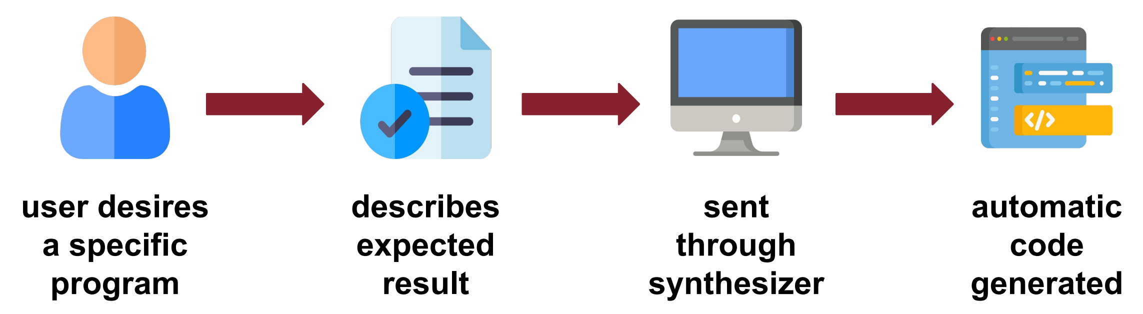
Program Analysis & Explainability

Shivani Wadhwa | shivaniwadhwa723@gmail.com | SHINE Lab
Arcadia High School, Class of 2024
USC Viterbi Department of Computer Science, SHINE 2023

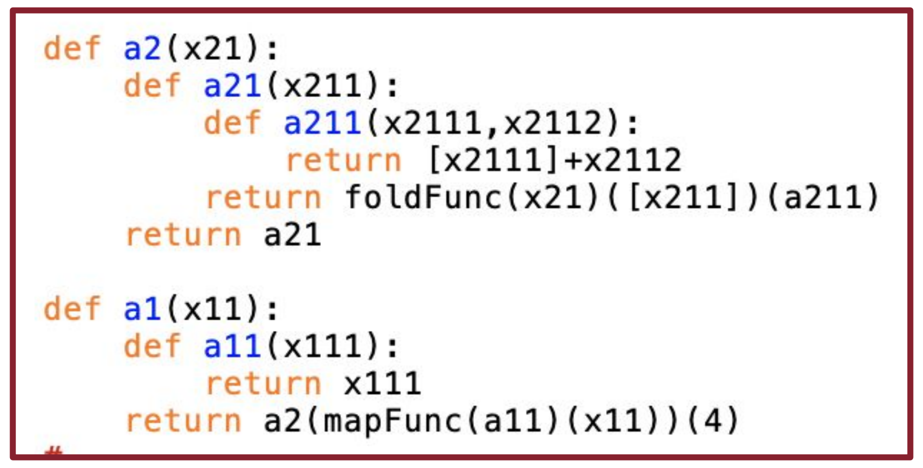
Introduction

Program synthesis is the process of automatically providing a program in a particular language.



For example, a user may provide an input of [4, 3, 5] and an output of [3, 4, 5]. Based on the specification, the synthesizer will return a program that sorts lists.

The issue lies in the **complexity** of the sub-functions employed by synthesizers. Users find difficulty in understanding their purpose within the program and are unable to have complete confidence in the synthesizer.



```
def a2(x21):
    def a21(x211):
        def a211(x2111,x2112):
            return [x2111]+x2112
        return foldFunc(x21)([x211])(a211)
    return a21

def a1(x11):
    def a11(x111):
        return x111
    return a2(mapFunc(a11)(x11))(4)
#
```

example code returned from a synthesizer

Objective of Professor's Research

Professor Raghothaman's research places a key emphasis on harnessing the potential of machine learning, program synthesis, and formal methods to **make the job of programmers easier**.

- Formal verification is the task of displaying the accuracy of sub-algorithms in a program that's provided by synthesizers through formal reasoning and mathematical methods.
- Static program analysis demonstrates program properties and behaviors, allowing errors to be easily identified without ever actually running the program.
- Machine learning, an extremely current field and still unknown field in technology, has the power to "learn" from users and data to be able to utilize resources and maximize customization like never seen before.

Project Summary

The solution proposed was to create an accurate algorithm that could read the code returned by the synthesizer, identify the specific purpose of each sub-function in the code, and assign them all a reliable name.

Four specific algorithms were created **(BU, F, R, S)**. Those, along with Open AI's Chat-GPT **(C)** were tested on how accurate and reliable their produced names were.

I went through 200 **higher-order sub-functions** and hand-assigned each one a list of suitable names. To ensure accurate results, I used a number of **methods** to analyze each sub-function:

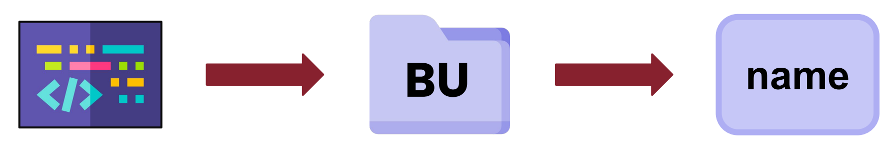
- modifying code with print() statements
- inspecting IO files (input, output)
- code tracing

With these names, I then had to verify my mentor's created names to ensure accuracy and that no bias existed.

Function Name	Name to be Checked	Verification
increment elements by one	increase elements by one	✓
count occurrences of x in list	count occurrences of k	✓
remove all even elements	remove odds	✗

snippet of name verification table

Once each of the 200 sub-functions had its specific list of names, I then sent the sub-functions through each of the five **algorithms**.

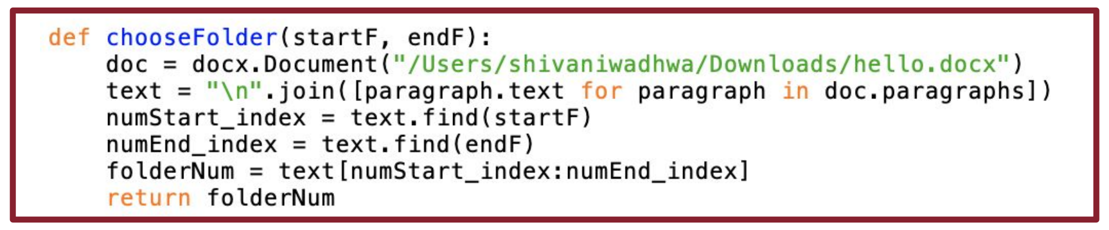


My generated names and the algorithmically-generated name results were **compared** using a **mathematical scale**. Each name was scored from 0 to 1. Names at or above a specific value were counted as "reliable" and "trustworthy".

Analyzing Results

After comparing all 200 sub-functions with each of the five algorithms, I was able to calculate an **average percent accuracy** for each algorithm.

To do this, I wrote a python program in Visual Studio Code that directly accessed data from the document (.docx) file of scores and specifically calculated each different algorithm's average.



```
def chooseFolder(startF, endF):
    doc = docx.Document("/Users/shivaniwadhwa/Downloads/hello.docx")
    text = "\n".join([paragraph.text for paragraph in doc.paragraphs])
    numStart_index = text.find(startF)
    numEnd_index = text.find(endF)
    folderNum = text[numStart_index:numEnd_index]
    return folderNum
```

snippet of written python program

RESULTS

	Comp Results	My Results
BU	85.1%	63.5%
C	22.0%	20.1%
F	83.2%	47.5%
R	79.3%	55.4%
S	58.0%	43.0%

The results I found verified the order of the predicted algorithms in terms of accuracy and reliability. **Note the "C" value (chatGPT) was measurably lower than any of the created algorithms.**

Next Steps

Based on a user-study, these algorithms have proven to be highly beneficial for enhancing programmer understanding. However, the current names merely provide a high-level overview of the code's function. Moving forward, the next crucial step is to update the algorithm to provide detailed explanations for **each line** within the sub-function. This advancement will allow **absolute clarity and comprehension** of the entire program.

Relating to STEM Coursework

While computer scientists often aim to create programs for external causes, my time at USC SHINE introduced me to the idea of designing tools to instead help and enhance a programmer's efficiency. Working with Python's higher-order functions challenged me and forced me to delve deeper into programming fundamentals. I also got to take a MATLAB course and earn certification. Additionally, I learned about the research and paper process and discovered the significance of related works, data from user studies, and abstracts.

Acknowledgements

I would like to thank Professor Mukund Raghothaman for providing me the opportunity to work in his lab and Monica and Marcus for their help in setting up this program for us. I'd also like to thank my lab mentor, PhD student Amirmohammad Nazari for all his help, guidance, and expertise that allowed me to complete this project. Lastly, I'd like to thank my center mentor Minsun for her role in making the whole CS sub-cohort feel welcome and comfortable on campus!

Citations

ALUR, R., BODIK, R., DALLAL, E., FISMAN, D., GARG, P., JUNIWAL, G., KRESS-GAZIT, H., MADHUSUDAN, P., MARTIN, M. M. K., RAGHOTHAMAN, M., SAHA, S., SESHIA, S. A., SINGH, R., SOLAR-LEZAMA, A., TORLAK, E., & UDUPA, A. "Syntax-Guided Synthesis", 2013, https://sygus.org/assets/pdf/Journal_SyGuS.pdf

Ko, A. J., & Myers, B. A. "Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior", 2008, https://faculty.washington.edu/ajko/papers/Ko2008JavaWhyline.pdf