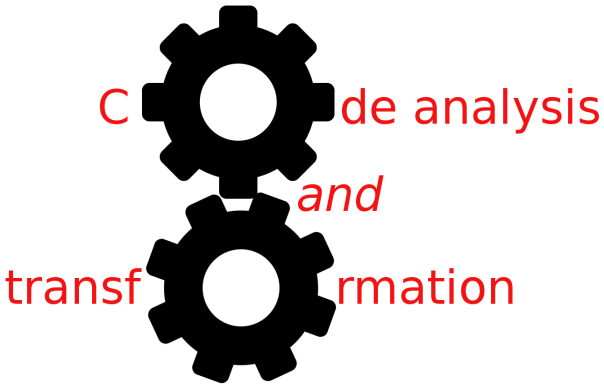

Homework H2

1 Description

Write an LLVM pass starting from the code you have developed for H1. The goal of this pass is to develop the first part of reaching definition data-flow analysis **for the CAT language** (not for generic C code). The definitions you need to analyze are those that define CAT variables only (see next section).

You need to define the GEN and KILL sets for every bitcode instruction of a program given as input. In more detail, you need to choose how to represent GEN and KILL sets (e.g., arrays, bitsets, graphs, trees, lists). Then you need to iterate over all instructions and compute GEN and KILL for each one. At the end of your pass, you need to have stored all GEN and KILL sets for all instructions in a data structure that you believe is going to be suitable for computing IN and OUT sets. Notice that having stored GEN and KILL sets is a requirement.

Before ending your pass, you need to print GEN and KILL sets for each instruction.

2 The CAT language

The CAT language is composed by operations performed by invoking the API defined in CAT.h, which is available in your tests. For example, `CAT_add` is the add operation of the CAT language.

Variables in this language are created by `CAT_new` and they are called "CAT variables". `CAT_new` returns a reference to the CAT variable just created; in other words, the return value isn't the CAT variable, but it points to the CAT variable.

The CAT function `CAT_get` takes a reference to a CAT variable as input and it returns its value stored in it.

The language used to invoke CAT functions is C.

2.1 Code example

The following program prints `5`.

```
#include <stdint.h>
#include <stdio.h>
#include <CAT.h>

int main (){
  CATData d1;
```

```
  CATData d2;
  CATData d3;

  d1  = CAT_new(2);
  d2  = CAT_new(3);
  d3  = CAT_new(0);

  CAT_add(d3, d1, d2);

  int64_t valueComputed = CAT_get(d3);
  printf("%ld", valueComputed);

  return 0;
}
```

2.2 Assumptions

For the H2 homework, you can take advantage of the following assumptions about the C code that invokes CAT functions.

1. A C variable used to store the return value of `CAT_new` (i.e., reference to a CAT variable) is defined statically not more than once in the C function it has been declared.
2. A C variable that includes a reference to a CAT variable cannot be copied to other C variables (no aliasing).
3. A C variable that includes a reference to a CAT variable cannot be copied into a data structure.
4. A C variable that includes a reference to a CAT variable does not escape the C function where it has been declared.

Run all tests Go to `H2/tests` and run

```
make
```

to test your work.

3 LLVM API and Friends

This section lists the set of LLVM APIs and headers I have used in my (multiple) H2 solutions that I did not use for the past assignments. You can choose whether or not using these APIs.

The next two lists are the union of the uses in all of my solutions.

- Method `getArgOperand` of the class `CallInst`
- Some methods of the classes `std::set`,`SmallBitVector`,`BitVector`,`SparseBitVector`

```
#include "llvm/IR/BasicBlock.h"
#include "llvm/Transforms/Utils/BasicBlockUtils.h"
#include "llvm/ADT/SmallBitVector.h"
#include "llvm/ADT/BitVector.h"
#include "llvm/ADT/SparseBitVector.h"
```

4 What to submit

Submit via Canvas the C++ file you've implemented (CatPass.cpp).

For your information: my solution for H2 added 123 lines of C++ code to H1 (computed by `sloccount`).

Good luck with your work!