Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

Week 2: What did I do?

- Studied the following approaches:
  - Petri net
  - „Intel“ method („Effective Static Deadlock Detection“ by Mayur et al.)
- Updated code in SVN to include early prototypes and simple tests

Petri net

- Describe states and transitions, analysis conducted on state space
- Pros:
  - Easy to model guard locks
  - Easy to describe unpaired locks spanning multiple methods
  - Possible to model more complex locks, such as ReaderWriterLock (multiple readers or single writer)
- Cons:
  - Costly analysis based on deadlock-preserving reductions
  - How to get back „counterexamples“ if deadlock is detected?
  - How to represent reentrant locks?

„Intel“ method

- Combination of several static analyses (Call graph, Aliasing analysis, Thread escape analysis, May happen in parallel analysis)
- Proves 6 deadlock properties for each (ta,la1,la2,tb,lb1,lb2), four of these conditions are proved soundly:
  - Reachable
  - Aliasing
  - Escaping
  - Parallel
  - Non-reentrant
  - Non-guarded
- Extensible for different lock types
- Detects only deadlocks between two threads

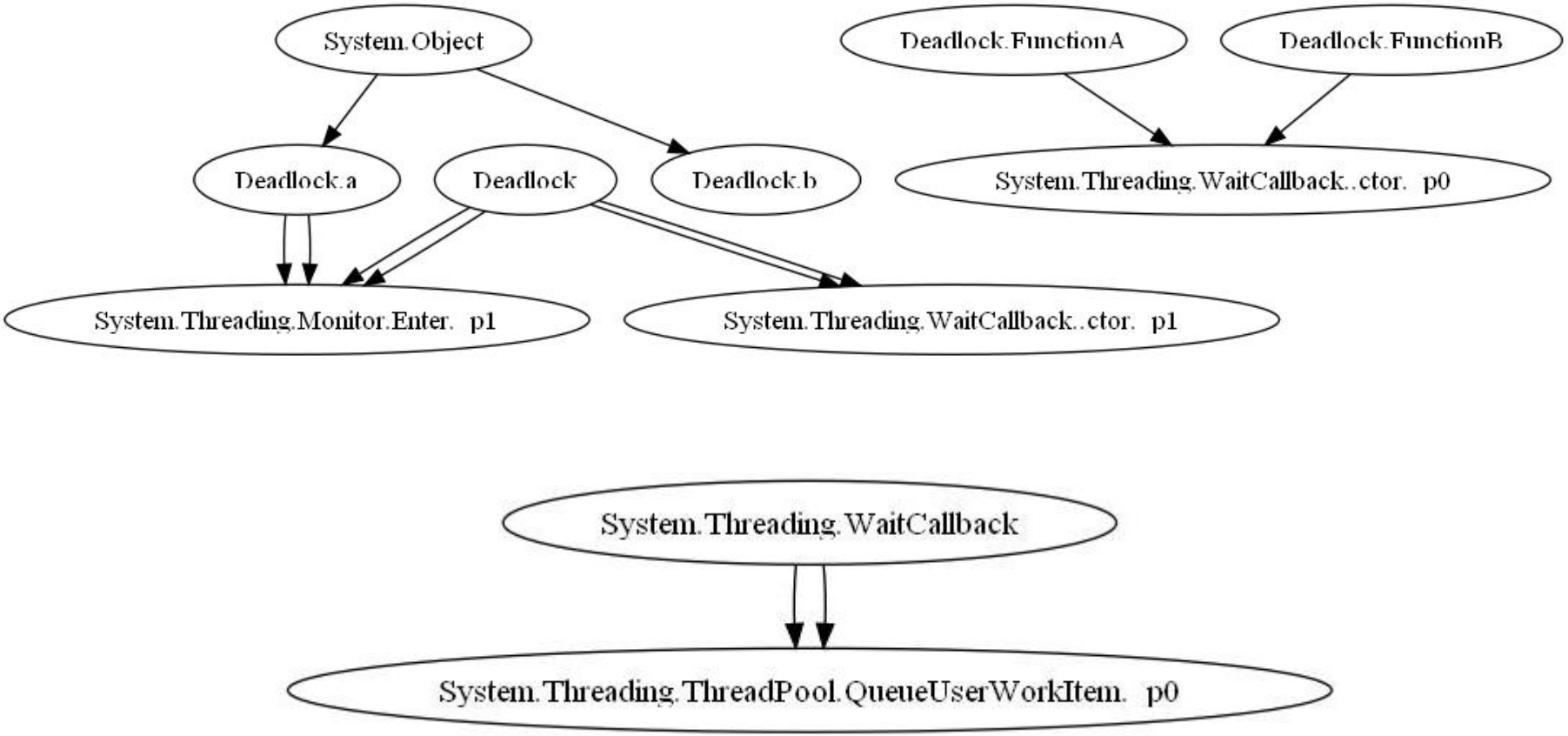
JChord – Implementation of „Intel“ method

- http://code.google.com/p/jchord/
- Library implemention in C / Java / Datalog
- Open source, new BSD license
- Possible use for prototyping
- As many other analyses works on three address code
  - Needs implementation of stack to TAC conversion

Code in SVN

- https://svn.assembla.com/svn/nodeadlock/
- Tests
  - Simple programs showing deadlocks
- Mono.Cecil
  - External library for parsing .NET assemblies
- StaticAnalysis library
  - Implementation of Control-Flow Graph
  - Place for reusable static analysis blocks (call graph, may-alias)
- LovePrototype
  - Implementation of original tool that detects strongly connected components in combined call / lock graph\
  - Updated to use the control-flow analysis in the library

One more thing...



Week 2: What do I plan to do?

- Implement simple call graph construction using Class Hiearchy Analysis
- Further study the Petri net and „Intel“ approaches
- Comprehend the „Automated deadlock detection in synchronized reentrant multithreaded call-graphs“ paper by Frank S. de Boer and Immo Grabe