Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

What did I do last week?

- Wired-up all the bits of the interprocedural analysis and modified the prototype to produce lock order graph using the William‘s approach
- Save the resulting lock order graph to file for manual inspection using yEd
- Use Tarjan‘s strongly connected components algorithm to find and report cycles

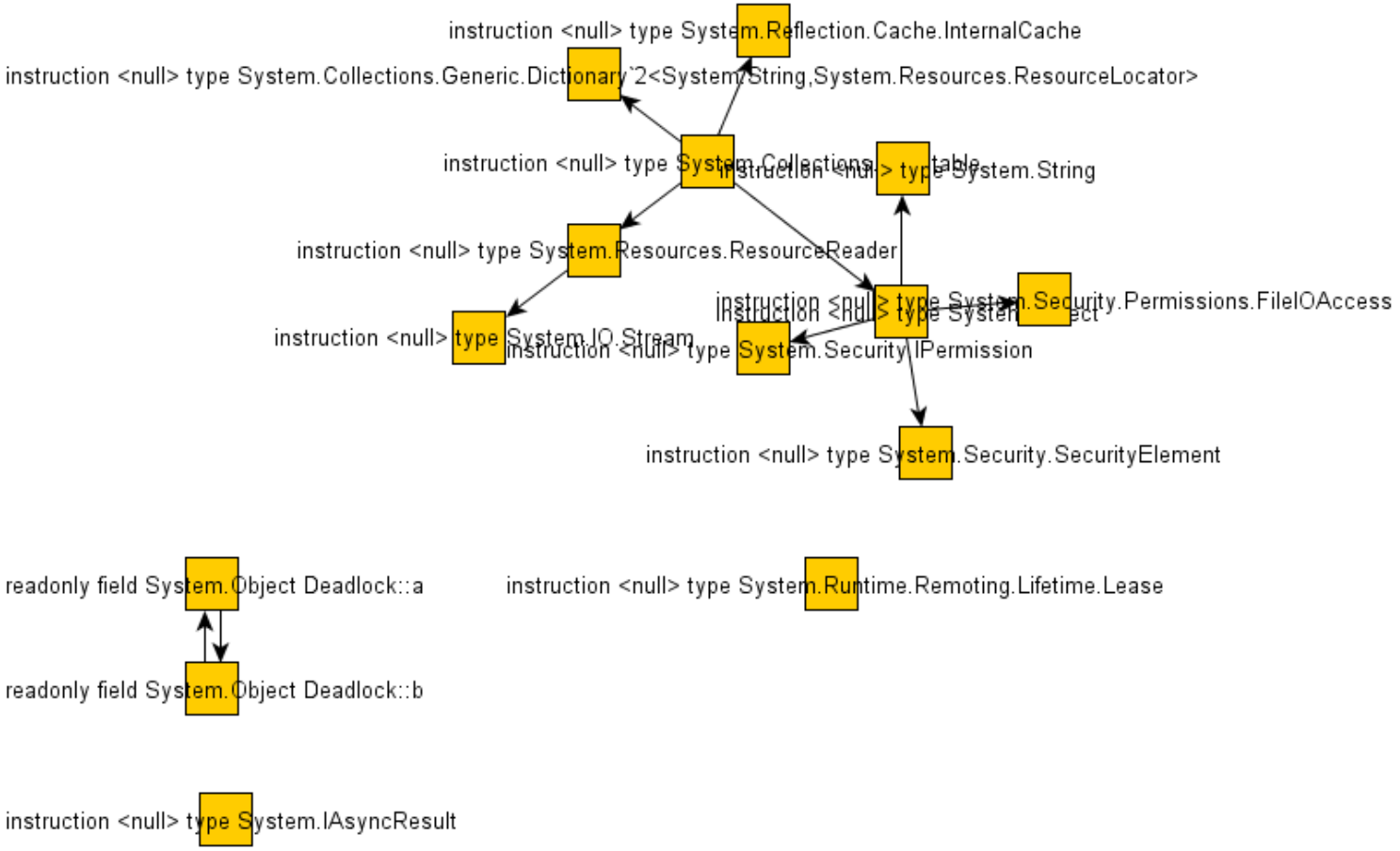
Results (so far)

- Analyzing basic example takes about 6 seconds, most of the time is spent in analyzing .NET framework internals
- Missing aliasing information causes misinterpreted Monitor.Exit calls, which is partially fixable by hacks described in the ECOOP 2005 paper
- Not enough information is recorded to pinpoint where the error actually happened

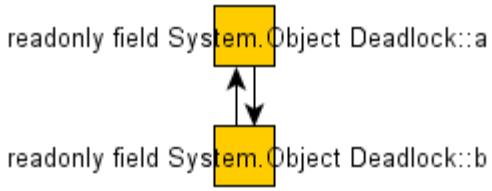
Lock Analysis: Basic example

```
public class Deadlock
{
        static readonly object a = new object();
        static readonly object b = new object();

        public static void FunctionA()
        {
                lock (b)
                {
                        lock (a)
                        {
                        }
                }
        }

        public static void FunctionB()
        {
                lock (a)
                {
                        lock (b)
                        {
                        }
                }
        }

        public static void Main()
        {
                Thread thread1 = new Thread(FunctionA);
                Thread thread2 = new Thread(FunctionB);
                thread1.Start();
                thread2.Start();
        }
}
```

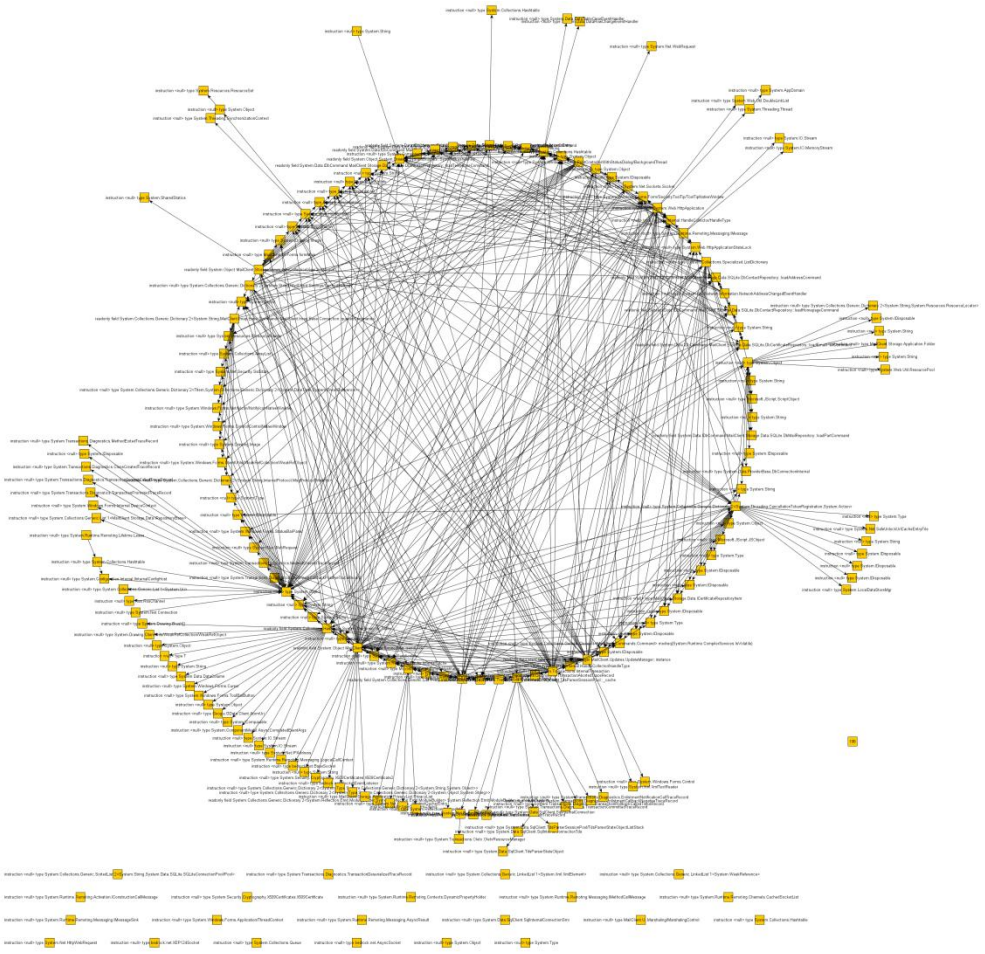
Lock Analysis: Basic example (cont.)



Lock Analysis: Basic example (cont.)



Lock Analysis: Real-world application



What do I plan to do next week?

- Analyze the results on a large scale application and try to pin-point / fix mistakes in the implementation
- Write up a summary that covers up what are locks, deadlocks, their representation in .NET and what we are trying to find