

بناام خدا



دانشکده فنی دانشگاه تهران
گروه برق و کامپیوتر

جزوه درس طراحی و ساخت کامپایلرها

مدرس : دکتر قاسم جابری پور

تهیه و تنظیم :
محسن احمدوند
هادی پرنده افشار

فهرست مطالب :

صفحه	عنوان
۲	۱- مفاهیم اولیه
۴	۲- واژه یاب
	۳- تعریف ساختار یاب بصورت گراف
	۱-۳- گراف عبارات ریاضی
	۲-۳- یافتن آرگومانهای دستورت کد ساز در Symbol Table
	۳-۳- گراف عبارت شرطی
	۴-۳- آرایه ها

۱- مفاهيم اوليه:

همانطور که میدانید زبانها از نظر پیچیدگی به ۴ دسته تقسیم میشوند.

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

از بین این دسته زبانهاي مستقل از متن برای برنامه هاي سطح بالا که نزدیک به زبان انسان

باشند مناسب ترند چون می توان با الگوریتمهاي از درجه $O(n)$ تعلق يك عبارت را به آن

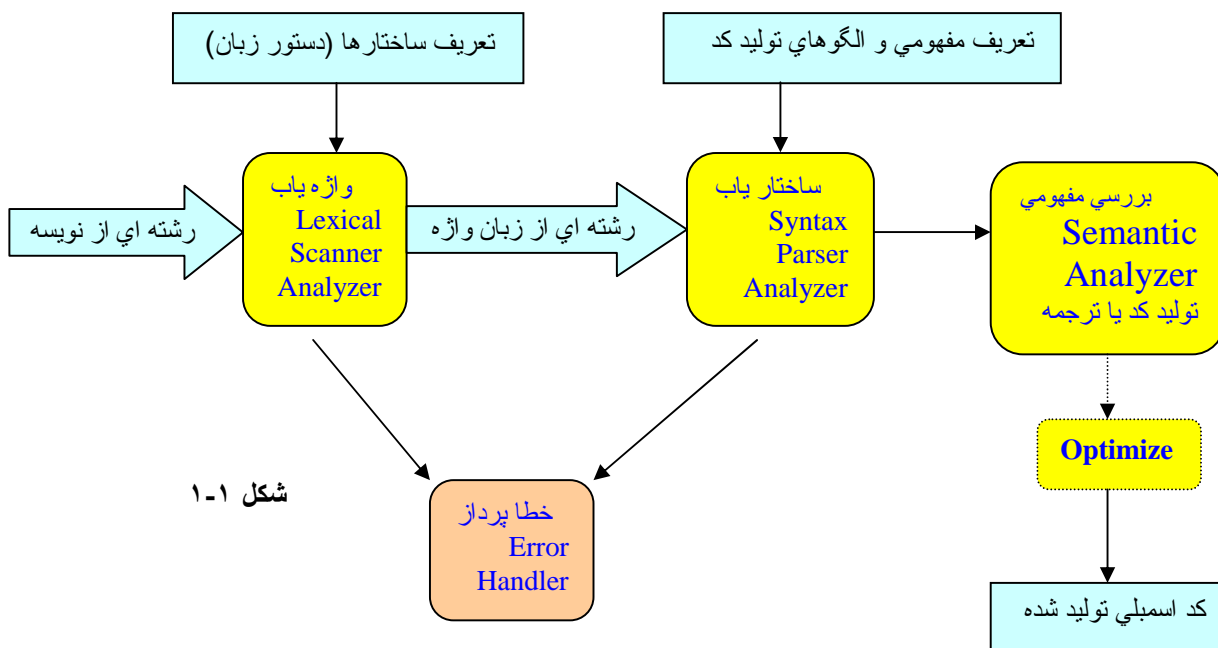
بررسی کرد که در آن n طول عبارت فوق می باشد. تشخیص این تعلق بوسیله عملیات

Parse انجام میشود.

کامپایلر ابزاري است که برنامه اي به زبان سطح بالا را گرفته و معادل همان برنامه را به زبان

سطح پایین بر می گرداند. یعنی رشته ای از نویسه ها را که برآحتي توسط ماشین اجرا می شود

تولید می کند. به شکل زیر توجه کنید:



شکل ۱-۱

ل

واژه یاب مجموعه ای از نویسه ها را گرفته و زبان واژه های استخراج شده را در اختیار Parser قرار می دهد.

واژه های زبان می توانند کلید واژه (Terminal یا Token) مانند $+$, \times , $=$, if , for , $begin$, $:=$, a , b , $temp$, get , ... باشند.

تشخیص Comment و Blank و رد کردن آنها توسط واژه یاب انجام می شود.
به مثال توجه نمایید:

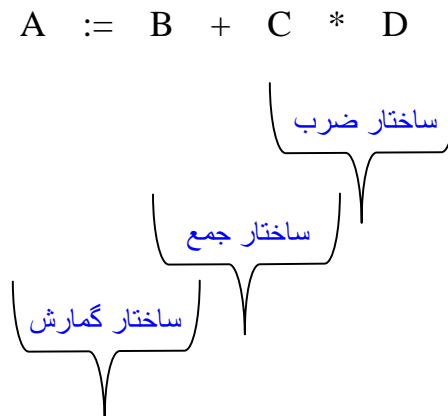
مثال :

$A := B + C * D$

واژه های زبان:

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

ساختارها:



توجه داریم که نوع C , D در تعیین ساختار ضرب تاثیری ندارد. زیرا در آن صورت گرامر

فوق حساس به متن خواهد شد که برای $O(n)$ نامناسب است. در واقع عمل $Check Type$

به بررسی مفهومی برمی گردد.

۲-واژه یاب:

شبه کد زیر واژه یابی مشابه زبان Pascal را نشان می دهد:

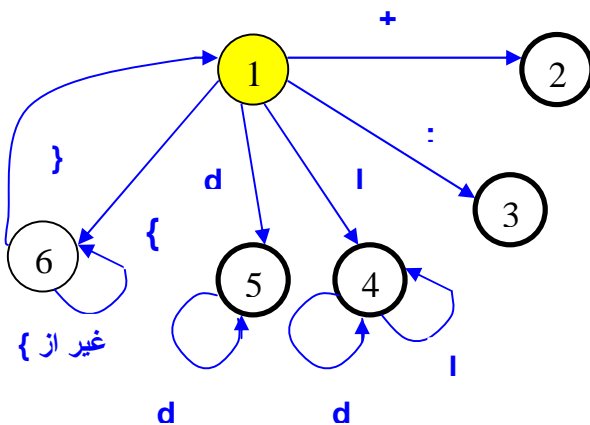
```

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
  
```

برنامه ۱-۲

واژه های يك زبان را میتوان با Finite Automata تولید کرد. ماشین زیر برای گمرم بالا

می باشدو همچنین Comment ها را هم رد می کنید:



شکل ۱-۲

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 از زبانهای 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 و یک 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 می‌دهیم. همراه پارسر یک پشته وجود دارد که به آن PS می‌گویند، که در آن آخرین راس ترک شده از گراف، هنگامی که به یک زیرگراف می‌رویم، قرار می‌گیرد.

نکته: هر زبان، یک گراف شروع و صفر یا بیشتر گراف فرعی دارد.

واژه زبان:

به واژه‌ای که در جملات زبان دیده می‌شود، واژه زبان می‌گویند.

واژه نحوی:

واژه‌ای که در جملات زبان دیده نمی‌شود و برای توصیف بکار می‌رود.

واژه مفهومی:

اسکنر با عبور از روی واژه مفهومی، روال مفهومی مربوطه را فراخوانی می‌کند. برای شناسایی واژه مفهومی از علامت @ استفاده می‌کنیم.

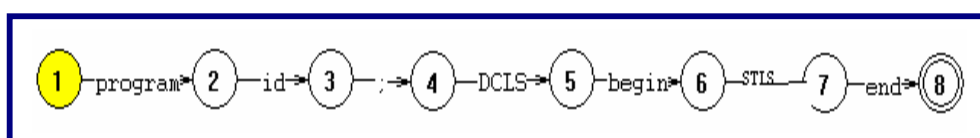
روالهای مفهومی:

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

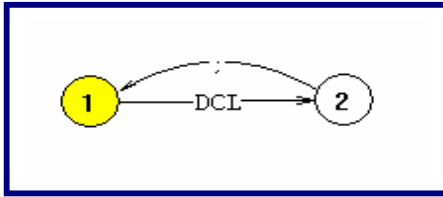
در واقع کد ساز مجموعه‌ای از روالهای مفهومی می‌باشد.
نکته:

این روال مفهومی است که سازگاری TYPE ها را بررسی می‌کند.

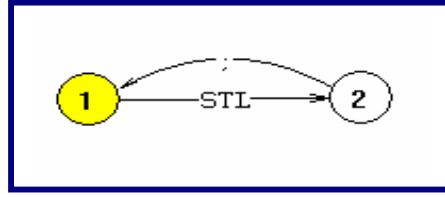
به عنوان مثال برای گرامر یک برنامه پاسکال گرافهای زیر را داریم:



DCLS :



STLS :



به عنوان مثال در گراف اصلی id و Program واژه‌های زبان و STLS و DCLS

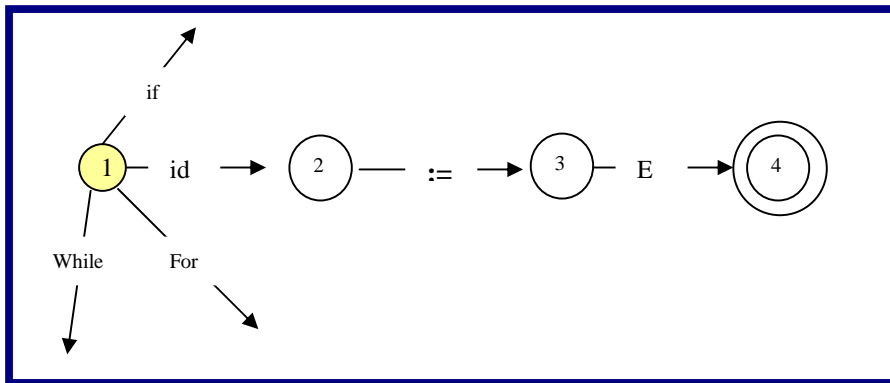
واژه‌های نحوی می‌باشند، که واژه‌های نحوی خود از یک گراف تشکیل شده است.

- به دلیل وجود ساختارهای Recursive (مانند if های تودرتو) نمی‌توان زبان را با DFA

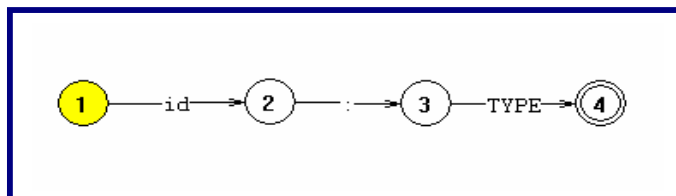
نشان داد.

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

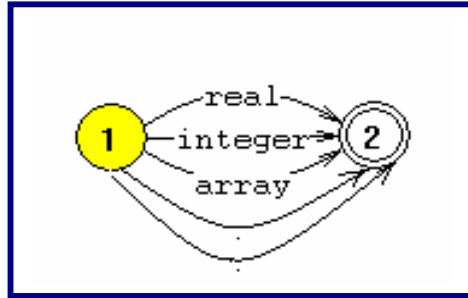
STL :



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



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

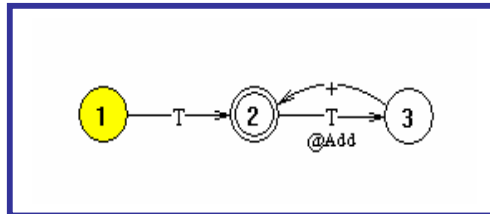


: (SS) Semantic Stack

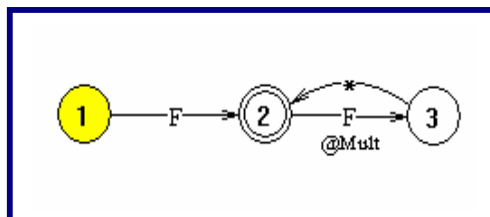
اسکنرپس از دیدن هر متغیر، شماره آن را در یک پشته قرار می دهد که به این پشته SS می گویند، و روالهای مفهومی برای انجام عمل مورد نظرشان بر روی متغیر ها از SS استفاده می کنند، به این ترتیب که بسته به نوع عملگر، یک یا دو عنصر بالای SS را که اندیس متغیرها در Symbol Table می باشد Pop می کنند و با مراجعه به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

گراف عبارات ریاضی:

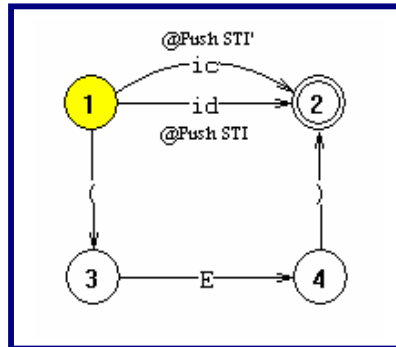
E:



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` کاملاً 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 مجدداً 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 و بهینه 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.

ل

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

این جدول همانطور که در زیر آمده است از اسم متغیر ، شماره متغیر ، نوع متغیر و 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	

Adr		
Adr		
Adr	Type	
Adr	Type	Flag
Adr	Type	Flag
Adr	Type	Flag

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

در Symtab علاوه بر متغیرهای دیده شده توسط اسکنر، متغیرهای موقت که جهت قرار

دادن نتایج میانی محاسبات مورد استفاده قرار می گیرند ، نیز وجود دارد که فیلد DSCP آنها

دارای سه بخش آدرس و نوع و پرچم 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 کلیدی یک زبان (مانند Begin و end در پاسکال) نیز در Symtab قرار داده

می شوند که در فیلد نوع آنها و اژه مستقل قرار می گیرد.

متغیر عمومی STI:

اسکنر با دیدن هر شناسه ، اندیس آنرا در متغیر عمومی STI قرار می دهد. چنانچه در بخش

DCL باشیم و متغیری که اسکنر آن را دیده در Symtab وجود نداشته باشد، آن متغیر در

Symtab قرار داده می شود و اندیس 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.

متغیر عمومی InDCL :

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

وقتی اسکندر شناسه را در Syntab پیدا کرد، اگر InDCL 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 مستقیم یا بلافصل باشد برای مشخص کردن مود آدرس دو بیت به ابتدای فیلد آدرس اضافه می کنیم. این دو بیت به صورت زیر کد گذاری می شوند:

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

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. در این جدول نوع عملگر و آدرس (های) عملوند (های) مربوطه قرار می گیرد.

ل

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 :

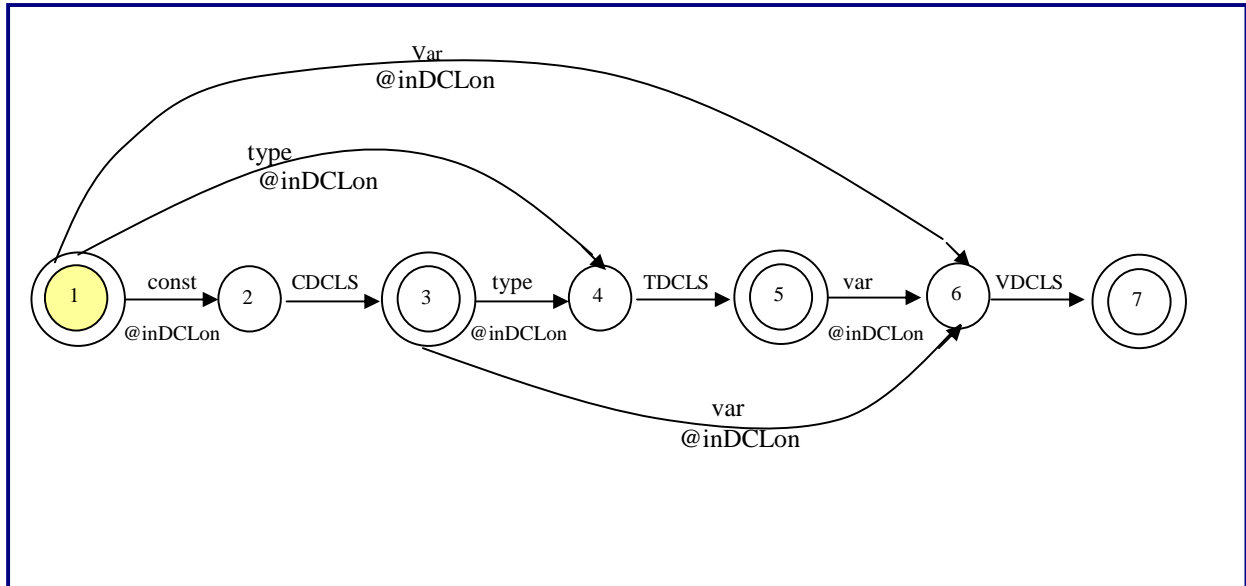
همواره به اولین خانه حافظه Allocate نشده اشاره می کند. در شروع کامپایل ، مقدار ADR برابر صفر است.

تمرین :

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

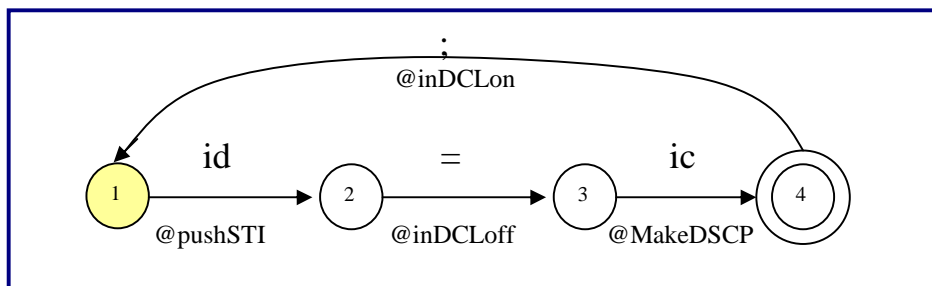
حال در ادامه ، گرافهای Declaration یک برنامه پاسکال را با جزییات بیشتری مورد بررسی قرار می دهیم:

DCLS :

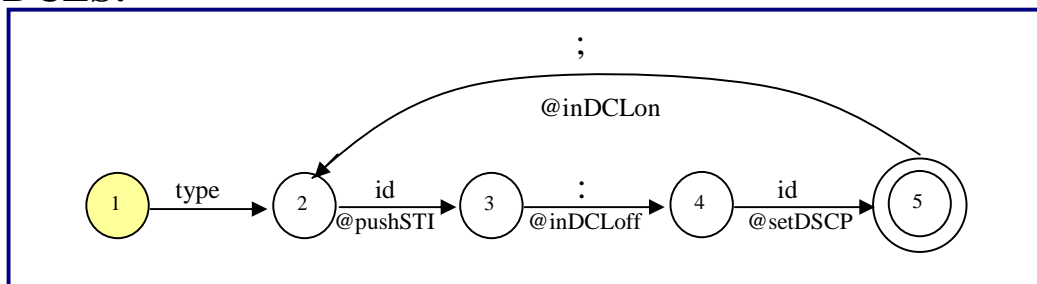


در گراف بالا CDLS مخفف Constant Declaration مخفف TDCLS و Variable Declaration مخفف VDCLS و Type Declaration مخفف می باشند.

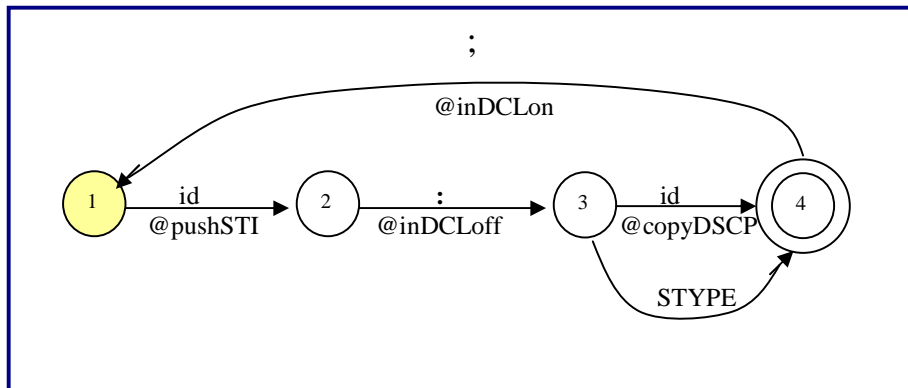
CDCLS:



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 ;
  
```

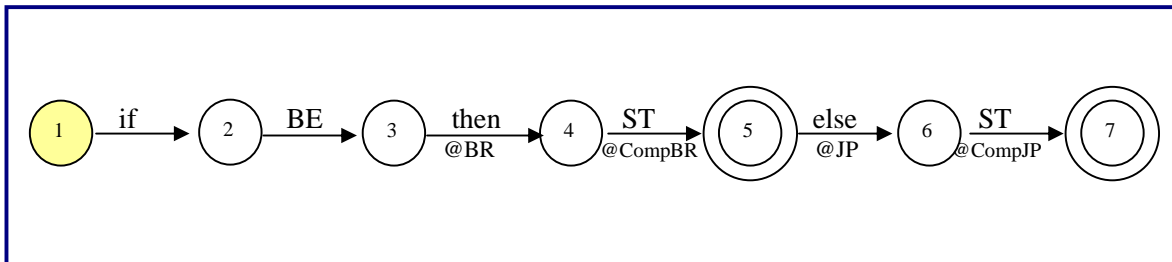
Copy DSCP:

```

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[Topss].DSCP.adr ;
Pushss (pc);
Pc := pc+1;
```

CompBR :

```
M [Topss].opr2 := pc+1 ;
Popss ;
```

JP :

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

CompJP :

```
M [Topss].opr1 := pc ;
Popss ;
```

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

همین مطلب در مورد روالهای @JP و @CompJP نیز صادق است.

تمرین :

روالهای مفهومی ساختار شرطی `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 Syntab :

Descriptor یک آرایه از چهار قسمت تشکیل شده است : `adr` ، `lb` ، `type` ، `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 و بعد از آن ، در کلیه آدرس دهی ها مورد استفاده

قرار می گیرد. (این مساله برای بهینه سازی بسیار مهم است.)

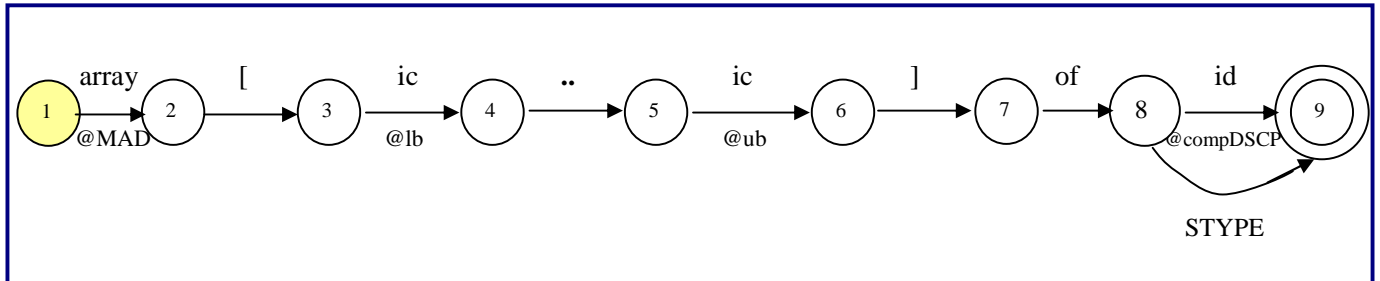
فیلد `type` نوع آرایه را و `ub` حد بالایی آرایه را مشخص می کند.

ل

تعریف یک آرایه :

گراف مربوط به تعریف آرایه در پاسکال به صورت زیر است:

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 ;
```

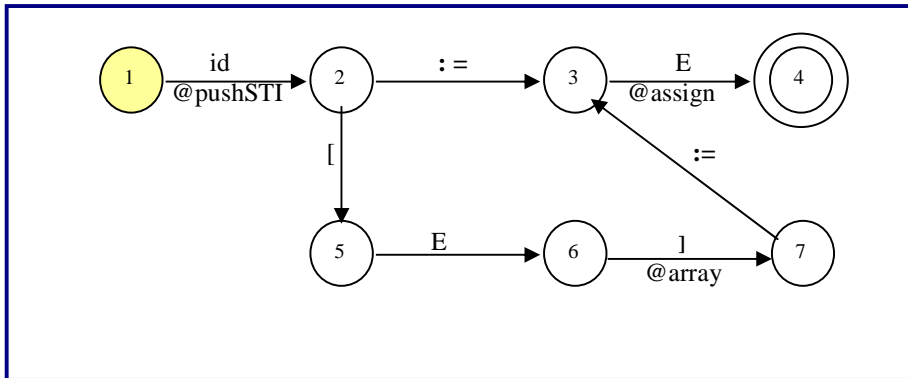
CompDSCP :

```
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 ;
```

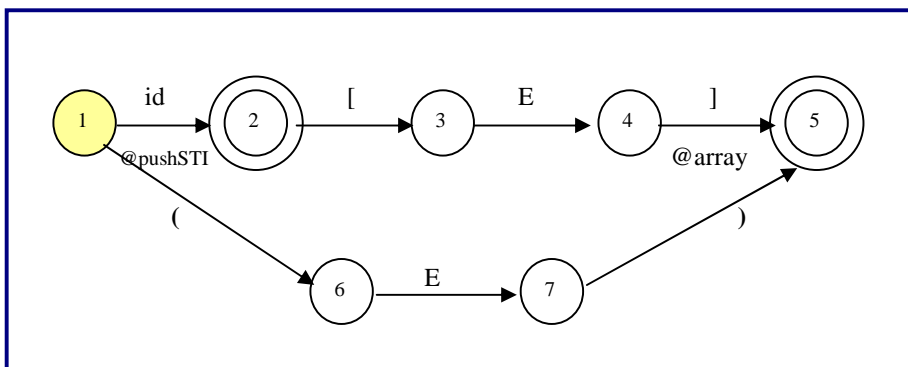
تمرین :
 در گراف بالا Scanner با دیدن 1..10 بازا 1 یک عدد اعشاری رد می کند ، در حالی که Parser انتظار دارد یک عدد صحیح رد شود. گراف را طوری اصلاح کنید که این اشکال بر طرف شود.

گرافهای ST و عبارات 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 :

```

STIe := TopSS ; PopSS ;
STILHS := TopSS ; PopSS ;
Rt := CheckType ( ` ` , STIe , STILHS ) ;
M[pc].op := ` ` ;
M[pc].opr1 := Symtab[STILHS].DSCP.adr ;
M[pc].opr2 := Symtab[STIe].DSCP.adr ; Release (STIe) ;

```

array :

```

STIe := TopSS ; PopSS ;
STIa := TopSS ; PopSS ;
RT := CheckType( `array` , STIa , STIe ) ;
M[pc].op := `*` ;
M[pc].opr1 := Symtab[STIe].DSCP.adr ; Release(STIe);
M[pc].opr2 := Sizeof(Symtab[STIa].DSCP.type) ;
STIT := GetTemp ( `int` ) ;
M[pc].Res := Symtab[STIT].DSCP.adr ;
Pc := pc+1 ;
M[pc].op := `+` ;
M[pc].opr1 := Symtab[STIa].DSCP.adr ;
M[pc].Res := M[pc].opr2 := Symtab[STIT].DSCP.adr ;
PushSS (STIT) ;
Symtab[STIT].DSCP.type := RT ;
Symtab[STIT].DSCP.adr + := 2K ;

```

در روال فوق STI_e اندیس جواب E است.

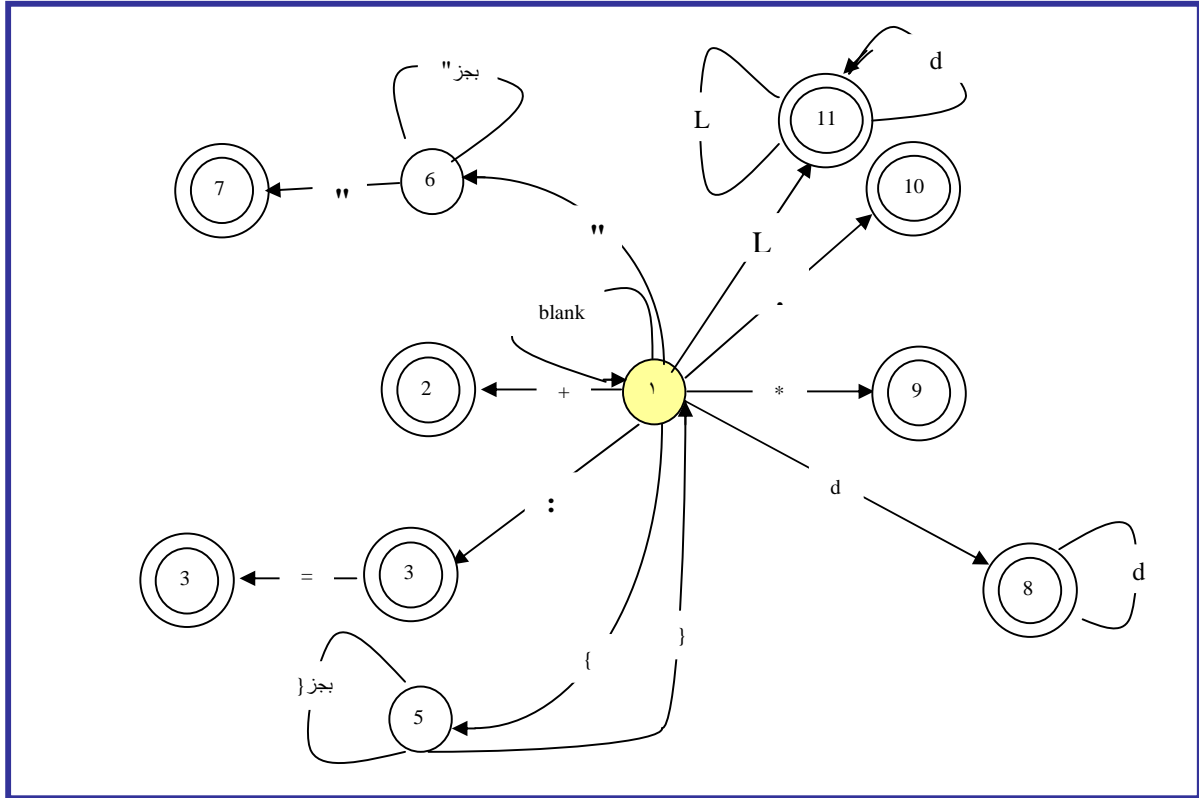
تمرین :

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 اسکنر بر 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);

ل

```
'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) Comment ها 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 هر بار بسط سمت چپ ترین متغیر برای بسط انتخاب می شود.

اشتقاق راست : (Rightmost Derivation)

در هر بار بسط سمت راست ترین متغیر برای بسط انتخاب می شود.

ساختار یاب کل به جز: (TopDown Parser)

```

Procedure TDParse
  Pushps($s);
  Token:=Scanner;
  Loop
  Case Topps of
    واژه نحوی
      Prod:=Getprod(Topps,Token);
      If Prod=0 then Error;
      Popps;
      Pushps(RHS[Prod]);
    واژه زبان
      If Topps<> Token Then Error;
      Popps;
      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 پشت 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

ل

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

RHS-۶ 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 و جدول 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 می‌دهد:

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, STI_b	
\$ E` @add T`	$*c$	6	STI_a, STI_b	
\$ E` @add T` @mult F *	$*c$		STI_a, STI_b	
\$ E` @add T` @mult id @pushsti	c		STI_a, STI_b	
\$ E` @add T` @mult id	c	7	STI_a, STI_b, 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- در ستون مربوط به PS و SS سمت چپ ترین عنصر، عنصر پایین 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 او ۲ بنویسید.

تمرین :

پارسر را برای $a*(b+c)$ اجرا و نتیجه 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 باشند. بنابراین تابع فوق با توجه به ورودیها یش خانه مناسب را انتخاب می نماید و عدد موجود در آن خانه را به عنوان خروجی برمی گرداند. در زیر جدول 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

ل

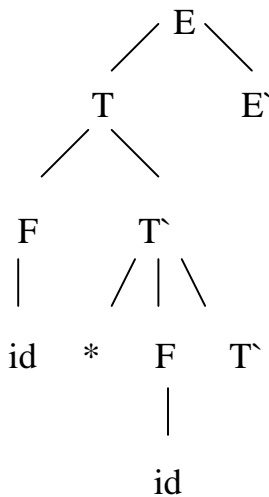
Table driven است. زیرا برای تغییرکد کافی است جدول پارس را تغییر داد.

	+	*	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:

هر اشتقاقی 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 TDParse.

نکته:

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

و حجیم است.

ل

مثال:

گرامر زیر LL(2) می باشد:

$$\left\{ \begin{array}{l} 7 \quad F \longrightarrow id \\ 9 \quad F \longrightarrow id [E] \end{array} \right.$$

و به طریق زیر LL(1) می شود:

$$\left\{ \begin{array}{l} 7 \quad F \longrightarrow id F^{\wedge} \\ 9 \quad F^{\wedge} \longrightarrow \\ 10 \quad F^{\wedge} \longrightarrow [E] \end{array} \right.$$

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 و به کمک آنها جدول پارس را می سازیم.

$$\begin{array}{l} 1 \quad E \longrightarrow T E^{\wedge} \\ 2 \quad E^{\wedge} \longrightarrow \\ 3 \quad E^{\wedge} \longrightarrow + T E^{\wedge} \\ 4 \quad T \longrightarrow F T^{\wedge} \\ 5 \quad T^{\wedge} \longrightarrow \\ 6 \quad T^{\wedge} \longrightarrow * F T^{\wedge} \\ 7 \quad F \longrightarrow id F^{\wedge} \\ 8 \quad F \longrightarrow (E) \\ 9 \quad F^{\wedge} \longrightarrow \\ 10 \quad F^{\wedge} \longrightarrow [E] \end{array}$$

:First

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

وبه طریق زیر محاسبه می شود:

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

تعریف :

A میرا است اگر:

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

بنابراین نتیجه می گیریم که:

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

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

$$\begin{aligned} \text{First}(F^{\backslash}) &= \text{First}(9) + \text{First}(10) = \{ \} \\ \text{First}(F) &= \text{Fisrt}(7) + \text{First}(8) = \{ \text{id} \} + \{ () \} = \{ \text{id}, () \} \\ \text{Fisrt}(T^{\backslash}) &= \text{Fisrt}(5) + \text{First}(6) = \{ * \} \\ \text{First}(T) &= \text{Fisrt}(4) = \text{First}(F) = \{ \text{id}, () \} \\ \text{Fisrt}(E^{\backslash}) &= \text{Fisrt}(2) + \text{First}(3) = \{ + \} \\ \text{First}(E) &= \text{Fisrt}(1) = \text{First}(F) = \{ \text{id}, () \} \end{aligned}$$

در مثال بالا واژه های نحوی $F^{\backslash}, T^{\backslash}, E^{\backslash}$ ، میرا (Nullable) می باشند.

:Follow

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

$$\text{Follow}(A) = \{ a \text{ Belong } T \mid s \Rightarrow^* \dots Aa \dots \}$$

بطور مثال برای قواعد تولید بالا داریم:

$$\text{Follow}(E) = \{ (), \}$$

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 برای قواعد تولید بالا داریم:

$$\text{Follow}(E') = \text{Follow}(E) = \{ \text{),] } \}$$

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 ، قرار می دهیم.

	+	*	Id	()	[]	\$
E			1	1				
E'	3							
T			4	4				
T'		6						
F			7	8				
F'						10		

ل

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

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 باشد ، زیرا واژه نحوی سمت چپ ، عیناً در ابتدای

سمت راست قاعده تولید آمده است :

$A \rightarrow Aa$

نکته:

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

راه حل کلی حذف چپ گردی :

فرض کنید گرامر چپ گردی به فرم زیر داشته باشیم:

$$1 \left\{ \begin{array}{l} A \rightarrow Aa_1 \\ A \rightarrow Aa_2 \\ \vdots \\ A \rightarrow Aa_n \end{array} \right.$$

ل

$$2 \left\{ \begin{array}{l} A \longrightarrow b_1 \\ A \longrightarrow b_2 \\ \vdots \\ A \longrightarrow b_m \end{array} \right.$$

قسمت ۱ را به فرم زیر می نویسیم:

$$A \Longrightarrow^* A(a_1 | a_2 | \dots | a_n)^*$$

قسمت ۲ را به فرم زیر می نویسیم:

$$A \Longrightarrow^* (b_1 | b_2 | \dots | b_m)$$

از تلفیق دو عبارت فوق به نتیجه زیر می رسیم:

$$A \Longrightarrow^* (b_1 | b_2 | \dots | b_m) (a_1 | a_2 | \dots | a_n)^*$$

عبارت فوق را می توان به صورت زیر نمایش داد:

$$A \longrightarrow BA'$$

که در آن :

$$B \Longrightarrow^* (b_1 | b_2 | \dots | b_m)$$

و

$$A' \Longrightarrow^* (a_1 | a_2 | \dots | a_n)^*$$

بدین ترتیب چپ گردی A برطرف شد.

گرامر مبهم:

اگر در گرامری 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 if BE then ST EP

EP \longrightarrow

EP \longrightarrow else ST

در گرامر فوق متن if be₁ then if be₂ then st₁ else st₂ به دو طریق زیر (با یک نوع

اشتقاق) به دست می آید:

ل

ST \longrightarrow if BE then ST EP \longrightarrow if BE then if BE then ST EP EP

به ازای ورودی else دو راه مختلف داریم:

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

(۲) بجای EP اول ، else st ، EP دوم ، Null قرار می دهیم.

بنابراین گرامر فوق مبهم است.

تمرین :

جدول پارس اچ-اگر امر if...then...else را بکشید.
این گرامر را با endif بنویسید و جدول آنرا بکشید.
گرامر if...then...else بدون endif را طوری بنویسید که قانون تطبیق else با
آخرین if رعایت شود.

پارسر (1) SLR (سار-۱) :

قاعده تولید زیر را در نظر بگیرید:

- 1 $E \longrightarrow E+T$
- 2 $E \longrightarrow T$
- 3 $T \longrightarrow T*F$
- 4 $T \longrightarrow F$
- 5 $F \longrightarrow id$
- 6 $F \longrightarrow (E)$

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

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

ل

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

قاعده :

متن گونه را از ابتدا به انتها می خوانیم و در آن به دنبال 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 (1) SLR بودن :

شرط (1) SLR 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 (1) SLR (سار-1) :

دیاگرام (1) SLR مجموعه ای از State ها می باشد. هر State دارای تعدادی ورودی و خروجی می باشد. در parser 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 قرار می 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 قاعده های 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.

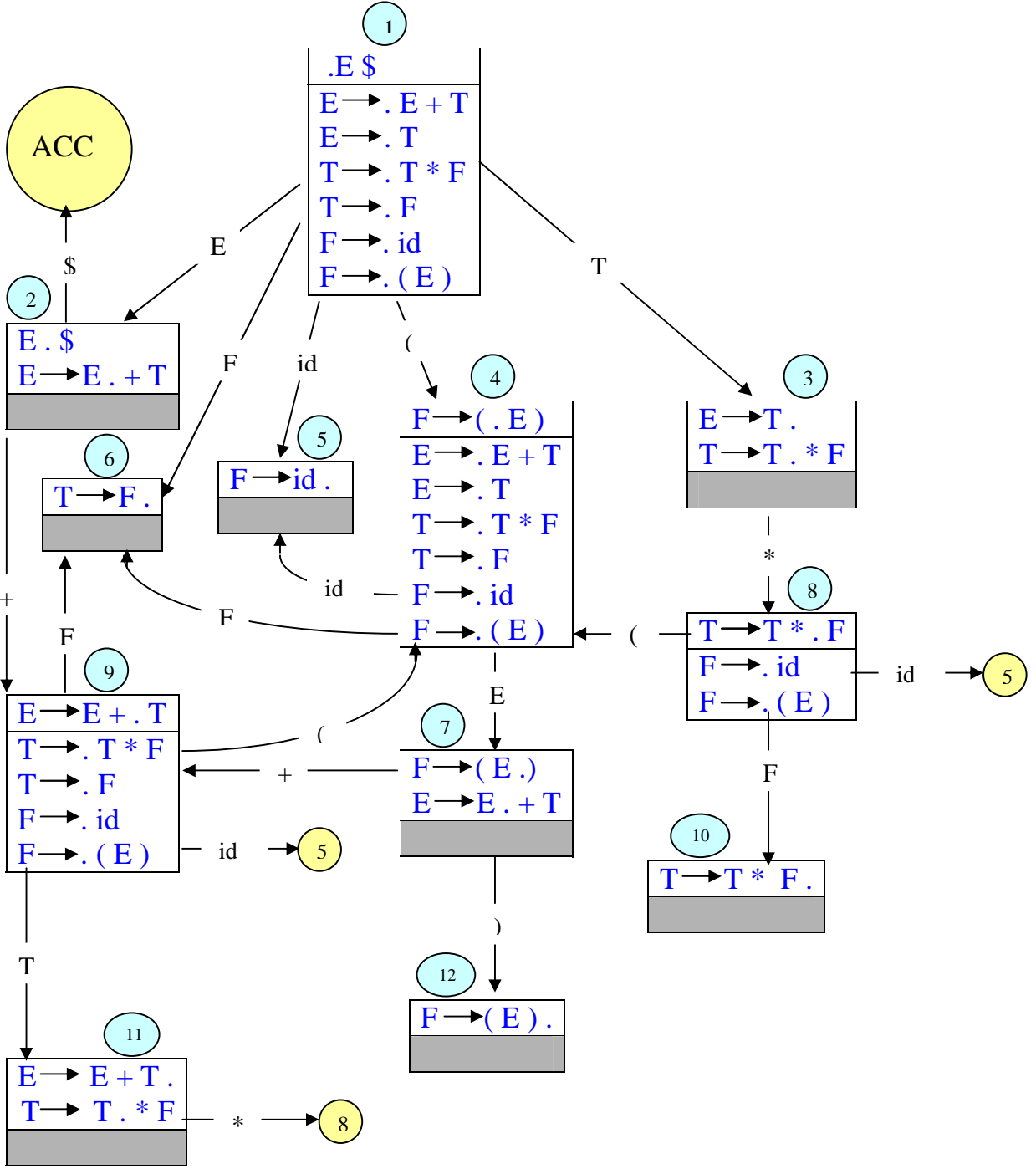
قسمت منتج :

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 در ابتدای 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 :

U



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. در مرحله بعد Parser با گرفتن ورودی بعدی اش ، وارد State دیگری می شود و شماره این State درون Parse Stack قرار می گیرد. اگر ورودی 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 شود و 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 در انتهای هر عبارت علامت \$ وجود دارد و Parser 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 دیگرام $SLR(1)$ فوق نشان 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 دیگرام $SLR(1)$ بدست می آید.

ل

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

- 1 $E \rightarrow E+T$
- 2 $E \rightarrow T$
- 3 $T \rightarrow T * F$
- 4 $T \rightarrow F$
- 5 $F \rightarrow id$
- 6 $F \rightarrow (E)$

مرحله ۱ :

ابتدا جدولی به صورت زیر تشکیل می دهیم که عناصر افقی مجموعه واژه های زبان و واژه های

نحوی و علامت \$ 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 Shift ها و GoTo های State i را به صورت خلاصه شده S و G 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 Reduce های 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 در زیرستونهای 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	R ₁	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	R ₁	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	R ₁	S ₈	E ₃₃	E ₃₄	R ₁	CE ₂₂	CE ₂₃	CE ₂₄	R ₁

توجه کنید که ردیف 10 و 6 را نمی توان حذف کرد. زیرا دستورهای G_6 و G_{10} وجود دارد.

نحوه ساختاریابی در یک گرامر SLR(1) با استفاده از جدول 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) و سپس 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 جدول GT برای ساختاریابی از Procedure های زیر استفاده می کنیم :

Procedure BUParser

```

PushPS(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
    
```

ل

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.

$E \longrightarrow E+E$
 $E \longrightarrow E * E$
 $E \longrightarrow id$
 $E \longrightarrow (E)$

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

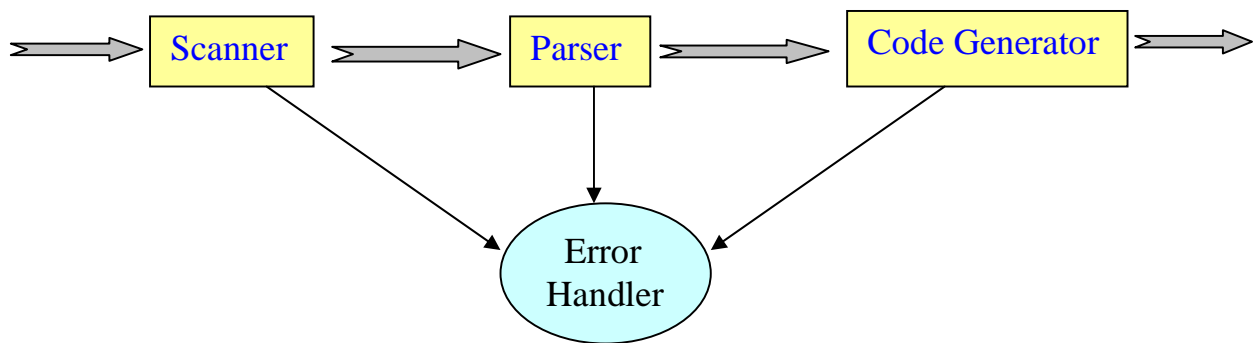
تعداد قواعد تولید می باشد.

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

حجم جدول SLR(1) بیش از دو برابر جدول LL(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 کامپایلر برای 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 کامپایلر برای 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

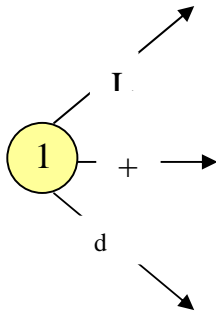
تقسیم بر صفر.

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 روبرو اگر 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 Syntab 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

TopSS 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` دنبال یک عبارت Boolean می گردد و از آنجایی که `A+B` عبارت Boolean نیست ، لذا Parser در اینجا Syntax 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 ابتدای شروع 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) `Topss` را به نحوی عوض کنیم ، تا `Match` صورت بگیرد.