Homework H2

1 Description

Write an LLVM pass starting from the template available in Canvas (Template.tar.bz2). 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 instruction of a program given as input. Specifically, 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. 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 defined by functions you can find in CAT.h of the CAT library. For example, `CAT_binary_add` is the add operation of the language.

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

The CAT function `CAT_get_signed_value` 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_create_signed_value(2);
  d2  = CAT_create_signed_value(3);
  d3  = CAT_create_signed_value(0);

  CAT_binary_add(d3, d1, d2);

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

  return 0;
}
```

To compile the above code, run

```
clang `pkg-config --cflags libCAT` `pkg-config --libs libCAT` program.c
```

where `program.c` is the file where you have stored the above code.

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 variable used to store the return value of `CAT_create_signed_value` (i.e., reference to a CAT variable) is defined statically not more than once in the function it has been declared.
2. A variable that includes a reference to a CAT variable does not get copied to other variables.
3. A variable that includes a reference to a CAT variable does not get copied in any data structure.
4. A variable that includes a reference to a CAT variable does not escape the function where it has been declared.

2.3 Installing the CAT library

The file `libCAT/README` includes instructions about how to install the library `libCAT`.

After installing the library, make sure that `pkg-config` is aware of this new installation. To test whether or not this is the case, run the following command:

```
pkg-config --cflags libCAT
```

The output of the above command will tell you whether `pkg-config` is aware of your new `libCAT` installation or not.

If `pkg-config` isn't aware of your `libCAT` , then you need to modify the `PKG_CONFIG_PATH` environment variable as next described. Let `/home/me/myinst` be the directory where you have installed `libCAT`. Make sure that there is a subdirectory `lib` in `/home/me/myinst` . The file that `pkg-config` needs to have access to is stored in `/home/me/myinst/lib/pkgconfig`. So, we need to modify `PKG_CONFIG_PATH` as following:

```
export PKG_CONFIG_PATH=/home/me/myinst/lib/pkgconfig:$PKG_CONFIG_PATH
```

Now `pkg_config` is aware of your `libCAT` installation.

3 Example

Consider the following program:

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

void CAT_execution (int userInput){
  CATData d1,d2,d3;

  d1  = CAT_create_signed_value(5);
  printf("H1:   Value 1 = %ld\n", CAT_get_signed_value(d1));

  d2  = CAT_create_signed_value(8);
  if (userInput > 10){
    CAT_binary_add(d2, d2, d2);
  }
  printf("H1:   Value 2 = %ld\n", CAT_get_signed_value(d2));

  d3  = CAT_create_signed_value(0);
  CAT_binary_add(d3, d1, d2);
  printf("H1:   Result = %ld\n", CAT_get_signed_value(d3));

  return ;
}

int main (int argc, char *argv[]){
  CAT_execution(argc);

  return 0;
}
```

your pass must generate the same output stored in `tests/test0/output/oracle_output`):

4 Testing your work

To test your work, go to a test you have available:

```
cd H2/tests/test0
```

set the path of your pass in `LLVMPASSPATH` of Makefile.

Now, invoke your pass

```
make
```

Check the output generated by your pass against the oracle output:

```
make check
```

If you've passed the test, you'll see the following output:

```
Test passed!
```

Otherwise, you'll see the following:

```
Test failed
Output differences can be found in "./diff_output"
```

and you can look at `diff_output` to find out the differences.

Good luck with your work!

5 What to submit

Submit via Canvas

- The C++ file you've implemented (CatPass.cpp)
- A PDP document where you describe in less than 500 words why you've chosen the data structures you used in CatPass.cpp to store GEN and KILL sets

For your information: my solution includes 190 lines of C++ code computed by `sloccount`.

6 Homework due

10/21