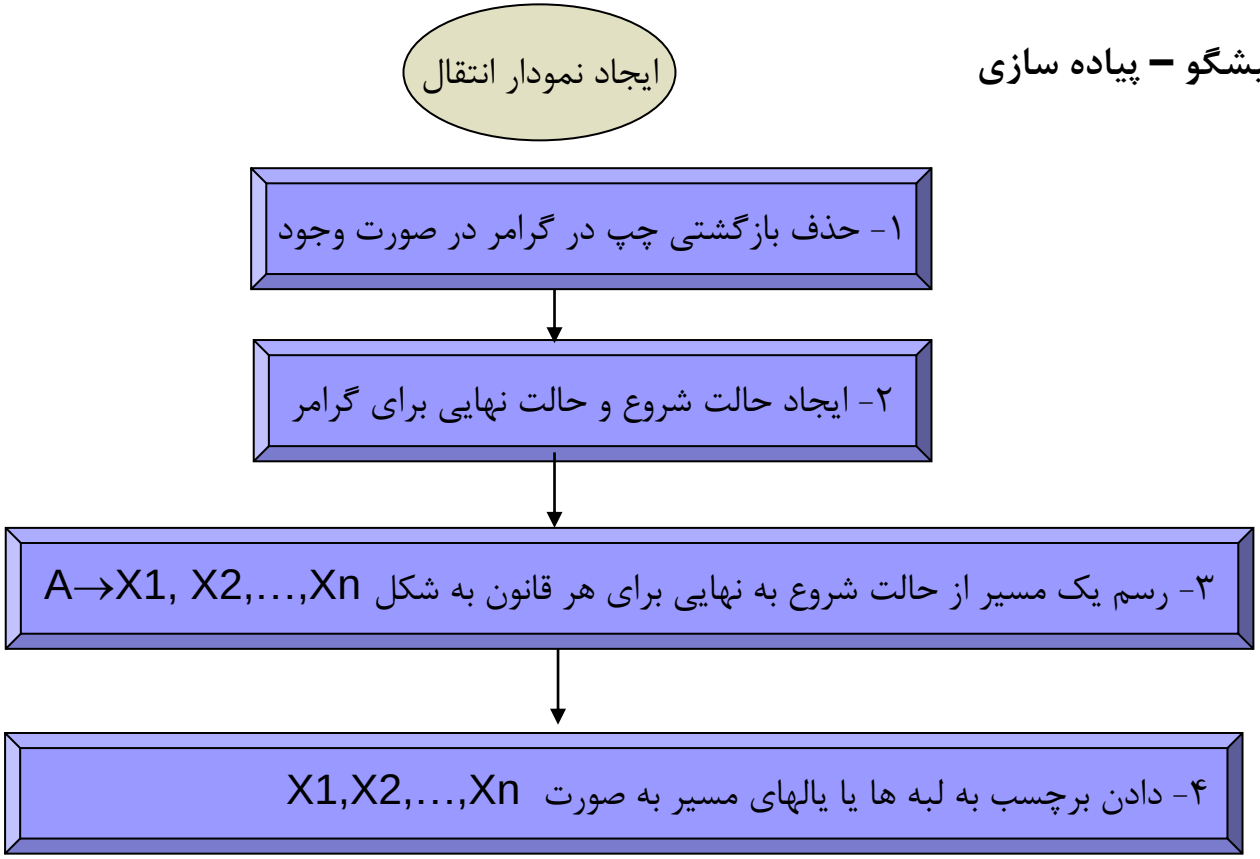
تجزیه بالا به پایین پیشگو

روش تجزیه بالا به پایین با نگاه به نماد پیش نگر در مورد استفاده از هر قانون درتصمیم گیری

نماد پیش نگر

مجموعه تمام پایانه هایی است که در قوانین مربوط به غیر پایانه در سمت چپ قرار می گیرند.

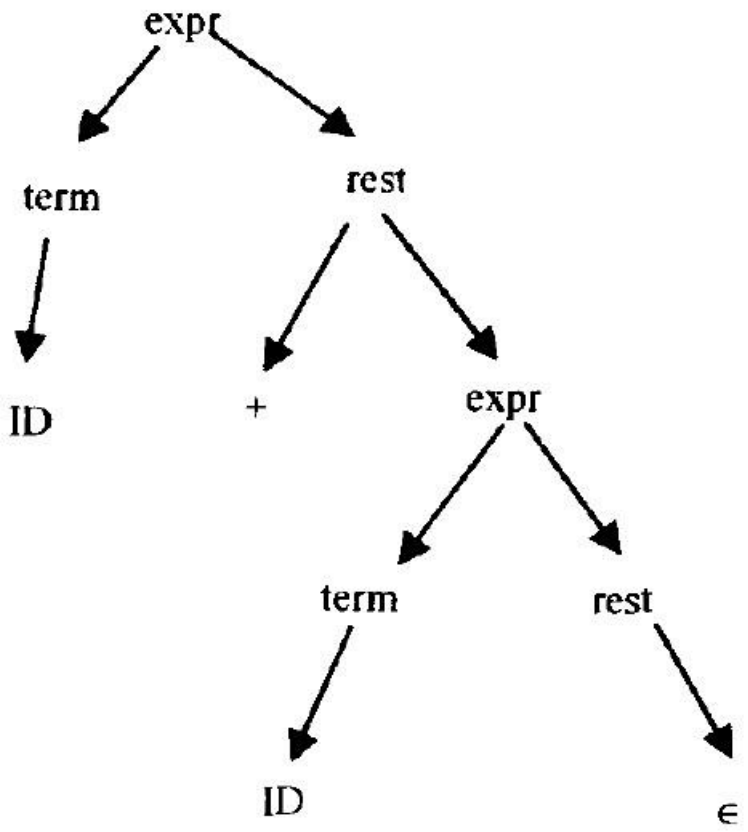
تجزیه کننده پیشگو – پیاده سازی



expr→ term rest
term→ ID
rest →'+' expr | '-' expr | ∈

id+id رشته ورودی

```
void   match(char symbol){
if ( lookahead== symbol)
     lookahead=getche();
else{
     cout<<" error";
    exit(0);
    }
return;
}
```



ورودی ۱+۲

expr→ term rest
term→ 1 | 2 | 3
rest→'+' expr | '-' expr | '*' expr | '/' expr | ∈

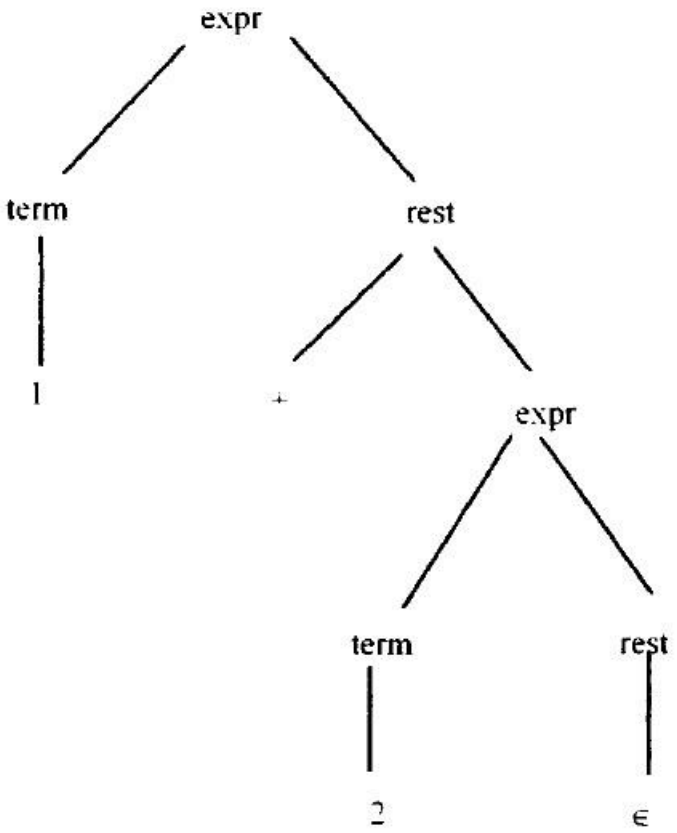
```
void term(){

if (lookahead=='1')

    match('1');

else if (lookahead=='2')

    match('2');

else if (lookahead=='3')

    match('3');

else {cout<<error";
       exit(0);
       }
}
```

```
void expr(){
   term();
   rest();
return;
```

```
int main(){
lookahead= getche();
expr();
cout<<"accepted";
return 0;
}
```

```
void rest(){

if (lookahead=='+'){
    match('+');
    expr();
}

else if (lookahead =='-'){
          match('-');
          expr();
}

else if (lookahead =='*'){
          match('*');
          expr();
}

else if (lookahead =='/'){
          match('/');
          expr();
}

else;
return;
}
```

ورودی ۱+۲



exp→ expr + term | term
term → 1 | 2 | 3

ورودی ۲+۱

```
char   lookahead;
expr(){
        expr();
        match('+');
        term();
        }

void term(){
if (lookahead=='1')
    match('1');

else if (lookahead=='2')
    match('2');
else if (lookahead=='3')
    match('3');
return 0;
}

int main(){
    lookahead=getche();
    expr();
    return 0;
}
```

$A \rightarrow a\ A | a\ B$
$B \rightarrow b\ B | c$

$first(aA) = \{a\}$
$first(aB) = \{a\}$

$A \rightarrow aR$
$R \rightarrow A | B$
$B \rightarrow bB | c$

$first(A) = \{a\}$
$first(B) = \{b, c\}$

قانون کلی اگر قاعده تولید A را به صورت ذیل در نظر بگیریم.

$A \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n$

اگر رابطه ذیل حداقل بین دو α_i و α_j به طوریکه $i \neq j$ برقرار باشد برخورد first / first رخ می‌دهد.

$first(\alpha_i) \cap first(\alpha_j) \neq \varnothing$

A→ aR
R→ A|B
B→ bB|c

```
void A(){
    match('a');
    R();
}

void R(){

if(lookahead=='a')
        A();

else if (lookahead=='b' || lookahead=='c')
        B();

else{ cout<"error";
        exit(0);
    }
}
```

```
void B(){

if(lookahead=='b'){
        match('b');
        B();
}

else if(lookahead=='c'){
match('c');
cout<<"Accepted";

 exit(0);
    }
 }

int main(){
    lookahead= getche();
    A();
    return 0;
}
```

با توجه به تعریف follow ، مشکل زمانی بروز می‌کند که در قاعده تولیدی به $A\rightarrow\alpha|\beta$ شرایط ذیل برقرار باشد.

۱- β بتواند $\in$ را تولید کند.

۲- حداقل یک نماد وجود دارد که هم می‌تواند در شروع α و هم بعد از A باشد. یا به عبارت دیگر رابطه ذیل برقرار باشد.

$first(\alpha) \cap follow(A)\neq\varnothing$

در این شرایط برخورد first/follow رخ می‌دهد.

$A\rightarrow Bed$

$B \rightarrow e|a| \in$

$first(e)=\{e\}$

$follow(B)=\{e\}$

$A\rightarrow eed \mid aed \mid ed$

$A\rightarrow eR|aed$

$R\rightarrow ed|d$

```
char  lookahead;
void A(){
    B();
    match('e');
    match('d');
}

void B(){
if (lookahead =='e'){
      match('e');
else  if (lookahead =='a')
      match('a');
else ;
return ;
}
int main(){
    lookahead=getche();
    A();
    return 0;
}
```

$A \rightarrow B \mid C$
$B \rightarrow bB \mid f$
$C \rightarrow cC \mid e$

	b	f	c	e
A	A → B	A → B	A→C	A→C
B	B→ bB	B→f		
C			C→cC	C→e

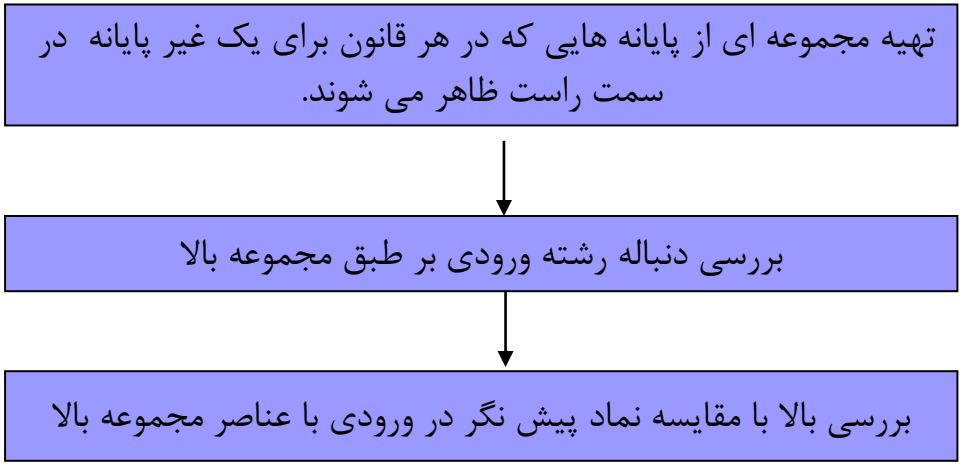
```
void  A(){

if ( lookahead=='b' || lookahead=='f')
      B();

else if ( lookahead=='c' || lookahead=='e')
      C();
}
else { cout<<"error";
       exit(0);
}
return;
}
```

```
void B(){

if (lookahead=='b'){
    match('b');
    B();
}

else if (lookahead=='f')
        match('f');
else {
       cout<<"error";
       exit(0);
     }
}
```

```
void C(){

if (lookahead=='c'){
    match('c');
    C();
}

else if (lookahead=='e')
        match('e');
else {
       cout<<"error";
       exit(0);
     }
}
```

تجزیه کننده غیر بازگشتی LL1

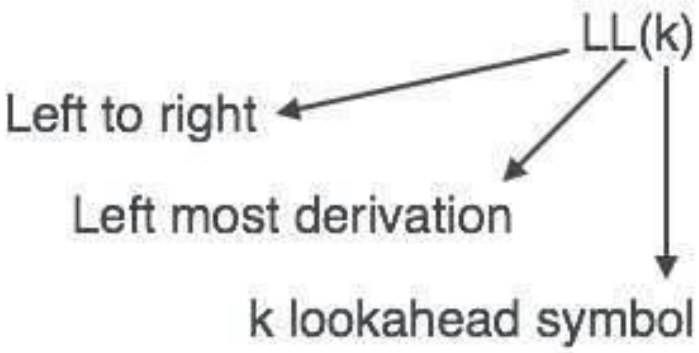
تجزیه کاهشی بازگشتی یک روش تجزیه **بالا به پایین** است که درخت تجزیه را از بالا ساخته و ورودی از **چپ به راست** خوانده می شود. این روش ها برای هر موجودیت پایانی و غیر پایانی استفاده می شود. این تکنیک تجزیه به صورت بازگشتی ورودی را تجزیه می کند تا یک درخت تجریه ایجاد کند، که ممکن است نیاز به پس گرد back-trackingداشته باشد. اما گرامر مرتبط با آن (اگر فاکتورگیری چپ نشده باشد) نمی‌تواند از پس گرد جلوگیری کند. نوعی از تجزیه پایین گرد که نیازی به پس گرد ندارد، به عنوان **تجزیه پیشگو** predictiveشناخته می شود. این تکنیک تجزیه به عنوان بازگشتی در نظر گرفته می شود، زیرا از گرامر مستقل از متن استفاده می کند که ماهیتی بازگشتی است.



تجزیه‌کننده LL

تجزیه‌کننده LLگرامر LLرا می‌پذیرد. گرامر LLزیرمجموعه‌ای از گرامر مستقل از متن است؛ اما برخی محدودیت‌ها بر آن اعمال شده تا پیاده‌سازی ساده‌تری داشته باشد و می‌توان گفت نسخه ساده‌شده‌ای از CFGمحسوب می‌شود. گرامر LLرا می‌توان به کمک هر دو الگوریتم پایین‌گرد و مطابق جدول پیاده‌سازی کرد.

تجزیه‌کننده LLبه صورت LL(k) نمایش داده می‌شود. حرف Lنخست در این عبارت به معنی تجزیه از ورودی به صورت چپ به راست left to rightو حرف Lدوم به معنی اشتقاق چپ‌ترین left-most derivation است. حرف k نیز نشان دهنده تعداد حروفی است که رو به جلو بررسی می‌شوند. به طور کلی k=1فرض می‌شود بنابراین LL(k) را می‌توان به صورت LL(1) نیز نوشت. البته روش LL(1)قابل تعمیم به LL(2)، LL(3)تا LL(k)نیز هست، ولی کاربرد کمتری دارد.

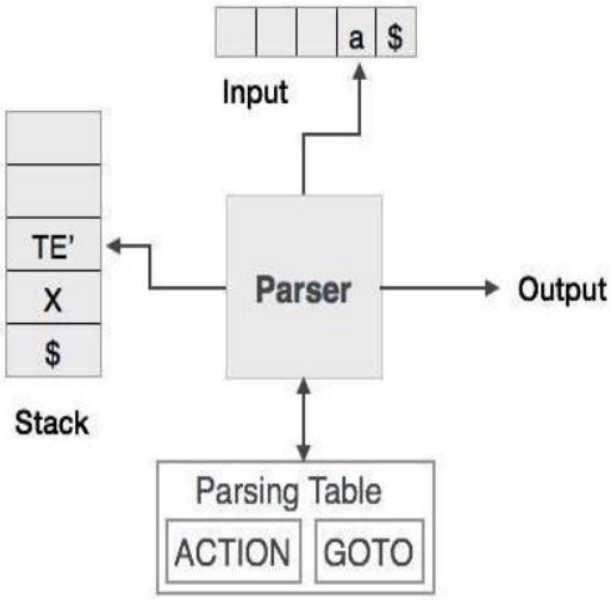


پارسر پیشگو Predictive Parser

تجزیه گر پیشگو یک تجزیه کننده کاهشی بازگشتی یا پایین گرد Recursive descent parserاست، که قابلیت پیشگویی این که کدام ترکیب برای استفاده با رشته ورودی جایگزین خواهد شد. تجزیه کننده پیشگو از پس گردی backtrackingرنج نمی برد.

برای انجام وظایف خود، تجزیه گر پیشگو از یک اشاره گر رو به جلو look-aheadاستفاده می‌کند، که به نمادهای ورودی بعدی اشاره دارد. برای این که تجزیه کننده دارای پس گردی نباشد، تجزیه گر پیشگو برخی محدودیت‌ها را بر روی گرامر قرار می‌دهد و فقط یک کلاس از گرامر معروف به (k) LL گرامر را می پذیرد.

تجزیه پیشگو از یک پشته و یک جدول تجزیه برای تجزیه ورودی و تولید یک درخت تجزیه استفاده می‌کند. هر دو پشته و ورودی حاوی نماد پایانی $ هستند که نشان می دهد پشته خالی می باشد و ورودی مصرف شده است. تجزیه کننده برای تصمیم گیری در مورد ترکیب عنصر ورودی و پشته ، از جدول تجزیه استفاده می‌کند.



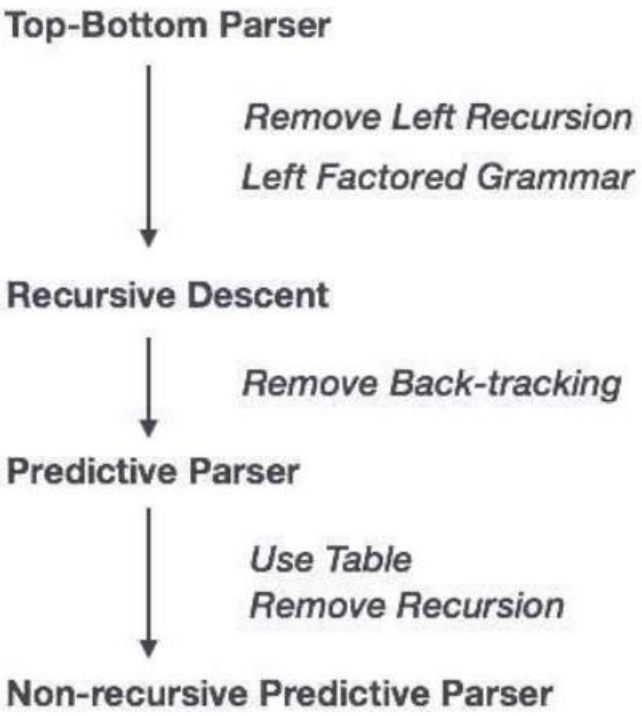
میانگیر ورودی حاوی رشته ورودی برای تجزیه با علامت $ در انتهای آن

پشته دنباله ای از نمادهای گرامر در هر لحظه برای تجزیه با $ در ته آن

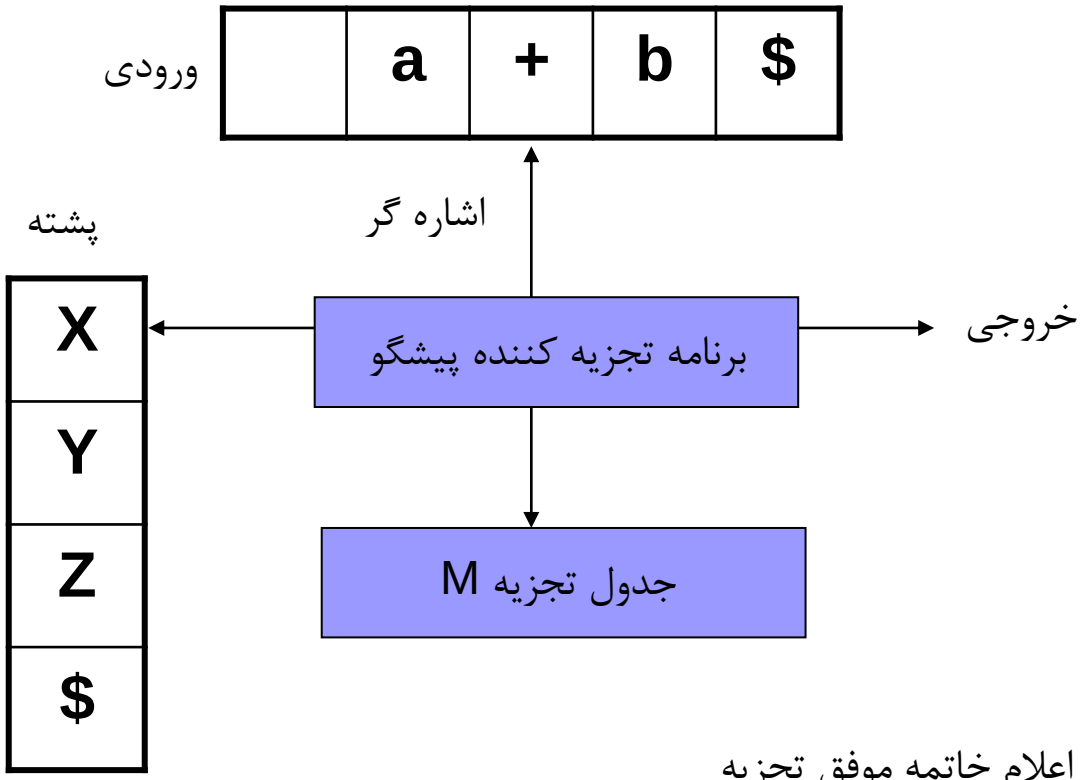
جدول تجزیه A، آرایه دو بعدی عناصر[A,a] ، Aیک غیر پایانه و a یک پایانه

دنباله خروجی دنباله تجزیه شده تا آن زمان

در تجزیه کاهشی بازگشتی یا پایین گرد Recursive descent parser، تجزیه کننده ممکن است بیش از انتخاب در مورد یک ورودی منفرد داشته باشد، در حالی که در تجزیه کننده پیشگو، در هر مرحله حداکثر یک ترکیب را می توان انتخاب نمود. ممکن است مواردی وجود داشته باشد که هیچ ترکیبی مطابق با رشته ورودی وجود نداشته باشد، و این امر باعث می شود که رویه تجزیه fail یا شکست بخورد.



تجزیه کننده پیشگوی غیر بازگشتی



۱- اگر $ = a =X باشد توقف تجزیه کننده اعلام خاتمه موفق تجزیه

۲- اگر $ ≠ a =X خروج X از پشته، انتقال اشاره‌گر ورودی به نماد بعدی در ورودی

۳-اگر x یک غیر پایانه باشد مراجعه به وارده در جدول [A , a]و جایگزینی قانون مناسب آن از جدول

۱- انتقال پیشگو: اگر بالای پشته غیر پایانه X و نماد جاری رشته ورودی a باشد، X از پشته حذف شده و قاعده تولید M[X,a] از جدول استخراج شده و به صورت معکوس در پشته درج می‌شود. به عنوان مثال اگر $M[X,a]=X\rightarrow\alpha\beta\gamma$ باشد، α در بالای پشته قرار می‌گیرد. در واقع با این عمل مانند تجزیه پیشگو، غیرپایانه گسترش می‌یابد. اگر M[X,a] خالی باشد خطا رخ داده است.

۲- انتقال تطبیق: اگر نماد بالای پشته، پایانه باشد و این پایانه با نشانه جاری رشته ورودی یکسان باشد، آنگاه پایانه بالای پشته حذف می‌گردد و نشانه بعدی در رشته ورودی به عنوان نشانه جاری در نظر گرفته می‌شود، ولی اگر نشانه بالای پشته و نشانه جاری رشته ورودی یکسان نباشد خطا رخ داده است.

تجزیه کننده دو انتقال تطبیق و پیشگو را دائما انجام می‌دهد. تجزیه وقتی تمام می‌شود که پشته و رشته ورودی خالی شده باشد.

ساخت جدول تجزیه پیشگوی غیر بازگشتی

expr → term rest
rest → + expr | - expr | ∈
term → id

first(term rest)=first(term)={id}
first(+expr)=first(+)={+}
first(-expr)=first(-)={-}
first(∈)={∈}
first(id)={id}
follow(expr)={$}
follow(term)={+,-,$}
follow(rest)={$}

	id	+	-	$
expr	expr→term rest			
term	term→id			
rest		rest→+expr	rest→ - expr	rest→∈

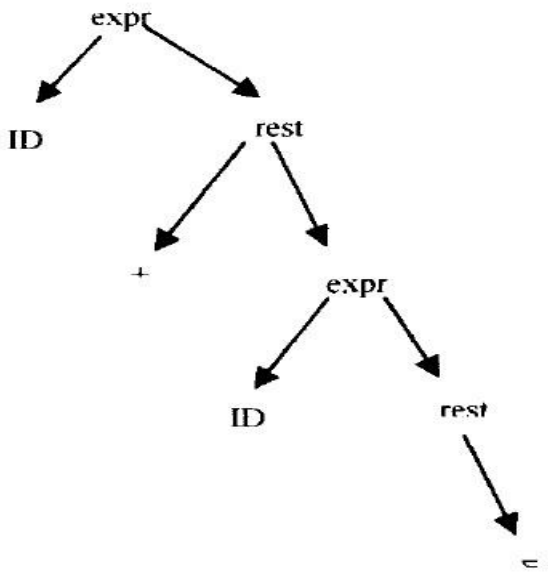
مراحل ساخت جدول تجزیه را می‌توان برای هر قاعده تولید $A\rightarrow\alpha$ در سه مرحله ذیل خلاصه کرد.

۱- برای هر پایانه a (به جز ϵ) در مجموعه first(α) قانون $A\rightarrow\alpha$ به جدول در M[A,a] اضافه می‌گردد.

۲- اگر ϵ در first(α) باشد برای هر پایانه b در follow(A)، $M[A,b]=A\rightarrow\epsilon$ می‌گردد.

۳- در خانه‌های خالی جدول، error قرار می‌دهیم.

expr→ ID rest
rest→'+' expr | '-' expr | ϵ



$A \rightarrow aB \mid aad$
$B \rightarrow bB \mid c$

با توجه به:

$first(aB)=\{a\}$
$first(aad)=\{a\}$

در نتیجه:

$first(aB) \cap first(aad)=\{a\} \neq \emptyset$

برخورد firts/first رخ داده است و بنابراین گرامر LL(1) نیست.

$A \rightarrow CB \mid \epsilon$
$B \rightarrow bB \mid \epsilon$
$C \rightarrow cC \mid \epsilon$

در $A \rightarrow CB \mid \epsilon$:

$\alpha=CB$
$\beta=\epsilon$

در قاعده تولید $A \rightarrow CB \mid \epsilon$ دو انتخاب CB و ϵ هر دو ϵ را تولید می‌کنند. در نتیجه گرامر LL(1) نیست.

– اگر در گرامر، قاعده تولیدی به صورت $A \rightarrow \alpha \mid \beta$ وجود داشته باشد به طوریکه α و β هر دو رشته تهی را تولد کنند، گرامر LL(1) نیست.

$S \rightarrow Aab$

$A \rightarrow a \mid \epsilon$

در قاعده تولید $A \rightarrow a|\epsilon$ ، $\alpha=a$ و $\beta=\epsilon$ است. با توجه به $S \rightarrow Aab$ ، پایانه a بعد از A قرار دارد در نتیجه $follow(A)=\{a\}$ است. با توجه به $A \rightarrow a|\epsilon$، $first(a)=\{a\}$ است، در نتیجه:

$first(a) \cap follow(A)=\{a\} \neq \varnothing$

برخورد first/follow رخ می‌دهد و در نتیجه گرامر LL(1) نیست.

- اگر در گرامر، قاعده تولیدی به صورت $A \rightarrow \alpha|\beta$ وجود داشته باشد به طوریکه β بتواند رشته تهی را تولید کند و نمادی در مجموعه‌های follow(A) و $first(\alpha)$ مشترک باشد، برخورد first/follow رخ می‌دهد در نتیجه گرامر LL(1) نیست. به عبارتی اگر رابطه ذیل برقرار باشد گرامر LL(1) نیست.

$first(\beta) \cap follow(A) \neq \varnothing$

$A \rightarrow aB \mid aad$
$B \rightarrow bB \mid c$

گرامر ذیل LL(1) نیست.

با استفاده از فاکتورگیری چپ گرامر فوق به گرامر ذیل تبدیل می‌گردد.

$A \rightarrow aR$
$R \rightarrow B \mid ad$
$B \rightarrow bB \mid c$

گرامر جدید دارای برخورد first/first نیست.

گرامرهایی وجود دارند که با هیچ تغییری نمی‌توان آنها را به LL(1) تبدیل نمود. اگر برای تعیین قاعده تولید بعدی نیاز به بررسی K نماد بعدی از ورودی باشد گرامر از نوع LL(k) است.

$A \rightarrow aCbAB \mid d$
$B \rightarrow eA \mid \epsilon$
$C \rightarrow c$

غیر پایانه	نماد ورودی					
	d	**c**	**e**	**a**	**b**	**$**
A	$A \rightarrow d$			$A \rightarrow aCbAB$		
B			$B \rightarrow \epsilon$ $B \rightarrow eA$			$B \rightarrow \epsilon$
C		$C \rightarrow c$				

$S \rightarrow Aa$
$S \rightarrow Bb$
$A \rightarrow \epsilon$
$B \rightarrow \epsilon$
$A \rightarrow cAb$
$B \rightarrow dAa$

غیر پایانه	نماد ورودی				
	a	b	c	d	$
S	$S \rightarrow Aa$	$S \rightarrow Bb$	$S \rightarrow Aa$	$S \rightarrow Bb$	
A	$A \rightarrow \epsilon$	$A \rightarrow \epsilon$	$A \rightarrow cAb$		
B		$B \rightarrow \epsilon$		$B \rightarrow dAa$	

$S \rightarrow aACb$
$A \rightarrow b \mid \epsilon$
$C \rightarrow cC \mid \epsilon$

غیر پایانه	نماد ورودی			
	a	b	c	$
S	$S \rightarrow aACb$			
A		$A \rightarrow b$ $A \rightarrow \epsilon$	$A \rightarrow \epsilon$	
B		$C \rightarrow \epsilon$	$C \rightarrow cC$	

با توجه به جدول، چون M[A,b] دارای دو قاعده تولید است بنابراین گرامر LL(1) نیست.