C de analysis
and
transf rmation

Homework H4

1 Description

Write an LLVM pass starting from the code you have developed for H3.

The goal of this new pass is to implement the constant propagation code transformation by using the IN and OUT sets you have computed in H3 with your reaching definition data-flow analysis. As it was the case for H3, the only variables you need to consider are the CAT variables.

1.1 Assumptions

For the H4 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 API

This section describes the set of LLVM APIs I have used in my H4 solution that I did not used in prior assignments. You can choose whether or not using these APIs.

These APIs are the following:

- Checking whether or not an instance of `Value` is an integer constant:

  ```
  isa<ConstantInt>(v)
  ```

  where `v` is an instance of `Value`.

- To fetch the actual constant value from an instance of `Value`:

  ```
  int64_t c = v->getSExtValue();
  ```

  where `v` is an instance of `Value`.

- To substitute all uses of a variable defined by an instruction with a constant:

  ```
  ReplaceInstWithValue(bb->getInstList(), ii, constValue)
  ```

  where `bb` is an instance of `BasicBlock`, `ii` is an instance of `BasicBlock::iterator` , and `constValue` is an instance of `Value`.

- To create an instance of `BasicBlock::iterator`:

  ```
  BasicBlock::iterator ii(i);
  ```

  where `i` is an instance of `Instruction`.

3 Testing your work

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

```
cd H3/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, the output message tells you what went wrong. Output differences are stored in the `diff` directory .
Good luck with your work!

4 What to submit

Submit via Canvas

- The C++ file you've implemented (CatPass.cpp)

For your information: my solution for H4 added 91 lines of C++ code to H3 (computed by `sloccount`).

5 Homework due

11/5 at 2am :)