SJTU

编译课程设计报告

基于 Tiger 语言

姓名：陈斌

学号：5080309505

日期：2011 年 1 月 18 日

目录

前言

本课程设计主要参考《Modern Compiler Implementation in Java》一书以及网站 http://www.cs.princeton.edu 的关于 Tiger 语言编译描述，JFlex 以及 cup 文档。基于 Tiger 语言的编译器较其他语言简单，并且使用我们熟悉的 java 语言编写，有利于学习编译器。由于时间以及能力有限，报告中有错漏请见谅。

感谢助教的入门指导以及在次过程中帮忙解决问题的各位同学。

陈斌

2011 年 1 月 8 日

概述

目标：

实现 Tiger 语言编译器设计

步骤：

1.词法分析(Lexical Analysis) :把 Tiger 源程序分割成符号(单词),要求制作自动机文件，词法分析程序由工具 JFlex 生成

2.语法分析(Syntax Parsing) :识别程序的语法结构,检查语法错误,要求制作文法文件,语法分析程序由工具 Java Cup 生成

3.抽象语法 (Abstract Syntax Tree) :根据语法结构生成抽象语法

4.语义分析(Semantic Analysis) :进行变量和类型检查等

5.活动记录 (Activation Record) :与函数调用相关的活动记录

6.中间代码生成(Intermediate Code):生成中间表示 (IR Trees)

7.规范化(Canonicalize) :优化表达式、条件分支等,只需复制代码

8.指令选择 (Instruction Selection): 生成基本的 MIPS 汇编指令

9.活性分析与寄存器分配 (Liveness Analysis and Register Allocation)

10.使之成为整体:生成完整的编译器程序

词法分析

编写 JFlex 或者 Jlex 文件，由工具自动生成 java 文件加入工程，能够分析源程序分割出一个个 Token，提供给后续的语法分析。
工具使用：使用 JFlex 工具运行 bat 导入 jflex 文件后自动生成 Yylex.java

生成 Token 规则

形式	词法规则	分析	相关代码
Identifier	以字母开头，数字、字母、下划线组成。区分大小写	在 YYINITIAL 状态中返回对应 id	alpha= [A-Za-z] digit= [0-9] id={alpha}({alpha}|{digit}|_)* <YYINITIAL> {id}{return tok(sym.ID,yytext());}
Comment	包含在/*和*/中	"/*": 进入 COMMENT 状态， "*/":返回 YYINITIAL 状态	<YYINITIAL> "/*"{count=1;yybegin(COMMENT);} <COMMENT> {"/*"{count++;} "*/" {count--; if (count==0) {yybegin(YYINITIAL);}} [^] {} }
Whitespace	空格 制表符 换行符 回车符 分页符	不做操作	delim= [\t\n\f\r] ws = {delim}+ <YYINITIAL> {ws} {}
String	包括在”之间	“进入 STRING 状态，”回到 YYINITIAL，回车分页异常	<YYINITIAL> \" {string.setLength(0);yybegin(STRING);}
Escape sequence	\n \t \” \\ \^c \ddd \…\	在 STRING 状态内出现，遇到\进入 INSLASH 状态	control = [\\^][@A-Z[\\]^_] [\t\\]+{string.append(yytext());} \\n{string.append('\n');} \\t{string.append('\t');} \\\" {string.append('\"');} \\\\{string.append('\\');} [\n\f] {err("String error");} [ ] {string.append(' ');} control{string.append(yytext());} \\{yybegin(INSLASH);}
Punctuation Symbol	, : ; ( ) [ ] { } . + - * / = <> < <= > >= & | :=	匹配则返回 sym 中相应符号	<YYINITIAL> "-" {return tok(sym.MINUS,null);}

要点

1、在编写 JFlex 文件中要注意编写的顺序，这个会涉及到优先级问题，比如 id 应该放在最后。
2、在转义字符\ddd 中用 Integer.parseInt 强制把 ddd 转换成整型数，检查是否在 0~255 之间，(char)返回 String，否则抛出错误。
3、在 Yylex 中调用 ErrorMsg.ErrorMsg

遇到问题

本部分在一开始就遇到拦路虎，由于本机 java 环境安装在 D:\Program Files (x86)\Java 内，在更改 JFlex bat 中 JAVA_HOME 路径无论如何都无法使用上 JFlex 工具，最后在终于找到原因路径中不能有空格，负责需要以双引号括起来。

在一开始没有仔细看 tiger manual 时候编写 jflex 根据平时碰到 java 或者 C++语言的规则，比如复制使用= 判断相等使用了==，后来细看才发现 manual 中给了详细的说明是:= 和=，这是个教训。

在 Comment 中使用的是一个变量记住遇到的/*和*/，这是词法分析中唯一有点技术含量的地方，不过似乎大家大同小异。

```
ID 82
ASSIGN 83
INT 85
TO 88
STRING 93
DO 95
ID 101
ASSIGN 103
ID 106
MINUS 108
INT 110
EOF 113
```

语法分析

编写 cup 文件生成 java 文件加入工程中，在工程中调用，利用 tiger 语言的语法规则来消除左递归，给出调用的顺序，并且输入抽象语法树。

使用 cup 工具导入 cup 文件生成 Parser.java

基本框架

Symbol.Symbol 提供符号,Symbol.Table 提供符号表
Dec: 类型、变量、函数声明
DecList:声明块结构
Exp:表达式
FieldExpList:用逗号分割的表达式列表，用于函数调用
Var:简单变量，域变量，下标变量
ExpList:用分号分割的表达式列表
Ty ： 数组、记录、自定义类型
FiledList: 域列表，用于记录声明或函数声明

语法数结点定义

	语法类型	名字
终结符号	String	ID，STRING
	Integer	INT
		US,MINUS,UMINUS,TIMES,DIVIDE,ASSIGN,Dec,EQ, NEQ, LT, LE, GT, GE, AND, OR, ARRAY, OF, VAR, TYPE, DOT, NIL, FUNCTION, IF, THEN, ELSE, WHILE, FOR, TO, DO, LET, IN, END, BREAK, COMMA, COLON, SEMICOLON, LPAREN, RPAREN, LBRACK, RBRACK, LBRACE, RBRACE
非终结符号	Exp	expr
	ExpList	expr_list
	FieldList	type_field,type_fields
	FieldExpLlist	field_list
	Dec	declaration
	DecList	declaration_list
	VarDec	variable_declaration
	Var	lvalue
	TypeDec	type_declaration
	Ty	type
	FunctionDec	Function_declaration,function_declarations
	SeqExp	expr_seq

语法树节点翻译

类型	Tiger 语言	形式	Absyn 结点
expr	string	STRING: s	StringExp(s)
	integer	INT: i	IntExp(i.intValue())
	lvalue	lvalue: lv	VarExp(lv)
	- expr	MINUS: m exp:e	OpExp(IntExp(0), MINUS, e)
	expr binary-operator expr	exp: e1 BINOP2 exp: e2	OpExp(e1, BINOP, e2)
	lvalue := expr	lvalue: l ASSIGN exp: v	AssignExp(l, v)
	id (expr-list)	ID:I LPAREN explist:elist RPAREN	CallExp(symbol(i), elist)
	(expr-seq)	LPAREN:l explist:elist RPAREN	SeqExp(elist)
	( )	LPAREN:l RPAREN	SeqExp(null)
	type-id { field-list }	ID:i LBRACE TypeList:flist RBRACE	RecordExp(symbol(i), flist)
	type-id { }	ID:i LBRACE RBRACE	RecordExp(symbol(i), null)
	type-id [expr] of expr	ID:i LBRACK exp:type LBRACK OF exp:value	ArrayExp(symbol(i), type, value)
	if expr then expr	IF:i exp:cond THEN exp:true	IfExp(cond, true, null)
	if expr then else expr	IF:I exp:cond THEN exp:true ELSE exp:false	IfExp(cond, true, false)
	while expr do expr	WHILE:i exp:cond DO exp:body	WhileExp(cond, body)
	for id := expr to expr do expr	FOR:f ID:i ASSIGN exp:low TO exp:high DO exp:body	ForExp(symbol(i), low, high, body)
	break	BREAK: i	BreakExp( )
	let declaration-list in expr-seqopt end	LET:i declarationlist:declist IN expseq:eseq END	LetExp(declist, eseq)
expr-list	expr	exp: e	ExpList(e, null)
	expr, expr-list	exp: e COMMA explist: elist	ExpList(e, elist)
expr-seq	expr	exp: e	
	expr; expr-seq	exp: e SEMICOLON epseq: eseq	SeqExp(e, eseq)
field-list	id = expr	ID:i EQ exp: type	FiledExpList(symbol(i), type, null)
	id = expr, field-list	ID:i EQ exp: type COMMA TypeList: flist	FiledExpList(symbol(i), type, flist)
lvalue	id	ID:i	SimpleVar(symbol(i));
	id . lvalue	ID:i DOT lvalue: l	FiledVar(symbol(i), l)

	lvalue [expr]	ID: i LBRACK exp:e RBRACK	SubscriptVar(symbol(i), e)
declaration-list	declaration	declaration: dec	DecList(dec, null)
	declaration, declaration-list	declaration: dec declarationlist: declist	DecList(dec, declist)
declaration	type-declaration	typedeclist: declist	declist
	variable-declaration	vardec: dec	vardec
	function-declaration	funcdeclist: declist	declist
type-declaration	type type-id = type	Type:i ID:id EQ type: t	TypeDec(symbol(id), t)
type	type-id	ID: i	NameTy(symbol(i))
	{ type-fields }	LBRACE:i typelist:tlist RBRACE	RecordTy(tlist)
	{ }	LBRACE:i RBRACE	RecordTy(null)
	array of type-id	ARRAY:i OF ID:i	ArrayTy(i)
type-fields	id: type	ID:i COLON ID:t	FiledList(symbol(i), symbol(t), null)
	id: type, type-fields	ID:i COLON ID:t COMMA typelist:tlist	FieldList(symbol(i), symbol(t), tlist)
variable-declaration	var id := expr	VAR:i ID:id ASSIGN exp: value	VarDec(symbol(id), null, value)
	var id : type-id := expr	VAR:i ID:id COMMA ID:tyASSIGN exp:value	VarDec(symbol(id), NameTy(symbol(ty)), vallue)
function-declaration	function id(type-fields) = expr	FUNCTION:f ID:i LPAREN typelist:list RPAREN EQ exp:v	FunctionDec(symbol(i), list, null, v)
	function id( ) = expr	FUNCTION:f ID:i LPAREN RPAREN EQ exp:v	FunctionDec(symbol(i), null, null, v)
	function id(type-fields):type-id = expr	FUNCTION:f ID:I LPAREN typelist:list RPAREN COLON ID:t EQ exp:v	FunctionDec(symbol(i),list,NameTy(symbol(t)), v)
	function id( ) :type-id = expr	FUNCTION:f ID:i LPAREN RPAREN COLON ID:t EQ exp:v	FunctionDec(symbol(i),null,NameTy(symbol(t)), v)

要点：

在 parse 中定义 parser parser = new parser(new Yylex(inp,errorMsg), errorMsg);在 Yylex 词法分析后调用语法分析。调用 Absyn.Print 打印出抽象语法树。Cup 文件按的函数声明中注意有关于是否有参数，是否有返回的四种情况。定义负数使用的是 0 减去正整数。

遇到问题:

使用 java-cup-11a 工具生成的 java 不能和教材中给的框架匹配，所以在询问同学发现使用 10 版本没有问题，故改用 java-cup-10。

```
ForExp(
 VarDec(i,
  IntExp(10),
  true),
 StringExp( ),
 AssignExp(
  SimpleVar(i),
  OpExp(
   MINUS,
   varExp(
    SimpleVar(i)),
   IntExp(1))))FOR 78
```

语义分析（类型检查）

基本框架：

Env.Env 储存符号、变量和函数。

Env.tEnv<Symbol.Symbol, Type.Type>: 符号-类型表。

Env.vEnv<Symbol.Symbol, Semant.VarEntry>, <Symbol.Symbol, Semant.FuncEntry>: 符号-变量函数表。

语法抽象树的节点传入 Semant.Semant, 由 Semant.Semant 对其进行类型检查，并进行相关的类型翻译。

类型检查规则

调用方法	传入参数	类型检查	返回值
transExp(e)	IntExp(value)	词法分析已经检查	INT
	StringExp(value)	同上	STRING
	NilExp( )	同上	NIL
	VarExp(var)	同上	VAR 类型
	OpExp(left, right, oper)	对于= !=不能为 void <<= >=必须为 String 或者 INT	INT
	CallExp(func, args)	获得 func 参数列表 fmls 检查 args 与 fmls 是否匹配 1、 args.head 不为 void 2、 args.head 能够强制转换为 fmls.head 3、 args 和 fmls 结点个数一样	对应函数的返回类型
	RecordExp(fields, typ)	获取 RECORD 入口 typ 的参数列表 fmls 逐个检查传入参数列表	typ 对应的类型

		1、fields 与 fmls 是否匹配 fields.head.name 等于 fmls.name 2、fields.head 类型能强制转换为 fmls.name 类型 3、fields 和 fmls 的节点个数相同	
	SeqExp(list)	list 为空，无需检查 list 非空，对 list 的翻译已保证正确性，无需检查	VOID list 最后一个节点的类型
	AssignExp(var, exp)	exp 类型不能为 VOID exp 类型能强制转换为 var 类型	VOID
	IfExp(test, thenclause, elseclause)	test 为 INT thenclause 与 elseclause 类型一样	VOID 对应的 RECORD 类型 thenclause 的类型
	WhileExp(test, body)	循环条件 test 类型为 INT 循环体 body 类型为 VOID	
	BreakExp()	设置全局变量 loops 录循环嵌套数 翻译 ForExp, WhileExp 时 loops++ 翻译完成时 loops-- loops 不能为 0	VOID
	ForExp(var, hi, body)	循环变量初始值 var.init 类型为 INT 循环变量上限 hi 类型为 INT 循环变量作用域为 ForExp，使用 beginScope()标记新域 循环体 body 类型应为 VOID 循环结束退出使用 endScope()退出当前域	VOID
	LetEp(decs, body)	在 decs 里申明的变量作用域为 LetExp 使用符号表的 beginScode()标记新域 对 decs 和 body(可为空)的翻译已保证正确性，无需检查 body 翻译结束调用 endScope()退出当前域	body 为空: VOID body 非空: body 的类型

	ArrayExp(size, init, typ)	符号 typ 属于 ARRAY 类型入口 获取 ARRAY 入口 typ 的参数类型 element 数组大小 size 类型为 INT 数组初始化 init 类型能强制转换为 element	typ 对应的类型
transVar(v)	SimpleVar(name)	符号 name 属于类型入口	name 对应的类型
	FieldVar(var, field)	var 类型为 RECORD 符号 field 应为对应 RECORD 类型的参数符号之一	
	SubscriptVar(var, index)	var 类型为 ARRAY index 类型为 INT	
transDec(d)	FunctionDec		
	VarDec(name, typ, init)	init 的类型能强制转化成 typ 对应的类型	null
transTy(t)	NameTy(e)	符号 e 属于类型入口	e 对应的类型
	ArrayTy(typ)	符号 typ 属于类型入口	ARRAY(typ 对应的类型)
	Record(fields)	fields 为空，无需检查 fields 非空，对 fields 的翻译已保证正确性，无需检查	fields 的类型
transExp(e)	FieldList(name, typ, tail)	设立集合 set 存储 RECORD 的参数名 符号 name 不在 Set 中 符号 typ 属于类型入口 转化成相应的 RECORD 节点	转化后的 RECORD 类型

要点：

上表是结合 tiger manual 考虑到的类型检查，可能有错漏或者不符合的地方，尚有待完善。Tiger manual 中定义的 11 类标准 func 在 Env 中加入

遇到问题：

到了这部分需要自己大量书写代码了，一开始是一片茫然的，后面仔细阅读了虎书以及向同学请教，才慢慢开始写起来，断断续续写得非常慢，由于时间以及个人能力关系，所以类型检查这部分不能算是全部完成，调试还没有成功，比较遗憾。对此自己深感愧疚。

最后一些话：

由于刚刚做完必修的部分，其他后续部分还没有完成，没有一个属于自己的完整的编译器，同时看到一些同学的进度让我自己都感到不好意思。希望自己在后面能够继续

完成，与分数无关，起码能够收获到一份喜悦。相信每一位编程者都会对一个完成的工程感到很高兴。

感谢张老师在前面半个学期中给我们教的理论知识，没有这些知识一切都是空谈，感谢助教的评测指导，你们都辛苦了，感谢身边的同学，无论是深夜写代码影响到的宿舍的哥们，还是一起去通宵写代码的兄弟。虽然写代码是一个人对着电脑，但是没有大家我相信自己都没法坚持下来。

因为报告没有规定形式，所以希望这份报告能够基本符合要求。

陈斌

2010 年 1 月 11 日