Modular and Verified Automatic Program Repair
Key Idea

This project aims to design a tool which would work with an abstract interpreter to automatically generate and suggest repairs to potential code problems at design time.
Research Questions

- Can we automate the process of suggesting code repairs at design time?
Research Questions

- How can we constrict the process to work within a single method?
Research Questions

• How can we be sure that our repairs are genuine?
Contributions

- Verified Repair
  - Formal Definition:

**Definition 1** (Verified repair, improvement). If $\alpha_A(\mathcal{G}_P) \subseteq \alpha_{\delta_{P',P}} \circ \alpha_A(\mathcal{G}_{P'})$ and $\alpha_A(\mathcal{B}_P) \supseteq \alpha_{\delta_{P',P}} \circ \alpha_A(\mathcal{B}_{P'})$, then we say that $\delta_{P,P'}$ is a verified repair for $P$ and that $P'$ is an improvement of $P$. 
Contributions

- What does this mean?
  - This is a definition defined in terms of execution traces. A verified repair is a repair that increases the number of good traces while preserving the current good traces.
Contributions

● Framework to achieve this goal
  ○ Uses Clousot to generate warnings
  ○ Acts on those to resolve:
    ● missing contracts
    ● incorrect locals and objects initialization
    ● wrong conditionals
    ● buffer overruns
    ● arithmetic overflow
    ● incorrect floating point comparisons.
Contributions

● An analysis of the tool’s performance on the .NET framework libraries
  ○ Program was tested on .NET framework libraries
  ○ Detected 54,000 errors in 3 and a half hours.
  ○ Program generated verified repairs for >80% of the warnings generated by their abstract interpreter.
Example Modification

```csharp
void ValidateOwnerDrawRegions(
    ComboBox c, Rectangle updateRegionBox)
{
    if (c == null){
        var r = new Rectangle(0, 0, c.Width); // (*)
        // use r and c
    }
}
```

ccheck suggests c == null be changed to c != null
Examples of usage

```csharp
string GetString(string key){
    var str = GetString(key, null);
    if (str == null){
        var args = new object[1];
        args[1] = key; // (*)
        throw new ApplicationException(args);
    }
    return str;
}
```

ccheck suggests array size be increased or to use index 0
Example modification

IMethodCallMessage ReadArray(
    object[] callA, object handlerObject)
    {
        if (callA == null) return null;
        var num = 0;
        if (NonDet())) num++;
        if (callA.Length < num) throw new SerializationException();
//    here callA.Length >= num
        this.args = (object[])callA[num++];
//    ...
    }

ccheck suggests to use <= instead of <.
Evaluation

Run against several shipped microsoft .NET libraries. These libraries had no contracts when shipped, and used traditional guards in the form of if statements.
## Evaluation

### Results of testing ccheck

<table>
<thead>
<tr>
<th>Library</th>
<th>Methods</th>
<th>Overall Time</th>
<th>Asserts</th>
<th>Validated</th>
<th>Warnings</th>
<th>Repairs</th>
<th>Time</th>
<th>Asserts with Repairs</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.Windows.forms</td>
<td>23,338</td>
<td>62:00</td>
<td>154,863</td>
<td>137,513</td>
<td>17,350</td>
<td>25,501</td>
<td>1:27</td>
<td>14,617</td>
<td>84.2</td>
</tr>
<tr>
<td>mscorlib</td>
<td>22,304</td>
<td>38:24</td>
<td>113,982</td>
<td>103,596</td>
<td>10,386</td>
<td>16,291</td>
<td>0:59</td>
<td>7,180</td>
<td>69.1</td>
</tr>
<tr>
<td>system</td>
<td>15,187</td>
<td>26:55</td>
<td>99,907</td>
<td>90,824</td>
<td>9,083</td>
<td>15,618</td>
<td>0:47</td>
<td>6,477</td>
<td>71.3</td>
</tr>
<tr>
<td>system.data.entity</td>
<td>13,884</td>
<td>51:31</td>
<td>95,092</td>
<td>81,223</td>
<td>13,869</td>
<td>28,648</td>
<td>1:21</td>
<td>12,906</td>
<td>93.0</td>
</tr>
<tr>
<td>system.core</td>
<td>5,953</td>
<td>32:02</td>
<td>34,156</td>
<td>30,456</td>
<td>3,700</td>
<td>9,591</td>
<td>0:27</td>
<td>2,862</td>
<td>77.3</td>
</tr>
<tr>
<td>custommarshaler</td>
<td>215</td>
<td>0:11</td>
<td>474</td>
<td>433</td>
<td>41</td>
<td>31</td>
<td>0:00</td>
<td>35</td>
<td>85.3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>80,881</td>
<td>3:31:03</td>
<td>498,474</td>
<td>444,045</td>
<td>54,429</td>
<td>95,680</td>
<td>4:51</td>
<td>44,077</td>
<td>80.9</td>
</tr>
</tbody>
</table>

---

Note: The table above summarizes the results of the testing process for various libraries, including the number of methods, overall time, number of assertions, validated assertions, warnings, repairs, and time required for repairs. The last column indicates the percentage of assertions with repairs.
Discussion question

How could this program be expanded in scope?

● Fix memory allocation errors, memory leaks.
● Visual studio plugin could automatically generate comments.
Discussion question

How can this tool be applied to other languages?

- Is it limited to statically typed languages?
- Could it use annotations for more languages?
Discussion question

- How complex is the code? Will developers have to write the code to suit the tool?
Discussion question

- Does this encourage developers to follow safe practices, or does it give a false sense of security?
Discussion question

- Are the bugs found in this experiment ones application developers care about?
Discussion question

- Would you use this tool?