Specification Mining With Few False Positives

Claire Le Goues
Westley Weimer
University of Virginia

March 25, 2009

Slide 0.5: Hypothesis

We can use measurements of the "trustworthiness" of source code to mine specifications with few false positives.

Slide 0.5: Hypothesis

We can use measurements of the "trustworthiness" of source code to mine specifications with few false positives.

Slide 0.5: Hypothesis

We can use measurements of the "trustworthiness" of source code to mine specifications with few false positives.

Slide 0.5: Hypothesis

We can use measurements of the "trustworthiness" of source code to mine specifications with few false positives.

Outline

- Motivation: Specifications
- Problem: Specification Mining
- Solution: Trustworthiness
- Evaluation: 3 Experiments
- Conclusions

Specifications

Why Specifications?

- Modifying code, correcting defects, and evolving code account for as much as 90% of the total cost of software projects.
- Up to 60% of maintenance time is spent studying existing software.
- Specifications are useful for debugging, testing, maintaining, refactoring, and documenting software.

Our Definition (Broadly)

A specification is a formal description of some aspect of legal program behavior.

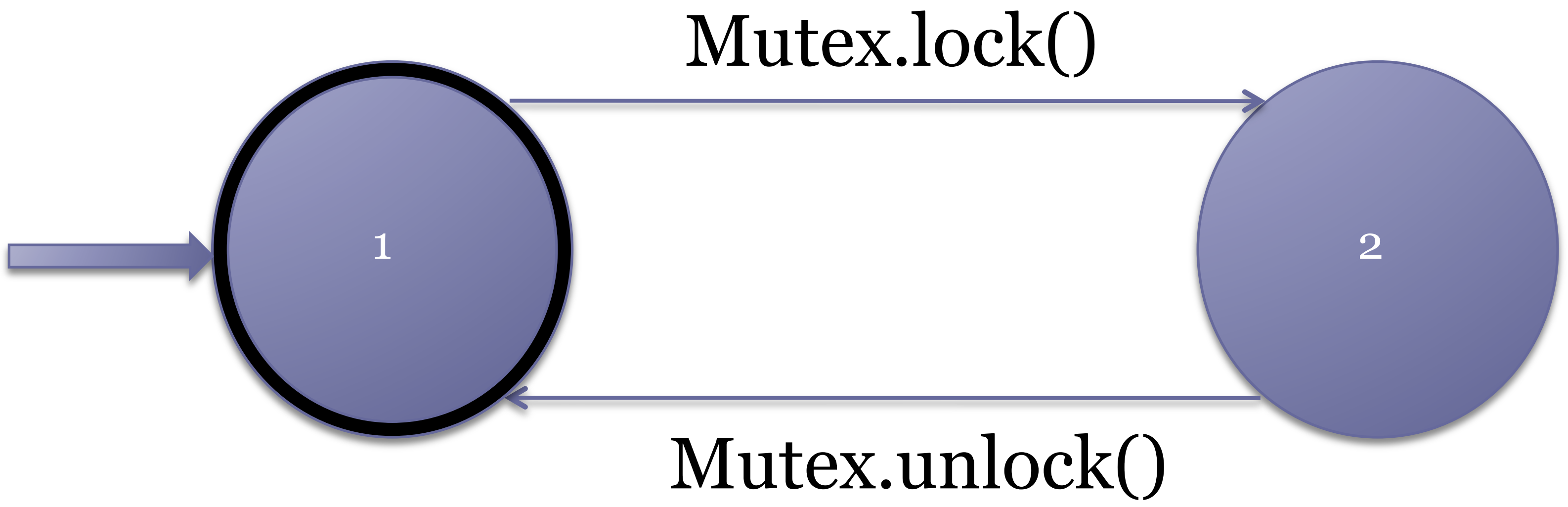
What kind of specification?

- We would like specifications that are simple and machine-readable
- We focus on partial-correctness specifications describing temporal properties
    - Describes legal sequences of events, where an event is a function call; similar to an API.
- Two-state finite state machines

Example Specification

Event A: Mutex.lock()
Event B: Mutex.unlock()

Example: Locks



Our Specifications

- For the sake of this work, we are talking about this type of two-state temporal specifications.
- These specifications correspond to the regular expression `(ab)*`
  - More complicated patterns are possible.

The Problem

Where do formal specifications come from?

- Formal specifications are useful, but there aren’t as many as we would like.
- We use specification mining to automatically derive the specifications from the program itself.

Mining 2-state Temporal Specifications

- **Input:** program traces – a sequence of events that can take place as the program runs.
  - ▫ Consider pairs of events that meet certain criteria.
  - ▫ Use statistics to figure out which ones are likely true specifications.
- **Output:** ranked set of candidate specifications, presented to a programmer for review and validation.

Problem: False Positives Are Common

Event A: Iterator.hasNext()
Event B: Iterator.next()

- This is very *common* behavior.
- This is not *required* behavior.
  - Iterator.hasNext() does not have to be followed eventually by Iterator.next() in order for the code to be correct.
- This candidate specification is a false positive.

Previous Work

Benchmark	LOC	Candidate Specs	False Positive Rate
Infinity	28K	10	90%
Hibernate	57K	51	82%
Axion	65K	25	68%
Hsqldb	71K	62	89%
Cayenne	86K	35	86%
Sablecc	99K	4	100%
Jboss	107K	114	90%
Mckoi-sql	118K	156	88%
Ptolemy2	362K	192	95%

* Adapted from Weimer-Necula, TACAS 2005

Previous Work

Benchmark	LOC	Candidate Specs	False Positive Rate
Infinity	28K	10	**90%**
Hibernate	57K	51	**82%**
Axion	65K	25	**68%**
Hsqldb	71K	62	**89%**
Cayenne	86K	35	**86%**
Sablecc	99K	4	**100%**
Jboss	107K	114	**90%**
Mckoi-sql	118K	156	**88%**
Ptolemy2	362K	192	**95%**

* Adapted from Weimer-Necula, TACAS 2005

Our Solution: Trustworthiness

The Problem (as we see it)

- Let’s pretend we’d like to learn the rules of English grammar.
- …but all we have is a stack of high school English papers.
- Previous miners ignore the differences between A papers and F papers.
- Previous miners treat all traces as though they were all equally indicative of correct program behavior.

Solution: Code Trustworthiness

- Trustworthy code is unlikely to exhibit API policy violations.
- Candidate specifications derived from trustworthy code are more likely to be true specifications.

What is trustworthy code?

Informally…

- Code that hasn’t been changed recently
- Code that was written by trustworthy developers
- Code that hasn’t been cut and pasted all over the place
- Code that is readable
- Code that is well-tested
- And so on.

Can you firm that up a bit?

- Multiple surface-level, textual, and semantic features can reveal the trustworthiness of code
  - Churn, author rank, copy-paste development, readability, frequency, feasibility, density, and others.
- Our miner should believe that lock() – unlock() is a specification if it is often followed on trustworthy traces and often violated on untrustworthy ones.

A New Miner

- Statically estimate the trustworthiness of each code fragment.
- Lift that judgment to program traces by considering the code visited along the trace.
- Weight the contribution of each trace by its trustworthiness when counting event frequencies while mining.

Incorporating Trustworthiness

- We use linear regression on a set of previously published specifications to learn good weights for the different trustworthiness factors.
- Different weights yield different miners.

Evaluation

Experimental Questions

- Can we use trustworthiness metrics to build a miner that finds useful specifications with few false positives?
- Which trustworthiness metrics are the most useful in finding specifications?
- Do our ideas about trustworthiness generalize?

Experimental Questions

- **Can we use trustworthiness metrics to build a miner that finds useful specifications with few false positives?**
- Which trustworthiness metrics are the most useful in finding specifications?
- Do our ideas about trustworthiness generalize?

Experimental Setup: Some Definitions

- **False positive**: an event pair that appears in the candidate list, but a program trace may contain only event A and still be correct.
- Our **normal** miner balances true positives and false positives (maximizes F-measure)
- Our **precise** miner avoids false positives (maximizes precision)

Experiment 1: A New Miner

	Normal Miner		Precise Miner		WN	
Program	**False**	**Violations**	**False**	**Violations**	**False**	**Violations**
Hibernate	53%	279	17%	153	82%	93
Axion	42%	71	0%	52	68%	45
Hsqldb	25%	36	0%	5	89%	35
jboss	84%	255	0%	12	90%	94
Cayenne	58%	45	0%	23	86%	18
Mckoi-sql	59%	20	0%	7	88%	69
ptolemy	14%	44	0%	13	95%	72
Total	**69%**	**740**	**5%**	**265**	**89%**	**426**

On this dataset:

- Our normal miner produces 107 false positive specifications.
- Our precise miner produces 1
- The previous work produces 567.

More Thoughts On Experiment 1

- Our normal miner improves on the false positive rate of previous miners by 20%.
- Our precise miner offers an order-of-magnitude improvement on the false positive rate of previous work.
- We find specifications that are more useful in terms of bug finding: we find 15 bugs per mined specification, where previous work only found 7.
- In other words: **we find useful specifications with fewer false positives.**

Experimental Questions

- Can we use trustworthiness metrics to build a miner that finds useful specifications with few false positives?
- **Which trustworthiness metrics are the most useful in finding specifications?**
- Do our ideas about trustworthiness generalize?

Experiment 2: Metric Importance

Metric	F	p
Frequency	32.3	0.0000
Copy-Paste	12.4	0.0004
Code Churn	10.2	0.0014
Density	10.4	0.0013
Readability	9.4	0.0021
Feasibility	4.1	0.0423
Author Rank	1.0	0.3284
Exceptional	10.8	0.0000
Dataflow	4.3	0.0000
Same Package	4.0	0.0001
One Error	2.2	0.0288

- Results of an analysis of variance (ANOVA).
- Shows the importance of the trustworthiness metrics.
- F is the predictive power (1.0 means no power).
- p is the probability that it had no effect (smaller is better).

More Thoughts on Experiment 2

Metric	F	p
Frequency	**32.3**	**0.0000**
Copy-Paste	12.4	0.0004
Code Churn	10.2	0.0014
Density	10.4	0.0013
Readability	9.4	0.0021
Feasibility	4.1	0.0423
Author Rank	1.0	0.3284
Exceptional	10.8	0.0000
Dataflow	4.3	0.0000
Same Package	4.0	0.0001
One Error	2.2	0.0288

- Statically predicted path frequency has the strongest predictive power.

More Thoughts on Experiment 2

Metric	F	p
Frequency	32.3	0.0000
Copy-Paste	12.4	0.0004
Code Churn	10.2	0.0014
Density	10.4	0.0013
Readability	9.4	0.0021
Feasibility	4.1	0.0423
Author Rank	**1.0**	**0.3284**
Exceptional	10.8	0.0000
Dataflow	4.3	0.0000
Same Package	4.0	0.0001
One Error	2.2	0.0288

- Statically predicted path frequency has the strongest predictive power.
- Author rank has no effect on the model.

More Thoughts on Experiment 2

Metric	F	p
Frequency	32.3	0.0000
Copy-Paste	12.4	0.0004
Code Churn	10.2	0.0014
Density	10.4	0.0013
Readability	9.4	0.0021
Feasibility	4.1	0.0423
Author Rank	1.0	0.3284
Exceptional	**10.8**	**0.0000**
Dataflow	**4.3**	**0.0000**
Same Package	**4.0**	**0.0001**
One Error	**2.2**	**0.0288**

- Statically predicted path frequency has the strongest predictive power.
- Author rank has no effect on the model.
- Previous work falls somewhere in the middle.

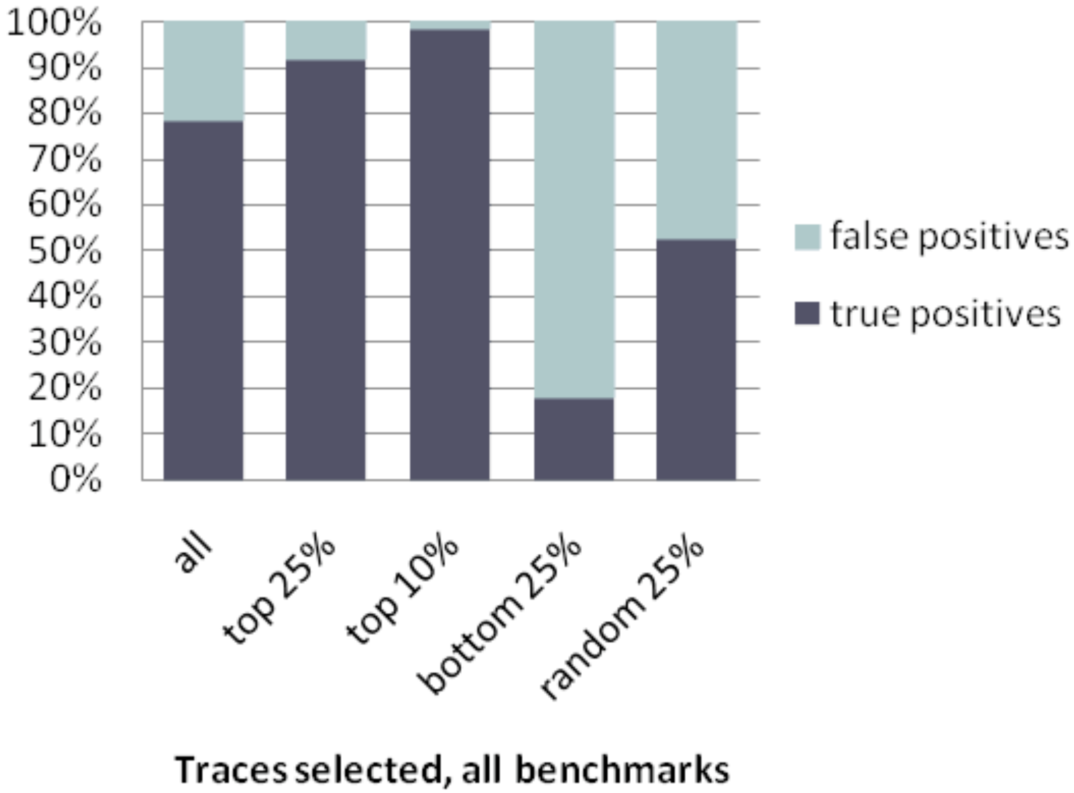
Experimental Questions

- Can we use trustworthiness metrics to build a miner that finds useful specifications with few false positives?
- Which trustworthiness metrics are the most useful in finding specifications?
- **Do our ideas about trustworthiness generalize?**

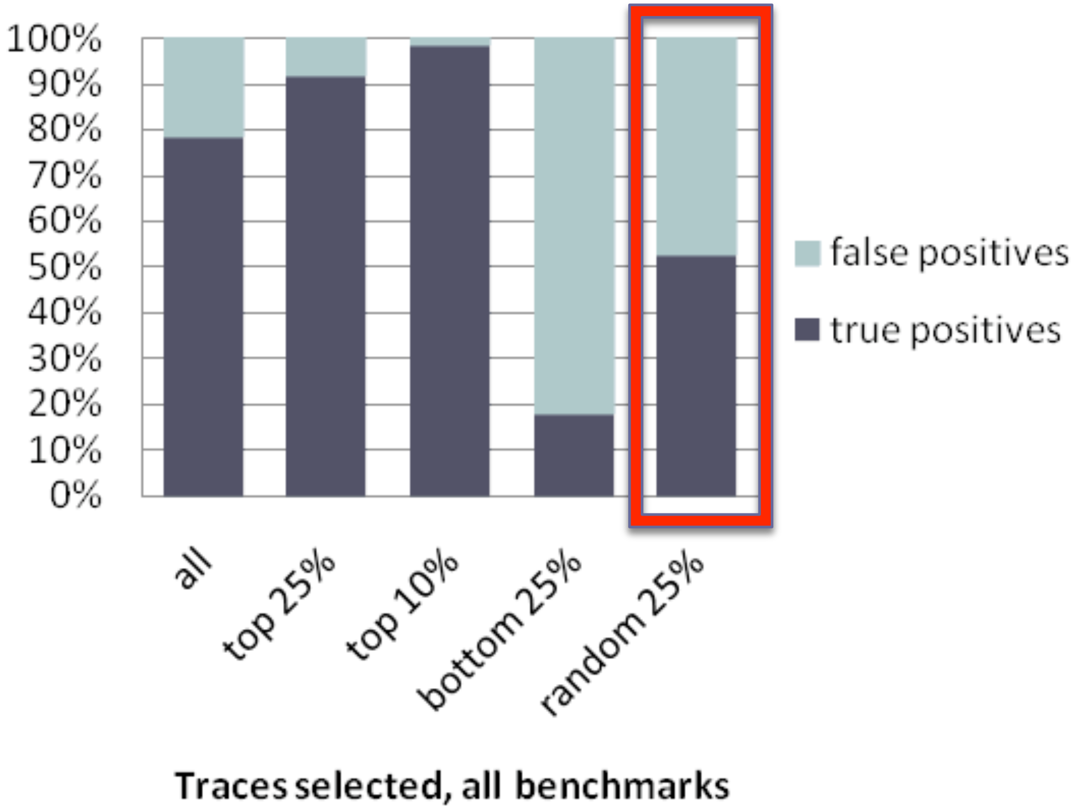
Experiment 3: Does it generalize?

- Previous work claimed that more input is necessarily better for specification mining.
- We hypothesized that smaller, more trustworthy input sets would yield more accurate output from previously implemented tools.

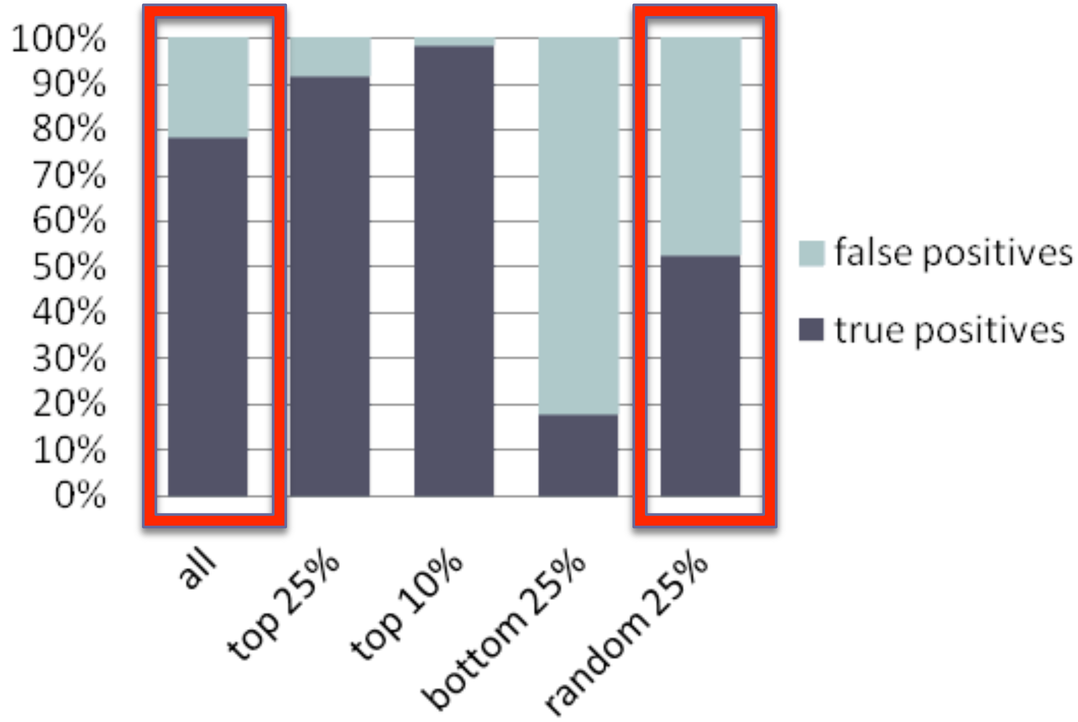
Experiment 3: Generalizing



Experiment 3: Generalizing

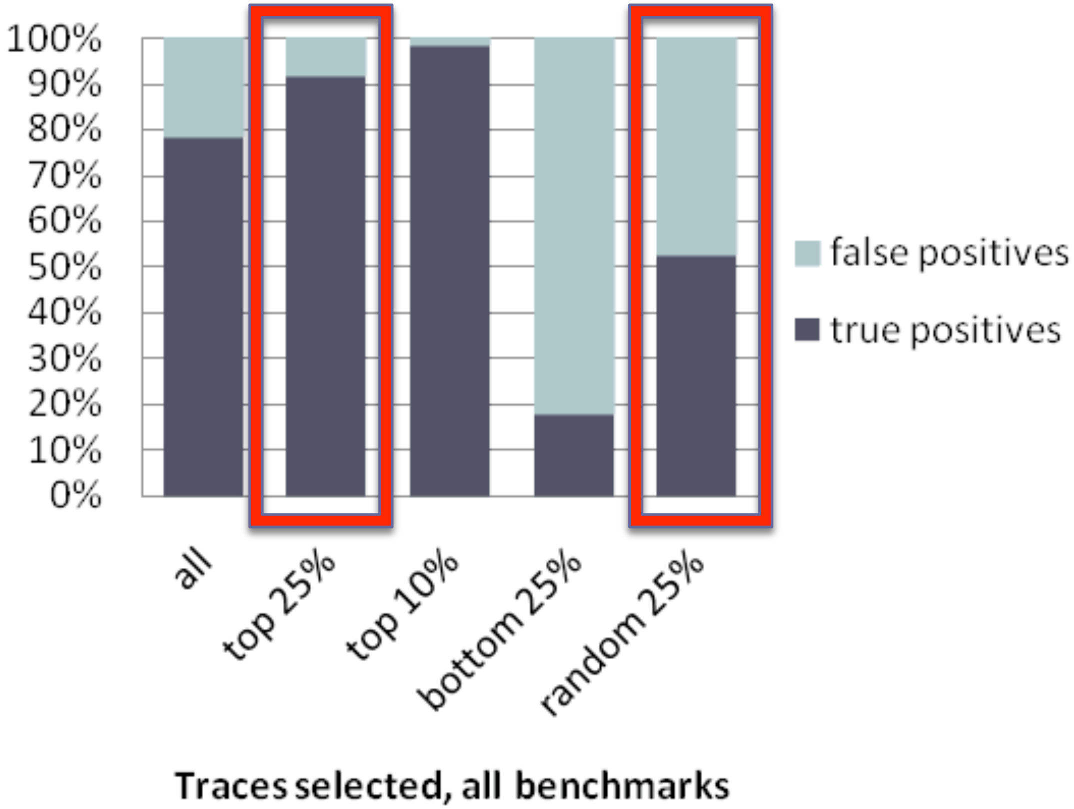


Experiment 3: Generalizing



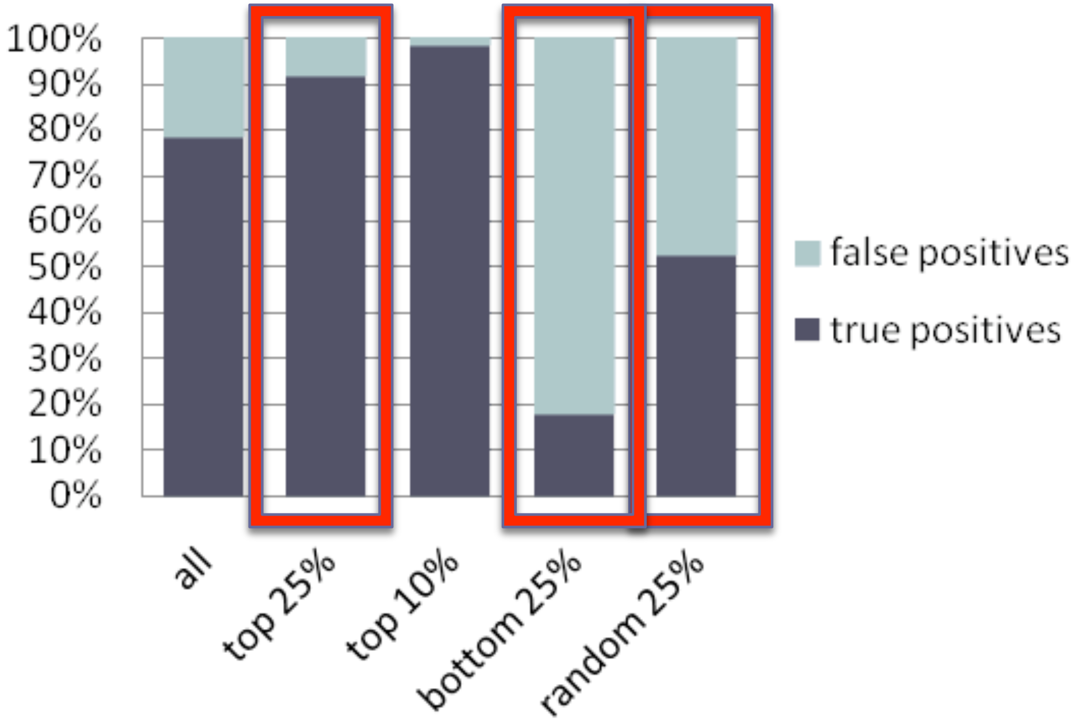
Traces selected, all benchmarks

Experiment 3: Generalizing



Experiment 3: Generalizing

- The top 25% “most trustworthy” traces make for a much more accurate miner; the opposite effect is true for the 25% “least trustworthy” traces.
- We can throw out the least trustworthy 40-50% of traces and still find the exact same specifications with a slightly lower false positive rate.
- **More traces != better, so long as the traces are trustworthy.**



Traces selected, all benchmarks

Experimental Summary

- We can use trustworthiness metrics to Build a Better Miner: our normal miner improves on the false positive rate of previous work by 20%, our precise miner by an order of magnitude, while still finding useful specifications.
- Statistical techniques show that our notion of trustworthiness contributes significantly to our success.
- We can increase the precision and accuracy of previous techniques by using a trustworthy subset of the input.

Conclusions

Summary

- Formal specifications are very useful.
- The previous work in specification mining yields too many false positives for industrial practice.
- We developed a notion of trustworthiness to evaluate the likelihood that code adheres to two-state temporal specifications.

Conclusion

A specification miner that incorporates notions of code trustworthiness can mine useful specifications with a much lower false positive rate.

The End
(questions?)