

Coverage from Homework 2

https://github.com/CharlesLiu-Umass/CS520_HW3

Part 1: Generate, Evaluate and Refine Specifications

Problem 1:

The first experiment was using code generated by Gemini using the CoT prompting strategy on problem 1, the close elements problem from the humaneval dataset. This experiment had 83.3% line coverage and 87.5% branch coverage. Full details on line 7 of the spreadsheet.

To get the formal specifications for this problem, I asked ChatGPT-5 to generate them. Below is the problem description and the prompt given to ChatGPT-5.

Function Signature: has_close_elements(numbers: List[float], threshold: float) -> bool

Natural Language Description: Check if in given list of numbers, are any two numbers closer to each other than given threshold

LLM Prompt:

Problem description: Write a function that checks if any pair of numbers from a list are closer to each other than a given threshold

Function Signature: has_close_elements(numbers: List[float], threshold: float) -> bool

Please write formal specifications as Python assertions that describe the correct behavior of this method.

Let ‘has_close’ denote the expected return value of ‘has_close_elements(numbers, threshold)’.

Do not call ‘has_close_elements()’ in your assertions.

Do not use methods with side effects such as System.out.println, file I/O, random number generation, or timing functions.

Express the relationship between ‘numbers’, ‘threshold’ and ‘has_close’ using boolean logic and native python to guarantee correctness.

The following code is the generated formal specifications

```
Python
# 1. Main semantic condition
assert has_close == any(
    abs(numbers[i] - numbers[j]) < threshold
    for i in range(len(numbers))
    for j in range(i + 1, len(numbers))
)
```

```

# 2. Fewer than two elements ⇒ must be False
assert not (len(numbers) < 2 and has_close)

# 3. If result is True, threshold must be positive
assert not (has_close and threshold <= 0)

```

All three specifications are correct therefore the accuracy rate is 100%

Problem 2:

The other experiment was code generated by ChatGPT using the SCoT prompting method on problem 8, the s-palendrom problem from APPS. This experiment had 84.6% line coverage and 66% branch coverage. Full details on line 45 on the spreadsheet.

Like the last experiment, to get the formal specifications for this problem, I asked ChatGPT-5 to generate them. Below is the problem description and the prompt given to ChatGPT-5.

Function Signature: `is_s_palindrome(s: str) -> str`

Natural Language Description: Let's call a string "s-palindrome" if it is symmetric about the middle of the string meaning characters at corresponding positions are mirror reflections of each other. For example, the string "oHo" is "s-palindrome", but the string "aa" is not. The string "aa" is not "s-palindrome", because the second half of it is not a mirror reflection of the first half. Return "TAK" if the string s is "s-palindrome" and "NIE" otherwise.

LLM Prompt:

Problem description: Let's call a string "s-palindrome" if it is symmetric about the middle of the string meaning characters at corresponding positions are mirror reflections of each other. For example, the string "oHo" is "s-palindrome", but the string "aa" is not. The string "aa" is not "s-palindrome", because the second half of it is not a mirror reflection of the first half. Return "TAK" if the string s is "s-palindrome" and "NIE" otherwise. Only consider the letters of the English alphabet, both uppercase and lowercase.

Function Signature: `is_s_palindrome(s: str) -> str`

Please write formal specifications as Python assertions that describe the correct behavior of this method.

Let 'result' denote the expected return value of `'is_s_palindrome(s)'`.

Do not call `'is_s_palindrome(s)'` in your assertions.

Do not use methods with side effects such as `System.out.println`, file I/O, random number generation, or timing functions.

Express the relationship between 'result', and 's' using native python to guarantee correctness. Additionally, please generate all possible mirrorings, not just the ones from the examples.

The following code was generated for the specifications of the problem:

```

Python

# mirror mapping describing every valid English letter mirror
mirror = {
    'A': 'A', 'H': 'H', 'I': 'I', 'M': 'M', 'O': 'O', 'T': 'T',
    'U': 'U', 'V': 'V', 'W': 'W', 'X': 'X', 'Y': 'Y',
    'b':'d', 'd':'b', 'i':'i', 'l':'l', 'o':'o', 'v':'v',
    'w':'w', 'x':'x'
}

# condition that must be true for s to be an s-palindrome
is_s_pal = all(
    (c in mirror and mirror[c] == s[len(s) - 1 - i])
    for i, c in enumerate(s)
)

# result must be "TAK" if and only if s is an s-palindrome
assert (result == "TAK") == is_s_pal

# result must be "NIE" exactly when it is not an s-palindrome
assert (result == "NIE") == (not is_s_pal)

```

The generated “mirror” dictionary and “is_s_pal” boolean are implemented correctly which makes both assertions correct as well. Even though the assertions are not standalone it is easy to convert the assertions into stand alone specifications by substituting the variables into the assertion statements; this would be at the cost of readability. The accuracy rate is 100%.

Part 2:

Problem 1:

Prompt:

Here are some specifications for a function that returns has_close and takes in parameters numbers and threshold. Please generate a list of test cases using the specifications given in the form of a python tuple ((input_parameters), (expected_output))

(#Added Specifications from part 1 problem 1)

Generated Test Cases:

Unit tests are of the form ((numbers, threshold), has_close).

Generated Unit Tests	Comments
----------------------	----------

(([1, 2], 2), True)	Correct general case where has_close is true.
(([1, 4, 9], 2), False)	Correct general case where has_close is false.
(([1, 3], 2), False)	Correct corner case where if smallest distance between all elements is equal to threshold then return false
(([10], 5), False)	Correct edge case where list of numbers less than 2 means no two elements can be close. Returns false
(([1, 2, 3], -5), False)	Correct edge case with negative threshold. has_close always returns false independent of numbers.

New Coverage:

Statement coverage: From 83.3% to 100%

Branch coverage: From 87.5% to 100%

Tests: All original tests and spec-guided tests passed.

Coverage increased due to spec-guided tests targeting the edge case when there are less than 2 numbers.

Problem 2:

Prompt:

Here are some specifications for a function that takes a parameter s which is a string and returns results which is also a string. Please generate a list of test cases using the specifications given in the form of a python tuple ((input_parameters), (expected_output)).

(#Added Specifications from part 1 problem 2)

Generated Test Cases:

Unit tests are of the form (s, result). result is essentially a boolean variable where “TAK” = true and “NIE” = false.

Generated Unit Tests	Comments
("aa", "NIE")	Correct test where even though “aa” is a palindrome its characters themselves do not mirror each other.

("HO","NIE")	Correct test where “HO” does not have a line of symmetry.
("oXxO","TAK")	Incorrect unit test. oXxO does not have a line of symmetry. This should return “NIE”
("A","TAK")	Correct test where the character A is symmetrical and a string of length 1 can be an s_palindrome
("TOT","TAK")	Correct case. “TOT” has a line of symmetry.

New Coverage:

Statement coverage: From 84.6% to 100%

Branch coverage: From 66% to 100%

Tests: The generated code for the function is_s_palindrome fails 2 of the spec-guided tests: “aa” and “TOT”. Additionally, it continues to fail many original unit tests.