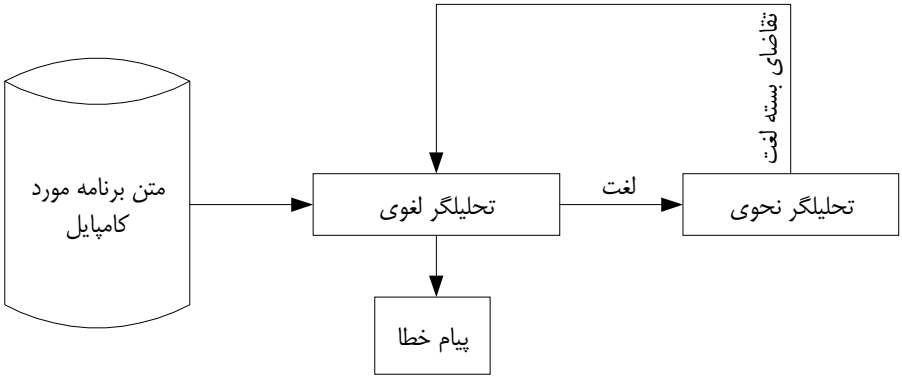
فصل دوم

تحلیلگر لغوی

۲-۱ مقدمه

تحلیلگر لغوی تابعی است که در ورودی خود متن برنامه مورد کامپایل را به عنوان یک فایل متن باز گشوده و کرکتر به کرکتر می خواند و با رسیدن به یک کرکتر جدا کننده لغت را تشخیص و تفکیک می‌نماید. جدا کننده ها مثل blank و tab یا new line فقط به منظور جدا سازی لغات به کار می‌روند. دسته دیگر از جدا کننده ها مثل semicolon یا + ارزش لغوی دارند. تحلیلگر لغوی در خروجی خود اطلاعات مربوط به لغت را در یک رکورد یا ساختار به نام بسته لغت قرار می‌دهد. بسته لغت شامل اطلاعاتی در مورد لغت تشخیص داده شده توسط تحلیلگر لغوی مانند سطر یا ستون و نوع لغت است که آن را در اصطلاح token گویند.



هر زبان برنامه سازی قوانین لغوی خاص خود را دارد. قوانین لغوی زبانها را می‌توان به صورت غیر مبهم و خلاصه در قالب نوعی خاص از عبارات به نام عبارت باقاعده بیان نمود. قوانین لغوی را نیز می‌توان توسط نوعی خاص از ساختار گراف به نام ماشین های خودکار مطرح کرد. ماشینهای خودکار ابزاری برای تبدیل

قوانین لغوی به کد برنامه هستند. چنانچه لغتی بنابر قوانین لغوی زبان برنامه سازی ایجاد نشده باشد، تحلیلگر لغوی اعلام خطا می‌نماید.

۲-۲ ساختار ورودی / خروجی

در داخل تحلیلگر لغوی برنامه مورد کامپایل به عنوان یک فایل متن باز می‌شود. این فـایل در ابتـدای برنامـه کامپایلر و خارج از محیط تحلیلگر لغوی تعریف و گشوده می‌شود و به عنوان یک پارامتر به تابع تحلیلگر لغوی ارسال می‌گردد. در قطعه کد ارائه شده در زیر چگونگی فراخوانی تابع تحلیلگر لغوی نشان داده شده است.

```
Viod main(int arg c, char *argv[])
{
        file * intext;
        if (argc <2)
        {
        clrscr();
        textcolor(red);
        gotoxy(10,5);
        cprintf(“نام برنامه مورد کامپایل فراموش شده است ”);
        if (!get()) getch();
        }
        intext=fopen(argv[ 1],”R”);
        while ! feof(intext)
        {
        token-type token;
        token=lexer(intext);
}
```

نقطه گنگ در مثال TokenType و در واقع نوع خروجی تحلیلگر لغوی است. معمولا اطلاعات زیر در بسته token قرار داده می‌شود.

```
TypeDef Struct Token
```

```
        {int ROW;                       //      شماره سطر
        int COL;                        //      شماره ستون
        int BLKORD;             //      شماره آشیانه
        enum SymbolType;        //      شماره ترتیب بلاک در برگیرنده
        char Name[39];                  //      نوع لغت
} TokenType;                    //      خود لغت
```

در ساختار ارائه شده برای تعریف لغت که در اینجا بسته لغت یا token نـامیده شـده اسـت شـماره سـطر و ستون ظاهر شده است. با استفاده از شماره سطر و ستون لغت، در هنگام صدور پیغام خطـا، کامپـایلر قـادر بـه بیان دقیق مکان خطاء خواهد بود. BLKNO شماره آشیانه ای بـلاک در بـرگیرنـده یـک لغـت را مشـخص می‌کند. تنها نام لغت برای تعیین و تشخیص آنها از یکدیگر کافی نیست بلکه، جهت تعیین یک لغت نیـاز بـه شماره آشیانه‌ای بلاک در برگیرنده آن نیز هست. برای نمونه به قطعه کد زیر توجه کنید:

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

در قطعه کد فوق همانگونه که مشاهده می کنید اگر چه که برای دو متغیر یک نام I تعییـن شـده امـا شـماره آشیانه‌ای بلاک در بر گیرنده، وجه تمایز این دو می‌باشد. گاهی اوقات شماره آشیانه‌ای بلاک نیز کافی نیسـت برای نمونه در قطعه کد زیر اگر چه که شماره آشیانه‌ای بلاک ها یکی است اما دو متغیر I وجود دارد.

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

شماره آشیانه ای با ورود بلاک افزایش یافته و با خروج از بلاک کاهش می یابد. با ظـهور هـر بـلاک جدیـد شماره بلاک یا BLKORD یک واحد اضافه می شود، در مثال فوق وجـه تمـایز دو متغـیر I شـماره ترتیـب بلاک در بر گـیرنده آن است.

به هر لغت یک نوع تخصیص داده می‌شود. نوعی شمارش پذیر یا در اصطلاح Enumerated بـرای نمونـه تحلیلگر لغوی اگر در متن فایل ورودی عدد 123 را تشکیل دهـد، در داخـل فیلـد Name عـدد 123 و در داخل فیلد Type، نوع آن یک عدد صحیح می‌باشد را مشخص می‌کند. اگر در متن ورودی، جملـه زیـر قـرار گرفته باشد:

ABC := 123

تحلیلگر لغوی با اجرای دستورالعملی مثل:

NextChar := getch(in-text);

ابتدا کرکتر A و سپس B و بعد از آن C را از متن فایل ورودی خوانده، حالا با مشـاهده Blank بـا مشـاهده علامت کولون (:)، خاتمه اولین لغت را تشخیص داده، در داخل فیلد Name رشته ABC و در داخـل فیلـد Type، نوع آن که یک شناسه می‌باشد را قرار می‌دهد.
در فراخوانی بعدی تحلیلگر لغوی با Blank مواجه می‌شود. از آن چشم پوشـی می‌کنـد و بـه جلـو مـی‌رود و علامت = : را به عنوان لغت بعدی تشـخیص داده در فیلـد Name، رشـته = : و در فیلـد Type عملکـرد تخصیص را قرار می‌دهد. البته شماره سطر و ستون و بلاک در برگیرندهٔ هر لغت نیز مشخص می‌شود.
نوع لغات در فیلد Type از ساختار TokenType مشخص شده است. این فیلد از نوع شمارش پذیر به نـام Symbols تعریف شده است. در داخل نوع شمارش پذیر یا در اصطلاح Enumerated به نـام Symbol انواع ممکن لغات در داخل زبان مورد نظر مشخص می‌شود. باید توجه داشته باشید که با تعریف متغـیر از نـوع شمارش پذیر مجموعه‌ای از مقادیر ثابت یا Constant تعریف مـی شـود. نـوع Symbols در زبـان C بـه صورت زیر برای نمونه تعریف می‌شود:

```
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_Colon, S_ParBAZ, S_ParBast, S_Set, S_of, S_BrackBaz, S_Var,
S_BrackBast, S_Pointer, S_ConstString, S_Function, S_Procedure, S_Begin,
S_Do, S_Comma, S_If, S_Then, S_Eles, 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 که مشاهده می‌شود انواع مختلف لغات باید در نوع Symbols گنجانده شود.

۳ – ۲ عبارات با قاعده

عبارات باقاعده، فرمی است چکیده و خلاصه برای بیان قوانین لغوی زبان ها. شکل لغـات در زبانـهای برنامـه نویسی دارای فرم کلی خاص است. براساس این فرم کلی است که انواع لغات از قبیل شناســه هـا (اسـامی) و اعداد، رشته های ثابت کرکتری، جملات تفسیری (Comment) و سایر لغات از یکدیگر تفکیک و تمیز داده می شوند.

۱ – ۳ – ۲ نمونه هایی از عبارات با قاعده

به عنوان نمونه تعریف شناسه ها در زبان c را در نظر بگیرید. یک شناسه یا Identifire در زبان c حتما بـاید با یک کرکتر آغاز گردد و در ادامه ممکن است به هر تعداد و با هر ترکیبی از ارقام صفر تا نه (0-9)و الفبـای انگلیسی (A-Z) و علامت خط زیر یا در اصطلاح (UnderLine) ادامه یابد. به عنوان مثال اسامی:

A, B__1, BAC2345__

از لحاظ زبان C به عنوان اسامی (شناسه)، شناخته می‌شوند. می‌توان با استفاده از یک عبارت بـا قـاعده بـه صورت زیر، فرم کلی شناسه ها را در زبان C تعریف نمود:

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

در عبارت ارائه شده برای شناسه ها، علامت '*' نمایانگر تکرار صفر یا بیشتر می باشد و علامت 'I' علامت یـا می باشد. می‌توان به صورت چکیده‌تر نیز Digit را تعریف نمود. به همین ترتیب، حروف الفبای انگلیسی را به صورت زیر می‌توان تعریف نمود:

Digit: [0..9]

به همین ترتیب، حروف الفبای انگلیسی را به صورت زیر می‌توان تعریف نمود:

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

شکل کلی اعداد صحیح را می‌توان با استفاده از یک عبارت به این صورت مشخص نمود:

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

بر طبق این عبارت، یک عدد صحیح با یک رقم بین صفر تا نه آغاز می‌شود و به هر تعدادی رقم می‌توانـد در ادامه آن ظاهر شود، برای مثال:

0, 5, 0123

همانگونه که مشخص است، یک عدد حداقل دارای حداقل یک رقم باید باشد یا به عبارت دیگر یـک عـدد از یک یا بیشتر ارقام تشکیل شده، می‌توان عدد را به صورت زیر تعریف کرد:

Number: digit^{+}

در اینجا علامت '+' نمایانگر تکرار یک یا بیشتر می‌باشد، یادآوری می‌کنیم کـه علامـت سـتاره '*' نمایـانگر تکرار صفر یا بیشتر است. اعداد می‌توانند دارای علامت و یا بدون علامت باشند.

Number: (+ | - |λ) digit $^{+}$

در این عبارت با قاعده، علامت لامبدا λ نمایانگر عدم وجود و یا تهی می‌باشد. عبارت فـوق بـه ایـن صـورت خوانده می‌شود: یک عدد دارای علامت به علاوه یا منها و یا ممکن است اصـلا علامتـی نداشـته باشـد و بـه دنبال آن به تعداد یک یا بیشتر ارقام بین صفر تا نه ظاهر گردد. اعداد را می‌توان به فرمی ساده‌تر با اسـتفاده از این قاعده که هر عبارت اختیاری مثل r بصورت r* نمایش داده می‌شود. به صورت زیر مشخص نمود:

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

به عنوان نمونه‌ای دیگر از عبارات با قاعده، فرم کلی رشته ها در زبان پاسکال را در نظـر بگیریـد. بـاید توجـه داشته باشید که در این زبان چنانچه در وسط رشته علامت کوتیشن ' نیاز باشد، می‌بایست آن را دو بار تکرار نمود. برای نمونه چنانچه جملهٔ This is your 's در خروجی مورد نظر باشد، می‌توان دستور writeln('This is your''s'); استفاده نمود. لذا، فرم کلی رشته ها با یک عبـارت بـا قـاعده بـه صـورت زیـر بیـان می‌شود:

String: '(Characters - ' ') * '

در عبارت فوق، مجموعه Characters نمایانگر هر گونه کرکتر قابل مشاهده منهای کوتیشن است.

۲-۳-۲ قوانین ایجاد عبارت باقاعده

برای ایجاد یک عبارت باقاعده، تعدادی از علائم مورد استفاده واقع می‌شوند. هر یک از این علائم دارای معنی و مفهوم خاصی می‌باشند. در حالت کلی اگر دلتا (Δ) مجموعه علائم بکار رفته شده در عبارات باشد، آنگاه:

۱. هر عنصر a € ? خود یک عبارت با قاعده است. برای مثال چنانچه [9 .. 0] = ? باشد آنگاه هر رقمی بین صفر تا نه مثل ۱، خود به تنهایی یک عبارت باقاعده است.
۲. چنانچه r و s دو عبارت با قاعده باشند، آنگاه rs نیز یک عبارت با قاعده است. به عبارت ساده‌تر در حالت کلی اگر دو عبارت با قاعده را در کنار یکدیگر قرار دهید، حاصل یک عبارت باقاعده خواهد بود.
۳. چنانچه r و s دو عبارت با قاعده باشند، آنگاه r یا s که به صورت (r- s) مشخص می‌شود نیز یک عبارت باقاعده است. عملگر λ دارای خاصیت جابجایی است. به عبارت دیگر:

r-s = s - r = (r-s)

۴. چنانچه r یک عبارت با قاعده باشد آنگاه r* نمایانگر تکرار صفر یا بیشتر از عبارت r است. برای نمونه اگر a، r یا b باشد آنگاه r* برابر است با:

r: a - b: (a-b)*

این عبارت گویای هر ترکیبی با هر تعدادی از a یا b که در کنار یکدیگر قرار گرفته‌اند می‌باشد. برای نمونه رشته های BBBB , BAABB , AAA همگی فرم های خاصی از عبارت r* هستند. بنابراین علامت لامبدا λ که نمایانگر عنصر تهی است، به تنهایی یک عبارت باقاعده است.

۵. چنانچه r یک عبارت باقاعده باشد، آنگاه r^+ نمایانگر تکرار یک یا بیشتر عبارت r است.

برای نمونه :

Number: $digit^+$

به این ترتیب واضح است که:

$r^* = (r^+ - \lambda)$

با استفاده از پنج قاعدهٔ فوق، می‌توان عبارت با قاعده را بنا نمود. با استفاده از نکات فوق، فرم کلی جملات تفسیری Comment در زبان پاسکال را می‌توان به صورت زیر معین نمود. باید توجه داشته باشید که در زبان پاسکال جملات تفسیری در بین علائم آکولاد باز و آکولاد بسته قرار می‌گیرند.

Comment: { C* } ,C: (کلیه کرکترها منهای آکولاد)

در زبان پاسکال جملات تفسیری را میتوان بین علائم (* *) نیز مشخص نمود. در این فرم از جملات تفسیری باید توجه داشته باشید که اگر ستاره در داخل جمله ظاهر شود، در پشت آن باید هر حرفی به غیر از ستاره و پرانتز بسته ظاهر شود. برای نمونه، لغت زیر یک جملهٔ تفسیری نیست:

(* abc *) abc *)

فرم کلی اینگونه جملات را می‌توان در قالب یک عبارت باقاعده به صورت زیر خلاصه نمود:

Comment2: ‘(‘* ((r |*+ s) |*+ ‘)’

کلیهٔ کرکتر ها منهای علامت :r

کلیهٔ کرکترها منهای علائم :s

به این ترتیب جملات تفسیری به صورت Comment: Comment | Comment2

تعریف می‌شوند.

با توجه به اینکه اعداد یا به صورت صحیح و یا به صورت اعشاری ظاهر می‌شوند و با در نظر گرفتن اینکه اگر قبل از ممیز حداقل یک رقم ظاهر شود آنگاه وجود ارقام پس از ممیز ضروری نیست و نیز اینکه بعد از ممیز حداقل یک رقم ظاهر شود وجود رقم قبل از ممیز ضروری نیست. به طور خلاصه عبارت با قاعده برای بیان شکل کلی اعداد به صورت زیر می‌تواند باشد:

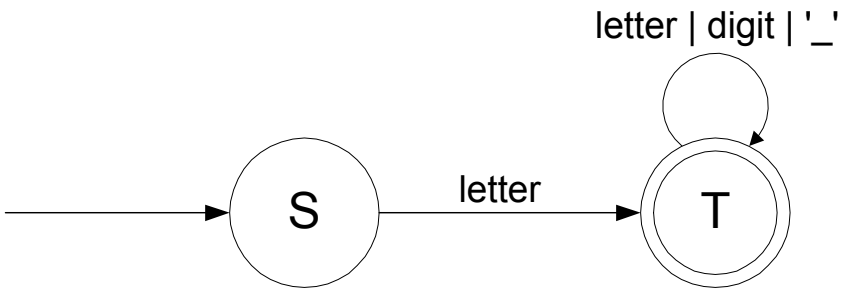
Number: (digit* . digit $^{+}$) | (digit^{+} . digit*) | digit

عبارات با قاعده مبین قوانین لغوی و نمایانگر فرم کلی لغات هستند. متأسفانه، این فرم کلی را به سادگی نمی‌توان تبدیل به کد برنامه نمود. لذا برای رفع این مشکل از ماشینهای خودکار استفاده می‌شود.

۲-۵ ماشین های خودکار

یک ماشین خودکار نوعی گراف می‌باشد که دارای تعدادی رئوس یا حالات (States) و تعدادی لبه یا یال (Edges) می‌باشد و در این گراف یال‌ها خطوط واصل بین حالات هستند. این ماشین‌ها، ابزاری برای تعیین قوانین لغوی و تشخیص لغات می‌باشند که به سادگی قابل تبدیل به کد برنامه بوده، به طوری که با استفاده از آنها می‌توان تحلیلگر لغوی را به صورت یک برنامه تبدیل کرد و یا Source یک برنامه را تولید نمود. برای

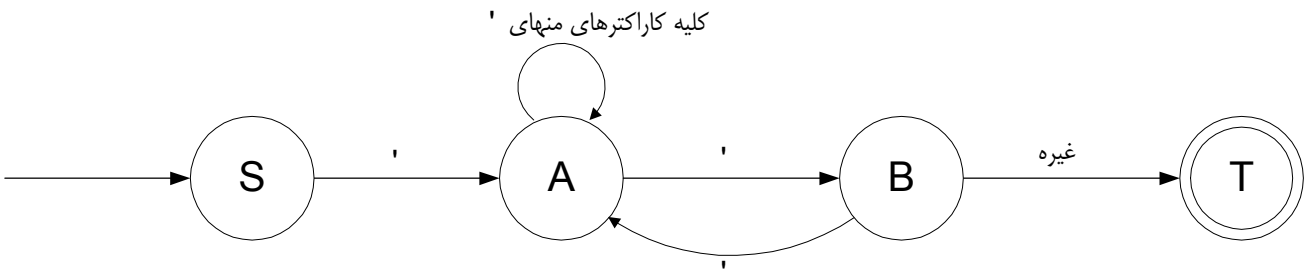
نمونه شناسه‌ها را در نظر بگیرید. شناسه‌ها توسط عبارت باقاعده letter (letter | digit)* در بخش قبل معین شدند. برای تشخیص شناسه‌ها می‌توان ماشین خودکار زیر را مورد استفاده قرار دارد:



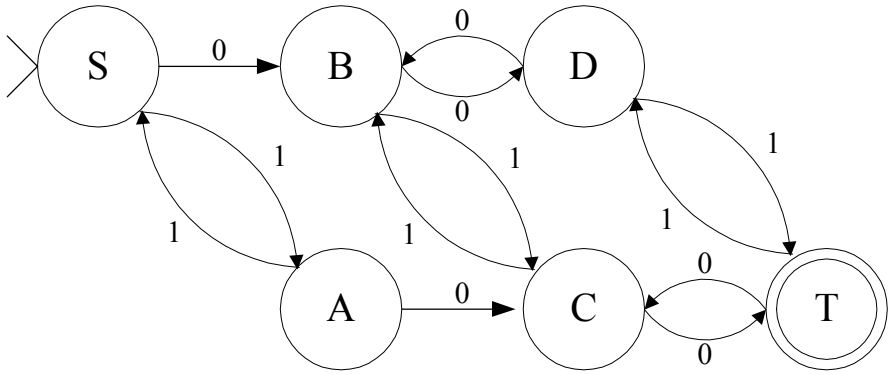
این ماشین خودکار را به صورت زیر نیز می‌توان نشان داد. بنابر دیاگرام فوق در صورتی که کاراکتر خوانده شده از متن فایل مورد تحلیل لغوی یکی از حروف الفبای لاتین یا در اصطلاح letter باشد، می‌توان وارد ماشین خودکار تشخیص شناسه‌ها شد. حالت شروع و در واقع نقطه ورود به ماشین خودکار با حرف S مشخص شده است. حالت بعدی T نامیده شده است. در این حالت اگر کاراکتر خوانده شده از فایل ورودی، از نوع Letter یا Digit و یا کاراکتر '-' باشد به حالت پذیرش یعنی T می‌رسد.
ماشین خودکار به خواندن کاراکترها بعدی ادامه می‌دهد. اما ورودی * اهمیت دیگری دارد. با دیدن * اگر در پشت آن علامت / ظاهر شود، کار خاتمه یافته تلقی می‌شود. حالت T حالت پذیرش است. در حالت C در واقع ماشین به خاطر دارد که * دیده شده است. اگر در حالت C در ورودی / ظاهر شود، کار خاتمه یافته است. در غیر این صورت به حالت B تغییر وضعیت داده می‌شود. پی مشاهده می‌کنید که ماشین‌های خودکار بسیار ساده‌تر از عبارات باقاعده قابل تولید هستند.
فرم کلی رشته‌ها را در زبان پاسکال با عبارت باقاعده زیر مشخص می‌نمایند:

String: '(c | '')*' کلیه کاراکترها منهای کوتیشن :c,

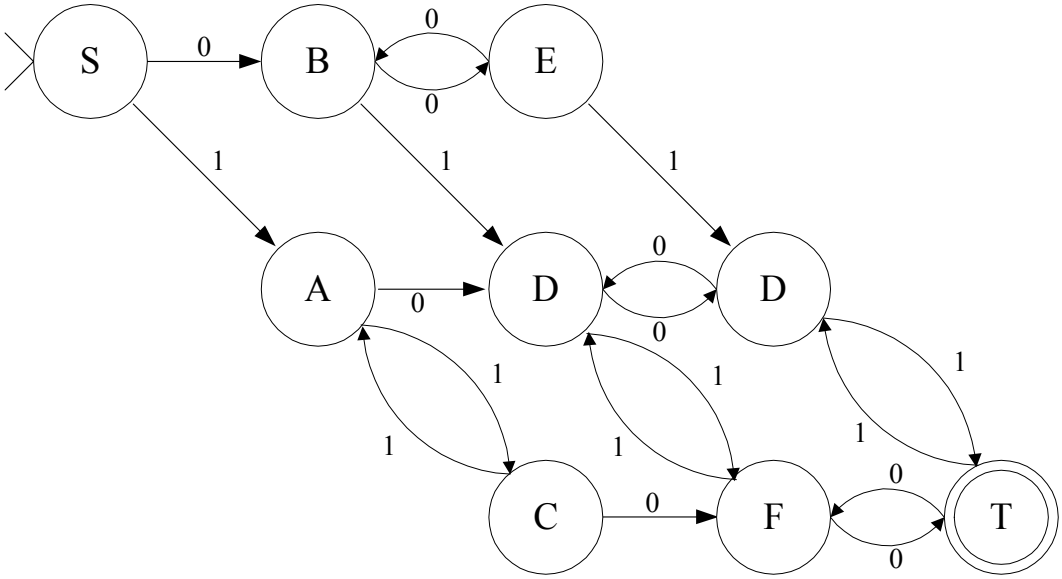
می‌توان رشته‌ها را با ماشین خودکار نیز معین نمود:



مثال: یک ماشین خودکار قطعی برای کلیه اعداد مبنای دو که تعداد صفرهای آن زوج و تعــداد یک‌هـای آن فرد است، ایجاد کنید. در ضمن لااقل دو صفر و یک عدد یک در این اعداد موجود باشند.

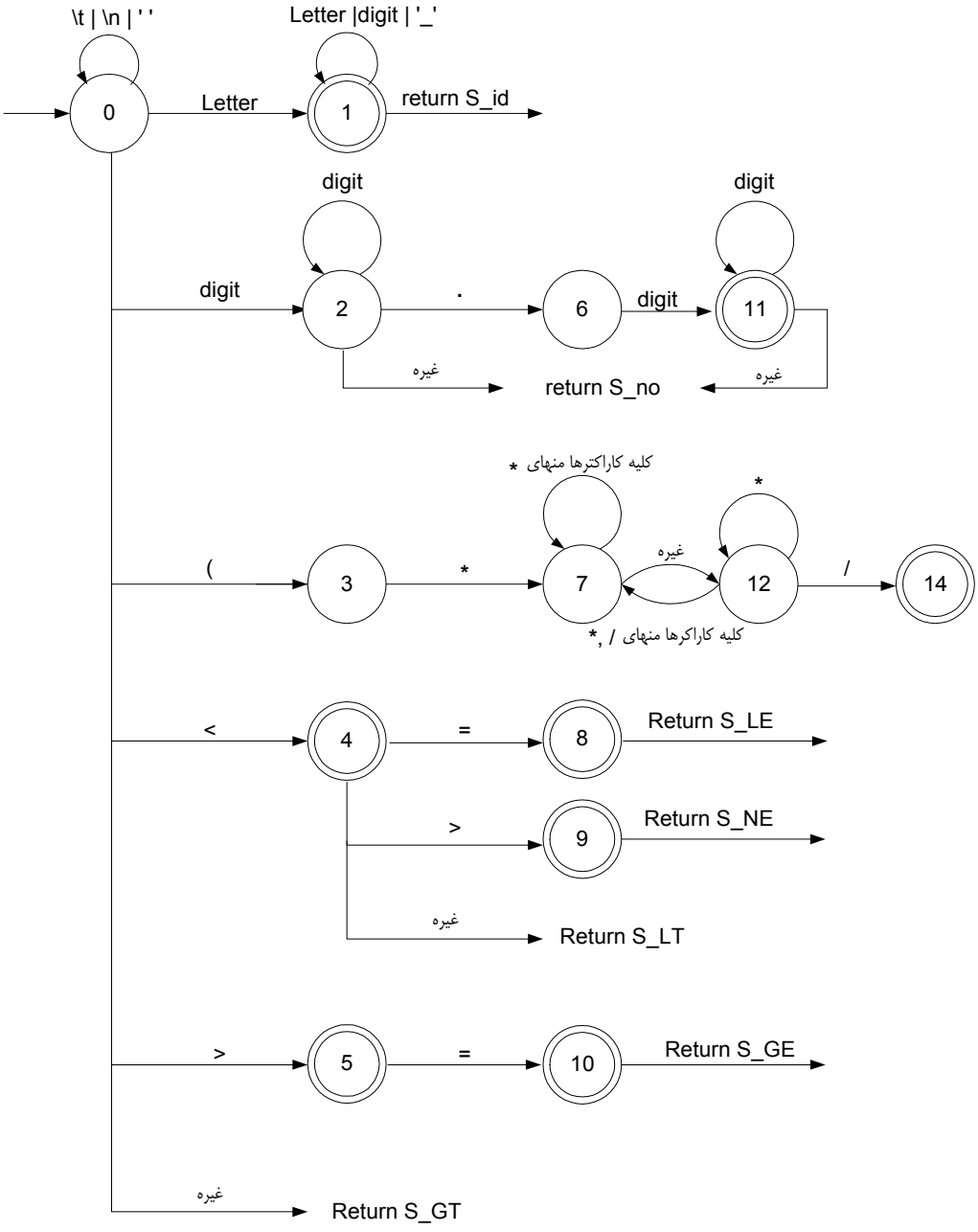


مثال: یک ماشین خودکار قطعی برای کلیه اعداد مبنای دو که تعداد صفرهای آن زوج و تعــداد یک‌هـای آن زوج است، ایجاد کنید. در ضمن لااقل دو صفر و دو عدد یک در این اعداد موجود باشند.



٦-٢ ایجاد تابع تحلیلگر لغوی

همانگونه که در بخش ۵-۲ ذکر شد، می‌توان با استفاده از ماشین‌های خودکار قوانین لغوی را بیان نمود. در این بخش نشان داده خواهد شد که چگونه می‌توان ماشین خودکار را به سادگی تبدیل به کد برنامه نمود. برای این منظور یک دیاگرام کلی برای نمایش برخی از لغات ارائه می‌شود. دیاگرام ارائه شده در شکل زیر نمایانگر ماشین خودکار کلی برای تشخیص تعدادی از لغات است. در واقع این دیاگرام بیان کننده الگوریتم تابع تحلیلگر لغوی است.



هرگاه تابع تحلیلگر لغوی مورد فراخوانی قرار گردد، در حالت شروع صفر قرار می‌گیرد. اگر در فراخوانی قبل، کاراکتری اضافه خوانده شده بود، آن کاراکتر اضافه مورد استفاده قرار می‌دهد، وگرنه کاراکتر بعدی را از داخل متن برنامه مورد کامپایل می‌خواند. تا زمانی که کاراکترهایی که ارزش لغوی ندارند، مانند NewLine، Tab و Blank خوانده شوند، تحلیلگر لغوی در همان حالت شروع صفر باقی می‌ماند. در غیر اینصورت وابسته به نوع کاراکتر، در یک جمله Case اقدام به تشخیص لغات مختلف می‌نماید. می‌توان با استفاده از دیاگرام شکل بالا، کد تحلیلگر لغوی را به شرح زیر در زبان C ایجاد نمود.

```
Struct TokenType lexer (FILE *InText)
{
        enum Symbols LexiconType;
        char NextChar, NextWord[80];
        int State, Length;
        static char LastChar = '\0';
        static int RowNi =0, ColNo= 0;
        State= 0;                              در وضعیت شروع قرار می‌گیرد//
        Length= 0;
        While (!Eof(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== '' | NexrChar== '\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;                          پایان حالت صفر//
        case 1:                                 تشخیص شناسه‌ها//
        NextWord[Length-2] ='\0';
        Return MakeToken(IsKeyWord(NextWord));
        Break;
        Case 3:                                 تشخیص اعداد//
        .
        .
        .
        }                                       پایان سویچ//
}                                               پایان حلقه//
```

در مثال فوق تابع MakeToken کار ایجاد بسته لغات یا در اصطلاح Token را برعهده دارد. ایـن تـابع در زبان C به شرح زیر است:

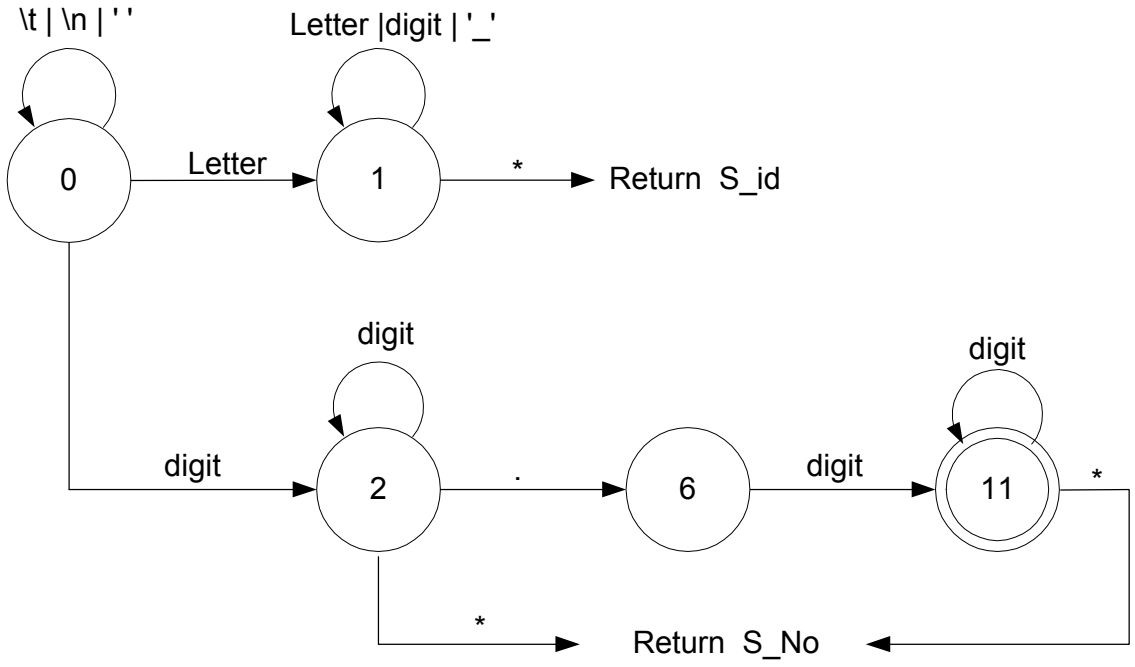
```
enum Symbols IsKeyWord(char *key)
{
        in I;
        struct KeyType
        {char *key; enum SymbolsType}
        KeyTab[]= {'if', S_If, 'while', S_While, 'then',
        S_Then, 'else', S_Else, 'integer', S_Integer,
        'type', S_Type, 'function', S_Function, ..}
        for (I= 0; KeyTab[I].key &&
        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

می‌توان به سادگی تابع مولد تحلیلگر لغوی ایجاد نمود. کافیست، تابعی برای پیمایش یک گراف بنویسید. این تابع در ورودی خود علاوه بر متن برنامه مورد تحلیل لغوی، فایل قوانین لغوی را در فرم یک ماشین خودکار می پذیرد.

از مولدهای شناخته شده تحلیلگر لغوی، یکی LEX می‌باشد. این مولد امروزه بخصوص بر روی سیستمهای عامل UNIX موجود می‌باشد. کار با LEX مشکل است چرا که نیاز به بیان قوانین لغوی، در فرم عبارات باقاعده دارد. در صورتی که همانگونه که تا کنون مشاهده کرده‌اید، به فرمی خیلی ساده‌تر می‌توان با استفاده از ماشینهای خودکار، قوانین لغوی را بیان نمود.

باید قوانین لغوی را در قالب یک ماتریس مشخص نموده و در داخل یک فایل متن قرار داد. سپس در داخل برنامه می‌توان به این فایل ارجاع و ماشین خودکار را در قالب یک ماتریس مشخص نمود. برای نمونه در زیر قوانین لغوی در قالب ماشین خودکار و ماتریسی مشخص شده است.



0, \t, 0 1, letter, 1 2, digit, 2

0, \n, 0	1, digit, 1	2, 6
0, ‘‘, 0	1, -, 1	2, Else, AccState_ID
0, letter, 1	1, Else, AcceptState_ID	6, digit, 11
0, digit, 2		11, digit, 11
		11, Else, AccState_NO

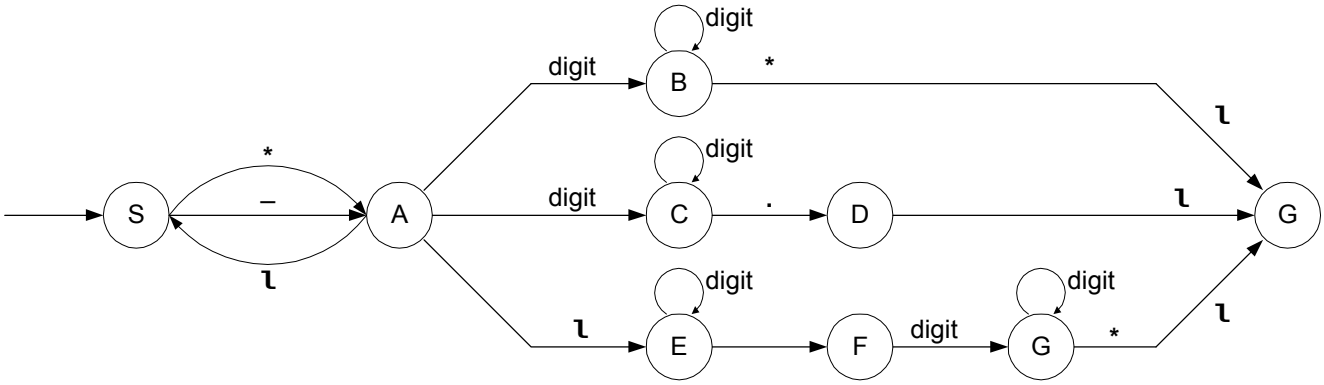
برای ایجاد یک ماتریس نیاز به یک زبان ساده است که بر اساس آن برای نمونه بتوان مشخص نمود که برای مثال Letter چیست و یا غیره، چه می‌باشد. شاید بهتر بود که در فایل ورودی در داخل ماتریس قوانین لغوی Letter به صورت زیر مشخص می‌شد:

0, [A..Z, a..z], 1

حالا در داخل برنامه، این ساختار را به عنوان یک فایل text، کرکتر به کرکتر خوانده، عینا در داخل یک ماتریس با تعداد سطرها که مساوی با تعداد رکوردهای فایل است و تعداد ستون‌ها که مساوی با طول هر رکورد یا هر سطر است، ضبط نمود.
اکنون با نوشتن یک برنامه پیمایش کننده ماشین خودکار و یا تولید متن یک برنامه C که بر اساس این ماشین خودکار برای انجام عمل تحلیل لغوی ایجاد شده، می‌توان لغات را تشخیص داد. همزمان با خواندن هر کرکتر از ورودی، مثلا با خواندن کرکتر کوچکتر از ورودی طبق ماتریس از حالت شروع صفر به حالت ۵ تغییر وضعیت داده می‌شود. در اینجا سرعت عملیات تحلیلگر لغوی، وابسته به سرعت جستجو در داخل ماتریس است.

۸-۲ مسا له عدم قطعیت

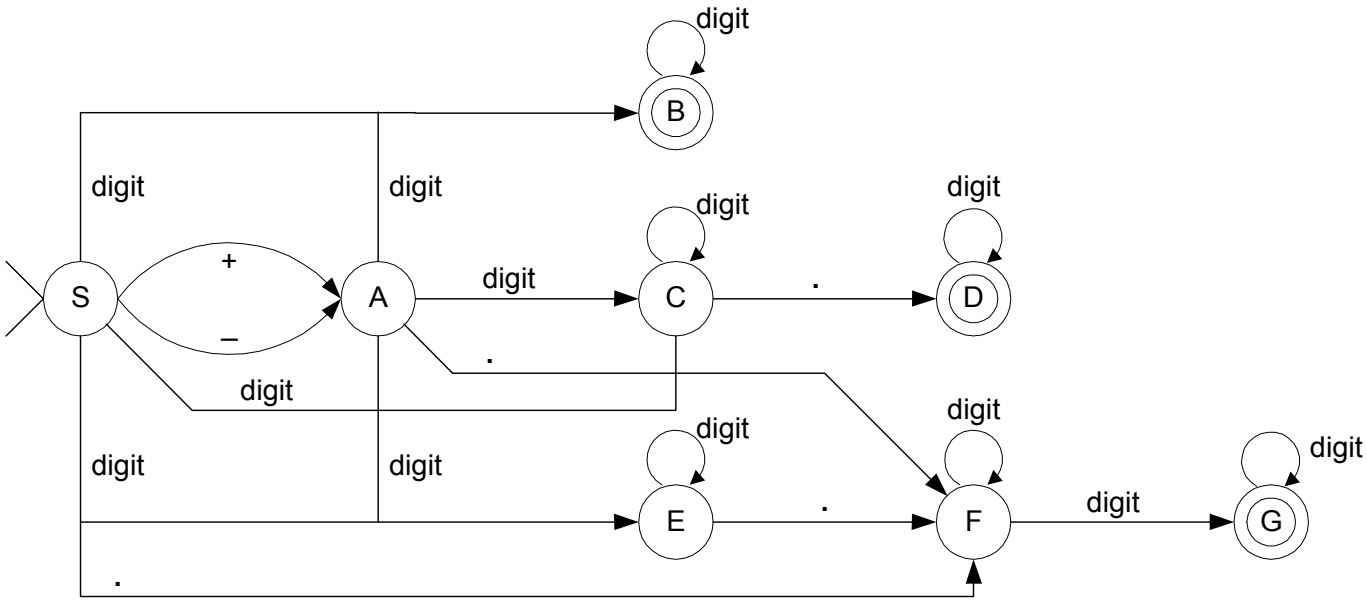
در اینجا ترسیم سه ماشین خودکار برای اعداد مستقل از یکدیگر ساده است. مشکل، ترکیب این سه حالت و ایجاد یک ماشین خودکار غیر قطعی می‌توان به سادگی و بدون در نظر گرفتن مشکل ترکیب گراف برای سه نوع متفاوت اعداد اقدام به ترسیم یک ماشین خودکار برای تشخیص سه نوع عدد نمود. زیبایی کار در اینجاست که می‌توان به صورت الگوریتمی و بدون نیاز به هیچگونه کار اضافی، این سه حالت را در قالب یک ماشین خودکار غیر قطعی با یکدیگر ادغام نمود و با استفاده از الگوریتم‌هایی که ارائه خواهد شد به طور اتوماتیک تبدیل به یک ماشین خودکار قطعی نمود.



همانگونه که در شکل بالا مشاهده می‌شود. یک ماشین خودکار غیر قطعی می‌تواند بیانگر حالات مختلف برای یک لغت باشد. در اینجا منظور از لغت، اعداد می‌باشند. اعداد یا دارای علامت هستند و یا علامت ندارند. این نداشتن علامت را با گذر تهی که با علامت l (لامبدا) مشخص شده، معین می‌کنند. همانگونه که مشاهده می‌شود، در حالت A با دیدن digit نمی‌توان مشخص نمود که حالت بعدی آیا حالت B یا C و یا حالت E است. باید توجه داشته باشید که گذر l در واقع یعنی هیچ چیز.
برای تبدیل ماشین خودکار غیر قطعی به قطعی در سه مرحله عمل می‌شود:

۱- حذف گذرهای تهی
۲- رفع عدم قطعیت
۳- بهینه سازی ماشین خودکار

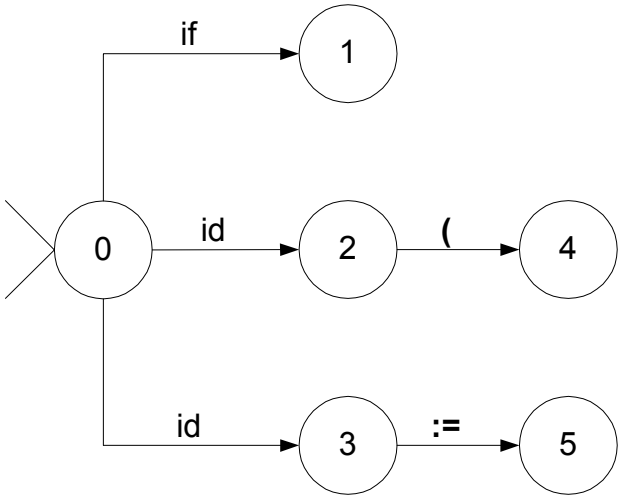
همانگونه که مشاهده می‌شود، کلیه واکنشهای حالت یک و یا به عبارت دیگر کلیه گذرهای خارج شونده از حالت یک، برای حالت صفر در نظر گرفته شد. به این ترتیب گذر تهی حذف گردید. در مرحله بعد به همین ترتیب ادامه می‌دهیم تا کلیه گذرهای تهی را از داخل ماشین خودکار قطع نماییم.



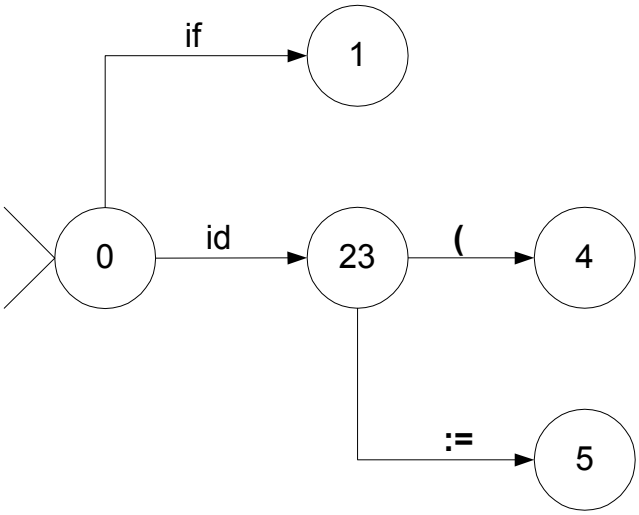
۱-۸-۲ رفع عدم قطعیت

همانگونه که در شکل بالا مشاهده می‌کنید، ماشین خودکار غیر قطعی است. برای نمونه در حالت A با دیدن digit نمی‌توان تشخیص داد که آیا حالت بعدی C ، B و یا اینکه E می‌باشد. اما نکته جالب توجه اینجاست که با دیدن digit حتماً به یکی از سه حالت گذر می‌شود. برای روشن‌تر شدن روش حذف عدم قطعیت بهتر است که کار تحلیلگر نحوی مطرح شود.

تحلیلگر نحوی، معمولاً با دیدن یک لغت در ورودی، ساختار جمله مورد نظر را می‌تواند پیش‌بینی کند. برای مثال اگر لغت دریافتی از تحلیلگر لغوی if باشد، تحلیلگر نحوی بلافاصله تصمیم می‌گیرد که جمله باید جمله if باشد و باید انتظار دیدن شرط جمله if را در ورودی داشته باشد. اما، بادیدن یکid یا شناسه، مشکل وجود دارد. در اینجا تحلیلگر نحوی نمی‌تواند مشخص کند که آیا جمله مورد نظر یک جمله فراخوانی زیر برنامه‌ها و یا یک جمله تخصیصی (Assignment) است. ماشین خودکار غیرقطعی در اینجا به صورت زیر قابل ترسیم است:



علیرغم وجود عدم قطعیت در حالت صفر از شکل بالا می‌توان قطعیتی را در نظر داشت. به این ترتیب کــه در حالت شروع صفر اگر یک شناسه یا id در ورودی ظاهر شود، قطعا حالت بعدی، حالت ۲ یا ۳ خواهد بــود. در حالت ۲ یا۳ اگر ’=:‘ در ورودی ظاهر شود جمله تخصیصی، اگر ’)‘ ظــاهر شــود، جملــه فراخوانــی و در غــیر اینصورت جمله غلط می‌باشد. پس حالات ۲ یا ۳ را به صورت یک حالت ترکیبی جدید با ادغــام خروجی‌هــای این دو حالت می‌توان به وجود آورد.



برای سهولت در امر تبدیل و یا نمایش ماشینهای خودکار به جای گراف از فرم جدول مانند اســتفاده می‌شــود. برای نمونه ماشین خودکار غیر قطعی اعداد در سه شکل قبل به صورت یک جــدول در شــکل زیــر مشــخص شده است.

	Digit	.	- / +
S	BCE	F	A
A	BCE	F	
B	B		
C	C	D	
D	D		
E	E	F	
F	G		
G	G		

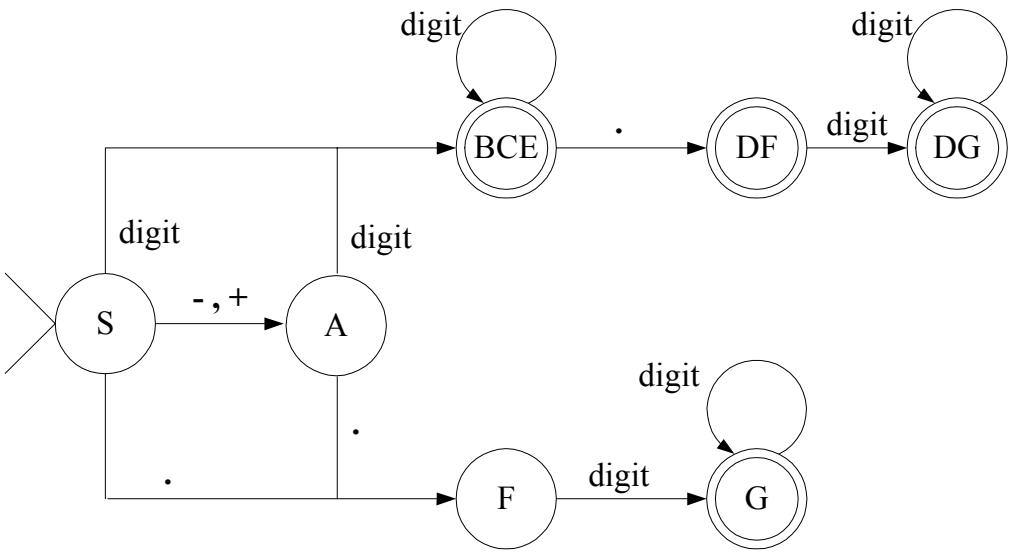
همانگونه که در شکل بالا مشاهده می‌شود در حالات S و A به یکی از سه حالت B یا C یا E به ازای دیدن digit گذری وجود دارد. لذا، می‌توان گفت کــه گذری بــه حــالت ترکیبــی BCE وجــود دارد. در ایــن حــالت

می‌توان واکنشهای هر سه حالت C، B و E را داشت. ردیف مربوط به حالت ترکیبی BCE در شکل زیر ترکیبی از واکنشهای هر سه حالت تشکیل دهنده آن است به عبارت دیگر از ترکیب '**یا**' سطرهای B، C و E، سطر جدید BCE به ماتریس افزوده می‌شود. BCE یک حالت پذیرشی است، زیرا یکی از حالات تشکیل دهنده آن یعنی B یک حالت پذیرش است.

همانگونه که در شکل زیر مشاهده می‌شود. در حالت جدید BCE به ازای ورودی '.' (نقطه اعشار) عدم قطعیت وجود دارد. از حالت BCE می‌توان به هر یک از دو حالت D یا F گذر نمود. لذا برای عدم قطعیت، حالت جدید دیگری به نام DF را به جدول حالات باید افزود. برای رفع عدم قطعیت در حالت DF حالت DG ایجاد می‌شود.

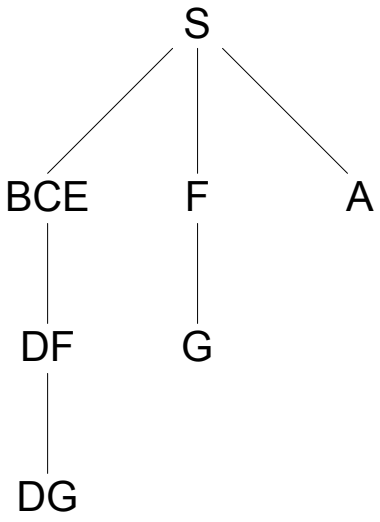
	Digit	.	- / +
S	BCE	F	A
A	BCE	F	
B	B		
C	C	D	
D	D		
E	E	F	
F	G		
G	G	DF	
BCE	BCE		
DF	DG		
DG	DG		

اکنون با استفاده از جدول فوق می‌توان ماشین خودکار قطعی اعداد را به صورت زیر ایجاد نمود.



همانگونه که مشاهده می‌کنید برخی از حالات درون جدول مثل حـالات B, C, D و E در عمـل غیرقـابل دسترسی هستند. علت افزایش حالات جدید ترکیبی است. می‌توان این حالات زاید را با ایجاد درخت دسترسی نیز مشخص نمود.

با افزایش حالات جدید جهت عدم قطعیت در ماشین خودکار، برخی از حالات زاید شده، از حالت شـــروع غـیر قابل دسترسی می‌شوند. جهت تشخیص این حالات یا می‌توان ماشین خودکار را از روی جدول ترسیم نمود. و یا اینکه با استفاده از یک درخت دسترسی حالات قابل دسترسی از حالت شـــروع S را مشـخص نمـود. بـرای نمونه درخت دسترسی برای ماتریس دو شکل قبل به صورت زیر است. باید توجه داشته باشید که نام حـــالات در درخت دسترسی تکراری نیست و این درخت صرفا جهت تعیین حالات قابل دسترسی از حالت شـروع S است.



۲-۸-۲ بهینه‌سازی ماشین‌های خودکار

هدف از بهینه سازی، تقلیل تعداد حالات در ماشین‌های خودکار است. برای این منظور باید حـــالات معـادل را تشخیص داد. دو حالت را در صورتی معادل گویند که به ازای ورودی‌های متفاوت با گذشت از یـــک سـری از حالات یا به عبارت دیگر در مسیری به طول صفر یا بیشتر و با گذشتن از تعداد n گره نهایتا به حالت پذیـرش برسند. حالات معادل را با روشی ابتکاری به ترتیب زیر می‌توان مشخص نمود:

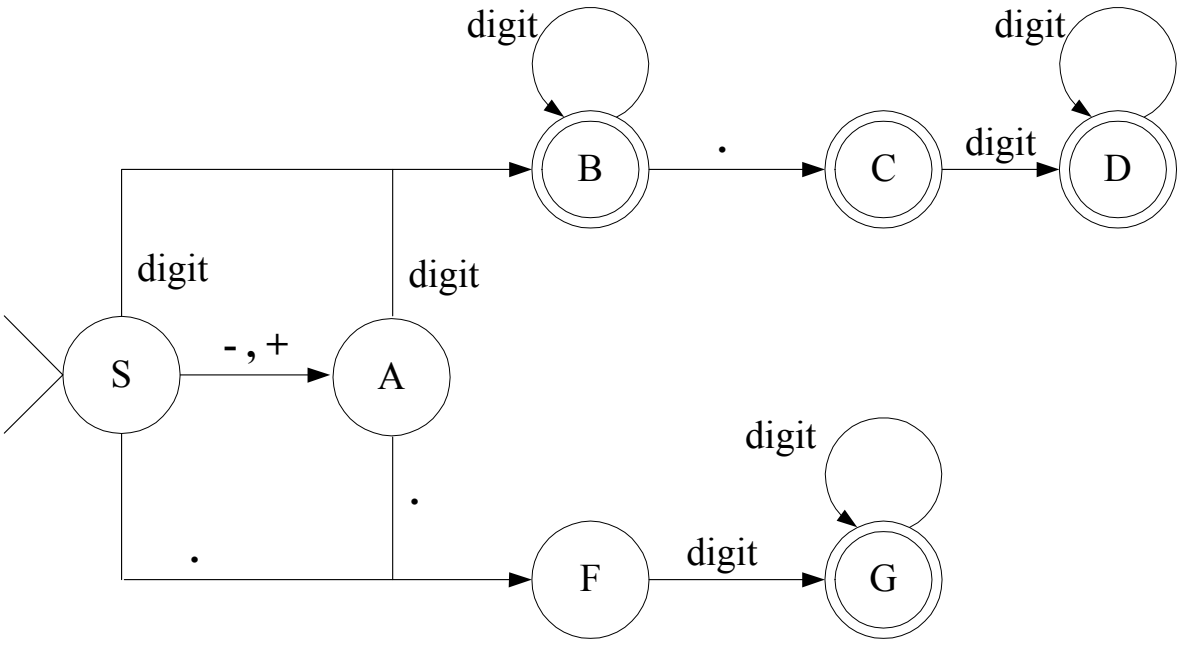
۱. ابتدا حالات معادل صفر که با مسیرهایی به طول صفر به پذیرش می‌رسـند را از سـایر حـالات مجـزا بـاید نمود. به عبارت دیگر، حالات پذیرش از حالات غیر پذیرش مجزا می‌شوند. به این ترتیب حالات به دو دســته مجزا از حالات پذیرش و غیر پذیرش تقسیم می‌شوند.

۲. به درون هر دسته حالات باید نگریست و مشخص نمود که آیا ورودی وجود دارد کـــه بـه ازای آن حـالات متعلق به یک دسته واکنش‌ها جدا از سایرین دارند. واکنش‌ها بر اساس گذر یک حالت به حالت دیگر سـنجیده نمی‌شوند. بلکه، اگر از یک حالت به حالت دیگر گذری وجود دارد، دســته مربـوط بـه آن حـالت را مشـخص

می‌کنند. مقصود مقایسه دسته‌هایی است که به آنها گذر می‌شود. حالات مشابه به ازا یک ورودی بـــه حـالات درون یک دسته گذر می‌کنند.
۳. آنقدر مرحلهٔ ۲ تکرار می‌شود تا دیگر گذری متفاوت بین دسته‌ها وجود نداشته باشد و دسته‌ای جدیدتر تولید نشود. برای نمونه اکنون ماشین خودکار که در مبحث قبل مطرح شد بهینه‌سازی می‌شود.

به طور خلاصه و با در نظر گرفتن الگوریتم فوق می‌توان حالات معادل را به صورت زیر تعریف نمود:
دو حالت را در صورتی معادل k گویند که به ازای هر رشته از ورودی‌ها به طول کوچکتر یا مساوی با k اگر از یکی از آنها بتوان به پذیرش رسید، از دیگری نیز بتوان به ازای همان رشته به حالت پذیرش رسید.
دو حالت را معادل گویند فقط و تنها فقط اگر به ازای هر رشته‌ای از ورودی‌های متوالی که موجـب رسـیدن از آن حالت به یک حالت پذیرش است بتوان از دیگری نیز به پذیرش رسید.

اکنون با توجه به تعریف حالت معادل بهینه‌سازی ماشین‌های خودکار، می‌توان مبادرت به بهینه‌سـازی ماشـین خودکار اعداد که در مبحث قبل مطرح شد، نمود. ماشین خودکار قطعـــی اعـداد و جـدول تولیـد آن در شـکل بخش مربوطه ارائه شده است.



	Digit	.	- / +
S	B	F	A
A	B	F	
B	B	C	
C	D		
D	D		
F	G		
G	G		

برای بهینه‌سازی ماشین‌های خودکار ابتدا باید گره‌ها را به دو دسته پذیرشی و غـیر پذیرشـی افـراز نمـود. بـه عبارت دیگر حالت معادل صفر که با مسیرهایی به طول صفر به پایان می‌رسند را از سایر حالات باید تفکیـک نمود. به این ترتیب حالات ماشین خودکار اعداد که در شکل‌های بالا ارایه شده است به دو دسته زیـر تقسـیم می‌شوند:

۱- حالات پذیرشی {B, C, D.G}

۲- حالات غیرپذیرشی {S, A, F}

حال باید مشخص نمود که آیا در داخل هر دسته به ازای هر ورودی، گره‌ها واکنشی یکسـان دارنـد یـا خـیر. منظور از واکنش یکسان این است که برای نمونه اگر از حالت M در دسته شـماره ۱۲ بـه ازای ورودی Digit گذری به حالتی در دسته شماره ۷ وجود دارد یا خیر. سایر حالات موجـود در دسـته شـماره ۱۲ بـاید بـه ازای ورودی Digit گذری به حالتی در دسته شماره ۷ داشته باشند. در غیر اینصورت، این حـالات از داخـل دسـته خارج شده و در دسته‌ای جدید قرار می‌گیرند.

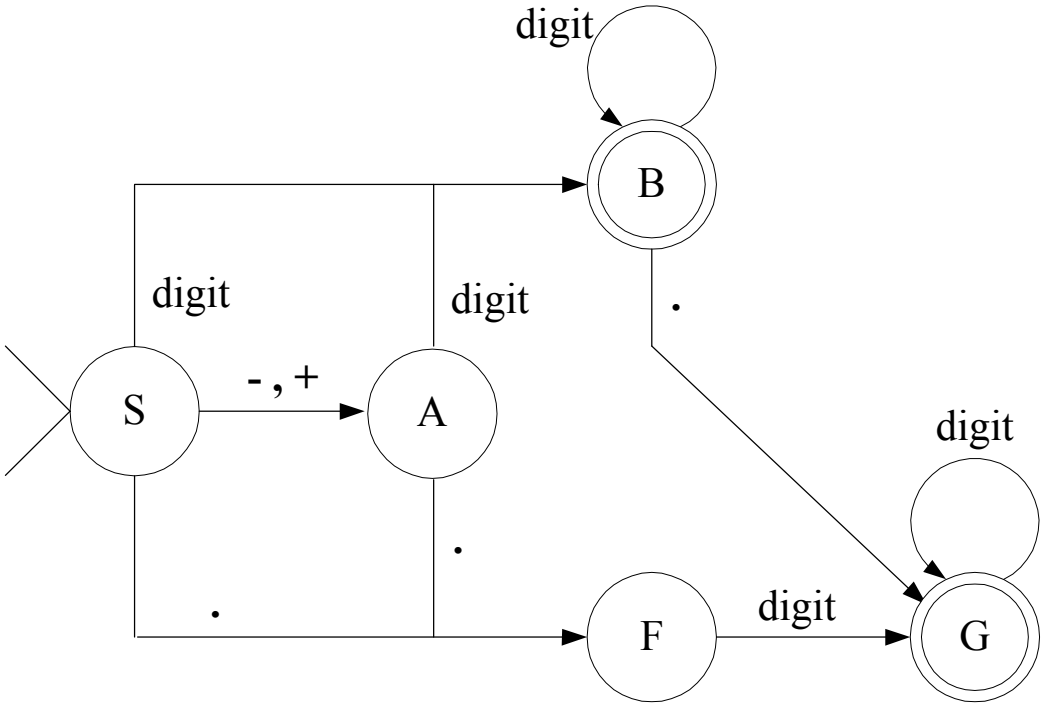
برای نمونه در داخل دسته شماره ۱ در بالا به ازای ورودی نقطه اعشار '.' از حـالت B یـک خروجـی و گذری وجود دارد در صورتی‌که در مورد سایر حالات این چنین نیست. پس این حالت از سایرین جـدا می‌شـود و سـه دسته به صورت زیر حاصل می گردد:

۱ {C, D.G}

۲ {B}

۳ {S, A, F} حالات غیرپذیرشی

به همین ترتیب در دسته شماره ۳ حالت S قابل تفکیک از A و F است. زیرا در S بـه ازای ورودی‌هـای + و – گذر و واکنشی وجود دارد در صورتی‌که در مورد A و F اینچنیـن نیسـت. پـس حـالت S از A و F قـابل تفکیک است. از سوی دیگر A و F نیز قابل تفکیک از یکدیگر هستند. بنابراین سه حـالت C، D و G غـیر قابل تفکیک از یکدیگر و معادل هستند. پس می‌توان هر یک از این سه حالت معـادل را بـه جـای دو حـالت دیگر در داخل ماشین خودکار قرار داد. به این ترتیب ماشین خودکار بهینه اعداد به صورت زیر خواهد بود:

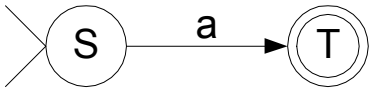


	Digit	.	- / +
S	B	F	A
A	B	F	
B	B	C	
F	G		
G	G		

۲-۱۰ تبدیل عبارات باقاعده به ماشین‌های خودکار

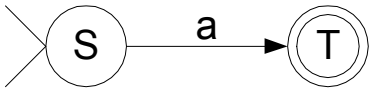
در این قسمت نشان داده خواهد شد که چگونه می‌توان عبارات باقاعده را به صورت ماشین‌های خودکار تبدیل نمود تا اینکه بتوان با استفاده از ماشین خودکار، قوانین لغوی که در فرم عبارات باقاعده خلاصه شده‌اند را عملا به صورت کد برنامه مورد بهره برداری قرار داد. برای تبدیل عبارات باقاعده به ماشین‌های خودکار معمولا مراحل زیر به کار می‌روند:

۱. هر عنصر a متعلق به الفبای زبان به صورت یک ماشین خودکار نمایش داده می‌شود:

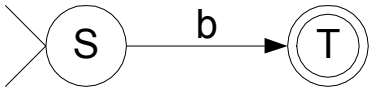


۲. ماشین خودکار برای عبارت ab را از ترکیب گراف‌ها برای a و b به صورت زیر می‌توان ایجاد نمود:

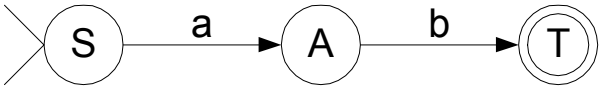
(الف) ماشین خودکار برای a



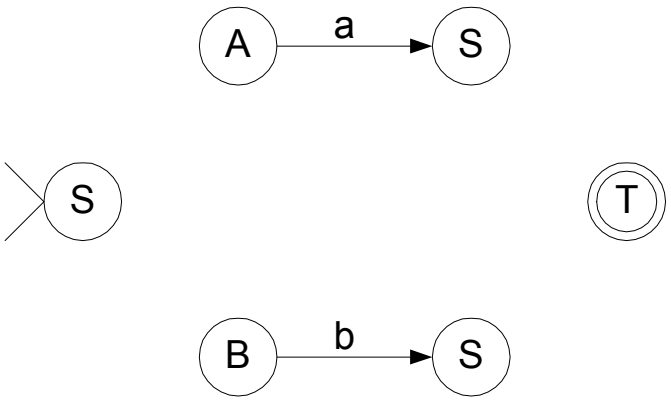
(ب) ماشین خودکار برای b



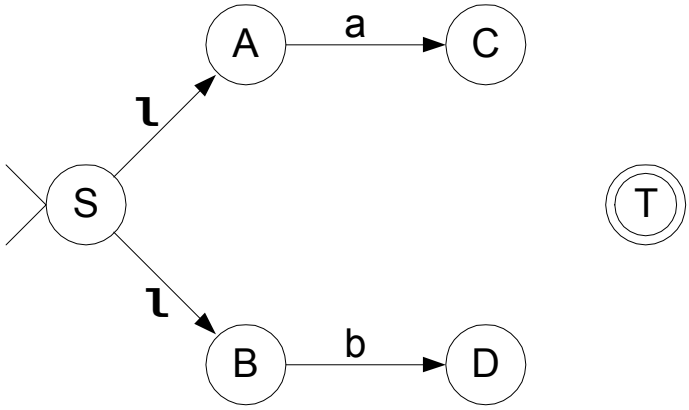
(ج) ماشین خودکار برای ab



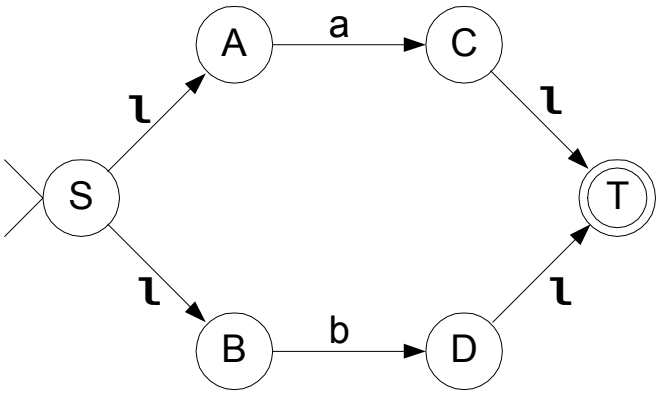
۳. ماشین خودکار برای عبارت a | b را با یک استدلال ساده می‌توان از ماشین‌های خودکار برای عبارات a و b ایجاد نمود. چنانچه حالت شروع ماشین خودکار برای عبارت a | b حالت S باشد:



حالت شروع S هیچ فرقی با حالت شروع برای عبارت a ندارد زیرا در شروع a | b می‌توان a را هم در ورودی دید. از سوی دیگر حالت شروع S هیچ تفاوتی با حالت شروع برای عبارت b ندارد.



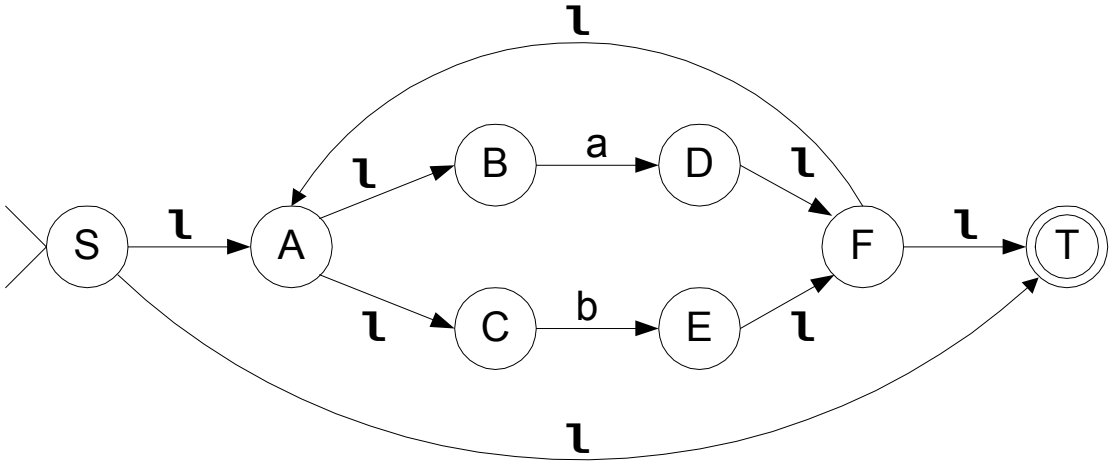
به همین ترتیب با مشاهده a در ورودی باید مطمئن بود که a | b در ورودی ظاهر شده است.
لذا حالت پذیرش برای ماشین خودکار عبارت a یعنی C هیچ تفاوتی با حالت پذیرش برای ماشین خودکار برای عبارت a | b یعنی T ندارد. به همین ترتیب حالت D هیچ تفاوتی با حالت پذیرش T ندارد. بنابراین ماشین خودکار برای عبارت a | b بصورت زیر خواهد بود:



در شکل فوق عبارت a | b از ترکیب ماشین‌های خودکار برای عبارت a و b ساخته شده است. مسلم است که حالت شروع S هیچ تفاوتی با حالت A و حالت B ندارد. اما بالعکس صادق نیست. و حالت شروع برای تشخیص a یعنی A متفاوت از حالت شروع برای a | b است. زیرا در حالت شروع برای a نمی‌توان در ورودی b را دید. لذا، حالت شروع a | b را با دو گذر تهی به حالت شروع برای a و حالت شروع برای b بایدمتصل نمود.

۴. با در دست داشتن ماشین خودکار برای a و b و در نتیجه وجود ماشین خودکار برای a | b می‌توان ماشین خودکار برای عبارت *(a | b) را با یک استدلال ساده ایجاد نمود. به این ترتیب که فرض کنید حالات شروع و خاتمه برای ماشین خودکار *(a | b) به ترتیب حالات S و T باشند. اولا، چون *(a | b) می‌تواند تهی باشد، پس حالت شروع آن ممکن است هیچ تفاوتی با حالت پذیرش نداشته باشد. زیرا *(a | b) ممکن است اصلا وجود نداشته باشد.

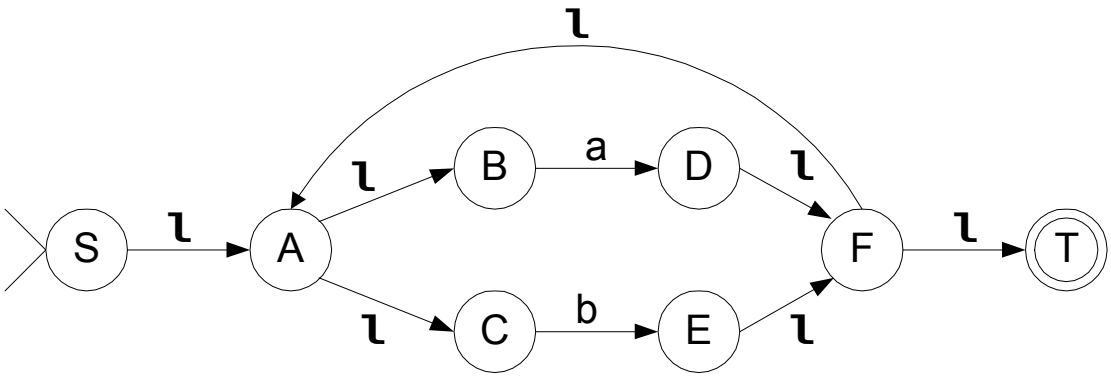
از طرف دیگر ممکن است حالت شروع S هیچ تفاوتی با حالت شروع (a | b) نداشته باشد. زیرا، در شروع *(a | b) می‌توان در شروع مشاهده a | b در ورودی بود. پس از مشاهده a | b دوباره می‌توان در آغاز مشاهده a | b جدیدتری قرار گرفت. این در واقع به خاطر خاصیت تکراری بودن *(a | b) است. پس، حالت پذیرش a | b درون *(a | b) هیچ تفاوتی با حالت شروع آن نخواهد داشت. با این استدلال می‌توان نتیجه گرفت که ماشین خودکار برای عبارت *(a | b) به صورت زیر است.



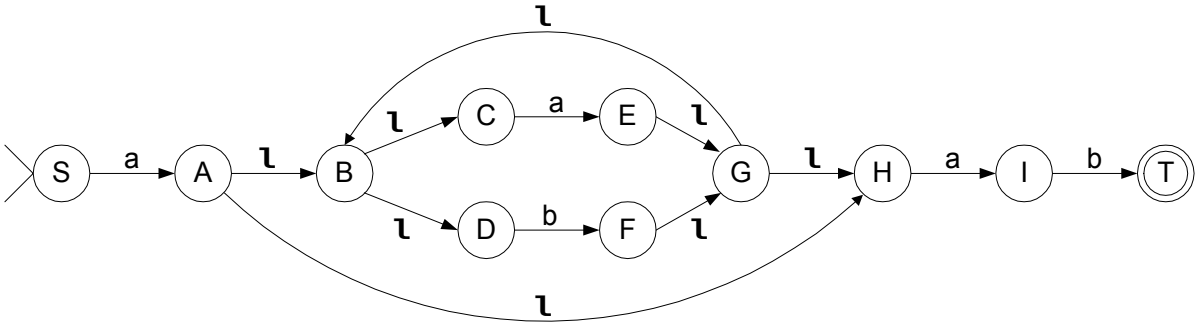
مسلما در ورودی *(a | b) چون رشته a یا b می‌تواند اصلا وجود نداشته باشد، در نتیجه حالت شروع می‌تواند معادل با حالت پذیرش باشد. لذا در شکل بالا، حالت شروع S با گذری تهی به حالت پذیرش T متصل شده است. از جهت دیگر در خاتمه دیدن a | b مثل اینکه دوباره در حالت شروع بوده و می‌توان a | b جدیدی را مشاهده نمود و این عمل تا به هر تعدادی قابل تکرار است. برای این منظور حالت F با گذری

تهی به حالت شروع a | b یعنی A متصل شده است. علاوه بر این حـالت، F ممکـن اسـت خاتمـه تکـرار و حالت پذیرش هم باشد.

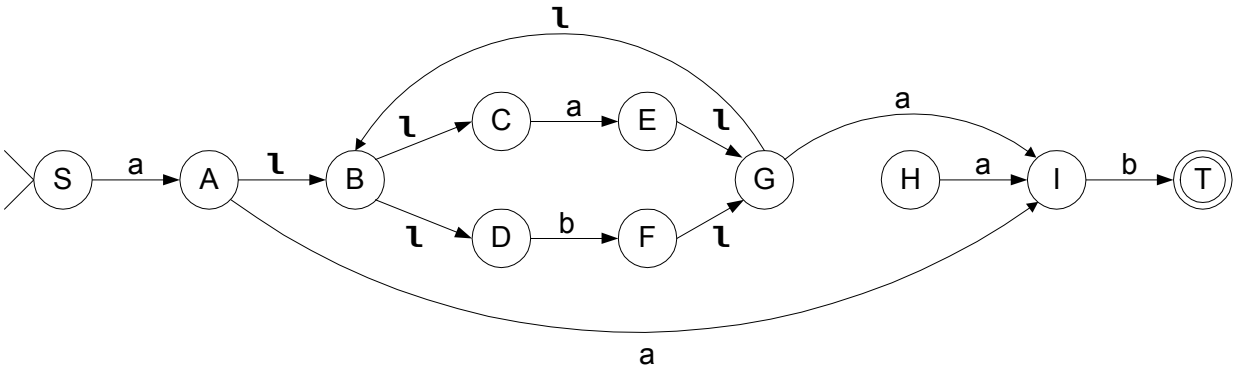
۵. عبارت +(a | b) را می‌توان با حذف گذر تهی از حالت شروع به حالت پایان در ماشین خودکار برای | a) *(b تولید نمود، زیرا در مورد +(a | b) حداقل یکبار a | b در ورودی باید ظاهر شود و حـالت شـروع آن بـا خاتمه یکسان نیست.



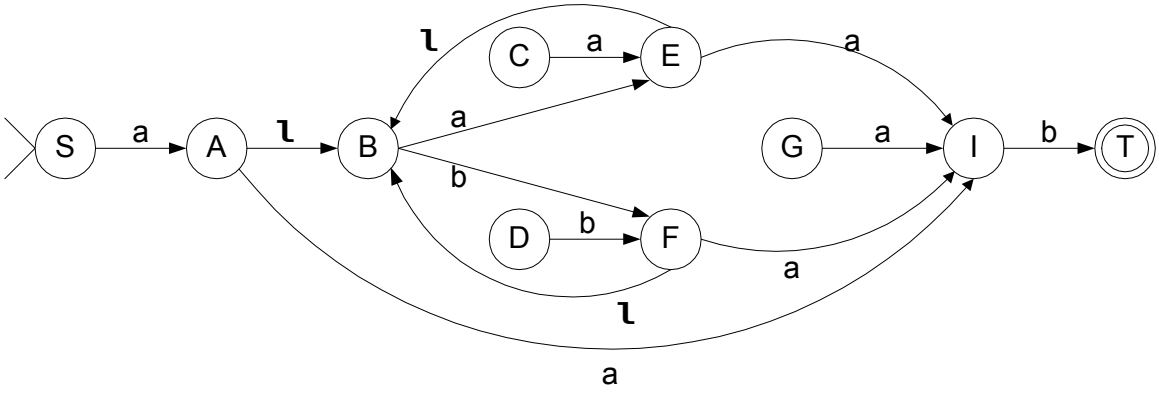
مثال ۱: برای عبارت a (a | b)* ab یک ماشین خودکار بهینه ایجاد نمایید.
با استفاده از ماشین خودکار ارایه شده برای عبارت *(a | b) که در دو شکل قبل ارایه شده است، می‌توان به سادگی ماشین خودکار غیرقطعی را ایجاد نمود.



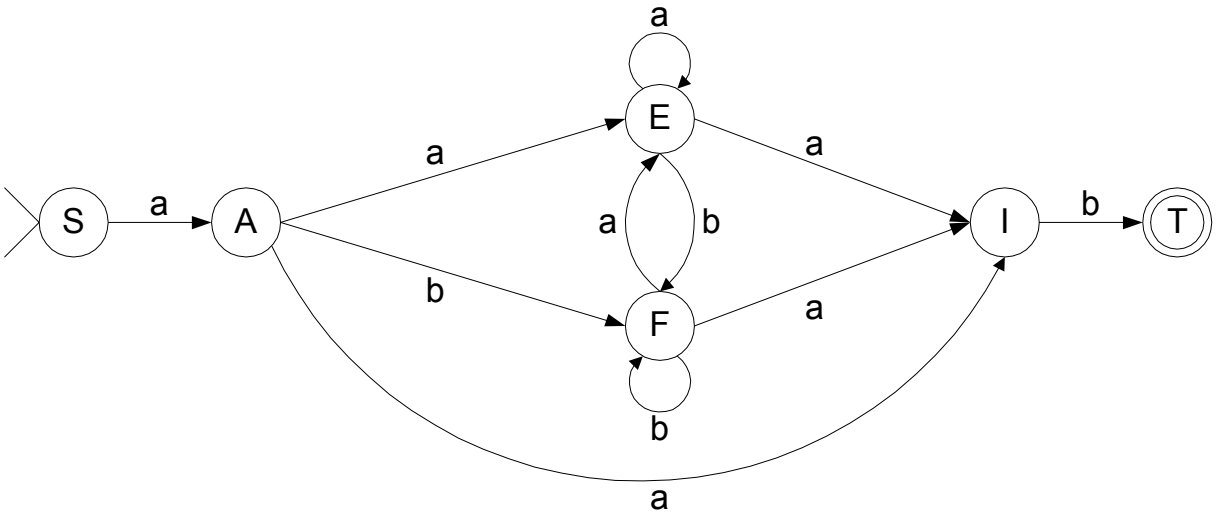
پس از حذف گذرهای تهی به H ماشین به صورت زیر تبدیل می‌شود.



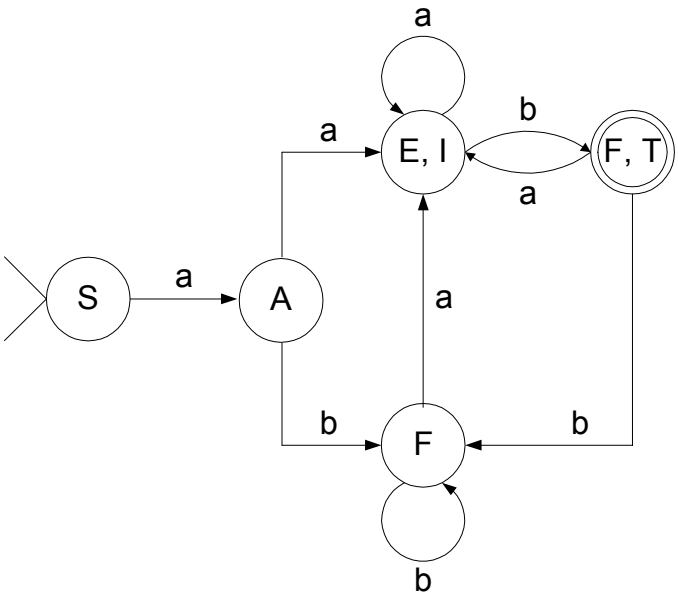
مشاهده می‌کنید که حالت H در شکل بالا غیرقابل دسترسی و قابل حذف است. در شکل زیر گذرهای تهی به حالات G، C و D حذف شده‌اند.



پس از حذف گذرهای تهی ماشین به صورت زیر تبدیل می‌شود.



ماشین خودکار فوق در شکل زیر با استفاده از نمایش ماتریسی به فرم قطعی تبدیل شده است.



	a	b
S	A	-
A	E,I	F
E	E,I	F
F	E,I	F
I	-	T
(T)	-	-
E,I	E,I	F,T
(F,T)	E,I	F

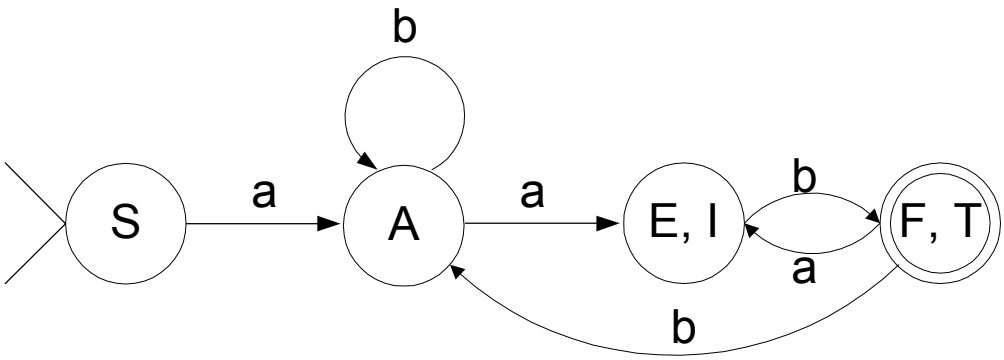
برای بهینه سازی ماشین خودکار فوق حالات مربوطه را به دو دسته پذیرشی و غیر پذیرشی به صورت زیر می‌توان تقسیم بندی نمود:

الف) 1- {FT} 2- {S, A, EI, F}

ب) 2- {FT} 2- {S} 3- {A, EI, F}

ج) 1- {FT} 2- {S} 3- {A, F} 4- {EI}

به این ترتیب مشاهده می‌کنید که حالات A و F با یکدیـگر معـادل هسـتند و یکـی را می‌تـوان بـا دیـگری جایگزین نمود. به این ترتیب ماشین خودکار بهینه برای عبارت به صورت زیر خواهد بود:



۲-۱۰ تمرین

تمرین ۱- فرم کلی اعدادی را مشخص کنید که یا صفر می‌باشند و یا اینکه اگر طولشان بیش از یک است حتماً با یک رقم غیر صفر آغاز می‌شوند.

تمرین ۲ – فرم کلی اعداد در مبنای هشت را مشخص کنید در صورتی که بدانیم اینگونه اعداد حتماً با یک رقم صفر آغاز و سپس حرف O ظاهر می‌گردد و بعد از آن عدد مبنای هشت مشخص می‌شود. برای نمونه OO127 یک عدد مبنای ۸ است.

تمرین ۳- فرم کلی اعداد در مبنای شانزده در زبان C با رقم صفر و سپس X آغاز می‌شود. یک عبارت با قاعده برای بیان اعداد در مبنای شانزده ارایه نمایید. به طور مثال 0XAF25 یک عدد در مبنای ۱۶ است.

تمرین ۴- فرم کلی رشته‌ها را در زبان C با یک عبارت با قاعده معین کنید. توجه داشته باشید که در زبان C علامت (Backslash) \ در داخل یک رشته مفهوم خاص دارد.

تمرین ۵- فرم کلی اعداد صحیحی را تعیین کنید که در آنها ارقام به ترتیب صعودی ظاهر می‌شوند.

تمرین ۶- فرم کلی کلیهٔ رشته‌های اعداد مبنای دو را مشخص کنید که در آنها زیر رشته 011 ظاهر نگردد.

تمرین* ۷- کلیه رشته‌های اعداد مبنای دو را مشخص کنید که تعداد صفرها در آنها فرد و تعداد یک ها زوج باشد.

تمرین ۸- ماشین خودکار قطعی برای شناسه‌ها ترسیم نمایید.

تمرین ۹- ماشین خودکار قطعی بهینه برای عبارت (a | b | c)* abb ایجاد نمایید و سپس برنامه تحلیلگر لغوی را برای تشخیص اینگونه رشته‌ها بنویسید. ابتدا عبارت را به فرم غیرقطعی تبدیل نمایید و سپس قطعی و بهینه نمایید.

تمرین ۱۰- یک ماشین خودکار قطعی برای کلیه اعداد در مبنای دو که تعداد صفرهای آنها فرد و تعداد یک ها فرد است ایجاد کنید. در ضمن لااقل دو عدد صفر و دو عدد یک باید در این اعداد موجود باشد.

تمرین ۱۱- ماشین خودکار قطعی برای اعداد مبنای که حتماً دارای یک رقم صفر و یک رقم یک هستند را ایجاد نمایید به قسمی که اعداد با تعداد یک‌ها و صفرهای زوج را در حالت پذیرشی مشخص نماید. اعداد با تعداد یک‌های فرد و صفرهای زوج را در یک حالت پذیرشی مشخص نمایید. اعداد با تعداد یک‌های فرد و تعداد صفرهای فرد را نیز در حالت پذیرش دیگر ماشین مشخص کند. برای اعداد تعداد یک‌ها فرد و صفرها

زوج و همچنین صفرها فرد و یک‌ها زوج نیز باید دو حالت پذیرش مجزا وجود داشته باشد. ابتدا برای هر یک از چهار حالت ذکر شده ماشینهای خودکار را ترسیم نمایید و سپس یک ماشین خودکار غیر قطعی برای ترکیب آنها ایجاد نمایید. سپس ماشین خودکار را قطعی کنید.

تمرین ۱۲- یک ماشین خودکار قطعی برای گرامر یکی از زبانهای C یا پاسکال ایجاد نمایید. سپس با استفاده از روش ارایه شده در این فصل یک تحلیلگر لغوی برای آن ایجاد کنید.