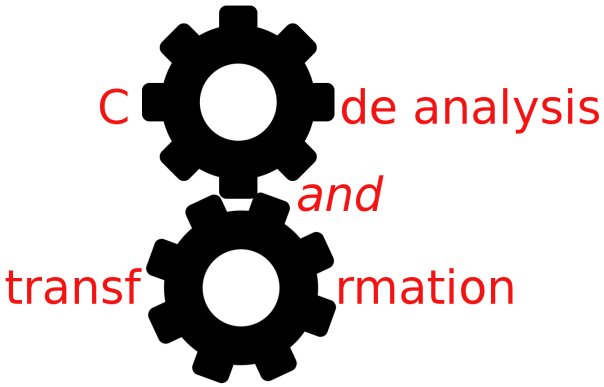

Homework H5

1 Description

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

For this homework you need to extend your H4 to remove all assumptions you have relied on in your previous assignments.

Finally, you cannot rely on alias 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 used in a C function was also defined in it. In other words, CAT variables couldn't be passed as arguments of a C function as well as they couldn't be returned.

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(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 both stored in function parameters and returned by other functions.

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_new` (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(d1);
  if (v > 10){
    d1 = CAT_new(50);
  } else {
    d1 = CAT_new(20);
  }

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

  return ;
}
```

2.3 No alias assumption

Let's remove also the following assumption:

- 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(d1);
  printf("%ld ", v2);

  return ;
}
```

2.4 Copied to/from memory

Let's remove the final assumption:

- 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:

```
typedef struct {
  CATData v;
} my_struct_t ;

void a_generic_C_function (my_struct_t *s){
  auto pointer = malloc(sizeof(CATData *));
```

```
  (*pointer) = CAT_new(50);
  s->v = *pointer;
  auto v2 = CAT_get(*pointer);
  printf("%ld ", v2);
  auto v3 = CAT_get(s->v);
  printf("%ld ", v3);

  free(pointer);

  return ;
}
```

3 Assumption you can rely on

Correctness comes first (before the ability to improve code). Therefore, since you cannot rely on an alias analysis, for H5 you have to implement a correct and conservative solution. A possible conservative solution is the following one: do not consider the CAT variables that are either stored in memory or passed to other functions.

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

Run all tests Go to `H5/tests` and run

```
make
```

to test your work.

4 LLVM API and Friends

This section lists the set of LLVM APIs and headers I have used in my H5 solutions that I did not use for the past 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`.

5 What to submit

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

For your information: my solution for H5 added 292 lines of C++ code to H4 (computed by `sloccount`).

Good luck with your work!