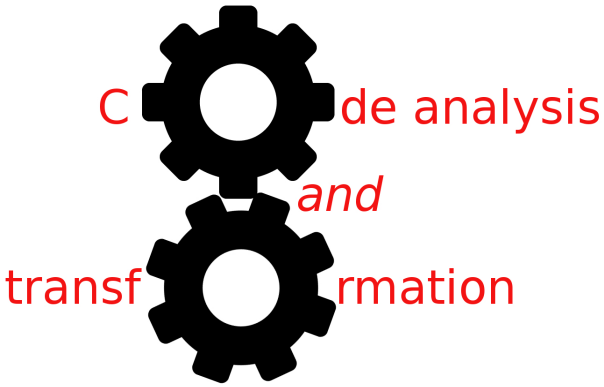

Homework H5

1 Description

Write an LLVM pass starting from the code you have developed for H4.

For H5 you need to extend your work (H4) to remove all assumptions you relied on to develop your H4 code. Finally, you cannot rely on data dependence analysis for H5.

2 Impact of Previous Assumptions

This section provides information about the impact of the assumptions you use to rely on.

2.1 No escaping variable assumption

Let's start removing the following assumption from the ones you relied on for H4:

- A C variable that includes a reference to a CAT variable does not escape the C function where it has been declared.

The above assumption allowed you to assume that a CAT variable was defined in the same function that it was used. Now it is not the case anymore; now you can have CAT variables as parameters of a function.

For example, now your pass has to handle the following C function:

```
void a_generic_C_function (CATData d1){
  int64_t v = CAT_get_signed_value(d1);
  printf("%ld ", v);
  return ;
}
```

Because your pass analyzes one function at a time (i.e., intra-procedural), you must assume that you know nothing about values stored in function parameters.

2.2 Unique definition assumption

Let's now remove the following assumption (in addition to the previous one already removed):

- A C 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 C function it has been declared.

Now your pass has to analyze and transform the following C function:

```
void a_generic_C_function (CATData d1){
  int64_t v = CAT_get_signed_value(d1);
  if (v > 10){
    d1 = CAT_create_signed_value(50);
  } else {
    d1 = CAT_create_signed_value(20);
  }

  int64_t v2 = CAT_get_signed_value(d1);
  printf("%ld ", v2);

  return ;
}
```

2.3 No alias assumption

Let's now remove the following assumption (in addition to the previous one already removed):

- A C variable that includes a reference to a CAT variable cannot be copied to other C variables (no aliasing).

Now your pass has to be able to analyze and transform the following C function:

```
void a_generic_C_function (CATData d_par){
  CATData d1;
  d1 = d_par;

  int64_t v2 = CAT_get_signed_value(d1);
  printf("%ld ", v2);

  return ;
}
```

2.4 Copied to/from memory

Let's now remove the following assumption (in addition to the previous ones already removed):

- A C variable that includes a reference to a CAT variable cannot be copied into a data structure.

Now your pass has to be able to analyze and transform the following C function:

```
void a_generic_C_function (void){
  CATData *pointer;
  (*pointer) = CAT_create_signed_value(50);

  int64_t v2 = CAT_get_signed_value(*pointer);
  printf("%ld ", v2);

  return ;
}
```

3 Assumption you can rely on

Correctness comes first (before the ability to improve code). Therefore, since you cannot rely on a data dependence analysis, for H5 you have to implement a correct and conservative solution: if any CAT variable get stored in memory, do not consider them as candidates for your constant propagation pass.

You can assume that a CAT variable escapes if the reference of that CAT variable get stored in memory.

Finally, the above C code are just examples of functions your pass has to be able to handle for this homework.

4 LLVM API and Friends

This section describes the set of LLVM APIs I have used in my H5 solution that I did not used in prior assignments. You can choose whether or not using these APIs.

- To create the 32-bits integer constant `0` :

  ```
  Constant *zeroConst = ConstantInt::get(IntegerType::get(m->getContext(), 32), 0, true);
  ```

  where `m` is an instance of `Module`.

- To create a new instruction "add" and insert it just before another instruction called `inst`:

  ```
  Instruction *newInst = BinaryOperator::Create(Instruction::Add, zeroConst, zeroConst, "", inst)
  ```

  where `inst` is an instance of `Instruction`, and `zeroConst` is an instance of `Constant`.

- To check if an instance of `Value` is an argument of a function:

  ```
  isa<Argument>(v)
  ```

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

- To delete an instruction from the function it belongs to:

  ```
  i->eraseFromParent()
  ```

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

- To check if an instance of the class `Value` is an instance of the class `PHINode`:

  ```
  isa<PHINode>(v)
  ```

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

- To fetch the variable stored in memory by a store instruction:

  ```
  Value *valueStored = storeInst->getValueOperand()
  ```

  where `storeInst` is an instance of `StoreInst`.

4.1 Testing your work

Copy `H5.tar.bz2` to your home at `hanlon.wot.eecs.northwestern.edu`. Login to `hanlon.wot.eecs.northwestern.edu` and go to `H5/tests` and run

```
make
```

to test your work.

5 What to submit

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

For your information: my solution for H5 added 62 lines of C++ code to H4 (computed by `sloccount`).

6 Homework due

11/2 at noon

Good luck with your work!