ΕΠΛ323 - Θεωρία και Πρακτική Μεταγλωττιστών

Lecture 4a

Lexical Analysis

Elias Athanasopoulos
eliasathan@cs.ucy.ac.cy

From a regular expression to an NFA

- Construct an NFA from a regular expression
- Simulate the behavior of the NFA with specific algorithms
- If run-time speed is essential
  - Convert NFA to DFA (see lecture 3b)

Thompson's construction

Construct an NFA from a regular expression

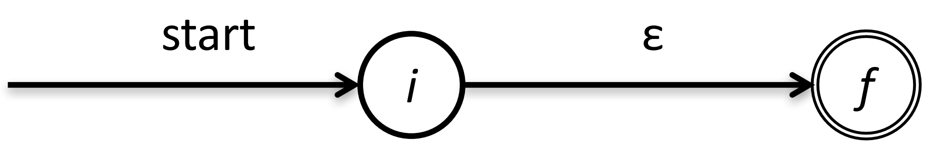
- Input
  - A regular expression *r* over an alphabet Σ
- Output
  - An NFA *N* accepting *L(r)*

Bootstrap

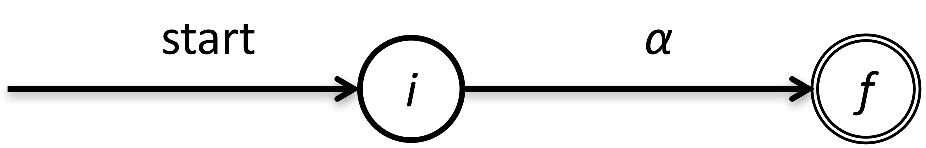
- We first parse *r* into its constituent expressions
- Then, using rules (1) and (2) (next slide), we construct NFAs for each of the basic symbols in *r*
- If a symbol *a* occurs several times in *r*, a separate NFA is constructed for each occurrence

Core rules

- Rule 1, for ε construct the NFA:

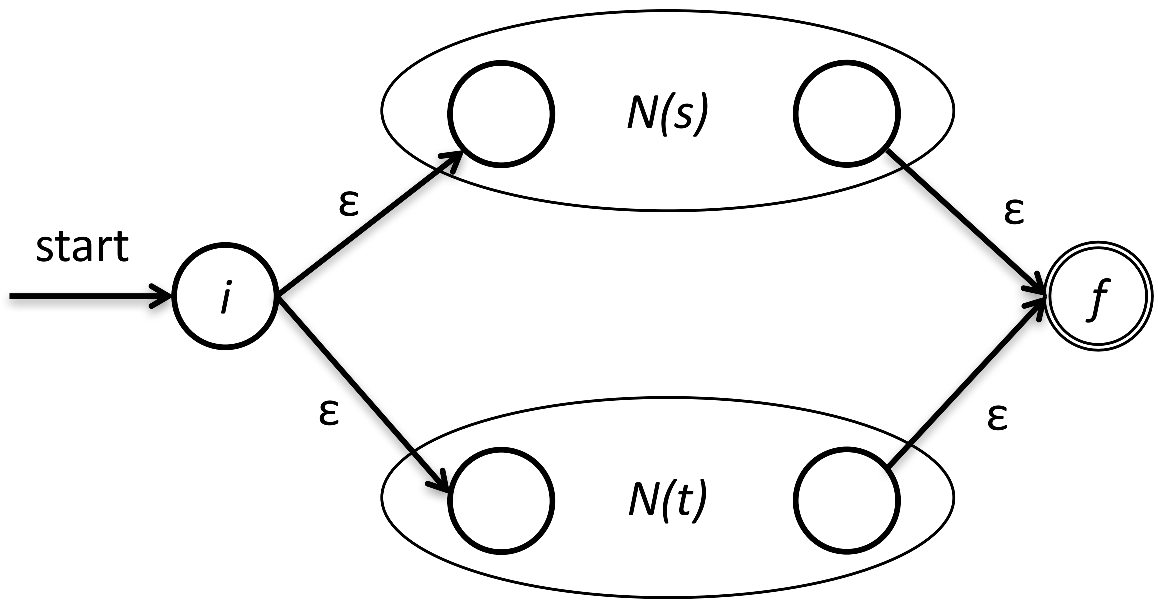


- Rule 2, for *a* in Σ, construct the NFA:



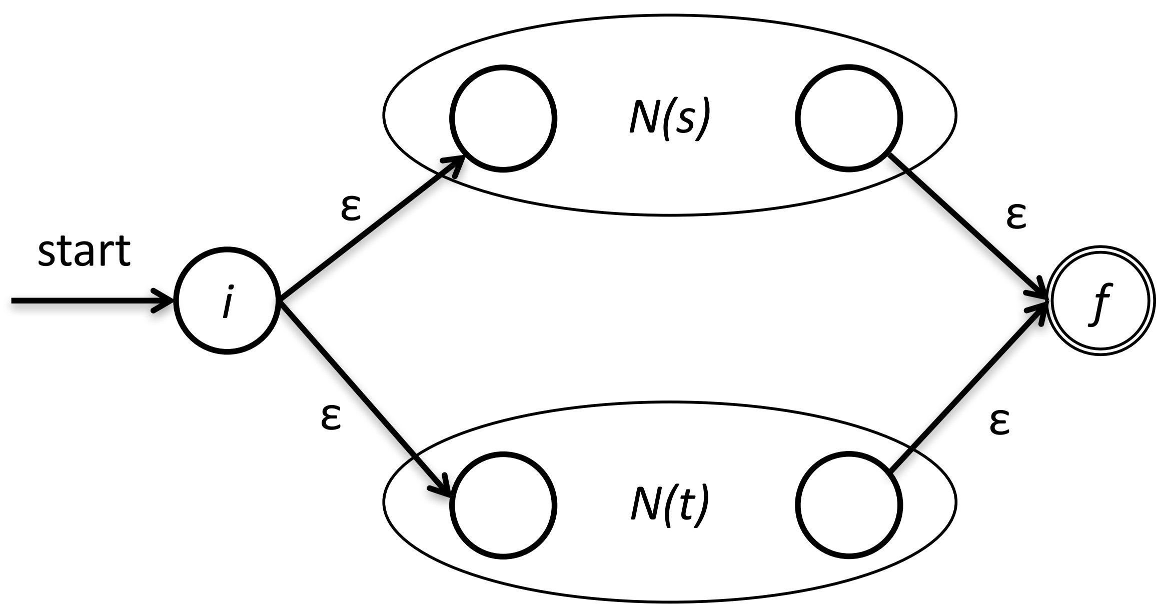
s | *t*

- *N(s)* and *N(t)* are NFAs for regular expressions *s* and *t*



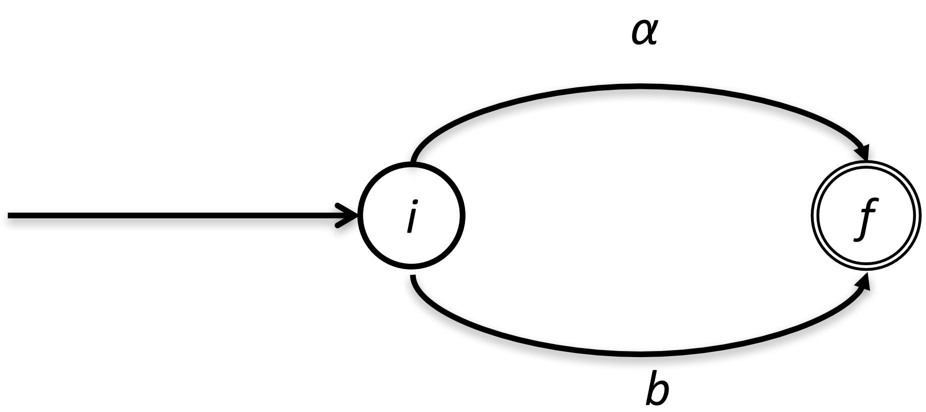
$s|t$

Here *i* is a new start state and *f* a new accepting state. There is a transition on ε from *i* to the start states of *N(s)* and *N(t)*. There is a transition on ε from the accepting states of *N(s)* and *N(t)* to the new accepting state f. The start and accepting states of *N(s)* and *N(t)* **are not start** or accepting states of *N(s|t)*. Note that any path from *i* to *f* must pass through either *N(s)* or *N(t)* exclusively. Thus, we see that the composite NFA recognizes *L(s)UL(t)*.



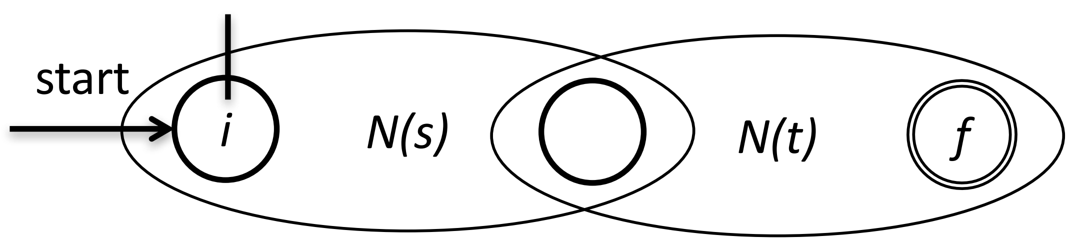
Simplification (no ε-transition)

- Example a/b



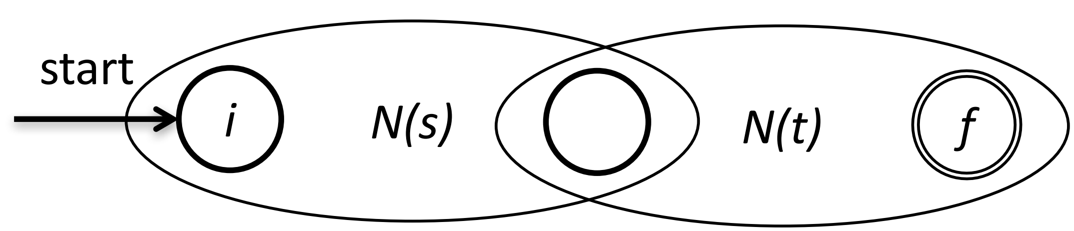
st

- *N(s)* and *N(t)* are NFAs for regular expressions *s* and *t*



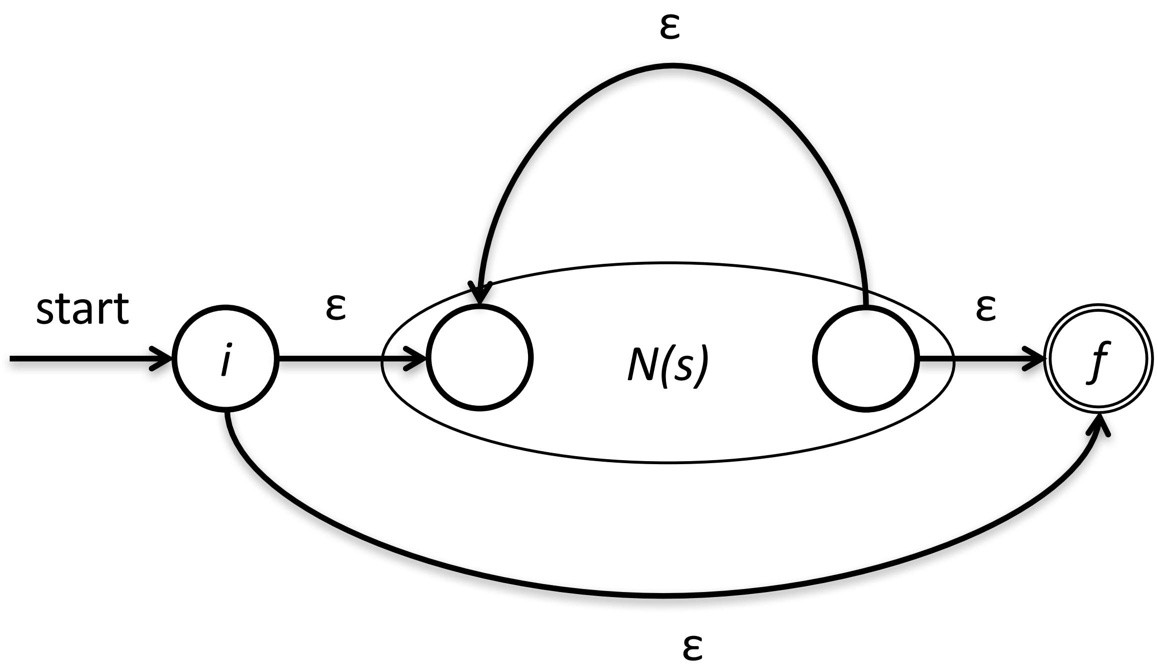
st

The start state of *N(s)* becomes the start state of the composite NFA and the accepting state of *N(t)* becomes the accepting state of the composite NFA. The accepting state of *N(s)* is merged with the start state of N(t); that is, all transitions from the start state of *N(t)* become transitions from the accepting state of *N(s)*. The new merged state loses its status as a start of accepting state in the composite NFA. A path from *i* to *f* must go first through *N(s)* and then through *N(t)*, so the label of that path will be a string in *L(s)L(t)*. Since no edge enters the start state of *N(t)* or leaves the accepting state of *N(s)*, there can be no path from *i* to *f* that travels from *N(t)* back to *N(s)*. Thus, the composite NFA recognizes *L(s)L(t)*.



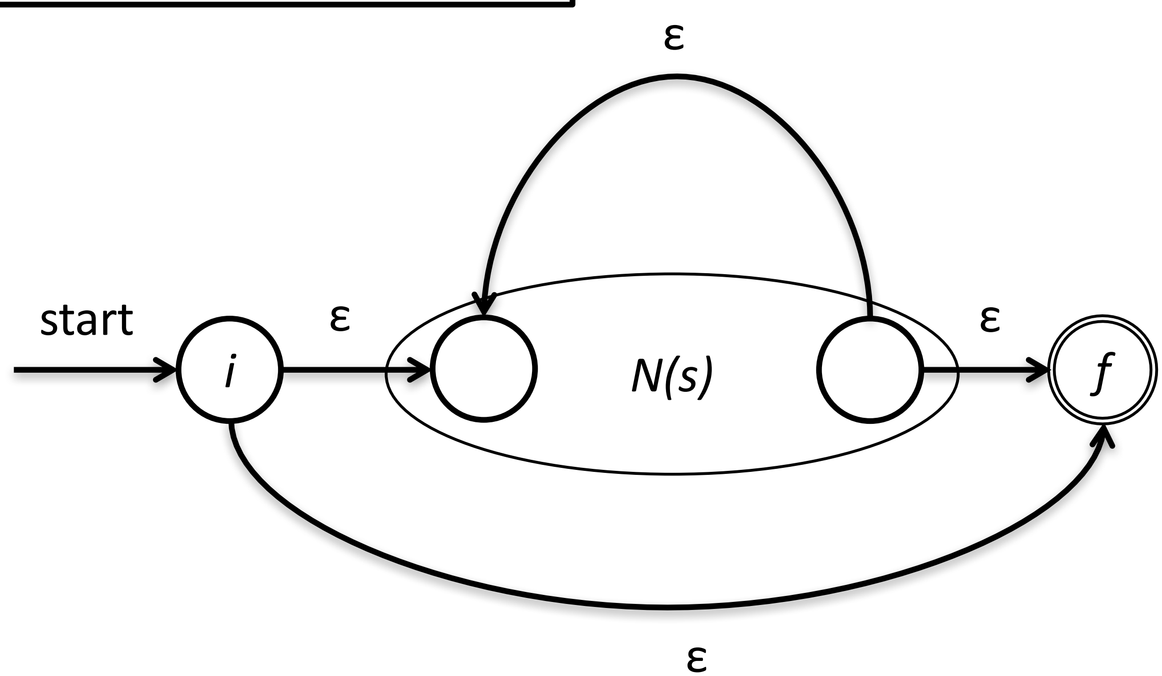
s^*

- *N(s)* is the NFA for the regular expression s^*



s^*

Here *i* is a new state and *f* a new accepting state. In the composite NFA, we can go from *i* to *f* directly, along an edge labeled ε, representing the fact that ε is in *(L(s))**, or we can go from *i* to *f* passing through *N(s)* one or more times.

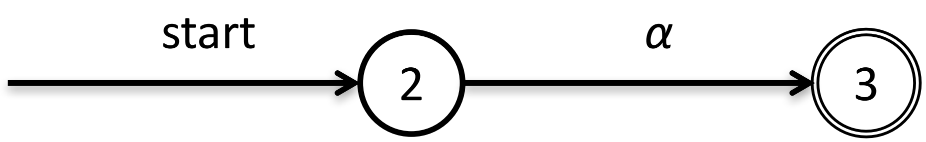


(s)

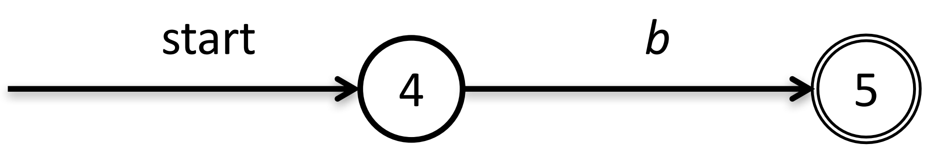
- For the parenthesized regular expression *(s)*, use *N(s)* itself as the NFA.

$(a|b)^*abb$

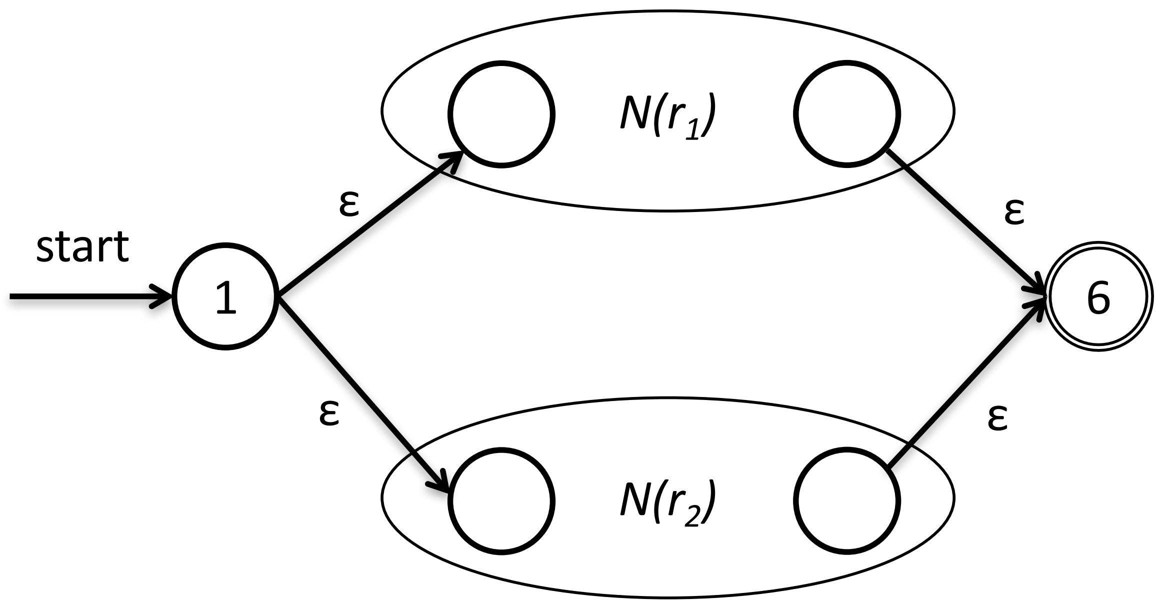
- $r_1\ (a)$



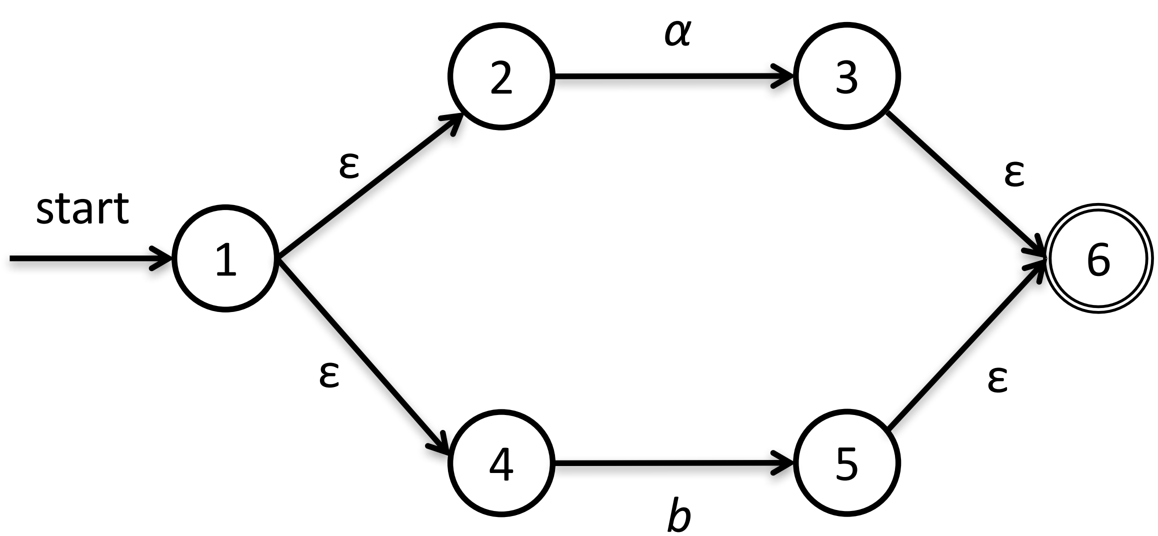
- $r_2\ (b)$



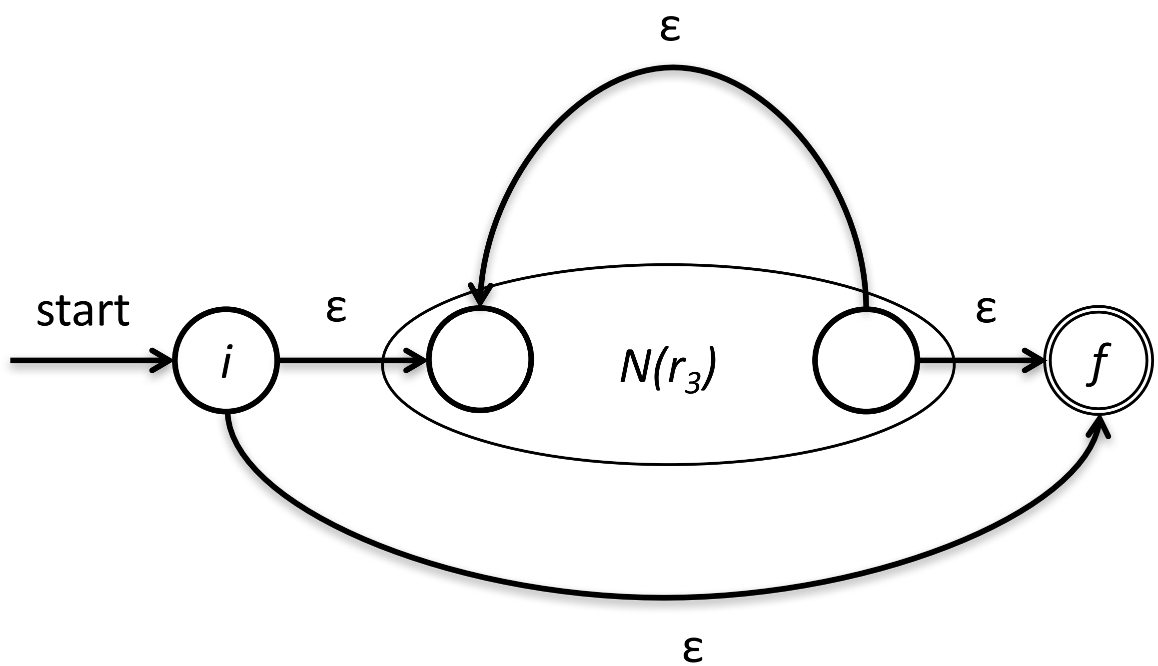
$r_3 = r_1 \mid r_2$

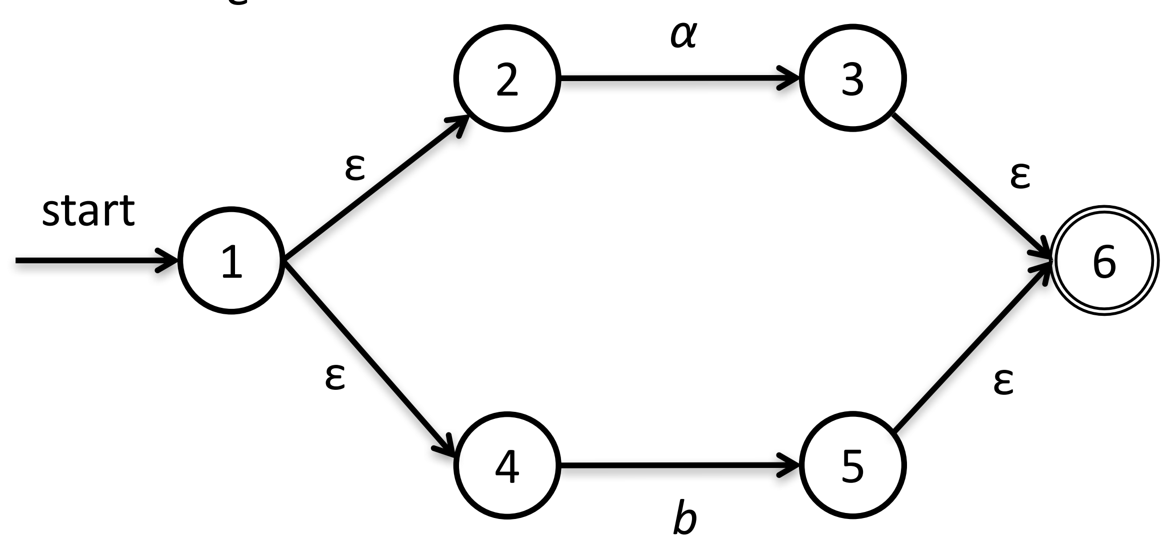


$r_3 = r_1 \mid r_2$

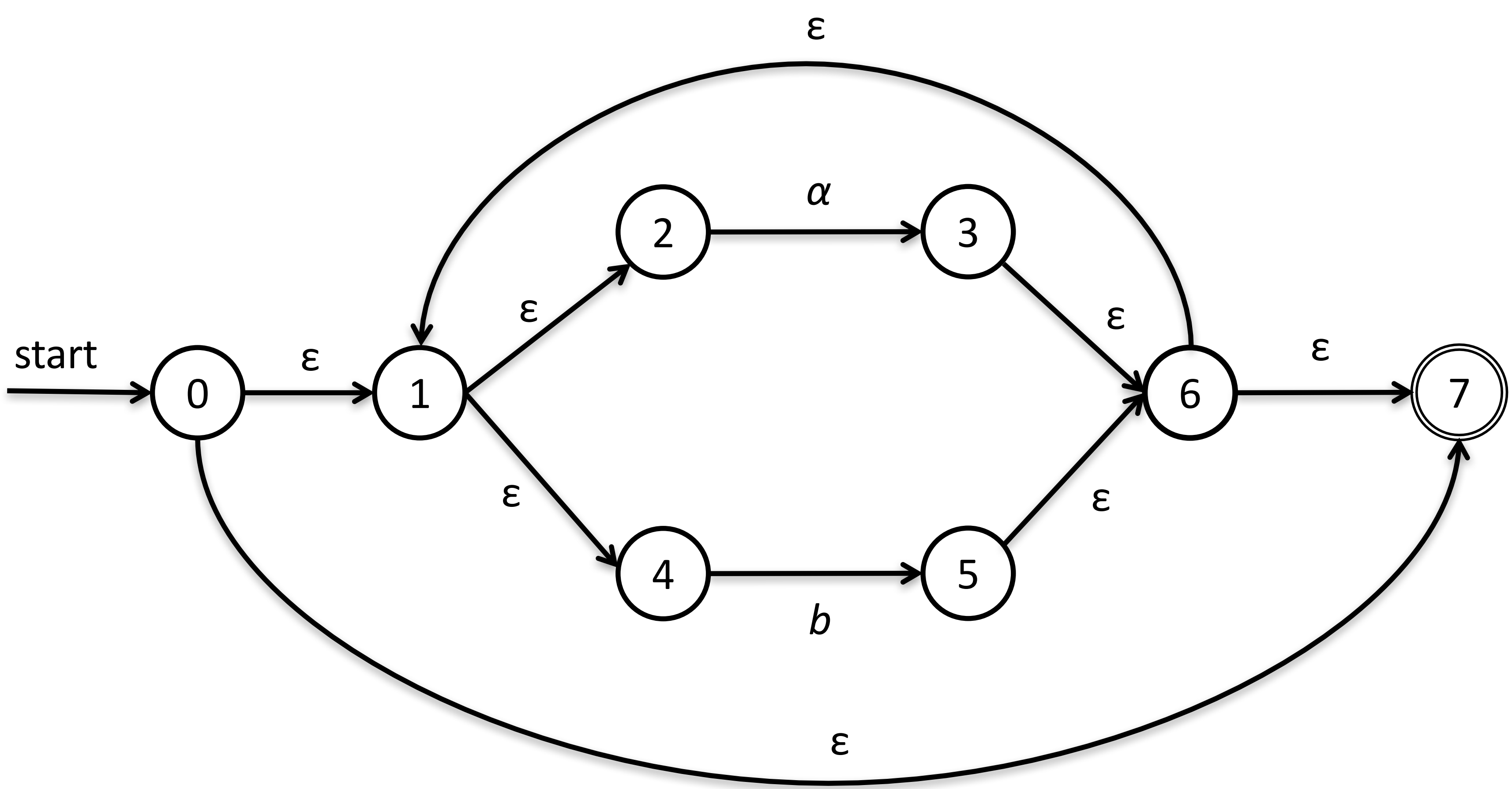


$r_5=(r_3)^*$

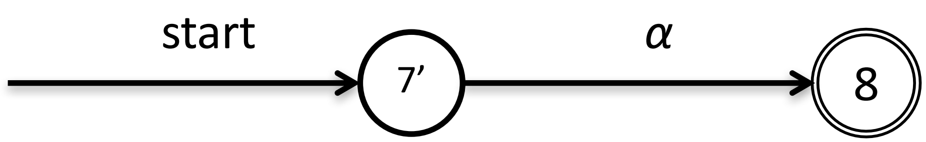




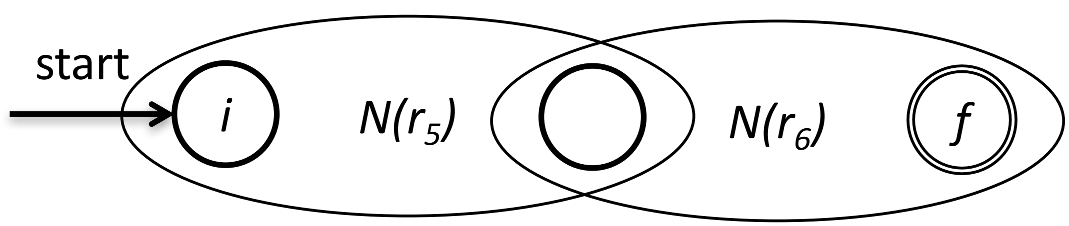
$r_5 = (r_3)^*$



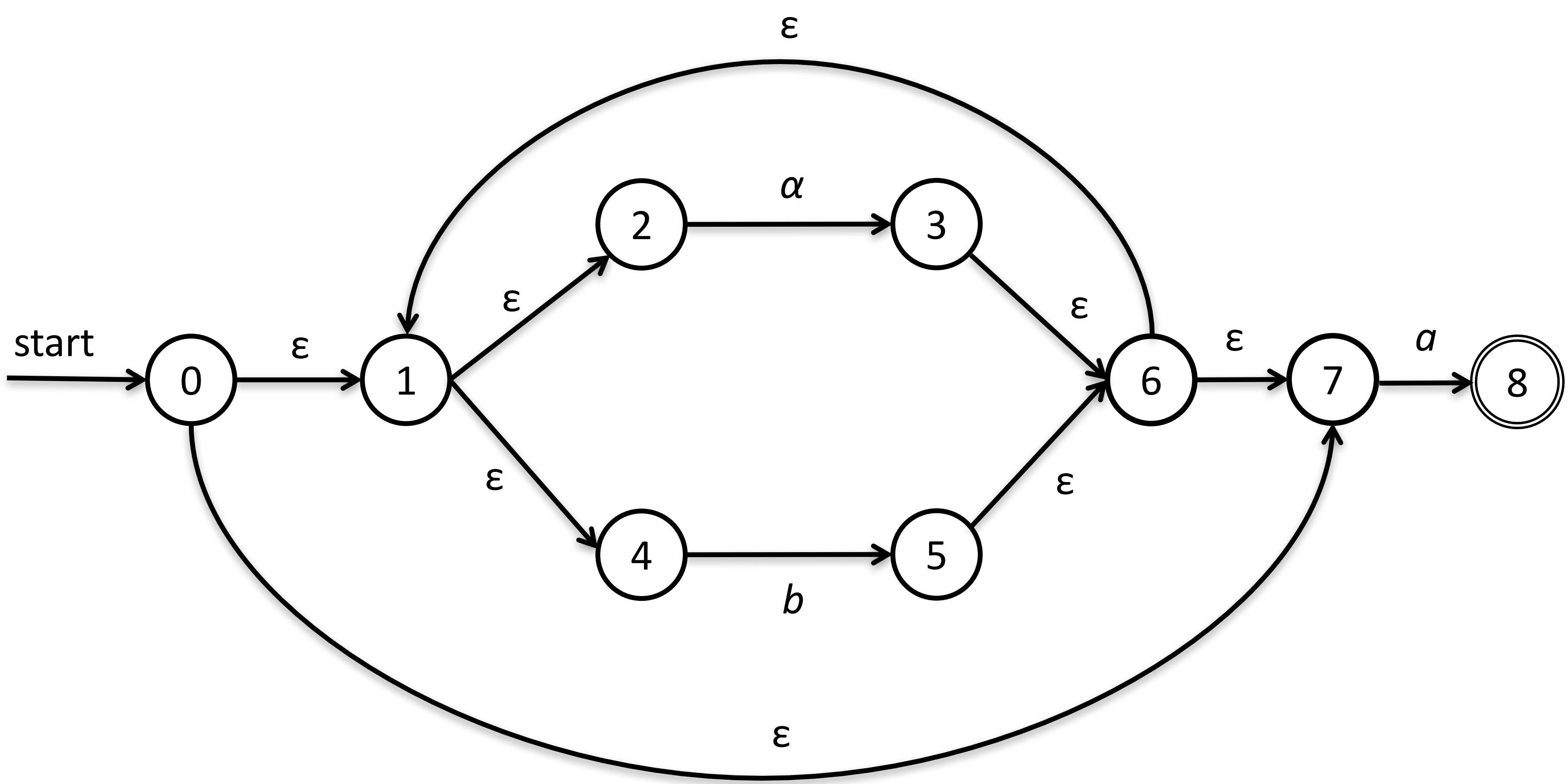
$r_6=\alpha$



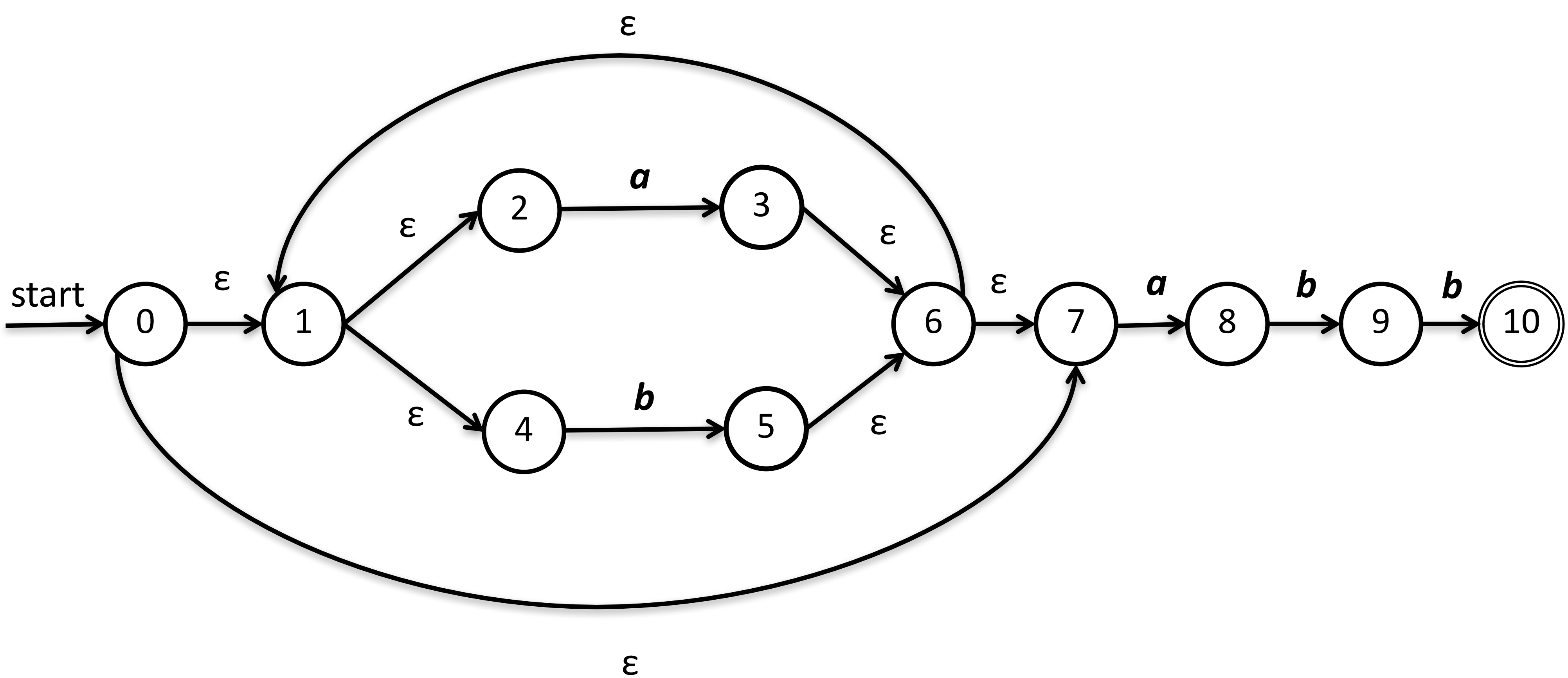
$r_7 = r_5 r_6$



$r_7 = r_5 r_6$



Final NFA



Coding the NFA

```
S := ε-closure({s0});
a := nextchar;
while a <> eof do begin
   S := ε-closure(move(S, a));
   a := nextchar;
end
if accepting-state in S then
   return "yes";
else
   return "no";
```

Example

- Is “*a*” part of the NFA of slide 22?
  - ε-closure({0}) = {0, 1, 2, 4, 7}
- On input symbol *a* there is a transition from 2 to 3 and from 7 to 8
  - ε-closure({3, 8}) = {1, 2, 3, 4, 6, 7, 8}
- None of these states is accepting, therefore the algorithm returns “no’

Time-space Tradeoffs

AUTOMATON	SPACE	TIME
NFA	$O(\|r\|)$	$O(\|r\| \times \|x\|)$
DFA	$O(2^{\|r\|})$	$O(\|x\|)$

We can construct NFA from *r*, and this can be done in $O(|r|)$ time, where $|r|$ is the length of $|r|$. The NFA has at most twice as many states as $|r|$, and at most two transitions from each state, so a transition table for the NFA can be stored in $O(|r|)$ space. The algorithm for coding the NFA takes time $O(|r| \times |x|)$ to resolve if x is accepted.

For DFA complexity, consider the regular expression *(a|b)*(a|b)(a|b)…(a|b)*, where there are *n-1* *(a|b)*s at the end. Then you need 2^n states to keep track of all sequences of *a* and *b*.