Masters Capstone Project - Type Annotation Analysis Using the .NET Compiler Platform
Student: Theodore Sill
Advisor: Professor Matthew Fluet

This file contains the contents of the Git repository created to manage the source code written for my capstone project.

CSharpQual

This solution contains a diagnostic analyzer and code fix created as part of milestone one. The primary goal was to become more familiar with the .NET Compiler Platform ("Roslyn") by addressing a common programming error. When formatting a string in C# (prior to C# 6.0 which introduces string interpolation) a format string is used which contains replacement tokens. If the number of additional arguments provided doesn't match the number of tokens to replace then a runtime error can occur. This project searches for occurrences of this mismatch and can correct the issue automatically.

EncryptedSandbox

This solution contains a test application which may be loaded in the Visual Studio (VS) experimental hive instance executing SharpChecker. It contains a variety of expressions and statements which exercise "Encrypted" attribute verification. The examples are primarily invocation expressions and assignment statements.

Inherited

This solution contains another test application used to test the rules associated with overriding methods. Here we are also verifying "Encrypted" attribute application. The examples are a variety of method declarations.

SharpChecker

This solution contains the SharpChecker framework, which is the primary deliverable. The purpose is to verify that target code respects the rules of a type annotation system. In its current state it only supports one such system, and expects explicit type annotations. There are comments in the code which identify opportunities for generalizing this approach to support more annotated type systems. The following is a description of the most important classes:

- SharpCheckerBaseAnalyzer – This is the entry point for SharpChecker. It is here that we register with the .NET Compiler Platform by extending the DiagnosticAnalyzer class and decorating our class with the "[DiagnosticAnalyzer(LanguageNames.CSharp)]" attribute. In the "Initialize" method we instantiate our ASTUtilites object (described below), and register actions to be performed when steps in the analysis life-cycle occur.
- ASTUtilities – This class is concerned with collecting the explicit type annotations, which occur in source code as attributes. The annotations are stored in a symbol dictionary where the key is a syntax node, and the value is a collection of attribute names.
- SCBaseSyntaxWalker – This class extends the CSharpSyntaxWalker class which allows us to hook into an implementation of the visitor design pattern provided by the .NET Compiler Platform. Using this mechanism we visit all of the syntax nodes which we are concerned with as we walk over the AST and verify that the appropriate effective type annotation is present. When not

present, we register a diagnostic as a side effect, and this diagnostic is presented to the user in the Visual Studio IDE.