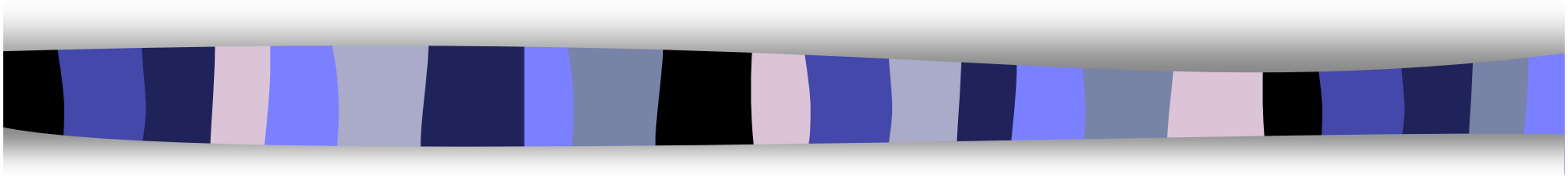
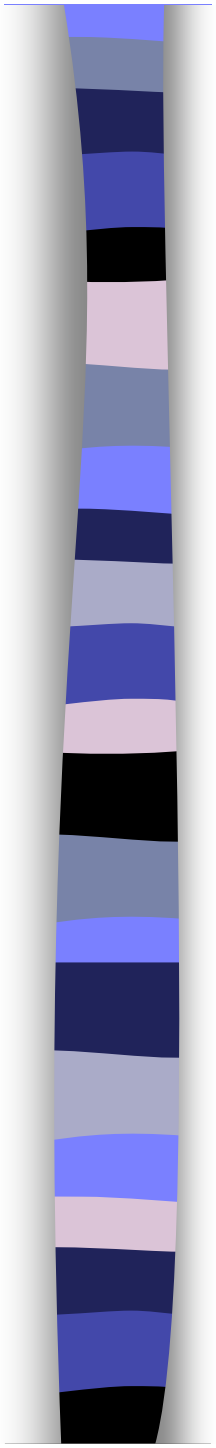
Perracotta: Mining Temporal API Rules from Imperfect Traces

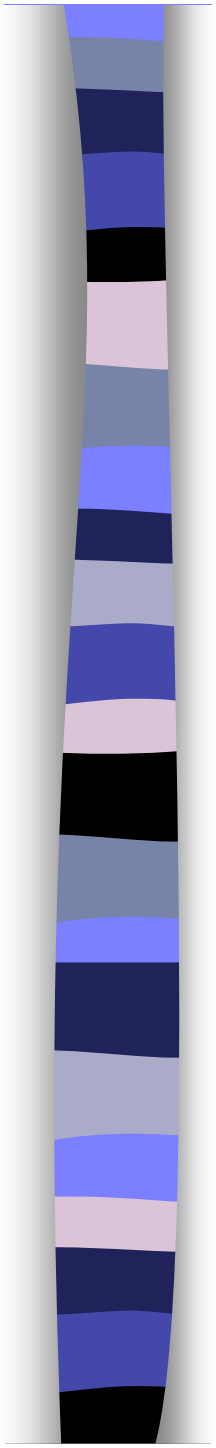


Authors: J. Yang, D. Evans, U of Virginia
D. Bhardwaj, T. Bhat, M. Das, Microsoft
Presented by: Elena Zheleva
Date: November 28, 2006



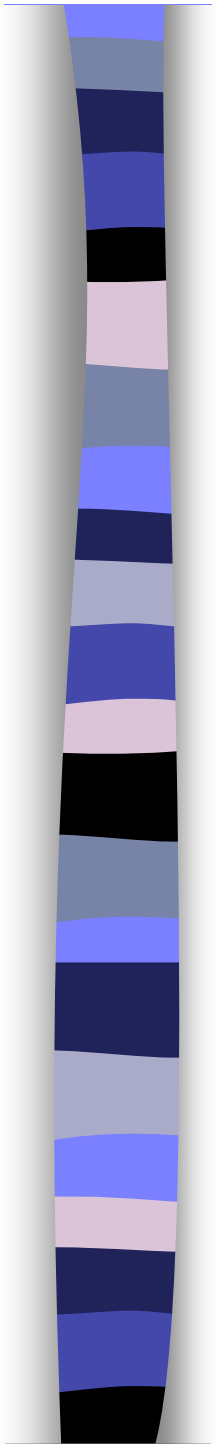
Problem Statement

- Software specifications can change or be non-existent
- Useful for:
  - Verification
  - Testing
  - Maintenance
- Specification inference and scalability
- Focus: Temporal properties and imperfect code



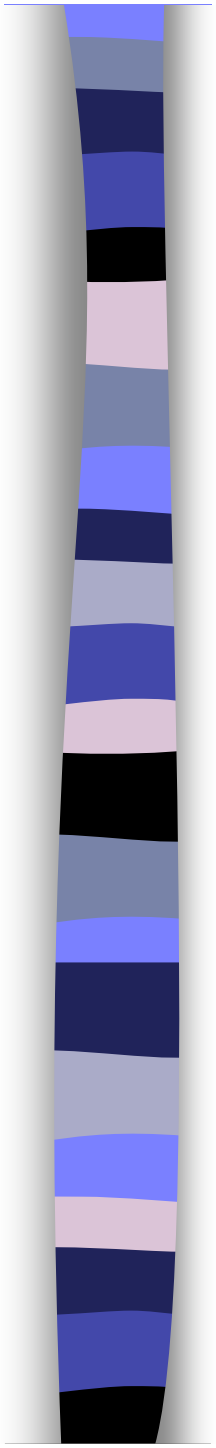
Overview

- Background
  - Static specification inference
  - Dynamic specification inference
- Solution
  - Approximate inference
  - Contextual properties
  - Dynamic inference
- Experiments



Background: Static Inference

- Static analysis of program text
- Extract temporal properties
  - Based on a set of predefined templates
- Disadvantages
  - Requires source code
  - Current research: only local properties



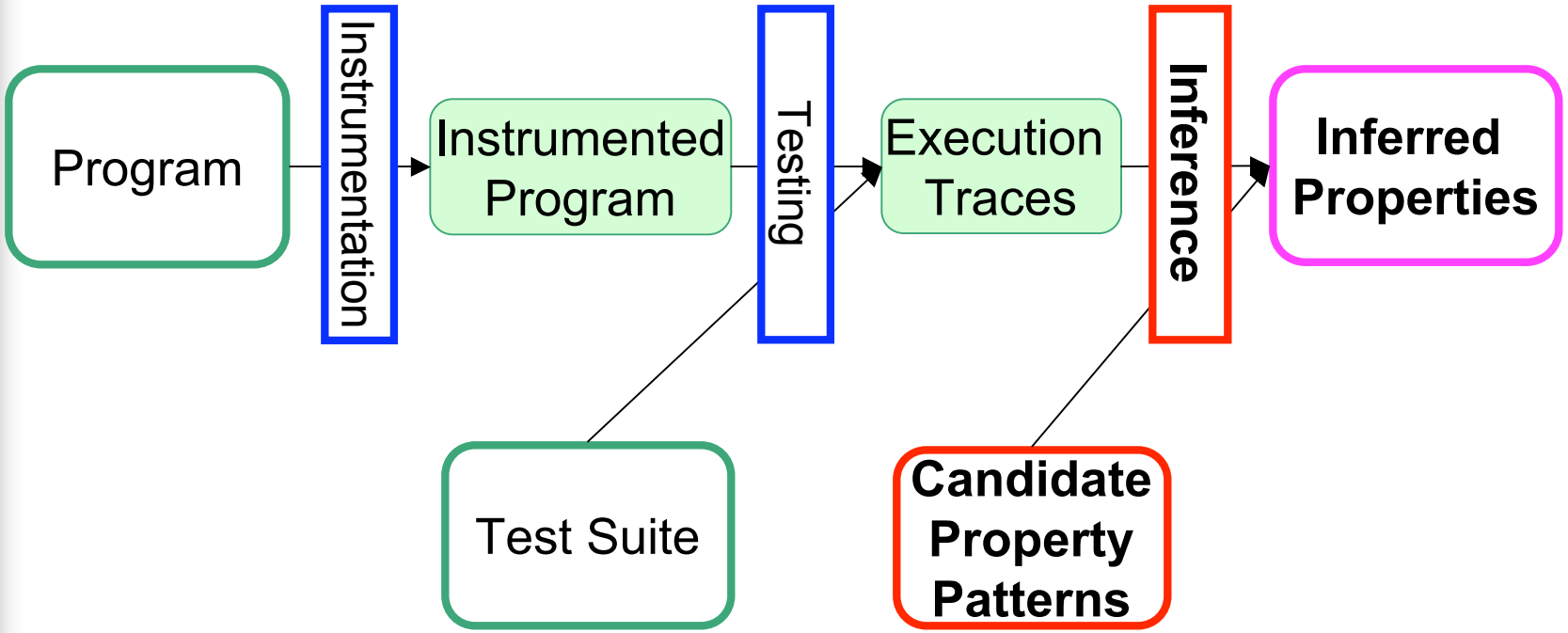
Background: Dynamic Inference

- Analysis of program execution trace
- Create state machines:
  - From particular events (if all: NP-hard)
- Related work:
  - Value-based invariant inference
  - Limited to one class
  - Recover thread model

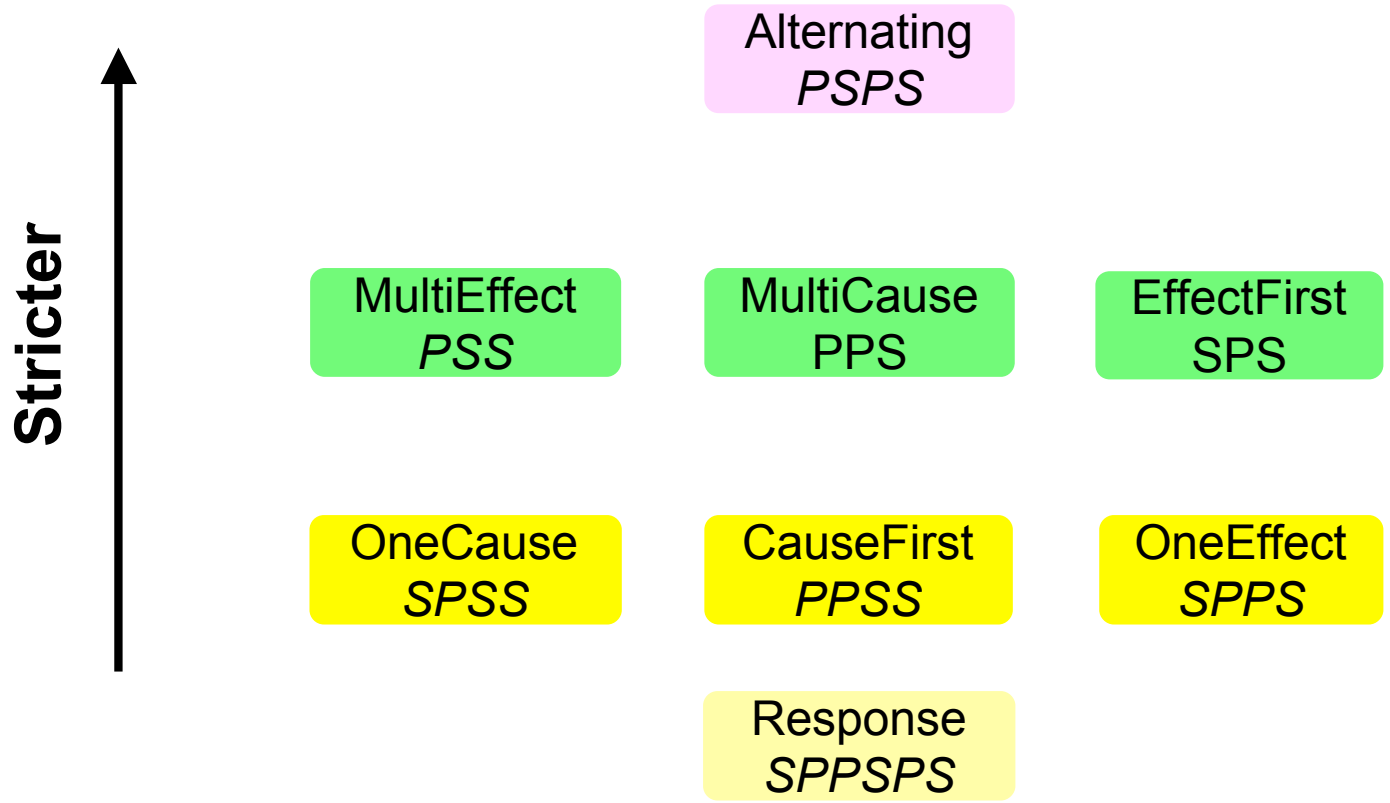
Background: Dynamic Inference

- Challenges in dynamic inference of temporal properties
  - Scalability
  - Imperfect traces
  - Missing context information
  - Finding the interesting properties
- Relationship between 2 or 3 events

Solution


Program
Instrumentation
Instrumented Program
Testing
Execution Traces
Inference
Inferred Properties
Test Suite
Candidate Property Patterns

Solution: Property Patterns



Solution: Property Patterns

- Alternating pattern
  - (PS)*
  - P and S represent events

Trace:

Lock.acq
Lock.rel
Lock.acq
Lock.rel

Possible event pairs:

(Lock.acq Lock.rel)* ✓

(Lock.rel Lock.acq)* ✗

Solution: Approximate Inference

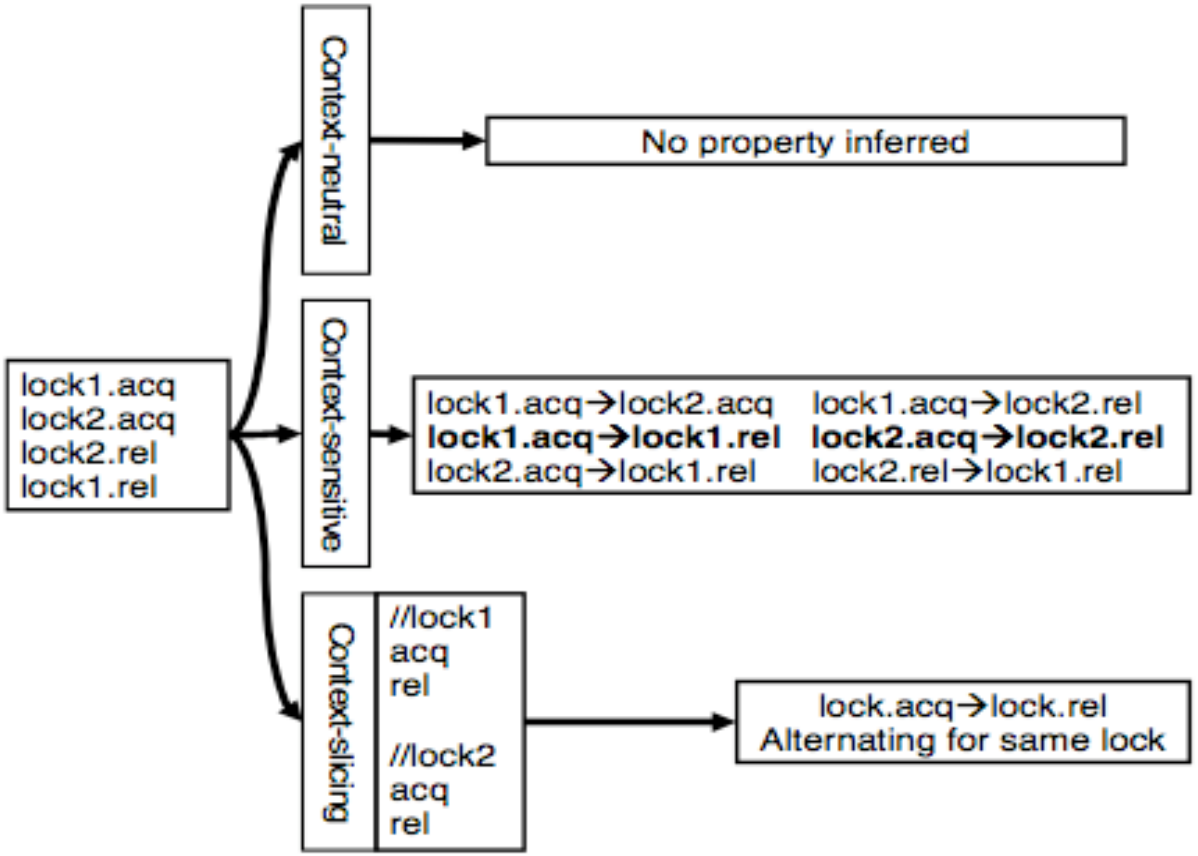
- Imperfect trace: PSPSPSPPP
- Add sub-trace partitioning: P+S+
- Property satisfaction rate:

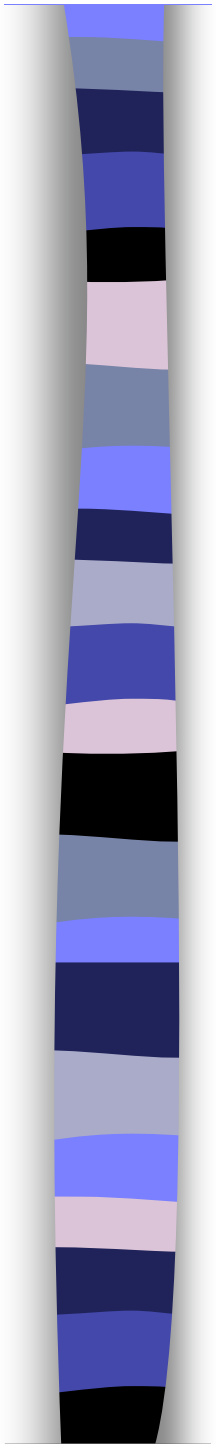
$$\frac{n_{\text{Alternating}}}{n} = .75$$

- Pick pairs above some threshold

Solution: Contextual Properties

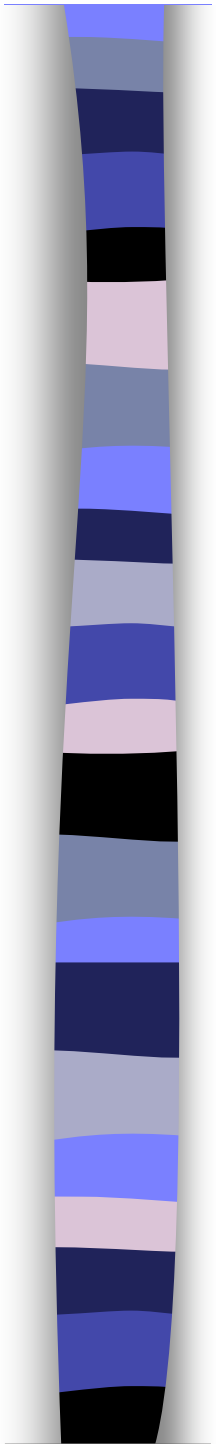
- Runtime thread, object, real params, etc.





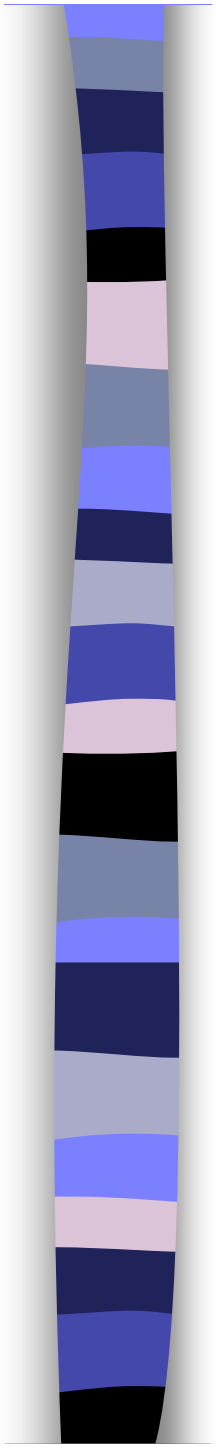
Solution: Selection Heuristic

- Goal: find interesting properties
- Non-reachability in call graph
  - X(){… foo(); … bar(); …}
- Event name similarity
  - ExAcquireFastMutexUnsafe
- Chaining
  - A->B, B->C, A->C => A->B->C



Experiments

- Perracotta
  - inference engine implementation
- Input: function call sequence
- Output: list of properties
- Tested programs:
  - Daisy, toy file system
  - JBoss, Java application server
  - Windows kernel API

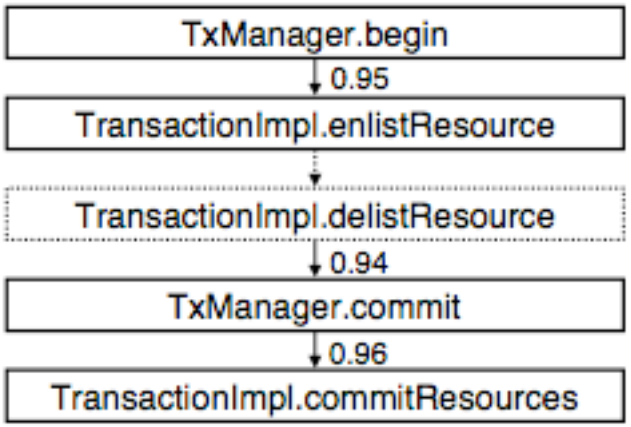


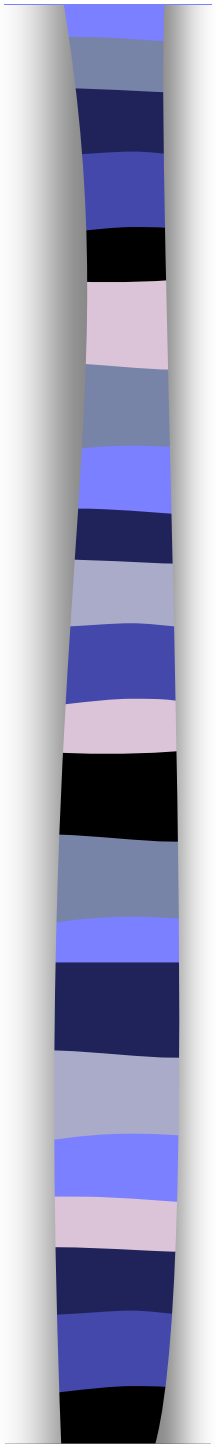
Experiments: Daisy

- Unix-like file system in 2000 lines of code
- 70,000 events - 40 distinct
- Inference
  - Inferred 70 properties (52 from imperfect)
  - Approximation and contextual slicing work well together
- Verified the inferred properties
  - Locks handled differently across layers

Experiments: JBoss

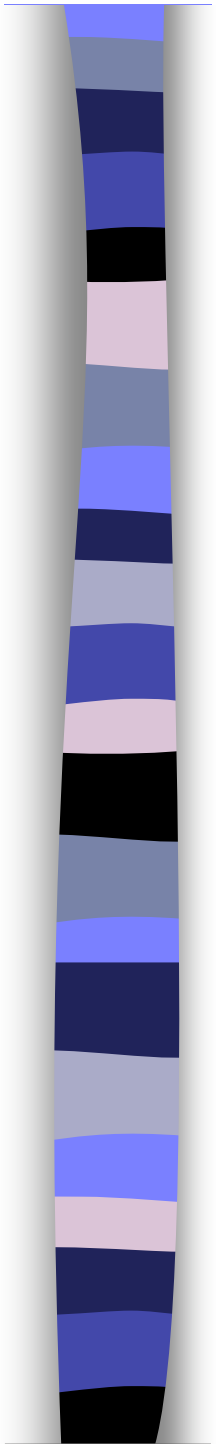
- 2.5 million events - 91 distinct
- Inference
  - 490 properties
  - Chaining and call graph reduce them
- Inferred properties vs. J2EE specs
  - Appx. inference finds extra properties
  - Mimics object Interaction diagram





Experiments: Windows kernel

- 17 execution traces
- 300K-5.8M traces
- 7,611 properties -> 142 (56 useful)
- Name similarity and call graph are good
- Inferred properties vs. MSDN document
  - Found extra properties
  - Found a new deadlock bug in NTFS



Summary

- Developed a mechanism to infer temporal invariants
- Deals with imperfect traces
- Can distinguish interesting invariants
- Works on big systems