1401 / 1 / 15

LL(1) Grammer : یک گرامر LL(1) است که در آن تجزیه فقط با یک non-terminal و یک input چک کنیم

ex: $S ::= aSB \mid dB$
$B ::= bB \mid a$
} → LL(1)

Date : 1401 / 1 / 15

statement : if statement :ex

| while_statement

| switch_statement

| call_statement

| Assignment_statement

;

#beta :

#beta_1

| beta_2

| beta_4

| beta_3

| beta_4

;

if foreach beta_i and beta_j and i not equal to j intersection(beta_i,beta_j)

= Null ⇒ the grammer is LL(1)

First(statement) = First(if_statement) + First(while_statement) +

First(switch_statement) + First(call_statement) + First(Assignment_statement)

if_statement : if '(' condition ')' then statement elsepart

while_statement : while '(' condition ')' statement

switch_statement : switch expression case parts

call_statement : identifier '(' parameters ')'

assignment_Statement : identifier ':=' expression

→ Not LL(1)

iman

هر گرامر غیر مبهم را می‌توان به LL(1) تبدیل کرد.

عمل فاکتورگیری از چپ

$A \rightarrow A' \mid \gamma$
$A' \rightarrow \alpha \mid \beta$
$\} \rightarrow A \rightarrow a(\alpha \mid \beta) \mid \gamma$

: Nullable

if_st → if condition then st else part

elsepart → else st | λ

Nullable ممکن است باشد یا نباشد

ex: A → Bbed
B → bd | λ
→ A → bdbed | bed → LL(1) ×

ex: پذیرایی → پیش غذا | غذا | دسر | کیک | λ

پیش غذا → سوپ | آش | کباب | λ

دسر → میوه | بستنی | ترافل | λ

کیک → جعبان | چوب

Date : 1401 / 1 / 22

LL(1) بررسی

$A \to P\ G\ D\ K$

$P \to s \mid a \mid m \mid \lambda$

$G \to a \mid k \mid c \mid \lambda$

$K \to o \mid b$

First(A) = First(B) = {s, a, m}

Follow(P) = First(G) = {a, k, c}

First(G) = {c, k, a} + Follow(G)

Follow(G) = First(K) = {o, b}

→ LL(1) ✗

برای اینکه گرامر nullable را به LL(1) تبدیل کنیم، باید λ را حذف کنیم.

To be LL(1) for any production: $Y \to \alpha\beta\gamma \wedge \alpha \to^{*} \lambda$

First(α) ∩ First(β) should be empty

ex: Bbed ← A

bd | λ ← B

→ First(A) ∩ First(B) = {b}

$A \to bed \mid bdbed$ ⟶ (LL(1) نیست) $A \to bA'$

$A' \to ed \mid dbed$ BNF

$A \to b(ed \mid dbed)$ EBNF

ex: $A \to Bde \mid B$

$C \to Bbe \mid \lambda$ → LL(1) ✗

$B \to a \mid d \mid \lambda$

Date : 1401 / 1 / 22

LL(1) پارسر

۲۷۸۱۳۶۲۹۴

D → cAd

A → eBF | eB

F → λ | fg

F → fg

B → gh | a

ex: S → aBd$

B → de | λ

→ since B is nullable ⇒ First(B) ∩ Follow(B) = ∅

⇒ {d} ∩ {d} = {d} ≠ ∅

S → ad$ | aBd$
B → de

→ حذف ε

S → aS'
S' → d$ | Bd$

→ Left Refactoring

S' → d$ | de$
S' → ds''

→ Left Refactoring

S'' → $ | e$

1401, 1, 27

'identifier' + '(' → عبارت فراخوانی

'identifier' + '=' → عبارت تخصیص (Assignment statement)

* در حالت Left recursive بودن، پارسر LL(1) نیست.

ex. A → Aα | β First(Aα) ∩ First(β) = First(β) ≠ ∅

First(Aα) = First(A) ⊇ First(β)

Date : 1401/1/27

درس (7) LL

حالات مختلف LL(1) نبودن

1. گذرش های مختلف با یک ترم یا پایان مشترک آغاز می شوند.

ex: $A \rightarrow a\alpha \mid a\beta \mid \gamma$ ⇒ Left factoring

$A \rightarrow aA' \mid \gamma$

$A' \rightarrow \alpha \mid \beta$

2. Nullable بودن

ex: $A \rightarrow B\alpha \mid \gamma$ ⇒ Null elimination

$B \rightarrow \lambda \mid \alpha\delta$

$A \rightarrow B\alpha \mid \alpha \mid \gamma$

$B \rightarrow \alpha\delta$

3. Left recursion داشتن

ex: $A \rightarrow A\alpha \mid \beta$ ⇒ جایگزین کردن با گرامر معادل

در اینجا ها بلوپرنت را مشاهده می کنید، در این حالت با گرامری معادل، جایگزین می کنیم.

در این مثال میاریم گرامری داریم که با یک β شروع شده و تعداد '*' از α دارد.

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \mid \lambda$

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$F \rightarrow id \mid No \mid (E)$

$E \rightarrow EE' \mid T$

$E' \rightarrow +T \mid -T$

$T \rightarrow TT' \mid F$

$T' \rightarrow *F \mid /F$

$E \rightarrow TE''$

$E'' \rightarrow E'E'' \mid \lambda$

$T \rightarrow FT''$

$T'' \rightarrow T'T'' \mid \lambda$

ex1: $S \rightarrow dAB \mid BaB$ ⇝ $Follow(B) = \$$

$A \rightarrow dA \mid Ba$ ⇝ $Follow(B) = a$

} ⇒ $Follow(B) = \{a, \$\}$

$B \rightarrow bB \mid \lambda$ $\quad First(B) = b$

$First(B) \cap Follow(B) = \emptyset$

«ادامه صفحه بعد»

ex2: $S \rightarrow dAB \mid BaBb$

$A \rightarrow dA \mid Ba$ ⇒ 2 روش

$B \rightarrow bB \mid \lambda$

$S \rightarrow dA \mid dAB \mid a\lambda b \mid BaBb \mid a\lambda b \mid Bab$

$A \rightarrow dA \mid a\lambda \mid Ba$

$B \rightarrow bB \mid b$

«ادامه صفحه بعد»

Date : 1401 / 1 / 29

$S \to dS_1 \mid aS_2 \mid BaS_2$: ex2

$S_1 \to \lambda \mid B$

$S_2 \to b \mid Bb$

$S_3 \to Bb \mid b$

$First(dAB) \cap First(BaB) = \phi$
$First(dA) \cap First(Ba) = \phi$
$\Big\} \to LL(1)$: ex1

1401 / 2 / 10

تولید تابع تحلیلگر نحوی (روش بالا به پایین)

1. تبدیل گرامر به فرم LL(1)
   - ایجاد مجموعه های First , Fallow
   - بررسی اینکه اشتراک مجموعه First برای ترمینال های هر ترم صفر باشد
     - Left Refactoring
     - Left Recursion Reversal
     - Omit nullables
   - ایجاد مجموعه های First , Fallow
   - تولید جداول تجزیه LL(1)

Recursive descent parsers

برای هر ترم میانی، یک تابع به اسم ترم میانی ایجاد می کند.

Date : 1401 / 2 / 10

روش کاهنده بازگشتی

1. تبدیل گرامر به فرم LL(1) (توسعه یافته)
2. نوشتن یک تابع برای تشخیص هر ترم میانی (غیر بازگشتی)
3. نوشتن تابعی به نام Expect برای ترم های پایانی

$E \to E+T \mid E-T \mid T$ $E \to E(+T \mid -T) \mid T$ $E \to T\{+T,-T\}$

$T \to T*F \mid T/F \mid F$ $T \to T(*F \mid /F) \mid F$ $T \to F\{*F \mid /F\}$

$F \to id \mid NO \mid (E)$

```
Typedef enum Symbols={S_id, S_NO, S_add, S_if,}
void main(){
    Symbols CurrentSymbol;
    CurrentSymbol=NextSymbol;
    E()...
    }
(2) /* T → F{*F | /T} */
void T(){
    F();
    while(CurrentSymbol=S_Mul ||
        CurrentSymbol=S_div){
        NextSymbol();
        F();
    }
(1) /* E → E(+T | -T) | T */
Void E(){
    T()
    While(CurrentSymbol=S_Add ||
        CurrentSymbol=S_minus){
        NextSymbol();
        //get NextSymbol is
        CurrentSymbol
        T();
    }
```

Date : 1401 / 2 / 10

(3) /* F → id | NO | (E) */

```
void F(){
    if(CurrentSymbol == S_id){
        NextSymbol();
    }
    else if(CurrentSymbol == S_no)
        NextSymbol();
    }
    else if(CurrentH Symbol == S_OpenPar){
        NextSymbol();
        E();
        Expect(S_closePar);
        {
    else{
        SyntaxError();
    }
```

```
// void Expect(Symbol Ex){
if (CurrentSymbol == Ex){
    NextSymbol();
}
else{
    SyntaxError();
}
```

ex: a + (3-b) * 2

⇓

currentSymal = S_id

E() ⇒ T() ⇒ F() = (detects S_id is)

currentSymbol = '+'

↵

currentSymbol = S_openPar

↳ T ⇒ F (CurentSymbol = 3)

↳ E ⇒ T ⇒ F → currentSymbol = S_minus

Date : 1401 / 2 / 10

→ F ⇒ T ⇒ E → current Symbal (=b) → T ⇒ F=(current Symbal

→ E ⇒ T ⇒ F() { Nextsymbal = S-close Par }

تجزیه گرهای کاهنده بازگشتی

مراحل : 1. تبدیل گرامر به فرم LL(1) توسعه یافته

2. نوشتن یک تابع به ازای هر ترم میانی برای تشخیص آن ترم میانی

3. نوشتن تابعی به نام Expect برای تشخیص ترم های پایانی