Playing Regex Golf with Genetic Programming

A. Bartoli, A. De Lorenzo, E. Medvet, F. Tarlao
University of Trieste, Italy
The problem ("Regex Golf")

- Specific kind of code golf
- Writing the shortest regular expression which:
  - matches all the strings in a given list
  - does not match any strings in another list
The Phantom Menace | Attack of the Clones | Revenge of the Sith | A New Hope | The Empire Strikes Back | Return of the Jedi

Positive examples:
- The Phantom Menace
- Attack of the Clones
- Revenge of the Sith
- A New Hope
- The Empire Strikes Back
- Return of the Jedi

Negative examples:
- The Wrath of Khan
- The Search for Spock
- The Voyage Home
- The Final Frontier
- The Undiscovered Country
- Generations
- First Contact
- Insurrection
- Nemesis
Regex Golf - Best solution

Positive examples
- The Phantom Menace
- Attack of the Clones
- Revenge of the Sith
- A New Hope
- The Empire Strikes Back
- Return of the Jedi

Negative examples
- The Wrath of Khan
- The Search for Spock
- The Voyage Home
- The Final Frontier
- The Undiscovered Country
- Generations
- First Contact
- Insurrection
- Nemesis

m | [tN] | B
16 difficult instances

<table>
<thead>
<tr>
<th>Match all of these...</th>
<th>and none of these...</th>
<th>Match all of these...</th>
<th>and none of these...</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ aerate arrest chant</td>
<td>✗ aerate arrest chant</td>
<td>✓ x</td>
<td>✗ x</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xx</td>
<td>✗ xx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxx</td>
<td>✗ xxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxx</td>
<td>✗ xxxxxxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxxxxxxxxxx</td>
<td>✗ xxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxxxxxxxxxxxxx</td>
<td>✗ xxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxxxxxxxxxxxxxx</td>
<td>✗ xxxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxxxxxxxxxxxxxx</td>
<td>✗ xxxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxxxxxxxxxxxxxx</td>
<td>✗ xxxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>✓ arrest arrest chant</td>
<td>✗ arrest arrest chant</td>
<td>✓ xxxxxxxxxxxxxxxxxxx</td>
<td>✗ xxxxxxxxxxxxxxxxxxx</td>
</tr>
</tbody>
</table>

- [32 chars]
- [64 chars]
- [128 chars]
- [256 chars]
- [512 chars]
- [1024 chars]
A lot of vibrant discussions...

GitHub Gist

jpsim / answers.md

Last active on 20 Dec 2013

Regex Golf Answers (http://regex.all.nu) HN: https://news.ycombinator.com/item?id=6941231

answers.md

1. Plain Strings (207): `foo`
2. Anchors (208): `x$
3. Ranges (202): `[a-f]`$
4. Backrefs (201): `(...)`$
5. Abba (169): `(\(?!\(\().()\)\)\)\)$\n6. A man, a plan (177): `[^0-9]`$
7. Prime (288): `[^!\(\)\[,]\ ]$`
8. Four (199): `(\(\)\(\)\(\))$`
9. Order (198): `[^0-9].\[\[\]\]`$
10. Triples (507): `(\(\)\(\)\(\))\((\)\(\)\(\))\((\)\(\)\(\))$`
11. Glob (364): `(\(?fs[f][l][p][t][v]\)\)\((\?[a][a][m]|\d|\d|\[p|l\]|\[r|l\]|\[e|l\])\)`$
13. Powers (58): `^\*\*(\(\)\(\))\((\)\(\))\((\)\(\))\((\)\(\))\((\)\(\))\((\)\(\))\((\)\(\))$`

Total Score: 3060
...really a lot...

@furyk: I don't think it is possible to create a regular expression that matches any list of words in alphabetical order (one of 6 letter combinations only would be incredibly long). I guess the hint is a reference back to #9 which is possible without cheating (even though it says to cheat). I stand corrected. ty @berndjendrissek.

You could create the brute force expression of every possible 6 letter word in order containing the letters aenrst (which is still terribly long), or the one containing all the words in this list (-69 points, goes: ^\(aen\)er^)"(erest?)"..., and is 406 characters long). Anything beyond that I feel is cheating so badly that it doesn't matter how much more you do.

I want to know what the meaning of the hitchhikers guide reference in long count v2 is all about. My solution there cheats a lot (it matches far more than the intended binary nibble count sequence, just happens to not match any of his counter examples because none of them swap more than 1 nibble out).
February 26-th, 2014

Best known Regex Golf solutions (SPOILERS)

See Regex Golf

See also:
Regex Golf Answers (Gist)
Best possible answers collected so far for Regex Golf (reddit)
using finite state machines

Regex Golf with Arbitrary Lists

We can define a convenience function to do this finding and verifying, and we might as well do it in both directions (e.g. separating winners from losers and losers from winners). We will also report the number of characters in the solution and the competitive ratio of the solution: the ratio between the length of a trivial solution and the solution found (a trivial solution for the set of winners \{'one', 'two', 'three'\} is the disjunction '^\(one|two|three\)$').

def findboth(A, B):
    "Find a regex to match A but not B, and vice-versa. Print summary."
    for (W, L, legend) in [(A, B, 'A-B'), (B, A, 'B-A')]:
        solution = findregex(W, L)
        assert verify(solution, W, L)
        ratio = len('^(' + OR(W) + ')$') / float(len(solution))
        print '%3d chars, %4.1f ratio, %2d winners %s: %r' % (len(solution), ratio, len(W), legend, solution)
Our previous GP-based tool

Automatic regex generation from examples

For data extraction

<table>
<thead>
<tr>
<th>Input string</th>
<th>String to match</th>
</tr>
</thead>
<tbody>
<tr>
<td>this is a valid ip 127.0.0.1</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>12.3 is just a number</td>
<td></td>
</tr>
<tr>
<td>ping from 192.168.0.1</td>
<td>192.168.0.1</td>
</tr>
<tr>
<td>today is 7/1/2012</td>
<td></td>
</tr>
<tr>
<td>msg to 06.231.55.67 sent</td>
<td>06.231.55.67</td>
</tr>
<tr>
<td>telnet 17.23.133.22:8080</td>
<td>17.23.133.22</td>
</tr>
<tr>
<td>this is old plain text</td>
<td></td>
</tr>
<tr>
<td>germany-italy 1-2</td>
<td></td>
</tr>
<tr>
<td>172.30.40.254 is a server ip</td>
<td>172.30.40.254</td>
</tr>
<tr>
<td>It's nine o'clock on a Saturday</td>
<td></td>
</tr>
</tbody>
</table>

IEEE Computer, GECCO Hot Off the Press
http://regex.inginf.units.it
Key observations

● We need a **classifier**---not an **extractor**
  ○ No need to identify boundaries

● No need to infer a **general pattern**

● No need to process streams with unknown items
  ○ We need to “overfit”
Our approach

- Candidate regex = tree

- Internal nodes: usual regex operators
  - No greedy/lazy quantifiers
    (execution time too long to be practical)

- Leave nodes:
  - a-z, A-Z, ^, $, ..
  - problem-dependent elements (next slide)
Problem-dependent elements

- All characters in matches
- All partial ranges including those characters
- All “most useful” n-grams (n=2,3,4)
  - Build all n-grams
  - Score each n-gram based on its frequency:
    +1 for each match, -1 for each unmatch
  - Rank n-grams
  - Select the smallest set totalling M points
    (M being the number of matches)
Evolutionary search

- 500 individuals, 1000 generations
- 32 independent searches

- Multiobjective fitness (NSGA-II)
- Minimize
  - Number of misclassifications
  - Length
Problem in detail

- 16 instances, each with:
  - Set of matches $M$
  - Set of unmatches $U$
  - Weight $w$ (a “difficulty” coefficient)

- Score of regex $r$ on a given instance:
  - $w \times \text{#misclassifications} - \text{length}(r)$
<table>
<thead>
<tr>
<th>Problem name</th>
<th>$M$</th>
<th>$U$</th>
<th>$w_T$</th>
<th>Ideal score</th>
<th>Best human score</th>
<th>Best human solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain strings</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>210</td>
<td>207</td>
<td>foo</td>
</tr>
<tr>
<td>Anchors</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>210</td>
<td>208</td>
<td>k$</td>
</tr>
<tr>
<td>Ranges</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>210</td>
<td>202</td>
<td><em>[a-f]</em>$^$</td>
</tr>
<tr>
<td>Backrefs</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>210</td>
<td>201</td>
<td>(...).*\1</td>
</tr>
<tr>
<td>Abba</td>
<td>21</td>
<td>22</td>
<td>10</td>
<td>210</td>
<td>193</td>
<td><em>(?!.</em>(.)(.)\2\1)</td>
</tr>
<tr>
<td>A man, a plan</td>
<td>19</td>
<td>21</td>
<td>10</td>
<td>190</td>
<td>177</td>
<td><em>(.[^p].!</em>$</td>
</tr>
<tr>
<td>Prime</td>
<td>20</td>
<td>20</td>
<td>15</td>
<td>300</td>
<td>286</td>
<td>*(?!(.+)\1$</td>
</tr>
<tr>
<td>Four</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>210</td>
<td>199</td>
<td>(.(.\1{3)</td>
</tr>
<tr>
<td>Order</td>
<td>21</td>
<td>21</td>
<td>10</td>
<td>210</td>
<td>199</td>
<td>^.5[^e]$?§</td>
</tr>
<tr>
<td>Triples</td>
<td>21</td>
<td>21</td>
<td>30</td>
<td>630</td>
<td>596</td>
<td>00($</td>
</tr>
<tr>
<td>Glob</td>
<td>21</td>
<td>21</td>
<td>20</td>
<td>420</td>
<td>397</td>
<td>a!\c$</td>
</tr>
<tr>
<td>Balance</td>
<td>32</td>
<td>32</td>
<td>10</td>
<td>320</td>
<td>289</td>
<td>^(&lt;&lt;&lt;&lt;&lt;&lt;??&gt;&gt;</td>
</tr>
<tr>
<td>Powers</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>110</td>
<td>93</td>
<td><em>(?!(.+)\1</em>$</td>
</tr>
<tr>
<td>Long count</td>
<td>1</td>
<td>20</td>
<td>270</td>
<td>270</td>
<td>254</td>
<td>((.+\0 \2?1{7}</td>
</tr>
<tr>
<td>Long count v2</td>
<td>1</td>
<td>21</td>
<td>270</td>
<td>270</td>
<td>254</td>
<td>((.+\0 \2?1{7}</td>
</tr>
<tr>
<td>Alphabetical</td>
<td>17</td>
<td>17</td>
<td>20</td>
<td>340</td>
<td>317</td>
<td>.r.{32}r</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td>4320</td>
<td>4072</td>
<td></td>
</tr>
</tbody>
</table>
Baseline

- GP-RegexExtract (our data extraction tool)
- “Norvig” (January 2014)
  - Deterministic algorithm widely discussed on the web
  - IMPORTANT
    - Not developed for this challenge
    - Designed for completely avoiding misclassifications
  - Comparison not fully fair...but the only algorithm we were aware of
Results: Warning

- The web site does not collect scores/rankings
- Programmers advertised solutions on forums
  - GitHub, Reddit, Hacker News
- Sometimes only scores without any evidence
- Sometimes slightly improving earlier results by other programmes

- No evidence of time spent
Great Results!

- 6-th/8-th worldwide
  - At the time
  - To the best of our knowledge

- Without any hint from other programmers
- Much better than the baseline
## Execution time, Actual regexes

<table>
<thead>
<tr>
<th>Problem</th>
<th>GP-RegexGolf</th>
<th>Actual regexes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>foo</td>
<td>k$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>(~a-f)[a-f]</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>vi[b]r[no][p]t</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>zo[nt]ca[lt]</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>oo[x]kled[ml]t</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>~(?:x[A-Zx]+)\ix</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>ellj\Wele\o\o\Mas1\da\do\ch[1-p]lo</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>24155</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>0o</td>
</tr>
</tbody>
</table>
| 10      |              | (?:=(?:<\>|\>|*)\))2?==(?:=(?:<(?:<(?:<(?:<(?:<(?:<(?:<
| 11      |              | 3?==(?:<(?:<(?:<(?:<(?:<(?:<(?:<(?:<
| 12      |              | \5?==(?:<(?:<(?:<(?:<(?:<(?:<(?:<
| 13      |              | ~(?:<(?:<(?:<(?:<(?:<(?:<(?:<
| 14      |              | 0000|0001|0010|0011|0100|0101|0110|0111|1000|1001|1010|1011|1100|1101|1110|1111|
| 15      |              | 1000|0001|0010|0011|0100|0101|0110|0111|1000|1001|1010|1011|1100|1101|1110|1111|
| 16      |              | tena[~en](?:<(?:<(?:<ren.|(?:<(?:<(?:<(?:<(?:<(?:<(?:<
| Total   | 820          |                |
Summary

- Evolutionary computation has reached a level in which it may successfully compete with human programmers.
- In scenarios explicitly designed to test their practical skills and creativity.
- And, it may do so without any starting hint or external help.
A web-based prototype is public available at http://regex.inginf.units.it/golf