فصل چهارم

تجزيه بالا به پائين

٤.١ تحليلگر ذهن

عمل تجزيه بالا به پائين بر مبناي روش تفكر مغز آدمي براي تحليل نحوي جملات استوار است. فردي فارسي زبان در مقابل شما قرار ميگيرد . ميخواهد با شما صحبت كند. به دهان وي مينگريد. پيش بيني مي كنيد كه جمله فارسي ميخواهد بيان كند. جمله فارسي سرترم جملات در دستورالعمل زبان فارسي است زيرا ، هر گفته اي در زبان فارسي بايد جمله فارسي باشد. بنابراين مغز پيش بيني ميكند كه جمله خارج شده از دهان سرترم گرامر جملات در زبان فارسي بايد باشد.

حالا فرض كنيد كه فرد كلمه ،اگر، از دهانش خارج شود. بلافاصله تحليلگر نحوي مغز به سراغ مجموعه لغات يا در اصطلاح ترمهاي پايانی اي ميرود كه مي توانند جملات فارسي را آغاز كنند. كلمه " گر" ميتواند آغاز كننده يك جمله فارسي باشد. در اصطلاح مجموعه ترم هاي پايانی كه ميتوانند آغاز كننده يك ترم باشند را مجموعه سرآغاز يا مجموعه First براي آن ترم گويند.

پس از اطمينان از اينكه لغت ،اگر، در مجموعه سرآغاز يا First جمله فارسي است بايد مشخص نمود كه كدام گسترش از گسترش‌هاي متفاوت جمله فارسي با كلمه ،گر، آغاز ميشود. پاسخ جملات سوألي است. كلمه اگر متعلق به مجموعه سرآغاز يا First جملات سوألي است. حالا طبق گرامر جملات سوألي پس از كلمه ،اگر، انتظار شنيدن يك شرط ميرود. باز هم بر طبق گذشته تحليلگر نحوي ذهن كلمه بعدي را گرفته در مجموعه سرآغاز شرط آنرا جستجو ميكند. و به اين ترتيب مراحل ادامه مي يابد.

٤.٢ ايجاد الگوريتم تحليل نحوی بر مبنای عملكرد ذهن

بطور خلاصه در بخش قبل ملاحظه نموديد كه براي انجام عمل تحليل نحوي مغز به صورت زير عمل مي نمايد :

1. انتظار سرترم گرامر زبان را در آغاز دارد. سرترم گرامر زبان فارسي جمله فارسي است.

2.لغت اولي دريافت ميشود . در بخش قبل در حاليكه انتظار جمله فارسي را تحليلگر ذهن مي داشت لغت "گر" دريافت شد.
3.در داخل مجموعه First براي ترم مورد انتظار بدنبال لغت دريافت شده ميگردد.
4.در صورت يافتن لغت ، در داخل مجموعه First براي گسترش‌هاي متفاوت ترم مياني مورد انتظار مي گردد تا مشخص كند كدام گسترش با لغت دريافت شده آغاز ميشود (در مثال فوق جمله شرطي با لغت گر، آغاز مي شد).
5. حالا با مشاهده ترم پايانی دريافت شده در گرامر ، طبق گرامر ترم مورد پيش بيني را مشخص ميكند. در مثال فوق با يافتن كلمه گر، طبق گرامر انتظار مشاهده يك شرط مي رفت. لذا ، لغت بعدي دريافت شده و در صورت عدم خاتمه جمله از مرحله 2 عمليات تكرار ميشود.

براي نمونه به گرامر زير توجه كنيد :

Statement → IfSt | WhileSt | ForSt | CaseSt | CompoundSt | AssignmentSt | CallSt

IfST → IF Expression THEN Statement
ElsePart

ElsePart → ELSE Statement |
∈

WhileSt → WHILE Expression DO
Statement

ForSt → FOR Variable := Expression TO Expression DO Statement

CaseSt → CASE Expression OF CaseParts
END

CaseParts → CaseLabels : Statement
OtherParts

OtherParts → ; CaseParts |
∈

CaseLabels → Constant
Constants

Constants → , CaseLabels |
∈

Constant → String | Identifier |
Number

CompoundSt → BEGIN Statements
END

Statements → Statement
OtherSts

OtherSts → ; Statements | ∈

AssignmentSt → Id :=
Expression

CallSt → Id '(' ActualParams

')'

ActualParams → Expression

OtherParams

OtherParams → , OtherParams | ∈

Expression → Id |

No

براي تجزيه جمله

If Then C := 1

A

بر اساس عملکرد ذهن بصورت زیر عمل میشود.

- بر اساس گرامر در ابتدا انتظار مشاهده سرترم گرامر يعني Statement ميرود.
- اولين لغت يعني If از تحليلگر لغوي دريافت ميشود.
- تحليلگر نحوي به جستجوي لغت If داخل مجموعه لغات آغاز كنده ترم مورد انتظار يعني

First(statement) = First(IfSt) + First(WhileSt) + First(ForSt) + First(CaseSt) +
First(CompoundSt) + First(AssignmentSt) + First(CallSt)
= [S_If, S_While, S_For, S_Case, S_Begin, S_Id]

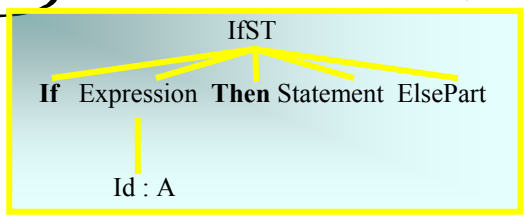
- با مشاهده نوع لغت If در مجموعه First(statement) تحليلگر لغوي بدرون مجموعه سرآغاز براي گسترش‌هاي متفاوت Statement مينگرد و لغت if را در مجموعه First(IfSt) پيدا ميكند. بنابراين تحليلگرتشخيص ميدهد كه جمله مورد نظر يك جمله شرطي If است. درخت زير بر اساس گسترش موجود براي جمله IfSt بر طبق گرامر ، ايجاد ميشود :

- حالا پس از مشاهده If ، بنابر گرامر جمله شرطي IfSt در ورودي انتظار مشاهده Expressiom ميرود. از تحليلگر لغوي لغت بعدي يعني A دريافت ميشود. درون مجموعه سرآغاز عبارات يعني :

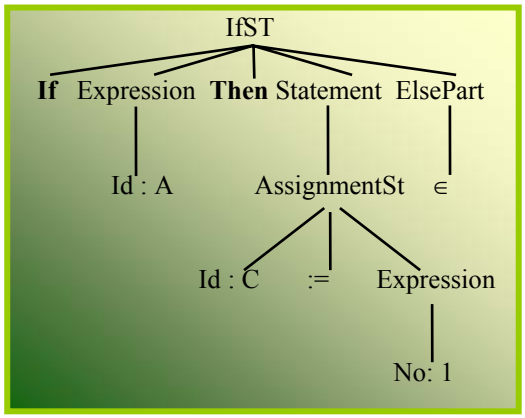
First(Expression) = [S_Id]

نوع لغت A يعني S_Id وجود دارد. بنابراين تحليلگر نحوي به كار خود ادامه ميدهد.

- حالا طبق گرامر انتظار ديدن Then در ورودي ميرود. تحليلگر لغوي لغت بعد از A را از ورودي ميخواند . اين لغت همانگونه كه طبق گرامر انتظار مي رفت لغت Then ميباشد. بنابراين تا اين لحظه درخت تجزيه زير توسط تحليلگر نحوي بنا شده است:



- حالا تحليلگر پيش بيني مشاهده يك Statement را بر اساس گرامر مينمايد. از تحليلگر لعوي لغت بعدي را تحليلگر نحوي دريافت ميكند. لغت C از نوع S_Id را تحليلگر نحوي دريافت ميكند. اين لغت در مجموعه First(Statement) وجود دارد اما ، نكته اينجا است كه هر دو گسترش CallSt و AssignmentSt با Id آغاز ميشوند. لذا ، تحليلگر نحوي نمي تواند بر اساس ترم پيش بيني Id تصميم بگيرد كه كدام گسترش بايد انتخاب شود. بنابراين ترم بعدي را دريافت مي كند ترم بعدي در مثال فوق := است. حالا ميتوان تصميم گرفت كه گسترش AssignmentSt بايد انتخاب شود. نهايتا" ، درخت تجزيه بصورت زير خواهد بود.



٤.٣ نتيجه تحليل عملكرد ذهن

از مطالب ارائه شده در بخش قبلي مي توان سه نتيجه بشرح زير گرفت:
اولا" در هر مرحله تحليلگر پيش بيني ميكند كه ترم بعدي چه بايد باشد. براي نمونه در ابتدا پيش بيني سر ترم گرامر را تحليلگر نحوي مي نمود. به اين نوع تحليلگرها در اصطلاح تجزيه گرهاي پيش بيني كننده يا Predictive Parsers گويند.
ثانيا" عمل خواندن لغات از چپ به راست صورت ميگيرد. درمثال فوق ابتدا سمت چپ ترين لغت يعني If از ورودي خوانده شد و عمل خواندن از چپ به راست ادامه پيدا كرد. لغت خوانده شده از ورودي را در اصطلاح ترم پيش بيني يا Look Ahead گويند زيرا ، بر اساس اين لغت است كه تحليلگر پيش بيني ميكند ترم بعدي چه بايد باشد. براي نمونه بر اساس ترم پيش بيني If تحليلگر نحوي تصميم گرفت كه گسترش IfSt را براي Statement بايد انتخاب نمايد. ثس پيش بيني مشاهده جمله IfSt در ورودي شد.

ثالثا" با در دست داشتن يك يا بيشتر از ترم هاي پيش بيني تحليلگر مي تواند تصميم گيري كند كه كدام گسترش از گسترش‌هاي متفاوت ترم مورد انتظار را بايد انتخاب نمايد. براي نمونه در بالا با در دست داشتن يك ترم پيش بيني If تحليلگر تصميم گرفت كه گسترش IfSt را براي ترم مورد پيش بيني يعني Statement بايد انتخاب كند اما، در هنگاميكه در ورودي ترم پيش بيني از نوع Id ظاهر شد چون دو گسترش متفاوت Statement يعني AssignmentSt و CallSt هر دو با ترم Id آغاز ميشدند ، تحليلگر نمي توانست تصميم بگيرد كه كداميك را انتخاب كند. در اين حالت ، با در دست داشتن دو ترم پيش بيني Id و := تحليلگر توانست تصميم بگيرد كه كدام گسترش بايد انتخاب شود.

اصولا" عمل تجزيه بالا به پائين از سر ترم گرامر آغاز و به ترمهاي پايانی درون جمله يا برنامه مورد كامپايل خاتمه مي يابد. براي انجام عمل تجزيه بالا به پائين فرم كلي جملات يا بعبارت ديگر گرامر زبان را آنقدر محدود ميكنند كه ، با در دست داشتن k ترم‌پيش بيني از متن برنامه مورد كامپايل بتوان عمل تحليل تحوي را بدرستي انجام داد.

تجزيه گرهاي پيش بيني كننده يا Predictive Parsers با در دست داشتن يك يا چند ترم پايانی بعدي در ورودي ، قادر به انجام عمل تحليل نحوي هستند . گرامرهاي مورد استفاده براي اين روش تجزيه را اصطلاحاً LL (k) يا Left Lookahead گويند . منظور اينست كه مي توان با استفاده از قواعد گرامر و با در دست داشتن k ترم پايانی بعدي در ورودي يا در اصطلاح k ترم پيش بيني ، عمل تجزيه بالا به پائين را انجام داد.

٤.٤ گرامرهای LL (1)

تجزيه گرهاي پيش بيني كننده براي تجزيه بالا به پائين با در دست داشتن يك يا چند ترم بعدي در ورودي بايد قادر به تشخيص اين باشند كه :

اولاً : جمله مورد نظر تا اين مرحله از لحاظ گرامري صحيح است .
ثانياً : چه ترمهاي مياني مورد انتظار هستند .
ثالثاً : چه ترمهاي پايانی در سر ورودي مي توانند ظاهر شوند .

چنانچه با در دست داشتن حداكثر K ترم بعدي از ورودي يا بعبارت ديگر با در دست داشتن حداكثر K ترم پيش بيني بتوان عمل تجزيه قابل پيش بيني را بر روي يك گرامر به انجام رساند آن گرامر را LL(K) گويند.

چنانچه با در دست داشتن يك ترم پايانی بعدي از چپ به راست در ورودي بتوان تشخيص داد كه از بين گسترش هاي متفاوت براي يك ترم مياني كداميك بايد انتخاب شوند ، گرامر را LL (1) گويند .

براي نمونه گرامر زير را در نظر بگيريد :

$S \rightarrow I \mid W \mid A \mid P \mid C$

$I \rightarrow$ if B do S E

$B \rightarrow$ id D

$D \rightarrow \varepsilon \mid R$ id

$R \rightarrow < \mid > \mid =$

$W \rightarrow$ while B do S

$A \rightarrow$ id := No

$P \rightarrow$ id '(' M ')'

$M \rightarrow \varepsilon \mid$ id

$C \rightarrow$ '{' T '}'

$T \rightarrow S\ G$

$G \rightarrow ; T \mid \varepsilon$

$E \rightarrow$ else S $\mid \varepsilon$

اين گرامر LL (1) نيست زيرا ، براي نمونه با ديدن id يا شناسه نميتوان مشخص نمود كه از بين قواعد مختلف براي گسترش S كداميك بايد انتخاب شود. اصولاً جهت تجزيه بالا به پائين بايد پس از خواندن يك ترم پايانی از ورودي مشخص نمود كه آيا ترم پايانی خوانده شده متعلق به مجموعه First براي ترم پايانی يا مياني مورد انتظار است يا خير ؟ . سپس بايد مشخص نمود كه كدام گسترش از گسترش هاي متفاوت ترم مياني مورد انتظار با ترم پايانی خوانده شده آغاز مي شوند. به عبارت ديگر بايد مشخص نمود كه ترم پايانی خوانده شده متعلق به مجموعه First براي كداميك از گسترش‌هاي مختلف ترم مياني مورد نظر است .

بنا بر اين اگر قرار باشد با داشتن يك ترم پيش بيني در هر مرحله از تجزيه بتوان مشخص نمود كه كدام گسترش براي ترم مياني مورد انتظار A با گسترش‌هاي

$A \rightarrow A1 \mid A2 \mid .... \mid An$

بايد انتخاب شود ، آنگاه بايد اشتراك مجموعه First براي هر دو گسترش متفاوت از A تهي باشد يا بعبارت ديگر بايد رابطه :

$\forall\ i,j\ \varepsilon\ 1 .. n\ ,\ i \neq j \Rightarrow first(Ai) \cap first(Aj) = \varnothing$

برقرار باشد. در غير اينصورت فرض كنيد كه براي نمونه

$i,j\ \varepsilon\ 1 .. n\ ,\ \neq j\ ,\ first(Ai) \cap first(Aj) = \{ a \}$

آنگاه با مشاهده ترم پایانی a در ورودي تحليلگر نمي تواند بلافاصله تصميم بگيرد كه آيا گسترش A به Ai بايد انتخاب شود. يا گسترش A به Aj.

پس بطور خلاصه ميتوان گفت :

يك گرامر LL(1) است اگر اشتراك مجموعه first هر دو گسترش متفاوت براي هر ترم مياني متعلق به آن گرامر تهي باشد .

براي نمونه درخت تجزيه را براي جمله :

{ a := 5 ; if b do f( ) }

با روش بالا به پائين ميتوان بصورت زير ايجاد نمود :

1- تحليلگر لغوي فراخواني ميشود .

2- لغت '{' توسط تحليلگر لغوي به تحليلگر نحوي برگردانده ميشود .

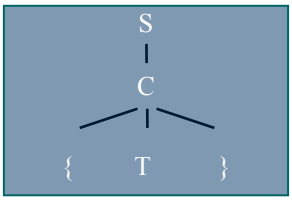
3- تحليلگر نحوي كه در انتظار مشاهده S در ورودي است ، ابتدا ميبايست مطمئن شود كه ترم خوانده شده يعني '{' آيا متعلق به مجموعه first (S) است.

'{' ε first (S) = { if , while , id , '{' }

سپس بايد مشخص شود كه '{' به مجموعه First براي كدام گسترش از گسترش‌هاي متفاوت S تعلق دارد . چون :

'{' ε first (C) = { '{' }

پس درخت تجزيه زير ايجاد ميشود :

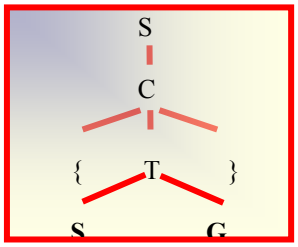


4- با مشاهده { در ورودي دو مرتبه مراحل 1 تا 4 بشرح زير تكرار مي شود .

- لغت بعدي a ميباشد كه از نوع id است .

id ∈ first (T) = first (S) = { if , while , id , '{' }

بنابر اين گسترش S به T انتخاب شده ، درخت تجزيه بصورت زير خواهد بود.



اما در اين مرحله مشاهده ميشود كه دو گسترش مختلف $S \rightarrow A$ و $S \rightarrow P$ هر دو با يك ترم id آغاز ميشوند بنابراين در اين مرحله به علت LL(1) نبودن گرامر

نمیتوان با در دست داشتن ترم پیش بیني a تصمیم قطعي در مورد گسترش S گرفت.

همانگونه که در بالا توضیح داده شد ، براي انجام تجزیه بالا به پائین نیاز به در دست داشتن مجموعه First براي ترمهاي گرامر است. روش محاسبه مجموعه سرآغاز یا مجموعه First در بخش بعدي توضیح داده خواهد شد .

٤.٥ مجموعه های سرآغاز و پیرو

مجموعه سرآغاز براي يك ترم شامل مجموعه ترمهاي پایانی است که گسترش‌هاي متفاوت و جملات مشتق از آن ترم را مي توانند آغازکنند. در حالت کلي براي ترم A مجموعه سرآغاز بصورت زیر محاسبه میشود:

- چنانچه براي A گسترش‌ي بصورت $A \rightarrow a\,\alpha$ وجود داشته باشد ، آنگاه :
  $First(A) = \{ a \}$

- چنانچه براي A گسترشی بصورت $A \rightarrow B\,\alpha$ وجود داشته باشد ، آنگاه :
  $First(A) = First(B)$

- چنانچه براي A گسترشی بصورت $A \rightarrow B\,\alpha$ وجود داشته باشد و $B \rightarrow \delta \mid \varepsilon$ ، آنگاه :
  $First(A) = First(\delta) + First(\alpha)$

واضح است که اگر B را تهي یا ε درنظر گرفته شود ، آنگاه قاعده $A \rightarrow B\,\alpha$ در واقع بصورت $A \rightarrow \alpha$ تبدیل میشود. بنابراین First(A) شامل First(α) نیز میشود.

- چنانچه براي A گسترشی بصورت $A \rightarrow X1\ X2\ X3\ \ldots\ Xn$ وجود داشته باشد وبراي X1 تا Xi قاعده تهي هم وجود داشته باشد یا به عبارت دیگر

$\forall\ 1 \le j \le i\ \ Xj \rightarrow \delta j \mid \varepsilon$

آنگاه :

$First(A) = First(X1) + First(X2) + \ldots + First(Xi+1)$

واضح است که در قاعده $A \rightarrow X1\ X2\ X3\ \ldots\ Xn$ اگر X1 وجود داشته باشد ، آنگاه

$First(A) = First(X1)$

وگرنه در صورتیکه X1 تهي یا ε باشد ، آنگاه قاعده بصورت زیر خواهد بود

$A \rightarrow X2\ X3\ \ldots\ Xn$

به این ترتیب :

$First(A) = First(X1) + First(X2)$

حالا اگر X1 و X2 هر دو تهي باشند آنگاه

First(A) = First(X1) + First(X2)
+First(X3)

به همين ترتيب مي توان نهايتا" نشان داد كه :

$$First(A) = \sum First(Xj) , 1 \le j \le I+1$$

چنانچه براي ترم A گسترش تهي نيز وجود داشته باشد يا بعبارت ديگر در حالت كلي قواعد مربوط به ترم A بصورت $A \rightarrow \alpha \mid \varepsilon$ باشد ، آنگاه با مشاهده يك عنصر متعلق به First(α) واضح است كه گسترش $A \rightarrow \alpha$ بايد انتخاب شود. اما چه عنصري بايد ظاهر شود تا در حاليكه انتظار مشاهده A ميرود ، بتوان دريافت كه گسترش $A \rightarrow \varepsilon$ بايد انتخاب شود. واضح است كه اگر ترم مورد انتظار يعني A در ورودي ظاهر نشود بايد ترمي كه پس از A انتظار مشاهده آن در ورودي ميرفت ظاهر شود. براي نمونه به گرامر زير توجه كنيد :

LabeledSt $\rightarrow$ Label Statement

Label $\rightarrow$ No : | ε

Statement $\rightarrow$ AssignmentSt | IfSt | WhileSt

AssignmentSt $\rightarrow$ Id :=
Expression

WhileSt $\rightarrow$ WHILE Expression DO Statement

IfSt $\rightarrow$ IF Expression DO Statement

طبق گرامر فوق جملات داراي برچسب يا Label و بدون برچسب هستند. در آغاز كار طبق گرامر فوق جهت مشاهده LabeledSt ابتدا انتظار مشاهده يك Label در ورودي ميرود. بنابراين انتظار ميرود كه اولين ترم در ورودي يك عدد يا No باشد زيرا :

No $\in$ First(Label) = { No,
ε)

اما ، اگر ترم خوانده شده از ورودي از نوع No نباشد ، مشخص است كه Label در ورودي وجود نداشته است. بتابراين ، انتظار ميرود كه ترم خوانده شده آغاز كننده ترم بعد از Label يعني Statement باشد. بنابراين چنانچه ترم مورد انتظار داراي گسترش تهي هم باشد ، درصورتي گسترش تهي انتخاب ميشود كه ترم پيش بيني متعلق به مجموعه First ترم بعدي آن در سمت راست قاعده مورد نظر باشد. مجموعه First ترمهايي كه پس از ترم مياني A در سمت راست قواعد متفاوت ظاهر ميشوند را در اصطلاح مجموعه پيرو يا Follow براي آن ترم مياني مي‌گويند. براي نمونه در گرامر فوق :

Follow(Label) = First(statement) = {S_Id, S_If, S_While}

اصولا" براي بدست آوردن مجموعه پيرو به روشهاي زير عمل ميشود:

- در صورتيكه ترم A در سمت راست يك قاعده بصورت زير ظاهر شود

$B \rightarrow A\,\alpha$

آنگاه :

$Follow(A) = First(\alpha)$

- در صورتيكه ترم A در سمت راست يك قاعده بصورت زير ظاهر شود

$B \rightarrow \alpha$

A

آنگاه مجموعه پيرو براي A شامل مجموعه پيرو B ميباشد

$Follow(A) \supseteq Follow(B)$

اما ، بالعكس صادق نيست . علت اين است كه هر جايي كه B درسمت راست قواعد ظاهر شود ميتوان بجاي آن $A\,\alpha$ را قرار داد اما ، بالعكس صادق نيست.

- همواره در مجموعه پيرو براي سرترم گرامر علامت خاتمه فايل يا End Of File وجود دارد. زيرا ورودي تحليلگر نحوي يك فايل است. براي نمونه يك برنامه C را در نظر بگيريد. اين برنامه بعنوان يك جمله ورودي به كامپايلر داده ميشود. چون برنامه در فايل است در انتهاي آن علامت خاتمه فايل قرار ميگيرد. برنامه ورودي در اينجا از سرترم گرامر يعني برنامه C مشتق ميشود. پس بعد از سرترم گرامر علامت خاتمه فايل قرار ميگيرد. بطور كلي هر جمله اي كه در ورودي كامپايلر قرار ميگيرد بايد از سرترم گرامر مربوطه مشتق شود. چون جمله ورودي در داخل يك فايل است لذا ، همواره در مجموعه پيرو سرترم گرامرعلامت خاتمه فايل وجود دارد. علامت خاتمه فايل را بطور اختصاري معمولاً با $ مشخص ميكنند.

بنابراين مشاهده ميكنيد چنانچه مجموعه First براي يك ترم شامل عنصر تهي ε باشد ، آنگاه بايد عنصر ε را از داخل مجموعه حذف و بجاي آن عناصر مجموعه پيرو را به داخل مجموعه First افزود . به اين ترتيب :

$First(Label) = \{ No, \varepsilon \} = First(Label) + Follow(Label)$
$= \{ No, \{S\_Id,\ S\_If,\ S\_While\}$

مشاهده ميكنيد مجموعه Follow براي يك ترم شاخص گسترش تهي براي آن ترم است. بنابراين ميتوان نتيجه گرفت :

يك گرامر LL(1) است اگر اشتراك مجموعه سرآغاز براي هر دو گسترش هر ترم متعلق به آن گرامر تهي باشد و چنانچه آن ترم داراي گسترش تهي باشد اشتراك مجموعه سرآغاز و پيرو آن ترم بايد تهي باشد.

براي نمونه گرامر زير LL(1) نيست :

$LabeledSt \rightarrow Label\ Statement$

Label → Id : | ε

Statement → AssignmentSt | IfSt | WhileSt

AssignmentSt → Id := Expression

WhileSt → WHILE Expression DO Statement

IfSt → IF Expression DO Statement

علت LL(1) نبودن گرامر فوق این است که :

FIrst(Label) ∩ Follow(label) = {S_Id}

بنابراین با مشاهده Id در ورودي نمي توان تصميم گرفت که آیا گسترش Label → Id : باید انتخاب شود یا گسترش Label → ε.

براي بدست آوردن مجموعه First میتوان از ماتریس تجانس نیز استفاده نمود. براي نمونه به گرامر زیر توجه کنید :

P → D B | B
B → A e
A → b S | S
S → a D | B a
D → a D | d

چنانچه در حالت کلي ترم A با ترم B آغاز شود يك رابطه بطول يك بين ترم A یا B وجود دارد. میتوان به این ترتیب يك گراف رواسط ایجاد نمود. اکنون مسإله بدست آوردن روابط بین هر دو گره در گراف است. یعني هدف بدست آوردن وجود روابط با هر طولي بین هر دو گره در داخل این گراف جهت دار روابط است. میتوان از ماتریس تجانس استفاده نمود. به این ترتیب که اگر بین دو ترم A و B يك رابطه آغاز شدن A توسط B وجود دارد در سطر مربوط به A و ستون مربوط به B در ماتریس مقدار يك قرار داده میشود. به این ترتیب براي گرامر فوق ماتریس زیر ایجاد میشود :

	A	B	D	P	S	a	b	d	e
A	1	0	0	0	1	0	0	0	0
B	1	1	0	0	0	0	0	0	0
D	0	0	1	0	0	1	0	1	0
P	0	1	1	0	1	0	0	0	0
S	0	1	0	0	1	1	0	0	0
a	0	0	0	0	0	1	0	0	0
b	0	0	0	0	0	0	1	0	0
d	0	0	0	0	0	0	0	1	0
e	0	0	0	0	0	0	0	0	1

اکنون آنقدر ماتریس را در خودش ضرب باید نمود تا اینکه در دو تکرار متوالي مقادیر ماتریس با یکدیگر تفاوتي نکند . در اینصورت ماتریس نهایي روابط First را مشخص میکند. باید توجه داشته باشید که در هنگام ضرب دو ماتریس بولین

بجاي ضرب از عمل و منطقي يا AND استفاده ميشود و بجاي جمع عمل از OR منطقي استفاده ميشود.

٤.٦ تبديل گرامرها به فرم LL(١)

همانگونه كه قبلا نيز توضيح داده شد اگر هر دو گسترش متفاوت هر ترم متعلق به يك گرامر نهايتاً با يك ترم مشترك آغاز نشوند آن گرامر LL(1) است . براي تبديل گرامر به فرم LL(1) بعضاً ميتوان از روشي موسوم به *فاكتوگيري* چپ استفاده نمود .

4.6.1 - فاكتورگيري چپ

چنانچه در حالت كلي براي ترم A گسترش‌هايي بصورت زير موجود باشد :

$$A \rightarrow a\,\alpha \mid a\,\beta \mid \gamma$$

آنگاه :

$$first(a\,\alpha) \cap first(a\,\beta) = \{ a \}$$

با فاكتورگيري a از سمت چپ دو گسترش aβ و aα ميتوان گرامر را بصورت زير بازسازي نمود :

$$A \rightarrow a\,B \mid \gamma$$

$$B \rightarrow \alpha \mid \beta$$

در فرم توسعه يافته ميتوان بدون استفاده از ترم كمكي B عمل فاكتور گيري را انجام داد و گسترش‌هاي ترم مياني A را بصورت زير تبديل نمود:

$$A \rightarrow a\,(\alpha \mid \beta) \mid \gamma$$

با استفاده از روش فاكتور گيري چپ ميتوان قواعد مربوط به سرترم S از گرامر ارائه شده در بخش قبلي را بصورت زير بفرم LL(1) تبديل نمود :

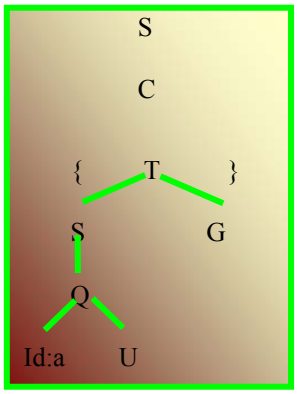
$$S \rightarrow I \mid W \mid Q \mid C$$

$$Q \rightarrow id\ U$$

$$U \rightarrow := No \mid \text{‘(‘ M ‘)’}$$

به اين ترتيب اكنون ميتوان بكار توليد درخت تجزيه كه در بخش قبل با مشاهده id يعني a در ورودي متوقف گرديد ادامه داد زيرا ، حالا با ديدن ترم پاياني **a** در

ورودي بلافاصله گسترش $Q \rightarrow S$ و پس از آن گسترش $Q \rightarrow id\ U$ انتخاب ميشود و درخت تجزيه به اين صورت تبديل ميگردد



حالا با فرا خواني تحليلگر لغوي در ورودي = : ظاهر ميشود كه در اينجا بطور قطعي ميتوان گفت كه گسترش $U \rightarrow :=No$ بايد انتخاب شود. در حالت كلي اگر در گرامر قواعد بصورت خود بازگشتي چپ وجود داشته باشد ، باز هم گرامر LL (1) نميتواند باشد.

4.6.2- تبديل قواعد خود بازگشتي چپ

وجود قواعد بصورت خود بازگشتي چپ مغاير با **LL (1)** بودن گرامرها ميباشد . براي نمونه به قواعد زير توجه نماييد :

$$A \rightarrow A\,\alpha \mid \beta$$

جهت نقض **LL (1)** بودن گرامر كافي است اثبات شود كه :

$$first(A\,\alpha) \cap first(\beta) \neq \varnothing$$

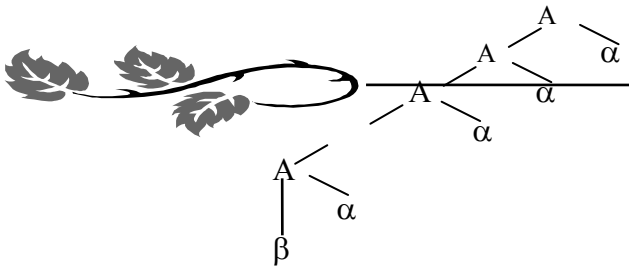
در اينجا واضح است كه :

$first(A\,\alpha) \cap first(\beta) = \beta$ چرا

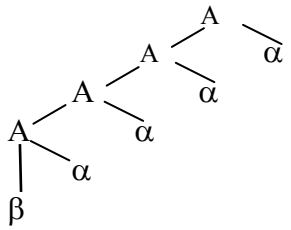
كه :

$$first(A\,\alpha) = first(A) = First(\beta)$$

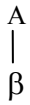
براي تبديل قواعد مربوط به A بصورت LL(1) به اين ترتيب ميتوان استدلال نمود كه دو قاعده $A \rightarrow A\,\alpha \mid \beta$ نمايانگر فرم كلي رشته هائي است كه با β آغاز و با تعداد صفر يا بيشتر α ادامه مي يابند. بنابراين هر رشته با فرم كلي $\beta\alpha\alpha\alpha$ ..... α از سر ترم A بايد مشتق شود.



به این ترتیب براي رشته β α α α درخت تجزیه بصورت زیر خواهد بود :



به همین ترتیب رشته β نیز از سز ترم گرامر مشتق میشود :

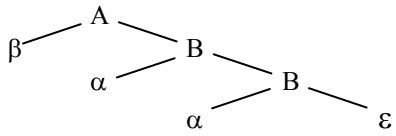


رشته هاي فوق را با گرامر زیر نیز میتوان تولید نمود:

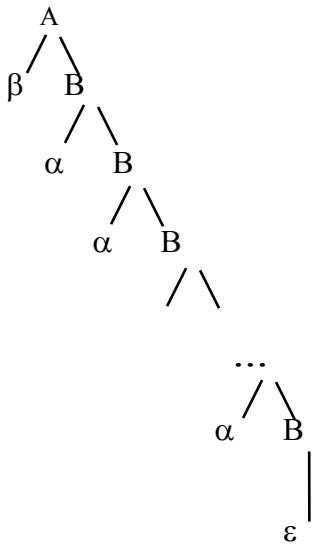
$A \rightarrow \beta B$

$B \rightarrow \alpha B \mid \varepsilon$

رشته β α α α را در نظر بگیرید. این رشته بر طبق گرامر فوق از سر ترم A مشتق میشود :



و به همین ترتیب هر رشته با فرم کلي α..... βααα از سر ترم A باید مشتق مي شود. درخت تجزیه در حالت کلي بصورت زیر است.



بنابراين ميتوان نتيجه گرفت كه براي حذف خودبازگشتي چپ از قواعد با فرم كلي :

$A \rightarrow A\,\alpha \mid \beta$

مي توان آنها را با قواعد معادل زير جايگزين نمود :

$A \rightarrow \beta\,B$

$B \rightarrow \alpha\,B \mid \varepsilon$

در فرم توسعه يافته گرامر بصورت زير تبديل ميشود :

$A \rightarrow \beta\,\{\,\alpha\,\}$

در فرم توسعه يافته آكولاد به مفهوم تكرار صفر يا بيشتر است.

مثال گرامر عبارات را بصورت **LL (1)** تبديل كنيد .

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

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

$F \rightarrow Id \mid No \mid (\,E\,)$

به قواعد E توجه نماييد. دو گسترش $E + T$ و $E - T$ با يك ترم مشترك E آغازميشوند. به همين ترتيب دو گسترش متفاوت ترم مياني T نيز با يك ترم آغاز ميشوند . پس از انجام عمل فاكتورگيري چپ گرامر توسعه يافته عبارات بصورت زير تبديل ميشود:

$E \rightarrow E\,(+ \mid -)\,T \mid T$

$T \rightarrow T\,(* \mid /)\,F \mid F$

$F \rightarrow id \mid NO \mid (\,E\,)$

حالا با حذف خود بازگشتي چپ ، گرامر به صورت زير تبديل مي شود :

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

$T \rightarrow F\,\{\,(* \mid /)\,F\,\}$

$F \rightarrow Id \mid NO \mid (\,E\,)$

درصورت وجود قواعد تهي با فرم كلي $A \rightarrow \varepsilon$ درگرامر نيز ممكن است گرامر LL(1) نباشد.

4.6.3- حذف قواعد تهي

چنانچه در گرامري قواعد تهي وجود داشته باشد ، آن گرامر ممكن است LL(1) نباشد. براي تبديل گرامر به فرم LL(1) بايد ابتدا قواعد تهي را حذف نمود و سپس

با استفاده از روشهاي فاكتور گيري و حذف خودبازگشتي چپ گرامر را به فرم LL(1) تبديل نمود. براي نمونه به گرامر زير توجه نمانيد :

LabeledSt ⟶ Label Statement

Label ⟶ Id : | ε

Statement ⟶ AssignmentSt | CallSt

AssignmentSt ⟶ Id := Expression

CallSt ⟶ Id '(' ActualParams ')' | Id

ActualParams ⟶ Expression | Expression , Params

Expression ⟶ Id | No

در گرامر فوق براي ترم مياني Label يك گسترش تهي وجود دارد. لذا بايد First (Label) ∩ Follow(label) = ∅ باشد. اما :

First(label) = { Id }

و

Follow(label) = First(Statement) = First(AssignmentSt) + First(CallSt) = { Id }

بنابراين :

First (Label) ∩ Follow(label) = { Id } ≠ ∅

بنابراين گرامر LL(1) نيست زيرا هنگاميكه انتظار مشاهده Label در ورودي مي رود ، با مشاهده Id در ورودي نمي توان تصميم گرفت كه آيا گسترش Label ⟶ Id : بايد انتخاب شود و يا گسترش Label ⟶ ε .

براي تبديل گرامر فوق به فرم LL(1) بايد گسترش Label ⟶ ε حذف شود و هر جايي كه از تزم مياني Label در سمت چپ يك قاعده استفاده شده است ، يك بار Label را تهي و بار ديگر بصورت غير تهي در نظر گرفت. بنابراين گرامر بصورت زير تبديل ميشود :

LabeledSt ⟶ Statement | Label Statement

Label ⟶ Id :

براي LabeledSt اكنون First(statement) ∩ Follow(label) = {Id} ≠ ∅ پس بايد با استفاده از عمل فاكتور گيري چپ گرامر را به فرم LL(1) تبذيل نمود. براي اين منظور بايد ابتدا ترم ها را جايگزين كرد .

LabeledSt ⟶ Id := Expression | Id '(' ActualParams ')' | Id : Statement

پس از فاکتورگیري از Id قاعده بصورت زیر تبدیل میشود :

LabeledSt $\longrightarrow$ Id

St

St $\longrightarrow$:= Expression | '(' ActualParams ')' | : Statement

در مورد Statement نیز باید عمل فاکتور گیري چپ را انجام داد:

Statement $\longrightarrow$ AssignmentSt | CallSt

با جایگزیني AssignmentSt و Callst قاعده مربوط به Statement بصورت زیر تبدیل میشود:

Statement $\longrightarrow$ Id := Expression | Id '(' ActualParams ')'

اکنون با استفاده از عمل فاکتورگیري چپ قاعده فوق بصورت زیر تبدیل میشود :

Statement $\longrightarrow$ Id StTail

StTail $\longrightarrow$:= Expression | Id '(' ActualParams ')'

مثال : گرامر زیر را به فرم LL(1) تبدیل.

S $\longrightarrow$ L A B

L $\longrightarrow$ d | ε

A $\longrightarrow$ d A | B a

B $\longrightarrow$ B b | ε

گرامر فوق LL(1) نیست زیرا :

First (L) $\cap$ Follow(L) = { d } $\neq \varnothing$

بنابراین باید گسترش تهي $L \longrightarrow \varepsilon$ حذف شود. در اینصورت باید در هر قاعده اي که L ظاهر میشود یك بار مقدار آنرا تهي و یك بار غیر تهي در نظر گرفت. بنابراین گرامر بصورت زیر تبدیل میشود :

S $\longrightarrow$ A B | L A B

L $\longrightarrow$ d

A $\longrightarrow$ d A | B a

B $\longrightarrow$ b B | ε

توجه داشته باشید که پس از حذف خود بازگشتي چپ قواعد $B \longrightarrow B\ b \mid \varepsilon$ را مي توان بصورت $B \longrightarrow b\ B \mid \varepsilon$ ساده نمود. البته هنوز گرامر فوق LL(1) نیست زیرا اشتراك مجموعه سرآغاز دو گسترش مختلف ترم میاني S تهي نمي باشد.

First (A B) $\cap$ Follow(L A B) = { d } $\neq \varnothing$

بنابراین باید با جایگزیني ترمهاي A و L در آغاز دو گسترش ترم S و سپس عمل فاکتور گیري چپ گرامر را LL(1) نمود.

S $\longrightarrow$ d A B | B a B | d A B

A $\longrightarrow$ d A | B a

$B \longrightarrow b\ B \mid \varepsilon$

در نتیجه گرامر بصورت زیر تبدیل میشود :

$S \longrightarrow d\ A\ B \mid B\ a\ B$

$A \longrightarrow d\ A \mid B\ a$

$B \longrightarrow b\ B \mid \varepsilon$

٤.٦ ایجاد جدول تجزیه بالا به پائین

جدول تجزیه ساختاري براي توليد برنامه تحليلگر نحوي براي گرامرهاي LL(1) است. براي نمونه به گرامر زير توجه نمائيد.

$S \longrightarrow d\ A\ B \mid B\ a\ B$

$A \longrightarrow d\ A \mid B\ a$

$B \longrightarrow b\ B \mid \varepsilon$

براي ايجاد تجزيه گر براي اين گرامر بايد ابتدا مجموعه هاي سرآغاز را براي ترمهاي مياني بدست آورد. با استفاده از اين مجموعه ها جدول تجزيه بدست مي آيد :

FIrst(S) = {d} + First(B)
First(A) = {d} + First(B)
First(B) = {b} + Follow(B)
= {b} + {a, $} = {a, b, $}

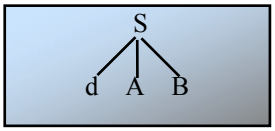
	a	b	d	$
S	BaB	BaB	dAB	
A	Ba	Ba	dA	
B	ε	bB		ε

اکنون با استفاده از جدول تجزيه فوق ميتوان عمل تجزيه بالا به پائين را انجام داد. براي نمونه جمله ddabbb را با استفاده از جدول تجزيه فوق ميتوان براحتي تجزيه نمود. براي اين منظور از يك پشته بنام پشته تجزيه استفاده ميشود.

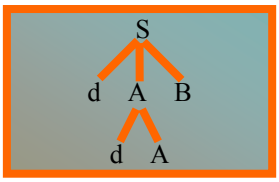
پشته	ورودي	قاعده تجزيه
$S	ddabbb$	$S \longrightarrow d\ A\ B$
$B A d	ddabbb$	
$B A	dabbb$	$A \longrightarrow d\ A$
$B A d	dabbb$	
$B A	abbb$	$A \longrightarrow B\ a$
$B a B	abbb$	$B \longrightarrow \varepsilon$
$ a B	abbb$	
$B	bbb$	$B \longrightarrow bB$
$bB	bbb$	
$B	bb$	$B \longrightarrow bB$
$bB	b b$	
$B	b $	$B \longrightarrow bB$
$bB	b$	

همانگونه که در جدول فوق مشاهده میکنید به انتهاي رشته ورودي علامت خاتمه فایل یعني $ افزوده میشود. عمل تجزیه از سرترم گرامر آغاز میشود. و پس از سرترم انتظار میرود که در ورودي علامت خاتمه فایل ورودي یعني $ ظاهر شود. لذا بنار این پیش بیني در آغاز کار رشته $S در داخل پشته قرار داده شده است.

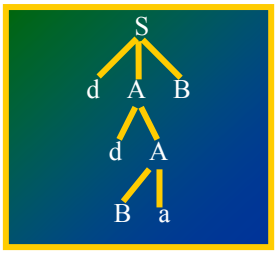
عمل تجزیه از ردیف مربوط به سرترم گرامر S آغاز میشود. با مشاهده اولین ترم در جمله **d**dabbb$ که ترم در ورودي که ترم پایانی d است به ستون d در سطر S در داخل جدول ارجاع میشود. گسترش BaB در مکان (S,d) از جدول مشخص شده است. لذا، درخت تجزیه بصورت زیر ایجاد میشود:



اکنون، ترم بعدي ازجمله dd**a**bbb$ خوانده میشود. این ترم پیش بیني نیز d است. با توجه به اینکه ترم پیش بیني هنوز d است. طبق درخت تجزیه انتظار مشاهده A در وردي میرود. درمکان (A,d) از جدول تجزیه گسترش dA قرار دارد. درخت تجزیه زیر حاصل میشود.

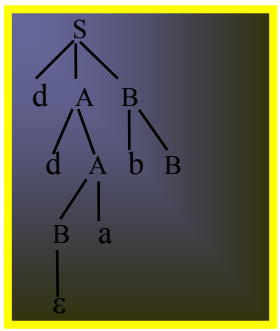


ورودي بعدي در جمله dd**a**bbb$ ترم پایانی a است. طبق درخت تجزیه انظار مشاهده A در ورودي میرود. درمکان (A,a) از جدول گسترش Ba قرار گرفته است. بنابراین درخت تجزیه بصورت زیر توسعه داده میشود :

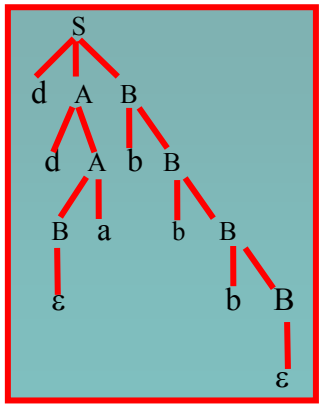


حالا با توجه به محتوي جدول تجزیه در مکان (B,a) باید گسترش B → ε انتخاب شود. بنابراین طبق درخت تجزیه انتظار دیدن a در ورودي میرود. در این لحظه ترم پیش بیني همان طوري که انتظار میرود a است. ورودي بعدي در جمله dda**b**bb ترم پایانی b است. طبق درخت تجزیه انظار مشاهده B در ورودي میرود. درمکان

(B,b) از جدول گسترش bB قرار گرفته است. بنابراین تا این مرحله درخت تجزیه بصورت زیر مي باشد:



به همین ترتیب اگر عملیات ادامه پیدا کند تهایتاً درخت تجزیه بصورت زیر ایجاد میشود :



مثال - گرامر عبارات را که در زیر ارائه شده تبدیل به فرم LL(1) نموده ، جدول تجزیه براي يك تحليگر پيش بيني كننده ايجاد كنيد:

Expression ⟶ Expression RelOp SimpleExp | SimpleExp

RelOp ⟶ < | <= | = | <> | >= | > |

IN

SimpleExp ⟶ SimpleExp '+' Term | SimpleExp '-' Term | SimpleExp Or Term | Term

Term ⟶ Term '/' Factor | Term '*' Factor | Term DIV Factor | Term AND Factor | Factor

Factor ⟶ Number | NOT Factor | '(' Expression ')' | Variable

Variable ⟶ Identifier

VarTale

VarTale ⟶ '[' Dim ']' .Variable |

∈

Dim ⟶ Expression

OtherDims

OtherDims ⟶ ' ,' Dim | ∈

قواعده ارائه شده براي Expression داراي وضعيت خودبازگشتي چپ مستقيم است.

Expression ⟶ Expression RelOp SimpleExp | SimpleExp

درحالت كلي $A \rightarrow A\alpha \mid \beta$ را مي توان با قواعد معادل زير جايگزين نمود :

$A \rightarrow \beta B$

$B \rightarrow \alpha B \mid \varepsilon$

حالا Expression را مشابه A و RelOp SimpleExp مشابه با α و SimpleExp را مشابه با β در نظر بگیرید. به این ترتیب قواعد مربوط به E بصورت زیر تبدیل میشوند :

Expression $\rightarrow$ SimpleExp E

E $\rightarrow$ RelOp SimpleExp E | ε

در مورد SimpleExp ابتدا باید فاکتورگیري چپ نمود .

SimpleExp $\rightarrow$ SimpleExp '+' Term | SimpleExp '-' Term | SimpleExp Or Term | Term

پس از فاکتورگیري چپ قواعد بصورت زیر تبدیل میشوند :

SimpleExp $\rightarrow$ SimpleExp Simple | Term

Simple $\rightarrow$ '+' Term | '-' Term | Or Term

پس از حذف خودبازگشتي چپ قواعد به صورت زیر تبدیل میشوند :

SimpleExp $\rightarrow$ Term S

S $\rightarrow$ Simple S | ε

Simple $\rightarrow$ '+' Term | '-' Term | Or Term

درقاعده مربوط به S میتوان Simple را با گسترش آن جایگزین نمود. بنابراین قواعد فوق بصورت زیر ساده میشود :

SimpleExp $\rightarrow$ Term S

S $\rightarrow$ '+' Term S | '-' Term S | Or Term S | ε

بطورخلاصه گرامر عبارات در فرم LL(1) بصورت زیر مي باشد:

Expression $\rightarrow$ SimpleExp E

E $\rightarrow$ RelOp SimpleExp E | ε

RelOp $\rightarrow$ < | <= | = | <> | >= | > | IN

SimpleExp $\rightarrow$ Term S

S $\rightarrow$ '+' Term S | '-' Term S | Or Term S | ε

Term $\rightarrow$ Factor T

T $\rightarrow$ '/' Factor T | '*' Factor T | DIV Factor T | AND Factor T | ε

Factor $\rightarrow$ Number | NOT Factor | '(' Expression ')' | Variable

Variable → Identifier

VarTale

VarTale → '[' Dim ']' .Variable |

ε

Dim → Expression

OtherDims

OtherDims → ' ,' Dim | ε

اکنون میتوان مجموعه سرآغاز را براي ترمها محاسبه نمود :

First(Expression) = First(SimpleExp) = First(Term) = First(factor) = {Number, NOT, (, Identifier}

First(E) = First(RelOp) + {∈} = { < , <= , = , <> , >= , > , IN , ε }

Follw(E) = Follow(Expression) = First(OtherDims) + { ')' , $

}

First(OtherDims) = {',', ε } = {','} + Follow(OtherDims) = = {','} + Follow(Dim) = { ',' , ']' }

First(E) = { < , <= , = , <> , >= , > , IN } + { ')' , $, ',' , ']' }

First(S) = { + , - , OR , ε } = { + , - , OR } +

Follow(S)

Follow(S) = Follow(SimpleExp) =

First(E)

First(S) = { + , - , OR } + { < , <= , = , <> , >= , > , IN , ')' , $, ',' , ']' }

First(T) = { / , *, DIV, AND, ε } = { / , *, DIV, AND} + Follow(T)

Follow(T) = Follow(Term) =

First(S)

First(T) = { / , *, DIV, AND} + {+ , - , OR , < , <= , = , <> , >= , > , IN , ')' , $, ',' , ']'}

Fisr(Variable) =

{Identifier}

First(VarTale) = {'[' } +

Follow(VarTale)

Follow(VarTale) = Follow(Variable) = Follow(factor) =

First(T)

First(Variable) = {'['}+{ /, *,DIV, AND,+ ,-,OR ,< ,<= ,=,<>,>=,>,IN,')',$,',',']'}

First(Dim) = First(Expression) = {Number, NOT, (,

Identifier}

حالا با استفاده از روابط فوق مي توان جدول تجزیه بالا به پائین را بسادگي ایجاد کرد.

مثال - گرامر زیر را به فرم LL(1) تبدیل نموده ، جدول تجزیه براي آن ایجاد کنید.

S → SA |

BdA

A → Aa

| b

B → ba | Bd |

ε

تبدیل گرامر به فرم LL(1) در فرم توسعه یافته بسیار ساده تر است. لذا، گرامر بصورت زیر تبدیل میشود.

S → BdA {A}
A → b {a
}
B → [ba] {d
}

اما چون First (B) ∩ Follow(B) = { d } ≠ ∅ گرامر LL(1) نمي باشد. بنابراين بصورت زير عمل بايد نمود :

S → [ba] {d }dA {A}
A → b {a
}

درمورد گسترش S ، مشکل {d }d است. زيرا مشخص نيست که چه هنگامي مي توان به d دومي رسيد. اما ، واضح است که {d }d = d{d } است. بنابراين گرامر بصورت زير تبديل ميشود :

S → [ba] d{d }A {A}
A → b {a
}

حالا ميتوان جهت توليد جدول تجزيه گرامر را به فرم ساده تبديل نمود. براي اين منظور ابتدا [b a] را با B ، d {d} را با C و A{A} را با D بايد جايگزين نمود.

S → B C D
A → b F

→ a F | ε
F

B → b E | a | ε
E → a | ε
C → d C | d
D → A D | ε

First(S) = First(B) = {b, a, d}
First(A) = { b
}

First(F) = { a , ε } Follow(F) = Follow(A) = First(D) = {b}
First(B) = {b , a, ε} Follow(B) First(C) = {d}
=

First(E) = { a, ε} Follow(E) = Follow(B) = {d}
First(C)
={d}

First(G) = { d , ε } Follow(G) = Follow(C) = First(D) = {b}
FIrst(D) = { b, ε } Follow(D) = Follow(S) = {$}

اکنون مي توان جدول تجزيه را بسادگي براي اين گرامر بدست آورد.

٤.٧ تجزیه گرهای کاهینه بازگشتی

تجزيه گرهاي *كاهينه بازگشتي* يا Recursive scent parser را مي توان براي گرامرهاي LL(1) مورد استفاده قرار داد. در اين روش براي هر ترم مياني و سرترم ، يك روال يا زير برنامه به همان نام ايجاد ميشود. به اين ترتيب ، قواعد گرامر را عيناً با استفاده از توابع خود بازگشتي تبديل به كد برنامه مينمايند. لذا، مزيت اين روش سادگي ايجاد وخوانايي كد برنامه تجزيه گر است.

در ساختاركلي يا در واقع برنامه اصلي براي اين نوع تحليلگر پس از انجام عمليات اوليه بنام Init تحليلگر لغوي بنام NextSymbol فراخواني فراخوانده ميشود تا نوع اولين ترم پيش بيني در يك متغير سرارسري بنام CurrentSymbol قرار گيرد. سپس روال تعيين شده براي سرترم گرامر كه در زير ProgramX ناميده شده ، مورد فراخواني قرار ميگيرد. بنابراين بدنه اصلي برنامه تحليلگر كاهينه بازگشتي بصورت زير است :

```
Begin
   init;
   NextSymbol;

                                                    ProgramX

End .
```

Current Symbol متغيري سراسري از نوع شمارش پذير Symbols است كه قبل از اين براي تعيين نوع لغات توسط تحليلگر لغوي مشخص شد. نوع Symbols حاوي انواع لغات موجود در زبان مورد نظر است. براي نمونه :

```
                                                    Type
Symbols = ( S_if , S_while, S_repeat , S_for, S_Case, S_then, S_else, S_do,
            S_program, S_uses, S_interface, S_unit,  S_begin, S_end,
              S_label, S_const, S_type, S_var, S_procedure, S_function,
            S_integer , S_real , S_char, S_array,  S_record, S_pointer,
             S_lt , S_gt , S_eq , S_le , S_ge, S_ne, S_add, S_sub, S_or, S_mul, S_div, S_and,
           S_id, S_no, S_not, S_comma, S_colon, S_semicolon, S_dot,
            S_OpBracket, S_ClBracket, S_OpCurlyB, S_ClCurlyB, S_OpSquB, S_ClSquB );

                                                    Var
CurrentSymbol  :  Symbols ;
```

پس از اينكه تحليلگرلغوي اولين لغت را از برنامه ورودي تشخيص و در داخل CurrentSymbol قرار داد ، ميبايست روال ProgramX فراخواني شود. در واقع ProgramX نام سر ترم گرامر است چرا كه در اين روش براي هر ترم مياني و سر ترم روالي به همان نام نوشته ميشود. روال ProgramX بر اساس قاعده مربوطه نوشته شده است. قاعده مربوط به ProgramX در زير با يك جمله تفسيري كامنت مشخص شده است:

```
(* Program  X  →   Program id ';' BlockBody  '.' * )
```

```
Procedure        } روال تشخیص سرترم گرامر
                                                    ProgramX ; {
    Begin
      Expect ( S_Program ) ;   { انتظار مشاهده S_Program در ورودي ميرود }
       Expect ( S_ id ) ;    { انتظار مشاهده S_Id در ورودي ميرود }
      Expect ( S_Semicolon ) ; { انتظار مشاهده S_Semicolon در ورودي ميرود }
BlockBody ;  {فراخواني روال ترم مياني BlockBody براي تشخيص بدنه برنامه
                                                                    {
      Expect ( S_dot ) { انتظار مشاهده S_Semicolon در ورودي ميرود }
    End ;
```

همانطوریکه مشاهده میشود اکنون در ابتداي روال ProgramX بنا بر قاعده مربوطه ، انتظار مشاهده لغت از نوع S_Prgram میرود. براي اين منظور تابع Expect با پارامتر S_Program مورد فراخواني قرار گرفته است. ترم پيش بيني قبل از اجراء دستورالعمل Expect (S_Program) در داخل برنامه اصلي و با فراخواني NextSymbol در داخل CurrentSymbol قرار داده شد. بنابراين محتوي CurrentSymbol در داخل Expect بنابرگرامر با S_Program مقايسه ميشود. کد اين روال بصورت زير است :

```
  Procedure EXPECT ( S : Symbols ) ;
  Bigin
If CurrentSymbol = S    { اگر ترم پيش بيني مساوي با پارامتر ارسالي است
                                                                    {
Then  NextSymbol        { آنگاه لغت بعدي بعنوان ترم پيش بيني خوانده شود
                                                                    {
   Else  SyntaxError    { وگرنه پيام خطا نحوي صادر شود }
   End ;
```

اکنون مي توان روال مربوط به ترم مياني BlockBody را نوشت. در ابتدا بايد قواعد مربوط به اين ترم مياني را مشخص نمود تا بتوان بر اساس آن گرامر روال مربوطه را مشخص کرد :

```
( * Blockbody → [ ConstantDef.part ]
                  [ typeDef.Part ]
                 [VarDefPart ]
                  {FunctionDef | ProcedureDef }
                  Compaund Statement * )
Procedure  BlockBody ;
          Begin
        if  CurrentSymbol =  S_const
            then  ConstantDef.Part ;
        if  CurrentSymbol =  S_type
           then  TypeDefPart ;
        if  CurrentSymbol =  S_Var
                  then  VarDefPart  ;
          While ( current Symbol =  S_ Procedure  |  Current symbol =  S_Function  )
                do if  CurrentSymbol  =  S_Procedure
                       then  ProcedureDef.
                     else  FunctionDef.
          Compound Statement  ;
    End ;
```

درگرامرفوق براكت علامت تكرار صفر و يا يك و آكولاد علامت تكرار صفر يا بيشتر است. نكته قابل توجه اين است كه در داخل ProgramX هنگاميكه BlockBody فراخواني ميشود . لغت بعدي در داخل CurrentSymbol قرار دارد. لذا، اگر اين لغت S_Const باشد روال مربوط به ثابتها فراخواني ميشود. اين روال بخش تعريف ثابتها را بنا بر گرامر ، مورد تحليل نحوي قرار ميدهد و در خاتمه لغت بعدي را در داخل CurrentSymbol قرار ميدهد . همين امر موجب ميشود كه تحليلگر نحوي بتواند بسادگي به كار خود ادامه دهد.

نكته قابل توجه در مورد تشخيص روالها و توابع است . به هر تعدادي و يا هر تركيبي از آنها را ميتوان داشت لذا در داخل يك حلقه While پس از اينكه يك روال يا تابع تشخيص داده ميشود بايد مطمئن بود كه لغت بعدي در CurrentSymbol است كه اگر S_Procedure يا S_Function باشد در داخل حلقه مربوط فراخواني ميشود :

```
  ( * ConstantDefPart → Const  ConstantDef {ConstantDef } *)
Procedure                Part;
                                                    ConstantDef
       Bigin
     Nextsymbol ;
          ConstantDef ;
      While  CurrentSymbol = S_id
           Do ConstantDef ;
End ;
```

```
( * constant Def  → id '='  ( No | id ) ' ; '  *
                                                              )
Procedure  ConstantDef
                                                              ;
        Begin
    Expect ( S_id ) ;
    Expect ( S_EQ ) ;
     If  CurrentSymbol  =  S_No
            Then  Nextsymbol
            else  Expect ( S_id ) ;
     Expect ( S_Semicolon ) ;
End ;
```

در بخش قبلي گرامر عبارات به برم LL(1) تبديل شد تا بتوان جدول تجزيه براي تحليلگر نحوي پيش بيني كننده براي تشخيص عبارات ايجاد نمود. براي ايجاد تحليلگر كاهينه بازگشتي بهتر است كه گرامر در فرم توسعه يافته بصورت LL(1) تبديل شود. گرامر عبارات در فرم توسعه يافته بصورت زير است :

Expression → SimpleExp {RelOp SimpleExp}

RelOp → < | <= | = | <> | >= | > | IN

SimpleExp → Term { ('+' | '-' | Or) Term }

Term → Factor { ('/' | '*' | DIV | AND) Factor}

Factor → Number | NOT Factor | '(' Expression ')' | Variable

Variable → Identifier { '[' Dim ']' }

Dim → Expression { ‘ ,’ Expression }

حالا میتوان بر اساس گرامر فوق تجزیه گر کاهینه بازگشتي را بصورت زیر ایجاد کرد :

```
Begin   Init;   NextSymbol; Expression;
                                                                End.

(*Expression      →    SimpleExp    {RelOp
                                                              SimpleExp}*)
Procedure
                                                              Expression;

                                                                Begin

                                                              SimpleExp;
  while CurrentSymbol in First(RelOp)
                                                                  do

        Begin RelOp ; SimpleExp  End;
      End;
(* RelOp → < | <= | = | <> | >= | > | IN
                                                                  *)
     Procedure RelOp;

Begin          if   CurrentSymbol   in      [S_Lt    ,   S_Le,   S_Eq,   S_Ne,   S_Ge,   S_Gt   ]

     then
                                                              NextSymbol
     else
                                                              Expect(S_In);

                                                                End;
(* SimpleExp  → Term { ( ‘+’  | ‘-’  |  Or  ) Term } *)

Procedure
                                                              SimpleExp;

                                                                Begin
          Term;
          while CurrentSymbol  in  [S_Add, S_Sub, S_Or)
             do  begin  NextSymbol;  Term
                                                                End;

                                                                End;
(* Term  → Factor  { (‘/’  |  ‘*’  |  DIV  | AND)  Factor}
                                                                  *)
Procedure
                                                              Term;

                                                                Begin
 Factor;
```

```
    while CurrentSymbol in [S_Div, S_Mul, S_IntDiv, S_And]
          do  begin  NextSymbol;    Factor
                                                                    End;

                                                                    End;
(*Variable  ⟶  Identifier {  '[' Dim ']'
                                                                    }*)
    Procedure Term;

                                                                    Begin

                                                                    Expect(S_Id);
        while CurrentSymbol =  S_OpenSquareBracket
              do begin
                 NextSymbol;  Dim;  Expect ( S_CloseSquareBracket );
                  End;

                                                                    End;
```

٤.٨ بهبود از خطاء

يكي از دامنه هاي تحقيقاتي بسيار وسيع در زمينه كامپايلر ها كه حتي امروزه به هوش مصنوعي نيز ارتباط پيدا كرده است ، مسئله بهبود از خطا است . مسئله اينجا است كه يك كامپايلر قادر باشد پس از مشاهده خطاي نحوي در متن برنامه داده شده به درستي بكار خود ادامه دهد و حد اكثر تعداد خطارا در يك مرحله از كامپايل تشخيص دهد .

نكته اين است كه براي نمونه اگر اشتباها" در برنامه اي بجاي If برنامه نويس اشتباها" Uf وارد كند ، كامپايلر با مشاهده لغت Uf كه يك شناسه است به جاي If ، گمراه نشده و پيامهاي خطاي اضافي صادر نكند و قادر باشد كه در يك مرحله از كامپايل حد اقل تعداد ثيام لازم براي حد اكثر تعداد خطا را ايجاد نمايد .

براي درك بهتر مسإله بهبود از خطا، در زير يك مثال ارائه ميشود. براي اين منظور روالهاي زير را كه قبلاً جهت تشخيص ثابتها ارائه شده بود در نظر بگيريد :

```
(  *      →  Const  ConstantDef  {ConstantDef  }    *)
                                                          ConstantDefPart
Procedure  ConstantDef Part
                                                          ;

                                                          Bigin
  NextSymbol ;
        ConstantDef ;
            While  CurrentSymbol = S_id
```

```
Do  CostantDef ;

                                                                    End ;

( *     → id   '='   ( No | id ) ' ; '   * )
                                                                    ConstantDef
Procedure  ConstantDef
                                                                    ;

                                                                    Begin
   Expect ( S_id ) ;
   Expect ( S_EQ ) ;
         If  CurrentSymbol  =  S_No
       Then  Nextsymbol
       Else  Expect ( S_id ) ;
   Expect ( S_Semicolon ) ;
     End ;
```

اکنون به قطعه کد مقابل که داراي خطاء نحوي است توجه نمائید .

```
    Const
  a := 5
                                                                    ;
  b  =
                                                                    ;
  c  = 8
                                                                    ;
```

براي تجزیه قطعه کد فوق روال ConstantDef فراخواني میشود. در ضمن عمل تجزیه هنگامي که پس از ترم پایانی a در ورودي انتظار مشاهده S_EQ میرود، برخلاف انتظار لغت := از نوع S_Assigin ظاهر میشود بنابراین در داخل Expect روال SyntaxError فراخواني میشود. در اینجا نکته چگونگی عملکرد SyntaxError است . اگر SyntaxError بصورت زیر نوشته شود :

```
Procedure SyntaxError ;
    Begin
     Error ( " Syntax ", LineNo , ColNo )  ;
    End;
```

چون در این حالت NextSymbol در داخل Expect مورد فراخواني قرار نمي گیرد ، محتواي CurrentSymbol همان S_Assigin باقي خواهد ماند. پس از اینکه پیغام خطا در سطر LineNo و ستون ColNo از متن برنامه مورد کامپایل اعلام شد ، کنترل به روال Expect و پس از آن به فراخواننده Expect یعني ConstantDef میرسد. امّا ، تغییر نکردن مقدار ترم پیش بیني یعني مقدار CurrentSymbol موجب میشود که به غلط پیام خطاء دومي اعلام شود. به این ترتیب زماني که دستورالعمل Expect (S_No) اجراء میشود چون محتواي CorrentSymbol مساوي با S_No نبوده و مساوي با S_Asigin است لذا ، مجدداً به غلط پیام خطاي دیگري صادر میشود و به همین ترتیب پیامهاي

خطاي بعدي ، يكي پس از ديگري صادر مي شوند. اگر NextSymbol بصورت زير در داخل SyntaxError فراخواني شود :

```
Procedure  SyntaxError ;
   Begin
        Error ( " Syntax ", LineNo , ColNo ) ;
     NextSymbol ;
   End;
```

آنگاه پس از اعلام پيام خطا ، لغت بعدي يعني عدد 5 كه از نوع S_No است از ورودي خوانده مي شد. به اين ترتيب با تغيير CurrentSymbol به S_No ، روال ConstantDefبه درستي ميتوانست بكار خود ادامه دهد. با وجود اين ، افزايش NextSymbol يا بعبارت ديگر چشم پوشي از مورد خطا كه در اينجا S_Assigin بود همواره مشكل گشا نيست. براي مثال در سطر بعدي يعني b = ; پس از تشخيص = يا S_EQ محتواي CurrentSymbol به غلط S_Semicolon خواهد بود. و در زماني كه دستورالعمل Expect (S_No) اجرا ميشود محتواي CurrentSymbol ، S_Semicolon است و در اينجا چشم پوشي از اين لغت يعني S_Semicolon موجب خطا ميشود و بهتر است كه NextSymbol از داخل روال SyntaxError حذف شود .

چه بايد كرد ؟ مشاهده نموديد كه در يك مورد وجود NextSymbol در داخل SyntaxError براي چشم پوشي از مورد خطاي نحوي و در واقع لغت غير قابل انتظار ، ضروري است و در مورد ديگر اين عمل غلط ميباشد . براي رفع اين مشكل در داخل هر قاعده براي هر ترم بطور مجزاء.مجموعه اي از ترمها كه پس از آن ترم در قاعده اي ظاهر مي شوند را به عنوان مجموعه STOP يا متوقف كننده در نظر گرفته مي شود. اكنون در داخل SyntaxError آنقدر ترمهاي بعدي خوانده شده و از آنها چشم پوشي مي شود تا طبق قاعده بتوان به يكي از ترمهاي بعدي مورد انتظار در داخل مجموعه Stop رسيد .

براي نمونه طبق گرامر در داخل ConstantDef پس از S_Id انتظار ديدن S_EQ و پس از آن انتظار S_Semicolon در ورودي ميرود. پس از S_Semicolon مسلماً بايد ترمهايي كه پس از ConstantDef ميتوانند ظاهر شوند ، قرار گيرند. اين ترمها شامل S_id كه شروع كننده ConstantDef ديگري است و همين طور ترمهايي كه بعد از خود ConstantDef.Part ميتوانند ظاهر شوند ، است. به اين ترتيب روالها را ميتوان بصورت زير باز نويسي كرد :

```
( *    →  Const     ConstantDef.  {ConstantDef }     *)
                                                  ConstantDefPart
   Procedure  ConstantDef Part  ( Stop : Set of Symbols ) ;

                                                       Bigin
  NextSymbol ;
   ConstantDef  ( [ S_id ] + Stop ) ;
   While  CurrentSymbol = S_id
```

```
        Do  CostantDef  ( [ S_id ] + Stop ) ;

                                                                End ;

( *   → Id '='   ( No | id ) ' ; '   * )
                                                        ConstantDef
Procedure  ConstantDef  ( Stop: Set of  Symbols )
                                                                ;

                                                                Begin
      Expect ( S_id  , [ S_EQ , S_No , S_Id , S_Semicolon ] + Stop ) ;
     Expect ( S_EQ , [ S_No , S_id , S_Semicolon ] + Stop ) ;
        If  CurrentSymbol  =  S_No
      Then  NextSymbol
       Else  Expect ( S_No ,  Stop + [ S_Semicolon ] ) ;
   Expect ( S_Semicolon  ,  Stop ) ;

                                                                End ;

Procedure EXPECT ( S : Symbols ,  Stop : Set of  Symbols
                                                                ) ;

                                                                Begin
 if  Currentsymbol = S
          Then   Nextsymbol
          else    SyntaxError( Stop ) ;

                                                                End;

Procedure  SYNTAXERROR  ( Stop : Set of Symbols
                                                                ) ;

                                                                Begin
    Error ( ' Syntax ' , LineNo , ColNo
                                                                ) ;
   While not  ( CurrentSymbol   in  Stop  )  DO  NextSymbol ;

                                                                End;
```

در حالت کلي واضح است که براي محاسبه مجموعه Stop بصورت زير عمل ميشود :

الف - چنانچه : $E \rightarrow e_1\ e_2\ e_3\ \ldots\ e_n$ آنگاه :

$$Stop(e_i) = \sum_{i=1\ldots n-1} First(e_j)_{j=(i+1)\ldots n} + Stop(E)$$

$$Stop(e_n) = Stop(E)$$

ب - چنانچه : $E \rightarrow e_1 \mid e_2 \mid \ldots\ldots\ldots \mid e_n$ آنگاه :

$$Stop(e_i) = Stop(E) \qquad i = 1\ldots n$$

ج - چنانچه : $E \rightarrow \{ e_1 \}$ آنگاه :

$$Stop(e_1) = Stop(E) + First(e_1)$$

به این ترتیب برنامه تحلیلگر لغوي که قبل از این نوشته شده بود بصورت زیر تبدیل میشود :

```
Begin
  Init ;   //   عملیات مقدماتي
Nextsymbol ;          //  گرفتن لغت بعدي از تحلیلگر لغوي
ProgramX ( [ S_EOF ]       //  انتظار سرترم و علامت خاتمه فایل پس از آن
                                                                    )

End.
```

همانگونه که مشاهده مي نمایید ، پس از سر ترم گرامر ، انتظار مشاهده علامت پایان فایل یا S_EOF مي رود زیرا ، هر جمله ورودي به کامپایلر در داخل یك فایل ظاهر میشود که با علامت خاتمه فایل انتهاي آن مشخص میشود. بنابراین پس از هر جمله ورودي علامت خاتمه فایل قرار میگیرد. جمله ورودي در صورت صحیح بودن از سرترم گرامر مشتق میشود. بنابراین پس از سرترم گرامر همواره انتظار مشاهده علامت خاتمه فایل میرود.

```
    (* ProgramX → Program id ';' CompoundSt '.' *)
       Procedure  ProgramX ( Stop : Set of  Symbols ) ;
 Begin
         Expect ( S_program , [ S_id , S_Semicolon ] + First ( Blockbody ) + [ S_dot ] + Stop ) ;
        Expect ( S_id , [ S_Semicolon , First ( Blockbody ) + [ S_dot ] + Stop ) ;
       Expect ( S_Semicolon , First ( Blockbody ) + [ S_dot ] + Stop  ) ;
      Blockbody ( [ S_dot ] + Stop ) ;
     Expect ( S_dot , Stop ) ;
End;
   ( *  Blockbody  →  [ ConstantDef.part ][ typeDef.Part ][VarDefPart ]
                        {FunctionDef  | ProcedureDef  } Compaund Statement * )
  Procedure  BlockBody ( Stop : Set of  Symbols);
        Begin
        if  CurrentSymbol =  S_const
        then  ConstantDef.Part(Stop + [S_Type, S_Var, S_Procedure, S_Function, S_Begin] );
        if  CurrentSymbol =  S_type
       then  TypeDefPart( Stop + [ S_Var, S_Procedure, S_Function, S_Begin] );
   if  CurrentSymbol =  S_var
       then  VarDefPart( Stop + [ S_Procedure, S_Function, S_Begin] );
     while ( current Symbol =  S_ Procedure  |  Current symbol =  S_Function  )
          do if  CurrentSymbol  =  S_Procedure
                then  ProcedureDef ( Stop + [ S_Procedure, S_Function, S_Begin] )
                else  FunctionDef ( Stop + [ S_Procedure, S_Function, S_Begin] );
   Compound Statement( Stop ) ;
End ;
```

مثال - گرامر زیر را به فرم LL(1) تبدیل نموده ، یك تحلیلگر كاهینه بازگشتي براي آن ایجاد نمائید.

$S \rightarrow aB \mid aC \mid dD$

$D \rightarrow Da \mid Db \mid d$

$B \rightarrow BC \mid b$

$C \rightarrow Cd \mid d$

براي تبدیل به LL(1) فرم توسعه یافته ساده تر است. در مورد S عمل فاكتور گیري چپ و در مورد D پس از عمل فاكتور گیري چپ ، باید حالت خود بازگشتي چپ حذف شود. در مورد سایرترم هاي گرامر باید حالت خودبازگشتي چپ از داخل قاعده حذف شود.

$S \rightarrow a\,(B \mid C) \mid dD$

$D \rightarrow d\ \{a \mid b\}$

$B \rightarrow b\ \{C\}$

$C \rightarrow d\ \{d\}$

با جایگزیني گسترش C در $B \rightarrow b\ \{C\}$ این قاعده بصورت زیر تبدیل میشود:

$B \rightarrow b\ \{d\}$

در بین قواعد فوق ، تنها قاعده $S \rightarrow a\,(B \mid C) \mid dD$ ممكن است عاملي براي LL(1) نشدن گرامر باشد. در سمت راست این قاعده 'B | C' را در نظر بگیرید. باید First(B) $\cap$ First(C) $= \varnothing$ باشد.

$$First(B) \cap First(C) = \{b\} \cap \{d\} = \varnothing$$

بنابراین گرامر LL(1) است.

```
Begin   init; NextSymbol;   S(S_EOF);
                                                                End.
(* S  →  a (B | C ) |
                                                                dD *)
Procedure S( Stop : Set of Symbols);


                                                                Begin
 If(CurrentSymbol = S_d)
        Then Begin  NextSymbol; D( Stop );  End
    Else Expect(S_a, [S_b, S_d] + Stop);


                                                                End;
(* D → d  { a  | b
                                                                } *)
    Procedure D( Stop : Set of Symbols);


                                                                Begin
 Expect(S_d, [S_b, S_d] + Stop);
  While(CurrentSymbol = S_a ) Or (CurrentSymbol = S_b) do
   If     CurrentSymbol   =   S_a   Then
                                                                NextSymbol
   Else Expect(S_a, [S_b, S_d] + Stop);
```

```
                                                                    End;
(*B → b { d }*)
    Procedure B( Stop : Set of Symbols);

                                                                    Begin
Expect(S_b,   [   S_d]   +
                                                                    Stop);
 While(CurrentSymbol = S_d ) do NextSymbol;

                                                                    End;
(*B → b { d }*)
    Procedure C( Stop : Set of Symbols);

                                                                    Begin
Expect(S_d,   [   S_d]   +
                                                                    Stop);
 While(CurrentSymbol = S_d ) do NextSymbol;

                                                                    End;
```

٤.٩ مولد تحليلگر نحوى

مولد تحليلگر نحوي برنامه اي است كه گرامر يك زبان را دريافت نموده و براي آن يك تحليلگر نحوي يا تجزيه گر ايجاد مي كند. در اين قسمت نوعي خاص از مولد ارائه شده است كه گرامر زبان را در داخل يك فايل ميپذيرد و بر اساس گرامر عمل تحليل نحوي را انجام ميدهد. اين مولد در واقع يك برنامه پيمايش گراف است. هر قاعده درگرامر LL(1) را ميتوان بعنوان يك گراف در نظر گرفت.

دربرنامه مولد تحليلگر نحوي كه درزير ارائه شده ، گرامر درون يك فايل متن يا در اصطلاح Text بنام Grammar.txt قرار دارد. براي قرار دادن گرامر در داخل اين فايل در هر سطر ابتدا ترم سمت چپ قاعده و بلافاصله گسترش آن مشخص ميشود. براي سادگي و خواناتر شدن برنامه فرض شده كه هر ترم مياني و سرترم با يك حرف لاتين درشت و هر ترم پايانى با يك حرف كوچك مشخص شده است. به اين ترتيب براي نمونه گرامر ساده :

S → cA | BdS
A → aAB |
d
B → bB |
d

بصورت زير در فايل Grammar.txt ذخيره ميشود :

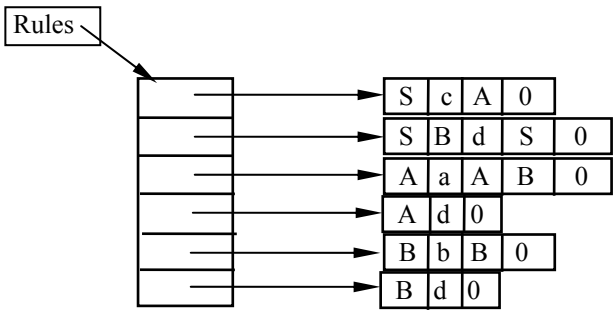
6
ScA
SBdS
AaAB
Ad
BbB
Bd

در اولين سطر از فايل Grammar.txt تعداد قواعد يا در واقع تعداد سطرهاي فايل مشخص شده است. بدنه برنامه اصلي main در زير ارائه شده است. بازهم بخاطر سادگي و خوانايي برنامه متن برنامه مورد كامپايل در داخل يك رشته بنام Statement در نظر گرفته شده است.

```
        void main()
{  int NoRules,        تعداد قواعد در گرامر
                                                    //
   len;                // تعداد ترمها در جمله ورودي
   char **rules,        ماتريس قواعد گرامر
                                                    //
```

```
*statement;   جمله ورودي كامپايلر
                                                                    //
// توليد ماتريس قواعد گرامر در آرايه دو بعدي
                                                                    1- rules
        NoRules = ReadGrammar(&rules, statement);
تحليل نحوي جمله داده شده statement با استفاده از ماتريس قواعد rules
                                                                    // 2-
        len = TopDownParse(rules, statement, 0, NoRules, 0);
//3. اعلام پيام خطا در صورتيكه كار تحليل نحوي تا آخر جمله داده شده ادامه پيدا نكرده
باشد.
    if(len != strlen(statement)) { clrscr(); puts("خطا نحوي"); }
            }
```

در اولين سطر از برنامه اصلي تابع ReadGrammar فراخواني ميشود. اين تابع گرامر را از داخل فايل Grammar.Txt خوانده ، در داخل يك آرايه از نشانگرها به آرايه ها قرار ميدهد. آدرس شروع آرايه نشانگرها در مكان مشخص شده توسط پارامتر Prod از تابع ظاهر ميشود. هر آرايه حاوي يك قاعده گرامر خواهد بود. به عنوان نمونه براي گرامري كه در ابتداي اين بخش ارائه شد ساختار آرايه كه Rules نام دارد به صورت زير است :



كد تابع ReadGrammar در زير ارائه شده است.

```
   int ReadGrammar(char ***prod, char *statement)
گرامر را از فايل ورودي به ماتريس مربوطه انتقال ميدهد . ازانتهاي فايل جمله ورودي
را ميخواند // {
          FILE *fp;
  char line[100], **rules;
         int i, NoRules;
         fp = fopen("Gammar2.txt","rt");
        if(!fp) fp = stdin;
         if(fp == stdin)
       {clrscr(); printf("NO. Rules : ");}
       fscanf(fp,"%d", &NoRules);
ايجاد آرايه از نشانگرها بسول تعداد قواعد گرامر باضافه
                                                                    يك //
    rules =(char **) malloc((NoRules+1) * sizeof(char *))  ;
        rules[NoRules] = NULL;
     for(i = 0; !feof(fp) && i < NoRules; i++)
قرار دادن قواعد درون ماتريس {
                                                                    // Rules
```

```
        fscanf(fp,"%s",line);
    rules[i] = malloc(strlen(line)+1);
        strcpy(rules[i], line);
    }
     خواندن جمله مورد
                                                                    کامپایل //
    if(!feof(fp)) fscanf(fp, "%s", statement) ;
*prod = rules;    // برگرداندن آدرس ماتریس قواعد در
                                                                    Prod
        برگرداندن تعداد قواعد بعنوان خروجي تابع return
                                                                NoRules; //
          }
```

اکنون با تشکلیل ماتریس قواعد درRules و خواندن جمله مورد کامپایل در رشته Statement میتوان با فراخواني تابع TopDownParse عمل تحلیل نحوي را انجام داد. این تابع براي انجام عمل تحلیل نحوي از Rules[start] شروع مي کند. در آغاز کار مقدار start مساوي با صفر است لذا، در آغاز کار تحلیلگر نحوي با سرترم گرامر آغاز میشود. ترم پیش بیني براي این تابع در Statement[ix] قرار گرفته است.

تابع تحلیلگر

نحوي //

```
int TopDownParse(char **rules, char *statement, int start, int NoRules, int ix)
    پارامتر Rules حاوي قواعد گرامر
                                                                    است .// {
      پارامتر start حاوي شماره قاعده ترم مياني مورد
                                                                انتظار است .//
    پارامتر ix حاوي انديس ترم پيش بيني در جمله ورودي
                                                                    است//
         int i, j, k, m, NewK;
        char leftterm, input;
        //1- Match the leftmost
     input = statement[ix];  // تعیین ترم پیش بیني در متغیر
                                                                    Input
با فراخواني تابع StartWhichRules مقدار i نشان میدهدکه در داخل آرایه //
                                                                    Rules
  کدام گسترش از گسترش‌هاي ترم مورد انتظار يعني rules[start] با ترم پيش بيني Input
آغاز میشود.//
      i = StartWhichRule(rules, input, start, NoRules);
          if(i < 0) return ix;
      // پیمایش سمت راست قاعده
                                                                rules[i][0]
    for( j = 1, k = ix; rules[i][j] && statement[k] && i < NoRules;  j++)
// اگر ترم بعدي در سمت راست قاعده يك ترم مياني است آنگاه
            if(isupper(rules[i][j]))
                {
جستجو براي يافتن قاعده مربوط به ترم مياني ظاهر شده در ضمن پيمايش سمت راست
                                                                    // 1-
```

```
        for(m = 0; rules[m][0] != rules[i][j] && m < NoRules; m++);
        NewK = TopDownParse(rules, statement, m, NoRules, k);
          if(k == NewK) return 0; else k = NewK;}
    وگرنه ترم بعدي در سمت راست قاعده يك ترم پايانی
                                                                    است //
  else    خواندن ترم پايانی بعدي در ورودي
                                                                    // 2.
          if(rules[i][j] == statement[k]) k++;
        return k;
  }
```

با فراخواني تابع StartWhichRule مشخص ميشود كه ترم پيش بيني input كدام گسترش از گسترش‌هاي ترم مورد انتظار يعني rules[start][0] را آغاز ميكند. در صورت يافتن گسترش مورد نظر شماره انديس قاعده آن در متغير i برگردانده ميشود. به اين ترتيب rules[i][0] بايد مساوي با rules[start][0] باشد. با يافتن يك قاعده مناسب ، سمت راست آن قاعده مورد پيمايش قرار ميگيرد. در ضمن پيمايش چنانچه تحليلگر به يك ترم مياني برسد ، با فراخواني خود بطور خود بازگشتي بر مبناي آن ترم مياني عمل تحليل نحوي را ادامه ميدهد. پس از اين فراخواني NewK نمايانگر انديس ترم پيش بيني درجمله ورودي يعني Statement است. واضح است كه مقدار اين انديس بايد متفاوت از انديس قبل از فراخواني خودبازگشتي تابع يعني K باشد.

تابع StartWhichRule مشخص ميكند كه ترم پيش بيني input آغاز كننده كدام گسترش از گسترش‌هاي متفاوت ترم rules[start][0] است. براي اين منظور چنانچه LeftNonTerm مساوي با rules[start][0] قرار داده شود، تحليلگر بدنبال گسترشی از اين ترم مياني است يا به عبارت ديگر بدنبال يك rules[i] ميگردد كه اولا" rules[i][0] مساوي با LeftNonTerm باشد و ثانياً ترم rules[i][1] يا ترم مورد پيش بيني يعني input آغاز شود. اگر rules[i][1] يك ترم مياني باشد ، تابع بصورت خود بازگشتي فراخواني ميشود تا مشخص شود آيا آن ترم مياني با ترم پيش بيني يعني input آغاز ميشود. اگر موفق نشد به سراغ گسترش ديگر ترم مياني ميرود. در اين مرحله ميتوان با استفاده از مجموعه سرآغاز كار تحليلگر را تسريع نمود. برنامه كامل مولد تحليلگر نحوي در زر ارائه شده است.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <malloc.h>
#include <ctype.h>


int ReadGrammar(char ***prod, char *statement)
        {
        FILE *fp;
char line[100], **rules;
```

```
        int i, NoRules;
         fp = fopen("Gammar2.txt","rt");
           if(!fp) fp = stdin;
            if(fp == stdin)
         {clrscr(); printf("NO. Rules : ");}
          fscanf(fp,"%d", &NoRules);
      rules =(char **) malloc((NoRules+1) * sizeof(char *));
           rules[NoRules] = NULL;
         for(i = 0; !feof(fp) && i < NoRules; i++)
                  {
             fscanf(fp,"%s",line);
           rules[i] = malloc(strlen(line)+1);
             strcpy(rules[i], line);
               }
         if(!feof(fp)) fscanf(fp, "%s", statement);
            *prod = rules;
            return NoRules;
    } // End of ReadGrammar

  //This function matches the leftmost term with input
int  StartWhichRule(char **rules, char input, int start, int NoRules)
       { int success, i, k;
         char LeftNonTerm;

        LeftNonTerm = rules[start][0];
     //1- Look for a left terminal in the production rule
         i = start;
      while((rules[i][1] != input) &&
                                  (rules[i][0] == LeftNonTerm)) i++;
 //Hint: here is a catch --- E R O R R -- Find it out..
      if(rules[i][1] == input)
         return i;
    //2- Look for a left nonterminal in the production rule
       for(i = start, success = -1;
        rules[i][0] == LeftNonTerm && success < 0;
          i++) if(isupper(rules[i][1]))
              { for(k = 0; rules[k][0] != rules[i][1] && k < NoRules; k++);
                success = StartWhichRule(rules, input, k, NoRules);
                    }
      if(success < 0 ) return success;
  return I-1;
        }


int TopDownParse(char **rules, char *statement, int start, int NoRules, int ix)
        {
int i, j, k, m, NewK;
       char leftterm, input;

//1- Match the leftmost
       input = statement[ix];
    i = StartWhichRule(rules, input, start, NoRules);
if(i < 0) return ix;

        for( j = 1, k = ix;
```

```
        rules[i][j] && statement[k] && i < NoRules; j++)
          if(isupper(rules[i][j]))
    {    for(m = 0; rules[m][0] != rules[i][j] && m < NoRules; m++);
        NewK = TopDownParse(rules, statement, m, NoRules, k);
           if(k == NewK) return 0; else k = NewK;}
         else if(rules[i][j] == statement[k]) k++;
           return k;
           }


void main()
        { int NoRules, len;
          char **rules, *statement;
       NoRules = ReadGrammar(&rules, statement);
     len = TopDownParse(rules, statement, 0, NoRules, 0);
     if(len != strlen(statement)) { clrscr(); puts("ERROE");  }
           }
```

٤.١٠ تمرین

تمرین 1- گرامر زیر براي ساختار لیست را در نظر بگیرید :

$S \rightarrow (L) \mid a$
$L \rightarrow L , S \mid S$

این گرامر را به فرم LL(1) تبدیل نموده براي آن جدول تجزیه بالا به پائین ایجاد کنید. با استفاده از این جدول تجزیه جمله (a, (a, a)) را مورد تحلیل نحوي قرار دهید.

تمرین 2- گرامر زیر را به فرم LL(1) تبدیل نموده جدول تجزیه براي یك تحلیلگر پیش بیني کننده ایجاد نمائید.

$S \rightarrow SAB \mid AB$
$A \rightarrow Aa \mid a$
$B \rightarrow Bb \mid \varepsilon$

تمرین 3- یك الگوریتم براي تحلیل گر پیش بیني کننده ایجاد کنید که با استفاده از جدول تجزیه و یك پشته تجزیه عمل تحلیل نحوي را انجام دهد. براي این الگوریتم یك تابع بصورت زیر ایجاد کنید :

```
PredictiveParser( char **ParseTable,  int  NoRows, int NoCols)
                                                              int
```

تمرین 4- یك مولد تحلیل گر پیش بیني کننده ایجاد کنید که گرامر LL(1) را در ورودي مي پذیرد. این مولد سپس ، مجموعه هاي سرآغاز براي ترمهاي میاني را محاسبه میکند. با استفاده از مجموعه هاي سرآغاز به راحتي میتوان جدول تجزیه را تولید کرد.

تمرین 5- گرامر زیر را به فرم توسعه یافته LL(1) تبدیل نموده ، یك تحلیل گر کاهینه بازگشتي براي آن ایجاد کنید. در ضمن مسإله بهبود از خطا را نیز در نظر بگیرید.

$S \rightarrow SbB \mid SaB \mid L\ a\ A$
$L \rightarrow L\ a\ B \mid LbB \mid \varepsilon$
$A \rightarrow b\ A \mid d$
$B \rightarrow Bb \mid \varepsilon$

تمرین 6- گرامر ارائه شده در تمرین 5 را تبدیل به فرم LL(1) نمائید و سپس جدول تجزیه بالا به پائین براي آن ایجاد کنید.

تمرین 7- گرامر زیر را به فرم LL(1) تبدیل کنید :

$A \rightarrow \alpha$
$A \rightarrow A_1\ \alpha_1$
$A_1 \rightarrow A_2\ \alpha_2$

... ...

... ...

$A_n \rightarrow A\ \alpha_{n+1}$

تمرين 8- يك الگوريتم كلي براي حذف گسترش‌هاي تهي در گرامر زبانها ارائه دهيد.

تمرين 9- گرامر زير را به فرم LL(1) تبديل نموده يك تجزيه گر كاهينه بازگشتي براي آن ايجاد كنيد. مسئله بهبود از خطا را نيز در نظر بگيريد.

$S \rightarrow SAB \mid Bd$

$A \rightarrow BdA \mid dB \mid \varepsilon$

$B \rightarrow Bb \mid \varepsilon$

تمرين 10- چگونه ميتوان براي يك تجزيه گر پيش بيني كننده مسإله بهبود از خطا را در نظر گرفت. الگوريتم خواسته شده در تمرين 3 را با در نظر گرفتن مسإله بهبود از خطا تكميل كنيد.

تمرين 11- برنامه يك تجزيه گر كاهينه بازگشتي براي گرامر يكي از زبانهاي رايج برنامه نويسي ، ايجاد كنيد. مسئله بهبود از خطاء در نظر گرفته شود.

تمرين 12- مولد تحليلگر نحوي ارائه شده در انتهاي فصل را آنچنان تكميل نمائيد كه تحليلگر لغوي را فراخواني كند تا ترم بعدي را از ورودي دريافت كند. ترمهاي مياني نيز به هر صورتي در گرامر ظاهر شوند. با محاسبه اتوماتيك مجموعه First براي ترمهاي مياني و سرترم ميتوان كار مولد تحليلگر را تسريع نمود. مسإله بهبود از خطا نيز در نظر گرفته شود.

تمرين 13- گرامر زير را به فرم LL(1) تبديل نماييد و سپس با در نظر گرفتن مسئله بهبود از خطاء براي آن يك تحليلگر كاهينه بازگشتي ايجاد كنيد.

$S \rightarrow Abd \mid Bd$

$A \rightarrow Aa \mid$

a

$B \rightarrow Bb \mid$

b

تمرين 14- گرامر زير را به فرم LL(1) تبديل نموده ، يك تحليلگر كاهينه بازگشتي براي آن ايجاد كنيد.

$S \rightarrow AB \mid A\ B\ C$

$A \rightarrow A\ a \mid$

ε

$B \rightarrow b\ A \mid A\ b\ b$

$C \rightarrow C\ c \mid \varepsilon$

تمرین 15- در حالت کلي چگونه میتوان اثبات کرد که یك گرامر به فرم LL(1) قابل تبدیل است.