Generating Evil Test Strings for Regular Expressions

Eric Larson and Anna Kirk

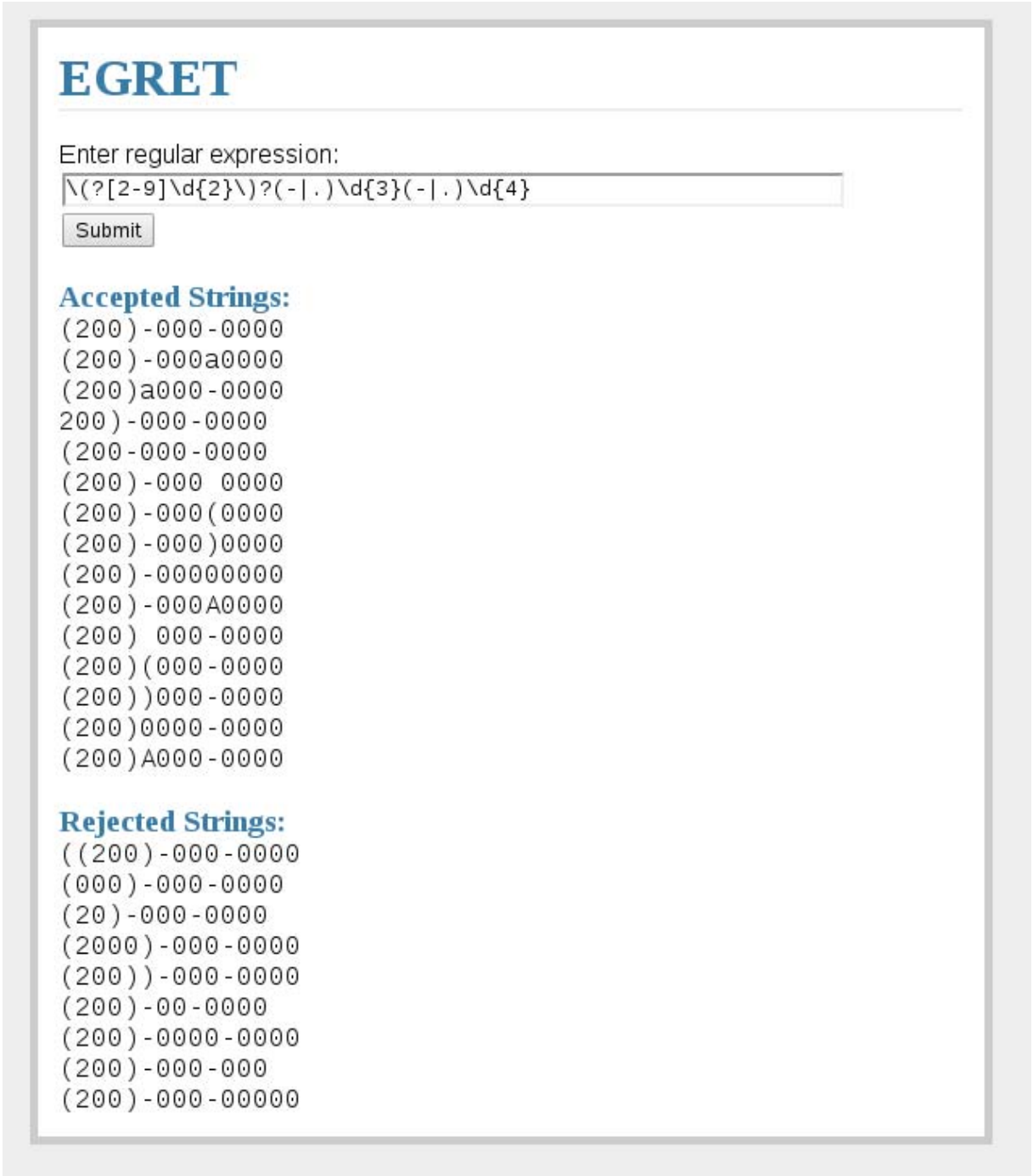
Seattle University

Work to be presented at the International Conference on Software Testing (ICST)
April 2016

Writing Regular Expressions

Writing a completely correct regular expression (regex) is challenging.

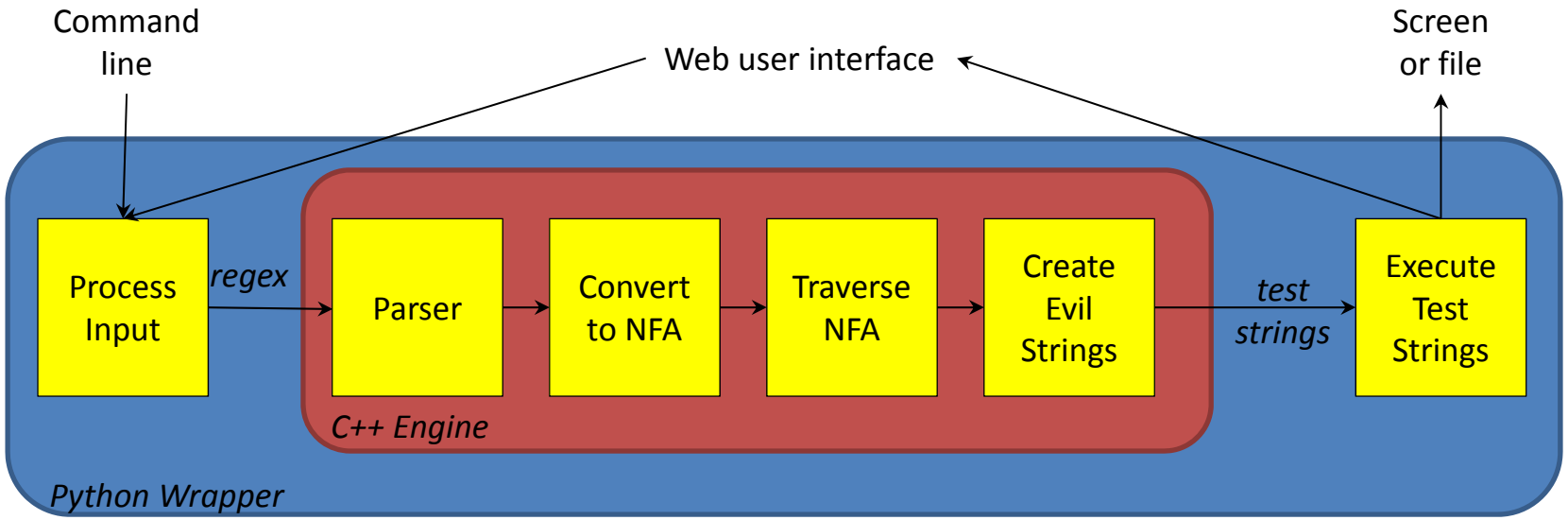
1. A regex often permits more strings than what should be accepted.
   - Example: `^(\d|,)*\.?\d*$`
2. The regex compiler catches very few errors.
   - Most regexes are syntactically correct.
   - Cannot emit warnings at run-time.
   - Example: `[A|M|P|M]{2}|[a|m|p|m]2}`



EGRET

Evil Generation of Regular Expression Tests

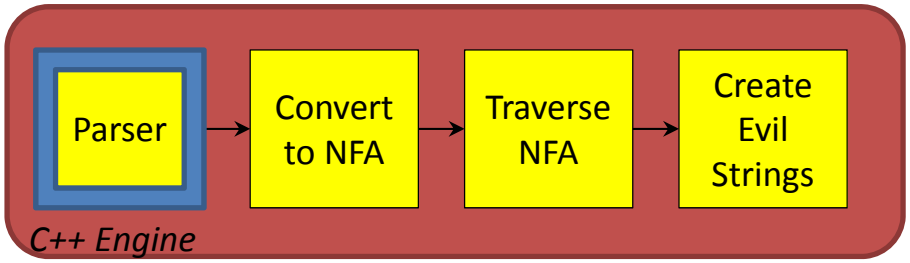
EGRET Overview



- Works on Python regular expressions.
- Supports most of the primary regular expression constructs.
- Limited support for extensions (?...)

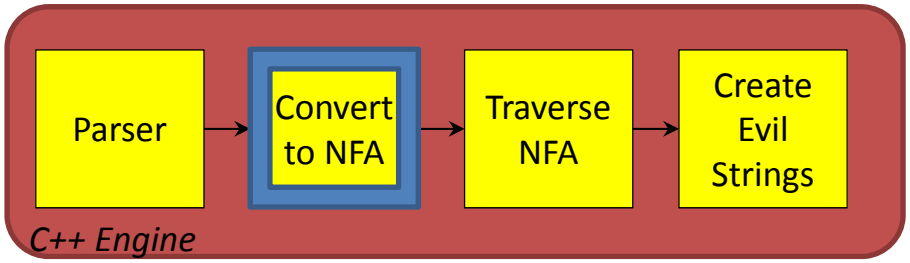
Parser

- Converts regular expression into a parse tree.
- Uses a “character set” node broadly to represent any construct that represents multiple characters:
  - Character sets such as `[A-Z\s_]` and `[^;]`
  - Character classes such as `\w` and `\D`
  - Wildcards using `.`
- Emits warnings on poorly formed character sets.



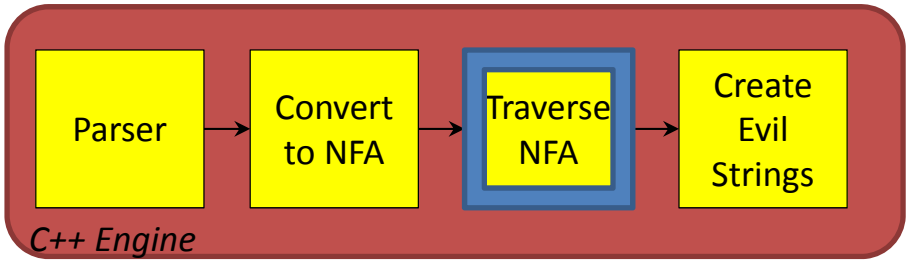
Convert to NFA

- The parse tree is converted to a NFA (non-deterministic finite state automata).
- Edges can be:
  - Individual characters
  - Character sets
  - Anchors such as ^ and $
  - Epsilon transitions
- Repeat quantifiers (such as * and +) are represented using special states.
  - The NFA is free of cycles.



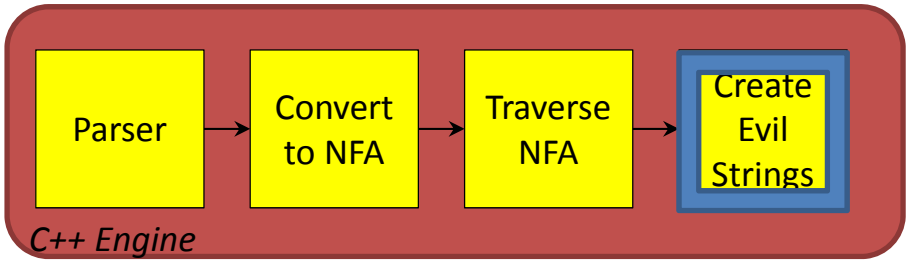
Traverse NFA

- NFA is traversed to obtain basis path coverage.
  - State or transition coverage is not sufficient.
  - Path coverage generates too many paths.
- An initial set of strings is generated.
  - Repeat quantifiers are iterated once (or *n* times for `{n,m}` or `{n}` if *n* > 1).
  - A valid character is chosen for each character set.
- Emits warning on poor anchor (^ and $) usage.



Create Evil Strings

- Using the initial set of strings, create new strings by:
  - altering the number of iterations for each repeat quantifier
  - changing the character used for a character set
- Changes are only applied to one string that contains that element.



Repeat Quantifier Iterations

Repeat Quantifier	Number of Iterations
* or {0,}	0, ***1***, 2
+ or {1,}	0, ***1***, 2
? or {0,1}	0, ***1***, 2
{0,n} or {,n}	0, ***1***, n, n + 1
{m,n} (m > 0 and n > m)	m – 1, ***m***, n, n + 1
{m} or {m,m}(m > 0)	m – 1, ***m***, m + 1
{m,} (m > 1)	m – 1, ***m***

The number of iterations in the initial string are ***bold and italicized***

Character Set Processing

- Each character specified individually in a character set will appear in a test string.
- For digits:
  - If all digits 0-9 are allowed, a test string using one digit will be generated.
  - If some digits are allowed and some are not, two test strings are generated: one with a good digit and one with a bad digit
  - Uppercase and lowercase letters separately are handled similarly.
- For character sets that accept all punctuation marks:
  - A test string is generated for all punctuation marks that appear somewhere in the regex.

Evaluation

		Regexes	***Not Supported***	***Tested Regexes***	***Buggy Regexes***
RegExLib.com Category	address / phone	104	6	98	40
	dates / time	135	18	117	59
	email	38	5	33	12
	markup / code	63	13	50	23
	misc	173	23	150	51
	numbers	107	5	102	23
	string	91	18	73	31
	uri	74	5	69	26
Python Program	advas (0.2.5)	16	0	16	0
	beautifulsoup4 (4.3.2)	7	0	7	3
	pychecker (0.8.19)	4	0	4	1
	pymetrics (0.8.1)	6	1	5	0
	tables (3.1.1)	8	0	8	0
	Trac (1.0.1)	63	4	59	15
	Library total	785	93	692	265
	Programs total	104	5	99	19
	Total	***889***	***98***	***791***	***284***

Classifying Bugs

Bug Type	Example Buggy Regex	Generated Bad String (or WARNING)	Number of Bugs
Bad Range	`^[D-d][K-k]-[1-9]{1}[0-9]{3}$`	WARNING	22
Anchor Usage	`^[-+]?\d+(\.\d+)?|[-+]?\.\d+?$`	WARNING	22
Character Set	`[AM|PM|am|pm]{2,2}`	`|m`	41
Delimiter Mismatch	`href[ ]*=[ ]*('|\")([^\"'])*('|\")`	`href = 'a"`	72
Insufficient Negation	`href=[\"\']?((?:[^>]|[^\s]|[^"]|[^'])+)[\"\']?`	`href='''`	31
Wildcard Too Wild	`(\w+?@\w+?\x2E.+)`	`a@a..`	41
Wrong Repeat	`^\d+(?:\.\d{0,2})?$`	`0.0`	28
Other	`(.*\.([wW][mM][aA])|([mM][pP][3])$)`	`MP3`	90

Other Key Results

- Number of generated strings is manageable.
  - 96% regexes generated less than 100 strings.
  - Most strings generated: 307
  - EXREX, another tool, generated over 1,000 strings for 472 regexes.
- Only 3 bugs were detected by generating a rejected string that should be accepted.
  - But there's value in confirming similar strings are actually rejected.

Future Work

- Add support for elements currently unsupported.
- Improve the algorithm.
  - Add more warnings to detect likely errors.
- Make the generated strings more readable.
  - Many character strings are only 1 or 2 characters long.
- Perform a usability study with likely users.

Questions?

- For more information:

http://fac-staff.seattleu.edu/elarson/web/egret.htm

- Tool (code and web interface) will be available on April 12 (start of ICST)