May LLMs tell how my system can go wrong?

Shing-Chi Cheung
What if seeking assistance over a buggy program?

The function $p_1(a, b)$ is a recursive function that takes two arguments $a$ and $b$. If $b$ is equal to 0, the function returns $a$. Otherwise, it calls itself with arguments $a$ and $a \% b$. This process continues until $b$ is equal to 0. The function then returns the value of $a$.

Here is the code for the function:

```python
#correct version
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

```python
def p1(a, b):
    if b == 0:
        return a
```
The $p_1(a,b)$ function ultimately returns the greatest common divisor (GCD) of two numbers $a$ and $b$. The function uses the Euclidean algorithm to compute the GCD recursively. The Euclidean algorithm is based on the principle that the GCD of two numbers does not change if the larger number is replaced by its remainder when divided by the smaller number. The algorithm continues until the remainder is 0, at which point the GCD has been found.
The intention of the $p_1(a, b)$ function is to find the greatest common divisor (GCD) of two numbers $a$ and $b$. The function uses the Euclidean algorithm to compute the GCD recursively. The Euclidean algorithm is based on the principle that the GCD of two numbers does not change if the larger number is replaced by its remainder when divided by the smaller number. The algorithm continues until the remainder is 0, at which point the GCD has been found.
Yes, the \( p_1(a, b) \) function is correct and will return the greatest common divisor (GCD) of two numbers \( a \) and \( b \). The function uses the Euclidean algorithm to compute the GCD recursively. The Euclidean algorithm is a well-known and widely used algorithm for computing the GCD of two numbers.

Can we trust Bing’s assessment on our code?
Sure! Here is a test case for p1:
Test case: a = 10 b = 25
Expected output: 5
The output of p1 when called with arguments a = 10 and b = 25 will be 5.

Can we trust the evidence given by Bing?

The actual output of p1(10, 25) is 10.
When \( p1(0, 0) \) is executed, the function will enter the if statement and return \( a \). The statement coverage of the test case is 100\% because both the if and else statements are executed at least once.

The branch coverage of the test case is 50\% because only one branch of the if-else statement is executed.

def \( p1(a, b) \):
    if \( b == 0 \):
        return \( a \)
    else:
        return \( p1(a, a \% b) \)

A 100\% statement coverage of \( p1 \) implies 100\% branch coverage.
Sure! A failure-inducing test case for $p1$ would be to call the function with arguments $a = 0$ and $b = 0$. The expected output for this test case is undefined because the GCD of 0 and 0 is not well-defined.

Another failure-inducing test case would be to call the function with arguments $a = 10$ and $b = -25$. The expected output for this test case is also undefined because the GCD of two negative numbers is not well-defined.
I apologize for the confusion. I just realized that I made a mistake in my previous response. A failure-inducing test case for `p1` would be to call the function with arguments `a = 10` and `b = 0`. The expected output for this test case is 10 because the GCD of any number and 0 is the number itself.

```python
def p1(a, b):
    if b == 0:
        return a
    else:
        return p1(a, a % b)
```

Actually, `p1(10, 0)` is not a failure-revealing test case. It returns 10, which is the same as the expected output.
I apologize for the confusion. Here's a correct implementation of the GCD function:

```python
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

This implementation uses the Euclidean algorithm to calculate the GCD of two numbers. The algorithm works by repeatedly taking the remainder of the larger number divided by the smaller number until the remainder is 0. The GCD is then the last non-zero remainder.
Generative Test Framework

Test Target
- Mobile Apps
- Web 3.0
- Application Software
- Foundation
  - TensorFlow
  - PyTorch
  - HarmonyOS
  - Matplotlib
  - MindSpore

Conventional
Deep Learning

Test Input Generation

Test Oracle Generation

Test Migration

Test Artifacts
- Test Inputs
- Mock Object
- Assertions
  - Test Assertions
  - Mock Assertions
- Matched Widgets

Textual Information
- Documentation
- Issue Reports
- Commit History

Existing Test

Note: We do not differentiate fuzzing from testing

4/12/2024
Shing-Chi Cheung
### Comparison on Buggy Subjects

<table>
<thead>
<tr>
<th>Buggy Subjects</th>
<th>FT-IA</th>
<th>FT-la</th>
<th>FT-ia</th>
<th>Passing</th>
<th>Invalid</th>
<th>No Soln</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pynguin [ASE22]</td>
<td>8%</td>
<td>2%</td>
<td>0%</td>
<td>32%</td>
<td>56%</td>
<td>0%</td>
<td>NA</td>
</tr>
<tr>
<td>Direct Use of ChatGPT</td>
<td>29%</td>
<td>19%</td>
<td>8%</td>
<td>30%</td>
<td>15%</td>
<td>0%</td>
<td>43.9%</td>
</tr>
<tr>
<td>Differential Prompting</td>
<td>75%</td>
<td>2%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>23%</td>
<td>97%</td>
</tr>
</tbody>
</table>

**Based on 40 Buggy QuixBugs (before training cutoff)**

<table>
<thead>
<tr>
<th>Buggy Subjects</th>
<th>FT-IA</th>
<th>FT-la</th>
<th>FT-ia</th>
<th>Passing</th>
<th>Invalid</th>
<th>No Soln</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pynguin [ASE22]</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>30%</td>
<td>70%</td>
<td>0%</td>
<td>NA</td>
</tr>
<tr>
<td>Direct Use of ChatGPT</td>
<td>7%</td>
<td>30%</td>
<td>44%</td>
<td>7%</td>
<td>11%</td>
<td>0%</td>
<td>9%</td>
</tr>
<tr>
<td>Differential Prompting</td>
<td>41%</td>
<td>0%</td>
<td>6%</td>
<td>0%</td>
<td>0%</td>
<td>53%</td>
<td>87%</td>
</tr>
</tbody>
</table>

**Based on 7 buggy CodeForces (after training cutoff)**
Summary

- ChatGPT outperforms state-of-the-art
- Test suites play an important role in using LLMs
- ChatGPT can incorrectly summarize a program based on the code that it has learnt
- Using ChatGPT alone to solve common programming tasks is not effective
- Augmenting ChatGPT with SE techniques can significantly boost its capability in solving common programming tasks