Assignment 3: May-Alias Analysis

Introduction to Program Analysis and Compiler Optimization

February 19, 2024

1 Alias Analysis

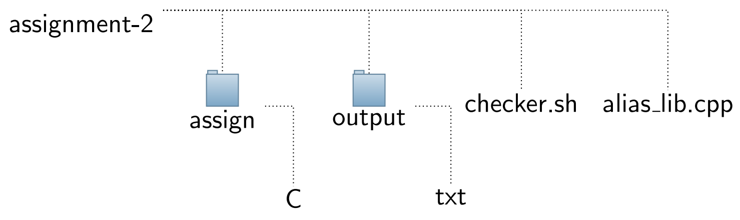
Alias Analysis (aka Pointer Analysis) is a class of techniques that attempt to determine whether two pointer variables can ever point to the same memory location. There are many different algorithms for alias analysis and many different ways of classifying them: flow-sensitive vs. flow-insensitive, context-sensitive 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 Example

Consider an example in Figure 1. Lines 9 and 10 contain pointer variables, `x` and `y` pointing to integer variables `a` and `b` respectively. At line 13, pointer `x` is pointing variable `a`. However, at line 14 pointer `y` is pointing to `a`, whereas earlier it was pointing to variable `b`. Hence, at the join point (line 15) we can conclude that `x` will point to variable `a` but the pointer variable `y` can point to either `a` or `b`, based on the branch at line 11. In this context, a **MayAlias** analysis can conclude that pointer variable `x` and pointer variable `y` may point to the same memory location.

3 Deliverables

The structure of the given template directory is mentioned above.



There are two folders, named `assign` and `output`. The former contains the CPP codes which you need to transform into LLVM IR. On the generated LLVM IR you will run your `mayAlias` analysis pass. The folder `output` will contain `text` files, with each `text` file containing the `mayAlias` relationship of each pointer variable pair.

See Figure 2 for a desired output of the given LLVM IR code in Figure 1. You need to store the alias relationship in a text file in the form of sets. While writing the sets you need to follow the below-mentioned rules: (1) Each pointer variable must have a set of pointer variables with which it may get aliased. (2) Each set should be printed with all the possible aliasing pointers inside two curly braces separated by commas. (3)

```c

int main()
{
    int a = 10;
    int b = 20;
    int c;
    scanf("%d", &c);

    int *x = &a;
    int *y = &b;
    if(c>0)
    {
        x = &a;
        y = &a;
    }
}
```

Figure 1: Example

```
    x -> {y}
    y -> {x}
```

Figure 2: Output to the given example

If the set is null, you must print the set with two curly braces (e.g., {}) (4) The printing of alias relationship must be done in the order of the declaration of corresponding pointer variables (e.g., in the output of given example code, the aliasing set of `y` should not occur before that of `x`).

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.
- 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 and folder structure. Note: `checker.sh` cannot validate the correctness of your assignment 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 same name as input IR files. For example, the output file corresponding to `file1.cpp` should be named as `file1.txt`.

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.