Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

What did I do last week?

- Rewrite the whole merging of caller/callee lock graphs to be non-recursive.
  - Also fix bug where certain edges where ignored and thus the merged lock graph was incomplete.
  - The fixed recursive version was too slow for merging graphs of 400+ edges, thus the rewrite was necessary.

What did I do last week?

- Add several aliasing hacks.
    - When emulating Monitor.Exit and the top lock is lock on field make sure that we release the lock even though the symbolic objects are not identical.
    - Make sure that UnaliasedFieldHeapObject are really treated as unaliased (hack in LockAcquisition.Equals).

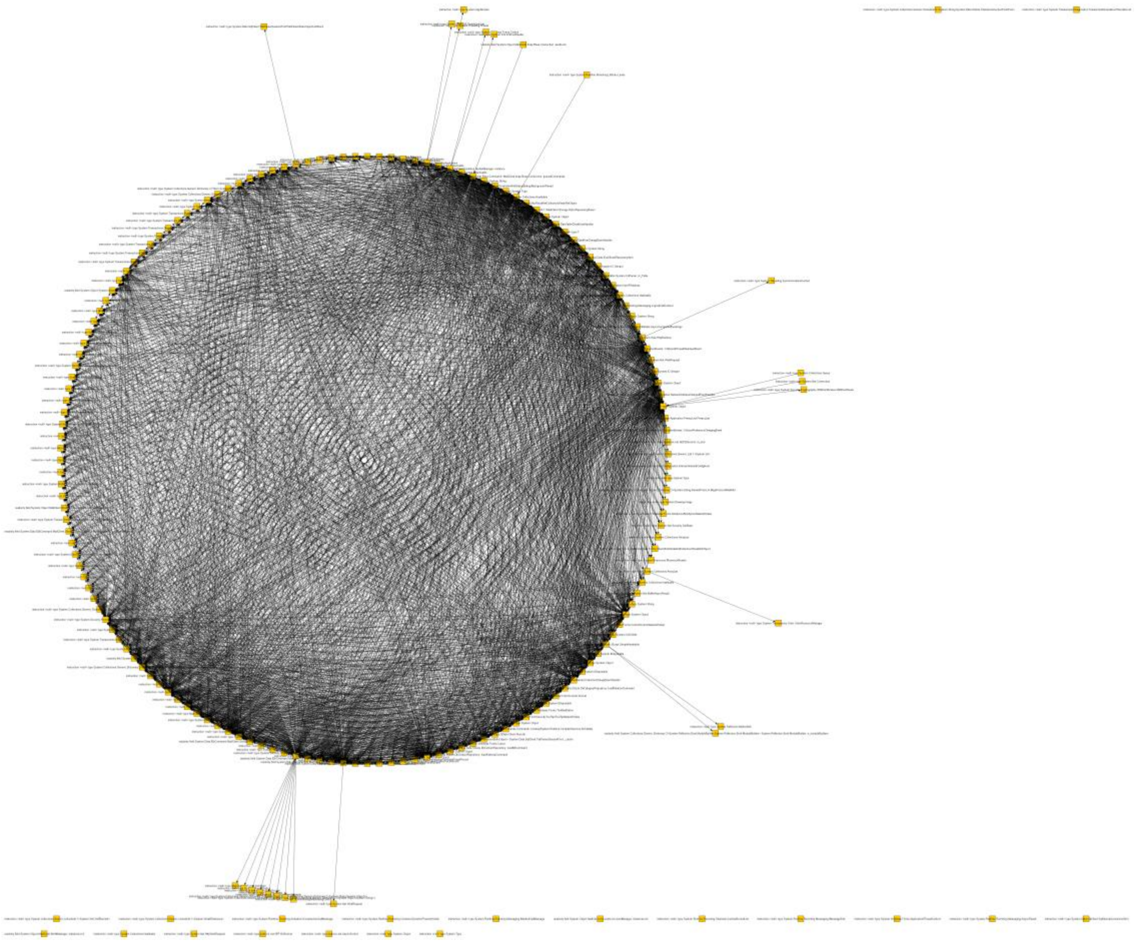
What did I do last week?

- Do not analyze the methods of System.Threading.Monitor type, we emulate them anyway.
- Rewrite the Tarjan implementation to report locks instead of edges since the older algorithm didn't work correctly and reported loops where there were none.

What did I do last week?

- Started writing the section on deadlocks for thesis paper...
- Updated the implementation to track more information that can help with understanding the results
  - Not commited yet
  - Raises the memory usage about 2x

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
- Finish the summary that covers up what are locks, deadlocks, their representation in .NET and what we are trying to find