---PanHengyu(njust)

概述

编译程序是现代计算机系统的基本组成部分之一，而且多数计算机系统都配有不止一种高级语言的编译程序。从功能上来看，编译程序就是语言翻译程序，把一种语言书写的程序翻译成另一种语言的等价程序。编译程序主要分为 6 大阶段：词法分析、语法分析、语义分析、中间代码生成、代码优化以及目标代码生成。本文主要阐述了前两个阶段的设计与实现过程，并清晰地展示了该编译程序中 java 代码实现，有助于读者进一步了解编译器的工作原理。

首先本文设计了一种较为复杂的高级语言--S 语言,作为该编译程序的测试语言，验证其词法分析、语法分析过程的合理性和正确性。

在编译程序的词法分析阶段中，本文采用通过以上述语言的正规文法作为输入，自动生成相关的 NFA(不确定有限状态机),然后通过子集算法转换为 DFA(有限状态机),形成特定语言的词法分析器。本文中的词法分析器可以对该语言的源代码可以进行词法分析，输出为三元组(行号,类型,内容)表。

在编译程序的语法分析阶段中，本文采用自底向上的 LR(1)分析方法，对于上述语言的二型文法描述构造 LR(1)项目集族,随之生成 LR(1)分析表作为源代码的语法分析依据，判断该源代码的正确性和符合性。

1 S 语言介绍

本文中设计的语言主要有变常量的定义语句、赋值语句、条件语句、当循环语句、复合语句以及双目运算符表达式等。S 语言有三种数据类型--整型、浮点型、波尔型,不允许过程的嵌套定义.

S 语言的常量可以是整数、浮点数、以及科学计数法表示。S 语言的变量名由英文字母(大小写均可)、下划线以及数字组成,且数字不能为变量名的首字符。

1.1 S 语言例子

程序的开始以 begin....end 结构层层嵌套。

语句命名	语句表达形式
输入语句	cin (表达式) ;
输出语句	cout (表达式) ;
赋值语句	变量名 = 表达式 ;
复合语句	begin 定义语句/赋值语句/条件语句/当循环语句等 ; end
当循环语句	while 表达式 do [begin] 定义语句/赋值语句/条件语句/当循环语句等 ; [end]
条件语句(if-then)	if 表达式 then [begin] 定义语句/赋值语句/条件语句/当循环语句等 ; [end]
条件语句 (if-then-else)	if 表达式 then [begin] 定义语句/赋值语句/条件语句/当循环语句等 ; [end] else [begin] 定义语句/赋值语句/条件语句/当循环语句等 ; [end]

注:[]表示可以省略

表(1)S 语言用例

1.2 S 语言的 2 型文法描述

我们把 G(V_N,V_T,P,S),若 P 中的每一个产生式 α -> β 满足: α 是一个非终结符, β ∈(V_N ∪ V_T),则称此文法为 2 型文法或上下文无关文法。每个语言的都有自己独特的 2 型文法描述，且每个语言的 2 型文法是语法分析器生成 LR(1)分析表的依据，因此正确设计 S 语言的 2 型文法至关重要。

1.2.1 终结符和非终结符的定义

由于倘若语言的 2 型文法中使用中文描述终结符和非终结符，则计算机识别该文法将过于冗余与复杂，因此我们要将中文描述的 2 型文法字母化，使之简洁。

中文描述	字母编号	数字编号
	终　结　符	
Const	a	256
<标识符>	b	257
<常量>	c	258
=	d	259
int|float|boolean	e	260
;	f	261
,	g	262
if	h	263
then	i	264
else	j	265
<关系运算符>	k	266
while	l	267
do	m	268
cin	n	269
(	o	270
)	p	271
cout	q	272
begin	r	273
end	s	274
<加法运算符+|->	t	275
<乘法运算符>	u	276

非 终 结 符		
<程序>	A	1
<语句>	B	2
<常量说明>	C	3
<变量说明>	D	4
<常量定义Ⅰ>	E	5
<常量定义Ⅱ>	F	6
<标识符Ⅰ>	G	7
<赋值语句>	H	8
<表达式>	I	9
<条件语句>	J	10
<条件>	K	11
<当循环语句>	L	12
<输入语句>	M	13
<输出语句>	N	14
<复合语句>	O	15
<语句Ⅰ>	P	16
<表达式串>	Q	17
<开始>	S	0

表(2)终结符和非终结符定义表

表(2)中的字母编号是 2 型文法中终结符和非终结符输入语法分析器时的编号，而数字编号是 2 型文法中终结符和非终结符在语法分析器的存在形式，我们规定在语法分析器中编号大于等于 256 的是终结符，反之小于 256 是非终结符。该规则是语法分析器中判定字符类型的依据。

1.2.2 S 语言的 2 型文法描述

根据 S 语言的语句语法特性和语句结构特性，设计了该 2 型文法描述如下：

模块名	表达式中文描述	表达式 字母描述
程序入口 语法描述	<开始> --> <程序>	S->A
	<程序> --> <语句>	A->B
	<程序> --> <常量说明>	A->C
	<程序> --> <变量说明>	A->D
常量语法描述	<常量说明> --> Const<int|float|boolean><常量定义Ⅰ>	C->aeE
	<常量定义Ⅰ> --> <常量定义Ⅱ>;	E->Ff
	<常量定义Ⅰ> --> <常量定义Ⅱ>,<常量定义Ⅰ>	E->FgE
	<常量定义Ⅱ> --> <标识符>=<常量>	F->bdc
变量语法描述	<变量说明> --> <int|float|boolean><标识符Ⅰ>	D->eG
	<标识符Ⅰ> --> <标识符>;	G->bf
	<标识符Ⅰ>--><标识符>,<标识符Ⅰ>	G->bgG
赋值语句 语法描述	<语句> --> <赋值语句>	B->H
	<赋值语句> --> <标识符>=<表达式>;	H->bdIf
条件语句 语法描述	<语句> --> <条件语句>	B->J
	<条件语句> --> if<条件>then<语句>	J->hHKiB
	<条件语句> --> if<条件>then<语句>else<语句>	J->hKiBjB
	<条件> --> <表达式> <关系运算符==|<=|<|>|>=|!=><表达式>	K->IkI
当循环语句 语法描述	<语句> --> <当循环语句>	B->L
	<当循环语句> --> while<条件>do<语句>	L->lKmB
输入语句 语法描述	<语句>--><输入语句>	B->M
	<输入语句>-->cin(<表达式>);	M->noIpf
输出语句 语法描述	<语句>--><输出语句>	B->N
	<输出语句>-->cout(<表达式>);	N->qoIpf
复合语句 语法描述	<语句>--><复合语句>	B->O
	<复合语句>-->begin<语句><语句Ⅰ>	O->rBP
	<语句Ⅰ>-->end	P->s
	<语句Ⅰ>--><语句><语句Ⅰ>	P->BP
	<表达式> --> <表达式串>	I->Q
	<表达式> --> (<表达式>)	I->oIp
	<表达式> -->(<常量>)	I->ocp
	<表达式>--><常量>	I->c

	<表达式> --><标识符>	I->b
	<表达式> -->(<标识符>)	I->obp
表达式	<表达式串>--><表达式><加法运算符><表达式>	Q->ItI
语法描述	<表达式串> --> <表达式><乘法运算符><表达式>	Q->IuI
复合语句	<语句> --> <变量说明>	B->D
常/变量描述	<语句> --> <常量说明>	B->C

表(3)S 语言 2 型文法描述

表(3)中 S 语言 2 型文法描述中表达式字母描述作为生成 LR(1)分析表的依据,输入至语法分析器中。

2、词法分析器

词法分析是编译的第一个阶段，它的主要任务是从左至右逐个字符地对源程序进行扫描,产生一个个单词序列，用于语法分析。在本词法分析器中输出三元组(行号,类型,内容)形式的单词序列，不仅如此，还能判断标识符的正确性以及识别各种常量表示，比如:整数、浮点数、科学记数法以及复数等。对于错误的单词则会进行一定的输出提示。

2.1 词法分析器设计

2.1.1 词法分析器输入

本词法分析器的输入:识别常量的正规文法和识别正确标识符的正规文法。对于正规文法的设计过程：先设计出相应的有限状态机的结构图，然后转换为相应的正规文法作为词法分析器的输入。如图(1)所示，有限状态机的结构图。而对于关键字、界定符、运算符的识别由于这些类单词具有可列举性，所以在词法分析器中用 Set 数据结构存放相关单词，作为输入单词的匹配依据。

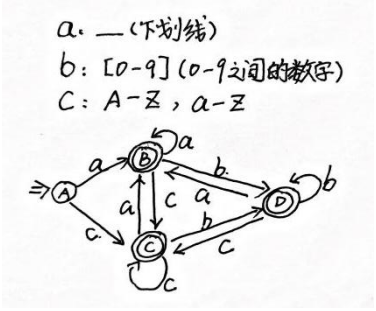


图 1.1 标识符状态机

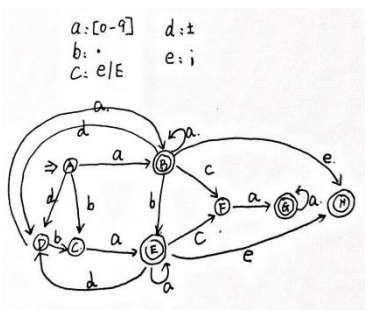


图 1.2 常量状态机

图(1) 有限状态机结构图

初步验证合理性：

(1)按照图 1.1 标识符状态机，如果单词为 abc,从状态 A 开始,字符 a 进入状态机后,由于字符 a 对应的输入符号为 c，因此机器进入状态 C,然后依次输入字符串 bc,最终机器仍然在状态 C,由于状态 C 为可接收状态，即与语言标识符设计相符。如果输入 1a_bc,从状态 A 开始，字符 1 进入状态机后，由于字符 1 对应的输入符号为 b，无法找到状态转换弧，此时输入的字符串 1a_bc 无法被接收。

(2)按照图 1.2 常量状态机,如果输入科学记数法 3.1e4,从状态 A 开始,字符 3 对应的输入符号为 a,则进入状态 B;然后字符.对应的输入符号为 b,进入状态 E;字符 1 对应的终结符为 a,则进入状态 E;字符 e 对应输入符号为 c,则进入状态 F;字符 4 对应的输入符号为 a，则进入状态 G,由于 G 为可接收状态，即与语言标识符设计相符。

有限状态机转换为正规文法，如表(5)所示。

正规 文法名	开始 符号	文法描述
isIdentier	A	A->aB A->cC B->aB B->bD B->cC B->* C->aB C->bD C->cC C->* D->aB D->bD D->cC D->*
isConst	A	A->aB A->bC A->dD B->aB B->dD B->cF B->eH B->bE C->aE D->aB D->bC E->aE E->dD E->cF E->eH F->aG F->dG G->aG H->* G->* B->* E->*
注:*表示空符号串		

表(5)正规文法表

2.1.2 词法分析器输出

本词法分析器的输出:三元组(行号,类型,内容)单词序列表。如表(5)所示,S 语言单词表。

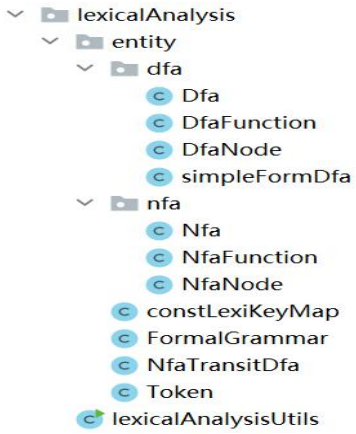
类型	内容
key	int,float,boolean,cin,cout,if,then,else,begin,end,do,while,Const
restriction	, ; ( )
operator	+ - * / % = == != < > <= >=
identiter	由字母数字和下划线组成，首字符不能为数字
const	整数，浮点数，科学计数法以及复数等

表(5)S 语言单词表

2.1.3 词法分析器的运行过程

首先将输入的正规文法转化为 NFA,然后将 NFA 通过子集法转化为 DFA。通过一系列转化获得的 DFA 将作为输入的源代码词法分析输出单词序列的依据。

在本工程 java 代码中的词法分析器相关文件目录,如图(2)所示。



图(2)词法分析器文件目录

constLexiKeyMap 文件记录了 S 语言的关键字、界限符以及运算符的映射表，用于判断输入单词是否属于关键字、界限符以及运算符。
lexicalAnalysisUtils 文件实现了正规文法转化为 NFA，NFA 转化为 DFA 的功能。

2.1.4 DFA 的生成算法

由于我们设计词法分析器的目的是面对各类语言而不是针对特定语言，因此我们在实现生成 DFA 功能时采用输入正规文法自动生成 DFA 的方式,而不是“手写”固定的 DFA，与此同时也增加了系统的鲁棒性和活泛性。

对于输入的正规文法 G(形如 A->a 或 A->aB),我们采用如下规则转化为 NFA M ,使得 L(M)=L(G)：

(1) M 的字母表 symbolTable 与 G 的终结符集 terminatorSet 命名相同。

(2) 为 G 中的每个非终结符生成 M 的一个以整数 Integer 表示的状态,G 的开始符号是 M 的开始状态 0。

(3) 增加一个新状态 Z，作为 M 的终态 endState,endState 数值为 G 的非终结符集合 noterminalSet 的大小+1。

(4) 对 G 中的形如 A->tB 的规则(其中 t 为终结符或空字符串,A 和 B 为非终结符的产生式)构造一个 M 的转化函数 NfaFunction (A,t)=B。

(5) 对 G 中形如 A->t 的产生式,构造 M 的转化函数 NfaFunction(A,t)=Z。

考虑到字符 char 数据类型表示范围远小于整数类型,因此 G 中非终结符转化为 NFA 的状态时映射为整数类型。

对于将 NFA 转化为等价的 DFA 需要借助于两个运算分别为闭包 e_closure 和弧转化。

e_closure:private Set<Integer> e_closure(Set<Integer> stateSet, Nfa nfa)表示状态集 stateSet 中的任意状态经过任意条空字符串 e 弧而能到达的状态集合。

弧转化:private Set<Integer>radian_move(Set<Integer> stateSet,Nfa nfa,char symbol)表示状态集 stateSet 中的任意状态经过一条 symbol 弧而到达的状态全体。因此子集法的核心思想就是让 DFA 的每一个状态对应 NFA 的一组状态,即通过 e_closure(radian_move(stateSet,nfa,symbol),nfa)函数求得 DFA 的每个状态。

假设 NFA N 按如下办法构造一个 DFA M 使得 L(M)=L(N);

(1) M 的状态集 stateSet 由 M 的状态集的一些子集构成(具体子集的计算方法如图(3)所示)。

```
①开始,令e_closure(NFA的开始状态集,nfa)为DFA状态集C中唯一成员,并且它是未被标记
的
②While(C中存在尚未被标记的子集T) do
{
   标记T;
   for 每个输入字母symbol do
      { U:=e_closure(radian_move(T,symbol,nfa),nfa);
       if U不在C中 then
         将U作为未被标记的子集加在C中
      }
}
```

图 3 子集构造算法

(2) M 和 N 的输入字母表是相同的，即是 symbolTable

(3) DFA 的转化函数定义为

radian_move(e_closure(NFA 状态集的子集,nfa),symbol)=NFA 状态集的子集，其中前后 NFA 状态集的子集可能不相同。

(4) DFA 的开始状态 beginState 为 e_closure(NFA 的 beginStateSet,nfa)。

(5) DFA 的结束状态 endState 为通过子集法求出的 DFA 状态对应 NFA 的一组状态，那么在对应 NFA 的一组状态中倘若存在一个 NFA 结束状态，那么该状态为 DFA 的结束状态。该类 DFA 状态组成的集合为 DFA 结束状态集 endStateSet。

2.2 词法分析器测试

2.2.1 测试源代码

```
begin
    Const int x* = 8+8i ;
    Const int y* = -9-2.2i ;
    float 1a = 1.3e-3 ;
    float b = 2.3e+3 ;
    cin ( a ) ;
    cout ( a ) ;
end
```

2.2.2 输出的 Token 表

```
!Error line:2 x* cause:标识符存在非法字符!
!Error line:3 y* cause:标识符存在非法字符!
!Error line:4 1a cause:标识符首字母不能为数字!
**词法分析器的token表(from testSource.txt)**********************************************************
LINE:2          TOKEN_CONTENT:Const             TYPE:key
LINE:2          TOKEN_CONTENT:int               TYPE:key
LINE:2          TOKEN_CONTENT:=                 TYPE:operator
LINE:2          TOKEN_CONTENT:8+8i              TYPE:const
LINE:2          TOKEN_CONTENT:;                 TYPE:restriction
LINE:3          TOKEN_CONTENT:Const             TYPE:key
LINE:3          TOKEN_CONTENT:int               TYPE:key
LINE:3          TOKEN_CONTENT:=                 TYPE:operator
LINE:3          TOKEN_CONTENT:-9-2.2i           TYPE:const
LINE:3          TOKEN_CONTENT:;                 TYPE:restriction
LINE:4          TOKEN_CONTENT:float             TYPE:key
LINE:4          TOKEN_CONTENT:=                 TYPE:operator
LINE:4          TOKEN_CONTENT:1.3e-3            TYPE:const
LINE:4          TOKEN_CONTENT:;                 TYPE:restriction
LINE:5          TOKEN_CONTENT:float             TYPE:key
LINE:5          TOKEN_CONTENT:b                 TYPE:identiter
LINE:5          TOKEN_CONTENT:=                 TYPE:operator
LINE:5          TOKEN_CONTENT:2.3e+3            TYPE:const
LINE:5          TOKEN_CONTENT:;                 TYPE:restriction
LINE:6          TOKEN_CONTENT:cin               TYPE:key
LINE:6          TOKEN_CONTENT:(                 TYPE:restriction
LINE:6          TOKEN_CONTENT:a                 TYPE:identiter
LINE:6          TOKEN_CONTENT:)                 TYPE:restriction
LINE:6          TOKEN_CONTENT:;                 TYPE:restriction
LINE:7          TOKEN_CONTENT:cout              TYPE:key
LINE:7          TOKEN_CONTENT:(                 TYPE:restriction
LINE:7          TOKEN_CONTENT:a                 TYPE:identiter
LINE:7          TOKEN_CONTENT:)                 TYPE:restriction
LINE:7          TOKEN_CONTENT:;                 TYPE:restriction
LINE:8          TOKEN_CONTENT:end               TYPE:key

**********************************************************************************************
```

2.2.3 测试用例分析

(1) 正确判断标识符正确性--首字母不能为数字且只由数字、字母和下划线组成:

源程序第二行 Const int x* = 8+8i; token 表中显示“!Error line:3 x* cause:标识符存在非法字符!”。

源程序第三行 Const int y* = -9-2.2i;token 表中显示“!Error line 4 y* cause:标识符存在非法字符”。

源程序第四行 float 1a = 1.3e-3; token 表中显示“!Error line:4 1a cause:标识符首字母不能为数字!”。

(2) 正确识别科学计数法表示的整数:

源程序第四行 float 1a = 1.3e-3;token 表中显示“token_content:1.3e-3 line:5 type:const”。

源程序第五行 float b=2.3e+3; token 表中显示“token_content:2.3e+3 line:6 type:const”。

(3) 正确识别复数表示形式:

源程序第二行 Const int x* = 8+8i;token 表中显示“token_content:8+8i line:3 type:const”。

源程序第三行 Const int y* =-9-2.2i;token 表中显示“token_content:-9-2.2i line:4type:const”。

(4) 正确识别系统中定义的运算符、关键字以及界限符。

3 语法分析器

语法分析是编译程序的核心功能之一。语法分析的作用是识别由词法分析给出的单词符号串是否是给定文法的正确句子(程序)。语法分析常用的方法可分为自顶向下分析和自底向上分析两大类。

自顶向上分析方法从文法的开始符号出发企图推导出与输入的单词符号串完全相匹配的句子。自顶向下的确定分析方法需对文法有一定的限制即文法中不能含有直接或间接左递归,或含有左公共因子。当然自顶向上的分析方法具有实现方法简单、直观等优点。

由于自底向上的分析方法能够识别大多数用无二义性上下文无关文法描述的语言，不仅如此还具有分析速度快，能准确、即时地指出出错位置的特点。

基于以上分析，我们最终采用 LR(1)自底向上的分析方法来进行语法分析，该分析方法不足之处就是不能分析存在“规约--规约冲突”的文法。

3.1 语法分析器的输入和输出

语法分析器的输入为表(2)所示 S 语言的 2 型文法描述中表达式的字母描述,基于以上 2 型文法描述先构造 LR(1)项目集族，随之建立 LR(1)分析表(Goto-Action 表),以该 LR(1)分析表作为分析依据,分析输入的句子或程序是否符合 S 语言的语法规则,并做出相应判断。该语法分析器在 2 型文法->LR(1)项目集族->LR(1)分析表的构造流程中会输出每一阶段的结果，便于调试以及理解语法

分析器的原理。

语法分析器中各个阶段的数据结构如表(6)所示,其中终结符和非终结符在语法分析器中均以整型 Integer 存在,**我们规定整数小于 256 的是非终结符大于等于 256 的是终结符。**

关键模块名	数据结构
2 型文法	HashMap<Integer,ArrayList<Production>>
LR(1)项目集族跳转函数	HashMap<GrammarState,HashMap<Integer,GrammarState>>
LR(1)分析表	HashMap<Integer,HashMap<Integer,Integer>>

表(6)关键数据结构

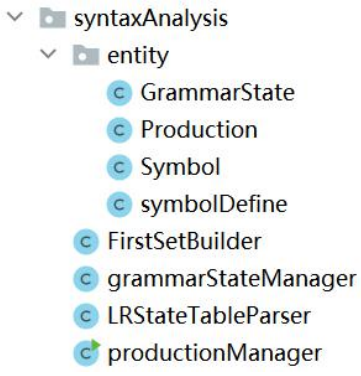
其中 2 型文法的 HashMap 的 key 表示非终结符,value 为产生式的列表,则每个元素表示相同的非终结符作为左式的所有产生式列表。

其中 LR(1)项目集族的外层 HashMap 的 key 表示项目集，内层 HashMap 的 key 表示输入的终结符,value 为跳转到的项目集,则每个元素表示原项目集在输入输入特定终结符下跳转到目的项目集。

其中 LR(1)分析表的外层 HashMap 的 key 表示状态编号，内层 HashMap 的 key 表示输入符号，value 表示移进规约操作。**我们规定该 value 小于 0 表示在 LR(1)分析时移进编号为 value 的状态进入状态栈中;大于 0 表示在 LR(1)分析时用编号为-value 的产生式进行规约;等于 0 表示在可接受,分析结束。**

3.2 语法分析器的运行过程

在本语法分析器的主要过程为 LR(1)项目集族的构造和 LR(1)分析表的构造。相关文件目录如图(3)所示。



图(3)语法分析器文件目录

3.2.1 LR(1)项目集族的构造

在构造 LR(1)项目集族的构造过程中,我们需要先计算 2 型文法中的所有终结符和非终结符的 First 集合。对于终结符来说，First 集合就是其本身;对于非终结符 A，遍历左式为符号 A 的所有产生式 Pro,获得产生式 Pro 的右式，依次从左向右扫描右式，直至遇到第一个终结符 b,则 FirstSet(A)=FirstSet(A)∪{b},在扫描期间遇到非终结符 B,则 FirstSet(A)=FirstSet(A)∪FirstSet(B)，然后进行 LR(1)项目集族的构造:

(1) 构造 LR(1)项目集的闭包函数

①假定 I 是一个项目集,I 的任何项目都属于 CLOSURE(I)。

②若有项目 $A \to \alpha \cdot B\beta, a$ 属于 CLOSURE(I), $B \to \gamma$ 是文法中的产生式,$\beta \in V^*$,$b \in FIRST(\beta\alpha)$，则 $B \to \cdot\gamma, b$ 也属于 CLOSURE(I)。

③重复②，直到 CLOSURE(I)不在增长为止。

(2) 构造转换函数 $GO(I,X) = CLOSURE(J)$,其中 I 是 LR(1)的项目集,X 是文法符号,$J = \{$任何形如$[A \to \alpha X \cdot \beta, a]$的项目$|[A \to \alpha \cdot X\beta, a] \in I\}$

在本语法分析器实现功能函数流程为:

计算 FirstSet 集合:FirstSetBuilder:runFirstSets()->addSymbolFirstSet(Symbol symbol)。构造 LR(1)项目集族:grammarStateManager:buildTransitionStateMachine()->GrammarState:createTransition()->GrammarState:makeClosure()->GrammarState:partition()->GrammarState:makeTransition()。

3.2.2 LR(1)分析过程

在预测分析来自词法分析的 token 表时,由于该 token 表中由三元组(行号,类型,内容)组成无法直接进行分析,且需保持单词原来的先后顺序,因此在分析前先根据 token 表中类型和内容信息转化为 S 语言 2 型文法描述中的终结符队列。依次从该终结符队列中取出终结符,结合 LR(1)分析表,按照以下规则进行分析:

(1)输入符号 a 为终结符(输入符号编号大于等于 256),在 LR(1)分析表中所对应的值为 i>0,此时是移进操作即把输入符号 a 和状态 i 分别移入文法符号栈和状态栈中。

(2)输入符号 A 为非终结符(输入符号编号小于 256),在 LR(1)分析表中所对应的值为 i<0,此时是规约操作即用编号为-i 的表达式规约。

(3)当输入符号为#(输入符号编号为-1),在 LR(1)分析表中所对应的值为 0，此时表示该输入字符串"接收",输出"the input can be accepted"。

(4)LR(1)分析表 lrStateTable 中没有相应的元素,则表示输入符号不满足 S 语言的语法，输出"the input is denied"。

3.3 语法分析器测试

3.3.1 测试 2 型文法

```
S->A
A->BB
B->SB
B->b
```

3.3.2 LR(1)构建过程

在语法分析器中所有的终结符和非终结符都是以整数的形式存在,其定义如下:非终结符为 S:0,A:1,B:2;非终结符为 a:256,b:257。
第一步:计算终结符和非终结符的 First 集合。

```
**********打印所有字符(非终结符和终结符)的FirstSet集合**********
(0) 非终结符1 firstSet:[257]
(1) 终结符256 firstSet:[256]
(2) 非终结符2 firstSet:[257]
(3) 终结符257 firstSet:[257]
(4) 非终结符0 firstSet:[257]

**********结束所有字符(非终结符和终结符)的FirstSet集合**********
```

第二步:构建 LR(1)项目集族。

```
开始进行LR(1)分析建立项目集组!(函数名:grammarStateManager.buildTransitionStateMachine)
I0:
流向:0(1) 1(2) 257(3) 2(4)
(0):0 -> . 1 ; -1
(1):1 -> . 2 2 ; -1
(2):2 -> . 0 2 ; 257
(3):2 -> . 257 ; 257
(4):0 -> . 1 ; 257
(5):1 -> . 2 2 ; 257

I1:
流向:0(1) 257(3) 1(5) 2(6)
(0):2 -> 0 . 2 ; 257
(1):2 -> . 0 2 ; 257
(2):2 -> . 257 ; 257
(3):0 -> . 1 ; 257
(4):1 -> . 2 2 ; 257

I2:
流向:
(0):0 -> 1 . ; -1
(1):0 -> 1 . ; 257

I3:
流向:
(0):2 -> 257 . ; 257
```

```
I4:
流向:0(8) 257(9) 1(5) 2(10)
(0):1 -> 2 . 2 ; -1
(1):1 -> 2 . 2 ; 257
(2):2 -> . 0 2 ; 257
(3):2 -> . 257 ; 257
(4):0 -> . 1 ; 257
(5):1 -> . 2 2 ; 257
(6):2 -> . 0 2 ; -1
(7):2 -> . 257 ; -1

I5:
流向:
(0):0 -> 1 . ; 257

I6:
流向:0(1) 257(3) 1(5) 2(7)
(0):2 -> 0 2 . ; 257
(1):1 -> 2 . 2 ; 257
(2):2 -> . 0 2 ; 257
(3):2 -> . 257 ; 257
(4):0 -> . 1 ; 257
(5):1 -> . 2 2 ; 257

I7:
流向:0(1) 257(3) 1(5) 2(7)
(0):1 -> 2 2 . ; 257
(1):1 -> 2 . 2 ; 257
(2):2 -> . 0 2 ; 257
(3):2 -> . 257 ; 257
(4):0 -> . 1 ; 257
(5):1 -> . 2 2 ; 257

I8:
流向:0(8) 257(9) 1(5) 2(11)
(0):2 -> 0 . 2 ; 257
(1):2 -> 0 . 2 ; -1
(2):2 -> . 0 2 ; -1
(3):2 -> . 257 ; -1
(4):0 -> . 1 ; 257
(5):1 -> . 2 2 ; 257
(6):2 -> . 0 2 ; 257
(7):2 -> . 257 ; 257

I9:
流向:
(0):2 -> 257 . ; 257
(1):2 -> 257 . ; -1

I10:
流向:0(1) 257(3) 1(5) 2(7)
(0):1 -> 2 2 . ; -1
(1):1 -> 2 2 . ; 257
(2):1 -> 2 . 2 ; 257
(3):2 -> . 0 2 ; 257
(4):2 -> . 257 ; 257
(5):0 -> . 1 ; 257
(6):1 -> . 2 2 ; 257
```

```
I11:
流向:0(1) 257(3) 1(5) 2(7)
(0):2 -> 0 2 . ; 257
(1):2 -> 0 2 . ; -1
(2):1 -> 2 . 2 ; 257
(3):2 -> . 0 2 ; 257
(4):2 -> . 257 ; 257
(5):0 -> . 1 ; 257
(6):1 -> . 2 2 ; 257

结束进行LR(1)分析建立项目集组!(函数名:grammarStateManager.buildTransitionStateMachine)
正在建立LR(1)语法分析表!(函数名:LRStateTableParser.LRStateTableParser)
```

第三步:根据第二步中构造的项目集族,构建 LR(1)分析表。

```
LR(1)分析表如下:**********************************************************
FROM:0                          INPUT:0                         TO:S1
FROM:0                          INPUT:1                         TO:S2
FROM:0                          INPUT:257                       TO:S3
FROM:0                          INPUT:2                         TO:S4
FROM:1                          INPUT:0                         TO:S1
FROM:1                          INPUT:257                       TO:S3
FROM:1                          INPUT:1                         TO:S5
FROM:1                          INPUT:2                         TO:S6
FROM:2                          INPUT:-1                        TO:S0
FROM:2                          INPUT:257                       TO:S0
FROM:3                          INPUT:257                       TO:R3
FROM:4                          INPUT:0                         TO:S8
FROM:4                          INPUT:257                       TO:S9
FROM:4                          INPUT:1                         TO:S5
FROM:4                          INPUT:2                         TO:S10
FROM:5                          INPUT:257                       TO:S0
FROM:6                          INPUT:0                         TO:S1
FROM:6                          INPUT:257                       TO:R2
FROM:6                          INPUT:1                         TO:S5
FROM:6                          INPUT:2                         TO:S7
FROM:7                          INPUT:0                         TO:S1
FROM:7                          INPUT:257                       TO:R1
FROM:7                          INPUT:1                         TO:S5
FROM:7                          INPUT:2                         TO:S7
FROM:8                          INPUT:0                         TO:S8
FROM:8                          INPUT:257                       TO:S9
FROM:8                          INPUT:1                         TO:S5
FROM:8                          INPUT:2                         TO:S11
FROM:9                          INPUT:-1                        TO:R3
FROM:9                          INPUT:257                       TO:R3
FROM:10                         INPUT:0                         TO:S1
FROM:10                         INPUT:-1                        TO:R1
FROM:10                         INPUT:257                       TO:R1
```

```
FROM:10                         INPUT:0                         TO:S1
FROM:10                         INPUT:-1                        TO:R1
FROM:10                         INPUT:257                       TO:R1
FROM:10                         INPUT:1                         TO:S5
FROM:10                         INPUT:2                         TO:S7
FROM:11                         INPUT:0                         TO:S1
FROM:11                         INPUT:-1                        TO:R2
FROM:11                         INPUT:257                       TO:R2
FROM:11                         INPUT:1                         TO:S5
FROM:11                         INPUT:2                         TO:S7
************************************************************************
结束建立LR(1)语法分析表!(函数名:LRStateTableParser.LRStateTableParser)
```

第四步:LR(1)分析过程：

输入字符串 bbbb 输出结果为“the input can be accepted”。输入字符串 aaab 输出结果为“the input is denied”。输入字符串 abab 输出结果为“the input can be accepted”。以上语法分析器输出结果与输入的 2 型文法描述相符合。

4 S 语言的测试

结合 S 语言的特性、词法分析器以及语法分析器的功能实现，以如下图中程序作为测试源代码。

```
1
2  begin
3      int input1 , input2 ;
4      Const int maxn = 100 ;
5      int out ;
6      cin ( input1 ) ;
7      cin ( input2 ) ;
8
9      if input1 < 0 then input1 = 0 ;
10
11      if input1 > input2 then
12      out = ( input1 + maxn ) * 100 ;
13      else
14      out = ( input2 + maxn ) * 100 ;
15
16      cout ( out ) ;
17
18      while input1 > input2 do input1 = input1 - 1 ;
19  end
```

4.1 Token 表

将以上源代码输入至项目 complier 文件夹下的 testSource.txt 文件中，然后运行 MainLexical.java 文件中 main 主函数，部分结果如下：

```
**词法分析器的token表(from testSource.txt)*******************************************************
LINE:2          TOKEN_CONTENT:begin             TYPE:key
LINE:3          TOKEN_CONTENT:int               TYPE:key
LINE:3          TOKEN_CONTENT:input1            TYPE:identiter
LINE:3          TOKEN_CONTENT:,                 TYPE:restriction
LINE:3          TOKEN_CONTENT:input2            TYPE:identiter
LINE:3          TOKEN_CONTENT:;                 TYPE:restriction
LINE:4          TOKEN_CONTENT:Const             TYPE:key
LINE:4          TOKEN_CONTENT:int               TYPE:key
LINE:4          TOKEN_CONTENT:maxn              TYPE:identiter
LINE:4          TOKEN_CONTENT:=                 TYPE:operator
LINE:4          TOKEN_CONTENT:100               TYPE:const
LINE:4          TOKEN_CONTENT:;                 TYPE:restriction
LINE:5          TOKEN_CONTENT:int               TYPE:key
LINE:5          TOKEN_CONTENT:out               TYPE:identiter
LINE:5          TOKEN_CONTENT:;                 TYPE:restriction
LINE:6          TOKEN_CONTENT:cin               TYPE:key
LINE:6          TOKEN_CONTENT:(                 TYPE:restriction
LINE:6          TOKEN_CONTENT:input1            TYPE:identiter
LINE:6          TOKEN_CONTENT:)                 TYPE:restriction
LINE:6          TOKEN_CONTENT:;                 TYPE:restriction
LINE:7          TOKEN_CONTENT:cin               TYPE:key
LINE:7          TOKEN_CONTENT:(                 TYPE:restriction
LINE:7          TOKEN_CONTENT:input2            TYPE:identiter
LINE:7          TOKEN_CONTENT:)                 TYPE:restriction
LINE:7          TOKEN_CONTENT:;                 TYPE:restriction
LINE:9          TOKEN_CONTENT:if                TYPE:key
LINE:9          TOKEN_CONTENT:input1            TYPE:identiter
LINE:9          TOKEN_CONTENT:<                 TYPE:operator
LINE:9          TOKEN_CONTENT:0                 TYPE:const
LINE:9          TOKEN_CONTENT:then              TYPE:key
LINE:9          TOKEN_CONTENT:input1            TYPE:identiter
LINE:9          TOKEN_CONTENT:=                 TYPE:operator
LINE:9          TOKEN_CONTENT:0                 TYPE:const
```

4.2 LR 分析表

将以上源代码输入至项目 complier 文件夹下的 testSource.txt 文件中，然后运行 MainSyntax.java 文件中 main 主函数，部分结果如下：

```
LR(1)分析表如下:*********************************************************
FROM:0                          INPUT:256                       TO:S1
FROM:0                          INPUT:1                         TO:S2
FROM:0                          INPUT:257                       TO:S3
FROM:0                          INPUT:2                         TO:S4
FROM:0                          INPUT:3                         TO:S5
FROM:0                          INPUT:4                         TO:S6
FROM:0                          INPUT:260                       TO:S7
FROM:0                          INPUT:263                       TO:S8
FROM:0                          INPUT:8                         TO:S9
```

最后输出：

```
the input can be accepted
```