Detecting deadlocks using static analysis in .NET

Filip Navara

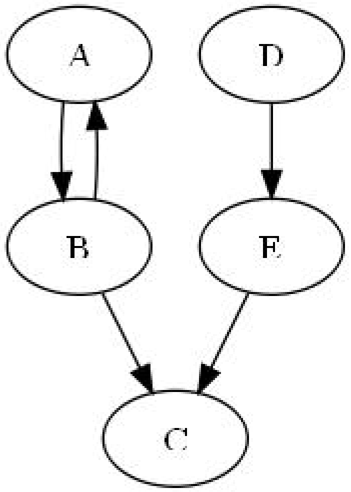
filip.navara@gmail.com

What did I do last week?

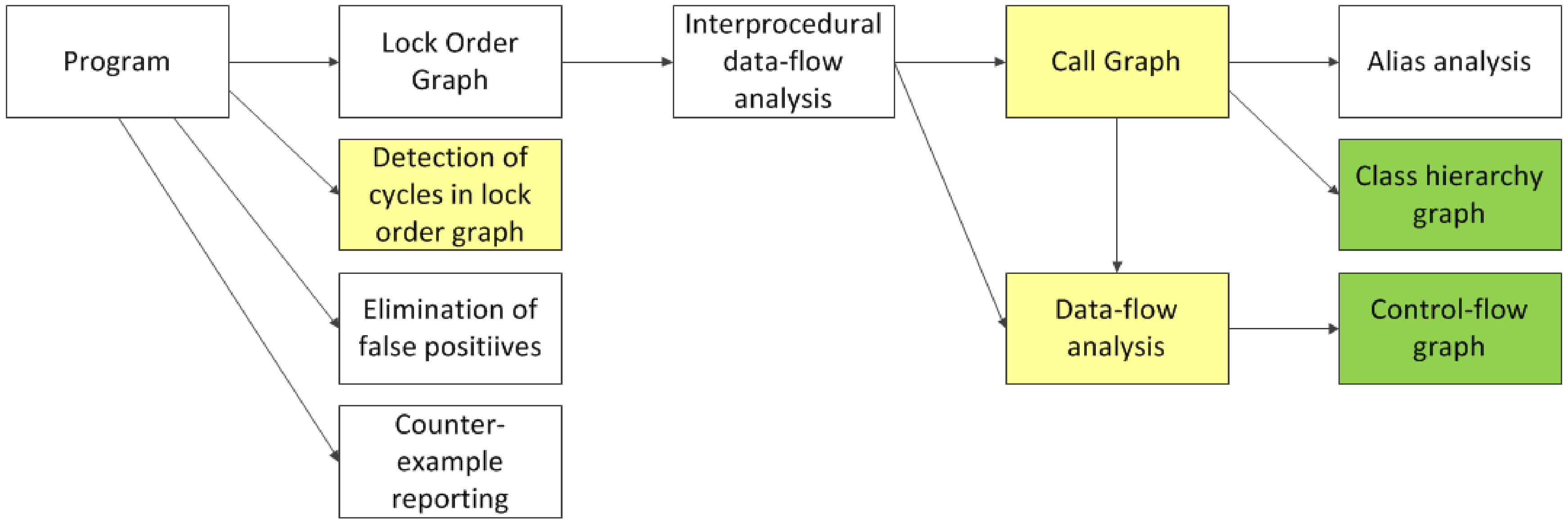
- Got the spanning tree generation working, so the graphs can be visualized using Walrus now
- Tested the basic dataflow compution and roworked some of the helper routines to be use better representation (ie. the flow function is now called per instruction instead of per basic block, ...)

Walrus

- The patological case for generating spanning tree using BFS:
  - The BFS is executed from „roots“, where „root“ is defined as vertex with no incomming edges
  - The A <-> B cycle on the right was never considered a „root“ even though it can‘t be reached by any edges outside of the cycle



Code plan


Program
Lock Order Graph
Detection of cycles in lock order graph
Elimination of false positiives
Counter-example reporting
Interprocedural data-flow analysis
Call Graph
Data-flow analysis
Alias analysis
Class hierarchy graph
Control-flow graph

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 Hieararchy Graph (currently the type of the called object is not considered in the implementation, but it can significantly reduce the number of possible called method for eg. System.Object.ToString)
  - Long term: Use the above building block to compute interprocedural lock order graph that takes reentrancy into account