Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

Conversion of an NFA to a DFA

- **The subset construction**

```
initially, ϵ-closure(s0) is the only state in Dstates, and it is unmarked;
while ( there is an unmarked state T in Dstates ) {
        mark T;
        for ( each input symbol a ) {
                U = ϵ-closure(move(T, a));
                if ( U is not in Dstates )
                        add U as an unmarked state to Dstates;
                Dtran[T, a] = U;
        }
}
```

-

توی DFA ما می خوایم عدم قطعیت هایی که توی NFA داشتیم رو از بین ببریم:

1- اپسیلون هست --> برای از بین بردن اپسیلون از اپسیلون closure استفاده می کنن که برای هر استیتی که توی NFA داریم می تونیم یک اپسیلون closure حساب بکنیم که این اپسیلون closure میشه اینکه از اون استیت با اپسیلون می تونیم به چه استیت های دیگه حرکت کنیم --> این برای این است که بتونیم اون اپسیلونه رو از بین ببریم

2- چند حالت بودن یا حالت نداشتن

تابع move ینی توی یک استیت یا یک مجموعه ای از استیت ها هستیم به ازای یک حرف خاصی به چه استیتی حرکت میکنیم

روش کلی:

می خوایم از استیت شروع که شروع می کنیم به کار بیایم اولا ببینیم با اپسیلون چه حرکت هایی میشه داشت و اون استیت هایی که با اپسیلون میشه بهش رفت رو میایم کوچکترش میکنیم ینی از بین می بریم اپسیلون رو

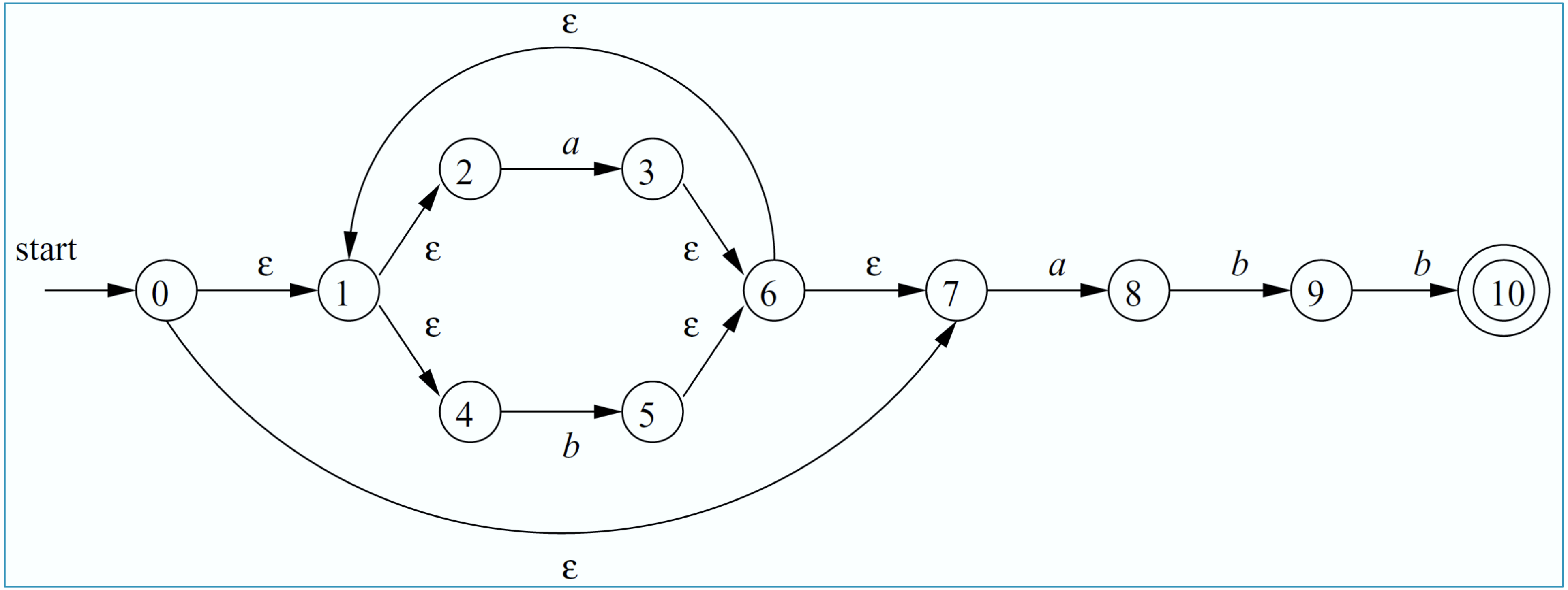
Conversion of an NFA to a DFA

- **Computing ϵ-closure(T)**

push all states of T onto *stack*;
initialize ϵ-*closure*(T) to T;
while (*stack* is not empty) {
    pop t, the top element, off *stack*;
    for (each state u with an edge from t to u labeled ϵ)
        if (u is not in ϵ-*closure*(T)) {
            add u to ϵ-*closure*(T);
            push u onto *stack*;
        }
}

Conversion of an NFA to a DFA

- **Example**



اولین کار برای ساخت DFA از روی NFA این است که:
اپسیلون closure هاشو در بیاریم --> اپسیلون closure ینی این که دقیقا از این استیت با کاراکتر اپسیلون به چه استیت هایی میشه رفت که اینجا برای استیت صفر این میشه 1 و 7

استیت های یک DFA زیرمجموعه ای از استیت های NFA ای است که از روش ساخته میشه. پس DFA حاصل حداکثر

$۲^{۸}$ وضعیت میتونه داشته باشه.

Conversion of an NFA to a DFA

- **Example**
  - The start state A of the equivalent DFA is ϵ-closure(0), or A = {0, 1, 2, 4, 7}

$$Dtran[A, a] = \epsilon\text{-}closure\big(move(A, a)\big) = \epsilon\text{-}closure(\{3, 8\}) = \{1, 2, 3, 4, 6, 7, 8\}$$

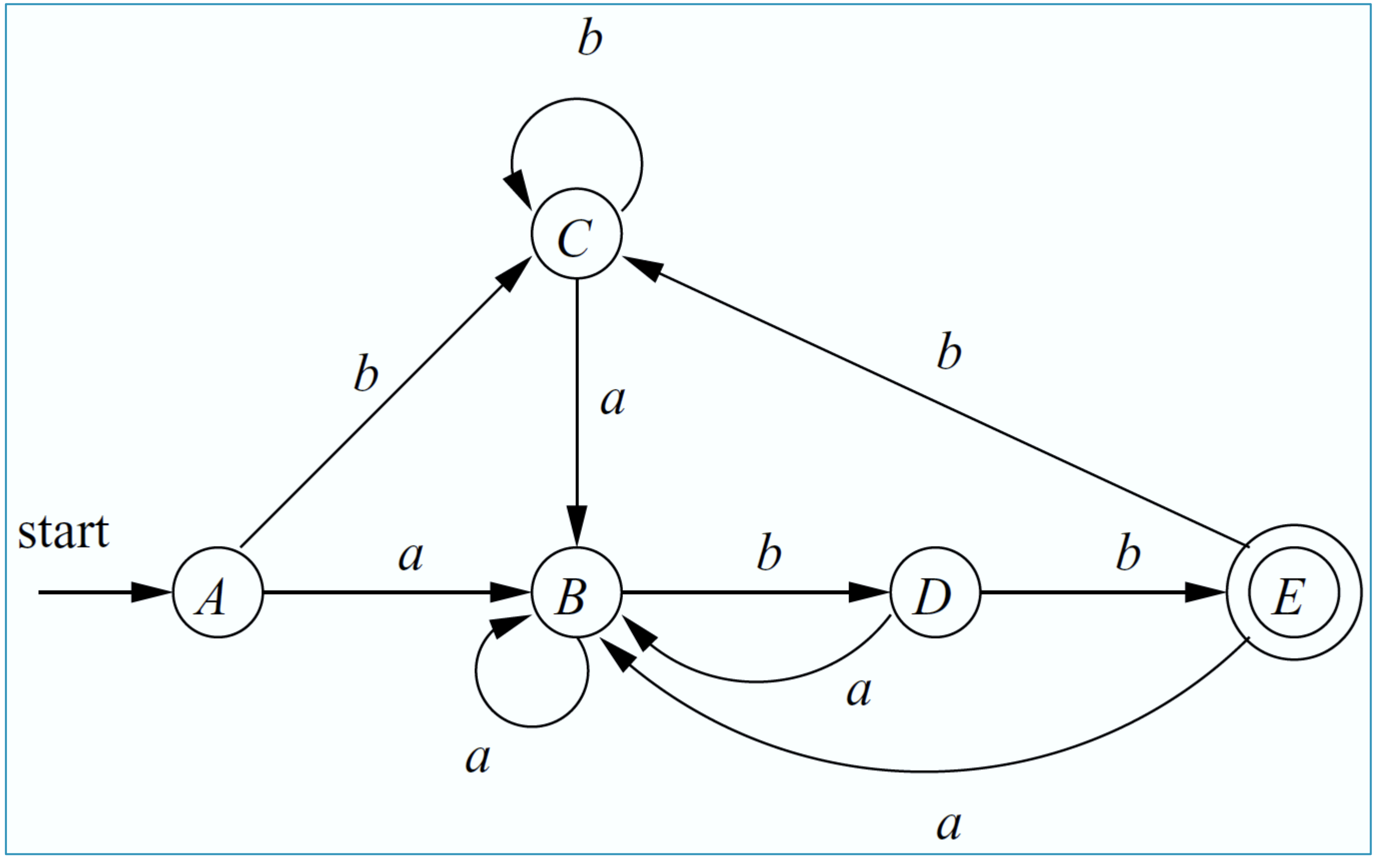
$$Dtran[A, b] = \epsilon\text{-}closure(\{5\}) = \{1, 2, 4, 5, 6, 7\}$$

move که از مجموعه A داریم که خودش 5 تا استیت است با حرف a این رو به دست بیاریم و بعد اون مجموعه که به دست میاد هم باز نگاه کنیم که اگر با اپسیلون حرکتی به استیت های دیگه داره اونو به دست بیاریم

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
$\{1, 2, 4, 5, 6, 7, 10\}$	E	B	C

Conversion of an NFA to a DFA

- **Example**



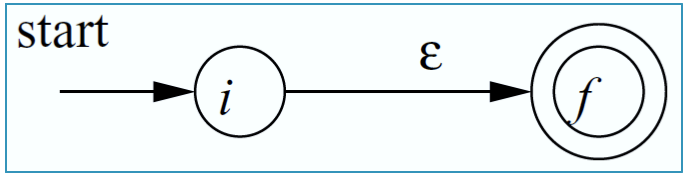
Simulation of an NFA

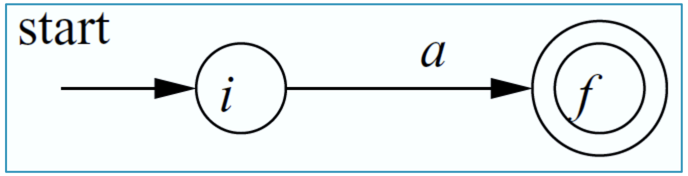
```
S = ε-closure(s0);
c = nextChar();
while ( c != eof ) {
        S = ε-closure(move(S, c));
        c = nextChar();
}
if ( S ∩ F != ∅ ) return "yes";
else return "no";
```

Construction of an NFA from a Regular Expression

- **Basis**

حالت پایه



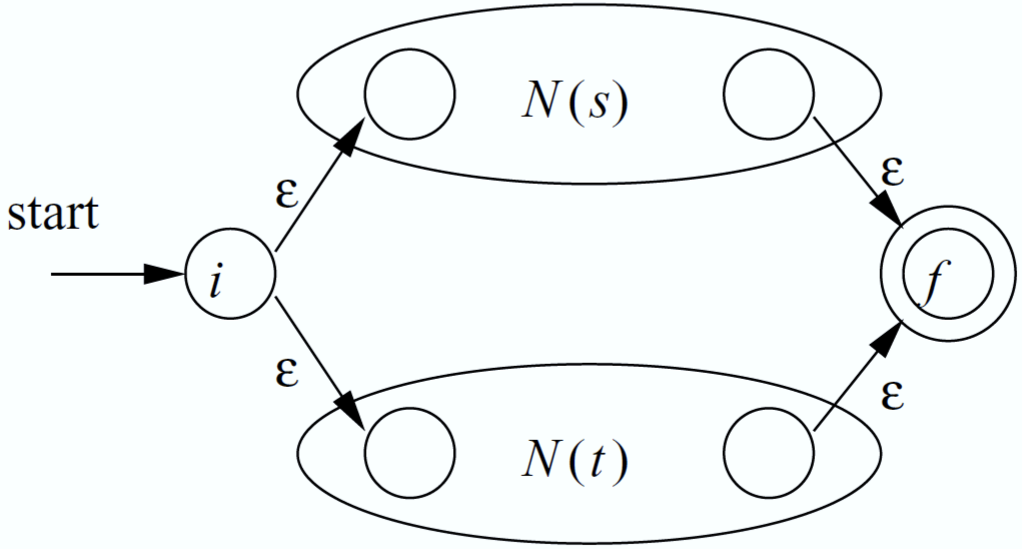


ساخت NFA از یک عبارت منظم:

وقتی عبارت منظم رو بخوایم به NFA تبدیل کنیم توی کوچکترین حالت ممکن، یا یک کاراکتر داریم یا اپسیلون

Construction of an NFA from a Regular Expression

- **Induction**

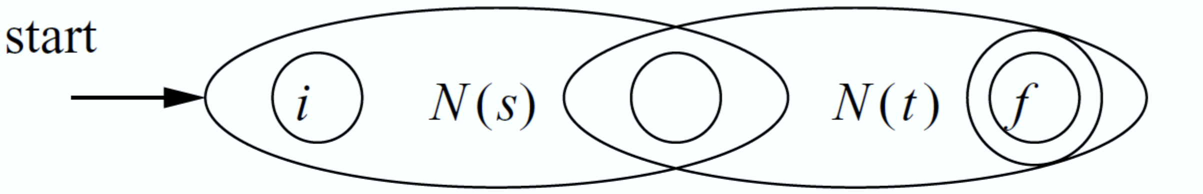


NFA for the union of two regular expressions

Construction of an NFA from a Regular Expression

- **Induction**

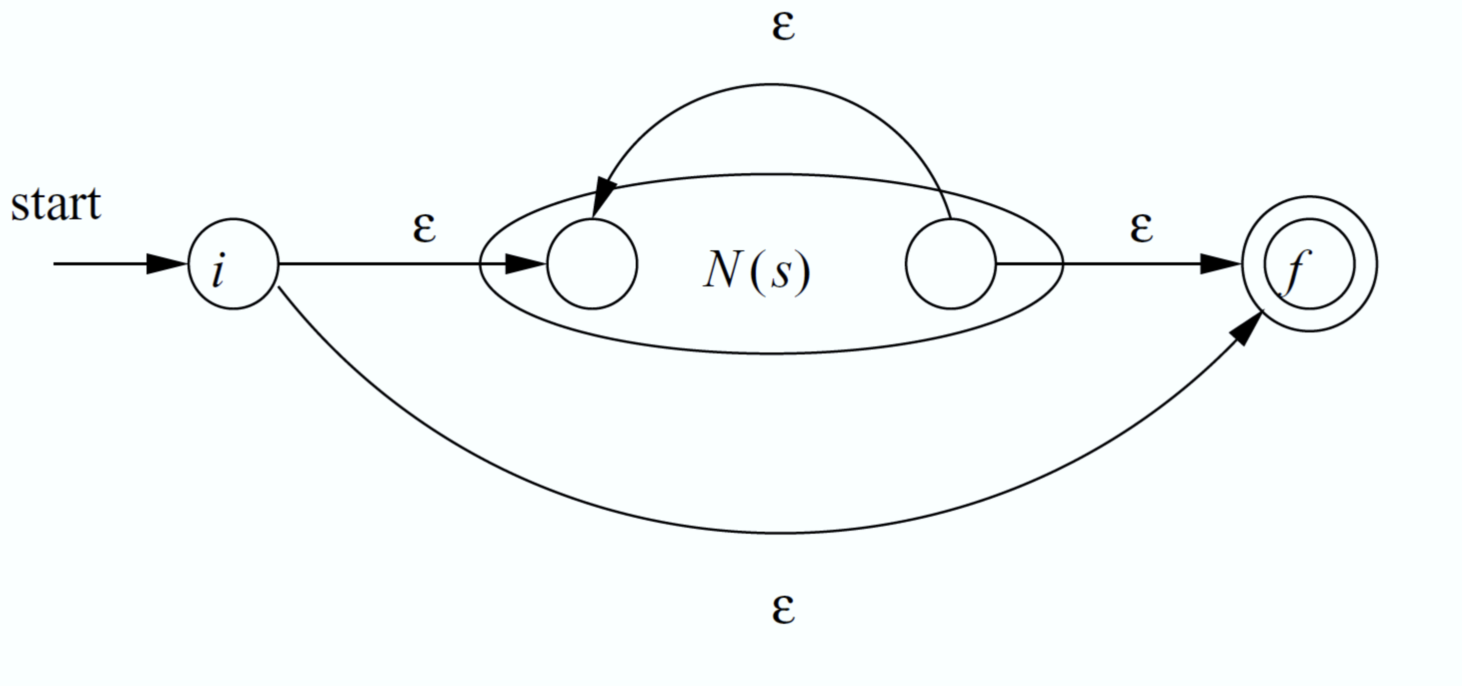
استیت اخر اولی میشه استیت اول دومی



NFA for the concatenation of two regular expressions

Construction of an NFA from a Regular Expression

- **Induction**

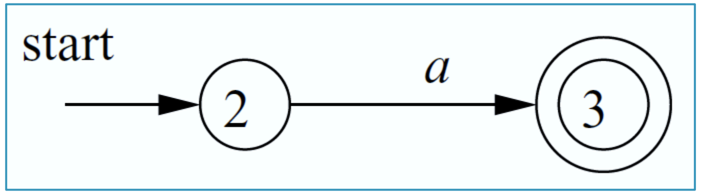


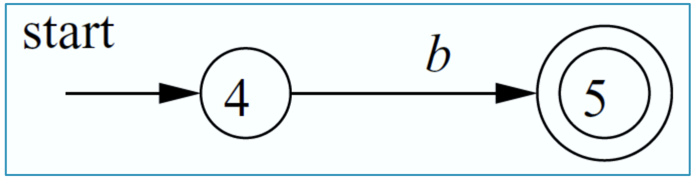
NFA for the closure of a regular expression

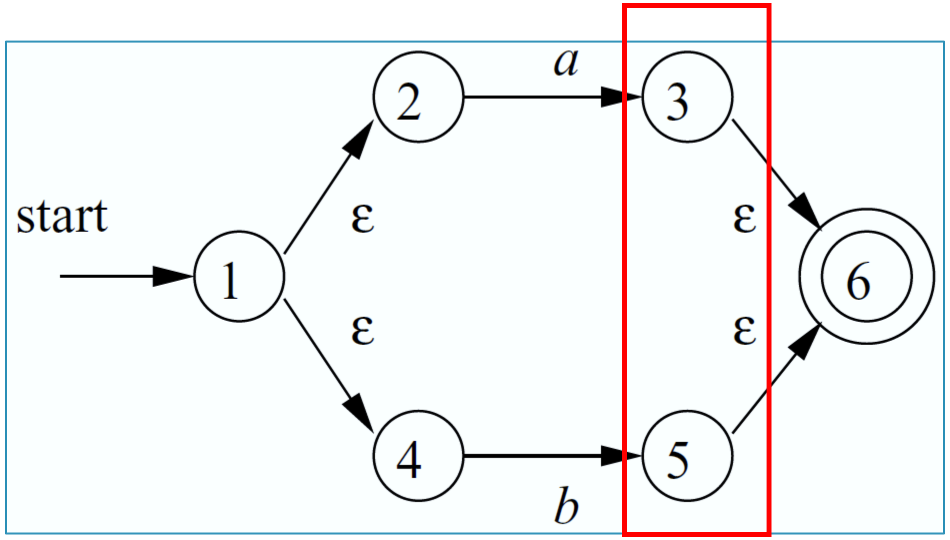
Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^*abb$

جز قاعده اش است که از حالت فاینال بودن در میان

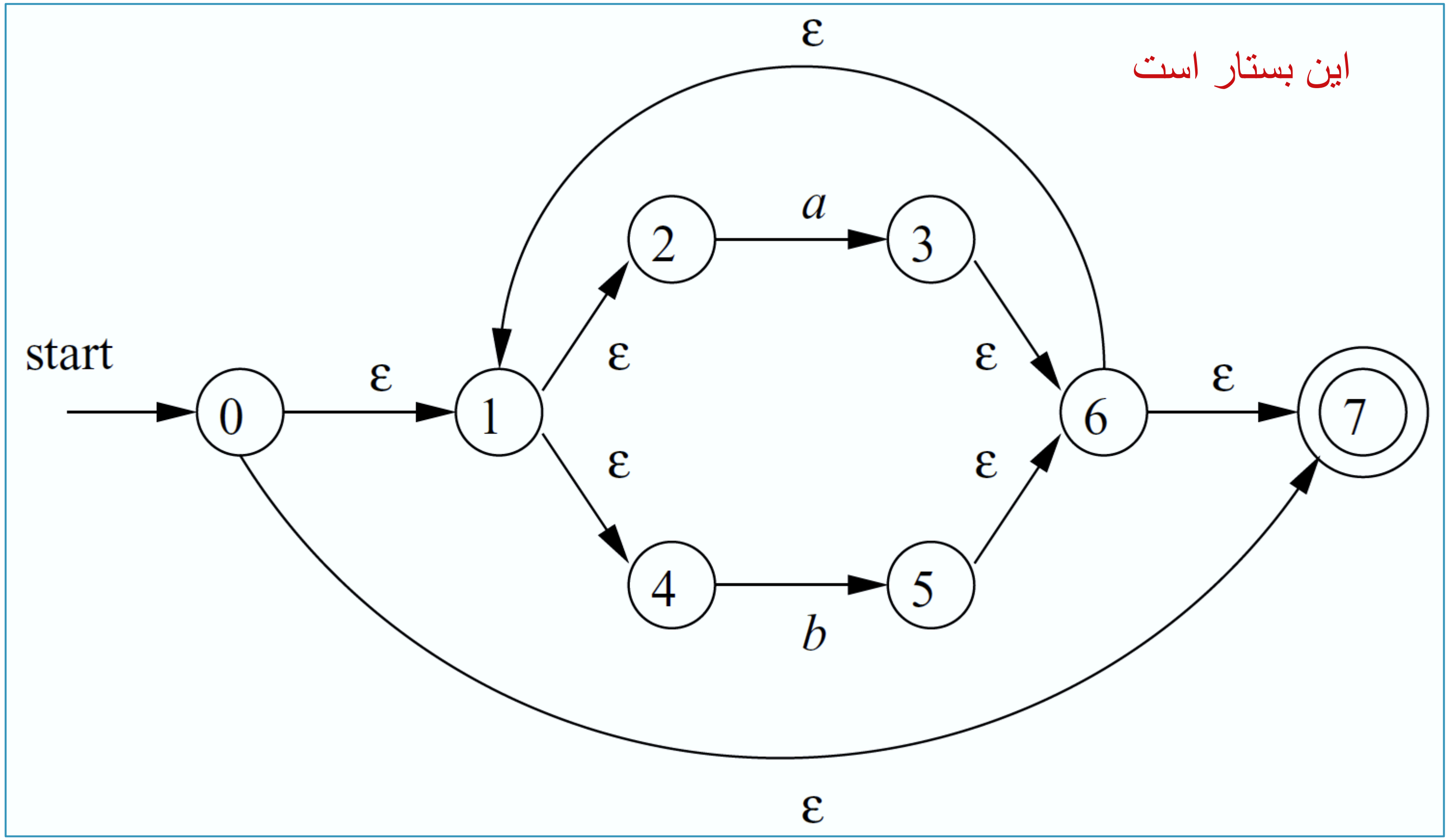






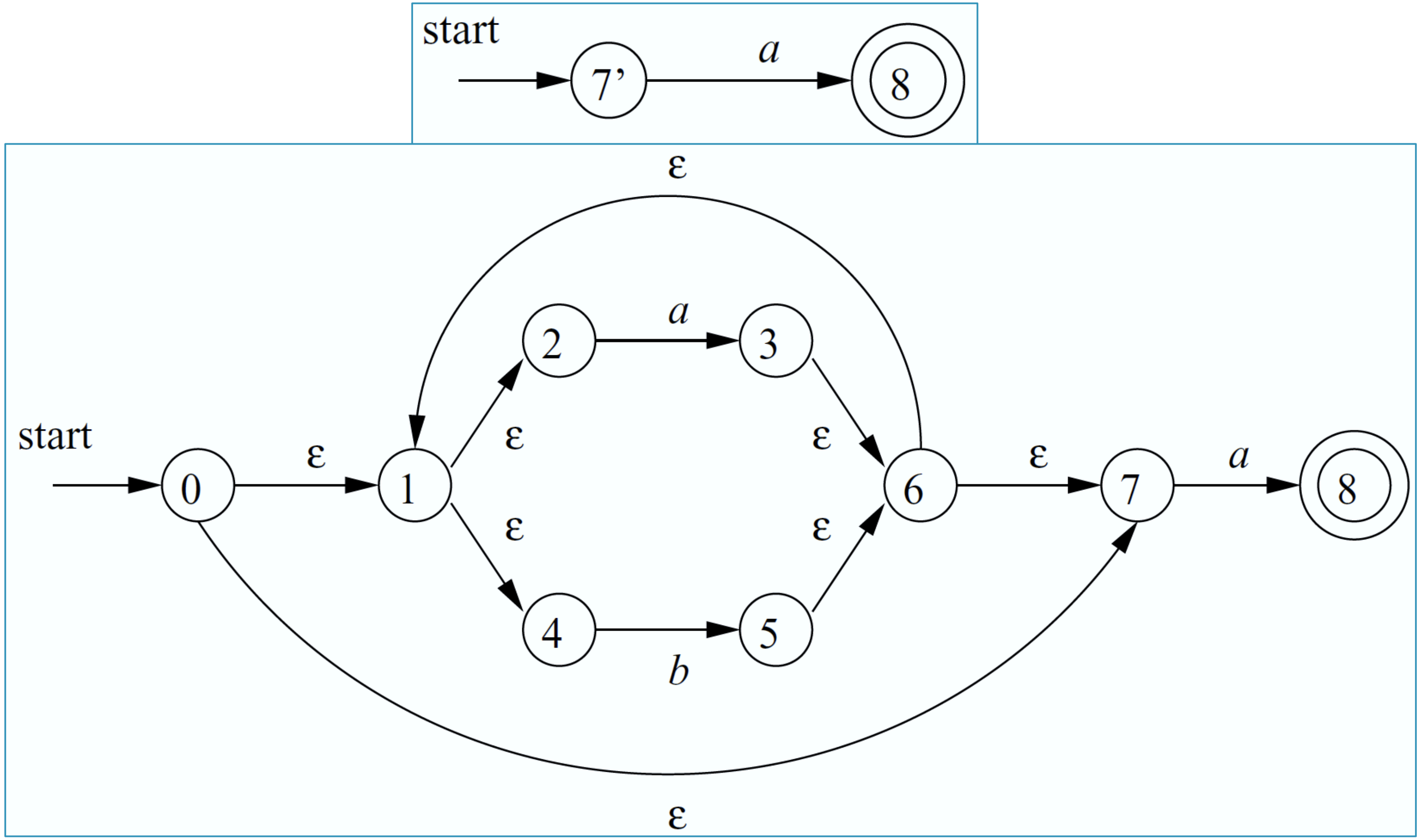
Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^*abb$



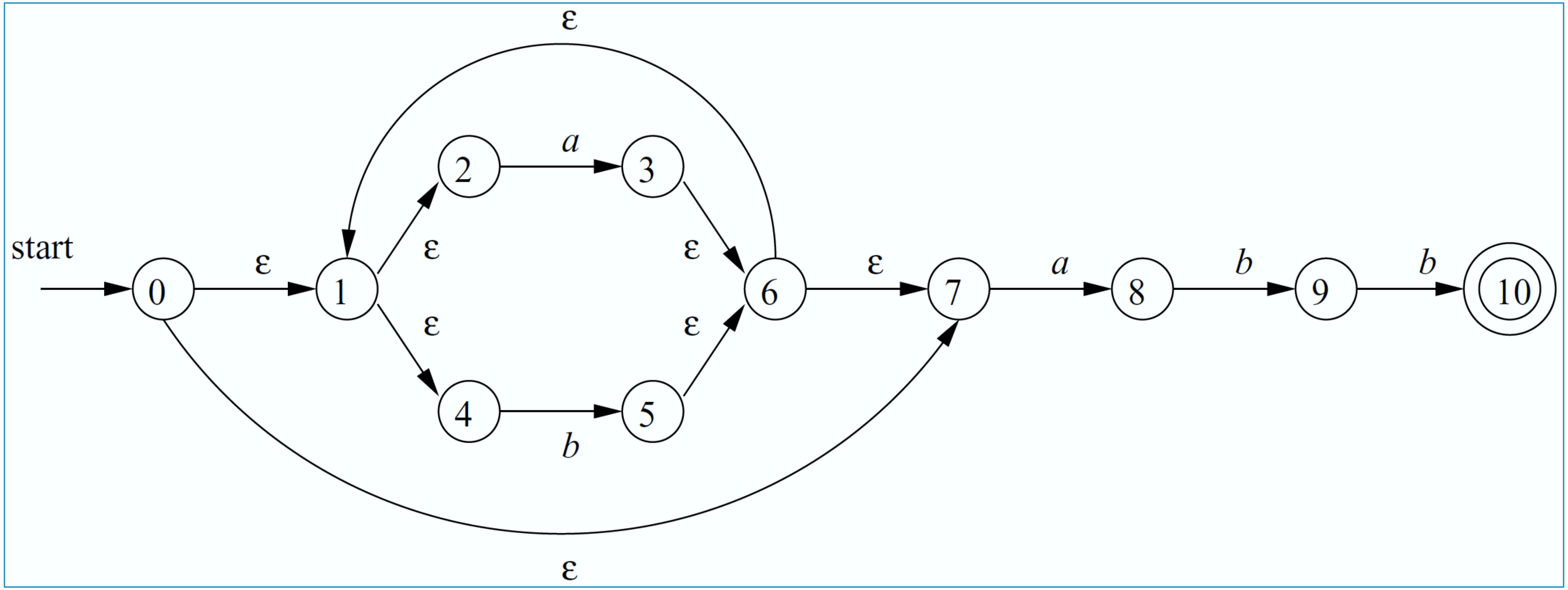
Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^* abb$



Construction of an NFA from a Regular Expression

- **Example:** Construct an NFA for $r = (a|b)^*abb$



Construction of an NFA from a Regular Expression

- **Example**

a) $(\mathbf{a}|\mathbf{b})^*$.

b) $(\mathbf{a}^*|\mathbf{b}^*)^*$.

c) $\left((\epsilon|\mathbf{a})\mathbf{b}^*\right)^*$.

d) $(\mathbf{a}|\mathbf{b})^*\mathbf{abb}(\mathbf{a}|\mathbf{b})^*$.

Converting a Regular Expression Directly to a DFA

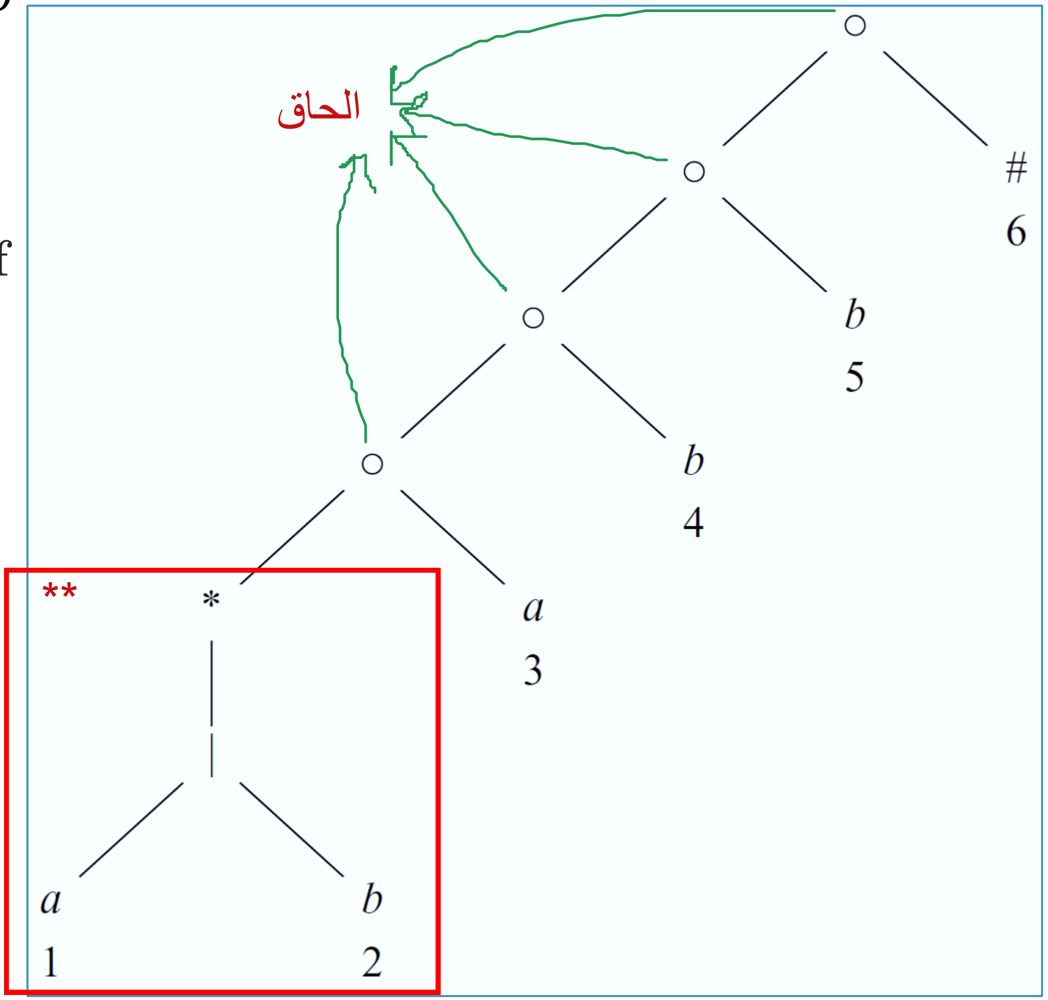
- **Algorithm**
  1. Construct a syntax tree T from the augmented regular expression $(r)\#$
  2. Compute *nullable*, *firstpos*, *lastpos*, and *followpos* for T
  3. Construct *Dstates*, the set of states of DFA D, and *Dtran*, the transition function for D, using the following algorithm

```
initialize Dstates to contain only the unmarked state firstpos(n0),
        where n0 is the root of syntax tree T for (r)#;
while ( there is an unmarked state S in Dstates ) {
        mark S;
        for ( each input symbol a ) {
                let U be the union of followpos(p) for all p
                        in S that correspond to a;
                if ( U is not in Dstates )
                        add U as an unmarked state to Dstates;
                Dtran[S, a] = U;
        }
}
```

-
تبدیل مستقیم به DFA:
الگوریتم:
1- در ابتدا به اون عبارت منظم # اضافه میکنیم و بعد از این کار یک درخت براش رسم می شه
2- قسمت اصلی: که یکسری توابع باید حساب بشه به اسم های nullable, firstpos, lastpos, and followpos که این موارد توی اون درخت باید برای همه نودها حساب بشه
3- و بعد از روی اون درخت میگیم DFA چجوری ساخته میشه

Converting a Regular Expression Directly to a DFA

- **Example:** Regular expression $(a|b)^*abb$
- **Syntax tree for $(\boldsymbol{a}|\boldsymbol{b})^*\boldsymbol{abb}\#$**
  - To each leaf not labeled ϵ, we attach a unique integer as the position of the leaf



-
مرحله اول یک درختی ساخته بشه از عبارات منظمی که # به اخرش اضافه شده

تک تک حرف ها توی برگ هاش است و بعد به ازای اون عملگرهایی که داریم میایم این برگ ها رو بهم وصل میکنیم

برای ورودی توابع: n منظور یک نود توی درخت است و p منظور پوزیشن است

Converting a Regular Expression Directly to a DFA

- **Functions Computed From the Syntax Tree**
  - ***nullable(n)*** is true for a syntax-tree node n if and only if the subexpression represented by n has ϵ in its language
  - ***firstpos(n)*** is the set of positions in the subtree rooted at n that correspond to the first symbol of at least one string in the language of the subexpression rooted at n
  - ***lastpos(n)*** is the set of positions in the subtree rooted at n that correspond to the last symbol of at least one string in the language of the subexpression rooted at n
  - ***followpos(p)***, for a position p, is the set of positions q in the entire syntax tree that can come after p

-

اول از همه باید برای هر یه دونه کاراکتری که توی این عبارت منظم داریم که توی برگ های درخت است یک پوزیشن در نظر بگیریم از همون اول تا اخر که شماره اینا از یک شروع میشه که توی صفحه قبل این شماره گذاری ها وجود داره

چون بعضی از این توابع روی این پوزیشن ها کار میکنند و این پوزیشن ها رو به عنوان ورودی میگیره نه کاراکتر رو

nullable: ینی null می تونه باشه یا نه --> برای هر گره ای که توی این درخت داریم کلا هر گره ای حالا چه برگ باشه و یا چه گره های میانی --> اگر بخوایم nullable رو حساب بکنیم ینی ساب تیری مربوط به اون گره می تونه null باشه یا نه --> در صورتی برای یک گره می شه true که ما بتونیم توی ساب تیری مربوط به اون گره به null برسیم مثلا توی ** به null می رسیم و براش true است

firstpos و lastpos هم باز روی گره ها حساب می شن --> firstpos: ینی اولین پوزیشن n و این ینی اون نوده توی ساب تیری که داره اولین پوزیشن رشته هایی که تولید میکنه چه کاراکترهایی می تونه باشه مثلا برای a|b رشته هاش یا a است یا b و می تونه با a شروع بشه یا b -

lastpos هم ینی با چه کاراکترهایی می تونه تموم بشه

followpos به ازای تک تک پوزیشن ها به دست میاد و نه به ازای تک تک نودها ینی توی صفحه قبل اینو برای 6 تا پوزیشن می تونیم به دست بیاریم --> اینی ینی ما بعد از اون پوزیشن خاص چه پوزیشن هایی ممکنه توی اون عبارت بیاد مثلا پوزیشن 4 الان داریم و این میگه بعد از پوزیشن 4 چه پوزیشن های دیگه ممکنه توی اون عبارت بیاد

Converting a Regular Expression Directly to a DFA

NODE n	$nullable(n)$	$firstpos(n)$
A leaf labeled ϵ	**true**	$\emptyset$
A leaf with position i	**false**	$\{i\}$
An or-node $n = c_1 \| c_2$	$nullable(c_1)$ **or** $nullable(c_2)$	$firstpos(c_1) \cup firstpos(c_2)$
A cat-node $n = c_1c_2$	$nullable(c_1)$ **and** $nullable(c_2)$	**if** ($nullable(c_1)$) $firstpos(c_1) \cup firstpos(c_2)$ **else** $firstpos(c_1)$
A star-node $n = {c_1}^*$	**true**	$firstpos(c_1)$

اگر یکیشون فقط null باشه این true میشه

باید هر دوتاش null باشه که این true بشه

این چون می تونه اپسیلون تولید بکنه کلا nullable اش true است

Converting a Regular Expression Directly to a DFA

- **Example**
  - ***firstpos*** and ***lastpos*** for nodes in the syntax tree for $(a|b)^*abb\#$

برای هر گره ای firstpos سمت چپش نوشته شده
و lastpos سمت راستش