Assignment 3: May-Alias Analysis

Introduction to Program Analysis and Compiler Optimization

March 4, 2025

1 Alias Analysis

Alias Analysis (aka Pointer Analysis) is a class of techniques that attempt to determine whether two pointer variables can ever point to the same memory location. There are many different algorithms for alias analysis and many different ways of classifying them: flow-sensitive vs. flow-insensitive, context-sensitive vs. context-insensitive, field-sensitive vs. field-insensitive, unification-based vs. subset-based, etc. Traditionally, alias analyses respond to a query with a Must, May, or No alias response, indicating that two pointers always point to the same object, might point to the same object, or are known to never point to the same object. The MayAlias response is used whenever the two pointers might refer to the same object. You must have already studied **MayAlias** analysis in the class. In this assignment, you have to build your own **intra-procedural**, **flow-sensitive**, **MayAlias** analysis. Your task will be to identify the **MayAlias** relationship between each pointer variable pair at the end of the program.

2 Deliverables

The structure of the given template directory is mentioned in Figure 1. Your task is to transform a given C program into LLVM IR and analyze pointer aliasing using the mayAlias analysis pass.

Files Provided.
`assign.c`: Contains the C program to be converted into LLVM IR.
`output.txt`: Stores the alias analysis results.
`private_test\test.c`: Stores private test cases provided by you (the student).

Task.

- Provide at least three functions as private test cases and corresponding alias analysis results in the file `private_test\test.c`. You should provide the alias analysis results as code comments in the prescribed format only.
- Convert `assign.c` to LLVM IR.

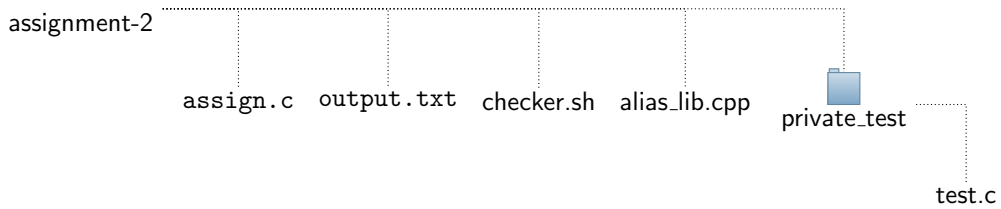

Figure 1: Folder Structure for Deliverables.

- Run your intra-procedural, flow-sensitive **mayAlias** analysis pass on the generated LLVM IR.
- Store alias relationships in `output.txt` file, following the format described below.

Output Format. Each aliasing relationship must be stored as a set in a text file, adhering to these rules:

- Print the function name first, followed by the alias relationships of the pointer variables. Note that only the alias relationships corresponding to the **exit** or **return** instruction of the function, after the analysis has converged, should be printed.
- Each pointer variable must have a set of pointer variables with which it may alias.
- Each set should be formatted as comma-separated pointers inside curly braces (e.g., {`p1, p2, p3`}).
- If a pointer does not alias with any other, print an empty set {}.
- The alias sets must be printed in the order of pointer variable declarations in the original C code.

Assumptions. When analyzing aliasing in a given C code, students can make the following assumptions to simplify the problem:

- All pointers are properly initialized before use. No dangling or wild pointers are present in the program.
- The program does not involve buffer overflows, out-of-bounds accesses, or uninitialized memory reads.
- If pointers are passed to a function, assume they may alias unless explicitly stated otherwise.
- Two pointers referencing two different indices of the same array may alias.

You can check the example in Section 3 for a clear understanding.

3 Example

```
1
2  #include <stdio.h>
3
4  void example() {
5      int x = 10, y = 20;
6      int *p = &x;
7      int *q = &y;
8
9      if (x > 5) {
10         q = p;
11     }
12
13     *p = 30;
14     printf("%d\n", *q);
15 }
16
17 void example2(int *z, int *w)
18 {
19     int a1 = 10, b1 = 20;
20     int *c = &a1, *d = &b1;
21     z = c;
22     w = d;
23 }
24
25 int main()
26 {
27     example();
28     int a = 5, b = 5;
29     example2(&a, &b);
30     return 0;
31 }
```

Figure 2: Example Input C code.

Consider an example in Figure 2. We will first analyze the function `example`, followed by `example2`.

In Lines 6 and 7, the pointer p initially points to variable `x`, and the pointer `q` initially points to variable y. At these lines, `p` and `q` are not aliases since they reference different memory locations. However, after

the conditional statement at Line 9, the assignment `q = p` may execute, causing `q` to point to the same memory location as `p`. As a result, `q` and `p` may become aliases in certain execution paths. Hence, at the exit instruction/return instruction (Line 14) of the function, `example`, we can say that the pointer variables `p` and `q` form an alias relationship.

Further, in the function `example2`, pointer variables `z` and `w` are passed as arguments (Line 17). According to our assumptions (see **Assumptions** in Section 2), `z` and `w` can be declared aliases. However, in Lines 21 and 22, `z` is assigned to the memory location referenced by `c`, and `w` is assigned to the memory location referenced by `d`. Since `c` and `d` point to distinct memory locations (as shown in Line 20), `z` and `w` do not form an alias relationship.

The output of your may-alias analysis pass on the input example (Figure 2) should be as follows:

```
1  Function: example
2  p -> {q}
3  q -> {p}
4  Function: example2
5  z -> {}
6  w -> {}
7  c -> {}
8  d -> {}
9  Function: main
```

4 Hints

- You may need to study the cpp STL containers (std::set, std::map, std::vector).
- You may need to implement Kildall's algorithm to maintain the points-to information.
- You may need `isPointerTy()` API also to identify pointer variables.

5 Additional Details

The marks distribution for the constant propagation assignment is as follows:
1. Correct Output on Public Test Cases. (30 pts)
2. Correct Output on Private Test Cases. (70 pts)

Here are some DOs and DONTs for the assignment.
DOs

- Use git commit to upload the assignment and the private test cases.
- Run the script in `checker.sh` file and submit the assignment only after receiving an "Accept" output from the script. It checks the naming conventions of the files and folder structure but it cannot validate the correctness of your analysis results.
- Clone the assignment repository inside the llvm-project folder.
- Write your pass only in the appropriate section of `alias_lib.cpp` file.
- Your output text file should have the analysis results of all the functions mentioned in `assign.c` file. You must not generate multiple output files.
- Try to submit within the deadline. There is a late penalty (**20**%) for each day after the deadline. We may consider 1-2 hours beyond the deadline time, but please do not expect further consideration.

DONTs

- Do not submit assignment if the `checker.sh` gives "Rejected". Your assignment will not be evaluated if your directory structure or naming conventions do not match.

- Do not change the name of any files or folders.
- Do not edit any other things (e.g., name of the pass) in the `alias_lib.cpp` file.
- Do not use any GPT tool. There will be a heavy penalty for such behaviour.
- Do not try to copy from any online resource or another student's assignment. In case of plagiarism, both the students (sink and source) will be seen as **equally** guilty.