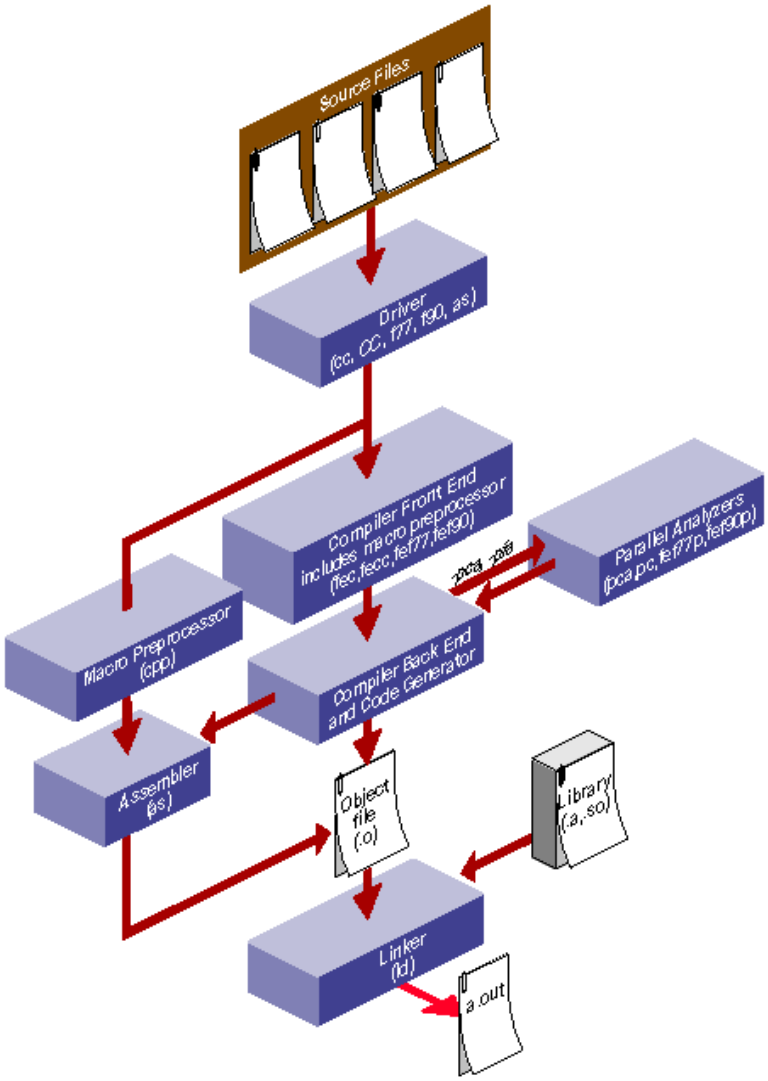
# اصول طراحی کامپایلرها





استاد : جناب آقای دکتر حاج سیدجوادی

# فهرست مطالب

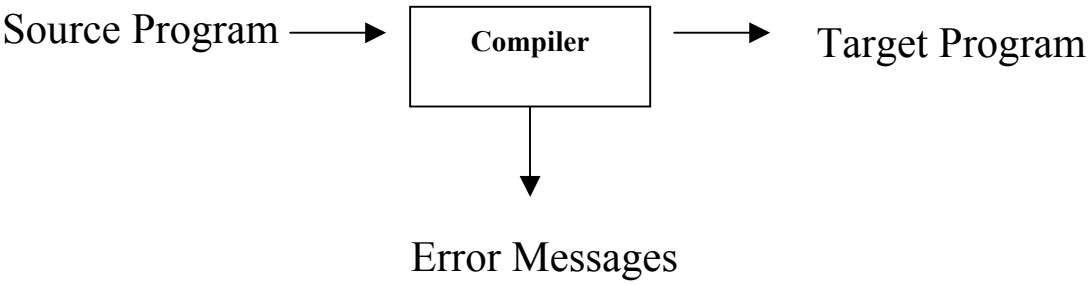
**عنوان** **شماره صفحه**

| عنوان | شماره صفحه |
|---|---|

## اصول طراحی و ساخت کامپایلرها

### ۱-۱ تعریف کامپایلر :

کامپایلر ( Compiler ) برنامه ای است که یک برنامه نوشته شده در یک زبان به نام زبان منبع ( Source Language ) را به برنامه ای معادل به زبانی دیگر به نام زبان مقصد ( Target Language ) ترجمه می کند.




### ۱-۲ مراحل کامپایل

عملیات کامپایل در شش مرحله زیر صورت می گیرد :

۱- تحلیل واژه ای ( Lexical Analysis )

۲- تحلیل نحوی ( Syntax Analysis )

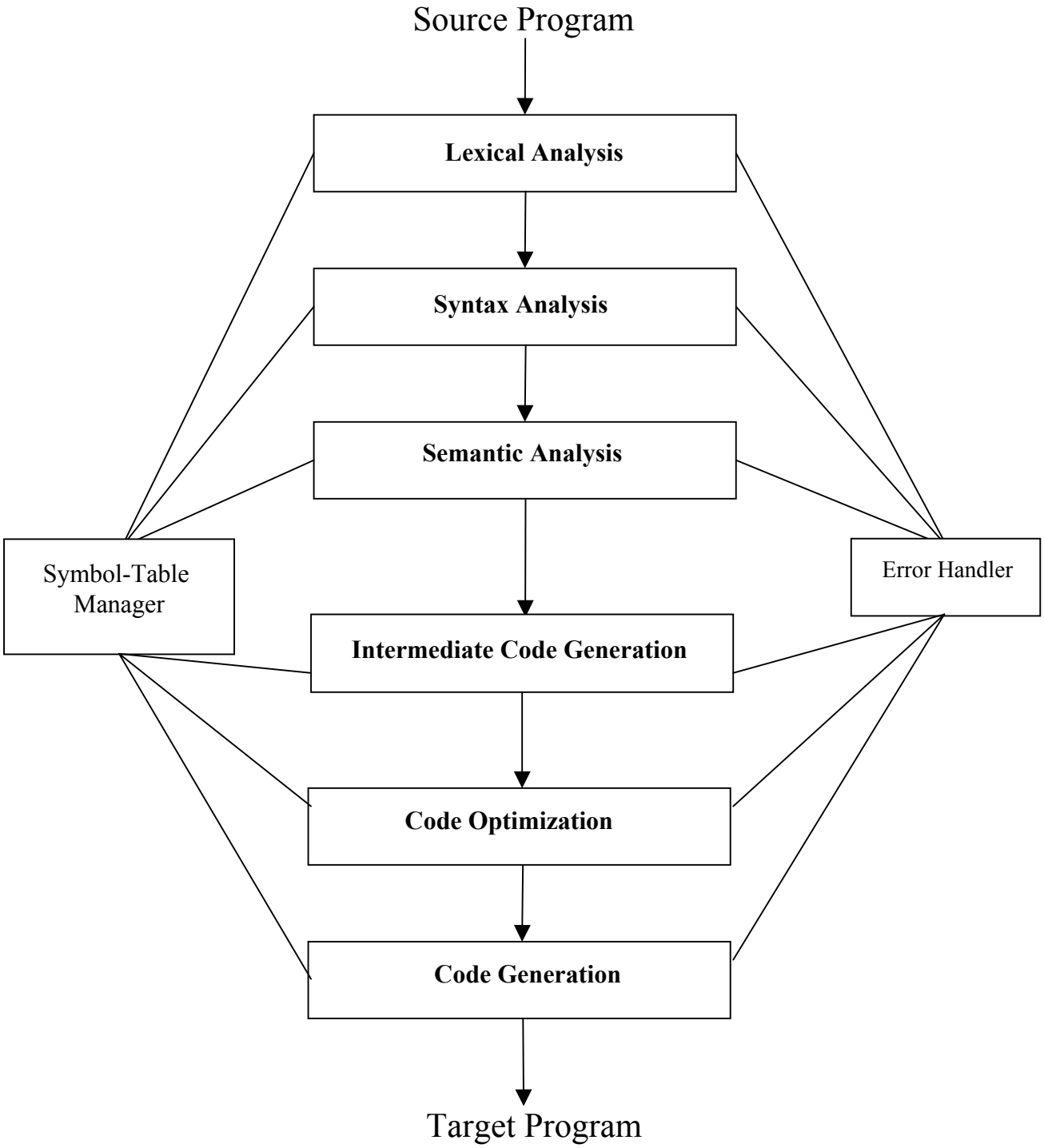
۳- تحلیل معنایی ( Semantic Analysis )

۴- تولید کد بینابینی ( Intermediate Code Generation )

۵- بهینه سازی کد ( Code Optimization )

۶- تولید کد نهایی ( Code Generation )

ارتباط بین این مراحل در شکل زیر نشان داده شده است.در کنار شش مرحله اصلی کامپایلر دو بخش دیگر بنام خطاپرداز ( Error Handler ) و جدول علائم ( Symbol Table ) نیز وجود دارد.



در مرحله اول کامپایل یعنی تحلیل واژه ای برنامه ورودی نویسه به نویسه خوانده شده و به دنباله ای از نشانه ها ( Tokens ) تبدیل می گردد.انواع مختلف نشانه ها عبارتند از : کلمات کلیدی ( Keywords ) , عملگرها ( Operators ) , جداکننده ها ( Delimeters ) , ثابت ها ( Literals ) , شناسه ها (Identifiers ) که به اسامی متغیرها و توابع و رویه ها و بطور کلی اسامی که کاربر انتخاب می کند گفته می شود.در اغلب زبانهای برنامه سازی کلمات کلیدی رزرو شده اند بدین معنی که کاربر مجاز نیست از هیچیک از آنها به عنوان اسم یک متغیر ، تابع و یا رویه استفاده نماید.اما در برخی از زبانها مثل PL/1 این محدودیت وجود ندارد.

در مرحله دوم برنامه از نظر خطاهای نحوی مورد بررسی قرار می گیرد و با استفاده از نشانه های تولید شده در مرحله تحلیل واژه ای یک درخت نحو (Syntax Tree ) ایجاد می گردد.

در مرحله سوم با استفاده از درخت نحو تولید شده در مرحله قبلی برنامه ورودی از نظر خطاهای مفهومی احتمالی بررسی می شود.

در مرحله تولید کد بینابینی یک برنامه که معادل برنامه اصلی است با یک زبان بینابینی تولید می شود.با ایجاد این کد بینابینی عملیات بعدی که کامپایلر باید انجام دهد آسان می گردد.در انتخاب زبان بینابینی باید موارد زیر در نظر گرفته شوند :

۱. تولید و بهینه سازی کد بینابینی باید ساده باشد.
۲. ترجمه آن به برنامه مقصد نیز به راحتی صورت پذیرد.

در مرحله بهینه سازی کوشش می شود تا کد بینابینی تولید شده در مرحله قبلی به نحوی بهبود داده شود بطوریکه این کار مسبب تولید کدی می شود که از لحاظ اجرایی سریعتر می باشد.

سرانجام در بخش تولید کد نهایی کد موردنظر بصورت برنامه مقصد تولید می شود.بعبارت دیگر هر کدام از کدهای بینابینی بهبود یافته به مجموعه ای از دستورات ماشین که کار مشابهی انجام می دهند تبدیل می گردند.

## خطاپرداز ( Error Handler )

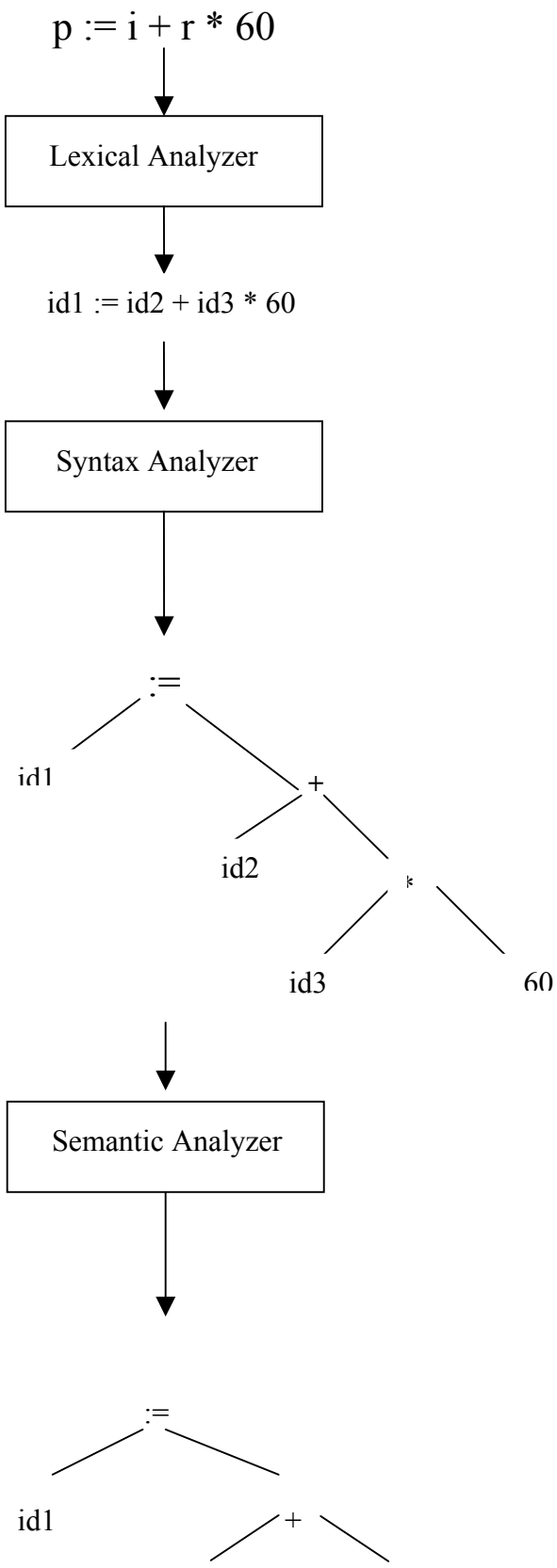
هر بار که خطایی در یکی از مرحله ها پیش بیاید رویه ای بنام خطاپرداز فراخوانده می شود. این بخش سعی می کند خطا را به نحوی برطرف کند که در نتیجه کامپایلر بتواند خطاهای بیشتری را در برنامه تشخیص دهد و با اولین خطای موجود در برنامه عمل کامپایل متوقف نگردد.معمولا پارسر و اسکنر بیشتر خطاهایی را که در یک برنامه ممکن است وجود داشته باشد تشخیص می دهند.

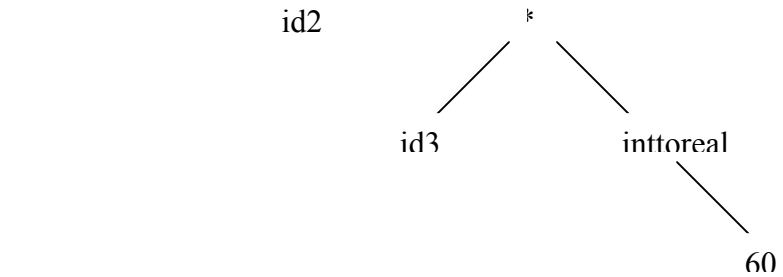
یکی از کارهای مهم و اساسی یک کامپایلر ثبت شناسه های استفاده شده در برنامه ورودی ( منبع ) و جمع آوری اطلاعات درباره مشخصات هر شناسه است.این مشخصات می توانند شامل : آدرس حافظه اختصاص داده شده به شناسه , نوع آن , محلی از برنامه که این شناسه در آن تعریف شده است (Scope ) , و در رابطه با رویه ها اسم آنها , تعداد و نوع آرگومانهای آنها , روشی که به آن طریق آرگومانها به رویه ها فرستاده می شوند.مثلا Call by Reference یا Call by Value و نوع نتیجه ای که رویه ها باز می گردانند باشد.

در جدول نشانه ها به ازای هر شناسه یک رکورد وجود دارد که این رکوردها شامل مشخصات شناسه ها می باشند.این جدول امکان دستیابی سریع به شناسه ها و مشخصات آنها را به ما می دهد.در کامپایلر و در مرحله تحلیل لغوی کلیه شناسه های موجود در برنامه اصلی وارد جدول نشانه ها می شوند.در مرحله های دیگر کامپایل این اطلاعات به جدول اضافه خواهند شد و سپس از آنها در موارد مختلف استفاده خواهد شد.

## ۳-۱ ابتدا ( Front-End ) و انتهای ( Back-End ) کامپایلرها

به چهار مرحله اول کامپایل و بخشی از مرحله بهینه سازی که بستگی به زبان منبع ( source Program ) دارد و مستقل از ماشین است Front-End کامپایلر گویند.به بخشی از مرحله بهینه سازی و مرحله آخر کامپایل که وابسته به ماشین مقصد است Back-End کامپایلر می گویند.معمولا این بخش از کامپایلر وابسته به زبان منبع نیست.شکل بعد ترجمه دستور p:=i+r*60 و تاثیر هر مرحله از کامپایلر بر روی این دستور را نشان می دهد.( i , p , r همگی از نوع real هستند)


p := i + r * 60
Lexical Analyzer
id1 := id2 + id3 * 60
Syntax Analyzer
:=
id1
+
id2
*
id3
60
Semantic Analyzer
:=
id1
+




Intermediate Code Generation

t1 := inttoreal(60)
t2 := id3 * t1
t3 := id2 + t2
id1 := t3

Code Optimizer

t1 := id3 * 60.0
id1 := id2 + t1

Code Generation

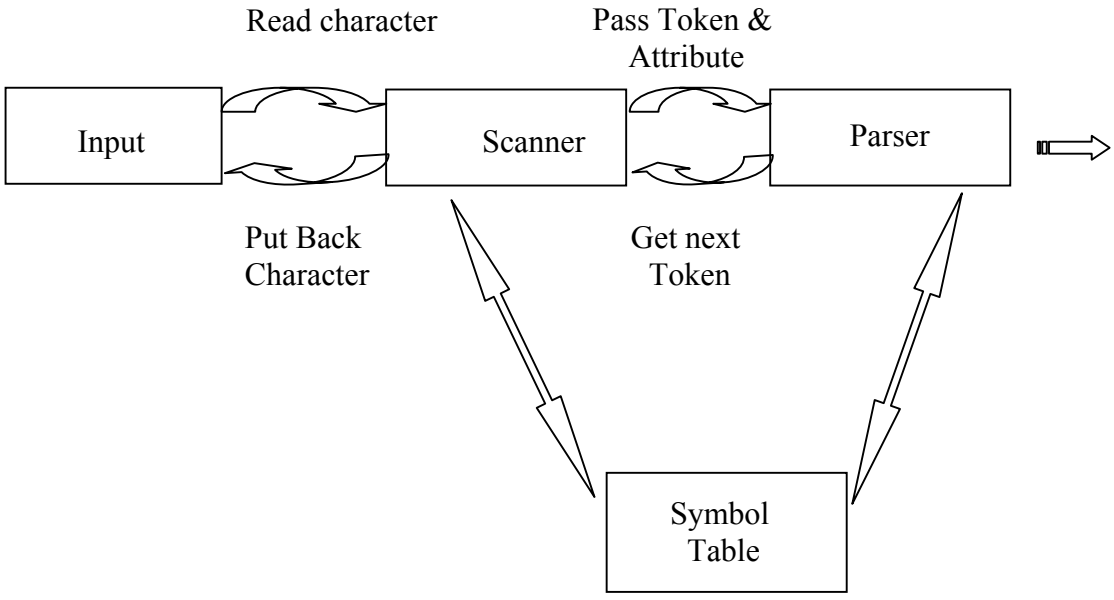
movF id3,R2
mulF #60.0,R2

movF id2,R1
ADDF R2,R1
movF R1,id1

---

## ۲- تحلیل واژه ای ( Lexical Analysis )

نخستین مرحله کامپایل تحلیل واژه ای است.به واحدی از کامپایلر که کار تحلیل واژه ای را انجام می دهد اسکنر (Scanner ) می گویند.اسکنر بین رشته ورودی و تحلیلگر نحوی یا پارسر ( Parser ) واقع است.وظیفه اصلی اسکنر این است که با خواندن کاراکترهای ورودی , توکن ها را تشخیص داده و برای پارسر ارسال نماید.رابطه اسکنر و پارسر بصورت زیر است :




به عنوان مثال در صورتی که رشته ورودی A := B + C باشد توکن های زیر تشخیص داده خواهند شد :

آدرس id , C > و < add. Op. > و < آدرس id , B > و < ass. Op. > و < آدرس id , A >
بنابراین اسکنر علاوه بر این که تشخیص می دهد که توکن یک شناسه است آدرس آن در جدول نشانه ها را نیز برای پارسر می فرستد.علاوه بر این اسکنر می تواند محل های خالی و توضیحات ( Comments ) موجود در برنامه اصلی را ضمن خواندن برنامه حذف نماید.
به آخرین توکنی که اسکنر یافته است علامت پیش بینی ( Lookahead Symbol ) و یا توکن جاری گفته می شود.

## ۲-۱ الگو (Pattern ) و واژه (Lexeme ) توکن ها

به فرم کلی که یک توکن می تواند داشته باشد الگوی آن توکن می گویند.به عبارت دیگر در ورودی رشته هایی وجود دارند که توکن یکسانی برای آنها تشخیص داده می شود .فرم کلی این رشته ها توسط الگوی آن توکن توصیف می شود.
به دنباله ای از نویسه ها که تشکیل یک توکن می دهند واژه ( Lexeme ) آن توکن می گویند.جدول زیر حاوی چند نمونه الگو و واژه است :

| Token | Lexeme | Pattern |
|---|---|---|
| Const | Const | Const |
| if | if | if |
| relation | < , > , <= , >= , <> , = | < or > or <= or >= or <> or = |
| id | pi | با حروف الفبا شروع و بدنبال آن حرف و رقم قرار می گیرد |
| Num | 3.1416 | هر ثابت عددی |
| literal | "core dumped" | هر رشته نویسه ای که بین دو علامت " قرار گیرد |

در بعضی وضعیت ها اسکنر قبل از اینکه تصمیم بگیرد چه توکنی را به پارسر بفرستد نیاز دارد که چند کاراکتر دیگر نیز از ورودی بخواند.بعنوان مثال اسکنر با دیدن علامت ' < ' در ورودی نیاز دارد که کاراکتر ورودی بعدی را نیز بخواند در صورتی که این کاراکتر = باشد توکن ' =< ' و در غیر اینصورت توکن ' < ' تشخیص داده می شود.در مورد آخر باید کاراکتر اضافی خوانده شده دوباره به ورودی بازگردانده شود.

یکی دیگر از مشکلاتی که اسکنر با آن روبرو است در زبانهایی نظیر Fortran است.در این قبیل زبانها محل خالی (Blank ) بجز در رشته های کاراکتری نادیده گرفته می شود.به عنوان مثال کلمه Do در فرترن در دستور زیر را در نظر بگیرید :

Do 5 I = 1.25 تا زمانی که به نقطه اعشار در 1.25 نرسیده باشیم نمی توان گفت که Do در این دستور کلمه کلیدی نیست بلکه بخشی از متغیر Do5I است.به همین ترتیب در دستور Do 5 I = 1 , 25 تا زمانی که علامت کاما دیده نشود نمی توان گفت که این یک حلقه Do است.

در زبانهایی که در آنها کلمات کلیدی ( Keyword ) جزو کلمات رزرو شده نیستند نظیر PL/1 اسکنر نمی تواند کلمات کلیدی را از شناسه های همنام آنها تشخیص دهد.به عنوان مثال در دستور زیر :

IF Then THEN Then = Else ; ELSE Else = Then ;

جداکردن کلمه کلیدی THEN از متغیر Then بسیار مشکل است.در این موارد معمولا پارسر تشخیص نهایی را خواهد داد.

## ۲-۲ خطاهای واژه ای ( Lexical Errors )

بطور کلی خطاهای محدودی را اسکنر می تواند بیابد زیرا اسکنر تمام برنامه ورودی را یکجا نمی بیند بلکه هر بار قسمت کوچکی از برنامه منبع را می بیند.بعنوان مثال هرگاه رشته fi در یک برنامه C برای بار اول مشاهده شود اسکنر قادر نیست تشخیص دهد که آیا fi یک املای غلط از کلمه کلیدی if است یا یک متغیر.

fi(x = = f(x))

از آنجایی که fi یک متغیر مجاز است اسکنر این توکن را به عنوان یک شناسه به پارسر می فرستد تا اینکه پارسر در این مورد تصمیم بگیرد.اما ممکن است خطاهایی پیش بیاید که اسکنر قادر به انجام هیچ عملی نباشد.در این حالت برنامه خطاپرداز (Error-Handler ) فراخوانده می شود تا آن خطا را به نحوی برطرف کند.روشهای مختلفی برای اینکار وجود دارد که ساده ترین آنها روشی موسوم به " Panic Mode " است.در این روش آنقدر از رشته ورودی حذف می شود تا اینکه یک توکن درست تشخیص داده شود.

سایر **روشهای تصحیح خطا ( Error-Recovery )** عبارتند از :

۱- حذف یک کاراکتر غیرمجاز ( تبدیل =$: به =: )

۲- وارد کردن یک کاراکتر گم شده ( تبدیل : به =: )

۳- تعویض کردن یک کاراکتر غلط با یک کاراکتر درست ( تبدیل :: به =: )

۴- جابجا کردن دو کاراکتر مجاز ( تبدیل := به =: )

---

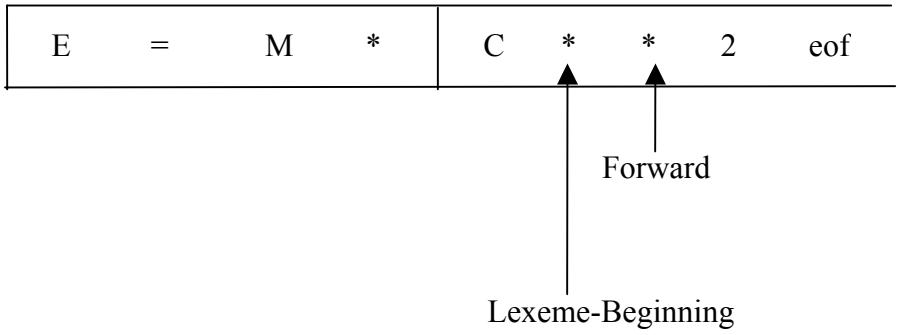
## ۲-۳ روشهایی جهت بهبود کار اسکنر

### ۲-۳-۱ استفاده از بافر

در بسیاری از زبانها اسکنر برای تشخیص نهایی توکن ها و مطابقت دادن آن با الگوهای موجود نیاز دارد که چند کاراکتر جلوتر را نیز مورد بررسی قرار دهد.از آنجایی که مقدار زیادی از زمان در جابجایی کاراکترها سپری می شود از تکنیک های بافرینگ برای پردازش کاراکترهای ورودی استفاده می گردد.

در یکی از این تکنیک ها از یک بافر که به دو نیمه N کاراکتری تقسیم شده است استفاده می کنند که در آن N تعداد کاراکترهایی است که در روی یک بلوک از دیسک جای می گیرند.به این ترتیب با یک فرمان read بجای خواندن یک کاراکتر می توان N کاراکتر ورودی را در هر نیمه بافر وارد کرد.درصورتیکه ورودی شامل تعداد کاراکترهای کمتر از N باشد بعد از خواندن این کاراکترها یک کاراکتر خاص eof نیز

وارد بافر می گردد.کاراکتر eof بیانگر پایان فایل منبع بوده و با سایر کاراکترهای ورودی به نوعی تفاوت دارد.




در بافر ورودی از دو نشانه رو استفاده می شود.رشته کاراکتری بین این دو نشانه رو معرف Lexeme جاری می باشد.در ابتدا هر دو نشانه رو به اولین کاراکتر Lexeme بعدی که باید پیدا شود اشاره می کنند.نشانه روی Forward پیش می رود تا اینکه یک توکن تشخیص داده شود.

این نحوه استفاده از بافر در بیشتر موارد کاملا خوب عمل می کند.با این وجود در مواردی که جهت تشخیص یک توکن , نشانه رو Forward ناچار است بیشتر از طول بافر جلو برود این روش کار نمی کند.به عنوان مثال دستور DECLARE (ARG1,ARG2,…,ARGn) را در یک برنامه PL/1 در نظر بگیرید.در این دستور تا زمانی که کاراکتر بعد از پرانتز سمت راست را بررسی نکنیم نمی توان گفت که DECLARE یک کلمه کلیدی است و یا یک اسم آرایه.

برای کنترل حرکت نشانه روی Forward و همچنین کنترل بافر می توان بصورت زیر عمل کرد :

```
If    Forward is at end of first half   Then begin
      reload  Second-half ;
      Forward := Forward + 1 ;
end
```

در این قسمت اگر نشانه روی Forward به انتهای نیمه اول بافر رسید , نیمه دوم با N کاراکتر جدید پر خواهد شد .

```
else if   Forward is at end of second-half   Then begin
          reload  first-half ;
          move Forward to beginning of first-half
end
else   Forward := Forward + 1 ;
```

در صورتیکه نشانه روی Forward به انتهای نیمه دوم بافر برسد نیمه اول بافر را با N کاراکتر جدید پر می کنیم و نشانه روی Forward را به آغاز بافر Set می کنیم.

### ۲-۳-۲ استفاده از نگهبانها ( Sentinels )

در حالت قبل با جلورفتن نشانه روی Forward باید چک می شد که آیا این نشانه رو به انتهای یک نیمه از بافر رسیده است یا خیر .در صورتیکه به انتهای یک نیمه بافر رسیده باشد باید نیمه دیگر را دوباره بار می کردیم.در الگوریتم فوق همان طوریکه مشاهده شد برای هر جلوروی نشانه روی Forward دو عمل مقایسه انجام می شود.می توان این دو را به یک بار تست تبدیل کرد.برای این کار باید در انتهای هر نیمه بافر یک کاراکتر نگهبان قرار دهیم ( Sentinel به عنوان قسمتی از برنامه منبع نخواهد بود )

| | |
|---|---|
| E = M * eof | C * * 2 eof |

به این ترتیب برای کنترل حرکت نشانه روی Forward می توانیم از الگوریتم زیر استفاده کنیم :

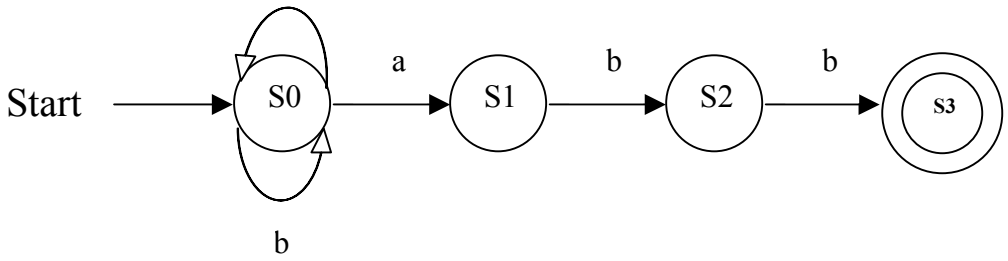
```
Forward := Forward + 1 ;
if  Forward = eof  Then  begin
    if  Forward  is  at  end   of  first-half  Then begin
        reload second-half ;
        Forward := Forward + 1
    end
    else  if  Forward  is  at  end  of  second-half  Then  begin
              reload first-half ;
              move  Forward  to  beginning  of  first-half
    end
    else
    /*  eof  within  a  buffer  signifying  end  of  input  */
    Terminate  Lexical  Analysis
end.
```

---

## ۲-۴ دیاگرام های انتقال

برای پیاده سازی دستی یک اسکنر از ابزاری بنام دیاگرام انتقال ( Transition Diagram ) کمک می گیریم.یک دیاگرام انتقال در واقع یک گراف جهت دار است که هریک از گره های آن معرف یک وضعیت ( State ) است.یکی از وضعیت ها بعنوان وضعیت شروع و یکی ( یا چند تا ) از آنها بعنوان وضعیت ( های ) خاتمه مشخص می گردد.برچسب ( Label ) هایی روی لبه های یک دیاگرام انتقال قرار داده می شود که مشخص می کند در چه صورتی می توان از یک وضعیت به وضعیت دیگر رفت.هر دیاگرام انتقال معرف یک زبان است.با خواندن کاراکترهای یک رشته تطبیق آنها با برچسب های دیاگرام انتقال معرف یک زبان و پیمایش آن دیاگرام می توان مشخص نمود که آیا آن رشته متعلق به زبان موردنظر است یا خیر.به عنوان مثال دیاگرام انتقال زیر و عبارت منظم (a|b)* abb هر دو زبانی را توصیف می کنند که شامل رشته های تشکیل شده از علائم " a " و " b " که به زیررشته " abb " ختم می شوند است.

a

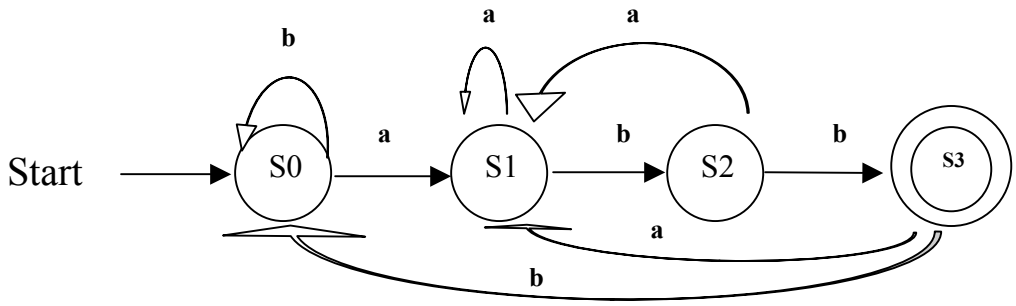



دیاگرام فوق یک دیاگرام غیرقطعی ( NonDeterministic ) است.یک دیاگرام انتقال غیر قطعی دیاگرامی است که یکی از دو خاصیت زیر را داشته باشد :

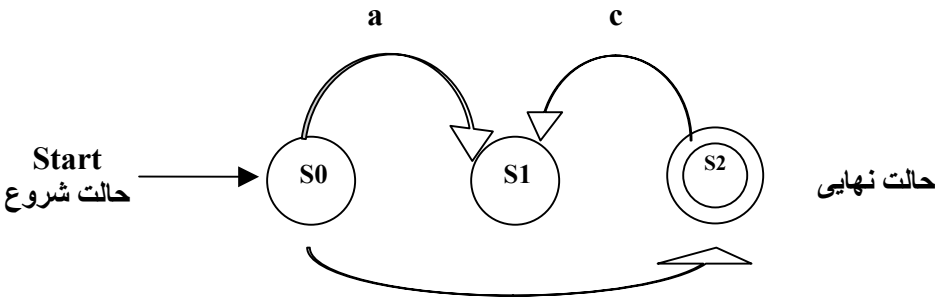
- لبه های خارج شده از برخی از وضعیت های آن برچسب مشابه داشته باشند.
- لبه های دارای برچسب € وجود داشته باشد ( برچسب € به این معنی است که بدون توجه به ورودی می توانیم از یک وضعیت به وضعیتی دیگر برویم )

دیاگرام فوق غیر قطعی است زیرا از وضعیت S0 دو لبه با برچسب مشترک " a " خارج شده است.

درصورتیکه دیاگرامی غیرقطعی نباشد آنرا قطعی ( Deterministic ) گویند.همچنین هر دیاگرام غیرقطعی را می توان به یک دیاگرام قطعی معادل تبدیل کرد.مثلا دیاگرام زیر معادل فوق لیکن قطعی است :



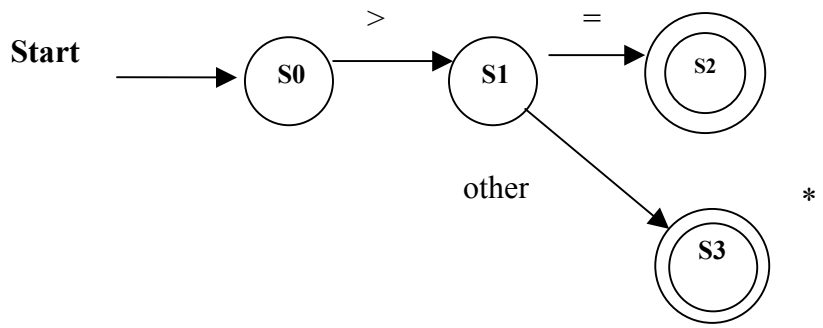

برنامه ای که از یک دیاگرام قطعی استفاده می کند پیاده سازی راحت تری نسبت به برنامه مبتنی بر یک دیاگرام غیرقطعی دارد.برنامه مبتنی بر یک دیاگرام غیرقطعی بایستی دارای قابلیت پی جویی یا Backtracking باشد.از طرف دیگر دیاگرام های انتقال قطعی معمولا تعداد وضعیت بیشتری نسبت به دیاگرام غیرقطعی معادل خود دارند. بنابراین برای پیاده سازی یک اسکنر ابتدا دیاگرام های انتقال معرف الگوی توکن های زبان موردنظر رسم می گردد.این دیاگرام ها برای بدست آوردن اطلاعات در مورد کاراکترهایی که بوسیله نشانه روی Forward در ورودی باید دیده شوند استفاده می گردد.به این ترتیب که همانطور که کاراکترهای ورودی خوانده می شوند از یک وضعیت در دیاگرام به وضعیتی دیگر حرکت می کنیم تا اینکه به یک وضعیت نهایی برسیم.پیمایش دیاگرام از وضعیت شروع (Start ) آغاز می شود.



هنگامیکه در وضعیت فعلی لبه ای که برچسب آن مساوی کاراکتر ورودی است قرار داشته باشیم، از آن حالت بوسیله آن لبه به حالت بعدی می رویم و در غیر اینصورت توکن توسط این دیاگرام قابل تشخیص نخواهد بود.

برچسب " other " در روی لبه یک وضعیت بیانگر هر کاراکتری است که توسط لبه های دیگر آن وضعیت ذکر نشده اند.

**مثال**- دیاگرام تشخیص توکن " =< "بصورت زیر است :

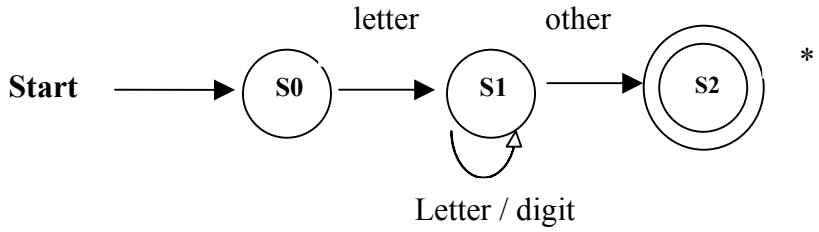



Other: یعنی هر کاراکتر دیگر غیر از =

علامت * یعنی اینکه بایستی آخرین ورودی به بافر بازگردد.

درصورتی که در کلیه دیاگرام ها نتوان یک توکن را تشخیص داد یک خطای واژه ای روی داده است و باید برنامه خطاپرداز فراخوانی شود.

**مثال** – دیاگرام تشخیص شناسه های زبان پاسکال




با توجه به اینکه کلمات کلیدی از یک دنباله کاراکتری تشکیل شده اند لذا می توان از دیاگرام فوق برای تشخیص کلمات کلیدی نیز استفاده نمود.تشخیص کلمات کلیدی

و شناسه ها توسط یک دیاگرام باعث کاهش تعداد وضعیت های دیاگرام انتقال اسکنر می گردد.برای آنکه کلمات کلیدی را از شناسه های همنامشان جدا سازیم یکی از ساده ترین روشها این است که در ابتدا در جدول نشانه ها کلمات کلیدی را وارد کنیم.به این ترتیب با رجوع به جدول نشانه ها می توان دریافت که توکن مورد نظر شناسه است یا کلمه کلیدی.

**جدول نشانه ها**

| if | ..... | K |
|---|---|---|
| | | |

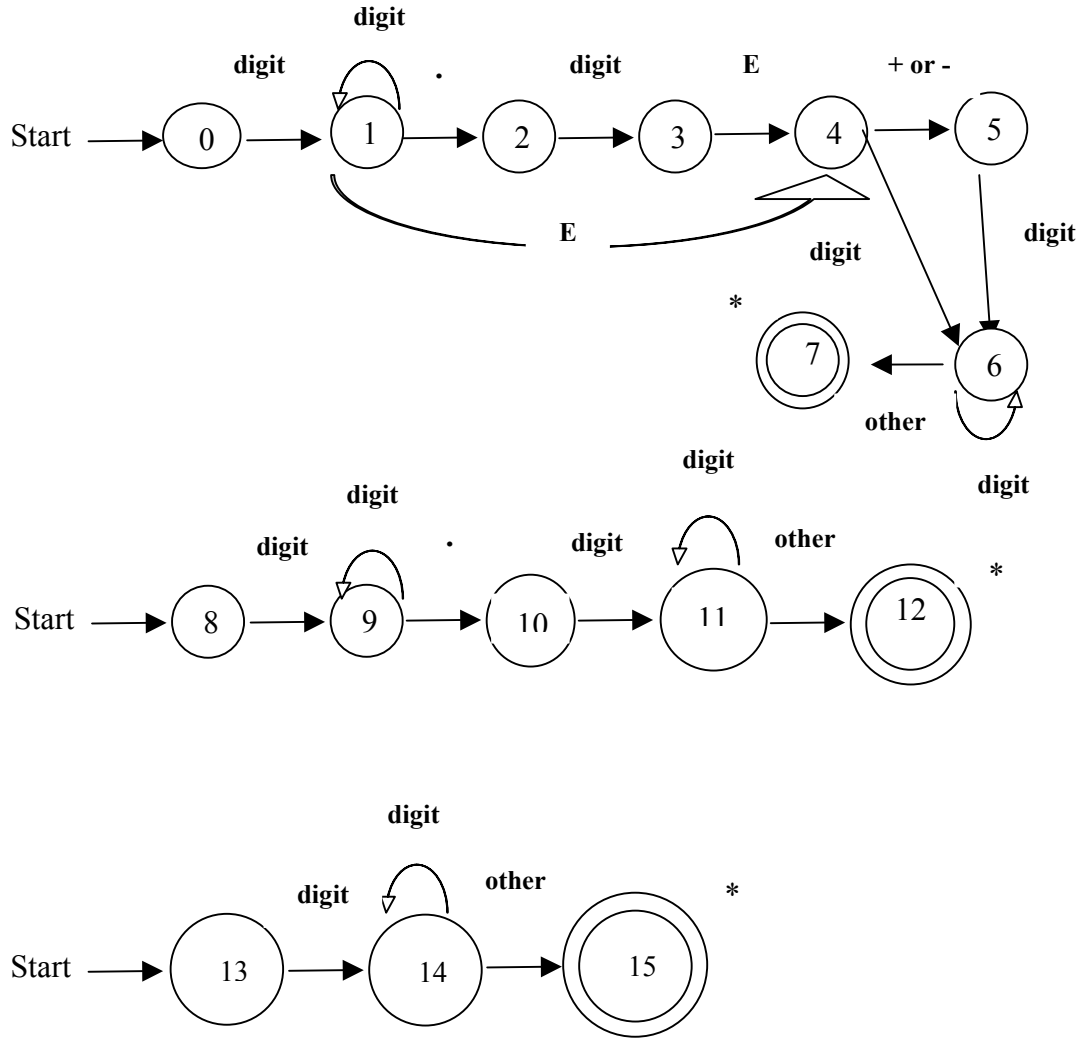
K : بیانگر Keyword بودن می باشد

برای این کار از دو تابع ( )gettoken و ( )install-id استفاده می شود.تابع install-id( ) جدول علائم را جستجو می کند و اگر واژه توکن در جدول نشانه ها بعنوان کلمه کلیدی آمده باشد این تابع عدد صفر را بازمی گرداند.در صورتی که واژه یک متغیر تشخیص داده شود و در جدول هم موجود باشد این تابع آدرس آن متغیر در جدول نشانه ها را بوسیله یک اشاره گر باز می گرداند.اگر چنانچه واژه در جدول موجود نباشد بعنوان ورودی جدید به جدول علائم وارد شده و مطابق حالت قبل آدرس آن بازگردانده خواهد شد.

( )gettoken نیز بطور مشابهی جدول نشانه ها را جستجو کرده و در صورتی که واژه موردنظر یک کلمه کلیدی باشد توکن متناظرش را مستقیما به پارسر می فرستد و در غیر اینصورت توکن " id " را انتقال می دهد.به این ترتیب در صورتی که تعداد کلمات

کلیدی تغییر کند دیاگرام انتقال بدون تغییر باقی خواهد ماند و به راحتی می توان این تغییر را در جدول نشانه ها ایجاد کرد.

در اینجا ذکر چند نکته در مورد نحوه قرار دادن دیاگرام انتقال توکن های مختلف ضروری است.اول آنکه اسکنر باید همواره سعی کند طولانی ترین توکن ممکن را تشخیص دهد.مثلا دیاگرام تشخیص اعداد اعشاری بایستی قبل از دیاگرامی باشد که اعداد صحیح را تشخیص می دهد.همچنین دیاگرام تشخیص توکن هایی که مورد استفاده بیشتری در برنامه ها دارند ( مثلا space , tab , ...) باید قبل از دیاگرام انتقال توکن های کمیاب تر قرار گیرد تا در نهایت تست کمتری برای تشخیص توکن ها انجام شود.مثلا شکل زیر ترتیب قرار گرفتن دیاگرام ها برای تشخیص اعداد را نشان می دهد.شماره وضعیت ها بیانگر ترتیب دیاگرام ها است.



بعد از اینکه دیاگرام تشخیص توکن ها رسم شد به راحتی می توان آنرا با دستور Case پیاده سازی نمود.

هر دیاگرام انتقال غیرقطعی را می توان به یک گرامر مستقل از متن/ منظم تبدیل کرد.برای اینکار باید مراحل زیر را انجام داد :

۱. به ازای هر حالت i یک غیرپایانه $A_i$ در نظر می گیریم.
۲. اگر از حالت i با ورودی $\alpha$ به حالت j می رویم قاعده ای بصورت $A_i \rightarrow \alpha A_j$ تولید می کنیم.
۳. اگر از حالت i با ورودی € به حالت j می رویم قاعده ای به فرم $A_i \rightarrow A_j$ تولید می کنیم.
۴. اگر i یک حالت نهایی باشد قاعده ای به فرم $Ai \rightarrow \varepsilon$ تولید می کنیم.
۵. اگر i حالت شروع باشد غیرپایانه $A_i$ را به عنوان علامت شروع گرامر در نظر می گیریم.

مثلا گرامر مستقل از متن دیاگرام غیرقطعی (a|b)*abb بصورت زیر خواهد بود :

$A_0 \rightarrow aA_1 \mid aA_0 \mid bA_0$
$A_1 \rightarrow bA_2$
$A_2 \rightarrow bA_3$
$A_3 \rightarrow \varepsilon$

اگرچه می توان قواعد لغوی زبانها را توسط گرامرهای مستقل از متن نیز بیان نمود لیکن دلایلی وجود دارد که بهتر است قواعد لغوی توسط عبارات منظم توصیف شوند :

۱. قواعد لغوی زبانها اغلب خیلی ساده هستند و برای توصیف آنها نیازی به نمایش قوی تر گرامر مستقل از متن نیست.
۲. عبارات منظم بطور کلی وسیله ای فشرده تر و گویاتر از گرامرهای مستقل از متن هستند.
۳. اسکنرهای سریعتری را می توان بصورت خودکار از روی عبارات منظم تولید کرد.

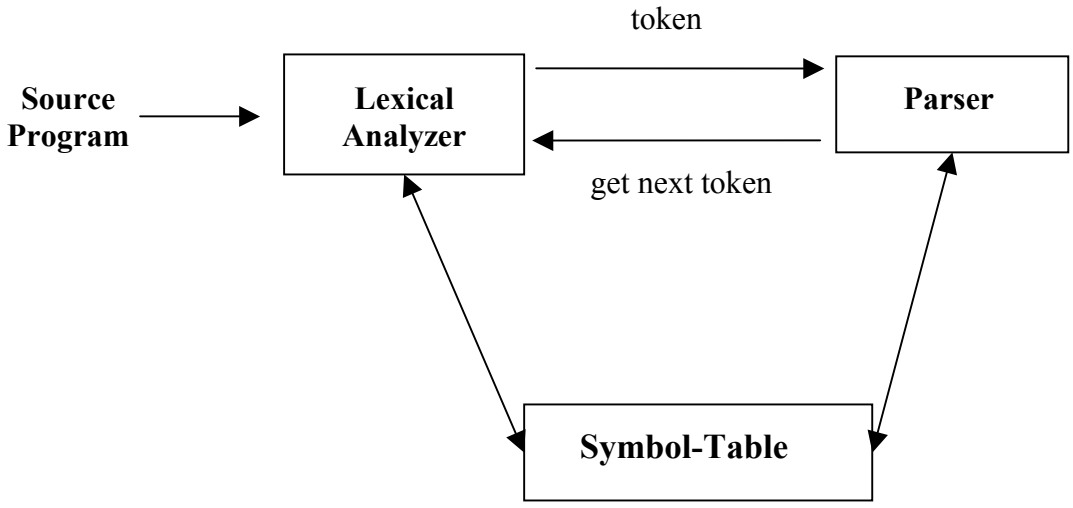
۴. جداکردن ساختار دستوری یک زبان به دو بخش لغوی و غیرلغوی کار پیاده سازی قسمت Front-End کامپایلرها بصورت پیمانه ای را راحت تر می سازد.

---

## ۳ – تحلیل نحوی ( Syntax Analysis )

در مرحله تحلیل نحوی برنامه ورودی از نظر دستوری بررسی می شود.تحلیلگر نحوی یا پارسر برنامه ورودی را که بصورت دنباله ای از توکن ها است از اسکنر گرفته و تعیین می کند که آیا این جمله می تواند بوسیله گرامر زبان موردنظر تولید شود یا خیر ؟

رابطه پارسر و اسکنر بصورت زیر است :




بطور کلی دو نوع **روش تحلیل نحوی** وجود دارد :

۱ – روشهای بالا به پائین ( Top-Down )

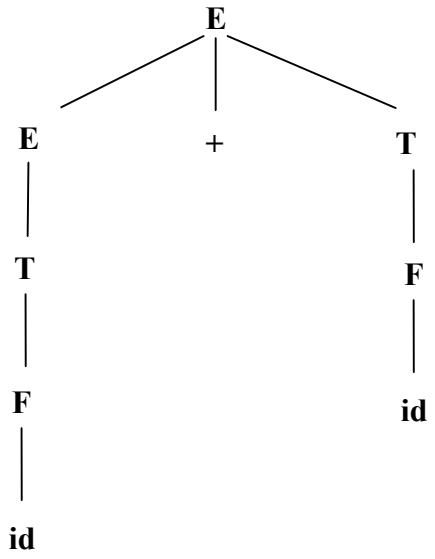
۲ – روشهای پائین به بالا ( Bottom-Up )

روشهای بالا به پائین , درخت تجزیه ( Parse Tree ) را از بالا به پائین می سازند در حالیکه روشهای پائین به بالا برعکس عمل می کنند یعنی درخت تجزیه را از پائین به بالا تولید می کنند.در هر دو روش ورودی از چپ به راست و در هر قدم فقط یک توکن بررسی می شود.

به عنوان مثال گرامر زیر را در نظر بگیرید.

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

درخت تجزیه جمله id + id بصورت زیر خواهد بود که در آن ریشه درخت علامت شروع گرامر است.( توضیح اینکه از این پس اگر علامت شروع گرامری بصورت صریح مشخص نشود غیرپایانه سمت چپ اولین قاعده گرامر بعنوان علامت شروع آن در نظر گرفته می شود.)




مهمترین **روشهای تجزیه بالا به پائین** عبارتند از :

۱ – روش پائینگرد ( Recursive Descent )

۲ – روش LL(1) که حالت خاصی از روش کلی LL(K) است.

در روش LL(1) منظور از ' L ' اول این است که ورودی از سمت چپ به راست خوانده شده و بررسی می گردد و ' L ' دوم یعنی پارسر از بسط چپ (Left-Most-Derivation) استفاده می کند و ' 1 ' نیز بیانگر این است که در هر قدم از تجزیه فقط یک توکن بررسی خواهد شد.

مهمترین **روشهای تجزیه پائین به بالا** عبارتند از :

۱ – روش تقدم عملگر ( Operator Precedence )

۲ – روش تقدم ساده ( Simple Precedence )

۳ – روش LR(1) که خود دارای سه فرم مختلف با نامهای SLR(1) , LALR(1) , CLR(1) است.

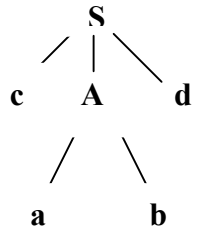
تمامی روشهای فوق به نوعی از یک روش کلی به نام انتقال – کاهش ( Shift-Reduce ) پیروی می کنند.اگرچه روشهای تجزیه بالا به پائین برای پیاده سازی دستی مناسب ترند اما ابزارهایی وجود دارد که با کمک آنها می توان بطور خودکار پارسرهای قوی پائین به بالا تولید نمود.در LR(1) منظور از ' L ' یعنی پارسر ورودی را از چپ به راست می خواند و منظور از ' R ' این است که پارسر از عکس بسط راست ( Right-Most-Derivation ) استفاده می کند.

## ۳-۱ تجزیه بالا به پائین ( Top-Down Parsing )

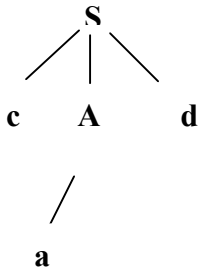
در حالت کلی یک پارسر بالا به پائین بایستی بتواند در صورت لزوم عمل پیجوئی ( Back Tracking ) انجام دهد.گرامر زیر را در نظر بگیرید.

$S \rightarrow cAd$
$A \rightarrow ab \mid a$

برای تجزیه رشته ورودی cad پارسر بصورت زیر عمل می کند :




چون d با b برابر نیست پارسر نیاز دارد که یک مرحله به عقب بازگردد و قاعده دیگر را مورد بررسی قرار دهد.




در اینجا پارسر نتیجه می گیرد که رشته ورودی رشته قابل قبول در گرامر می باشد.
اگرچه همانگونه که توضیح داده شد عمل نحوی در حالت کلی می تواند به روش آزمایش و خطا اجرا گردد لیکن بهتر است پارسرها بگونه ای طراحی و پیاده سازی شوند که نیازی به پیجوئی نداشته باشند.به پارسرهایی که عمل عقبگرد انجام نمی دهند پارسر پیشگو ( Predictive ) می گویند.از طرفی پارسرها معمولا به صورت حریصانه ، ( greedy ) عمل می کنند یعنی با دریافت هر توکن درخت تجزیه را تا حد امکان گسترش می دهند و تنها هنگامی که دیگر امکان گسترش درخت پارس وجود نداشته باشد توکن بعدی را درخواست می کنند.

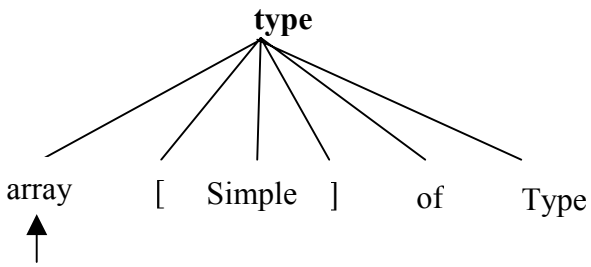
به عنوان مثال گرامر زیر را که معرف گونه ( Type ) در زبان پاسکال است در نظر بگیرید.

Type → Simple
| ↑ id
| array [ Simple ] of Type
Simple → integer
| char
| num dot dot num

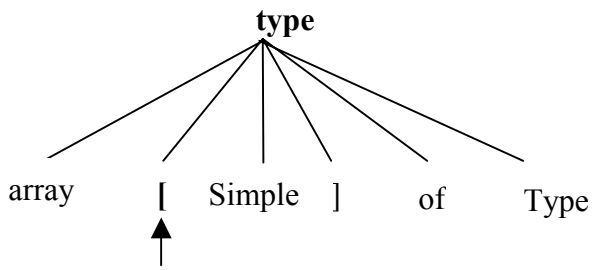
در روشهای بالا به پائین تولید درخت تجزیه از ریشه درخت که همان غیرپایانه شروع گرامر است آغاز و در ادامه کار مراحل زیر بطور مکرر انجام می گیرد و درخت تجزیه بصورت بالا به پائین و از چپ به راست ساخته می شود :

۱. در گره n با غیرپایانه A یکی از قواعد گرامر که A در سمت چپ آن قرار دارد را انتخاب کرده و سمت راست این قاعده را بعنوان فرزندان گره n در درخت پارس قرار می دهد.
۲. گره بعدی را که از آنجا یک زیردرخت دیگر باید ایجاد شود را پیدا می کند.

مراحل فوق در حین آنکه رشته ورودی از چپ به راست خوانده می شود انجام می گیرد.توکنی که پارسر در حال بررسی آن است را توکن جاری می گویند.فرض کنید رشته ورودی بصورت " array [ num dot dot num ] of integer " باشد.مراحل تشکیل درخت پارس بصورت زیر خواهد بود :

| ورودی | درخت تجزیه |
|---|---|
| array [ num dot dot num ] of integer ↑ | Type |
| array [ num dot dot num ] of integer ↑ |  |

array [ num dot dot num ] of integer




معمولا انتخاب یک قاعده برای یک غیرپایانه بصورت سعی و خطا ( Trial-and-Error ) انجام می شود.بدین معنی که در صورتی که قاعده انتخابی اول مناسب ادامه عمل تجزیه نباشد با عمل عقبگرد یا پیجوئی قاعده دیگری انتخاب می گردد.

## ۳-۲ تجزیه پائینگرد ( Recursive Descent Parsing )

یکی از انواع پارسرها که بصورت پیشگویانه عمل می کند پارسر پائینگرد ( Recursive Descent ) است.این پارسر بصورت بالا به پائین عمل می کند و در آن یک مجموعه رویه ها بطور بازگشتی رشته ورودی را مورد پردازش قرار می دهند.این رویه ها که برای پردازش رشته ورودی فراخوانی می شوند یک درخت پارس برای ورودی ایجاد می کنند.یک پارسر پائینگرد به ازای هر غیرپایانه یک رویه دارد که دو کار انجام می دهد :

۱- تصمیم می گیرد که از کدام قاعده گرامر استفاده شود.

۲- از قاعده انتخاب شده استفاده می کند.

علاوه بر رویه هایی که به ازای هر غیرپایانه وجود دارد یک پارسر پائینگرد از رویه دیگری بنام match برای تطبیق توکن های ورودی و پایانه های درخت تجزیه در حال ساخت استفاده می کند.بعنوان مثال پارسر پائینگرد برای گرامر تعریف Type در زبان پاسکال دو رویه برای غیرپایانه های Type و Simple خواهد داشت.رویه match نیز بصورت زیر است :

```
Procedure    match ( t : token );
  begin
```

```
  if  Lookahead = t  Then
        Lookahead := nexttoken
  else  Error
end
```

رویه فوق برای راحتی کار رویه های Type و Simple استفاده می شود و متغیر Lookahead را تغییر می دهد.

```
Procedure  Type ;
 begin
   if  Lookahead is in {integer,char,num}  Then  Simple
   else if  Lookahead = array  then  begin
        match ( array ) ;
        match ( ' [ ' ) ;
        Simple ;
        match ( ' ] ' ) ;
        match (of) ;
        Type
   end
   else  if  Lookahead = ' ↑ '  Then Begin
       match ( ' ↑ ' ) ;
       match( ' id ' )
   end
   else  Error
end ;

Procedure  Simple ;
 begin
   if  Lookahead = integer  Then   match ( integer )
   else  if  Lookahead = char  Then  match ( char )
   else  if  Lookahead = num  Then  begin
      match ( num ) ; match ( dot dot ) ; match ( num )
   end
   else  Error
end ;
```

پارسر پائینگرد با استفاده از حاصل اعمال تابعی بنام " First " بر روی رشته سمت راست قواعد تعیین می کند که از کدامیک از قواعد گرامر باید استفاده شود .تابع

First روی رشته ای از پایانه ها و غیرپایانه ها عمل می کند.حاصل تابع First(α) مجموعه ای از پایانه ها است که در سمت چپ ترین قسمت از رشته های تولیدشده از رشته α قرار می گیرند.تعریف رسمی تر این تابع بصورت زیر است :

$$First(\alpha) = \{a \mid \alpha \overset{*}{\Rightarrow} a\beta \, , \, a \in T \, , \, \beta \in (N \cup T)^* \}$$

الگوریتم بدست آوردن تابع First یک رشته در بخش های بعدی آورده شده است.بعنوان مثال گرامر زیر را در نظر بگیرید.

$S \rightarrow cAd$
$A \rightarrow ab \mid a$

$First(cAd) = \{c\}$
$First(S) = \{c\}$
$First(A) = \{a\}$

برای گرامر Type خواهیم داشت :

$First(Simple) = \{integer,char,num\}$
$First(\uparrow id) = \{\uparrow\}$
$First(array[Simple] \text{ of } Type) = \{array\}$
$First(Type) = \{\uparrow,array,integer,char,num\}$

به این ترتیب در گرامری که دو قاعده بصورت $A \rightarrow \alpha$ و $A \rightarrow \beta$ داشته باشد پارسر پائینگرد با استفاده از First سمت راست این قواعد , قاعده مناسب را تعیین می کند بدون اینکه نیاز به عمل عقبگرد داشته باشد البته مشروط بر اینکه در چنین گرامرهایی شرط زیر برقرار باشد :

$First(\alpha) \cap First(\beta) = \phi$

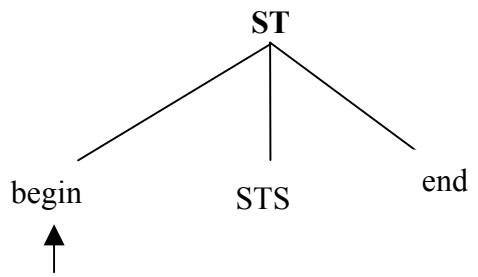
### ۳-۳ استفاده از قاعده اپسیلون ( $A \rightarrow \varepsilon$ )

هرگاه در گرامری قاعده اپسیلون وجود داشته باشد پارسر پائینگرد از آن بعنوان قاعده پیش فرض ( default ) استفاده می کتد.بدین معنی که اگر هیچ قاعده دیگری در گرامر مناسب تشخیص داده نشد پارسر از قاعده اپسیلون برای ادامه عمل تجزیه استفاده می کند.

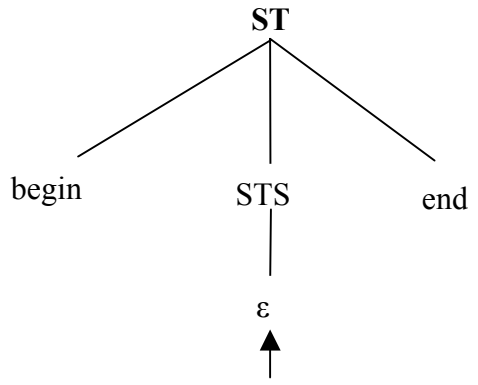
مثال :

$ST \rightarrow$ begin STS end
$STS \rightarrow STL \mid \varepsilon$
$STL \rightarrow a$

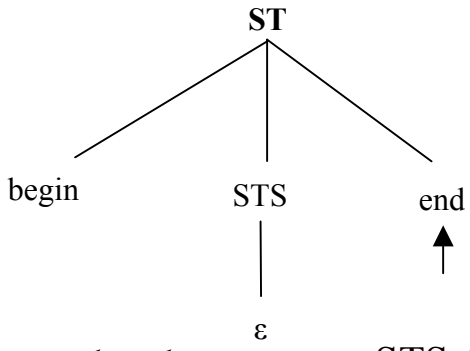
فرض کنید جمله ورودی begin end باشد.درخت تجزیه پس از خواندن توکن begin و بسط علامت شروع گرامر بصورت زیر خواهد بود :




پس از تطبیق توکن begin , توکن بعدی یعنی end از اسکنر دریافت می گرددو نوبت به بسط غیرپایانه STS می رسد.چون end متعلق به مجموعه First(STS) نیست لذا از قاعده $STS \rightarrow \varepsilon$ بعنوان قاعده پیش فرض استفاده می شود.درخت تجزیه بصورت زیر درخواهد آمد :




سپس توکن end نیز با پایانه end در درخت تجزیه تطبیق می شود و عمل تجزیه خاتمه می یابد.




باید توجه داشت که انتخاب قاعده ε برای بسط STS تنها در صورتی که توکن جاری در آن لحظه end باشد انتخاب درستی خواهد بود.

---

## ۴-۳ مشکل چپ گردی ( Left Recursion )

گرامری را چپ گرد گویند اگر غیرپایانه سمت چپ یک قاعده به عنوان اولین علامت سمت راست آن قاعده ظاهر شده باشدوبعبارت دیگر غیرپایانه ای در گرامر وجود داشته باشد که قاعده ای بصورت $A \rightarrow A\alpha$ داشته باشد.

روشهای پارس بالا به پائین را نمی توان برای گرامری که چپ گردی داشته باشد بکار برد.از اینرو باید چپ گردی گرامر را حذف کنیم یعنی گرامر را به گرامر معادلی تبدیل کنیم که در آن چپگردی وجود نداشته باشد.برای مثال گرامر چپ گرد $A \rightarrow A\alpha \mid \beta$ را می توان بفرم زیر که چپ گردی ندارد تبدیل نمود :

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

هر دو گرامر فوق رشته هایی بفرم $\alpha\beta^*$ را توصیف می کنند.

روش کلی حذف چپ گردی بصورت زیر است ( توجه کنید که اهمیتی ندارد که چه تعداد از قواعد چپ گرد باشند) :

بطور کلی اگر داشته باشیم :

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

در قواعد فوق فرض بر این است که $\beta_i$ ها نباید با A شروع شوند و هیچکدام از $\alpha_i$ ها نبایدε باشند.در اینصورت می توان قواعد زیر را بجای قواعد چپ گردی فوق بکار برد.

$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$
$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$

اینگونه چپ گردی ها را " چپ گردی آشکار " ( Immediate Left recursion ) گویند.
ممکن است چپ گردی در بیش از یک قدم ظاهر شود که به آن چپ گردی ضمنی گویند.به عنوان مثال گرامر زیر دارای چپ گردی ضمنی است:

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd$

در غیرپایانه S چپ گردی ضمنی داریم زیرا :

$S \Rightarrow Aa \Rightarrow Sda$

## ۳-۵ حذف چپ گردی ضمنی ( Implicit Left Recursion )

ورودی الگوریتم گرامر G با این شرط که قاعده اپسیلون نداشته باشد و هیچ دوری نیز در گرامر موجود نباشد یعنی بسطی بصورت $A \overset{+}{\Rightarrow} A$ در گرامر نباشد.خروجی الگوریتم گرامری معادل گرامر G اما فاقد چپ گردی است.ابتدا غیرپایانه های گرامر را به ترتیب دلخواه $A_1, A_2, \ldots, A_n$ مرتب می کنیم.سپس اعمال زیر را بصورت مشخص شده در حلقه های تکرار اجرا می کنیم :

For i := 1 to n do begin
For j := 1 to i-1 do begin
بجای هر قاعده به شکل $A_i \rightarrow A_j\gamma$ قواعد $A_i \rightarrow \delta_1\gamma \mid \delta_2\gamma \mid \ldots \mid \delta_k\gamma$ را قرار دهید که در آن $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$ قواعد فعلی $A_j$ هستند
end
حال چپ گردی آشکار قواعد $A_i$ را حذف کنید
end

حال بعنوان نمونه الگوریتم فوق را برای گرامر زیر بکار می بریم :

$S \rightarrow Aa \mid b$
$A \rightarrow Ac \mid Sd$

ابتدا غیرپایانه های گرامر را به ترتیب A و S ( از چپ به راست ) مرتب می کنیم.از روی قاعده $A \rightarrow Sd$ خواهیم داشت :

$A \rightarrow Aad \mid bd$

به این ترتیب قواعد A بصورت زیر درخواهند آمد که دارای چپ گردی آشکار هستند :

$A \rightarrow Aad$
$A \rightarrow Ac$
$A \rightarrow bd$

با حذف چپ گردی آشکار قواعد فوق گرامر زیر بدست می آید :

$S \rightarrow Aa \mid b$
$A \rightarrow bdA'$
$A' \rightarrow cA' \mid adA' \mid \varepsilon$

---

## ۳-۶ فاکتورگیری از چپ ( Left factoring )

با استفاده از فاکتورگیری از چپ می توان گرامرهایی که در آنها برای غیرپایانه A دو قاعده بصورت $A \rightarrow \alpha\beta_1$ و $A \rightarrow \alpha\beta_2$ وجود دارد را طوری تغییر داد که بتوان پارس بالا به پائین را برای این گرامرها استفاده کرد.

مشکل این قبیل گرامرها در این است که روشن نیست که از کدامیک از این قواعد باید برای بسط غیرپایانه A استفاده کرد.بعنوان مثال :

Stmt → **if** exp **then** Stmt **else** Stmt
| **if** Exp **then** Stmt

با دیدن if در ورودی بلافاصله نمی توان گفت که از کدام قاعده برای بسط غیرپایانه Stmt می توان استفاده کرد.در حالت کلی اگر $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ دو قاعده موجود در گرامری باشند و ورودی با رشته $\alpha$ شروع شده باشد نمی توان گفت که A را باید بصورت $\alpha\beta_1$ بسط داد و یا بصورت $\alpha\beta_2$ .برای رفع این مشکل از $\alpha$ فاکتور می گیریم.گرامر را بصورت زیر تبدیل می کنیم :

$A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 \mid \beta_2$

الگوریتم فاکتورگیری از چپ در حالت کلی بصورت زیر است :

قواعد زیر را در نظر بگیرید :

$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_m \mid \delta_1 \mid \delta_2 \mid \ldots \mid \delta_n$

که در آن $\delta_i$ بیانگر قواعدی است که سمت راست هیچ کدام با $\alpha$ آغاز نشده است.با فاکتورگیری از چپ قواعد زیر حاصل می شوند :

$A \rightarrow \alpha A' \mid \delta_1 \mid \delta_2 \mid \ldots \mid \delta_n$
$A' \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_m$

که در آن $A'$ یک غیرپایانه جدید است.

مثال: قواعد گرامری بصورت زیر می باشد :

$S \rightarrow i\ E\ t\ S$
$S \rightarrow i\ E\ t\ S\ e\ S \mid a$
$E \rightarrow b$

این قواعد بعد از انجام عمل فاکتورگیری از چپ بصورت زیر تبدیل می شوند :

$S \rightarrow i\ E\ t\ S\ S' \mid a$
$S' \rightarrow e\ S \mid \varepsilon$
$E \rightarrow b$

---

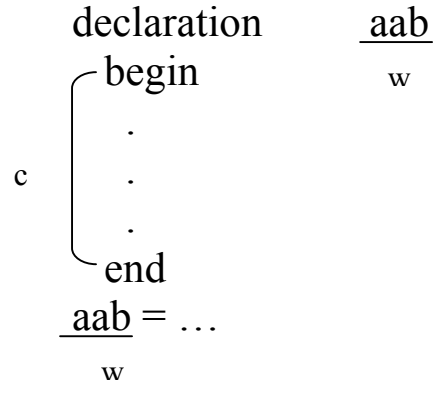
## ۳-۷ زبانهای غیر مستقل از متن ( Non-Context Free Languages )

زبانهایی هستند که نمی توان آنها را توسط یک گرامر مستقل از متن توصیف کرد.همچنین محدودیت هایی در زبانهای برنامه نویسی وجود دارد که نمی توان آنها را توسط گرامرهای مستقل از متن اعمال کرد.به مثالهای زیر توجه کنید .

**مثال ۱** – زبان زیر را در نظر بگیرید :

$L_1 = \{ wcw \mid w \text{ is in } (a|b)^* \}$

این زبان رشته هایی بصورت aabcaab تولید می کند.این رشته ها را می توان مشابه این محدودیت در زبانهای برنامه سازی در نظر گرفت که تعریف متغیرها بایستی قبل از استفاده از آنها قرار گیرد.به این ترتیب که w اول در wcw بیانگر تعریف متغیر بوده و w دوم نشان دهنده استفاده از متغیر می باشد.




**مثال ۲** – زبان $L_2 = \{ a^n b^{n1} c^n d^{n1} \mid n1 >= 1 \text{ and } n >= 1 \}$ نیز مستقل از متن نیست.این زبان رشته هایی بصورت $a^+b^+c^+d^+$ تولید می کند که در آنها تعداد تکرار a با c برابر است و تعداد تکرارهای b با d برابر است.این زبان مشابه این محدودیت در زبانهای برنامه سازی است که تعداد پارامترهای رسمی در تعریف یک رویه بایستی با تعداد آرگومانها در فراخوانی همان رویه برابر باشد.در اینجا می توان $a^n$ و $b^{n1}$ را بیانگر پارامترهای رسمی در تعریف دو رویه که به ترتیب دارای n و n1 پارامتر ورودی هستند در نظر گرفت.به همین ترتیب $d^{n1}$ و $c^n$ را می توان بعنوان تعداد آرگومانها در فراخوانی این رویه ها در نظر گرفت :

```
dcl     proc1( a,a,a )
dcl     proc2( b,b )
  .
  .
  .
Call    proc1( c,c,c )
Call    proc2( d,d )
```

**مثال ۳** – زبان $L_3 = \{ a^n b^n c^n \mid n >= 1 \}$ نیز مستقل از متن نیست.این زبان رشته هایی بفرم $a^+ b^+ c^+$ تولید می کند که در آنها تعداد a , b , c برابر است.این زبان مشابه مساله ایجاد کلماتی که در زیر آنها خط کشیده شده باشد (Underlined Word ) است. اینگونه کلمات به این صورت چاپ می شوند که ابتدا کاراکترهای یک کلمه چاپ شده و بدنبال آن به تعداد کاراکترهای آن کلمه به عقب برگشته ( با کمک کاراکتر BackSpace ) و سپس به همان تعداد کاراکتر " _ " چاپ می شود.مثلا کلمه Read .در زبان $L_3$ اگر a بیانگر حروف , b بیانگر کاراکتر BackSpace و c نیز بیانگر کاراکتر " _ " باشد آنگاه این زبان کلمات Underlined را تولید می کند.

گرامرهایی وجود دارند که با وجود شباهت بسیار به گرامرهای $L_1 , L_2 , L_3$ در مثالهای فوق , مستقل از متن هستند.بعنوان مثال زبان $L_1'$ که بصورت زیر تعریف شده است مستقل از متن است :

$L_1' = \{ wcw^R \mid w \text{ is in } (a \mid b)^* \}$

این زبان را می توان بوسیله گرامر زیر تولید کرد :

$S \rightarrow aSa \mid bSb \mid c$

زبان $L_2' = \{ a^n b^n c^{n1} d^{n1} \mid n >= 1 \text{ and } n1 >= 1 \}$ نیز مستقل از متن است و با گرامر زیر توصیف می شود :

$S \rightarrow AB$
$A \rightarrow aAb \mid ab$
$B \rightarrow cBd \mid cd$

و سرانجام زبان $L_3' = \{ a^n b^n \mid n >= 1 \}$ نیز مستقل از متن و گرامر تولید آن بصورت زیر است :

$S \rightarrow aSb \mid ab$

## ۳- ۸ استفاده از دیاگرام های انتقال برای پیاده سازی پارسرهای پیشگو

برای طراحی پارسرهای پیشگو نیز می توانیم از دیاگرام های انتقال استفاده کنیم.به این ترتیب که ابتدا ساختار نحوی زبان را با استفاده از گرامر ایجاد می کنیم.سپس از روی گرامر دیاگرام های انتقال را رسم می کنیم.در پارسرها برای هر غیرپایانه یک دیاگرام انتقال داریم .برچسب های لبه های این دیاگرام ها می توانند پایانه و یا غیرپایانه باشند.انتقال از طریق لبه های با برچسب پایانه به منزله اینست که در ورودی نیز نشانه ای معادل با برچسب مشاهده شده است.در انتقال از طریق لبه ای با برچسب غیرپایانه مانند A به منزله فراخوانی رویه ای برای A می باشد.برای ایجاد دیاگرام یک گرامر ابتدا باید چپ گردی گرامر را در صورت وجود برطرف کرد.همچنین هرکجا که لازم باشد بایستی عمل فاکتورگیری از چپ انجام شود.سپس برای رسم دیاگرام مراحل زیر را برای هر غیرپایانه انجام می دهیم :

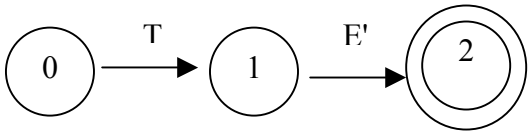
۱- یک حالت شروع و یک حالت نهایی ایجاد کنید.

۲- برای هر قاعده به فرم $A \rightarrow X_1X_2\ldots X_m$ یک مسیر از حالت شروع به حالت نهایی رسم می کنیم و به لبه ها برچسب های $X_1X_2\ldots X_m$ می دهیم.

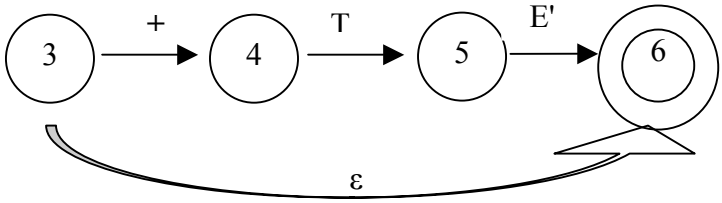
حال بعنوان نمونه دیاگرام های انتقال گرامر زیر را رسم می کنیم.

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid id$

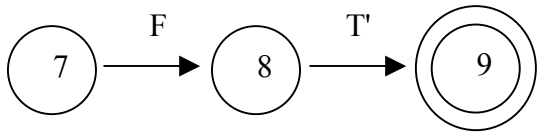
E :



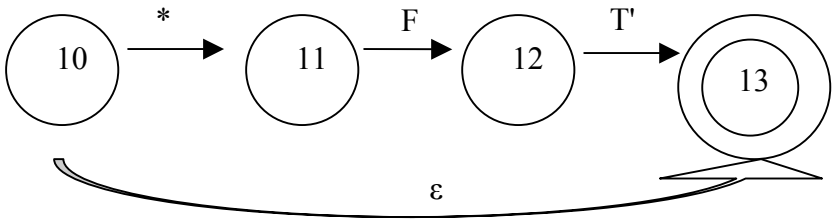

E' :



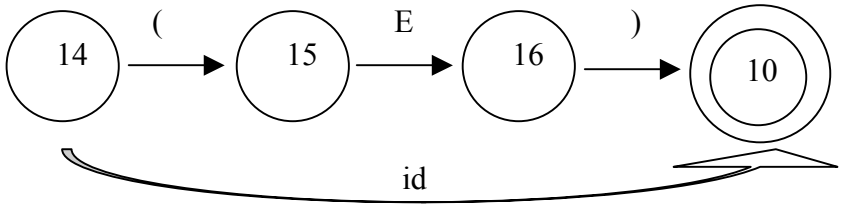
T :




T' :

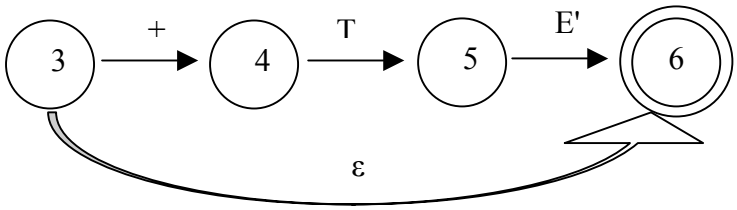



F :

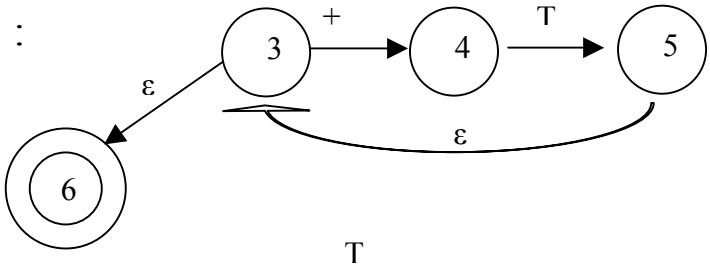



گاهی اوقات دیاگرام های انتقال را می توان ساده تر کرد.به عنوان مثال دیاگرام های فوق را در نظر بگیرید.

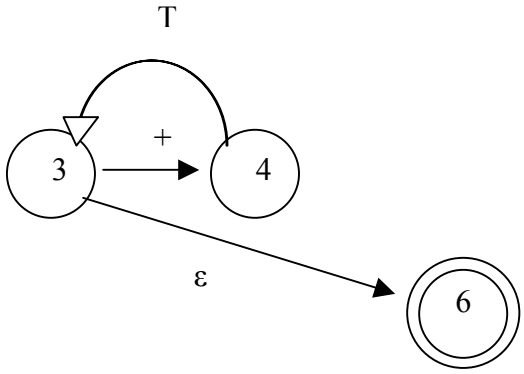
E' :




E' :

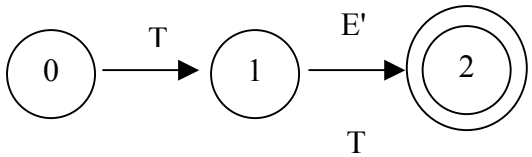



E' :

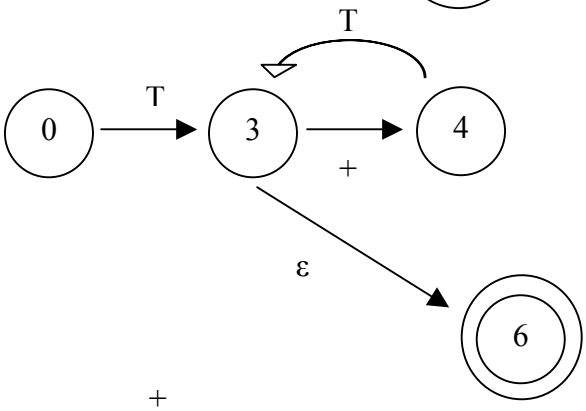


و با جاگذاری این دیاگرام ساده شده در دیاگرام E خواهیم داشت :

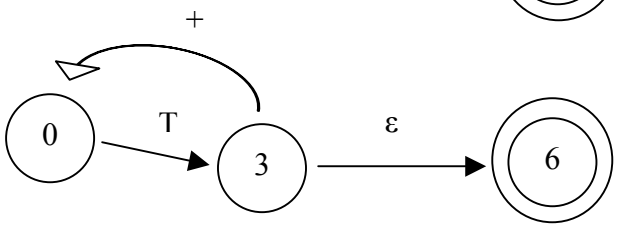
E :




E :

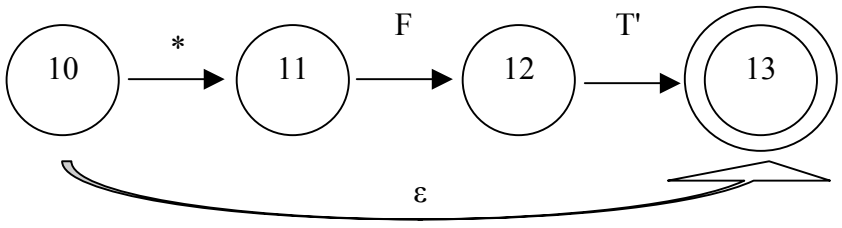



E :

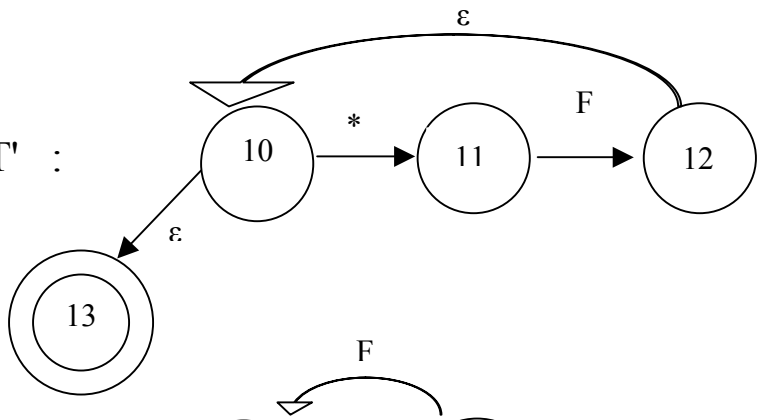



به همین ترتیب برای دیاگرام های T' و T خواهیم داشت :

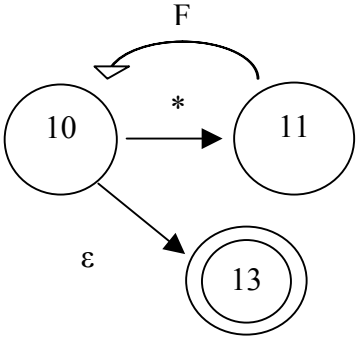
T' :



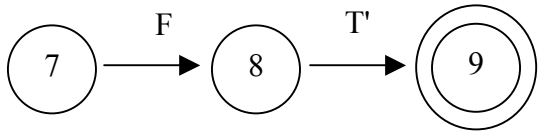

T' :



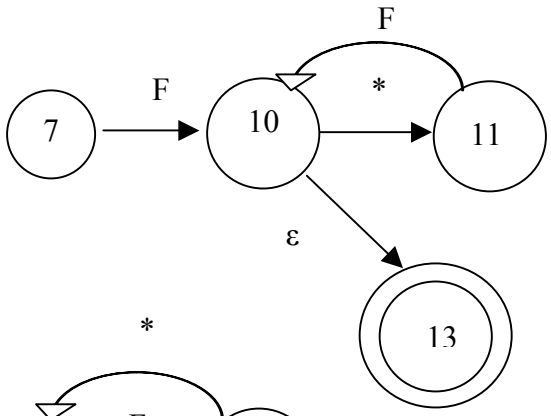

T' :



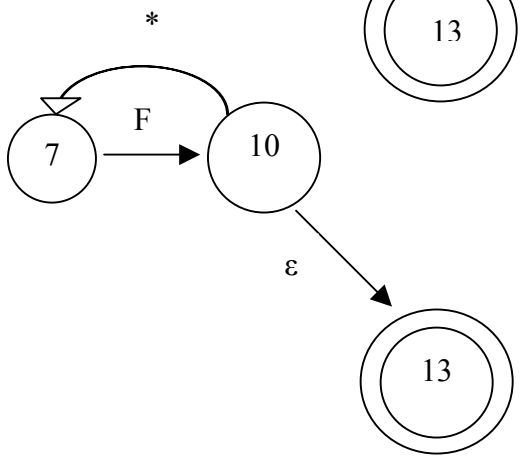
T :




T :



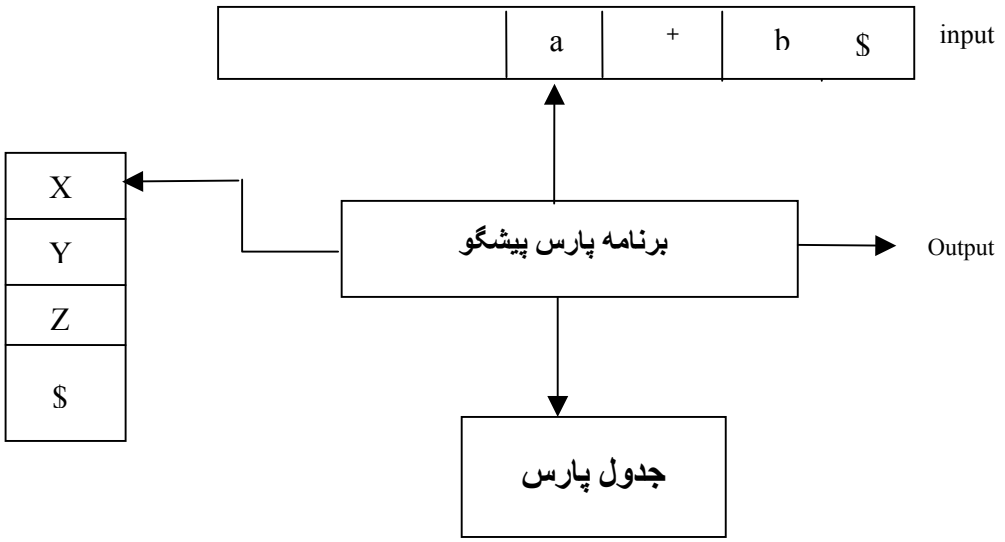

T :




---

## ۳- ۹ تجزیه پیشگویانه غیربازگشتی ( Non-Recursion Predictive Parsing )

در این روش تجزیه از یک انباره ( Parse Stack ) و یک جدول به نام جدول تجزیه ( Parsing Table ) استفاده می گردد.بافر ورودی شامل رشته ای است که باید تجزیه شود.در انتهای رشته ورودی علامتی مثلا $ قرار می گیرد.ساختار کلی این نوع پارسر بفرم زیر است :




در ابتدای پارس علامت $ وارد انباره می شود و روی آن علامت شروع گرامر قرار می گیرد.در انتهای پارس هم در انباره بافر هم در ورودی تنها علامت $ باقی می ماند.جدول پارس یک آرایه دوبعدی بصورت M [ A , a ] است که در آن A یک غیرپایانه و a یک پایانه و یا علامت $ است.فرم کلی جدول بصورت زیر است :

| غیرپایانه | پایانه ها و علامت $ | | | |
|---|---|---|---|---|
| | | | ..... | |
| | | | | |
| | | | | |

برخی از خانه های جدول حاوی شماره یک قاعده از گرامر و برخی از آنها خالی است.

در هر قدم از پارس برنامه پارس علامت X بالای انباره و Token جاری a در ورودی را مورد بررسی قرار می دهد و بصورت زیر تصمیم می گیرد :

۱- اگر $X = a = \$$ باشد پارسر پایان موفقیت آمیز پارس را گزارش می دهد.

۲- اگر $X = a \neq \$$ باشد پارسر X را از بالای انباره حذف و توکن بعدی را دریافت می کند.اگر X پایانه باشد و با a مطابقت نکند یک خطای نحوی رخ داده است.

۳- اگر X یک غیرپایانه باشد برنامه به خانه M [ X , a ] مراجعه می کند که در آن یا شماره قاعده ای به فرم $A \rightarrow ABC$ قرار دارد و یا خالی است.در صورت اول , پارسر X را از بالای انباره حذف و بجای آن ABC را وارد انباره می کند به نحوی که A بالای انباره قرار گیرد.در صورتیکه خانه M [ X , a ] خالی باشد یک خطای نحوی رخ داده است.

---

## ۳- ۱۰ توابع First و Follow

برای پر کردن جدول پارس از توابعی به نامهای First و Follow استفاده می شود.همانگونه که قبلا توضیح داده شد First(α) مجموعه پایانه هایی است که بعنوان سمت چپ ترین علامت رشته های بدست آمده از α قرار می گیرند.در صورتیکه $\alpha \stackrel{*}{\Rightarrow} \varepsilon$ در اینصورت ε نیز جزو First(α) خواهد بود.در ادامه الگوریتم محاسبه First یک علامت مثل X توضیح داده شده است.اگرچه الگوریتم در مورد یک علامت بیان می گردد لیکن با کمک آن می توان مجموعه First را برای رشته ها نیز محاسبه نمود.

برای پیدا کردن **First(X)** بصورت زیر عمل می شود( X می تواند یک پایانه یا یک غیرپایانه باشد ) :

۱- اگر X یک پایانه باشد در آنصورت $First(X) = \{X\}$

۲- اگر قاعده ای بصورت $X \rightarrow \varepsilon$ در گرامر باشد ε را به First(X) اضافه می کنیم.

۳- اگر قاعده ای به فرم $X \rightarrow y_1y_2 \ldots y_k$ در گرامر موجود باشد ابتدا $First(y_1) - \varepsilon$ ( یعنی مجموعه $First(y_1)$ منهای $\varepsilon$ ) را به $First(X)$ اضافه می کنیم.درصورتیکه $y_1 \Rightarrow^* \varepsilon$ , $First(y_2) - \varepsilon$ را نیز به $First(X)$ اضافه می کنیم.( در غیر اینصورت کار یافتن $First(X)$ از طریق قاعده فوق خاتمه می یابد.)در صورتیکه $y_2 \Rightarrow^* \varepsilon$ , $First(y_3) - \varepsilon$ را هم به $First(X)$ می افزائیم.این کار آنقدر ادامه می یابد تا آنکه $\varepsilon$ عضو مجموعه $First(y_i)$ , به ازای $1 <= i <= k$ نباشد.در صورتیکه $y_{k-1} \Rightarrow^* \varepsilon$ باید $First(y_k)$ را نیز به $First(X)$ اضافه بکنیم.

به عنوان مثال گرامر زیر را در نظر بگیرید.

$A \rightarrow aB$
$B \rightarrow Cb \mid d$
$C \rightarrow \varepsilon \mid c$

$First(A) = \{a\}$ , $First(B) = \{ c,b,d \}$ , $First(C) = \{\varepsilon , c \}$

برای بدست آوردن **Follow(A)** ( A یک غیرپایانه است ) اعمال زیر را آنقدر ادامه می دهیم تا اینکه دیگر چیزی به مجموعه $Follow(A)$ اضافه نشود :

۱. $ را در $Follow(S)$ قرار می دهیم.( S علامت شروع گرامر است )
۲. اگر قاعده ای بصورت $X \rightarrow \alpha A \beta$ داشته باشیم هرچه در $First(\beta)$ قرار دارد ( بغیر از $\varepsilon$ ) را به مجموعه $Follow(A)$ اضافه می کنیم.
۳. اگر قاعده ای بفرم $X \rightarrow \alpha A$ داشته باشیم و یا آنکه قاعده ای بفرم $X \rightarrow \alpha A \beta$ و $\beta \Rightarrow^* \varepsilon$ , هرچه در $Follow(X)$ قرار دارد را به مجموعه $Follow(A)$ اضافه می کنیم.

به عنوان مثال گرامر زیر را در نظر بگیرید :

1 $E \rightarrow T\ E'$

2 – 3 $E' \rightarrow + T E' \mid \varepsilon$

4 $T \rightarrow F T'$

5 – 6 $T' \rightarrow * F T' \mid \varepsilon$

7 – 8 $F \rightarrow ( E ) \mid id$

| مجموعه های Follow | مجموعه های First |
|---|---|
| Follow(E) = { ) , $ } | First(E) = { ( , id } |
| Follow(T) = { + , ) , $ } | First(T) = { ( , id } |
| Follow(F) = { * , + , $ , ) } | First(F) = { ( , id } |
| Follow(E') = { ) , $ } | First(E') = { ε , + } |
| Follow(T') = { + , ) , $ } | First(T') = { ε , * } |

## نحوه تشکیل جدول پارس برای پارسر های پیشگو :

۱- برای هر قاعده بصورت $A \rightarrow \alpha$ در گرامر قدمهای ۲ و ۳ را انجام می دهیم.

۲- برای هر پایانه a در $First(\alpha)$ , شماره قاعده $A \rightarrow \alpha$ را به خانه $M [ A , a ]$ اضافه می کنیم.

۳- اگر ε در $First(\alpha)$ وجود داشت شماره قاعده $A \rightarrow \alpha$ را در خانه های $M [ A , b ]$ به ازای هر $b \in Follow(A)$ قرار می دهیم.

جدول پارس گرامر مثال قبل بفرم زیر است :

| | id | + | * | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | 1 | | | 1 | | |
| E' | | 2 | | | 3 | 3 |
| T | 4 | | | 4 | | |
| T' | | 6 | 5 | | 6 | 6 |
| F | 8 | | | 7 | | |

حال با استفاده از جدول فوق عبارت id + id * id را تجزیه می کنیم :

| محتوای انباره | ورودی | قواعد استفاده شده |
|---|---|---|
| $ E | id + id * id $ | E → T E' |
| $ E' T | id + id * id $ | T → F T' |
| $ E' T' F | id + id * id $ | F → id |
| $ E' T' id | id + id * id $ | |
| $ E' T' | + id * id $ | T' → ε |
| $ E' | + id * id $ | E' → + T E' |
| $ E' T + | + id * id $ | |
| $ E' T | id * id $ | T → F T' |
| $ E' T' F | id * id $ | F → id |
| $ E' T' id | id * id $ | |
| $ E' T' | * id $ | T' → * F T' |
| $ E' T' F * | * id $ | F → id |
| $ E' T' F | id $ | |
| $ E' T' id | id $ | |
| $ E' T' | $ | T' → ε |
| $ E' | $ | E' → ε |
| $ | $ | پایان پارس |

## گرامرهای LL(1)

در صورتیکه از روش فوق برای ایجاد جدول پارس گرامرهای گنگ و یا چپ گرد استفاده شود در برخی از خانه های جدول پارس بیش از یک شماره قاعده خواهیم داشت.بعبارت دیگر اگر در خانه های جدول پارس یک گرامر مستقل از متن حداکثر یک شماره قاعده باشد گرامر مربوطه را LL(1) گویند.

مثال – گرامر زیر را در نظر بگیرید :

1 – 2 $S \rightarrow i\ E\ t\ S\ S' \mid a$
3 – 4 $S' \rightarrow e\ S \mid \varepsilon$
5 $E \rightarrow b$

جدول پارس گرامر فوق بصورت زیر است :

| | i | t | a | e | b | $ |
|---|---|---|---|---|---|---|
| S | 1 | | 2 | | | |
| S' | | | | 3,4 | | 4 |
| E | | | | | 5 | |

این گرامر LL(1) نیست زیرا در خانه M[S',e] جدول تجزیه آن دو شماره قاعده قرار دارد.

برای پی بردن به LL(1) بودن یک گرامر لازم نیست که حتما جدول تجزیه آن بدست آید.با بررسی شرایط زیر نیز می توان LL(1) بودن یک گرامر را بررسی نمود.بعبارت دیگر گرامری LL(1) است که شرایط زیر در مورد قواعد بصورت $A \rightarrow \alpha \mid \beta$ آن صدق کند :

۱. $First(\beta) \cap First(\alpha) = \phi$
۲. حداکثر یکی از رشته های $\alpha$ و $\beta$ رشته $\varepsilon$ را تولید کنند.
۳. اگر $\alpha \Rightarrow^* \varepsilon$ در آنصورت $First(\beta) \cap Follow(A) = \phi$

بعنوان مثال گرامر زیر را در نظر بگیرید :

1 $E \rightarrow T\ E'$
2 – 3 $E' \rightarrow + T\ E' \mid \varepsilon$
4 $T \rightarrow F\ T'$
5 – 6 $T' \rightarrow * F\ T' \mid \varepsilon$
7 – 8 $F \rightarrow ( E ) \mid id$

شرایط LL(1) بودن را برای گرامر فوق چک می کنیم.از آنجا که غیرپایانه های E و T تنها یک قاعده دارند نیازی به بررسی ندارند.در مورد غیرپایانه E' :

$First(+TE') = \{ + \}\ ,\ First(\varepsilon) = \{ \varepsilon \}\ ,\ \{ \varepsilon \} \cap \{ + \} = \phi$
$First(+TE') = \{ + \}\ ,\ Follow(E') = \{ \$,) \}\ ,\ \{ + \} \cap \{ \$,) \} = \phi$

و برای غیرپایانه T' داریم :

$First(*FT') = \{ * \}\ ,\ First(\varepsilon) = \{ \varepsilon \}\ ,\ \{ \varepsilon \} \cap \{ * \} = \phi$
$First(*FT') = \{ * \}\ ,\ Follow(T') = \{ \$,),+ \}\ ,\ \{ * \} \cap \{ \$,),+ \} = \phi$

حال شرایط را برای غیرپایانه F بررسی می کنیم :

$First((E)) = \{ ( \}\ ,\ First(id) = \{ id \}\ ,\ \{ ( \} \cap \{ id \} = \phi$

لذا گرامر LL(1) است.

حال بعنوان یک مثال دیگر گرامر زیر را در نظر بگیرید :

$S \rightarrow i\ e\ t\ S\ S' \mid a$
$S' \rightarrow e\ S \mid \varepsilon$
$E \rightarrow b$

با توجه به آنکه ε عضو First(S') است اگر شرط سوم را در مورد غیرپایانه S' چک کنیم مشخص خواهد شد که این گرامر LL(1) نیست.

$First(eS) = \{ e \}\ ,\ Follow(S') = \{ e,\$ \}\ ,\ \{ e,\$ \} \cap \{ e \} \neq \phi$

گرامرهایی که چپ گردی داشته باشند LL(1) نیستند.برخی از گرامرها را می توان با حذف چپ گردی و فاکتورگیری از چپ به گرامر LL(1) تبدیل کرد.ولی

فاکتورگیری و حذف چپ گردی باعث از بین رفتن خوانایی گرامرها می شوند.در ضمن تولید کد را نیز مشکل تر می نمایند.

---

## ۳- ۱۱ روش های اصلاح خطای نحوی در روش تجزیه LL(1)

از مهمترین روشهای اصلاح خطای قابل استفاده در تجزیه LL(1) عبارتند از :

- روش Panic Mode
- روش Phrase Level

بطور کلی در روش LL(1) زمانی یک خطای نحوی تشخیص داده می شود که یا پایانه بالای انباره با توکن جاری تطبیق نکند و یا با غیرپایانه بالای انباره A و توکن جاری a خانه M [ A , a ] خالی باشد.

در روش Panic Mode اگر پارسر با مراجعه به یک خانه خالی جدول تجزیه یک خطای نحوی بیابد آنقدر از رشته ورودی حذف می کند تا به یکی از اعضا مجموعه ای موسوم به مجموعه Synchronizing برسد.در روش Panic Mode به ازای هر غیرپایانه در گرامر یک مجموعه Synchronizing در نظر گرفته می شود.کارایی روش Panic Mode نیز بستگی به انتخاب مناسب مجموعه Synchronizing دارد.این مجموعه باید یه گونه ای تعیین شود که عمل تجزیه بتواند بدون حذف قسمت زیادی از ورودی به کار خود ادامه دهد.یک انتخاب مناسب , در نظرگرفتن مجموعه Follow هر غیرپایانه به عنوان مجموعه Synchronizing آن غیرپایانه است.با این وجود در نظرگرفتن مجموعه Follow تنها برای Synchronizing کافی نیست.برای اینکه حذف کمتری در برنامه ورودی صورت بگیرد می توان نمادهای بیشتری را به این مجموعه افزود.مثلا می توان مجموعه First غیرپایانه ها را نیز به مجموعه Synchronizing آنها افزود.به عنوان یک مثال از نحوه عمل پیاده سازی روش Panic Mode در تجزیه LL(1) گرامر زیر را در نظر بگیرید :

1 $E \rightarrow T\ E'$

2 – 3 $E' \rightarrow + T\ E' \mid \varepsilon$

4 $T \rightarrow F\ T'$

5 – 6 $T' \rightarrow * F T' \mid \varepsilon$
7 – 8 $F \rightarrow ( E ) \mid id$

مجموعه Follow غیرپایانه ها را به عنوان مجموعه Synchronizing آنها در نظر گرفته و در جدول تجزیه در مقابل Follow غیرپایانه ها با گذاردن علامتی مثل " S " مجموعه Synchronizing هر غیرپایانه را معین می کنیم.به این ترتیب جدول پارس گرامر فوق بصورت زیر در خواهد آمد :

| | id | + | * | ( | ) | $ |
|---|---|---|---|---|---|---|
| E | 1 | | | 1 | S | S |
| E' | | 2 | | | 3 | 3 |
| T | 4 | S | | 4 | S | S |
| T' | | 6 | 5 | | 6 | 6 |
| F | 8 | S | S | 7 | S | S |

برای **اصلاح خطا به روش Panic Mode** در الگوریتم تجزیه LL(1) بصورت زیر عمل می کنیم :

۱- اگر پارسر خانه M [ A , a ] را خالی ببیند علامت a را در ورودی نادیده می گیرد.

۲- اگر در محل خانه M [ A , a ] علامت " S " باشد غیرپایانه بالای انباره حذف می شود.مشروط برآنکه A تنها غیرپایانه موجود در انباره نباشد.

۳- اگر پایانه بالای انباره با ورودی جاری تطبیق نکند پایانه بالای انباره حذف می شود.

## ۳ - ۱۲ تجزیه پایین به بالا ( Bottom-Up Parsing )

یک روش کلی تجزیه پائین به بالا به روش انتقال – کاهش ( Shift – Reduce ) است.در این روش عکس تجزیه بالا به پائین عمل می شود.به این ترتیب که از رشته ورودی شروع کرده و ساخت درخت تجـزیه از برگ ها آغاز گـشته و به طرف ریـشه ( علامت شروع ) پیش می رود.

ترتیب بکارگیری قواعد در پارس بالا به پائین درست مطابق بسط چپ است درحالیکه ترتیب بکارگیری قواعد در اکثر روشهای تجزیه پائین به بالا درست عکس بسط راست است.گرامر زیر را در نظر بگیرید :

1 $S \rightarrow aABe$

2 – 3 $A \rightarrow Abc \mid b$

4 $B \rightarrow d$

جمله abbcde را مورد بررسی قرار می دهیم.بسط راست این جمله بصورت زیر است :

$$S \underset{rm}{\overset{1}{\Rightarrow}} aABe \underset{rm}{\overset{4}{\Rightarrow}} aAde \underset{rm}{\overset{2}{\Rightarrow}} aAbcde \underset{rm}{\overset{3}{\Rightarrow}} abbcde$$

که در آن ترتیب بکارگیری قواعد گرامر بصورت 1,4,2,3 ( از چپ به راست ) است.تجزیه پائین به بالای رشته فوق در جدول زیر آمده است :

| مرحله تجزیه | فرم جمله ای تحت تجزیه | شماره قاعده | دستگیره |
|---|---|---|---|
| | S | | |
| ۴ | <u>aABe</u> | ۱ | aABe |
| ۳ | aA<u>d</u>e | ۴ | d |
| ۲ | a<u>Abc</u>de | ۲ | Abc |
| ۱ | a<u>b</u>bcde | ۳ | b |

به این ترتیب جمله abbcde به علامت شروع گرامر کاهش می یابد.ترتیب عملیات در این کاهش درست برعکس بسط راست صورت گرفته است.در هر مرحله از کاهش در پارس پائین به بالا این مشکل وجود دارد که پارسر کدام زیر رشته را به عنوان دستگیره انتخاب و سپس از کدام قاعده برای کاهش آن استفاده نماید.در ادامه به ارائه چند تعریف در ارتباط با تجزیه پائین به بالا می پردازیم :

**عبارت ( Phrase ) :** بخشی از یک فرم جمله ای است که از یک غیرپایانه بوجود آمده باشد.به عنوان مثال در بسط زیر $\beta$ یک عبارت محسوب می شود.

$$S \overset{*}{\Rightarrow} \alpha A \gamma \overset{+}{\Rightarrow} \alpha\beta\gamma$$

**عبارت ساده ( Simple Phrase ) :** عبارتی است که در یک قدم بوجود آمده باشد.به عنوان مثال در بسط زیر $\beta$ یک عبارت ساده است.

$$S \overset{*}{\Rightarrow} \alpha A \gamma \Rightarrow \alpha\beta\gamma$$

**دستگیره ( Handle ) :** عبارت ساده ای است که در جهت عکس یک بسط راست تولید شده باشد.در مثال زیر $\beta$ یک دستگیره است.توجه داشته باشید که از آنجائیکه دستگیره در رابطه با بسط راست مطرح است سمت راست دستگیره هیچ غیرپایانه ای نیست.به همین خاطر در مثال زیر از " x " برای نمایش زیر رشته سمت راست دستگیره استفاده شده است.

$$S \overset{*}{\Rightarrow} \alpha A x \Rightarrow \alpha\beta x$$

اگر گرامر مورد استفاده گنگ نباشد در هر مرحله از تجزیه پائین به بالا تنها یک دستگیره وجود دارد.لیکن در صورت استفاده از یک گرامر گنگ ممکن است در بعضی از قدم ها بیشتر از یک دستگیره موجود باشد.به مثال زیر توجه کنید :

$$1-4 \quad E \rightarrow E + E \mid E * E \mid ( E ) \mid id$$

از آنجا که گرامر فوق گنگ است برای جمله id + id * id دو بسط راست و در نتیجه دو مسیر تجزیه پائین به بالا وجود دارد.این دو بسط در ادامه نشان داده می شود.همانگونه که مشاهده می شود در قدم سوم تجزیه دو انتخاب برای دستگیره وجود دارد.

$E \Rightarrow \underline{E + E}$ $E \Rightarrow \underline{E * E}$

$\Rightarrow E + \underline{E * E}$ $\Rightarrow E * \underline{id}$

$\Rightarrow E + E * \underline{id}$ $\Rightarrow \underline{E + E} * id$

$\Rightarrow E + \underline{id} * id$ $\Rightarrow E + \underline{id} * id$

$\Rightarrow \underline{id} + id * id$ $\Rightarrow \underline{id} + id * id$

---

### ۳- ۱۳ پیاده سازی روش تجزیه انتقال – کاهش با استفاده از یک انباره

در این روش از یک انباره و یک بافر ورودی جهت نگهداری رشته ای که باید تجزیه شود استفاده می گردد.در وضعیت شروع تجزیه به انتهای ورودی یک علامت " $ " اضافه می گردد که خاتمه رشته ورودی برای پارسر مشخص می گردد.درون انباره نیز یک علامت " $ " وارد می گردد.

پارسر آنقدر دو عمل انتقال و کاهش را انجام می دهد که یا یک خطای نحوی مشاهده گردد و یا اینکه به وضعیت خاتمه پارس برسد.وضعیت خاتمه تجزیه به این صورت است که توکن جاری علامت " $ " است و درون انباره نیز تنها علامت شروع گرامر بر روی علامت " $ " که در ابتدای تجزیه وارد انباره گردیده است قرار دارد.

بطور رسمی تر اعمالی که یک پارسر انتقال – کاهش انجام می دهد عبارتند از :

۱- **انتقال ( Shift ) :** تعدادی از علائم ورودی به بالای انباره انتقال می یابد.عمل انتقال تا زمانی ادامه می یابد که دستگیره در بالای انباره تشخیص داده شود.

۲- **کاهش ( Reduce ) :** دستگیره ای در بالای انباره ظاهر شده است.دستگیره از بالای انباره حذف و بجای آن غیرپایانه سمت چپ قاعده ای که سمت راست آن مطابق دستگیره است وارد انباره می شود.

۳- **قبول ورودی ( Accept ) :** پارسر پایان موفقیت آمیز تجزیه را اعلام می کند.

۴- **تشخیص خطا ( Error ) :** پارسر یک خطای نحوی را تشخیص داده و رویه خطاپرداز را فرا می خواند.

به عنوان نمونه تجزیه رشته id + id * id به روش انتقال – کاهش بصورت زیر است :

| عمل انجام شده | باقیمانده ورودی | محتوای انباره |
|---|---|---|
| انتقال id | id + id * id $ | $ |
| کاهش بوسیله E → id | + id * id $ | $ id |
| انتقال + | + id * id $ | $ E |
| انتقال id | id * id $ | $ E + |
| کاهش بوسیله E → id | * id $ | $ E + id |
| انتقال * | * id $ | $ E + E |
| انتقال id | id $ | $ E + E * |
| کاهش بوسیله E → id | $ | $ E + E * id |
| کاهش بوسیله E → E*E | $ | $ E + E * E |
| کاهش بوسیله E → E+E | $ | $ E + E |
| Accept | $ | $ E |

در تجزیه به روش انتقال – کاهش مشکلات زیر وجود دارد :

۱- تصمیم گیری در مورد اینکه کدام زیررشته تشکیل یک دستگیره می دهد.

۲- انتخاب قاعده ای که باید برای کاهش استفاده شود.این مشکل زمانی بروز می کند که سمت راست بیش از یک قاعده با دستگیره مطابقت می کند.به چنین وضعیتی " تداخل کاهش– کاهش" (Reduce / Reduce Conflict ) گفته می شود.

## ۳ – ۱۴ انواع تداخل در تجزیه انتقال – کاهش

در پارس انتقال – کاهش دو نوع تداخل می تواند روی دهد :

**۱- تداخل انتقال – کاهش ( Shift / Reduce Conflict ) :**

زمانی روی می دهد که پارسر نتواند تصمیم بگیرد که عمل انتقال باید انجام دهد یا عمل کاهش.

**۲- تداخل کاهش – کاهش ( Reduce / Reduce Conflict ) :**

اگر بیش از یک قاعده جهت کاهش موجود باشد اینگونه تداخل روی خواهد داد.

به عنوان مثالی از تداخل نوع اول گرامر زیر را در نظر بگیرید :

```
Stmt → if  expr  then  Stmt
     | if  expr  then  Stmt  else  Stmt
     | other
```

فرض کنید توکن جاری " else " و رشته " if expr then Stmt " بالای انباره قرار داشته باشد.با توجه به وضعیت انباره , پارسر هم می تواند با استفاده از قاعده اول عمل کاهش انجام دهد و هم می تواند ابتدا توکن جاری را به بالای انباره انتقال داده و در زمان مناسب با استفاده از قاعده دوم عمل کاهش را انجام دهد.

حال به مثالی از تداخل نوع دوم توجه کنید.گرامر زیر را در نظر بگیرید :

```
1 – 2   Stmt → id ( Parameter-list ) | Expr := Expr
3 – 4   Parameter-list → Parameter-list , Parameter | Parameter
  5     Parameter → id
6 – 7   Expr → id ( Expr-list ) | id
8 – 9   Expr-list → Expr-list , Expr | Expr
```

قاعده شماره 1 این گرامر جهت توصیف فراخوانی رویه ها و قاعده 6 گرامر جهت توصیف مراجعه به آرایه ها است.فرض کنید به عنوان بخشی از ورودی رشته "A (I,J)" آمده باشد.این بخش از ورودی بوسیله اسکنر بصورت " id(id,id) " تبدیل می گردد.همچنین فرض کنید در وضعیتی از تجزیه قرار داریم که باقیمانده ورودی بصورت " $...(id, " و زیررشته " id(id " بالای انباره ظاهر شده باشد.در این حالت از دو

قاعده جهت عمل کاهش " id " می توان استفاده نمود ( قواعد 5 و 7 ).یعنی یک تداخل کاهش / کاهش رخ داده است.در این مثال انتخاب قاعده درست بستگی به نوع متغیر A دارد.در صورتی که A یک رویه باشد باید از قاعده 5 و اگر A یک آرایه باشد باید از قاعده 7 برای عمل کاهش استفاده شود.یعنی پارسر باید با مراجعه به جدول نشانه ها و پی بردن به نوع A این تداخل را حل کند.راه حل دیگری نیز جهت رفع این نوع مشکل وجود دارد.اگر اسکنر در هنگامی که متغیر ورودی یک رویه است بجای توکن " id " توکن دیگری مثلا " procid " به پارسر انتقال دهد دراینصورت در موقع برخورد با وضعیت تداخل کافیست پارسر داخل انباره و توکن زیر " ) " را بررسی کند.اگر این توکن procid باشد پارسر از قاعده شماره 5 و در غیر اینصورت از قاعده شماره 7 جهت کاهش id بالای انباره استفاده می کند.توجه داشته باشید که در اینصورت قاعده شماره 1 بایستی بصورت زیر اصلاح گردد :

1 Stmt → procid( Parameter-list)

## ۳ – ۱۵ روش تجزیه تقدم – عملگر ( Operator-Precedence Parsing )

**گرامر عملگر :** گرامری است که دارای خصوصیات زیر باشد :

۱. قاعده ε نداشته باشد.

۲. در سمت راست هیچ قاعده ای از آن دو غیرپایانه مجاور نباشند.

به عنوان مثال گرامر زیر یک گرامر عملگر نیست زیرا سمت راست قاعده $E \rightarrow EAE$ دو غیرپایانه مجاور دارد.

$E \rightarrow EAE \mid (E) \mid -E \mid id$
$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$

اگر این گرامر را بصورت زیر تبدیل کنیم گرامر عملگر خواهد شد :

$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E\uparrow E \mid (E) \mid -E \mid id$

**تذکر :** یکی از شرایطی که باید وجود داشته باشد تا بتوان از روش تجزیه تقدم – عملگر استفاده نمود این است که گرامر باید یک گرامر عملگر باشد.

### معایب روش پارس تقدم – عملگر

روش تقدم – عملگر علیرغم داشتن مزیت پیاده سازی آسان دارای معایبی نیز است.این معایب عبارتند از :

۱- به دلیل محدودیت هایی که دارد گرامرهای کمی وجود دارند که بتوان از این روش برای آنها استفاده کرد.

۲- در مورد اپراتورهایی مانند ' – ' (minus ) که دارای دو تقدم متفاوتند ( بسته به اینکه منهای unary است یا binary ) این روش کار نمی کند.

۳- روش چندان دقیقی نیست.یعنی ممکن است برخی از خطاهای نحوی را نتواند کشف کند.

در پارس تقدم – عملگر از سه رابطه تقدم مابین عملیات جهت هدایت عمل تجزیه استفاده می گردد.در این روش روابط تقدم تنها بین پایانه های گرامر و بصورت زیر تعریف می گردد :

۱- $a < b$ یعنی پایانه a از پایانه b تقدم کمتری دارد.مانند $+ < *$

۲- $a = b$ یعنی پایانه a و پایانه b از تقدم یکسانی برخوردارند.مانند ( = )

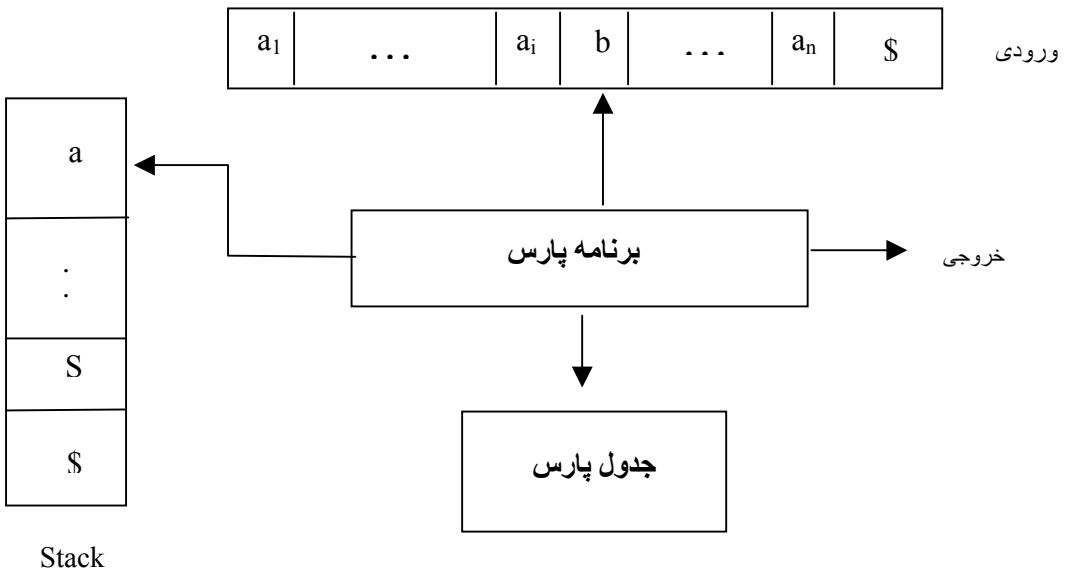
۳- $a > b$ یعنی پایانه a از پایانه b تقدم بیشتری دارد.مانند $id > *$

روابط تقدمی که در اینجا بین پایانه ها توصیف می شود با روابط $<$ , $=$ , $>$ معمولی که در بین اعداد طبیعی برقرار است تفاوت زیادی دارند.به عنوان نمونه در اینجا با دانستن اینکه رابطه $a < b$ برقرار است نمی توان نتیجه گرفت که $b > a$ .از طرفی ممکن است در پایانه هیچیک از این سه رابطه تقدم را با هم نداشته باشند و یا اینکه دو پایانه دو رابطه تقدم متفاوت داشته باشند.مثلا داشته باشیم که $* > -$ و هم $* < -$ .

## ۳ – ۱۵ – ۱ الگوریتم تجزیه تقدم – عملگر

ساختار پارسر در روش تقدم – عملگر کاملا مشابه ساختار یک پارسر LL(1) است.در این ساختار که در شکل زیر نشان داده شده است مولفه اصلی پارسر یک برنامه است که از یک طرف ورودی خود را از اسکنر دریافت می کند و از یک انباره برای ذخیره اطلاعات و از یک جدول تجزیه برای هدایت عمل تجزیه استفاده می کند.در این روش پارسر ابتدا یک علامت $ به انتهای رشته ورودی اضافه می کند.انباره نیز در ابتدای کار فقط شامل یک علامت $ است.

جدول پارس در این روش یک جدول دو بعدی مربع است که به تعداد پایانه های بعلاوه یک ( به خاطر علامت $ ) سطر و ستون دارد.در داخل جدول نیز در برخی خانه های جدول یکی از علامت های $<$ , $>$ و یا $=$ قرار دارد و مابقی خانه های جدول خالی است.



در این روش برنامه پارس در هر قدم از پارس با استفاده از بالاترین پایانه انباره (a) و توکن جاری (b) ( به غیرپایانه بالای انباره توجهی ندارد ) به عنوان اندیس جدول تجزیه و مراجعه به این جدول یکی از اعمال زیر را انجام می دهد :

۱- اگر رابطه پایانه بالای انباره و توکن جاری بصورت $a < b$ باشد پارسر ابتدا علامت $<$ و سپس توکن جاری b را به بالای انباره انتقال می دهد.

۲- اگر رابطه پایانه بالای انباره و توکن جاری بصورت $a = b$ باشد پارسر فقط توکن جاری را به بالای انباره انتقال می دهد.

۳- اگر رابطه پایانه بالای انباره و توکن جاری بصورت $a > b$ باشد پارسر عمل کاهش را انجام می دهد.برای اینکار در داخل انباره آنقدر پائین می رود تا به اولین علامت $<$ برسد , دستگیره رشته مابین این علامت و بالای انباره است ( بعلاوه غیرپایانه زیر علامت $<$ در صورت وجود ) .پارسر برای انجام عمل کاهش دستگیره پیدا شده را از انباره حذف و به جای آن یک غیرپایانه نوعی ( مثلا N ) وارد انباره می کند.( در روش تقدم – عملگر پس از تهیه جدول تجزیه از روی گرامر دیگر بین غیرپایانه های گرامر تمایزی قائل نمی شویم و بجای همه آنها می توان از یک غیرپایانه نوعی استفاده نمود.همچنین این عامل باعث ضعف این روش در کشف برخی از خطاهای نحوی گردیده است.)

۴- اگر پایانه بالای انباره با ورودی جاری رابطه ای نداشته باشد یک خطای نحوی است و پارسر رویه اصلاح خطا را فرا می خواند.

حال به عنوان مثال به تجزیه رشته $id + id * id$ با استفاده از جدول تجزیه گرامرغیر گنگ عبارات جبری توجه کنید.جدول تجزیه و مراحل تجزیه بصورت قدم به قدم در شکل های صفحه بعد آمده است.در قدم هایی که عمل کاهش صورت گرفته زیر دستگیره خط کشیده شده است.

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid id$

| | + | * | ( | ) | id | $ |
|---|---|---|---|---|---|---|
| + | > | < | < | > | < | > |
| * | > | > | < | > | < | > |
| ( | < | < | < | = | < | |
| ) | > | > | | > | | > |
| id | > | > | | > | | > |
| $ | < | < | < | | < | = |

| انباره | ورودی | عمل انجام شده |
|---|---|---|
| $ | id + id * id $ | انتقال < و id |
| $ <u>< id</u> | + id * id $ | کاهش بوسیله E → id |
| $ E | + id * id $ | انتقال > و + |
| $ E < + | id * id $ | انتقال < و id |
| $ E < + <u>< id</u> | * id $ | کاهش بوسیله E → id |
| $ E < + E | * id $ | انتقال > و * |
| $ E < + E < * | id $ | انتقال < و id |
| $ E < + E < * <u>< id</u> | $ | کاهش بوسیله E → id |
| $ E < + <u>E < * E</u> | $ | کاهش بوسیله E → E*E |
| $ <u>E < + E</u> | $ | کاهش بوسیله E → E+E |
| $ E | $ | پایان پارس |

## ۳ – ۱۵ – ۲ نحوه یافتن روابط تقدم

برای تعیین روابط تقدم از دو تابع با تعریف زیر استفاده می کنیم.این دو تابع روی غیرپایانه ها تعریف شده اند و حاصل آنها مجموعه ای از پایانه ها است.

$Firstterm(A) = \{ a \mid A \overset{+}{\Rightarrow} a\alpha \quad or \quad A \overset{+}{\Rightarrow} Ba\alpha \}$
$Lastterm(A) = \{ a \mid A \overset{+}{\Rightarrow} \alpha a \quad or \quad A \overset{+}{\Rightarrow} \alpha aB \}$

که در آن a یک پایانه , B یک غیرپایانه و α یک رشته از پایانه و غیرپایانه است.

با توجه به تعاریف توابع فوق رابطه های تقدم بصورت زیر تعریف می شوند :

$a = b$ iff $\exists\ U \rightarrow \ldots ab \ldots$ or $U \rightarrow \ldots aWb \ldots$

$a < b$ iff $\exists\ U \rightarrow \ldots aW \ldots$ and $b \in Firstterm(W)$

$a > b$ iff $\exists\ U \rightarrow \ldots Wb \ldots$ and $a \in Lastterm(W)$

که W یک غیر پایانه است.

حال به عنوان مثال گرامر زیر را در نظر بگیرید :

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow ( E ) \mid id$

اگر Firstterm و Lastterm را بر روی غیرپایانه های این گرامر اعمال کنیم حاصل بصورت زیر خواهد شد :

$Firstterm(E) = \{ + , * , ( , id \}$

$Firstterm(T) = \{ * , ( , id \}$

$Firstterm(F) = \{ ( , id \}$

$Lastterm(E) = \{ + , * , ) , id \}$

$Lastterm(T) = \{ * , ) , id \}$

$Lastterm(F) = \{ ) , id \}$

برای بدست آوردن رابطه علامت $ و سایر پایانه ها قاعده ای بفرم $N \rightarrow \$\ S\ \$$ که در آن N یک غیرپایانه جدید و S علامت شروع گرامر است به گرامر اضافه می کنیم.

حال با توجه به تعاریف روابط تقدم – عملگر که در بالا عنوان شد رابطه تساوی به سادگی با بررسی قواعد بدست می آید.در گرامر فوق در نظر گرفتن قاعده جدیدی که بخاطر علامت $ اضافه می گردد دو رابطه تساوی وجود دارد : $\$ = \$$ و $( = )$ .توجه داشته باشید که $( = )$ هیچ اطلاعی در مورد رابطه " ( " و " ) " به ما نمی دهد.مثلا از این نمی توان نتیجه گرفت که $) = ($ .

برای یافتن روابط $<$ , با توجه به تعریف آن به دنبال نقاطی در گرامر می گردیم که یک پایانه در سمت چپ یک غیرپایانه قرار گرفته باشد.مثلا در قاعده پنجم این گرامر " ) "

سمت چپ غیرپایانه E قرار گرفته است.حال اگر پایانه ای مثلا b عضو مجموعه Firstterm(E) باشد رابطه تقدم ( < b ) برقرار است.در واقع این رابطه بین " ) " و همه اعضا Firstterm(E) برقرار است.یعنی در اینجا می توان نتیجه گرفت که روابط ( + < , * < ) , ( ( < ) , ( id < ) برقرار است.به همین ترتیب می توان سایر روابط تقدم < را نیز یافت.

برای یافتن روابط > , با توجه به تعریف آن به دنبال نقاطی در گرامر می گردیم که یک پایانه در سمت راست یک غیرپایانه قرار گرفته باشد.مثلا در قاعده پنجم این گرامر , ( " " سمت راست غیرپایانه E قرار گرفته است.حال اگر پایانه ای مثلا a عضو مجموعه Lastterm(E) باشد رابطه تقدم ( a > ) برقرار است.در واقع این رابطه نیز بین " ( " و همه اعضا Lastterm(E) برقرار است.یعنی در اینجا می توان نتجه گرفت که روابط ( + > ) , ( * > ) , ( ( > ) و ( id > ) برقرار است.به همین ترتیب می توان سایر روابط تقدم > را نیز یافت.

در نهایت جدول روابط تقدم گرامر فوق بصورت زیر خواهد بود.محل های خالی در جدول نشانگر آن است که دو پایانه با هم رابطه ندارند.مثلا در اینجا id با id رابطه ندارد.این بدان معنی است که با استفاده از گرامر فوق نمی توان فرم جمله ای تولید نمود که در آن دو id مجاور هم قرار بگیرند.( در اینجا منظور از مجاور بودن دو پایانه این است که یا دقیقا مجاور باشند و یا اینکه بین آنها یک غیرپایانه باشد.توجه داشته باشید که به دلیل محدودیت خاصی که روی قواعد گرامر عملگر وجود دارد امکان ندارد که در یک فرم جمله ای دو غیرپایانه مجاور هم قرار بگیرند.بنابراین حداکثر ممکن است بین دو پایانه یک غیرپایانه قرار داشته باشد که در اینصورت نیز طبق تعریف آن دو پایانه مجاور محسوب می شوند.)

| | + | * | ( | ) | id | $ |
|---|---|---|---|---|---|---|
| + | .> | <. | <. | .> | <. | .> |
| * | .> | .> | <. | .> | <. | .> |
| ( | <. | <. | <. | = | <. | |
| ) | .> | .> | | .> | | .> |
| id | .> | .> | | .> | | .> |
| $ | <. | <. | <. | | <. | = |

### ۳ – ۱۵- ۳ اصلاح خطا در روش تقدم – عملگر

در این روش کلا به دو صورت یک خطای نحوی تشخیص داده می شود.اول وقتیکه هیچ رابطه ای بین پایانه بالای انباره و ورودی جاری نباشد و دوم هنگامی که دستگیره بالای انباره با سمت راست هیچ قاعده ای تطبیق نکند.

برای اصلاح خطاهای نوع اول در خانه های خالی جدول نشانه روهایی به زیر روال های اصلاح خطا می گذاریم بطوریکه اگر در عمل تجزیه به یک خانه خالی جدول رجوع شده زیرروال مربوطه فراخوانی شده و خطا به نحو مقتضی اصلاح گردد.

به عنوان نمونه گرامر زیر را در نظر بگیرید : $E \rightarrow E + T \mid T$

$T \rightarrow (E) \mid id$

جدول تجزیه این گرامر بصورت زیر است :

| | id | ( | ) | $ | + |
|---|---|---|---|---|---|
| $ | < | < | $e_1$ | = | < |
| ) | $e_2$ | $e_2$ | > | > | > |
| id | $e_2$ | $e_2$ | > | > | > |
| ( | < | < | = | $e_3$ | < |
| + | < | < | > | > | > |

روال های اصلاح خطا بصورت زیر تعریف می شوند :

- e1 : توکن " ( " را حذف و پیغام " در ورودی یک پرانتز بسته اضافی وجود دارد " را چاپ کن.
- e2 : پایانه " + " را به ورودی اضافه و پیغام " یک عملگر در برنامه کم است " را چاپ کن.
- e3 : پایانه " ) " را از بالای انباره حذف و پیغام " یک پرانتز بسته در ورودی کم است " را چاپ کن.

در صورت تشخیص خطای نوع دوم یعنی عدم تطبیق دستگیره با سمت راست هیچیک از قواعد گرامر پارسر به دنبال قاعده ای که سمت راست آن شبیه دستگیره باشد ( در یک یا

دو علامت تفاوت داشته باشند ) جستجو می کند و با توجه به اختلاف دستگیره و سمت راست قاعده پیدا شده پیغام مناسبی چاپ می کند و عمل کاهش را انجام می دهد.
مثلا فرض کنید دستگیره aNbc باشد و قاعده ای بصورت (aEc → ... ) پیدا شود.از آنجایی که غیرپایانه ها در این روش تجزیه اهمیتی ندارند و تنها محل آنها در انباره اهمیت دارد لذا در مقایسه دستگیره با سمت راست قواعد تنها به موقعیت غیرپایانه ها اهمیت داده می شود.در این مثال با توجه به اینکه اختلاف دستگیره و سمت راست قاعده در پایانه " b " است پیغام زیر صادر می شود.توجه داشته باشید که پایانه های اضافی در دستگیره نشانه علائم اضافی در برنامه ورودی است.

Illegal " b " on line ....

حال اگر دستگیره بصورت abEc باشد و سمت راست قاعده پیدا شده بصورت abEdc باشد پیغام زیر صادر خواهد شد :

Missing " d " on line ....

ممکن است اختلاف در مورد یک غیرپایانه باشد.بعنوان مثال فرض کنید abc دستگیره و aEbc سمت راست قاعده ای از گرامر باشد.دراینصورت صدور پیغامی بصورت " Missing " E " on line ... " مجاز نیست.زیرا کاربر یک کامپایلر اطلاعی در مورد غیرپایانه های گرامر ندارد و لذا در چاپ پیغامها نبایستی از غیرپایانه ها استفاده نمود.در این حالت بایستی با توجه به ساختار نحوی که غیرپایانه موردنظر توصیف می کند درباره خطای کشف شده گزارش داد.مثلا اگر E معرف یک عبارت جبری ( در ساده ترین شکل یک عملوند ) است می توان پیغام زیر را صادر نمود :

Missing Operand on line ....

توابع اولویت

کامپایلر هایی که از تجزیه کننده های عملگر- اولویت استفاده می کنند ، نیازی به ذخیره ی جدول روابط اولویت ندارند. در اکثر موارد ، این جدول می تواند توسط دو تابع اولویت f و g که نماد های پایانه را به اعداد صحیح تبدیل می نمایند ، کد گذاری شود. سعی بر این است که توابع f و g به گونه ای انتخاب شوند که برای هر نماد a و b :

۱- $f(a) < g(b)$ هرگاه $a < b$

۲- $f(a) = g(b)$ هرگاه $a = b$

۳- $f(a) > g(b)$ هرگاه $a > b$

بنا بر این رابطه ی اولویت بین a و b می تواند توسط مقایسه ی عددی بین f(a) و g(b) انجام گیرد. به هر حال توجه داشته باشید که ورودی های خطا در ماتریس اولویت ، مهم می باشند. زیرا یکی از حالت های (۱) ، (۲) و یا (۳) بدون توجه به این که f(a) و g(b) دارای چه مقادیری هستند ، برقرار می باشد. فقدان توانایی آشکار سازی خطا ، عموما به اندازه ای مهم نیست که مانع استفاده از توابع اولویت در موقع لزوم شود. هنوز هم امکان گرفتن خطا در زمانی که کاهش درخواست می شود ولی دستگیره یافت نمی گردد ، وجود دارد.

اینچنین نیست که هر جدول روابط اولویت ، دارای توابع اولویت به منظور کد گذاری آن باشد ، اما در عمل ، این توابع معمولا وجود دارند.

مثال )

جدول اولویت زیر را در نظر بگیرید :

| | + | − | * | / | ↑ | id | ( | ) | $ |
|---|---|---|---|---|---|---|---|---|---|
| + | > | > | < | < | < | < | < | > | > |
| − | > | > | < | < | < | < | < | > | > |
| * | > | > | > | > | < | < | < | > | > |
| / | > | > | > | > | < | < | < | > | > |
| ↑ | > | > | > | > | < | < | < | > | > |
| id | > | > | > | > | > | | | > | > |
| ( | < | < | < | < | < | < | < | = | |
| ) | > | > | > | > | > | | | > | > |
| $ | < | < | < | < | < | < | < | | |

توابع اولویت جدول مذکور به شکل زیر است :

| | + | − | * | / | ↑ | ( | ) | id | $ |
|---|---|---|---|---|---|---|---|---|---|
| f | 2 | 2 | 4 | 4 | 4 | 0 | 6 | 6 | 0 |
| g | 1 | 1 | 3 | 3 | 5 | 5 | 0 | 5 | 0 |

برای مثال ، $id > *$ و $g(id) > f(*)$ . توجه داشته باشید که $f(id) > g(id)$ بیان می دارد که $id > id$ ، اما در حقیقت هیچ رابطه ی اولویت بین id و id وجود ندارد. ورودی های دیگر خطا در جدول اولویت بالا به طور مشابه با یکی از روابط اولویت جایگزین می

گردند. روشی ساده به منظور یافتن توابع اولویت برای یک جدول ، اگر چنین توابعی وجود داشته باشند ، به ترتیب زیر است.

الگوریتم ساخت توابع اولویت:

ورودی : ماتریس عملگر- اولویت

خروجی : توابع اولویت نمایانگر ماتریس ورودی، یا نشان دهنده این است که چنین جدولی وجود ندارد.

روش:

۱- نماد fa وga را برای هر a که یک پایانه یا $ است ایجاد نمایید.

۲- نماد های ایجاد شده را به تعداد ممکن گروه تقسیم بندی کنید به شکلی که اگر a=b در این صورت faوgb در یک گروه قرار می گیرند. توجه داشته باشید که ممکن است مجبور شوید نمادهایی را در یک گروه قرار دهید، حتی اگر با رابطه ی = با یکدیگرمرتبط نشده باشند. برای مثال اگر a=bوc=b انگاه faو fc باید در یک گروه قرار گیرند، زیرا هردوی آنها در همان گروه gb قرار دارند.اگر علاوه بر آن،c=d آنگاه faو gd در یک گروه قرار دارند حتی اگر a=d وجود نداشته باشد.

۳- گراف جهت داری ایجاد کنید که گره های آن، گروه های ایجاد شده در مرحله ۲ باشند. برای هر aوb اگر a<b،آنگاه یک لبه از گروه gbبه گروه fa رسم کنید.اگر a>b انگاه یک لبه از گروه faبهgb رسم نمایید. توجه داشته باشید که یک لبه یا مسیر از fa به gb به این معنی است که fa باید بیش از g(b) باشد؛یک مسیر از gbبه fa به این معنی است که g(b) باید بیش از f(a) باشد.

۴- اگر کراف بدست آمده در ۳ دارای دوره باشد،توابع اولویت وجود ندارند. اگر دوره وجود نداشته باشد ،مقدار f(a) ، طول طولانی ترین مسیری است که از گروه fa شروع می شود مقدار ga طول طولانی ترین مسیر از گروهی است که ga در آن قرار دارد.

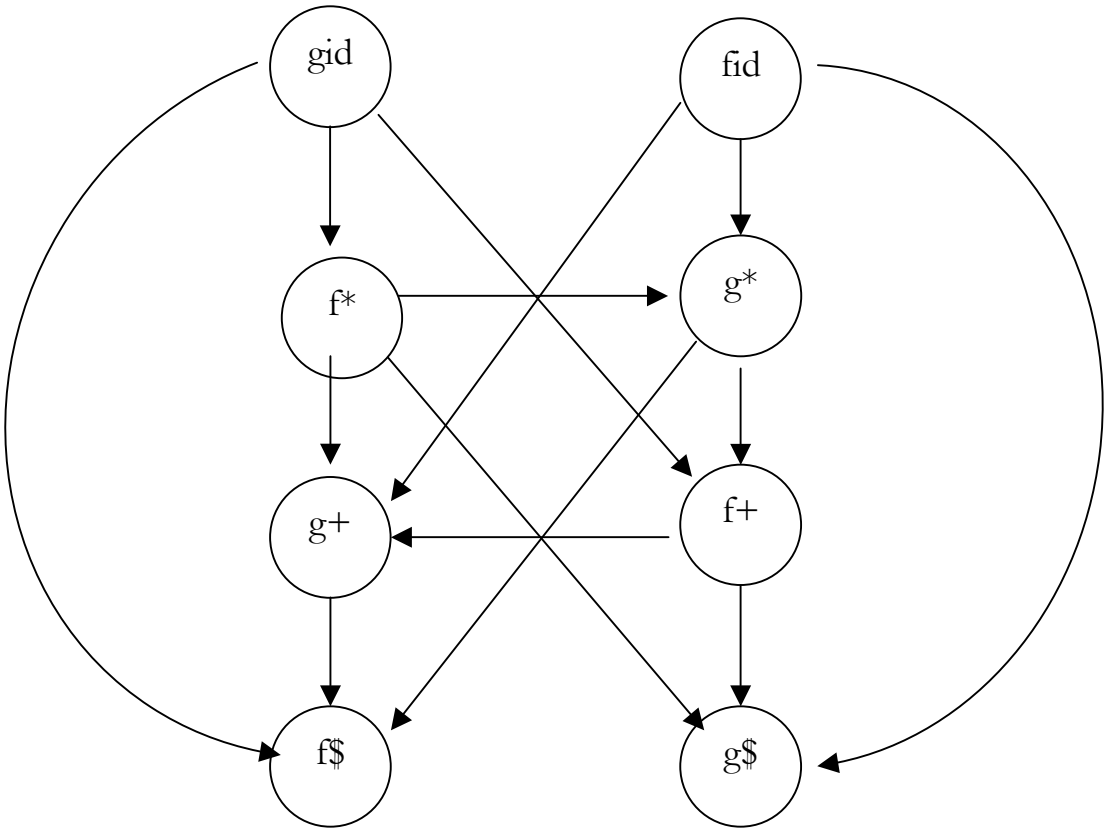
مثال:

ماتریس زیر را در نظر بگیرید :

| | id | + | * | $ |
|---|---|---|---|---|
| id | | .> | .> | .> |
| + | <. | .> | <. | .> |
| * | <. | .> | .> | .> |
| $ | <. | <. | <. | |

در این ماتریس هیچ رابطه = وجود ندارد ، بنابراین هر نماد،با خودش در یک گروه است.
شکل زیر گراف بدست آمده با الگوریتم بالا را نشان می دهد.
در این گراف دوره وجود ندارد،بنابراین توابع اولویت وجود دارند. از آنجایی که f$ و g$ لبه ی خروجی ندارند،f($)=g($)=0. طولانی ترین مسیر از g+ دارای طول 1 است،بنابراین g(+)=1 . یک مسیر از gid به f* و g* و f+ و g+ و f$ وجود دارند، بنابراین g(id)=5.توابع اولویت نتیجه عبارت اند از:

| | + | * | id | $ |
|---|---|---|---|---|
| f | 2 | 4 | 4 | 0 |
| g | 1 | 3 | 5 | 0 |



## ۳ – ۱۶ روش تجزیه تقدم ساده

این روش تجزیه بسیار شبیه روش تجزیه تقدم – عملگر است و در واقع بهبود یافته تقدم – عملگر است.در این روش روابط تقدم بین همه عناصر گرامر تعریف شده در حالیکه در تقدم – عملگر این روابط فقط بین پایانه ها تعریف می شود.برای استفاده از این روش محدودیت های کمتری نسبت به مورد تقدم – عملگر وجود دارد که باعث می شود که روش تقدم ساده طیف بیشتری از گرامرها را در بربگیرد.به عنوان نمونه در اینجا وجود غیرپایانه های مجاور در سمت راست قواعد مجاز است لیکن مانند حالت قبل وجود قواعد اپسیلون مجاز نیست.از آنجا که در روش تقدم ساده بر خلاف روش تقدم – عملگر بین غیرپایانه ها تمایز قائل می شویم .در اینجا یک محدودیت جدید داریم که سمت راست هیچ دو قاعده ای نباید یکسان باشد زیرا در غیراینصورت در بعضی از قدم ها تداخل کاهش – کاهش پیش خواهد آمد.البته این محدودیت چندان مهمی نیست.در هر دو مورد این روشها یک محدودیت مشترک وجود دارد که در خانه های جدول تجزیه بایستی حداکثر یک رابطه تقدم وجود داشته باشد.

در روش تقدم ساده هم برای هدایت عملیات از روابط سه گانه تقدم استفاده می شود.البته در روش تقدم ساده این روابط بین کلیه علائم گرامر ( پایانه و غیرپایانه و $ ) تعریف می شود.جدول تجزیه تقدم ساده یک جدول مربع است که به تعداد حاصل جمع تعداد پایانه ها و غیرپایانه های گرامر بعلاوه یک ( بخاطر علامت $ ) سطر وستون دارد.

برای تعیین روابط تقدم ساده از توابع با نامهای Head و Tail استفاده می شود که تعریف رسمی آنها بصورت زیر است ( هر دوی این توابع بر روی یک غیرپایانه عمل کرده و حاصل آنها مجموعه ای از علائم گرامر است ) :

$$Head(U) = \{ X \mid U \overset{+}{\Rightarrow} X\alpha \}$$
$$Tail(U) = \{ X \mid U \overset{+}{\Rightarrow} \alpha X \}$$

با استفاده از دو تابع فوق روابط تقدم ساده بصورت زیر تعریف می شوند :

$X = Y$ iff $\exists\, U \rightarrow \ldots XY \ldots$

$X < Y$ iff $\exists\, U \rightarrow \ldots XA \ldots$ and $Y \in Head(A)$

$X > Y$ iff $\exists\, U \rightarrow \ldots AB \ldots$ and $X \in Tail(A)$ and $Y \in Head(B)$ or $Y=B$

به عنوان مثال گرامر زیر را در نظر بگیرید :

$S \rightarrow ( S S )$
$S \rightarrow c$

مانند روش تقدم – عملگر ابتدا قاعده ای بفرم $N \rightarrow \$ S \$$ به قواعد گرامر اضافه کرده سپس مطابق روالی که در آنجا ذکر گردید به دنبال قواعدی می گردیم که شرایط تعاریف فوق در مورد آنها صدق نماید.حاصل این کار در مورد مثال فوق بصورت جدول تجزیه زیر خواهد بود.برای تفکیک روابط تقدم ساده از روابط تقدم – عملگر , روابط تقدم ساده با استفاده از علائم متفاوتی نمایش داده خواهد شد

.

| | S | $ | ( | ) | c |
|---|---|---|---|---|---|
| S | $\circledast$ | $\circledast$ | $\circledcirc$ | $\circledast$ | $\circledcirc$ |
| $ | $\circledast$ | | $\circledcirc$ | | $\circledcirc$ |
| ( | $\circledast$ | | $\circledcirc$ | | $\circledcirc$ |
| ) | | $\circledcirc$ | $\circledcirc$ | $\circledcirc$ | $\circledcirc$ |
| c | | $\circledcirc$ | $\circledcirc$ | $\circledcirc$ | $\circledcirc$ |

## ۳ – ۱۶ – ۱ الگوریتم تجزیه به روش تقدم ساده

در این روش پارسر در هر قدم از تجزیه با استفاده از توکن جاری b و عنصر بالای انباره X ( که می تواند پایانه یا غیرپایانه باشد ) به جدول تجزیه مراجعه کرده و بصورت یکی از حالات زیر عمل می کند :

۱. در صورتی که رابطه علامت بالای انباره و توکن جاری بصورت $X \circledcirc b$ باشد پارسر عمل انتقال را انجام می دهد.در این مورد ابتدا علامت $\circledcirc$ و سپس توکن جاری b را به بالای انباره منتقل می کند.
۲. در صورتی که رابطه دو عنصر مزبور بصورت $X \circledast b$ باشد پارسر فقط توکن جاری را به بالای انباره انتقال می دهد.
۳. در صورتیکه رابطه بصورت $X \circledcirc b$ باشد پارسر عمل کاهش را انجام می دهد.در این حالت دستگیره رشته بالای انباره تا اولین علامت $\circledcirc$ است.پارسر ابتدا دستگیره را از بالای انباره حذف می کند.اگر عنصر بالای انباره ( پس از

حذف دستگیره ) را Top بنامیم و سمت چپ قاعده ای را که پارسر از آن جهت کاهش استفاده می کند Lhs بنامیم پارسر رابطه بین Lhs و Top را از جدول استخراج نموده و یکی از اعمال زیر را انجام می دهد :

- اگر رابطه Top و Lhs بصورت Top $\lessdot$ Lhs باشد پارسر ابتدا علامت $\lessdot$ وسپس Lhs را وارد انباره می کند.
- اگر رابطه Top و Lhs بصورت Top $=$ Lhs باشد پارسر فقط Lhs را وارد انباره می کند.
- اگر Top و Lhs رابطه ای نداشته باشند یک خطای نحوی رخ داده است و بایستی رویه اصلاح خطا فراخوانده شود.

۴. در صورتی که عنصر بالای انباره X و ورودی جاری b رابطه ای نداشته باشند یک خطای نحوی رخ داده است و بایستی رویه اصلاح خطا فراخوانده شود.

۵. در صورتی که توکن جاری $ و در داخل انباره $S ( S علامت شروع گرامر است ) باقی مانده باشد پارسر پایان موفقیت آمیز تجزیه را اعلام می کند.

---

## ۳ – ۱۶ – ۲ مشکلات چپ گردی و راست گردی در روش تقدم ساده

هرگاه در قواعد گرامر وضعیتی بصورت زیر باشد که در آن قاعده اول یک قاعده چپ گرد است بین علامت X و غیرپایانه U دو رابطه تقدم بصورت زیر وجود دارد:

$U \rightarrow U \ldots$
$V \rightarrow \ldots xU \ldots$
$X \doteq U \ , \ X \lessdot U$

برای حل این مشکل قواعد فوق را بصورت زیر تبدیل می کنیم که در آن W یک غیرپایانه جدید است :

$U \rightarrow U \ldots$
$V \rightarrow \ldots XW \ldots$
$W \rightarrow U$

حال روابط تقدم بین این علائم بصورت زیر است :

$X \doteq W \quad , \quad X \lessdot U$

همچنین هرگاه در قواعد گرامر وضعیتی بصورت زیر باشد که در آن قاعده اول یک قاعده راست گرد است بین غیرپایانه $U$ و علامت $X$ دو رابطه تقدم وجود دارد :

$U \rightarrow \ldots U$
$V \rightarrow \ldots UX \ldots$

برای رفع این مشکل قواعد را بصورت زیر تغییر می دهیم :

$U \rightarrow \ldots U$
$V \rightarrow \ldots WX \ldots$
$W \rightarrow U$

در ادامه به عنوان یک نمونه از تجزیه به روش تقدم ساده به تجزیه جمله (c(cc)) توجه کنید :

| عمل انجام شده | ورودی | انباره |
|---|---|---|
| انتقال | ( c ( c c ) ) $ | $ |
| انتقال | c ( c c ) ) $ | $ < ( |
| کاهش بوسیله $S \rightarrow c$ | ( c c ) ) $ | $ < ( < c |
| انتقال | (c c ) ) $ | $ < ( S |
| انتقال | c ) ) $ | $ < ( S < ( |
| کاهش بوسیله $S \rightarrow c$ | ) ) $ | $ < ( S < ( < c |
| انتقال | ) ) $ | $ < ( S < ( S |
| کاهش بوسیله $S \rightarrow c$ | ) ) $ | $ < ( S < ( S < c |
| انتقال | ) ) $ | $ < ( S < ( S S |
| کاهش بوسیله $S \rightarrow (SS)$ | ) $ | $ < ( S < ( S S |
| انتقال | ) $ | $ < ( S S |
| کاهش بوسیله $S \rightarrow (SS)$ | $ | $ < ( S S ) |
| پایان | $ | $ S |

## روشهای تجزیه LR :

یکی از امن ترین روشهای تجزیه پائین به بالا که می تواند در مورد اکثر گرامرهای مستقل از متن اعمال شود روش LR است.مزایای روشهای تجزیه LR عبارتند از :

۱- تقریبا تمامی ساختارهای زبان برنامه نویسی را می توان توسط پارسرهای LR تشخیص داد.

۲- روش تجزیه LR کلی ترین روش تجزیه غیر بازگشتی به طریقه انتقال – کاهش است که تا کنون شناخته شده و می توان آن را به کارایی هر روش دیگری پیاده سازی کرد.

۳- مجموعه زبانهایی که توسط روش LR تجزیه می شوند شامل مجموعه زبانهایی است که توسط پارسرهای پیشگو تجزیه می شوند.

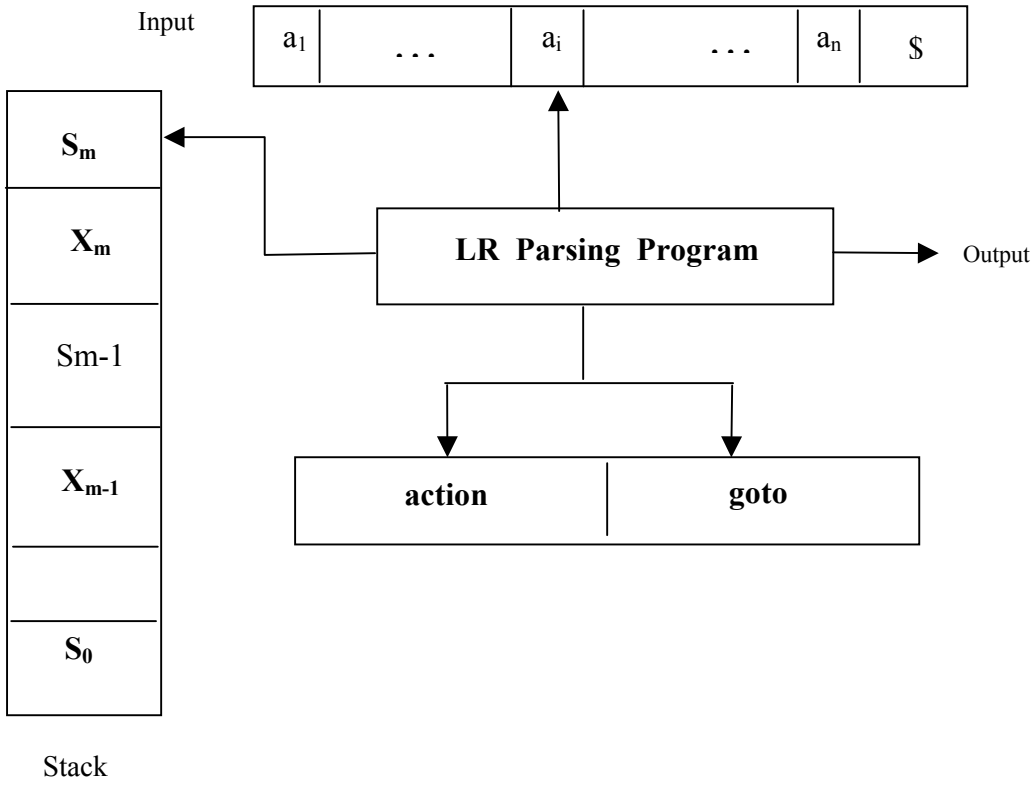
۴- یک پارسر LR خطای نحوی را در کمترین زمان توسط بررسی چپ به راست ورودی پیدا می کند.

تجزیه LR خود از سه روش زیر تشکیل می شود :

1. SLR ( Simple LR )
2. LALR ( Look Ahead LR )
3. CLR ( Canonical LR )

یکی از معایب روش LR آن است که حجم کار بسیار زیادی دارد که در مورد پیاده سازی دستی آن را مشکل می نماید ولی به هر حال نرم افزارهای خودکاری وجود دارند که می توانند به پارسرهای LR کمک کنند.مانند نرم افزار Yacc

ساختار پارسرهای LR به فرم زیر است :




## الگوریتم تجزیه LR :

اگر در حال حاضر پیکربندی ( $S_0X_1S_1X_2S_2 \ldots X_mS_m$ , $a_ia_{i+1} \ldots a_n\$$ )را داشته باشیم یعنی اینکه پارسر LR در قدمی از تجزیه قرار گرفته باشد که شماره وضعیت بالای پشته $S_m$ و توکن جاری $a_i$ باشد.بنابراین در این حال پارسر به خانه action [ $S_m$ , $a_i$ ] رجوع می کند اگر این خانه خالی نباشد یکی از سه حالت زیر رخ می دهد :

1. Shift 2. Reduce $A \rightarrow \alpha$ 3. Accept

۱- اگر action [ $S_m$ , $a_i$ ] = ShiftS باشد در اینصورت $a_i$ از رشته ورودی حذف شده و به Stack , Push می شود و سپس وضعیت S روی آن قرار می گیرد.

۲- اگر action [ $S_m$ , $a_i$ ] = Reduce $A \rightarrow \alpha$ باشد در آنصورت پارسر عمل کاهش را انجام می دهد و از پیکربندی

( $S_0X_1S_1X_2S_2 \ldots X_mS_m$, $a_ia_{i+1} \ldots a_n\$$ )

به پیکربندی ( $S_0X_1S_1X_2S_2 \ \dots \ X_{m-r}S_{m-r}\ AS$ , $a_ia_{i+1} \dots a_n\ \$$ ) دست خواهیم یافت که در آن :

$S = goto [ S_{m-r} , A ]$ , $r = |\alpha| \rightarrow$ طول $\alpha$

بنابراین تعداد حروف حذف شده از Stack به اندازه $2r$ خواهد بود که r به تعداد وضعیتها یا تعداد علائم گرامر A است.

۳- اگر $action [ S_m,a_i ] = accept$ باشد در اینصورت عمل پارس به شکل موفقیت آمیزی انجام پذیرفته است.

اگر $action [ S_m,a_i ]$ خالی باشد یک خطای نحوی رخ داده است که در این حال رویه خطاپرداز فراخوانی می شود.

قرارداد : منظور از $r_n$ در جدول پارس , به منظور عمل کاهش ( reduce ) توسط قانون شماره n است و منظور از $S_n$ یعنی انتقال یا Shift است که n شماره یک وضعیت می باشد.

مثال – گرامر زیر را در نظر بگیرید :

1 ) $E \rightarrow E + T$

2 ) $E \rightarrow T$

3 ) $T \rightarrow T * F$

4 ) $T \rightarrow F$

5 ) $F \rightarrow ( E )$

6 ) $F \rightarrow id$

| State | id | + | * | ( | ) | $ | E | T | F |
|---|---|---|---|---|---|---|---|---|---|
| 0 | $S_5$ | | | $S_4$ | | | 1 | 2 | 3 |
| 1 | | $S_6$ | | | | acc | | | |
| 2 | | $r_2$ | $S_7$ | | $r_2$ | $r_2$ | | | |
| 3 | | $r_4$ | $r_4$ | | $r_4$ | $r_4$ | | | |
| 4 | $S_5$ | | | $S_4$ | | | 8 | 2 | 3 |
| 5 | | $r_6$ | $r_6$ | | $r_6$ | $r_6$ | | | |
| 6 | $S_5$ | | | $S_4$ | | | | 9 | 3 |
| 7 | $S_5$ | | | $S_4$ | | | | | 10 |
| 8 | | $S_6$ | | | $S_{11}$ | | | | |
| 9 | | $r_1$ | $S_7$ | | $r_1$ | $r_1$ | | | |
| 10 | | $r_3$ | $r_3$ | | $r_3$ | $r_3$ | | | |
| 11 | | $r_5$ | $r_5$ | | $r_5$ | $r_5$ | | | |

| State | Input | Action |
|---|---|---|
| 0 | id*id+id $ | Shift |
| 0id5 | *id+id $ | Reduce by F → id |
| 0F3 | *id+id $ | Reduce by T → F |
| 0T2 | *id+id $ | Shift |
| 0T2*7 | id+id $ | Shift |
| 0T2*7id5 | +id $ | Reduce by F → id |
| 0T2*7F10 | +id $ | Reduce by T → T*F |
| 0T2 | +id $ | Reduce by E → T |
| 0E1 | +id $ | Shift |
| 0E1+6 | id $ | Shift |
| 0E1+6id5 | $ | Reduce by F → id |
| 0E1+6F3 | $ | Reduce by T → F |
| 0E1+6T9 | $ | E → E+T |
| 0E1 | $ | Accept |

---

## نحوه تهیه جدول SLR(1) :

در میان سه روش LR ساده ترین روش از نظر پیاده سازی روشSLR ( Simple LR) است.قبل از توضیح راجع به نحوه بدست آوردن جدول SLR(1) ذکر چند مفهوم زیر ضروری است :

۱. یک آیتم LR(0) یک قاعده از گرامر است که در محلی درسمت راست آن یک علامت خاص صفر یا point وجود داشته باشد.مثلا اگر قاعده ای به شکل زیر داشته باشیم A → xyz میتوان از روی آن 4 آیتم زیر را بدست آورد :

A → .xyz
A → x.yz
A → xy.z
A → xyz.

۲. اگر قاعده ε داشته باشیم یعنی A → ε تنها آیتم ما . → A است.

**تعریف ( I )Closure :** اگر I یک مجموعه از آیتم های یک گرامر باشد در آنصورت ( I )Closure را بستار ( I ) می نامیم که آن نیز یک مجموعه از آیتم هاست و برای محاسبه آن بایستی دو قدم زیر را پیمایش کرد :

۱- هر آیتمی که در I وجود دارد را به ( I ) Closure می افزائیم .

۲- اگر قاعده ای به شکل $A \rightarrow \alpha.B\beta$ در بستار I داشته باشیم و $B \rightarrow \delta$ یک قاعده از گرامر باشد آنگاه قاعده $B \rightarrow .\delta$ را به بستار I می افزائیم.

---

**رسم دیاگرام انتقال SLR :**

همانطوری که گفته شد برای تهیه جدول تجزیه روشهای LR بایستی یک دیاگرام انتقال رسم کرد.این امر در مورد روش SLR(1) به شکل زیر خواهد بود :

۱. ابتدا قاعده ای به شکل $S' \rightarrow .S$ که در آن S علامت شروع گرامر است و 'S یک nonterminal جدید است اضافه می کنیم.به این گرامر یک گرامر افزوده گوئیم.

۲. رسم دیاگرام را با وضعیت $I_0$ و آیتم $S' \rightarrow .S$ شروع می کنیم و سپس بستار این آیتم را محاسبه کرده و در $I_0$ قرار می دهیم.سپس درصورتیکه در حالت $I_0$ یا بطور کلی در حالت $I_i$ باشیم آیتم هایی که به شکل زیر اند که در آنها X می تواند یک پایانه یا غیرپایانه باشد وجود دارد.

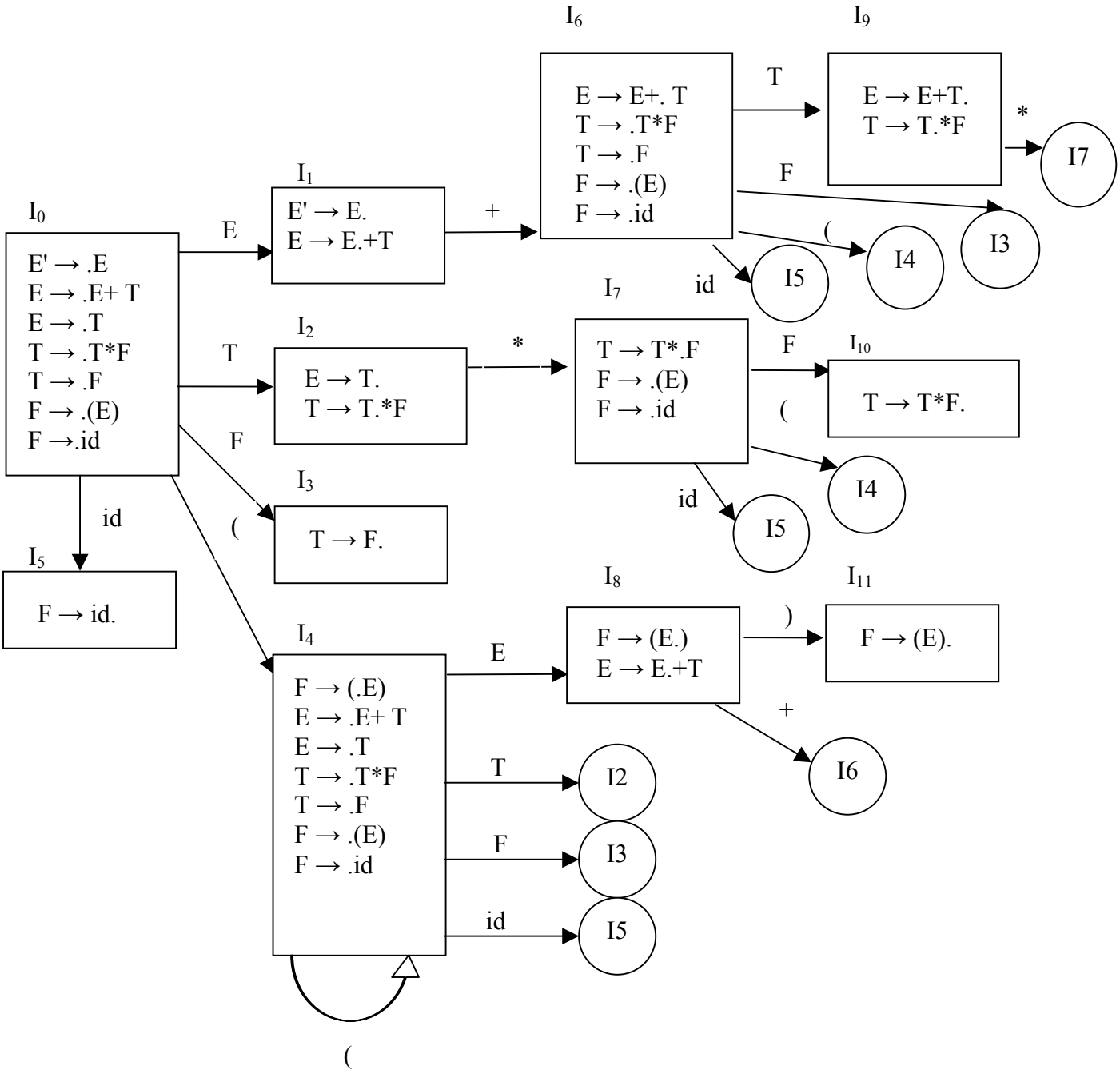
$A_1 \rightarrow \alpha_1.X\beta_1$
$A_2 \rightarrow \alpha_2.X\beta_2$
$A_n \rightarrow \alpha_n.X\beta_n$

با بوجود آمدن وضعیت جدید $I_i$ میتوان $I_i$ را توسط لبه هایی با برچسب X به $I_j$ متصل نمائیم و پس از اینکه در همه عبارات علامت نقطه را به بعد از علامت X منتقل کردیم در وضعیت جدیدی قرار خواهیم گرفت.بدین ترتیب آنقدر این عمل را ادامه می دهیم تا دیگر حالت جدیدی به دیاگرام اضافه نشود.

1 – 2 $E \rightarrow E + T \mid T$

3 – 4 $T \rightarrow T * F \mid F$

5 – 6 $F \rightarrow ( E ) \mid id$



## تهیه جدول تجزیه از روی دیاگرام SLR(1) :

پس از رسم دیاگرام SLR(1) یک گرامر جدول تجزیه آنرا بصورت زیر تکمیل می نمائیم :

۱. نحوه پر شدن بخش action :

۱-۱. اگر با ورودی a از وضعیت I به وضعیت J برویم در خانه action[I,a] دستور Shift J را قرار می دهیم.

۱-۲. اگر در حالت I آیتمی بفرم $A \rightarrow \alpha.$ داشته باشیم در اینصورت به ازای هر $a \in$ Follow(A) خانه یا شماره قاعده مذکور را در action[I,a] = reduce $A \rightarrow \alpha$ در جدول قرار می دهیم.

۱-۳. اگر در حالت I آیتمی بفرم $S' \rightarrow S.$ داشته باشیم در آنصورت در خانه action[I,$] علامت Accept قرار می دهیم.

۱-۴. در خانه های خالی بخش action علامتی را بعنوان خطا میتوان قرار داد.

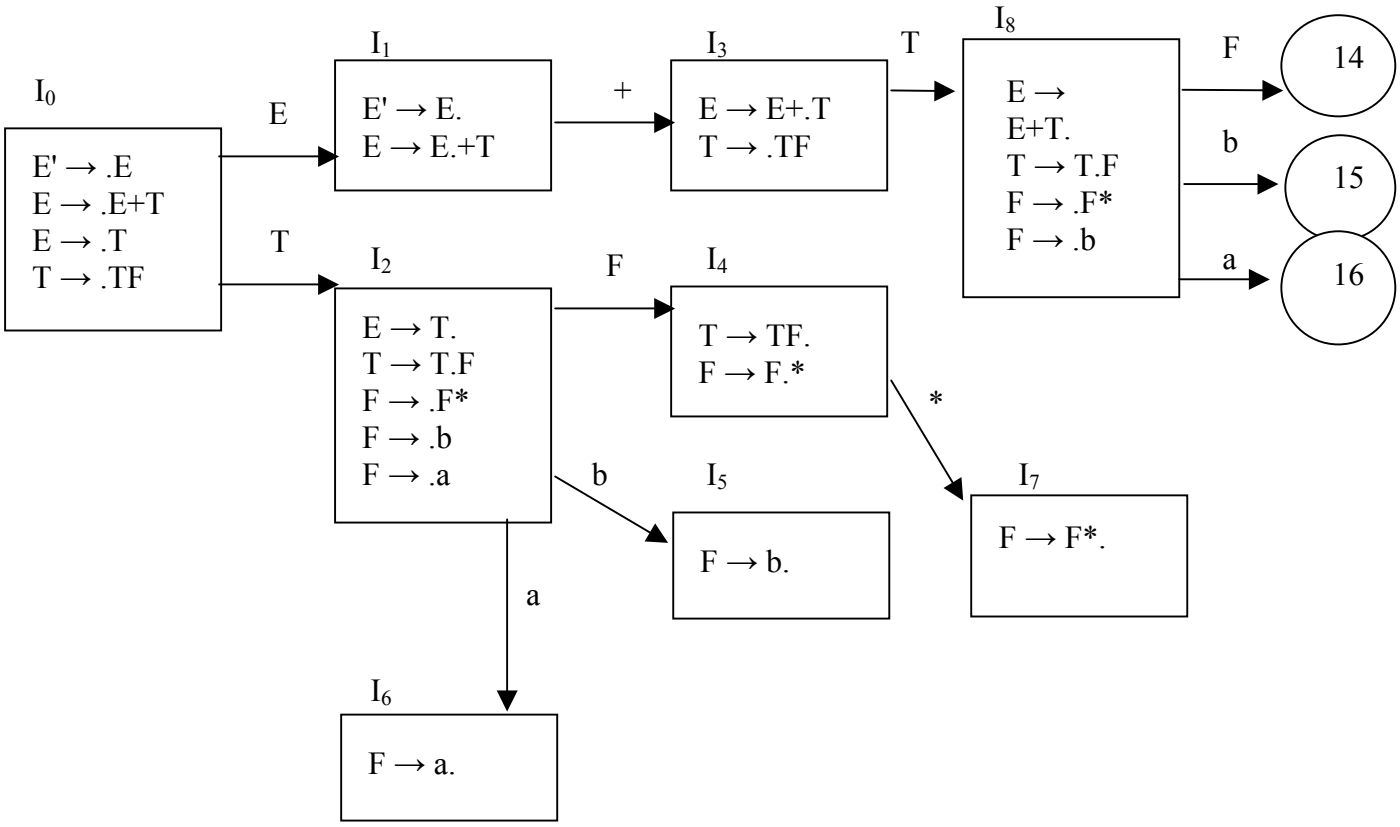
۲. نحوه پر شدن بخش goto :

برای پر کردن بخش goto از جدول تجزیه بدین ترتیب عمل می کنیم که اگر با غیرپایانه A از حالت I به حالت J برویم در خانه goto[I,A] مقدار J را قرار می دهیم یعنی goto[I,A] = J .

مثال :

---

$E \rightarrow E + T \mid T$
$T \rightarrow T\ F$
$F \rightarrow F * \mid a \mid b$



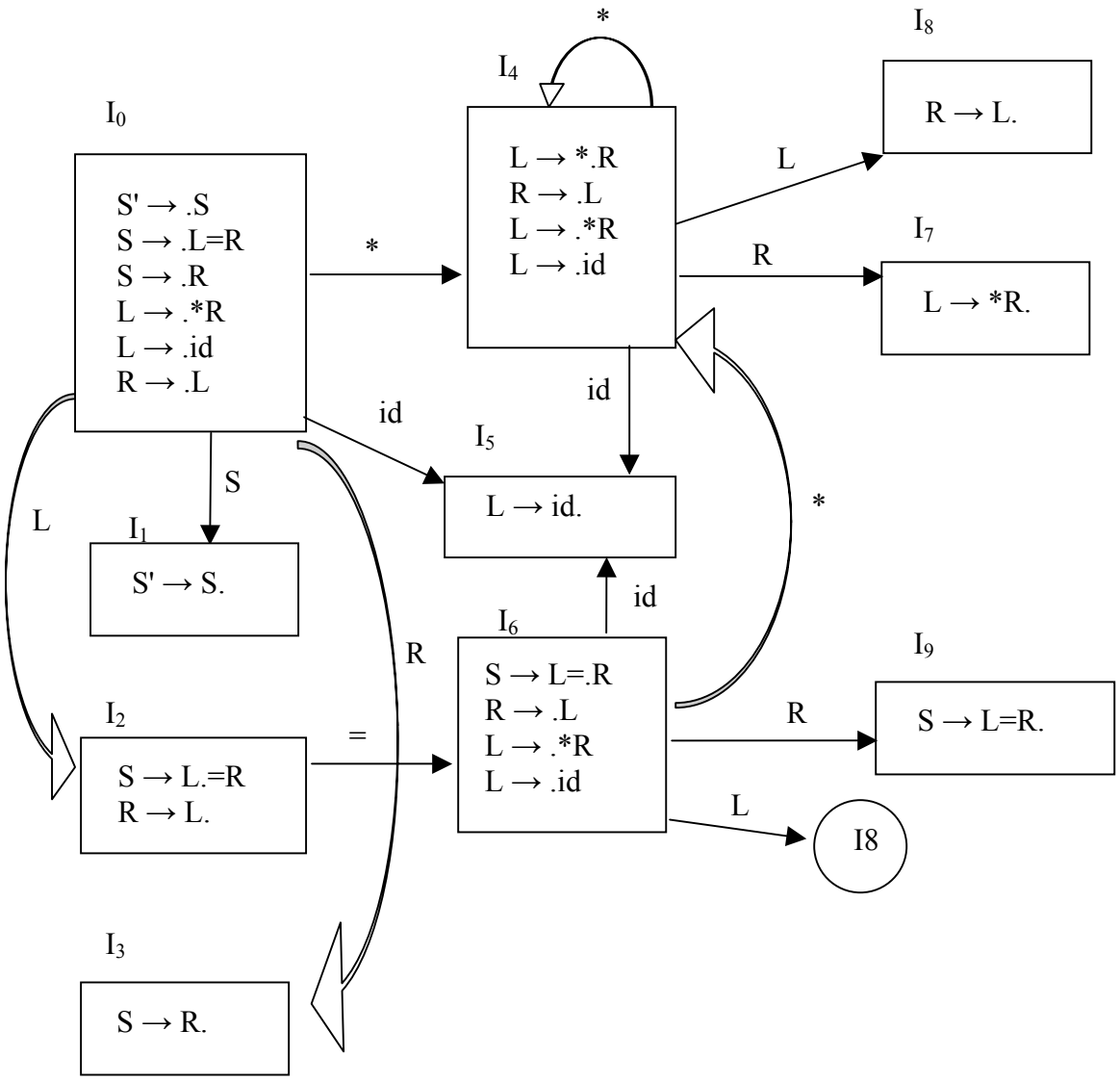

| | * | + | $ | a | b | E | T | F |
|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | 1 | 2 | |
| 1 | | $S_3$ | acc | | | | | |
| 2 | | $r_2$ | $r_2$ | $S_6$ | $S_5$ | | | 4 |
| 3 | | | | | | | 8 | |
| 4 | $S_7$ | $r_3$ | $r_3$ | $r_3$ | $r_3$ | | | |
| 5 | $r_6$ | $r_6$ | $r_6$ | $r_6$ | $r_6$ | | | |
| 6 | $r_5$ | $r_5$ | $r_5$ | $r_5$ | $r_5$ | | | |
| 7 | $r_4$ | $r_4$ | $r_4$ | $r_4$ | $r_4$ | | | |
| 8 | | $r_1$ | $r_1$ | $S_6$ | $S_5$ | | | 4 |

Follow(T) = { $ , + , a , b }
Follow(F) = { * , $ , + , a , b }
Follow(E) = { $ , + }

این گرامر SLR(1) است.

تمرین:

0 $S' \rightarrow .S$
1 $S \rightarrow L = R$
2 $S \rightarrow R$
3 $L \rightarrow *R$
4 $L \rightarrow id$
5 $R \rightarrow L$



| State | = | * | id | $ | L | R | S |
|---|---|---|---|---|---|---|---|
| 0 | | $S_4$ | $S_5$ | | 2 | 3 | 1 |
| 1 | | | | acc | | | |
| 2 | $S_6/r_5$ | | | $r_5$ | | | |
| 3 | | | | $r_2$ | | | |
| 4 | | $S_4$ | $S_5$ | | 8 | 7 | |
| 5 | $r_4$ | | | $r_4$ | | | |
| 6 | | $S_4$ | $S_5$ | | 8 | 9 | |
| 7 | $r_3$ | | | $r_3$ | | | |
| 8 | $r_5$ | | | $r_5$ | | | |
| 9 | | | | $r_1$ | | | |

این گرامر SLR(1) نیست.

---

## دیاگرام جدول تجزیه CLR و LALR :

یک روش برای تهیه دیاگرام LALR از طریق رسم دیاگرام CLR است.برای این منظور مفهوم آیتم های LR(1) را تعریف می کنیم.

یک آیتم LR(1) عبارتست از یک زوج مرتب متشکل از یک آیتم LR(0) و یک مجموعه از پایانه ها بنام مجموعه پیش بینی و معمولا این آیتم را به شکل کلی ( $A \rightarrow \alpha.B\beta$ , LA ) نمایش می دهیم.رابطه بین مجموعه پیش بینی LA و مجموعه Followی غیرپایانه های A به شکل زیر است :

$LA \in Follow(A)$

الگوریتم محاسبه تابع بستار در مورد الگوریتم تابعهای LR(1) دقیقا مشابه الگوریتم محاسبه بستار آیتم LR(0) است. در مورد آیتم $[ A \rightarrow \alpha . B \beta , \{a\} ]$ چنانچه $B \rightarrow S$ یک مجموعه از قاعده گرامر باشد در اینصورت مجموعه پیش بینی زیر را می افزائیم :

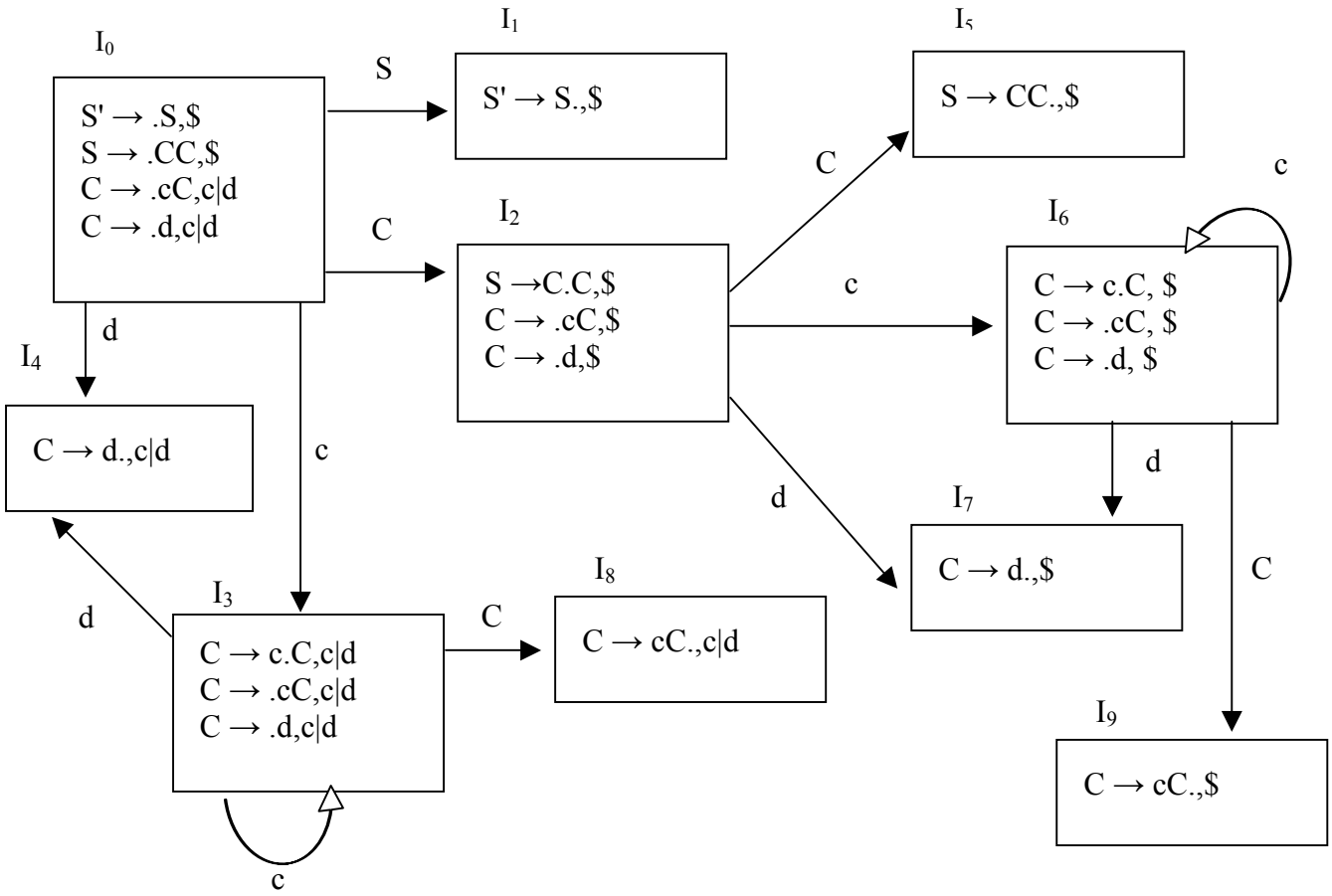
$[ B \rightarrow .S , \{b\} ]$ $\quad b \in First(\beta a)$

## طریقه رسم دیاگرام CLR :

رسم دیاگرام CLR دقیقا مشابه دیاگرام SLR است با این تفاوت که برای شروع قاعده [ S' → .S,$ ] را بعنوان آیتم LR(1) به مجموعه قواعد گرامر اضافه می کنیم.

مثال :

1 S → CC
2,3 C → cC | d
a=$ First(βa) = First($)
B=S {$}
α=ε S → CC
S'=A [ S → .CC,$ ] [C→.cC,c|d]
β=ε A → α.Bβ,a



## نحوه تشکیل جدول CLR(1) :

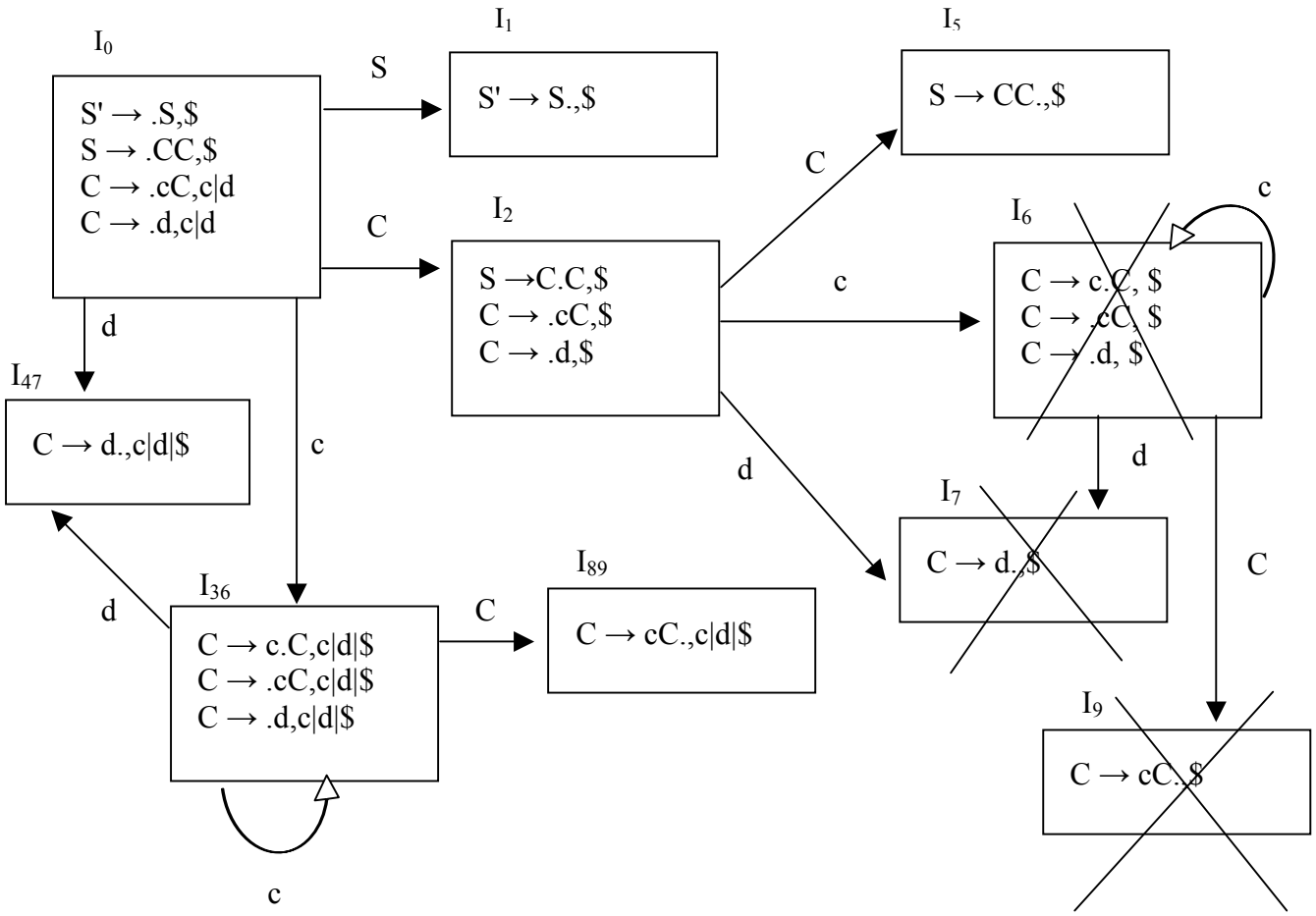
روش تهیه جدول CLR(1) دقیقا مطابق جدول SLR(1) است با این تفاوت که در اینجا در وضعیت I اگر آیتمی به فرم [ $A \rightarrow \alpha$ . , {b} ] داشته باشیم آنوقت در خانه های action[I,b] شماره دستور Reduce $A \rightarrow \alpha$ را قرار می دهیم یعنی در اینجا از Follow ها کمک نمی گیریم.

| State | c | d | $ | S | C |
|---|---|---|---|---|---|
| 0 | $S_3$ | $S_4$ | | 1 | 2 |
| 1 | | | acc | | |
| 2 | $S_6$ | $S_7$ | | | 5 |
| 3 | $S_3$ | $S_4$ | | | 8 |
| 4 | $r_3$ | $r_3$ | | | |
| 5 | | | $r_1$ | | |
| 6 | $S_6$ | $S_7$ | | | 9 |
| 7 | | | $r_3$ | | |
| 8 | $r_2$ | $r_2$ | | | |
| 9 | | | $r_2$ | | |

## رسم دیاگرام و جدول تجزیه LALR :

برای رسم دیاگرام LALR ابتدا دیاگرام CLR را ترسیم کرده و سپس وضعیتهای این دیاگرام را به شکل زیر ادغام می کنیم.بدین ترتیب که در وضعیتهای متفاوتی که آیتم های LR(0) آنها یکسان است و علامت پیش بینی آنها متفاوت است آنها را یکی فرض می نمائیم و بخش پیش بینی آنها را به شکل اجتماع پیش بینی های دو وضعیت در نظر می گیریم.

مثال – برای مثال قبل وضعیتهای 7,4 و 8,9 و 3,6 یکی می شوند.



برای مثال قبل تغییرات را اعمال می کنیم.

| | c | d | $ | S | C |
|---|---|---|---|---|---|
| 0 | $S_{36}$ | $S_{47}$ | | 1 | 2 |
| 1 | | | acc | | |
| 2 | $S_{36}$ | $S_{47}$ | | | 5 |
| 36 | $S_{36}$ | $S_{47}$ | | | 89 |
| 47 | $r_3$ | $r_3$ | $r_3$ | | |
| 5 | | | $r_1$ | | |
| 89 | $r_2$ | $r_2$ | $r_2$ | | |

اگر گرامری CLR باشد و پس از بدست آوردن جدول تجزیه LALR تداخلی بوجود نیاید میتوان نتیجه گرفت که گرامر LALR است.

اگر گرامری CLR باشد ولی LALR نباشد پس از ادغام در جدول تجزیه LALR از گرامر ممکن است تداخل نوع کاهش – کاهش رخ دهد.

توجه : در جدول LALR هیچ وقت تداخل انتقال – کاهش رخ نمی دهد مگر آنکه گرامر از نوع CLR نباشد.

---

تمرینی راجع به مباحث قبلی :

$S \rightarrow ( SS )$ Head(s) = { ( , c }

$S \rightarrow c$ Tail(s) = { ) , c }

$X = Y$ iff $\exists$ $U \rightarrow \ldots XY \ldots$

$X < Y$ iff $\exists$ $U \rightarrow \ldots XA \ldots$ and $Y \in Head(A)$

$X > Y$ iff $\exists$ $U \rightarrow \ldots AB \ldots$ and $X \in Tail(A)$ and $Y \in Head(B)$ or $Y = B$

| | S | $ | ( | ) | C |
|---|---|---|---|---|---|
| S | = | = | < | = | < |
| $ | = | | < | | < |
| ( | = | | < | | < |
| ) | | > | > | > | > |
| c | | > | > | > | > |

## فشرده سازی جداول تجزیه LR :

یک روش مفید به منظور فشرده سازی فیلد action تشخیص این نکته است که اغلب تعداد زیادی از سطر های جدول action مشابه هستند. بنا بر این در زمان کوتاه امکان صرفه جویی قابل توجه در حافظه وجود خواهد داشت. اشاره گر ها برای حالت هایی که action مشابه دارند، به مکان مشابه اشاره می کنند. به منظور دسترسی به اطلاعات این آرایه، به هر پایانه یک عدد بین صفر تا یک واحد کمتر از تعداد پایانه ها نسبت می دهیم، و این عدد صحیح را به عنوان تفاوت مکان از محلی که مقدار اشاره گر برای هر حالت مشخص می کند، مورد استفاده قرار می دهیم. برای یک حالت داده شده عمل action در تجزیه برای پایانه i ام، در i محل بعد از محل اشاره گر برای آن حالت، به دست می آید.

علاوه بر آن، کارایی بیشتر حافظه می تواند در مقابل تجزیه کند تر به دست آید. این عمل با ایجاد یک لیست برای actionهای هر حالت بدست می آید. ای لیست شامل زوج (نماد پایانه ، action) می باشد. متداول ترین action برای یک حالت می تواند در انتهای لیست قرار گیرد و به جای یک پایانه ممکن است کلمه ی any را قرار دهیم. به این معنی که اگر نماد ورودی جاری تا کنون در لیست یافت نشده است، این action باید بدون توجه به ورودی انجام شود.علاوه بر این وارده های خطا می توانند به منظور یکنواختی بیشتر در یک سطر با reduceجایگزین شوند. این خطا ها بعدا قبل از یک حرکت انتقال آشکار خواهند شد.

**مثال**– جدول تجزیه زیر را در نظر بگیرید:

.....

نخست به این نکته توجه نمایید که بخش action از حالت های 0,4,6,7 با یکدیگر مشابه می باشند. همه ی آن ها را می توان با لیستی به صورت زیر نمایش داد:

| نماد | عمل |
|---|---|
| id | S5 |
| ( | S4 |
| any | Error |

حالت ۱ نیز لیستی مشابه دارد:

| | |
|---|---|
| + | S6 |
| $ | Acc |
| any | Error |

در حالت ۲ تمامی وارده های خطا را می توان با r2 جایگزین نمود:

| | |
|---|---|
| * | S7 |
| any | R2 |

لیست حالت ۸ عبارت است از:

| | |
|---|---|
| + | S6 |
| ) | S11 |
| any | Error |

و برای حالت ۹ به صورت:

| | |
|---|---|
| * | S7 |
| any | R1 |

جدول goto را نیز می توان با استفاده از یک لیست کد گذاری نمود، اما به نظر می رسد ساخت یک لیست از زوج ها برای هر غیر پایانه مانند A کارایی بیشتری داشته باشد. هر زوج در این لیست برای غیر پایانه A به شکل (حالت بعدی ، حالت جاری) می باشد که نشان دهنده: حالت بعدی=[ A ، حالت جاری]goto

می باشد.

به منظور کاهش بیشتر فضا به این نکته توجه داشته باشید که به وارده های خطا در بخش goto از جدول هرگز مراجعه نمی شود. به این ترتیب هر وارده خطا را می توان با متداول ترین حالت غیر خطا در همان ستون جایگزین نمود.

**مثال**- جدول تجزیه مثال قبل را در نظر بگیرید. ستون مر بوط به F برای حالت ۷ مقدار ۱۰ را دارد، و تمام وارده های دیگر مقدار ۳ یا خطا دارند، می توان خطا را با مقدار ۳ جایگزین کرد و برای ستون F لیستی به صورت زیر ایجاد کرد:

| حالت جاری | حالت بعدی |
|---|---|
| ۷ | ۱۰ |
| any | ۳ |

به طور مشابه برای ستون T داریم:

| | |
|---|---|
| ۶ | ۹ |
| any | ۲ |

و برای ستون E :

| | |
|---|---|
| ۴ | ۸ |
| any | ۱ |

## استفاده از اولویت و شرکت پذیری به منظور رفع تناقض های بخش action :

گرامر زیر برای عبارت های محاسباتی، با عملگر های + ، * مبهم است زیرا شرکت پذیری یا اولویت *,+ را مشخص نمی کند :

E **** E+E | E*E |(E) | id

گرامر غیر مبهم زیر همان زمان را تولید میکند اما به + اولویت کمتری از * می دهد و هر دو عملگر را با شر کت پذیری چپ تعریف می کند.

E E+T |T
T T*F | F
F (E) | id

دو دلیل وجود دارد که از گرامر اولی بجای دومی استفاده می شود:

۱- تجزیه کننده گرامر دوم زمان زیادی را صرف کاهش با مولد های E T , T F می نماید.

۲- به راحتی می توان شر کت پذیری و اولویت را برای عملگر های *,+ بدون اضافه کردن تعداد حا لت ها در گرامر تعریف کرد.

شکل زیر مجموعه Itemهای LR(0) را برای گرامر

E **** E+E | E*E |(E) | id

که E` E به آن اضافه شده نشان می دهد.

.....

**I0 :**

E` .E
E .E+E
E .E*E
E .(E)
E .id

**I1:**

E` E.
E E.+E
E E.*E

**I2:**

E (.E)
E .E+E
E .E*E
E .(E)
E .id

**I3:**

E id.

**I4:**

E E+.E
E .E+E
E .E*E
E .(E)
E .id

**I5:**

E E* .E
E .E+E
E .E *E
E .(E)
E .id

**I6:**

E (E.)
E E.+E
E E.*E

**I7:**

E E+E.
E E.+E
E E.*E

**I8:**

E E*E.
E E.+E
E E.*E

**I6:**

E (E).

از آن جایی که این گرامر مبهم است، هنگام تولید جدول تجزیه LR تناقض هایی در بخش action ایجاد خواهد شد. حالت هایی مشابه I7 وI8 این تناقض ها را تولید می

کنند. اگر از روش SLR برای ایجاد جدول تجزیه استفاده کنیم، تناقض ایجاد شده توسط I7 بین کاهش با E E+E و انتقال با + و* قابل حل نیست، زیرا + و* هر دو در follow(E) قرار دارند. بنابر این هر دوی این action ها در صورت در یافت ورودی های +و* فرا خوانی خواهند شد.

برای مثال ورودی id+id*id را در نظر بگیرید.در نهایت به حالتی می رسیم که در پشته 0E1+4E7 و در ورودی $id* داریم. فرض کنید * نسبت به جمع اولویت داشته باشد، تجزیه کننده باید * را به پشته منتقل کند و برای کاهش با * و idهای طرفین آن آماده شود. این همان چیزی است که تجزیه کننده SLR انجام می دهد، و همان چیزی است که یک تجزیه کننده عملگر- اولویت انجام می دهد. اگر + اولویتی بالا تر از * داشته باشد، تجزیه کننده E+E را به E کاهش خواهد داد. بنا بر این اولویت نسبی جمع که بعد از ضرب می باشد، به طور منحصر به فرد مشخص می سازد که چگونه تناقض action در تجزیه بین کاهش با E E+E و انتقال با * در حالت ۷ برطرف خواهد شد.

اگر به جای ورودی قبلی مقدار id+id+id به عنوان ورودی باشد، پس از پردازش بخش id+id تجزیه کننده به وضعیتی خواهد رسید که در آن محتوای پشته 0E1+4E7 خواهد بود. با ورودی + مجددا در حالت I7 تناقض انتقال – کاهش وجود خواهد داشت. به هر حال شرکت پذیری عملگر+ مشخص خواهد کرد که این تناقض چگونه باید حل شود. اگر + شرکت پذیری چپ داشته باشد action صحیح، کاهش E+E بهE خواهد بود.

فرض کنید + شرکت پذیری چپ داشته باشد، در حالت ۷، action با ورودی + باید کاهش با E E+E باشد، و فرض کنید که * اولویت بیشتری نسبت به جمع داشته باشد، actionدر حالت ۷ با ورودی * انتقال خواهد بود.

به طور مشابه فرض کنید * شرکت پذیری چپ دارد و اولویت آن بالا تر از+ می باشد، این نکته قابل بحث است که حالت ۸ در بخش action ، باید عمل کاهش با E E*E را به ازای هر دو ورودی + و* انجام دهد و این که این حالت فقط زمانی در بالای پشته قرار می گیرد که E*E سه نماد بالای پشته باشند. در رابطه با+ ، دلیل آن این است که * اولویتی بالاتر از + دارد، در حالی که در مورد* علت شرکت پذیری چپ * می باشد.

با ادامه این روش جدول تجزیه LR به شکل زیر به دست می آید:

| حالت | id | + | * | ( | ) | $ | E |
|---|---|---|---|---|---|---|---|
| 0 | S3 | | | S2 | | | 1 |
| 1 | S4 | S5 | | | | acc | |
| 2 | S3 | | | S2 | | | 6 |
| 3 | | R4 | R4 | | R4 | R4 | |
| 4 | S3 | | | S2 | | | 8 |
| 5 | S3 | | | S2 | | | 8 |
| 6 | | S4 | S5 | | S9 | | |
| 7 | | R1 | S5 | | R1 | R1 | |
| 8 | | R2 | R2 | | R2 | R2 | |
| 9 | | R3 | R3 | | R3 | R3 | |

## جدول های نماد

کامپایلر از جدول نماد برای نگهداری اطلاعات و محدوده تعریف نام های برنامه استفاده می نماید . هر زمان نامی در متن برنامه یافت می شود ، جدول نماد جستجو می گردد . هنگامی که نام جدیدی یا اطلاعات جدیدی در رابطه با نام های موجود در جدول یافت شود ، جدول تغییر می نماید .

مکانزیم جدول نماد باید این امکان را فراهم سازد که اضافه نمودن وارده های جدید و جستجوی وارده های موجود به صورت کارآمد انجام گیرد . دو مکانیزم جدول نماد که در این بخش ارائه شده است ، لیست های خطی و جدول در هم می باشند . هر یک از این طرح ها بر مبنای زمان لازم جهت افزودن n وارده و انجام e در خواست بررسی می گردند .لیست خطی آسانترین آن ها برای ساخت می باشد . اما زمانی که n وe به سمت مقادیر بزرگ میل کنند کارایی کاهش می یابد . طرح های درهم کارایی بهتری را با فعالیت برنامه نویسی و صرف فضای بیشتر ارائه میدهند . هر دو روش می توانند به گونه ای سازگار شوند که بتوانند اکثر قوانین محدوده برنامه را اداره نمایند .

برای کامپایلرمفید است که بتواند در صورت لزوم در زمان کامپایل اندازه جدول نماد را به صورت پویا افزایش دهد . اگر در زمان نوشتن کامپایلر ، اندازه جدول نماد ثابت باشد ، اندازه آن باید به اندازه کافی بزرگ انتخاب شود تا بتواند هر برنامه مبدا ممکن را اداره نماید . این چنین اندازه ثابتی برای اکثر برنامه ها بسیار بزرگ و برای بعضی نامناسب است.

## وارده های جدول نماد

هر وارده در جدول نماد ، برای اعلان نام است . قالب یا شکل وارده ها لزومی ندارد که یکنواخت باشد ، زیرا اطلاعات ذخیره شده برای نام ، به استفاده ازآن نام بستگی دارد . هر وارده می تواند به عنوان رکوردی که شامل دنباله ای از کلمات متوالی حافظه است ، پیاده سازی شود . به منظور حفظ یکنواختی رکوردهای جدول نماد ، بعضی از اطلاعات مربوط به نام مناسب تر است که درخارج ازاین وارده جدول نگهداری شوند و تنها یک اشاره گر به اطلاعات ذخیره شده در این رکورد ، قرار داده شود .

اطلاعات در زمان های متفاوتی به جدول نماد وارد می شوند . کلمات کلیدی در ابتدا به جدول وارد می شوند . تحلیل گر لغوی ، دنباله ای از حروف و ارقام را درجدول نماد جستجو می نماید تا مشخص شود که یک کلمه کلیدی و یا نام تشخیص داده شده است . دراین روش ، کلمات کلیدی قبل از اینکه تحلیل گر لغوی کار خود را آغاز نماید ، باید در جدول وجود داشته باشند . در مقابل ، اگر تحلیل گر لغوی در ضمن کار کلمات کلیدی رزرو شده را تشخیص می دهد ، نیازی به حضورآنها در جدول نماد نمی باشد . اگر زبان کلمات کلیدی را رزرو نکند، لازم است که کلمات کلیدی به جدول وارد شوند با این هشدار که امکان استفاده آنها به عنوان کلمه کلیدی وجود دارد .

هنگامی که نقش یک نام روشن شود ، وارده ای از جدول نماد می تواند به آن اختصاص داده شود و به محض اینکه اطلاعات مربوط به آن در دسترس قرار گرفت مقادیر صفت

آن وارده در جدول تکمیل می گردد .در برخی از موارد ، به محض اینکه تحلیل گر لغوی نام را در ورودی دید وارده ای برای آن در جدول نماد اختصاص می دهد اغلب اوقات ، یک نام می تواند مشخص کننده چندین شیء باشد حتی در یک بلوک یارویه . برای مثال اعلان های C به صورت :

(۱- الف)

```
Int x;
Struct x { float y , z ;}
```

x را هم به عنوان متغیر وهم به عنوان یک ساختار با دو فیلد معرفی می نماید . درچنین مواردی ، تحلیل گر لغوی می تواند بجای اشاره گری به وارده جدول نماد ( یا یک اشاره گر به لغت تشکیل دهنده آن نام ) فقط نام را به تجزیه کننده برگرداند . رکورد جدول نماد ، هنگامی که ایجاد می شود که نقش نحوی این نام روشن گردد . برای اعلان های (۱- الف ) دو وارده در جدول نماد برای x ایجاد خواهد شد . یکی با x به عنوان یک عدد صحیح و دیگری به عنوان یک ساختار .

صفات در پاسخ به اعلان ها که ممکن است ضمنی باشند ، به جدول وارد می شوند . برچسب ها ، اغلب شناسه هایی هستند که به دنبال آنها دو نقطه (:) قرار می گیرد ، بنابراین پس از تشخیص چنین شناسه ای باید اطلاع مربوط به بر چسب بودن آن را در وارده مربوط درجدول نماد وارد کرد . بطور مشابه ، ساختار نحوی اعلان رویه ها مشخص می نماید که بعضی شناسه ها از نوع پارامترهای رسمی هستند .

## کاراکترهای موجود در یک نام

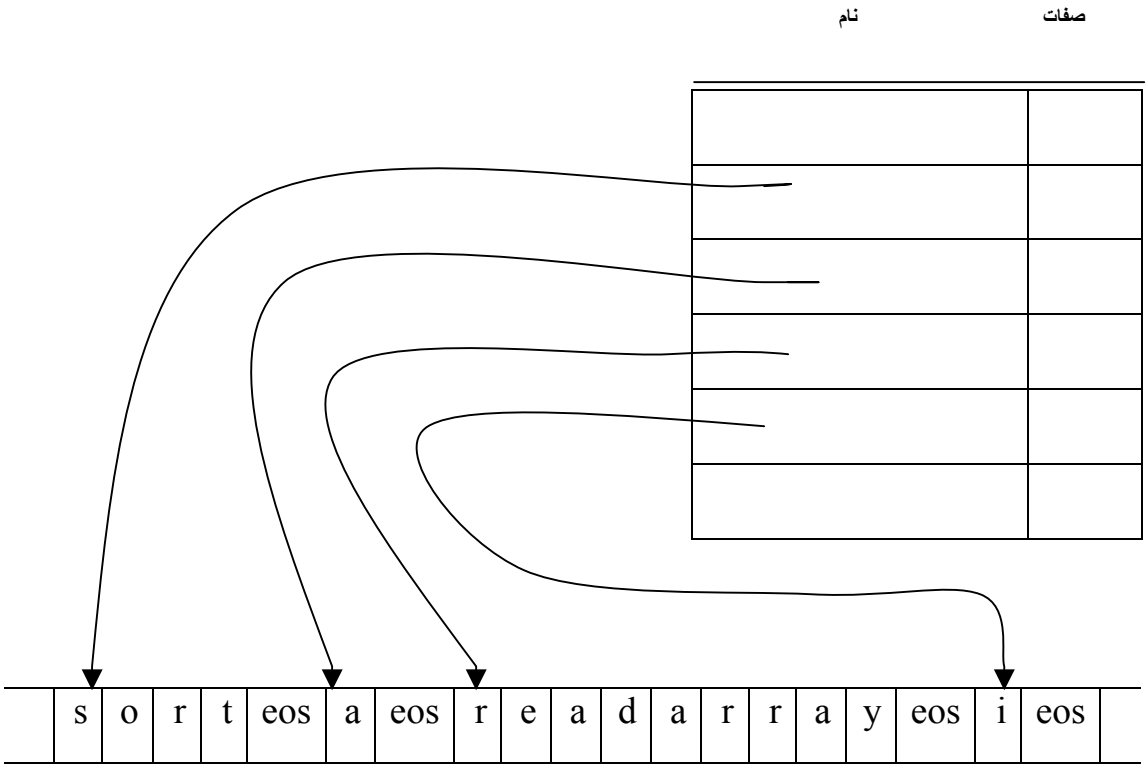
بین نشانه id برای یک شناسه یا نام ، لغت شامل رشته کاراکترهای تشکیل دهنده نام و صفات این نام تفاوت وجود دارد . بطور گسترده ای ممکن است با رشته هایی از کاراکترها کار نشود ، بنابراین کامپایلر نمایشی با طول ثابت از آن نام را بجای لغت آن استفاده می نماید . این لغت زمانی مورد نیازاست که وارده جدول نماد برای اولین بار وارد می شود . همچنین هنگامی که برای لغت یافت شده در ورودی به جدول مراجعه می شود تا مشخص شود آیا این نام قبلا وجود داشته است یا خیر. نمایشی مرسوم از نام ، اشاره گری است به وارد ه جدول نماد مربوط به آن . اگر حد بالای متوسطی برای طول نام وجود داشته باشد ، کاراکترهای موجود در نام می توانند در وارده جدول نماد مشابه زیر ذخیره شوند :

صفات

نام

| | |
|---|---|
| s o r t | |
| a | |
| r e a d a r r a y | |
| i | |
| | |

اگر محدودیتی برای طول نام وجود نداشته باشد  یا اگر این حد بندرت قابل حصول باشد طرح غیر مستقیم شکل زیر می تواند استفاده شود :




بجای تخصیص حداکثر فضای ممکن برای نگهداری یک لغت در هر وارده جدول نماد ، اگر فقط فضا برای اشاره گری در وارد ه جدول نماد اختصاص داده شود ، می تواند از فضا بطور کارآمدی استفاده نماید . در رکود یک نام ، اشاره گری به آرایه ای از کاراکترها (جدول رشته ) قرار داده می شود که موقعیت اولین کاراکتر آن لغت را مشخص می نماید . طرح غیرمستقیم اجازه می دهد اندازه فیلد نام وارده جدول نماد ثابت باقی بماند .

برای هر نام ، لغت کامل تشکیل دهنده آن باید ذخیره شود تا اطمینان حاصل گردد که تمام کاربردهای آن نام به یک رکورد از جدول نماد مرتبط می شوند . به هر حال رخدادهای لغت مشابه که در محدوده های اعلان های متفاوت قرار دارند باید قابل تفکیک باشند .

## اطلاعات تخصیص حافظه

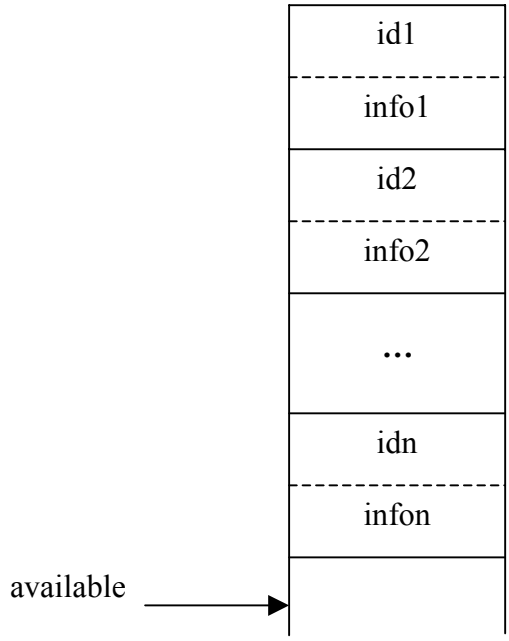
اطلاعات مربوط به مکان های حافظه که در زمان اجرا به نام ها اختصاص خواهد یافت ، در جدول نماد نگهداری می شود . ابتدا نام هایی را با حافظه ایستا در نظر بگیرید . اگر کد هدف زبان اسمبلی باشد ، اسمبلر این اجازه را خواهد داشت که در مورد مکان های حافظه برای نام های گوناگون ، مر اقبت لازم را انجام دهد . تمام آن چیزی که باید پس از تولید کد اسمبلی برای برنامه انجام گیرد ، پویش جدول نماد و تولید تعاریف داده های زبان اسمبلی نامی می باشد که باید به برنامه اضافه شود .

اگر کد ماشین را کامپایلر باید تولید کند موقعیت هر داده مقصود نسبت به یک مبدا ثابت مانند ابتدای یک رکورد فعالیت ، باید مشخص شود . توضیحات مشابهی در مورد یک بلوک ازداده ها که به عنوان یک پیمانه مجزا از برنامه بار می شود ، می تواند بکار رود. برای مثال بلوک های COMMON در فرتن به صورت مجزا بار می شوند و موقعیت نام ها نسبت به ابتدای بلوکCOMMON در فرترن به صورت مجزا بار می شوند و موقعیت نام ها نسبت به ابتدای بلوک COMMONکه درآن قرار دارند ، باید مشخص شود .

در رابطه با نام هایی که حافظه آنها در پشته یا کپه اختصاص یافته ، به هیچ وجه کامپایلر حافظه را اختصاص نمی دهد .

## ساختمان داده لیست برای جدول های نماد

ساده ترین ساختمان داده برای ساخت جدول نماد ، لیست خطی از رکورد ها می باشد ، که در شکل زیر نشان داده شده است :




از یک آرایه یا چندین آرایه معادل ، برای ذخیره نام ها و اطلاعات مرتبط با آنها استفاده می شود . نام های جدید به همان تر تیی که یافت می شوند ، به لیست اضافه می گردند موقعیت انتهای آرایه با اشاره گر available علامت گذاری می شود ، که به محلی اشاره می کند که وارده بعدی باید در جدول نماد قرار گیرد . جستجو برای نام از انتهای آرایه تا ابتدا به صورت معکوس انجام می گیرد . هنگامی که نام مورد نظر یافت شود ، اطلاعات

مرتبط با آن می تواند در کلمات بعد ازآن یافت شود . اگر بدون یافتن نام به ابتای آرایه برسد ، خطایی رخ می دهد مبنی براینکه نام مورد نظر در جدول وجود ندارد .

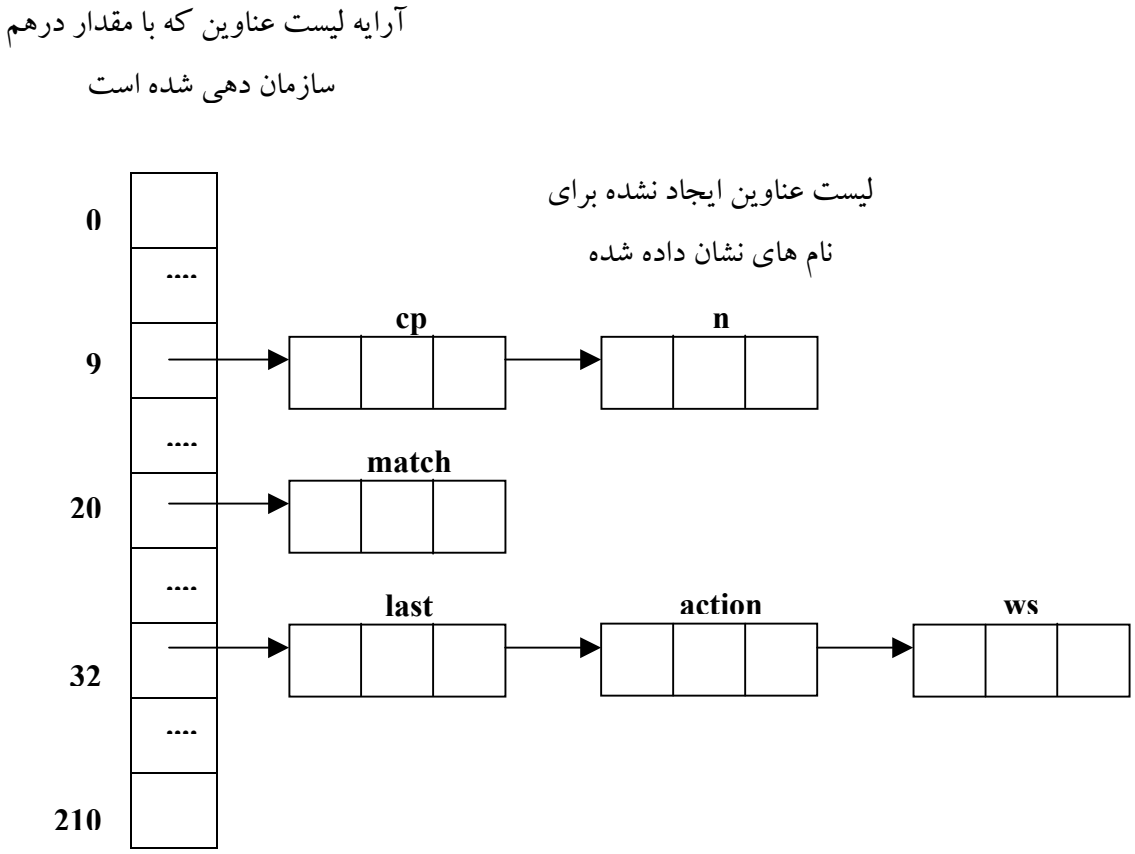
توجه داشته باشید که ایجاد یک وارده برای نام و یافتن نام در جدول نماد ، عملیات مستقلی هستند ، یعنی می توان یکی ازآنها را بدون دیگری انجام داد. در زبانی با ساختار بلوکی ، رخدادی از نام در محدوده نزدیکترین اعلان متداخل آن نام است . این قانون محدوده را می توا ن با استفاده از ساختمان داده لیست و با ایجاد یک وارده جدیدی برای نام ، هر زمانی که اعلان می شود ، پیاده سازی کرد . وارده جدید در کلمات بلافاصله بعد از اشاره گرد available قرار داده می شود و این اشاره گر به مقدار اندازه رکود جدول نماد افزایش می یابد . چون ورودی ها با شروع از ابتدای آرایه به ترتیب درج می شوند ، آنها به ترتیبی که اعلان شده اند قرار دارند . با جستجو ازمحل available و حرکت به سمت ابتدای آرایه ، مطمئن خواهیم بود که جدیدترین وارده ایجاد شده را خواهیم یافت . اگر جدول نماد داری n نام باشد ، میزان کار لازم برای درج کردن یک نام جدید ثابت خواهد بود چنانچه بررسی برای وجود آن نام از قبل در جدول انجام نگیرد . اگر چندین ورودی برای نام ها مجاز نباشد ، برای مشخص شدن عدم وجود نام در جدول ، تمام جدول باید جستجو شود ، در این فرایند میزان کار انجام شده متناسب با n است . به منظور یافتن داده های مربوط به یک نام ، به طور متوسط n/2 از نام ها باید جستجو شود ، بنابراین هزینه درخواست نیز متناسب با n است . به این ترتیب با توجه به اینکه درج کردن

و درخواست ، زمانی متناسب با n نیاز دارند ،کل کار انجام شده برای درج کردن n نام و انجام e درخواست حداکثر cn(n+e) می باشد که c ثابتی است نشان دهنده زمان لازم برای انجام چند دستور زبان ماشین . در یک برنامه با اندازه متوسط ، ممکن است e=1000 و n=100 بنابراین چند صد هزار دستور ماشین در فرایند نگهداری اطلاعات صرف می شود . این مقدار ممکن است نگران کننده نباشد ، زیرا در مورد زمان کمتر از یک ثانیه صحبت می شود . درهر صورت ، اگر e و n در 10 ضرب شوند ، هزینه در 100 ضرب خواهد شد . و زمان نگهداری اطلاعات عاملی باز دارنده خواهد بود . محاسبه زمان اجرا ، اطلاعات زیادی را در رابطه با اینکه کامپایلر زمان را در کجا صرف می نماید ، در اختیار قرار می دهد و می تواند به منظور تشخیص اینکه آیا زمان بیش از حد برای جستجو درلیست های خطی تلف می شود استفاده گردد .

## جداول درهم

روش متفاوتی در جستجو به نام جستجوی در هم در بسیاری از کامپایلرها بکار گرفته شده است . در اینجا حالت ساده ای به نام درهم باز بررسی می گردد که « باز » اشاره به این ویژگی دارد که محدودیتی برای تعداد ورودی هایی که می توانند در جدول قرار گیرند وجود ندارد . حتی این طرح نیز قابلیت انجام e درخواست با n نام را در زمانی متناسب با n(n+e)/m با هر ثابت دلخواه m را فراهم می سازد . با توجه به این که m می تواند به هر اندازه دلخواه بزرگ باشد ، حداکثر تا n این روش بسیار کاراتر از لیست های خطی است

و روش انتخاب شده ساخت جداول نماد در اکثر موارد است . همانطور که انتظار می رود فضای اشغال شده توسط این ساختمان داده ، با m افزایش می یابد ، بنابراین تعادلی بین زمان وفضا ایجاد شده است . طرح اصلی جستجوی در هم در شکل زیر نمایش داده شده است :




این ساختمان داده دارای دو بخش است :

۱-جدول در هم شامل آرایه ثابت با m اشاره گر به وارده های جدول .

۲- ورودی های جدول به صورت m لیست پیوندی مجزا به نام buckets سازمان دهی شده اند (بعضی از buckets ها ممکن است خالی باشند ) . هر رکورد جدول نماد دقیقا در یکی از این لیست ها ظاهر می شود . حافظه برای رکوردها می تواند ازآرایه ای از رکورد

ها داده شود ، همانطور که در بخش بعد بحث شده است . از طرف دیگر امکانات تخصیص حافظه پویای زبان پیاده سازی کننده ، می تواند برای گرفتن فضابرای رکوردها استفاده شود ، و اغلب با کاهش کارایی همراه است .

به منظور تشخیص وجود یک وارده در جدول نماد برای رشته s ، یک تابع در هم از h به s استفاده می شود که h(s) عدد بین 0 تا m-1 را بر می گرداند. اگر s در جدول نماد وجود داشته باشد ، در فهرستی با شماره h(s) قرار دارد . اگر s هنوز در جدول وجود ندارد ، با ایجاد یک رکورد برای s یعنی قرار دادن در جلوی فهرست شماره h(s) به جدول اضافه می شود .

با یک حساب سرانگشتی اگر n نام ، در جدولی با اندازه m وجود داشته باشد ، بطور متوسط فهرست دارای طولی برابر n/m رکورد می باشد . با انتخاب m به شکلی که n/m به یک ثابت کوچک محدود شود ،مثلا 2 ، زمان لازم برای دسترسی به یک ورودی از جدول ضرورتا ثابت است .

فضای اشغال شده توسط جدول نماد شامل m کلمه برای جدول درهم و cn کلمه برای وارده های جدول است که c تعداد کلمات برای هر وارده جدول است . بنابراین فضای لازم برای جدول در هم فقط به m بستگی دارد ، و فضای لازم برای وارده جدول ، فقط به تعداد وارده ها وابسته است .

انتخاب m به کاربرد مورد نظر برای جدول نماد وابسته است . با انتخاب مقدار چند صد برای m زمان مراجعه به جدول به اندازه کسری ناچیز از کل زمان مصرف شده توسط کامپایلر خواهد بود حتی برای برنامه هایی با اندازه متوسط. هنگامی که ورودی کامپایلر توسط برنامه دیگری تولید می شود ، تعداد نام ها نسبت به اکثر برنامه های تولید شده توسط انسان با همان اندازه افزایش چشمگیری خواهد داشت . بنابراین جدول با اندازه بزرگتر تر جیح دارد .این سوال مورد توجه است که چگونه تابع در هم طراحی شود که به سادگی برای رشته هایی از کاراکترها قابل محاسبه باشد و رشته ها را بطور یکنواخت در میان m لیست توزیع نماید. یک روش مناسب برای محاسبه توابع در هم به صورت زیر است :

۱- یک عددصحیح مثبت مانند h از کاراترهای $C_k,\ldots C_2,C_1$ در رشته s مشخص می گردد . تبدیل یک کاراکتر به عدد صحیح معمولا توسط زبان پیاده سازی کننده پشتیبانی می گردد . پاسکال تابعی به نام ord رابرای این منظور فراهم نموده است . c بطور خود کار یک کاراکتر را به عدد صحیح تبدیل می نماید اگر یک عمل محاسباتی بر روی آن انجام گیرد .

۲- عدد صحیح h که در مرحله قبل بدست آمده به شماره لیست تبدیل می گردد ، یعنی عدد صحیحی بین 0 و m-1 . فقط با تقسیم بر m و استفاده از باقیمانده آن روشی معقول

است . اگر m عدد اول باشد ، استفاده از باقیمانده به نظر بهتر عمل می کند بنابراین در جدول درهم مثال زده شده انتخاب 211 به جای 200 انجام می شود .

توابع در هم که به تمام کاراکترهای رشته توجه دارند ، کمتر از توابعی که فقط به تعدادی از کاراکترهای انتها یا وسط رشته توجه دارند دچار خطا می شوند .بخاطر آورید ورودی به کامپایلر ممکن است توسط یک برنامه ایجاد شود و بنابراین به منظور جلوگیری از تناقض با نام های استفاده شده توسط اشخاص یا برنامه دیگر ممکن است دارای شکل خاص باشند اشخاص تمایل به دسته بندی نام ها دارند مانند انتخاب های :

baz1,new baz,baz و مانند آن .

یک روش ساده برای محاسبه h، جمع نمودن مقادیر صحیح کاراکترهای رشته است . ایده بهتر ، ضرب مقدار قبلی h با ثابت a قبل از جمع با کاراکتر بعدی است . یعنی

$h_i=ah_{i-1}+c_i$ , $h_0=0$ برای $h = h_k, 1 \le i \le k$ که k طول رشته است . (بخاطر آورید ، مقدار درهم که شماره لیست را مشخص می نماید h mod m است ) تنها جمع کاراکترها با یکدیگر ، حالتی است که a=1 می باشد . یک استراتژی مشابه این است که $c_i$ ها بجای جمع شدن با مقدار $a\ h_{i-1}$ ، exclusive-or شوند.

برای اعداد صحیح 32 بیتی ، اگر a = 65599 باشد ،که یک عدد اول نزدیک $2^{16}$ است در این صورت بزودی در محاسبه $a\ h_{i-1}$ سر ریز رخ خواهد داد . با توجه به اینکه a عدد اول است ، صرفنظر کردن از سرریزها و حفظ 32 بیت مرتبه پایین به نظر مناسب است .

در یک سری آزمایش ، تابع درهم hashpjw در شکل زیر برای کامپایلر C از P.J.Weinberger با تمام اندازه ها جدول امتحان شده به شکل مناسبی عمل می نماید.

```
(1) #define PRIME 211
(2) #define Eos ’ \0 ‘

(3) int hashpjw (s)
(4) char *.s;
(5) {
(6)       char *p;
(7)       unsigned h = 0 , g;
(8)       for ( p = s; *p != Eos; p =p+1 )  {
(9)          h = ( h << 4 )  +  (*p);
(10)       if ( g= h & 0xf0000000 )  {
(11)          h = h ^ ( g >> 24 );
(12)          h = h ^ g;
(13)        }
(14)      }
(15)       return h % PRIME ;
(16) }
```

این اندازه ها شامل اولین اعداد اول بزرگتر از 100و200و.....و1500 می باشند . دومین تابع نزدیک ، تابعی بود که h را با ضرب مقدار قبلی آن در 65599 و صرف نظر از سر ریز و جمع آن با کاراکتر بعدی محاسبه نموده تابع hashpjw با شروع از h=0 محاسبه می شود. برای هر کاراکتر c بیت های h به اندازه 4 مکان به چپ انتقال می یابند و با c جمع می شوند . اگر هر یک از 4 بیت مرتبه بالای h (۱) باشند این 4 بیت ، 24 مکان به راست

انتقال داده می شود و با exclusive-or, h می گردند و هر یک از چهار بیت مر تبه بالا که 1 بوده ، به صفر تبدیل می شود.

**مثال 1- الف** به منظور رسیدن به بهترین نتایج ، اندازه جدول درهم و ورودی مورد انتظار در زمان طراحی تابع در هم باید مورد توجه قرار گیرد . برای مثال مطلوب است که مقادیر در هم برای نام هایی که بطور مکرر در زبان رخ می دهند متفاوت باشند اگر کلمات کلیدی نیز به جدول نماد وارد شوند ، کلمات کلیدی نیز در گروه نام هایی هستند که بطور مکرر استفاده می شوند . اگر چه در یک نمونه از برنامه های c نام i تا بیش از سه بار ، مشابه while استفاده شده است .

یک راه امتحان نمودن تابع درهم تعیین تعداد رشته هایی است که در لیست مشابه قرار می گیرند . با داشتن یک فایل به نام F شامل n رشته ، فرض کنید تعداد $b_j$ رشته در لیست j قرار گیرند ،برای $0 \le j \le m-1$ یک اندازه گیری از میزان یکنواختی رشته هایی که در لیست ها توزیع شده اند با محاسبه :

$$\sum_{j=0}^{m-1} b_{j(}b_{j+1)/2}$$

بدست می آید . یک بازبینی استقرایی برای این عبارت این است که برای یافتن اولین وارده در لیست j نیاز بررسی یک عنصر ازلیست می باشد ، برای یافتن دومین عنصر ، دو عضو باید بررسی شود ، و به همین ترتیب تا برای آخرین عنصر به $b_j$ بررسی نیاز است . مجموع 1و2و....$b_j$ برابر با $b_j(b_j+1)/2$ می باشد .

## نمایش اطلاعات محدوده

وارده های جدول نماد ، برای اعلان نام ها می باشند . هنگامی که برای رخدادی از نام ، در جدول نماد جستجو می شود ، وارده مربوط به اعلان مناسب آن نام باید بر گردانده شود . قوانین محدوده در زبان مبدا مشخص می نمایند که کدام اعلان مناسب است .

یک روش ساده استفاده از جدول نماد مجزا برای هر محدوده است . در نتیجه جدول نماد برای یک رویه یا محدوده معادل رکورد فعالیت در زمان کامپایل است . اطلاعات غیر محلی یک رویه با استفاده از پویش جدول های نماد مربوط به رویه های موجود در حیطه قوانین محدوده زبان یافت می شود . بطور مشابه اطلاعات محلی یک رویه می تواند به گروه مربوط به آن رویه در درخت نحو مربوط به آن برنامه الحاق گردد . با این گرایش ، جدول نماد در نمایش میانی ورودی مجتمع می شود .

اکثر قوانین محدوده که به شکل نزدیکی متداخل هستند ، می توانند با اقتباس از ساختمان داده های ارائه شده در این بخش پیاده سازی گردند . نام های محلی هر رویه با اختصاص عدد منحصر به فردی به هر رویه قابل پیگیری است . اگر زبان دارای ساختار بلوکی است بلوک ها نیز باید شماره گذاری شوند . شماره هر رویه می تواند به صورت نحوگرا با استفاده از قوانین معنایی که ابتدا و انتهای هر رویه را مشخص می نمایند ، محاسبه گردد. شماره رویه به عنوان بخشی از تمام اطلاعات محلی آن رویه قرار می گیرد ، نمایش نام محلی در جدول نماد زوجی است شامل نام و شماره رویه. (در بعضی از موارد مانند آن

هایی که در زیر توصیف شده اند ظاهر شدن شماره رویه به طور واقعی لزومی ندارد ، زیرا می تواند ازموقعیت رکورد در جدول نماد مشخص شود ).

هنگامی که برای نام تازه پویش شده به جدول نماد مراجعه می شود ، تنها هنگامی که انطباق رخ می دهد که کاراکترهای نام به صورت کاراکتر به کاراکتر با وارده منطبق گردد و شماره همراه آن در وارده جدول نماد با شماره رویه ای که در حال پردازش است برابر باشد. اکثر قوانین محدوده ای که بطور نزدیکی با هم متداخلند می توانند در قالب عبارت هایی از اعمال زیر بر روی یک نام پیاده سازی شوند :

Lookup : یافتن جدیدترین وارده ایجاد شده

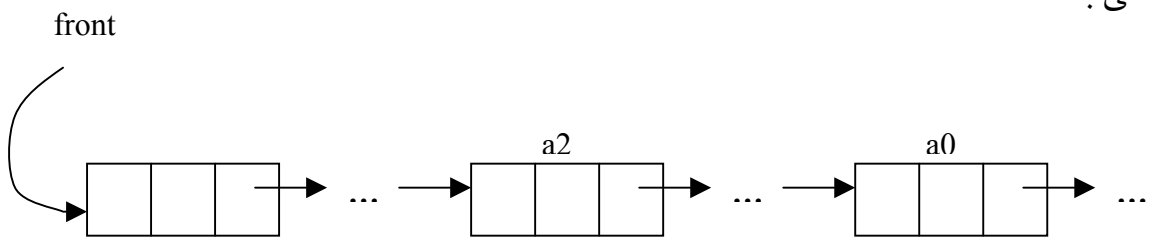
Insert: ایجاد یک وارده جدید

Delete: حذف جدیترین وارده ایجاد شده .

وارده های حذف شده باید نگهداری شوند ، آنها فقط از جدول نماد فعال حذف می شوند در کامپایلر تک گذره ، اطلاعات جدول نماد در مورد یک محدوده شامل بدنه رویه ، پس از پایان پردازش بدنه رویه مورد نیاز نمی باشد . به هر حال ممکن است در زمان اجرا مورد نیاز باشد ، مخصوصاً اگر یک سیستم تشخیصی زمان اجرا پیاده سازی شود . در این حالت ، اطلاعات جدول نماد ، باید به کد تولید شده برای استفاده الحاق گر یاسیستم تشخیصی زمان اجرا اضافه گردد .

هر یک از ساختمان  داده های بحث شده دراین بخش ، لیست ها و جداول در هم می توانند به شکلی به کار گرفته شوند که اعمال فوق را حمایت نمایند .

هنگامی که لیست خطی شامل آرایه ای از رکورد ها در قسمت های قبلی این بخش توصیف شد ، به این نکته اشاره شده که چگونه Lookup می تواند با درج کردن وارده ها در یک طرف به شکلی که ترتیب وارده ها درآرایه مشابه ترتیب درج کردن وارده ها باشد ، پیاده سازی شود . یک پویش با شروع از انتها و ادامه به سمت ابتدای آرایه ، جدیترین وارده را برای آن نام می یابد. این وضعیت شبیه لیست پیوندی شکل زیر می باشد :




اشاره گر front به جدیترین وارده اضافه شده به لیست اشاره می نمایند . پیاده سازی insert زمان ثابتی نیاز دارد زیرا وارده جدید در جلوی لیست قرار می گیرد . پیاده سازی Lookup با پویش لیست و شروع از وارده اشاره شده توسط front و دنبال نمودن اتصال ها تا زمان یافتن وارد ه مورد نظر یا فرارسیدن انتهای لیست ، انجام می گیرد . در شکل فوق وارده برای a که دربلوک $B_2$ اعلان شده است ، که خود در داخل بلوک $B_0$ قرار دارد ، از وارده a که در B0 اعلان شده ، به ابتدای لیست نزدیک تر است .

برای عمل delete توجه داشته باشید که وارده های مربوط به اعلان های رویه ای که در عمیق ترین رویه داخلی قرار دارند ، در نزدیک ترین قسمت به ابتدای لیست قرار می گیرند. بنابراین ، نیازی به نگهداری شماره رویه با هر وارده نمی باشد . اگر اولین وارده برای هر رویه مشخص شود تمام وارده ها تا اولین وارده هنگامی که پردازش محدوده این رویه خاتمه می یابد ، می تواند از جدول نماد فعال حذف گردد . جدول در هم شامل m لیست ، با استفاده از آرایه دستیابی می شود . با توجه به اینکه یک نام همیشه با استفاده از تابع درهم به لیست منتقل می شود هر یک از فهرست ها مشابه شکل فوق نگهداری می شود . به هر حال برای پیاده سازی عمل delete نیازی به پویش تمام جدول در هم برای یافتن لیستی شامل وارده هایی که باید حذف شود نمی باشد . روش زیر می تواند مورد استفاده قرار گیرد .فرض کنید هر وارده دارای دو اتصال است :

۱- یک اتصال در هم که این وارده را به وارده های دیگری که نام آنها توسط تابع در هم به مقدار مشابه تبدیل می شود به زنجیر متصل می نماید .

۲- یک اتصال محدوده که تمام وارده هایی را که درمحدوده مشابه قرار دارند به صورت زنجیر به یکدیگر متصل می نماید .

اگر اتصال محدوده در زمان حذف یک وارده از جدول در هم دست نخورده باقی بماند ، زنجیری که با اتصالات محدوده تشکیل شده است ، جدول نماد مجزایی را برای محدوده مورد بحث (غیر فعال ) تشکیل می دهد .

حذف وارده از جدول در هم باید با دقت انجام گیرد ، زیرا حذف یک وارده بر مقدار قبلی آن در فهرست تاثیر دارد . بخاطر آورید که حذف وارده شماره i با برقراری ارتباط وارده شماره i-1 با وارده شماره i+1 انجام می گیرد بنابراین تنها استفاده از اتصالات محدوده برای یافتن وارد شماره i کافی نیست . اگر اتصالات در هم یک لیست پیوندی حلقوی را تشیکل دهند که درآن آخرین وارده به اولین وارده اشاره نماید ، وارده i-1 ام نیز می تواند یافت شود . در مقابل می توان از پشته به منظور پی گیری تعیین لیست هایی شامل وارده هایی که باید حذف شوند استفاده نمود . هنگامی که رویه جدیدی پویش می شود ، یک علامت در پشته قرار می گیرد . در بالای علامت ، شماره لیست هایی شامل وارده هایی برای نام های اعلان شده در این رویه قرار دارند . هنگامی که پردازش این رویه خاتمه می یابد ، اعداد فهرست می توانند از پشته خارج شوند تا زمانی که علامت آن رویه فرابرسد.