Type Annotation Analysis Using the .NET Compiler Platform

Theodore Sill with Professor Matthew Fluet

It ain't what you don't know that gets you into trouble,
It's what you know for sure that just ain't so.

-Mark Twain

Why Use Type Annotation Analysis?

```
8     Velocity vel = new Velocity(5.0, 2.0);
9     double speed = vel.GetVelocity();
10  }
11
12  public class Velocity{
13    private double meters = 0.0;
14    private double seconds = 0.0;
15    public Velocity(double m, double s)
16    {
17        this.meters = m;
18        this.seconds = s;
19    }
20    public double GetVelocity()
21    {
22        if(seconds != 0.0)
23        {
24            return meters / seconds;
25        }
26        return -1.0;
27    }
28  }
```

```
8     double meters = 5.0;
9     double seconds = 2.0;
10    double speed = meters;
```

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

```
8     @m double meters = 5.0 * UnitsTools.m;
9     @s double seconds = 2.0 * UnitsTools.s;
10    @mPERs double speed = meters / seconds;
```

```
8     @m double meters = 5.0 * UnitsTools.m;
9     @s double seconds = 2.0 * UnitsTools.s;
10    @mPERs double speed = meters;
```

Executed command: javac -processor org.checkerframework.checker.units.UnitsChecker afile.java

No.	Type	Description
1	error	Error: [assignment.type.incompatible] incompatible types in assignment. found : @m double required: @mPERs double

Existing Frameworks

Checker Framework

- 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 uninitiated 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, Interning, 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

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

```
public static void SendText([Encrypted] string unencrypted) { }
public static string UnencryptedProp { get; set; }
static void Main(string[] args)
{
    SendText(UnencryptedProp);
}
```

string Program.UnencryptedProp { get; set; }

Attribute application error Encrypted

```
[Untainted]
private static string GetCustomers = "Select * from dbo.Customers";
static void Main(string[] args)
{
    var dbAccess = new DatabaseAccess();
    dbAccess.ExecuteNonQuery(GetCustomers);
    dbAccess.ExecuteNonQuery(ReadUserInput());
}
[return: Tainted]
public static string ReadUserInput()
{
```

string Program.ReadUserInput()

Attribute application error Untainted

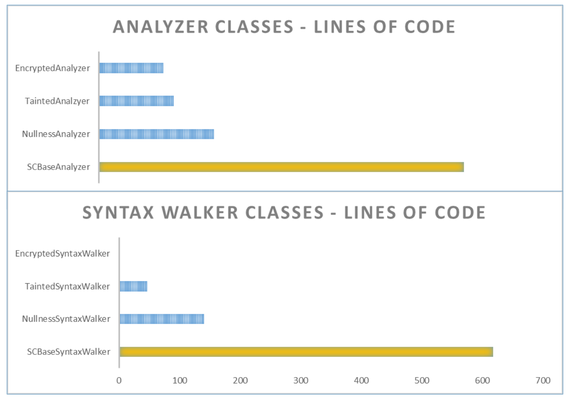
```
Debug.Assert(conditional.WhenTrue != null, "conditional.WhenTrue:NonNull");
if(expectedAttr != null)
    VerifyExpectedAttrsInSyntaxNode(expectedAttr, conditional.WhenTrue);
```

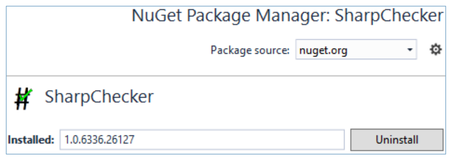
```
internal virtual void VerifyExpectedAttrsInSyntaxNode([NonNull] List<string> expectedAttributes, [NonNull] SyntaxNode node)
{
    //If there are no expected attributes, or there is no node to analyze then short circuit
    if(expectedAttributes == null || expectedAttributes.Count() == 0 || node == null) { return; }
```

Experimental Results and Conclusions

Checker	Target Application	Lines of Code	Annotations	Assertions	Limitations	Bugs Discovered
Encrypted	RijndaelEncryption	100	2	2	0	0
Nullness	SharpChecker	4368	10	12	3	2
Nullness	EventCloud	268144	15	2	1	0
Tainted	EventCloud	268144	10	20	3	0

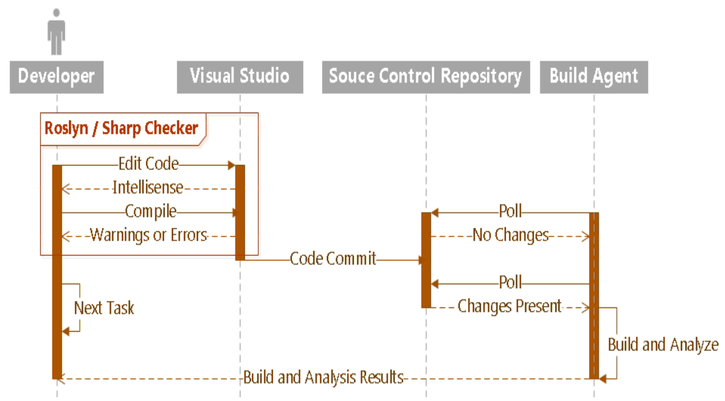
- Applying type annotations requires an investment of time not necessary for lightweight analysis
- Bridges the gap between constraints enforced by the C# type system and those which are desired
- Results come with guarantees
- Conservative by default
- There is always room to permit more use cases which 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





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

Analysis Life Cycle



.NET Compiler Platform ("Roslyn")

Compiler API

- Syntax Tree
- Symbol
- Binding and Flow
- Emit

Highlights

- Redesigned and written in C#
- Open source
- Immutable data structures
- Efficient managed code

Diagnostic API

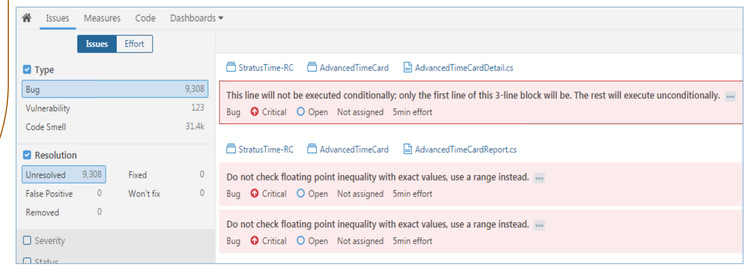
- IDE Feedback
- Code Fix
- Refactoring

Workspace API

- Solutions, Projects, and Documents
- Formatting
- Find All References

Traditional C# Static Analysis

- Lint tools – Some stand alone tools (jslint, pylint, etc.) but many full featured tools like Sonar or Fortify include this type of syntax driven analysis
- FxCop – Partially ported to the .NET Compiler Platform
- SonarQube – Pictured below, features a Roslyn plugin



References

[1] Matthew M. Papi, Mahmood Ali, Telmo Luis Correa, Jr., Jeff H. Perkins, and Michael D. Ernst. Practical pluggable types for java. In Proceedings of the 2008 International Symposium on Software Testing and Analysis, ISSTA '08, pages 201–212, New York, NY, USA, 2008. ACM

[2] David Greenfieldboyce and Jeffrey S. Foster. Type qualifier inference for java. In Proceedings of the 22Nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications, OOPSLA '07, pages 321–336, New York, NY, USA, 2007. ACM.