C de analysis
and
transf rmation

Homework H1

1 Description

First we describe the target programs of your assignment, and then we describe the assignment.

1.1 The CAT API

From now on your homework will need to analyze (and transform in later assignments) C programs that invoke a known set of functions called CAT API. Next are the C functions that your homework needs to consider: `CAT_add`, `CAT_sub`, `CAT_new`, `CAT_get`, `CAT_set`.

Your work will have to understand how these functions are invoked by the C program given as input. The semantics of these functions are the following:

1. `CATData CAT_new (int64_t value)`: Create a new object called "CAT variable". This object includes an integer 64 bit value, which is initialized by this function before returning the object. The initial value is set to `value`. The CAT variable is stored in the heap.

2. `void CAT_add (CATData result, const CATData v1, const CATData v2)`: It adds the values included inside `v1` and `v2` and it stores the new resulting value inside the CAT variable `result`.

3. `void CAT_sub (CATData result, const CATData v1, const CATData v2)`: It subtracts the value included inside `v2` from the value included within `v1` (e.g., `v1 - v2`) and it stores the new resulting value inside the CAT variable `result`.

4. `const int64_t CAT_get (const CATData v)`: It returns the value included within the CAT variable `v`.

5. `void CAT_set (CATData v, int64_t value)`: It stores `value` inside the CAT variable `v`.

1.2 Homework

Write an LLVM pass to print statistics about invocations of public functions included in the CAT API that are **reachable** from the entry point of a program's function.

Specifically, for each program function, you must print

- the name of a CAT function that is invoked in the current bitcode function `f` such that this invocation is reachable from the entry point of `f`, and

- the number of instructions of `f` that invoke it

The order of CAT functions to print is

1. `CAT_add`
2. `CAT_sub`
3. `CAT_new`
4. `CAT_get`
5. `CAT_set`

Finally, CAT functions that are not invoked by a bitcode function (or not reachable) are not printed.

2 CAT sources

You can find the CAT API in CAT.h available in the distributed tests.

3 Examples of Output of Your Work

Next we show a few examples of outputs that your pass will have to produce.
`H1.tar.bz2` includes a few programs you can use to test your work.

3.1 Example 0

Consider the following program:

```
#include <CAT.h>

int CAT_execution (void){
  CATData d1;
  CATData d2;
  CATData d3;

  d1  = CAT_new(5);
  d2  = CAT_new(8);
  d3  = CAT_new(0);

  CAT_add(d3, d1, d2);

  return CAT_get(d3);
}

int main (int argc, char *argv[]){
  return CAT_execution();
}
```

your pass must generate the following output (stored in `compiler_output`):

```
H1: "CAT_execution": CAT_add: 1
H1: "CAT_execution": CAT_new: 3
H1: "CAT_execution": CAT_get: 1
```

3.2 Example 1

Consider the following program:

```
#include <CAT.h>

int CAT_execution (void){
  CATData d1;
  CATData d2;
  CATData d3;

  d1  = CAT_new(5);
  d2  = CAT_new(8);
  d3  = CAT_new(0);
  return CAT_get(d3);

  CAT_add(d3, d1, d2);
  return 0;
}

int main (int argc, char *argv[]){
  return CAT_execution();
}
```

your pass must generate the following output (stored in `compiler_output`):

```
H1: "CAT_execution": CAT_new: 3
H1: "CAT_execution": CAT_get: 1
```

Run all tests Go to `H1/tests` and run `make` to test your work.
The following output means you passed all tests:

```
./misc/run_tests.sh
SUMMARY: 6 tests passed out of 6
```

If you didn't pass a test, then the output will include all tests that have failed.

4 LLVM API and Friends

This section lists the set of LLVM APIs and headers I have used in my (multiple) H1 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

You can choose whether or not using these APIs.

- Method `getFunction` of the class `Module`
- `isa<LLVM CLASS>(LLVM OBJECT)`. For example, `isa<CallInst>(i)` where `i` is an instance of the class `Instruction`
- `cast<LLVM CLASS>(LLVM OBJECT)`. For example, `CallInst *callInst = cast<CallInst>(i)` where `i` is an instance of the class `Instruction`

- `getCalledFunction` of the class `CallInst`
- Method `write_escaped` of the class `raw_ostream`

Next are some headers that you did not see in slides yet, and you might find them useful:

```
#include "llvm/Pass.h"
#include "llvm/IR/Module.h"
#include "llvm/IR/Function.h"
#include "llvm/IR/Instructions.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/Transforms/IPO/PassManagerBuilder.h"
```

5 What to submit

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

For your information: my solution for H1 added 63 lines of C++ code to H0 (computed by `sloccount`).

Good luck with your work!