编译原理课设报告

设计任务

一个PASCAL语言子集（PL/0）编译器的设计与实现。其编译过程采用一趟扫描方式，以语法分析程序为核心，词法分析和代码生成程序都作为一个独立的过程，当语法分析需要读单词时就调用词法分析程序，而当语法分析正确需要生成相应的目标代码时，则调用代码生成程序。

用表格管理程序建立变量、常量和过程标识符的说明与引用之间的信息联系。

用出错处理程序对词法和语法分析遇到的错误给出在源程序中出错的位置和错误性质。

当源程序编译正确时，PL/0编译程序自动调用解释执行程序，对目标代码进行解释执行，并按用户程序的要求输入数据和输出运行结果。

系统设计

编译程序就是能够把源语言程序转换为目标语言程序的程序。编译器的结构主要包括：词法分析器、语法分析器、语义分析与中间代码产生器、优化器以及目标代码生成器。PL/0语言的编译程序包括词法分析器、语法分析器、语义分析与中间代码产生器以及目标代码生成器，同时设计一个解释器对生成的P代码进行解释运行。PL/0语言编译程序采用那个以语法分析为核心，词法分析和语法分析及中间代码生成器作为独立的子程序共语法分析程序调用，成功生成中间代码后，调用后端目标代码生成器将中间代码翻译为P代码，在调用解释器对其解释运行。接下来对程序中的每个部分进行介绍。

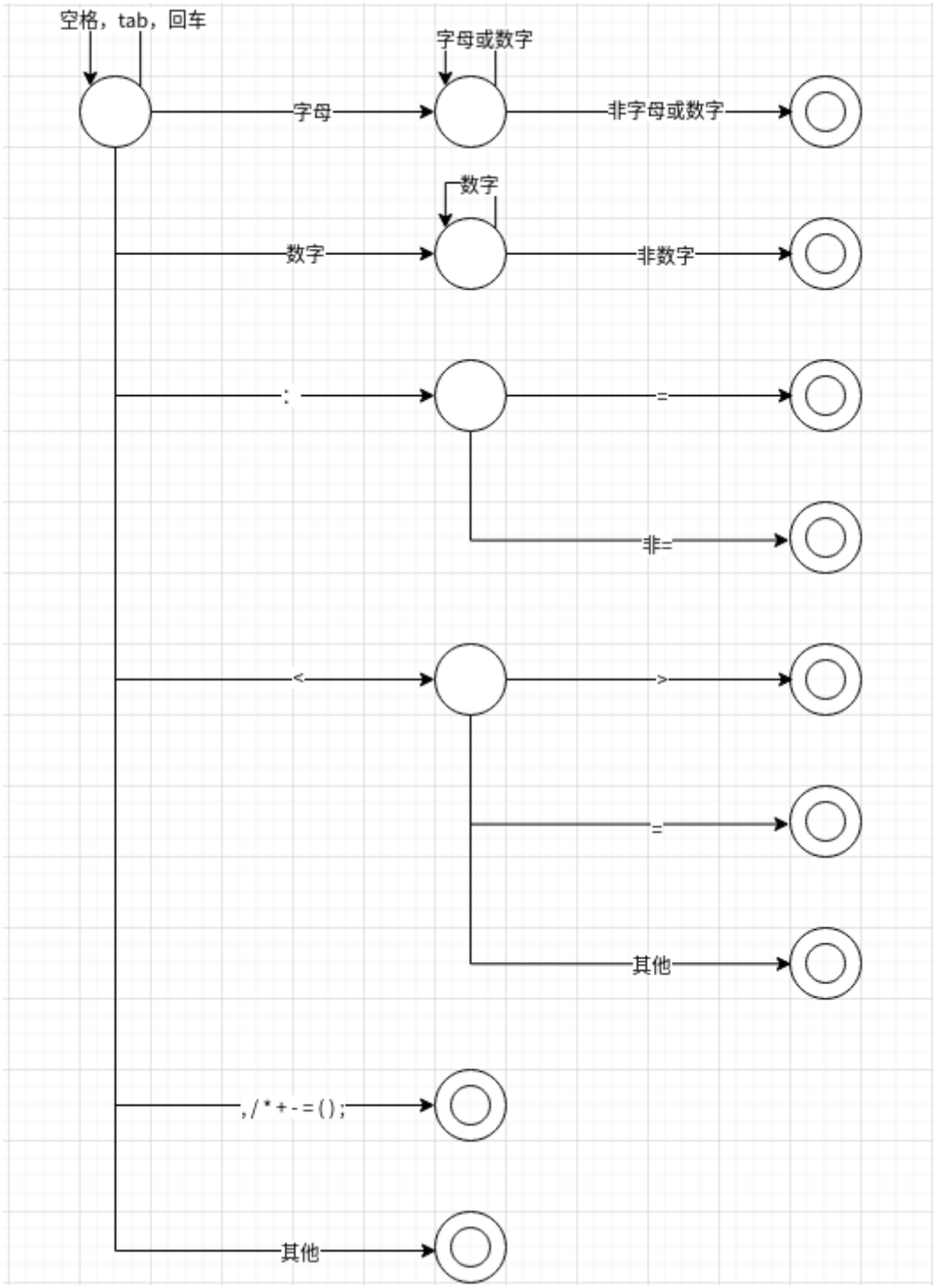
词法分析器

任务：

词法分析的任务是输入源程序，对构成源程序的字符串进行扫描和分解，识别出一个一个的单词。识别出的单词用于后续的语法分析阶段。

设计思路：

词法分析器的设计思想主要是利用有限状态机对输入的字符进行处理，得到单词符号或产生错误类型。当有限自动机成功识别出字母、PL/0语言所含的符号以及数字后，输出得到的单词，否则调用错误输出程序error_print（）输出对应的错误。每次语法分析器对一个单词进行匹配以后，都调用词法分析器对源程序进行分析，得到下一个单词。



语法分析器

任务：

语法分析的任务是在词法分析的基础上，根据语言的语法规则，把单词符号串分解成各类语法单位（语法范畴），如“短语”、“子句”、“句子”和“程序”等。

设计思路：

在语法分析阶段，采用自顶向下的语法分析模式。其主要步骤是：

1.消除左递归

2.提取公共左因子

3.求解First集合

4.对读进的单词进行匹配

由于PL/0语言不存在左递归以及公共左因子，我们求解PL/0语言的First集合：

```
<prog> = {'program'}
<block> = {'const','var','procedure','begin'}
<condecl> = {'const'}
<const> = {'l'}
<vardecl> = {'var'}
<body> = {'begin'}
<proc> = {'procedure'}
<statement> = {'if','while','call','read','write','l','begin'}
<lexp> = {'odd','(','l','d'}
<exp> = {'+','-','(','l','d'}
<term> = {'(','l','d'}
<factor> = {'(','l','d'}
<lop> = {'=','<>','<','<=','>','>='}
<aop> = {'+','-'}
<mop> = {'*','/'}
<id> = {'l'}
<integer> = {'d'}
```

l代表字母，d代表数字。

接着对得到的单词进行匹配。

1.获取单词，存到全局alpha中，若获取成功，跳转到2,否则跳转到4

2.对于每个词法分析器得到的字母a，当前非终结符A，若a ∈FIRST（αi）（αi为非终结符A的产生式），则用αi去匹配a。跳转到1

3.若不存在αi与a匹配，由于PL/0语言不存在产生式为空，所以这是一种语法错误，输出对应的错误。跳转到1

4.结束

在语法分析阶段，每一个非终结符都是一个过程，我们递归的调用非终结符形成的过程，对每个字母进行匹配。

语义分析与中间代码产生器

任务：

语法分析的任务是对语法分析所识别出的各类语法范畴分析其含义，并进行初步翻译，产生中间代码。包括两方面的工作：语法分析和中间代码的生成。

设计思路：

该阶段主要借助属性文法和翻译模式进行分析和产生代码。该阶段除了语法分析程序，还有表格管理程序对过程中的符号表进行变量、常量和过程标识符的说明与引用之间的信息联系。符号表用于记录该过程中的变量，常量，标识符等信息，具体我们的设计的符号表为：

```
符号表（table）：
表头：
前一张表（pre） 宽度（width） 层次（level） 程序起始地址（quad）
表项（item）：
名字（name） 类型（type） 偏址（offset） 数值（value）
```

为过程定义几个函数：

mktable:
输入：
pre 指向一张先前创建的符号表
idname 该过程的名称
layer 该过程的层次
功能：
在表格管理程序中创建一张新的符号表，记录过程中的变量，常量等信息，放着表格栈的顶端

pop：
功能：
将符号表栈栈顶的符号表出栈

enter：
输入：
idname 变量/常量名
type 类型
offset 偏址
value 值
功能：
在表格栈顶端的表格中加入新的一项（idname，type，offset，value）

enterproc：
输入：
idname 过程名
offset 偏址
value 该过程的符号表
功能：
在表格栈顶端的表格中加入新的一项（idname，’table‘，value）

lookup：
输入：
idname 变量/常量名称
输出：
变量/常量对应的入口
功能：
寻找idname对应变量/常量的入口，以便对变量/常量的访问
具体实现：
1.l=0，t=符号表栈栈顶的符号表
2.若t=None，跳转到4，;否则在t中寻找该变量/常量，若存在，则offset=该变量/常量在符号表的位置，type=该变量/常量的类型，跳转到3;否则t=该符号表的pre，l+=1,跳转到2
3.返回offset，type，l
4.输出错误（未定义）

emit:
输入：
code 三地址代码
功能：
生成中间代码

newtemp：
输出：
新临时变量ti
功能：
创建临时变量

backpath：
输入：

```
nextlist 回填列表
quad 回填地址
功能：
将回填列表中的第四部分用quad替换
```

创建空转换：

```
用于创建第一个符号表，插入到prog过程中的block前
M -- ep
{t = mkable(nil);
push(t, tblpter);
push(0, offset);}

用于创建新的符号表，插入到proc括号前
N -- ep
{t = mkable(top(tblptr));
push(t, tblpter);
push(0, offset);}

用于获取下一条代码地址，插入到body的statement之后（每个）/当statement翻译为if时，插入到then之后，若翻译出else，else之后也需插入/当statement翻译为while时，插入到while之后以及do之后
H -- ep
{H.quad = nextquad;}
```

设计属性文法：

```
<id>.name 变量/常量名称
<statement>.nextlist 回填列表
<lexp>.truelist 判断真出口
<lexp>.falselist 判断假出口
<exp>.place 运算量入口
<term>.place 运算量入口
<factor>.place 运算量入口
<lop>.c 判断
<aop>.c 加减运算符
<mop>.c 乘除运算符
<integer>.c 常数值
```

语法分析过程我们采用自顶向下的分析方法，每当一个过程分析完后，我们就知道了我们用那些单词对该过程进行归约，即可调用每个过程对应的语义分析来产生中间代码。

在这个任务中，对于P代码，我们生成的中间代码将访问常量直接翻译为常数，对于变量，翻译为相对于当前符号表的层差与偏址。同时在每个过程body部分的开始阶段和结束阶段设置标识符（begin、end）

目标代码生成器

任务：

将语义分析产生的中间代码转换为目标机上的目标代码

设计思路：

解释器

任务：

将产生的目标代码进行解释运行

设计思路：

解释器通过识别每一条P代码进行解释运行。运行环境为：

1.两个存储器（code存储P代码，stack数据栈）

2.寄存器I存储当前运行的代码

3.寄存器T存储当前数据栈栈顶

4.寄存器B存储当前活动记录基地址

5.寄存器P存储下一条要运行的代码

具体语句翻译思路：

```
LIT指令 将常数a直接放入数据栈栈顶，I=P+1
OPR指令
当a部分为0时，取出基地址B偏移3地址处的数，存到k中;将B处数据存到orib中;I=基地址B偏移2地址处的数;B=基地址B偏移1地址处的数;数据栈数据出栈，直到T=orib;数据栈出栈k+1个元素（k个形式变量，1个基地址）
当a部分为2/3/4/5/8/9/10/11/12/13时，栈顶两个元素出栈，进行相应的运算，将结果存入栈顶
当a部分为6时，数据栈栈顶元素出栈，存到k中，如果k为奇数，则将1放入到数据栈栈顶
当a部分为7时，
```

课设总结

参考资料