Output C Code

Compiler Construction '16 Final Report

Cdric Viaccoz   Bastian Nanchen

EPFL

{cedric.viaccoz}@epfl.ch

1. Introduction

The first part of the lab was to implement a compiler for Toy Object-Oriented Language (Tool), which is a small object-oriented programming language.

This has led us to conceive a Lexer, a $LL(1)$ grammar, a Parser, a Type Checking system and to generate binary code for the Java Virtual Machine.

The aim of this project is to replace code generation for the Java Virtual Machine with a pretty printer, which outputs C code.

The difficulty is to write C code, which is an imperative programming language, from an object-oriented programming language.

2. Examples

First it is important to see the representation of an object in C.

```
class Rectangle {
    var width: Int;
    var height: Int;

    def area(): Int = {
        return width * height;
    }
}
```

This Tool code will be pretty-printed in C like this:

```
struct Rectangle{
    int width;
    int heigth;
    int (*area)(struct Rectangle*);
}

int Rectangle_area(struct Rectangle* this) {
    return this->width * this->height;
}
```

The class are represented in C by a structure. The variables of the class are members of the structure. They are declared in the same fashion.

Contrariwise the method of the class is declared outside of the structure. In C, a structure can't have a function declaration as a member. The function shall be declared outside. But the structure has a pointer to the function. The function is declared with the name of the class as prefix followed by an underscore. Thus the belonging to the "Rectangle" class is suggested.

Therefore this is how the C code mimics the behavior of an object-oriented class.

There exists other dilemma like inheritance, dynamic dispatch,...

3. Implementation

This is a very important section, you explain to us how you made it work.

3.1 Theoretical Background

If you are using theoretical concepts, explain them first in this subsection. Even if they come from the course (eg. lattices), try to explain the essential points *in your own words*. Cite any reference work you used like this [?]. This should convince us that you know the theory behind what you coded.

3.2 Implementation Details

Two files were created to implement this project.

The first one is "CDataType.scala". It contains a class that depicts the C's structure.

```
class StructDef(val name: String, val membersList: ListBuffer[St
```

A trait "StructMember" is extended by two class to implement the two sort of structure's members, which are a variable or a pointer to a function (as explained in

section 2. Examples). Without forgetting to define a class "FunctionPtr". It defines the fact that each function have as first parameter a generic pointer to the calling structure.

```scala
class StructVar(val name: String, val tpe: CType) extends StructMember {
  // some code
}

class StructFunctionPtr(val ptr: FunctionPtr, var mtDcl: MethodDecl) extends StructMember {
  // some code
}

class FunctionPtr(val name: String, val retType: CType, val args: List[CType]) {
  // some code
}
```

The second file is "COutputGeneration.scala". Its skeleton and behavior is based on the "CodeGeneration.scala" completed to output binary code for the Java Virtual Machine.

4. Possible Extensions

If you did not finish what you had planned, explain here what's missing.

In any case, describe how you could further extend your compiler in the direction you chose. This section should convince us that you understand the challenges of writing a good compiler for high-level programming languages.

References