Can Large Language Models Transform Natural Language Intent into Formal Method Postconditions?

Madeline Endres (University of Michigan)
Sarah Fakhoury, Saikat Chakraborty, Shuvendu Lahiri (Microsoft Research)
Software requirements are often specified informally.

Informal + Ambiguous

Consume/define requirements

Write an implementation

Program

Formal (yet operational)
Software requirements are often specified informally

- Natural Language
  - Docstring
  - API Ref.
  - RFC

Informal + Ambiguous

Consume/define requirements

Write an implementation

Program

Significant gap ("what" vs. "how")

Formal (yet operational)
Software requirements are often specified informally

Further compounded by AI use

Significant gap ("what" vs. "how")

Formal (yet operational)
Proposal: Formal specifications can reduce the gap

Consumes/defines requirements

Write an implementation

Natural Language
- Docstring
- API Ref.
- RFC

Specifications

Informal + Ambiguous

Formal (declarative)

Formal (yet operational)
Proposal: Formal specifications can reduce the gap

Informal + Ambiguous

Smaller gap (both capture “what”)

Formal (declarative)

Enforceable by PL methods (tests, formal verification, refinement)

Formal (yet operational)

Natural Language

Docstring  API Ref.  RFC

Specifications

Program

Consume/define requirements

Write an implementation
Proposal: Formal specifications can reduce the gap

Formal (yet operational)

Smaller gap (both capture “what”)

Enforceable by PL methods (tests, formal verification, refinement)

Formal (declarative)

Informal + Ambiguous

Consume/define requirements

Write an implementation

Natural Language

Docstring  API Ref.  RFC

Program
Motivating example

[1,2,3,2,4] -> [1,3,4]

def remove_duplicates(numbers: List[int]):
    """ From a list of integers, remove all elements that occur more than once,
    Keep order of elements left the same as in the input."""
Motivating example

```
[1,2,3,2,4] -> [1,3,4]
```

```python
def remove_duplicates(numbers: List[int]):
    
    """ From a list of integers, remove all elements that occur more than once,  
    Keep order of elements left the same as in the input. """
```

Formal Specifications in Python

```
assert len(set(numbers)) == len(set(return_list))
```
Motivating example

```
[1,2,3,2,4] -> [1,3,4]
```

```python
def remove_duplicates(numbers: List[int]):
    """ From a list of integers, remove all elements that occur more than once, 
    Keep order of elements left the same as in the input."""
```

**Formal Specifications in Python**

```
assert len(set(numbers)) == len(set(return_list))
```
Motivating example

```python
def remove_duplicates(numbers: List[int]):
    # """From a list of integers, remove all elements that occur more than once,
    # Keep order of elements left the same as in the input."""
```

**Formal Specifications in Python**

```python
assert len(set(numbers)) == len(set(return_list))
```

```python
assert all(numbers.count(i) == 1 for i in return_list)
```
Motivating example

def remove_duplicates(numbers: List[int]):
    """From a list of integers, remove all elements that occur more than once,
    Keep order of elements left the same as in the input."""

assert len(set(numbers)) == len(set(return_list))

assert all(numbers.count(i) == 1 for i in return_list)

assert all(i in return_list for i in numbers if numbers.count(i) == 1)
Problem formulation

• Given
  • NL description $nl$ for a method $m$

• Generate a postcondition $S$ of $m$ from $nl$

Research Questions:
1. Benchmark and metrics
   1. How do we characterize if a specification $S$ captures the intent in $nl$?
   2. How good are LLMs at user-intent-formalization?
2. What are good real-world application of user-intent-formalization?
Contributions

1. **Semantics-based metrics** for evaluating user-intent-formalization (similar to code generation)

2. **Empirical evaluation of LLMs** for the task of user-intent-formalization

3. Application: **Finding historical real-world bugs**
Problem formulation (ideal)

• Given
  • NL description $nl$ for a method $m$
  • (hidden) reference implementation $I$

• Generate a postcondition $S$ of $m$ from $nl$

• Evaluation metrics (intuition)
  • **Soundness**: $I$ satisfies $S$
  • **Completeness**: $S$ discriminates $I$ from any buggy implementations
Problem formulation (based on tests)

• Given
  • NL description \( nl \) for a method \( m \)
  • (hidden) reference implementation \( I \) + a set of input/output tests \( T \)

• Generate a postcondition \( S \) of \( m \) from \( nl \)

• Evaluation metrics (intuition)
  • **Test-set Soundness**: \( S \) is consistent with \( I \) for each test \( t \) in \( T \)
  • **Test-set Completeness**: \( S \) discriminates \( I \) from any buggy implementations on some test \( t \) in \( T \)

• Score = 
  \[
  \begin{cases}
    0 & \text{if unsound} \\
    \frac{|\text{buggy mutants discriminated}|}{|\text{mutants}|} & \text{else}
  \end{cases}
  \]
Buggy mutant generation

Leverage LLMs!
1. Prompt GPT-3.5 to enumerate 200 solutions to nl prompt
2. Group mutants by the subset of tests in \(T\) they pass [natural bugs]
3. If too few distinct mutants,
   1. Prompt GPT-3.5 to enumerate 200 “buggy” solutions to nl prompt
   2. Group mutants by the subset of tests in \(T\) they pass [artificial bugs]
Buggy mutant generation

Leverage LLMs!
1. Prompt GPT-3.5 to enumerate 200 solutions to nl prompt
2. Group mutants by the subset of tests in $T$ they pass [natural bugs]
3. If too few distinct mutants,
   1. Prompt GPT-3.5 to enumerate 200 “buggy” solutions to nl prompt
   2. Group mutants by the subset of tests in $T$ they pass [artificial bugs]

Hypothesis
• More space of mutations (compared to traditional mutant generation through mutating program elements)
• More natural and subtly incorrect mutants?
RQ1: How good are LLMs at generating specs from Natural Language?

*Evaluation Methodology: EvalPlus*


For each problem in HumanEvall, we used LLMs to generate a set of postconditions. We consider the following ablations¹:

1. Model (GPT 3.5 and GPT 4 and StarCoder)
2. Prompting with NL only vs. NL + reference solution
RQ1: Postcondition *Soundness*

### Evaluation

<table>
<thead>
<tr>
<th>Model</th>
<th>Prompt</th>
<th>Prompt has: NL Only=(\times) ref code=✓</th>
<th>Accept @ 1</th>
<th>Accept @ 5</th>
<th>Accept @ 10</th>
<th>x/164 correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-3.5</td>
<td>base</td>
<td>(\times)</td>
<td>0.46</td>
<td>0.80</td>
<td>0.87</td>
<td>143</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>base</td>
<td>✓</td>
<td>0.49</td>
<td>0.81</td>
<td>0.88</td>
<td>145</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>simple</td>
<td>(\times)</td>
<td>0.55</td>
<td>0.82</td>
<td>0.87</td>
<td>143</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>simple</td>
<td>✓</td>
<td>0.56</td>
<td>0.82</td>
<td>0.88</td>
<td>144</td>
</tr>
<tr>
<td>GPT-4</td>
<td>base</td>
<td>(\times)</td>
<td>0.63</td>
<td>0.83</td>
<td>0.88</td>
<td>144</td>
</tr>
<tr>
<td>GPT-4</td>
<td>base</td>
<td>✓</td>
<td>0.71</td>
<td>0.89</td>
<td>0.91</td>
<td>150</td>
</tr>
<tr>
<td>GPT-4</td>
<td>simple</td>
<td>(\times)</td>
<td>0.77</td>
<td>0.94</td>
<td>0.96</td>
<td>158</td>
</tr>
<tr>
<td>GPT-4</td>
<td>simple</td>
<td>✓</td>
<td>0.76</td>
<td>0.92</td>
<td>0.96</td>
<td>157</td>
</tr>
<tr>
<td>StarChat</td>
<td>base</td>
<td>(\times)</td>
<td>0.21</td>
<td>0.61</td>
<td>0.82</td>
<td>134</td>
</tr>
<tr>
<td>StarChat</td>
<td>base</td>
<td>✓</td>
<td>0.20</td>
<td>0.59</td>
<td>0.77</td>
<td>126</td>
</tr>
<tr>
<td>StarChat</td>
<td>simple</td>
<td>(\times)</td>
<td>0.25</td>
<td>0.69</td>
<td>0.85</td>
<td>139</td>
</tr>
<tr>
<td>StarChat</td>
<td>simple</td>
<td>✓</td>
<td>0.23</td>
<td>0.67</td>
<td>0.86</td>
<td>141</td>
</tr>
</tbody>
</table>
RQ1: Postcondition Completeness

<table>
<thead>
<tr>
<th>Model</th>
<th>Prompt</th>
<th>Prompt has: NL Only=✓ ref code=✓</th>
<th>% bug-complete</th>
<th>% problems with bug-complete</th>
<th>% problems union bug-complete</th>
<th>Avg bug-completeness-score for correct postconditions</th>
<th>All bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-3.5</td>
<td>base</td>
<td>X</td>
<td>15.4</td>
<td>42.1</td>
<td>48.2</td>
<td>0.62</td>
<td>0.72</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>base</td>
<td>✓</td>
<td>18.5</td>
<td>47.0</td>
<td>49.4</td>
<td>0.70</td>
<td>0.76</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>simple</td>
<td>X</td>
<td>8.1</td>
<td>29.3</td>
<td>33.5</td>
<td>0.44</td>
<td>0.55</td>
</tr>
<tr>
<td>GPT-3.5</td>
<td>simple</td>
<td>✓</td>
<td>14.0</td>
<td>37.2</td>
<td>41.5</td>
<td>0.58</td>
<td>0.62</td>
</tr>
<tr>
<td><strong>GPT-4</strong></td>
<td>base</td>
<td>X</td>
<td>35.1</td>
<td>61.6</td>
<td>62.2</td>
<td>0.81</td>
<td>0.85</td>
</tr>
<tr>
<td>GPT-4</td>
<td>base</td>
<td>✓</td>
<td>34.9</td>
<td>58.0</td>
<td>61.6</td>
<td>0.78</td>
<td>0.82</td>
</tr>
<tr>
<td>GPT-4</td>
<td>simple</td>
<td>X</td>
<td>9.2</td>
<td>26.2</td>
<td>29.3</td>
<td>0.40</td>
<td>0.52</td>
</tr>
<tr>
<td>GPT-4</td>
<td>simple</td>
<td>✓</td>
<td>8.9</td>
<td>29.3</td>
<td>36.0</td>
<td>0.47</td>
<td>0.56</td>
</tr>
<tr>
<td>StarChat</td>
<td>base</td>
<td>X</td>
<td>0.8</td>
<td>7.3</td>
<td>8.5</td>
<td>0.13</td>
<td>0.24</td>
</tr>
<tr>
<td>StarChat</td>
<td>base</td>
<td>✓</td>
<td>1.4</td>
<td>9.1</td>
<td>11.0</td>
<td>0.23</td>
<td>0.30</td>
</tr>
<tr>
<td>StarChat</td>
<td>simple</td>
<td>X</td>
<td>1.5</td>
<td>6.7</td>
<td>7.3</td>
<td>0.16</td>
<td>0.24</td>
</tr>
<tr>
<td><strong>StarChat</strong></td>
<td>simple</td>
<td>✓</td>
<td><strong>3.0</strong></td>
<td><strong>17.1</strong></td>
<td><strong>17.7</strong></td>
<td><strong>0.23</strong></td>
<td><strong>0.36</strong></td>
</tr>
</tbody>
</table>
Common postcondition categories on HumanEval

<table>
<thead>
<tr>
<th>Category</th>
<th>Example Postcondition</th>
<th>% Prevalent</th>
<th>Avg. Bug-complete-score (Natural/All)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type Check</td>
<td><code>isinstance(return_val, int)</code></td>
<td>47.4</td>
<td>0.14 / 0.27</td>
</tr>
<tr>
<td>Format Check</td>
<td><code>return_val.startswith(&quot;ab&quot;)</code></td>
<td>11.2</td>
<td>0.43 / 0.57</td>
</tr>
<tr>
<td>Arithmetic Bounds</td>
<td><code>return_val &gt;= 0</code></td>
<td>30.8</td>
<td>0.23 / 0.34</td>
</tr>
<tr>
<td>Arithmetic Equality</td>
<td><code>return_val[0] == 2 * input_val</code></td>
<td>17.5</td>
<td><strong>0.82 / 0.89</strong></td>
</tr>
<tr>
<td>Container Property</td>
<td><code>len(return_val) &gt; len(input_val)</code></td>
<td>27.0</td>
<td>0.45 / 0.57</td>
</tr>
<tr>
<td>Element Property</td>
<td><code>return_val[0] % 2 == 0</code></td>
<td>12.6</td>
<td>0.39 / 0.53</td>
</tr>
<tr>
<td>Forall-Element Property</td>
<td><code>all(ch.isalpha() for ch in return_val)</code></td>
<td>8.3</td>
<td>0.23 / 0.44</td>
</tr>
<tr>
<td>Implication</td>
<td><code>(return_val==False) if 'A' not in string</code></td>
<td>12.7</td>
<td>0.58 / 0.64</td>
</tr>
<tr>
<td>Null Check</td>
<td><code>return_val is not None</code></td>
<td>4.4</td>
<td>0.40 / 0.50</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td><strong>0.32 / 0.46</strong></td>
<td></td>
</tr>
</tbody>
</table>
**RQ2:** Can GPT-4 generated specifications find real-world bugs?

Evaluate on **Defects4J** dataset of real-world bugs and fixes in mature Java projects

Our postconditions leverage functional Java syntax introduced in Java 8. Not all bugs in Defects4J are Java 8 syntax compatible.

Our NL2Spec Defects4J subset contains 525 bugs from 11 projects. These bugs implicate 840 buggy Java methods.

[Defects4J]: a database of existing faults to enable controlled testing studies for Java programs. 2014. Rene Just, Darioush Jalali, Michael Ernst]
RQ2: **Bug Finding: Experiments**

We use GPT-4 to generate 10 postconditions and 10 preconditions for each **buggy** function.

We consider two ablations (33,600 total GPT-4 calls)
- NL + Buggy Method Code + Relevant File Context
- NL + Relevant File Context

For each, we measure:

<table>
<thead>
<tr>
<th>Correctness</th>
<th>Bug-discriminating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does the <strong>spec</strong> pass the tests on <strong>correct code</strong>?</td>
<td>If it is correct, does the <strong>spec fail</strong> any of the tests on <strong>buggy code</strong>?</td>
</tr>
</tbody>
</table>
Defects4J results

<table>
<thead>
<tr>
<th>Model</th>
<th>Prompt has: NL Only = x buggy code = ✓</th>
<th>Compiles</th>
<th>Test-set correct</th>
<th># distinguishable bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>@1</td>
<td>@5</td>
<td>@10</td>
</tr>
<tr>
<td>GPT-4</td>
<td>x</td>
<td>0.65</td>
<td>0.86</td>
<td>0.89</td>
</tr>
<tr>
<td>GPT-4</td>
<td>✓</td>
<td>0.73</td>
<td>0.90</td>
<td>0.93</td>
</tr>
<tr>
<td>StarChat</td>
<td>x</td>
<td>0.25</td>
<td>0.68</td>
<td>0.83</td>
</tr>
<tr>
<td>StarChat</td>
<td>✓</td>
<td>0.29</td>
<td>0.72</td>
<td>0.84</td>
</tr>
</tbody>
</table>

Across ablations, **65 bugs (12.5% of all bugs) are plausibly caught by generated specifications**

- We manually verify a subset of bug catching conditions
Defects4J results

<table>
<thead>
<tr>
<th>Model</th>
<th>Prompt has:</th>
<th>Compiles</th>
<th>Test-set correct</th>
<th># distinguishable bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NL Only = x</td>
<td>@1 0.65</td>
<td>@5 0.86</td>
<td>@1 0.32</td>
</tr>
<tr>
<td>GPT-4</td>
<td>buggy code = ✓</td>
<td>@10 0.89</td>
<td></td>
<td>@5 0.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>@10 0.66</td>
</tr>
<tr>
<td>GPT-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StarChat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>StarChat</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Across ablations, **65 bugs (12.5% of all bugs) are plausibly caught** by generated specifications

- We manually verify a subset of bug catching conditions

Complementary to prior assertion generation approaches TOGA [Dinella, Ryan, Mytkowicz, Lahiri, ICSE’22] and Daikon [Ernst et al. ICSE’99]

- TOGA mostly finds expected exceptional bugs. TOGA can only tolerate bugs during testing, and cannot prevent bugs in production.
- Daikon specs overfit the regression tests and bug-discriminating specs are unsound
RQ2: Example triggered bug from **Defects4J**

```java
/*
 * @param sb The StringBuffer to place the rendered text into.
 * @param width The number of characters to display per line
 * @param nextLineTabStop The position on the next line for the first tab.
 * @param text The text to be rendered.
 * @return the StringBuffer with the rendered Options contents.
 */

protected StringBuffer renderWrappedText(StringBuffer sb, int width, int nextLineTabStop, String text) {
  int pos = findWrapPos(text, width, 0);
  if (pos == -1) {
    sb.append(rtrim(text));
    return sb;
  }

  sb.append(rtrim(text.substring(0, pos))).append(defaultNewline);

  final String padding = createPadding(nextLineTabStop);
  final String padding = createPadding(0);

  while (true) {
    text = padding + text.substring(pos).trim();
  }

  // All lines must be less than or equal to the specified width
  assert returnValue.toString().lines().allMatch(line -> line.length() <= width);
}
```

https://issues.apache.org/jira/browse/CLI-151
Ongoing works around user-intent-formalization

Evaluating user-intent-formalization for verification-aware languages (Verus, Dafny, F*) [Lahiri FMCAD’24]


Real-world application on generating verified parsers through user-intent-formalization of RFC documents