Motivation

Analysing application code is beneficial in a number of different ways. The implementation of variable analysis helps give a better understanding of how a program works, and a larger scale system analysis reveals how different sections of code interact with each other. This project aims to utilise both of these types of analysis on Python source code to achieve the following goals:

- Generate a system dependence graph (SDG): This provides a way to better understand how a program works as well as forming a foundation for further analysis.
- Design an API: This provides a way for the information in the SDG to be accessed and used by other programs.
- Provide information about the variables that are used.
- Create a way to visualise the analysed data.

Basic Block Generation

Python basic blocks are generated in the PBlockGen module. This module uses the Python ast module to break down the source file into its abstract syntax tree (AST). PBBlockGen subclasses ast’s NodeVisitor class in order to efficiently traverse the AST. As it traverses the tree, it builds basic blocks and stores information about them in custom block objects. These objects are designed to optimise the analysis done with them in the later modules.

Control Flow Graph Builder

The control flow graph (CFG) is generated in the GraphBuilder module. A CFG is a graph wherein nodes are basic blocks and whose edges show which blocks can have control flow between them. The GraphBuilder module takes the list of basic blocks from PBBlockGen as its input and iterates through the list to create the graph. It is also able to optimise the block list and CFG, for example removing redundant nodes.

Data Dependence Analysers

Data dependence analysis is done in the Analysers module. Analysers use the block list and CFG from PBBlockGen and GraphBuilder to detect the dependences. It does this by traversing the CFG with a depth first search and looks for instances where a variable is either read from or written to. A link is made when there is a write to and then a read from the same variable. The links are stored in a list of Link objects that is accessible to the API.

Front End Visualiser

The front end visualiser is a command line tool that uses the Analysers API to take a source file, analyse it, and then visualises the data in one of two different ways. The first way is to print all the information to the console or to a specified file. This can be useful for testing but is not very useful for visualising the program. The second way is to create a graph using Graphviz – a third-party graphing tool. The graph visually shows the SDG and helps the user to understand the flow of the program.

Examples

These example demonstrate the SDG analysis of certain control structures. The first example shows the analysis of code made up of functions and function calls. As you can see, the blocks in the SDG are not sequential, rather, a function is only entered when it is called. The second example shows the control flow for if-statements and loops. You can see that the control flow loops back from the LoopFooter to the ForHeader.

```
1 def func(a, b):
2   x = 4
3   def nested_func(c, d):
4     nonlocal x
5     x = 5
6     print("in here")
7     y = 4
8     return x, y
9     nested_func(a, b)
10    print("me")
11   fun2(x, y)
```

Listing 1. A sample program to demonstrate the control flow for function calls

```
1   x = 5
2   if x > 1:
3     for y in range(x):
4       z += 1
5
6   print(x)
```

Listing 2. A sample program to demonstrate the control flow for if-statements and loops

```
Figure 1. The SDG output from the code in Listing 1.
```

Summary

This project achieves its goals of creating a platform for enabling better analysis of Python source code. The SDG generation and visualisation provides a way to visualise the code, thus enabling a greater understanding of how different sections of the code fit together. In addition, the API provides a way for other programs and tools to perform further analysis and visualise the data in different ways.

```
Figure 2. The SDG output from the code in Listing 2.
```

The project is also designed in a way that is scalable and easily enables new features and further improvements. By designing the underlying system architecture in this way, this project can be used a foundation for tools that perform further analysis and do more with the analysed data. Evidently, this project makes successful progress towards the ultimate goal of being able to parallelise Python code running on embedded systems.