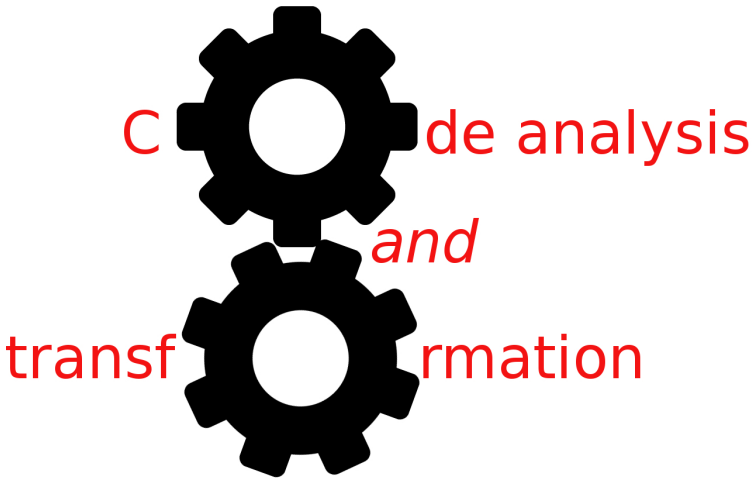

Homework H3

Contents

1 Description

Write an LLVM pass starting from the code you have developed for your previous assignment.

For this homework you need to extend your previous assignment to remove some assumptions (explained next) you have relied on in your previous assignments.

2 Homework

2.1 Assumptions

You can rely only on the following assumption:

1. A reference to a CAT variable in C cannot be stored into memory. CAT variable references can only exists within C variables.
2. A function cannot return a CAT reference.
3. As for your previous homework, your pass will be invoked until a fixed point is reached.

2.2 Constraints

Next are the constraints for your solution:

- H3 has to be intra-procedural as all prior assignments.
- You can only implement constant propagation, constant folding, and algebraic simplification. No other transformations are allowed (e.g., no deadcode elimination).
- You cannot delete calls to `CAT_new`.
- Operations of CAT variables (e.g., add, set) have to be either performed at compile time by your pass or they have to be performed via the CAT APIs.
- Your solution must complete each test in less than 10 minutes.
- Your pass cannot save a state as in your previous assignment.

No other constraints exist. In other words, no constraints are inherited from prior assignments.

2.3 Testing your work

Correctness comes first (before the ability to improve code). Therefore, you have to implement a correct and conservative solution. Finally, the above C functions are just examples of code your pass has to be able to handle for this homework.

The same conditions that you had to pass the tests for your previous homework assignment applies for this one. Also, you have the same options to run tests as you had for your previous assignment.

2.4 LLVM API and Friends

This section lists the set of LLVM APIs and headers I have used in my (multiple) solutions (this is the union of all APIs across solutions) such that

1. I did not use for the past assignments and
2. I did not list them yet in slides and
3. I did not use them in the LLVM examples I shared via Canvas in the directory `Code`.

You can choose whether or not using these APIs.

These APIs are the following:

- 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`.

2.5 What to submit

Submit via Canvas the file `sources.tar.bz2` generated by `collect_src.sh` script of the middle-end git repository you cloned.

3 Impact of the Lost Assumptions

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

3.1 No escaping variable assumption

This homework assignment cannot rely on the following assumption you had in your previous assignment:

- A C variable that includes a reference to a CAT variable cannot be given as argument to a call to a function.

In other words, the above assumption allowed you to assume that a CAT variable used in a C function was also defined in it. 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 stored in function parameters. Hence, your solution cannot propagate any constant in the example above.

Another example is the following:

```
void a_generic_C_function (void){
  CATData d = CAT_new(42);
  another_function(d);
  printf("%ld ", CAT_get(d));
  return ;
}
```

In the example above your solution cannot propagate any constant. This is because the CAT variable escapes to `another_function` and you must conservatively assume such function modifies the CAT variable given as input.

3.2 Unique definition assumption

This homework assignment cannot rely on the following assumption (in addition to the previous one already removed) that you had in your previous assignment:

- 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 where it has been allocated.

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 ;
}
```

For the example above, your solution cannot propagate any constant.

```
void a_generic_C_function (CATData d1){
  int64_t v = CAT_get(d1);
  if (v > 10){
    d1 = CAT_new(42);
  } else {
    d1 = CAT_new(42);
  }

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

  return ;
}
```

For this example above, your solution must propagate the constant `42`.

3.3 No alias assumption

This homework assignment cannot rely on the following assumption (in addition to the previous one already removed) that you had in your previous assignment:

- 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 (int p){
  CATData d;
  CATData d1 = CAT_new(5);
  CATData d2 = CAT_new(5);

  if (p > 5){
    d = d1;
  } else
    d = d2;
  }

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

  return ;
}
```

For the example above, your solution must be able to propagate the constant `5`.

Good luck with your work!