Why Use Type Annotation Analysis?

- Lightweight compile time verification
- Extensible
- Similar to an automatic code review
- Annotations are concise

Sharp Checker for C# Features

- Leverage the .NET Compiler Platform to provide feedback within the Visual Studio IDE
- Several type systems implemented: Encrypted, Nullness, and Tainted
- Designed with extensibility in mind
- Subtyping among type annotations
- Warning suppression with assertions
- Type refinement when explicitly comparing with null

Type Annotation Analysis Using the .NET Compiler Platform

Why Use Type Annotation Analysis?

- Powerful extensible framework for Java
- Niceties which make it possible to create new annotated type systems declaratively or procedurally
- Empirical analysis of the ease of use and efficacy of annotation application by uninformed users

JQual
- Type annotation inference instead of type checking
- Type qualifier subtyping
- Opt-in field sensitive analysis
- Context sensitive analysis

Future Work

- Support stubs or another mechanism to annotate libraries
- Expand upon warning suppression behavior to permit ignoring or targeting namespaces, projects, or classes
- Declarative alternatives to procedural mechanisms
- Instantiate Sharp Checker to categorically different type systems such as Units, Inverting, and Lock
- Consider side effects when refining types
- Introduce feature switches for conservative decisions
- Introduce type qualifier polymorphism to allow reuse of instrumented code, as with generics
- Create mechanism for assigning annotations to syntax elements which do not accept C# attributes

Experimental Results and Conclusions

<table>
<thead>
<tr>
<th>Checker</th>
<th>Target Application</th>
<th>Lines of Code</th>
<th>Annotations</th>
<th>Assertions</th>
<th>Limitations</th>
<th>Bugs Discovered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted</td>
<td>RijndaelEncryption</td>
<td>100</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Nullness</td>
<td>SharpChecker</td>
<td>4368</td>
<td>10</td>
<td>12</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Nullness</td>
<td>EventCloud</td>
<td>268144</td>
<td>15</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Tainted</td>
<td>EventCloud</td>
<td>268144</td>
<td>10</td>
<td>20</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

- Applying type annotations requires an investment of time not necessary for lightweight analysis
- Bridges the gap between constraints enforced by the C# type system and those which are desired
- Results come with guarantees
- Conservative by default
- There is always room to permit more use cases which a human would judge to be safe
- The facilities exposed by Roslyn are categorically different than Java batch annotation processing
  - Analyses are initiated continuously
  - Partial programs are analyzed

References


GitHub Repository: https://github.com/tcs1896/SharpChecker