Introduction to Language Theory and Compilation: Exercises

Session 8: Code generation

Code generation

Two questions:

1. What code to generate? For which architecture?
    - Low Level Virtual Machine Intermediate Representation (LLVM IR) for an abstract machine because it is intermediate code (see `http://www.llvm.org`)
2. At what point should the code be generated? How can this be specified formally?
    - Attribute grammars

Abstract machine

In this course, we'll use LLVM IR as target language, which acts as a kind of assembly language for an abstract machine.

- LLVM IR is a intermediary language developed (originally) for a Virtual Machine (LLVM), the file extension is `*.ll`
- LLVM IR is can be optimized and compiled to a specific architecture by using LLVM tools.

For the complete documentation, go to
`http://llvm.org/docs/LangRef.html`

LLVM IR – generals

First of all, we present a simplified LLVM because we avoid objects and visibility.

We also avoid to call specific architecture commands like commands to access on registers.

Inline comments starts with a ';' until the end of line.

LLVM IR – identifier

Two types

Global identifiers (*functions*, *global variables*) begin with the **'@'** character

Local identifiers (*register names*, *types*) begin with the **'%'** character

LLVM IR – identifier

Three formats

Named values[a] '[%@][a-zA-Z$._][a-zA-Z$._0-9]*'

Unamed values '[%@]integer' looks like '%0'. It is temporaries values and they are numbered sequentially (using a per-function incrementing counter, starting with 0).

Constants classical form for numeric ('null' for pointers, 'true'/'false' for boolean)

[a] other characters can be surrounded with quotes and special characters may be escaped using '\xx' where 'xx' is the hexadecimal ASCII code.

LLVM IR – primitive types

i*N* is an integer defined on *N* bits (i.g. *i*1 for boolean, *i*32 for classic integer)

half 16-bit floating point value

float 32-bit floating point value

double 64-bit floating point value

void does not represent any value and has no size

label represents code labels

array [<# elements> x <elementtype>]

LLVM IR – function

Syntax

```
define <ResultType> @<FunctionName> ([argument list])
{
   entry:
      ...
}
```

Example: $sum(a, b)$

```
define i32 @add1(i32 %a, i32 %b)
{
   entry:
      %varTmp1 = add i32 %a, %b
      ret i32 %varTmp1
}
```

Simple usage

as an **Interpreter**

1. Produce the byte-code file

```
llvm-as code-source.ll -o=code-source.bc
```

2. Run the interpreter

```
lli code-source.bc
```

as an **Compiler**

1. Produce the byte-code file

```
llvm-as code-source.ll -o=code-source.bc
```

2. Run the compiler

```
llc code-source.bc -o=code-source.bin
```

3. Run your program

```
./code-source.bin
```

Exercise 1

Assuming that you have defined these functions:

```
define i32 @readInt()
define void @println(i32 %value)
```

Write a LLVM function that computes and outputs the value of:

$$(3 + x) * (9 - y)$$

where x is a value read on input and y is a global variable.

Operations

The list of all operations are available on `http://llvm.org/docs/LangRef.html`.

The most useful subset is explained in the remainder section of the statement sheet.

Exercise 2

Assuming that you have defined these functions:

```
define i32 @readInt()
define void @println(i32 %value)
```

Write a function that:

- Allocates memory for two variables we will call *a* and *b*
- Initializes *a* and *b* with values read on input
- Adds 5 to *a*
- Divides *b* by 2
- If $a > b$, output *a*, else output *b*

Exercise 3

Define this function

```
define i32 @readInt()
```

which reads an integer of the form `[0-9]+` in base 10 by using

```
; External declaration of the getchar function
declare i32 @getchar()
```

Remember that the character 0 is the ASCII code 48.

Exercise 4

Translate this C program in LLVM IR.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int getNumber(void){
   return rand() % 100;
}

int main(void){
   //initialization of randomizer
   srand(time(NULL));
   int guess = getNumber();
   int i;
   for(i=0;i<5;i++){
      int try;
      scanf("%d",&try);
      if(try > guess){//greater
         putchar(45);//-
         putchar(10);//\n
```

Exercise 4 (ctd.)

```
      }else if(try < guess){//lower
         putchar(43);//+
         putchar(10);//\n
      }else{//success
         putchar(79);//O
         putchar(75);//K
         putchar(10);//\n
         return 0;
      }
   }
   //failure
   putchar(75);//K
   putchar(79);//O
   putchar(10);//\n
   return 0;
}
```