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

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

```java
/**
* 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

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.

```java
/**
 * @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.

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

(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.

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

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

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

11 /** @public_exceptional_behavior
12 * @requires index < 0 || index >= this.size();
13 * @signals_only java.lang.IndexOutOfBoundsException;
14 * @public_normal_behavior
15 * @requires 0 <= index && index < this.size();
16 * @ensures \result == \old(this.get(index));
17 * @ensures \forallall i; index<=i && i<\old(this.size()-1);
18 public Object remove(int index){ ... }
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” \(\rightarrow\) “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 $\pi_i@t$ (the $i$-th parameter with type $t$).
<table>
<thead>
<tr>
<th>ID</th>
<th>Natural Language Comment</th>
<th>Formal Program Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Removes the element at the specified position in this list</td>
<td><code>\forall\text{int }i; \text{index}&lt;=\text{old}(\text{this.size()}-1)\text{; this.get}(i)==null \&amp;\&amp; \text{old}(\text{this.get}(i+1))==null \mid \text{this.get}(i).equals(\text{old}(\text{this.get}(i+1)));</code></td>
</tr>
<tr>
<td>ii</td>
<td>Returns the element that was removed from the list</td>
<td><code>\text{result}=\text{old}(\text{this.get}(\text{index}))</code></td>
</tr>
<tr>
<td>iii</td>
<td>Throws IndexOutOfBoundsException if the index is out of range (index &lt; 0</td>
<td></td>
</tr>
</tbody>
</table>

Comment-Specification Pairs

<table>
<thead>
<tr>
<th>Word</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove</td>
<td><code>\forall</code>, <code>equals</code>, <code>\old</code>, <code>get</code>, <code>&amp;&amp;</code>, <code>...</code></td>
</tr>
<tr>
<td>return</td>
<td><code>\result</code>, <code>\old</code>, <code>==</code>, <code>this</code>, <code>get</code>, <code>...</code></td>
</tr>
<tr>
<td>element</td>
<td><code>this</code>, <code>null</code>, <code>equals</code>, <code>get</code>, <code>p1@int</code>, <code>...</code></td>
</tr>
<tr>
<td>if</td>
<td><code>throw</code>, <code>=</code>, <code>&amp;&amp;</code>, `</td>
</tr>
<tr>
<td>add</td>
<td><code>contains</code>, <code>this</code>, <code>\result</code>, <code>true</code>, <code>p1@E</code>, <code>...</code></td>
</tr>
<tr>
<td>empty</td>
<td><code>isEmpty</code>, <code>this</code>, <code>size</code>, <code>0</code>, <code>==</code>, <code>...</code></td>
</tr>
<tr>
<td>first</td>
<td><code>0</code>, <code>this</code>, <code>get</code>, <code>\old</code>, <code>==</code>, <code>...</code></td>
</tr>
</tbody>
</table>

Word-Token Pairs

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

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

\[ \text{\}result \} == -1 \Rightarrow !\text{this.contains}(p1@Object) \]
Specification Generator

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

```java
/** 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 existing developer test cases.

```java
String ret = list.remove(2);

java.util.LinkedList: remove(int index)
\result == \old(this.get(index))
ret == oldList.get(2)
```

```java
01 LinkedList oldList = list.clone();
02 String ret = list.remove(2);
03 org.junit.Assert.assertTrue(ret == oldList.get(2));
```
Evaluation Setup

• Hardware
  • CPU: Intel® i5-8259U
  • RAM: 8GB

• Operating System
  • MacOS High Sierra 10.13.6

• JDK version: 8
Generated Specification Summary

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

- 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

<table>
<thead>
<tr>
<th>Tool</th>
<th>Pre</th>
<th>Except Post</th>
<th>Normal Post</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return</td>
<td>Non-return</td>
</tr>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>@tComment</td>
<td>0.98</td>
<td>0.64</td>
<td>0.80</td>
<td>0.18</td>
</tr>
<tr>
<td>Toradocu</td>
<td>n.a.</td>
<td>0.00</td>
<td>0.58</td>
<td>0.42</td>
</tr>
<tr>
<td>Jdoctor</td>
<td>0.94</td>
<td>0.92</td>
<td>0.93</td>
<td>0.77</td>
</tr>
<tr>
<td>C2S</td>
<td>0.98</td>
<td>0.97</td>
<td>0.98</td>
<td>0.91</td>
</tr>
</tbody>
</table>

- @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

- 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

<table>
<thead>
<tr>
<th>APK</th>
<th>No TW</th>
<th>Jdoctor TW</th>
<th>C2S TW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#P</td>
<td>#T</td>
<td>#P</td>
</tr>
<tr>
<td>ArrayAccess1</td>
<td>1</td>
<td>3.85</td>
<td>1</td>
</tr>
<tr>
<td>Alipay</td>
<td>39</td>
<td>1450.89</td>
<td>40</td>
</tr>
<tr>
<td>Broncos News</td>
<td>1</td>
<td>32.26</td>
<td>1</td>
</tr>
<tr>
<td>OpenTable</td>
<td>32</td>
<td>1251.142</td>
<td>32</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>0</td>
<td>5.20</td>
<td>0</td>
</tr>
<tr>
<td>TencentNews</td>
<td>89</td>
<td>1061.75</td>
<td>89</td>
</tr>
<tr>
<td>DroidKungFu</td>
<td>1</td>
<td>11.02</td>
<td>1</td>
</tr>
<tr>
<td>santander</td>
<td>4</td>
<td>11.86</td>
<td>4</td>
</tr>
<tr>
<td>enriched1</td>
<td>1</td>
<td>4.83</td>
<td>1</td>
</tr>
<tr>
<td>Avira Antivirus Security</td>
<td>20</td>
<td>1682.14</td>
<td>26</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>188</strong></td>
<td><strong>-</strong></td>
<td><strong>195</strong></td>
</tr>
</tbody>
</table>

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