Detecting deadlocks using static analysis in .NET

Filip Navara

filip.navara@gmail.com

What did I do last week?

- Investigated current implementations and papers about data-flow analyses
  - Frameworks
  - Applications
  - Interprocedural analysis

Intraprocedural data-flow analyses

- Liveness analysis
  - Determines for each statement which variables are used beyond that statement
  - Start with an empty set of live variables
  - Walks the control-flow graph backwards, mark every used variable as live for the given statement and propagate the information to next statement
  - When two disjoint control-flow graph paths reach a common node do an union of the live variable sets

Intraprocedural data-flow analyses

- Reaching definitions
  - Determines for each assignment statement up to which statement the assigned definition is unchanged (used for constant propagation, loop invariant motion)
  - Walks the control-flow graph forward

Intraprocedural data-flow analyses

- The above two algorithms can be solved using the same method!
- Both walk the control-flow graph, one in forward direction, the other in backward direction
- Both operate in a specific domain with a set of flow values in the given domain
- Both specify a merge operation over two paths

Work-List Algorithm for IDFA

```
for each node n
        in[n] = υ; out[n] = υ
worklist = {entry node}
while worklist not empty
        Remove some node n from worklist
        out' = out[n]
        in[n] = ∩ out[p]
        out[n] = transfer(in[n], n)
        if out[n] ≠ out'
                for each s ∈ succ[n]
                        if s ∉ worklist, add s to worklist
```

May vs. Must

- May identifies possibilities
  - Initial guess
    - Empty set
  - Transfer function
    - Add everything that might be true
    - Remove only facts that are guaranteed to be false
  - Merge function
    - Union
- Must implies a guarantee

Generalization of the IDFA

- Forward vs. Backward
- Transfer function (also called flow function)
- Meet operator (also called merge operator)
- Flow values (also called „facts“)
- If the domain is finite and the transfer function is monotonic then the work-list algorithm is guaranteed to reach a fix-point and finish
- Called „lattice framework“ or „monotone framework“ in literature

Interprocedural analysis

- Flow-sensitive vs. flow-insensitive
- Context-sensitive vs. context-insensitive
- Path-sensitive vs. path-insensitive
- Top-down vs. bottom-up

Flow sensitivity

- Flow-sensitive analysis
  - Computes one answer for every statement
  - Requires iterative data-flow analysis
- Flow-insensitive analysis
  - Ignores control flow
  - Computes one answer for every method
  - Can be computed in linear time
  - Less accurate than flow-sensitive

Context sensitivity

- Context-sensitive analysis (also called polyvariant analysis)
  - Re-analyzes callee for each caller
- Context-insensitive analysis (also called monovariant analysis)
  - Perform one analysis or method independent of callers

Path sensitivity

- Path-sensitive analysis
  - Computes one answer for every execution path
  - Practically a model checking approach
- Path-insensitive analysis
  - Much faster

Top-down vs. Bottom-up

- Top-down
  - Summarizes information from caller for callees
- Bottom-up
  - Summarizes information form calles for callers

Solving IPA: Supergraphs

- Combine control-flow graphs of all methods using a call graph and produce a control-flow supergraph
- Work-list algorithm works unchanged
- Context-insensitive
- Flow-sensitive
- Potentially slow, each call creates a cycle

Solving IPA: Brute force

- Use an invocation graph, which distinguishes all calling chains
- Re-analyze callee for all distinct calling paths
- Pro: precise
- Cons: exponentially expensive, recursion is tricky

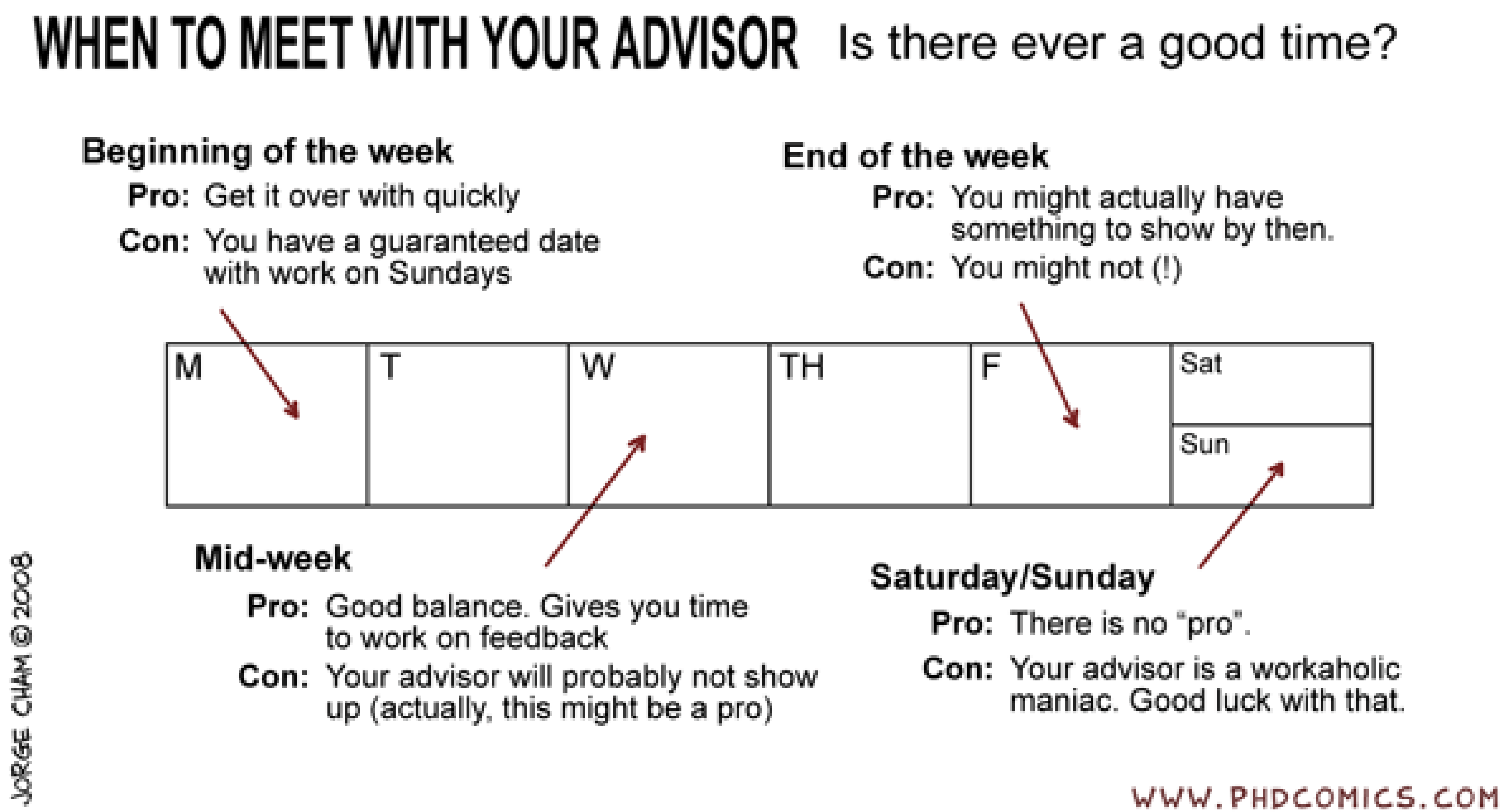
Solving IPA: Call Graph + IDFA

- Summarize effect of called method for callers (eg. compute IDFA for called method and use out[exit node])
- Use work-list algorithm on the call graph
- Context-insensitive, flow-sensitive
- Walking the call graph:
  - Recurisive method calls form strongly connected components
  - All other methods can be analyzed individually in a topological order (top-down) or reverse topological order (bottom-up)

How is this all related to deadlocks?!

- May-alias, must-alias, escape analyses can be defined as IPA
  - http://www.cis.upenn.edu/~cis570/slides/lecture10.pdf
  - http://www.cis.upenn.edu/~cis570/slides/lecture12.pdf
  - http://www.cis.upenn.edu/~cis570/slides/lecture13.pdf
  - http://www.cis.upenn.edu/~cis570/slides/lecture17.pdf
- „Static Deadlock Detection for Java Libraries“ (ECOOP 2005) uses IPA to detect deadlocks:
  - Context-insensitive, flow-sensitive
  - Defined in the terms of „lattice framework“

Obligatory comic strip



What do I plan for next week(s)?

- Write down a draft of thesis text that describes the basic concepts of static analyses (control-flow graph, data-flow analysis, interprocedural analysis)
- Define what is needed for interprocedural analysis framework and design an interface for defining such analyses