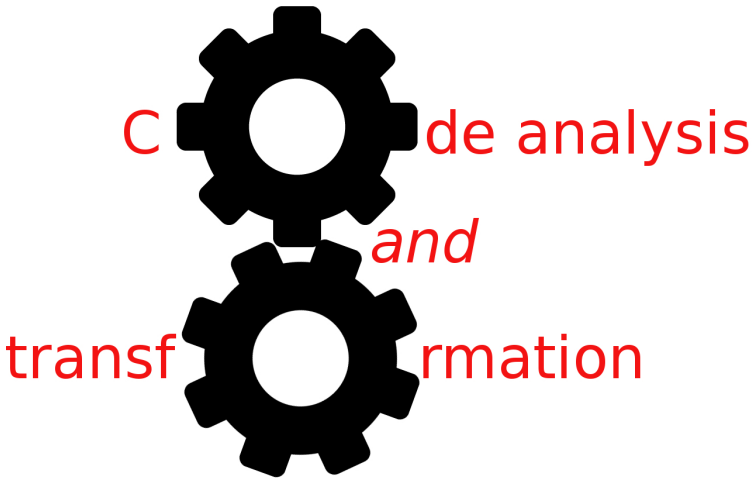

Homework H2

Contents

1 Description

Write an LLVM pass starting from your H1 code.

The goal of this new pass is to implement constant propagation, constant folding, and algebraic simplification. As it was the case for H1, the only variables you need to consider are the CAT variables.

2 Homework

2.1 Constraints

Next are the constraints for your solution:

- H2 has to be intra-procedural as all prior assignments.
- 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. For example, you cannot create a file where you store what you have performed during the previous invocations. Another example of solution you cannot implement is saving information in metadata attached to the IR.

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

2.2 Assumptions

You can make the same code assumptions that you had for the H2 homework with the only exception of how many times your pass will run. From now on, your pass **will be invoked until a fixed point is reached**. In more detail, the IR file generated by your pass (i.e., program_optimized.bc) is checked against the current input one. If they differ, then your pass will be invoked again to further modify the IR previously generated. This will continue for as long as your pass modifies the IR code given as input.

2.3 Testing your work

`H2.tar.bz2` includes some examples of C programs with multiple functions. The tests are included in `H2/tests` and your pass needs to pass them all.

2.3.1 To pass the homework

To pass this homework (and subsequent ones), you need to:

- Pass each test individually. To pass a test, your work needs to decrease the computation cost as specified by the oracle output (under the output folder of a given test). The computation cost paid for each invocation to a CAT API is described in `H0.pdf` inside `H0.tar.bz2`.
- Reduce the cumulative CAT API cost among all tests. The file `tests/optimization.txt` reports the total cost accumulated between all tests. This is the maximum value your pass can obtain to pass the related homework assignment.
- Your pass includes only correct C++ code (bug free)
- The algorithm you have designed and implemented is correct (this check cannot be automated)

2.3.2 To run all tests

You have two ways of checking your work against the tests included.

- **Parallel:** to run all tests in parallel and wait for all of them, go to `H2/tests` and run `make`.
- **Sequential early stopping:** to run one test at a time and stop as soon as one test fails, go to `H2/tests` and run `make sequential`.

2.3.3 Memory Corruption

Your pass needs to include correct C++ code. The most common bugs when writing C++ code are memory corruptions. To help you identify memory corruptions in your code, the directory `tests` includes scripts to run the tool `valgrind` on your pass. If you do not know `valgrind`, then it is now the time to learn it. It is a well established tool, so please google it.

To run your pass using `valgrind` to check for memory corruptions in your code, follow the next steps:

- `cd tests`
- `make clean`
- `make USE_VALGRIND=1`

If your pass shows a memory corruption while analyzing a test, then that test will fail independently on the correctness of the generated IR file.

2.4 LLVM API and Friends

This section lists the set of LLVM APIs and headers I have used in my (multiple) H2 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:

- 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`.

- To get the number of pairs in a phi node:

```
uint64_t numberOfPairs = phiNode->getNumIncomingValues();
```

where `phiNode` is an instance of `PHINode`.

- To get the value of the i-th pair (starting form `0`) of a phi node:

```
Value *v = phiNode->getIncomingValue(i);
```

where `phiNode` is an instance of `PHINode` and `i` is an integer value.

I've also used the following new header:

```
#include "llvm/IR/Constants.h"
```

2.5 What to submit

Submit via Canvas the file `sources.tar.bz2` generated by `collect_src.sh` script of the middleend git repo you cloned.

Good luck with your work!