C2S: Translating Natural Language Comments to Formal Program Specifications

Juan Zhai, Yu Shi, Minxue Pan, Guian Zhou, Yongxiang Liu, Chunrong Fang,

Shiqing Ma, Lin Tan, Xiangyu Zhang

RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

PURDUE
UNIVERSITY

南京大學
NANJING UNIVERSITY

Motivation

- Formal program specifications are essential for various software engineering tasks, such as program verification, program synthesis, code debugging and software testing.
- Manually composing formal specifications:
  - Time-consuming
  - Error-prone
  - Requires substantial expertise
  - …

Provide Automation Support in Formal Specification Generation

Where do we Infer Specifications from?

Q: How do we know the semantics of a method?
A: By reading abundant natural language comments which describe semantics informally.

```
/** Removes and returns the first element from this list.
  * @throws NoSuchElementException if this list is empty
  * @return the first element from this list                */
 public boolean removeFirst(){ ... }
```

Java Documentation of Method LinkedList.removeFirst()

Can we translate the informal comments into formal specifications?

Existing Work Based on Comments

- Existing approaches all rely on patterns summarized manually from comments to derive specifications
  - Require substantial manual work
  - Have limited generality
  - Context-unaware

```
/** Returns the first component of this vector.
  * @returns the first component of this vector. */
 public E firstElement(){ ... }
```

Java Documentation of Method Vector.firstElement()

```
/** Removes and returns the first element from this list.
  * @throws NoSuchElementException if this list is empty
  * @return the first element from this list              */
 public boolean removeFirst(){ ... }
```

Java Documentation of Method LinkedList.removeFirst()

Devise a General Approach which Addresses the Limitations

Java Modeling Language

- Java modeling language (JML) is one formal specification language.
- It has been widely used by developers to provide specifications for JDK library methods.

```
/** @public exceptional_behavior
  * @requires index < 0 || index >= this.size();
  * @signals_only java.lang.IndexOutOfBoundsException;
  * @public normal_behavior
  * @requires 0 <= index && index < this.size();
  * @ensures \result == \old(this.get(index)); */
 public Object remove(int index){ ... }
```

JML Specifications of Method ArrayList.remove(int)

Idea

- Treat natural language comments and JML specifications as two languages expressing the same semantics, and formulate the specification translation task as a syntax-guided synthesis problem by assembling primitive tokens.

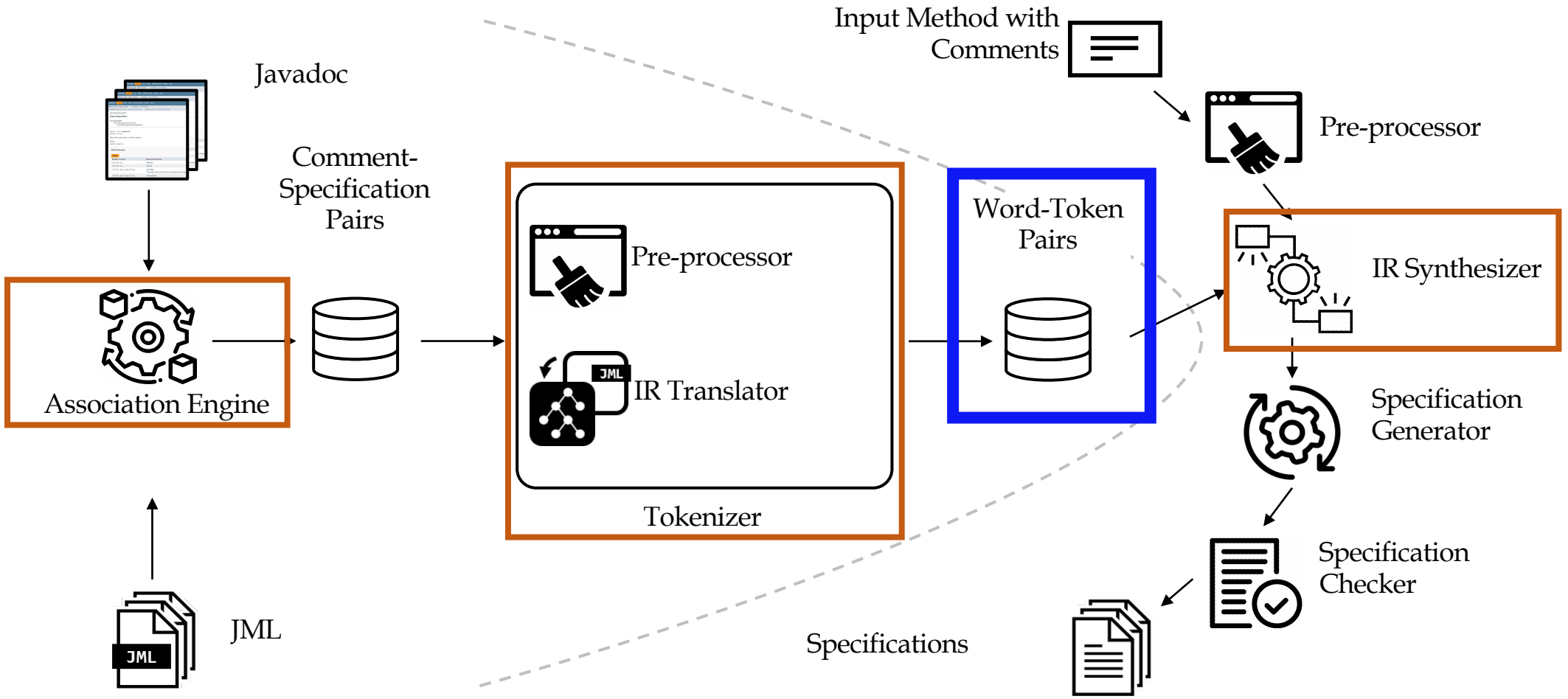
```
01 /** Removes the element at the specified position in this list. ①
02   * @throws IndexOutOfBoundsException if the index is out of
03   *         range (index < 0 || index >= size())                ②
04   * @param index - the index of the element to be removed
05   * @returns the element that was removed from the list      */ ③
06 public Object remove(int index){ ... }
```

(a) Java Documentation of Method ArrayList.remove(int)

```
11 /** @public exceptional_behavior
12   * @requires index < 0 || index >= this.size();                ④
13   * @signals_only java.lang.IndexOutOfBoundsException;
14   * @public normal_behavior
15   * @requires 0 <= index && index < this.size();
16   * @ensures \result == \old(this.get(index));                  ⑤
17   * @ensures \forall int i; index<=i && i<\old(this.size()-1);
     this.get(i)==null && \old(this.get(i+1))==null ||            ⑥
     this.get(i).equals(\old(this.get(i+1)));  */
18  public Object remove(int index){ ... }
```

(b) JML Specifications of Method ArrayList.remove(int)

Overarching Design



Search Space Preparation

Specification Synthesis

Association Engine

- Automatically couple specifications with corresponding comments based on annotations to prepare comment-specification pairs.

```
01 /** Removes the element at the specified position in this list. ①
02   * @throws IndexOutOfBoundsException if the index is out of
03   *         range (index < 0 || index >= size())                ②
04   * @param index - the index of the element to be removed
05   * @returns the element that was removed from the list      */ ③
06 public Object remove(int index){ ... }
```

(a) Java Documentation of Method ArrayList.remove(int)

```
11 /** @public exceptional_behavior
12   * @requires index < 0 || index >= this.size();                ④
13   * @signals_only java.lang.IndexOutOfBoundsException;
14   * @public normal_behavior
15   * @requires 0 <= index && index < this.size();
16   * @ensures \result == \old(this.get(index));                  ⑤
17   * @ensures \forall int i; index<=i && i<\old(this.size()-1);
     this.get(i)==null && \old(this.get(i+1))==null ||            ⑥
     this.get(i).equals(\old(this.get(i+1)));  */
18  public Object remove(int index){ ... }
```

(b) JML Specifications of Method ArrayList.remove(int)

Pre-processor

- Remove unnecessary information and normalize texts to get more general comments.
  - Remove stop words (common words appearing frequently) like “the”
  - Reduce derived words to their word stem, namely root form, by applying the Porter stemming algorithm, e.g., “inserts” →”insert”
  - Lowercase all the words

IR Translator

- Generalize a JML specification in the text form to an abstract IR form (represented using AST).
  - Substitute all concrete parameter names with parameter placeholders in the form of `pi@t` (the `i-th` parameter with type `t`).

Tokenizer

ID	Natural Language Comment	Formal Program Specification
i	Removes the element at the specified position in this list	\forall int i; index<=i && i<\old(this.size()-1); this.get(i)==null && \old(this.get(i+1))==null || this.get(i).equals(\old(this.get(i+1)));
ii	Returns the element that was removed from the list	\result==\old(this.get(index))
iii	Throws IndexOutOfBoundsException if the index is out of range (index < 0 || index >= size())	index<0 || index>=this.size() ⟶ throw IndexOutOfBoundsException
...	...	...

Comment-Specification Pairs

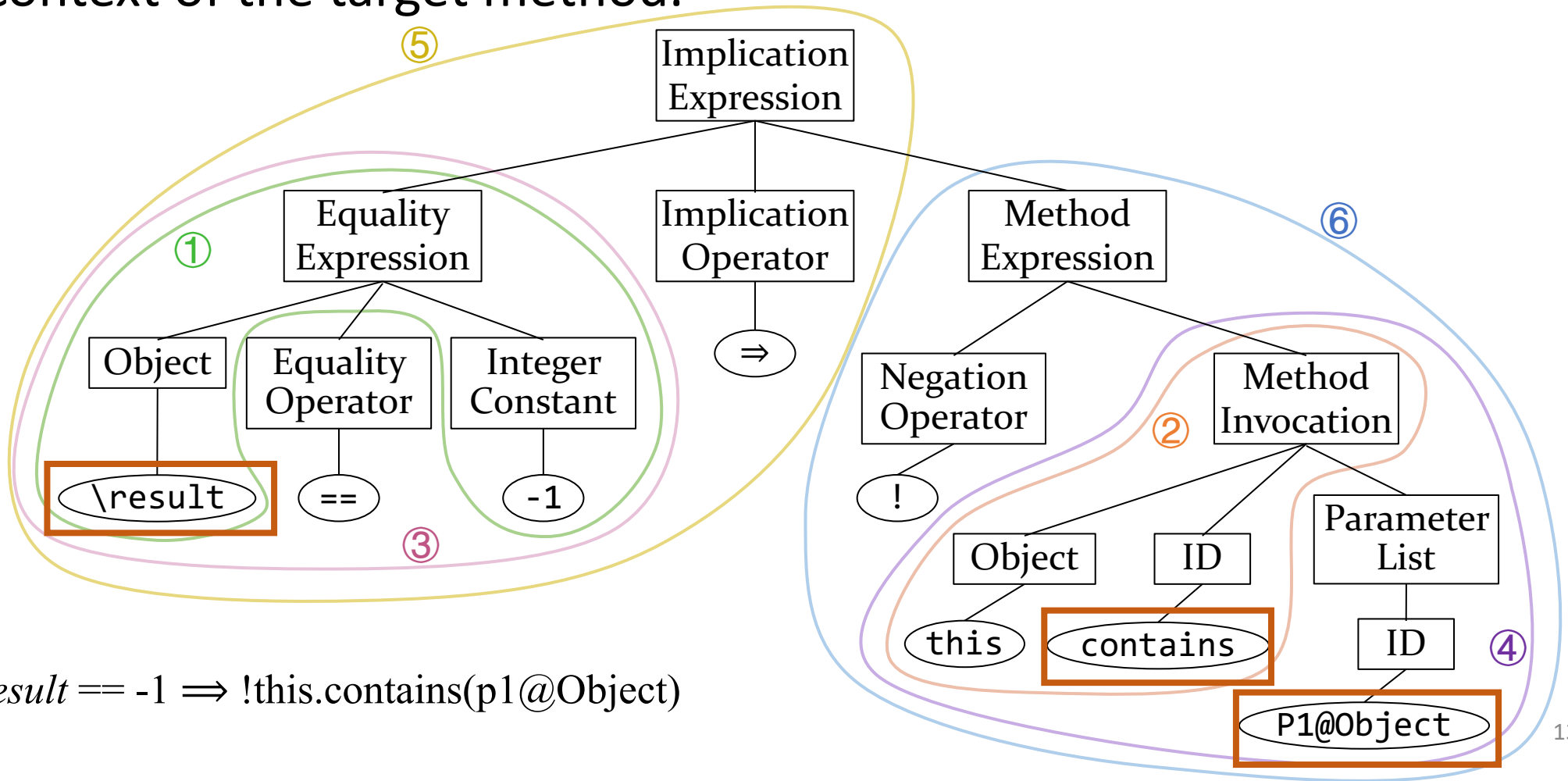
Transform a comment-specification pair into pairs of NL words and AST tokens.

Word	Token
remove	\forall, equals, \old, get, &&, …
return	\result, \old, ==, this, get, …
element	this, null, equals, get, p1@int, …
if	throw, ⇒, &&, ||, !, …
add	contains, this, \result, true, p1@E, …
empty	isEmpty, this, size, 0, ==, …
first	0, this, get, \old, ==, …
...	...

Word-Token Pairs

IR Synthesizer

- Generate IR candidates by assembling tokens based on syntax rules and the context of the target method.



$\backslash result == -1 \Longrightarrow$!this.contains(p1@Object)

Specification Generator

- Generate a formal specification by instantiating each IR candidate with the context of the target method (e.g., parameters of the method)

```
/** Adds the specified vertex to this graph if not already present.
  * @param v - vertex to be added to this graph.
  * @returns true if this graph did not already contain the specified vertex.   */
 public boolean addVertex(V v){ ... }
```

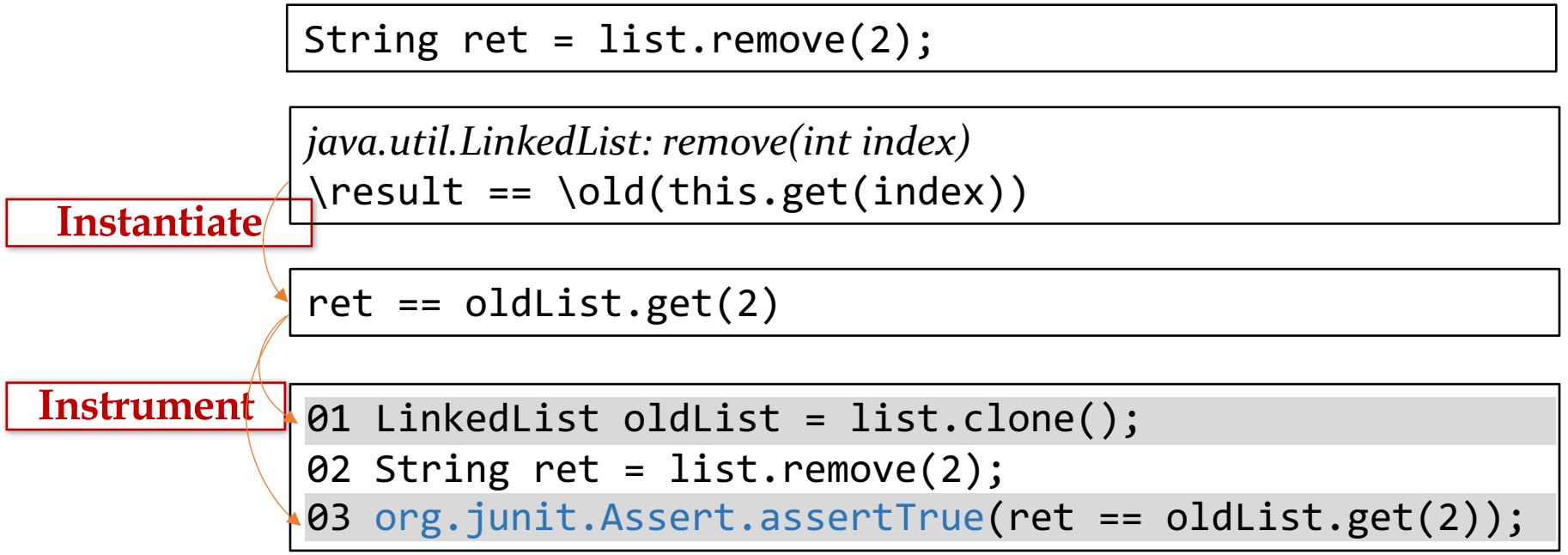
Java Documentation of Method DirectedAcyclicGraph.addVertex(V)

IR: `this.contains(p1@E)`

Specification: `this.containsVertex(v)`

Specification Checker

- Eliminate invalid specification candidates by leveraging exsiting developer test cases.



Evaluation Setup

- Hardware
  - CPU: Intel® i5-8259U
  - RAM: 8GB
- Operating System
  - MacOS High Sierra 10.13.6
- JDK version: 8

Generated Specification Summary

Project	#Class	#Method	#Pre	#Except Post	#Nor Post
JDK 8.0	10	201	64	99	348
Commons Collections 4.1	27	170	140	115	187
Guava 19	8	81	10	13	98
GraphStream 1.3	4	25	0	6	32
JGraphT 0.9.2	15	34	4	10	19
Total	64	511	218	243	684

- In the 5 projects, 1,145 specifications for 511 methods of 64 classes are generated.
- C2S is cross-project, given that the search space of IR tokens are extracted only from project JDK.

Specification Precision and Recall

Tool	Pre		Except Post		Normal Post				Overall	
					Return		Non-return			
	P	R	P	R	P	R	P	R	P	R
@tComment	0.98	0.64	0.80	0.18	n.a.	0.00	n.a.	0.00	0.91	0.26
Toradocu	n.a.	0.00	0.58	0.42	n.a.	0.00	n.a.	0.00	0.58	0.41
Jdoctor	0.94	0.92	0.93	0.77	0.66	0.39	n.a.	0.00	0.85	0.76
C2S	0.98	0.97	0.98	0.91	0.93	0.90	0.92	0.88	0.96	0.91

- `@tComment` does not handle normal post-conditions.
- `Toradocu` does not generate pre-conditions or normal post-conditions.
- `Jdoctor` does not generate normal post-conditions that are unrelated to return values.
- The precision of `C2S` is comparable with the state-of-the-art approaches while the recall of `C2S` is substantially higher.

Improving Automatic Test Case Generation

Project	Jdoctor					C2S				
	#FC	#TA	#FA	%	#NO	#FC	#TA	#FA	%	#NO
JDK 8.0	60	40	20	33.33%	9	40	40	5	12.50%	178
Collections 4.1	105	30	75	71.43 %	17	30	30	10	33.33 %	106
Guava 19	20	10	10	50.00 %	2	10	10	0	0.00 %	22
GraphStream 1.3	20	20	0	0.00 %	1	20	20	0	0.00 %	12
JGraphT 0.9.2	114	64	50	43.86 %	0	64	64	10	15.63 %	5
Total	356	189	167	46.91 %	29	189	189	25	13.23 %	323

- Our specifications lead to much lower false alarm than Jdoctor's specifications do (12.23% vs. 46.91% on average).
- The number of new oracles generated using our specifications is much higher than that of Jdoctor (323 vs. 29 on in total).

Improving Identifying Leak Paths

APK	No TW		Jdoctor TW			C2S TW		
	#P	#T	#P	#T	#ATP	#P	#T	#ATP
ArrayAccess1	1	3.85	1	4.56	0	1	5.37	0
Alipay	39	1450.89	40	1482.32	1	52	1542.43	13
Broncos News	1	32.26	1	30.35	0	5	32.35	4
OpenTable	32	1251.142	32	1253.52	0	47	1799.66	15
Wikipedia	0	5.20	0	5.20	0	2	9.43	2
TencentNews	89	1061.75	89	1293.74	0	93	1305.57	4
DroidKungFu	1	11.02	1	19.41	0	5	27.12	4
santander	4	11.86	4	14.20	0	5	16.85	1
enriched1	1	4.83	1	5.08	0	1	5.50	0
Avira Antivirus Security	20	1682.14	26	1786.43	6	27	1927.86	7
Total	188	-	195	-	7	238	-	50

Our specifications can identify much more leak paths than Jdoctor's specifications (50 vs. 7)

Related Work

- Specification Inference from Natural Language Comments
  - Blasi [ISSTA ’18], Zhou [ICSE ’17], Goffi [ISSTA ’16], Pandita [ICSE ’12], Tan [ICST ’12], Tan [ICSE ’11, SOSP ’07]
- Specification Inference from Code
  - Astorga [DSN ’18], Nguyen [ESEC/FSE ’14], Cousot [VMCAI ’13], Seghir [ESOP ’13], Csallner [ICSE ’08], Ernst [TSE ’01]

Conclusion

- We propose an automatic technique to derive formal program specifications from natural language comments.
  - Assemble primitive tokens guided by specification syntax and properties of the target method
- Generated specifications can improve other software engineering tasks.
  - Automated testing
  - Static taint analysis

Thank You