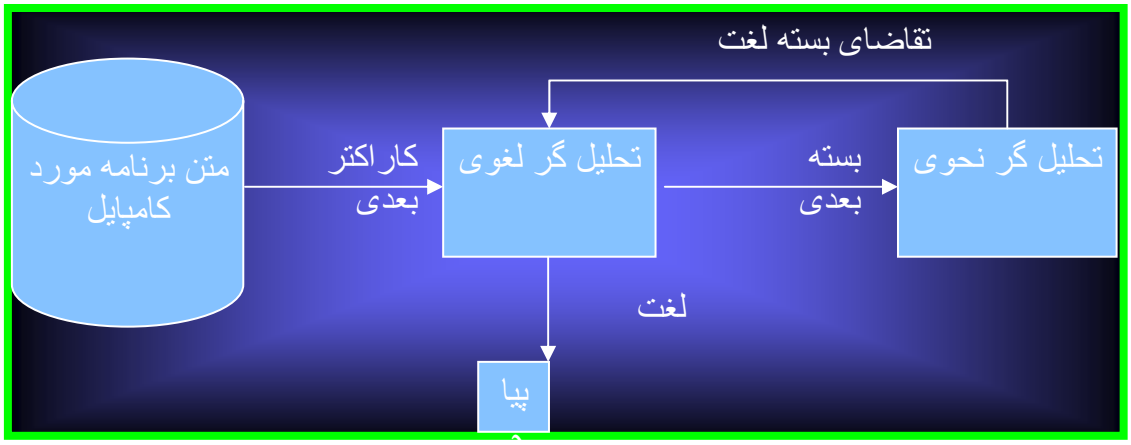
فصل دوم

تحليلگر لغوی

۲.۱ مقدمه

تحليلگرلغوي تابعي است كه در ورودي خود برنامه مورد كامپايل را بعنوان يك فايل متن بازگشوده و كاراكتر به كاراكتر مي خواند وبا رسيدن به يك كاراكتر جداكننده لغت را تشخيص و تفكيك مينمايد. جداكننده ها مثل : Blank , Tab يا New line فقط به منظور جداسازي لغات بكار ميروند. دسته ديگر از جدا كننده ها مثل Semicolon يا + ارزش لغوي دارند. تحليلگر لغوي در خروجي خود اطلاعات مربوط به لغت را در يك ركورد يا ساختار بنام بسته لغت قرارمي دهد بسته لغت شامل اطلاعاتي در مورد لغت تشخيص داده شده توسط تحليلگر لغوي مانند سطر ، ستون و نوع لغت است بسته لغت را در اصطلاح Token گويند.



شكل 2.1- شماي كلي تحليلگر لغوي

هر زبان برنامه سازي قوانين لغوي خاص خود را دارد. قوانين لغوي زبانها را ميتوان بصورت غير مبهم و خلاصه در قالب نوعي خاص از عبارات بنام عبارت با قاعده بيان نمود. قوانين لغوي را نيز ميتوان توسط نوعي خاص از ساختارگراف بنام ماشين هاي خودكار مطرح كرد. ماشينهاي خودكار ابزاري براي تبديل قوانين لغوي به كد برنامه هستند. چنانچه لغتي بنا بر قوانين لغوي زبان برنامه سازي ايجاد نشده باشد ، تحليلگر لغوي اعلام خطا مينمايد.

۲.۲ ساختار ورودی / خروجی

در داخل تابع تحليلگر لغوي برنامه مورد كامپايل به عنوان يك فايل متن دسترسي ميشود. اين فايل در ابتداي برنامه كامپايلر وخارج از محيط تحليلگر لغوي تعريف وگشوده مي شودو به عنوان يك پارامتر به تابع تحليلگر لغوي ارسال مي گردد. در قطعه كد ارائه شده در شكل 2.2 چگونگی فراخواني تابع تحليلگر لغوي نشان داده شده است..

```
                    Void  main ( int arg c , char  *argv [  ] )
{
    File   *InText ;
      if  (  argc  <  2  )
            {
              clrscr  (  ) ;
              textcolor  ( RED  ) ;
              gotoxy  ( 10 , 5 )
                 cprintf  ( “  نام برنامه مورد كامپايل فراموش شده است “ ) ;
               if  (  !getch ( )  )   getch  ( ) ;
                                        }
       InText  =  fopen ( argv [ 1 ] , “  R  “ ) ;
        while  ! feof ( InText ) ;
                {
                  Token-type   Token ;    .
                 Token =  Lexer ( InText ) ;
                                  .
```

شكل 2.2- فراخواني تابع تحليلگر لغوي

نقطه گنگ در مثال فوق TokenType و در واقع نوع خروجي تحليلگر لغوي است. معمولا" اطلاعات زير در بسته Token قرار داده مي شود :

```
Typedef  struct  Token
        {  int    ROW ;  //                    شماره سطر
           int    COL; //                     شماره ستون
             int  BLKNO ;  //                  شماره آشيانه اي
         int  BLKORD ; //  شماره  ترتيب بلاك در برگيرنده
            enum  Symbols  type ;  //          نوع لغت
             char Name [ 30 ] ;  //            خود لغت
        }  Token-type ;
```

شكل 2.7- ساختار بسته لغت

در ساختار ارائه شده براي تعريف لغت كه در اينجا بسته لغت يا Token ناميده شده است شماره سطر وستون ظاهر شده است. با استفاده از شماره سطر و ستون لغت ، در هنگام صدور پيام خطا كامپايلر قادر به بيان دقيق مكان خطاء خواهد بود. BLKNO شماره آشيانه اي بلاك در بر گيرنده يك لغت را مشخص مي كند. تنها نام لغات براي تعيين و تشخيص آنها از يكديگر كافي نيست بلكه ، جهت تعيين يك لغت نياز به شماره آشيانه اي بلاك در برگيرنده آن نيز هست . براي نمونه به قطعه كد زير توجه كنيد.

```
{
   int  I ;
   I=5 ;
        { int I ;
          I=6 ;
          printf ( " 2nd blk   %d " / I ) ;
        }
              printf ( " \n  1st blk  %d " / I ) ;
          }
```

در قطعه كد فوق همانگونه كه مشاهده مي كنيد اگرچه كه براي دو متغير يك نام I تعيين شده اما شماره آشيانه اي بلاك در بر گيرنده ، وجه تمايز اين دو مي باشد. گاهي اوقات شماره آشيانه اي بلاك نيز كافي نيست براي نمونه درقطعه كد زيراگرچه كه شماره آشيانه اي بلاكها يكي است اما دو متغيير وجود دارد.

```
{  int  I  ;
    I = I + 1
}
{  int  I  ;
    I = I + 3
}
```

شماره آشيانه اي با ورود به بلاك افزايش وبا خروج از بلاك كاهش مي يابد. با ظهور هر بلاك جديد شماره بلاك يا BLKORD يك واحد اضافه مي شود در مثال فوق وجه تمايز دو متغيير شماره ترتيب بلاك در بر گيرنده آن است .

به هر لغت يك نوع تخصيص داده ميشود. نوعي شمارش پذير يا در اصطلاح Enumenated . براي نمونه تحليلگر لغوي اگردر متن فايل ورودي عدد 123 را تشكيل دهد، در داخل فيلد Name عدد 123 و در داخل فيلد Type ، نوع آن كه يك عدد صحيح مي باشد را مشخص مي كند . اگر در متن ورودي ، جمله زير قرار گرفته باشد :

```
ABC := 123 ;
```

تحليلگر لغوي با اجراي دستورالعملي مثل :

```
NextChar = getc ( in-text ) ;
```

ابتدا كاراكتر **A** وسپس **B** وبعد از آن **C** را از متن فايل ورودي خوانده ، حالا با مشاهده Blank و در صورت عدم وجود Blank با مشاهده علامت كولون (:) ، خاتمه اولين لغت را تشخيص داده ، در داخل فيلد Name رشته ABC و در داخل فيلد Type ، نوع آن كه يك شناسه مي باشد را قرار مي دهد.

در فراخواني بعدي تحليلگر لغوي با Blank مواجه مي شود. ازآن چشم پوشي مي كند وبه جلو مي رود و علامت =: را به عنوان لغت بعدي تشخيص داده در فيلد Name ، رشته =: و در فيلد Type ، عملگر تخصيص را قرار مي دهد. البته شماره سطر وستون وبلاك در بر گيرنده هر لغت نيز مشخص مي شود.

نوع لغات در فيلد Type از ساختار TokenType مشخص شده است. اين فيلد از نوع شمارش پذير بنام Symbols تعريف شده است. در داخل نوع شمارش پذير يا در اصطلاح Enumerated بنام Symbols انواع ممكن لغات در داخل زبان مورد نظر مشخص ميشود. بايد توجه داشته باشيد كه با تعريف متغير از نوع شمارش پذير مجموعه اي از مقادير ثابت يا Constant تعريف ميشود. نوع Symbols در زبان C بصورت زير براي نمونه تعريف ميشود.

```
enum  Symbols { S_Program, S_Const, S_Eq, S_Semi, S_Id, S_No, S_Type,
                S_Record, S_End, S_Int, S_Real, S_Char, S_Array, S_String,
                S_Begin, S_Var, S_Colon, S_ParBaz, S_ParBast, S_Set, S_of,
                S_BrackBaz, S_BrackBast, S_Case, S_Pointer, S_ConstString,
                S_Function, S_Procedure, S_Begin, S_Dot, S_Comma,
                S_If, S_Then, S_Else, S_While, S_Do, S_Repeat, S_Until,
                S_For, S_Add, S_Sub, S_Div, S_Mul,  S_Mod, S_Lt, S_Le,
                S_Gt, S_Ge, S_Gt, S_Ne, S_Not, S_And, S_Or};
```

همانگونه كه مشاهده ميشود انواع مختلف لغات بايد در نوع Symbols گنجانده شود.

۲.۳ عبارات با قاعده

عبارت با قاعده[1] ، فرمي است چكيده وخلاصه براي بيان قوانين لغوي زبانها. شكل لغات درزبانهاي برنامه سازي داراي فرم كلي خاص است. بر اساس اين فرم كلي است كه انواع لغات از قبيل شناسه ها (اسامي) ، اعداد ، رشته هاي ثابت كاراكتري ، جملات تفسيري (Comment) وساير لغات از يكديگر تفكيك و تميز داده مي شوند.

۲.۳.۱ نمونه هايي از عبارات با قاعده

به عنوان نمونه تعريف شناسه ها در زبان **C** را در نظربگيريد. يك شناسه يا Identifire در زبان **C** حتما" بايد با يك كاراكتر آغاز گردد و در ادامه ممكن است به هر تعداد وبا هر تركيبي از ارقام صفر تا نه (0-9) ، حردف الفباي انگليسی (A-Z) و علامت خط زير يا در اصطلاح (UnderLine) ادامه يابد. به عنوان مثال اسامي :

A , B_ _ 1 , BAC2345_ _

ازلحاظ زبان **C** به عنوان اسامي (شناسه) ، شناخته مي شوند. ميتوان با استفاده از يك عبارت با قاعده به صورت زير، فرم كلي شناسه ها را در زبان **C** تعريف نمود :

Identifire : Letter (Letterr | Digit | '_') *
Letter : A | B | ..... | Z | a | b | ..... | z
Digit : 0 | 1 | 2 | ..... | 9

در عبارت ارااه شده براي شناسه ها ، علامت '*' نمايانگر تكرار صفر يا بيشتر مي باشد و علامت ' | ' علامت يا مي باشد. مي توان بصورت چكيده نيز Digit را تعريف نمود :

Digit : [0 ... 9]

به همين ترتيب ، حروف الفباي انگليسی را بصورت زير مي توان تعريف نمود :

Letter : [a ... z , A ... Z]

شكل كلي اعداد صحيح را ميتوان با استفاده از يك عبارت به اين صورت مشخص نمود:

Number : digit digit *
Digit : [0 ... 9]

بر طبق اين عبارت ، يك عدد صحيح با يك رقم بين صفر تا نه آغاز مي شود و به هر تعدادي رقم مي تواند در ادامه آن ظاهر شود. براي مثال :

0 , 5 , 0123

[1] Regular Expression

همانگونه كه مشخص است ، يك عدد حداقل داراي يك رقم بايد باشد يا به عبارت ديگر يك عدد از يك يا بيشتر ارقام تشكيل شده ، مي توان عدد را بصورت زير نيز تعريف كرد :

Number : digit +

در اينجا علامت '+' نمايانگرتكرار يك يا بيشتر مي باشد، يادآوري مي كنيم كه علامت ستاره '*' نمايانگر تكرار صفر يا بيشتراست.

اعداد مي توانند داراي علامت و يا بدون علامت باشند

Number : (+ | - | ε) digit +

در اين عبارت با قاعده ، علامت اپسيلون ε نمايانگر *عدم وجود* و يا *تهي* مي باشد . عبارت فوق به اين صورت خوانده مي شود :

يك عدد داراي علامت بعلاوه يا منها و يا ممكن است اصلا" علامتي نداشته باشد و بدنبال آن به تعداد يك يا بيشتر ارقام بين صفر تا نه ظاهر مي گردد.

اعداد را ميتوان به فرمي ساده تر با استفاده از اين قاعده كه هر عبارت اختياري مثل **r** بصورت **r ?** نمايش داده ميشود ، بصورت زير مشخص نمود :

Number : (+ | -) ? digit +

بعنوان نمونه اي ديگر از عبارات با قاعده ، فرم كلي رشته ها در زبان پاسكال را در نظر بگيريد. بايد توجه داشته باشيد كه در اين زبان چنانچه در وسطح رشته علامت كوتيشن ' نياز باشد ، مي بايست آنرا دوبار تكرار نمود. براي نمونه چنانچه جمله This is your's در خروجي مورد نظر باشد، ميتوان دستور Writeln (' This is your '' s ') ; استفاده نمود. لذا، فرم كلي رشته ها با عبارت با يك عبارت با قاعده بصورت زير بيان ميشود :

String : ' (Characters | ' ') * '

در عبارت فوق ، مجموعه Characters نمايانگر هرگونه كاراكتر قابل مشاهده منهاي كوتيشن است.

۲.۳.۲ قوانين ايجاد عبارات با قاعده

براي ايجاد يك عبارت با قاعده ، تعدادي از علائم مورد استفاده واقع مي شوند. هريك از اين علائم داراي معني و مفهوم خاصي مي باشد. در حالت كلي اگر دلتا (Δ) مجموعه علائم بكار رفته شده در عبارات باشد، آنگاه :

1. هر عنصر $a \in \Delta$ خود يك عبارت با قاعده است. براي مثال چنانچه $\Delta = [\ 0\ ..\ 9\ ]$ باشد آنگاه هر رقمي بين صفر تا نه مثل 1، خود به تنهايي يك عبارت با قاعده است.
2. چنانچه r و s دو عبارت با قاعده باشند ، آنگاه rs نيز يك عبارت با قاعده است ε به عبارت ساده تر در حالت كلي اگر دو عبارت با قاعده را در كنار يكديگر قرار دهيد ، حاصل يك عبارت با قاعده خواهد بود.
3. چنانچه r و s دو عبارت با قاعده باشند، آنگاه r يا s كه بصورت (r|s) مشخص ميشود نيز يك عبارت با قاعده است ε عملگر | داراي خاصيت جابجايي است ، به عبارت ديگر:

$$r|s = s|r = (r|s)$$

4. چنانچه r يك عبارت با قاعده باشد آنگاه r* نمايانگر تكرار صفر يا بيشتر از عبارات r است ε براي نمونه اگر r , a يا b باشد آنگاه r* برابر است با :

$$r : a|b$$
$$: (a|b)*$$

اين عبارت گويای هر تركيبي با هر تعدادي از a يا b كه در كنار يكديگر قرار گرفته اند مي باشد. براي نمونه رشته هاي

AAA , BAABB , BBBB

همگی فرم هاي خاصي از عبارت r* هستندε بنابراين علامت اپسيلون ε كه نمايانگر عنصر تهي است ، به تنهايي يک عبارت با قاعده است

5. چنانچه r يك عبارت با قاعده باشد، آنگاه r^+ نمايانگر تكرار يك يا بيشتر عبارت r است. براي نمونه :

Number : digit $^+$

به اين ترتيب واضح است كه : $r^+ = (r^*|\varepsilon)$

با استفاده از پنج قاعده فوق ، مي توان عبارات با قاعده را بنا نمود.
با استفاده از نكات فوق ، فرم كلي جملات تفسيري (Comment) در زبان پاسكال را ميتوان بصورت زير معين نمود. بايد توجه داشته باشيد كه در زبان پاسكال جملات تفسيري در بين علائم آكولاد باز و آكولاد بسته قرار مي گيرند.

Comment1 : { C* }
C : (كليه كاراكترها منهاي آكولاد)

در زبان پاسكال جملات تفسيري را ميتوان بين علائم (* و *) نيز مشخص نمود. در اين فرم از جملات تفسيري بايد توجه داشته باشيد كه اگر ستاره در داخل جمله

ظاهر شود، در پشت آن بايد هر حرفي به غير از ستاره و پرانتز بسته ظاهر شودء براي نمونه ، لغت زير يك جمله تفسيري **نيست** :

(* abc *) acb *)

فرم كلي اينگونه جملات را ميتوان در قالب يك عبارت با قاعده بصورت زير خلاصه نمود:

Comment2 : '(' * ((r | * $^{+}$ s) * r) * * $^{+}$ ')'

r : كليه‌كاراكترها منهاي علامت *

s : كليه‌كاراكترها منهاي علائم * و ')'

به اين ترتيب جملات تفسيري بصورت زير

Comment : Comment1 | Comment2

تعريف مي شوند.

با توجه به اينكه اعداد يا به صورت صحيح و يا به صورت اعشاري ظاهر مي شوند و با در نظر گرفتن اينكه اگر قبل از ممیز حداقل يك رقم ظاهر شود آنگاه وجود ارقام پس از ممیز ضروري نيست و نيز اينكه اگر بعد از ممیز حداقل يك رقم ظاهر شود وجود رقم قبل از ممیز ضروري نيست . بطور خلاصه عبارت با قاعده براي بيان شكل كلي اعداد بصورت زير ميتواند باشد:

Number : (digit*.digit^{+}) | (digit^{+}.digit*) | digit^{+}

عبارات با قاعده مبين قوانين لغوي و نمايانگر فرم كلي لغات هستند. متاسفانه ، اين فرم كلي را بسادگی نميتوان تبديل به كد برنامه نمود. لذا ، براي رفع اين مشكل از ماشينهاي خودكار استفاده ميشود. .

۲.۵- ماشينهای خودكار

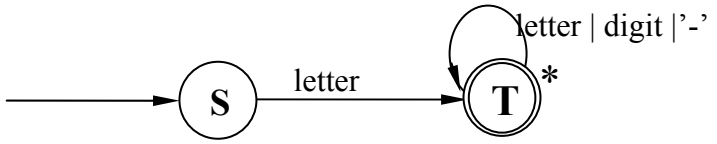
يك ***ماشين خودكار***[2] نوعي گراف مي باشدكه داراي تعدادي رئوس يا حالات (States) و تعدادي لبه يا يال (Edges) مي باشد و در اين گراف يالها خطوط واصل بين حالات هستند. اين ماشينها، ابزاري براي تعيين قوانين لغوي و تشخيص لغات مي باشند كه بسادگی قابل تبديل به كد برنامه بوده بطوري كه با استفاده از آنها مي توان تحليلگر لغوي را بصورت يك برنامه تبديل كرد و يا Source يك برنامه را توليد نمود.

براي نمونه شناسه ها را در نظر بگيريد. شناسه ها توسط عبارت با قاعده :

[2] Finite State Automata

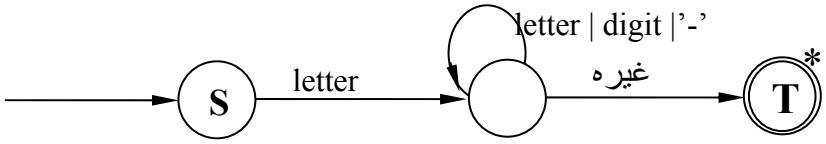
letter (letter | digit)*

در بخش قبل معين شدند. براي تشخيص شناسه ها مي توان ماشين خودكار زير را مورد استفاده قرار داد:



اينن ماشين خودكار را بصورت زير نيز ميتوان نمايش داد. بنابر دياگرام فوق در صورتيكه كاراكتر خوانده شده ازمتن فايل مورد تحليل لغوي يكي از حروف الفباي لاتين يا در اصطلاح letter باشد ميتوان وارد ماشين خودكار تشخيص شناسه ها شد. حالت شروع و در واقع نقطه ورود به ماشين خودكار با حرف S مشخص شده است. حالت بعدي T ناميده شده است. در حالت T اطمينان از تشخيص يك شناسه است زيرا ، يك حرف نيز به تنهايي يك شناسه در نظر گرفته ميشود. پس در اين حالت لغت خوانده شده بعنوان يك شناسه پذيرفته شده است.

حالت پذيرش ماشين خودكار را با دو دايره تو در تو نشان ميدهند. در حالت پذيرش T آنقدر از ورودي كاراكتر هاي بعدي خوانده ميشوند تا بهكاراكتري غير از آنچه برروي اين حالت مشخص شده در ورودي برسيد. پس براي تشخيص يك شناسه نيز نياز بخواندن يك كاراكتر اضافي از ورودي هست. در شكل فوق علامت * نمايانگر خواندن يك كاراكتر اضافي تر است . براي مثال در AB12< با ديدن علامت < است كه مي توان فهميد شناسه AB12 به پايان رسيده است. ميتوان شناسه ها را با ماشين خودكار زير نيز نمايش داد:



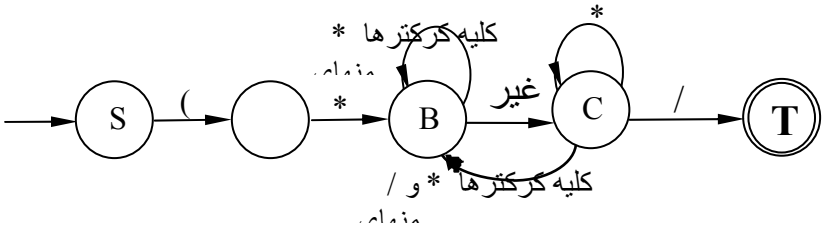
جملات تفسيري Comment توسط تحليلگر لغوي بعنوان يك لغت شناسايي ميشوند. جملات تفسيري پاسكال را ميتوان توسط عبارت زير مشخص نمود :

Comment2 : '(*' ((r | *+ s) * r) * *+ ')'

r : كليه كاراكترها منهاي علامت *

s : كليه كاراكترها منهاي علائم * و ')'

ميتوان جملات تفسيري را با يك ماشين خودكار بصورت زير مشخص نمود:



همانگونه كه مشاهده مي كنيد در حالت شروع S با ديدن كاراكتر / ، گذرى (Transision) از حالت شروع S به حالت **A** وجود دارد و در حالت A اگر * ديده شود گـذري به حالت B وجود دارد. در غير اينصورت اين ماشين خودكار لغت خوانده شده را نمي پذيرد و در اصطلاح Reject مي كند . در حالت B با ديدن هركاراكتري غير از * ماشين خودكار به كار خواندن كاراكترهاي بعدي ادامه ميدهد. اما ورودي * اهمّيت ديگري دارد . با ديدن * اگر در پشت آن علامت / ظاهر شود كار خاتمه يافته تلقي مي شود .

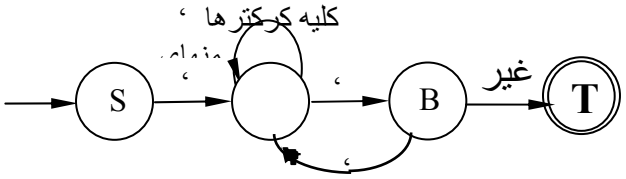
حالت **T** ، حالت پذيرش است . در حالت C در واقع ماشين بخاطر دارد كه * ديده شده است . اگر در حالت C در ورودي علامت / ظاهر شود ، كار خاتمه يافته است. در غير اينصورت ، به حالت B تغيير وضعيت داده ميشود. پس مشاهده مي كنيد كه ماشينهاي خودكار بسيار ساده تر از عبارات با قاعده قابل توليد هستند

فرم كلي رشته هارا در زبان پاسكال با عبارت با قاعده

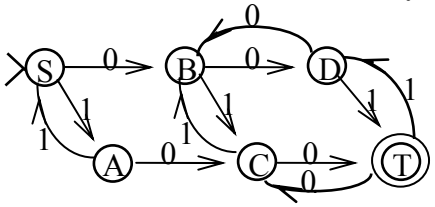
String : ‘ (c | ‘’)* ‘

كليه‌كاراكترها منهاي علامت كوتيشن ' : C

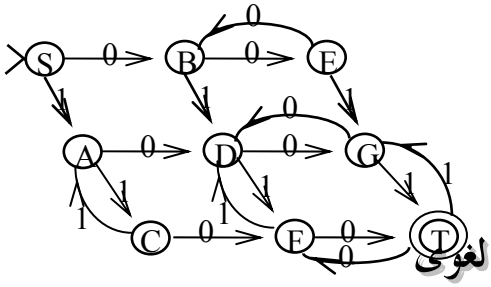
مشخص ميشود. ميتوان رشته ها را با ماشين خودكار زير نيز معين نمود:



مثال: يك ماشين خودكار قطعي براي كليه اعداد مبناي دوكه تعداد صفرهاي آنها زوج و تعداد يك ها فرد است ايجاد كنيد. در ضمن لااقل دو عدد صفر و يك عدد يك بايد در اين اعداد موجود باشد.

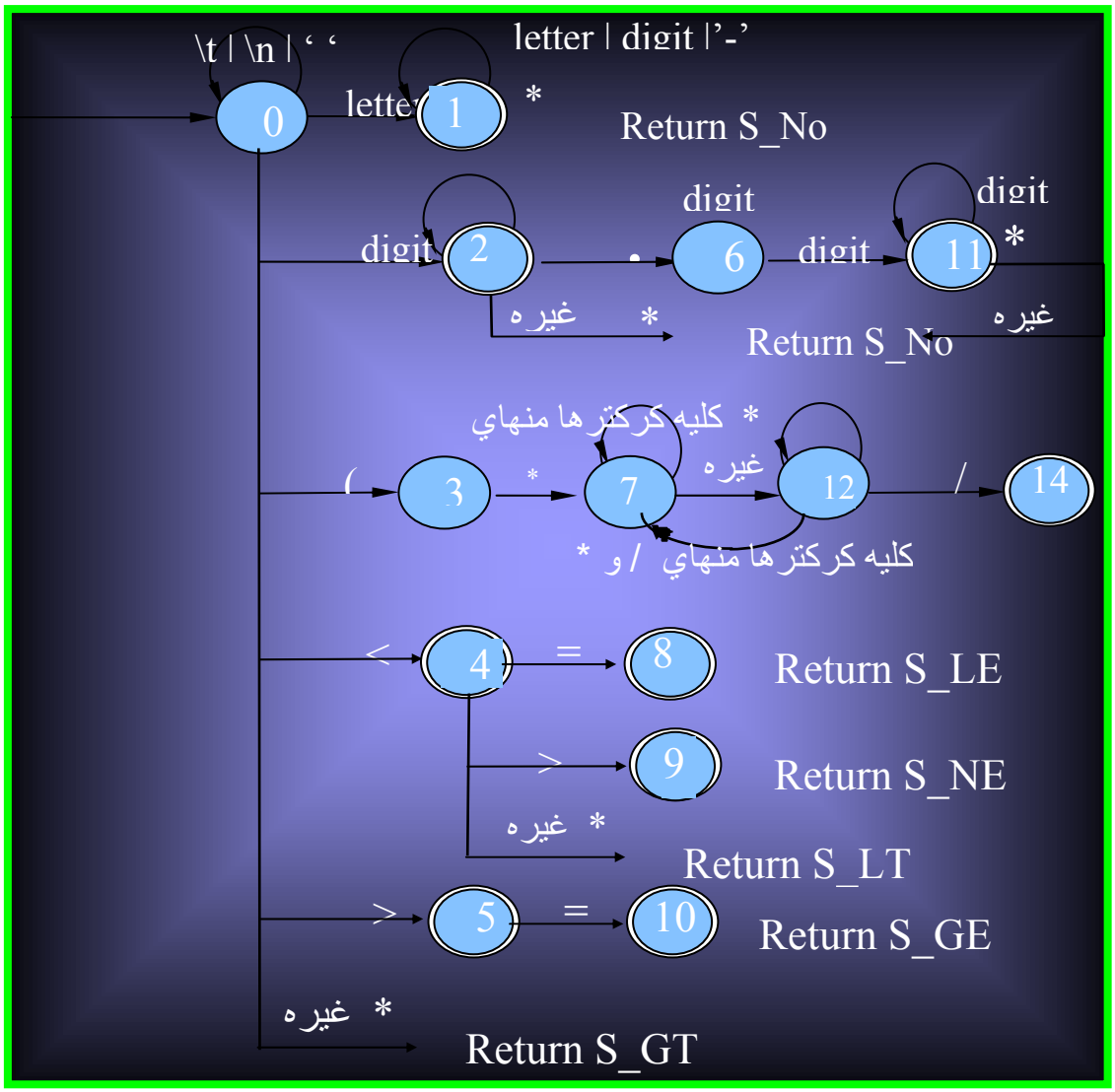


مثال: يك ماشين خودكار قطعي براي كليه اعداد مبناي دوكه تعداد صفرهاي آنها زوج و تعداد يك ها زوج است ايجاد كنيد. در ضمن لااقل دو عدد صفر و دو عدد يك بايد در اين اعداد موجود باشد.



٢.٦ ايجاد تابع تحليلگر لغوى

همانگونه كه در بخش 2و5 ذكر شد ، مي توان با استفاده از ماشينهاي خودكار قوانين لغوي را بيان نمود. در اين بخش نشان داده خواهد شدكه چگونه ميتوان ماشين خودكار را بسادگی تبديل به كد برنامه نمود . براي اين منظور يك دياگرام كلي براي نمايش برخي از لغات ارا ئه مي شود . دياگرام ارائه شده در شكل 2.8 نمايانگر ماشين خودكار كلي براي تشخيص تعدادي از لغات است. در واقع اين دياگرام بيان كننده الگوريتم تابع تحليلگرلغوي است.



شكل 2.8 - دياگرام تابع تحليلگر لغوي

هرگاه كه تابع تحليلگر لغوي مورد فراخواني واقع مي گردد ، در حالت شروع صفر قرار مي گيرد . اگر در فراخواني قبل، كاراكتري اضافه خوانده شده بود ، آن كاراكتر اضافه را مورد استفاده قرار مي دهد ، وگرنه كاراكتر بعدي را از داخل متن برنامه مورد كامپايل مي خواند. تا زماني كه كاراكترهايي كه ارزش لغوي ندارند مانند Tab ، Blank و Newline خوانده شوند ، تحليلگر لغوي در همان حالت شروع صفر باقي ميماند. در غير اينصورت وابسته به نوع كاراكتر ، در يك جمله **Case** اقدام به تشخيص لغات مختلف مي نمايد .

ميتوان با استفاده از دياگرام شكل 2.8 كد تابع تحليلگر لغوي را بشرح زير در زبان C ايجاد نمود.

```
struct TokenType  lexer ( FILE   *InText
                                                    )
{
   enum   Symbols    LexiconType  ;
   char   NextChar  ;    NextWord[80 ]  ;
   int   State , Lentgh  ;
static char LastChar = '\0';
static int RowNo =0, ColNo = 0;
     State   =  0  ;    //  در وضعيت شروع قرار مي‌گيرد
   Length  =  0 ;
   while  ( ! feof  (  InText  ) )
    {
      if  ( LastChar  )
         {  Nextchar  =  LastChar   ;   LastChar  =  '\0'  ;  }
      else
        NextChar   =     fgetc  (  InText  ) ;
       NextWord[Length++] = NextChar;
       switch   State
          {
        case 0:                  //      حالت شروع صفر
              if  (NextChar == '\n') {RowNo++ ;  ColNo =  0;  }
                                        else  ColNo++;
              if  ( Nextchar  = =  '  '  ||  Nextchar = = ' \ t '  ||
                   Nextchar  =  =  ' \ n '  )  Length = 0;
              else if  (( Nextchar < =  ' z '  & &
                          Nextchar  > = ' a '  )  ||
                         ( Nextchar  < = ' Z '  & &
                          Nextchar  > = ' A '  )      State  = 1;
               else  if  ( Nextchar  < =  ' 9 '  & & Nextchar  > =  ' 0 '  )
                                                  State = 2 ;
               else if  ( Nextchar  =  =  ' ( '  )        State 3;
```

```
        else if ( Nextchar  = =  ‘ < ‘ )        State  4
        else if ( Nextchar  = =  ‘ > ‘ )        State  5
        else    LexerError(NextWord, Length);
break  ; //  پايان حالت صفر
    case    1 :   //   تشخيص شناسه ها
        if( Isalpha( Nextchar ) || Isdigit( Nextchar ) ||
                        NextChar == ‘_’)  State = 1;
      else  { Lastchar  = Nextchar ;
              NextWord[Length-2] = ‘\0’;
              return MakeToken(IsKeyWord(NextWord)); }

        Break  ;
case   3  : // تشخيص اعداد
        .
        .
  } //  پايان سوئچ
} //  پايان حلقه
```

در مثال فوق تابع MakeToken کار ایجاد بسته لغات یا در اصطلاح Token را بر عهده دارد. IsKeyWord کار تشخیص نوع لغات کلیدي را بعهده دارد. این تابع در زبان C بشرح زیر است:

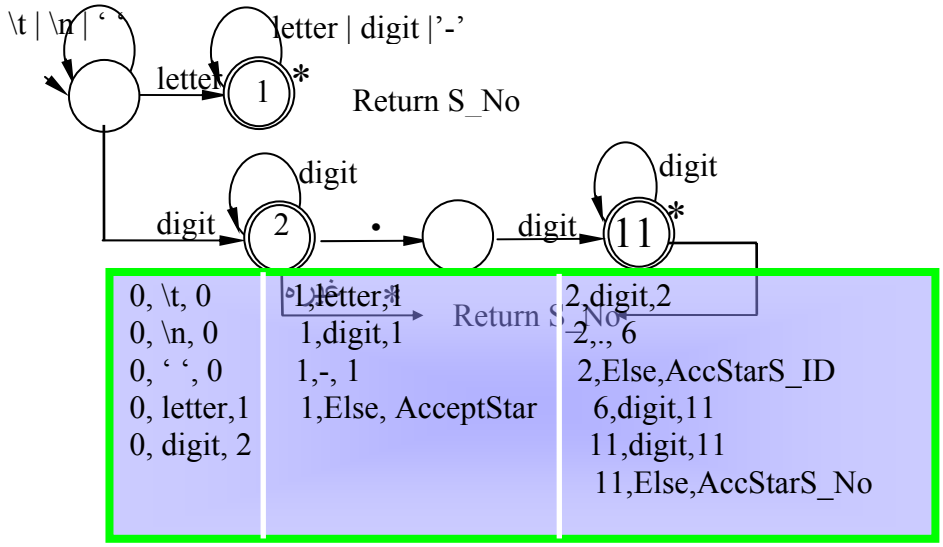
```
enum Symbols IsKeyWord( char *key)
  {
    int I ;
    struct  KeyType
        { char *key;
          enum Symbols Type
        } KeyTab[] = { “if”, S_If,
                       “while”, S_While,
                        “then”, S_Then,
                        “else”, S_Else,
                        “integer”, S_Integer,
                        “type”, S_Type,
                        “function”, S_Function,
                          0, 0};
    for( I  =0;  KeyTab[ I ].key &&
                 strcmp(KeyTab[I].key, Key); I++) ;
    if(KeyTab[I].key) return KeyTab[I].Type;
    Return S_Identifier;
} // EOF IsKeyWord
```

۲.۷ ایجاد مولد تحلیلگر لغوی

ميتوان به سادگی تابع مولد تحليلگر لغوي ايجاد نمود. كافيست ، تابعي براي پيمايش يك گراف بنويسيد. اين تابع در ورودي خود علاوه بر متن برنامه مورد تحليل لغوي ، فايل قوانين لغوي را در فرم يك ماشين خودكارمي پذيرد. اين تابع با خواندن هركاراكتراز داخل فايل متن برنامه مورد تحليل لغوي به پيمايش ماشين خودكار ميپردازد تا نوع لغت را تشخيص دهد.

از مولدهاي شناخته شده تحليلگر لغوي ، يكي **LEX** ميباشد. اين مولد امروزه بخصوص بر روي سيستمهاي عامل **UNIX** موجود مي باشد. كار با **LEX** مشكل است چرا كه نياز به بيان قوانين لغوي ، در فرم عبارات با قاعده دارد. در صورتي كه همانگونه كه تا كنون مشاهده كرده ايد، به فرمي خيلي ساده تر مي توان با استفاده از ماشينهاي خودكار، قوانين لغوي را بيان نمود .

بايد قوانين لغوي را در قالب يك ماتريس مشخص نموده و در داخل يك فايل متن قرار داد. سپس در داخل برنامه ميتوان به اين فايل متن ارجاع و ماشين خودكار را در قالب يك ماتريس مشخص نمود. براي نمونه در زير قوانين لغوي در قالب ماشين خودكار و ماتريسي مشخص شده است.



براي ايجاد ماتريس نياز به يك زباني ساده است كه براساس آن براي نمونه بتوان مشخص نمود كه براي مثال Letter ئيست و يا **غيره** ، كه مي باشد . شايد بهتر بود كه در فايل ورودي در داخل ماتريس قوانين لغوي Letter بصورت زير مشخص ميشد:

0 ; [A - Z , a - z] ; 1

حالا در داخل برنامه ، اين ساختار را به عنوان يك فايل **TEXT** ،كاراكتر به كاراكتر خوانده ، عينا در داخل يك ماتريس با تعداد سطرها كه مساوي با تعداد ركوردهاي فايل است وتعداد ستونها كه مساوي با طول هر ركورد يا هر سطر است ، ضبط نمود.
اكنون يا با نوشتن يك برنامه پيمايش كننده ماشين خودكار و يا توليد متن يك برنامه **C** كه بر اساس اين ماشين خودكار براي انجام عمل تحليل لغوي ايجاد شده ، مي توان لغات را تشخيص داد. همزمان با خواندن هركاراكتر از ورودي ، مثلا با خواندن كاراكتر **كوچكتر** از ورودي طبق ماتريس از حالت شروع صفر به حالت **5** تغيير وضعيت داده مي شود ء در اينجا سرعت عمليات تحليلگر لغوي ، وابسته به سرعت جستجو در داخل ماتريس است .
مي توان با ايجاد يك تحليلگر لغوي براي تشخيص متن برنامه ها با لغات كليدي فارسي اقدام نمود. براي مثال جمله زير را ميتوان به يك جمله if تبديل نمود.

حقوق > 1000

اگر

مالیات = حقوق / 100 : آنگاه

۲.۸ تبديل برنامه ها به زبان فارسی

ميتوان متن برنامه هاي فارسي را كه بر اساس قواعد زبانهاي متفاوت نوشته شده است را با استفاده از يك جدول تبديل كلي به زبان انگليسي تبديل و بالعكس پس از كامپايل و اشكال زدايي برنامه ها دوباره با استفاده از همان جدول از انگليسی به فارسي تبديل نمود .

اصولا زبانهاي سطح بالا به خاطر خوانا بودنشان ، **سطح بالا** خوانده مي شوند. متاسفانه اين انگليسی بودن زبانها، خوانايي برنامه ها را با مشكل مواجه نموده است . بخصوص در انتهاي مراحل تجزيه وتحليل سيستم ها، برنامه سازان را در تبديل متن عمليات تحليل شده از شبه دستورالعمل يا در اصطلاح **Pseudo code** به كد برنامه عملاً با مشكل زياد روبرو نموده است. شبه دستورالعمل را فارسي زبانها نمي توانند به سادگی به زبان انگليسی بنويسند. شبه دستورالعمل ها معمولاً از زبانهاي سطح بالا الهام گرفته مي شوند. مشكل زبان فارسي از راست به چپ بودن جملات وبالعكس ، چپ به راست بودن عبارات است . البته ، در اينجا اين مشكل موردي ندارد.

در مورد شناسه ها نيز در زبان فارسي مشكل وجود دارد . شناسه ها از انتها تشخيص داده مي شوند . مشكل ديگر اين است كه در زبان انگليسی 28 و در زبان فارسي 32 حرف وجود دارد . البته ، با تركيب حروف مي توان اين مشكل را حل نمود .

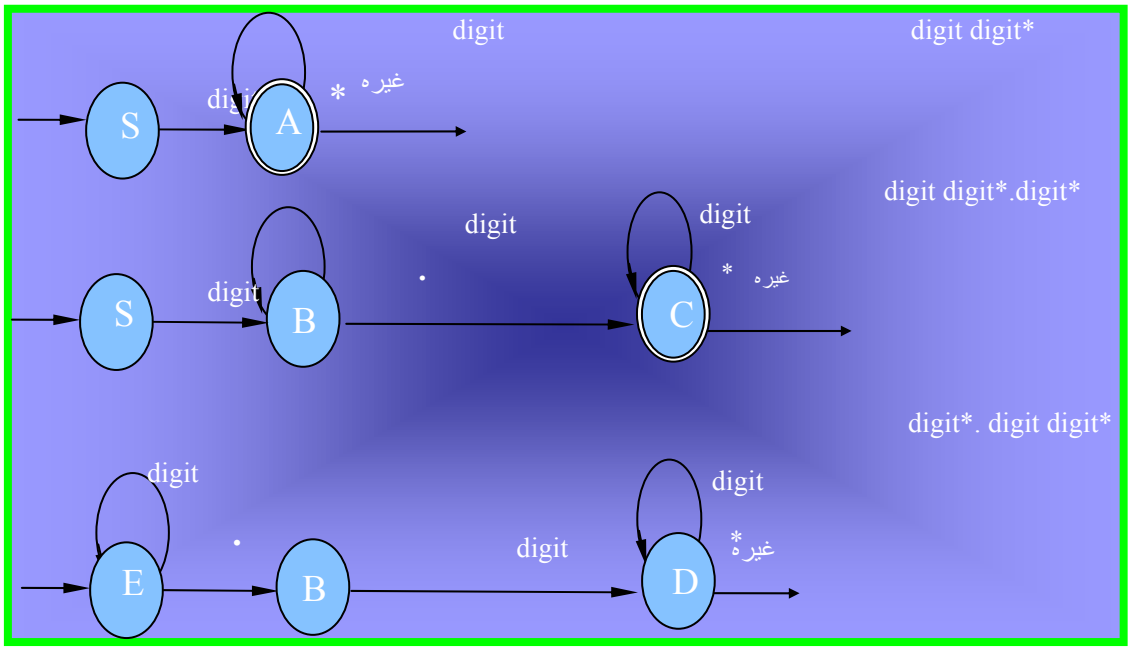
نكته ديگر، محيط ويرايش متن برنامه هاست . دراغلب محيط هاي ارائه شده براي كامپايلرها، **Syntax Directed Editors** ويرايشگرهايي مي باشند كه بنابر قوانين زبان به برنامه ساز كمك مي كنند تا بتواند برنامه خود را با Syntax صحيح ايجاد كند . ميتوان يك محيط قوي ويرايش وابسته به زبان مورد نظر براي ويرايش برنامه هاي فارسي ايجاد نمود.

به اين ترتيب ميتوان برنامه ها را به زبان فارسي با استفاده از يك ويرايشگر باهوش براي زبان خاص نوشت . سپس با استفاده از جدول تبديل برنامه به زبان اصلي ترجمه و پس از اشكالزدايي دوباره از زبان اصلي با استفاده از جدول تبديل لغات را به فارسي برگرداند.

۲.۹ ماشينهاى خودكار غير قطعى

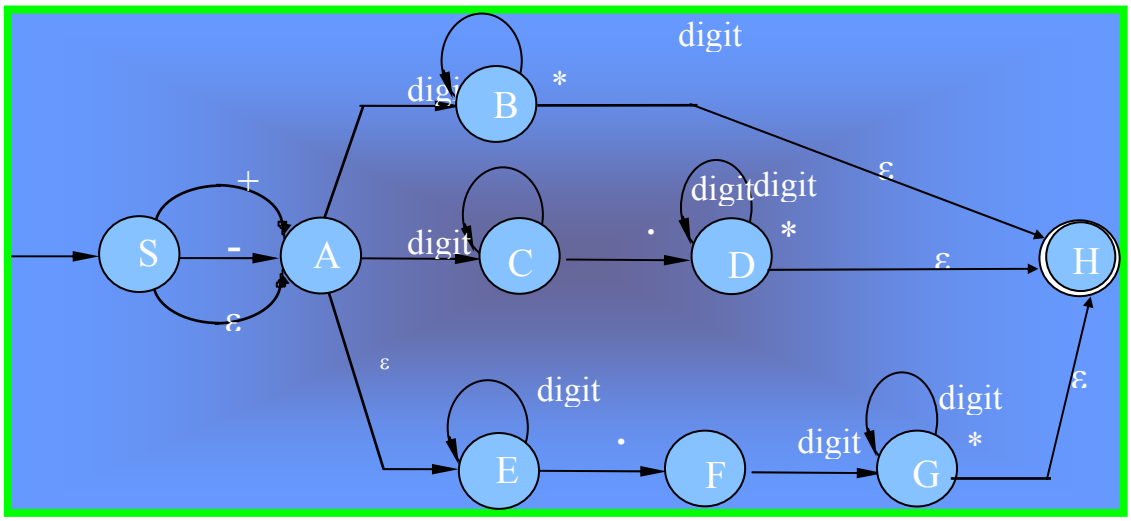
اصولا يك ماشين خودكار در واقع ابزاري براي نمايش چگونگى تصميم گيريهاست . ماشين خودكار يا بصورت قطعي تصميم گيريها را انجام مي دهد يا غير قطعي. اگر به ماشينهاي خودكاري كه تا به حال ترسيم شده توجه كنيم ، مشخص است كه در هر حالتي بطور قطعي مشخص ميكنند كه به ازاي ديدن يك رشته در ورودي از يك حالت به كدام حالت مي بايست گذر نمود . ماشينهاي خودكار قطعي را اصطلاحا" Dterermisitic Finite State Automata گويند.

در ماشينهاي غير قطعي يا در اصطلاح NonDeterministic ، عدم قطعيت در تصميمات وجود دارد . البته اين عدم قطعيت اگرچه كه كار تصميم گيري را مشكل ميكند، اما كار ايجاد ماشين را بسيار ساده تر مي نمايد. حالا، نكته در اينجاست كه چگونه ميتوان يك ماشين خودكار غير قطعي را بصورت قطعي تبديل نمود. در واقع زيبايي الگوريتمها ها در اين زمينه تبلور خود را نشان مي دهد. براي اين منظور اعداد را در نظر بگيريد . بطور مجزا مي توان گفت كه در حالت كلي سه فرم براي اعداد صحيح واعشاري مي تواند وجود داشته باشد :



شكل 2.8- انواع اعداد صحيح و اعشاري

در اينجا ترسيم سه ماشين خودكار براي اعداد مستقل از يكديگر ساده است . مشكل ، تركيب اين سه حالت و ايجاد يك ماشين خودكار براي سه نوع عدد مي باشد. با استفاده از ماشينهاي خودكار غير قطعي ميتوان به سادگي و بدون در نظر گرفتن مشكل تركيب گراف براي سه نوع متفاوت اعداد اقدام به ترسيم يك ماشين خودكار براي تشخيص سه نوع عدد نمود. زيبايي كار در اينجاست كه ميتوان بصورت الگوريتمي و بدون نياز به هيچگونه كار اضافي ، اين سه حالت را در قالب يك ماشين خودكار غير قطعي با يكديگر ادغام نمود و با استفاده از الگوريتمهايي كه ارائه خواهد شد بطور اتوماتيك تبديل به يك ماشين خودكار قطعي نمود.



شكل 2.9- ماشين خودكار غير قطعي

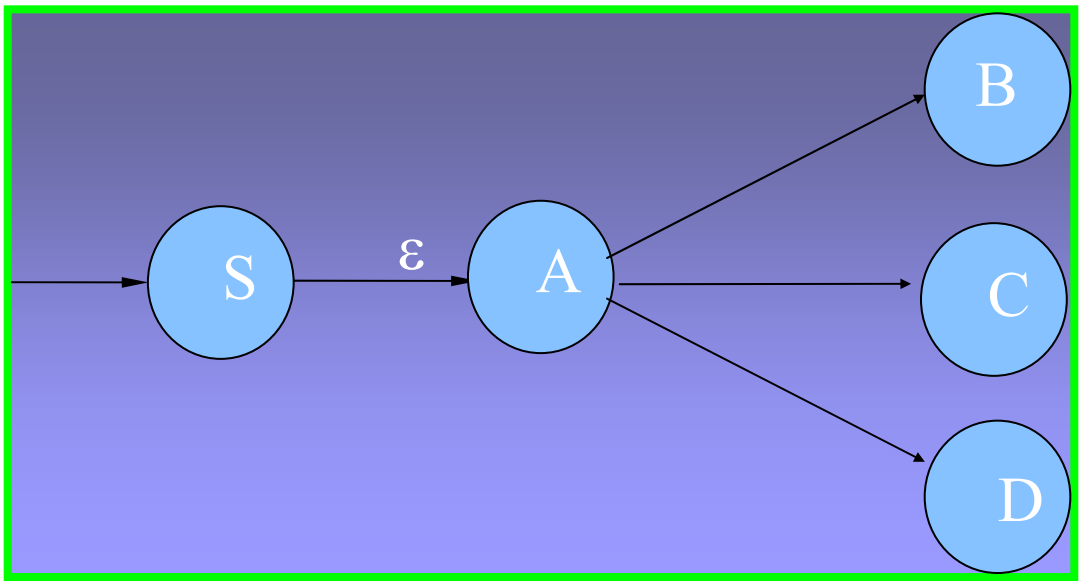
همانگونه كه در شكل 2.9 مشاهده مي شود ، يك ماشين خودكار غير قطعي مي تواند بيان گر حالات مختلف براي يك لغت باشد. در اينجا منظور از لغت ، اعداد مي باشند . اعداد يا داراي علامت هستند و يا علامت ندارند. اين نداشتن علامت را با گذر تهي كه با علامت ε (اپسيلون) مشخص شده ، معين مي كنند. همانگونه كه مشاهده مي شود، در حالت A با ديدن digit نمي توان مشخص نمود كه حالت بعدي آيا حالت B يا C ويا حالت E است . بايد توجه داشته باشيد كه گذر ε در واقع يعني هيچ چيز .

براي تبديل ماشين خودكار غير قطعي به قطعي در سه مرحله عمل ميشود:

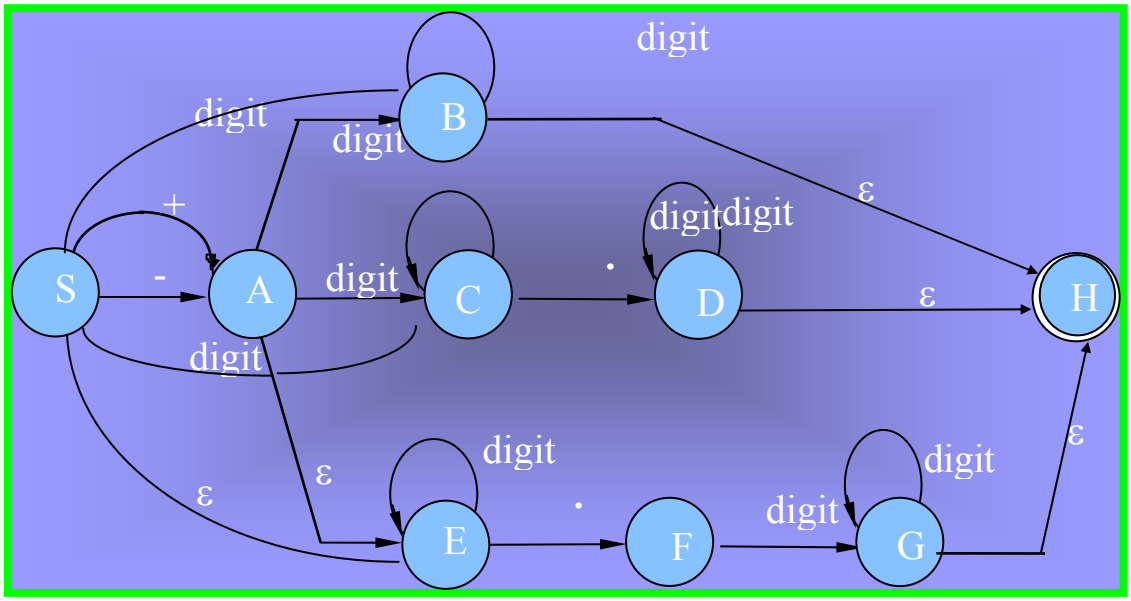
1. حذف گذرهاى تهي
2. رفع عدم قطعيت
3. بهينه سازي ماشين خودكار

۲.۹.۱ حذف گذرهای تهی

اگر از حالت S گذری تهي به حالت A وجود دارد، اين نمايانگر يك تساوي يك طرفه مي باشد كه نكته ايست جالب توجه . در اينجا S با A هيچ تفاوتي ندارد اما A متمايز از S است. ميتوان هر واكنشي كه در A وجود دارد را براي S نيز در نظر گرفت اما بالعكس صادق نيست.

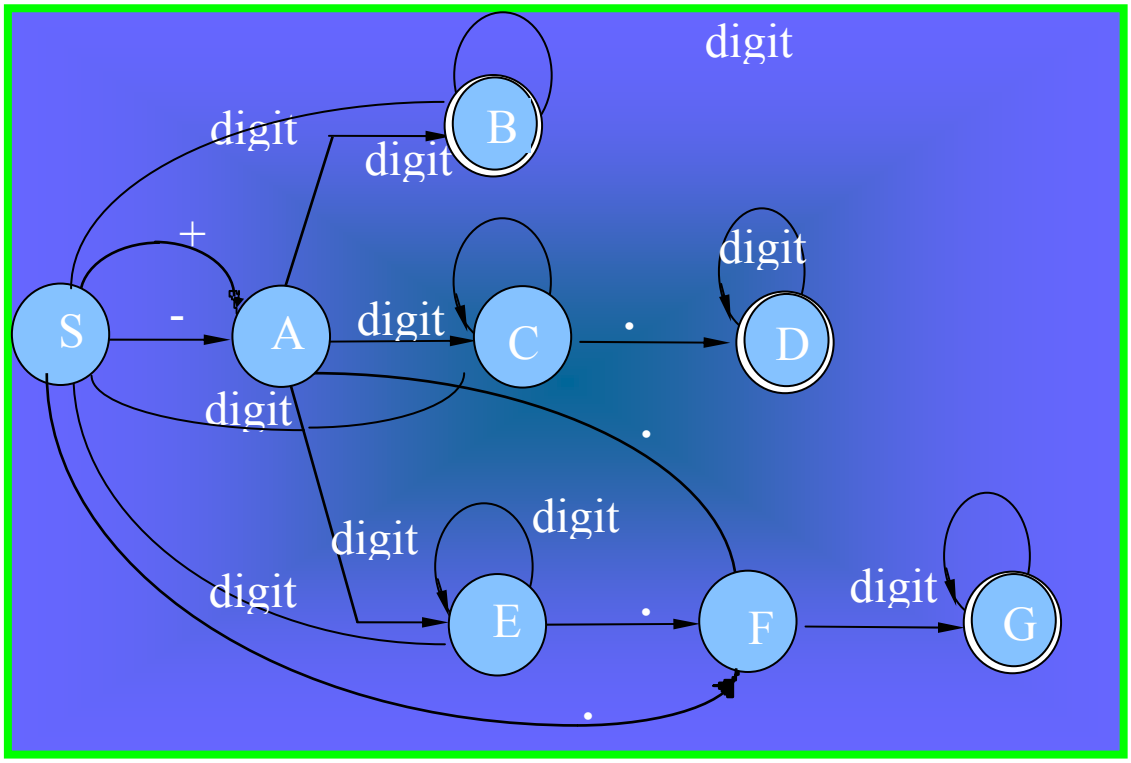


براي درك بهتر مطلب ، گذر تهي از حالت صفر به حالت يك در شكل 2.9 را در نظر بگيريد اگر عدد علامت نداشته باشد، حالت صفر دقيقاً مثل حالت يك است. اما، اگر عدد علامت داشته باشد ، حالت صفر واكنشي ديگر دارد كه واكنش اين حالت ، گـذري به حالت يك مي باشد. حالت صفر از حالت يك متمايز گرديد ، چرا كه ممكن بود عدد داراي علامت باشد، اما اگر عدد علامتي نداشته باشد اين دو حالت يكي خواهند بود. پس هيچگونه نيازي به جداسازي آنها نخواهد بود .



شكل 2.10 حذف گذر تهي از حالت صفر به يك

همانگونه كه مشاهده مي شود، كليه واكنشهاي حالت يك و يا به عبارت ديگر كليه گذرهاي خارج شونده از حالت يك ، براي حالت صفر در نظر گرفته شد . به اين ترتيب گذر تهي حذف گرديد. در مرحله بعد به همين ترتيب ادامه مي دهيم تا كليه گذرهاي تهي را از داخل ماشين خودكار قطع نما ييم .

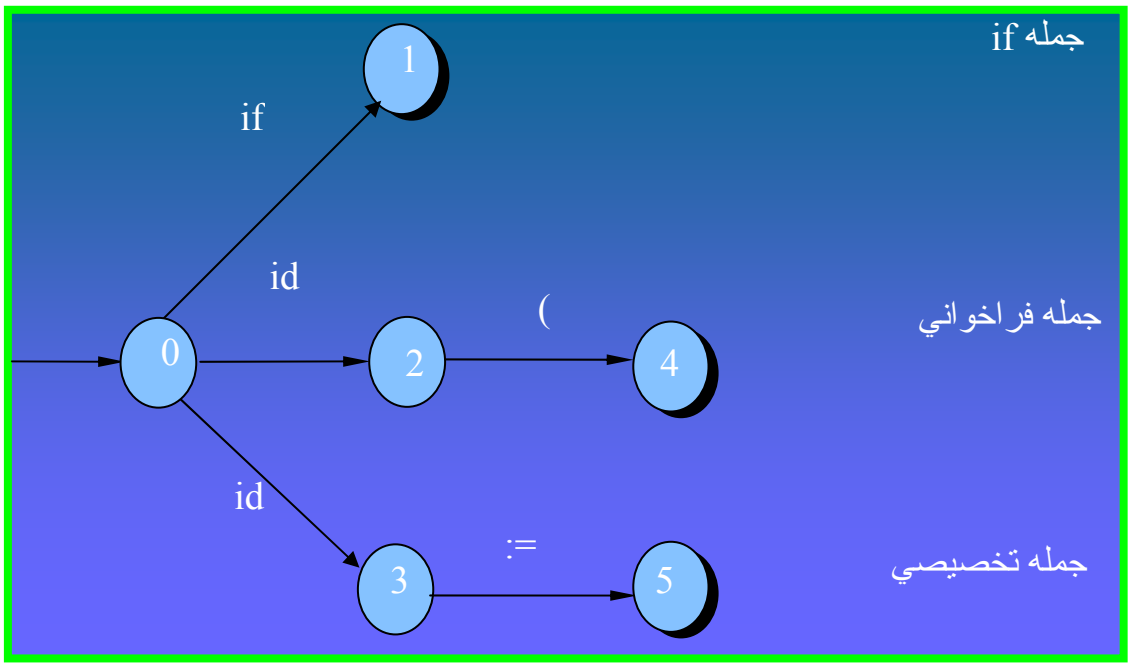


شكل 2.11- ماشين خودكار بدون گذرهاي تهي

۲.۹.۲ رفع عدم قطعيت

همانگونه كه در شكل **2.11** مشاهده مي كنيد، ماشين خودكار غير قطعي است. براي نمونه در حالت A با ديدن digit نمي توان تشخيص داد كه آيا حالت بعدي **B** ، **C** و يا اينكه **E** مي باشد. اما نكته جالب توجه اينجاست كه با ديدن digit حتماً به يكي از اين **سه** حالت گذر مي شود . براي روشن تر شدن روش حذف عدم قطعيت بهتر است كه كار تحليلگر نحوي مطرح شود .

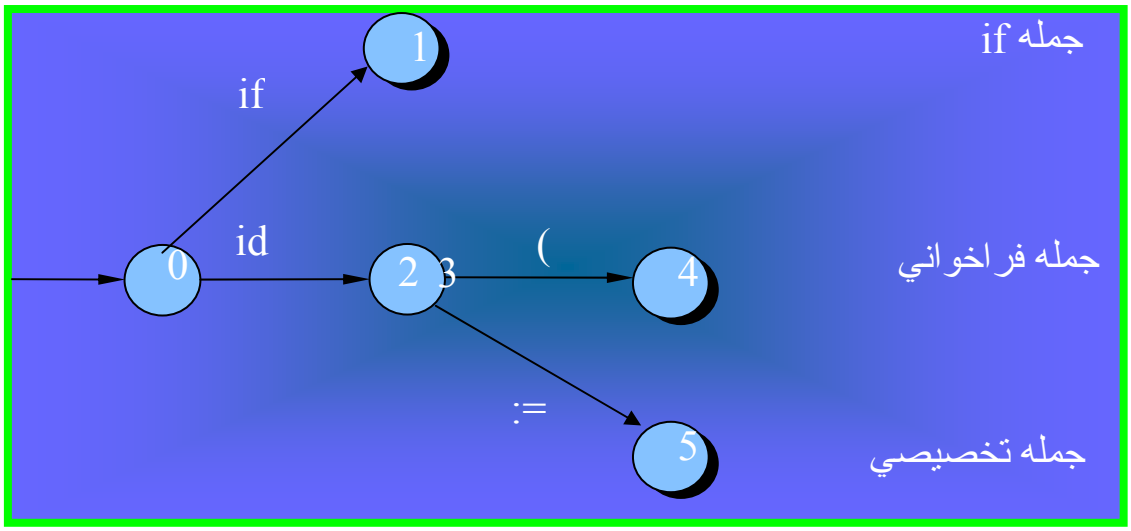
تحليلگر نحوي ، معمولاً با ديدن يك لغت در ورودي ، ساختار جمله مورد نظر را مي تواند پيش بيني كند. براي مثال اگر لغت دريافتي از تحليلگر لغوي **if** باشد، تحليلگر نحوي بلافاصله تصميم مي گيرد كه جمله بايد جمله **if** باشد و بايد انتظار ديدن شرط جمله **if** را در ورودي داشته باشد. اما، با ديدن يك اسم ، يا به عبارت ديگر يك شناسه ، مشكل وجود دارد. در اينجا تحليلگر نحوي نمي تواند مشخص كند كه آيا جمله مورد نظر يك جمله **فراخواني** زير برنامه ها ويا يك جمله **تخصيصي** (**Assignment**) است. ماشين خودكار غير قطعي در اينجا بصورت زير قابل ترسيم است :



شكل 2.12 تشخيص جملات if ، فراخواني و تخصيصي

عليرغم وجود عدم قطعيت در حالت صفر از شكل 2.12 ، مي توان قطعيتي را در نظر داشت. به اين ترتيب كه در حالت شروع صفر اگر يك شناسه يا id در

ورودي ظاهر شود، قطعاً حالت بعدي ، حالت 2 يا 3 خواهد بود. در حالت 2 يا 3 اگر ':=' در ورودي ظاهرشود جمله تخصيصي ، اگر '(' ظاهر شود جمله فراخواني و در غير اينصورت جمله غلط مي باشد. پس حالات 2 يا 3 را بصورت يك حالت تركيبي جديد با ادغام خروجي هاي اين دو حالت مي توان بوجود آورد :



شكل 2.13 ماشين خودكار قطعي

براي سهولت در امر تبديل ويا نمايش ماشينهاي خودكار به جاي گراف از فرم جدول مانند استفاده ميشود. براي نمونه ماشين خودكارغير قطعي اعداد در شكل **2.11** ، بصورت يك جدول در شكل 2.14 مشخص شده است.

	digit	.	+/-
S	B,C,E	F	A
A	B,C,E	F	
(B)	B		
C	C	D	
(D)	D		
E	E	F	
F	G		
(G)	G		

شكل 2.14 جدول ماشين خودكار اعداد

همانگونه كه در شكل 2.14 مشاهده ميشود در حالات S و A به يكي از سه حالت **B** يا **C** يا **E** به ازاي ديدن digit گذري وجود دارد. لذا ، ميتوان گفت كه گذري به حالت تركيبي BCE وجود دارد. در اين حالت ميتوان واكنشهاي هر سه حالت B ، C و E را داشت. رديف مربوط به حالت تركيبي BCE در شكل 2.15 ، تركيبي از واكنشهاي هر سه حالت تشكيل دهنده آن است به عبارت ديگر از تركيب "يا"

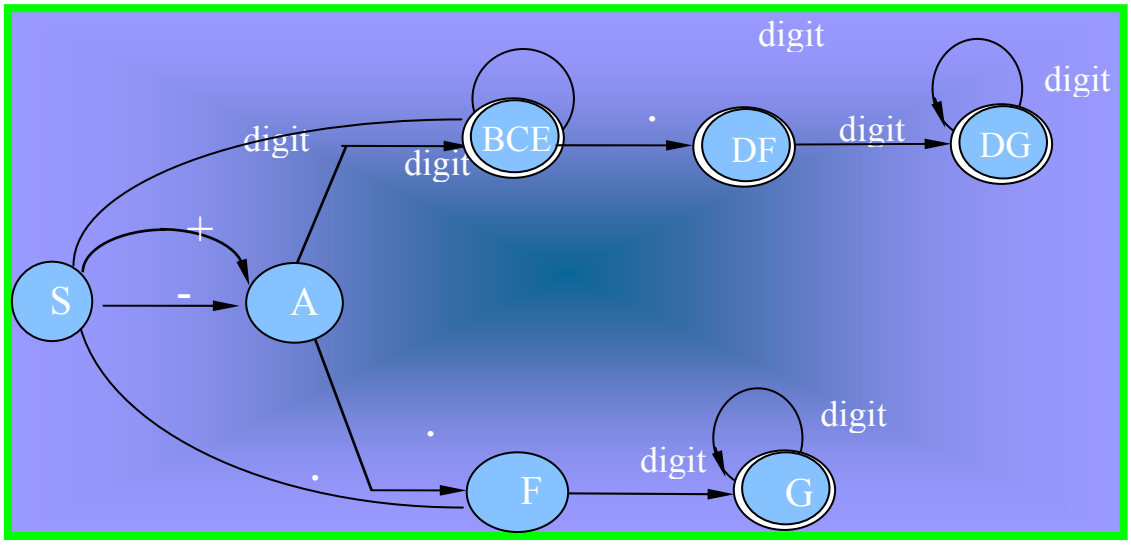
سطرهاي B ، C و E، سطر جديد BCE به ماتريس افزوده مي شود . BCE يك حالت پذيرشي است ، زيرا يكي از حالات تشكيل دهنده آن يعني B يك حالت پذيرش است .

همانگونه كه در شكل 2.15 مشاهده ميشود ، در حالت جديد BCE به ازاء ورودي '.' (نقطه اعشار) عدم قطعيت وجود دارد . از حالت BCE ميتوان به هر يك از دو حالت D يا F گذر نمود. لذا براي رفع عدم قطعيت، حالت جديد ديگري بنام DF را به جدول حالات بايد افزود. براي رفع عدم قطعيت در حالت DF حالت DG ايجاد ميشود.

	digit	.	+/-
S	BCE	F	A
A	BCE	F	
(B)	B		
C	C	D	
(D)	D		
E	E	F	
F	G		
(G)	G		
(BCE)	BCE	DF	
(DF)	DG		
(DG)	DG		

شكل 2.15 جدول ماشين خودكار اعداد

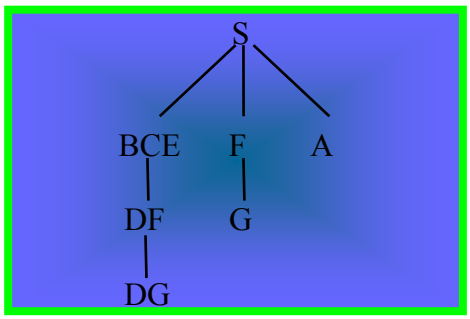
اكنون با استفاده از جدول فوق ميتوان ماشين خودكار قطعي اعداد را بصورت زير ايجاد نمود.



شكل 2.16 ماشين خودكار قطعي اعداد

همانگونه كه مشاهده ميكنيد برخي از حالات درون جدول مثل حالات B ، C ، D و E در عمل غير قابل دسترسي هستند. علت افزايش حالات جديد تركيبي است. ميتوان اين حالات زااد را با ايجاد درخت دسترسي نيز مشخص نمود.

با افزايش حالات جديد جهت حذف عدم قطعيت در ماشين خودكار ، برخي از حالات زائد شده ، از حالت شروع غير قابل دسترسي ميشوند . جهت تشخيص اين حالات يا مي توان ماشين خودكار را از روي جدول ترسيم نمود و يا اينكه با استفاده از يك درخت دسترسي حالات قابل دسترسي از حالت شروع S را مشخص نمود. براي نمونه درخت دسترسي براي ماتريس شكل 2.15 بصورت زير است. بايد توجه داشته باشيد كه نام حالات در درخت دسترسي تكراري نيست و اين درخت صرفا"جهت تعيين حالات قابل دسترسي ازحالت شروع S است.



شكل 2.17 درخت حالات قابل دسترسي

۲.۹.۳ بهينه سازى ماشينهاى خودكار

هدف از بهينه سازي ، تقليل تعداد حالات در ماشينهاي خودكار است . براي اين منظور بايد حالات معادل را تشخيص داد . دو حالت را در صورتي معادل گويند كه به ازاي ورورديهاي متفاوت با گذشت از يك سري از حالات يا به عبارت ديگر در مسيري به طول صفر يا بيشتر و با گـذشتن از تعداد **n** گره نهايتا" به حالت پذيرش برسند. حالات معادل را با روشي ابتكاري به ترتيب زير مي توان مشخص نمود :

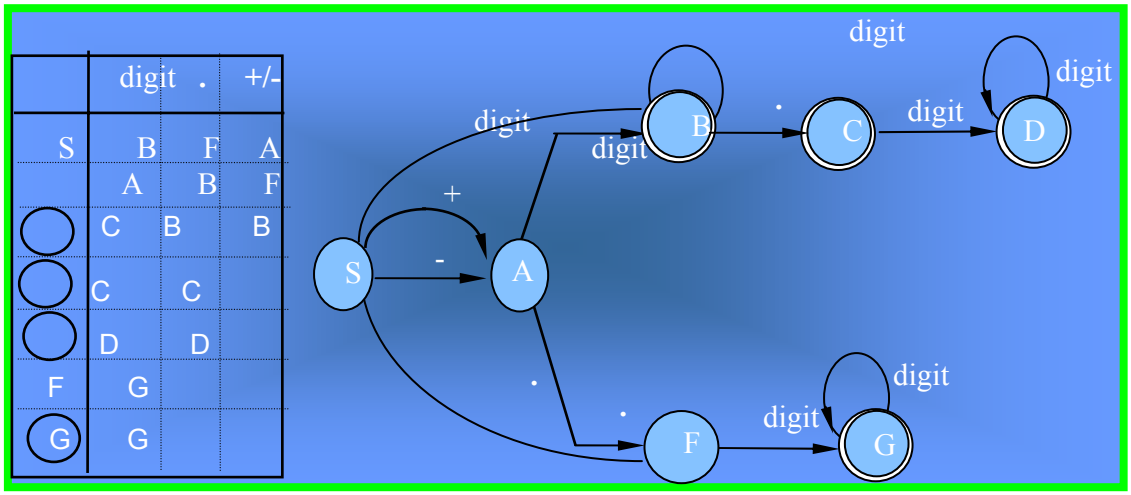
1. ابتدا حالات معادل صفر كه با مسيرهائـي به طول صفر به پذيرش ميرسند را از ساير حالات مجزا بايد نمود. بعبارت ديگر ، حالات پذيرش از حالات غير پذيرش مجزا مي شوند. به اين ترتيب حالات به دو دسته مجزا از حالات پذيرش و غير پذيرش تقسيم ميشوند.
2. به درون هر دسته حالات بايد نگريست ومشخص نمود كه آيا ورودي اي وجود دارد كه به ازاي آن حالات متعلق به يك دسته واكنشهای جدا از سايرين دارند. واكنشها بر اساس گذر يك حالت به حالت ديگر سنجيده نمي شود بلكه ، اگر از يك حالت به حالت ديگر گذري وجود دارد دسته مربوط به آن حالت را مشخص ميكنند. مقصود مقايسه دسته هاإی است كه به آنها گذر ميشود. حالات مشابه به ازا يك ورودي به حالات درون يك دسته گذر مي كنند .
3. آنقدر مر حله **2** تكرار مي شود تا ديگر گذري متفاوت بين دسته ها وجود نداشته باشد ودسته اي جديدتر توليد نشود . براي نمونه اكنون ماشين خودكار كه در مبحث قبل مطرح شد بهينه سازي مي شود.

بطور خلاصه و با در نظر گرفتن الگوريتم فوق مي توان حالات معادل را بصورت زير تعريف نمود :

دو حالت را در صورتي معادل **k** گويند كه به ازاي هر رشته از ورورديها به طول كوچكتر يا مساوي با **k** اگر از يكي از آنها بتوان به پذيرش رسيد ، از ديگري نيز بتوان به ازاء همان رشته به حالت پذيرش رسيد.

دو حالت را معادل گويند اگر و تنها فقط اگر به ازاي هر رشته اي از ورورديهاي متوالي كه موجب رسيدن از آن حالت به يك حالت پذيرش است بتوان از ديگري نيز به پذيرش رسيد.

اكنون با توجه به تعريف حالات معادل و مراحل بهينه سازي ماشينهاي خودكار ، مي توان مبادرت به بهينه سازي ماشين خودكار اعداد كه در مبحث قبل مطرح شد ، نمود. ماشين خودكار قطعي اعداد و جدول توليد آن در شكل 2.18 ارائه شده است.



شكل 2.17- ماشين خودكارقطعي اعداد

براي بهينه سازي ماشينهاي خودكار ابتدا بايد گره ها را به دو دسته پذيرشی و غير پذيرشی افراز نمود . بعبارت ديگر حالات معادل صفركه با مسيرهايي به طول صفر به پذيرش مي رسند را از ساير حالات بايد تفكيك نمود. به اين ترتيب حالات ماشين خودكار اعداد كه در شكل 2.17 ارائه شده ، به دو دسته زير تقسيم ميشوند:

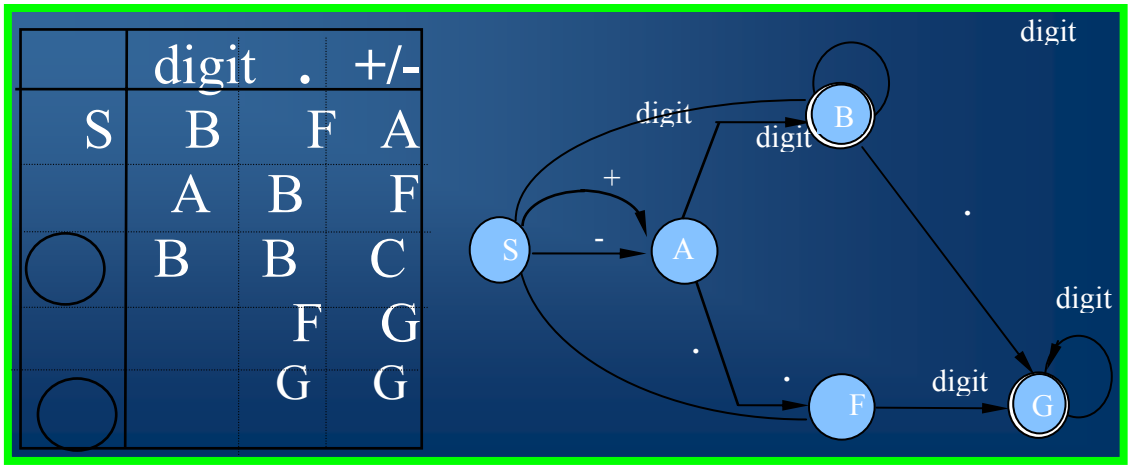
1	{B,C,D,G}	حالات پذيرشی
2	{S,A,F}	حالات غير پذيرشی

حال بايد مشخص نمود كه آيا در داخل هر دسته به ازاء هر ورودي گره ها واكنشي يكسان دارند . منظور از واكنش يكسان اين است كه براي نمونه اگر از حالت M در دسته شماره 12 به ازاء ورودي digit گذري به حالتي در دسته شماره 7 وجود دارد ساير حالات موجود در دسته شماره 12 بايد به ازاء ورودي digit گذري به حالتي در دسته شماره 7 داشته باشند. در غير اينصورت اين حالات از داخل دسته خارج شده و در دسته اي جديد قرار مي‌گيرند. براي نمونه در داخل دسته شماره 1 در بالا به ازاء ورودي نقطه اعشار '.' از حالت B يك خروجي و گذري وجود دارد در صورتيكه در مورد ساير حالات اينچنين نيست. پس اين حالت از سايرين جدا ميشود و سه دسته به صورت زير حاصل مي گردد:

1	{ C ,D ,G}	
2	(B)	
3	{S ,A ,F}	حالات غير پذيرشی

به همين ترتيب در دسته شماره 3 حالت S قابل تفكيك از A و F است. زيرا در S به ازاء ورودیهاي + و - گذر و واكنشي وجود دارد در صورتيكه در مورد A و F

اينچنين نيست. پس حالت S از A و F قابل تفكيك است. از سوي ديگر A و F نيز قابل تفكيك از يكديگر هستند. بنابراين سه حالت C ، D و G غير قابل تفكيك از يكديگر و معادل هستند. پس ميتوان هر يك از اين سه حالت معادل را بجاي دو حالت ديگر در داخل ماشين خودكار قرار داد. به اين ترتيب ماشين خودكار بهينه اعداد بصورت زير خواهد بود.



شكل 2.18- ماشين خودكاربهينه اعداد

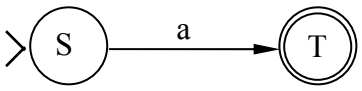
۲.۱۰ تبديل عبارات با قاعده به ماشينهاى خودكار

در اين قسمت نشان داده خواهد شد كه چگونه مي توان عبارات با قاعده را به صورت ماشينهاي خودكار تبديل نمود تا اينكه بتوان با استفاده از ماشين خودكار، قوانين لغوي كه در فرم عبارات باقاعده خلاصه شده اند را عملا" به صورت كد برنامه مورد بهره برداري قرار دادء براي تبديل عبارات با قاعده به ماشينهاي خودكار معمولا" مراحل زير بكار مي روند :

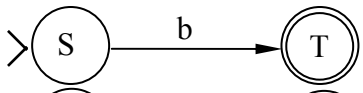
1 . هر عنصر **a** متعلق به الفباي زبان به صورت يك ماشين خودكار نمايش داده مي شود :

2 . ماشين خودكار براي عبارت **ab** را از تركيب گرافها براي **a** و **b** به صورت زير مي توان ايجاد نمود.

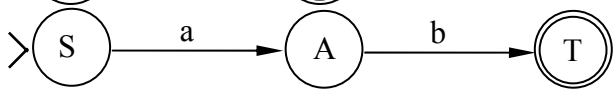
(الف) ماشين خودكار براي a



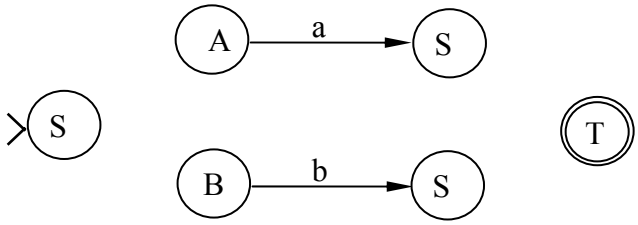
(ب) ماشين خودكار براي b



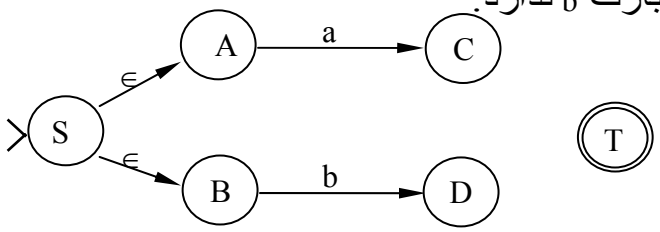
(ج) ماشين خودكار براي ab



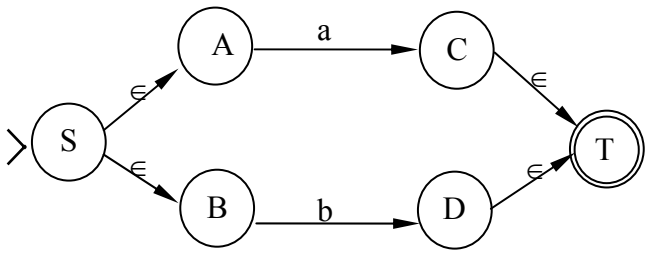
3 . ماشين خودكار براي عبارت **a | b** را با يك استدلال ساده ميتوان از ماشين هاي خودكار براي عبارات a و b ايجاد نمود. چنانچه حالت شروع ماشين خودكار براي عبارت **a | b** حالت S باشد :



حالت شروع S هيچ فرقي با حالت شروع براي عبارت a ندارد زيرا در شروع a | b ميتوان a را هم در ورودي ديد. از سوي ديگر حالت شروع S هيچ تفاوتي با حالت شروع براي عبارت b ندارد.



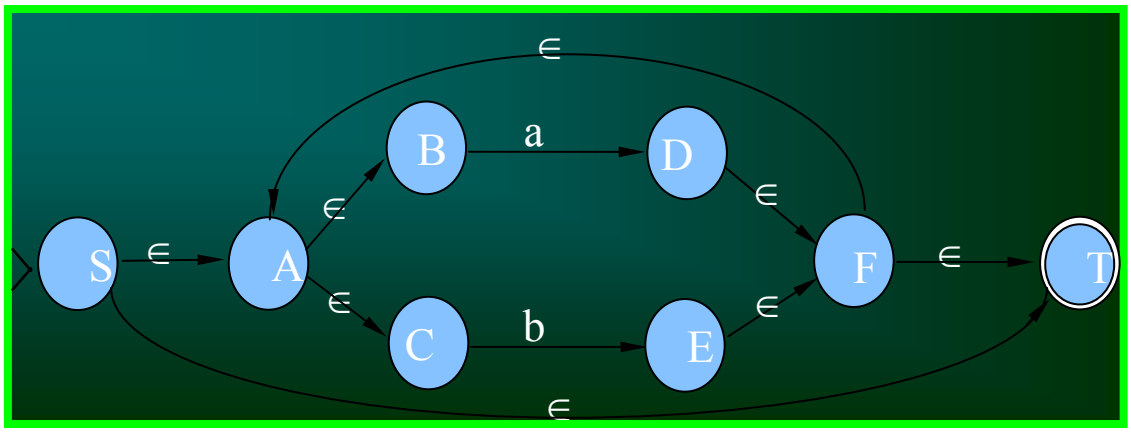
به همين ترتيب با مشاهده a در ورودي بايد مطمئن بود كه a|b در ورودي ظاهر شده است. لذا ، حالت پذيرش براي ماشين خودكار عبارت a يعني C هيئ تفاوتي با حالت پذيرش براي ماشين خودكار براي عبارت a | b يعني T ندارد. به همين ترتيب حالت D هيچ تفاوتي با حالت پذيرش T ندارد. بنابراين ماشين خودكار براي عبارت a | b بصورت زير خواهد بود:



در شكل فوق عبارت **a | b** از تركيب ماشينهاي خودكار براي عبارات **a** و **b** ساخته شده است. مسلم است كه حالت شروع S هيچ تفاوتي با حالت A وحالت B ندارد. اما بالعكس صادق نيست و حالت شروع براي تشخيص a يعني A متفاوت از حالت شروع براي a | b است ء زيرا در حالت شروع براي a نمي توان در ورودي b را ديد. لذا، حالت شروع a | b را با دو گذر تهي به حالت شروع براي a و حالت شروع براي b متصل بايد نمود.

4 . با در دست داشتن ماشين خودكار براي a و b و درنتيجه وجود ماشين خودكار براي a | b مي توان ماشين خودكار براي عبارت (a | b)* را با يك استدلال ساده ايجاد نمود. به اين ترتيب كه فرض كنيد حالات شروع و خاتمه براي ماشين خودكار (a | b)* به ترتيب حالات S و T باشند . اولا" ، چون (a | b)* ميتواند تهي نيز باشد پس حالت شروع آن ممكن است هيچ تفاوتي با حالت پذيرش نداشته باشد. زيرا رشته (a | b)* ممكن است اصلاً وجود نداشته باشد.

از طرف ديگر ممكن است حالت شروع S هيئ تفوتي با حالت شروع (a | b) نداشته باشد زيرا ، در شروع (a | b)* ميتوان در شروع مشاهده a | b در ورودي بود. پس از مشاهده a | b دو باره ميتوان در آغاز مشاهده a | b جديدتري قرار گرفت . اين در واقع بخاطر خاصيت تكراري بودن (a | b)* است.پس ، حالت پذيرش a | b درون (a | b)* ، هيچ تفاوتي با حالت شروع آن نخواهد داشت. با اين استدلال ميتوان نتيجه گرفت كه ماشين خودكار براي عبارت (a | b)* بصورت زير است.

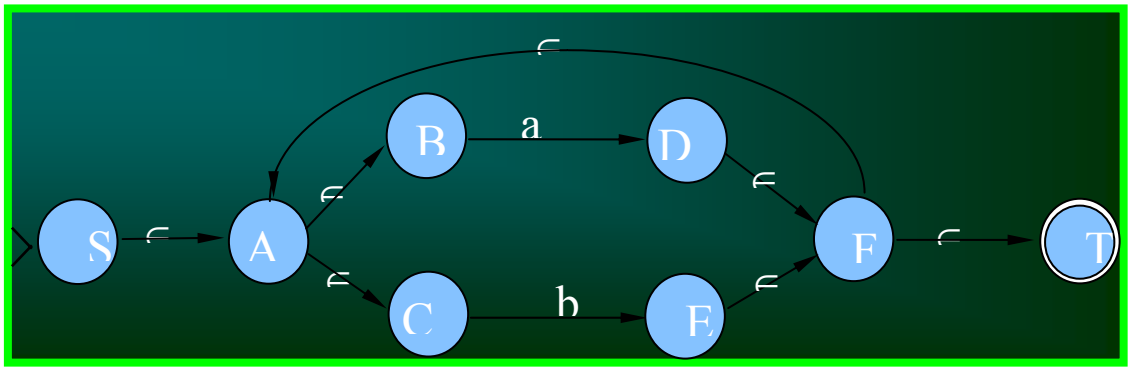


شكل 2.19- ماشين خودكار براي عبارت (a | b)*

مسلما" در ورودي (a | b)* ئون رشته a يا b مي تواند اصلا" وجود نداشته باشد با اين نتيجه حالت شروع مي تواند معادل با حالت پذيرش باشد. لذا ، در شكل 2.19 حالت شروع S با گذري تهي به حالت پذيرش T متصل شده است. از جهت ديگر در خاتمه ديدن a | b مثل اينكه دوباره در حالت شروع بوده و مي توان a | b

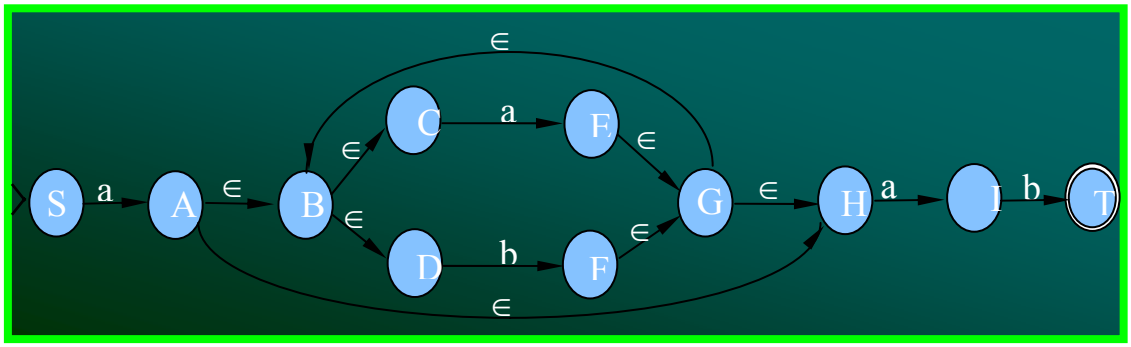
جديدي را مشاهده نمود و اين عمل تا به هر تعدادي قابل تكرار است . براي اين منظور حالت F با گذري تهي به حالت شروع a | b يعني A متصل شده است. علاوه بر اين حالت F ممكن است خاتمه تكرار و حالت پذيرش هم باشد.

5 . عبارت $(a \mid b)^+$ را مي توان با حذف گذر تهي از حالت شروع به حالت پايان در ماشين خودكار براي $(a \mid b)^*$ توليد نمود زيرا درمورد $(a \mid b)^+$ حداقل يكبار a | b در ورودي بايد ظاهر شود و حالت شروع آن با خاتمه يكسان نيست .



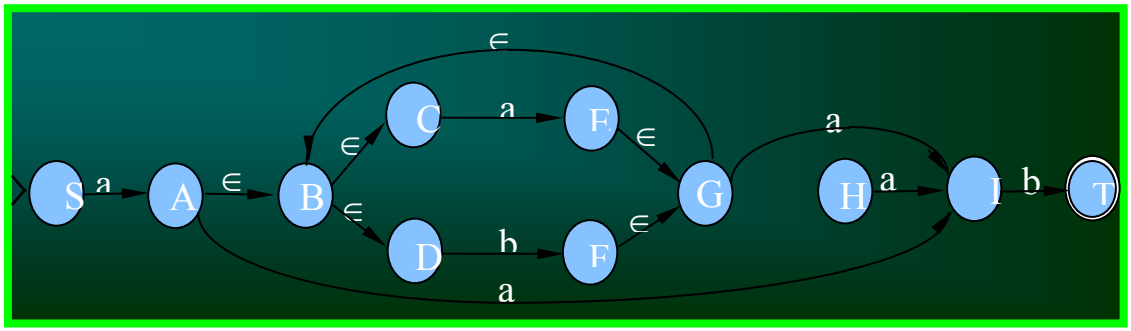
شكل 2.20- ماشين خودكار براي عبارت $(a \mid b)^+$

مثال 1- براي عبارت a (a | b)*a b يك ماشين خودكار بهينه ايجاد نماييد.
با استفاده از ماشين خودكار ارائه شده براي عبارات (a | b)* كه در شكل 2.19 ارائه شده است ، ميتوان بسادگی ماشين خودكار غير قطعي را ايجاد كرد.



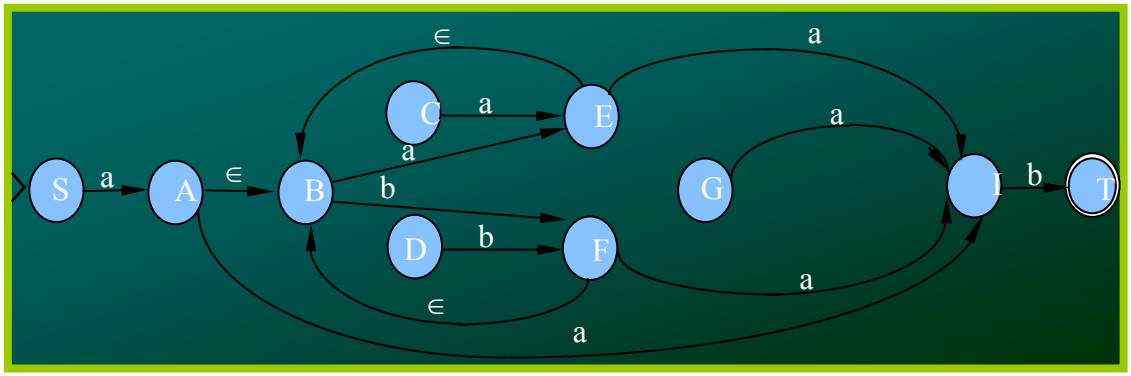
شكل 2.21- ماشين خودكار غير قطعي براي عبارت

پس از حذف گذرهاي تهي به H ماشين بصورت زير تبديل ميشود.



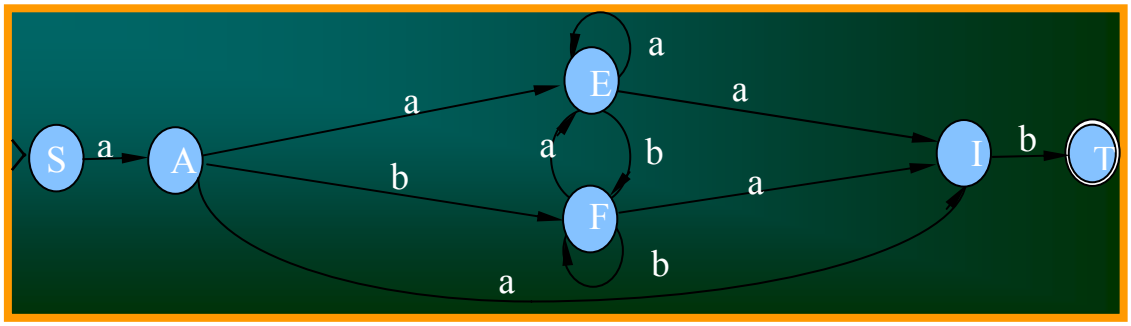
شكل 2.22- حذف گذرهاي تهي از A و G به H

مشاهده ميكنيد كه حالت H در شكل 2.22 غير قابل دسترسي و قابل حذف است. در شكل 2.23 گذرهاي تهي به حالات G ، C و D حذف شده اند.



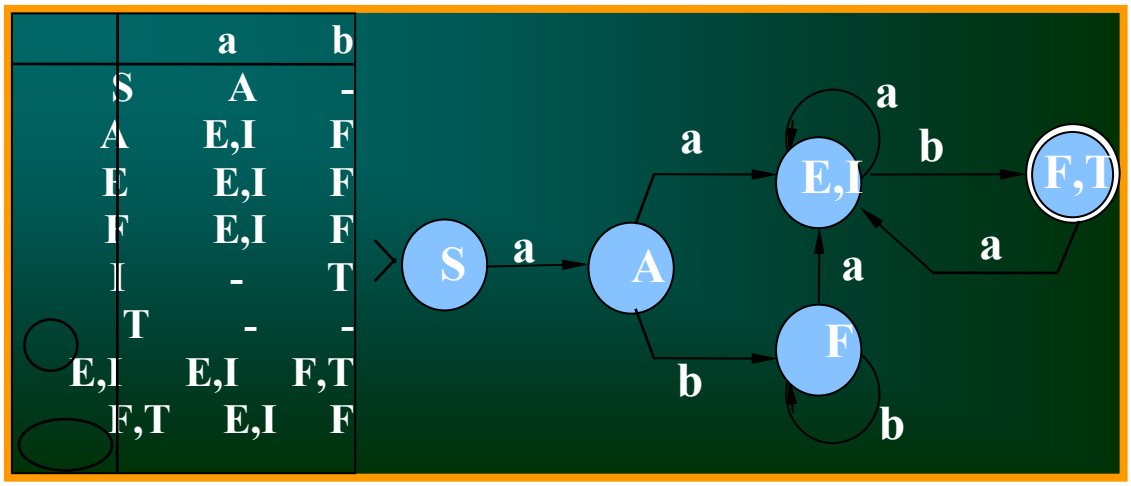
شكل 2.23- حذف گذرهاي تهي به حالات G ، C و D

پس از حذف گذرهاي تهي ماشين بصورت زير تبديل ميشود.



شكل 2.24- ماشين خودكارپس از حذف گذرهاي تهي

ماشين خودكار فوق در شكل 2.25 با استفاده از نمايش ماتريسي به فرم قطعي تبديل شده است.



شكل 2.25- ماشين خودكار قطعي براي عبارت a(a|b)*ab

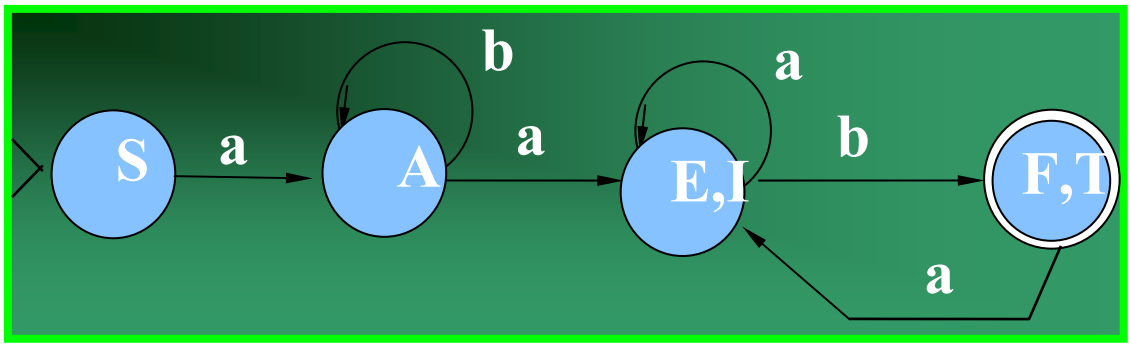
براي بهينه سازي ماشين خودكار فوق حالات مربوطه را به دو دسته پذيرشي و غير پذيرشي بصورت زير ميتوان تقسيم بندي نمود:

الف) 1- {FT} 2- {S, A, EI, F}

ب) 1- {FT} 2- {S} 3-{ A, EI, F}

ج) 1- {FT} 2- {S} 3-{ A, F} 4- {EI}

به اين ترتيب مشاهده ميكنيد كه حالات A و F با يكديگر معادل هستند و يكي را ميتوان با ديگري جايگزين نمود. به اين ترتيب ماشين خودكار بهينه براي عبارت a(a|b)*ab بصورت زير خواهد بود.



۲.۱۰ تمرين

تمرين 1- فرم كلي اعدادي را مشخص كنيد كه يا صفرمي باشندويا اينكه اگرطولشان بيش از يك است حتما" با يك رقم غير صفرآغاز مي شوند.

تمرين 2- فرم كلي اعداد در مبناي هشت را مشخص كنيددر صورتي كه بدانيم اينگونه اعداد حتماً با يك رقم صفر آغاز و سپس حرف **O** ظاهرمي‌گردد و بعد از آن عدد مبناي هشت مشخص مي شود. براي نمونه 0O127 يك عد مبناي 8 است.

تمرين 3- فرم كلي اعداد در مبناي شانزده در زبان C با رقم صفر وسپس X آغاز مي شود. يك عبارت با قاعده براي بيان اعداد در مبناي شانزده ارايه نماييد . بطور مثال 0XAF25 يك عدد در مبناي 16 است.

تمرين 4- فرم كلي رشته ها را در زبان C را با يك عبارت با قاعده معين كنيد. توجه داشته باشيد كه در زبان **C** علامت \ (Backslash) در داخل يك رشته مفهوم خاص دارد.

تمرين 5- فرم كلي اعداد صحيحي را تعيين كنيد كه در آنها ارقام به ترتيب صعودي ظاهر ميشوند.

تمرين 6- فرم كلي كليه رشته اعداد مبناي دو را مشخص كنيد كه در آنها زير رشته 011 ظاهر نگردد.

تمرين 7 *- كليه رشته هاي اعداد مبناي دو را مشخص كنيد كه تعداد صفرها در آنها فرد و تعداد يك ها زوج باشد.

تمرين 8- ماشين خودكار قطعي براي شناسه ها ترسيم نمائيد

تمرين 9- ماشين خودكار قطعي بهينه براي عبارت (a | b | c)* a b b ايجاد نماإيد و سپس برنامه تحليلگر لغوي براي تشخيص اينگونه رشته ها بنويسيد. ابتدا عبارت را به فرم غير قطعي تبديل نمائيد سپس قطعي و بهينه نمائيد.

تمرين 10- يك ماشين خودكار قطعي براي كليه اعداد مبناي دوكه تعداد صفرهاي آنها فردو تعداد يك ها فرد است ايجاد كنيد. در ضمن لااقل دو عدد صفر و دو عدد يك بايد در اين اعداد بايد موجود باشد.

تمرين 11- ماشين خودكار قطعي براي اعداد مبناي 2 كه حتما" داراي يك رقم صفر و يك رقم يك هستند راايجاد نمائيد بقسمي كه اعداد با تعداد يك ها و صفرهاي زوج را در يك حالت پذيرشی مشخص نمايد . اعداد با تعداد يك هاي فرد و تعداد صفرهاي فرد را نيز در حالت پذيرش ديگر بايد ماشين مشخص كند. براي اعداد با تعداد يك ها فرد و صفرها زوج و همچنين صفرها فرد و يك ها زوج نيز بايد دو حالت پذيرش مجزا وجود داشته باشد. ابتدا براي هر يك از چهار حالت ذكر شده ماشينهاي خودكار را ترسيم نمائيد و سپس يك ماشين خودكار غير قطعي براي تركيب آنها ايجاد نمائيد. سپس اين ماشين خودكار را قطعي كنيد.

تمرين 12- يك ماشين خودكار قطعي براي گرامر يكي از زبانهاي C يا پاسكال ايجاد نمائيد. سپس با استفاده از روش ارائه شده در اين فصل يك تحليلگر لغوي براي آن ايجاد كنيد.