Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

What did I do last week(s)?

- Focused on implementation:
  - Learned basics of QuickGraph
  - Class Hiearchy Graph
  - Virtual method resolution
  - CHA-based call graph
  - Tested MoonWalker
- Skimmed over the „Bensalem“ and „CalFuzzer“ papers

QuickGraph

- C# library, MS-PL license
- Principles borrowed from Boost.Graph in C++
- Data structures
- Algorithms
- Serialization
- Visualization
- Generic – algorithm can work on graphs represented by different data structures

QuickGraph (contd.)

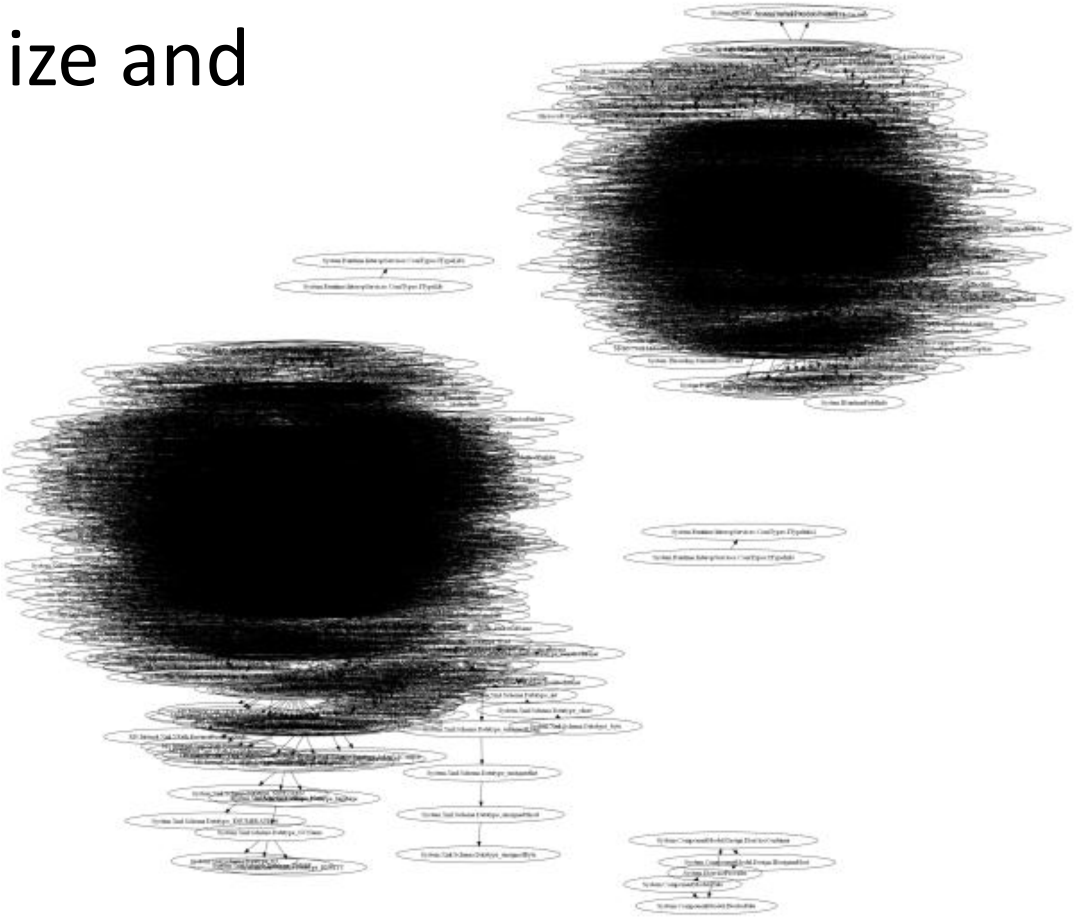
- Graph data structures
  - Traditional - Adjacency list, Incidence matrix, ...
  - Dynamic - Callbacks generate edges on the fly
  - Compressed sparse row
  - Mutable, Immutable
- Graph algorithms
  - Search, Shortest path, Connected components, Strongly connected components, Eulerian trails, Page Rank and many others

Class Hierarchy Graph

- Graph showing type inheritence of a program
- Generated from single root method and recursively for all referenced classes and thier methods (a weak form of rapid type analysis)
- Class hierarchy graph for the program that itself generates the graph has about 4700 vertices, mostly framework classes

Class Hierarchy Graph

- Difficult to visualize and thus verify



Virtual method resolution

- Objective: Given a virtual method reference and class hierarchy graph, return all the possible overrides of the given method
- Turns out to be a bit harder than I expected due to some CIL byte code features that I wasn‘t aware of (overriding method with different name)
- Can yield a list as large as the CH graph for methods such as System.Object.ToString or System.Object.GetType

CHA-based call graph

- Static call graph generated from a root method with the help of CH graph and virtual method resolution
- Generating the graph in advance is very time-consuming due to methods such as ToString
- Generating the edges on the fly is feasible and may be adaquate for some of the graph algorithms, needs to be evaluated

Moonwalker

- Model checker for Mono („the other .NET implementation“)
- Very incomplete, depends on the Mono runtime
- Fails badly even on the simplest programs
- Bugs in both implementation of the threading constructs as well as in the instruction interpretation (eg. type cast of a delegate yields incorrect results or crashes the tool)

CalFuzzer

- Adjusting thread schedules to simulate deadlocks or data race conditions
- Adready implemented for .NET and still subject of research
  - http://research.microsoft.com/en-us/projects/chess/

What do I plan for next week(s)?

- Finish the bits of implementation I didn‘t manage to complete this week, ie. encapsulate the call graph implementation
- Study the algorithms used for aliasing analysis
- Rewrite L.O.V.E. to take advantage of the virtual method resolution and test the speed and bottlenecks