Compilers

by
Marwa Yusuf

Lecture 4
Tues. 30-3-2021

Chapter 3 (3.7 to 3.9)

Lexical Analysis

From Regular Expressions to Automata

- Pattern processing software (like lexical analyzer) is implemented using simulating automata.
- DFA simulation is more straightforward (because of nondeterminism of NFA).

1) Convert NFA to DFA.

2) Simulate NFA directly (hen it is more effective).

3) Convert regular expression to NFA.

NFA to DFA

- Each state of DFA corresponds to a set of NFA states.
- It is possible that No. of states in DFA is exponential in No. of NFA states. However, for real languages NFA and DFA have approximately the same number of states.

Subset Construction Algorithm

- **INPUT** : An NFA *N*.
- **OUTPUT** : A DFA *D* accepting the same language as *N*.
- **METHOD** : Our algorithm constructs a transition table *Dtran* for *D*. Each state of *D* is a set of NFA states, and we construct *Dtran* so *D* will simulate "in parallel" all possible moves *N* can make on a given input string.

Operations on NFA States

OPERATION	DESCRIPTION
ε-*closure*(s)	Set of NFA states reachable from NFA state s on ε-transitions alone.
ε-*closure*(T)	Set of NFA states reachable from some NFA state s in set T on ε-transitions alone; $= \mathrm{U}_{\mathrm{i\ in\ T}}$ ε-*closure*(s).
move(T, a)	Set of NFA states to which there is a transition on input symbol a from some state s in T.

NFA to DFA

- N initially can be in any state from ε-*closure*(s_0).
- After reading input string x, N can be in set of states T.
- From T, upon reading a, N moves to any of states *move*(T, a).
- As it could make any ε transition, N can be in any state of ε-*closure*(*move*(T, a)) after reading xa.

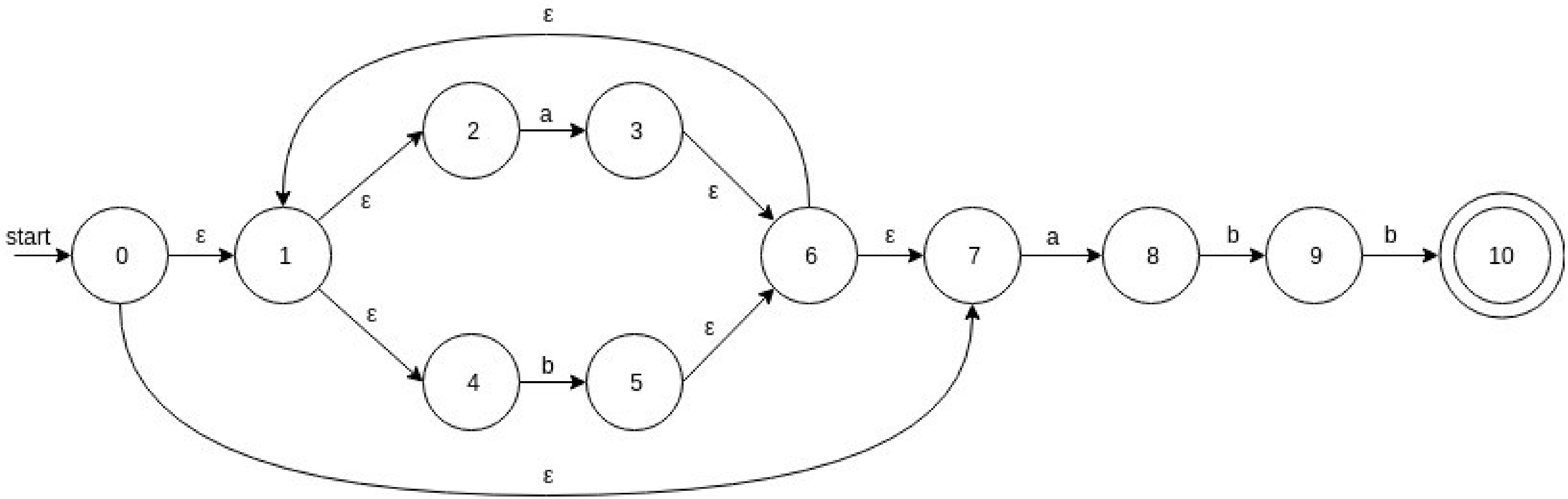
Subset Construction

```
initially, ε-closure(s) 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;
  }
}
```

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;
  }
}
```

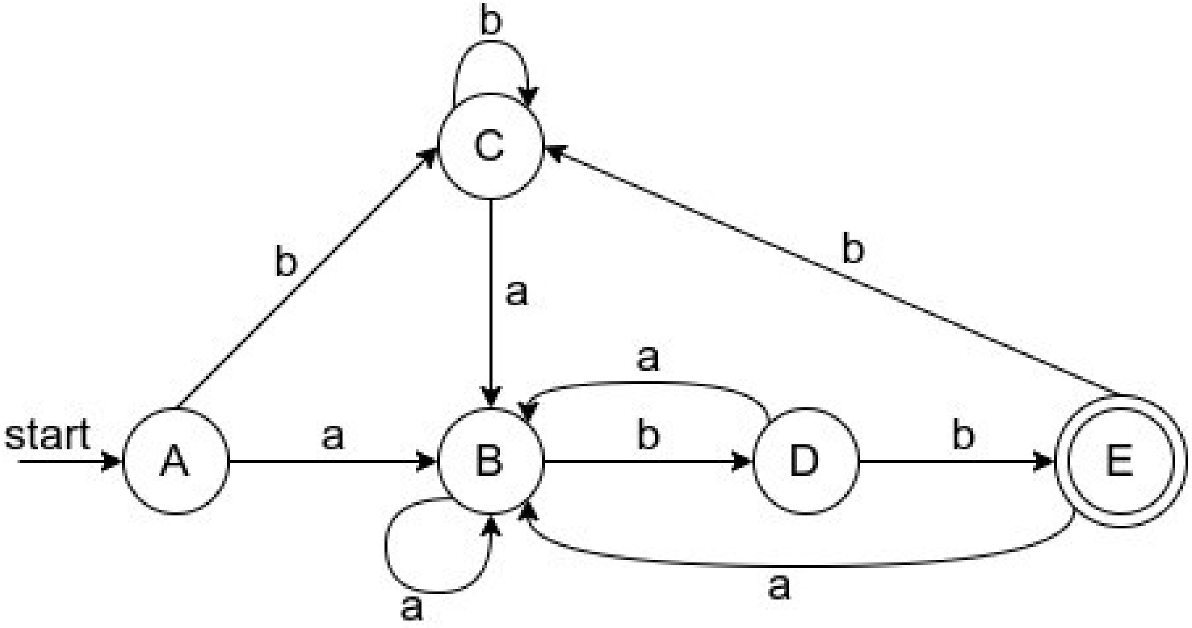
NFA to DFA Example



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
{1,2,3,5,6,7,10}	E	B	C

NFA to DFA Example

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
{1,2,3,5,6,7,10}	E	B	C



Simulation of an NFA

- **INPUT** : An input string x terminated by an end-of-file character **eof**. An NFA N with start state s_0, accepting states F, and transition function *move*.
- **OUTPUT** : Answer "yes" if N accepts x; "no" otherwise.
- **METHOD**: The algorithm keeps a set of current states S, those that are reached from s_o following a path labeled by the inputs read so far. If c is the next input character, read by the function *nextChar*(), then we first compute *move*(S,c) and then close that set using *ε-closure*().

Simulation of an NFA

```
$S$ = $\varepsilon$-closure($s_0$);
$c$ = nextChar();
while ( $c$ != eof ) {
    $S$ = $\varepsilon$-closure(move($S$,$c$));
    $c$ = nextChar();
}
if ( $S \cap F$ != $\Phi$ ) return "yes";
else return "no";
```

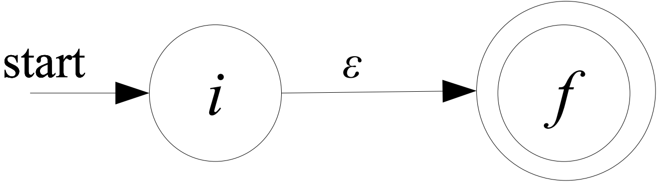
Efficiency of NFA Simulation

- Section 3.7.3 skipped.

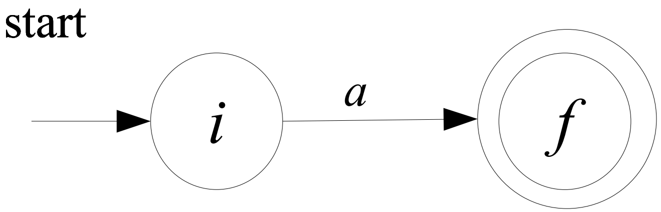
Construction of an NFA from a Regular Expression

(McNaughton-Yamada-Thompson)

- **INPUT**: A regular expression *r* over alphabet Σ.
- **OUTPUT**: An NFA *N* accepting *L*(*r*).
- **METHOD**: Parse *r* into it subexpressions, then use basis rules and inductive rules.
- **BASIS**: For expression ε construct:



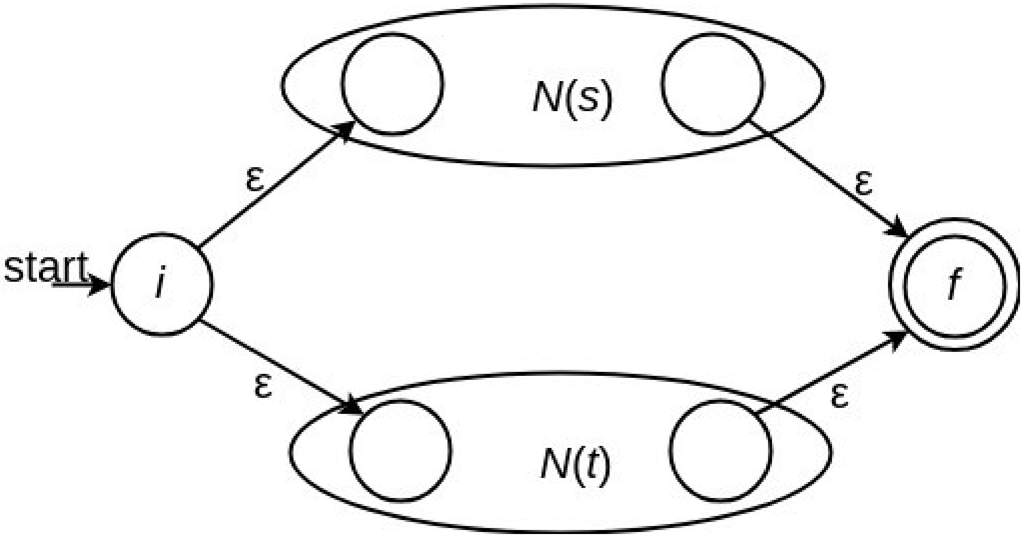
For any subexpression *a* in Σ construct:



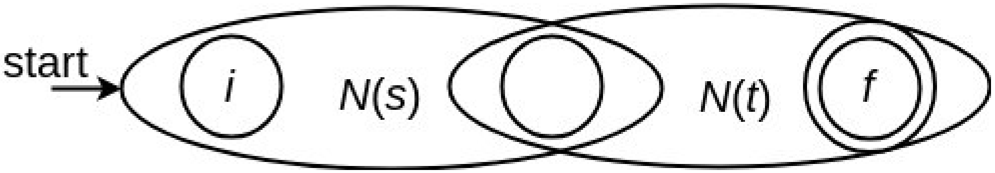
Construction of an NFA from a Regular Expression

- **INDUCTION**: Suppose *N*(*s*) and *N*(*t*) are NFA's for regular expressions *s* and *t*:

  a) If *r* = *s* | *t* then *N*(*r*):

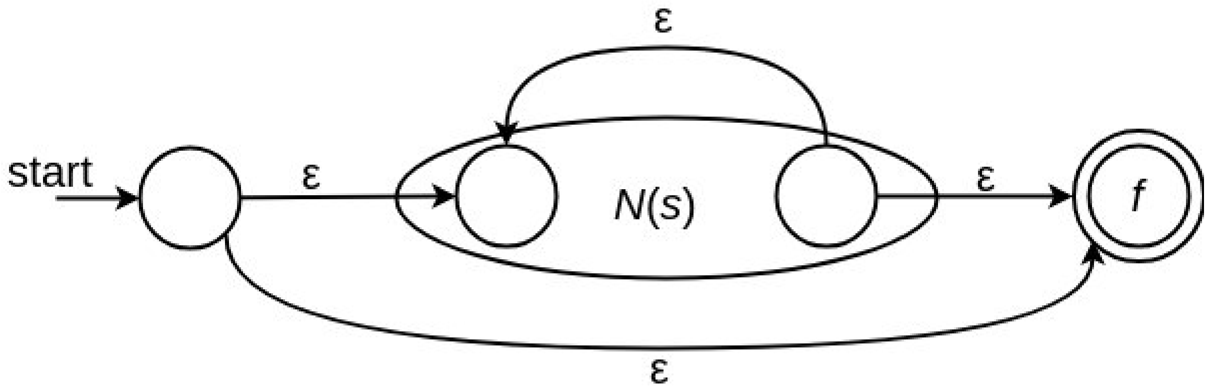


  b) If *r* = *s* | *t* then *N*(*r*):



Construction of an NFA from a Regular Expression

a) If $r = s^*$ then $N(r)$:



b) If $r = (s)$ then $N(r) = N(s)$.

Properties of the constructed NFA

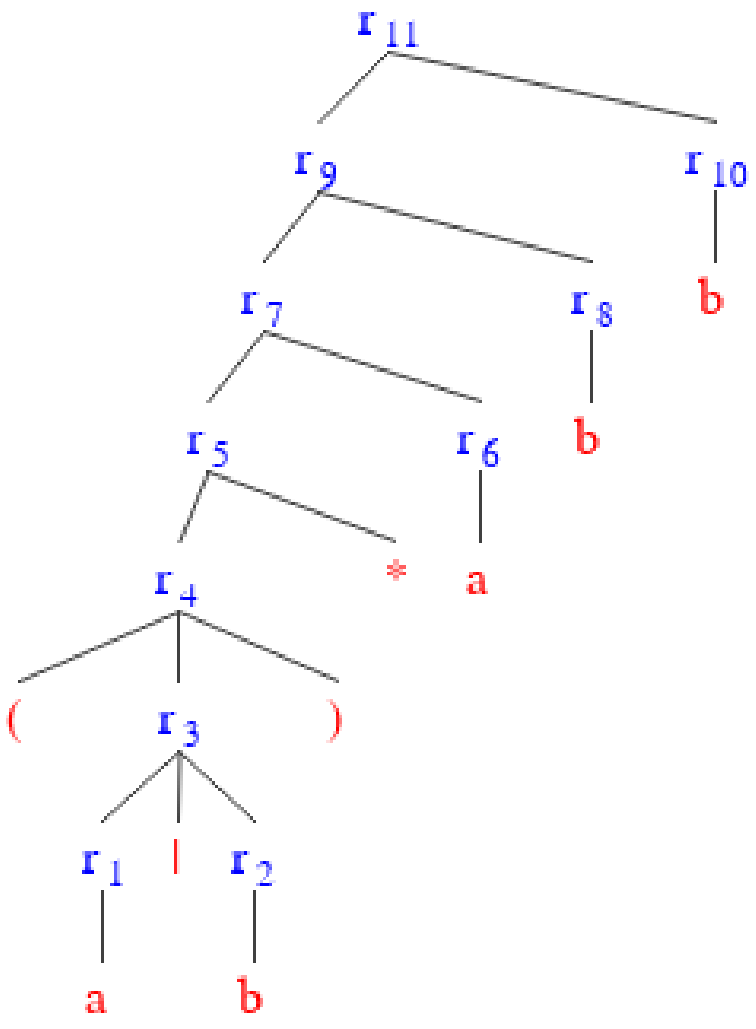
1) *N*(*r*) has at most twice as many states as there operators and operands in *r*.

2) *N*(*r*) has one start state with no ingoing transitions, and one accepting state with no outgoing transitions.

3) Each state of *N*(*r*) other that the accepting state has either one outgoing transition on a symbol in Σ or up to two outgoing transitions both on ε.

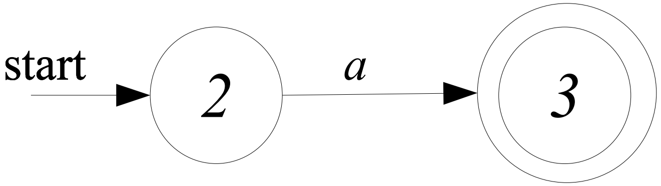
Example

- $r = (\mathbf{a}|\mathbf{b})^{*}\mathbf{abb}$

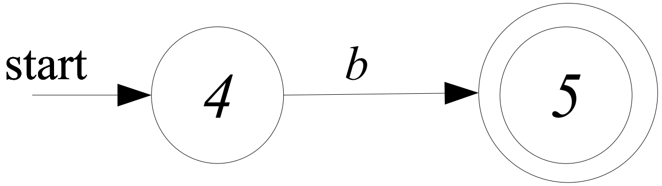


Example

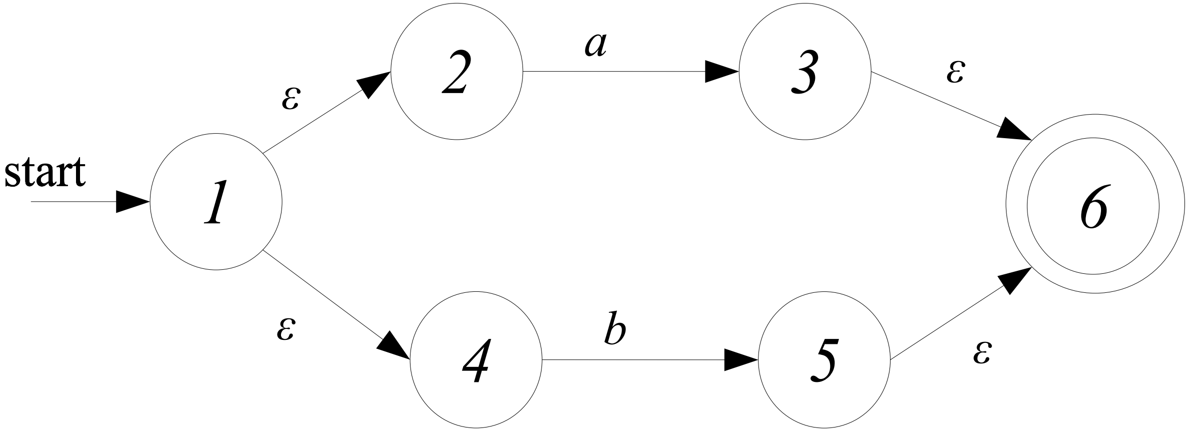
r_1



r_2



r_3

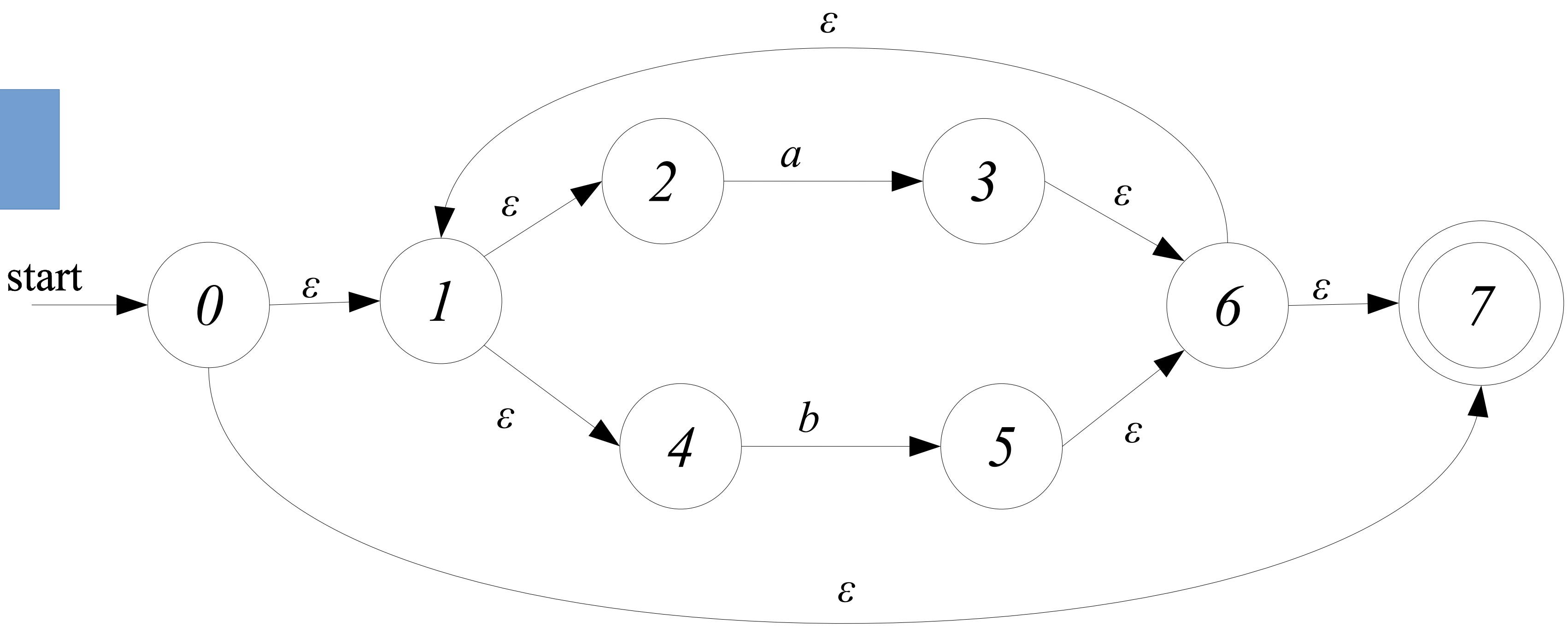


r_4

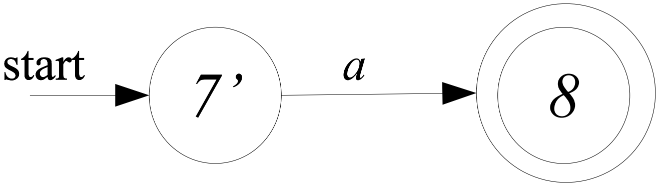
The same as r_3

Example

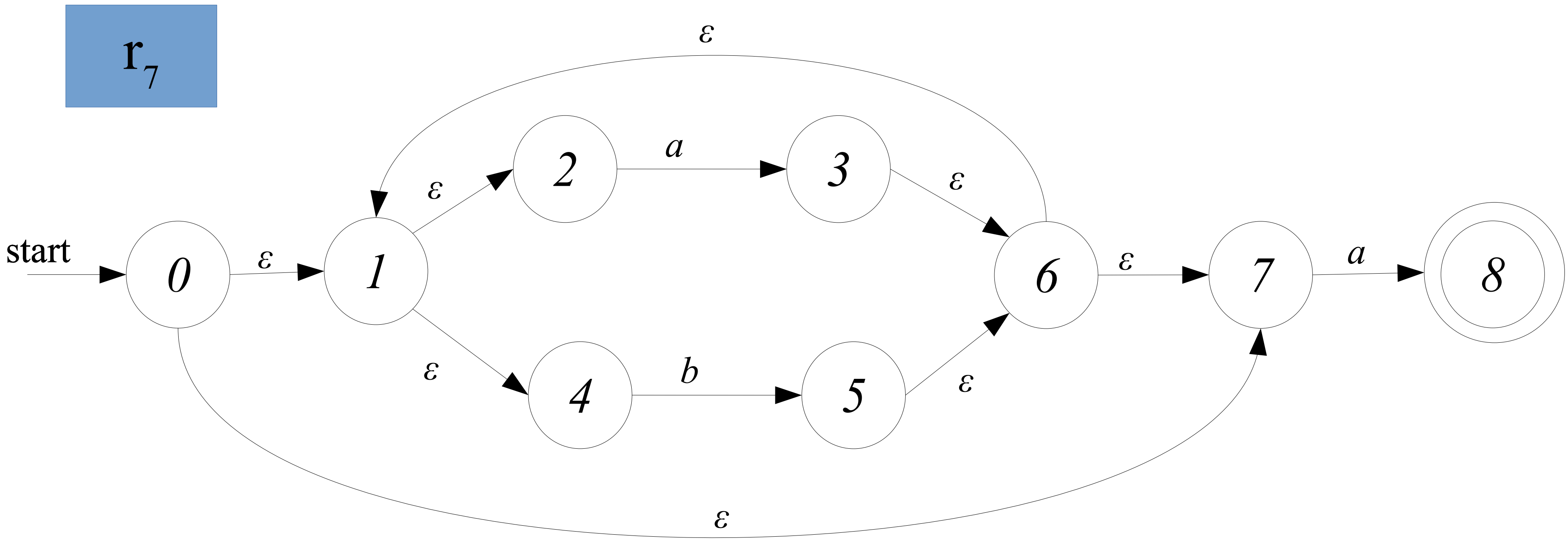
r_5



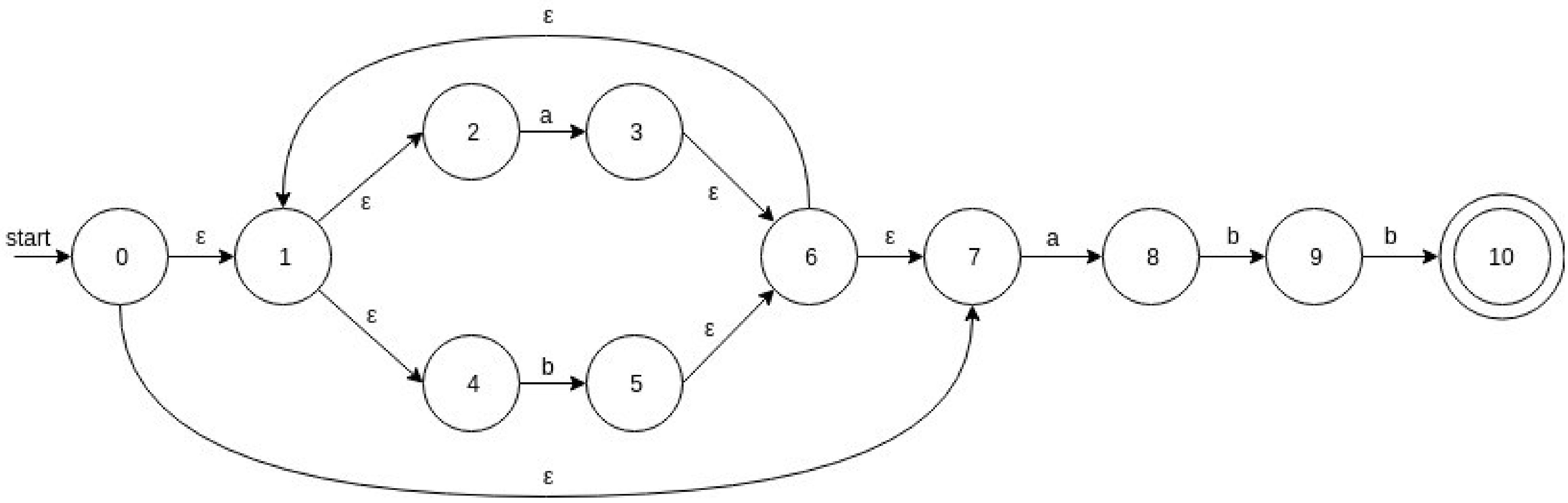
r_6



Example



Example

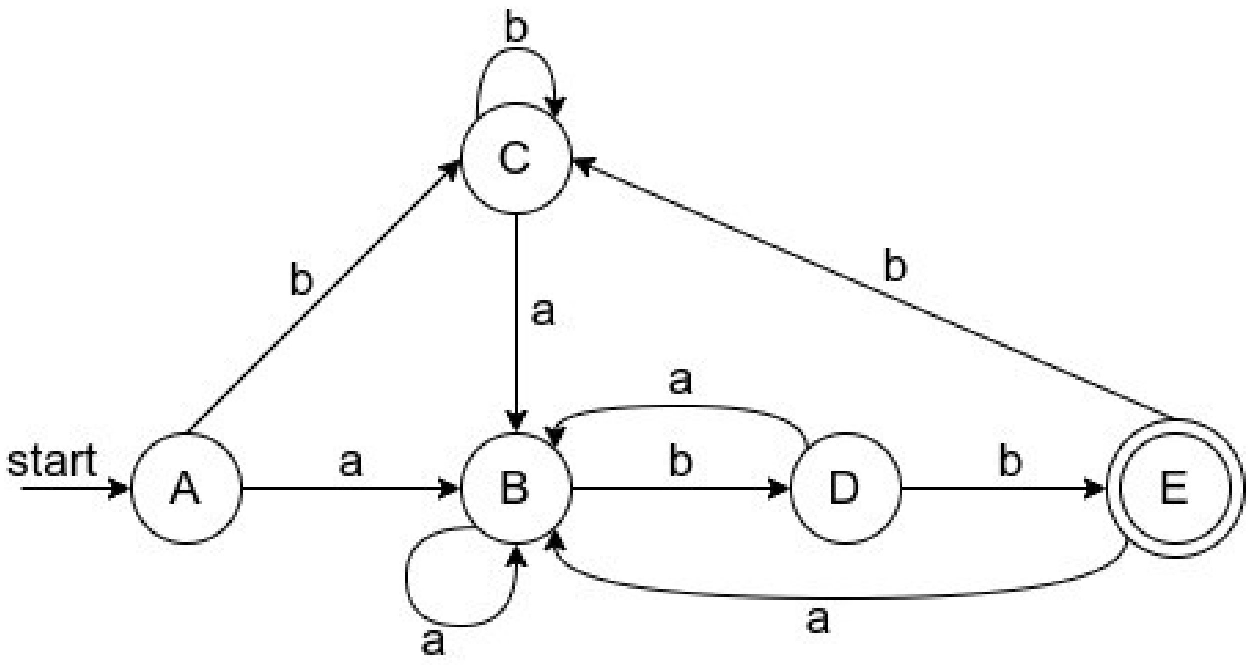


Skipped

- From section 3.7.5 to section 3.9.5 are skipped.

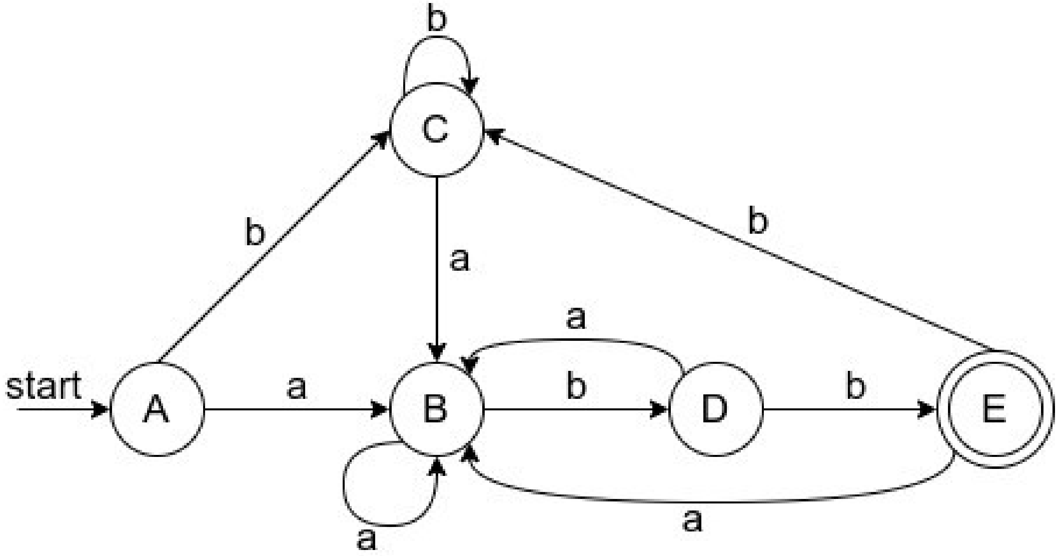
Minimizing the Number of States of a DFA

- There maybe more than DFA accepting the same language, with different no. of states.
- In building a lexical analyzer simulating a DFA, fewer no. of state is preferable.
- Ex: States A & C are equivalent (same moves to B&C on a&b respectively).
- There is always a unique minimum state DFA for any regular language, and it can be constructed from any other DFA for the same language by grouping equivalent states.



Minimizing the Number of States of a DFA (cont.)

- We say that string *x* **distinguishes** state *s* from state *t* if exactly one of the states reached from *s* and *t* by following the path with label *x* is an accepting state. State *s* is **distinguishable** from state *t* if there is some string that distinguishes them.
- **Ex**: ε **distinguishes** any accepting state from any non-accepting state. String *bb* **distinguishes** state A from state B.
- The state-minimization algorithm works by partitioning the states of a DFA into groups of states that cannot be distinguished.



Minimizing the Number of States of a DFA Algorithm

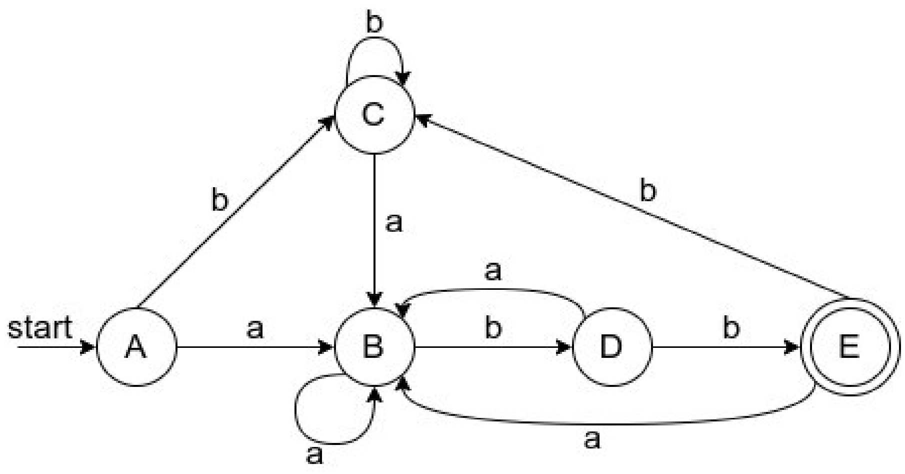
- **INPUT**: A DFA *D* with set of states *S*, input alphabet Σ, start state s_0, and set of accepting states *F*.
- **OUTPUT**: A DFA *D'* accepting the same language as *D* and having as few states as possible.
- **METHOD**:

1) Start with an initial partition Π with two groups, *F* and *S-F*, the accepting and nonaccepting states of *D*.
2) Apply the following procedure to construct a new partition Π_{new}.

```
initially, let $\Pi_{new} = \Pi$;
for ( each group $G$ of $\Pi$) {
  partition $G$ into subgroups such that two states $s$ and $t$
    are in the same subgroup if and only if for all
    input symbols $a$, states $s$ and $t$ have transitions on $a$
    to states in the same group of $\Pi$;
  /* at worst, a state will be in a subgroup by itself */
  replace $G$ in $\Pi_{new}$ by the set of all subgroups formed;
}
```

3) If $\Pi_{new} = \Pi$, let $\Pi_{final} = \Pi$ and continue with step (4). Otherwise, repeat step (2) with Π_{new} in place of Π.
4) Choose one state in each group of Π_{final} as the *representative* for that group. The representatives will be the states of the minimum-state DFA *D'* . The other components of *D'* are constructed as follows:
    a) The start state of *D'* is the representative of the group containing the start state of *D* .
    b) The accepting states of *D'* are the representatives of those groups that contain an accepting state of *D* . Note that each group contains either only accepting states, or only nonaccepting states, because we started by separating those two classes of states, and the prev. procedure always forms new groups that are subgroups of prev. constructed groups.
    c) Let *s* be the representative of some group *G* of Π_{final} , and let the transition of *D* from *s* on input *a* be to state *t*. Let *r* be the representative of *t* 's group *H*. Then in *D'* , there is a transition from *s* to *r* on input *a*. Note that in *D* , every state in group *G* must go to some state of group *H* on input *a*, or else group *G* would have been split.

Example



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
{1,2,3,5,6,7,10}	E	B	C

- Π_0 = {A,B,C,D} {E}
  - Non-accepting and accepting.
- Π_1={A,B,C} {D} {E}
  - They split on input *b*.
- Π_2={A,C} {B} {D} {E}
  - They split on input *b*.
- No more splits. $\Pi_{final} = \Pi_2$

State	a	b
A	B	A
B	B	D
D	B	E
E	B	A

Example

	x	y
A	B	F
B	J	C
C	A	C
D	C	J
E	K	F
F	C	J
J	J	E
K	J	C

- Π_0 = {A, B, D, E, F, J, K} {C}
- Π_1={A, E, J} {B, K} {D, F} {C}
- Π_2={A,E} {J} {B, K} {D, F} {C}
- No more splits. $\Pi_{final} = \Pi_2$

	x	y
A	B	F
J	J	E
B	J	C
D	C	J
C	A	C

Skipped

- Sections 3.9.7 & 3.9.8 are skipped.