Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

What did I do last week?

- Replace the ad-hoc state structures with the HeapObjects and state structure resambling the ones defined by Amy Williams
  - Partially done, the code using the new structure can be compiled and executed, but the computed results are not used yet
- Use data-flow work list algorithm to compute the per-method lock graphs
  - Same as above. Tested on the basic tests and several bugs in static analysis code were fixed, particularly in the control-flow graph computation and processing of data-flow equations
- Implemented exception block reconstruction in control-flow graph and iteration over *finally* blocks
  - Required for proper treatment of Monitor.Exit calls, which the compiler puts inside *finally blocks*

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

- Start adapting the „L.O.V.E.“ prototype
  - Replace the ad-hoc state structures with the HeapObjects and state structure resambling the ones defined by Amy Williams
  - Use data-flow work list algorithm to compute the per-method lock graphs
  - Implement better resolution of virtual methods using Class Hiearchy Graph (currently the type of the called object is not considered in the implementation, but it can significantly reduce the number of possible called methods for eg. System.Object.ToString)
  - Long term: Use the above building block to compute interprocedural lock order graph that takes reentrancy into account