Compilers for Modern Architectures Project Report

Author: Papadopoulos Charalampos
Topic: Reaching definitions and constant propagation at LLVM level.
Date: 5/1/2015

At this project, I created an LLVM pass that performs constant propagation analysis for a llvm bitcode file. The pass works at function level and prints after each instruction a list of all the variables that are constant until that point.
Since LLVM works at static single assignment form (often abbreviated as SSA form or simply SSA), building that pass was simplified. SSA is a property of an intermediate representation (IR), which requires that each variable is assigned exactly once, and every variable is defined before it is used. There were no kill sets for our constant propagation, since once a constant was assigned to a variable, that variable could be considered as constant till the end of the program.

Instructions: In order to run my demo, after downloading the tar file you need to type at the command prompt :

./sript

The script does the following:

```
rm pass.so
./parser code.c 2>&1 | opt -dot-cfg | llvm-dis> test.ll
./parser code.c 2>&1 | opt -mem2reg | llvm-dis > test2.ll
g++ -fPIC -shared pass.cc -o pass.so -std=c++11 `llvm-config --cppflags`
cat test.ll | opt -load ./pass.so -print
```

The parser is the one that we built at previous labs. The parser produces the llvm bitcode file .ll . The llvm bitcode file that is used by the pass is the test.ll file. The opt -dot-cfg is useless, it was just used because without it, the output was not redirected to test.ll.

The test2.ll file was built to demonstrate the problems that I faced when I used the mem2reg optimization. That optimization promotes memory references to be register references. Most important, the mem2reg optimization implements the SSA functionality. However, the –mem2reg optimization eliminates several instructions, as someone can easily inspect. That way the functionality of my pass cannot be shown.

For the above reasons,I used test.ll instead of test2.ll . The test.ll is not characterized by SSA properties, but the code.c was chosen in a way that the test.ll looks like SSA code.

Another problem that I faced, was that I could not find the correct way to print the operand of a function, when the operand was a constant. For that reason, I could not assign the value of its constant to its name. This was a big issue and did not allow me to create a more complex pass.

APPENDIX - Example

code.c

```
void f()
{
  int x;
  int y;
  int mul;
  int div;
  int add;
  int sub;
  x=1;
  mul=x*y;
  div=9/x;
  add=x+y;
  sub=3-y;
}
```

test.ll

```
; ModuleID = '<stdin>'

define void @f() {
L0:
  %t0 = alloca i32
  %t1 = alloca i32
  %t2 = alloca i32
  %t3 = alloca i32
  %t4 = alloca i32
  %t5 = alloca i32
  store i32 1, i32* %t0
  %t6 = load i32* %t0
  %t7 = load i32* %t1
  %t8 = mul i32 %t6, %t7
  store i32 %t8, i32* %t2
  %t9 = load i32* %t0
  %t10 = sdiv i32 9, %t9
  store i32 %t10, i32* %t3
  %t11 = load i32* %t0
  %t12 = load i32* %t1
  %t13 = add i32 %t11, %t12
  store i32 %t13, i32* %t4
  %t14 = load i32* %t1
```

```
  %t15 = sub i32 3, %t14
  store i32 %t15, i32* %t5
  ret void
}
```

Terminal output after typing ./script

```
neu@ubuntu:~/Desktop/compilers_deliverables$ ./script

Pass on function f
----------------------------------------
 Instruction 1 : alloca
 Constants:  { }
----------------------------------------
 Instruction 2 : alloca
 Constants:  { }
----------------------------------------
 Instruction 3 : alloca
 Constants:  { }
----------------------------------------
 Instruction 4 : alloca
 Constants:  { }
----------------------------------------
 Instruction 5 : alloca
 Constants:  { }
----------------------------------------
 Instruction 6 : alloca
 Constants:  { }
----------------------------------------
 Instruction 7 : store
 Adding constant variable t0
 Constants:  { t0 }
----------------------------------------
 Instruction 8 : load
  Adding constant variable t6
 Constants:  { t0 t6 }
----------------------------------------
 Instruction 9 : load
 Constants:  { t0 t6 }
----------------------------------------
 Instruction 10 : mul
 Constants:  { t0 t6 }
```

```
----------------------------------------
 Instruction 11 : store
 Constants:  { t0 t6 }
----------------------------------------
 Instruction 12 : load
  Adding constant variable t9
 Constants:  { t0 t6 t9 }
----------------------------------------
 Instruction 13 : sdiv
  Adding constant variable t10
 Constants:  { t0 t10 t6 t9 }
----------------------------------------
 Instruction 14 : store
 Adding constant variable t3
 Constants:  { t0 t10 t3 t6 t9 }
----------------------------------------
 Instruction 15 : load
  Adding constant variable t11
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 16 : load
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 17 : add
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 18 : store
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 19 : load
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 20 : sub
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 21 : store
 Constants:  { t0 t10 t11 t3 t6 t9 }
----------------------------------------
 Instruction 22 : ret
 Constants:  { t0 t10 t11 t3 t6 t9 }
```