Formal Verification in Python:
Applying `CrossHair` to Analyze Program Behavior

Earl T. Barr[*]

1 Overview

This coursework challenges students to explore formal verification using `CrossHair`[1], a symbolic execution engine for Python.

Students will:

- Gather a small data set suitable for verifying Python functions.
- Apply `CrossHair` to verify Python functions via symbolic execution.
- Specify function contracts (preconditions, postconditions, and invariants).
- Reflect on the strengths and weaknesses of automated verification through the lens of applying `CrossHair` to your data set.

This coursework provides hands-on experience with **formal verification** and its practical applications in software correctness.

2 Coursework Description

Students will

1. Select at least three Python functions or classes from open-source software or their own projects.
2. Specify contracts (preconditions, postconditions, invariants).
3. Run `CrossHair` to check contract violations.
4. Document their findings, challenges, workarounds for those challenges, and insights in a report.

Note: In addition to checking contracts, `CrossHair` can help you discover and refine them.

3 Step-by-Step Instructions

This section outlines the process for analyzing Python code using `CrossHair`, from selecting code and adding contracts to verifying results and addressing any limitations encountered.

[*]Thanks to Panaki Chakraborty and Hadrien Renaud for their comments and criticisms.

[1]https://github.com/pschanely/CrossHair

3.1 Select Code for Analysis

Students should choose at least three functions that:

- Contain non-trivial logic, e.g., manipulate data structures or strings or perform numerical computations.
- Include branching conditions, e.g., `if-else` logic.
- Have loops or recursion.
- Are written in Python 3 and avoid excessive external dependencies.

3.2 Document, Add Contracts and Verify with CrossHair

Students must first document each function. It's fine to re-use the function's existing documentation, with appropriate acknowledgement. We expect that you will, nearly always, need to augment whatever documentation exists.

Students will annotate their functions with preconditions, postconditions, class invariants and loop invariants using Python's `assert` statement.

Crosshair does support other syntax than Python's `assert`, as documented here. You are free to use any of them. You are, of course, also welcome to define your own decorators, if you wish. Note: The Python community did not ratify PEP 316 and experimentation shows that, despite its documentation claiming that it does recognise PEP 316 syntax, Crosshair appears to ignore it.

Example: Binary Search Verification with `CrossHair`:

```
from typing import List

def binary_search(arr: List[int], x: int) -> int:
    assert arr == sorted(arr), "Precondition: Input list must be sorted."
    left, right = 0, len(arr) - 1
    while left <= right:
        assert (x not in arr[:left] and x not in arr[right + 1:]), "Loop invariant violated."
        mid = (left + right) // 2
        if arr[mid] == x:
            return mid
        elif arr[mid] < x:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

To check a function's postcondition, you must follow *each* call to the function with the postcondition:

```
    result = binary_search(arr, x)
    assert ((result == -1 and x not in arr) or arr[result] == x), "Postcondition failed."
```

3.3 Run CrossHair and Analyze Results

Students will run `CrossHair` on their data set and document:

- Invariants
  - Identify and describe the *preconditions, postconditions, and loop invariants* that CrossHair does not invalidate for your functions.

  - Explain how these invariants align with the intended behavior of the function and what they reveal about its specification.

- Verified Properties
  - Identify and describe any usually partial properties that CrossHair is able to verify for your functions.
  - As above, explain how these properties align with the intended behavior of the function and what they reveal about its specification.

- Limitations and Partial Coverage
  - Describe any limitations encountered during the analysis, such as paths that CrossHair could not fully explore or properties that could not be verified.
  - If your analysis achieves only partial coverage (i.e., explores a subset of the function's paths), document which paths were covered and which were not, and explain how this partial analysis still provides insight into the function's behavior.
  - Acknowledge any under-approximate properties (properties verified for a subset of paths) and discuss their significance.

If CrossHair does discover any counterexamples, document them and explain what they reveal about the function's behavior. Note that finding counterexamples is *not expected* and is not required to achieve a high grade.

3.4 Reflect on Verification Using CrossHair

Reflect on the process of using symbolic execution to analyze the functions. How did it compare to testing with concrete inputs? What challenges did you face, and how did you address them? Discuss any interesting findings or surprises about the function's behavior or specification.

The primary goal is to *propose and analyze meaningful preconditions, postconditions, and loop invariants*, not to find bugs or achieve full path coverage and fully verify your functions.

- **Partial coverage** and **under-approximate properties** are perfectly valid and sufficient for a high grade.
- Documenting your process, findings, and limitations is just as important as the results themselves.
  1. Each time `CrossHair` handles one of your verification goals on one of your functions, explain why the verification goal and the function are representative and important.
  2. Each time `CrossHair` struggles with one of your verification goals on one of your functions, you will
     (a) Describe the problem, *e.g.*, "`CrossHair` fails to verify recursion depth"; and
     (b) Propose a workaround, *e.g.*, rewriting the function or adding invariants, or explain why none is possible.

It is possible, albeit unlikely, that your data set does not expose any of `CrossHair`'s limitations and therefore does not require workarounds. In this case, your task will be to convincingly argue that your data set is representative of an important class of programs.

4 Final Report

The report must include:

Introduction: A *brief* (~ 1 paragraph) overview of `CrossHair`'s role in formally verifying Python code, emphasizing what you have learned about its relevance to ensuring code correctness.

Data Set: A clear justification for your chosen data set, including its source, characteristics, and why it is suitable for verification. Highlight any unique properties or challenges it presents.

Verification Results: Preconditions, postconditions, and findings.

Reflection: Insights on formal verification in Python. This is where you should discuss any `CrossHair` limitations exposed by your data set and workarounds for them.

There is no set report length. 2000–2500 words will be sufficient. Using the Calibri at 10pt with single spacing, this comes to 3–4 pages. We have limited time to assess coursework submissions, so we reserve the right not to read past 2500 words.

5 Grading Criteria

- Selection of interesting programs
- Quality of contract specifications
- Analysis of verification results using `CrossHair`
- The thoroughness and insightfulness of your analysis on verifying Python code with `CrossHair`, including any limitations encountered and potential solutions to address them.
- The clarity, organization, and comprehensiveness of the final report.

When assessing your submission, we will take into account the amount of time you have to do the work, from the release of this problem set to the due date. We understand that this is just one of many demands on your time.

6 Submitting Your Coursework

To submit your coursework, follow these steps:

1. **Create a zip file**: This zip file must include
   (a) **Your Dataset**: Ensure it is properly documented and includes any necessary metadata.
   (b) **Your Scripts**: Submit all scripts used to apply `CrossHair` to your dataset and analyse or visualise the results. Clearly label and organize these scripts for ease of review.
2. **Produce a PDF Version of Your Final Report**: Ensure the report is well-structured, clear, and includes all relevant sections, *e.g.*, introduction, methodology (data set), results, and reflection.
3. **Submit via Moodle**: Upload your zipfile and final report to the designated coursework submission area on Moodle. Ensure all files are properly named and organized.

7 Advice

- **Data Set Challenge:** Gathering a suitable data set is likely the hardest part. Choose data that exposes interesting behavior, even if it pushes `CrossHair`'s limits.
- **Time Management:** Budget $\sim \frac{1}{3}$ of your time for writing the report. Interleave report writing with data collection and analysis to stay organized.
- **Start Small:** Begin with a small data set, fully analyze it, and write a draft report. You may, for instance, wish to write up each function in your data set before continuing to the next one. Expand your work only if time permits.
- **Time Boxing:** Expect to spend $\sim$30 hours in total — plan your tasks and adjust based on your pace.
- **Limitations Are OK:** If `CrossHair`'s limitations prevent your verification goal, document this clearly — it's a valid outcome.