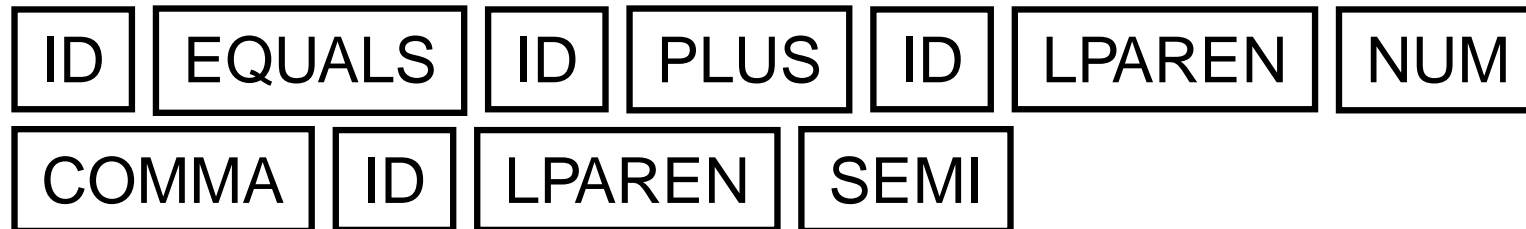


Lexical Analysis (Scanning)

Lexical Analysis (Scanning)

Translates a stream of characters to a stream of tokens

`f o o _ = _ a + _ bar(2, _ q);`



Token	Lexemes	Pattern
EQUALS	=	an equals sign
PLUS	+	a plus sign
ID	a foo bar	letter followed by letters or digits
NUM	0 42	one or more digits

Lexical Analysis

Goal: simplify the job of the parser.

Scanners are usually much faster than parsers.

Discard as many irrelevant details as possible (e.g., whitespace, comments).

Parser does not care that the the identifier is
“supercalifragilisticexpialidocious.”

Parser rules are only concerned with tokens.

Describing Tokens

Alphabet: A finite set of symbols

Examples: $\{ 0, 1 \}$, $\{ A, B, C, \dots, Z \}$, ASCII, Unicode

String: A finite sequence of symbols from an alphabet

Examples: ϵ (the empty string), Stephen, $\alpha\beta\gamma$

Language: A set of strings over an alphabet

Examples: \emptyset (the empty language), $\{ 1, 11, 111, 1111 \}$,
all English words, strings that start with a letter followed by
any sequence of letters and digits

Operations on Languages

Let $L = \{ \epsilon, \text{wo} \}$, $M = \{ \text{man}, \text{men} \}$

Concatenation: Strings from one followed by the other

$LM = \{ \text{man}, \text{men}, \text{woman}, \text{women} \}$

Union: All strings from each language

$L \cup M = \{ \epsilon, \text{wo}, \text{man}, \text{men} \}$

Kleene Closure: Zero or more concatenations

$M^* = \{ \epsilon, M, MM, MMM, \dots \} =$
 $\{ \epsilon, \text{man}, \text{men}, \text{manman}, \text{manmen}, \text{menman}, \text{menmen},$
 $\text{manmanman}, \text{manmanmen}, \text{manmenman}, \dots \}$

Regular Expressions over an Alphabet Σ

A standard way to express languages for tokens.

1. ϵ is a regular expression that denotes $\{\epsilon\}$
2. If $a \in \Sigma$, a is an RE that denotes $\{a\}$
3. If r and s denote languages $L(r)$ and $L(s)$,
 - $(r)|(s)$ denotes $L(r) \cup L(s)$
 - $(r)(s)$ denotes $\{tu : t \in L(r), u \in L(s)\}$
 - $(r)^*$ denotes $\cup_{i=0}^{\infty} L^i$ ($L^0 = \{\epsilon\}$ and $L^i = LL^{i-1}$)

Regular Expression Examples

$$\Sigma = \{a, b\}$$

RE	Language
$a b$	$\{a, b\}$
$(a b)(a b)$	$\{aa, ab, ba, bb\}$
a^*	$\{\epsilon, a, aa, aaa, aaaa, \dots\}$
$(a b)^*$	$\{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, \dots\}$
$a a^*b$	$\{a, b, ab, aab, aaab, aaaab, \dots\}$

Specifying Tokens with REs

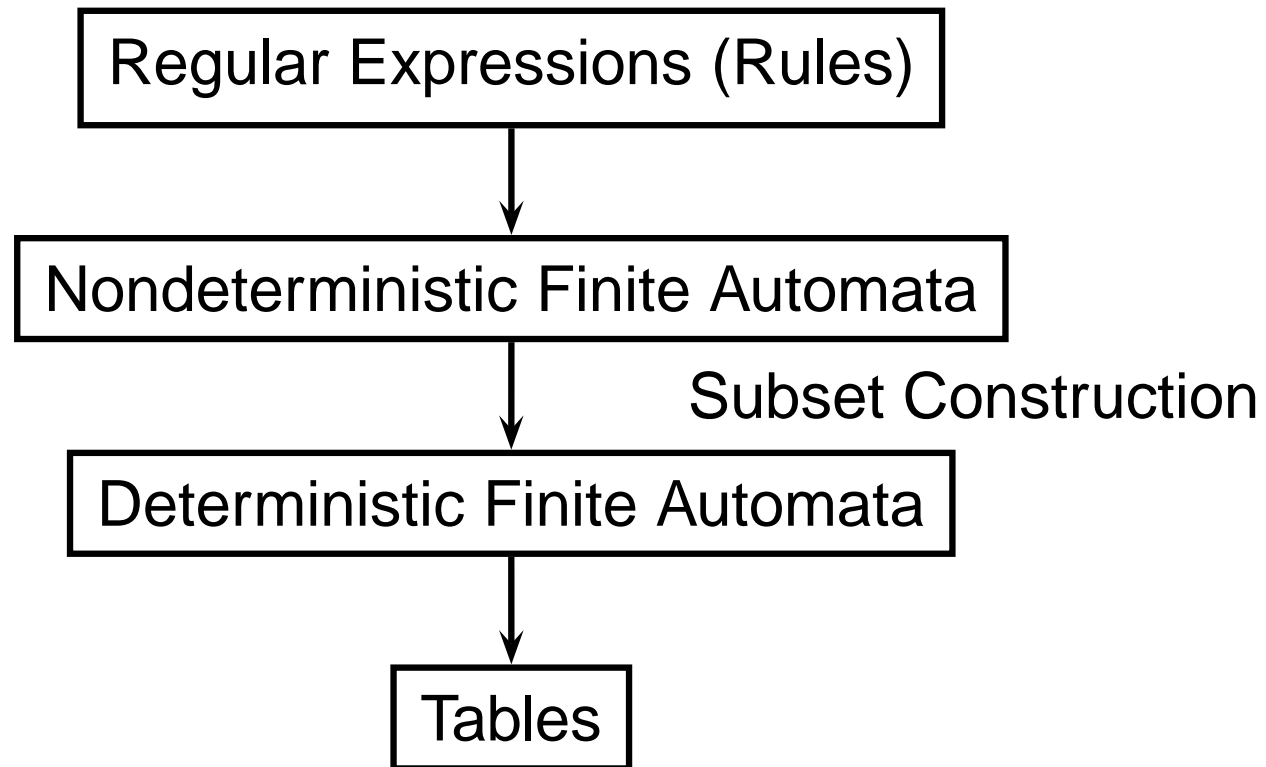
Typical choice: $\Sigma = \text{ASCII characters, i.e.,}$
 $\{_, !, ", \#, \$, \dots, 0, 1, \dots, 9, \dots, A, \dots, Z, \dots, \sim\}$

letters: $A|B|\dots|Z|a|\dots|z$

digits: $0|1|\dots|9$

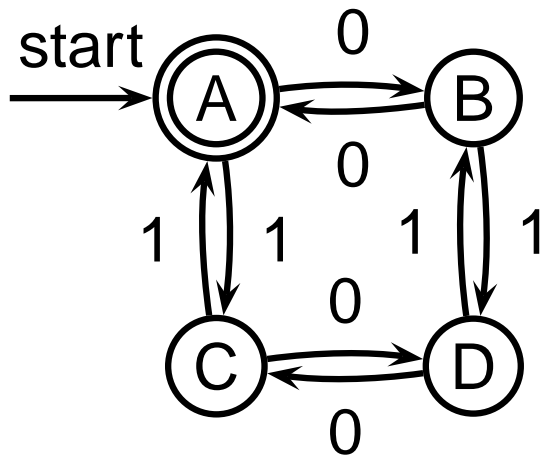
identifier: $\text{letter} (\text{letter} | \text{digit})^*$

Implementing Scanners Automatically



Nondeterministic Finite Automata

“All strings containing an even number of 0’s and 1’s”



1. Set of states S : $\{ \textcircled{\textcircled{A}}, \textcircled{B}, \textcircled{C}, \textcircled{D} \}$

2. Set of input symbols Σ : $\{0, 1\}$

3. Transition function $\sigma : S \times \Sigma_{\epsilon} \rightarrow 2^S$

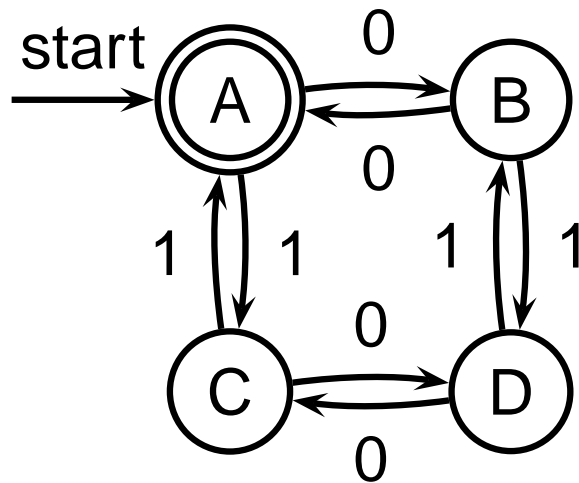
state	ϵ	0	1
A	—	{B}	{C}
B	—	{A}	{D}
C	—	{D}	{A}
D	—	{C}	{B}

4. Start state s_0 : $\textcircled{\textcircled{A}}$

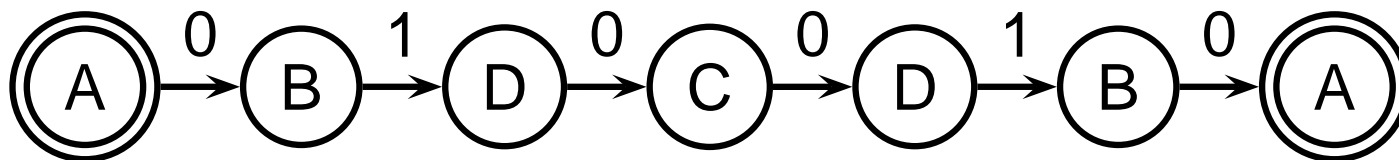
5. Set of accepting states F : $\{ \textcircled{\textcircled{A}} \}$

The Language induced by an NFA

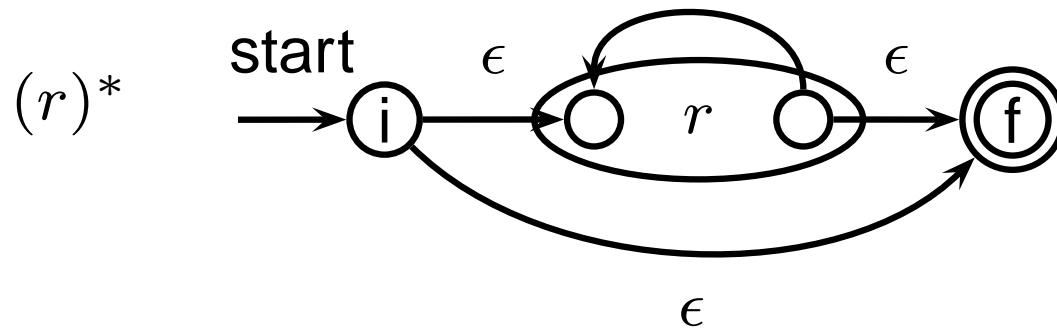
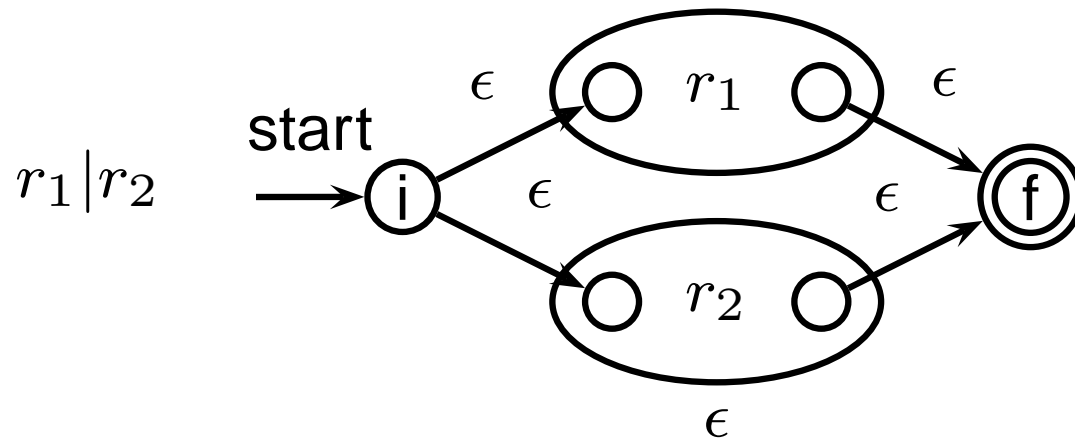
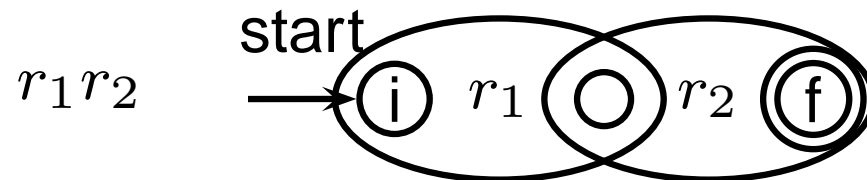
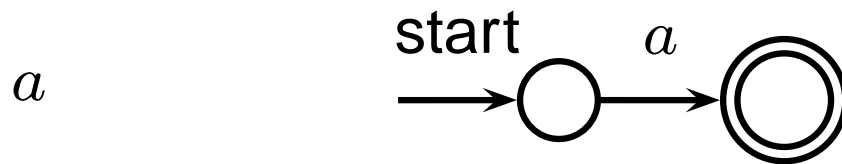
An NFA accepts an input string x iff there is a path from the start state to an accepting state that “spells out” x .



Show that the string
“010010” is accepted.

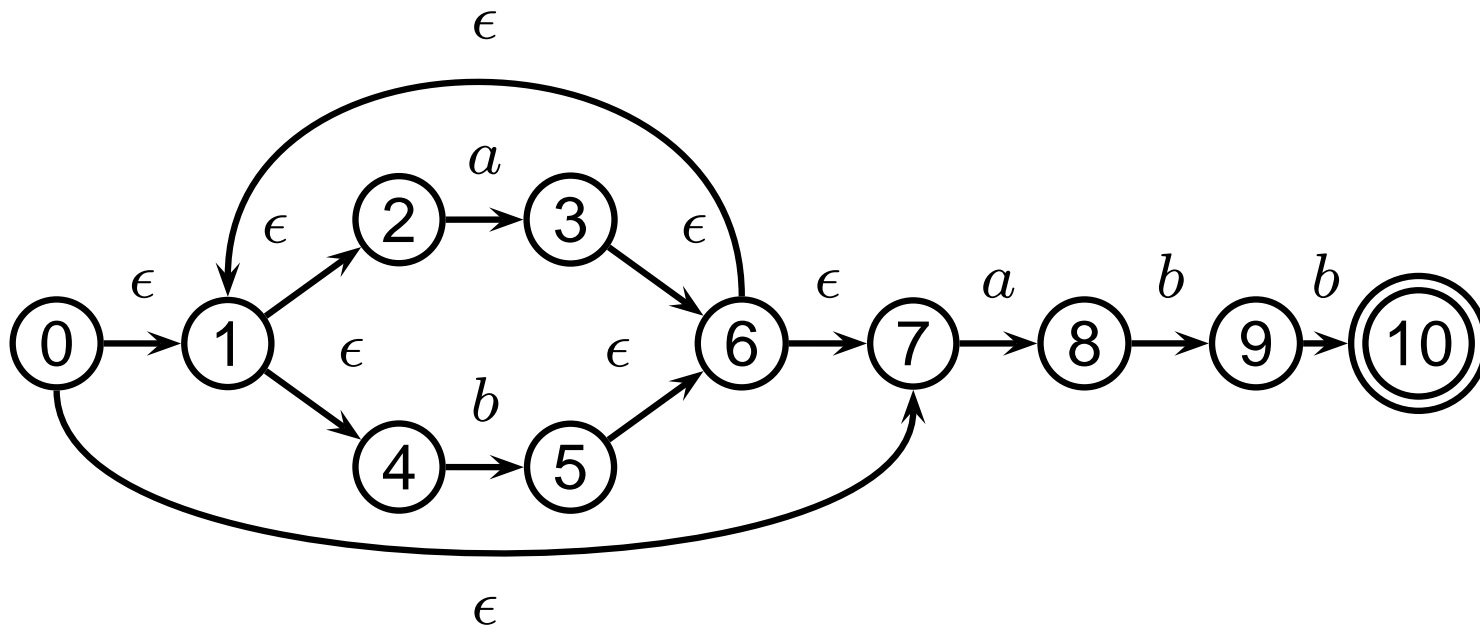


Translating REs into NFAs

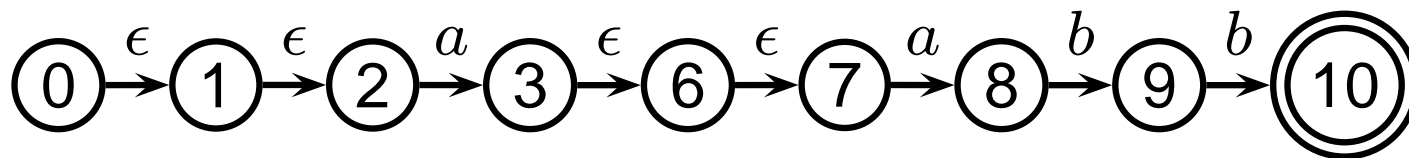


Translating REs into NFAs

Example: translate $(a|b)^*abb$ into an NFA



Show that the string “ $aabb$ ” is accepted.



Simulating NFAs

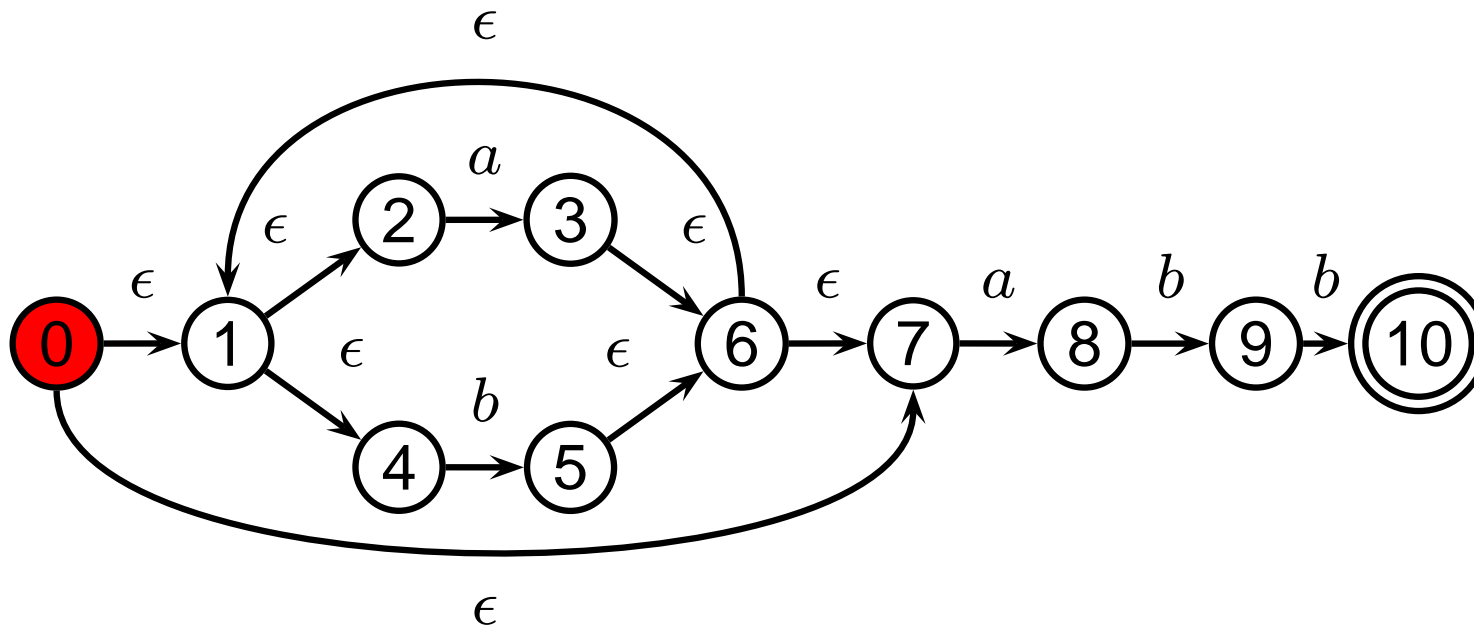
Problem: you must follow the “right” arcs to show that a string is accepted. How do you know which arc is right?

Solution: follow them all and sort it out later.

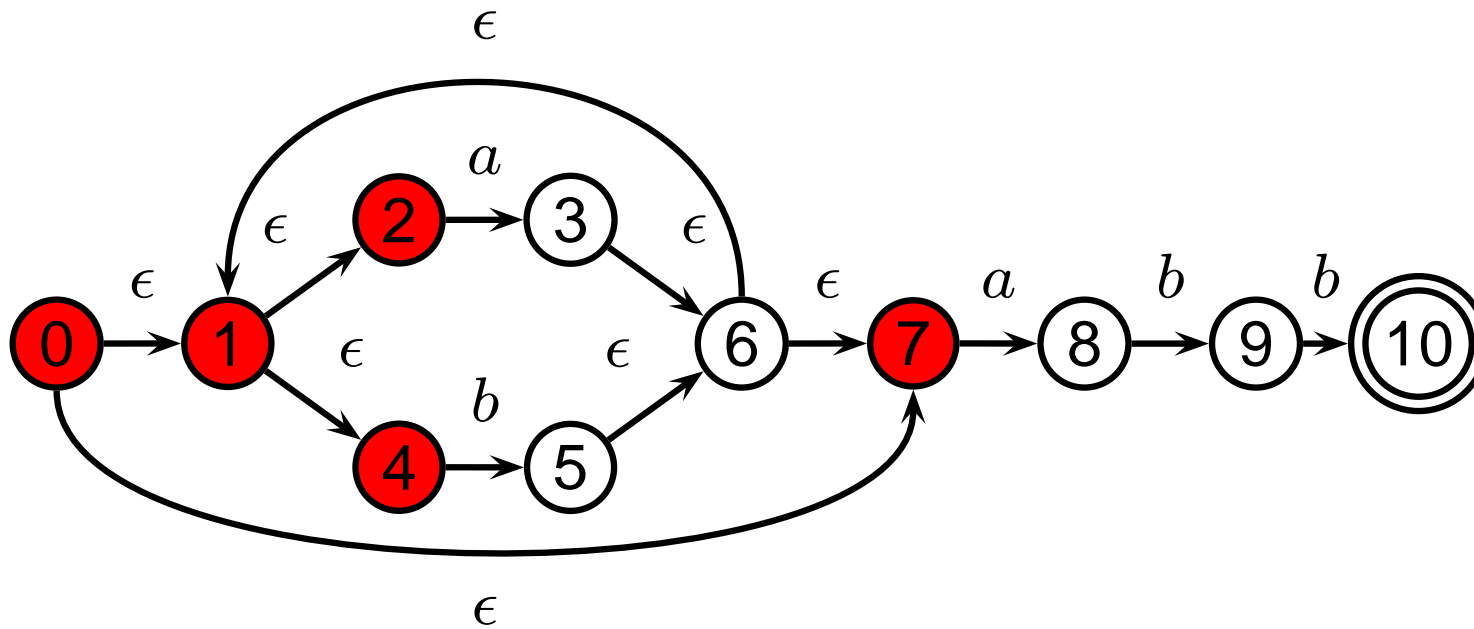
“Two-stack” NFA simulation algorithm:

1. Initial states: the ϵ -closure of the start state
2. For each character c ,
 - New states: follow all transitions labeled c
 - Form the ϵ -closure of the current states
3. Accept if any final state is accepting

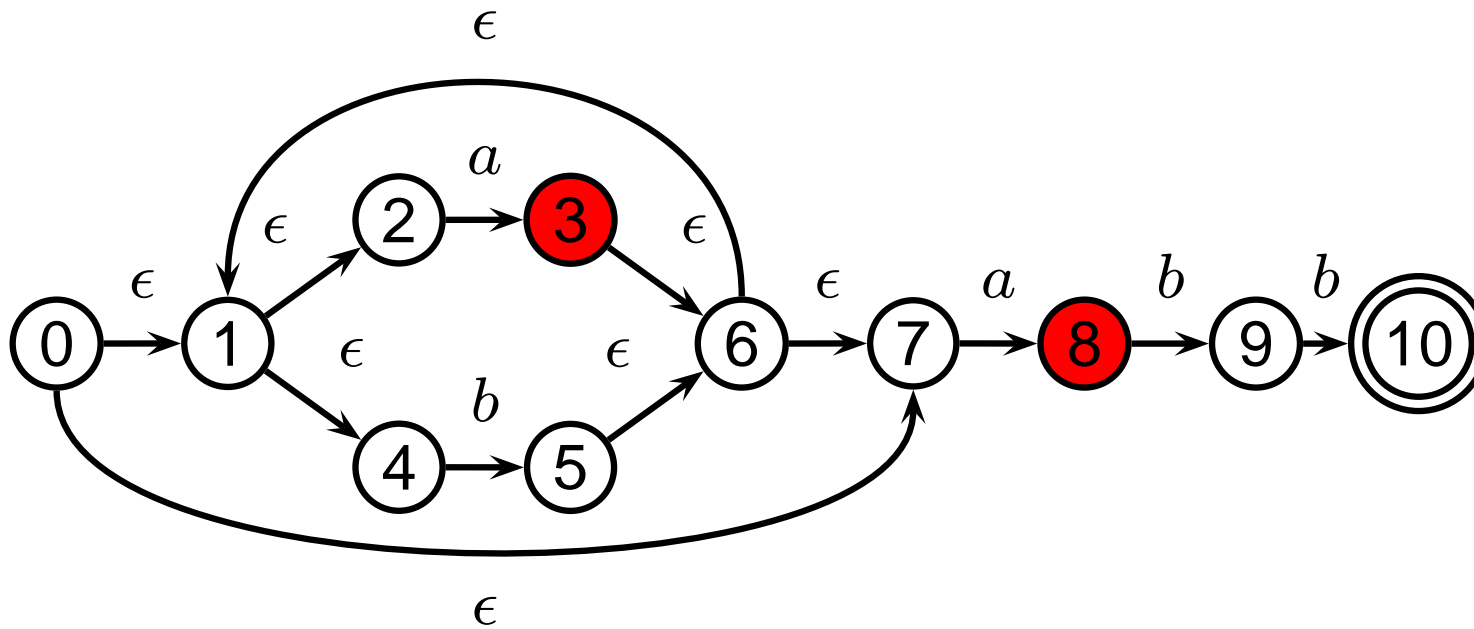
Simulating an NFA: $\cdot aabb$, Start



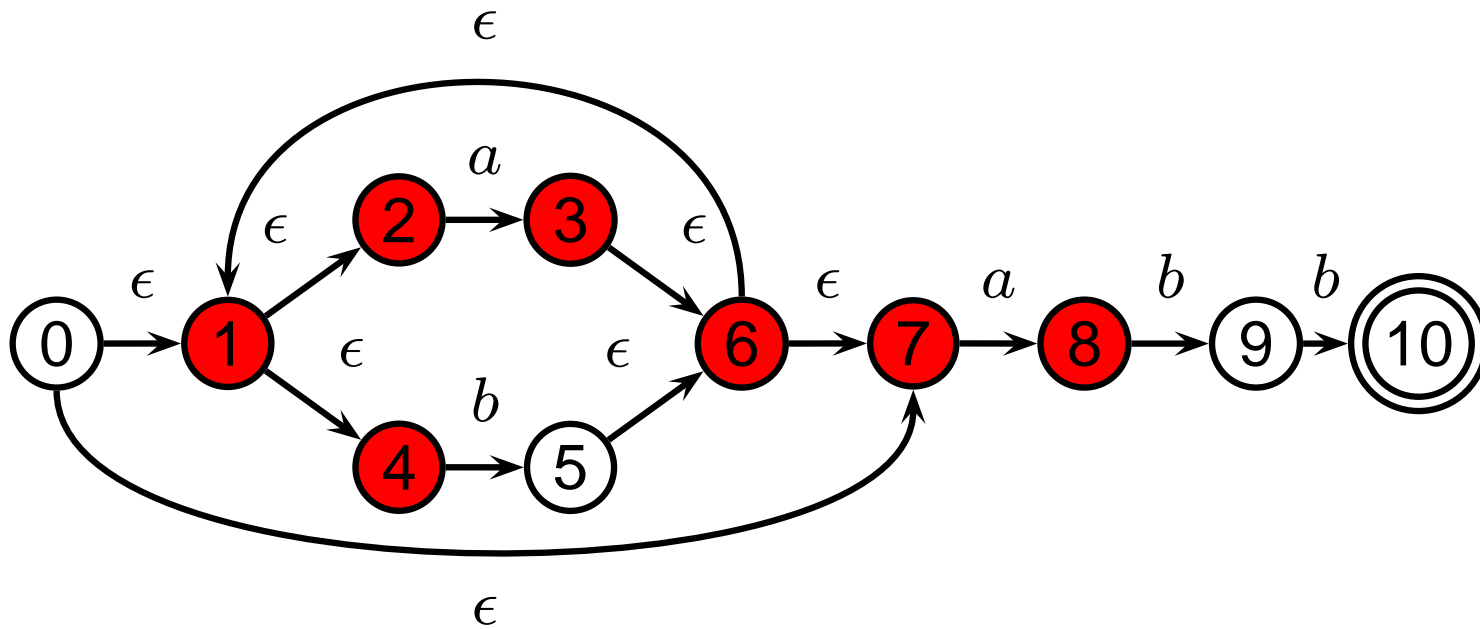
Simulating an NFA: $\cdot aabb$, ϵ -closure



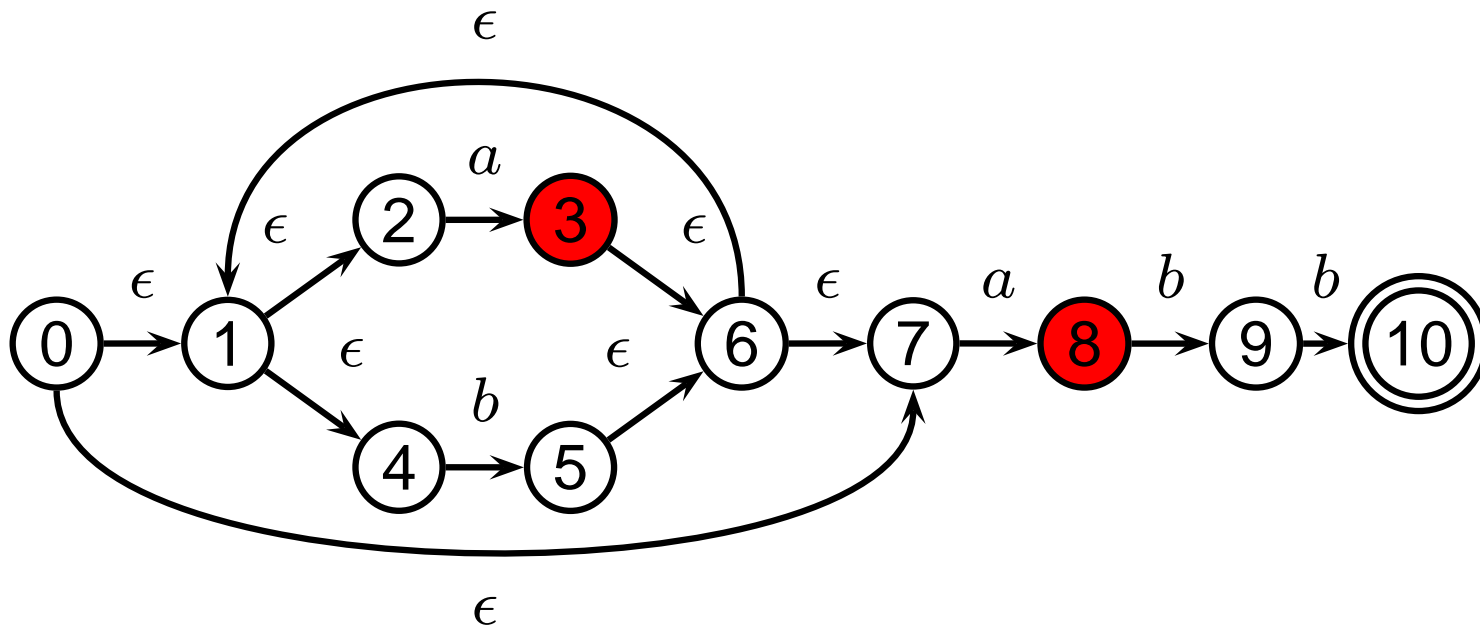
Simulating an NFA: $a \cdot abb$



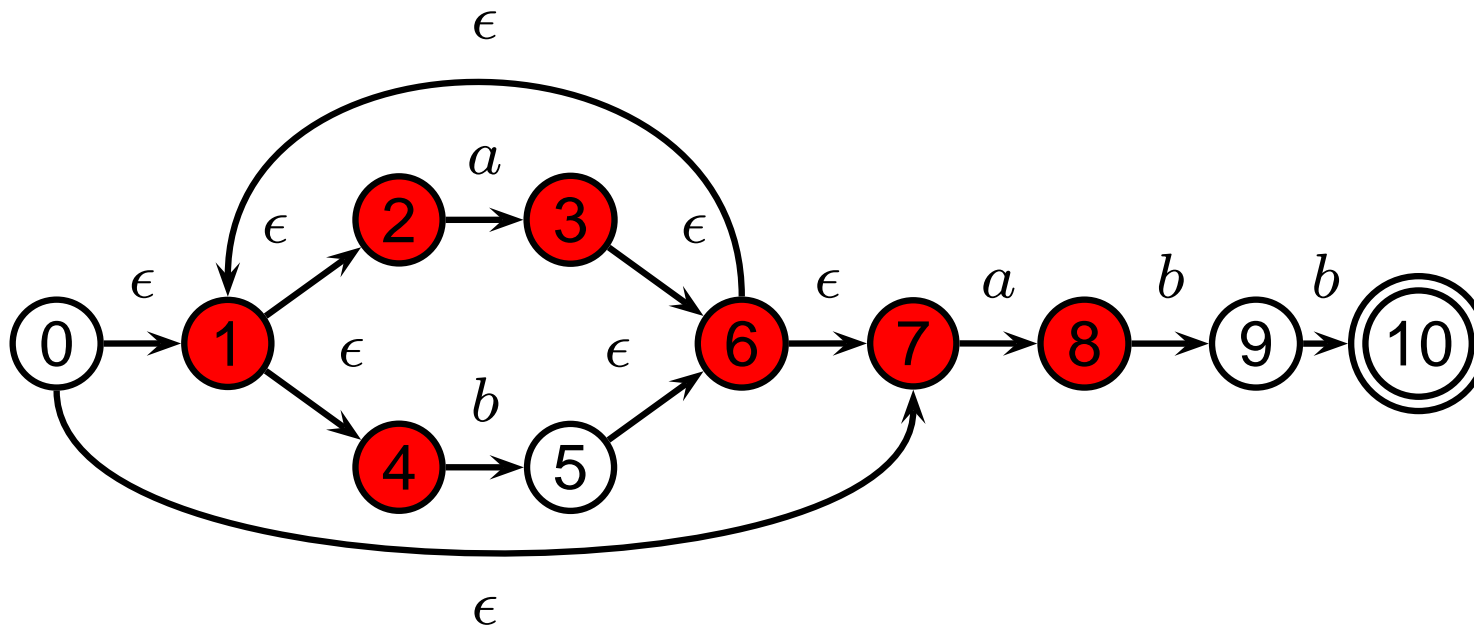
Simulating an NFA: $a \cdot abb$, ϵ -closure



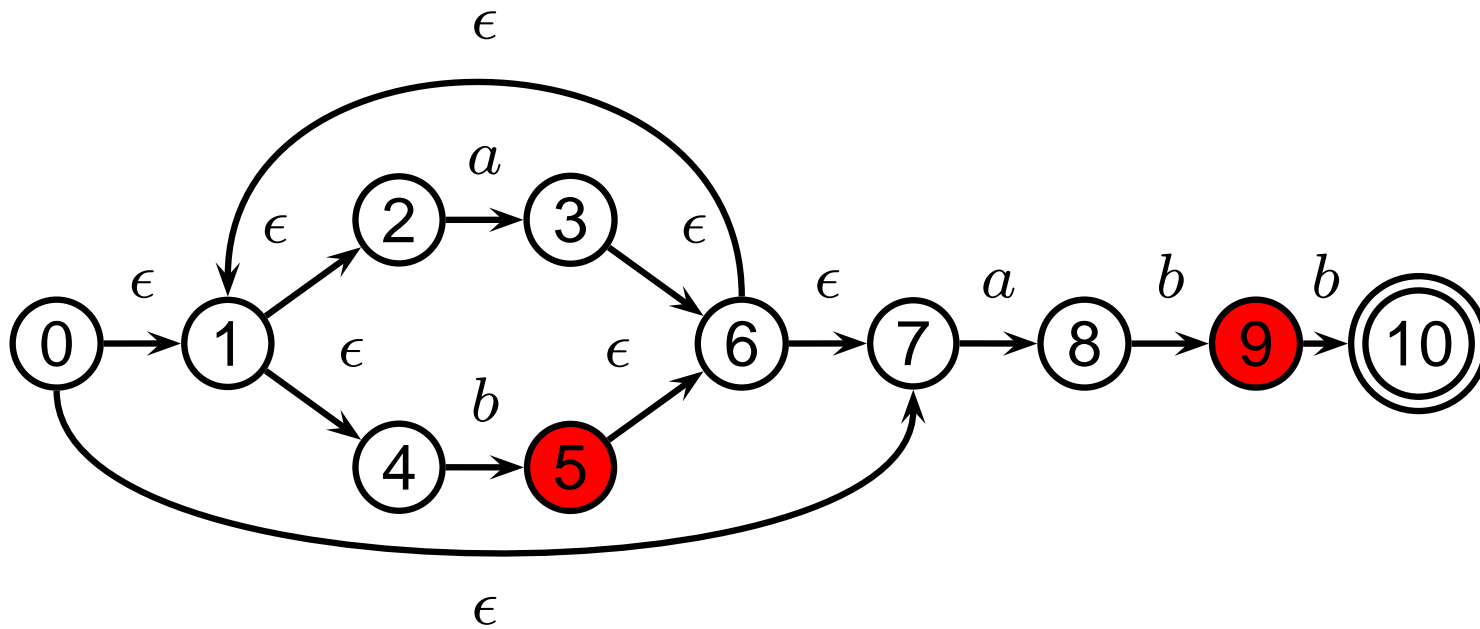
Simulating an NFA: $aa \cdot bb$



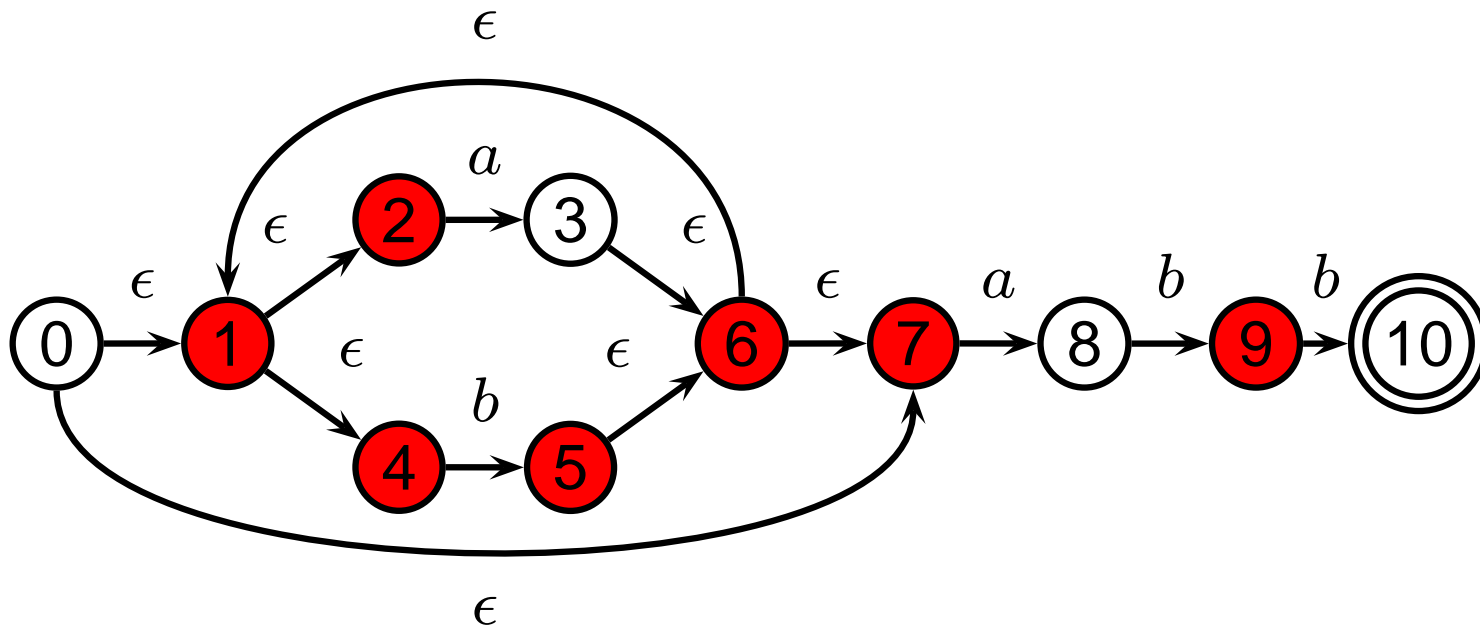
Simulating an NFA: $aa \cdot bb$, ϵ -closure



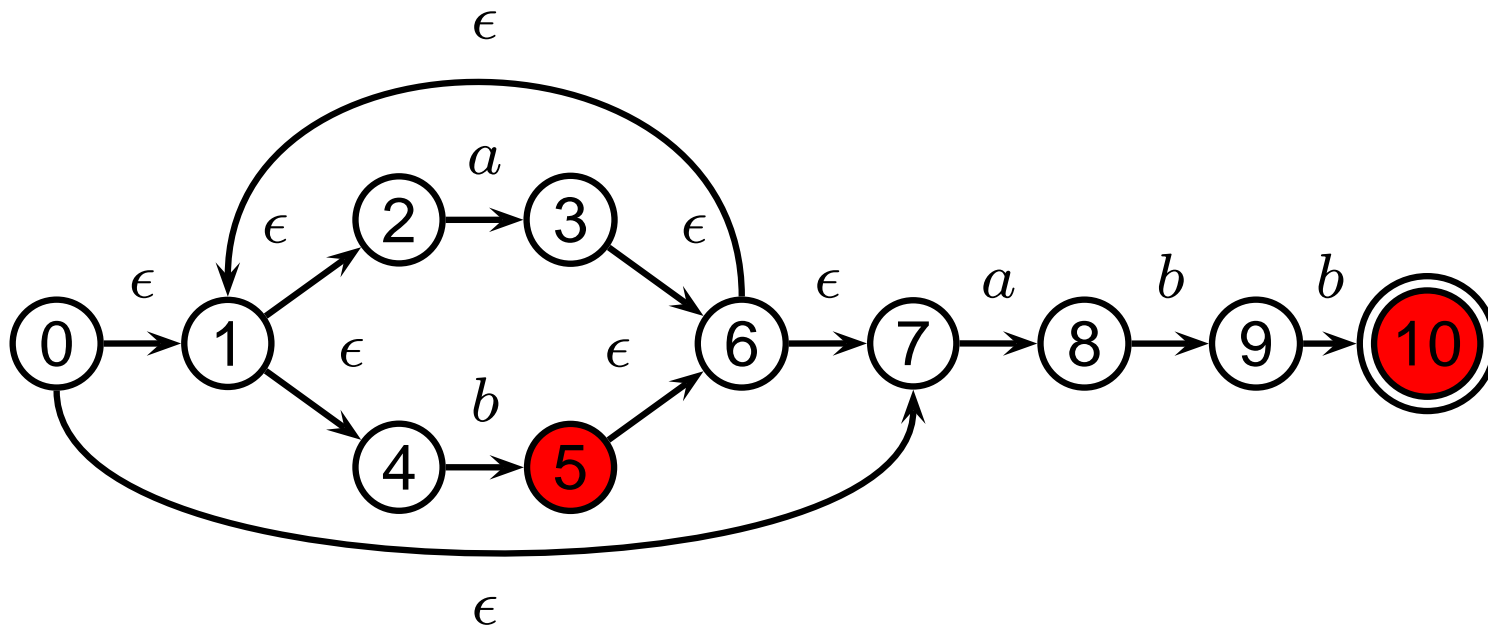
Simulating an NFA: $aab \cdot b$



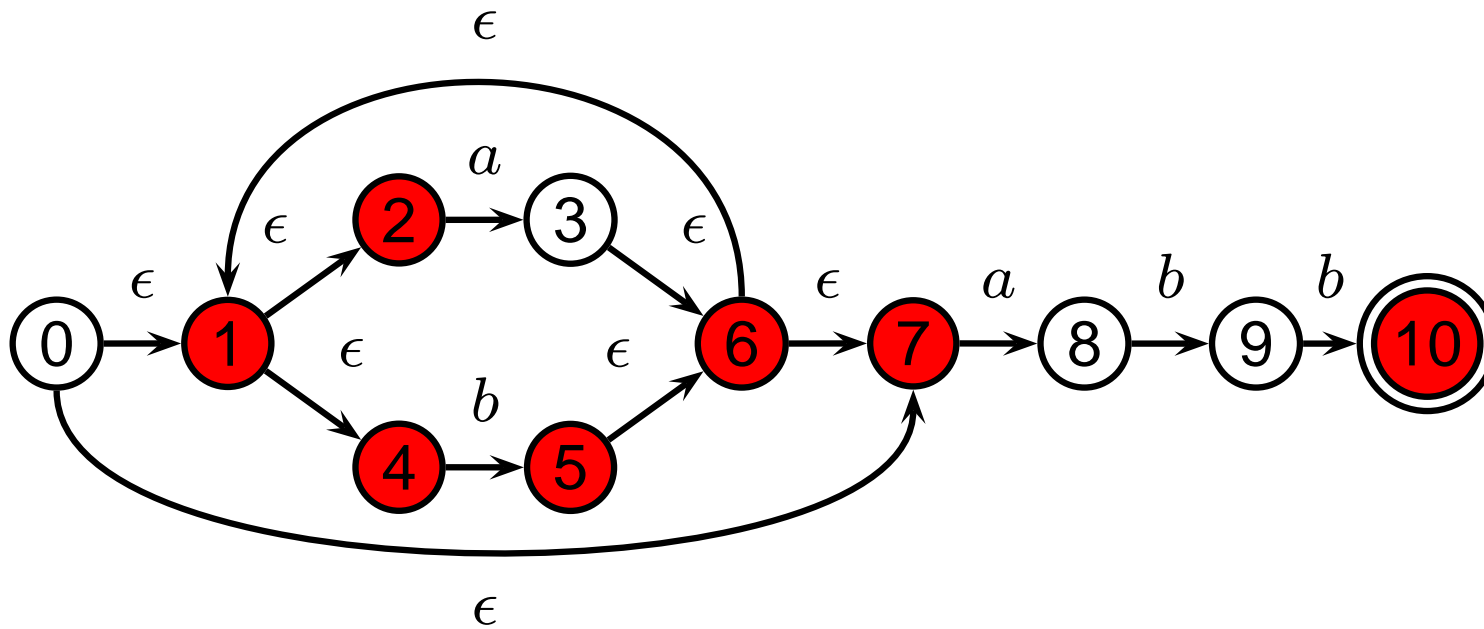
Simulating an NFA: $aab \cdot b$, ϵ -closure



Simulating an NFA: *aabb*.



Simulating an NFA: *aabb.*, Done



Deterministic Finite Automata

Restricted form of NFAs:

- No state has a transition on ϵ
- For each state s and symbol a , there is at most one edge labeled a leaving s .

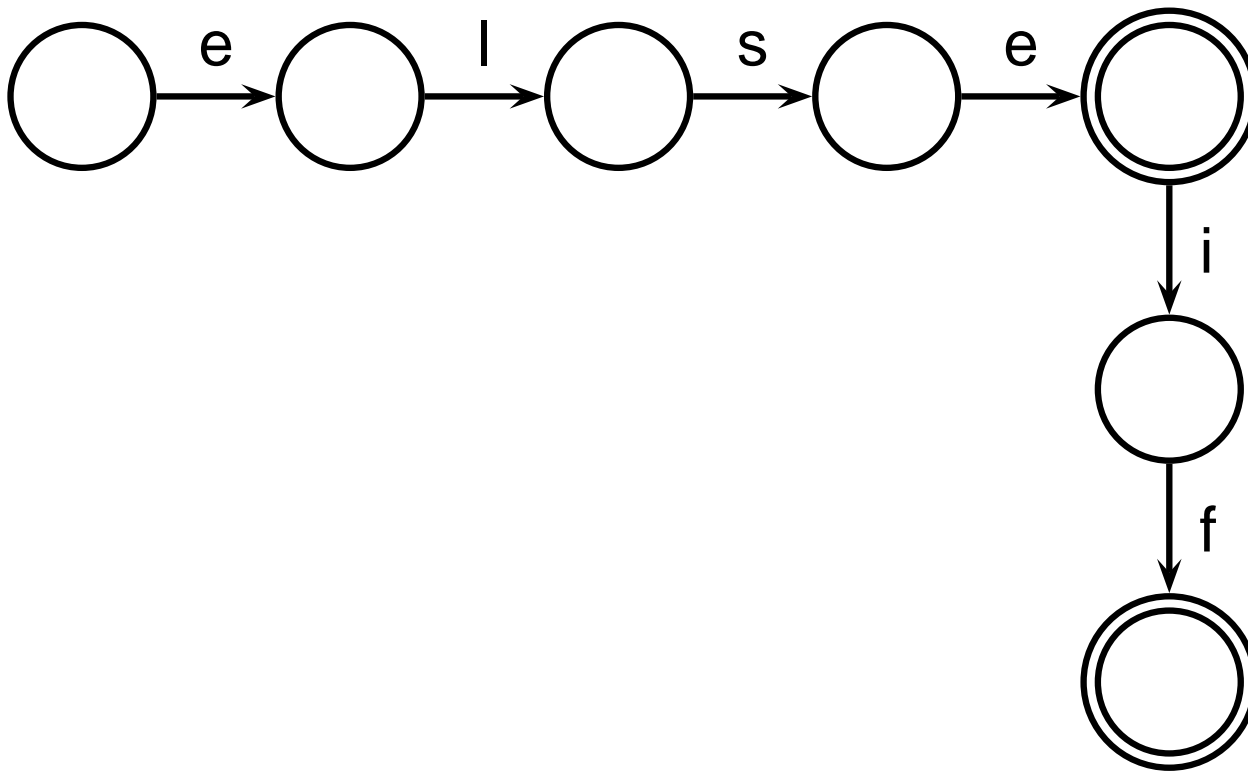
Differs subtly from the definition used in COMS W3261
(Sipser, *Introduction to the Theory of Computation*)

Very easy to check acceptance: simulate by maintaining current state. Accept if you end up on an accepting state. Reject if you end on a non-accepting state or if there is no transition from the current state for the next symbol.

Deterministic Finite Automata

ELSE: "else" ;

ELSEIF: "elseif" ;

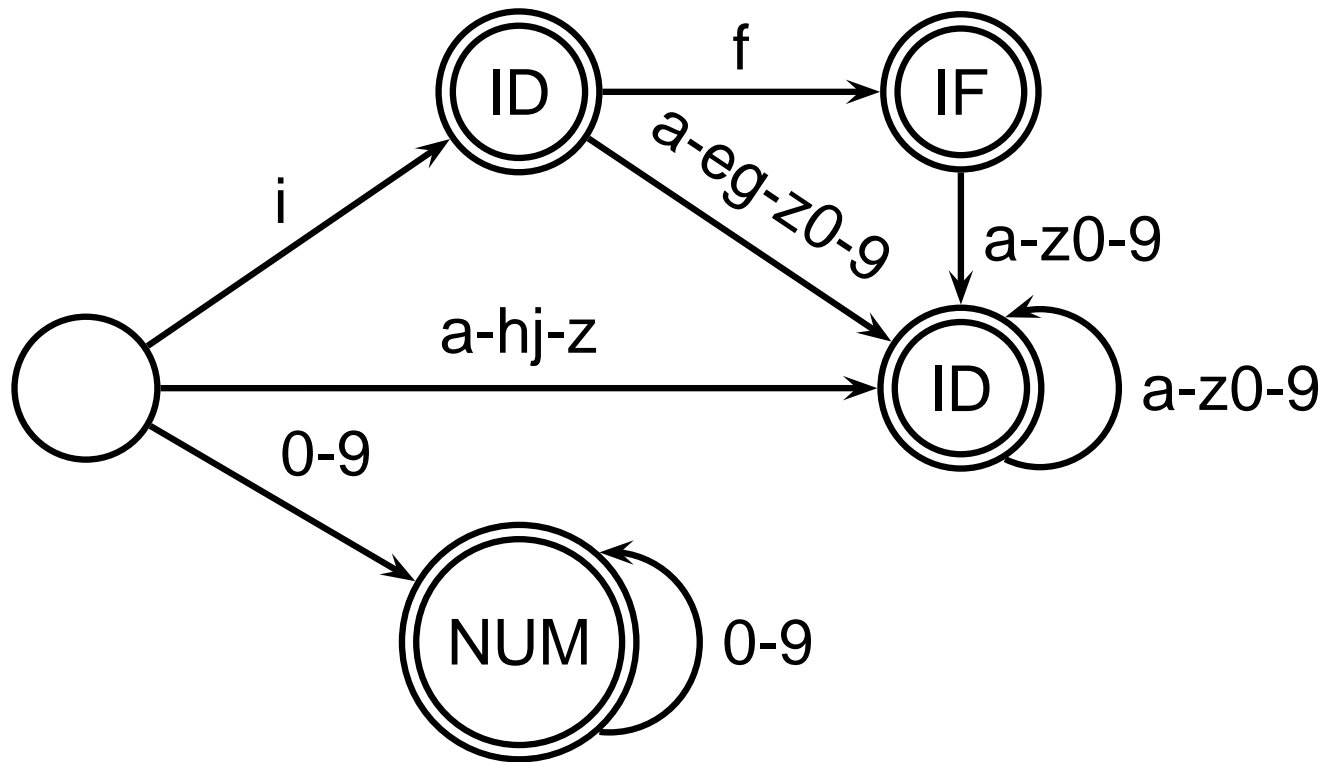


Deterministic Finite Automata

IF: "if" ;

ID: 'a'..'z' ('a'..'z' | '0'..'9')* ;

NUM: ('0'..'9')+ ;



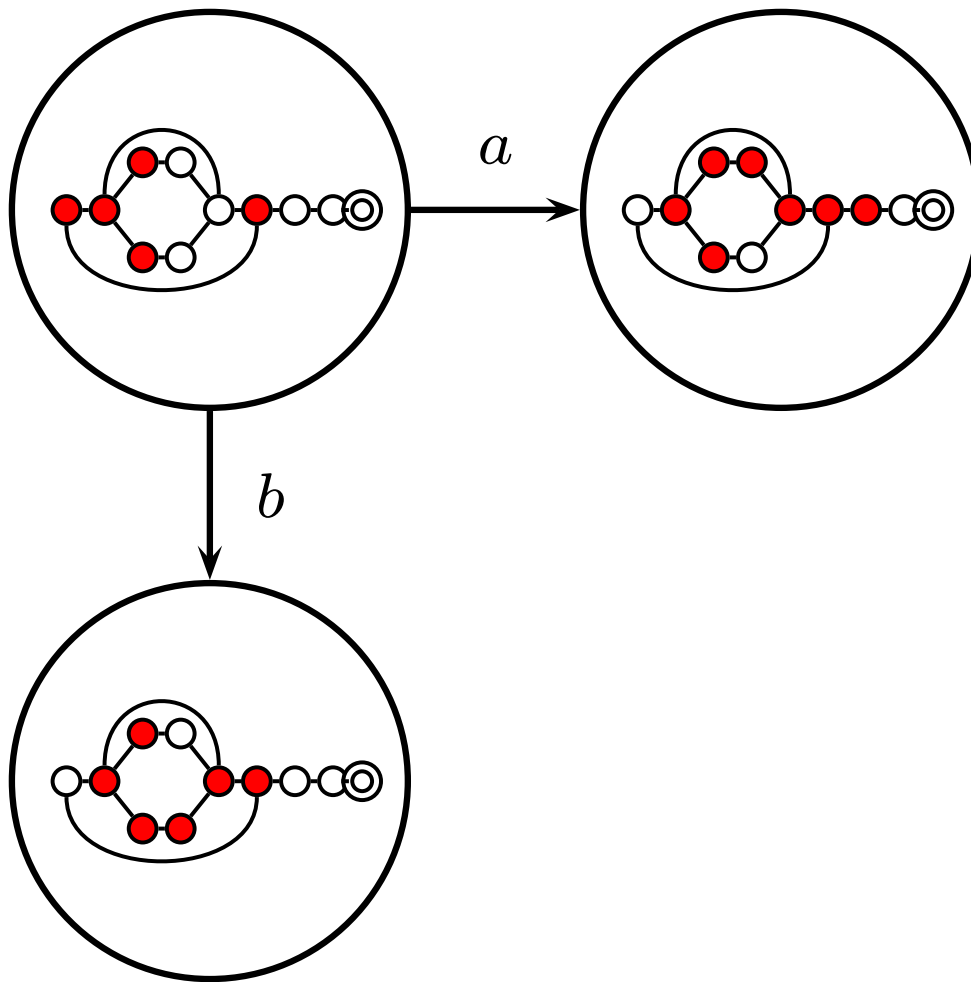
Building a DFA from an NFA

Subset construction algorithm

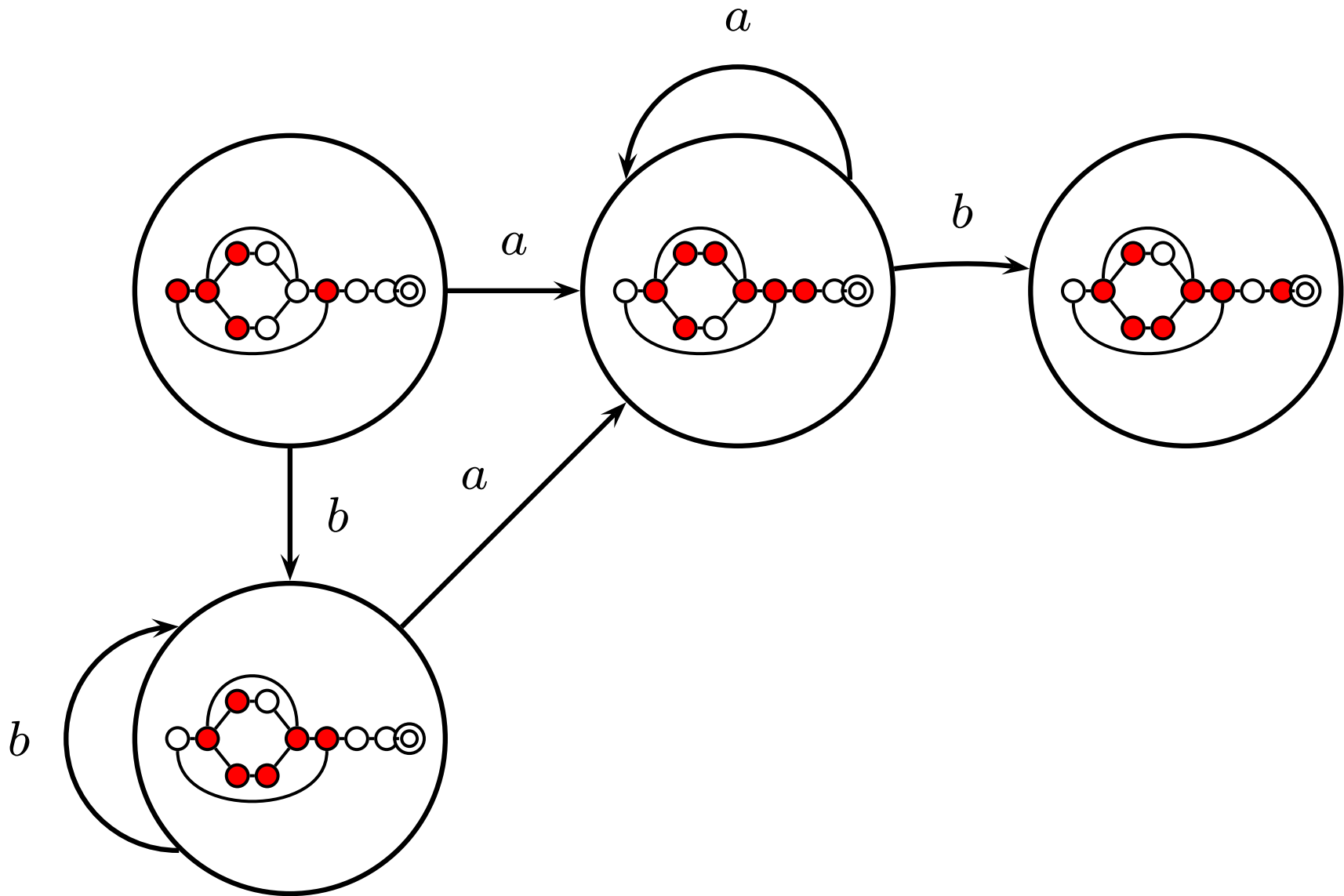
Simulate the NFA for all possible inputs and track the states that appear.

Each unique state during simulation becomes a state in the DFA.

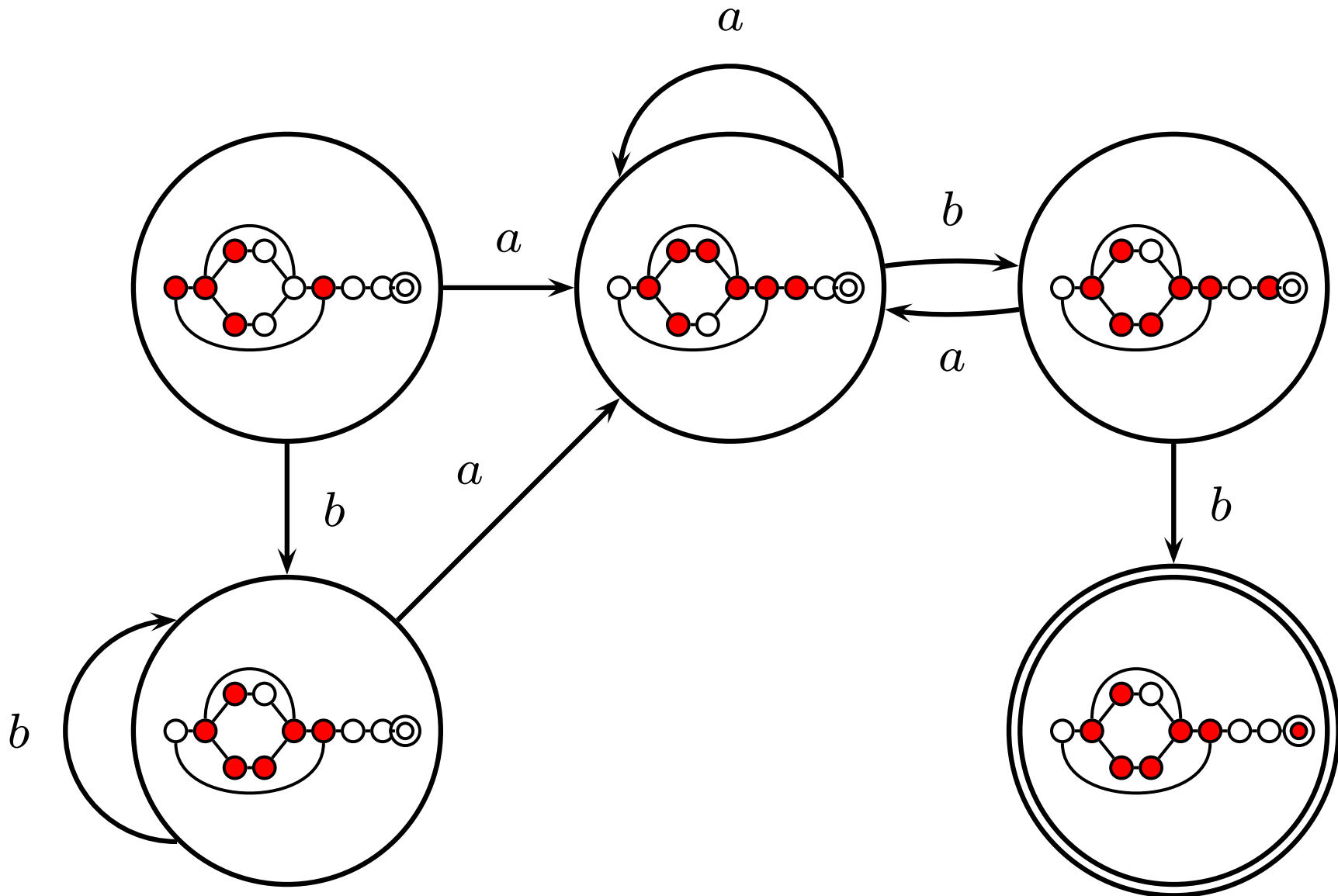
Subset construction for $(a|b)^*abb$ (1)



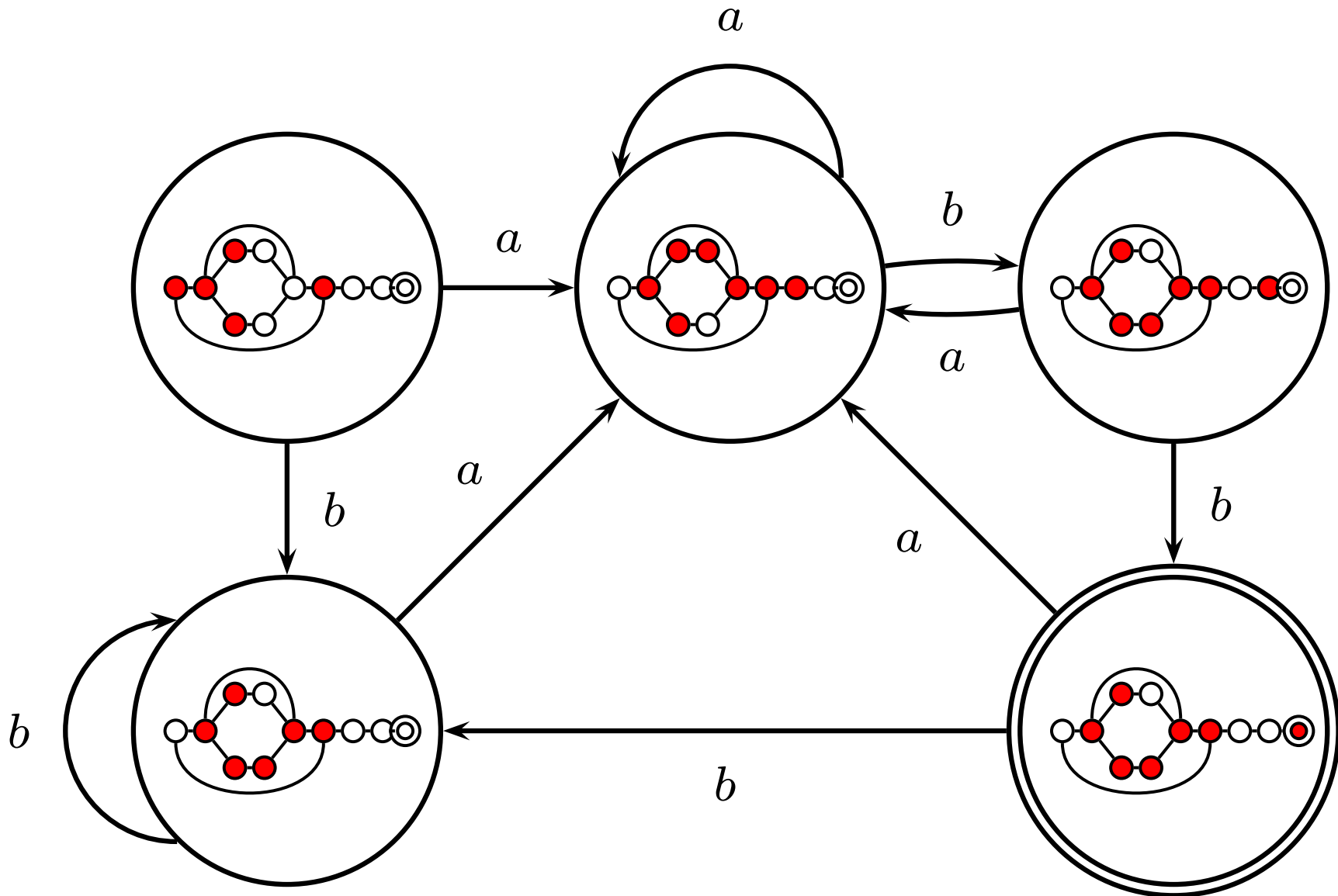
Subset construction for $(a|b)^*abb$ (2)



Subset construction for $(a|b)^*abb$ (3)



Subset construction for $(a|b)^*abb$ (4)



Subset Construction

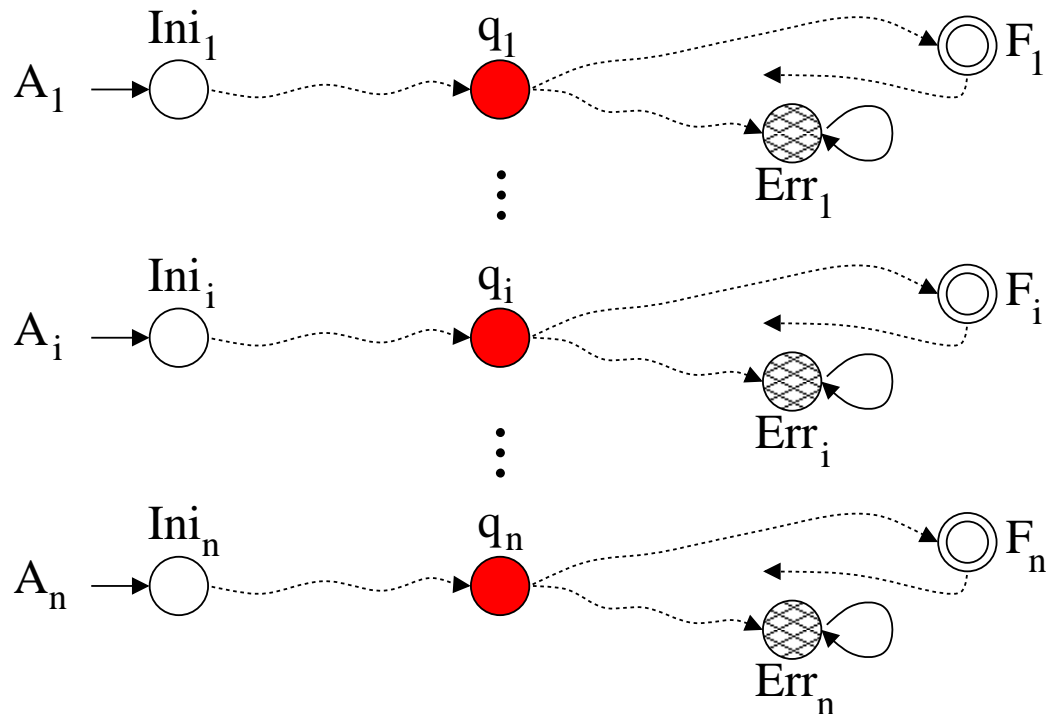
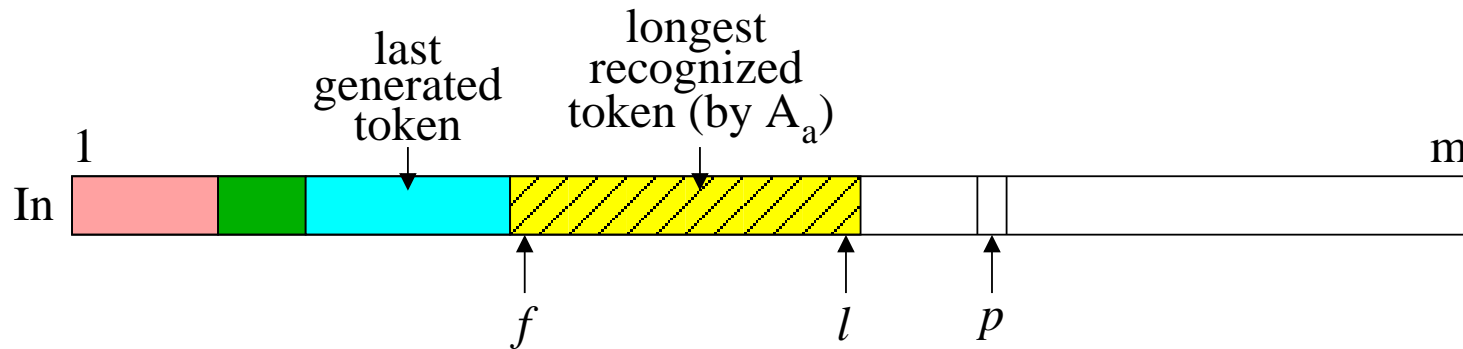
An DFA can be exponentially larger than the corresponding NFA.

n states versus 2^n

Tools often try to strike a balance between the two representations.

ANTLR uses a different technique.

An algorithm for scanning



```

 $p = f = 1; l = 0; \forall i : q_i = \text{Ini}_i;$ 
while  $p \leq m$  do
     $\forall i : q_i = \delta(q_i, \text{In}[p]); p = p + 1; //$ State transition
    if  $\exists i : q_i \in F_i$  then
         $l = p - 1; a = \text{smallest } i \text{ such that } q_i \in F_i;$ 
    elseif  $\forall i : q_i = \text{Err}_i$  then
        if  $l \geq f$  then
            generate token of type  $a$  with word  $\text{In}[f \dots l]$ ;
             $p = f = l + 1; l = f - 1;$ 
             $\forall i : q_i = \text{Ini}_i;$ 
        else
            Lexical error
        endif
    endif
endif
endwhile
if  $l < f$  then Lexical error endif

```

The ANTLR Compiler Generator

Language and compiler for writing compilers

Running ANTLR on an ANTLR file produces Java source files that can be compiled and run.

ANTLR can generate

- Scanners (lexical analyzers)
- Parsers
- Tree walkers

An ANTLR File for a Simple Scanner

```
class CalcLexer extends Lexer;
```

```
LPAREN : '(' ;           // Rules for punctuation
```

```
RPAREN : ')' ;
```

```
STAR : '*' ;
```

```
PLUS : '+' ;
```

```
SEMI : ';' ;
```

```
protected           // Can only be used as a sub-rule
```

```
DIGIT : '0'..'9' ; // Any character between 0 and 9
```

```
INT : (DIGIT)+ ;    // One or more digits
```

```
WS : (' ' | '\t' | '\n' | '\r') // Whitespace  
    { setType(Token.SKIP); } ; // Action: ignore
```

ANTLR Specifications for Scanners

Rules are names starting with a capital letter.

A character in single quotes matches that character.

LPAREN : ' (' ;

A string in double quotes matches the string

IF : "if" ;

A vertical bar indicates a choice:

OP : '+' | '-' | '*' | '/' ;

ANTLR Specifications

Question mark makes a clause optional.

```
PERSON : ("wo")? 'm' ('a' | 'e') 'n' ;
```

(Matches man, men, woman, and women.)

Double dots indicate a range of characters:

```
DIGIT : '0'..'9' ;
```

Asterisk and plus match “zero or more,” “one or more.”

```
ID : LETTER (LETTER | DIGIT)* ;
```

```
NUMBER : (DIGIT)+ ;
```

Exercises

- Define regular expressions for the following languages:

1. Floating point numbers. Examples:

3.1416 -3e4 +1.0e-5 .567e+8

In case of using a ' . ', a decimal digit is always requested.

2. All strings from the alphabet $\{a, b, c\}$ in which the first appearance of symbol b is always preceded by at least one appearance of symbol a .
3. All strings of lowercase letters that contain the five vowels in order (each vowel must appear only once). Example: **zfaehipojksuj**.
4. All strings of lowercase letters in which the letters are in ascending lexicographic order. Examples: **afhmnqsyz**, **abcdz**, **dgky**, However, the string **bdeaz** does not belong to the language.
5. Financial quantities in American notation. Examples:

\$2,345.67 \$12,452,183.16 \$****12**

These have a leading dollar sign (\$), an optional string of asterisks (* – used on checks to discourage a fraud), a string of decimal digits, and an optional fractional part consisting of a decimal point (.) and two decimal digits. The string of digits to the left of the decimal point may consist of a single zero (0). Otherwise it must not start with a zero. If there are more than three digits to the left of the decimal point, groups of three (counting from the right) must be separated by commas (,).

6. Inexact constants in Scheme. Scheme allows real numbers to be explicitly inexact (imprecise). A programmer who wants to express all constants using the same number of characters can use sharp signs (#) in place of any lower-significance digits whose values are not known. A base-ten constant without exponent consists of one or more digits followed by zero or more sharp signs. An optional decimal point can be placed at the beginning, the end, or anywhere in between. Examples:

35## 35#.# ## 356.3## .35## 35. 35#.

- Given regular expressions for the identifiers of different languages: C, C++, C#, Java, Lisp, Fortran and SQL (use the reference manual or web pages to obtain this information).
- Define finite automata (deterministic or nondeterministic) for the following languages:

$(ab|ac)^*$ $a(a|b)^*a$ $((\varepsilon|a)b^*)^*$ $(a|b)^*a(a|b)(a|b)$