Program Boosting: Program Synthesis via Crowd-Sourcing

Cochran, Robert A., et al.

presenter name(s) removed for FERPA considerations
What is a regular expression (regex)?

- A regular expression defines a pattern that strings must match

- What would a regex for **phone numbers** look like?

  **Ans1:** `^[0-9]{3}-[0-9]*-[0-9]{4}$`?

  and

  **Ans2:** `^[0-9]{3}-[0-9]{3}-[0-9]*$`?

  becomes

  `^[0-9]{3}-[0-9]{3}-[0-9]{4}$`

  A regular expression for phone numbers with **hyphens**!
”A regex for phone numbers?”

- Was “^[0-9]{3}-[0-9]{3}-[0-9]{4}$” also your answer in mind?


People think differently!
Observation: People think differently!

- It might be due to **ambiguity** in task **specification**
- Or they just consider different cases, cover different bases

- **Simple** task can be really **challenging**!
Research Questions

- Can crowd-sourcing improve solution accuracy on difficult programming tasks, by synthesizing over individual programmers’ inaccurate solutions?
  - *Task such as coming up with a regular expression to recognize email addresses, URLs, phone numbers, and dates*
Key Idea: Crowdsourcing and Two-Crowds

- While people may get different parts wrong, blending these partially incorrect programs may provide better solutions.

- Experts/Developers and non-professional crowd each can contribute different aspects to a task in the program boosting process.
Example: Regex for URL

- **Bountify**

  - http://foo.com/blah_blah/(wikipedia)_(again)
  - http://www.example.com/wpstyle/?p=364
  - https://www.example.com/foo/?bar=baz&inga=42&quux
  - http://df.ws/123

  **Positive**

  - http://userid@example.com:8080
  - http://userid@example.com:8080/
  - http://userid:password@example.com

  **Negative**

  - http://-.z_.tr/?\+=%40%80%2f%00@example%com[...
  - http://%@live.com
  - http://userid@-example.com:8080/
  - http://userid@example.com:808%
Example: Regex for URL
- From experts
(Crowd #1)

Regular expression to validate URLs

Please write a regular expression that validates URLs that may start with http, https, or ftp. Note that we are asking for original work. Please do not copy your answer from other sites.

All inputs below should be accepted by your regular expression.

<table>
<thead>
<tr>
<th>Regex source</th>
<th>Regex length</th>
<th>True positive</th>
<th>True negative</th>
<th>Overall accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>@krijnhoetmer</td>
<td>115</td>
<td>.78</td>
<td>.41</td>
<td>.59</td>
</tr>
<tr>
<td>@gruber</td>
<td>71</td>
<td>.97</td>
<td>.36</td>
<td>.65</td>
</tr>
<tr>
<td>@gruber v2</td>
<td>218</td>
<td>1.00</td>
<td>.33</td>
<td>.65</td>
</tr>
<tr>
<td>@cowboy</td>
<td>1,241</td>
<td>1.00</td>
<td>.15</td>
<td>.56</td>
</tr>
<tr>
<td>@mattfarina</td>
<td>287</td>
<td>.72</td>
<td>.44</td>
<td>.57</td>
</tr>
<tr>
<td>@stephenhay</td>
<td>38</td>
<td>1.00</td>
<td>.64</td>
<td>.81</td>
</tr>
<tr>
<td>@scottgonzales</td>
<td>1,347</td>
<td>1.00</td>
<td>.15</td>
<td>.56</td>
</tr>
<tr>
<td>@rodneyrehm</td>
<td>109</td>
<td>.83</td>
<td>.36</td>
<td>.59</td>
</tr>
<tr>
<td>@imme_emosol</td>
<td>54</td>
<td>.97</td>
<td>.74</td>
<td>.85</td>
</tr>
<tr>
<td>@diegoperini*</td>
<td>502</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 1: Representative regular expressions for URLs obtained from http://mathiasbynens.be/demo/url-regex. For every possible solution we show its length, true and false positive rates, and the overall accuracy. The last row is the winner.
Representation: Symbolic Finite Automata (SFA)

- Extension of classical finite state automata to represent regexes
- Allow transitions to be labeled with predicates
  - Needed to handle UTF-16 (\(2^{16}\) characters)
Program Boosting: System Architecture

Crowd #1
Expert/ Developers

Crowd #2
Non-professional
Program Boosting: Iterative Genetic Programming

Crowd #1: Crowd-Sourced initial programs

Golden examples

Perfect or budget = 0?

Program with best fitness

Updated programs & updated examples

Crossovers

Mutations

Customized operations

Fitness

Crowd #2 Consensus
Fitness Scoring

How well does a program perform on a given data set?

The dashed region is where the program is considered correct.

\[
\text{Accuracy} = \frac{|A \cap P| + |N \setminus A|}{|P \cup N|}
\]

A = Examples a program considered positive
P = Positive examples
N = Negative examples
Experiment Evaluation

Initial Condition

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Initial Test Set</th>
<th>Candidate regexes</th>
<th>Candidate regex source:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Phone numbers</td>
<td>20</td>
<td>29</td>
<td>8</td>
</tr>
<tr>
<td>Dates</td>
<td>31</td>
<td>36</td>
<td>6</td>
</tr>
<tr>
<td>Emails</td>
<td>7</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>URLs</td>
<td>36</td>
<td>39</td>
<td>9</td>
</tr>
</tbody>
</table>

Boosting Process

- Tested on 465 genetic programmed regexes
- Limited the boosting iteration to 10 for each regex
- Generated 0-207 test strings from each regex
## Results

<table>
<thead>
<tr>
<th>EVALUATED ON...</th>
<th>Initial Test Set</th>
<th>Crowd-Sourced Test Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Boosted</td>
</tr>
<tr>
<td></td>
<td>initial</td>
<td>no crowd</td>
</tr>
<tr>
<td>Phone numbers</td>
<td>0.80</td>
<td>0.90</td>
</tr>
<tr>
<td>Dates</td>
<td>0.85</td>
<td>0.99</td>
</tr>
<tr>
<td>Emails</td>
<td>0.71</td>
<td>0.86</td>
</tr>
<tr>
<td>URLs</td>
<td>0.67</td>
<td>0.91</td>
</tr>
</tbody>
</table>

The more comprehensive test set for final evaluation

Significant improvement on already high quality initial input.

Average 16.25% accuracy boost
Time and Cost

Time

- Main time cost is the classification latency from Mechanical Turk
- Between the four test categories, the averages ranged from 4 minutes to 37 minutes

Monetary Cost:

- Bountify: $5-$10 per question for total 4 questions
- Mechanical Turk: average $0.41-$3 per regex for 465 regexes
Contributions

- **Program Boosting**
  Method for *semi-automatic* program synthesis by blending a set of initial *crowd-sourced programs*

- **CROWDBOOST**
  Tool for generations of *complex regular expressions*
  - First to adopt *symbolic setting* with genetic programming using *Symbolic Finite Automata* (SFAs)
    - Supports complex alphabets such as UTF-16
Discussions Q1

Any other possible applications? Other than Regex?
Money:
How do we compensate the programmers?
What if we can’t afford to pay them?
Discussions Q3

When to stop collecting answers?
How confident are you that the majority is correct?
How to counteract overfitting to the data?
Discussions Q5

There may be slow response times from the crowd. How can this be fixed?
• Information Flocking Applied to Genetic Programming Visualization, slides
• In search of the perfect URL validation regex https://mathiasbynens.be/demo/url-regex
• Program Boosting: Program Synthesis via Crowd-Sourcing, POPL ‘15, slides
Thank You!
Regular expression to validate US phone numbers

Please write a regular expression that validates a US 10-digit phone numbers, with an optional US country code. Note that we are asking for original work.

Please do not copy your answer from other sites

All inputs below should be accepted by your regular expression.

- 800 555 1212
- (828) 112 5555
- 404-555-1234
- 2141231243
- 1-241-123-1212

All of these inputs should be rejected by your regular expression:

- 789 789
- 44a 555 1234
- 2-555-214-7899
- 828)112-3555

Please provide the regular expression in the form /^ YOUR ANSWER IS HERE $/ as part of your answer. Please test your regex on the samples provided before submitting.
### Example: Twitter Example for Regex

<table>
<thead>
<tr>
<th>URL</th>
<th>Spoon Library</th>
<th>@krijnhoetmer</th>
<th>@gruber</th>
<th>@gruber v2</th>
<th>@cowboy</th>
<th>Jeffrey Friedl</th>
<th>@mattfarina</th>
<th>@stephenhay</th>
<th>@scottgonzales</th>
<th>@rodneyrehm</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://foo.com/blah_blah">http://foo.com/blah_blah</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://foo.com/blah_blah/">http://foo.com/blah_blah/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://foo.com/blah_blah_(wikipedia)">http://foo.com/blah_blah_(wikipedia)</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://foo.com/blah_blah_(wikipedia)_again">http://foo.com/blah_blah_(wikipedia)_again</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://www.example.com/wpsstyle?page=364">http://www.example.com/wpsstyle?page=364</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="https://www.example.com/foo/?bar=baz&amp;inga=42&amp;quux">https://www.example.com/foo/?bar=baz&amp;inga=42&amp;quux</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://Odf.ws/123">http://Odf.ws/123</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid:password@example.com:8080">http://userid:password@example.com:8080</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid:password@example.com:8080/">http://userid:password@example.com:8080/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid@example.com">http://userid@example.com</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid@example.com/">http://userid@example.com/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid@example.com:8080">http://userid@example.com:8080</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid@example.com:8080/">http://userid@example.com:8080/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid:password@example.com">http://userid:password@example.com</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://userid:password@example.com/">http://userid:password@example.com/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://142.42.1.1/">http://142.42.1.1/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://142.42.1.1:8080/">http://142.42.1.1:8080/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://ws/">http://ws/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://ws/">http://ws/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><a href="http://ws/">http://ws/</a></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

These URLs should match (1 → correct)

---

- **URL validation challenge from Mathias Bynens, Dec. 10**
Genectic Programming Operation: Crossover

- Redirecting Edges
- Collapsing Stretches
- One-way Crossovers
Genetic Programming Operation: Mutation

- Positive Mutation: ftp://foo.com
  - Add edge for “f”

- Negative Mutation: http://#
  - Remove “#”

Augmenting mutations  Diminishing mutations
Example:
Regex for URL

```javascript
var re_weburl = new RegExp(
  // Code
);
```

<table>
<thead>
<tr>
<th>Source</th>
<th>Length</th>
<th>True Positive</th>
<th>True Negative</th>
<th>Overall Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>@krijnhoetmer</td>
<td>115</td>
<td>.78</td>
<td>.41</td>
<td>.59</td>
</tr>
<tr>
<td>@gruber</td>
<td>71</td>
<td>.97</td>
<td>.36</td>
<td>.65</td>
</tr>
<tr>
<td>@gruber v2</td>
<td>218</td>
<td>1.00</td>
<td>.33</td>
<td>.65</td>
</tr>
<tr>
<td>@cowboy</td>
<td>1,241</td>
<td>1.00</td>
<td>.15</td>
<td>.56</td>
</tr>
<tr>
<td>@mattfarina</td>
<td>287</td>
<td>.72</td>
<td>.44</td>
<td>.57</td>
</tr>
<tr>
<td>@stephenhay</td>
<td>38</td>
<td>.64</td>
<td>.81</td>
<td></td>
</tr>
<tr>
<td>@scottgonzales</td>
<td>1,347</td>
<td>1.00</td>
<td>.15</td>
<td>.56</td>
</tr>
<tr>
<td>@rodneyrehm</td>
<td>109</td>
<td>.83</td>
<td>.36</td>
<td>.59</td>
</tr>
<tr>
<td>@imme_emosol</td>
<td>54</td>
<td>.97</td>
<td>.74</td>
<td>.85</td>
</tr>
<tr>
<td>@diegoperini*</td>
<td>502</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

**Figure 1:** Representative regular expressions for URLs obtained from http://mathiasbynens.be/demo/url-regex. For every possible solution we show its length, true and false positive rates, and the overall accuracy. The last row is the winner.