# 

# 编译原理综合训练实验报告

学院：软件学院

班级：软件21班

学号：2121601003

姓名：李晓萌

2015年7月25日

1. 语言文法的形式化描述(BNF范式)

**#开始部分**

S ::= <语句组>

语句组 ::= <语句>

语句组 ::= <语句组><语句>

语句 ::= <声明语句>

语句 ::= <执行语句>

**#声明语句**

声明语句 ::= <全局变量定义语句>

声明语句 ::= <函数声明语句>

函数声明语句 ::= <函数头>{<局部语句组或空>

函数头 ::= <类型和函数名>(<形式参数列表>)

类型和函数名 ::= <类型>id

形式参数列表 ::= <形式参数>

形式参数列表 ::= <形式参数列表>com<形式参数>

形式参数列表 ::=

形式参数 ::= <类型>id

局部语句组或空 ::= <局部语句组>}

局部语句组或空 ::= }

局部语句组 ::= <局部语句>

局部语句组 ::= <局部语句组><局部语句>

局部语句 ::= <局部变量定义语句>

局部语句 ::= <执行语句>

局部语句 ::= <return语句>

return语句 ::= return<值>semi

局部变量定义语句 ::= <局部声明部分>semi

局部声明部分 ::= <类型><局部声明元>

局部声明部分 ::= <局部声明部分>com<局部声明元>

局部声明元 ::= id

局部声明元 ::= <局部数组参数表>]

局部数组参数表 ::= <局部数组参数表>com number

局部数组参数表 ::= id [ number

**#全局变量定义部分**

全局变量定义语句 ::= <全局声明部分>semi

全局声明部分 ::= <全局声明部分>com<全局声明元>

全局声明部分 ::= <类型><全局声明元>

全局声明元 ::= <全局数组参数表>]

全局声明元 ::= id

全局数组参数表 ::= <全局数组参数表>com number

全局数组参数表 ::= id [ number

**#执行语句部分**

执行语句 ::= <控制语句>

执行语句 ::= <求值语句>

控制语句 ::= <if语句>

控制语句 ::= <while语句>

if语句 ::= <没有else的if语句>

if语句 ::= <没有else的if语句><else子句>

没有else的if语句 ::= <if语句头>{<执行语句组或空>

if语句头 ::= if(<布尔表达式>)

else子句 ::= <else字符串>{<执行语句组或空>

else字符串 ::= else

while语句 ::= <while语句头>{<执行语句组或空>

while语句头 ::= <while字符串> ( <布尔表达式> )

while字符串 ::= while

执行语句组或空 ::= <执行语句组> }

执行语句组或空 ::= }

执行语句组 ::= <执行语句>

执行语句组 ::= <执行语句组><执行语句>

**#表达式部分**

布尔表达式 ::= <与或值>

与或值 ::= <与前缀><非值>

与或值 ::= <非值>

与或值 ::= <或前缀><非值>

与前缀 ::= <与或值> and

或前缀 ::= <与或值> or

非值 ::= not <元布尔值>

非值 ::= <元布尔值>

元布尔值 ::= <值><关系运算符><值>

元布尔值 ::= ( <布尔表达式> )

元布尔值 ::= true

元布尔值 ::= false

**#关系运算符部分**

关系运算符 ::= <

关系运算符 ::= >

关系运算符 ::= >=

关系运算符 ::= ==

关系运算符 ::= <=

关系运算符 ::= <>

**#求值语句部分**

求值语句 ::= <值>semi

值 ::= <左值> ass <加值>

值 ::= <加值>

加值 ::= <加值>-<乘积值>

加值 ::= <加值>+<乘积值>

加值 ::= <乘积值>

乘积值 ::= <乘积值>\*<自增值>

乘积值 ::= <自增值> 自增值 ::= <元值> 自增值 ::= ++<左值> 自增值 ::= <左值>++ 元值 ::= <左值> 元值 ::= number

元值 ::= (<值>)

元值 ::= <函数调用>

左值 ::= <数组访问参数表>]

左值 ::= id

数组访问参数表 ::= id [ <数组访问参数>

数组访问参数表 ::= <数组访问参数表> com <数组访问参数>

数组访问参数 ::= <值>

函数调用 ::= id ( <实参列表>)

实参列表 ::= <实参> 实参列表 ::= <实参列表> com <实参>

实参列表 ::=

实参 ::= <值>

**#类型定义部分**

类型 ::= bool

类型 ::= int

类型 ::= char

类型 ::= real

1. 单词种别定义

采用一符一种的定义方式，并且为每个符号取名字和设置优先级，对于同一个输入字符串可识别的单词符号不止一种时，将其识别为优先级较高的单词种类。

|  |  |  |
| --- | --- | --- |
| **正规式** | **种别** | **优先级** |
| i.n.t | int | 2 |
| f.a.l.s.e | false | 2 |
| t.r.u.e | true | 2 |
| b.o.o.l | bool | 2 |
| r.e.a.l | real | 2 |
| c.h.a.r | char | 2 |
| w.h.i.l.e | while | 2 |
| i.f | if | 2 |
| e.l.s.e | else | 2 |
| r.e.t.u.r.n | return | 2 |
| :.= | ass | 2 |
| < | < | 2 |
| > | > | 2 |
| >.= | >= | 2 |
| <.= | <= | 2 |
| < .> | < > | 2 |
| ( | ( | 2 |
| ) | ) | 2 |
| &.& | and | 2 |
| | .| | or | 2 |
| ~.~ | not | 2 |
| [ | [ | 2 |
| ] | ] | 2 |
| { | { | 2 |
| } | } | 2 |
| ; | semi | 2 |
| , | com | 2 |
| +.+ | ++ | 2 |
| + | + | 2 |
| - | - | 2 |
| \* | \* | 2 |
| (\_|a-z|A-Z).(0-9|\_|a-z|A-Z)\* | id | 1 |
| (0-9)+|((0-9)+.\..(0-9)+) | number | 1 |

源程序

1. 编译器的总框架

词法分析

利用DFA识别单词符号，作为语法分析器的输入

语法分析

将单词符号转化为语法单位

语义分析

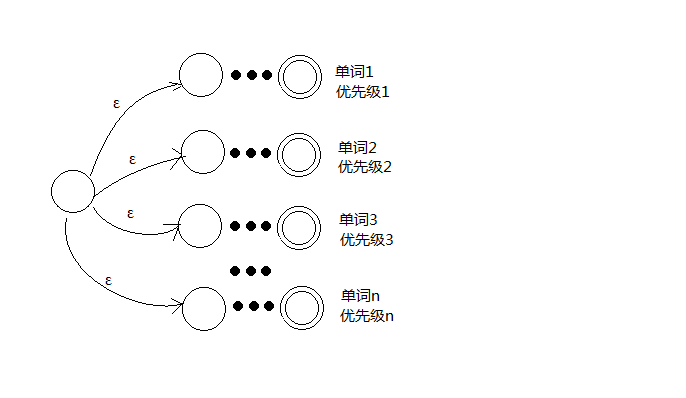
在语法分析同时进行语义分析，执行语义动作

生成中间代码

生成符号表和四元式序列

1. 关键算法的文字解释
2. 识别单词符号的DFA的构造算法：

DFA由NFA转化而来。因此首先构造NFA。NFA在wordlist.properties文件的解析过程中逐步构造；文件的每一条目的格式为：正规式=优先级，种别，例如：i.n.t=2,int，i.n.t为正规式，2为优先级，int为单词种别。正规式中运算符包括(、)、+、\*、.、|，其作用与在课程所学的正规式中一致。正规式运算的对象为自动机的子集，得到的最终结果的开始状态将被用epsilon弧连接在一个开始节点上，形成类似下图的NFA：



然后对这个自动机运行确定化算法，将其确定为DFA即可用于识别单词符号。在确定化过程中若有某个状态含有多个终态节点，保留优先级最高的节点的信息。

识别单词符号时，先将自动机复位，然后从下一个非空白字符开始，逐个输入自动机，直到自动机对于当前输入字符没有状态转移为止。此时回吐最后输入的字符，查看最后一次状态，若为终态节点，则将之前匹配的字符（长度大于0）标记为该种别的单词，输出给语法分析器；否则报告一个词法错误。

1. SLR(1)分析表的构造算法：

首先从grammar.properties得到产生式的集合，文件的条目格式为:产生式左部=右部1@语义动作的路径1|右部2@语义动作的路径2，例如：S=a，E@action.a | b@action.b，它表示两条左部相同的产生式和相应的语义动作路径，其中逗号为文法符号的分隔符。得到所有的产生式后，依次求出文法的非终结符和终结符、所有产生式的项目和项目集规范族、所有非终结符的FIRST集、所有非终结符的FOLLOW集， 然后依照课本中SLR(1)分析表的构造方法构造SLR(1)分析表

1. 语法分析器的执行算法：

语法分析器维护状态栈和符号栈。开始时将状态‘0’压入状态栈，‘#’压入符号栈。循环从词法分析器中取一个单词符号，查看状态栈的栈顶的状态，根据此状态和当前输入单词符号对照分析表，执行相应的动作，若查看分析表时该表项为空白(null)，则出错，抛出语法异常，此时执行错误建议算法；动作类型分为移进、规约、GOTO和ACC四种，若动作类型为移进，将当前输入符号压入符号栈，将动作所示的下一状态压入状态栈；若动作类型为规约，则用动作所示的产生式进行规约，计算出产生式右部的文法符号的数目，并在状态栈和符号栈弹出同等数目的元素，将产生式左部符号压入符号栈，同时执行语义动作；若动作类型为GOTO，将动作所示的下一状态压入状态栈；若动作类型为ACC，从符号栈和状态栈中弹出与动作所示的产生式右部文法符号相等数目的元素，执行语义动作，分析结束。

1. 错误建议算法；

此错误建议算法仅限于程序中缺少一个符号的时候给出可能的建议方案。算法的工作原理如下：例如有关系表达式a < 2，如果不小心输入了a 2，程序在2处出错，则错误建议算法在2之前尝试每一个终结符，将其填在那个空位上，模拟语法分析器进行分析，如果能顺利将2规约掉，则将此终结符作为建议输入之一，直到尝试完所有的终结符，形成建议方案；对于程序中输入错误或者缺少不止一个符号的情况，则此错误建议算法失效，无法给出建议方案。

1. 程序结构简介：

本程序共7个包

packege1：dtaautomat ，存放自动机的相关信息。

main class: DFAAutomat类，该类为构造识别单词符号的DNA类，可 以构造NFA、将NFA确定化为DFA、执行DFA的所有功 能等。

State类为构成自动机的状态类，记录状态节点的类型和优 先级等信息。

packege2：language ，定义对文法操作相关的一些类。

main class:AnalyzeTable类，保存SLR（1）分析表，可以根据当前状态和输入符号判断下一个执行动作，可以将分析表作为对象保存为文件，避免每次执行程序都要生成分析表，提高执行效率。

GrammarAnalyzer类，根据文法输入文件将其解析为产生式，并由产生式得到项目和项目集规范族、得到文法符号的FIRST集和FOLLOW集，进而构造出SLR（1）分析表。

Action类，记录了动作的类型，动作所需状态和产生式等信息。

Production类，记录产生式的所有信息，包括产生式左部left，产生式右部文法符号列表right和执行动作的方法的路径。

packege3：lexical 定义词法分析执行相关的类。

main class: LexicalAnalyzerImpl类，LexicalAnalyzer接口实现类，利用DFA将输入程序识别为单词符号串，同时记录符号的位置信息。

TableReader类，解析由正规式列表定义的单词文件，获得DFA。

packege4：main

main class:Entry类，整个程序的入口。

该包中还存放文法文件grammar.properties和单词定义文件wordlist.properties和一些测试输入文件inputX.txt

packege5：main.syntax.action ，定义不同类型的语义动作类。

main class:ControlAction类，定义控制语句执行的语义动作。

ExpAction类，定义表达式语句执行的语义动作。

FunAction类，定义函数定义语句执行的语义动作。

GlobalAction类，定义全局语句执行的语义动作。

ValueAction类，定义与值相关的语句执行的语义动作。

Vstmt类，定义声明语句执行的语义动作。

packege6：syntax 定义

main class:SymbolTable类，定义与符号相关的一些操作，如增加、查询符号是否定义、获得和释放临时变量、为全局变量和临 时变量分配地址空间等。

SyntaxAnalyzerImpl类,实现SyntaxAnalyzer接口，从词法分析器中取单词符号，利用分析表进行语法分析并执行语义动作，时会生成中间代码，如四元式序列和符号等信息。

Quad类，定义四元式的结构和输出形式。

Symbol类，定义对符号操作，如增加属性、由属性名查询属性值、输出的格式等。

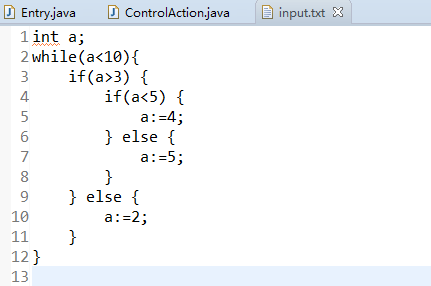
V类，记录文法符号的信息和操作，如文法符号的名称、具有的属性、是否为终结符、增加和查询属性等。

packege7：syntax.action 定义语义动作类需要的操作。

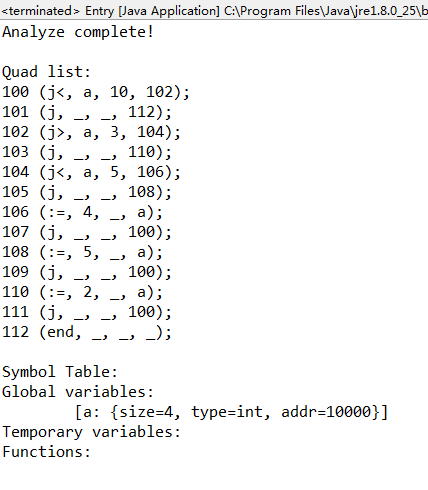
main class: SemanticAction类，所有语义动作类的父类，定义了四元式的操作，如获得下一条四元式的编号、生成一条四元式、由编号查询四元式、设置符号表等。

1. 测试样例(输入 & 输出)
2. 测试控制语句

输入:

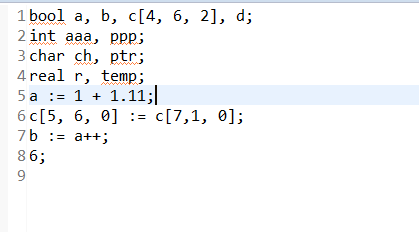


输出:

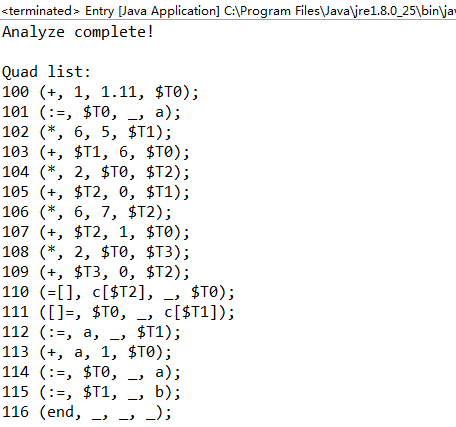


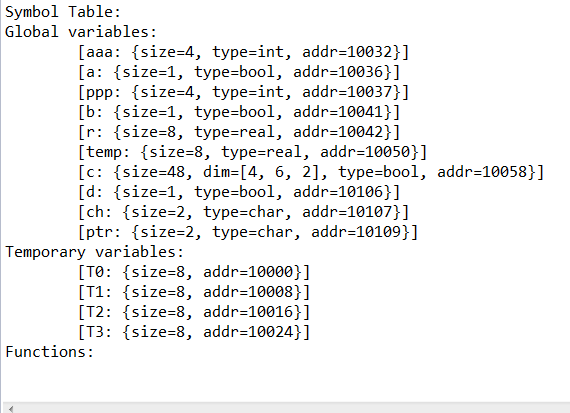
2、测试声明语句和赋值语句(包括四种数据类型，数组类型)

输入:



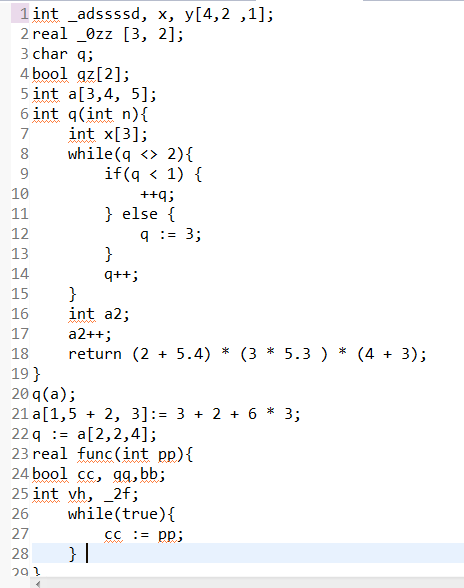
输出:



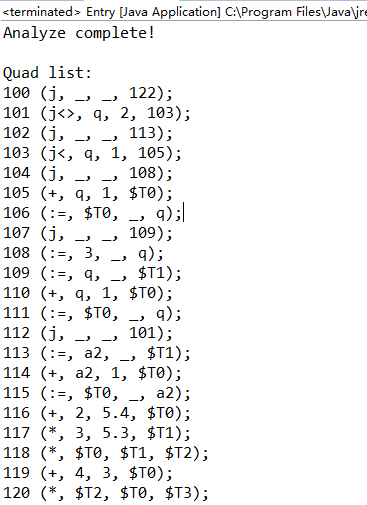


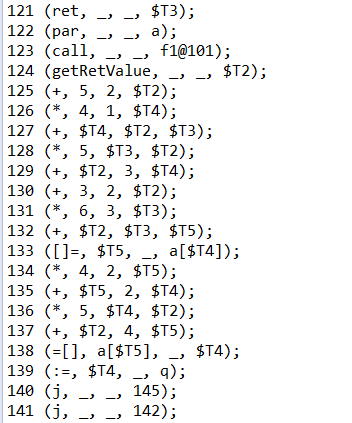
3、测试函数声明和函数调用语句

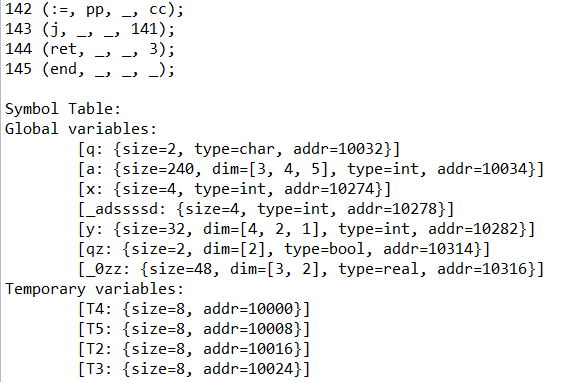
输入：



输出：







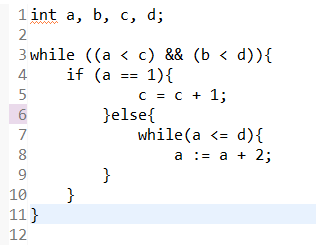
Functions:

f1: RetType = int,BaseAddr = 101, [Parameters: [n: {type=int}]], [LocalVariables: [a2: {type=int}], [x: {dim=[3], type=int}], [n: {type=int}]]

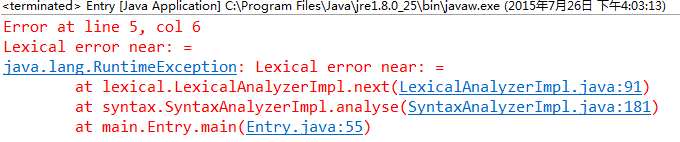
f2: RetType = real,BaseAddr = 141, [Parameters: [pp: {type=int}]], [LocalVariables: [pp: {type=int}], [cc: {type=bool}], [qq: {type=bool}], [bb: {type=bool}], [vh: {type=int}], [\_2f: {type=int}]]

4、测试词法错误处理

输入：（定义的赋值符号为:=，不是=，故c=c+1处出错）

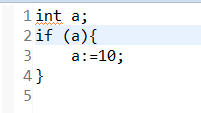


输出：

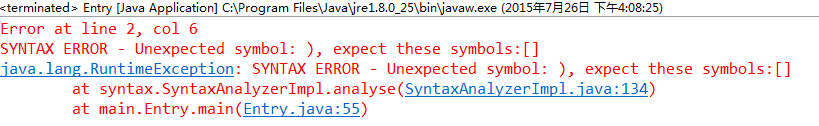


5、测试语法错误处理

输入：（if后面的括号中必须是一个布尔表达式，只有一个符号是错误的）

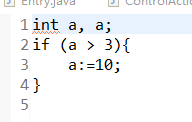


输出：

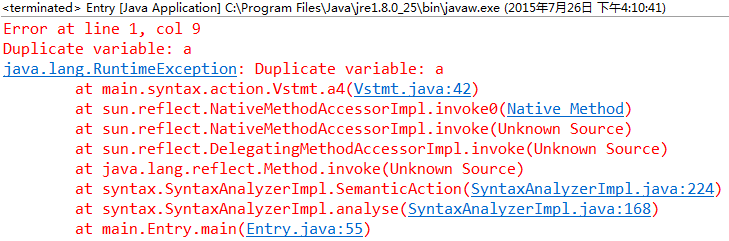


6、测试语义错误处理

输入：

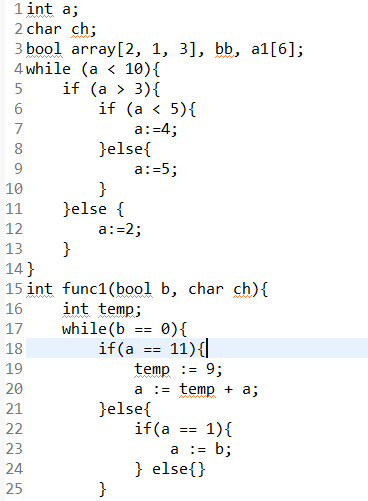


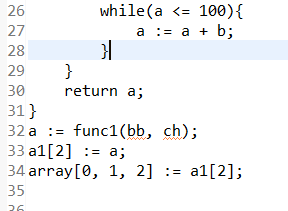
输出：



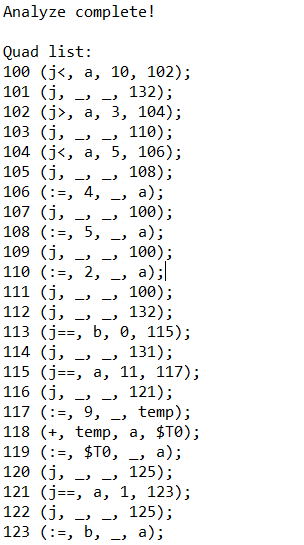
7、测试复杂程序的翻译

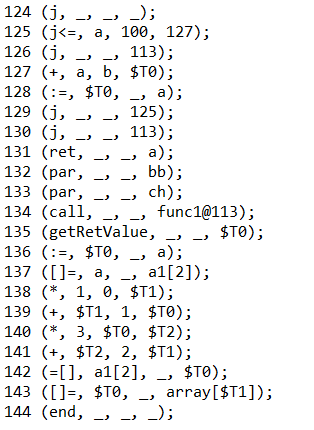
输入：

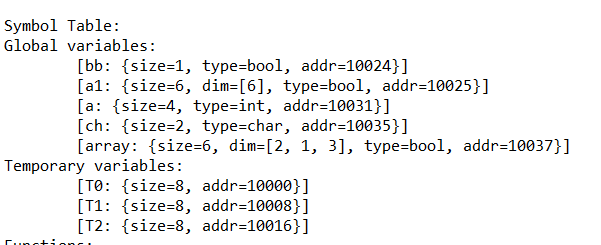




输出：







Functions:

func1: RetType = int,BaseAddr = 113, [Parameters: [b: {type=bool}], [ch: {type=char}]], [LocalVariables: [b: {type=bool}], [temp: {type=int}], [ch: {type=char}]]

1. 实验总结

我做这个编译器大概经历了二十天，一开始信心满满，因为刚学完编译原理这门课，我觉得对编译器的理解还可以，只要理解老师所讲的东西，一步一步做下去就足够了。开始着手做了，构造DFA、词法分析器、SLR(1)分析表、语法分析器，这几步还不是很难，没有遇到很多问题。之前还没有构造文法，用的都是一些比较简单的文法片段，仅仅是为了验证程序的正确性。当开始写语义动作的时候，就需要设计本编译器的文法了。此时我才觉得老师讲的东西并不够，设计文法还要考虑其他的东西，比如如何把书上讲的零散的部分整合起来，如何添加新的部分构成完整的文法。在研究了好久老师讲的知识之后，我突然意识到其实老师讲的是方法，不是结果。在我真正了解了文法构造的方法之后，文法设计起来并非想象的那么难了。在之后的工作中，有时候做不下去了，又回过头来检查之前写的，进行修改、完善，也不断的修改文法。语义动作是最难的部分，也是做得最久的部分。为语义添加动作时，每个产生式的动作应该是怎样的，如何把他们联系起来，由于自己的考虑不周，或者遗漏了一些信息，或者组织不当，遇到了很多bug，在经过认真思考之后，才将他们解决。符号表没语义动作那么难，在明白符号表如何组织、各种符号如何存放之后，做起来还算顺利。当然在做实验的过程中，我也请教同学或者上网查资料解决问题。

虽然实验做完了，但由于我对编译器的理解有限以及自己的能力有限，实验存在很多不完美的地方需要去改进和完善。

这次的实验对我来说是一次挑战，一次锻炼自己、提升自己编程能力和解决问题能力的机会。编译原理为我打开了探索编译器奥秘之门。而经过这次实验，为对编译器的组织结构和工作原理也有了更深的认识和理解。最后，感谢老师给我们提供这样一次提升自我的机会，感谢实验中给予我帮助的小伙伴。

1. 附录

程序源文件清单

DFAAutomat.java

package dfaautomat;

import java.io.InputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.OutputStream;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

/\*\*

\* DFAAutomat类,

\* DFA自动机,实现Serializable接口,将保存到文件中

\* @author lxm

\*

\*/

public class DFAAutomat implements Serializable{

public static final char EPSLON = '\0'; // EPSLON表示空字符

private static final long serialVersionUID = -4274687974256578446L; // 序列号ID

private State startState; // 开始状态

private Map<Group, State> converts; // 状态转换表

private Set<State> finalStates; // 终态集

private transient State currState; // 记录当前状态(暂态,对象文件中不保存)

private DFAAutomat(State startState,

Map<Group, State> converts, Set<State> finalStates) {

this.startState = startState;

this.converts = converts;

this.finalStates = finalStates;

reset();

}

/\*\*

\* 重置自动机

\*/

public void reset(){

currState = startState;

}

/\*\*

\* 根据转换表返回下一状态

\* @param input

\* @return currState

\*/

public State next(char input){

return currState = converts.get(Group.getComparer(currState, input));

}

public State next(String input){

for (char ch: input.toCharArray()) {

next(ch);

}

return currState;

}

/\*\*

\* 判断是否到达终态

\* @param s

\* @return

\*/

public boolean isFinal(State s){

return finalStates.contains(s);

}

public boolean isFinal(){

return finalStates.contains(currState);

}

/\*\*

\* 将类对象写入文件

\* @param os

\*/

public void saveTo(OutputStream os){

try {

ObjectOutputStream oos = new ObjectOutputStream(os);

oos.writeObject(this); // 向文件写入对象

oos.close();

} catch(Exception e) {

e.printStackTrace();

}

}

/\*\*

\* 从文件中加载类对象

\* @param is

\* @return ret

\*/

public static DFAAutomat load(InputStream is) {

DFAAutomat ret = null;

try {

ObjectInputStream ois = new ObjectInputStream(is);

ret = (DFAAutomat)ois.readObject(); // 读取对象

ret.reset();

ois.close();

} catch(Exception e) {

e.printStackTrace();

}

return ret;

}

/\*\*

\* 调用内部类,构造自动机

\* @return

\*/

public static DFAConstructor constructor(){

return new DFAConstructor();

}

/\*\*

\* 内部类

\* 构造自动机

\* @author lxm

\*

\*/

public static class DFAConstructor {

private Map<Group, State> map = new HashMap<Group, State>(); // 转换表

private Set<State> finals = new HashSet<State>(); // 终态集

private State begin; // 初始状态

private DFAConstructor() {}

/\*\*

\* 增加一条边

\* @param from

\* @param via

\* @param to

\*/

public void edge(State from, char via, State to){

map.put(new Group(from, via), to);

}

/\*\*

\* 得到终态集

\* @param s

\*/

public void finals(State... s) {

finals.addAll(Arrays.asList(s));

}

public void begin(State s) {

begin = s;

}

/\*\*

\* 构造DFA

\* @return

\*/

public DFAAutomat finish(){

return new DFAAutomat(begin, map, finals);

}

}

/\*\*

\* 调用内部类NFAConstructor,构造NFA

\* @return

\*/

public static NFAConstructor constructorN(){

return new NFAConstructor();

}

/\*\*

\* 内部类

\* NFA自动机

\* @author lxm

\*

\*/

public static class NFAConstructor {

private Map<Group, Set<State>> map = new HashMap<Group, Set<State>>(); // 转换表

private Set<State> finals = new HashSet<State>(); // 终态集

private Set<State> begin = new HashSet<State>(); // 初始状态集

private Set<Character> alpha = new HashSet<Character>(); // 字母表

private NFAConstructor() {}

/\*\*

\* 增加一条边

\* @param from

\* @param via

\* @param to

\*/

public void edge(State from, char via, State... to){

if(via != EPSLON){

alpha.add(via);

}

Set<State> mSet = map.get(Group.getComparer(from, via));

if(mSet == null){

mSet = new HashSet<State>();

map.put(new Group(from, via), mSet);

}

mSet.addAll(Arrays.asList(to));

}

public void finals(State... s) {

finals.addAll(Arrays.asList(s));

}

public void begin(State... s) {

begin.addAll(Arrays.asList(s));

}

/\*\*

\* 将NFA转换为DFA

\* @return

\*/

public DFAAutomat finish(){

State begin0 = null; // DFA的初始状态

Map<Group, State> map0 = new HashMap<Group, State>(); // 状态转换集合

Set<State> finals0 = new HashSet<State>(); // DFA的终态集

Map<Set<State>,Integer> clMap = new HashMap<Set<State>,Integer>(); // 状态集对应编号

Set<State> s = new HashSet<State>();

s.addAll(begin);

List<State> sl = new ArrayList<State>(); // 为状态重新编号

Set<State> bg = closure(s);

clMap.put(bg, 0);

sl.add(new State());

List<Set<State>> il = new ArrayList<Set<State>>();

il.add(bg);

for(int i = 0; i < sl.size(); i ++){

for(char chx : alpha){

Set<State> tmp;

if((tmp = closure(il.get(i), chx)).size() != 0){

Integer to = clMap.get(tmp);

if(to == null){

to = sl.size();

clMap.put(tmp, to);

sl.add(new State());

il.add(tmp);

}

map0.put(new Group(sl.get(i) , chx),sl.get(to));

}

}

}

for(int i = 0; i < sl.size(); i++){

boolean isFinal = false;

State x = sl.get(i);

for(State sx : il.get(i)){

isFinal = isFinal || finals.contains(sx);

if (x.priority < sx.priority) {

x.priority = sx.priority;

x.type = sx.type;

}

}

if (isFinal)

finals0.add(x);

}

begin0 = sl.get(0);

return new DFAAutomat(begin0, map0, finals0);

}

/\*\*

\* 求EPSLON闭包

\* @param set

\* @return

\*/

private Set<State> closure(Set<State> set){

return closure(set, EPSLON);

}

/\*\*

\* 计算相同输入字符的状态集的闭包

\* @param set

\* @param ch

\* @return

\*/

private Set<State> closure(Set<State> set, char ch) {

Set<State> ret = new HashSet<State>(); // 当前状态集

Set<State> tmp = new HashSet<State>();

// 计算输入字符的闭包

for(State sx : set) {

Set<State> snext;

if ((snext = map.get(Group.getComparer(sx, ch))) != null) {

ret.addAll(snext);

}

}

// 计算EPSLON闭包

boolean change = true;

while(change) {

change = false;

tmp.clear();

tmp.addAll(ret);

for(State sx : tmp) {

Set<State> snext;

if ((snext = map.get(Group.getComparer(sx, EPSLON))) != null) {

change = ret.addAll(snext) || change;

}

}

}

return ret;

}

}

}

/\*\*

\* Group类

\* 记录当前状态和输入字符

\* @author lxm

\*

\*/

class Group implements Serializable{

private static final long serialVersionUID = -6497235445783214316L;

private static Group BUF = new Group(null, (char)0);

public static Group getComparer(State s, char ch) {

BUF.state = s;

BUF.ch = ch;

return BUF;

}

private State state;

private char ch;

public Group(State state, char ch) {

this.state = state;

this.ch = ch;

}

@Override

public boolean equals(Object o) {

if (o instanceof Group){

return ((Group) o).state == state && ch == ((Group) o).ch;

}

return false;

}

@Override

public int hashCode() {

return (state == null? 0 : state.hashCode() << 8) + ch;

}

}

AnalyzeTable.java

package language;

import java.io.InputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.OutputStream;

import java.io.Serializable;

import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

/\*\*

\* SLR(1)分析表,记录当前状态,输入符号和执行动作

\* @author lxm

\*

\*/

public class AnalyzeTable implements Serializable{

private static final long serialVersionUID = 8106377333742804836L;

private Map<Group, Action> analyzeTable = new HashMap<Group, Action>(); // 分析表map

private Group gQuery = new Group(0, null);

private Set<String> vtset = null; //

AnalyzeTable(Map<Group, Action> analyzeTable) {

this.analyzeTable = analyzeTable;

}

public void setVT(Set<String> vt) {

this.vtset = vt;

}

public Set<String> getVT(){

return vtset;

}

public Action query(int state, String input) {

gQuery.setFrom(state);

gQuery.setVia(input);

return analyzeTable.get(gQuery);

}

public static AnalyzeTable load(InputStream is){

AnalyzeTable table = null;

try {

ObjectInputStream ois = new ObjectInputStream(is);

table = (AnalyzeTable) ois.readObject();

ois.close();

} catch(Exception e) {

throw new RuntimeException(e);

}

return table;

}

public void save(OutputStream os){

try {

ObjectOutputStream oos = new ObjectOutputStream(os);

oos.writeObject(this);

oos.close();

} catch(Exception e) {

throw new RuntimeException(e);

}

}

public void show() {

for(Entry<Group, Action> ex : analyzeTable.entrySet()){

System.out.println("from: " + ex.getKey().getFrom() +

" via: " + ex.getKey().getVia() + " Action: " + ex.getValue());

}

}

}

GrammarAnalyser.java

package language;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Properties;

import java.util.Set;

/\*\*

\* 语法分析类</br>

\* 得到SLR(1)分析表

\* @author lxm

\*

\*/

public class GrammarAnalyser {

public static final String EPSLON = "\_", TERMINATOR = "#", START = "S";

private Properties p;

private Map<String, Set<Production>> map = new HashMap<String, Set<Production>>(); // 左部相同的产生式集合

private List<Set<Item>> itemfamily = new ArrayList<>(); // 项目集规范族

private Set<String> vset = new HashSet<>(); // 所有符号的集合(包括终结符和非终结符)

private Set<Item> psset = new HashSet<Item>(); // 所有项目的集合

private Set<String> vnset = new HashSet<String>(); // 非终结符集合

private Map<String, Set<String>> firstSet = new HashMap<String, Set<String>>(); // 所有非终结符的FIRST集合

private Map<String, Set<String>> followSet = new HashMap<String, Set<String>>(); // 所有非终结符的FOLLOW集合

private Map<Group, Integer> gotomap = new HashMap<Group, Integer>(); // GOTO转换表

private AnalyzeTable at;

public GrammarAnalyser(InputStream is) {

p = new Properties();

try {

p.load(new InputStreamReader(is, "utf-8"));

} catch (IOException e) {

e.printStackTrace();

}

// 提取产生式集合

for(Entry<Object, Object> x : p.entrySet()){ // 遍历属性表

Set<Production> xset = new HashSet<Production>();

String left = ((String) x.getKey()).trim(); // 得到产生式的左部

vset.add(left); // 将左部文法符号加入文法符号集合

String[] ss = ((String) x.getValue()).split("\\|"); // 用|提取每一右部

for(int i = 0;i < ss.length; i++){

Production p = new Production();

String[] str = ss[i].split("@");

if(str.length == 1){

p.setAction(null);

} else if(str.length == 2){

p.setAction(str[1].trim()); // 设置产生式的动作路径

} else {

throw new RuntimeException("Incorrect number of @: " + ss[i]);

}

String[] arr = str[0].trim().split(" \*, \*"); // 提取右部所有的文法符号 // 提取每一右部的执行动作的方法的路径

if(arr.length == 1 && arr[0].equals(EPSLON)){

p.setRight(new String[0]);

} else {

p.setRight(arr);

}

vset.addAll(Arrays.asList(p.getRight())); // 加入文法符号集合

p.setLeft(left); // 设置产生式的左部

vnset.add(left);

xset.add(p);

}

map.put(left, xset);

}

Set<Item> sItems = getItemSet(map);

//System.out.println(sItems.toString());

//System.out.println("文法符号:\n" + vset.toString());

//System.out.println("非终结符集:\n" + vnset.toString());

automat(sItems);

//System.out.println("GOTO表:\n" + mGotos.toString());

getFirstSet(map);

getFollowSet(map);

//System.out.println("FIRST SET:" + .toString());

//System.out.println("FOLLOW SET:" +.toString());

Set<String> vt = new HashSet<String>();

vt.addAll(vset);

vt.removeAll(vnset);

at = constructor();

at.setVT(vt);

//System.out.println(getBNFGrammar());

}

/\*\*

\* 得到BNF范式文法

\* @return

\*/

public String getBNFGrammar(){

StringBuilder sb = new StringBuilder();

List<BNFRecord> outList = new ArrayList<>();

Set<String> remaining = new HashSet<String>(vnset);

dfsSearch(outList, remaining, START, 0);

for(int i = 0; i < outList.size(); i++) {

BNFRecord br = outList.get(i);

for(Production px : map.get(br.vn)){

for(int k = 0; k < br.w; k++){

sb.append('\t');

}

sb.append(px.getLeft()).append(" ::= ");

for(String rx : px.getRight()) {

if (vnset.contains(rx)) {

sb.append("<").append(rx).append(">");

} else {

sb.append(rx);

}

}

sb.append("\n");

}

sb.append("\n");

}

return sb.toString();

}

/\*\*

\* 深度优先遍历BNF列表

\* @param outList

\* @param remaining

\* @param curr

\* @param w

\*/

private void dfsSearch(List<BNFRecord> outList, Set<String> remaining, String curr, int w) {

outList.add(new BNFRecord(curr, w));

remaining.remove(curr);

for(Production px : map.get(curr)) {

for(String rx : px.getRight()) {

if (remaining.contains(rx)) {

dfsSearch(outList, remaining, rx, w + 1);

}

}

}

}

public AnalyzeTable getAnalyzeTable() {

return at;

}

public static void main(String[] args) throws IOException{

String savePath = "e:\\table1.atab";

AnalyzeTable at = new GrammarAnalyser(GrammarAnalyser.class.getResourceAsStream("/test/grammar.properties")).getAnalyzeTable();

at.show();

at.save(new FileOutputStream(savePath));

System.out.println("File saved at '" + savePath + "'.");

//new GrammarAnalyser(GrammarAnalyser.class.getResourceAsStream("/test/grammar.properties"));

}

/\*\*

\* 返回项目集

\* @param map

\* @return psset

\*/

private Set<Item> getItemSet(Map<String, Set<Production>> map){

for(Entry<String, Set<Production>> ex : map.entrySet()){

for(Production px : ex.getValue()){

int pos = 0;

int len = px.getRight().length; // 得到右部长度

Item last = null;

// 产生项目

while(pos < len + 1){

Item ps = new Item();

ps.setDotPos(pos);

ps.setPd(px);

psset.add(ps);

pos++;

if (last != null) {

last.setNext(ps);

}

last = ps; // 保存上一个项目

}

}

}

return psset;

}

/\*\*

\* 获得非终结符的First集

\* @param pros

\* @return firstSet

\*/

private Map<String, Set<String>> getFirstSet(Map<String, Set<Production>> pros){

Set<Map.Entry<String, Set<Production>>> entries = pros.entrySet();

for(Entry<String, Set<Production> > ex : entries){

String vn = ex.getKey();

firstSet.put(vn, new HashSet<String>()); // 初始化first集

}

boolean isChanged = true;

while(isChanged){ // 循环直到所有的first集不再改变

isChanged = false; // 标记first集是否变化

for(Entry<String, Set<Production> > ex : entries){ // 遍历所有的产生式

String left = ex.getKey();

Set<String> currFirstSet = firstSet.get(left);

int beginSize = currFirstSet.size();

for(Production p : ex.getValue()){

if(p.getRight().length != 0){

String tmp = p.getRight()[0];

if(vnset.contains(tmp)){ // 产生式右部第一个符号是非终结符

currFirstSet.addAll(firstSet.get(tmp)); // 将此非终结符的first集加入当前的first集

if(firstSet.get(tmp).contains(EPSLON)){ // 非终结符的first集包含EPSLON

int len = p.getRight().length;

int i = 1;

// 判断其后的符号是否为非终结符且其first集是否含有EPSLON

while(i < len &&

firstSet.get(p.getRight()[i - 1]).contains(EPSLON)){

if(vnset.contains(p.getRight()[i])){ // 是非终结符

currFirstSet.addAll(firstSet.get(p.getRight()[i])); // 将其first集加入

i++;

} else {

currFirstSet.add(p.getRight()[i]); // 将终结符加入

break;

}

}

if(i == len // 若之前所有符号均为非终结符

&& (firstSet.get(p.getRight()[i - 1]).contains(EPSLON))){// 最后一个符号其first集包含EPSLON

currFirstSet.add(EPSLON); // 将EPSLON加入当前first集

} else {

currFirstSet.remove(EPSLON); // 否则从first集中去除EPSLON

}

}

} else {

currFirstSet.add(tmp);

}

} else {

currFirstSet.add(EPSLON);

}

}

isChanged = isChanged || currFirstSet.size() > beginSize;

}

}

return firstSet;

}

/\*\*

\* 获得非终结符的follow集

\* @param map

\* @return followSet

\*/

private Map<String, Set<String>> getFollowSet(Map<String, Set<Production>> map){

Set<Map.Entry<String, Set<Production>>> entries = map.entrySet();

for(Entry<String, Set<Production>> ex : entries){

String vn = ex.getKey();

followSet.put(vn, new HashSet<String>()); // 初始化follow集

if (vn.equals(START)){

followSet.get(vn).add(TERMINATOR);

}

}

boolean isChanged = true;

while(isChanged){

isChanged = false; // 标记follow集是否变化

for(Entry<String, Set<Production>> ex : entries){

String left = ex.getKey();

for(Production p : ex.getValue()){

String[] right = p.getRight();

for(int i = 0; i < right.length; i++){

String tmp = right[i]; // 取出产生式右部第i个符号

if(vnset.contains(tmp)){ // 若为非终结符

Set<String> follow = followSet.get(tmp);

int followsize = follow.size();

if(i < right.length){

int x = i + 1;

while(x < right.length){ // 判断之后是否还有非终结符

if(!vnset.contains(right[x])){ // 若为终结符则终止循环

follow.add(right[x]);

break;

} else {

follow.addAll(firstSet.get(right[x]));

if(firstSet.get(right[x]).contains(EPSLON)){ // 若有且first集包含EPSLON,则查看下一个符号

follow.remove(EPSLON);

i++;

} else break;

}

}

if(x == right.length) {

follow.addAll(followSet.get(left));

}

}

isChanged = isChanged || follow.size() > followsize;

}

}

}

}

}

return followSet;

}

/\*\*

\* 构造GOTO状态转换的自动机

\* @param psset

\* @return gotomap

\*/

private Map<Group, Integer> automat(Set<Item> psset){

Set<Item> begin = new HashSet<Item>(); // 初始项目集合

for(Item psx : psset){

if(psx.getPd().getLeft().equals(START) && psx.getDotPos() == 0){

begin.add(psx);

}

}

getClosure(begin);

itemfamily.add(begin);

for(int i = 0; i < itemfamily.size(); i++){

// System.out.println( " index :" + i + ", itemset: " + itemfamily.get(i).toString());

for(String str : vset){ // 对每一个文法符号进行GOTO动作

Set<Item> sx = getItemFamily(itemfamily.get(i), str); // 得到下一个状态的项目集

if(sx.size() == 0) continue;

getClosure(sx); // 求该项目集的闭包

int to = getKey(itemfamily, sx);

if(to <= -1){ // 该项目集不存在则创建并产生GOTO转换

gotomap.put(new Group(i, str), itemfamily.size());

itemfamily.add(sx);

} else { // 否则只产生GOTO转换不创建

gotomap.put(new Group(i, str), to);

}

}

}

//System.out.println("项目集规范族集合: \n" + itemfamily.toString());

return gotomap;

}

/\*\*

\* 返回项目集

\* @param j

\* @param prefix

\* @return set

\*/

private Set<Item> getItemFamily(Set<Item> j, String prefix){

Set<Item> set = new HashSet<Item>();

for(Item ix : j){

if((ix.getDotPos() < (ix.getPd().getRight().length))

&& (ix.getPd().getRight()[ix.getDotPos()].equals(prefix))){ // 项目未到达末尾且匹配活前缀

set.add(ix.getNext()); // 将其指向的下一个项目加入集合

}

}

return set;

}

/\*\*

\* 返回项目集j的CLOSURE闭包

\* @param j

\* @return j

\*/

private Set<Item> getClosure(Set<Item> j){

int size = 0;

Set<Item> tmp = new HashSet<Item>(j); // 临时变量

while(size < j.size()) {

size = j.size();

for(Item ix : j){

String prefix = null;

if(ix.getDotPos() < ix.getPd().getRight().length){

prefix = ix.getPd().getRight()[ix.getDotPos()]; // 得到活前缀

}

for(Item xx : psset){

if(xx.getPd().getLeft().equals(prefix) && xx.getDotPos() == 0){ // 匹配左部等

tmp.add(xx);

}

}

}

Set<Item> change = j; // 交换j和tmp

j = tmp;

tmp = change;

}

return j;

}

/\*\*

\* 由项目集获得相应编号

\* @param itemfamily

\* @param si

\* @return key

\*/

private int getKey(List<Set<Item> > itemfamily, Set<Item> si){

int key = -1; // 不存在则返回-1

for( int i = 0; i < itemfamily.size(); i++){ // 扫描项目集列表

if(itemfamily.get(i).equals(si)){

key = i;

}

}

return key;

}

/\*\*

\* 将元素加入分析表

\* @param table

\* @param g

\* @param a

\*/

private void tablePut(Map<Group, Action> table, Group g, Action a){

Action ax = table.get(g);

if (ax != null && !ax.equals(a)) { // 若已存在一条转移记录,则发生冲突

System.err.println("WARNING: CONFLICT on input " + g.getVia()); // 出错

System.err.println("Existing: " + table.get(g));

System.err.println("Want to insert: " + a);

System.err.println("Position:" + g);

System.err.println("Item family:" + itemfamily.get(g.getFrom()));

return;

}

table.put(g, a);

}

/\*\*

\* 根据当前状态和输入符号得到下一个状态

\* @param curr

\* @param via

\* @return next

\*/

private int getNextState(int curr, String via){

int next = -1;

Integer ret = gotomap.get(new Group(curr, via));

if (ret != null){

next = ret;

}

return next;

}

/\*\*

\* 构造SLR(1)分析表

\* @return

\*/

private AnalyzeTable constructor(){

Map<Group, Action> table = new HashMap<>(); // 初始化分析表

for(Entry<Group, Integer> ex : gotomap.entrySet()){ // 先遍历GOTO表,加入非终结符的GOTO动作

Group g = ex.getKey();

String via = g.getVia();

int nextState = ex.getValue();

if(vnset.contains(via)){

Action a = new Action(Action.GOTO, nextState);

tablePut(table, g, a);

continue;

}

}

for(int i = 0; i < itemfamily.size(); i++){ // 遍历每一个项目集中的每一个项目的右部

Set<Item> from = itemfamily.get(i);

for(Item ix : from){

int dotpos = ix.getDotPos();

int len = ix.getPd().getRight().length; // 根据每一个规则填充分析表

if(dotpos < len){

String v = ix.getPd().getRight()[dotpos];

int next = getNextState(i, v);

if(next != -1 && !vnset.contains(v)){

Group t = new Group(i, v);

Action a = new Action(Action.STEPINTO, next);

tablePut(table, t, a);

}

} else if(ix.getPd().getLeft().equals(START)){ // 找到ACC出口

Action a = new Action();

a.setP(ix.getPd());

tablePut(table, new Group(i, TERMINATOR), a);

} else {

Set<String> follow = followSet.get(ix.getPd().getLeft());

for(String x : follow){

Group gx = new Group(i, x);

Action a = new Action(ix.getPd());

tablePut(table, gx, a);

}

}

}

}

return new AnalyzeTable(table);

}

}

/\*\*

\* group类

\* 记录上一个状态和经由的文法符号

\* from表示状态编号

\* via表示文法符号

\* @author lxm

\*/

class Group implements Serializable {

private static final long serialVersionUID = 3969890776875195720L;

private int from;

private String via;

public int getFrom() {

return from;

}

public void setFrom(int from) {

this.from = from;

}

public String getVia() {

return via;

}

public void setVia(String via) {

this.via = via;

}

public Group(int from, String via) {

super();

this.from = from;

this.via = via;

}

@Override

public int hashCode() {

final int prime = 31;

int result = 1;

result = prime \* result + from;

result = prime \* result + ((via == null) ? 0 : via.hashCode());

return result;

}

@Override

public boolean equals(Object obj) {

if(obj instanceof Group){

return ((Group) obj).from == this.from && ((Group) obj).via.equals(this.via);

}

return false;

}

@Override

public String toString() {

return "\nGroup [from=" + from + ", via=" + via + "]";

}

}

/\*\*

\* 项目类

\* 记录产生式pd,圆点位置dotPos和下一个项目next

\* @author lxm

\*

\*/

class Item implements Serializable {

private static final long serialVersionUID = 8008233021526569262L;

private Production pd;

private int dotPos;

private Item next = null;

public void setNext(Item next) {

this.next = next;

}

public Item getNext() {

return next;

}

public int getDotPos() {

return dotPos;

}

public void setDotPos(int dotPos) {

this.dotPos = dotPos;

}

public Production getPd() {

return pd;

}

public void setPd(Production pd) {

this.pd = pd;

}

@Override

public String toString() {

StringBuilder sb = new StringBuilder();

sb.append("\n");

sb.append(pd.getLeft()).append("-->");

for(int i = 0; i < pd.getRight().length; i++) {

if (i != 0)

sb.append(" ");

if (dotPos == i)

sb.append("・");

sb.append(pd.getRight()[i]);

}

if (dotPos == pd.getRight().length){

sb.append("・");

}

return sb.toString();

}

@Override

public int hashCode() {

return pd.hashCode() << 4 + dotPos;

}

}

/\*\*

\* BNF记录

\* @author lxm

\*

\*/

class BNFRecord{

public String vn; // 非终结符

public int w; // 深度

public BNFRecord(String vn, int w) {

this.vn = vn;

this.w = w;

}

}

AnalyzeTable.java

package language;

import java.io.InputStream;

import java.io.ObjectInputStream;

import java.io.ObjectOutputStream;

import java.io.OutputStream;

import java.io.Serializable;

import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

/\*\*

\* SLR(1)分析表,记录当前状态,输入符号和执行动作

\* @author lxm

\*

\*/

public class AnalyzeTable implements Serializable{

private static final long serialVersionUID = 8106377333742804836L;

private Map<Group, Action> analyzeTable = new HashMap<Group, Action>(); // 分析表map

private Group gQuery = new Group(0, null);

private Set<String> vtset = null; //

AnalyzeTable(Map<Group, Action> analyzeTable) {

this.analyzeTable = analyzeTable;

}

public void setVT(Set<String> vt) {

this.vtset = vt;

}

public Set<String> getVT(){

return vtset;

}

public Action query(int state, String input) {

gQuery.setFrom(state);

gQuery.setVia(input);

return analyzeTable.get(gQuery);

}

public static AnalyzeTable load(InputStream is){

AnalyzeTable table = null;

try {

ObjectInputStream ois = new ObjectInputStream(is);

table = (AnalyzeTable) ois.readObject();

ois.close();

} catch(Exception e) {

throw new RuntimeException(e);

}

return table;

}

public void save(OutputStream os){

try {

ObjectOutputStream oos = new ObjectOutputStream(os);

oos.writeObject(this);

oos.close();

} catch(Exception e) {

throw new RuntimeException(e);

}

}

public void show() {

for(Entry<Group, Action> ex : analyzeTable.entrySet()){

System.out.println("from: " + ex.getKey().getFrom() +

" via: " + ex.getKey().getVia() + " Action: " + ex.getValue());

}

}

}

TableReader.java

package lexical;

import static dfaautomat.DFAAutomat.EPSLON;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.EmptyStackException;

import java.util.Map.Entry;

import java.util.Properties;

import java.util.Stack;

import dfaautomat.DFAAutomat;

import dfaautomat.DFAAutomat.NFAConstructor;

import dfaautomat.State;

/\*\*

\* 解析单词文件

\* 文件为正规式列表

\* @author lxm

\*

\*/

public class TableReader {

private DFAAutomat mat;

private Properties p; //读取可识别的单词列表

private int priority(char ch){ //定义单词的优先级

switch(ch) {

case ')':

return 1;

case '.':

return 2;

case '|':

return 3;

case '\*':

case '+':

return 4;

case '(':

return 5;

}

return -1;

}

/\*\*

\* 构造函数, 解析文件

\* @param is

\*/

public TableReader(InputStream is) {

p = new Properties();

try {

p.load(new InputStreamReader(is, "utf-8"));

} catch (IOException e) {

throw new RuntimeException(e);

}

Stack<Character> sop = new Stack<Character>(); // 操作符栈

Stack<SubMat> snum = new Stack<SubMat>(); // 操作数栈

NFAConstructor c = DFAAutomat.constructorN(); // 构造NFA

State s = new State();

c.begin(s);

for(Entry<Object, Object> ex : p.entrySet()) { // 解析文件中的每一条记录

String key = (String) ex.getKey(); // 得到正规式

int len = key.length();

for(int i = 0; i < len; i++) {

char chx = key.charAt(i); // 提取正规式的每一个符号

switch (chx) {

case '(':

case ')':

case '\*':

case '+':

case '|':

case '.':

while(!sop.isEmpty() && priority(sop.peek()) > priority(chx)){

// pop and calculate

if (!calc(sop, snum, c)) break;

}

if(chx != ')')

sop.push(chx);

else if (sop.pop() != '(') {

throw new RuntimeException("Too many right brackets.");

}

break;

case '\\':

chx = key.charAt(++i);

default:

SubMat tmp = new SubMat();

tmp.start = new State();

tmp.end = new State();

c.edge(tmp.start, chx, tmp.end);

snum.push(tmp);

break;

}

}

while(!sop.isEmpty()) {

try {

if (!calc(sop, snum, c))

throw new RuntimeException("Too maly left bracket at " + key);

}catch (EmptyStackException e) {

throw new RuntimeException("Lack of Operand at " + key);

}

}

if (snum.size() != 1) {

throw new RuntimeException("Lack of Operator at " + key);

}

SubMat sm = snum.pop();

c.edge(s, EPSLON, sm.start);

c.finals(sm.end);

String[] val = ((String) ex.getValue()).split(",");

sm.end.priority = Integer.parseInt(val[0].trim());

sm.end.type = val[1].trim();

}

mat = c.finish();

}

public DFAAutomat getMat() {

return mat;

}

/\*\*

\* 计算正规式,构造NFA

\* @param sop

\* @param snum

\* @param c

\* @return

\*/

private boolean calc(Stack<Character> sop, Stack<SubMat> snum, NFAConstructor c) {

char opx = sop.pop();

SubMat p1, p2, tmp;

State s1, s2;

switch (opx) {

case '(':

sop.push('(');

return false;

case '\*':

p1 = snum.pop();

s1 = new State();

s2 = new State();

c.edge(s1, EPSLON, s2);

c.edge(s2, EPSLON, s1);

c.edge(s1, EPSLON, p1.start);

c.edge(p1.end, EPSLON, s2);

tmp = new SubMat();

tmp.start = s1;

tmp.end = s2;

snum.push(tmp);

break;

case '+':

p1 = snum.pop();

s1 = new State();

s2 = new State();

c.edge(s2, EPSLON, s1);

c.edge(s1, EPSLON, p1.start);

c.edge(p1.end, EPSLON, s2);

tmp = new SubMat();

tmp.start = s1;

tmp.end = s2;

snum.push(tmp);

break;

case '|':

p1 = snum.pop();

p2 = snum.pop();

s1 = new State();

s2 = new State();

c.edge(s1, EPSLON, p1.start);

c.edge(s1, EPSLON, p2.start);

c.edge(p1.end, EPSLON, s2);

c.edge(p2.end, EPSLON, s2);

tmp = new SubMat();

tmp.start = s1;

tmp.end = s2;

snum.push(tmp);

break;

case '.':

p1 = snum.pop();

p2 = snum.pop();

c.edge(p2.end, EPSLON, p1.start);

tmp = new SubMat();

tmp.start = p2.start;

tmp.end = p1.end;

snum.push(tmp);

break;

default:

throw new RuntimeException("" + opx);

}

return true;

}

public static void main(String[] args) throws IOException{

TableReader tr = new TableReader(TableReader.class.getResourceAsStream("wordlist.properties"));

DFAAutomat df = tr.getMat();

//df.next("++");

//System.out.println(df.isFinal());

df.saveTo(new FileOutputStream("e:\\mat1.automat"));

}

}

/\*\*

\* SubMat类

\* 记录每一个子正规式产生的自动机

\* @author lxm

\*

\*/

class SubMat {

public State start, end;

@Override

public String toString() {

return "[" + start.hashCode() + ", " + end.hashCode() + "]";

}

}

LexicalAnalyzerImpl.java

package lexical;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.util.Arrays;

import java.util.HashSet;

import java.util.Set;

import dfaautomat.DFAAutomat;

import dfaautomat.State;

import syntax.V;

/\*\*

\* LexicalAnalyzer接口实现类

\* 将输入程序识别为单词符号串

\* @author lxm

\*

\*/

public class LexicalAnalyzerImpl implements LexicalAnalyzer {

private static final Set<Character> BLANKCH = new HashSet<Character>(Arrays.asList(' ','\t','\n','\r'));//空白字符集

private int line = 1, col = 1;

private int lastLine = 1, lastCol = 1;

private BufferedReader br; //从输入流读取字符的缓冲区

private int buf = -1; //缓冲区

private DFAAutomat dfa; //DFA自动机对象

private StringBuffer sb = new StringBuffer(); //保存未识别完的输入字符串

public int getLine() {

return lastLine;

}

public int getCol() {

return lastCol;

}

/\*\*

\* 获得DFA对象

\* @param is

\*/

public LexicalAnalyzerImpl(InputStream is) {

dfa = DFAAutomat.load(is);

}

/\*\*

\* 获得输入字符流

\*/

@Override

public void load(InputStream is) {

br = new BufferedReader(new InputStreamReader(is));

}

/\*\*

\* 返回一个单词符号

\*/

@Override

public V next() {

if(br == null){ throw new RuntimeException("please load file first !"); } // 读取文件失败

V v = new V();

//boolean isOver = false;

sb.setLength(0); // 清空字符串缓冲区

dfa.reset(); // 重置DFA

State curr = null; // 记录当前状态

State pre = null; // 记录前一状态

try {

// 找到非空字符

char r = getChar();

while(BLANKCH.contains(r)) {

if(r == '\n') {

line++;

col = 1;

}

r = getChar();

}

// enable circle

lastLine = line;

lastCol = col;

do { // 开始循环

pre = curr;

curr = dfa.next(r);

if(curr != null){

sb.append(r);

r = getChar();

}

} while(curr != null);

buf = r; // 保存多读的字符

if(dfa.isFinal(pre)){

v.name = pre.type;

v.attr("value", sb.toString());

} else {

throw new RuntimeException("Lexical error near: " + sb.toString() + r);

}

}catch (FileEndException e){ // 读到文件末尾

if (sb.length() > 0) {

if(dfa.isFinal(curr)){

v.name = curr.type;

v.attr("value", sb.toString());

} else {

throw new RuntimeException("Lexical error near: " + sb.toString());

}

} else

return null;

} catch (IOException e) {

e.printStackTrace();

}

return v;

}

/\*\*

\* 读取一个字符

\* @return ch

\* @throws IOException

\*/

private char getChar() throws IOException{

char ch = 0;

if(buf != -1){

ch = (char) buf;

buf = -1;

} else {

int read;

if((read = br.read()) > -1){

ch = (char) read;

if(ch != '\n' && ch != '\r') {

col++;

}

} else {

throw new FileEndException();

}

}

return ch;

}

public void close(){

try {

br.close();

} catch (IOException e) {

throw new RuntimeException(e);

}

}

}

class FileEndException extends RuntimeException {

private static final long serialVersionUID = -4700426986853540136L;

}

grammar.properties

S=语句组@main.syntax.action.GlobalAction.a1

语句组=语句@main.syntax.action.GlobalAction.a2|语句组,语句@main.syntax.action.GlobalAction.a3

语句=声明语句|执行语句@main.syntax.action.GlobalAction.a4

#声明语句

声明语句=全局变量定义语句|函数声明语句

函数声明语句=函数头,{,局部语句组或空@main.syntax.action.FuncAction.f1

函数头=类型和函数名,(,形式参数列表,)@main.syntax.action.FuncAction.f2

类型和函数名=类型,id@main.syntax.action.FuncAction.f17

形式参数列表=形式参数@main.syntax.action.FuncAction.f3|形式参数列表,com,形式参数@main.syntax.action.FuncAction.f4|\_

形式参数=类型,id@main.syntax.action.FuncAction.f5

局部语句组或空=局部语句组,}@main.syntax.action.FuncAction.f6|}@main.syntax.action.FuncAction.f22

局部语句组=局部语句@main.syntax.action.FuncAction.f7|局部语句组,局部语句@main.syntax.action.FuncAction.f8

局部语句=局部变量定义语句|执行语句@main.syntax.action.FuncAction.f9|return语句@main.syntax.action.FuncAction.f19

return语句=return,值,semi@main.syntax.action.FuncAction.f18

局部变量定义语句=局部声明部分,semi@main.syntax.action.FuncAction.f10

局部声明部分=类型,局部声明元@main.syntax.action.FuncAction.f11|局部声明部分,com,局部声明元@main.syntax.action.FuncAction.f12

局部声明元=id@main.syntax.action.FuncAction.f13|局部数组参数表,]@main.syntax.action.FuncAction.f14

局部数组参数表=id,[,number@main.syntax.action.FuncAction.f15|局部数组参数表,com,number@main.syntax.action.FuncAction.f16

#全局变量定义部分

#多条全局变量定义语句=全局变量定义语句|多条全局变量定义语句,全局变量定义语句

全局变量定义语句=全局声明部分,semi

全局声明部分=类型,全局声明元@main.syntax.action.Vstmt.a1|全局声明部分,com,全局声明元@main.syntax.action.Vstmt.a2

类型=int@main.syntax.action.Vstmt.a3|bool@main.syntax.action.Vstmt.a3|char@main.syntax.action.Vstmt.a3|real@main.syntax.action.Vstmt.a3

全局声明元=id@main.syntax.action.Vstmt.a4|全局数组参数表,]@main.syntax.action.Vstmt.a5

全局数组参数表=id,[,number@main.syntax.action.Vstmt.a6|全局数组参数表,com,number@main.syntax.action.Vstmt.a7

#执行语句部分

执行语句=控制语句@main.syntax.action.ControlAction.pass|求值语句

控制语句=if语句@main.syntax.action.ControlAction.pass|while语句@main.syntax.action.ControlAction.pass

if语句=没有else的if语句@main.syntax.action.ControlAction.c9|没有else的if语句,else子句@main.syntax.action.ControlAction.c7

没有else的if语句=if语句头,{,执行语句组或空@main.syntax.action.ControlAction.c6

if语句头=if,(,布尔表达式,)@main.syntax.action.ControlAction.c5

else子句=else字符串,{,执行语句组或空@main.syntax.action.ControlAction.c4

else字符串=else@main.syntax.action.ControlAction.c10

while语句=while语句头,{,执行语句组或空@main.syntax.action.ControlAction.c3

while语句头=while字符串,(,布尔表达式,)@main.syntax.action.ControlAction.c2

while字符串=while@main.syntax.action.ControlAction.c8

执行语句组或空=执行语句组,}@main.syntax.action.ControlAction.pass|}

执行语句组=执行语句@main.syntax.action.ControlAction.c11|执行语句组,执行语句@main.syntax.action.ControlAction.c1

布尔表达式=与或值@main.syntax.action.ExpAction.e1

与或值=与前缀,非值@main.syntax.action.ExpAction.e2|或前缀,非值@main.syntax.action.ExpAction.e3|非值@main.syntax.action.ExpAction.e4

与前缀=与或值,and@main.syntax.action.ExpAction.e11

或前缀=与或值,or@main.syntax.action.ExpAction.e12

非值=not,元布尔值@main.syntax.action.ExpAction.e5|元布尔值@main.syntax.action.ExpAction.e6

元布尔值=false@main.syntax.action.ExpAction.e7|true@main.syntax.action.ExpAction.e7|值,关系运算符,值@main.syntax.action.ExpAction.e8|(,布尔表达式,)@main.syntax.action.ExpAction.e9

关系运算符=<@main.syntax.action.ExpAction.e10|>@main.syntax.action.ExpAction.e10|<>@main.syntax.action.ExpAction.e10|\=\=@main.syntax.action.ExpAction.e10|>\=@main.syntax.action.ExpAction.e10|<\=@main.syntax.action.ExpAction.e10

求值语句=值,semi@main.syntax.action.ValueAction.a11

值=加值@main.syntax.action.ValueAction.pass|左值,ass,加值@main.syntax.action.ValueAction.a10

加值=乘积值@main.syntax.action.ValueAction.pass|加值,+,乘积值@main.syntax.action.ValueAction.a9|加值,-,乘积值@main.syntax.action.ValueAction.a21

乘积值=自增值@main.syntax.action.ValueAction.pass|乘积值,\*,自增值@main.syntax.action.ValueAction.a8

自增值=元值@main.syntax.action.ValueAction.pass|++,左值@main.syntax.action.ValueAction.a6|左值,++@main.syntax.action.ValueAction.a7

元值=number@main.syntax.action.ValueAction.a1|(,值,)@main.syntax.action.ValueAction.a2|函数调用@main.syntax.action.ValueAction.pass|左值@main.syntax.action.ValueAction.a5

左值=id@main.syntax.action.ValueAction.a12|数组访问参数表,]@main.syntax.action.ValueAction.a13

数组访问参数表=id,[,数组访问参数@main.syntax.action.ValueAction.a14|数组访问参数表,com,数组访问参数@main.syntax.action.ValueAction.a15

数组访问参数=值@main.syntax.action.ValueAction.pass

函数调用=id,(,实参列表,)@main.syntax.action.ValueAction.a16

实参列表=实参@main.syntax.action.ValueAction.a17|实参列表,com,实参@main.syntax.action.ValueAction.a18|\_

实参=值@main.syntax.action.ValueAction.a20

wordlist.properties

i.n.t=2,int

f.a.l.s.e=2,false

t.r.u.e=2,true

b.o.o.l=2,bool

r.e.a.l=2,real

c.h.a.r=2,char

w.h.i.l.e=2,while

i.f=2,if

e.l.s.e=2,else

r.e.t.u.r.n=2,return

\:.\==2,ass

<=2,<

>=2,>

>.\==2,>\=

<.\==2,<\=

\=.\==2,\=\=

<.>=2,<>

\\(=2,(

\\)=2,)

&.&=2,and

\\|.\\|=2,or

~.~=2,not

[=2,[

]=2,]

{=2,{

}=2,}

;=2,semi

,=2,com

\\+.\\+=2,++

-=2,-

\\+=2,+

\\\*=2,\*

(\_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z).(0|1|2|3|4|5|6|7|8|9|\_|a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z)\*=1,id

(0|1|2|3|4|5|6|7|8|9)+|((0|1|2|3|4|5|6|7|8|9)+.\\..(0|1|2|3|4|5|6|7|8|9)+)=1,number

Entry.java

package main;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.PrintWriter;

import java.lang.Thread.UncaughtExceptionHandler;

import language.GrammarAnalyser;

import lexical.LexicalAnalyzer;

import lexical.LexicalAnalyzerImpl;

import lexical.TableReader;

import syntax.Quad;

import syntax.SyntaxAnalyzer;

import syntax.SyntaxAnalyzerImpl;

public class Entry {

public static void main(String[] args) throws Exception{

boolean wordChanged = false;

boolean grammaChanged = true;

// 自动机和分析表的输入路径

String automatPath = "d:\\syntaxtest.automat", atablePath = "d:\\syntaxtest.atable";

if (wordChanged) {

TableReader ta = new TableReader(Entry.class.getResourceAsStream("wordlist.properties"));

ta.getMat().saveTo(new FileOutputStream(automatPath));

}

if (grammaChanged) {

GrammarAnalyser ga = new GrammarAnalyser(Entry.class.getResourceAsStream("grammar.properties"));

ga.getAnalyzeTable().save(new FileOutputStream(atablePath));

//System.out.println("BNF Grammar:");

//System.out.print(ga.getBNFGrammar());

}

// 开始分析

LexicalAnalyzer la;

// 加载自动机

la = new LexicalAnalyzerImpl(new FileInputStream(automatPath));

// 加载输入文件

la.load(Entry.class.getResourceAsStream("input6.txt"));

// 提示出错位置等信息

Thread.setDefaultUncaughtExceptionHandler(new UncaughtExceptionHandler() {

@Override

public void uncaughtException(Thread arg0, Throwable e) {

Throwable t = e;

while(t.getCause() != null) t = t.getCause();

if (t instanceof RuntimeException) {

System.err.println("Error at line " + la.getLine() + ", col " + (la.getCol() - 1));

System.err.println(t.getMessage());

}

t.printStackTrace();

}

});

SyntaxAnalyzer sa = new SyntaxAnalyzerImpl(new FileInputStream(atablePath));

sa.setInput(la);

sa.analyse();

File output = new File("d:\\output.txt");

PrintWriter pw = new PrintWriter(output);

System.out.println("\nQuad list:");

for(Quad qx : sa.getQuad()) {

System.out.println(qx.toString());

pw.println(qx.toString());

}

System.out.println("\nSymbol Table:");

pw.println(sa.getSymbolTable().toString());

pw.close();

System.out.println(sa.getSymbolTable());

}

}

ControlAction.java

package main.syntax.action;

import syntax.Quad;

import syntax.V;

import syntax.action.SemanticAction;

/\*\*

\* 控制语句的语义动作

\* @author lxm

\*

\*/

public class ControlAction extends SemanticAction {

// 执行语句=控制语句@test.syntax.action.ControlAction.pass

// 控制语句=if语句@test.syntax.action.ControlAction.pass

// 控制语句=while语句@test.syntax.action.ControlAction.pass

public void pass(V left, V[] right) {

left.attr("chain", right[0].attr("chain"));

}

// 执行语句组 = 执行语句组,执行语句@test.syntax.action.ControlAction.c1

public void c1(V left, V[] right){

backPatch((Integer)right[0].attr("chain"), (Integer)right[0].attr("nextq"));

left.attr("nextq", nextQuad());

left.attr("chain", right[1].attr("chain"));

}

// 执行语句组=执行语句@test.syntax.action.ControlAction.c11

public void c11(V left, V[] right){

left.attr("nextq", nextQuad());

left.attr("chain", right[0].attr("chain"));

}

// while语句头=while字符串,(,布尔表达式,)@test.syntax.action.ControlAction.c2

public void c2(V left, V[] right){

Integer tc = (Integer)right[2].attr("TC"), fc = (Integer)right[2].attr("FC");

backPatch(tc, nextQuad());

left.attr("chain", fc);

left.attr("quad", right[0].attr("quad"));

}

// while语句=while语句头,{,执行语句组或空@test.syntax.action.ControlAction.c3

public void c3(V left, V[] right){

left.attr("chain", right[0].attr("chain"));

backPatch((Integer)right[2].attr("chain"), (Integer)right[0].attr("quad"));

newQuad().field("j", null, null, right[0].attr("quad"));

}

// else子句=else字符串,{,执行语句组或空@test.syntax.action.ControlAction.c4

public void c4(V left, V[] right){

left.attr("elseq", right[0].attr("elseq"));

left.attr("elsechain", right[0].attr("elsechain"));

left.attr("chain", right[2].attr("chain"));

}

// if语句头=if,(,布尔表达式,)@test.syntax.action.ControlAction.c5

public void c5(V left, V[] right){

Integer tc = (Integer)right[2].attr("TC"), fc = (Integer)right[2].attr("FC");

backPatch(tc, nextQuad());

left.attr("chain", fc);

}

// 没有else的if语句=if语句头,{,执行语句组或空@test.syntax.action.ControlAction.c6

public void c6(V left, V[] right){

left.attr("chain", right[0].attr("chain"));

left.attr("chain2", right[2].attr("chain"));

}

// if语句=没有else的if语句,else子句@test.syntax.action.ControlAction.c7

public void c7(V left, V[] right){

left.attr("chain", merge((Integer)right[1].attr("elsechain"), (Integer)right[0].attr("chain2")));

backPatch((Integer)right[0].attr("chain"), (Integer)right[1].attr("elseq"));

}

// while字符串=while@test.syntax.action.ControlAction.c8

public void c8(V left, V[] right){

left.attr("quad", nextQuad());

}

// if语句=没有else的if语句@test.syntax.action.ControlAction.c9

public void c9(V left, V[] right){

left.attr("chain", merge((Integer)right[0].attr("chain"), (Integer)right[0].attr("chain2")));

}

// else字符串=else@test.syntax.action.ControlAction.c10

public void c10(V left, V[] right){

left.attr("elsechain", nextQuad());

newQuad().field("j");

left.attr("elseq", nextQuad());

}

/\*\*

\* 合并

\* @param head1

\* @param head2

\* @return

\*/

private Integer merge(Integer head1, Integer head2){

Integer newhead = 0;

if (head1 == null){

newhead = head2;

} else if (head2 == null){

newhead = head1;

} else {

Quad qx = getQuad(head2);

while(qx.field[3] != null){

qx = getQuad((Integer)qx.field[3]);

}

qx.field[3] = head1;

newhead = head2;

}

return newhead;

}

/\*\*

\* 回填

\* @param head

\* @param nextQuad

\*/

private void backPatch(Integer head, int nextQuad) {

while(head != null && head >= Quad.STARTNUM){

Quad q = getQuad(head);

head = (Integer)q.field[3];

q.field[3] = nextQuad;

}

}

}

ExpAction.java

package main.syntax.action;

import syntax.Quad;

import syntax.Symbol;

import syntax.V;

import syntax.action.SemanticAction;

/\*\*

\* 表达式的语义动作

\* @author lxm

\*

\*/

public class ExpAction extends SemanticAction{

// 布尔表达式=与或值@test.syntax.action.ExpAction.e1

public void e1(V left, V[] right){

left.attr("TC", right[0].attr("TC"));

left.attr("FC", right[0].attr("FC"));

}

// 与或值=与前缀,非值@test.syntax.action.ExpAction.e2

public void e2(V left, V[] right){

left.attr("FC", merge((Integer)right[0].attr("FC"), (Integer)right[1].attr("FC")));

left.attr("TC", right[1].attr("TC"));

}

// 与或值 =或前缀,非值@test.syntax.action.ExpAction.e3

public void e3(V left, V[] right){

left.attr("TC", merge((Integer)right[0].attr("TC"), (Integer)right[1].attr("TC")));

left.attr("FC", right[1].attr("FC"));

}

// 与前缀=与或值,and@test.syntax.action.ExpAction.e11

public void e11(V left, V[] right){

left.attr("FC", right[0].attr("FC"));

backPatch((Integer)right[0].attr("TC"), nextQuad());

}

// 或前缀=与或值,or@test.syntax.action.ExpAction.e12

public void e12(V left, V[] right){

left.attr("TC", right[0].attr("TC"));

backPatch((Integer)right[0].attr("FC"), nextQuad());

}

private Integer merge(Integer head1, Integer head2){

Integer newhead = 0;

if (head1 == null){

newhead = head2;

} else if (head2 == null){

newhead = head1;

} else {

Quad qx = getQuad(head2);

while(qx.field[3] != null){

qx = getQuad((Integer)qx.field[3]);

}

qx.field[3] = head1;

newhead = head2;

}

return newhead;

}

// 与或值=非值@test.syntax.action.ExpAction.e4

public void e4(V left, V[] right){

left.attr("TC", right[0].attr("TC"));

left.attr("FC", right[0].attr("FC"));

}

// 非值=not,元布尔值@test.syntax.action.ExpAction.e5

public void e5(V left, V[] right){

left.attr("TC", right[1].attr("FC"));

left.attr("FC", right[1].attr("TC"));

}

// 非值=元布尔值@test.syntax.action.ExpAction.e6

public void e6(V left, V[] right){

left.attr("TC", right[0].attr("TC"));

left.attr("FC", right[0].attr("FC"));

}

// 元布尔值=false@test.syntax.action.ExpAction.e7|true@test.syntax.action.ExpAction.e7

public void e7(V left, V[] right){

Quad q = newQuad();

q.field("j");

left.attr("true".equals(right[0].attr("value"))?"TC":"FC", q.num);

}

// 元布尔值=值,关系运算符,值@test.syntax.action.ExpAction.e8

public void e8(V left, V[] right){

Object o1 = right[0].attr("value"), o2 = right[2].attr("value");

Quad qx = newQuad();

qx.field("j" + right[1].attr("value"),o1, o2);

left.attr("TC", qx.num);

qx = newQuad();

qx.field("j");

left.attr("FC", qx.num);

}

// 元布尔值=(,布尔表达式,)@test.syntax.action.ExpAction.e9

public void e9(V left, V[] right){

left.attr("TC", right[1].attr("TC"));

left.attr("FC", right[1].attr("FC"));

}

// 关系运算符=<@test.syntax.action.ExpAction.e10

// 关系运算符=>@test.syntax.action.ExpAction.e10

// 关系运算符=<>@test.syntax.action.ExpAction.e10

// 关系运算符=\=\=@test.syntax.action.ExpAction.e10

// 关系运算符=>\=@test.syntax.action.ExpAction.e10

// 关系运算符=<\=@test.syntax.action.ExpAction.e10

public void e10(V left, V[] right){

left.attr("value", right[0].attr("value"));

}

/\*\*

\* 回填

\* @param head

\* @param nextQuad

\*/

// 元布尔值=id@test.syntax.action.ExpAction.e11

public void e13(V left, V[] right){

String name = (String) right[0].attr("value");

Symbol s = st.lookup(name);

if(s == null){

throw new RuntimeException("Symbol undefined: " + name);

} else {

if(s.attr("type").equals(ValueAction.BOOL)){

Quad q = newQuad();

q.field("jnz",s);

left.attr("TC", q.num);

q = newQuad();

q.field("j");

left.attr("FC", q.num);

} else {

throw new RuntimeException("Symbol is not a boolean: " + name);

}

}

}

private void backPatch(Integer head, int nextQuad) {

while(head != null && head >= Quad.STARTNUM){

Quad q = getQuad(head);

head = (Integer)q.field[3];

q.field[3] = nextQuad;

}

}}

FuncAction.java

package main.syntax.action;

import java.util.ArrayList;

import java.util.List;

import syntax.Quad;

import syntax.Symbol;

import syntax.V;

import syntax.action.SemanticAction;

/\*\*

\* 函数定义语句的语义动作

\* @author lxm

\*

\*/

public class FuncAction extends SemanticAction {

// 函数声明语句=函数头,{,局部语句组或空@test.syntax.action.FuncAction.f1

public void f1(V left, V[] right) {

Symbol s = (Symbol) right[0].attr("func");

s.attr("localVar", st.exitFunc());

backPatch((Integer)right[0].attr("skip"), nextQuad());

}

// 函数头=类型和函数名,(,形式参数列表,)@test.syntax.action.FuncAction.f2

public void f2(V left, V[] right) {

Symbol s = (Symbol) right[0].attr("func");

s.attr("parlist", right[2].attr("parlist"));

left.attr("func", s);

left.attr("skip", nextQuad());

newQuad().field("j");

s.attr("addr", nextQuad());

}

// 类型和函数名=类型,id@test.syntax.action.FuncAction.f17

public void f17(V left, V[] right) {

String funcName = (String) right[1].attr("value");

if (st.lookupFunc(funcName) != null) {

throw new RuntimeException("Duplicate function declaration: " + funcName);

}

Symbol s = st.addFunc(funcName);

s.attr("type", right[0].attr("type"));

st.enterFunc();

left.attr("func", s);

}

// 形式参数列表=形式参数@test.syntax.action.FuncAction.f3

public void f3(V left, V[] right) {

List<Symbol> parlist = new ArrayList<>();

Symbol s = st.add((String)right[0].attr("name"));

s.attr("type", right[0].attr("type"));

parlist.add(s);

left.attr("parlist", parlist);

}

// 形式参数列表=形式参数列表,com,形式参数@test.syntax.action.FuncAction.f4

public void f4(V left, V[] right) {

@SuppressWarnings("unchecked")

List<Symbol> parlist = (List<Symbol>) right[0].attr("parlist");

String name = (String) right[2].attr("name");

if(st.lookupLocal(name) != null) {

throw new RuntimeException("Duplicated local variable: " + name);

}

Symbol s = st.add(name);

s.attr("type", right[2].attr("type"));

parlist.add(s);

left.attr("parlist", parlist);

}

// 形式参数=类型,id@test.syntax.action.FuncAction.f5

public void f5(V left, V[] right) {

left.attr("type", right[0].attr("type"));

left.attr("name", right[1].attr("value"));

}

// 局部语句组或空=局部语句组,}@test.syntax.action.FuncAction.f6

public void f6(V left, V[] right) {

if((String)right[0].attr("ret") == null){

throw new RuntimeException("Need a return statement...");

}

if(right[0].attr("nextq") != null)

backPatch((Integer)right[0].attr("chain"), (Integer)right[0].attr("nextq"));

}

// 局部语句组或空=}@test.syntax.action.FuncAction.f22

public void f22(V left, V[] right){

throw new RuntimeException("Need a return statement...");

}

// 局部语句组=局部语句@test.syntax.action.FuncAction.f7

public void f7(V left, V[] right) {

left.attr("chain", right[0].attr("chain"));

left.attr("nextq", right[0].attr("nextq"));

String ret = (String) right[0].attr("ret");

if(ret !=null){

left.attr("ret", ret);

} else {

throw new RuntimeException("Need a return statement...");

}

}

// 局部语句组=局部语句组,局部语句@test.syntax.action.FuncAction.f8

public void f8(V left, V[] right) {

if(right[1].attr("exe") != null) {

backPatch((Integer)right[0].attr("chain"), (Integer)right[0].attr("nextq"));

left.attr("chain", right[1].attr("chain"));

} else {

left.attr("chain", right[0].attr("chain"));

}

String ret1 = (String) right[1].attr("ret");

String ret0 = (String) right[0].attr("ret");

if(ret0 != null || ret1 != null ){

left.attr("ret", ret0 != null? ret0 : ret1);

}

//left.attr("ret", ret0 != null? ret0: ret1);

left.attr("nextq", nextQuad());

}

// 局部语句=return语句@test.syntax.action.FuncAction.f19

public void f19(V left, V[] right) {

left.attr("exe", 1);

left.attr("chain", right[0].attr("chain"));

left.attr("nextq", nextQuad());

left.attr("ret", right[0].attr("ret"));

}

// 局部语句=执行语句@test.syntax.action.FuncAction.f9

public void f9(V left, V[] right) {

left.attr("exe", 1);

left.attr("chain", right[0].attr("chain"));

left.attr("nextq", nextQuad());

}

// 局部变量定义语句=局部声明部分,semi@test.syntax.action.FuncAction.f10

public void f10(V left, V[] right) {

}

// 局部声明部分=类型,局部声明元@test.syntax.action.FuncAction.f11|

public void f11(V left, V[] right) {

String type = (String) right[0].attr("type");

String name = (String) right[1].attr("name");

if(st.lookupLocal(name) != null){

throw new RuntimeException("Duplicate local variable: " + name);

}

Symbol s = st.add(name);

if(right[1].attr("dim") != null){

s.attr("dim", right[1].attr("dim"));

}

s.attr("type", type);

left.attr("type", type);

}

// 局部声明部分=局部声明部分,com,局部声明元@test.syntax.action.FuncAction.f12

public void f12(V left, V[] right) {

String type = (String) right[0].attr("type");

String name = (String) right[2].attr("name");

if(st.lookupLocal(name) != null) {

throw new RuntimeException("Dupilicate local variable: " + name);

}

Symbol s = st.add(name);

if(right[2].attr("dim") != null)

s.attr("dim", right[2].attr("dim"));

s.attr("type", type);

left.attr("type", type);

}

// 局部声明元=id@test.syntax.action.FuncAction.f13

public void f13(V left, V[] right) {

left.attr("name", right[0].attr("value"));

}

// 局部声明元=局部数组参数表,]@test.syntax.action.FuncAction.f14

public void f14(V left, V[] right) {

left.attr("name", right[0].attr("name"));

left.attr("dim", right[0].attr("dim"));

}

// 局部数组参数表=id,[,number@test.syntax.action.FuncAction.f15

public void f15(V left, V[] right) {

left.attr("name",right[0].attr("value"));

List<Integer> dim = new ArrayList<>();

String val = (String) right[2].attr("value");

if(val.indexOf(".") != -1) {

throw new RuntimeException("Array Dimension can only be integer: " + val);

}

dim.add(Integer.parseInt(val));

left.attr("dim", dim);

}

// 局部数组参数表=局部数组参数表,com,number@test.syntax.action.FuncAction.f16

public void f16(V left, V[] right) {

@SuppressWarnings("unchecked")

List<Integer> dim = (List<Integer>) right[0].attr("dim");

String val = (String) right[2].attr("value");

if(val.indexOf(".") != -1) {

throw new RuntimeException("Array Dimension can only be integer: " + val);

}

dim.add(Integer.parseInt(val));

left.attr("name", right[0].attr("name"));

left.attr("dim", dim);

}

// return语句=return,值,semi@test.syntax.action.FuncAction.f18

public void f18(V left, V[] right) {

left.attr("ret", right[0].attr("value"));

Object val = right[1].attr("value");

newQuad().field("ret",null, null, val);

st.releaseTemp(val);

}

/\*\*

\* 回填

\* @param head

\* @param nextQuad

\*/

private void backPatch(Integer head, int nextQuad) {

while(head != null && head >= Quad.STARTNUM){

Quad q = getQuad(head);

head = (Integer)q.field[3];

q.field[3] = nextQuad;

}

}

}

GlobalAction.java

package main.syntax.action;

import syntax.Quad;

import syntax.V;

import syntax.action.SemanticAction;

/\*\*

\* 定义全局语句执行的语义动作

\* @author lxm

\*

\*/

public class GlobalAction extends SemanticAction {

// S=语句组@test.syntax.action.GlobalAction.a1

public void a1(V left, V[] right) {

backPatch((Integer)right[0].attr("chain"), nextQuad());

newQuad().field("end");

}

// 语句组=语句@test.syntax.action.GlobalAction.a2

public void a2(V left, V[] right) {

left.attr("nextq", nextQuad());

left.attr("chain", right[0].attr("chain"));

}

// 语句组=语句组,语句@test.syntax.action.GlobalAction.a3

public void a3(V left, V[] right) {

if(right[1].attr("exe") != null) {

backPatch((Integer)right[0].attr("chain"), (Integer)right[0].attr("nextq"));

left.attr("chain", right[1].attr("chain"));

} else {

left.attr("chain", right[0].attr("chain"));

}

left.attr("nextq", nextQuad());

}

// 语句=执行语句@test.syntax.action.GlobalAction.a4

public void a4(V left, V[] right) {

left.attr("chain", right[0].attr("chain"));

left.attr("exe", 1);

}

/\*\*

\* 回填

\* @param head

\* @param nextQuad

\*/

private void backPatch(Integer head, int nextQuad) {

while(head != null && head >= Quad.STARTNUM){

Quad q = getQuad(head);

head = (Integer)q.field[3];

q.field[3] = nextQuad;

}

}

}

ValueAction.java

package main.syntax.action;

import java.util.ArrayList;

import java.util.List;

import syntax.Quad;

import syntax.Symbol;

import syntax.V;

import syntax.action.SemanticAction;

/\*\*

\* 值的属性为: value - Integer, Double或Symbol, 代表当前符号的值<br>

\* 如果是数组左值，则附带属性为offset<br>

\* type - String,可能为{@link main.syntax.action.ValueAction}.INT, REAL, CHAR, BOOL

\* @author lxm

\*/

public class ValueAction extends SemanticAction{

public static final String INT = "int", REAL = "real", CHAR = "char", BOOL="bool"; // 四种类型

private static final String[] PREC = new String[]{BOOL, CHAR, INT, REAL}; // 类型数组

// 元值=number@test.syntax.action.ValueAction.a1

public void a1(V left, V[] right){

String val = (String) right[0].attr("value");

if (val.indexOf('.') == -1) {

// int

left.attr("value",Integer.parseInt(val));

left.attr("type", INT);

} else{

// real

left.attr("value", Double.parseDouble(val));

left.attr("type", REAL);

}

}

// 元值=(,值,)@test.syntax.action.ValueAction.a2

public void a2(V left, V[] right){

left.attr("type", right[1].attr("type"));

left.attr("value", right[1].attr("value"));

}

// 值=加值@test.syntax.action.ValueAction.pass

// 加值=乘积值@test.syntax.action.ValueAction.pass

// 乘积值=自增值@test.syntax.action.ValueAction.pass

// 自增值=元值@test.syntax.action.ValueAction.pass

// 数组访问参数=值@test.syntax.action.ValueAction.pass

// 元值=函数调用@test.syntax.action.ValueAction.pass

public void pass(V left, V[] right){

left.attr("value", right[0].attr("value"));

left.attr("type", right[0].attr("type"));

}

// 元值 = 左值@test.syntax.action.ValueAction.a5

public void a5(V left, V[] right){

Symbol s = (Symbol)right[0].attr("value");

Object offset = right[0].attr("offset");

if (offset != null) {

Symbol tmp = st.getTemp();

newQuad().field("=[]", s.name + "[" + offset + "]", null, tmp);

st.releaseTemp(offset);

left.attr("value", tmp);

} else {

left.attr("value", s);

}

left.attr("type", s.attr("type"));

}

// 自增值=++,左值@test.syntax.action.ValueAction.a6

public void a6(V left, V[] right){

Symbol sx = (Symbol)right[1].attr("value");

Object offset = right[1].attr("offset");

if(offset == null) {

Symbol temp = st.getTemp();

Quad qx = newQuad();

qx.field("+", sx, 1, temp);

qx = newQuad();

qx.field(":=", temp, null, sx);

st.releaseTemp(temp);

left.attr("type", sx.attr("type"));

left.attr("value", sx);

} else {

Symbol tmp1 = st.getTemp(), tmp2 = st.getTemp();

newQuad().field("=[]", sx.name + "[" + offset + "]", null, tmp1);

newQuad().field("+", tmp1, 1, tmp2);

newQuad().field("[]=", tmp2, null, sx.name + "[" + offset + "]");

st.releaseTemp(tmp1);

left.attr("type", sx.attr("type"));

left.attr("value", tmp2);

}

}

// 自增值=左值,++@test.syntax.action.ValueAction.a7

public void a7(V left, V[] right){

Symbol sx = (Symbol)right[0].attr("value");

Object offset = right[1].attr("offset");

if (offset == null) {

Symbol temp = st.getTemp(), temp2 = st.getTemp();

Quad qx = newQuad();

qx.field(":=", sx, null, temp2);

qx = newQuad();

qx.field("+", sx, 1, temp);

qx = newQuad();

qx.field(":=", temp, null, sx);

st.releaseTemp(temp);

left.attr("type", sx.attr("type"));

left.attr("value", temp2);

} else {

Symbol tmp1 = st.getTemp(), tmp2 = st.getTemp();

newQuad().field("=[]", sx.name + "[" + offset + "]", null, tmp1);

newQuad().field("+", tmp1, 1, tmp2);

newQuad().field("[]=", tmp2, null, sx.name + "[" + offset + "]");

st.releaseTemp(tmp2);

left.attr("type", sx.attr("type"));

left.attr("value", tmp1);

}

}

// 乘积值 = 乘积值,\*,自增值@test.syntax.action.ValueAction.a8

public void a8(V left, V[] right){

Object p1 = right[0].attr("value"), p2 = right[2].attr("value");

Symbol temp = st.getTemp();

Quad qx = newQuad();

qx.field("\*", p1, p2, temp);

st.releaseTemp(p1);

left.attr("value", temp);

left.attr("type", getResultType((String)right[0].attr("type"), (String)right[2].attr("type")));

}

// 加值 = 加值,+,乘积值@test.syntax.action.ValueAction.a9

public void a9(V left, V[] right){

Object p1 = right[0].attr("value"), p2 = right[2].attr("value");

Symbol temp = st.getTemp();

Quad qx = newQuad();

qx.field("+", p1, p2, temp);

st.releaseTemp(p1);

st.releaseTemp(p2);

left.attr("value", temp);

left.attr("type", getResultType((String)right[0].attr("type"), (String)right[2].attr("type")));

}

public void a21(V left, V[] right){

Object p1 = right[0].attr("value"), p2 = right[2].attr("value");

Symbol temp = st.getTemp();

Quad qx = newQuad();

qx.field("-", p1, p2, temp);

st.releaseTemp((Symbol)p1);

st.releaseTemp(p2);

left.attr("value", temp);

left.attr("type", getResultType((String)right[0].attr("type"), (String)right[2].attr("type")));

}

// 值=左值,ass,加值@test.syntax.action.ValueAction.a10

public void a10(V left, V[] right){

Quad qx = newQuad();

Object val = right[2].attr("value");

Symbol leftv = (Symbol) right[0].attr("value");

Object offset = right[0].attr("offset");

qx.field(offset == null? ":=": "[]=", val, null, offset == null?leftv.name : leftv.name + "[" + offset + "]");

st.releaseTemp(offset);

left.attr("value", val);

left.attr("type", ((Symbol)leftv).attr("type"));

}

// 求值语句=值,semi@test.syntax.action.ValueAction.a11

public void a11(V left, V[] right){

Object val = right[0].attr("value");

st.releaseTemp(val);

}

// 左值 = id@test.syntax.action.ValueAction.a12

public void a12(V left, V[] right){

Symbol sx = st.lookup((String) right[0].attr("value"));

if (sx == null) {

throw new RuntimeException("Symbol Undefined:" + (String) right[0].attr("value"));

}

left.attr("type", sx.attr("type"));

left.attr("value", sx);

}

// 左值 = 数组访问参数表,]@test.syntax.action.ValueAction.a13

public void a13(V left, V[] right){

Symbol s = (Symbol)right[0].attr("value");

int currdim =(Integer)right[0].attr("currdim");

if(currdim != ((List<?>)s.attr("dim")).size()) {

throw new RuntimeException("Array dimension mismatch with " + s.name + ": " + currdim);

}

left.attr("value", s);

left.attr("offset", right[0].attr("offset"));

left.attr("type", s.attr("type"));

}

// 数组访问参数表 = id,[,数组访问参数@test.syntax.action.ValueAction.a14

public void a14(V left, V[] right){

Symbol sx = st.lookup((String) right[0].attr("value"));

if (sx == null) {

throw new RuntimeException("Symbol Undefined: " + (String) right[0].attr("value"));

}

if (sx.attr("dim") == null){

throw new RuntimeException("Symbol is not an array: " + (String) right[0].attr("value"));

}

left.attr("value", sx);

left.attr("currdim", 1);

String type = (String) right[2].attr("type");

if (type.equals(REAL)) throw new RuntimeException("Array dimension cannot be a real number:" + right[2].attr("value"));

left.attr("offset", right[2].attr("value"));

}

// 数组访问参数表 = 数组访问参数表,com,数组访问参数@test.syntax.action.ValueAction.a15

public void a15(V left, V[] right){

int currdim = (Integer)right[0].attr("currdim") + 1;

Object offset = right[0].attr("offset"), dimx = right[2].attr("value");

Symbol s = (Symbol) right[0].attr("value");

@SuppressWarnings("unchecked")

List<Integer> dim = ((List<Integer>)s.attr("dim"));

if(dim.size() < currdim) {

throw new RuntimeException("Array Dimension out of range at " + s + ": " + currdim);

}

if (REAL.equals(right[2].attr("type"))) throw new RuntimeException("Array dimension cannot be a real number:" + dimx);

Symbol tmp = st.getTemp(), tmp2 = st.getTemp();

newQuad().field("\*", dim.get(currdim - 1), offset, tmp2);

newQuad().field("+", tmp2, dimx, tmp);

st.releaseTemp(dimx);

st.releaseTemp(tmp2);

st.releaseTemp(offset);

left.attr("offset", tmp);

left.attr("currdim", currdim);

left.attr("value", s);

left.attr("type", s.attr("type"));

}

// 函数调用=id,(,实参列表,)@test.syntax.action.ValueAction.a16

public void a16(V left, V[] right){

String name = (String)right[0].attr("value");

Symbol func = st.lookupFunc(name);

if(func == null) {

throw new RuntimeException("Undefined function: " + name);

}

@SuppressWarnings("unchecked")

List<Object> paras = (List<Object>) right[2].attr("paras");

if(paras != null){

for(Object ox : paras){

Quad qx = newQuad();

qx.field("par", null, null, ox);

st.releaseTemp(ox);

}

}

newQuad().field("call", null, null, func + "@" + func.attr("addr"));

left.attr("func", func);

left.attr("type", func.attr("type"));

Symbol temp = st.getTemp();

newQuad().field("getRetValue", null, null, temp);

left.attr("value", temp);

}

// 实参列表=实参@test.syntax.action.ValueAction.a17

public void a17(V left, V[] right){

List<Object> paras = new ArrayList<>();

paras.add(right[0].attr("value"));

left.attr("paras", paras);

}

// 实参列表=实参列表,com,实参@test.syntax.action.ValueAction.a18

public void a18(V left, V[] right) {

@SuppressWarnings("unchecked")

List<Object> paras = (List<Object>)right[0].attr("paras");

if(paras == null) {

throw new RuntimeException("Lack of Parameter before ','.");

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*没有记录实参类型\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

paras.add(right[2].attr("value"));

left.attr("paras", paras);

}

// 实参=值@test.syntax.action.ValueAction.a20

public void a20(V left, V[] right){

left.attr("value", right[0].attr("value"));

left.attr("type", right[0].attr("type"));

}

public String getResultType(String t1, String t2){

int p1 = -1, p2 = -1;

for(int i = 0; i < PREC.length; i++) {

if (PREC[i].equals(t1)) {

p1 = i;

}

if (PREC[i].equals(t2)) {

p2 = i;

}

}

if (p1 == -1 || p2 == -1) throw new RuntimeException("Type error:" + t1 + ", " + t2);

return p1 > p2? PREC[p1] : PREC[p2];

}

}

Vstmt.java

package main.syntax.action;

import java.util.ArrayList;

import java.util.List;

import syntax.Symbol;

import syntax.V;

import syntax.action.SemanticAction;

/\*\*

\* 声明语句部分

\* @author lxm

\*

\*/

public class Vstmt extends SemanticAction {

// 声明语句=类型,声明元@test.syntax.action.Vstmt.a1

public void a1(V left, V[] right){

String type = (String) right[0].attr("type");

left.attr("type", type);

Symbol s = (Symbol) right[1].attr("symbol");

s.attr("type", type);

}

// 声明语句=声明语句,com,声明元@test.syntax.action.Vstmt.a2

public void a2(V left, V[] right){

String type = (String) right[0].attr("type");

left.attr("type", type);

Symbol s = (Symbol) right[2].attr("symbol");

s.attr("type", type);

}

// 类型=int@test.syntax.action.Vstmt.a3

// 类型=bool@test.syntax.action.Vstmt.a3

// 类型=char@test.syntax.action.Vstmt.a3

// 类型=real@test.syntax.action.Vstmt.a3

public void a3(V left, V[] right){

left.attr("type", right[0].attr("value"));

}

// 声明元=id@test.syntax.action.Vstmt.a4

public void a4(V left, V[] right){

String name = (String) right[0].attr("value");

if (st.lookup(name) != null) {

throw new RuntimeException("Duplicate variable: " + name);

}

Symbol s = st.add(name);

left.attr("symbol", s);

}

// 声明元=声明元数组参数表,]@test.syntax.action.Vstmt.a5

public void a5(V left, V[] right){

Symbol s = (Symbol)right[0].attr("symbol");

left.attr("symbol", s);

}

// 数组参数表=id,[,值@test.syntax.action.Vstmt.a6

public void a6(V left, V[] right){

String name = (String) right[0].attr("value");

if (st.lookup(name) != null) {

throw new RuntimeException("Duplicate variable: " + name);

}

Symbol s = st.add(name);

left.attr("symbol", s);

List<Integer> dim;

s.attr("dim", dim = new ArrayList<Integer>());

String x = (String)right[2].attr("value");

if (x.indexOf('.')!= -1) {

throw new RuntimeException("Array dimension can only be integer: " + x);

}

dim.add(Integer.parseInt(x));

}

// 数组参数表=数组参数表,com,值@test.syntax.action.Vstmt.a7

@SuppressWarnings("unchecked")

public void a7(V left, V[] right){

List<Integer> dim = (List<Integer>)(((Symbol)(right[0].attr("symbol"))).attr("dim"));

String x = (String)right[2].attr("value");

if (x.indexOf('.')!= -1) {

throw new RuntimeException("Array dimension can only be integer: " + x);

}

dim.add(Integer.parseInt(x));

left.attr("symbol", right[0].attr("symbol"));

}

}

SemanticAction.java

package syntax.action;

import java.util.List;

import syntax.Quad;

import syntax.SymbolTable;

/\*\*

\* 所有语义动作类的父类

\* public void actionName(V left, V[] right) {

\*

\* }

\*/

public class SemanticAction {

protected SymbolTable st; // 符号表

private List<Quad> qlist; // 四元式队列

protected final Quad newQuad() {

Quad q = new Quad();

q.num = Quad.STARTNUM + qlist.size();

qlist.add(q);

return q;

}

public final void setSt(SymbolTable st) {

this.st = st;

}

public final void setQlist(List<Quad> qlist) {

this.qlist = qlist;

}

/\*\*

\* 根据四元式编号返回四元式

\* @param num

\* @return

\*/

public final Quad getQuad(int num){

return qlist.get(num - Quad.STARTNUM);

}

/\*\*

\* 获得下一条四元式的编号

\* @return

\*/

public final int nextQuad(){

return qlist.size() + Quad.STARTNUM;

}

}

SymbolTable.java

package syntax;

import java.util.HashMap;

import java.util.HashSet;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

/\*\*

\* 符号表类,存放符号的信息

\* @author lxm

\*

\*/

public class SymbolTable {

private final Map<String, Symbol> syms = new HashMap<>(), funcs = new HashMap<>(); // 符号的集合

private Map<String, Symbol> localSyms = null; // 局部符号

private Set<Symbol> temp = new HashSet<>(), pool = new HashSet<>(); // 临时变量池

private static int startAddr = 10000; // 符号表起始地址

private static final int INTSIZE = 4, CHARSIZE = 2, REALZISE = 8, BOOLSIZE = 1; // 定义不同类型的分配空间

/\*\*

\* 在全局符号表和当前局部符号表中查看符号是否存在</br>

\* 并返回符号信息

\* @param name

\* @return

\*/

public Symbol lookup(String name){

if(localSyms != null) {

Symbol sx = localSyms.get(name);

if (sx != null) return sx;

}

return syms.get(name);

}

/\*\*

\* 仅在当前局部符号表中查看符号是否存在</br>

\* 返回符号的信息

\* @param name

\* @return

\*/

public Symbol lookupLocal(String name) {

if (localSyms != null) {

return localSyms.get(name);

}

return null;

}

/\*\*

\* 增加一个符号

\* @param name

\* @return

\*/

public Symbol add(String name) {

Symbol sx = new Symbol();

sx.name = name;

if (localSyms == null) {

syms.put(name, sx);

} else {

localSyms.put(name, sx);

}

return sx;

}

/\*\*

\* 获取一个临时变量

\* @return

\*/

public Symbol getTemp(){

Symbol sx;

if (pool.isEmpty()) {

sx = new TempSymbol();

sx.name = "T" + temp.size();

} else {

Iterator<Symbol> itx = pool.iterator();

sx = itx.next();

itx.remove();

}

temp.add(sx);

return sx;

}

/\*\*

\* 释放临时变量

\* @param s

\*/

public void releaseTemp(Object s) {

if (s instanceof TempSymbol) {

if(temp.remove(s)){

pool.add((Symbol)s);

}

}

}

/\*\*

\* 判断是否为临时变量

\* @param s

\* @return

\*/

public boolean isTemp(Symbol s) {

if(pool.contains(s))

throw new RuntimeException("Temporary Variable Leak!" + s);

return temp.contains(s);

}

/\*\*

\* 增加一个函数,将函数作为符号存放

\* @param name

\* @return

\*/

public Symbol addFunc(String name){

Symbol sx = new Symbol();

sx.name = name;

funcs.put(name, sx);

return sx;

}

public Symbol lookupFunc(String name){

return funcs.get(name);

}

/\*\*

\*为函数中的局部变量开辟空间

\*/

public void enterFunc() {

if(localSyms == null) {

localSyms = new HashMap<>();

} else {

throw new RuntimeException("Cannot define function within a function.");

}

}

/\*\*

\* 退出函数时,回收存放数据的空间

\* @return

\*/

public Map<String, Symbol> exitFunc(){

Map<String, Symbol> ret = localSyms;

localSyms = null;

return ret;

}

@SuppressWarnings("unchecked")

@Override

public String toString() {

StringBuffer sb = new StringBuffer();

sb.append("Global variables: \n");

for(Symbol sx : syms.values()) {

sb.append("\t").append(sx.details()).append("\n");

}

sb.append("Temporary variables:\n");

for(Symbol sx : pool){

sb.append("\t").append(sx.details()).append("\n");

}

sb.append("Functions:\n");

for(Symbol sx : funcs.values()) {

sb.append('\t').append(sx.name).append(": RetType = ").append(sx.attr("type")).

append(",BaseAddr = ").append(sx.attr("addr")).append(", [Parameters: ");

List<Symbol> parlist = (List<Symbol>)sx.attr("parlist");

if(parlist != null) {

for(Symbol syx : parlist) {

sb.append(syx.details()).append(", ");

}

sb.setLength(sb.length() - 2);

} else {

sb.append("No parameter");

}

sb.append("], [LocalVariables: ");

boolean hasVariable = false;

for(Entry<String, Symbol> ex: ((Map<String, Symbol>)sx.attr("localVar")).entrySet()) {

sb.append(ex.getValue().details()).append(", ");

hasVariable = true;

}

if(hasVariable)

sb.setLength(sb.length() - 2);

sb.append("]\n");

//.append(sx.attr("parlist")).append("\n");

}

return sb.toString();

}

public void allocateAddr() {

int allocPointer = startAddr;

// 为临时变量分空间

for(Symbol sx : pool){

sx.attr("addr", allocPointer);

sx.attr("size", 8);

allocPointer += 8;

}

// 全局变量分配空间

for(Entry<String, Symbol> sx : syms.entrySet()){

Symbol s = sx.getValue();

s.attr("addr", allocPointer);

int num = 1;

@SuppressWarnings("unchecked")

List<Integer> dim = (List<Integer>)s.attr("dim");

if(dim != null){

for(Integer ix : dim){

num \*= ix;

}

}

switch((String)s.attr("type")) {

case "int":

s.attr("size", INTSIZE \* num);

allocPointer += INTSIZE \* num;

break;

case "char":

s.attr("size", CHARSIZE \* num);

allocPointer += CHARSIZE \* num;

break;

case "bool":

s.attr("size", BOOLSIZE \* num);

allocPointer += BOOLSIZE \* num;

break;

case "real":

s.attr("size", REALZISE \* num);

allocPointer += REALZISE \* num;

break;

default:

throw new RuntimeException("wrong type of symbol: " + s.name + " type :" + s.attr("type"));

}

}

}

}

class TempSymbol extends Symbol{

@Override

public String toString() {

return "$" + name;

}

}

SyntaxAnalyzerImpl.java

package syntax;

import java.io.InputStream;

import java.lang.reflect.Method;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.HashSet;

import java.util.List;

import java.util.Map;

import java.util.Set;

import java.util.Stack;

import language.Action;

import language.AnalyzeTable;

import language.GrammarAnalyser;

import language.Production;

import lexical.LexicalAnalyzer;

import syntax.action.SemanticAction;

/\*\*

\* SyntaxAnalyzerImpl类,实现SyntaxAnalyzer接口

\* 实现语法分析的功能

\* @author lxm

\*

\*/

public class SyntaxAnalyzerImpl implements SyntaxAnalyzer {

private AnalyzeTable at; // SLR(1)分析表

private LexicalAnalyzer la; // 词法分析器

private SymbolTable st = new SymbolTable(); // 符号表

private Stack<Integer> sstate = new Stack<Integer>(); // 状态栈

private Stack<V> ssymbol = new Stack<>(); // 符号栈

private List<Quad> quadlist = new ArrayList<>(); // 生成的四元式队列

private Map<String, SemanticAction> saObjs = new HashMap<>(); // 语义动作映射

private Map<String, Method> samethods = new HashMap<String, Method>(); // 动作方法映射

public SyntaxAnalyzerImpl(InputStream analyzeTable) {

// 加载分析表

at = AnalyzeTable.load(analyzeTable);

if (at == null) {

throw new NullPointerException(" cannot be null");

}

}

@Override

public void setInput(LexicalAnalyzer la) {

if (la == null) {

throw new NullPointerException("Lexical Analyzer cannot be null");

}

this.la = la;

}

public SymbolTable getSymbolTable() {

return st;

}

@Override

public List<Quad> analyse() {

quadlist.clear(); // 清空四元式队列

int currState = 0;

Action action;

sstate.push(0); // 初始状态压栈

V term = new V();

term.name = GrammarAnalyser.TERMINATOR;

ssymbol.push(term); // '#'压栈

V v = la.next();

V[] param;

Production p;

while(true){

v = (v == null) ? term : v;

currState = sstate.peek(); // 查看栈顶的状态

String input = v.name; // 当前输入

action = at.query(currState, input); // 获得对于的动作

if (action == null){

/\*

\* 出错建议机制

\* 如果Action为NULL, 说明我们遇到了一个不能顺利进行分析的符号;

\* 为了帮助用户解决问题，我们可以采取以下策略：在该符号前插入

\* 所有可能输入的终结符中的一个，尝试继续分析过程，如果能够成功，

\* 并且能将原来不能顺利分析的符号移进掉，则认为该输入是有效的，

\* 并将其作为建议之一

\*/

// 取得不能被规约符号的符号名

String nextVt = (String) v.name;

Set<String> proposal = new HashSet<>();

// 遍历所有可能的终结符

outer: for(String vtx : at.getVT()) {

// 模拟

Action ax;

// 复制状态栈，无需复制符号栈。

Stack<Integer> sstack2 = new Stack<Integer>();

sstack2.addAll(sstate);

// 进行模拟规约，直到插入的终结符被规约掉。

int currstate2;

inner: while(true) {

currstate2 = sstack2.peek();

ax = at.query(currstate2, vtx);

// 如果走不通，直接看下一个终结符

if(ax == null) continue outer;

switch(ax.getType()) {

// 如果是归约而非移进，继续

case Action.REDUCTION:

int poplen = ax.getP().getRight().length;

for(int i = 0; i < poplen; i++)

sstack2.pop();

ax = at.query(sstack2.peek(), ax.getP().getLeft());

if(ax == null || ax.getType() != Action.GOTO) continue outer;

sstack2.push(ax.getState());

break;

case Action.GOTO:

// this should not occur.

continue outer;

case Action.ACC:

// 如果能接受ACC，则认为该符号可用

proposal.add(vtx);

continue outer;

case Action.STEPINTO:

// 移进掉了插入的符号

sstack2.push(ax.getState());

break inner;

}

}

// 规约原有的符号

currstate2 = sstack2.peek();

ax = at.query(currstate2, nextVt);

// 如果仍然不能被规约，说明该插入终结符无效

if(ax == null) continue;

proposal.add(vtx);

}

// 生成格式化的错误信息

StringBuffer err = new StringBuffer();

err.append("SYNTAX ERROR - Unexpected symbol: ").append(v.attr("value"));

err.append(", expect these symbols:").append(proposal);

throw new RuntimeException(err.toString());

}

// 根据不同类型执行不同的动作

switch (action.getType()) {

// 接受

case Action.ACC:

p = action.getP();

param = new V[p.getRight().length];

for(int i = 0; i < param.length; i++){

param[param.length - 1 - i] = ssymbol.pop();

sstate.pop();

}

V vnx = new V();

vnx.name = p.getLeft();

vnx.isFinal = false;

SemanticAction(p, vnx, param);

System.out.println("Analyze complete!");

st.allocateAddr();

break;

// 跳转

case Action.GOTO:

throw new RuntimeException("Unexpected 'Goto' action: " + action);

// 规约

case Action.REDUCTION:

p = action.getP();

param = new V[p.getRight().length];

for(int i = 0; i < param.length; i++){

param[param.length - 1 - i] = ssymbol.pop();

sstate.pop();

}

String tmp = p.getLeft();

V vn = new V();

vn.name = tmp;

vn.isFinal = false;

SemanticAction(p, vn, param);

action = at.query(sstate.peek(), tmp);

if(action != null && action.getType() == Action.GOTO && action.getState() != -1){

ssymbol.push(vn);

sstate.push(action.getState());

} else {

throw new RuntimeException("Reduction error at: " + v.attr("value"));

}

break;

// 移进

case Action.STEPINTO:

sstate.push(action.getState());

ssymbol.push(v);

v = la.next();

break;

// 出错

default:

throw new RuntimeException("Stepinto error at: " + v.attr("value"));

}

if(action.getType() == Action.ACC){

break;

}

}

return quadlist;

}

/\*\*

\* 根据要规约的产生式,执行相应的语义动作

\* @param p

\* @param left

\* @param param

\*/

private void SemanticAction(Production p, V left, V[] param){

String a = p.getAction(); // 获得动作执行路径

if (a == null) return;

int pos;

String clazz = a.substring(0, pos = a.lastIndexOf('.'));

String method = a.substring(pos + 1);

SemanticAction obj = saObjs.get(clazz);

if (obj == null) {

try {

// 获得类对象的实例

Class<?> cx = Class.forName(clazz);

obj = (SemanticAction) cx.newInstance();

obj.setQlist(quadlist);

obj.setSt(st);

} catch(Exception e) {

throw new RuntimeException(e);

}

}

Method mx = samethods.get(method);

if(mx == null){

try {

// 获得对象方法,调用方法

mx = obj.getClass().getMethod(method, V.class, V[].class);

samethods.put(a, mx);

mx.invoke(obj, left, param);

} catch (Exception e) {

throw new RuntimeException(e);

}

}

}

@Override

public List<Quad> getQuad() {

return quadlist;

}

}