بنام خدا

دانشکده فنی دانشگاه تهران
گروه برق و کامپیوتر

جزوه درس طراحی و ساخت کامپایلرها

مدرس : دکتر قاسم جابری پور

فهرست مطالب :

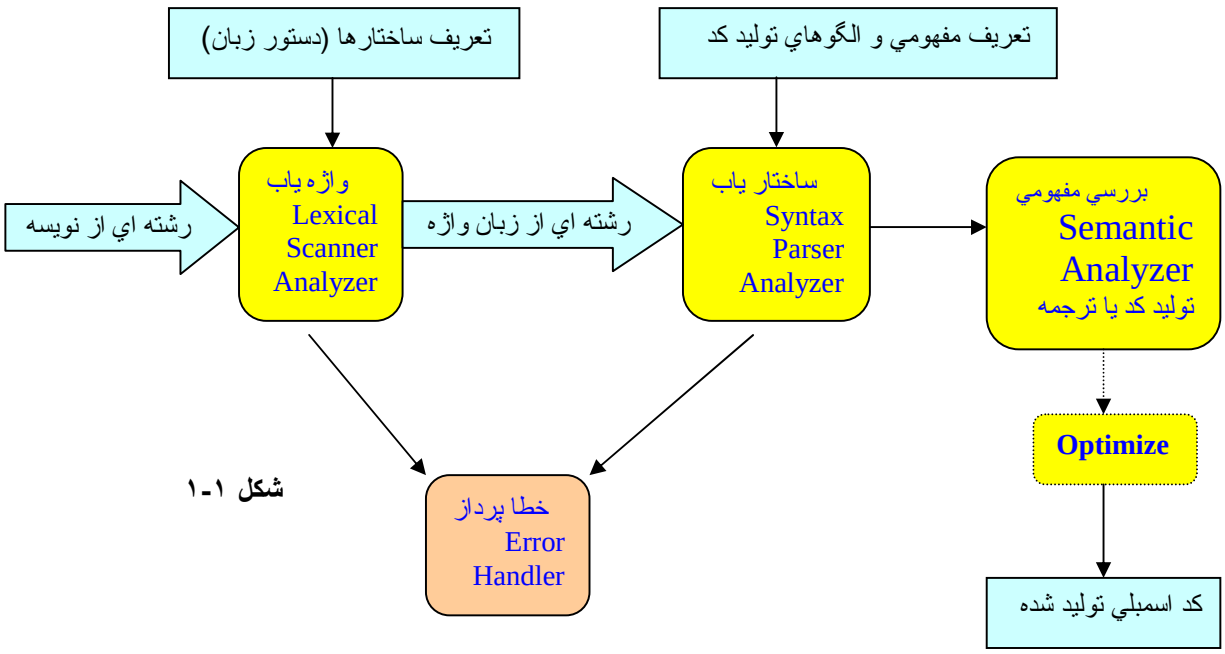
۱ـمفاهيم اوليه:

همانطور كه ميدانيد زبانها از نظر پيچيدگي به ۴ دسته تقسيم ميشوند.

۱- زبانهاي منظم Regular Languages
۲- زبانهاي مستقل از متن Context Free Languages
۳- زبانهاي حساس به متن Context Sensitive Languages
۴- زبانهاي بدون محدوديت

از بين اين دسته زبانهاي مستقل از متن براي برنامه هاي سطح بالا كه نزديك به زبان انسان باشند مناسب ترند چون مي توان با الگوريتمهاي از درجه O(n) تعلق يك عبارت را به آن بررسي كرد كه در آن n طول عبارت فوق مي باشد. تشخيص اين تعلق بوسيله عمليات Parse انجام ميشود.

كامپايلر ابزاري است كه برنامه اي به زبان سطح بالا را گرفته و معادل همان برنامه را به زبان سطح پايين بر می گرداند. يعني رشته ای از نويسه ها را كه براحتي توسط ماشين اجرا مي شود توليد مي كند. به شكل زير توجه كنيد:



شكل ۱-۱

واژه یاب مجموعه اي از نویسه ها را گرفته و زبان واژه هاي استخراج شده را در اختیار Parser قرار مي دهد.

واژه های زبان مي توا نند کلید واژه (Terminal یا Token) ما نند if,for,begin,:=,×,+,... باشند و یا شنا سه (ID) ما نند a,b,temp,get,... باشند.

تشخیص Comment و Blank و رد کردن آنها توسط واژه یاب انجام مي شود.

به مثال توجه نمایید:

مثال :

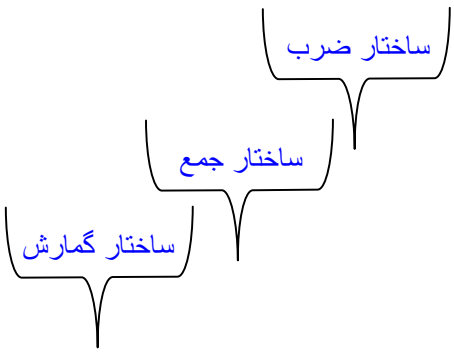
A := B + C * D

واژه های زبان:

A	: =	B	+	C	*	D
شناسه	علامت گمارش	شناسه	علامت جمع	شناسه	علامت ضرب	شناسه

ساختار ها:

A := B + C * D



توجه داریم که نوع C, D در تعیین ساختار ضرب تاثیري ندارد. زیرا در آن صورت گرامر فوق حساس به متن خواهد شد که براي O (n) نا مناسب است.در واقع عمل Check Type به بررسي مفهومي برمی گردد.

۲-واژه یاب:

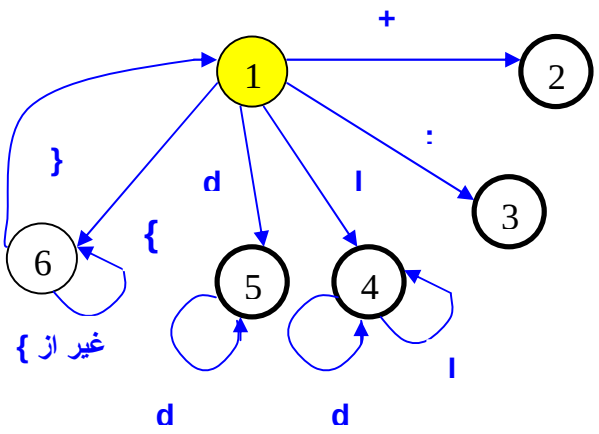
شبه کد زیر واژه یابي مشابه زبان Pascal را نشان مي دهد:

```
Function Scanner: TokenType
Begin
        Case Ch of
        '+':    Ch = getChar (); return (Plus);
        ';':    Ch = getChar (); return (Semicolon);
        ':':    Ch = getChar ();
                If Ch <> '=' then return (Colon);
                Ch = getChar (); return (Assign);
        'A'..'Z':
                IDstring = Ch;
                Ch = getChar ();
                While Ch in ['A'..'Z','0'..'9'] do
                {
                        IDstring = IDString + Ch;
                        Ch = getChar ();
                }
                Return (Id);

End
```

برنا مه ۲-۱

واژه هاي يك زبان را ميتوان با Finite Automata توليد كرد. ماشين زير براي گرمر بالا

مي باشدو همچنين Comment ها را هم رد مي كنيد:



شكل ۲-۱

در فصلهای بعدی قسمتهایی به کد فوق اضافه خواهیم کرد.

تمرین : برنامه واژه یاب را برای موارد زیر تغییر دهید :

- Commen با فرمت C /*....*/
- اعداد اعشاری (ممیز شناور)

۳- ساختار یا ب:

زبانهای برنامه سازی از نوع زبانهای Context Free هستند.

کامپایلر باید کارش را به صورت خطی (O(n)) تمام کند.

در حالت کلی برای Parse کردن زبانهای مستقل از متن بهترین الگوریتم $O(n^3)$است ، ولی زبانهای برنامه سازی حالت خاصی اززبانهای Context Free که برای آنها الگوریتم O(n) وجود دارد.

راههای توصیف یک زبان برنامه سازی :

۱- Syntax Graph

۲- Grammer

:Syntax graph

گرافی جهتدار است ، که در آن مجاز به استفاده Lambda نیستیم.

هر گراف شامل یک راس شروع ویک یا چند راس نهائی می باشد. هر گراف می تواند خود شامل چند زیر گراف باشد. در صورت رسیدن به گرههای پایانی ، اگر در گراف اصلی بودیم کار تمام است ، و اگر در یک گراف فرعی بودیم ، به گراف اصلی بر می گردیم و کار را ادامه می دهیم .

بروی یالهای گراف ،واژههای زبان ، واژههای نحوی و حداکثر یک واژه مفهومی وجود دارد.

واژههای زبان را با حروف کوچک و واژههای نحوی را با حروف بزرگ نشان می دهیم.

همراه پارسر یک پشته وجود دارد که به آن PSمی گویند ، که درآن آخرین راس ترک شده از گراف ، هنگامی که به یک زیرگراف می رویم ، قرارمی گیرد.

نکته: هرزبان ، یک گرا ف شروع وصفر یا بیشتر گرا ف فرعی دارد.

واژه زبان:

به واژه ا ی که در جملات زبان دیده می شود ، واژه زبان می گویند.

واژه نحوی :

واژه ای که در جملات زبان دیده نمی شود و برای توصیف بکارمی رود.

واژه مفهومی:

اسکنر با عبور از روی واژه مفهومی ، روال مفهومی مربوطه را فراخوانی می کند. برای شناسایی واژه مفهومی از علامت @ استفاده می کنیم.

روالهای مفهومی:

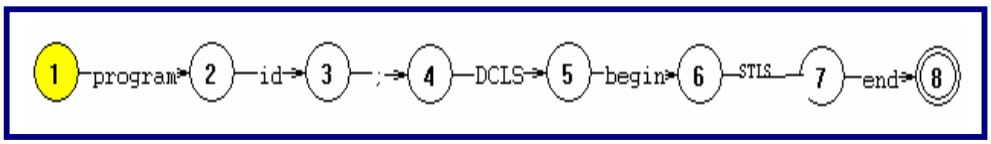
این روالها با استفاده ازعنصر بالای Semantic Stack کد لازم برای انجام یک عمل خاص را تولید کرده و آنرا در M-Table قرار می دهند.

در وا قع کد ساز مجموعه ای از روالهای مفهومی می باشد.

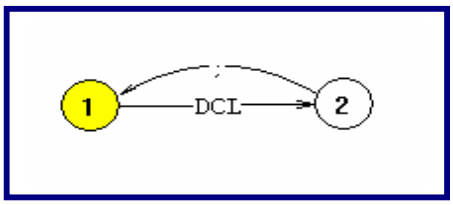
نکته:

این روال مفهومی است که سازگاری TYPE ها را بررسی می کند.

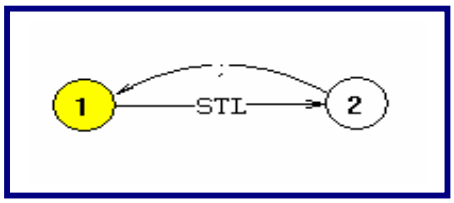
به عنوان مثال برای گرامر یک برنامه پاسکال گرافهای زیررا داریم:



DCLS :



STLS :

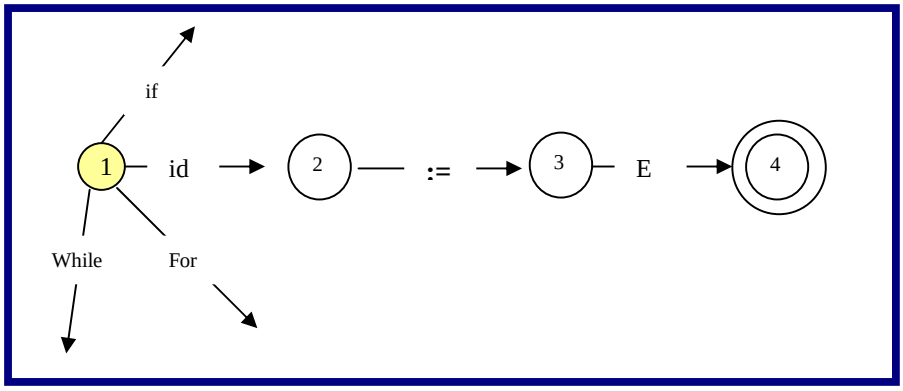


به عنوان مثال در گراف اصلی id و Program واژههای زبان و STLS وDCLS واژههای نحوی می باشند،که واژههای نحوی خود از یک گراف تشکیل شده است.

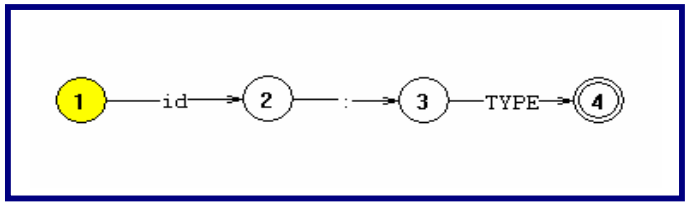
- به دلیل وجود ساختارهای Recursive (مانند if های تودرتو) نمی توان زبان را با DFA نشان داد.

زیر گراف مربوط به واژه نحوی STL به صورت زیر است :

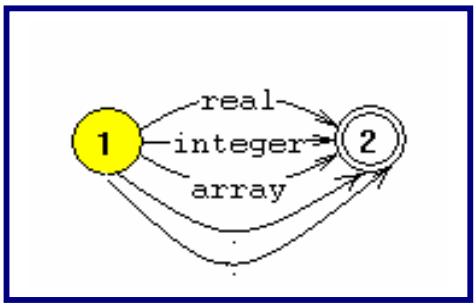
STL :



زیر گراف مربوط به واژه نحوی DCLS به صورت زیر است :



زیر گراف مربوط به واژه نحوی TYPE به صورت زیر است :

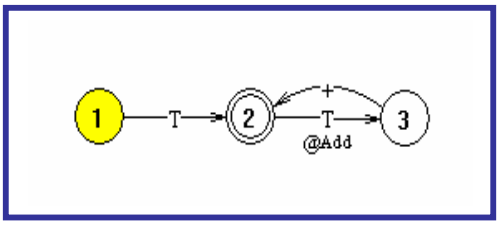


: (SS) Semantic Stack

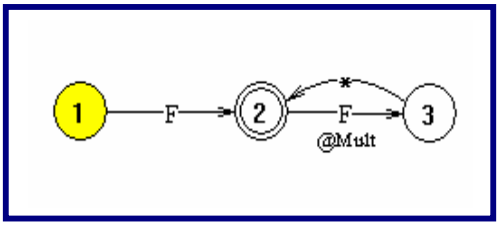
اسکنرپس از دیدن هر متغیر، شماره آن را در یک پشته قرار می دهد که به این پشته SS می گویند ،و روالهای مفهومی برای انجام عمل مورد نظرشان بر روی متغیر ها از SS استفاده می کنند، به این ترتیب که بسته به نوع عملگر، یک یا دوعنصر بالای SS را که اندیس متغیرها در Symbol Table می باشد Pop می کنند وبا مراجعه به Symbol Table نوع و آدرس متغیر را می یابد.

گراف عبارات ریاضی:

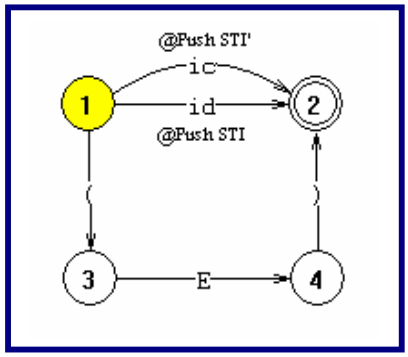
E:



T:



F:



روالهای مفهومی استفاده شده در گرافهای فوق در زیر آمده است:

```
Add:
        sti1:=Topss;
        Popss;
        sti2:=Topss;
        Popss;
        M[pc].op:='+';
        M[pc].opr1:=st[sti1].dscp.adr;
        M[pc].opr2:=st[sti2].dscp.adr;
        Release(sti1);
        Release(sti2);
        Stit:=gettemp;
        M[pc].res:=[stit].dscp.adr;
        Pushss(stit);
        St[stit].dscp.Type=RT;
        Pc++;
```

```
Push sti:
            pushss(sti);
```

```
Push sti' :
              If  Symyab [sti].DSCP := NULL   Then
                    MakeDscp (ic,ICV);
              Pushss (sti);
```

وظیفه روال MakeDscp ، ساختن Descriptor یک مقدار Integer در Symtab می باشد. روال مفهومی @Mult کاملا مشابه @Add می باشد ، با این تفاوت که به جای عملگر + ، از عملگر * استفاده می شود.

توجه :

- در روالهای مفهومی فوق ، M معرف M_Table و ST معرف Symbole Table می باشند که در زیر توضیح داده می شوند.
- کنترل سازگاری TYPE ها با توجه به عملگر داده شده توسط تابع CheckType انجام می شود که در صورت عدم تطابق کنترل به Error Handler داده می شود .
- تابع GetTemp به سراغ متغیرهای موقت در Symtab می رود و اولین متغیر خالی را پیدا کرده و اندیس آنرا بر می گرداند.
- تابع Release متغیر موقت را آزاد می کند تا مجددا قابل استفاده باشد.

تمرین :

گراف عبارت ریاضی را با / ، - و neg روالهای مفهومی @div @minus را بنویسید.

تبدیل کد میانی به کد قابل اجرا توسط برنامه مبدل (یا کامپایلر) انجام می شود و همزمان بهینه سازی کد(استفاده موثر وبهینه از رجیسترها) انجام می شود.

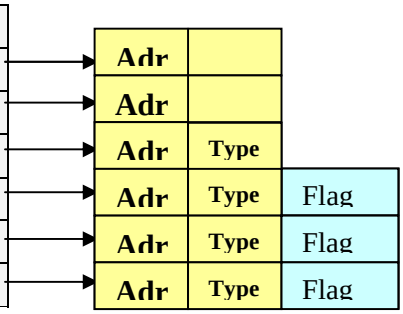
Symbole Table:

جدولی است که در آن شناسه هایی که در Declaration تعریف شده است ،در آن قرار می گیرند.

هر متغیر دارای Descriptor می باشد که بسته به نوع متغیرها DSCPهای متفاوتی داریم.

این جدول همانطور که در زیر آمده است از اسم متغیر ، شماره متغیر ، نوع متغیرو DSCP آن تشکیل شده است.

اندیس	نام متغیر	نوع متغیر	Dscp
0	A	ساده	
1	B	ساده	
2	C	ساده	
3	T1	ساده	
4	T2	ساده	
5	T3	ساده	



Dscp بطور پیش فرض از فیلد های نوع و آدرس تشکیل شده است.

درSymtab علاوه بر متغیرهای دیده شده توسط اسکنر،متغیرهای موقت که جهت قرار دادن نتایج میانی محاسبات مورد استفاده قرارمی گیرند ، نیز وجود دارد که فیلدDscp آنها دارای سه بخش آدرس و نوع وپرچم می باشد.هنگامی که یک متغیر موقت تعریف می شود، فیلد پرچم آن فعال می شود تا زمانی که این فیلد فعال است این متغیر را نمی توان دوباره تعریف کرد.

کلمات کیلدی یک زبان (مانند Begin و end در پاسکال) نیزدر Symtab قرار داده می شوند که در فیلد نوع آنها واژه مستقل قرار می گیرد.

متغیرعمومی STI:

اسکنر با دیدن هر شناسه ، اندیس آنرا در متغیرعمومی STI قرار می دهد. چنانچه دربخش DCL باشیم و متغیری که اسکنر آن را دیده درSymtab وجود نداشته باشد،آن متغیردر Symtab قرار داده می شودواندیس اختصاص داده شده به آن در متغیر عمومی STI قرار می گیرد و در غیر این صورت اندیس متناظر با آن در STI کپی می شود.

متغیرعمومی InDCL :

اگر اسکنر در قسمت Declaration یک زبان باشداین پرچم فعال می شود،در غیر این صورت پرچم خاموش است.

وقتی اسکنریک شناسه را در Symtab پیدا کرد ،اگرInDCL فعال باشد خطایی رخ می دهد زیرا این شناسه دوبار تعریف شده است.

مودهای آدرس دهی:

عددی که در فیلد آدرس قرار می گیرد می تواند آدرس مستقیم یا غیر مستیقیم یا بلافصل باشد.برای مشخص کردن مود آدرس دو بیت به ابتدای فیلد آدرس اضافه می کنیم.این دو بیت به صورت زیر کد گشایی می شوند:

مقدار	نماد	مود آدرس دهی
00	None	مستقیم
01	@	غیرمستقیم
10	#	بلافصل

تمرین :

برای عبارت زیر ، کدی که کامپایلر تولید می کند را بنویسید.

A=b*(c+d)/e

M-Table:

کدی که روال مفهومی تولید می کند درون این جدول قرار می گیرد.دراین جدول نوع عملگر و آدرس (های) عملوند (های) مربوطه قرار می گیرد.

به عنوان مثال کد زیر برای عمل ضرب مقابل تولید شده است:

T=B*C

شماره	عملگر	آدرس عملوند ۱	آدرس عملوند ۲	آدرس نتیجه
0	*	آدرس B	آدرس C	آدرس T
1				

PC →

متغیر عمومی PC :

این متغیر همواره به اولین خانه خالی M-Table اشاره می کند.

متغیر عمومی ICV:

اسکنر اعداد ثابت را به صورت کارکتر می پذیرد و مقدار عددی آنها را در این متغیر قرار می دهد.

متغیر عمومی ADR :

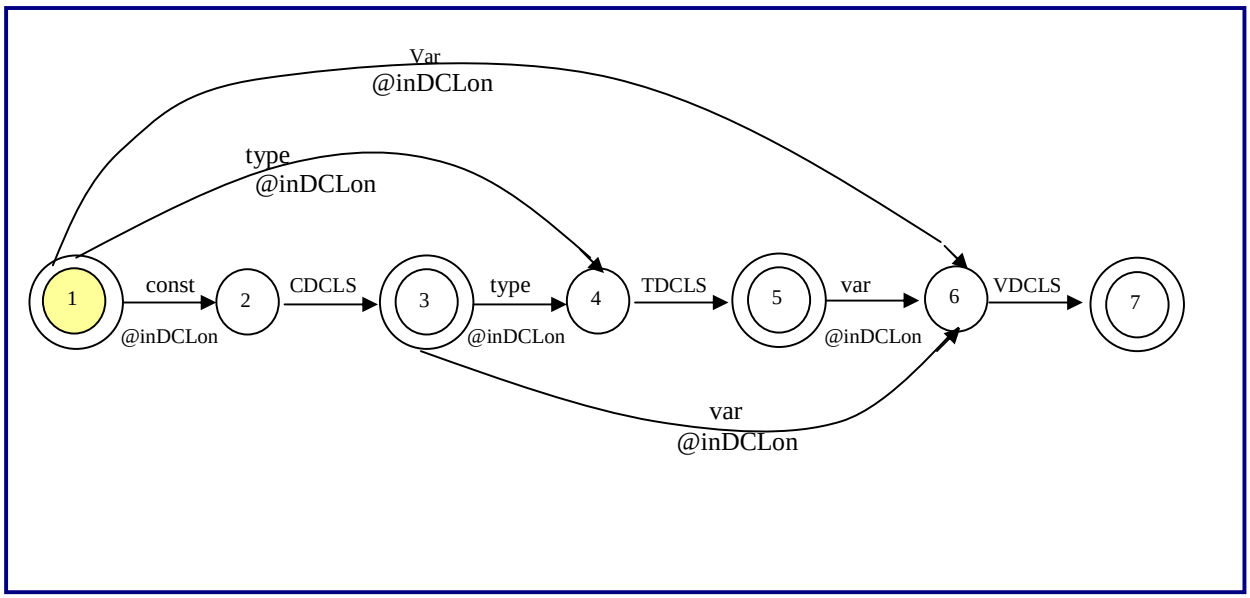
همواره به اولین خانه حافظه Allocate نشده اشاره می کند. در شروع کامپایل ، مقدار ADR برابر صفر است.

تمرین :

عبارت ریاضی مثال بزنید که وقتی کد آن تولید می شود ، به بیش از دو Temporary نیاز داشته باشیم.

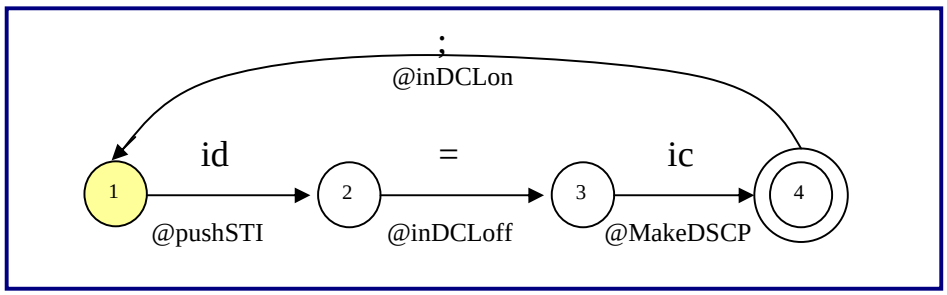
حال در ادامه ، گرافهای Declaration یک برنامه پاسکال را با جزییات بیشتری مورد بررسی قرار می دهیم:

DCLS :

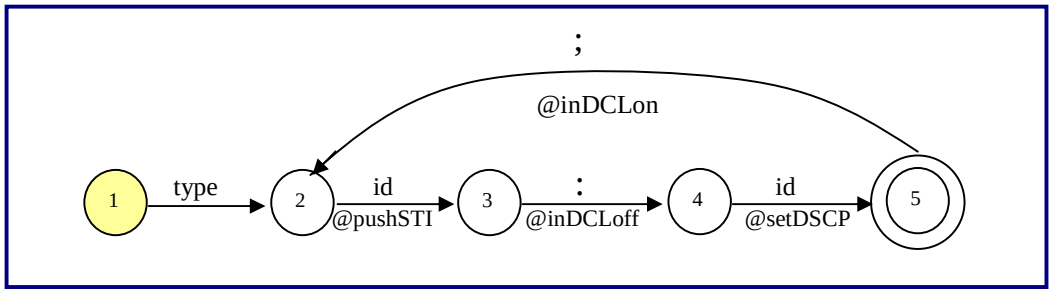


در گراف بالا CDLS مخفف Constant Declaration و TDCLS مخفف Type Declaration و VDCLS مخفف Variable Declaration می باشند.

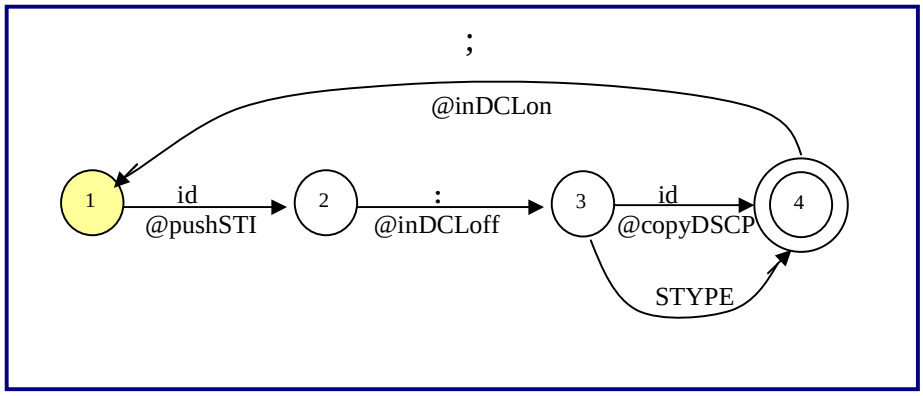
CDCLS:



TDCLS:



VDCLS:



توجه : گراف واژه نحوی STYPE در مبحث آرایه ها آمده است.

روالهای مفهومی @inDCLon و @inDCLoff متغیر عمومی InDCL را بترتیب روشن (True) و خاموش (False) می کند . بقیه روالهای مفهومی گرافهای بالا در زیر آمده است:

```
Make DSCP:

        Symtab[TopSS].DSCP := Make DSCP(ic,ICV);
        PopSS ;
```

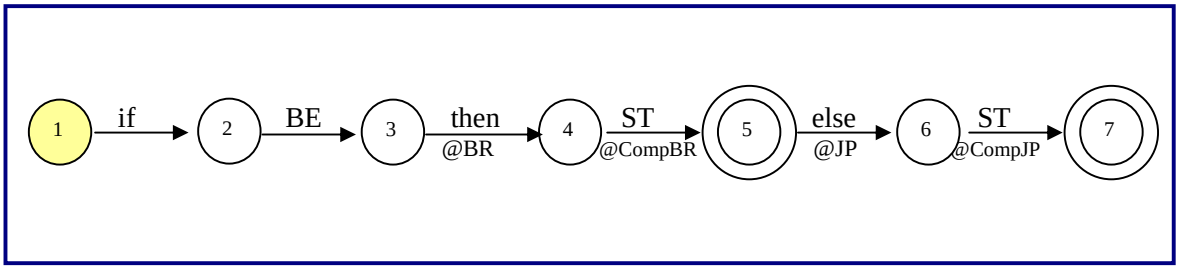
توجه کنید که وظیفه تابع Make DSCP(ic,IC□) ساختن فیلد Descriptor برای یک مقدار ثابت (ic) می باشد.

```
Set DSCP:

        Symtab[TopSS].DSCP := Symtab[STI].DSCP ;
        PopSS ;
```

```
Copy DSCP:

        Symtab[TopSS].DSCP := Copy DSCP(STI) ;
        Symtab[TopSS].DSCP.adr := ADR ;
        ADR + := size of (Symtab[TopSS].DSCP.type) ;
        PopSS  ;
```

کارتابع Copy DSCP کپی کردن فیلد Descriptor می باشد.

گراف ساختارهای شرطی :



در گراف فوق شرط Match شدن else با آخرین if رعایت شده است . در بعضی زبانها از Endif برای نشان دادن پایان if استفاده می شود که رسم گرامر آن بعنوان تمرین به خواننده واگذار شده است.

تمرین :
گراف ساختار if…then….else….endif را رسم کنید.

تمرین :
گراف BE را با رعایت حق تقدم رسم کنید.

توجه کنید که عبارات or not و and not قابل قبول و عبارت not not غیر قابل قبول است.

توجه :

از این به بعد فرض می کنیم که ST چیزی از خود در Semantic Stack باقی نمی گذارد.

روالهای مفهومی ساختار شرطی if به صورت زیر است :

```
BR :

    M [pc].op :='BRZ' ;
    M [pc].opr1 := Symtab[Top_SS].DSCP.adr ;
    Push_SS (pc);
    Pc := pc+1;
```

```
CompBR :

    M [Top_SS].opr2 :=pc+1 ;
    Pop_SS ;
```

```
JP :

    M [pc].op :='JP' ;
    Push_SS (pc) ;
```

```
CompJP :

    M [Top_SS].opr1 :=pc ;
    Pop_SS ;
```

توجه کنید، وقتی که به واژه مفهومی BR@ می رسیم ، در صورتی که BE درست نباشد ، کامپایلر باید دستورپرش به عبارت بعد از else را در M-Table قرار دهد ولی چون آدرس پرش فعلا معلوم نیست ، لذاpc را در Semantic Stack ذخیره کرده وموقتا از روی دستور BRZ عبور می کند و در روال مفهومیCompBR@ فیلد دوم دستور BRZ(همان آدرس پرش) کامل شده وpc از Semantic Stack ، Pop می شود.

همین مطلب در مورد روالهای JP@ و CompJP@ نیزصادق است.

تمرین :
روالهای مفهومی ساختارشرطی if…then….else….endif را بنویسید.

تمرین :
گراف Assignment را رسم کرده و روالهای مفهومی آنرا بنویسید.

آرایه ها :

چگونگی فیلد Descriptor یک آرایه در Symtab :

Descriptor یک آرایه از چهار قسمت تشکیل شده است : adr ، type ، lb ، ub .

در adr ، آدرس شروع مجازی آرایه($a`_B$) قرار می گیرد و مقدار آن از رابطه a_B-lb*sb محاسبه می شود. در رابطه فوق a_B آدرس شروع واقعی آرایه و lb حد پایینی آرایه و sb اندازه هر عضو آرایه می باشد.

نکته :

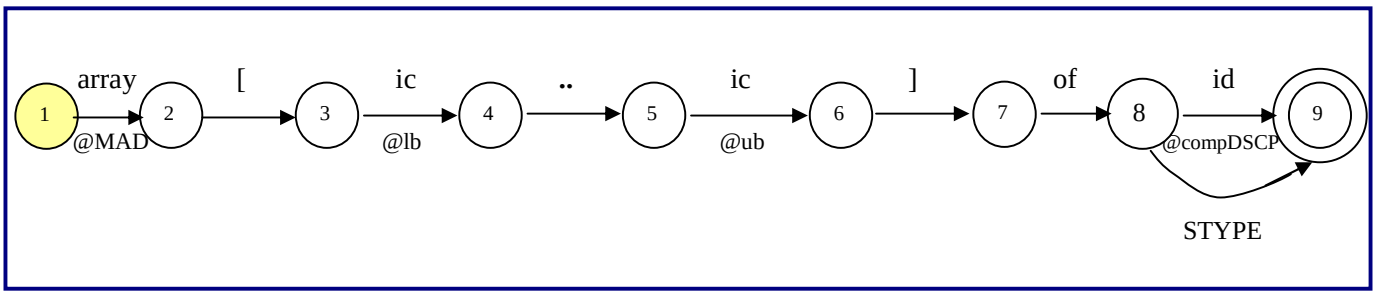
مقدار $a`_B$ تنها یکبار حساب می شود وبعد از آن ، در کلیه آدرس دهی ها مورد استفاده قرار می گیرد. (این مساله برای بهینه سازی بسیار مهم است.)

فیلد type نوع آرایه را و ub حد بالایی آرایه را مشخص می کند.

تعریف یک آرایه :

گراف مربوط به تعریف آرایه درپاسکال به صورت زیراست:

STYPE :



روالهای مفهومی گراف بالا بصورت زیر است :

```
MAD :

    Symtab[TopSS].DSCP := MakeDSCP('array') ;
```

تابع MakeDSCP فیلد Descriptor یک آرایه را می سازد . Descriptor ساخته شده در این قسمت ، دارای مقدار اولیه NULL می باشد.

```
lb :

    Symtab[TopSS].DSCP.lb := ICV ;
```

```
ub :

    Symtab[TopSS].DSCP.ub := ICV ;
```

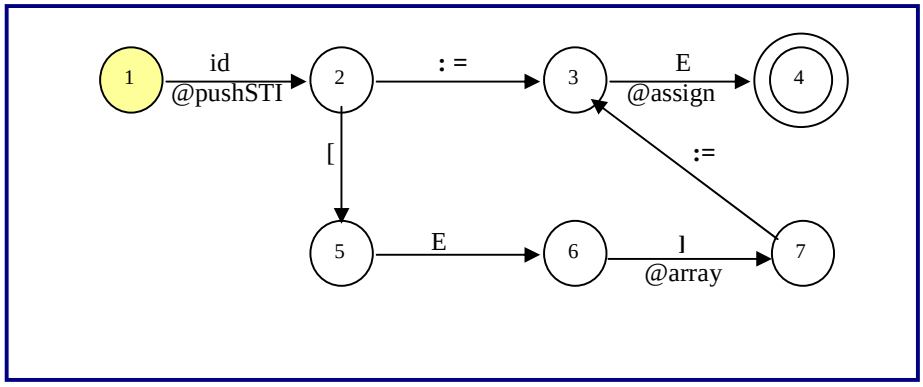
```
CompDSCP :

     With  Symtab[TopSS].DSCP do
        begin
           type :=Symtab[STI].DSCP.type ;
           adr := ADR-lb*sizeof (type) ;
           ADR := ADR+(ub-lb+1)*sizeof(type) ;
        End
    PopSS ;
```

تمرین :

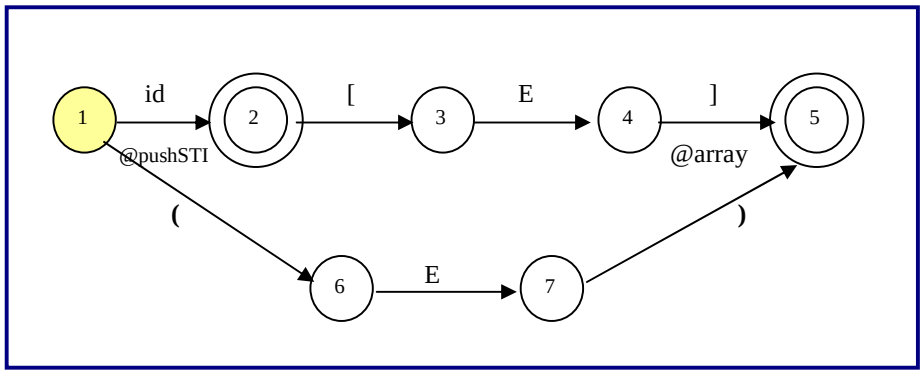
در گراف بالا Scanner با دیدن 1..10 بازا .1 یک عدد اعشاری رد می کند ، در حالی که Parser انتظار دارد یک عدد صحیح رد شود. گراف را طوری اصلاح کنید که این اشکال بر طرف شود.

گرافهای ST وعبارات ریاضی با در نظر گرفتن آرایه ها :

ST :



F :



بقیه گرافها (E ,T) هیچ تغییری نمی کنند.

روالهای مفهومی گرافهای بالا بصورت زیر است:

assign :

STI_e :=Top_{SS} ; Pop_{SS} ;
STI_{LHS} :=Top_{SS} ; Pop_{SS} ;
Rt := CheckType (`=` , STI_e ,STI_{LHS}) ;
M[pc].op :=`=` ;
M[pc].opr1 := Symtab[STI_{LHS}].DSCP.adr ;
M[pc].opr2 := Symtab[STI_e].DSCP.adr ; Release (STI_e) ;

array :

STI_e := Top_{SS} ; Pop_{SS} ;
STI_a := Top_{SS} ; Pop_{SS} ;
RT := CheckType(`array` , STI_a , STI_e) ;
M[pc].op := `*` ;
M[pc].opr1 := Symtab[STI_e].DSCP.adr ; Release(STI_e);
M[pc].opr2 := Sizeof(Symtab[STI_a].DSCP.type) ;
STI_T :=GetTemp (`int`) ;
M[pc].Res := Symtab[STI_T].DSCP.adr ;
Pc :=pc+1 ;
M[pc].op := `+` ;
M[pc].opr1 :=Symtab[STI_a].DSCP.adr ;
M[pc].Res := M[pc].opr2 := Symtab[STI_T].DSCP.adr ;
$Push_{SS}$ (STI_T) ;
Symtab[STI_T].DSCP.type := RT ;
Symtab[STI_T].DSCP.adr + := 2^K ;

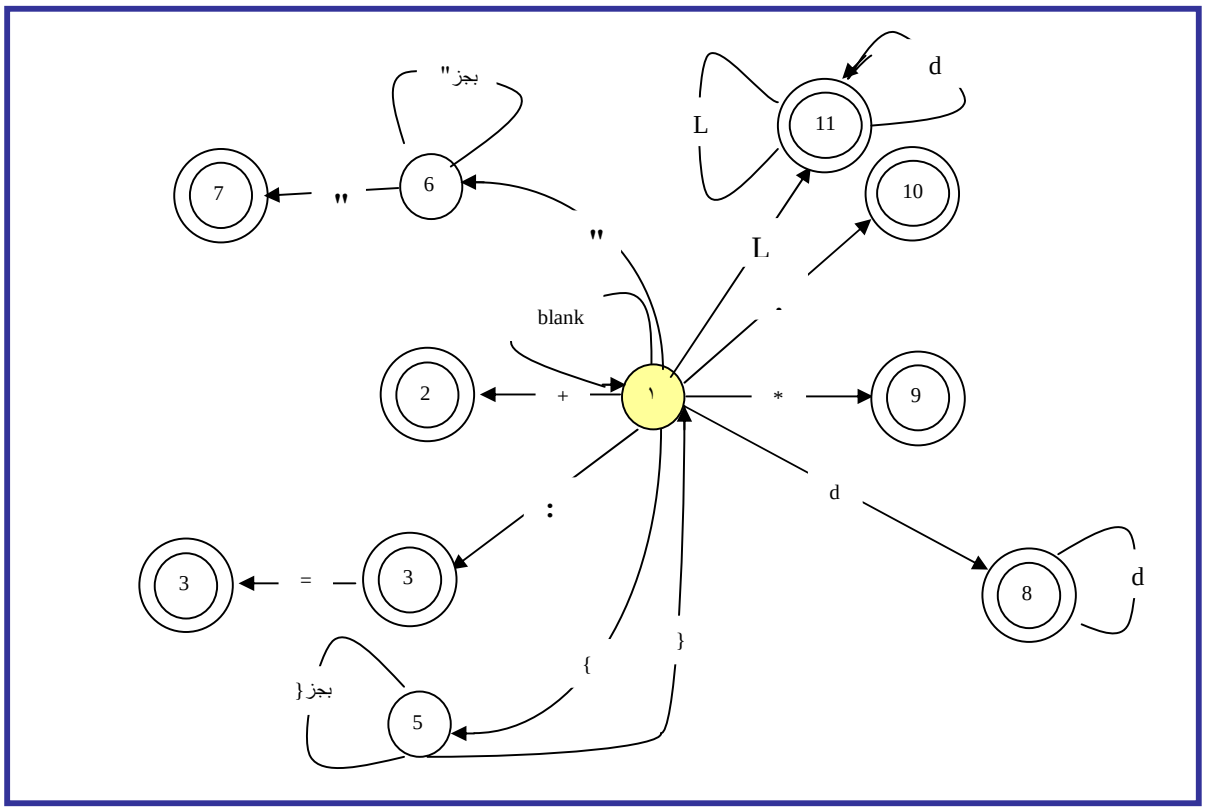
در روال فوق STI_e اندیس جواب E است.

تمرین :
کامپایلر برای عبارت زیر چه کدی تولید می کند.

A[B[I+J]-2] :=A[-1]

واژه یاب (Scanner) :

گراف Scanner زبان پاسکال در زیر آمده است:



الگوریتمی که اسکنربر اساس آن ، واژه یابی می کند به صورت زیر است:

```
Function Scanner (ch) ;
    1: Case  ch  of
        Blank  :  while ch in [blank, eoln,lf]  do   { read (ch);  goto 1; }
        '+' :  read (ch); return (pluse);
        ':'  :  read (ch) ; if  ch= '='  then   {read (ch); return (assign)}
                          else   return (colon);
        '{' :  while  ch <> '}'  do  read (ch);
              read (ch);   goto 1;
        '"' :   string := "" ; read (ch);
               while  ch <> ' " ' do  { string +: = ch ; read (ch);}
                read (ch);  return (SC);
```

```
‘0’..’9’ : ICV : = 0; string : ="" ;
            while  ch  in [‘0’..’9’] do
                { ICV :=ICV*10+ord(ch) - ord(‘0’) ;
                  string + := ch; read (ch);
                 }
             Find ic Symtab (string);  return (ic);
‘A’..’Z’ : string := "" ; while  ch  in  [‘0’..’9’,’A’..’Z’] do
                                    {string + :=ch ; read (ch);}
                 if  InDCL  then  {put id symtab (string)}
                  else Token := find id Symtab (string);  return (Token);
      endcase
endprocedure
```

تمرین :

گراف Scanner را طوری اصلاح کنید که

الف) درون رشته بتوانیم " نیز داشته باشیم ، یعنی اگر ورودی به صورت "....."....." باشد خروجی برابر ....."..... باشد.

ب) Commentها با */ شروع و با /* خاتمه پذیرد.

توجه کنید که / و * خود می توانند یک واژه باشند.

اشتقاق قانونمند : (Canonical Derivation)

اشتقاق چپ: (Leftmost Derivation)

در هربار بسط سمت چپ ترین متغیربرای بسط انتخاب می شود.

اشتقاق راست: (Rightmost Derivation)

در هربار بسط سمت راست ترین متغیربرای بسط انتخاب می شود.

ساختار یاب کل به جز: (TopDown Parser)

```
Procedure TDParser
      Push_Ps($s);
      Token:=Scanner;
    Loop
      Case Top_ps of
          واژه نحوی:
                  Prod:=Getproud(Top_ps,Token);
                  If Prod=0 then Error;
                  Pop_ps;
                  Push_ps(RHS[Prod]);
          واژه زبان:
                 If Top _ps<> Token Then Error;
                 Pop_ps;
                  Token:=Scanner;
               $:
                  if Token=$ then EXIT
                  else Error;
```

چند نکته:

۱- $ واژه پایان ورودی می باشد.اسکنر وقتی که به پایان ورودی می رسد،واژه $ را به عنوان Token برمی گرداند.

۲- ابتدا یک $ در پشته گذاشته می شود تا علامت پایان پشته باشد.

۳- تابع Getprod نیز $ را علامت هیچ چیز می گیرد.

۴- تابع Getprod قاعده تولیدی را انتخاب می کند که سمت چپ آن Top ps و ورودی آن Token است.

۵-اگر Getprod به گونه ای باشد که یک قاعده تولید را برگرداند(فقط یک قاعده تولید مناسب

باشد)، می توان پارسر خطی داشت وگرنه باید Back track داشت، که در نتیجه پارسر خطی نخواهد بود.

۶-RHS جدولی است که قواعد تولید به ترتیب عکس در آن قرار دارند.

به عنوان مثال جدول RHS برای قواعد تولید عبارات ریاضی به صورت زیر می باشد:

Prod Num	Products
1	E` T
2	
3	E` T +
4	T` F
5	
6	T` F *
7	Id
8	) E (

۷- از Code generator (CG) برای تولید کد(روالهای مفهومی) استفاده می شود.

```
Procedure CG(action)
    Case action of
        @add:
            ----------
            ----------
            ----------
        @mult:
            ----------
            ----------
            ----------
        @Pushsti:
            ----------
```

تمرین :

در ساختار یاب فوق دستور if در قسمت " واژه زبان" برای چک کردن وجود ")" می باشد.
اولاً:این گفته را اثبات کنید.
ثانیاً: گرامر را به گرامری تبدیل کنید که دیگر نیازی به بررسی فوق نباشد.

تمرین :
واژههای مفهومی را به گرامر عبارات ریاضی اضافه کنید وجدول RHS را کامل کنید.

مثال:
اگر در ورودی جمله a+b*c را داشته باشیم ، جدول زیر مراحل اجرا را توسط ساختاریاب کل به جز نشان می دهد:

PS	بقیه ورودی	Prod	SS	کد
$ E	a+b*c$	1		
$ E` T	a+b*c$	4		
$ E` T` F	a+b*c$	7		
$ E` T` id @pushsti	a+b*c$		STI_a	
$ E` T` id	a+b*c$		STI_a	
$ E` T`	+b*c$	5	STI_a	
$ E`	+b*c$		STI_a	
$ E` @add T +	+b*c$	3	STI_a	
$ E` @add T	+b*c$		STI_a	
$ E` @add T` F	b*c$	4	STI_a	
$ E` @add T` id @pushsti	b*c$	7	STI_a	
$ E` @add T` id	b*c$		STI_a , STIb	
$ E` @add T`	*c$	6	STI_a , STIb	
$ E` @add T` @mult F *	*c$		STI_a , STIb	
$ E` @add T` @mult id @pushsti	c$		STI_a , STIb	
$ E` @add T` @mult id	c$	7	STI_a , STIb,STI_c	
$ E` @add T` @mult	$		STI_a , STI_T,STI_c	
$ E` @add T`	$	5	STI_a , STI_T	*,b,c,T
$ E` @add	$	2	STI_a , STI_T	
$ E`	$		STI_T	+,a,T,T
$			STI_T	

چند نکته در مورد جدول فوق:

۱- درستون مربوط به PS و SS سمت چپ ترین عنصر، عنصرپایین پشته را نشان می دهد.

۲- ستون مربوط به کد، کدهای تولید شده مربوط به روالهای مفهومی را نشان می دهد.

۳- ستون Prod شماره قا عده تولید استفاده شده در هر مرحله را نشان می دهد .

تمرین :
اشتقاق راست وچپ a*(b+c) را با گرامر ۱و۲ بنویسید.

تمرین :
پارسر را برای a*(b+c) اجرا ونتیجه را در جدول نشان دهید.

پیاده سازی تابع Getprod :

برای پیاده سازی تابع Getprod از جدولی به نام Parse table استفاده می کنیم .همانطور که در قبل هم دیدیم تابع Getprod دو ورودی Top_{ps} و Token دارد که سطروستونهای جدول پارس می با شند . بنابراین تابع فوق با توجه به ورودیها یش خانه مناسب را انتخاب می نماید وعدد موجود در آن خانه رابه عنوان خروجی برمی گرداند.

در زیرجدول پارس مربوط به عبارات ریاضی را مشاهده می کنیم. نحوه ساخت این جدول در ادامه آمده است.

نکته:

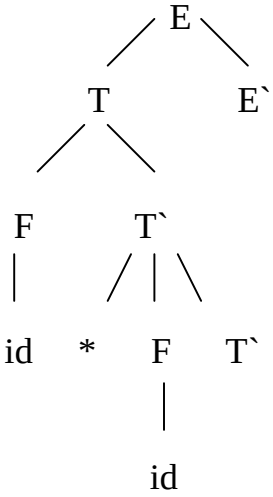
با داشتن این جدول می توان گفت که کد پارسر ، یک کد ثابت است و یا به عبارت دیگر

Table deriven است. زیرا برای تغییرکد کافی است جدول پارس را تغییر داد.

	+	*	Id	(	)	$
E	0	0	1	1	0	0
E`	3	0	0	0	2	2
T	0	0	4	4	0	0
T`	5	6	0	0	5	5
F	0	0	7	8	0	0

نکته:

هراشتقاقی را می توان به صورت یک درخت نشان داد. به عنوان مثال درخت زیر اشتقاق چپ عبارت ریاضی id*id را نشان می دهد.



LL(1) Parser :

در این نوع پارسر از اشتقاق چپ استفاده می شود . پارسر برای اینکه قاعده تولید مناسب را انتخاب کند ، فقط به یک واژه زبان نیاز دارد ،مانند TDParser .

نکته:

LL(2) Parser به دو واژه زبان برای انتخاب قاعده تولید نیاز دارد، و جدول آن سه بعدی وحجیم است.

مثال:
گرامر زیر LL(2) می باشد:

7 F ⟶ id
9 F ⟶ id [E]

و به طریق زیر LL(1) می شود:

7 F ⟶ id F`
9 F`⟶
10 F`⟶ [E]

ساختاریاب اچ-۱ (LL(1) Parser) :

گرامر اچ-۱ همان LL(1) Parser می باشد. برای تشخیص اینکه گرامری اچ-۱ می باشد یا خیر کافی است جدول پارس آنرا ساخت. در صورتی که خانه های این جدول دارای حداکثر یک شماره باشند ، گرامر اچ-۱ می باشد.

برای ساخت جدول پارس باید با مفاهیم First و Follow آشنا شد. برای این منظور قواعد تولید زیر را در نظر می گیریم وبه کمک آنها جدول پارس را می سازیم.

1 E ⟶ T E`
2 E`⟶
3 E`⟶ + T E`
4 T ⟶ F T`
5 T`⟶
6 T`⟶* F T`
7 F ⟶ id F`
8 F ⟶ (E)
9 F`⟶
10 F`⟶ [E]

First:

به طور کلی به ازای A⟶ a i ، First (a) اولین واژه زبانی است که از a تولید می شود،

وبه طریق زیر محاسبه می شود:

$$First(A)=\begin{cases} NULL \text{ if } a=Lambda \\ \{s\} \text{ if } a = sa` \mid s \text{ Belong } T \\ First(B) \quad \text{if } \{ (a = Ba`) \text{ And } (B \not\Longrightarrow Lambda) \mid B \text{ Belong } V\} \\ First(a`) \quad \text{if } \{ (B = Ba`) \text{ And } (B \Longrightarrow^* Lambda) \mid B \text{ Belong } V\} \end{cases}$$

تعریف :

A میرا است اگر:

$$A \text{ is Nullable if } A \Longrightarrow^* Lambda$$

بنابراین نتیجه می گیریم که:

$$First(A)= \text{union of } \{First(a) \mid A \longrightarrow a\}$$

با توجه به روابط بالا ، برای قواعد تولید بیان شده داریم:

First(F`)=First(9)+First(10)={[}
First(F)=Fisrt(7)+First(8)={id}+{(}={id,(}
Fisrt(T`)= Fisrt(5)+First(6)={*}
First(T)= Fisrt(4)=First(F)={id,(}
Fisrt(E`)= Fisrt(2)+First(3)={+}
First(E)= Fisrt(1)=First(F)={id,(}

در مثال بالا واژه های نحوی F`,T`,E` ، میرا (Nullable) می باشند.

Follow:

عبارت است از واژه هایي که در یک متن گونه ، بعد از واژه نحوی قرار می گیرند.

$$Follow(A)=\{a \text{ Belong } T \mid s \Longrightarrow^* \ldots.Aa\ldots\}$$

بطور مثال برا ی قواعد تولید بالا داریم:

Follow(E)={),]}

نکته:

برای محاسبه Follow در صورتی که واژه نحوی در انتهای سمت راست باشد،یا سمت راست آن میرا باشد، Follow واژه نحوی سمت چپ قاعده تولید را محاسبه می کنیم.

بطور مثال برا ی قواعد تولید بالا داریم:

Follow(E`)=Follow(E)={),] }

ساختن جدول ساختاریابی:

مرحله ۱:

ابتدا جدولی به صورت زیر تشکیل می دهیم که عناصر افقی واژه های زبان و عناصر عمودی واژه های نحوی می باشند:

	+	*	Id	(	)	[	]	$
E								
E`								
T								
T`								
F								
F`								

مرحله ۲ :

سپس First واژه های نحوی را محاسبه می کنیم و شماره قانونی که First از آن محاسبه شده است را، در سطر واژه نحوی مربوطه ، در زیرستونی که First آن می باشد ، قرارمی دهیم.

	+	*	Id	(	)	[	]	$
E			1	1				
E`	3							
T			4	4				
T`		6						
F			7	8				
F`						10		

مرحله ۳ :

اگر واژه نحوی میرا باشد، Follow آنرا حساب می کنیم و جدول را مانند مرحله قبل کامل

می کنیم:

	+	*	Id	(	)	[	]	$
E			1	1				
E`	3				2		2	2
T			4	4				
T`	5	6			5		5	5
F			7	8				
F`	9	9			9	10	9	9

نکته:

$ همواره در Follow واژه نحوی شروع وجود دارد.

تمرین : گرامر عبارات مقایسه ای را بنویسید و جدول پارسر را بکشید.

قاعده تولید چپ گرد:

قاعده تولید زیر یک قاعده تولید چپ گرد می با شد ، زیرا واژه نحوی سمت چپ ، عیناً در ابتدای

سمت راست قاعده تولید آ مده است :

$$A \longrightarrow Aa$$

نکته:

هرگاه در گرامری قاعده تولید چپ گرد داشته باشیم ، آن گرامر LL(1) نیست.

راه حل کلی حذ ف چپ گردی :

فرض کنید گرامر چپ گردی به فرم زیر داشته باشیم:

$$1 \begin{cases} A \longrightarrow Aa_1 \\ A \longrightarrow Aa_2 \\ \vdots \\ A \longrightarrow Aa_n \end{cases}$$

$$2\begin{cases} A \longrightarrow b_1 \\ A \longrightarrow b_2 \\ \vdots \\ A \longrightarrow b_m \end{cases}$$

قسمت ۱ را به فرم زیر می نویسیم:

$$A \Longrightarrow^* A(a_1 \mid a_2 \mid \ldots\ldots\ldots \mid a_n)^*$$

قسمت ۲ را به فرم زیر می نویسیم:

$$A \Longrightarrow^* (b_1 \mid b_2 \mid \ldots\ldots\ldots \mid b_m)$$

از تلفیق دو عبارت فوق به نتیجه زیر می رسیم:

$$A \Longrightarrow^* (b_1 \mid b_2 \mid \ldots\ldots\ldots \mid b_m)\,(a_1 \mid a_2 \mid \ldots\ldots\ldots \mid a_n)^*$$

عبارت فوق را می توان به صورت زیر نمایش داد:

$$A \longrightarrow BA`$$

که در آن :

$$B \Longrightarrow^* (b_1 \mid b_2 \mid \ldots\ldots\ldots \mid b_m)$$

و

$$A` \Longrightarrow^* (a_1 \mid a_2 \mid \ldots\ldots\ldots \mid a_n)^*$$

بدین ترتیب چپ گردی A برطرف شد.

گرامر مبهم:

اگردرگرامری برای یک متن دو یا چند پارس مختلف(با یک نوع اشتقاق) وجود داشته باشد، آن گرامر مبهم خواهد بود. به عنوان مثال گرامر زیر مبهم است:

$$ST \longrightarrow \text{if BE then ST EP}$$
$$EP \longrightarrow$$
$$EP \longrightarrow \text{else ST}$$

در گرامر فوق متن $\text{if } be_1 \text{ then if } be_2 \text{ then } st_1 \text{ else } st_2$ به دو طریق زیر(با یک نوع اشتقاق) به دست می آید:

ST ⟶ if BE then ST EP ⟶ if BE then if BE then ST EP EP

به ازای ورودی else دو راه مختلف داریم:

۱) بجایEP اول ، Null و EP دوم ، st else قرار دهیم .

۲) بجایEP اول ، st else و EP دوم ، Null قرار می دهیم.

بنابرا ین گرامر فوق مبهم است.

تمرین :
جدول پارس اچ-۱گرامر if...then...else را بکشید.
این گرامر را با endif بنویسید وجدول آنرا بکشید.
گرامر if...then...else بدون endif را طوری بنویسید که قانون تطبیق else با آخرین if رعایت شود.

پارسر SLR (1) (سار-۱) :

قاعده تولید زیر را در نظر بگیرید:

1 E ⟶ E+T
2 E ⟶ T
3 T ⟶ T*F
4 T ⟶ F
5 F ⟶ id
6 F ⟶ (E)

فرض کنید برای پارس کردن عبارت id+id*id به ترتیب از قاعده تولیدهای زیر استفاده کنیم :

$$E \overset{1}{\Longrightarrow} E+T \overset{3}{\Longrightarrow} E+T*F \overset{5}{\Longrightarrow} E+T*id \overset{4}{\Longrightarrow} E+F*id \overset{5}{\Longrightarrow} E+id*id \overset{2}{\Longrightarrow} T+id*id \overset{4}{\Longrightarrow} F+id*id \overset{5}{\Longrightarrow} id+id*id$$

حال سوا ل این است که اگر در جهت عکس حرکت کنیم ،با رعا یت چند قاعده می توان فقط با دانستن یک ورودی ، زبان را پارس کرد؟

قاعده :

متن گونه را از ا بتدا به ا نتها می خوانیم و در آن به د نبال سمت راست ترین واژه نحوی می گردیم و آنرا با یکی از قواعد تولید که همان سمت راست را دارد ،تطبیق می دهیم (به شرطی که ورودی بعدی در Follow سمت چپ باشد) و از آن استفاده می کنیم .

شرط SLR(1) بودن :

شرط SLR (1) بودن یک گرامر این است که وقتی Parser می خواهد از بین دو یا چند قاعده تولید ، قاعده ای را انتخاب کند ، ورودی بعدی به Parser کمک کند تا فقط یک قاعده تولید را انتخاب کند.

دیاگرام SLR(1) (سار-۱) :

دیاگرام SLR(1) مجموعه ای از State ها می باشد. هر State دارای تعدادی ورودی و خروجی می باشد . parser در هر State بر اساس اینکه ورودی اش با کدام خروجی از State منطبق است ، آن State را به مقصد State بعدی ترک خواهد کرد. ترک یک State با پذیرفتن یک واژه نحوی را Goto و با یک واژه زبان را Shift می نامند. در درون هر State تعدادی قاعده تولید قرارمی گیرد که در موقعیتی از سمت راستشان علامت • (نقطه) قرار گرفته است.

علامت • (نقطه) قبل از هر واژه (چه نحوی و چه زبان) نشاندهنده این است که Parser در ورودیاش منتظر دریافت آن واژه می باشد.

State ای که علامت • (نقطه) به انتهای سمت راست همه قاعدهای تولید موجود در آن رسیده

باشد را Reduce می نامند.

هر State از دو قسمت تشکیل شده است :

۱- قسمت پایه

۲ – قسمت منتج

قسمت پایه :

از این قسمت برای ساختن قسمت منتج استفاده می شود. قسمت پایه state شروع همواره به صورت $E. می باشد، که E واژه نحوی شروع گرامر می باشد.

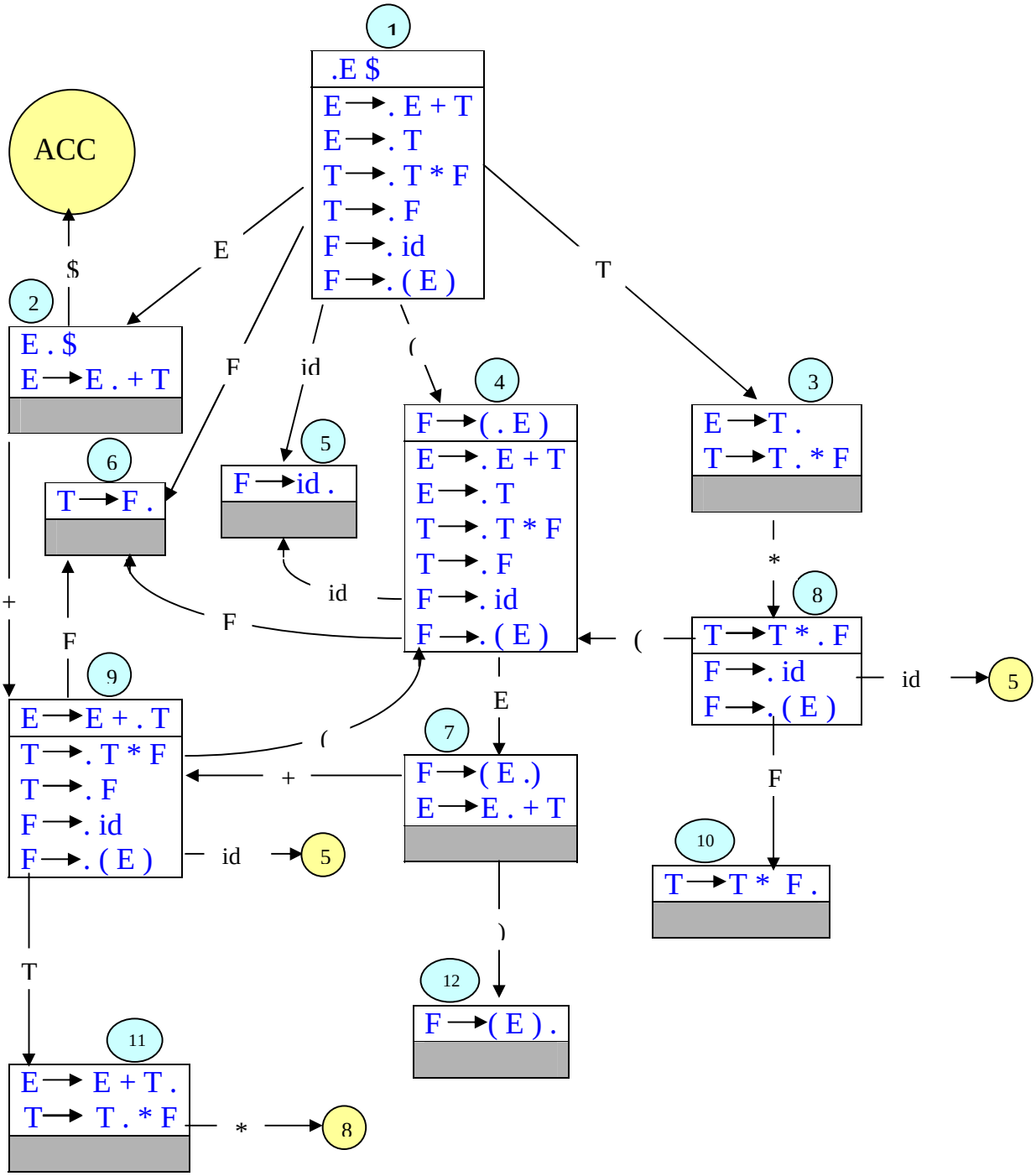
قسمت منتج :

- این قسمت از روی قسمت پایه ساخته می شود ، بدین صورت که هر گاه در قسمت پایه علامت (نقطه) قبل از یک واژه نحوی قرار داشته باشد ،در قسمت منتج تمام قاعده تولیدهایی که سمت چپشان آن واژه نحوی قرار دارد ، نوشته می شوند ، با این تفاوت که درابتدای سمت راستشان علامت (نقطه) قرار می گیرد و دوباره قاعده فوق در مورد این قاعده تولید (های) جدید اجرا می شود. این کار تا زمانی که ممکن باشد، ادامه می یابد.

نکته :

همیشه State اول گرامر ، شامل تمام قواعد تولید آن گرامر نیست.

به عنوان مثال، دیاگرام SLR(1) گرامر مطرح شده در ابتدای بحث پارسر SLR(1) ،ترسیم شده است :


1
.E $
E → . E + T
E → . T
T → . T * F
T → . F
F → . id
F → . (E)
ACC
$
E
F
id
(
T
2
E . $
E → E . + T
6
T → F .
5
F → id .
4
F → (. E)
E → . E + T
E → . T
T → . T * F
T → . F
F → . id
F → . (E)
3
E → T .
T → T . * F
*
8
T → T * . F
F → . id
F → . (E)
id
5
F
9
E → E + . T
T → . T * F
T → . F
F → . id
F → . (E)
+
(
id
5
7
F → (E .)
E → E . + T
E
)
12
F → (E) .
10
T → T * F .
T
11
E → E + T .
T → T . * F
*
8

تعریف Reduction (دستور Reduce):

به قاعده تولیدی که در آن علامت • به انتهای سمت راستش رسیده است، یک دستور Reduce (Reduction) می گویند.

نحوه ساختاریابی در یک گرامر SLR(1) با استفاده از دیاگرام آن :

در هر ساختاریابی همیشه شماره State شروع ، اولین عددی است که درون Parse Stack قرار می گیرد. درمرحله بعد Parser با گرفتن ورودی بعدی اش ،وارد State دیگری می شود و شماره این State درون Parse Stack قرار می گیرد.اگرورودی Parser طوری بود که با پذیرفتن آن ، در State جدید به یک Reduction رسیدیم ، در این حالت Parse Stack به اندازه تعداد واژه های قبل از نقطه Pop می شود و از State ای که شماره آن در Top ، Parse Stack قرار دارد با واژه نحوی موجود در سمت چپ قاعده Reduction ، خارج می شویم ، در غیر اینصورت با توجه به ورودی وارد State بعدی می شویم. در هر Reduction یک واژه زبان توسط Parser پذیرفته می شودو برای مراحل بعدی از واژه های باقیمانده استفاده می شود.

باید بتوانیم با تکرار کارهای فوق به State پایانی (ACC) برسیم ،در غیر این صورت عبارت داده شده از لحاظ ساختاری ، دارای خطا است .

نکته :

توجه داشته باشید که درانتهای هر عبارت علامت $ وجود دارد وParser با پذیرفتن این علامت وارد State پایانی (ACC) می شود.

برای درک بهتر ، به مثال صفحه بعد توجه کنید.

مثال :

نحوه ساختاریابی عبارت a+b*c را با استفاده از دیاگرام SLR(1) فوق نشا ن دهید.

حالات PS	ورودی باقیمانده	قاعده تولید
1	a+b*c$	
1 5	+b* c$	5
1 6	+b* c$	4
1 3	+b* c$	2
1 2	+b* c$	
1 2 9	b* c$	
1 2 9 5	* c$	5
1 2 9	* c$	
1 2 9 6	* c$	4
1 2 9 11	* c$	
1 2 9 11 8	c$	
1 2 9 11 8 5	$	5
1 2 9 11 8	$	
1 2 9 11 8 10	$	3
1 2 9 11	$	1
1	$	
1 2		

تمرین :

با گرامر SLR(1) گرامر اچ-۱ عبارات ریاضی را کشیده و نحوه ساختاریابی عبارت a*b+c را نشان دهید.

جدول SLR(1) :

جدول SLR(1) از روی دیاگرام SLR(1) بدست می آید.

مراحل تکمیل جدول SLR(1) گرامر زیر توضیح داده شده است :

1 E ⟶ E+T
2 E ⟶ T
3 T ⟶ T*F
4 T ⟶ F
5 F ⟶ id
6 F ⟶ (E)

مرحله ۱:

ابتدا جدولی به صورت زیر تشکیل می دهیم که عناصر افقی مجموعه واژه های زبان و واژه های نحوی وعلامت $ و عناصر عمودی شماره State ها می باشند:

	+	*	Id	(	)	E	T	F	$
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									

مرحله ۲ :

سپس Shiftها و GoToهای State i, را به صورت خلاصه شده S وG و با اندیس شماره State بعد ، در سطر i و در ستونهای مربوطه جدول قرار می دهیم:

توجه: اگر به State پایانی رسیدیم ، واژه ACC را در جدول قرار می دهیم.

	+	*	Id	(	)	E	T	F	$
1			S_5	S_4		G_2	G_3	G_6	
2	S_9								ACC
3	S_8								
4			S_5	S_4		G_7	G_3	G_6	
5									
6									
7	S_9				S_{12}				
8			S_5	S_4				G_{10}	
9			S_5	S_4			G_{11}	G_6	
10									
11		S_8							
12									

مرحله ۳ :

در این مرحله Reduceهای State, i را به صورت خلاصه شده R و با اندیس شماره قاعده تولیدی که Reduce در آن اتفاق افتاده است ، را در سطر i ، و درزیرستونهای Follow واژه نحوی سمت چپ در جدول قرار می دهیم:

	+	*	Id	(	)	E	T	F	$
1			S_5	S_4		G_2	G_3	G_6	
2	S_9								ACC
3	S_8								
4			S_5	S_4		G_7	G_3	G_6	
5	R_5	R_5			R_5				R_5
6	R_4	R_4			R_4				R_4
7	S_9				S_{12}				
8			S_5	S_4				G_{10}	
9			S_5	S_4			G_{11}	G_6	
10	R_3	R_3			R_3				R_3
11	R1	S_8			R_1				R_1
12	R_6	R_6			R_6				R_6

مرحله ۴ :

در این مرحله خانه های خالی باقی مانده را پر می کنیم بدین ترتیب که اگر خانه خالی در زیر ستون ، یک واژه نحوی بود ،در آن خانه CE (Compiler Error) و در غیر اینصورت E (Error) نوشته می شود:

	+	*	Id	(	)	E	T	F	$
1	E_1	E_2	S_5	S_4	E_3	G_2	G_3	G_6	E_4
2	S_9	E_5	E_6	E_7	E_8	CE_1	CE_2	CE_3	ACC
3	S_8	E_8	E_9	E_{10}	E_{11}	CE_4	CE_5	CE_6	E_{12}
4	E_{13}	E_{14}	S_5	S_4	E_{15}	G_7	G_3	G_6	E_{16}
5	R_5	R_5	E_{17}	E_{18}	R_5	CE_7	CE_8	CE_9	R_5
6	R_4	R_4	E_{19}	E_{20}	R_4	CE_{10}	CE_{11}	CE_{12}	R_4
7	S_9	E_{21}	E_{22}	E_{23}	S_{12}	CE_{13}	CE_{14}	CE_{15}	E_{24}
8	E_{24}	E_{25}	S_5	S_4	E_{26}	CE_{16}	CE_{17}	G_{10}	E_{26}
9	E_{27}	E_{28}	S_5	S_4	E_{29}	CE_{18}	G_{11}	G_6	E_{30}
10	R_3	R_3	E_{31}	E_{32}	R_3	CE_{19}	CE_{20}	CE_{21}	R_3
11	R1	S_8	E_{33}	E_{34}	R_1	CE_{22}	CE_{23}	CE_{24}	R_1
12	R_6	R_6	E_{35}	E_{36}	R_6	CE_{25}	CE_{26}	CE_{27}	R_6

تمرین : دیاگرام SLR(1) گرامر زیر را بکشید و از روی آن جدول SLR(1) را بدست آورید.

توجه : در بعضی خانه ها دو دستور قرار می گیرد که یکی از آنها با توجه به قواعد تقدم حذف خواهد شد.

E ⟶ E+E
E ⟶ E*E
E ⟶ id
E ⟶ (E)

بهینه کردن جدول SLR(1) :

در ردیفهایی که فقط دستور Reduce وجود دارد ، امکان نوعی بهینه سازی وجود دارد.

فرض کنید که ردیف i ام فقط شامل دستور R_k (Reduce) باشد، اگر در جدول ، دستور G_i وجود نداشته باشد، می توان ردیف i ام را حذف کردو در جدول بجای دستور S_i دستور SR_k را قرار داد. نحوه عملکرد دستور SR به صورت زیر است:

```
SR:  Token := Scanner;
     Pop_PS (GT[N].RHSL-1)
     Push_PS(PT[Top_PS , GT[N].LHS].Number)
     CG(N);
```

به عنوان مثال جدول SLR(1) صفحه قبل بهینه شده است:

	+	*	Id	(	)	E	T	F	$
1	E_1	E_2	SR_5	S_4	E_3	G_2	G_3	G_6	E_4
2	S_9	E_5	E_6	E_7	E_8	CE_1	CE_2	CE_3	ACC
3	S_8	E_8	E_9	E_{10}	E_{11}	CE_4	CE_5	CE_6	E_{12}
4	E_{13}	E_{14}	SR_5	S_4	E_{15}	G_7	G_3	G_6	E_{16}
6	R_4	R_4	E_{19}	E_{20}	R_4	CE_{10}	CE_{11}	CE_{12}	R_4
7	S_9	E_{21}	E_{22}	E_{23}	SR_6	CE_{13}	CE_{14}	CE_{15}	E_{24}
8	E_{24}	E_{25}	SR_5	S_4	E_{26}	CE_{16}	CE_{17}	G_{10}	E_{26}
9	E_{27}	E_{28}	SR_5	S_4	E_{29}	CE_{18}	G_{11}	G_6	E_{30}
10	R_3	R_3	E_{31}	E_{32}	R_3	CE_{19}	CE_{20}	CE_{21}	R_3
11	R1	S_8	E_{33}	E_{34}	R_1	CE_{22}	CE_{23}	CE_{24}	R_1

توجه کنید که ردیف 10 و6 را نمی توان حذف کرد. زیرا دستورهای G_6 و G_{10} وجود دارد.

نحوه ساختاریابی در یک گرامر SLR(1) با استفاده ازجدول آن:

در این نوع ساختاریابی ابتدا جدول GT را تشکیل می دهیم.

نحوه تشکیل جدول GT :

جدولی به صورت زیر تشکیل می دهیم که عناصر افقی به ترتیب ابتدا واژه نحوی سمت چپ یک قاعده تولید(LHS) وسپس طول سمت راست همان قاعده تولید (RHSL) و عناصر عمودی شماره قاعده تولید می باشند:

	LHS	RHSL
1	E	3
2	E	1
3	T	3
4	T	1
5	F	1
6	F	1

نکته :

RHSL قاعده تولید میرا برابر 0 است.

بعد از تشکیل جدولGT برای ساختاریابی از Procedure های زیر استفاده می کنیم :

```
Procedure BUParser
      Push PS(1);
      Token := Scanner;
      Loop
          (A,N) := PT [TopPS ,Token];
           Case A of
               S : PushPS(N) ;  Token :=Scanner;
               R : PopPS (GT [N].RHS);
                    PushPS (PT [TopPS ,GT [N].LHS].N);
                    CG (N);
               E : Error handler (N);
               CE : Compiler error ;
               ACC : Exit ;
           endCase
     endLoop

endProcedure
```

```
Procedure CG(prod)
   Case  prod  of

      1: .....
         .....        } add
         .....

      2: .....
         .....        } mult
         .....

         :

   endcase
endprocedure
```

تمرین : برای گرامر زیر نحوه ساختاریابی عبارت a+b*c را با استفاده از جدول SLR(1) مشخص کنید.

E ⟶ E+E
E ⟶ E*E
E ⟶ id
E ⟶ (E)

مقایسه SLR(1) و LL(1) :

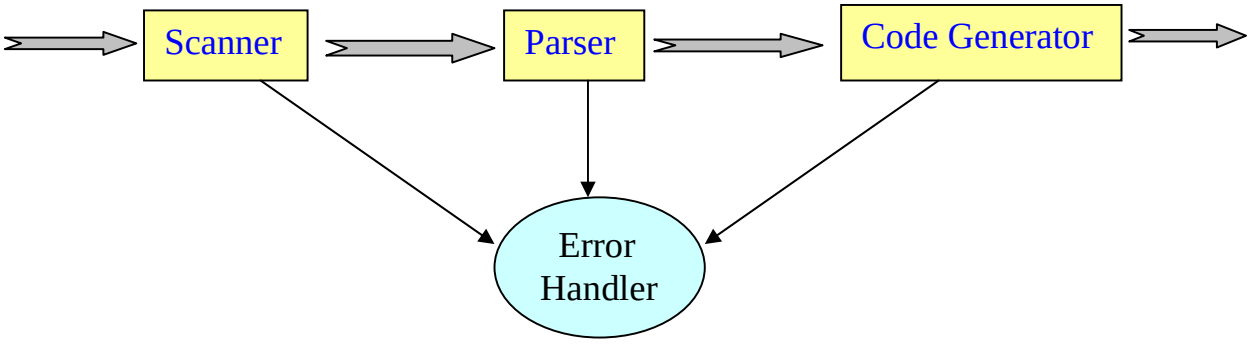
Order در جدول SLR(1) برابر P*L می باشد که در آن L ماکزیمم طول سمت راست و P تعداد قواعد تولید می باشد.

Order در جدول LL(1) برابر N*T می باشد که در آن T تعداد Tokenها می باشد.

حجم جدول SLR(1) بیش از دو برابر جدولLL(1) می باشد.

پارسر SLR(1) هیچ مزیتی بر پارسر LL(1) ندارد و تنها در حالتی از آن استفاده می کنند که گرامر LL(1) نشود.

خطا پردازی (Error handling) :



Error Handler خطاهای زمان Compile را کنترل می کند.

انواع خطاهای زمان اجرا:

۱- خطاهایی که کامپایلربرای کشف آن نمی تواند کد تولید کند و یا اینکه اگر هم کد تولید کند ،آن کد هزینه بر است و به صرفه نیست . کشف این نوع خطا را بهتر است به عهده سیستم عامل گذاشت مانند خطای Overflow.

۲- بعضی از خطاها هم توسط سیستم عامل قابل تشخیص نیستند که در این مورد کامپایلر باید کد کشف خطا را تولید کند مانند خطای Subscript out of range .

۳- خطاهایی که کامپایلربرای کشف آن می تواند کد تولید کند ولی این کار هزینه بسیار بالایی دارد. کشف این نوع خطا را بهتر است به عهده سیستم عامل گذاشت مانند خطای تقسیم بر صفر.

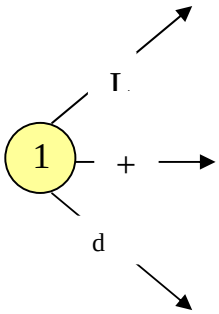
انواع خطاهای زمان کامپایل:

این نوع خطا توسط Scanner، Parser و Code Generator کشف می شوند.

خطاهای قابل کشف توسط Scanner :

۱- خطاهای واژه ای (Lexical):

مثلا در گراف روبرواگر ورودی Scanner در State شماره یک



علامت ؟ باشد ، چنین خطایی رخ می هد.

۲- خطاهای مفهومی (Semantic):

به عنوان مثال اگر در حالتی که متغیر InDCL روشن است ،id در Symtab پیدا شود ،این Error رخ می دهد. کشف این خطا خارج از وظایف واژه یابی Scanner است.

خطاهای قابل کشف توسط Parser :

خطاهای دستوری (Syntax Error) را کشف می کند. مثلا در گرامر LL(1) ، Match نشدن Top_{SS} با ورودی و یا در گرامر SLR(1) ، مراجعه به خانه ای که در آن E یا CE نوشته شده است ، خطاهای دستوری هستند که توسط Parser کشف می شوند.

خطاهای قابل کشف توسط Code Generator :

خطاهای مفهومی (Semantic Error) را کشف می کند. این خطاها عمدتا توسط روتین Check Type کشف می شوند.مثلا خطای جمع یک کاراکتر با یک عدد صحیح اینجا کشف می شود.

نکته :

فرض کنید در ورودی عبارت if A+B then ... را داشته باشیم که A و B اعداد صحیح هستند ، چون در Parser داریم:

st ⟶ if BE then ...

Parser بعد از if بدنبال یک عبارت Boolean می گردد و از آنجایی که A+B عبارت Boolean نیست ، لذا Parser در اینجاSyntax Error می گیرد.اگر در روال مفهومی Branch از Chek Type استفاده می کردیم ، خطای فوق به عنوان یک خطای مفهومی ، کشف می شد.

پس از کشف خطا ، کامپایلر دو راه دارد:

۱- برنامه را متوقف ساخته و پیغام خطا بدهد که در اینحالت می تواند دو روش را انتخاب کند:

الف) پس از رفع خطا توسط برنامه نویس ، کامپایلر از همان خط به بعد کامپایل را ادامه بدهد.

برای این منظور نیاز به یک Editor هوشمند داریم.

ب) پس از رفع خطا توسط برنامه نویس ، کامپایلر از ابتداشروع به کامپایل کند.

۲- کامپایلر خطاها را رد کند و در نهایت لیستی از خطاها را بر گرداند که در اینحالت نیز می تواند دو روش را انتخاب کند:

الف) یک یا بیشتر کاراکتر را در ورودی Insert یا از ورودی Delete کنیم، تا Match صورت بگیرد.

ب)Top_{SS} را به نحوی عوض کنیم ، تا Match صورت بگیرد.