Report for Compiler Course Project

Qinglin Li
Shanghai Jiaotong University
lostleaf@icloud.com

Abstract

This report describes the compiler course project. The design of Abstract syntax tree and immediate representative along with some optimization are included in this report.

Categories and Subject Descriptors D.3.4 [*Programming Languages*]: compiler

Keywords Compiler, Abstract Syntax Tree, Immediate Representative, Code Optimization, Register Allocation, Constant Folding

1. Introduction

This project aims at building a compiler for a subset of C language. It removed float numbers, some confusing grammars and most library functions in C language. And, of course, the compiler translate C code to MIPS code with ANTLR4 parser generate tool.

2. Abstract Syntax Tree

The Abstract Syntax Tree(AST for short) is generated while parsing and the whole process is contained in *C.g4* file under *parser* directory.
The AST is similar with Parsing Tree, but removes useless information. Every node in Parsing Tree is corresponded to a node in AST.
And since addition, multiplication, and other binary operator expressions are similar, a generic type is used here. And it brings much benefit when generate Immediate Representative.
The inheritance of AST is shown below:

- Node
  - Program
  - Declaration
  - . . .
  - Stmt(Correspond to Statement)
    - CompStmt(Correspond to Compound-Statement)
    - . . .
  - Expression
    - AssExpr(Correspond to Assignment-Statement)
    - BinExpr< ExprType >(generic type)
      - AddExpr:BinExpr< MulExpr >
      - MulExpr:BinExpr< CastExpr >
      - . . .

3. Semantic Checking

The semantic checking producure is called after AST generating. Semantic checking mainly check the following items:

1. Type
2. Left value
3. Declaration and use before declaraed
4. Other items including breaks, returns, etc.

3.1 Type

Types all have upcase class names in case of mixing up with Java type names.
The inheritance:

- TYPE
  - CHAR
  - FUNCTION
  - INT
  - VOID
  - STRING
  - NAME
  - POINTER
    - ARRAY
  - RECORD
    - UNION
    - STRUCT

CHAR, **INT**, and **VOID** are singleton classes.
Type checking mainly happened in expressions and some statements.
In expressions, if the oprands and operators doesn't match, a error would be reported. For instance, if a structure is multiplied by an integer, a "type not match" error would be reported.
And some statements require special types. For instance, the condition of **if** or **while** statements must be integer.

3.2 Left Value

Most left values checking happens in assignments. And some operator such as "&"(get address) and "++"(self increment)

3.3 Declaration

The check about declaration and use before declarated is based on symbol table. If a variable cannot be found in the symbol table while used, a "variable not declaraded" error would be reported.

3.4 Returns and Breaks

Returns and breaks are checked by some counters.

3.5 Other items

The details not metioned can be found in *Semantic.java* under *semantic* directory

A. Appendix Title

This is the text of the appendix, if you need one.

Acknowledgments

Acknowledgments, if needed.

References

[1] P. Q. Smith, and X. Y. Jones. ...reference text...