## سوال اول:

**Token:** یک جفت است که از یک نام رمز و یک مقدار اختیاری تشکیل شده است. در برنامه‌نویسی، توکن‌ها به عنوان واحدهای معنایی در کد منبع شناخته می‌شوند. مثال: در زبان C، int value = 100;، توکن‌ها شامل کلیدواژه (int)، شناسه (value)، عملگر (=) و ثابت (۱۰۰) هستند.

**Patten:** توصیفی از شکلی است که واژگان یک نشانه ممکن است داشته باشند. الگوها معمولاً در تحلیل واژه‌ای (Lexical Analysis) کامپایلرها استفاده می‌شوند. مثلاً الگوی یک عدد مثبت می‌تواند [۹-۱]+ باشد.

**Lexeme:** دنباله ای از کاراکترها در برنامه منبع است که با الگوی یک نشانه مطابقت دارد. به عبارت دیگر، نمایانگر واژه‌ای است که به یک توکن تبدیل می‌شود. مثلاً در عبارت ۴۲ + ۳.۱۴، Lexemes ها شامل ۴۲، + و ۳.۱۴ هستند.

## سوال دوم:

**کامپایلر:**

برنامه‌ای نرم‌افزاری است که کدهای نوشته شده توسط برنامه‌نویس را به زبان پایه ماشین تبدیل می‌کند. کامپایلر ابتدا کدهای منبع را به کدهای سطح پایین ترجمه می‌کند. سپس این کدها توسط اسمبلر به کدهای دودویی یا همان باینری قابل درک برای ماشین تبدیل می‌شوند. مثال: در زبان C ، کامپایلر کد منبع را به کد اسمبلی ترجمه کرده و سپس اسمبلر آن را به کد دودویی تبدیل می‌کند. برنامه‌های کامپایل شده سریع‌تر اجرا می‌شوند و خطایابی در زبان‌های کامپایلری آسان‌تر است.

**مفسر:**

مفسر دستورات برنامه را به صورت خط به خط اجرا می‌کند. مفسر نتیجه ترجمه خود را از برنامه اصلی تولید می‌کند. به عبارت دیگر، کدها خط به خط تفسیر و سپس اجرا می‌شوند. مثال: زبان‌های مفسری مانند Python و JavaScript از مفسر استفاده می‌کنند. مفسر سرعت ترجمه بالاتری نسبت به کامپایلر دارد و خطایابی در زبان‌های مفسری ممکن است پیچیده‌تر باشد.

## سوال سوم:

یک regex برای کلمه کلیدی select می تواند بصورت زیر باشد:

[Ss][Ee][Ll][Ee][Cc][Tt]

## سوال چهارم:

a)

[bcdfghjklmnpqrstvwxyz]* a ([abcdfghjklmnpqrstvwxyz] | a)* e ([bcdfghjklmnpqrstvwxyz] | e)* i ([bcdfghjklmnpqrstvwxyz] | i)* o ([bcdfghjklmnpqrstvwxyz] | o)* u ([bcdfghjklmnpqrstvwxyz] | u)*.

b)

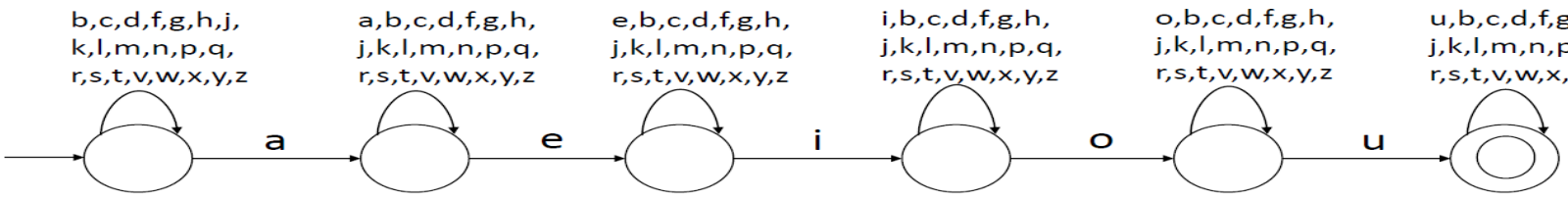
a* b* ... z*.

c)

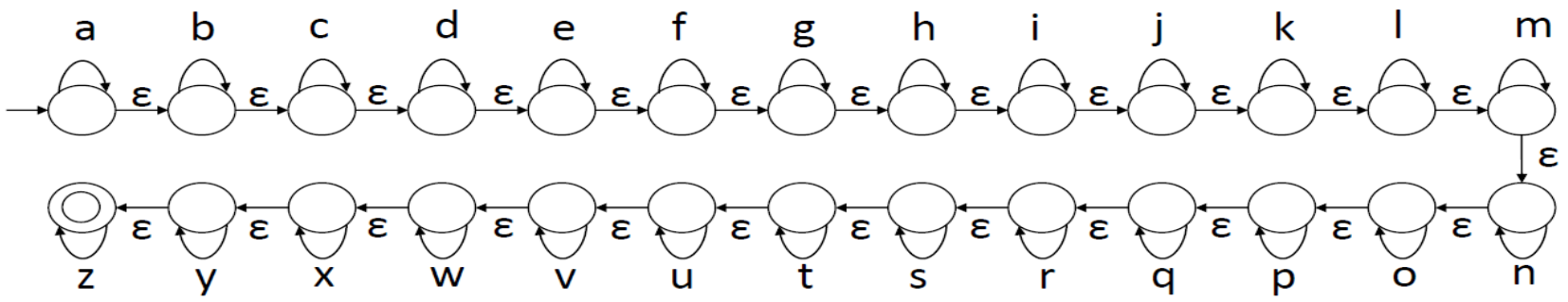
((a(aa)*b(b(aa)*b)*b(aa)*ab|a) |((aa)*b)) ((b(aa)*b)|( a(aa)*b(b(aa)*b)*b(aa)*ab|a)).

## سوال پنجم:

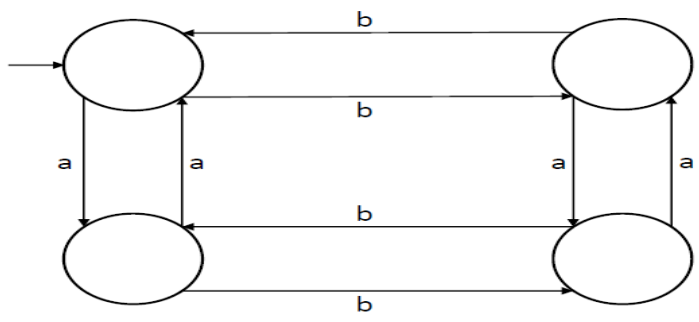
a)




b)

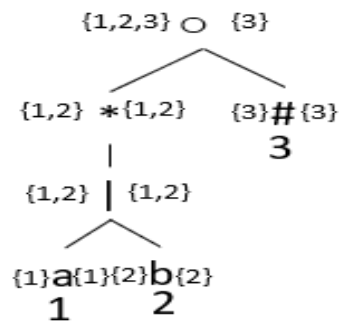



c)

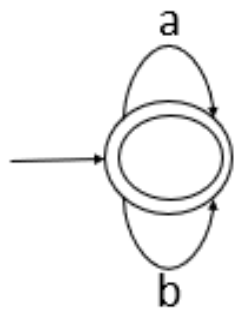



## سوال ششم:

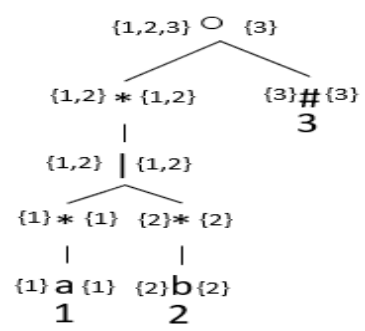
**(a|b)*:**



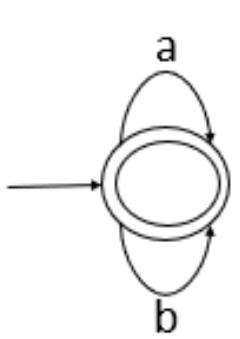

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3} |
| 2 | {1,2,3} |
| 3 | ----- |



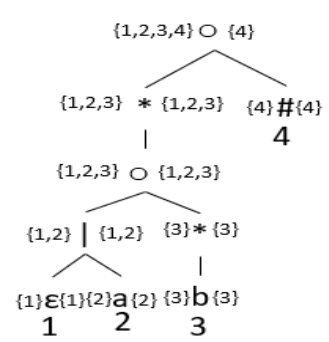

**(a*|b*)*:**



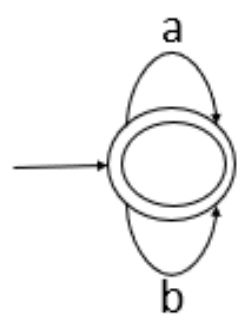

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3} |
| 2 | {1,2,3} |
| 3 | ----- |




**((ε|a)b*)*** :



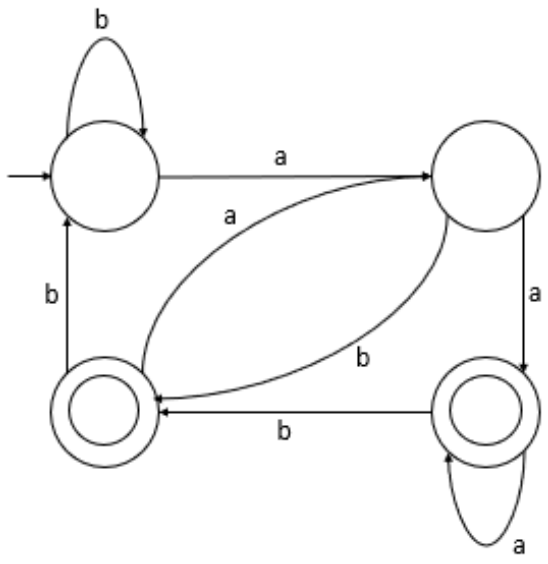

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3,4} |
| 2 | {1,2,3,4} |
| 3 | {1,2,3,4} |
| 4 | ------ |



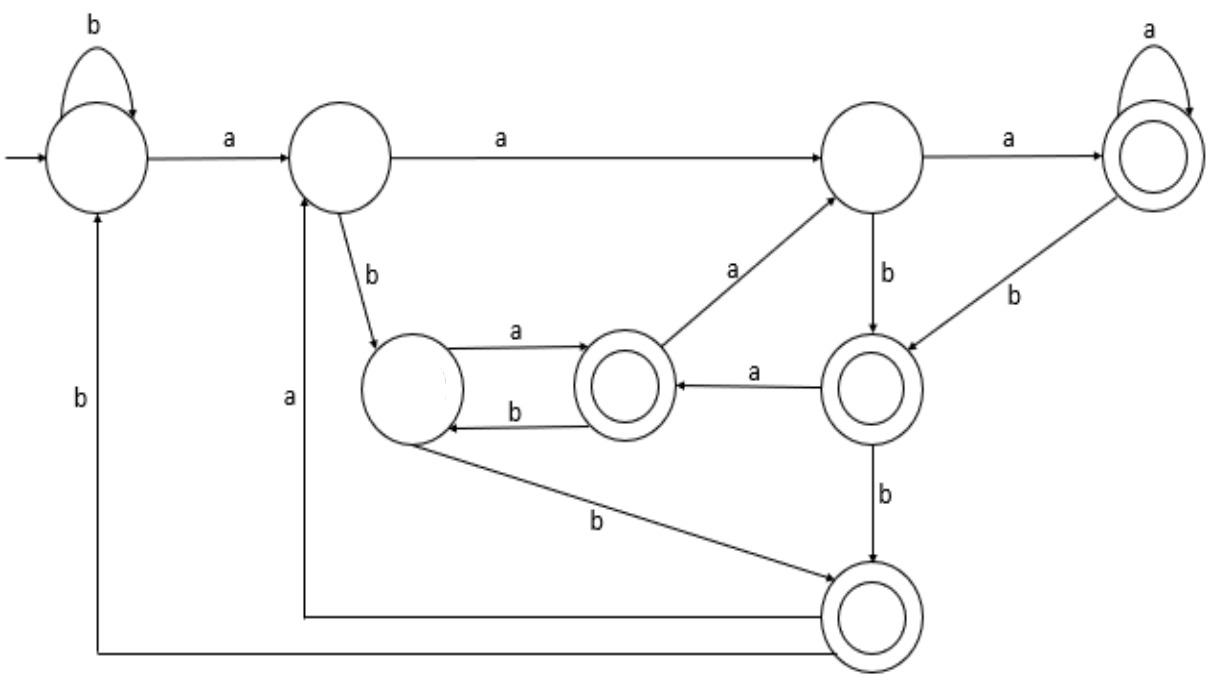

همانطور که می بینیم هر سه عبارت دارای DFA مشابه هستند، پس می توان گفت که هر سه معادل هم هستند.

## سوال هفتم:

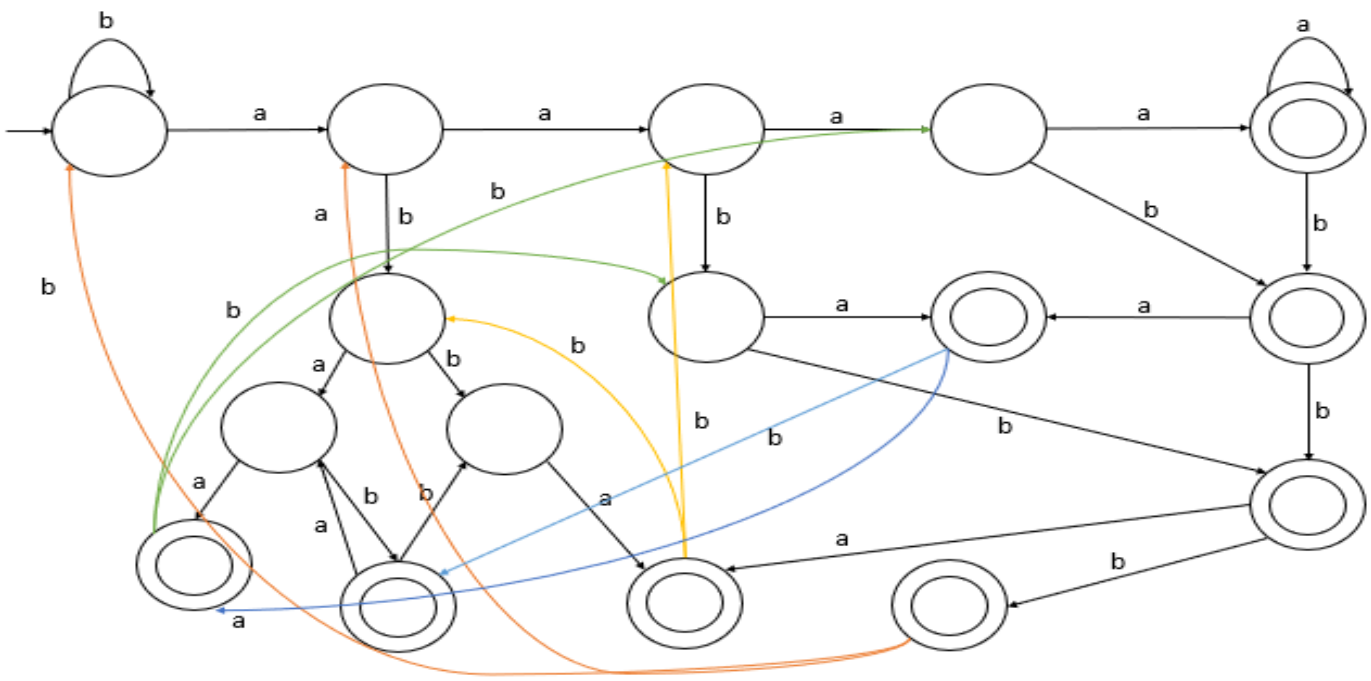
**(a|b)*a(a|b)** :




**(a|b)*a(a|b)(a|b)** :




**(a|b)*a(a|b)(a|b)(a|b)** :




تعداد استیت‌های نهایی به ازای هر (a|b) ۲ برابر می شود. همچنین تعداد کل استیت‌ها دوبرار می شود.

## سوال هشتم:

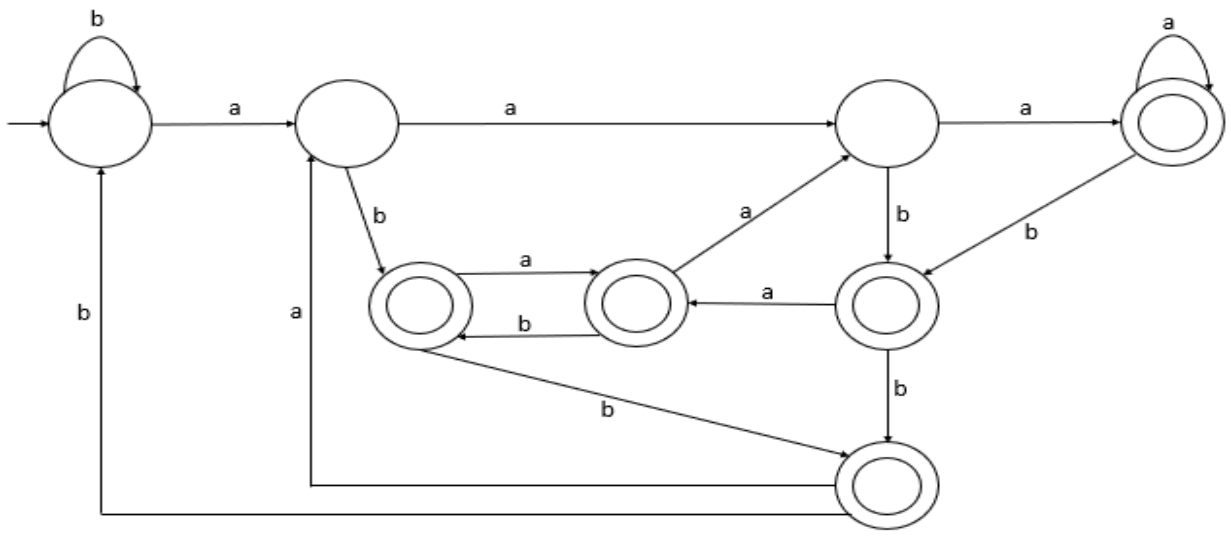
نداریم(:

## سوال نهم:

با استفاده از استقرا می‌توانیم.

پایه: نشان می‌دهیم برای n=2 صحیح است.

$2^2 = 4 \rightarrow$ (a|b)*a(a|b)(a|b):



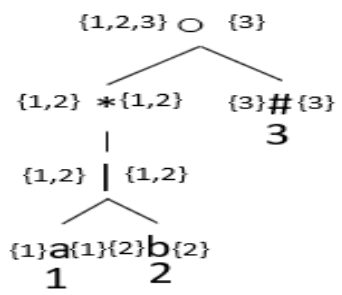

حال فرض می کنیم که برای n=k صحیح است. باید نشان دهیم که آیا برای k+1 حالت هم صحیح است یا نه. در حالت قبلی فرض کردیم $2^k$ درست است. حال میدانیم به ازای هر (a|b) تعداد استیت ها دوبرابر می شود.پس داریم:

$2^k * 2 = 2^{k+1}$

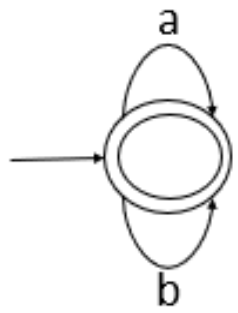
پس یعنی نشان دادیم برای n=k+1 هم نیاز به $2^{k+1}$ استیت داریم.

## سوال دهم:

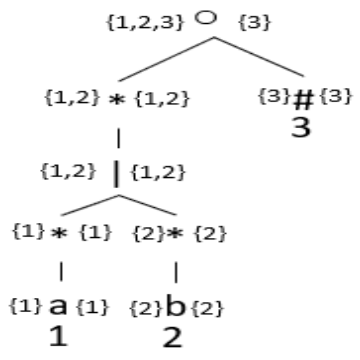
**(a|b)*:**



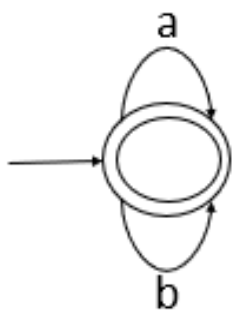

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3} |
| 2 | {1,2,3} |
| 3 | ----- |



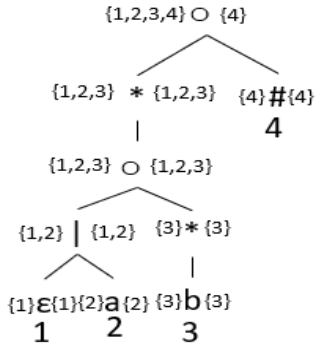

**(a*|b*)*:**



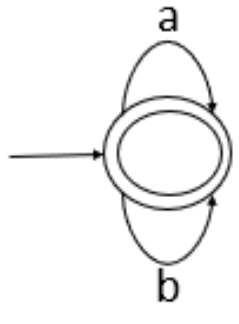

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3} |
| 2 | {1,2,3} |
| 3 | ----- |



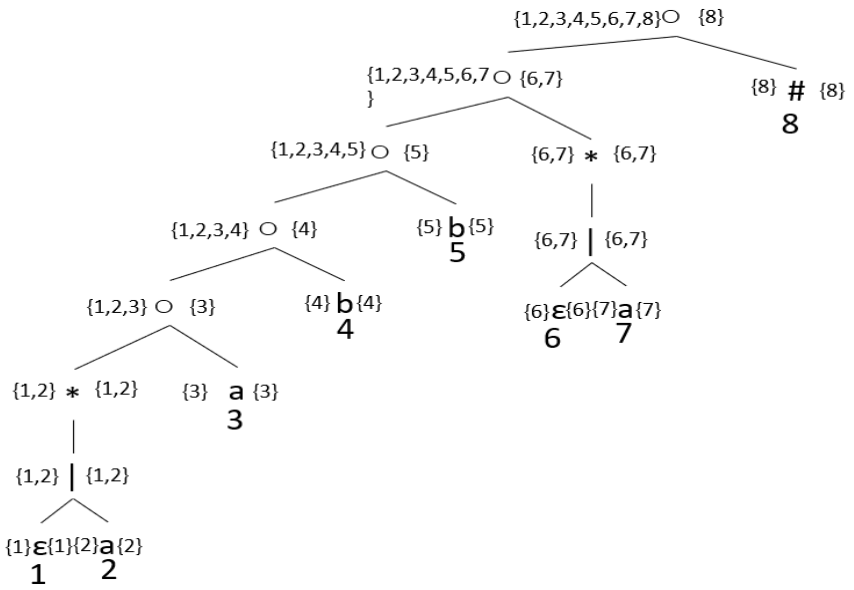

**((ε|a)b*)*** :



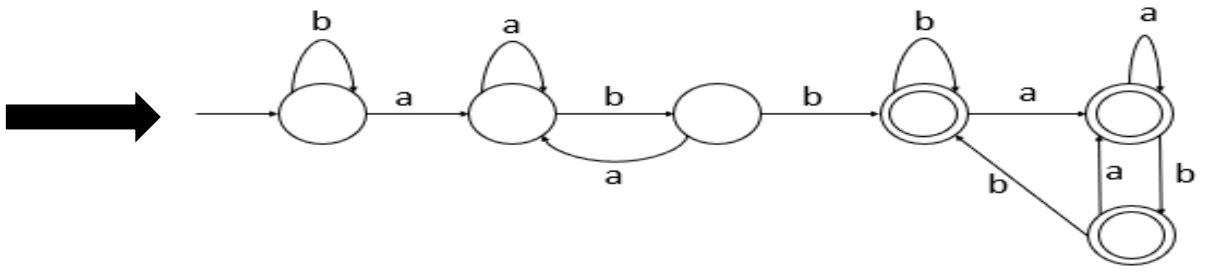

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3,4} |
| 2 | {1,2,3,4} |
| 3 | {1,2,3,4} |
| 4 | ------ |



**(a|b)*abb(a|b)*** :



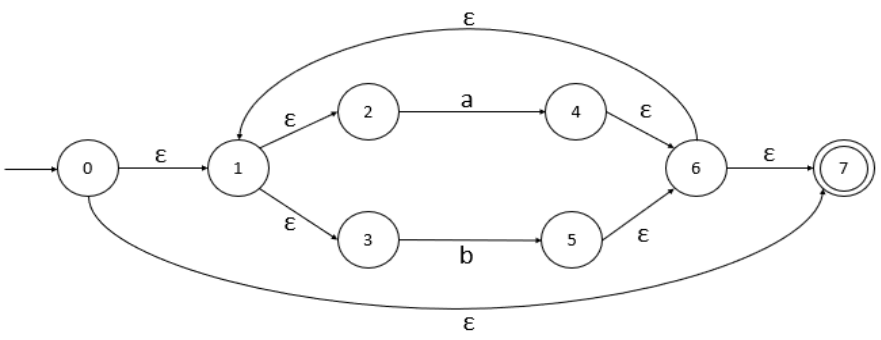

| Node n | Followpos(n) |
|---|---|
| 1 | {1,2,3} |
| 2 | {1,2,3} |
| 3 | {4} |
| 4 | {5} |
| 5 | {6,7,8} |
| 6 | {6,7,8} |
| 7 | {6,7,8} |
| 8 | ------ |




**سوال یازدهم:**

**(a|b)*:**

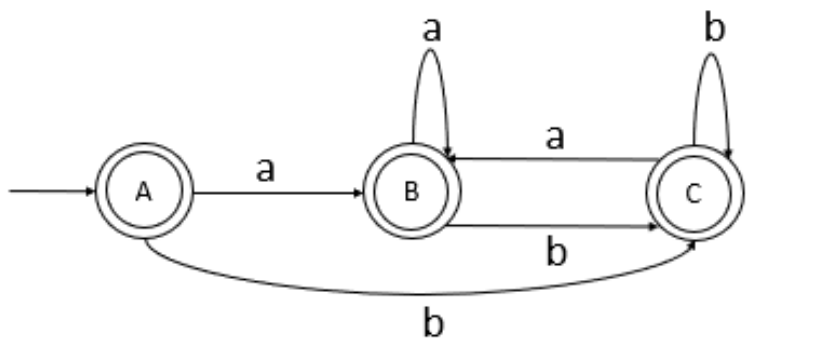
**NFA:**




**Transition Table**

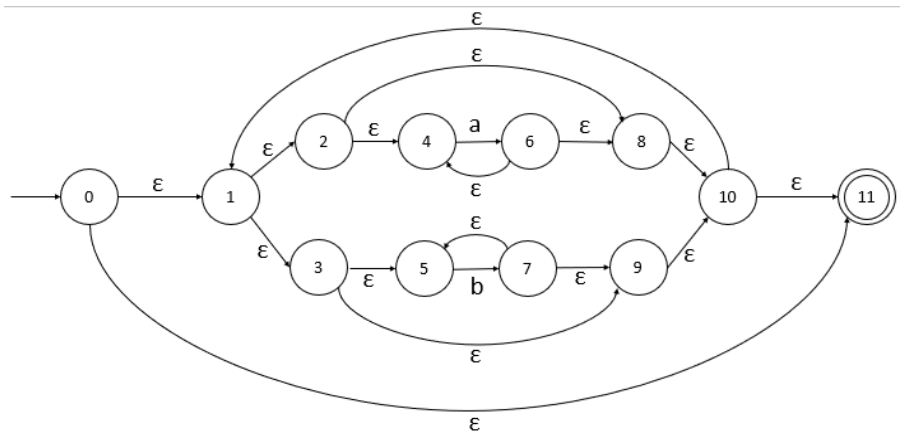
| NFA State | DFA State | a | b |
|---|---|---|---|
| {0,1,2,3,7} | A | B | C |
| {1,2,3,4,6,7} | B | B | C |
| {1,2,3,5,6,7} | C | B | C |

**DFA**




**(a*|b*)***:

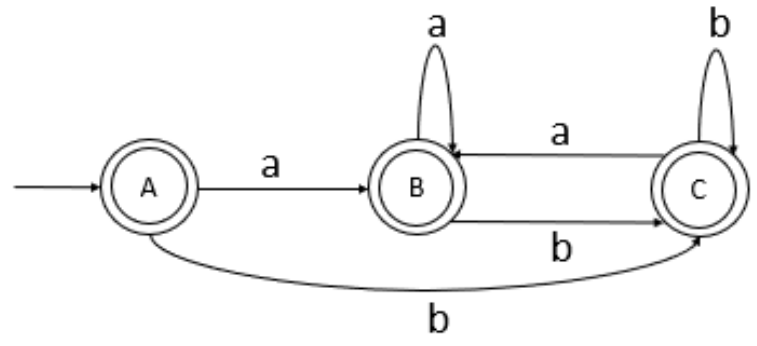
**NFA:**




**Transition Table**

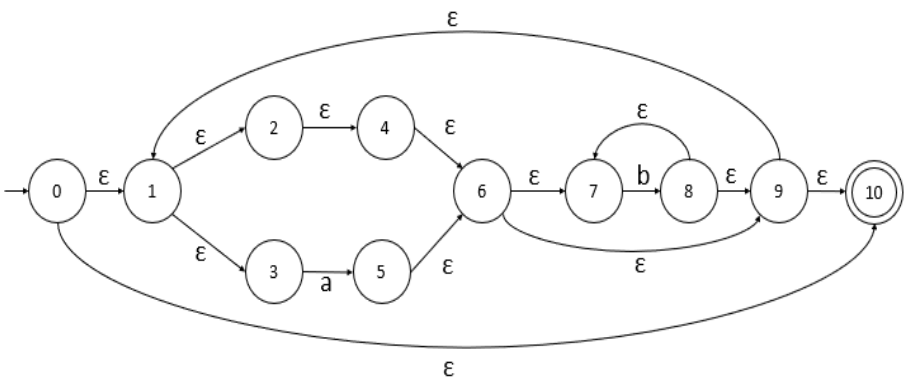
| NFA State | DFA State | a | b |
|---|---|---|---|
| {0,1,2,3,4,5,8,9,10,11} | A | B | C |
| {1,2,3,4,5,6,8,9,10,11} | B | B | C |
| {1,2,3,4,5,7,8,9,10,11} | C | B | C |

**DFA**



**((ε|a)b*)***:

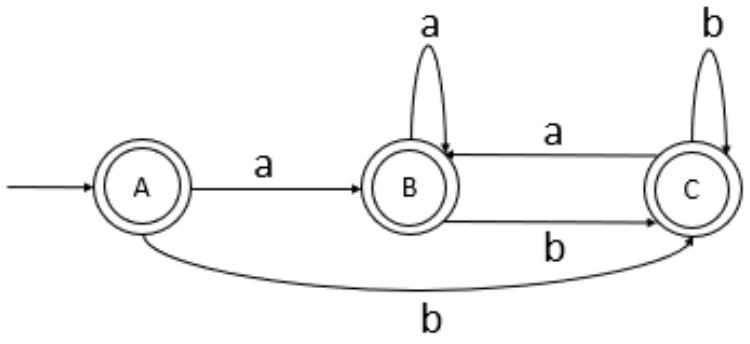
**NFA:**




**Transition Table**

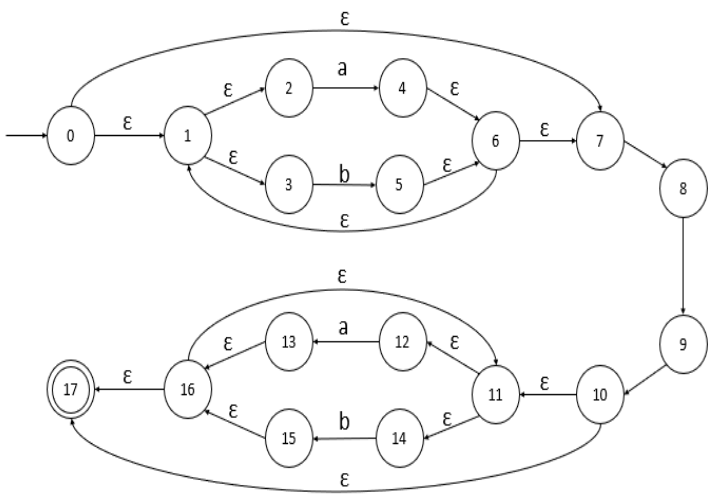
| NFA State | DFA State | a | b |
|---|---|---|---|
| {0,1,2,3,4,6,7,9,10} | A | B | C |
| {1,2,3,4,5,6,7,9,10} | B | B | C |
| {1,2,3,4,6,7,8,9,10} | C | B | C |

**DFA**




**(a|b)*abb(a|b)***:

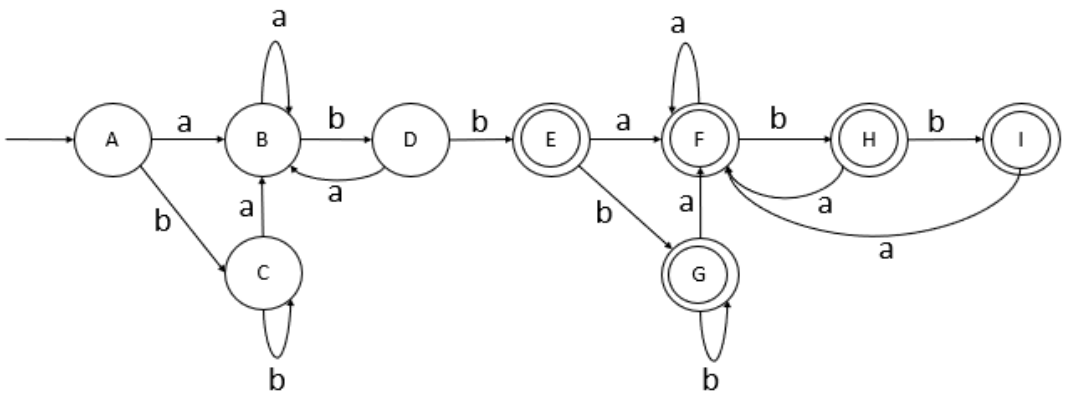
**NFA:**




**Transition Table**

| NFA State | DFA State | a | b |
|---|---|---|---|
| {0,1,2,4,7} | A | B | C |
| {1,2,3,4,6,7,8} | B | B | D |
| {1,2,4,5,6,7} | C | B | C |
| {1,2,4,5,6,7,9} | D | B | E |
| {1,2,4,5,6,7,10,11,12,14,17} | E | F | G |
| {1,2,3,4,6,7,8,11,12,13,14,16,17} | F | F | H |
| {1,2,4,5,6,7,11,12,13,15,16,17} | G | F | G |
| {1,2,4,5,6,7,9,11,12,14,15,16,17} | H | F | I |
| {1,2,4,5,6,7,10,11,12,14,15,16,17} | I | F | G |

**DFA**




## سوال دوازده:

```
main ()
{
char ch= 'A';
int x, y;
x = y = 20;
x ++;
printf("%d% d", x, y);
}
```

**انواع:**

ID - LPAREN - RPAREN - LBRACE - CHAR - ID - ASSIGN - NUM - SEMI - INT - ID - COMMA - ID - SEMI - ID - ASSIGN - ID - ASSIGN - NUM - SEMI - STRING - COMMA - ID - COMMA - ID - ID - INC - SEMI - ID - LPAREN - RPAREN - SEMI - RBRACE – EOF

**تعداد:**

33 + EOF = 34

```
main( )
{
    int *a, b;
    b = 10;
    a = &b;
    printf("%d%d", b, *a);
    b = */* pointer*/b;
}
```

**انواع:**

ID - LPAREN – RPAREN - LBRACE – INT – STAR - ID - COMMA - ID - SEMI - ID - ASSIGN – NUM – SEMI - ID - ASSIGN - AND – ID – SEMI - ID - LPAREN – STRING - COMMA – ID – COMMA – STAR – ID - RPAREN - SEMI - ID - ASSIGN – STAR - ID - SEMI - RBRACE - EOF

**تعداد:**

35 + EOF = 36

```
int strange (int x)
{
    if (x <= 0) return 0;
    if ((x%2) != 0) return x-1;
    return 1+strange(x-1);
}
```

**انواع:**

INT – ID(strange) – LPAREN – INT – ID(x) – RPAREN – LBRACE – IF - LPAREN - ID(x) - LEQ - NUM( 0 ) - RPAREN - RETURN - NUM( 0 ) - SEMI - IF - LPAREN - LPAREN - ID(x) - MOD - NUM( 2 ) – RPAREN - NOTEQ - NUM( 0 ) - RPAREN - RETURN - ID(x) - MINUS - NUM( 1 ) - SEMI - RETURN - NUM( 1 ) - PLUS - ID(strange) - LPAREN - ID(x) - MINUS - NUM( 1 ) - RPAREN - SEMI - RBRACE – EOF

**تعداد:**

43 + EOF = 44