Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

What did I do last week?

- Interprocedural analysis using callee summaries and work list approach for recomputing recursive chains
  - Implemented the basic algorithm
  - Fixed several test cases and tested the underlying analysis on subset of the .NET Framework
    - Fixed bugs in emulation of stack behavior
    - Fixed bugs in instruction interpretation (casting, array handling)

Williams vs. my program

- Intraprocedural data-flow analysis
  - Several instructions probably still have incorrect implementation
  - Comparing of states is incomplete, ie. each cycle is processed only twice instead of reaching an endpoint (~ 1 week)
- Interprocedutal data-flow analysis
  - Basic framework is in place
  - Incomplete implementation of merging callee lock order graph into caller's lock order graph (~ 1 week)
- Postprocessing
  - Not done, but easy enough (~ 1 week)

Williams vs. my program (cont.)

- Optimizations leading to less false positives
  - Detecting unaliased fields (~ 1 - 2 weeks)
  - Detecting readonly fields (~ 1 week)
  - Callee/caller type optimization (not investigated yet)

.NET 4 construct problem

- .NET 4 introduced a new "lock" construct
  - lock (a) { ... } translates to
  - try {
    Monitor.Enter(a, out acquiredA);
    ...
    }
    finally {
    if (acquiredA) Monitor.Exit(a);
    }
  - Requires path-sensitive data-flow analysis for proper handling of the construct. Current workaround is to assume that when merging branches the one with least locks is the correct one.

What do I plan to do next week (s)?

- Get the interprocedural analysis in the "L.O.V.E." prototype to generate first lock order graph form the William's structures.