs”	✓				
3	“R0R”	✓				
4	“qrs”	×				
5	“98”	×				
	Score	1.00				

✓ = match, ×= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	‘RR*’	Oracle	P1	P2	P3	P4
1	“ARROW”	✓	✓			
2	“qRs”	✓	✓			
3	“R0R”	✓	✓			
4	“qrs”	✗	✓			
5	“98”	✗	✗			
	Score	1.00				

✓ = match, ✗= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	‘`RR*`’	**Oracle**	**P1**	**P2**	**P3**	**P4**
1	“ARROW”	✓	✓			
2	“qRs”	✓	✓			
3	“R0R”	✓	✓			
4	“qrs”	✗	✓			
5	“98”	✗	✗			
	Score	1.00	0.80			

✓= match, ✗= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	‘RR*’	Oracle	P1	P2	P3	P4
1	“ARROW”	✓	✓	✓		
2	“qRs”	✓	✓	✗		
3	“R0R”	✓	✓	✓		
4	“qrs”	✗	✓	✗		
5	“98”	✗	✗	✗		
	Score	1.00	0.80	0.80		

✓= match, ✗= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	‘RR*’	Oracle	P1	P2	P3	P4
1	“ARROW”	✓	✓	✓	✓	
2	“qRs”	✓	✓	✗	✗	
3	“R0R”	✓	✓	✓	?	
4	“qrs”	✗	✓	✗	✓	
5	“98”	✗	✗	✗	✗	
	Score	1.00	0.80	0.80		

✓ = match, ✗= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	`‘RR*’`	Oracle	P1	P2	P3	P4
1	“ARROW”	✓	✓	✓	✓	
2	“qRs”	✓	✓	✗	✗	
3	“R0R”	✓	✓	✓	?	
4	“qrs”	✗	✓	✗	✓	
5	“98”	✗	✗	✗	✗	
	Score	1.00	0.80	0.80	0.50	

✓ = match, ✗= not a match, ? = unsure, – = left blank

Comprehension Metrics

1. **Matching**
2. Composition

String	‘RR*’	Oracle	P1	P2	P3	P4
1	“ARROW”	✓	✓	✓	✓	✓
2	“qRs”	✓	✓	✗	✗	?
3	“R0R”	✓	✓	✓	?	-
4	“qrs”	✗	✓	✗	✓	-
5	“98”	✗	✗	✗	✗	-
	Score	1.00	0.80	0.80	0.50	1.00

✓ = match, ✗ = not a match, ? = unsure, – = left blank

Comprehension Metrics

1. Matching
2. **Composition**

	Regex	Composition score
P1	`(q4fab|ab)`	
P2	`(q4fab|ab)`	

Comprehension Metrics

1. Matching
2. **Composition**

	Regex	Composition score
P1	`(q4fab|ab)`	`xyzq4fab`
P2	`(q4fab|ab)`	

Comprehension Metrics

1. Matching
2. **Composition**

	Regex	Composition	score
P1	`(q4fab|ab)`	`xyzq4fab`	1
P2	`(q4fab|ab)`		

Comprehension Metrics

1. Matching
2. **Composition**

	Regex	Composition	score
P1	`(q4fab|ab)`	`xyzq4fab`	1
P2	`(q4fab|ab)`	`acb`	

Comprehension Metrics

1. Matching
2. **Composition**

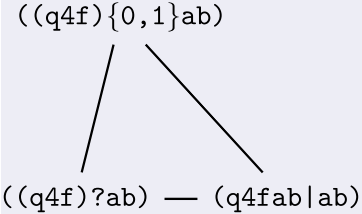
	Regex	Composition	score
P1	`(q4fab|ab)`	`xyzq4fab`	1
P2	`(q4fab|ab)`	`acb`	0

Which representations are most understandable?

Double-Bounded Groups

Which representations are most understandable?

Double-Bounded Groups

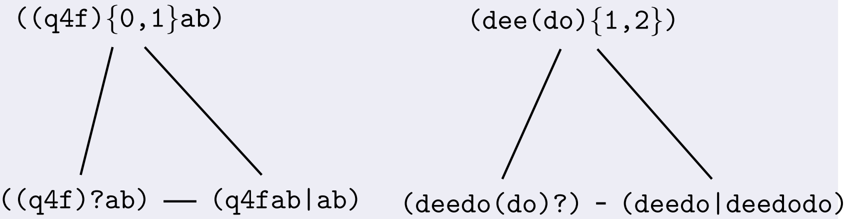


`((q4f){0,1}ab)`

`((q4f)?ab)` — `(q4fab|ab)`

Which representations are most understandable?

Double-Bounded Groups



Which representations are most understandable?

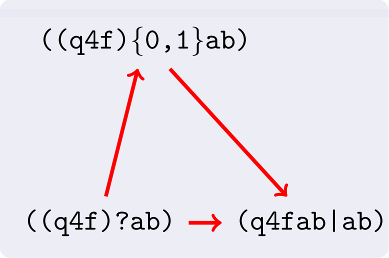
Regex	Match	Comp
`((q4f){0,1}ab)`	82.93	50.00
`((q4f)?ab)`	79.25	40.00
`(q4fab|ab)`	84.50	60.00

`((q4f){0,1}ab)`

`((q4f)?ab)`   `(q4fab|ab)`

Which representations are most understandable?

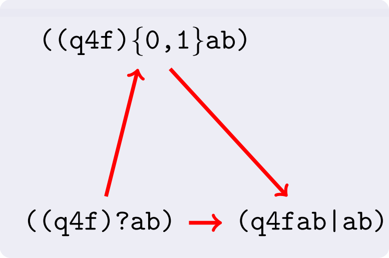
Regex	Match	Comp
`((q4f){0,1}ab)`	82.93	50.00
`((q4f)?ab)`	79.25	40.00
`(q4fab|ab)`	84.50	60.00



Which representations are most understandable?

Regex	Match	Comp
`((q4f){0,1}ab)`	82.93	50.00
`((q4f)?ab)`	79.25	40.00
`(q4fab|ab)`	84.50	60.00

Regex	Match	Comp
`(dee(do){1,2}`	84.83	66.67
`(deedo(do)?)`	77.17	60.00
`(deedo|deedodo)`	90.00	63.33



((q4f){0,1}ab)

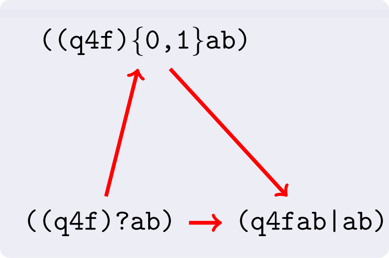
((q4f)?ab)

(q4fab|ab)

Which representations are most understandable?

Regex	Match	Comp
`((q4f){0,1}ab)`	82.93	50.00
`((q4f)?ab)`	79.25	40.00
`(q4fab|ab)`	84.50	60.00

Regex	Match	Comp
`(dee(do){1,2}`	84.83	66.67
`(deedo(do)?)`	77.17	60.00
`(deedo|deedodo)`	90.00	63.33



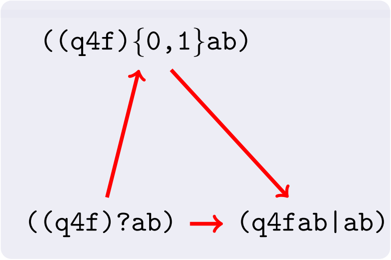
`(dee(do){1,2})`

`(deedo(do)?)` `(deedo|deedodo)`

Which representations are most understandable?

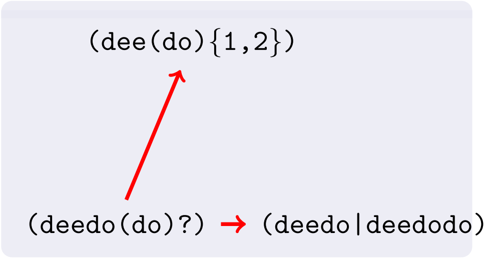
Regex	Match	Comp
`((q4f){0,1}ab)`	82.93	50.00
`((q4f)?ab)`	79.25	40.00
`(q4fab|ab)`	84.50	60.00

Regex	Match	Comp
`(dee(do){1,2}`	84.83	66.67
`(deedo(do)?)`	77.17	60.00
`(deedo|deedodo)`	90.00	63.33



((q4f){0,1}ab)

((q4f)?ab) → (q4fab|ab)



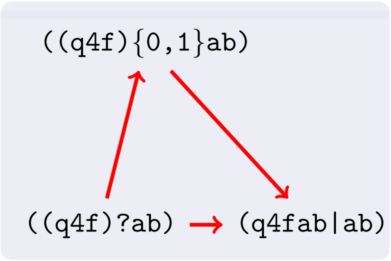
(dee(do){1,2})

(deedo(do)?) → (deedo|deedodo)

Which representations are most understandable?

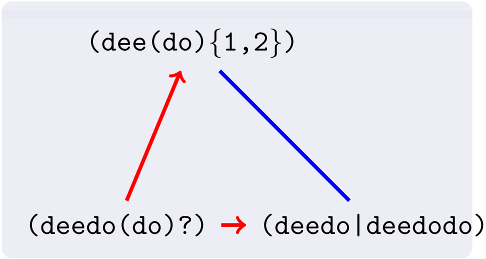
Regex	Match	Comp
`((q4f){0,1}ab)`	82.93	50.00
`((q4f)?ab)`	79.25	40.00
`(q4fab|ab)`	84.50	60.00

Regex	Match	Comp
`(dee(do){1,2}`	84.83	66.67
`(deedo(do)?)`	77.17	60.00
`(deedo|deedodo)`	90.00	63.33



`((q4f){0,1}ab)`

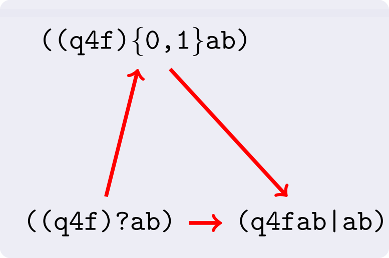
`((q4f)?ab)` → `(q4fab|ab)`



`(dee(do){1,2})`

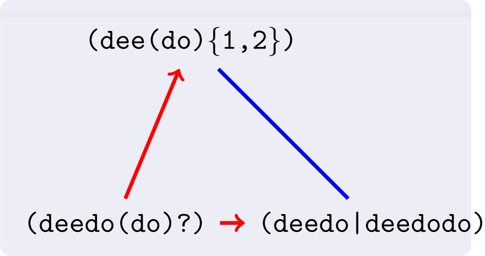
`(deedo(do)?)` → `(deedo|deedodo)`

Topological Ordering



((q4f){0,1}ab)

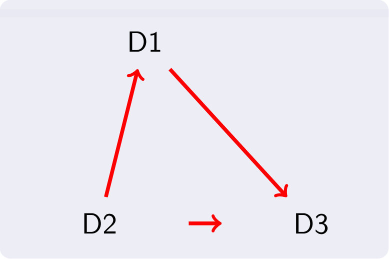
((q4f)?ab) → (q4fab|ab)



(dee(do){1,2})

(deedo(do)?) → (deedo|deedodo)

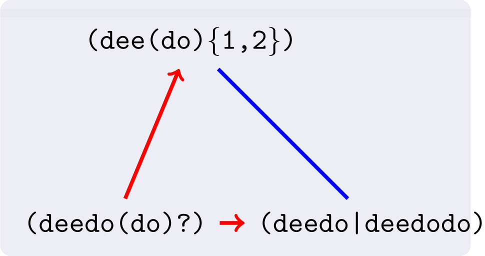
Topological Ordering



D1

D2

D3

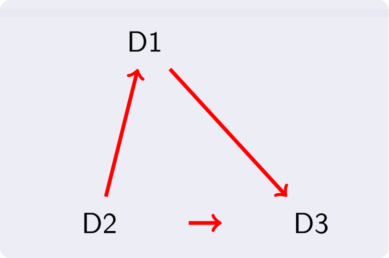


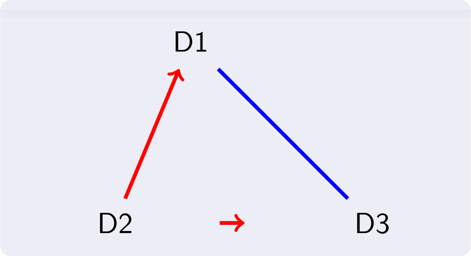
`(dee(do){1,2})`

`(deedo(do)?)`

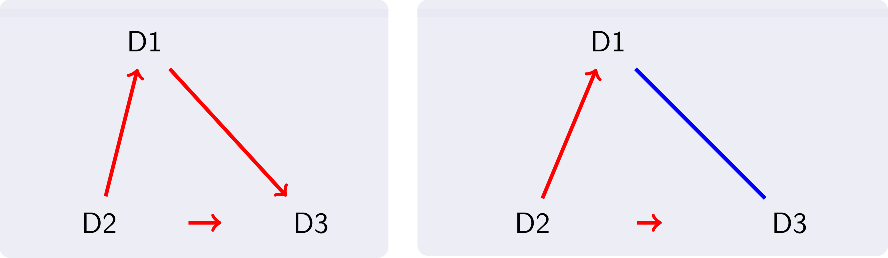
`(deedo|deedodo)`

Topological Ordering





Topological Ordering



Understandability Ordering

$D3 > D1 > D2$

Community Study

RQ2

Which representations have the strongest community support based on frequency?

Community Study

RQ2

Which representations have the strongest community support based on frequency?

- 13,597 distinct regex patterns from 1,544 Github Python projects
- Mapping regexes to representations: PCRE feature, string pattern, token stream

Frequent Representations

Rep	Example	nPatterns	% patterns	nProjects	% projects
D1	`((q4f){0,1}ab)`	346	2.5%	234	15.2%
D2	`((q4f)?ab)`	1,871	13.8%	646	41.8%
D3	`(q4fab|ab)`	10	.1%	27	1.7%

Frequent Representations

Rep	Example	nPatterns	% patterns	nProjects	% projects
D1	`((q4f){0,1}ab)`	346	2.5%	234	15.2%
D2	`((q4f)?ab)`	1,871	13.8%	646	41.8%
D3	`(q4fab|ab)`	10	.1%	27	1.7%

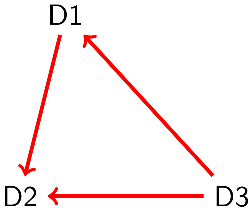
D1

D2

D3

Frequent Representations

Rep	Example	nPatterns	% patterns	nProjects	% projects
D1	`((q4f){0,1}ab)`	346	2.5%	234	15.2%
D2	`((q4f)?ab)`	1,871	13.8%	646	41.8%
D3	`(q4fab|ab)`	10	.1%	27	1.7%



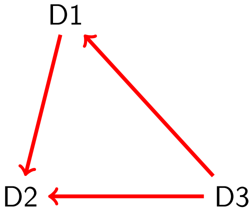
D1

D2

D3

Frequent Representations

Rep	Example	nPatterns	% patterns	nProjects	% projects
D1	`((q4f){0,1}ab)`	346	2.5%	234	15.2%
D2	`((q4f)?ab)`	1,871	13.8%	646	41.8%
D3	`(q4fab|ab)`	10	.1%	27	1.7%



Community Ordering

$D2 > D1 > D3$

Desirability Analysis

RQ3

Which regex representations should we use?

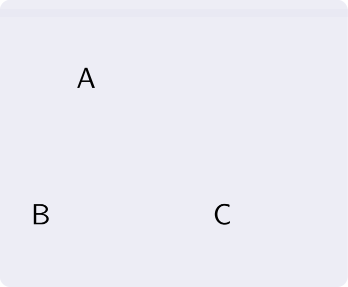
- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`

Desirability Analysis

RQ3

Which regex representations should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`



A

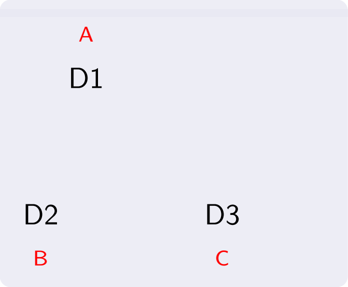
B C

Desirability Analysis

RQ3

Which regex representations should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`



A

D1

D2

D3

B

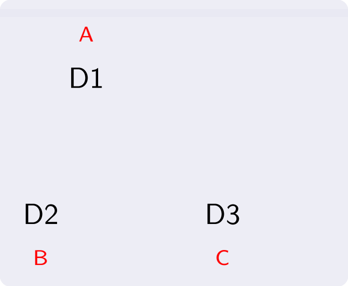
C

Desirability Analysis

RQ3

Which regex representations should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`



A

D1

D2

D3

B

C

Topological Ordering

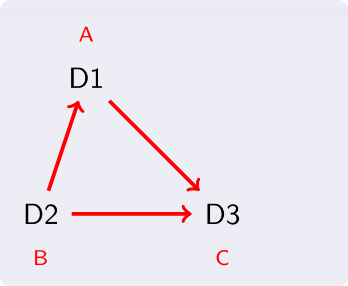
Understandability: $D3 > D1 > D2$

Desirability Analysis

RQ3

Which regex representations should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`



Topological Ordering

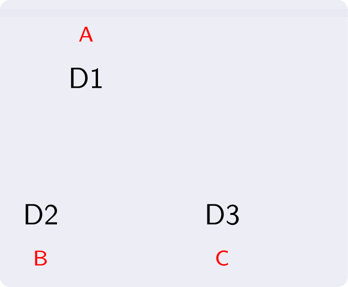
Understandability: $D3 > D1 > D2$

Desirability Analysis

RQ3

Which regex representations should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`


A

D1

D2

D3

B

C

Topological Ordering

Understandability: $D3 > D1 > D2$

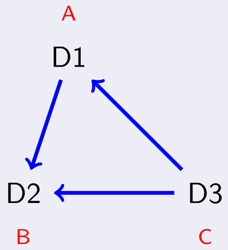
Community: $D2 > D1 > D3$

Desirability Analysis

RQ3

Which regex representations should we use?

- A = `[1-9][0-9]{0,2}`
- B = `[1-9][0-9]?[0-9]?`
- C = `[1-9]|[1-9][0-9]|[1-9][0-9][0-9]`



Topological Ordering

Understandability: $D3 > D1 > D2$
Community: $D2 > D1 > D3$

Ordering Results

Equivalence Class	Understandability	Community
Custom Character Class	C1 C5 C3 C4 C2	C1 C3 C2 C4 C5
Double-Bounded	D3 D1 D2	D2 D1 D3
Lower-Bounded	L3 L2	L3 L2 L1
Single-Bounded	S2 S1	S2 S1 S3
Literal	T1 T3 T2 T4	T1 T3 T2 T4

What We Learn

1. Commonly used regexes are NOT always easier to understand!
2. Replace * with + when possible.
3. Use literal character! If not possible, use hex encoding.
4. Use range feature for character sets when possible.
   - letters a to g: **[a-g]**, [abcdefg], `[a|b|c|d|e|f|g]`

Limitations

- Five types of equivalence classes
- Python code
- Regex length is short
  - `ab|abab`
  - `thisbadchoice|thisbadchoicethisbadchoice`
- DFA size is small: 2 to 8
- . . .

Post Analysis

ANOVA analysis: which factor can impact comprehension?

- Regex representation
- DFA size (matching: *$\alpha = 0.05$, composition: **$\alpha = 0.01$)
- Regex length

Opportunities for Future Work!

DFA size

How does DFA size impact comprehension?

Opportunities for Future Work!

DFA size

How does DFA size impact comprehension?

More types of equivalence classes

Consider multiline option, case insensitive, backreference?

Opportunities for Future Work!

DFA size

How does DFA size impact comprehension?

More types of equivalence classes

Consider multiline option, case insensitive, backreference?

Automatic identification

Could we automatically build equivalence classes?

Questions?

Peipei Wang
pwang7@ncsu.edu
North Carolina State University