Automatic Patch Generation Learned from Human-Written Patches
Conventional Bug-fixing

- A human manually writes a patch to fix the bug
- Slow and tedious
- Uninteresting
- Takes time away from developing new features, or other interesting things to work on
Automated Bug-fixing

- Previous attempts to automate this process have been made
- GenProg uses a genetic algorithm to create patches by trying different mutations of the code
- Low acceptance rate of generated patches (lots of nonsensical suggestions), therefore not very useful in practice
New Technique: PAR

- Pattern-based Automatic program Repair
- PAR uses templates created from actual human-written patches to offer better suggestions
- Better acceptance rate than other automatic patch generation systems (like GenProg)
How does PAR work?

1. Fault Localization

2. Template Based Patch Candidate Generation

3. Patch Evaluation
Line 1920 throws an *Array Index Out of Bound* exception when `getShort(iCode, pc+1)` is equal to or larger than `strings.length` or smaller than 0.
Patch Generated by GenProg

1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
1920+   lhs = ((Scriptable)lhs).getDefaultValue(null);
1921 }  
1922 Scriptable calleeScope = scope;

- Nonsensical Patch
- Patch removed the “strings[]”
- Passes all the given tests
- Developer would not accept the patch
Human-written Patch

1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
1920+   i = getShort(iCode, pc + 1);
1921+   if (i != -1)
1922+     lhs = strings[i];
1923  }
1924 Scriptable calleeScope = scope;

- Checks if the value is not -1
- Do not check for >=0
- Do not check if getShort() < “array length”
Patch Generated by PAR

1918  if (lhs == DBL_MRK) lhs = ...;
1919  if (lhs == undefined) {
1920+   if (getShort(iCode, pc + 1) < strings.length &&
1921+     getShort(iCode, pc + 1) >= 0)
1922+   {
1923+     lhs = strings[getShort(iCode, pc + 1)];
1924+   }
1925  } Scriptable calleeScope = scope;

- Checks if getShort(iCode, pc+1) is less than the array size
- Checks if getShort(iCode, pc+1) >= 0
- Acceptable by developers and passes all the tests
What are the benefits?

- Patches are more acceptable by developers compared to GenProg
- Pattern-based and useful to improve other automatic patch generation
- Useful to generate more program variant in model-based technique
- Generates fix templates from human-written patch patterns
Evaluation!

RQ1 (Flexibility): How many bugs are fixed successfully?

RQ2 (Acceptability): Can PAR generate more acceptable patch compared to GenProg?
Experiment Design:

- 119 real bugs from various open-source projects
- Applied both PAR and GenProg
- Total 23,800 runs (100 run x 119 bugs x 2 approaches)
- Stopped after 8 hours of run = failed patch generation
- Several machines with two hexa-core 3GHz CPU and 16Gb RAM
RQ1: Fixability

- PAR fixed 27 bugs out of 119, while GenProg fixed 16.
- Identifying more fix templates from existing patches might improve fixability further.
RQ2: Acceptability

- H1: Patches generated by PAR and GenProg have no acceptability difference from each other
- H2: Patches generated by PAR have no acceptability difference from human-written patches
- H3: PAR generates more acceptable patches than GenProg
- H4: Patches generated by PAR are more acceptable than human-written patches
RQ2: Acceptability

- Subjects: Computer science students and developers
- Study Design:
  - One of the five bugs fixed by PAR and GenProg
  - Web-Based online survey
RQ2: Acceptability

<table>
<thead>
<tr>
<th>Bugs</th>
<th>Human (Std)</th>
<th>PAR (Std)</th>
<th>GenProg (Std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math #280</td>
<td>1.33 (0.62)</td>
<td>2.27 (0.59)</td>
<td>2.40 (0.83)</td>
</tr>
<tr>
<td>Rhino #114493</td>
<td>2.00 (0.54)</td>
<td>1.33 (0.62)</td>
<td>2.67 (0.72)</td>
</tr>
<tr>
<td>Rhino #192226</td>
<td>1.47 (0.64)</td>
<td>1.67 (0.62)</td>
<td>2.67 (0.72)</td>
</tr>
<tr>
<td>Rhino #217379</td>
<td>1.69 (0.70)</td>
<td>1.50 (0.63)</td>
<td>2.81 (0.40)</td>
</tr>
<tr>
<td>Rhino #76683</td>
<td>2.13 (0.51)</td>
<td>1.07 (0.26)</td>
<td>2.80 (0.41)</td>
</tr>
<tr>
<td>Average</td>
<td>1.72 (0.67)</td>
<td>1.57 (0.68)</td>
<td>2.67 (0.64)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bugs</th>
<th>Human (Std)</th>
<th>PAR (Std)</th>
<th>GenProg (Std)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math #280</td>
<td>1.92 (0.76)</td>
<td>2.00 (0.82)</td>
<td>2.08 (0.95)</td>
</tr>
<tr>
<td>Rhino #114493</td>
<td>1.60 (0.63)</td>
<td>2.40 (0.74)</td>
<td>2.00 (0.93)</td>
</tr>
<tr>
<td>Rhino #192226</td>
<td>2.00 (0.68)</td>
<td>1.79 (0.98)</td>
<td>2.21 (0.80)</td>
</tr>
<tr>
<td>Rhino #217379</td>
<td>1.62 (0.77)</td>
<td>1.69 (0.63)</td>
<td>2.69 (0.63)</td>
</tr>
<tr>
<td>Rhino #76683</td>
<td>1.92 (0.64)</td>
<td>1.23 (0.43)</td>
<td>2.85 (0.38)</td>
</tr>
<tr>
<td>Average</td>
<td>1.81 (0.70)</td>
<td>1.82 (0.80)</td>
<td>2.36 (0.90)</td>
</tr>
</tbody>
</table>

PAR generates more acceptable patch than GenProg.
Discussion Questions

- How useful is it in Practice?
- How will PAR perform on code written in different languages?
- How would a poorly written test suit effect PAR’s performance?
- How can PAR be modified to work for closed source projects as it did for open source projects?
- Can PAR learn a bad way to generate patches since it learns the pattern from human written patches and humans are prone to make mistakes?
- Can PAR technique be used in another domain?
Questions?