Top-Down Parsing

1

Contents

1. Modify grammar
   - Left Recursion Removal
   - Left Factoring
2. First and Follow Sets
3. Top-Down Parsing by Recursive-Descent
4. LL(1) Parsing

2

Left Recursion Removal

- **Immediate left recursion:**

  exp → *exp* + *term* | *exp* - *term* |*term*
- **Indirect left recursion:**

  A → *B*b |…

  B → *A*a |…
- **A top-down parsing cannot terminate if there are left recursions in productions.**

3

Simple Immediate Left Recursion Removal

- **G:** $P \rightarrow P\alpha|\beta$， $P \in N$
- **After Left Recursion Removal**

  G': $P \rightarrow \beta P'$

  $P' \rightarrow \alpha P'|\varepsilon$

4

General Immediate Left Recursion Removal

- $P \rightarrow P\alpha_1|P\alpha_2|\ldots|P\alpha_m|\ \beta_1|\ \beta_2|\ldots|\ \beta_n$
- **The solution is similar to the simple case:**

  $P \rightarrow \beta_1 P'|\beta_2 P'|\ldots|\beta_n P'$

  $P' \rightarrow \alpha_1 P'|\alpha_2 P'|\ldots|\alpha_m P'|\varepsilon$

5

Example

- *exp* → *exp* + *term* | *exp* - *term* |*term*
- **Remove the left recursion as follows:**

  exp → *term exp'*

  exp' → +*term exp'* | - *term exp'* |ε

6

Left Factoring

- $A \to \delta\beta_1|\delta\beta_2|\ldots|\delta\beta_n|\gamma_1|\gamma_2|\ldots|\gamma_m$
- **Example:**

  If-stmt → if (exp) statement

  | if (exp) statement else statement
- A top-down parsing **cannot distinguish** between the production choices in such a situation.

7

Left Factoring

- $A \to \delta\beta_1|\delta\beta_2|\ldots|\delta\beta_n|\gamma_1|\gamma_2|\ldots|\gamma_m$
- The solution is to "factor" the δ out on the left and rewrite the rule as two rules:

  $A \to \delta A'|\gamma_1|\gamma_2|\ldots|\gamma_m$

  $A' \to \beta_1|\beta_2|\ldots|\beta_n$

8

Example

- Consider the following grammar for if-statements:

  if-stmt → **if** (*exp*) *statement*

  |**if**(*exp*) *statement* **else** *statement*
- The left factored form

  if-stmt → **if** (*exp*) *statement else-part*

  else-part → **else** *statement* |ε

9

Contents

1. Modify grammar
   - Left Recursion Removal
   - Left Factoring
2. First and Follow Sets
3. Top-Down Parsing by Recursive-Descent
4. LL(1) Parsing

10

Definition of First(α)

First(α) = {a | $\alpha \overset{*}{\Rightarrow}$ a……， a∈T}

α可能为单个符号X或一串符号

(1)First(X)，X∈(N∪T∪ε)。

a. First(ε)={ε};

b. if X∈T， then First(X)={X};

11

c. if X∈N, then for each production choice $X \to X_1X_2\ldots X_n$,

- First(X) contains First(X_1)-{ε};
- If ε∈First(X_1), then First(X) also contains First(X_2)-{ε};
- ……
- If ε∈First(X_1)… ε∈First(X_i), then First(X) also contains First(X_{i+1})-{ε};
- If ε∈First(X_1) … ε∈First(X_n), then ε∈First(X).

12

(2)First(α), if $\alpha=X_1X_2\ldots X_n$,

- First(α) contains First(X_1)-{ε};
- If ε∈First(X_1), then First(α) also contains First(X_2)-{ε};
- ……
- If ε∈First(X_1)… ε∈First(X_i), then First(α) also contains First(X_{i+1})-{ε};
- If ε∈First(X_1) … ε∈First(X_n), then ε∈First(α).

13

Example

G: *E*→*TE'*
E'→+ *TE'*| ε
T→*FT'*
T'→* *FT'*| ε
F→(*E*) | i

First（*E*）={(，i）
First（*E'*）={+，ε}
First（*T*）={(，i}
First（*T'*）={*，ε}
First（*F*）={(，i}

14

Computing First(α)

G: *E*→*TE'*
E'→+ *TE'*| ε
T→*FT'*
T'→* *FT'*| ε
F→(*E*) | i

First（*F*）={(，i}
First（*T'*）={*，ε}
First（*T*）={(，i}
First（*E'*）={+，ε}
First（*E*）={(，i}

First(α):
First(*TE'*)={(，i}
First(+*TE'*)={+}
First(*FT'*)={(，i}
First(**FT'*)={*}
First(ε)={ε}
First((*E*))={(}
First(i)={i}

15

(3) Algorithm for Computing First (A)

```
for all nonterminals A do First(A)={};
while any First(A) is changed do
  for each A→X1X2 …Xn do
    { k=1;  Continue=T;
      while Continue=T and k<=n do
        { add First(Xk)-{ε} to First(A);
          if ε∉First(Xk)  then Continue=F;
          k=k+1;}
      if Continue=T  then add ε to First(A);
    }
```

16

Example

- The Expression Grammar:

E→TE'
E'→+TE' |ε
T→FT'
T'→*FT' |ε
F→（E）|i

17

The computing process

(1) E→TE'
(2) E'→+TE'
(3) E'→ε
(4) T→FT'
(5) T'→*FT'
(6) T'→ε
(7) F→(E)
(8) F→i

Pass 1
•First(E)={ }
•First(E')={+}
•First(E')={+, ε}
•First(T)={}
•First(T')={*}
•First(T')={*, ε}
•First(F)={ (}
•First(F)={(, i}

18

The computing process

(1) E→TE'
(2) E'→+TE'
(3) E'→ε
(4) T→FT'
(5) T'→*FT'
(6) T'→ε
(7) F→(E)
(8) F→i

Pass 2
- First(E)={ }
- First(E')={+, ε}
- First(T)={(, i}
- First(T')={*, ε}
- First(F)={(, i}

19

The computing process

(1) E→TE'
(2) E'→+TE'
(3) E'→ε
(4) T→FT'
(5) T'→*FT'
(6) T'→ε
(7) F→(E)
(8) F→i

Pass 3
- First(E)={(, i}
- First(E')={+, ε}
- First(T)={(, i}
- First(T')={*, ε}
- First(F)={(, i}

20

Definition of Follow(A)

Follow(A)= {a|$S \overset{*}{\Rightarrow}$ …Aa…, a∈T}

- **For start symbol S, \$∈Follow(S);**
- **If $B \to \alpha A \beta$, then FIRST(β)-{ε} is in Follow(A);**
- **If $B \to \alpha A$ or $B \to \alpha A \beta$ and ε∈First(β), then Follow(A) contains Follow(B). (Add Follow(B) to Follow(A))**

21

G: E→TE'
E'→+TE'|ε
T→FT'
T'→*FT' |ε
F→(E)|i

First（F）={(, i}
First（T'）={*, ε}
First（T）={(, i}
First（E'）={+, ε}
First（E）={(, i}

Follow（E）={\$,) }
Follow（E'）={\$,) }
Follow（T）={\$, +,) }
Follow（T'）={\$, +,) }
Follow（F）={\$,), +, *}

22

Algorithm for Computing Follow(A)

Follow(S)={\$};
for all other A∈N do Follow(A)={};
while any Follow(A) is changed do
  for each $A \to X_1X_2 \ldots X_n$ do
    for each X_i∈N do
      {add First($X_{i+1} \ldots X_n$)-{ε} to Follow(X_i);
       if ε∈First($X_{i+1} \ldots X_n$) then
         add Follow(A) to Follow(X_i);
      }

23

Example

- **The Expression Grammar:**

E→TE'
E'→+TE'|ε
T→FT'
T'→*FT' |ε
F→(E)|i

First Sets
First（F）={(, i}
First（T'）={*, ε}
First（T）={(, i}
First（E'）={+, ε}
First（E）={(, i}

24

E→TE'
E'→+TE'
T→FT'
T'→*FT'
F→(E)

First Sets
First（F）={(，i}
First（T'）={*，ε}
First（T）={(，i}
First（E'）={+，ε}
First（E）={(，i }

Pass 1
•Follow(E)={$}
•Follow(T)={+, $}
•Follow(E')={$}
•Follow(F)={*, +, $}
•Follow(T')={+, $}
•Follow(E)={$,)}

25

E→TE'
E'→+TE'
T→FT'
T'→*FT'
F→(E)

First Sets
First（F）={(，i}
First（T'）={*，ε}
First（T）={(，i}
First（E'）={+，ε}
First（E）={(，i }

Pass 1结果
•Follow(E)={$,)}
•Follow(T)={+, $}
•Follow(E')={$}
•Follow(F)={*, +, $}
•Follow(T')={+, $}

Pass 2
•Follow(E)={$,)}
•Follow(T)={+,$,)}
•Follow(E')={$,)}
•Follow(F)={*, +, $,)}
•Follow(T')={+, $,)}

26

Contents

1. Modify grammar
   - Left Recursion Removal
   - Left Factoring
2. First and Follow Sets
3. Top-Down Parsing by Recursive-Descent
4. LL(1) Parsing

27

The idea of Recursive-Descent Parsing

- $A \rightarrow X_1X_2 \ldots X_n$
- Viewing the grammar rule for a non-terminal *A* as a definition for a procedure to recognize an *A*.
- The right-hand side of the grammar for *A* specifies the structure of the code for this procedure.

28

具体实现方式

每个非终结符A对应一个过程A( )，过程体：
- 选择A的一个产生式；
- 产生式中的终结符对应一个匹配操作；
- 产生式中的非终结符对应一个对其过程的调用。

```
void A( ){
  选择一个A的产生式，A→X1X2…Xn
  for(i=1 to n){
    if(Xi为非终结符）
      调用Xi( );
    else if(Xi与输入符号匹配）
      读入下一个符号;
    else 出错处理;
  }
}
```

29

The examples

- The Expression Grammar:

exp→exp addop term
　| *term*
addop→ + | -
term→term mulop factor
　| *factor*
*mulop →**
factor→(*exp*) | number

```
void factor( )
{
  switch token {
    case ( :  match( ( );
              exp();
              match( ) );
    case number:
              match (number);
    default: error;
  }
}
```

30

Problems in Recursive-Descent Parsing

- *exp*→*exp addop term* | *term* | ε

用First集、Follow集解决

```
void exp()
{ 选择一个产生式；//选择哪一个产生式？
  //若选择了exp→exp addop term
    exp(); //递归调用自己
    addop();
    term();
   ……
}
```

修改文法，消除左递归

31

- *exp*→*exp* + *term* | *term* | ε   *term*→num
- 消除左递归

  exp→ *term E* | *E*   *E*→ + *term E*| ε

- First(exp)={num, +, ε}   First(E)={+, ε}

  Follow(exp)={$}   Follow(E)={$}

```
void exp()
{ if(token==num) {term(); E();}
  else if(token==+) E();
  else if(token==$) return;
}
```

```
void E()
{ if(token==+) {
    match(+); term(); E();}
  else if(token==$) return;
}
```

32

Problems in Recursive-Descent Parsing

- The grammar rule for an if-statement:

  If-stmt → if (*exp*) *statement*

  ｜ if (*exp*) *statement* else *statement*

```
void ifstmt()
{ // 在没有看到后面是否有else之前，应该选择
  哪一个产生式？
}
```

修改文法，提取左因子

33

Problems in Recursive-Descent Parsing

- The grammar rule for an if-statement:

  If-stmt → if (*exp*) *statement*

  ｜ if (*exp*) *statement* else *statement*

- 提取左因子

  If-stmt → if (*exp*) *statement E*

  E → else *statement* ｜ ε

```
void ifstmt()
{ match(if);
  ……
}
```

```
void E()
{ if(token==else)
    {match(else); statement();}
  else if(token==$)  return; }
```

34

Contents

1. **Modify grammar**
   - **Left Recursion Removal**
   - **Left Factoring**
2. **First and Follow Sets**
3. **Top-Down Parsing by Recursive-Descent**
4. **LL(1) Parsing**

35

(1) Main idea of LL(1) Parsing

36

General Schematic

- A LL(1) parser begins by pushing the start symbol onto the stack
- It accepts an input string if, after a series of actions, the stack and the input become empty
- A general schematic for a successful LL(1) parse:

Parsing Stack	Input	Action
StartSymbol$	*Inputstring*$	action
…	…	action
…	…	action
$	$	accept

37

Two Actions

- **The two actions**
  - Output: Replace a non-terminal ***A*** at the top of the stack by a string α(in reverse order) using a grammar rule ***A***→α
  - Match: Match a token on top of the stack with the next input token

38

(2) The LL(1) Parsing Table and Algorithm

39

Purpose and Example of LL(1) Parsing Table

- **Purpose of the LL(1) Parsing Table:**
  - **To express the possible rule choices for *A* when *A* is at the top of stack based on the current input token (the look-ahead).**
- **The LL(1) Parsing table for grammar: *S*→(*S*) *S* | ε**

M[N,T]	(	)	$
S	*S*→(*S*) *S*	*S*→ ε	*S*→ ε

40

S→(*S*) *S* | ε

M[N,T]	(	)	$
S	*S*→(*S*) *S*	*S*→ ε	*S*→ ε

步骤	分析栈	输入串	动作
1	*S*$	( ) $	output: *S*→(*S*) *S*
2	(*S*)*S*$	( ) $	match
3	*S*)*S*$	)$	output: *S*→ ε
4	)*S*$	)$	match

LL(1)分析表M：当*A*在栈顶、输入为a时，使用M[*A*,a]位置的产生式进行output。

41

The General Definition of Table

- **The table is a 2-dimensional array indexed by non-terminals and terminals**
- **Containing production choices to use at the appropriate parsing step called M[N,T]**
  - **N: set of non-terminals**
  - **T: set of terminals(including $)**
- **Any entrances remaining empty**
  - **Representing potential errors**

42

The LL(1) parsing table constructing rules

算法4.17（课本4.4.3节 P131）

For each production choice $A \to \alpha$,

- For each $a \in First(\alpha)$, add $A \to \alpha$ to the table entry M[A,a];
- If $\varepsilon \in First(\alpha)$，then for each $b \in Follow(A)$, add $A \to \alpha$ to the table entry M[A,b].

43

$E \to TE'$
$E' \to +TE' \mid \varepsilon$
$T \to FT'$
$T' \to *FT' \mid \varepsilon$
$F \to (E) \mid i$

First(TE')={(,i}
First(+ TE')={+}
First(FT')={(,i}
First(* FT')={*}
First((E))={(}
First(i)={i}

Follow(E)={$,)}
Follow(E')={$,)}
Follow(T)={$,+,)}
Follow(T')={$,+,)}
Follow(F)={$,),+,*}

	i	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to i$			$F \to (E)$		

44

Theorem

A grammar is LL(1) if the following conditions are satisfied.

- For every $A \to \alpha_1|\alpha_2|\ldots|\alpha_m$,
  $First(\alpha_i) \cap First(\alpha_j) = \Phi$ （$i \neq j$）
- If $\varepsilon \in First(A)$, $First(A) \cap Follow(A) = \Phi$.

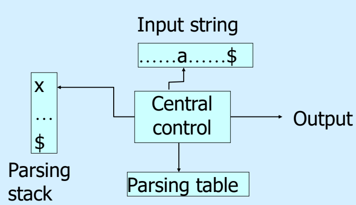
45

Theorem

- In other words: A grammar is an LL(1) grammar if the associated LL(1) parsing table has at most one production in each table entry
- An LL(1) grammar cannot be ambiguous.

46

Table-based LL(1) Parsing Algorithm



47

Table-based LL(1) Parsing Algorithm

/*use $ marks the bottom of the stack and the end of the input, assumes X is the current symbol on top of the stack, a is the next input token */
push *S* onto the top of the parsing stack;
while X≠$ **do**
{ if X==a
then (* match *)
pop the parsing stack;
advance the input;

48

else if X∈*N* and M[X,a] is $X \to X_1X_2 \ldots X_n$
then (* output *)
output $X \to X_1X_2 \ldots X_n$;
pop the parsing stack;
for i:=n downto 1 do
push X_i onto the parsing stack;
else error;
let X be the top stack symbol;
} //end of while
if *X*==$ and a==$ then accept
else error.

49

表驱动的LL(1)分析方法

(设栈顶符号为X，当前输入符号为a)
将$和文法开始符号先后入栈；然后执行下述：

- 若X=a='$'，则分析成功，结束；
- 若X=a≠'$'，则a匹配成功，X出栈，处理下一个输入符号；
- 若X为非终结符，则查分析表：
  - 若M[X,a]是一条产生式，则X出栈，同时将产生式的右部所有符号按反序进栈；
  - 若M[X,a]为Error，则出错处理。

重复上述过程，直至成功或出错。

50

Table-based LL(1) Parsing example

S *statement* → *if-stmt* | **other**
I *if-stmt*→**if**(*exp*) *statement else-part*
L *else-part* →**else** *statement* |ε
E *exp* → 0 | 1

	if	other	else	0	1	$
S	*S→I*	*S*→other				
I	*I*→if (*E*) *S L*					
L			*L*→else *S* *L*→ε			*L*→ε
E				*E*→*0*	*E*→1	

51

Parsing steps: if (0) if (1) other else other

Steps	Parsing Stack	Input	Action
1	S$	if(0)if(1)other else other$	S→I
2	I$	if(0)if(1)other else other$	I→if(E)SL
3	if(E)SL$	if(0)if(1)other else other$	Match
4	(E)SL$	(0)if(1)other else other$	Match
5	E)SL$	0)if(1)other else other$	E→0
6	0)SL$	0)if(1)other else other$	Match
7	)SL$	)if(1)other else other$	Match
8	SL$	if(1)other else other$	S→I
9	IL$	if(1)other else other$	I→if(E)SL
10	if(E)SLL$	if(1)other else other$	Match

52

Steps	Parsing Stack	Input	Action
11	(E)SLL$	(1)other else other$	Match
12	E)SLL$	1)other else other$	E→1
13	1)SLL$	1)other else other$	Match
14	)SLL$	)other else other$	match
15	SLL$	other else other$	S→other
16	otherLL$	other else other$	match
17	LL$	else other$	L→else S
18	elseSL$	else other$	Match
19	SL$	other$	S→other
20	otherL$	other$	match
21	L$	$	L→ ε
22	$	$	accept

53

课堂练习

Consider the grammar:

S → **aBc**| **aAB**
A → **aAb** | **a**
B → **b**|ε

(a) Left factor this grammar.
(b) Construct First and Follow sets for the non-terminals of the resulting grammar;
(d) Construct the LL(1) parsing table for the resulting grammar.
(c) Show the actions of the corresponding LL(1) parser, given the input string **aaabb**.

54