فصل ۳

تحلیلگر لغوی

فصل سوم: تحلیلگر لغوی

اهداف رفتاری:

دانشجو پس از مطالعه این فصل با مفاهیم زیر آشنا خواهد شد:

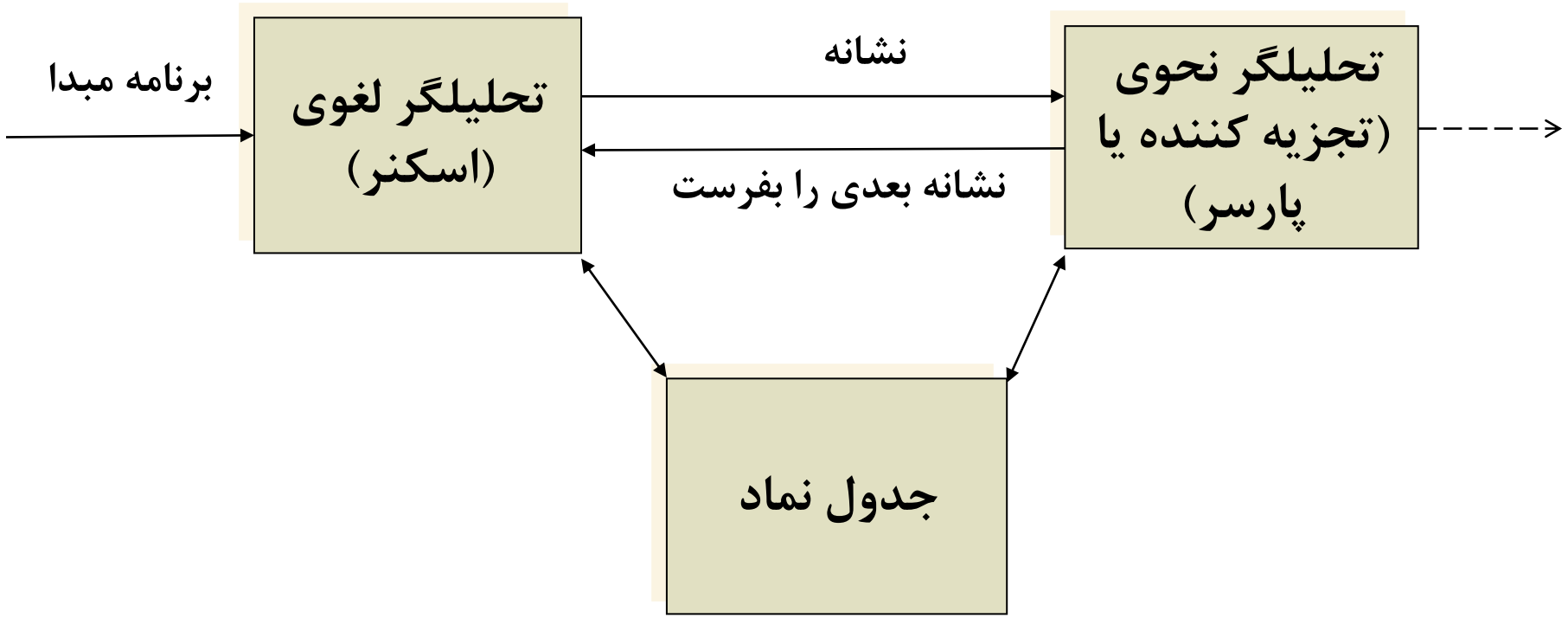
- **آشنایی با تحلیلگر لغوی**
- **عبارات و گرامر باقاعده**
- **تحلیلگر لغوی Lex**
- **ماشین خودکار قطعی و غیر قطعی و تبدیل آنها به یکدیگر**

۳-۱ وظایف تحلیلگر لغوی

۱. خواندن کاراکترهای ورودی
۲. تولید دنباله ای از توکن ها (نشانه ها) که تجزیه کننده برای تحلیل از آن استفاده می کند
۳. ثبت توکن ها در جدول نمادها
۴. حذف توضیحات برنامه، کاراکترهای فضای خالی (blank)، tab و خط جدید
۵. ایجاد ارتباط بین پیام های خطا از طرف کامپایلر به برنامه مبدا

- به عنوان مثال تحلیلگر لغوی ممکن است تعداد کاراکترهای خط جدید را شمارش کند از این رو همراه با هر پیام خطا، یک شماره خط چاپ می شود.

۳-۲ ارتباط با تجزیه کننده



ارتباط متقابل تحلیلگر لغوی با تجزیه کننده

دلایل جدا کردن فاز تحلیل به دو فاز تحلیل لغوی و تجزیه

۱- ساده تر بودن طراحی دو فاز

۲- افزایش کارایی کامپایلر به دلیل استفاده از
میانگیر برای خواندن کاراکترهای ورودی و پردازش توکن ها

۳-افزایش قابلیت حمل کامپایلر و محدود شدن مشخصه های
الفبای ورودی و دیگر خصوصیات وابسته به ماشین به تحلیلگر لغوی

نشانه (token)، الگو (pattern) و لغت (lexeme)

Token	Lexeme	Pattern
const	Const	Const
if	If	If
relation	<, <=, =, <>, >=, >	< or <= or = or <> or >= or >
Id	Pi, count D2	با حروف الفبا شروع و بدنبال آن می تواند هر تعداد حرف و رقم قرار داد.
Num	3.14, 0, 6, 02E23	هر ثابت عددی
literal	"Book" "core dumped"	هر رشته ای که بین دو علامت " " قرار گیرد.

نشانه (token)، الگو (pattern) و لغت (lexeme)

- به طور کلی در ورودی، مجموعه‌ای از رشته‌ها وجود دارند که برای آنها در خروجی، توکن‌های مشابهی تولید می‌شوند. این مجموعه از رشته‌ها با قانونی به نام **الگوی مربوط به نشانه** توصیف می‌شود. بنابراین **الگو**، قاعده ای است که مجموعه ای از لغات را توصیف می کند که می تواند نمایش یک نشانه خاص در برنامه مبدا باشد. این الگو با هر رشته در مجموعه منطبق است. یک **لغت**، دنباله‌ای از چند کاراکتر در برنامه‌ی مبدا است که سازگار با الگوی یک توکن می باشد. به عبارت دیگر، دنباله‌ای از کاراکترها که تشکیل یک توکن می‌دهند **لغت** آن توکن است.
- در اکثر زبان های برنامه نویسی، موارد زیر به عنوان توکن در نظر گرفته می شوند:
  - کلمات کلیدی
  - عملگرها
  - شناسه ها
  - ثابت ها
  - رشته های حرفی
  - نمادهای علامت گذاری مانند پرانتزها، کاماها و سمی کالن ها

نشانه (token)، الگو (pattern) و لغت (lexeme)

مثال:

در دستور پاسکال const pi = 3.1416 توکن ها و لغت ها را مشخص نمایید.

خصیصه توکن ها

- هرگاه بیش از یک الگو با یک لغت سازگار باشد تحلیلگر لغوی باید اطلاعات بیشتری درباره یک لغت خاص که سازگار با فازهای بعدی کامپایلر است فراهم نماید.
- مثال: الگوی num با هر دو رشته 0 و 1 سازگار است اما مولد کد باید بداند الگو واقعا با کدام رشته سازگار است.
- تحلیلگر اطلاعات مربوط به توکن ها را در خصیصه آن توکن ذخیره می نماید.
- توکن ها بر روی تصمیم تجزیه کننده و خصیصه ها بر ترجمه توکن ها تاثیر می گذارند.
- خصیصه اشاره گری است که به عضوی از جدول نمادها اشاره می کند و در آن، اطلاعات مربوط به توکن نگهداری می شود.
- گاهی نیاز است لغت مربوط به شناسه و نیز شماره خط که لغت اولین بار در آن دیده شده بدانیم. هر دوی این اطلاعات را می توان در در عضوی از جدول نمادها که مربوط به توکن است ذخیره نمود.

خصیصه توکن ها

- مثال: توکن ها و مقادیر خصیصه آن ها برای دستور فورترن E = M * C ** 2 به صورت زیر است:

<id, E اشاره گر به عضوی از جدول نمادها برای >

<assign_op,>

<id, M اشاره گر به عضوی از جدول نمادها برای >

<mult_op,>

<id, C اشاره گر به عضوی از جدول نمادها برای >

<exp_op>

<num, 2 مقدار صحیح>

۳-۳ خطای مرحله تحلیل لغوی

- در سطح لغوی، تنها تعداد کمی از خطاها قابل تشخیص هستند زیرا تحلیلگر لغوی، یک دید محلی و بسیار محدود از برنامه مبدا دارد و تمام برنامه ورودی را یکجا نمی بیند بلکه هر بار قسمت کوچکی از برنامه منبع را می بیند.
- مثال: اگر رشته fi در یک برنامه C برای اولین بار به صورت fi (a == f(x)) ظاهر شود، تحلیلگر لغوی نمی تواند تشخیص دهد که آیا fi اشتباه املایی کلمه کلیدی if است یا یک شناسه تابعی اعلام نشده. اسکنر توکن fi 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 خطا پرداز (Error- Handler) فراخوانده می شود تا آن خطا را به نحوی برطرف کند.

۳-۳ خطای مرحله تحلیل لغوی

منطبق نبودن هیچ کدام از الگوهای مربوط به تشخیص نشانه ها در زبان مبدا با پیشوندی از ورودی

روشهای تصحیح خطا

۱- Panic Mode
۲- حذف کاراکتر اضافی (تبدیل =:$ به =:)
۳- درج کاراکتر از قلم افتاده (تبدیل : به =:)
۴- جایگزینی یک کاراکتر بجای کاراکتر غلط (تبدیل :: به =:)
۵- جابجا نمودن دو کاراکتر مجاور هم (تبدیل := به =:)

۳-۳-۱ پوشش خطا- Panic mode

- ساده ترین شیوه پوشش خطا
- حذف کاراکترهای متوالی از ورودی باقیمانده تا پیدا شدن توکن قابل قبول تحلیلگر لغوی
- کافی بودن برای یک سیستم محاسباتی محاوره ای

۳-۴ تحلیلگر لغوی – پیاده سازی

روشهای پیاده سازی

استفاده از یک مولد تحلیلگر لغوی مانند کامپایلر Lex
تولید تحلیلگر لغوی را از روی مشخصاتی که بر پایه عبارات منظم استوار است
استفاده از روال هایی برای خواندن و میانگیری ورودی

نوشتن تحلیلگر لغوی به زبان های متداول برنامه نویسی برای سیستم ها و خواندن رشته ورودی با استفاده از امکانات I/O

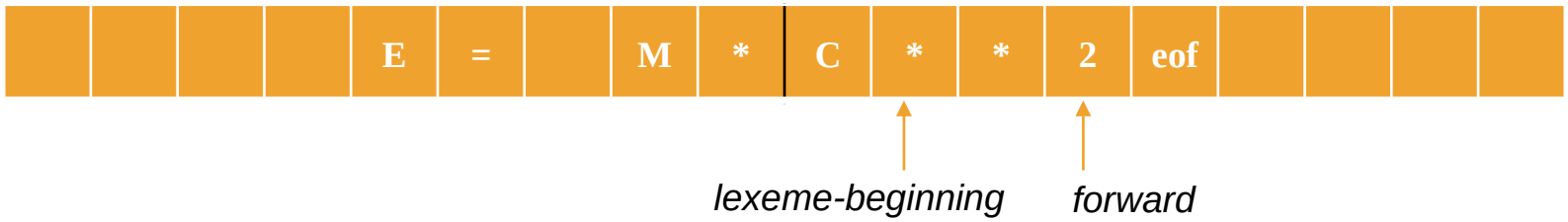
نوشتن تحلیلگر لغوی به زبان اسمبلی و مدیریت صریح خواندن رشته ورودی

روش هایی جهت بهبود کار اسکنر – بافر کردن ورودی

- در بسیاری از موارد اسکنر برای تشخیص نهایی توکن‌ها و مطابقت دادن آن با الگوهای موجود، نیاز دارد که تعداد بیشتری از کاراکترهای ورودی را بخواند. واضح است که مقدار زیادی از زمان در جابجایی کاراکترها سپری می‌شود و برای جلوگیری از این امر می‌توان از تکنیک‌های استفاده شده از بافر بهره مند شد. در این قسمت با تعدادی از روش‌های مفید برای افزایش سرعت تحلیل‌گر لغوی مانند استفاده از نگهبان برای مشخص کردن پایان میانگیر (بافر) آشنا می‌شویم.

جفت میانگیرها

- در بسیاری از زبان های مبدا، تحلیلگر لغوی قبل از پیدا کردن یک سازگاری باید به چند کاراکتر بعد از لغت نگاه کند تا بتواند الگوی مورد نیاز برای شناسایی آن لغت را پیدا کند. تحلیلگر لغوی برای این منظور از یک تابع برای برگرداندن کاراکترهای پیش نگر (look ahead) استفاده می کند. از آنجا که ممکن است زمان زیادی صرف انتقال کاراکترها شود از این رو از روش های بافرگیری استفاده می شود تا سربار مورد نیاز برای پردازش کاراکتر ورودی را کاهش دهد. در این روش از یک بافر که مانند شکل زیر به دو قسمت n کارکتری تقسیم شده است استفاده می شود که معمولا n تعداد کاراکتر موجود بر روی یک بلاک از دیسک است.



- بدین ترتیب بجای فراخوانی دستور read برای هر کاراکتر ورودی می توان دستور read را برای خواندن nکاراکتر به کار برد و آنها را در هر نیمه بافر قرار داد. اگر کمتر از n کاراکتر ورودی باقی مانده باشد آنگاه بعد از اتمام کاراکترهای ورودی یک کاراکتر مخصوص بنام eof (end - of - file) داخل بافر قرار می گیرد.
- Eof پایان فایل مبدا را مشخص می کند و با کاراکترهای ورودی متفاوت است.

جفت میانگیرها - کنترل حرکت اشاره گر forward

if *forward* at end of first half **then begin**

reload second half;

forward := *forward* +1;

end

else if *f**orward*** at end of second half **then begin**

reload first half;

move *forward* to beginning of first half

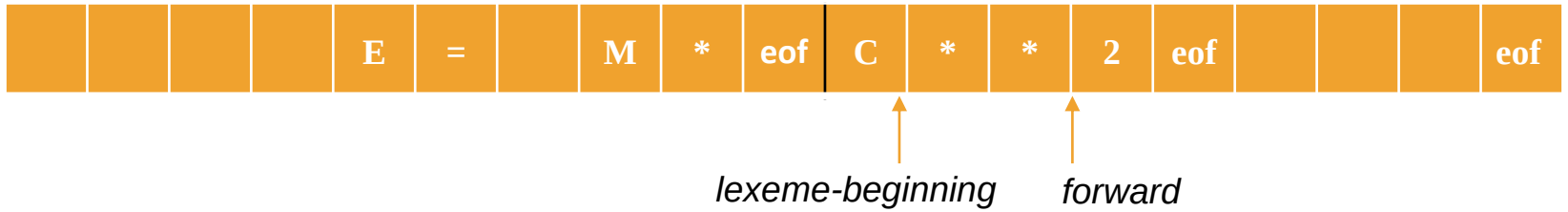
end

else

forward := *forward* +1;

نگهبان (sentinel)

- در روش قبل، در هر بار حرکت forward باید چک شود که آیا به انتهای هر یک از دو نیمه بافر رسیده است یا خیر یعنی دو عمل مقایسه باید انجام شود. حال اگر هر انتهای نیمه بافر را با استفاده از کاراکتر مخصوص eof بعنوان کاراکتر نگهبان مشخص کنیم این دو مقایسه به یک مقایسه کاهش می یابد. نگهبان کارکتر خاصی است که نمی‌تواند قسمتی از برنامه مبدا باشد.



نگهبان - کنترل حرکت اشاره گر forward

```
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
```

۳-۵ مشخصات نشانه ها

عبارات منظم یک نماد گذاری مهم برای مشخص کردن الگو ها هستند. هر الگو با مجموعه ای از رشته ها سازگار است. از این رو از عبارت های منظم می توان برای مجموعه هایی از رشته ها استفاده کرد.

قواعد تعریف

۱- عبارت باقاعده (منظم) ε زبان $\{\varepsilon\}$ (مجموعه حاوی رشته تهی) را مشخص می نماید.

۲- {a} عبارت باقاعده ای است که زبان a (نمادی از Σ) را می سازد.

۳- اگر a و b عبارات باقاعده باشند، اجتماع، الحاق، Kleene Star و positive closure آنها هم باقاعده است.

مثال عبارات باقاعده (منظم)

اگر $\Sigma=\{a,b\}$ باشد آنگاه:

۱- از عبارت باقاعده aUb مجموعه {a , b} ساخته می شود.

۲- از عبارت باقاعده (aUb).(aUb) مجموعه {aa, ab,ba,bb} تولید می شود.

۳- عبارت باقاعده a* کلیه رشته هایی با صفر یا چند a را تولید می کند.

۴- عبارت باقاعده * (aUb) رشته هایی با صفر یا چند نماد از b یا a را تولید می کند.

۳-۵-۱ عبارات باقاعده – خواص جبری

توصیف	اصل
/ جابجا پذیر است.	a U b = b U a
/ شرکت پذیر است	a U (b U c)=(a U b) U c
الحاق شرکت پذیر است.	(a.b).c = a.(b.c)
الحاق نسبت به / توزیع پذیر است	a.(b U c) = ab U ac (b U c).a = ba U ca
	a* = (a U ε)*
	a** = a*

مثال عبارات منظم در زبان پاسکال

id → letter (letter | digit)*
letter → A | B | … | Z | a | b | … | z
digit → 0 | 1 | … | 9

تعریف منظم مربوط به شناسه ها

digit → 0 | 1 | … | 9
digits → digit digit*
optional_fraction → . digits | ε
optional_exponent → (E (+ | - | ε) digits) | ε
num → digits optional_fraction optional_exponent

تعریف منظم اعداد بی علامت

اختصارات در نمادگذاری

1. **یک یا چند نمونه:**
   - $L^+ = \bigcup_{i=1}^{\infty} L^i$ (positive closure of L)
   - دارای اولویت و شرکت پذیری یکسان با عملگر *
   - $r^* = r^+ | \varepsilon$ و $r^+ = rr^*$
2. **صفر یا یک نمونه:**
   - عملگر یگانی پسوندی ?
   - نماد r? علامت اختصاری $r|\varepsilon$ است.
3. **کلاس های کاراکترها:**
   - نماد [abc]، عبارت منظم a|b|c را نمایش می دهد.
   - کلاس کاراکتری خلاصه [a-z]

اختصارات در نمادگذاری

digit → 0 | 1 | … | 9
digits → digit digit*
optional_fraction → . digits | ε
optional_exponent → (E (+ | - | ε) digits) | ε
num → digits optional_fraction optional_exponent

digit → 0 | 1 | … | 9
digits → digit^{+}
optional_fraction → (. digits)?
optional_exponent → (E (+ | -)? digits)?
num → digits optional_fraction optional_exponent

۳-۶ مجموعه های بی قاعده (نامنظم)

بعضی از زبان ها را نمی توان با هیچ عبارت منظمی توصیف کرد. برای نشان دادن محدودیت قدرت توصیفی عبارات منظم می توان موارد زیر را به عنوان نمونه ارائه داد:

عبارات منظم را نمی توان برای این موارد استفاده کرد:

۱. **ساختارهای متوازن یا تو در تو**
   - مجموعه تمام رشته ها با پرانتزهای تو در تو (گرامر مستقل از متن و نه منظم)

۲. **رشته های تکراری**
   - $\{wcw \mid w \in \{a,b\}^*\}$ نه منظم است و نه مستقل از متن

۳. **رشته هایی برای مقایسه دو عدد دلخواه**
   - عبارات منظم برای تعداد ثابتی از تکرار یا تعداد نامشخصی از تکرار یک ساختار هستند.
   - در عبارت منظم امکان مقایسه دو عدد دلخواه وجود ندارد.

۳-۷ گرامر با قاعده

فرم قوانین گرامر باقاعده

$A \rightarrow a$

$A \rightarrow \lambda$

$A \rightarrow aB$

a عنصری از الفبا و پایانه

B و A غیر پایانه

تشخیص نشانه ها

stmt → **if** *expr* **then** *stmt* | **if** *expr* **then** *stmt* **else** *stmt* | *ε*

expr → *term* **relop** *term* | *term*

term → **id** | **num**

delim → blank | tab | newline
ws → delim^{+}

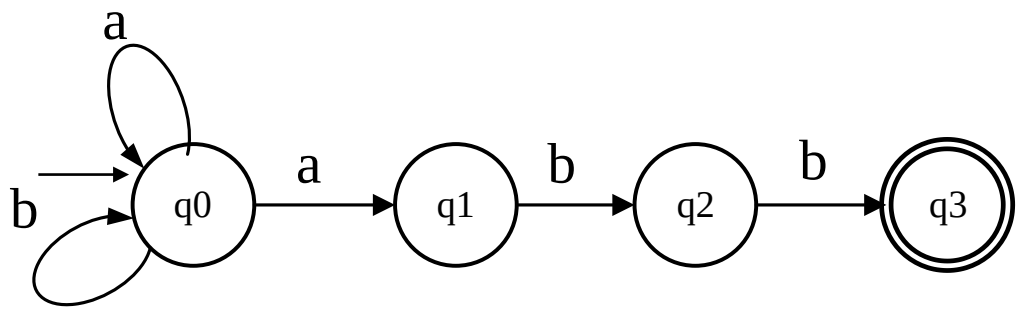
if → `if`
then → `then`
else → `else`
relop → < | <= | = | <> | > | >=
id → letter (letter | digit)*
num → digit^{+} (. digit^{+})? (E (+|-)? digit^{+})?

نمودار های تبدیل یا نمودار تغییر حالت یا دیاگرام‌های انتقال

- برای پیاده سازی دستی یک اسکنر از ابزاری بنام دیاگرام انتقال کمک می‌گیریم. بدین معنی که بعنوان یک مرحله میانی در ساختن تحلیل‌گر لغوی باید ابتدا یک نمودار گردشی ایجاد کنیم. یک دیاگرام انتقال در واقع یک گراف جهت دار می باشد که هر یک از گره‌های آن معرف یک وضعیت (state) است. یکی از وضعیت‌ها به عنوان وضعیت شروع و یک (یا چند) وضعیت به عنوان وضعیت خاتمه مشخص می‌گردند. برچسب (label)هایی روی لبه‌های یک دیاگرام انتقال قرار داده می‌شوند که مشخص می‌کنند در چه صورتی می‌توان از یک وضعیت به وضعیت دیگر رفت. برچسب other مربوط به هر کاراکتری دیگری غیر از کاراکترهایی است که با سایر یال ها از یک وضعیت خارج شده است. هر دیاگرام انتقال معرف یک زبان است. با خواندن کاراکترهای یک رشته و تطبیق آنها با برچسب‌های دیاگرام انتقال معرف یک زبان و پیمایش آن دیاگرام می‌توان مشخص نمود که آیا آن رشته متعلق به زبان مورد نظر است یا خیر.
- فرض می کنیم نمودارهای تبدیل این بخش **معین** یا **قطعی** یا deterministic هستند (DFA)
-

DFA و NFA

- دیاگرام انتقال زیر و عبارت منظم (a|b)*abb هر دو زبانی را توصیف می‌کنند که شامل رشته‌های تشکیل شده از نشانه های a و b است که به abb ختم می‌شوند.

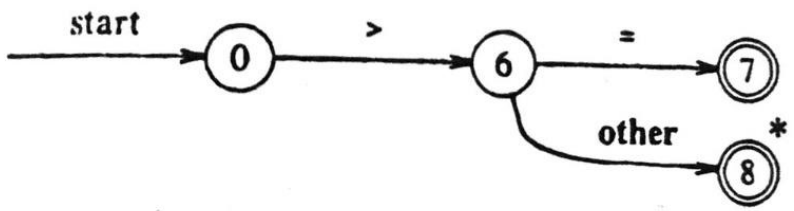


- دیاگرام فوق یک دیاگرام انتقال نامعین (NFA) است که در آن یال‌های خارج شده از یک گره آن یکسان است و یا دارای لبه‌هایی با برچسب λ است.

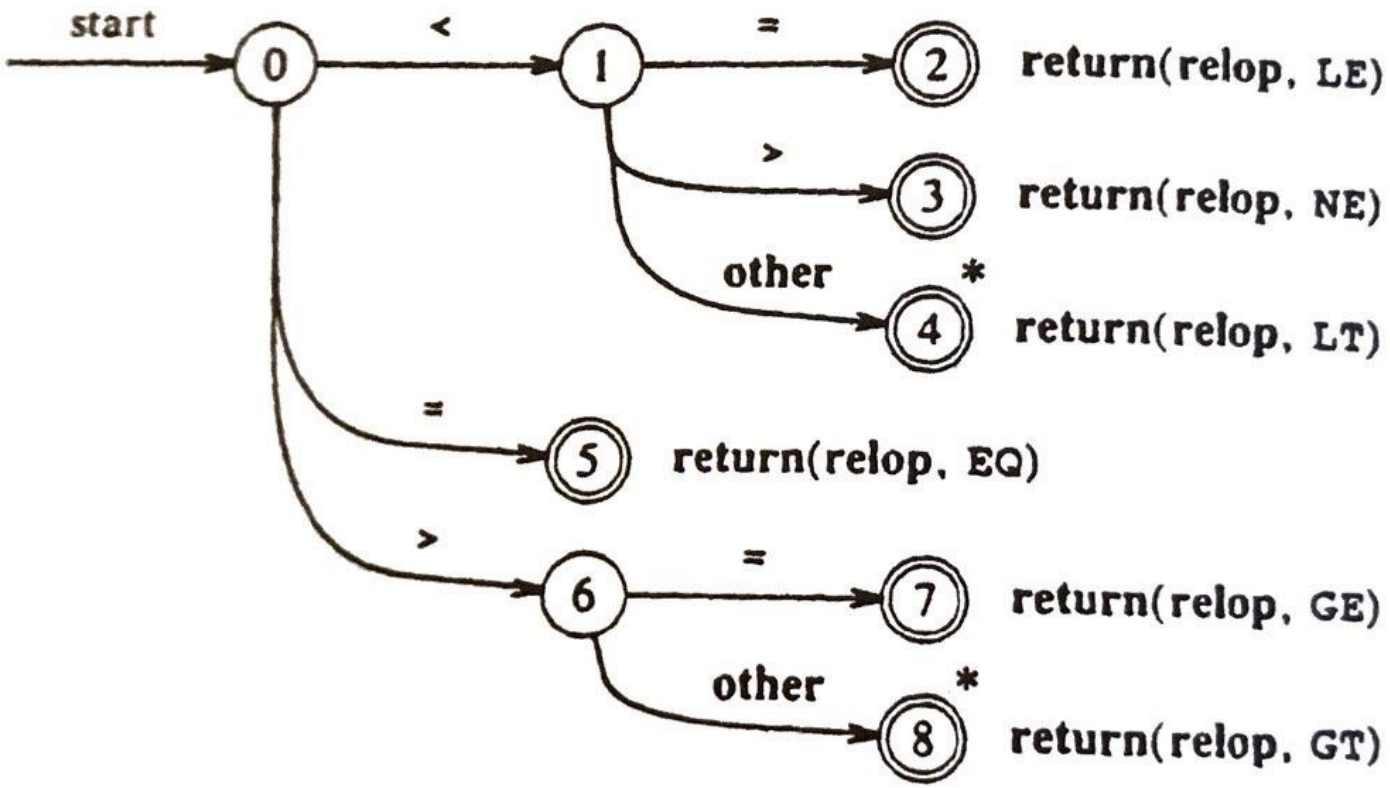
- هر دیاگرام نامعین (NFA) را می‌توان به یک دیاگرام انتقال معین (DFA) معادل تبدیل کرد.

REGULAR EXPRESSION	TOKEN	ATTRIBUTE-VALUE
ws	-	-
if	if	-
then	then	-
else	else	-
id	id	pointer to table entry
num	num	pointer to table entry
<	relop	LT
<=	relop	LE
=	relop	EQ
<>	relop	NE
>	relop	GT
>=	relop	GE

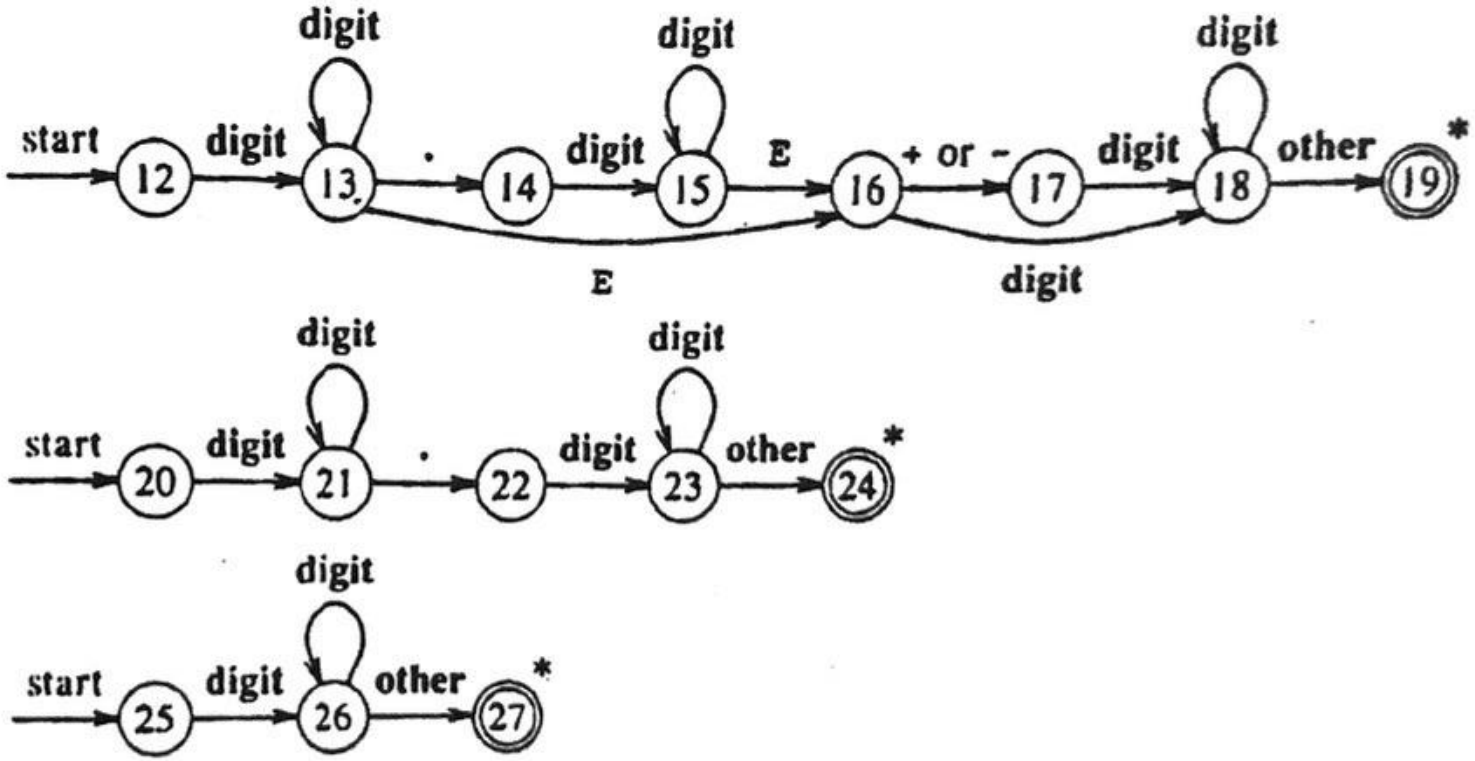
شکل ۳-۱۰: الگوهای عبارت منظم برای نشانه‌ها .



شکل ۳-۱۱: نمودار تبدیل =<.

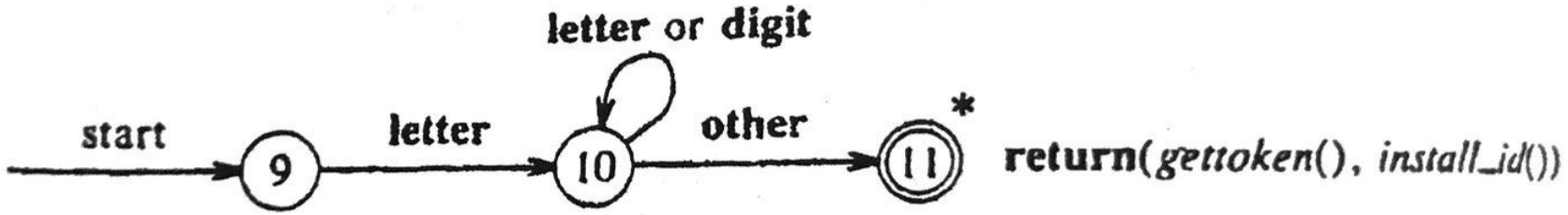


شکل ۳-۱۲: نمودار تبدیل برای عملگرهای رابطه‌ای .



شکل ۳-۱۴: نمودار تبدیل حالت برای اعداد بدون علامت در پاسکال.

$$num \rightarrow digit^{+} (. digit^{+})? (E (+|-)? digit^{+})?$$

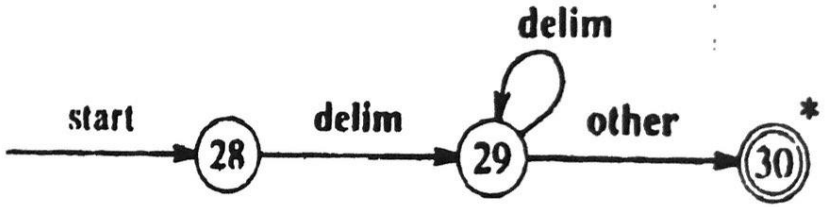


شکل ۳-۱۳: نمودار تبدیل برای شناسه‌ها و کلمه‌های کلیدی .

id → letter (letter | digit)*

ws → delim^{+}

delim → blank | tab | newline



پیاده سازی نمودار تبدیل حالت

```
int state = 0, start = 0;
int lexical_value;
    /* to "return" second component of token */

int fail()
{
    forward = token_beginning;
    switch (start) {
        case 0:   start = 9; break;
        case 9:   start = 12; break;
        case 12:  start = 20; break;
        case 20:  start = 25; break;
        case 25:  recover(); break;
        default:   /* compiler error */
    }
    return start;
}
```

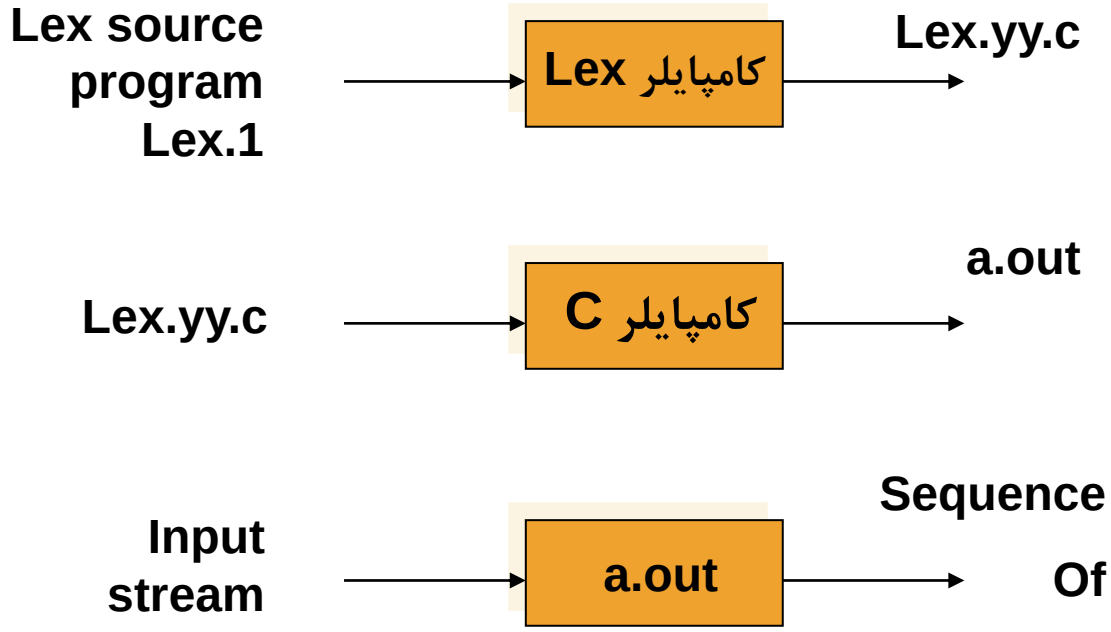
شکل ۳-۱۵: دستور C برای پیدا کردن حالت شروع بعدی.

```
token nexttoken()
{   while(1) {
        switch (state) {
        case 0:   c = nextchar();
            /* c is lookahead character */
            if (c==blank || c==tab || c==newline) {
                state = 0;
                lexeme_beginning++;
                    /* advance beginning of lexeme */
            }
            else if (c == '<') state = 1;
            else if (c == '=') state = 5;
            else if (c == '>') state = 6;
            else state = fail();
            break;
            .../* cases 1-8 here */
        case 9:   c = nextchar();
            if (isletter(c)) state = 10;
            else state = fail();
            break;
        case 10:  c = nextchar();
            if (isletter(c)) state = 10;
            else if (isdigit(c)) state = 10;
            else state = 11;
            break;
        case 11:  retract(1); install_id();
            return ( gettoken() );
            .../* cases 12-24 here */
        case 25:  c = nextchar();
            if  (isdigit(c)) state = 26;
            else state = fail();
            break;
        case 26:  c = nextchar();
            if (isdigit(c)) state = 26;
            else state = 27;
            break;
        case 27:  retract(1); install_num();
            return ( NUM );
        }
    }
}
```

شکل ۳-۱۶: یک برنامهٔ C برای تحلیل‌گر لغوی

۳-۸ تولیدکننده تحلیلگر لغوی - Lex

روش ایجاد تحلیلگر لغوی توسط Lex



۱- آماده شدن فایلی حاوی مشخصه تحلیلگر لغوی برای Lex

۲- تبدیل محتوای فایل به برنامه در زبان C

۳- کامپایل برنامه تولیدی همراه کتابخانه برنامه تحلیل لغوی

۴- خروجی: برنامه تحلیلگر لغوی

۳-۸-۱ Lex - اجزای برنامه

بخش های برنامه مبدا **Lex**

- اعلان ها
- قواعد ترجمه
- زیربرنامه های کمکی c

ترتیب در متن برنامه مبدا **Lex**

اعلان ها

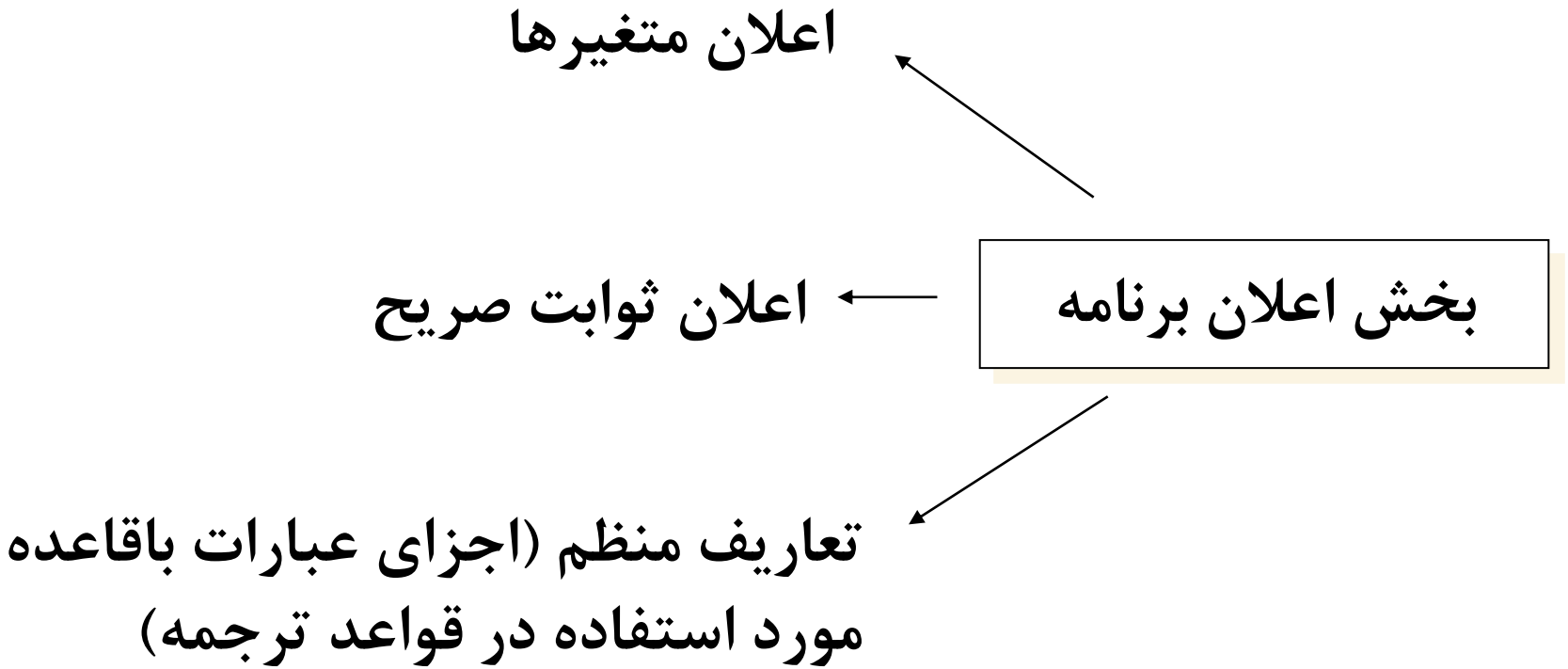
%%

قواعد ترجمه

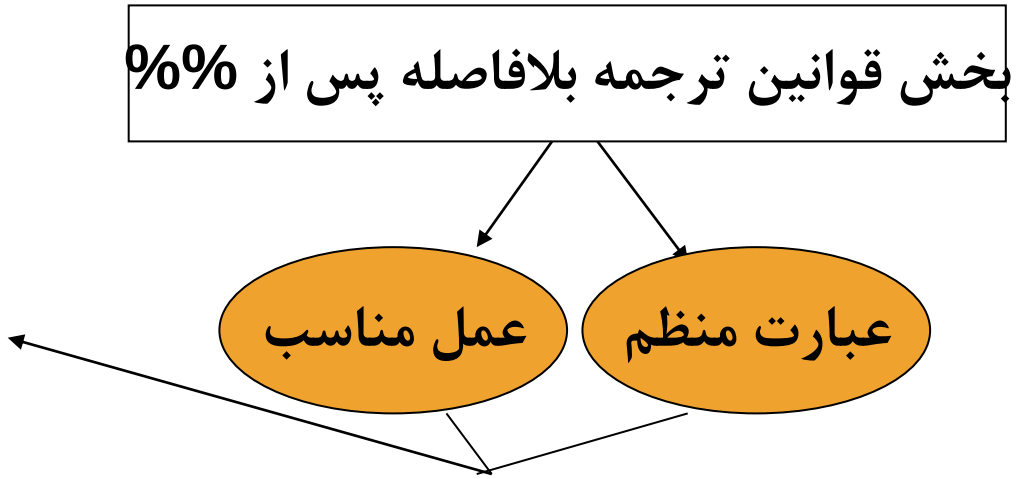
%%

زیربرنامه های کمکیc

۳-۸ تولیدکننده تحلیلگر لغوی



۳-۸-۱ Lex - اجزای برنامه



P1 {action 1}

P2 {action 2}

P3 {action 3}

.

.

.

Pn {action n}

زیربرنامه های کمکی C

مورد استفاده دراجرای action ها

۳-۹ ماشین خودکار متناهی

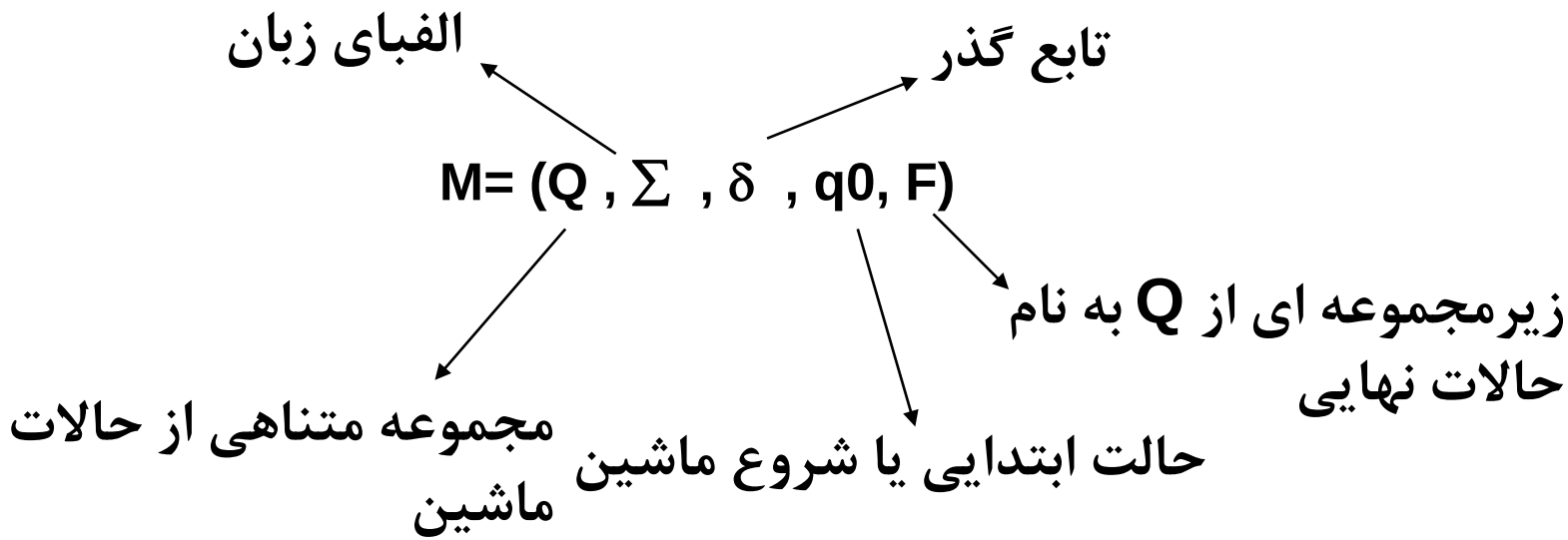
ابزاری برای تشخیص ساختارهای موجود زبان در دنباله ورودی از نشانه ها و پذیرفتن یا نپذیرفتن دنباله کاراکترهای ورودی

انواع ماشین های خودکار

- **۱- ماشین خودکار متناهی قطعی DFA**
- **۲- ماشین خودکار متناهی غیر قطعی NFA**

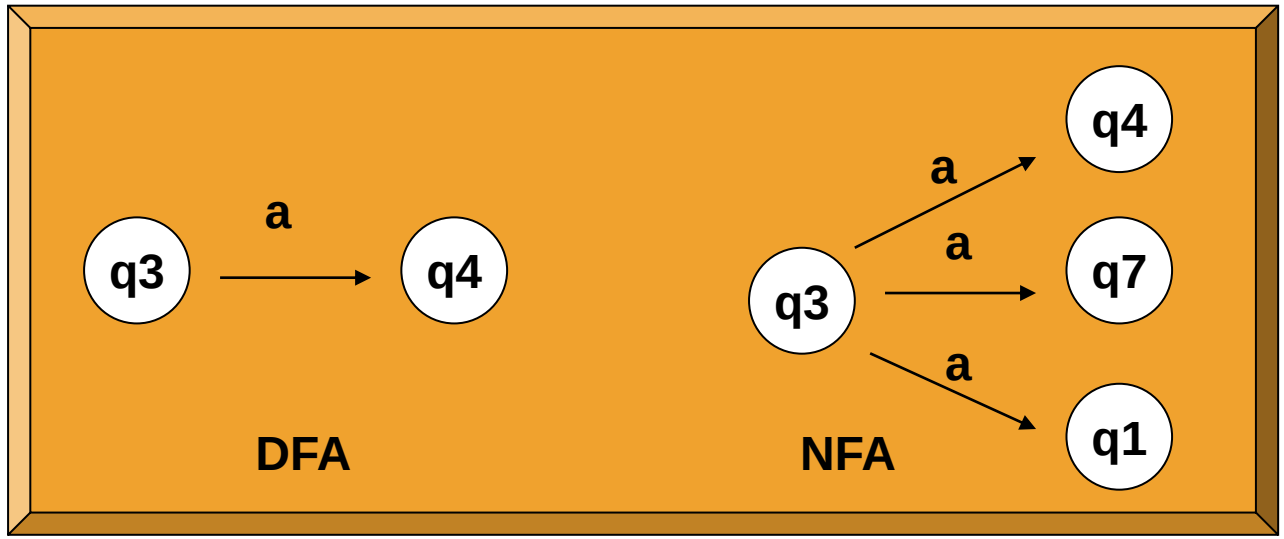
۳-۹-۱ ماشین خودکار قطعی

یک مدل ریاضی است متشکل از ۵تایی



۳-۹-۲ ماشین خودکار غیر قطعی

DFA ای که می توان از هر حالت با عناصر ورودی یکسان به حالات مختلفی رسید.

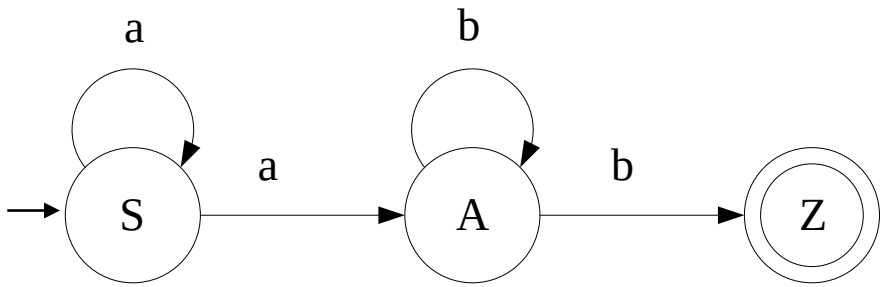


مثال از NFA

$S \rightarrow aS \mid aA$

$A \rightarrow bA \mid b$

تبدیل به ماشین غیرقطعی

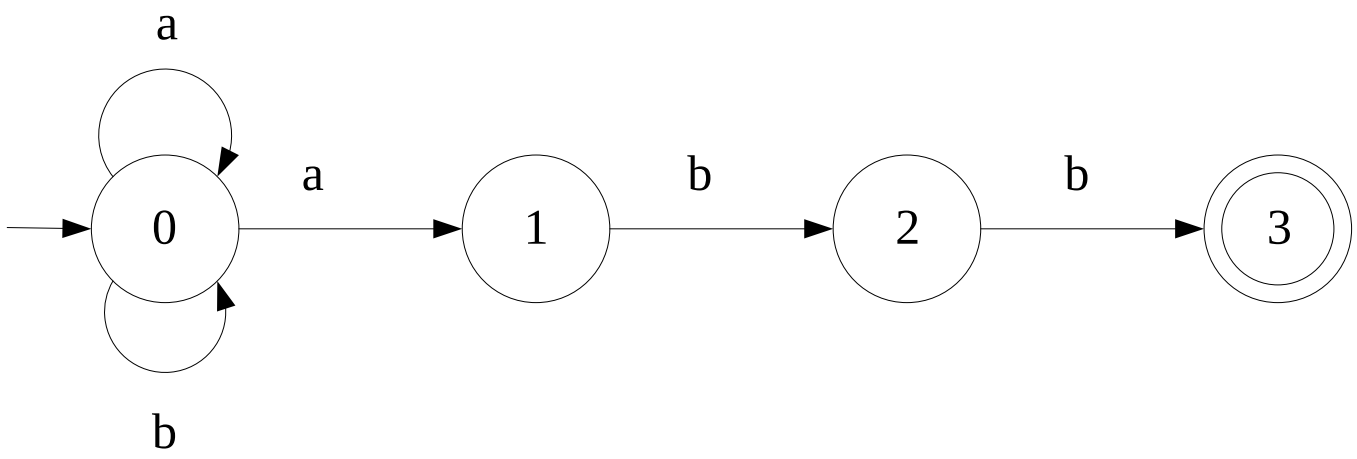


اشتقاق	محاسبه	رشته پردازش شده
$S \rightarrow aS$	[S , aabb] → [S , abb]	a
$\rightarrow aaA$	[A,bb]	aa
$\rightarrow aabA$	[A , b]	aab
$\rightarrow aabb$	[Z,]	aabb

پذیرفته توسط ماشین

مثال از NFA

زبان: **(a U b)* abb**



۳-۹-۳ تبدیل NFA به DFA

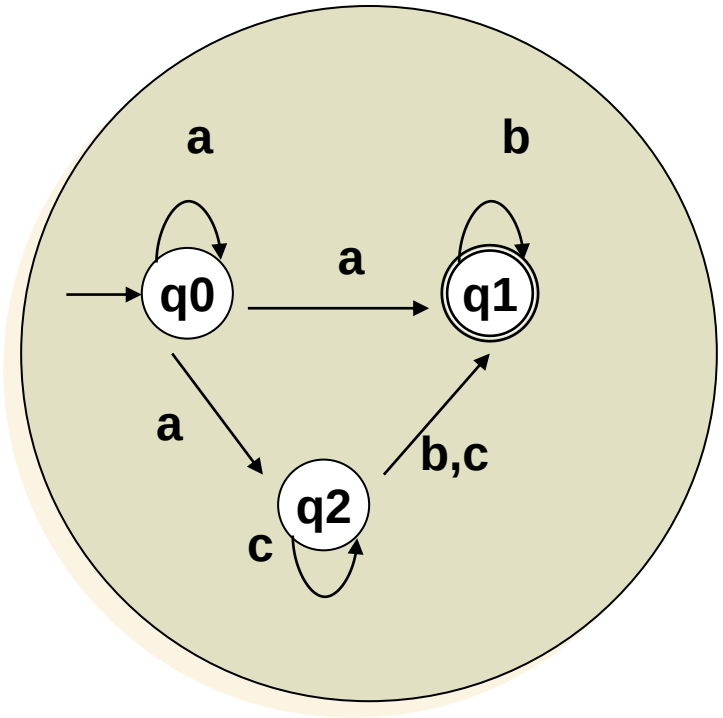
تعریف : هبستگی لامبدا یا λ- closure (qi) به صورت بازگشتی :

۱- پایه: qi ∈ λ -closure (qi)

۲- مرحله بازگشت: اگر qj یک عنصر از λ –closure(qj) باشد و اگر qk∈ (gj , λ) آنگاه qk ∈ λ -closure (qj) است.

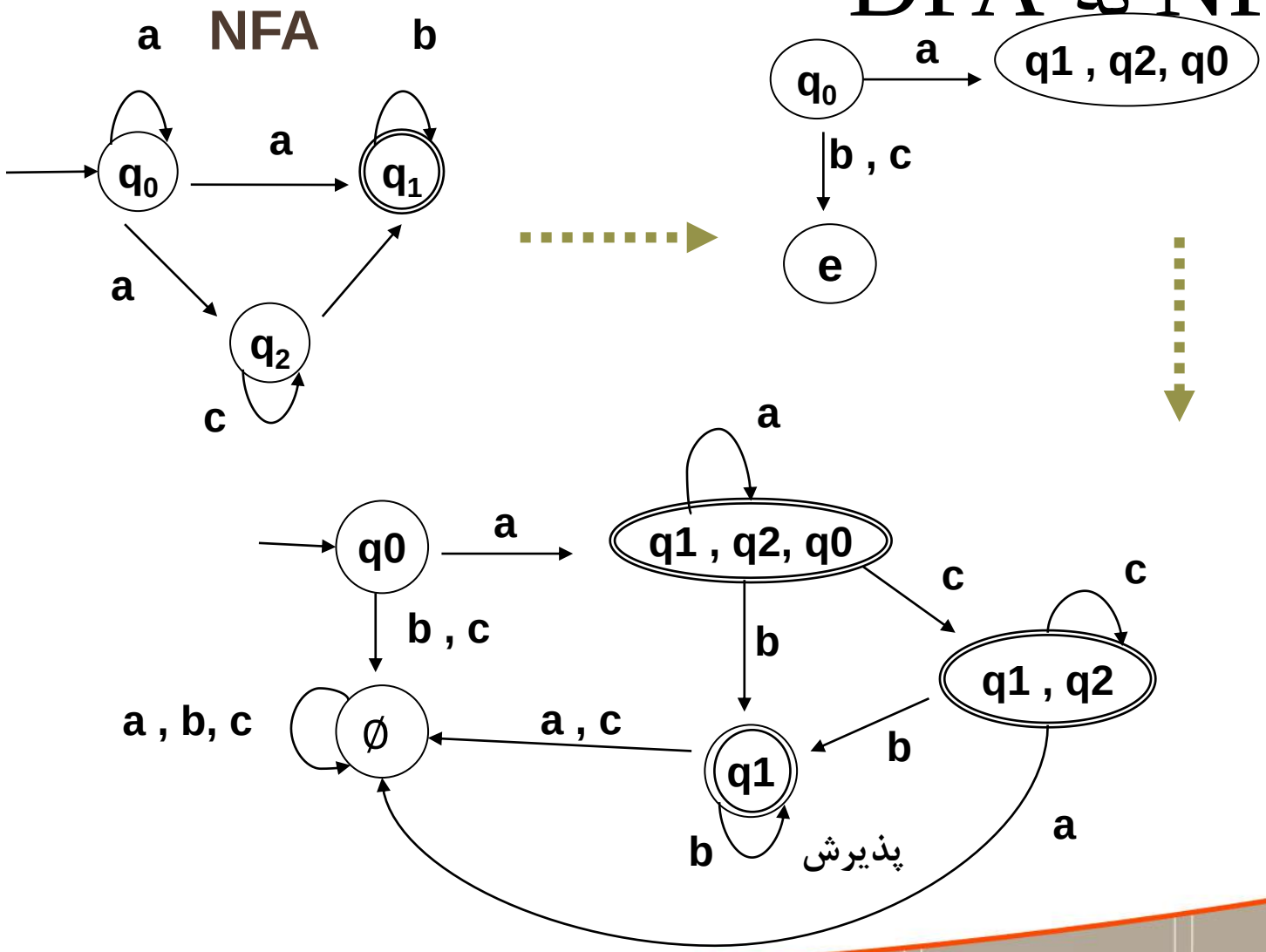
۳- همبستگی : qj در λ -closure (qi) است اگر بتواند با تکرار متناهی از مرحله بازگشت بدست آید.

مثال همبستگی لامبدا



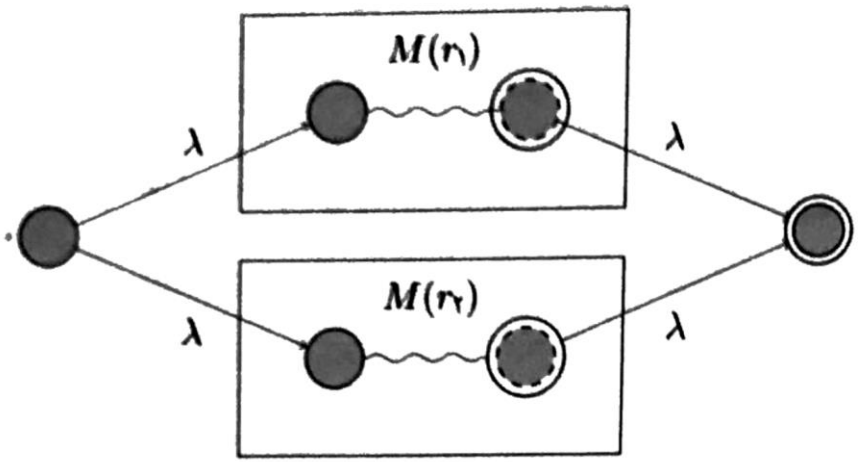
حالت	a	b	c
q0	{ q1 , q2, q0 }	-	-
q1	-	{q1}	-
q2	-	{q1}	{ q1 , q2}

مثال تبدیل NFA به DFA



۳-۹-۴ ساخت NFA از عبارات با قاعده

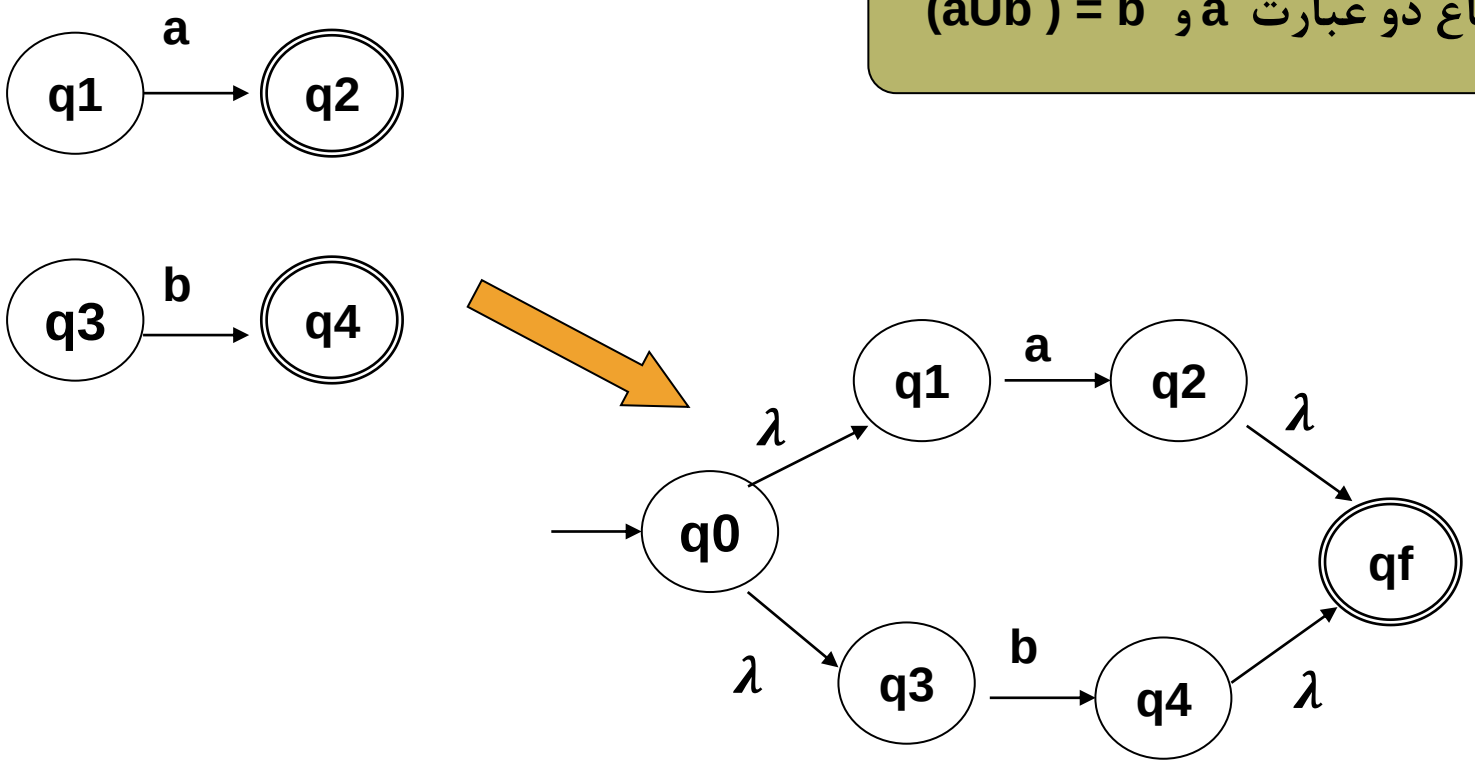
قاعده اول = اجتماع



L (M2) U L (M1)

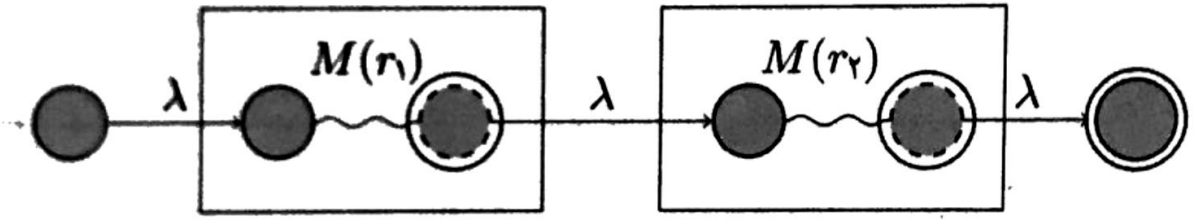
مثال اجتماع دو عبارت با قاعده

اجتماع دو عبارت a و b = (aUb)



۳-۹-۴ ساخت NFA از عبارات با قاعده

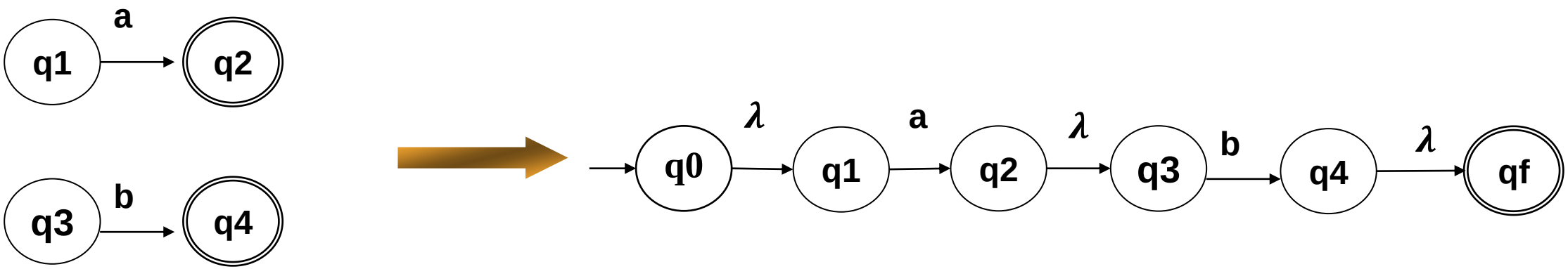
قاعده دوم= الحاق



الحاق L (m1) , L (M2) یا L(M1) . L (M2)

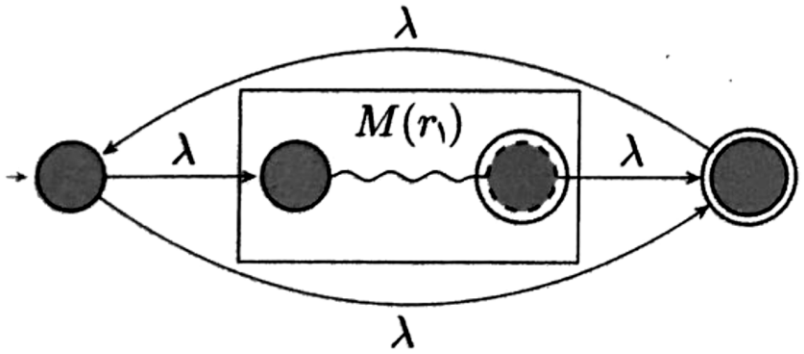
مثال الحاق دو عبارت با قاعده

الحاق دو عبارت a , b یا ab



۳-۹-۴ ساخت NFA از عبارات با قاعده

قاعده سوم- kelin star

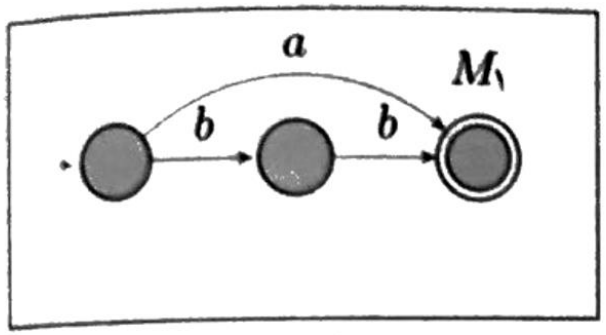


kelin star L (M1) یا L (M1) *

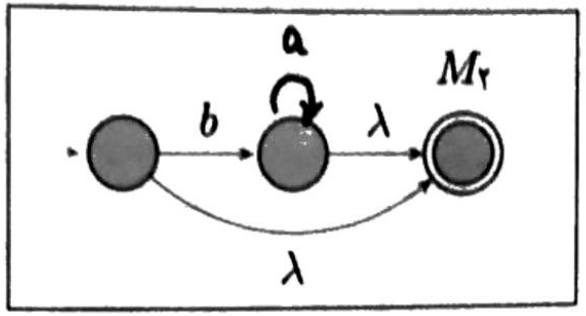
مثال تشکیل NFA از عبارات باقاعده

$$r = (a + bb)^*(ba^* + \lambda)$$

(الف) M_1، $L(a + bb)$ را می‌پذیرد.
(ب) M_2، $L(ba^* + \lambda)$ را می‌پذیرد.



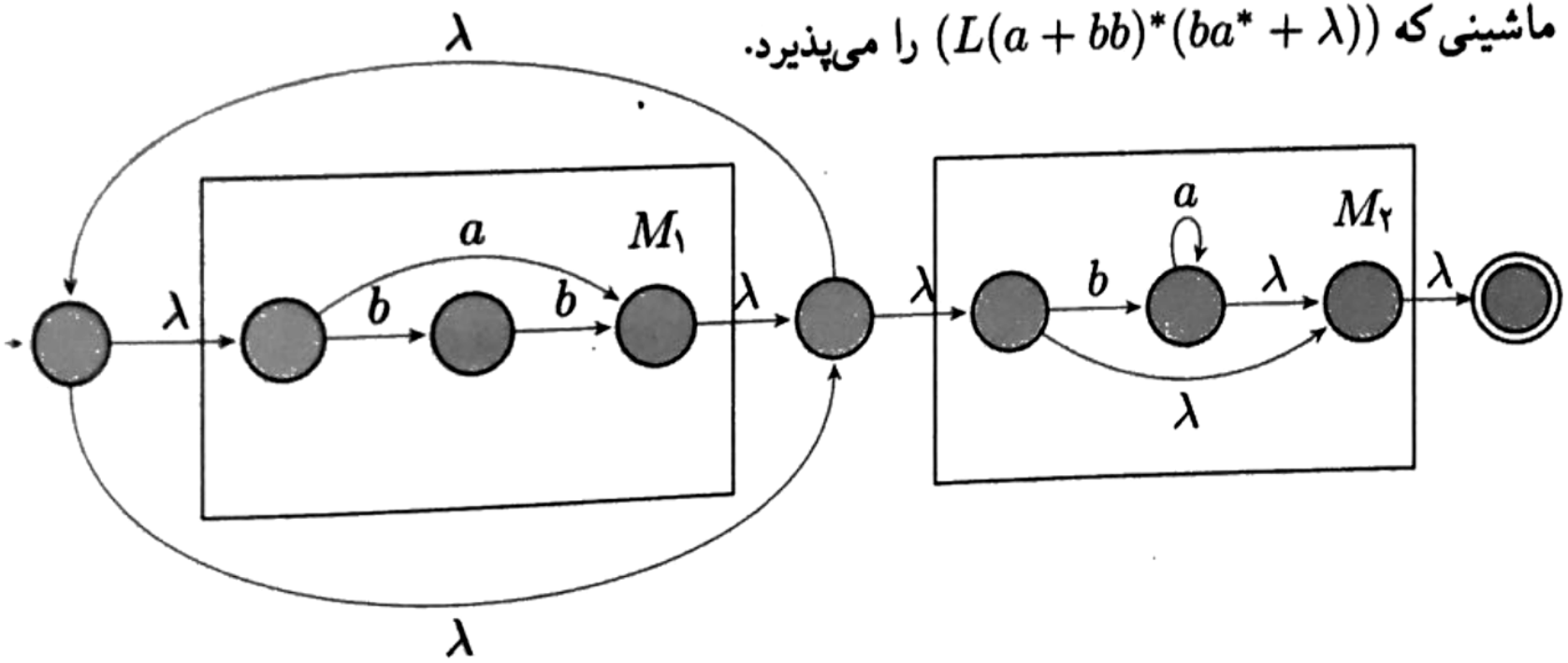
(الف)



(ب)

مثال تشکیل NFA از عبارات باقاعده

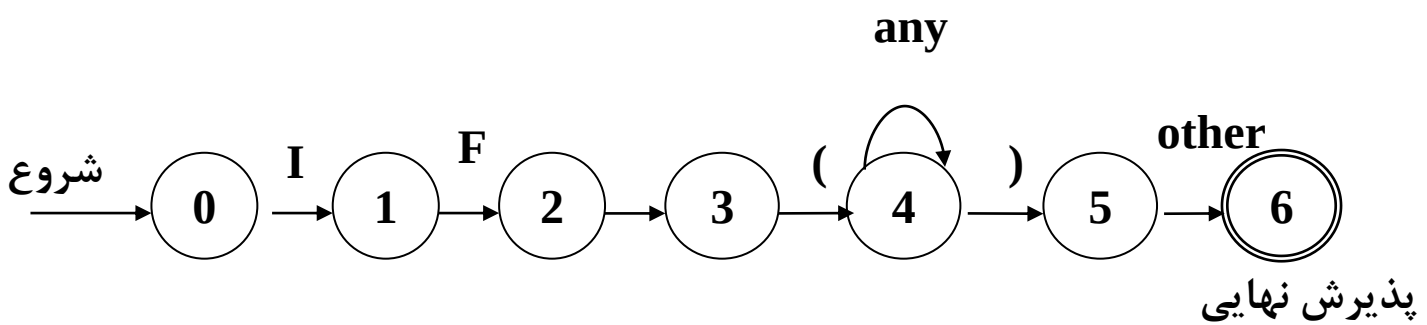
ماشینی که $(L(a+bb)^*(ba^*+\lambda))$ را می‌پذیرد.



مثال یک الگو برای تحلیل گر لغوی

با کمک ماشین قطعی IF الگوی تشخیص ساختار

other = رقم یا حرف



پایان فصل ۳

تحلیلگر لغوی