IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Tackling Lack of Software Specifications

A Sustained, Sustainability and Productivity Crisis

Hridesh Rajan

Collaboration with Hoan A. Nguyen, Tien Nguyen, Gary Leavens, Samantha Khairunnesa, John Singleton, Hung Phan, Robert Dyer, and Vasant Honavar

Sustainability and productivity challenge

- To produce critical software infrastructure so it is:
  - of highest quality and free of defects,
  - produced ethically and within budget, and
  - maintainable, upgradeable, portable, scalable, secure.

- Pervasiveness of software infrastructures in such critical areas as power, banking and finance, air traffic control, telecommunication, transportation, national defense, and healthcare need us to address this challenge.

Software specifications* can help achieve this sustainability and productivity challenge.

* Software specifications: formal, often machine readable, description of software’s intended behavior, e.g. {Pre} S {Post} behavioral specifications

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.
  - Maintenance of code can become easier
  - Lower cost of code understanding & total lifecycle cost
  - Specification-guided code optimization
  - Prevent introducing new bugs during maintenance
  - Code reuse
  - Specification-guided synthesis
  - Modular analysis and verification, scalable tools

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

Despite these benefits useful, non-trivial specifications aren’t widely available

  - Modular analysis and verification leading to scalable tools

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

Why aren’t software specifications widely available?

  - Modular analysis and verification leading to scalable tools

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

Cost

Education

Tools

Libraries

  - Modular analysis and verification leading to scalable tools

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

Cost

Education

Tools

Libraries

  - Modular analysis and verification leading to scalable tools

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

Unspecified libraries are root cause

- increase cost of specification

- make education harder

- make tool support difficult

- make specifying libraries harder

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

How to Solve it? Specify key libraries

- decrease cost of specification

- make education easier (examples)

- make tool support easier

- make specifying libraries easier

Sustainability and productivity challenge

- If specifications are widely available, a wide variety of techniques for addressing the sustainability and productivity crisis can be enabled.

How to Solve it? Specify key libraries

Challenge #1: lower manual cost of specifying libraries, infer most

Challenge #2: infer rich, but practical specifications, allow code evolution

Mining Preconditions of APIs in Large-scale Code Corpus, FSE'14.

Robert Dyer*

Hoan Nguyen

Tien N. Nguyen

Key Ideas

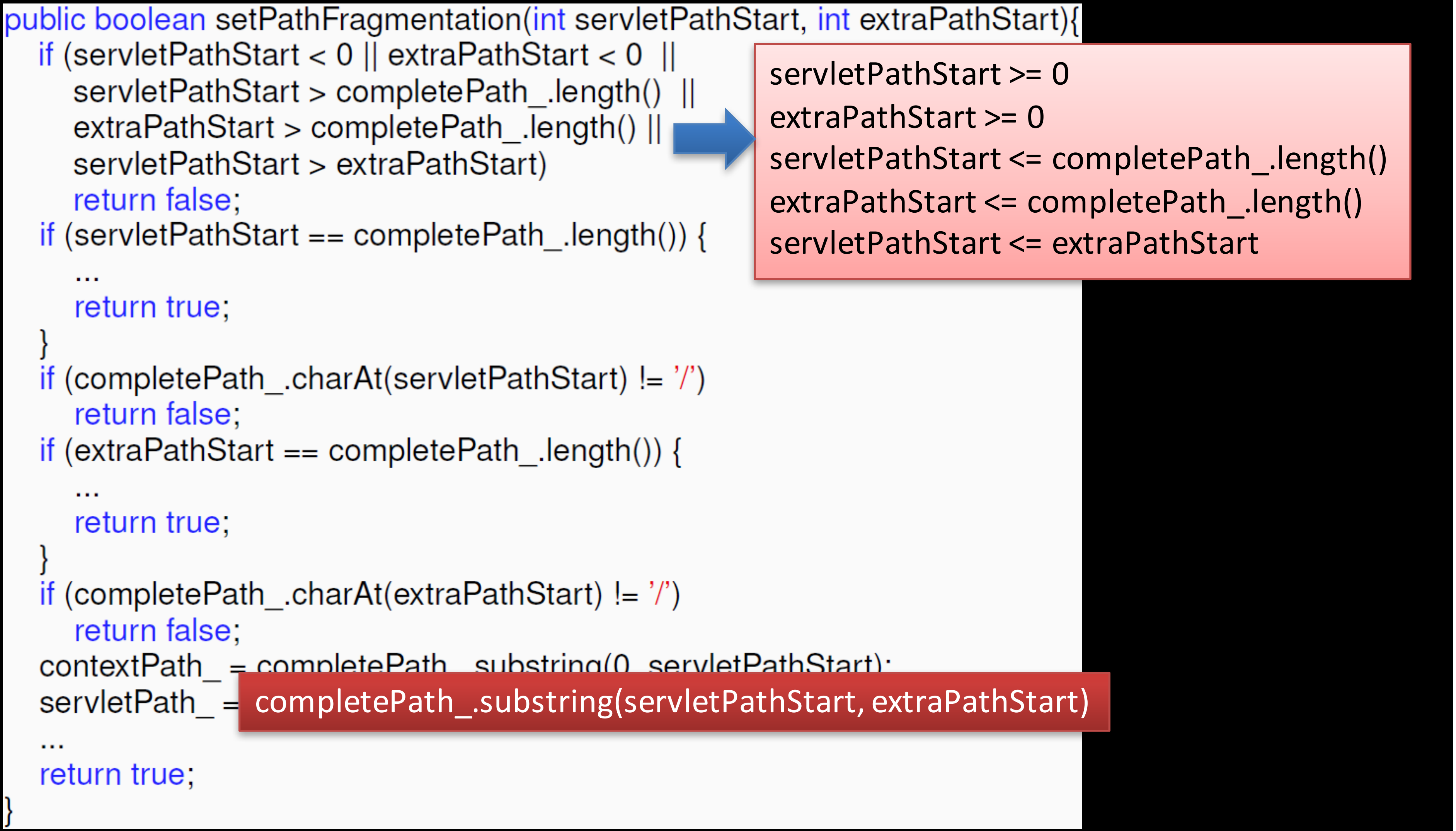
Preconditions can be mined from **guarded conditions** at the call sites of the code using the APIs

```
void m(...) {
   ...
   if (pred)
      lib.api();
   ...
}
```

Preconditions mined from **multiple projects** in a **large-scale code corpus** can be used to filter out chaff

Key Ideas

- Preconditions can be mined from guarded conditions at the call sites of the code using the APIs
- Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff



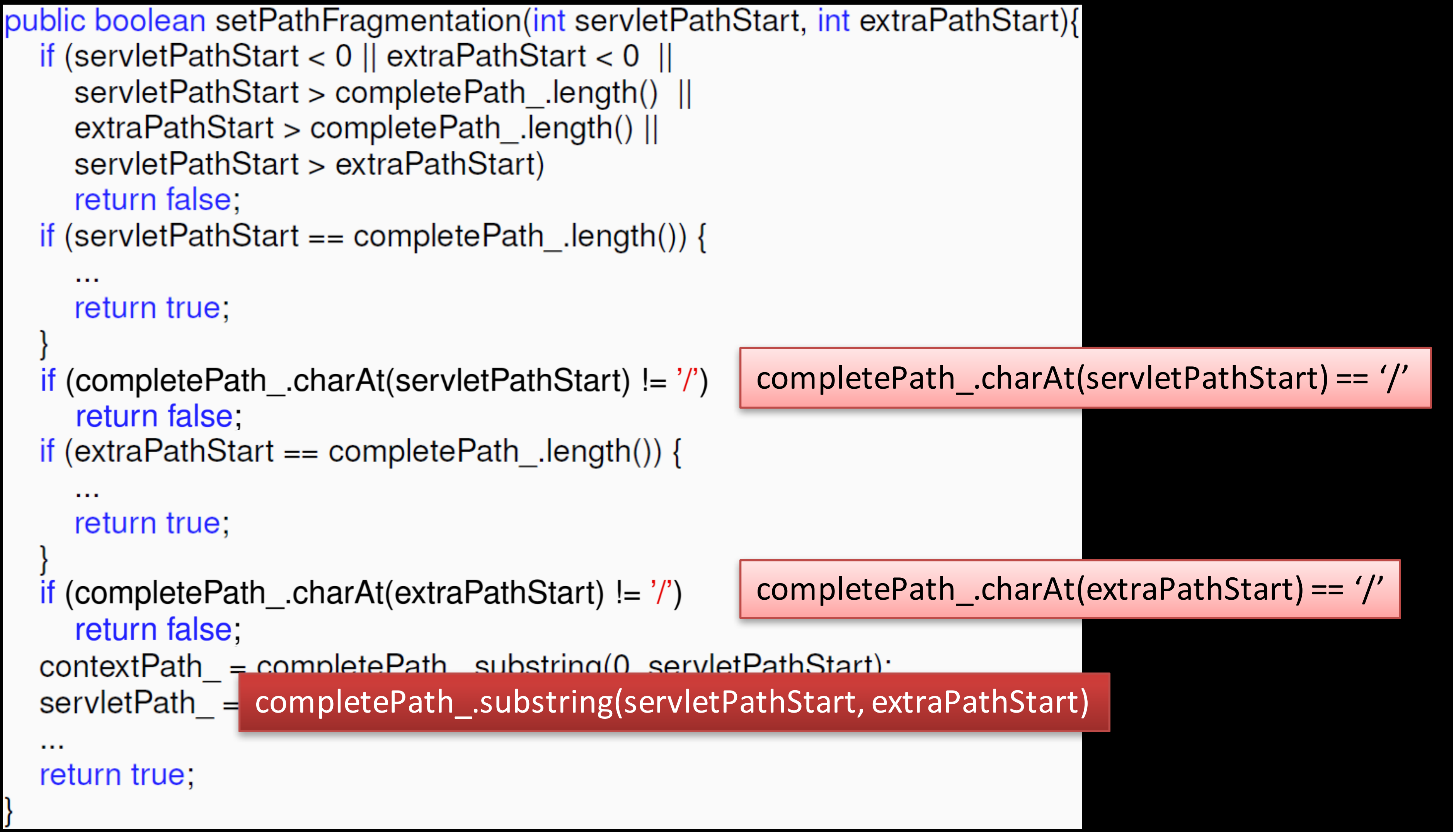
```
public boolean setPathFragmentation(int servletPathStart, int extraPathStart){
   if (servletPathStart < 0 || extraPathStart < 0  ||
      servletPathStart > completePath_.length()  ||
      extraPathStart > completePath_.length() ||
      servletPathStart > extraPathStart)
      return false;
   if (servletPathStart == completePath_.length()) {
      ...
      return true;
   }
   if (completePath_.charAt(servletPathStart) != '/')
      return false;
   if (extraPathStart == completePath_.length()) {
      ...
      return true;
   }
   if (completePath_.charAt(extraPathStart) != '/')
      return false;
   contextPath_ = completePath_.substring(0, servletPathStart);
   servletPath_ = completePath_.substring(servletPathStart, extraPathStart)
   ...
   return true;
}
```

servletPathStart >= 0
extraPathStart >= 0
servletPathStart <= completePath_.length()
extraPathStart <= completePath_.length()
servletPathStart <= extraPathStart

Client code of API **String.substring(int,int)** in project SeMoA at revision 1929

Key Ideas

- Preconditions can be mined from guarded conditions at the call sites of the code using the APIs
- Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff



```
public boolean setPathFragmentation(int servletPathStart, int extraPathStart){
    if (servletPathStart < 0 || extraPathStart < 0  ||
        servletPathStart > completePath_.length()  ||
        extraPathStart > completePath_.length() ||
        servletPathStart > extraPathStart)
        return false;
    if (servletPathStart == completePath_.length()) {
        ...
        return true;
    }
    if (completePath_.charAt(servletPathStart) != '/')
        return false;
    if (extraPathStart == completePath_.length()) {
        ...
        return true;
    }
    if (completePath_.charAt(extraPathStart) != '/')
        return false;
    contextPath_ = completePath_.substring(0, servletPathStart);
    servletPath_ =
    ...
    return true;
}
```

completePath_.charAt(servletPathStart) == '/'

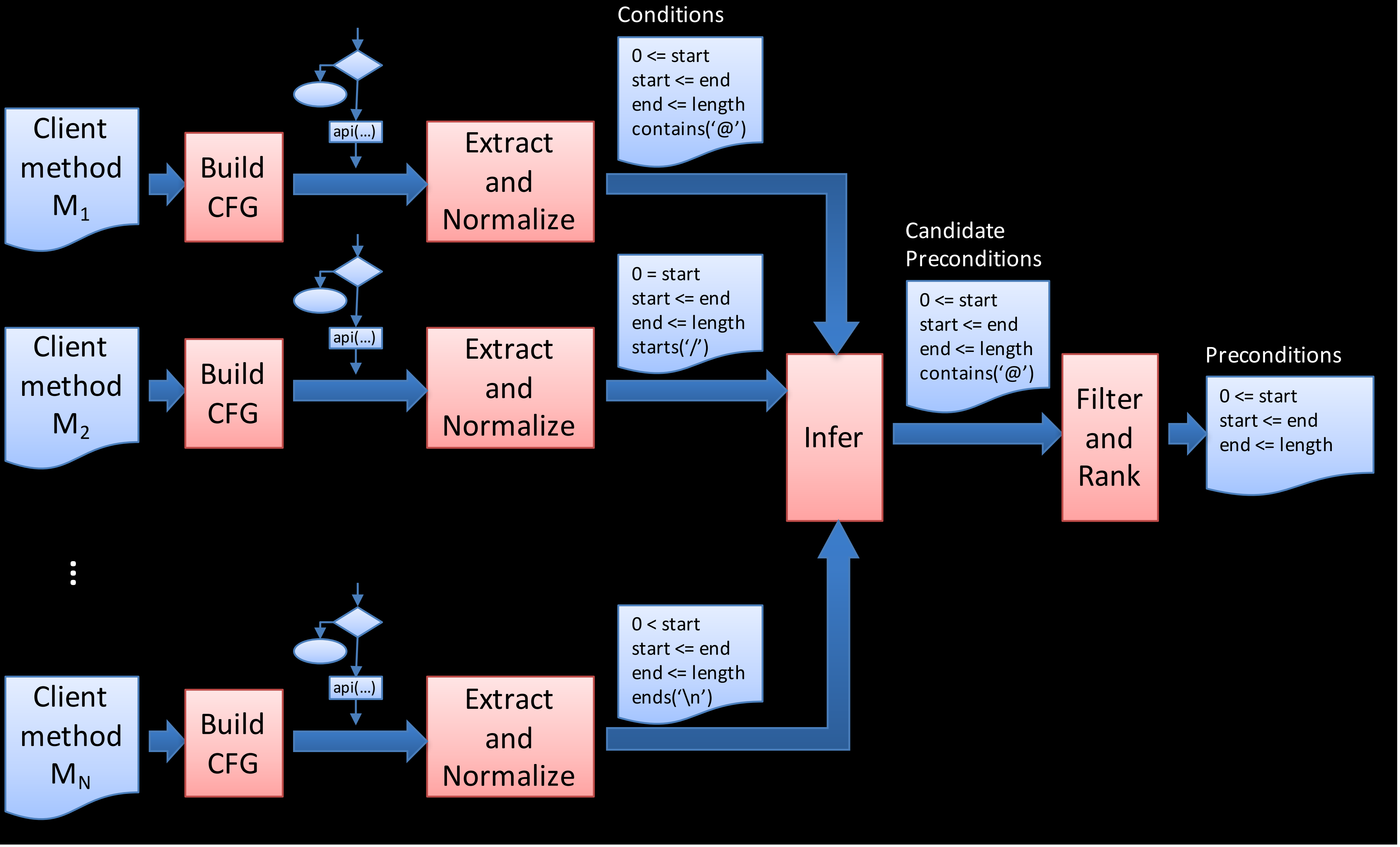
completePath_.charAt(extraPathStart) == '/'

completePath_.substring(servletPathStart, extraPathStart)

Key Ideas

Preconditions can be mined from guarded conditions at the call sites of the code using the APIs

Preconditions mined from multiple projects in a large-scale code corpus can be used to filter out chaff: a. infer, b. filter and rank



Evaluation – Accuracy

Data collection

	SourceForge	Apache
Projects	3,413	146
Total source files	497,453	132,951
Total classes	600,274	173,120
Total methods	4,735,151	1,243,911
Total SLOCs	92,495,410	25,117,837
Total used JDK classes	806 (63%)	918 (72%)
Total used JDK methods	7,592 (63%)	6,109 (55%)
Total method calls	22,308,251	5,544,437
Total JDK method calls	5,588,487	1,271,210

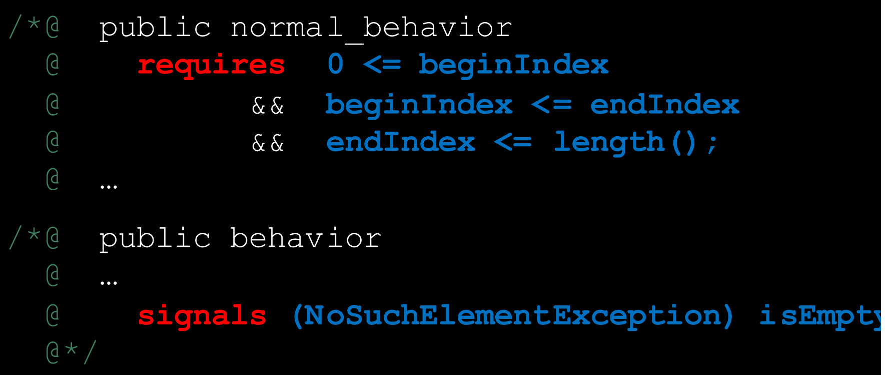
Almost 120 millions SLOCs

Ground Truth

www.jmlspecs.org

Extracted preconditions from published formal specification for JDK APIs on JML website

- 797 Methods
- 1155 preconditions



```
/*@  public normal_behavior
  @    requires   0 <= beginIndex
  @          &&  beginIndex <= endIndex
  @          &&  endIndex <= length();
  @  …
/*@  public behavior
  @  …
  @    signals (NoSuchElementException) isEmpty
  @*/
```

Accuracy of Preconditions Mining

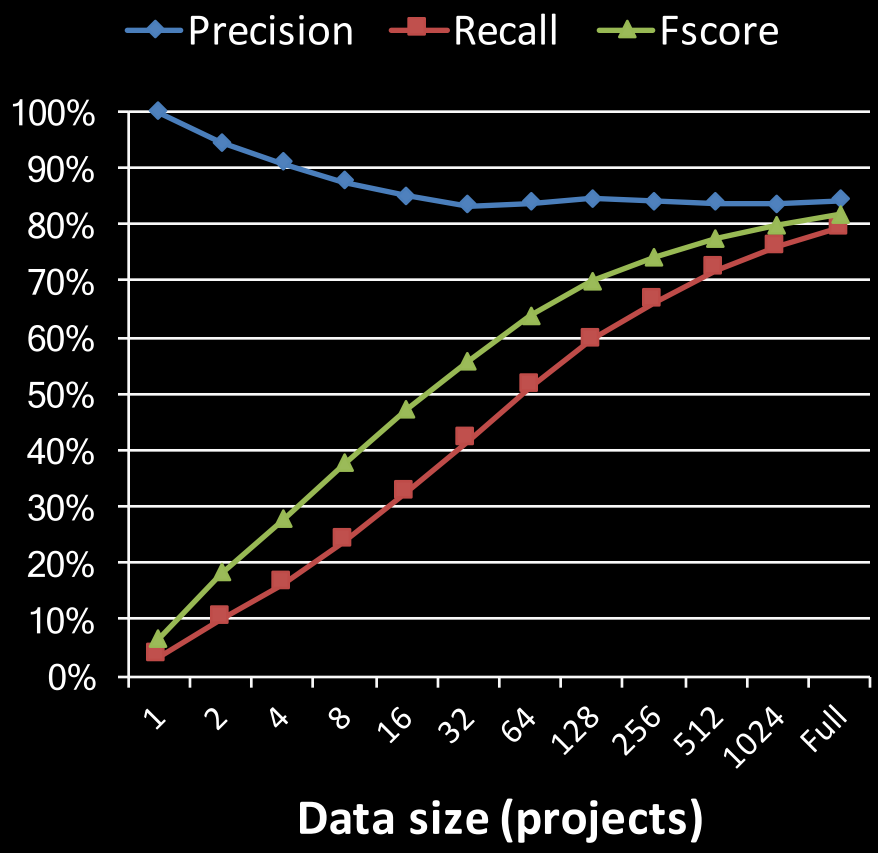
	Precision	Recall	Time
SourceForge	84%	79%	17h35m
Apache	82%	75%	34m
Both	83%	80%	18h03m

Performance

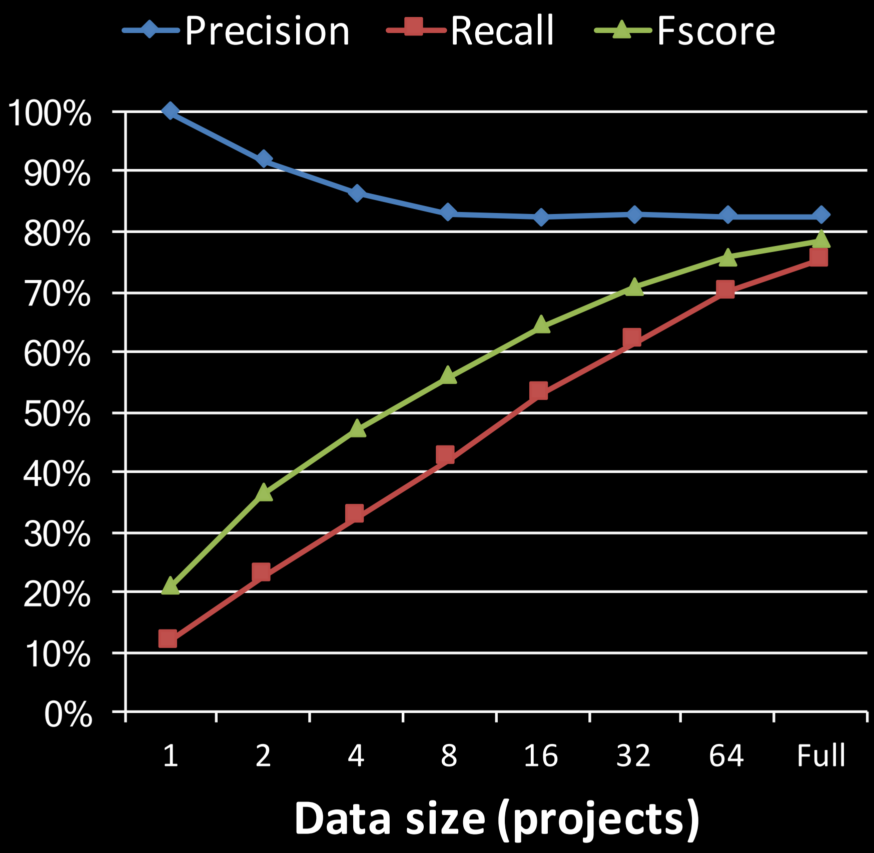
- ~ 1 minute/condition
- 5 preconditions are newly found for the JDK API methods that has already had JML specifications
- Effective for new specs

Class	Method	Suggest	Accept
StringBuffer	delete(int,int)	3	Y
	replace(int,int,String)	2	Y*
	setLength(int)	1	Y
	subSequence(int,int)	3	Y
	substring(int,int)	3	Y
LinkedList	add(int,Object)	2	Y
	addAll(int,Collection)	3	Y
	get(int)	2	Y
	listIterator(int)	2	Y
	remove(int)	2	Y
	set(int,Object)	2	Y
2 classes	11 methods	25	

Accuracy by size



SourceForge

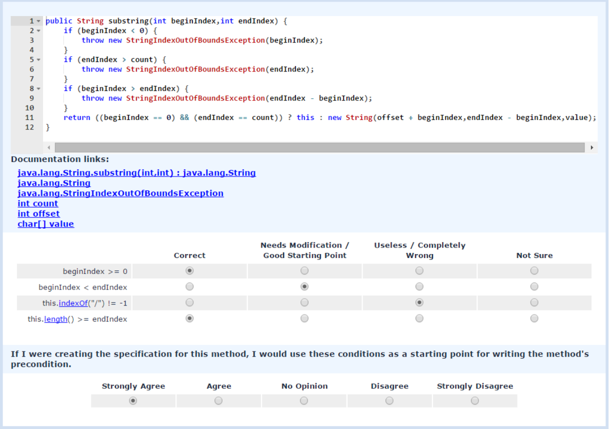


Apache

Usefulness Evaluation

Web-based Survey

http://boa.cs.iastate.edu/jml



```java
public String substring(int beginIndex,int endIndex) {
    if (beginIndex < 0) {
        throw new StringIndexOutOfBoundsException(beginIndex);
    }
    if (endIndex > count) {
        throw new StringIndexOutOfBoundsException(endIndex);
    }
    if (beginIndex > endIndex) {
        throw new StringIndexOutOfBoundsException(endIndex - beginIndex);
    }
    return ((beginIndex == 0) && (endIndex == count)) ? this : new String(offset + beginIndex,endIndex - beginIndex,value);
}
```

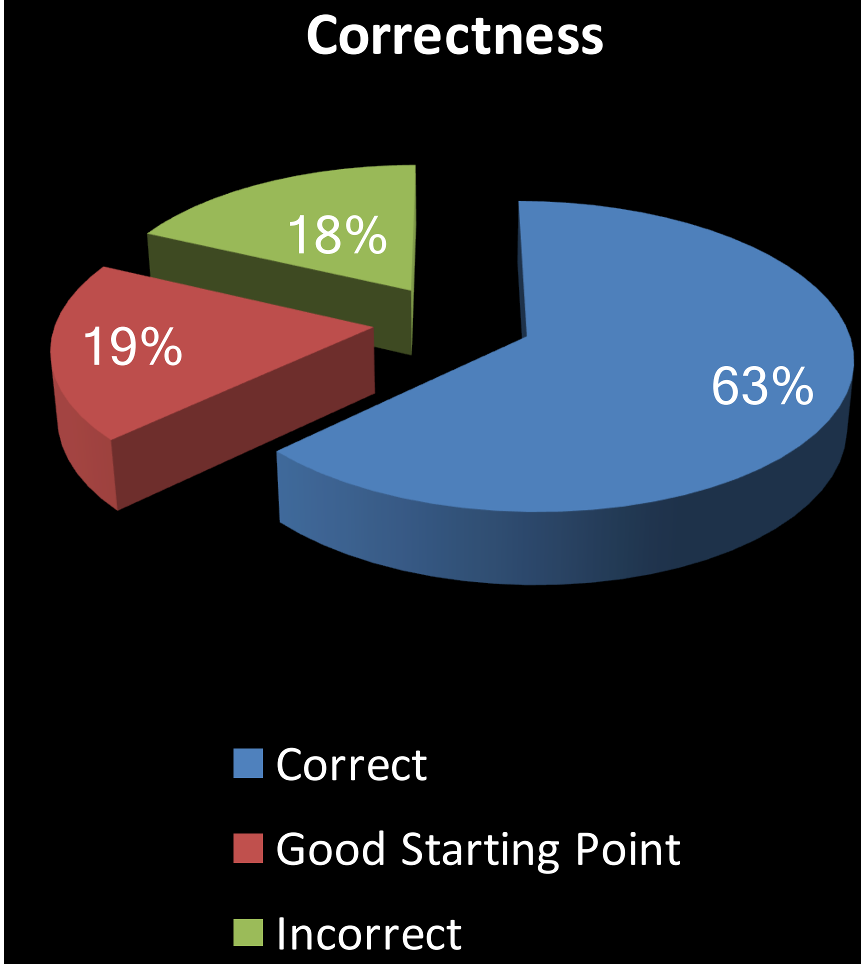
Documentation links:

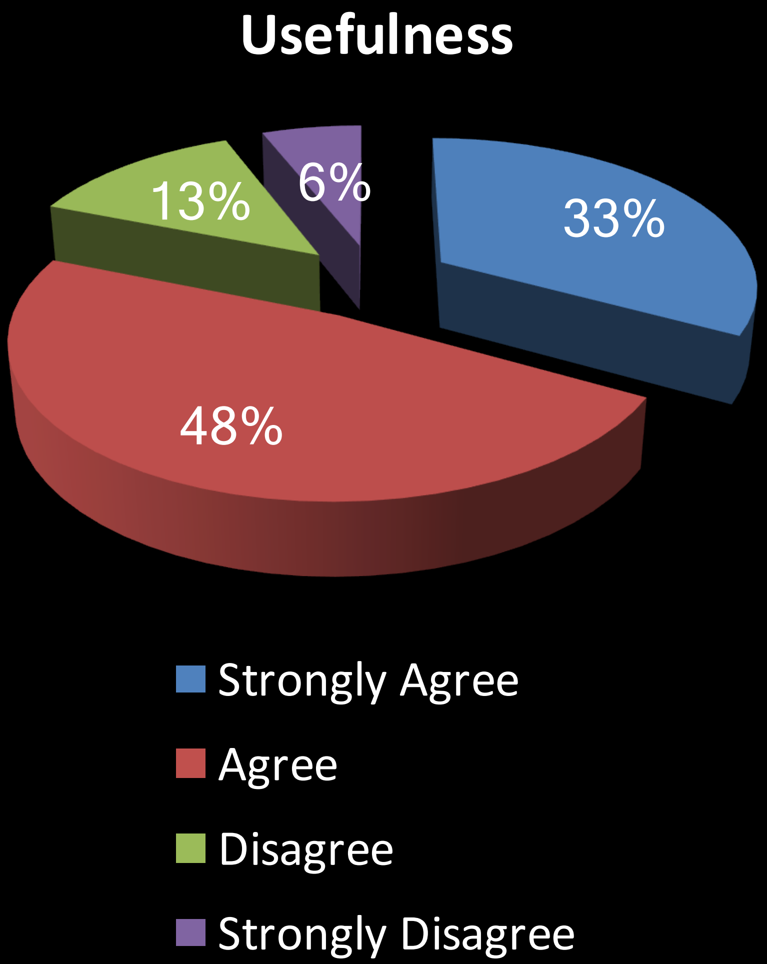
java.lang.String.substring(int,int) : java.lang.String
java.lang.String
java.lang.StringIndexOutOfBoundsException
int count
int offset
char[] value

	Correct	Needs Modification / Good Starting Point	Useless / Completely Wrong	Not Sure
beginIndex >= 0	●	○	○	○
beginIndex < endIndex	○	●	○	○
this.indexOf("/") != -1	○	○	●	○
this.length() >= endIndex	●	○	○	○

If I were creating the specification for this method, I would use these conditions as a starting point for writing the method's precondition.

Strongly Agree	Agree	No Opinion	Disagree	Strongly Disagree
●	○	○	○	○





Exploiting Implicit Beliefs to
Resolve Sparse Usage Problem
in Usage-based Specification
Mining, OOPSLA’17.

S. Khairunnessa

Hoan Nguyen

Tien N. Nguyen

Problem: Sparse labels in mined code corpus

Key Ideas

- Additional labels can be mined from **implicit beliefs** at the call sites of the code using the APIs

```
void m(…) {
   …
   O o = new O()
   lib.api(o);
    …
}
```

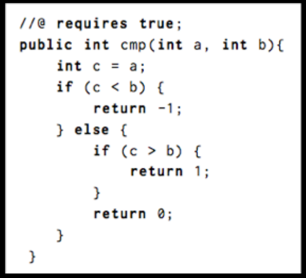
- Implicit beliefs mined from **multiple projects** in a **large-scale code corpus** can be used to strengthen explicit labels

An Algorithm and Tool to Infer Practical Postconditions, Ongoing work.

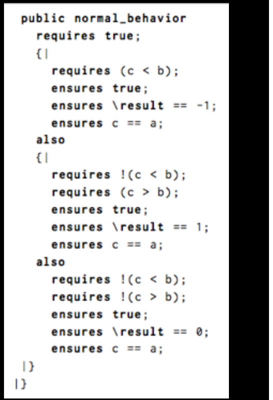
John Singleton

Gary T. Leavens

Problem: Using extant work , e.g. strongest postcondition (sp), for postcondition inference produces impractical specs



```
//@ requires true;
public int cmp(int a, int b){
    int c = a;
    if (c < b) {
        return -1;
    } else {
        if (c > b) {
            return 1;
        }
        return 0;
    }
 }
```



```
public normal_behavior
  requires true;
  {|
    requires (c < b);
    ensures true;
    ensures \result == -1;
    ensures c == a;
  also
  {|
    requires !(c < b);
    requires (c > b);
    ensures true;
    ensures \result == 1;
    ensures c == a;
  also
    requires !(c < b);
    requires !(c > b);
    ensures true;
    ensures \result == 0;
    ensures c == a;
 |}
|}
```

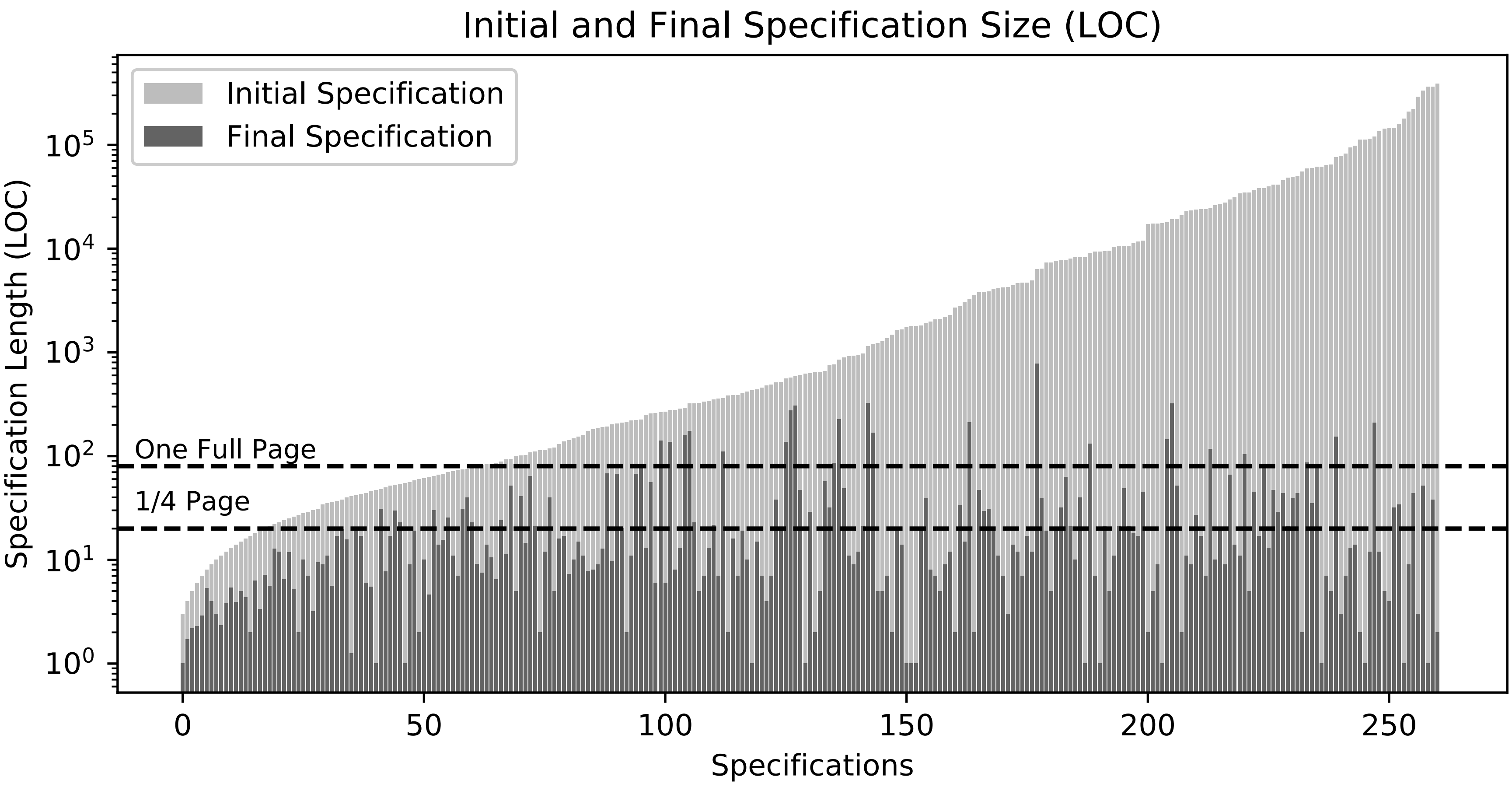
Key Ideas

Strongest postcondition inference produces **implicitly parallel formulas**

$$\texttt{sp}\ (\texttt{IF}\ B\ \texttt{THEN}\ S_1\ \texttt{ELSE}\ S_2)\ P = (\texttt{sp}\ S_1(P \wedge B)) \vee (\texttt{sp}\ S_2(P \wedge \neg B))$$

Flattening, and **recombining** parallel formulas can lead to much simpler inferred specifications.

Specification Reduction



Impact: 84% of specifications < ¼ page in length

We are overcoming lack of software specifications, a critical hurdle for high assurance SE, by combining program analysis and data mining.

boa.cs.iastate.edu

hridesh@iastate.edu