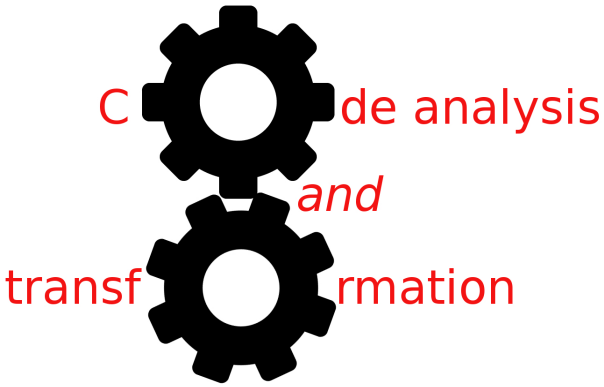

Homework H6

1 Description

Write an LLVM pass starting from the code you have developed for H5.

The goal of this assignment is to reduce the conservativeness of H5 related to variables that escape to memory.

In more detail, instead of blocking the constant propagation for such variables as you did in H5, you now need to rely on the dependencies identified by LLVM to properly handle them both in the reaching definition analysis and in your constant propagation algorithm.

2 LLVM API and Friends

This section describes the set of LLVM APIs I have used in my H6 solution that I did not used in prior assignments. You can choose whether or not using these APIs.

- To get the output of the dependence analysis :

  ```
  DependenceAnalysis &deps = getAnalysis<DependenceAnalysis>();
  ```

  you also need to include the appropriate header:

  ```
  #include "llvm/Analysis/DependenceAnalysis.h"
  ```

- To declare your pass now depends on dependence analysis:

  ```
  AU.addRequiredTransitive<DependenceAnalysis>();
  ```

- To check if there is a direct dependence from an instruction `i1` to another instruction `i2`:

  ```
  deps.depends(i1, i2, false)
  ```

  where `i1,i2` are instances of `Instruction`.

2.1 Testing your work

`H6.tar.bz2` includes examples of C programs with escaping variables. The tests included in the package do not check whether or not your H6 compiler generates better code than H5. These tests check only whether or not the generated binary preserves the same program output of the original C program.

This is because there are many possible solutions for your H6 compiler that can lead to many different optimized bitcode. I will check your H6 improvements manually. Comments are welcome.

3 What to submit

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

For your information: my solution for H6 added 56 lines of C++ code to H5 (computed by `sloccount`).

4 Homework due

11/9 at noon