DF00100 Compiler Labs: Part 1 - LLVM IR

Erik Hansson

1 LLVM

LLVM (`www.llvm.org`) stands for low level virtual machine and is an unlimited register machine. Register values can only be set once, and has a type. LLVM has three equivalent forms of intermediate representations (IR), assembly form, in memory representation and bit code.

This part of the labs focus on how to work with the IR of LLVM.

1.1 Installation

We prefer to install from source code. We will also use the Clang front-end. The following steps are done on 32 bit Ubuntu 9.10.

- Download LLVM 2.6 source
- Unpack it (into `LLVM_DIR`)
- Download Clang source
- Unpack Clang source into `LLVM_DIR/tools/clang/`
- Run `./configure` and `make -j 4`
- Add `LLVM_DIR/Release/bin/` to your `PATH`

For more information see LLVM documentation (`http://llvm.org/docs/`). Of course you can also use the latest svn release of LLVM, if you do so please write in you report which revision you used.

1.2 First step

```
#include <stdio.h>
int main(  )
{
  printf("Hello LLVM + CLANG! \n");
}
```

Compile it using Clang (`clang -o test1 test1.c`). This will produce native code, then run it (`./test1`). To generate LLVM IR, in assembly form, run `clang test1.c -S -emit-llvm` which will produce `test1.s`.

Now run the same command but with optimization flag (`-o3`) and compare the output. Compile other input programs, with and with out optimization, for example

```
int main()
{
  int x =0;
  return x++;
}
```

and compare the IR code.

1.3 A simple pass

Write a simple pass that calculates how many calls there are to `printf()` function. Output can for example be something like:

```
main:
  printf(): 12 calls
foo:
  printf(): 23 calls
```

See, `http://llvm.org/docs/WritingAnLLVMPass.html`, for a good introduction about how to get started writing passes. To convert `file.s` to bit code format, use `llvm-as file.s` which produces `file.s.bc`.

What will your pass report if `printf()` is called inside a for loop?

1.4 Exercise 1

Rewrite your `printf` calculation pass so it handles loops. At least if the number of times the loop body will execute can be determined statically, and the loops are perfectly nested. Example:

```
for(i=2; i<4; i++)
        {
          printf("Hello␣LLVM␣+␣CLANG!␣\n");
        }
```

Feel free to run/use already existing passes that may be useful, for example:

- mem2reg
- indvars
- loop-rotate

1.5 Exercise 2a

Write a pass (or a set of passes) that recognizes the vector init (initialization of a vector with a constant), example:

```
for(i=0; i<5; i++)
    v[i]=42;
```

The pass should report: Matched computation, operand, size etc. Alternatively replace matched loops by an equivalent function call.

1.6 Exercise 2b - optional (gives bonus in the exam)

Write a pass that recognizes dot product calculation,

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=0}^{n} a_i b_i$$

where **a** and **b** are vectors with elements a_0, ..., a_n respectively b_0, ..., b_n. The corresponding C code could look like:

```
double DotProduct(double *x, double *y, int size)
{
 int i;
 double result=0;

 for(i=0; i< size; i++)
   result+= x[i]*y[i];

 return result;
}
```

Also consider implementations with double for-loops:

```
for(i=.., .., ..)
  for(j=.., .., ..)
    a[i]+=b[i][j]*c[j]
```

Example of things that should *not* be matched:

```
s=s+a[i]*b[j]
```

and

```
s=t+a[i]*b[i]
```

The pass should report: Matched computation, operand, size etc. Alternatively replace matched loops by an equivalent function call.

1.7 What should be in your report

- Strategy for solving the problems.
- Results of your tests, with comments.
- Well commented source code for your LLVM passes.
- Test programs, in C and LLVM assembly. Do not forget to write which built in passes you used (mem2reg etc.) and if you used any optimizations. Their invocation order is interesting as well.

Also write which version of LLVM you used and which platform you used. If you downloaded it via svn please write the revision number you used. It would also be nice if you had a (small) discussion part where you can take up problems you had.