

**سوال ۱:****:Token**

در واقع Token در زبان‌های برنامه نویسی و تجزیه و تحلیل زبان‌ها یک واحد کوچکتر از یک لغت یا یک واژه است.

Token‌ها کوچکترین واحدهای معنادار در یک زبان برنامه نویسی هستند که توسط parser یا سایر اجزای پردازش زبان‌ها شناسایی می‌شوند. هر Token نمایانگر یک مفهوم یا نوع خاصی از اطلاعات است که توسط برنامه قابل استفاده است.

توی هر سطری از جدول نماد تقریباً یک زوجی به صورت `<token-name, attribute-value>` نگهداری می‌شود یعنی یک token توی هر سطرش نگهداری می‌شود که ان token یک اسم دارد و یکسری attribute که این attribute ممکن است یک یا چندتا باشد.

اسامی که token می‌تواند بگیرد به صورت زیر است:

**Identifier:** Strings of letters or digits, starting with a letter

**Integer:** A non-empty string of digits

**Keyword:** "else" or "if" or "for" or ...

**Whitespace:** A non-empty sequence of blanks, newlines, and tabs

مثلا برای متغیر position که توسط کاربر تعریف شده است token به صورت زیر می‌شود:

`<id,1>` که id نام ان توکن است و 1 هم شماره سطر اطلاعات این متغیر در جدول نماد می‌باشد یعنی در سطر اول جدول نماد اطلاعات یا همان attribute مربوط به این متغیر وجود دارد.

**:Pattern**

همان pattern هم ان الگویی است که کامپایلر متوجه می‌شود که این lexeme مربوط به چه است مثلًا متغیر position ---> یک الگویی وجود داشته باشد که متوجه بشود این lexeme مربوط به token باشد Identifier هست.

قسمت اصلی که باید توی فاز Lexical Analysis طراحی بشود همین pattern ها است. همچنین این الگوها معمولاً با استفاده از عبارات منظم تعریف می‌شوند.

**:Lexeme**

به هر لغتی که استخراج می‌شود یک Lexeme گفته می‌شود یا به عبارت دیگر یک واحد جداگانه از متن اشاره دارد که توسط فرآیند تجزیه (parsing) شناسایی می‌شود. مثلاً int خودش می‌شود یک Lexeme

## سوال 2:

:1

کامپایلر کل کد منبع برنامه را در یک مرحله به کد ماشین ترجمه می‌کند، در حالی که مفسر به طور ترتیبی هر خط از کد منبع را ترجمه و اجرا می‌کند.

کامپایلر کل کد منبع رو میگیرد و ان را به زبان ماشین ترجمه میکند و بعد ان را اجرا میکند در حالی که مفسر به صورت خط به خط پیش میرود یعنی جمله به جمله ترجمه و اجر میکند.

مثال: در زبان C++ اگر یک برنامه با استفاده از کامپایلر (مانند g++) کامپایل شود فایل اجرایی مستقلی تولید می شود که می توان آن را بدون نیاز به مفسر بر روی هر سیستمی اجرا کرد اما اگر از مفسر Cling (مانند C++) استفاده شود برنامه به طور ترتیبی ترجمه و اجرا می شود.

:2

مفسر وقتی که داره خط به خط برنامه رو ترجمه و اجرا می کند بهتر می تواند خطاها مربوط به زمان اجرا رو نشان بدهد و ان را دیباگ کند ولی در برنامه های کامپایل شده ممکن است این دیباگ کردن دشوارتر باشد چون کد منبع در زمان اجرا در دسترس نیست

مثال: در برنامه های C++ کامپایل شده اگر در حین اجرا خطایی رخ دهد ممکن است برنامه به دلایل امنیتی یا دیگر محدودیت های سیستمی متوقف شود و اطلاعات خطایی به برنامه نویس نشان داده نشود اما در برنامه های Python تفسیر شده، مفسر ممکن است پیام خطای دقیق تری را نشان دهد که برنامه نویس می تواند از آن برای دیباگ کردن استفاده کند.

:3

در مورد تشخیص و گزارش خطا کامپایلرها خطاها موجود در کد را به طور کامل شناسایی کرده و به برنامه نویس گزارش می دهند اما مفسرها خطاها را به طور ترتیبی در طول اجرا شناسایی کرده و در صورت وقوع آنها را به برنامه نویس گزارش می دهند.

مثال: در C++ کامپایلر ممکن است در هنگام کامپایل پیام خطایی مانند "error: 'x' was not declared in this scope" را نشان دهد. اما در زبان Python، مفسر در حین اجرا می تواند خطایی مانند "NameError: name 'x' is not defined" را نشان دهد.

:4

برنامه های کامپایل شده باید بر روی سخت افزار یا سیستم عامل خاصی که توسط کامپایلر پشتیبانی می شود اجرا می شوند اما برنامه های تفسیر شده می توانند بر روی هر سیستمی که یک مفسر متناسب با زبان برنامه نویسی نصب شده است اجرا شوند.

مثال: یک برنامه کامپایل شده با C++ باید بر روی سیستمی که از ساختار ماشینی مشخصی پشتیبانی می‌کند اجرا شود اما یک برنامه تفسیر شده با Python می‌تواند بر روی هر سیستمی که Python Interpreter نصب شده باشد اجرا شود.

:5

در برنامه‌های کامپایل شده چون کل برنامه قبل از اجرا ترجمه می‌شود پس به حافظه کمتری نیاز دارد نسبت به برنامه‌های تفسیر شده چون در برنامه‌های تفسیر شده مفسر باید در حین اجرای خط باید اطلاعات دیگر وضعیت برنامه را پیگیری کند پس حافظه بیشتری را می‌گیرد.

مثال: یک برنامه C++ کامپایل شده معمولاً حافظه کمتری نیاز دارد تا یک برنامه Python تفسیر شده به علت اینکه باید تمامی کدها قبل از اجرا ترجمه شوند و فقط کد اجرایی در حافظه بارگذاری می‌شود اما یک مفسر Python نیاز دارد که در حین اجرا، خطوط کد را ترجمه کرده و در حافظه نگه دارد.

regex:

←  $\cup^w$

$$(s|S)^+ (e|E)^+ (l|L)^+ (e|E)^+ (c|C)^+ (t|T)^+$$

←  $\cup^r$

a →

$$(\Sigma - \{a, e, i, o, u\})^* a (\Sigma - \{a, e, i, o, u\} | a)^* c (\Sigma - \{a, e, i, o, u\} | e)^*$$
$$i (\Sigma - \{a, e, i, o, u\} | i)^* o (\Sigma - \{a, e, i, o, u\} | o)^* u (\Sigma - \{a, e, i, o, u\} | u)^*$$

b →

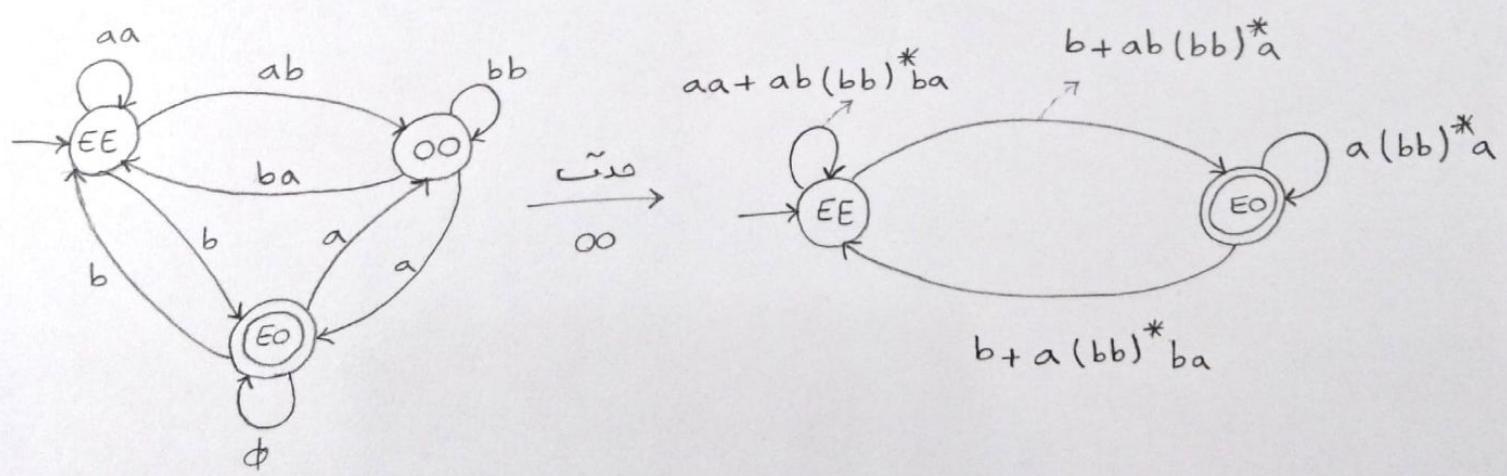
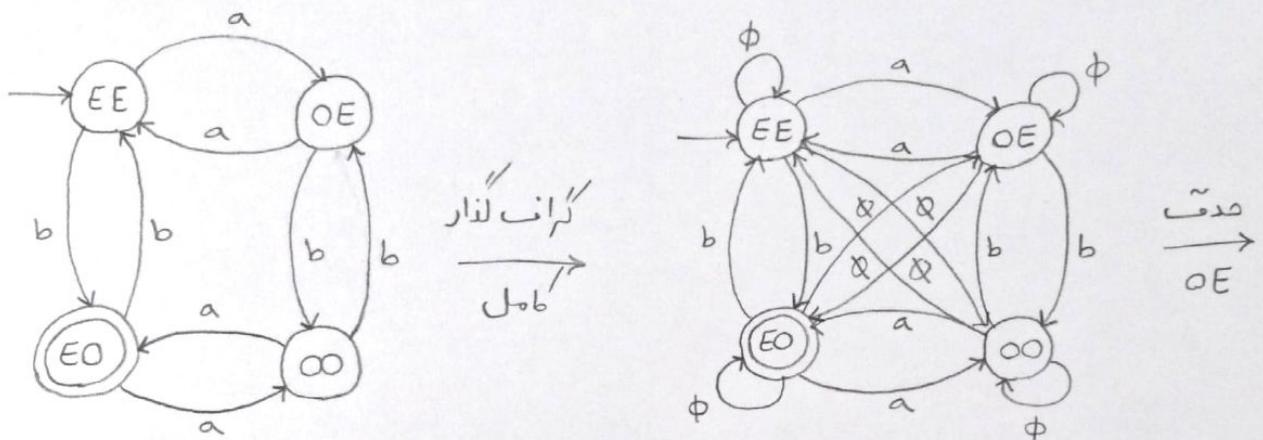
$$a^* b^* c^* d^* e^* \dots x^* y^* z^*$$

c →

: C ← F

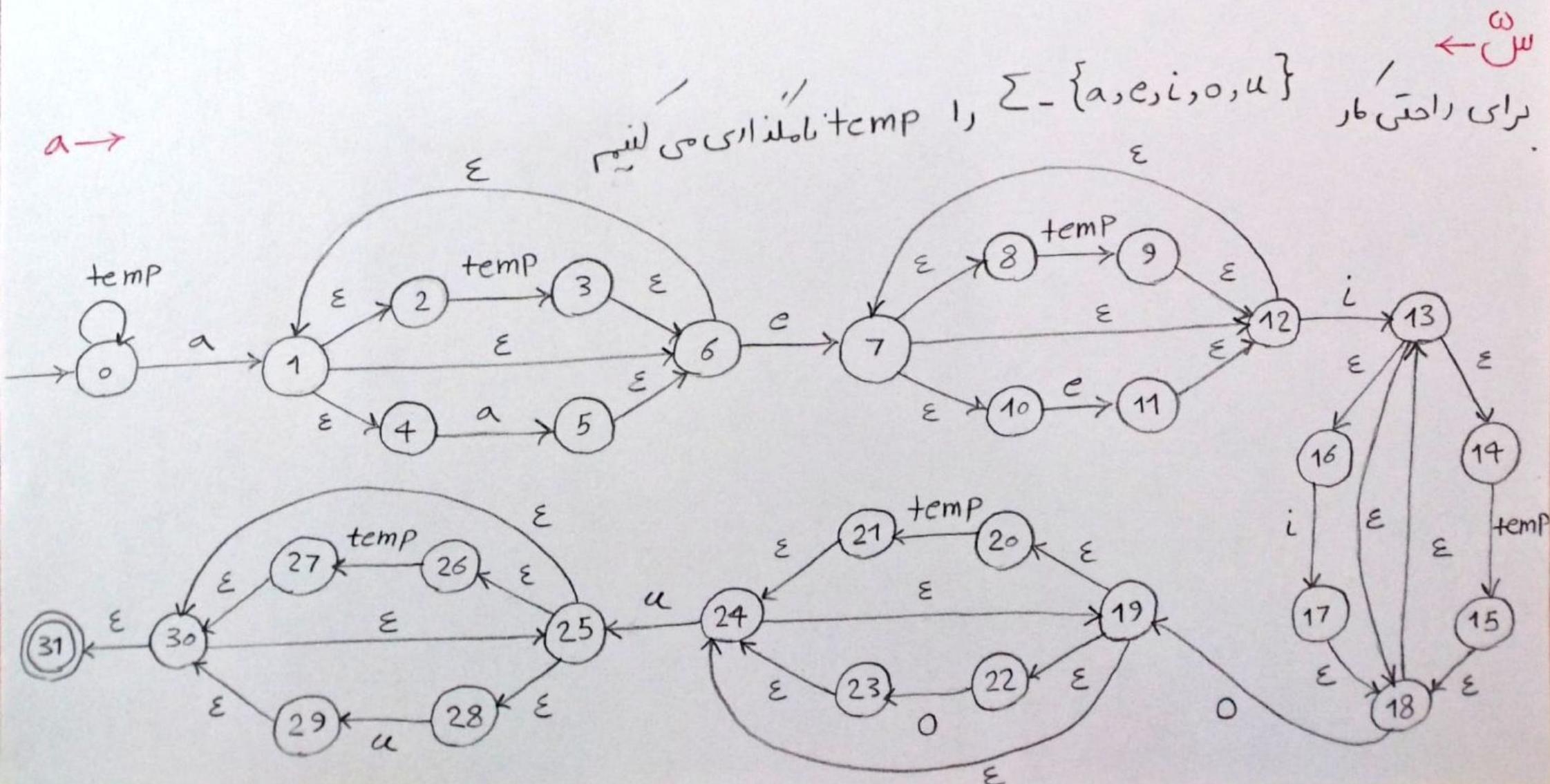
NFA زیر است:

حالی است که در آن تعدادی زوج از a و b حداکثر سده است  
 مفرد از a و تعدادی زوج از b حداکثر سده است  
 مفرد از b و زوج از a و زوج از b و زوج از a و b حداکثر سده است  
 فرد از a و b حداکثر سده است

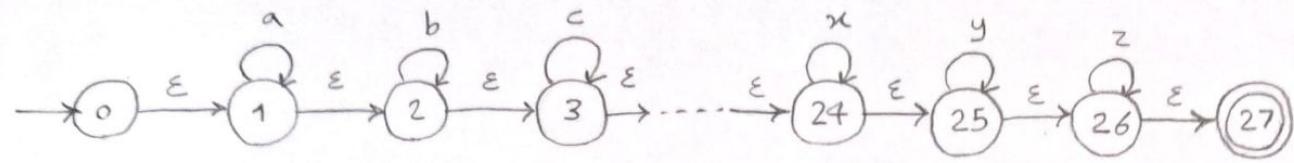


نماینده عبارت مفهومی  $\Rightarrow (aa + ab(bb)^*ba)^*(b + ab(bb)^*a) (a(bb)^*a + (b + a(bb)^*ba)$

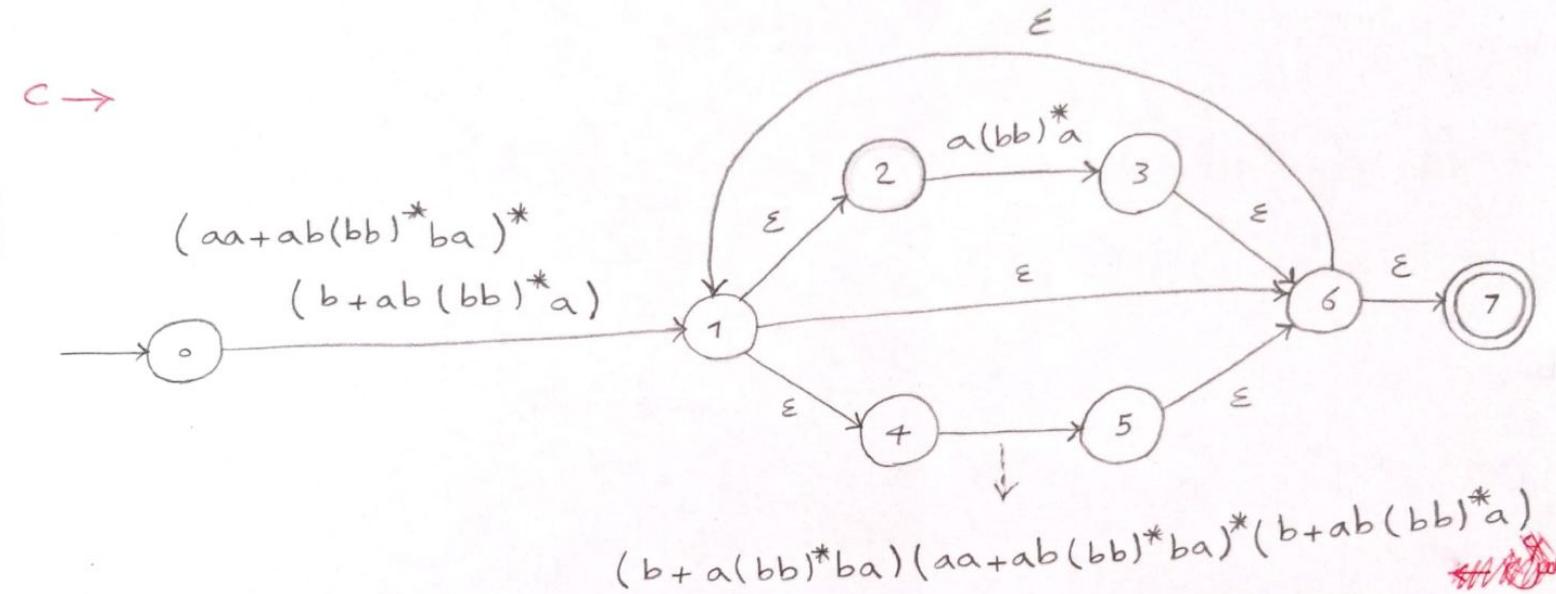
$\Rightarrow (aa + ab(bb)^*ba)^*$   
 $\Rightarrow (b + ab(bb)^*a)^*$



b →

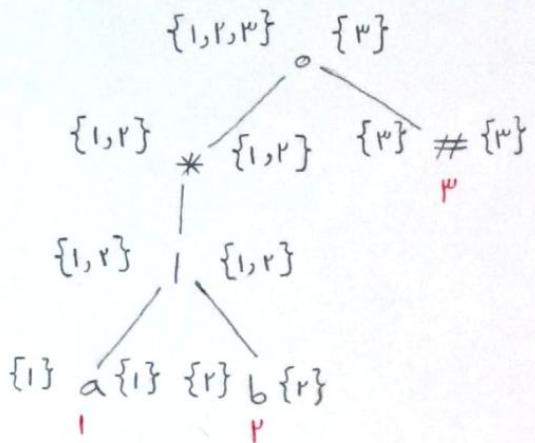


c →

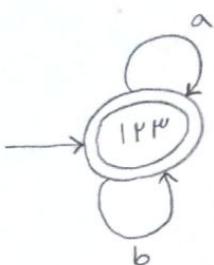


$$(a|b)^* \rightarrow (a|b)^* \#$$

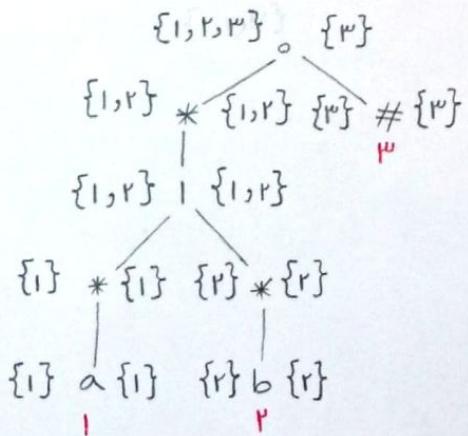
$\leftarrow \text{ج}\rightleftharpoons \text{ج}$



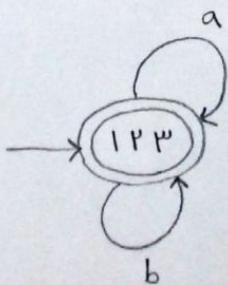
position $n$	followpos( $n$ )
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\emptyset$



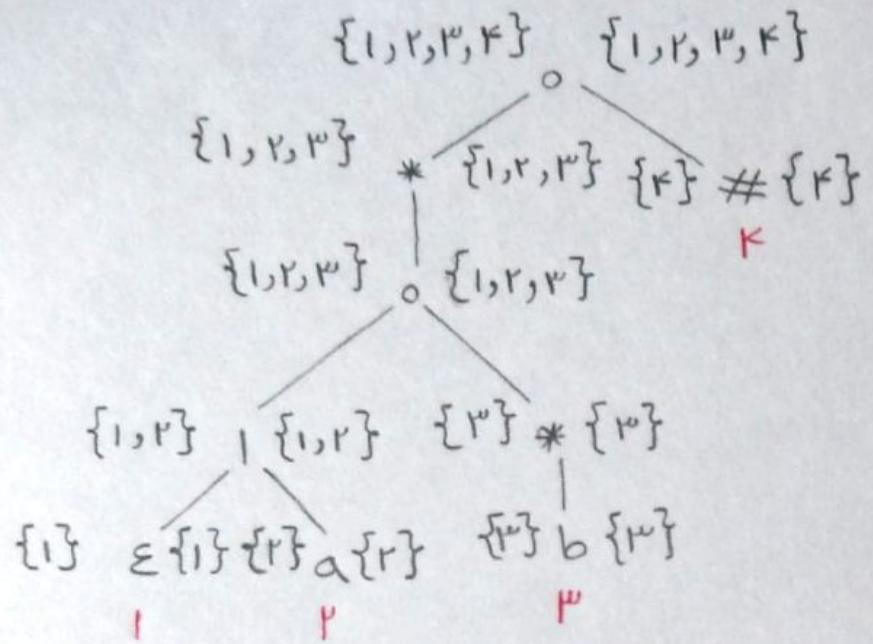
$$(a^*|b^*)^* \rightarrow (a^*|b^*)^* \#$$



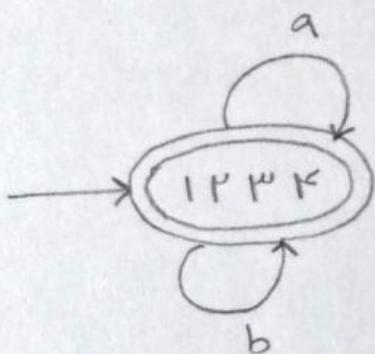
position $n$	followpos( $n$ )
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\emptyset$



$$((\varepsilon | a)b^*)^* \rightarrow ((\varepsilon | a)b^*)^* \#$$



position n	followpos(n)
1	$\{\varepsilon, a, b, \#\}$
b	$\{\varepsilon, a, b, \#\}$
#	$\{\varepsilon, a, b, \#\}$
$\#$	$\emptyset$

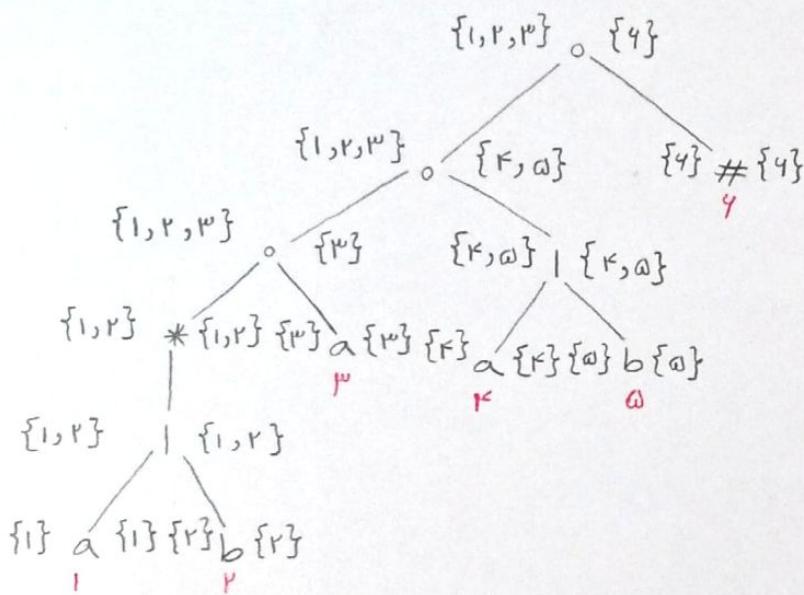


برای این سه عبارت منظم طبق روندی که مشاهده کردیم و اثبات سوال بعدی هر کدام از این عبارت‌ها حداقل 2 به توان n حالت دارند برای عبارت اولی 4 حالت داریم که حداقل ما است و عبارت دومی و سومی هم به همین صورت پس از انجایی که حداقل حالت‌ها برای این سه عبارت اینجا رخ داده است پس تمامی dfa هایی که رسم شده است کمترین تعداد حالت را دارند

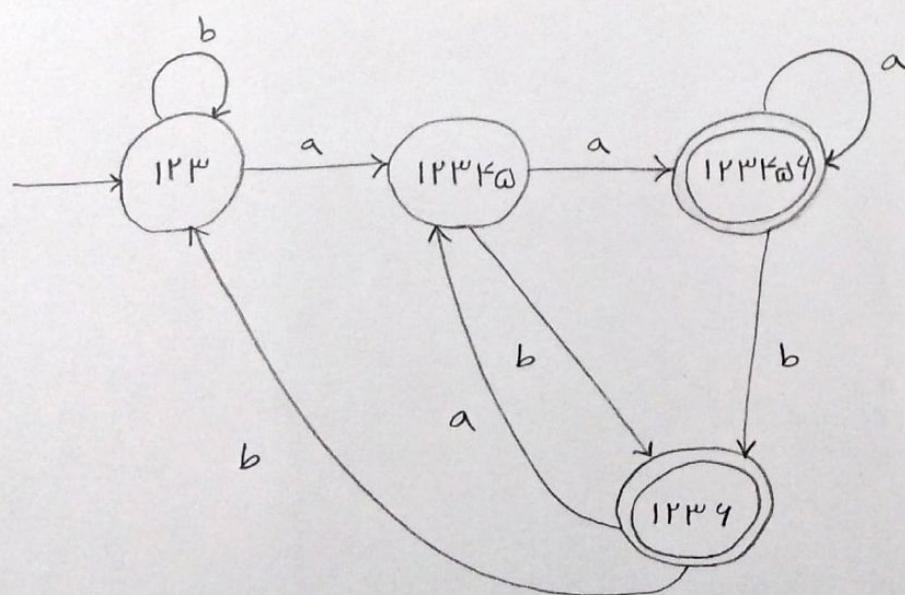
$(\alpha\beta)^*\alpha(\alpha\beta)$

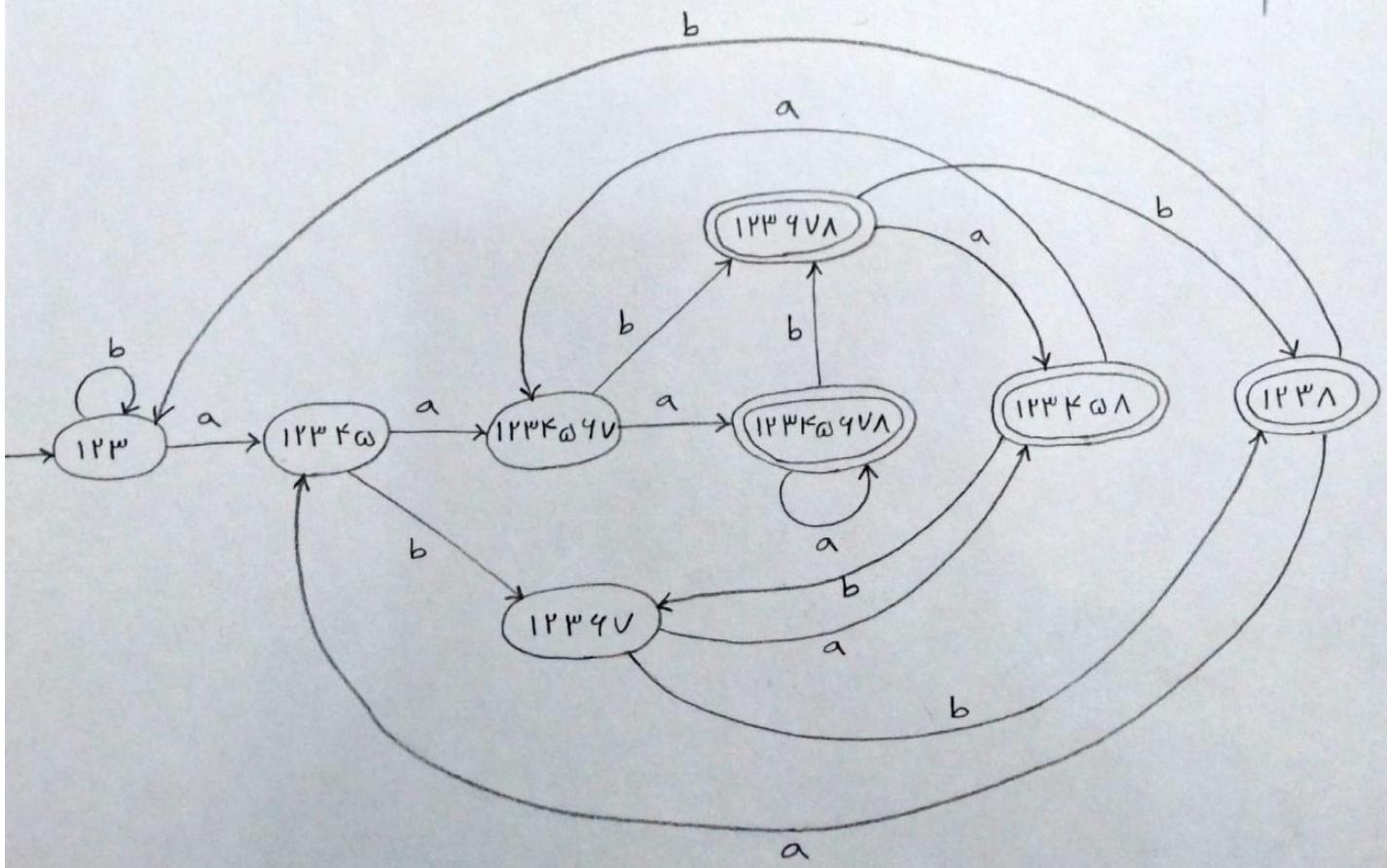
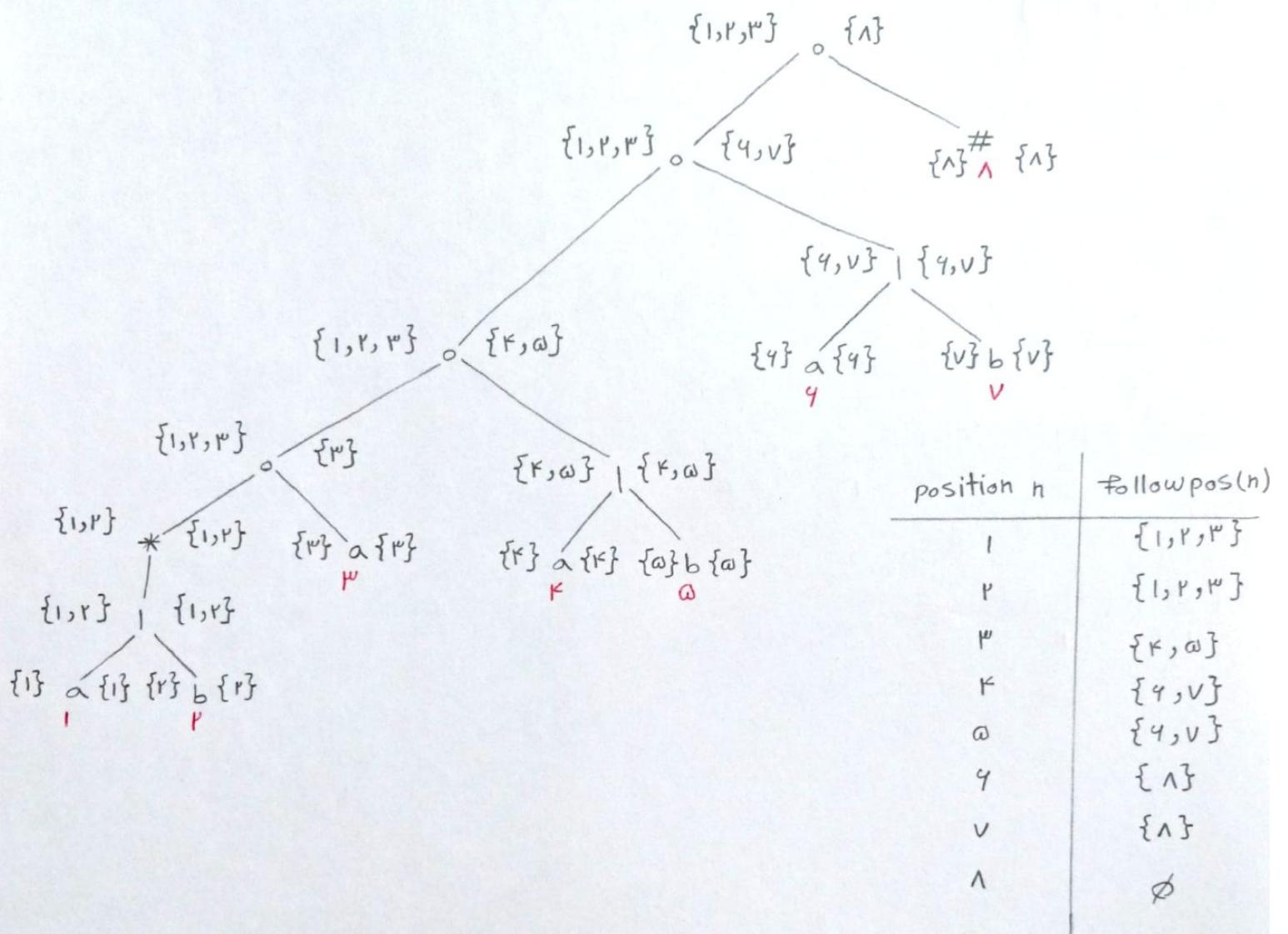
نلتے  $\leftarrow$  برای هر چهار سمت firstpos چیز نوشت و lastpos چیز نوشت را سین

$\hookrightarrow (\alpha\beta)^*\alpha(\alpha\beta) \#$

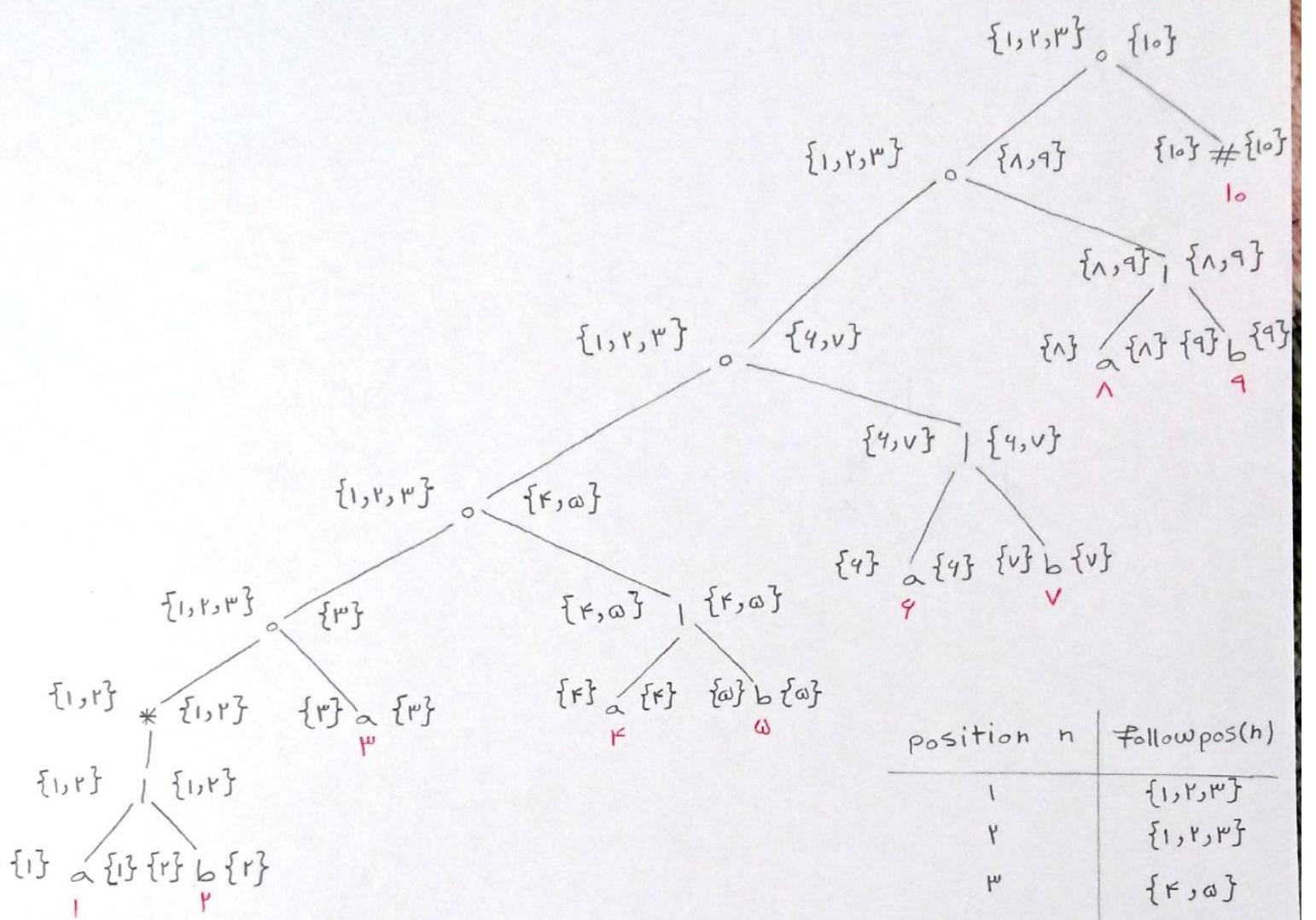


position $n$	followpos( $n$ )
1	{1, 2, 3}
2	{1, 2, 3}
3	{F, \alpha}
4	{4}
5	{4}
6	$\emptyset$

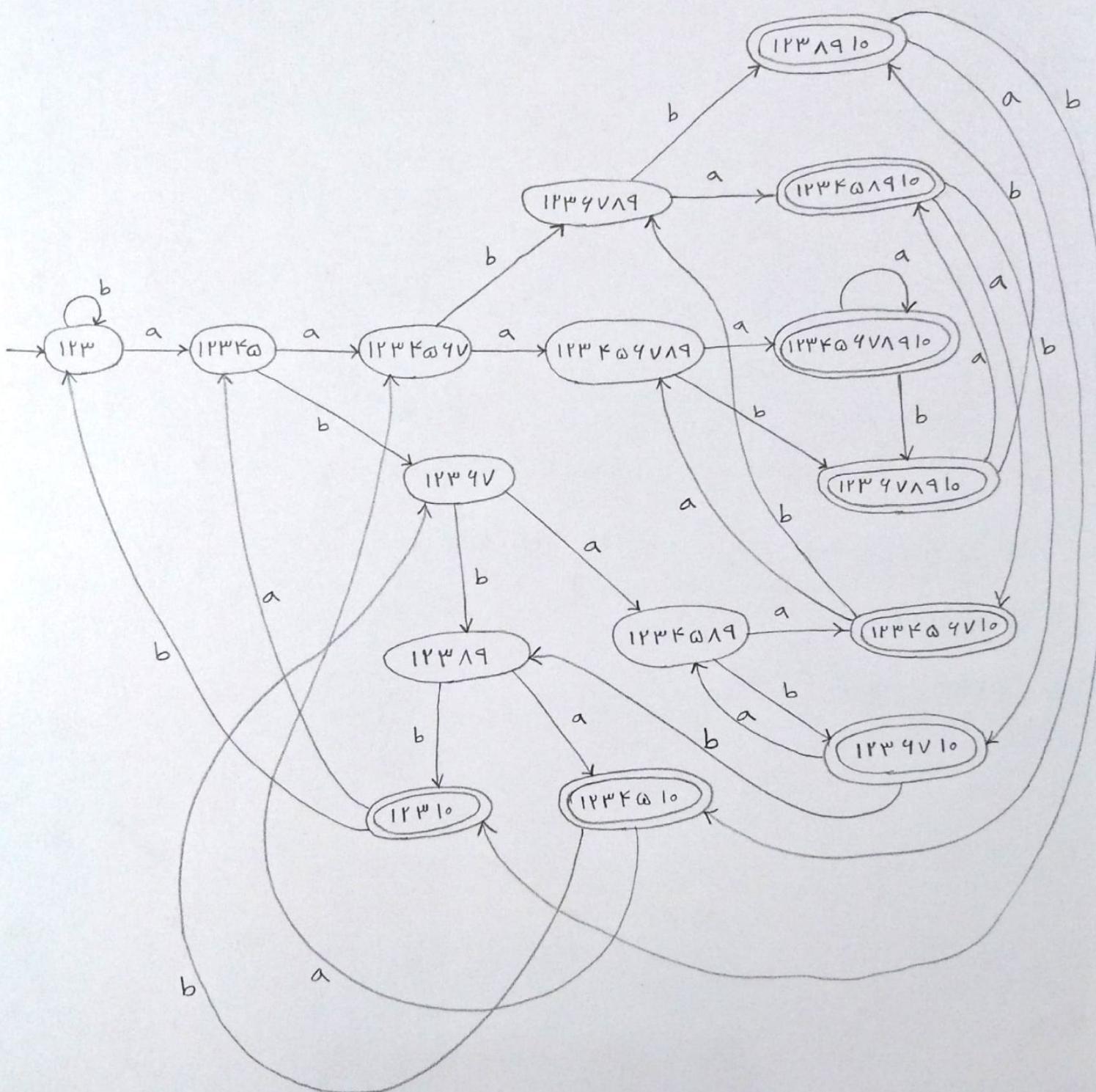


$$(a|b)^* a (a|b) (a|b) \rightarrow (a|b)^* a (a|b) (a|b) \#$$


$(a|b)^* a (a|b) (a|b) (a|b) \rightarrow (a|b)^* a (a|b) (a|b) (a|b) \#$



position n	followpos(n)
1	{1, 2, 3}
2	{1, 2, 3}
3	{F, o}
4	{4, v}
5	{4, v}
6	{1, q}
7	{1, q}
8	{10}
9	{10}
10	$\emptyset$



تعداد این مقصوم است لجه نهادن از این در عبارات  
سیدالحسن ← بله . رویی که مساهده می سود بضریت زیر است :

$$(alb) + 1 \leq \text{تعداد اسیست ها}$$

۹)  $\left( \alpha \mid b \right)^* a \left( \alpha \mid b \right)^{n-1}$  از مatic استر این موضع را ایجاب می کنیم:

حکم باید  $\left( \alpha \mid b \right)^* a \left( \alpha \mid b \right)^n$  داریم:  
ل برای این تعداد حالت ممکن  $2^n$  سرد چون یا هاست  
band یا

فرض اسقرا  $\rightarrow$  فرض می کنیم  $n = k$  است  $\rightarrow$  عبارت  $\left( \alpha \mid b \right)^k a \left( \alpha \mid b \right)^{k-1}$  حالت دارد

حکم اسقرا  $\rightarrow$  محو این ایجاب می کنیم این موضع برای  $n = k+1$  برقرار است:

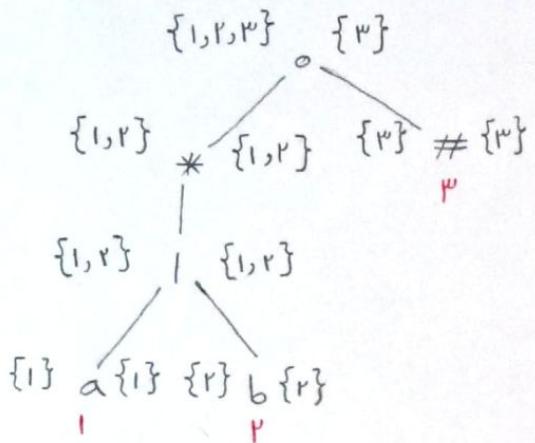
$\left( \alpha \mid b \right)^* a \left( \alpha \mid b \right)^k \rightarrow \underbrace{\left( \alpha \mid b \right)^* a \left( \alpha \mid b \right)^{k-1}}_{\text{ل حق فرض اسقرا حداقل } 2^k \text{ حالت دارد}} \underbrace{\left( \alpha \mid b \right)}_{\substack{\text{یا میتواند باشد یا طبع} \\ \text{2 حالت دارد}}}$

$$2^k \times 2 = 2^{k+1} = 2^n \quad \text{تعداد حالت} =$$

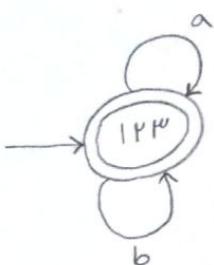
س تعداد حالت ممکن برای عبارت  $\left( \alpha \mid b \right)^* a \left( \alpha \mid b \right)^{n-1}$  است.

$$(a|b)^* \rightarrow (a|b)^* \#$$

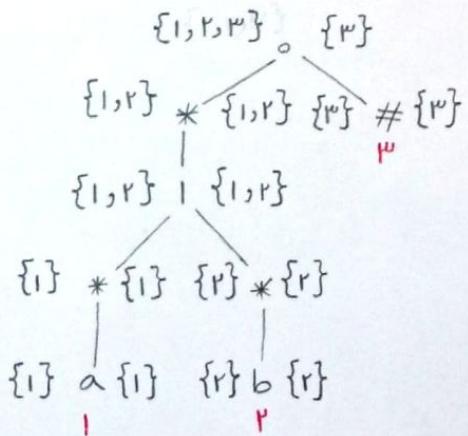
$\leftarrow \text{ج}\rightleftharpoons \text{ج}$



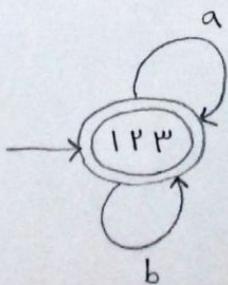
position $n$	followpos( $n$ )
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\emptyset$



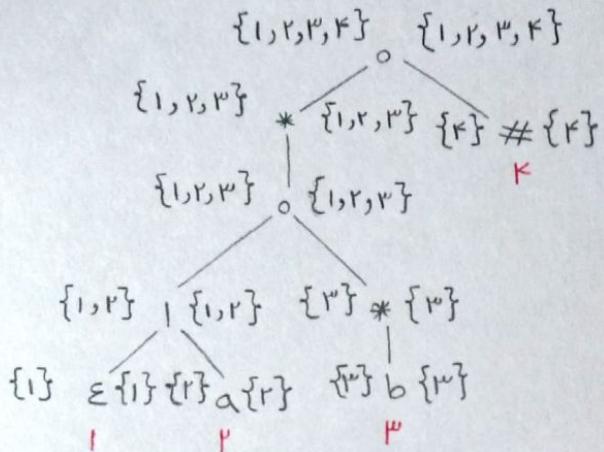
$$(a^*|b^*)^* \rightarrow (a^*|b^*)^* \#$$



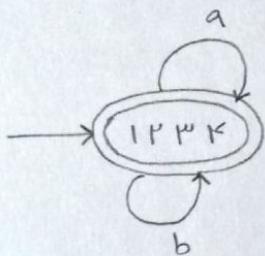
position $n$	followpos( $n$ )
1	$\{1, 2, 3\}$
2	$\{1, 2, 3\}$
3	$\emptyset$



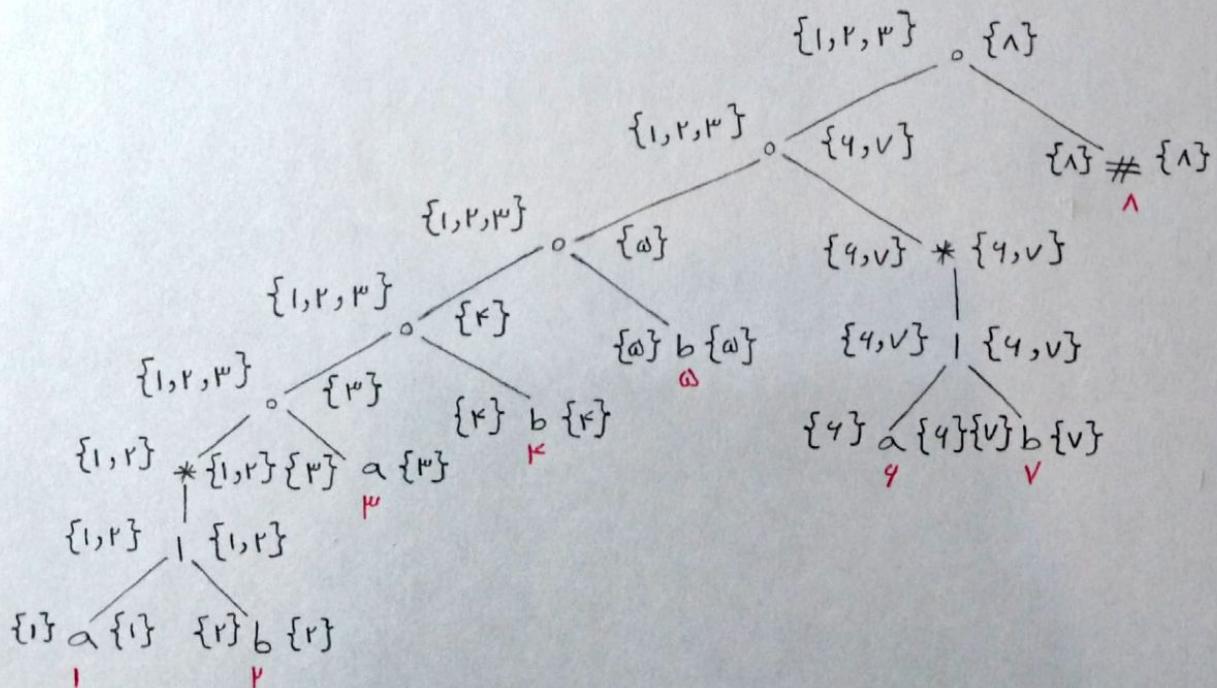
$$((\varepsilon | a)b^*)^* \rightarrow ((\varepsilon | a)b^*)^* \#$$

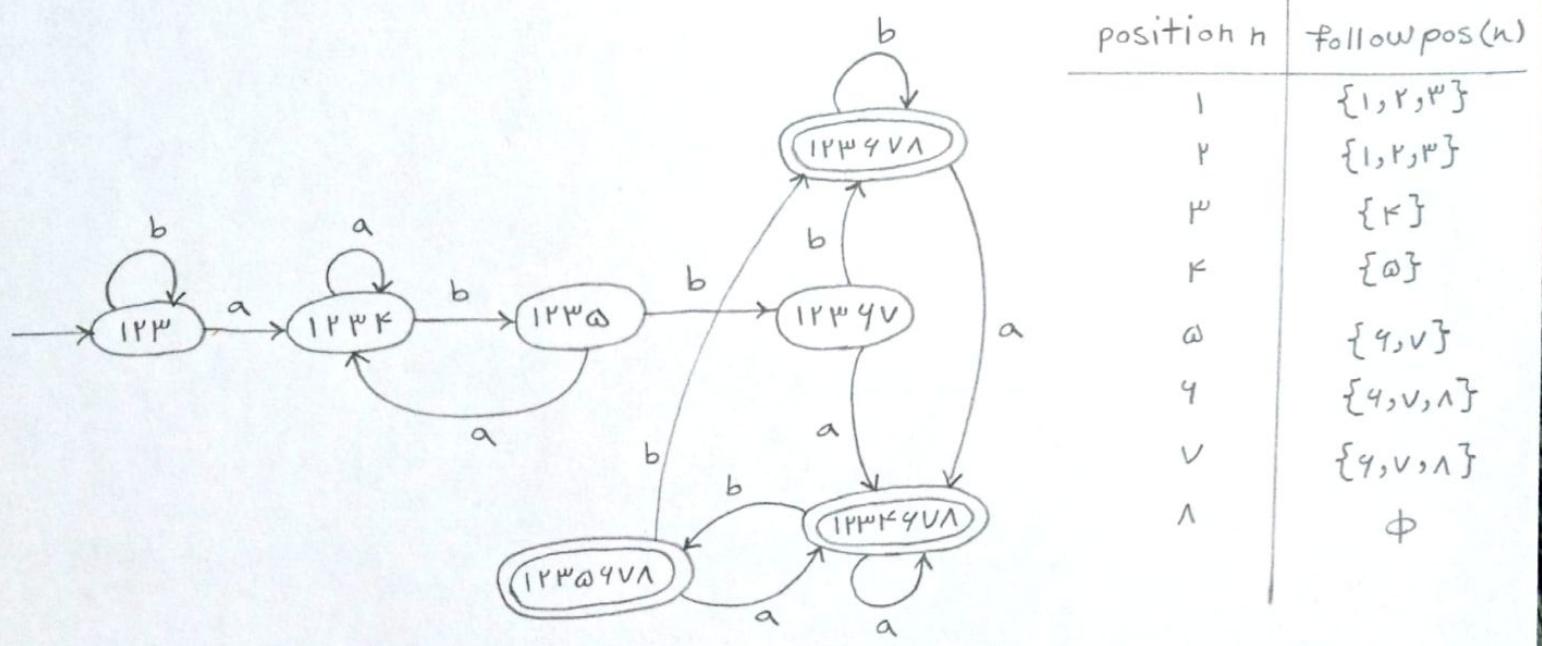


position n	followpos(n)
1	$\{1, 2, 3, 4\}$
2	$\{1, 2, 3, 4\}$
3	$\{1, 2, 3, 4\}$
4	$\emptyset$



$$(a|b)^*abb(a|b)^* \rightarrow (a|b)^*abb(a|b)^*\#$$

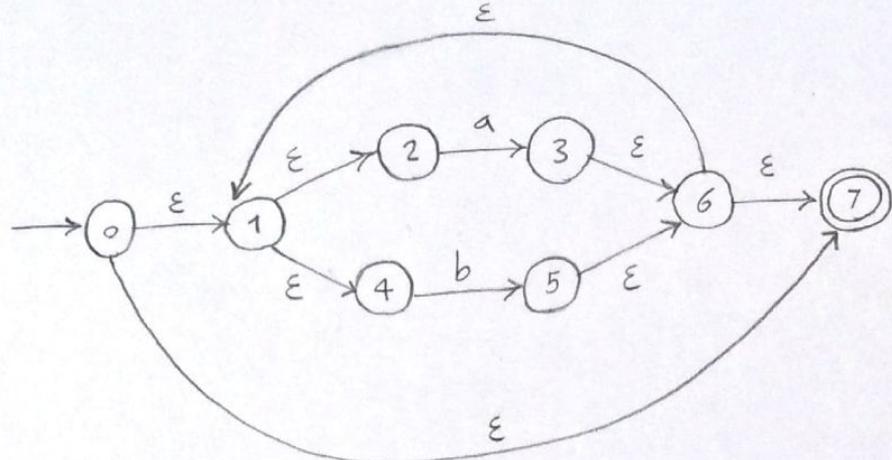




$(alb)^*$  →

← C w

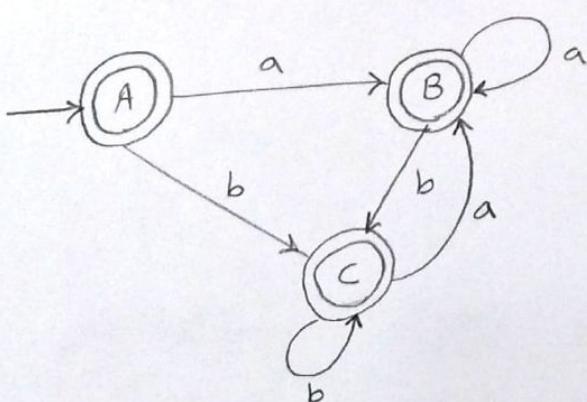
NFA →



Transition tables:

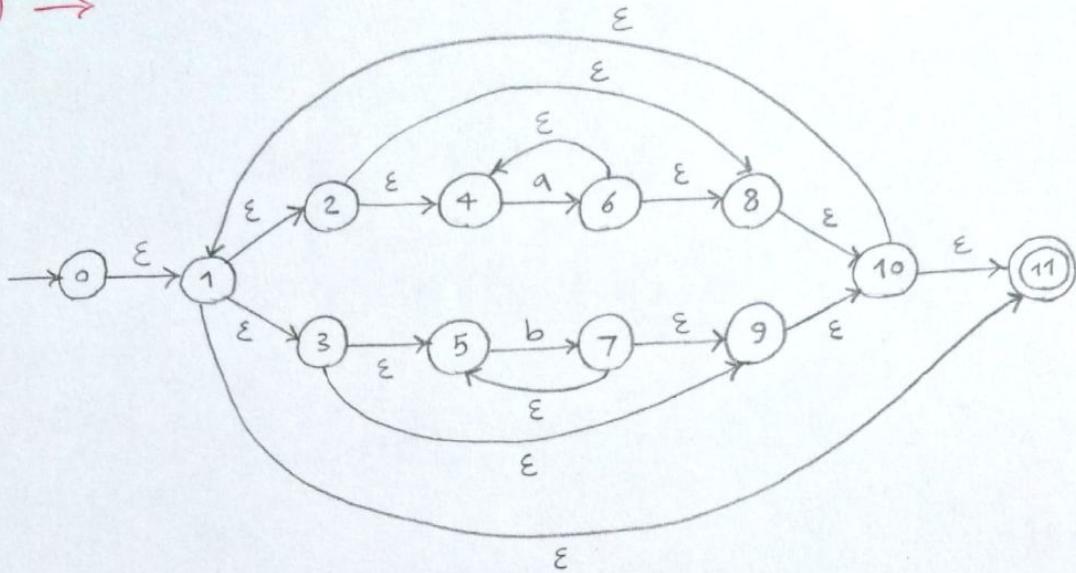
NFA state	DFA state	a	b
$\{0, 1, \varphi, \text{F}, \text{V}\}$	A	B	C
$\{1, \varphi, \text{F}, \text{F}, \text{V}, \text{V}\}$	B	B	C
$\{1, \varphi, \text{F}, \omega, \text{V}, \text{V}\}$	C	B	C

DFA:



$(a^* | b^*)^* \rightarrow$

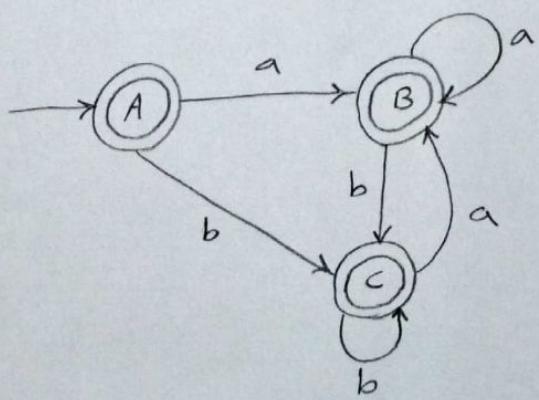
NFA  $\rightarrow$



Transition Table:

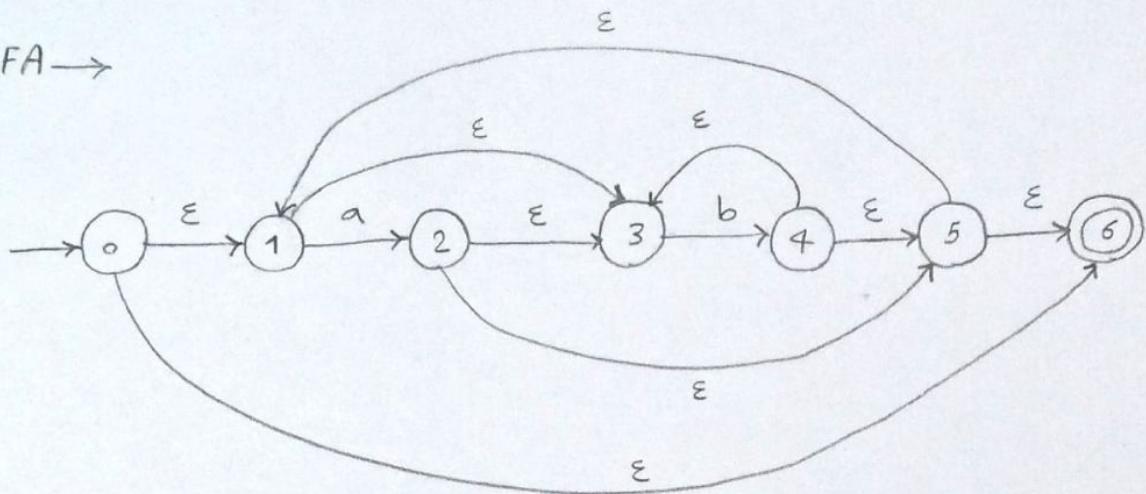
NFA state	DFA state	a	b
$\{0, 1, 2, 4, 6, 8, 10, 11\}$	A	B	C
$\{0, 1, 2, 3, 5, 7, 9, 10, 11\}$	B	B	C
$\{0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11\}$	C	B	C

DFA:



$((\epsilon|a)b^*)^*$

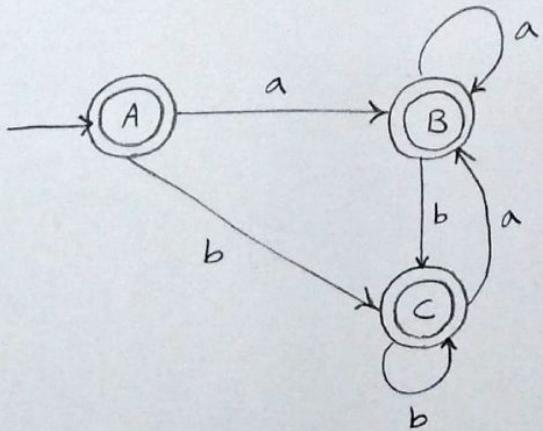
NFA →



Transition table:

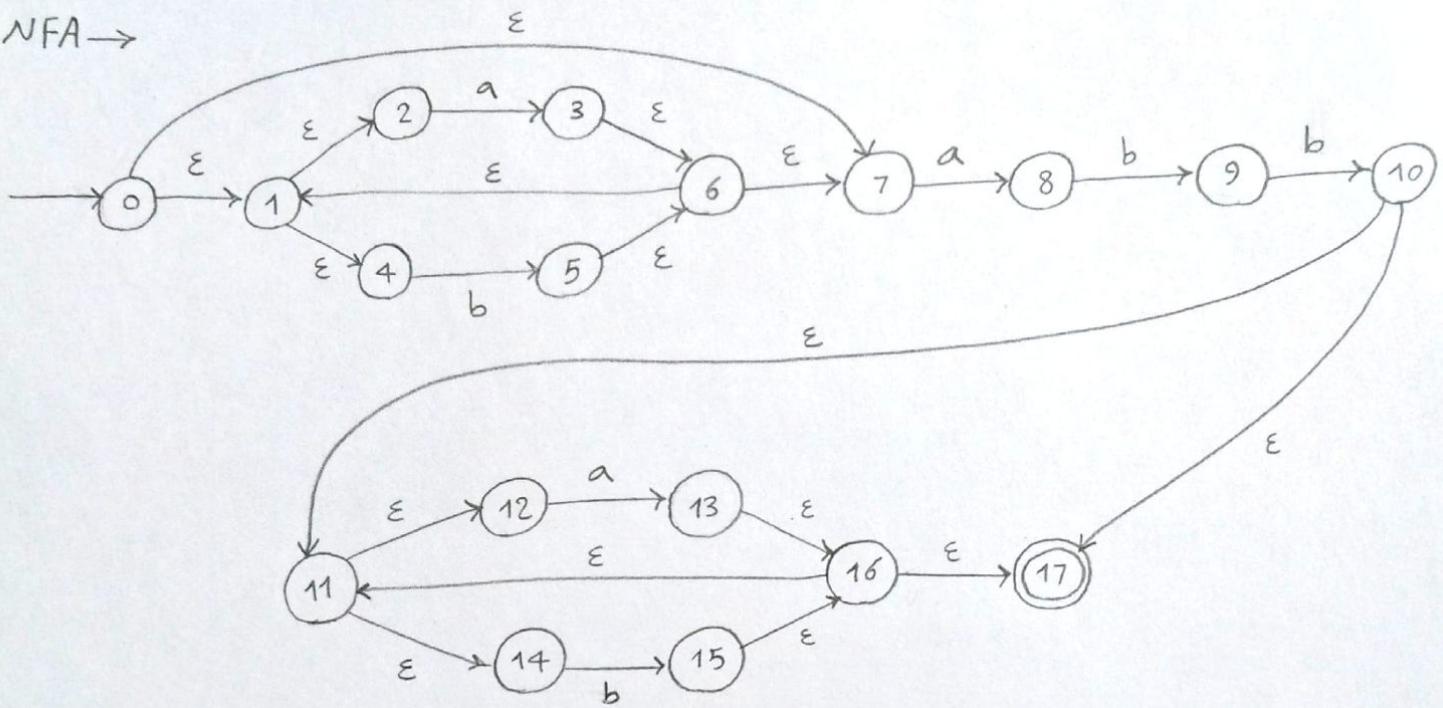
NFA state	DFA state	a	b
{0, 1, 2, 4}	A	B	C
{1, 2, 3, 5, 6}	B	B	C
{1, 2, 3, 6}	C	B	C

DFA:



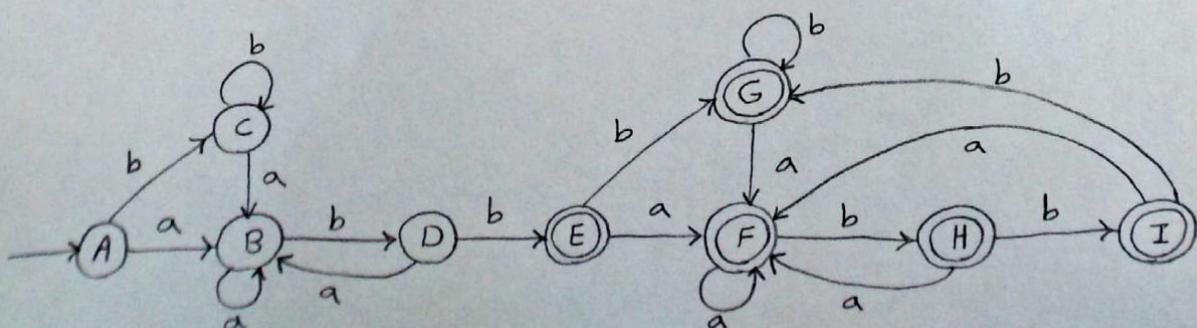
$(alb)^*abb(alb)^* \rightarrow$

NFA  $\rightarrow$



Transition table:

NFA state	DFA state	a	b
$\{0, 1, 2, 4, 5\}$	A	B	C
$\{1, 2, 3, 4, 5, 6, 7, 8\}$	B	B	D
$\{1, 2, 4, 5, 6, 7, 8, 9\}$	C	B	C
$\{1, 2, 4, 5, 6, 7, 8, 9, 10\}$	D	B	E
$\{1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$	E	F	G
$\{1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$	F	F	H
$\{1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17\}$	G	F	G
$\{1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$	H	F	I
$\{1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19\}$	I	F	G



## سؤال 12:

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

أنواع توكن:

1	ID(main)	13	ID(y)	25	ID(sprintf)
2	LPAREN	14	SEMI	26	LPAREN
3	RPAREN	15	ID(x)	27	STRING(%d%d)
4	LBRACE	16	ASSIGN	28	COMMA
5	CHAR	17	ID(y)	29	ID(x)
6	ID(ch)	18	ASSIGN	30	COMMA
7	ASSIGN	19	NUM(20)	31	ID(y)
8	CHARACTER(A)	20	SEMI	32	RPAREN
9	SEMI	21	ID(x)	33	SEMI
10	INT	22	PLUS	34	RBRACE
11	ID(x)	23	PLUS	35	EOF
12	COMMA	24	SEMI	36	

تعداد توكن: 35 تا

```

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

```

أنواع توكن:

1	INT	12	NUM(0)	23	RPAREN	34	PLUS
2	ID(strange)	13	RPAREN	24	NOTEQ	35	ID(strange)
3	LPAREN	14	RETURN	25	NUM(0)	36	LPAREN
4	INT	15	NUM(0)	26	RPAREN	37	ID(x)
5	ID(x)	16	SEMI	27	RETURN	38	MINUS
6	RPAREN	17	IF	28	ID(x)	39	NUM(1)
7	LBRACE	18	LPAREN	29	MINUS	40	RPAREN
8	IF	19	LPAREN	30	NUM(1)	41	SEMI
9	LPAREN	20	ID(x)	31	SEMI	42	RBRACE
10	ID(x)	21	MOD	32	RETURN	43	EOF
11	LEQ	22	NUM(2)	33	NUM(1)	44	

تعداد توكن: 43 تا

```

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

```

أنواع توكن:

1	ID(main)	13	NUM(10)	25	COMMA
2	LPAREN	14	SEMI	26	STAR
3	RPAREN	15	ID(a)	27	ID(a)
4	LBRACE	16	ASSIGN	28	RPAREN
5	INT	17	AND	29	SEMI
6	STAR	18	ID(b)	30	ID(b)
7	ID(a)	19	SEMI	31	ASSIGN
8	COMMA	20	ID(sprintf)	32	STAR
9	ID(b)	21	LPAREN	33	ID(b)
10	SEMI	22	STRING(%d%d)	34	SEMI
11	ID(b)	23	COMMA	35	RBRACE
12	ASSIGN	24	ID(b)	36	EOF

تعداد توكن: 36 تا