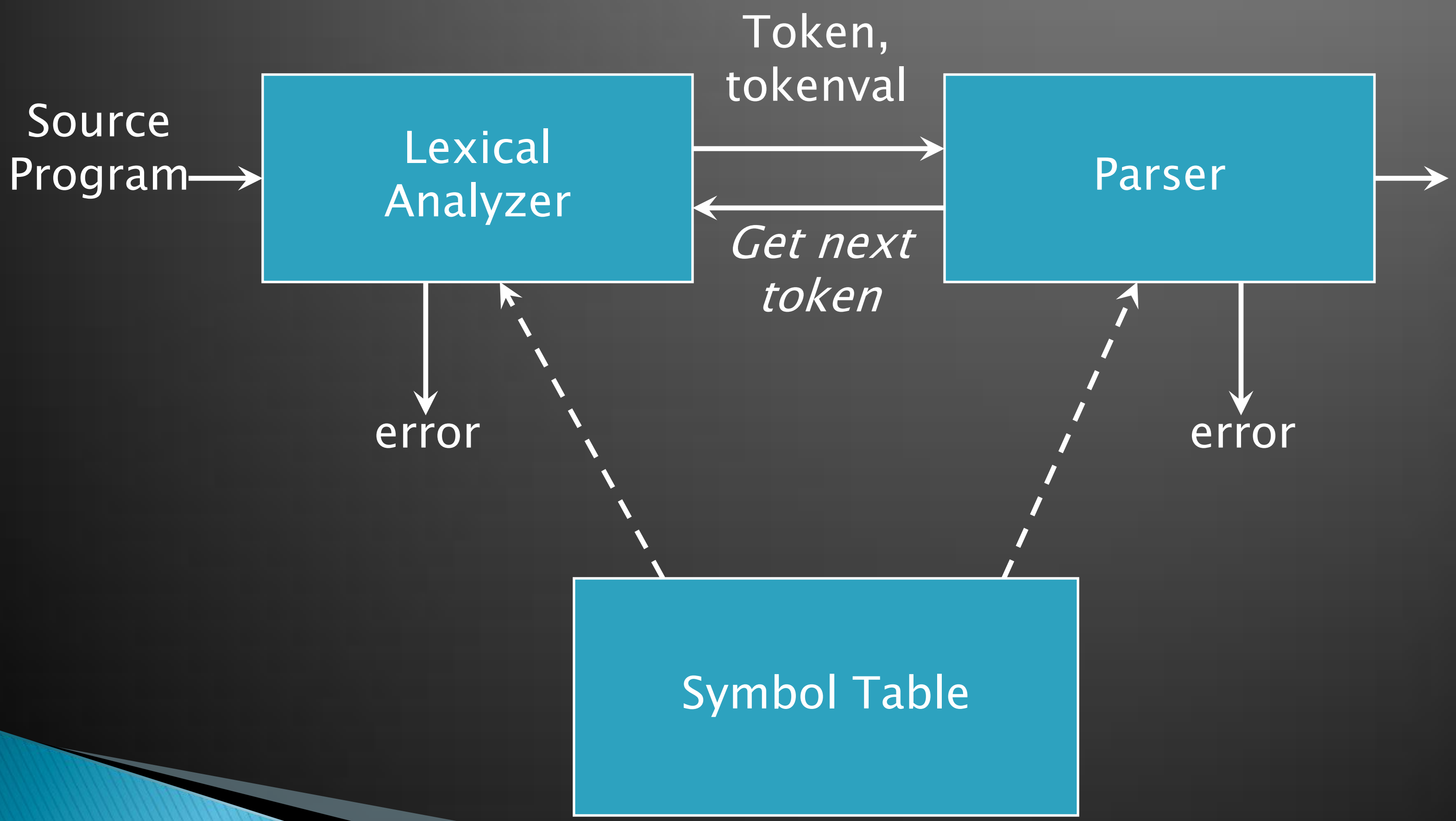
Lecture # 5

Lexical Analysis

Role of Lexical Analyzer

- It is the first phase of compiler
- Its main task is to read the input characters and produce as output a sequence of tokens that the parser uses for syntax analysis
- Reasons to make it a separate phase are:
  - Simplifies the design of the compiler
  - Provides efficient implementation(read the source code)
  - Improves portability

Interaction of the Lexical Analyzer with the Parser



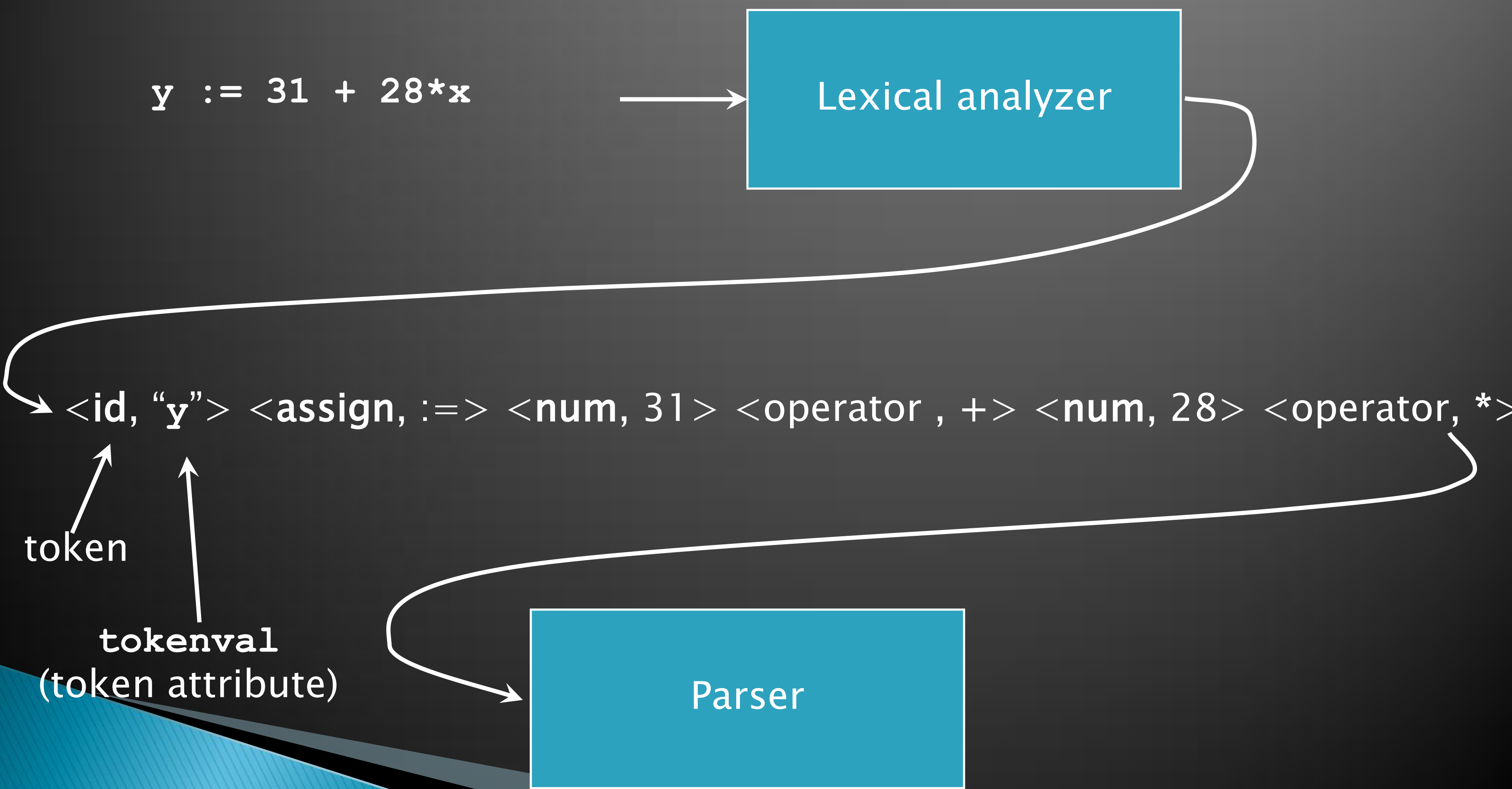
Tokens, Patterns, and Lexemes

- A *token* is a classification of lexical units
  - For example: **id** and **num**
- *Lexemes* are the specific character strings that make up a token
  - For example: `abc` and `123`
- *Patterns* are rules describing the set of lexemes belonging to a token
  - For example: "*letter followed by letters and digits*" and "*non-empty sequence of digits*"

Diff b/w Token, Lexeme and Pattern

Token	Lexeme	Pattern
if	if	if
relation	<, <=,=,<>,>,>=	< or <= or = or <> or > or >=
id	y, x	Letter followed by letters and digits
num	31 , 28	Any numeric constant
operator	+ , *, – ,/	Any arithmetic operator + or * or – or /

Attributes of Tokens



Specification of Tokens

- Alphabet: Finite, nonempty set of symbols
  Example: $\Sigma = \{0, 1\}$ binary alphabet
  Example: $\Sigma = \{a, b, c, \ldots, z\}$ the set of all lower case letters
- Strings: Finite sequence of symbols from an alphabet e.g. 0011001
- Empty String: The string with zero occurrences of symbols from alphabet. The empty string is denoted by ϵ

Continue…

- <u>Length of String:</u> Number of positions for symbols in the string. |w| denotes the length of string w

  Example $|0110| = 4$; $|\epsilon| = 0$
- <u>Powers of an Alphabet:</u> $\Sigma^k =$ the set of strings of length k with symbols from Σ

  Example:

$$\Sigma = \{0, 1\}$$

$$\Sigma^1 = \{0, 1\}$$

$$\Sigma^2 = \{00, 01, 10, 11\}$$

$$\Sigma^0 = \{\epsilon\}$$

Continue..

- The set of all strings over Σ is denoted Σ^*

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

Continue..

- Language: is a specific set of strings over some fixed alphabet Σ

  Example:

  The set of legal English words

  The set of strings consisting of n 0's followed by n 1's

  LP = the set of binarv numbers whose value is prime $\{\epsilon, 01, 0011, 000111, \ldots\}$

$$\{10, 11, 101, 111, 1011, \ldots\}$$

Concatenation and Exponentiation

- The *concatenation* of two strings x and y is denoted by xy
- The *exponentiation* of a string s is defined by

$$s^0 = \varepsilon$$
$$s^i = s^{i-1}s \quad \text{for } i > 0$$

note that $s\varepsilon = \varepsilon s = s$

Language Operations

- <u>Union</u>
  $L \cup M = \{s \mid s \in L \text{ or } s \in M\}$
- <u>Concatenation</u>
  $LM = \{xy \mid x \in L \text{ and } y \in M\}$
- <u>Exponentiation</u>
  $L^0 = \{\varepsilon\}; \quad L^i = L^{i-1}L$
- <u>Kleene closure</u>
  $L^* = \cup_{i=0,\ldots,\infty} L^i$
- <u>Positive closure</u>
  $L^+ = \cup_{i=1,\ldots,\infty} L^i$

Regular Expressions

- Basis symbols:
  - ε is a regular expression denoting language $\{\varepsilon\}$
  - $a \in \Sigma$ is a regular expression denoting $\{a\}$
- If r and s are regular expressions denoting languages $L(r)$ and $M(s)$ respectively, then
  - $r|s$ is a regular expression denoting $L(r) \cup M(s)$
  - rs is a regular expression denoting $L(r)M(s)$
  - r^* is a regular expression denoting $L(r)^*$
  - (r) is a regular expression denoting $L(r)$
- A language defined by a regular expression is called a *Regular set or a Regular Language*

Regular Definitions

- Regular definitions introduce a naming convention:

  $$d_1 \to r_1$$
  $$d_2 \to r_2$$
  $$\ldots$$
  $$d_n \to r_n$$

  where each r_i is a regular expression over

  $$\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$$

- Example:

$$\textbf{letter} \rightarrow \mathtt{A} \mid \mathtt{B} \mid \ldots \mid \mathtt{Z} \mid \mathtt{a} \mid \mathtt{b} \mid \ldots \mid \mathtt{z}$$
$$\textbf{digit} \rightarrow 0 \mid 1 \mid \ldots \mid 9$$
$$\textbf{id} \rightarrow \textbf{letter} \ (\ \textbf{letter} \mid \textbf{digit} \ )^*$$

- The following shorthands are often used:

$$r^+ = rr^*$$
$$r? = r \mid \varepsilon$$
$$[\mathtt{a}-\mathtt{z}] = \mathtt{a} \mid \mathtt{b} \mid \mathtt{c} \mid \ldots \mid \mathtt{z}$$

- Examples:

$$\textbf{digit} \rightarrow [0-9]$$
$$\textbf{num} \rightarrow \textbf{digit}^+ \ (.\ \textbf{digit}^+)? \ (\ \textbf{E} \ (+ \mid -)? \ \textbf{digit}^+ \ )?$$

Regular Definitions and Grammars

Grammar

$stmt \rightarrow$ **if** $expr$ **then** $stmt$

$\quad\mid$ **if** $expr$ **then** $stmt$ **else** $stmt$

$\quad\mid \varepsilon$

$expr \rightarrow term$ **relop** $term$

$\quad\mid term$

$term \rightarrow$ **id**

$\quad\mid$ **num**

Regular definitions

if $\rightarrow$ `if`

then $\rightarrow$ `then`

else $\rightarrow$ `else`

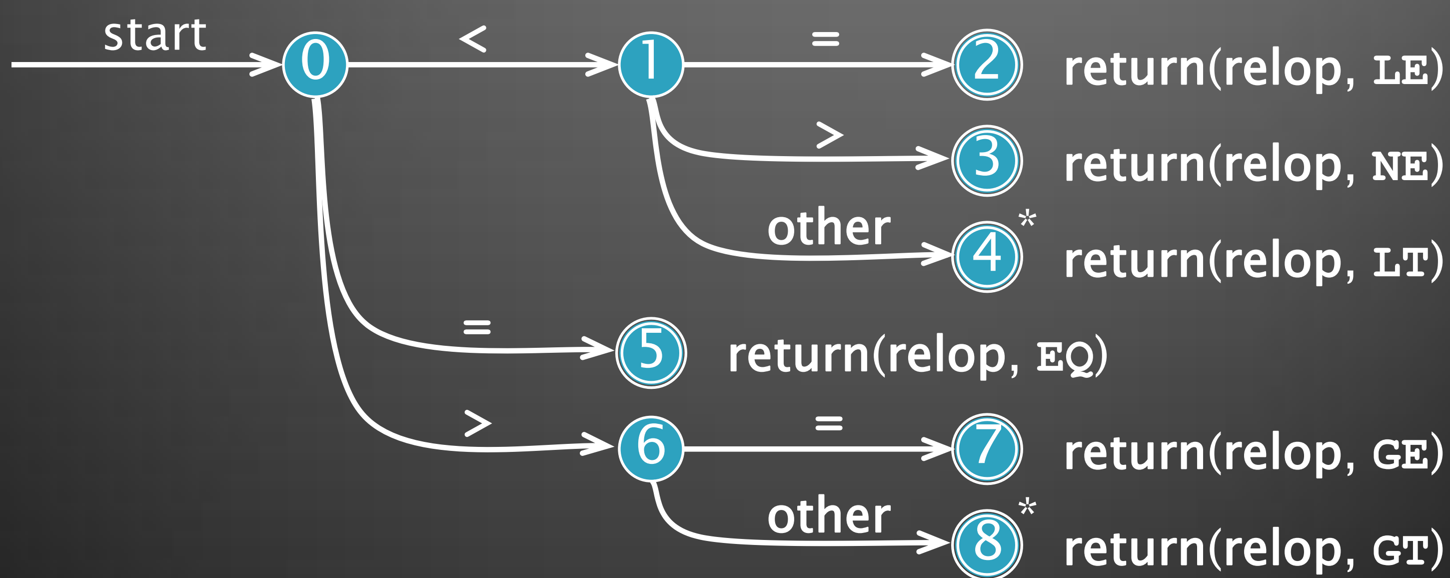
relop $\rightarrow$ < | <= | <> | > | >= | =

id $\rightarrow$ **letter (letter | digit)***

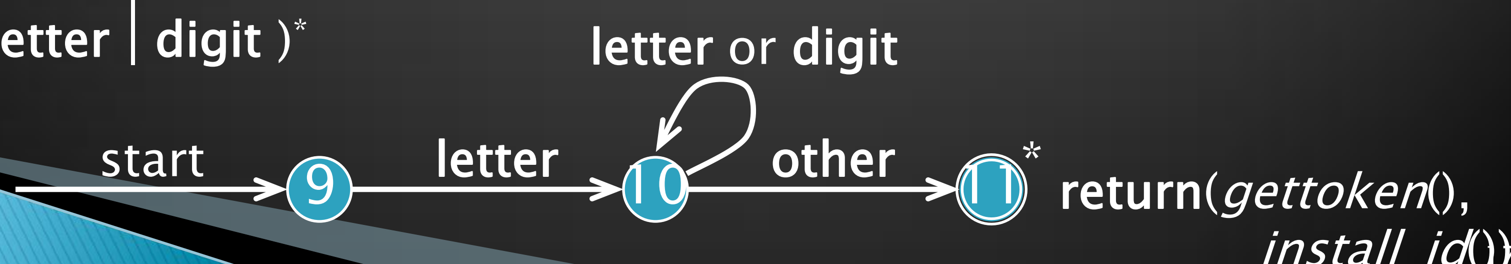
num $\rightarrow$ **digit**$^+$ (. **digit**$^+$)? (**E** (+ | –)? **digit**$^+$)?

Coding Regular Definitions in Transition Diagrams

relop → < | <= | <> | > | >= | =



id → letter (letter | digit)*

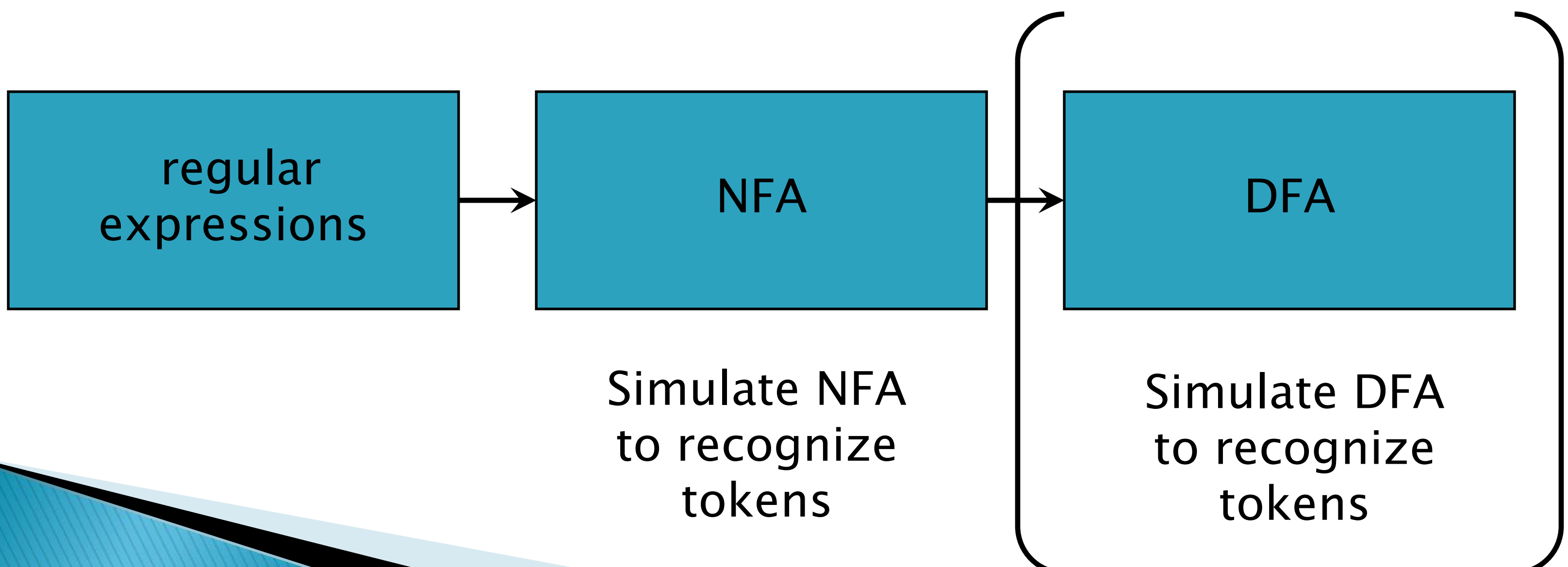


Finite Automata

- Finite Automata are used as a model for:
  - Software for designing digital circuits
  - Lexical analyzer of a compiler
  - Searching for keywords in a file or on the web.
  - Software for verifying finite state systems, such as communication protocols.

Design of a Lexical Analyzer Generator

- Translate regular expressions to NFA
- Translate NFA to an efficient DFA



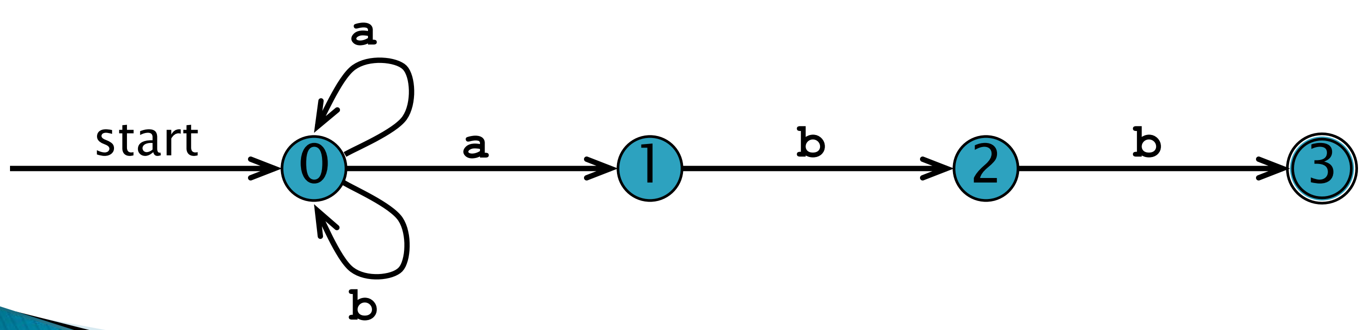
Nondeterministic Finite Automata

- An NFA is a 5-tuple $(S, \Sigma, \delta, s_0, F)$ where

  S is a finite set of *states*
  Σ is a finite set of symbols, the *alphabet*
  δ is a *mapping* from $S \times \Sigma$ to a set of states
  $s_0 \in S$ is the *start state*
  $F \subseteq S$ is the set of *accepting* (or *final*) *states*

Transition Graph

- An NFA can be diagrammatically represented by a labeled directed graph called a *transition graph*



$S = \{0,1,2,3\}$
$\Sigma = \{\mathbf{a},\mathbf{b}\}$
$s_0 = 0$
$F = \{3\}$

Transition Table

- The mapping δ of an NFA can be represented in a *transition table*

$$\delta(0,\mathtt{a}) = \{0,1\}$$
$$\delta(0,\mathtt{b}) = \{0\}$$
$$\delta(1,\mathtt{b}) = \{2\}$$
$$\delta(2,\mathtt{b}) = \{3\}$$

$\longrightarrow$

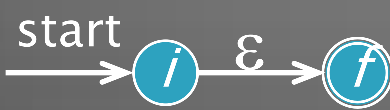
State	*Input* **`a`**	*Input* **`b`**
0	{0, 1}	{0}
1		{2}
2		{3}

The Language Defined by an NFA

- An NFA *accepts* an input string *x* if and only if there is some path with edges labeled with symbols from *x* in sequence from the start state to some accepting state in the transition graph
- A state transition from one state to another on the path is called a *move*
- The *language defined by* an NFA is the set of input strings it accepts, such as (**`a`**|**`b`**)*`abb` for the example NFA

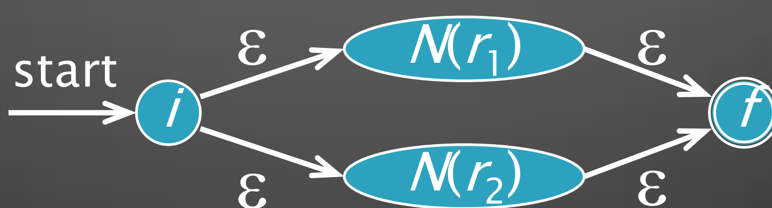
From Regular Expression to ε-NFA (Thompson's Construction)

ε



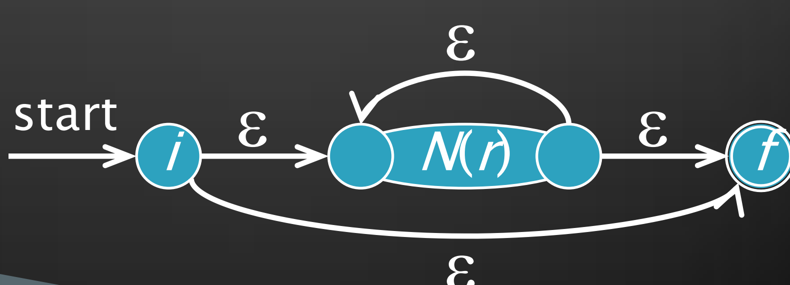
a

$r_1 \mid r_2$



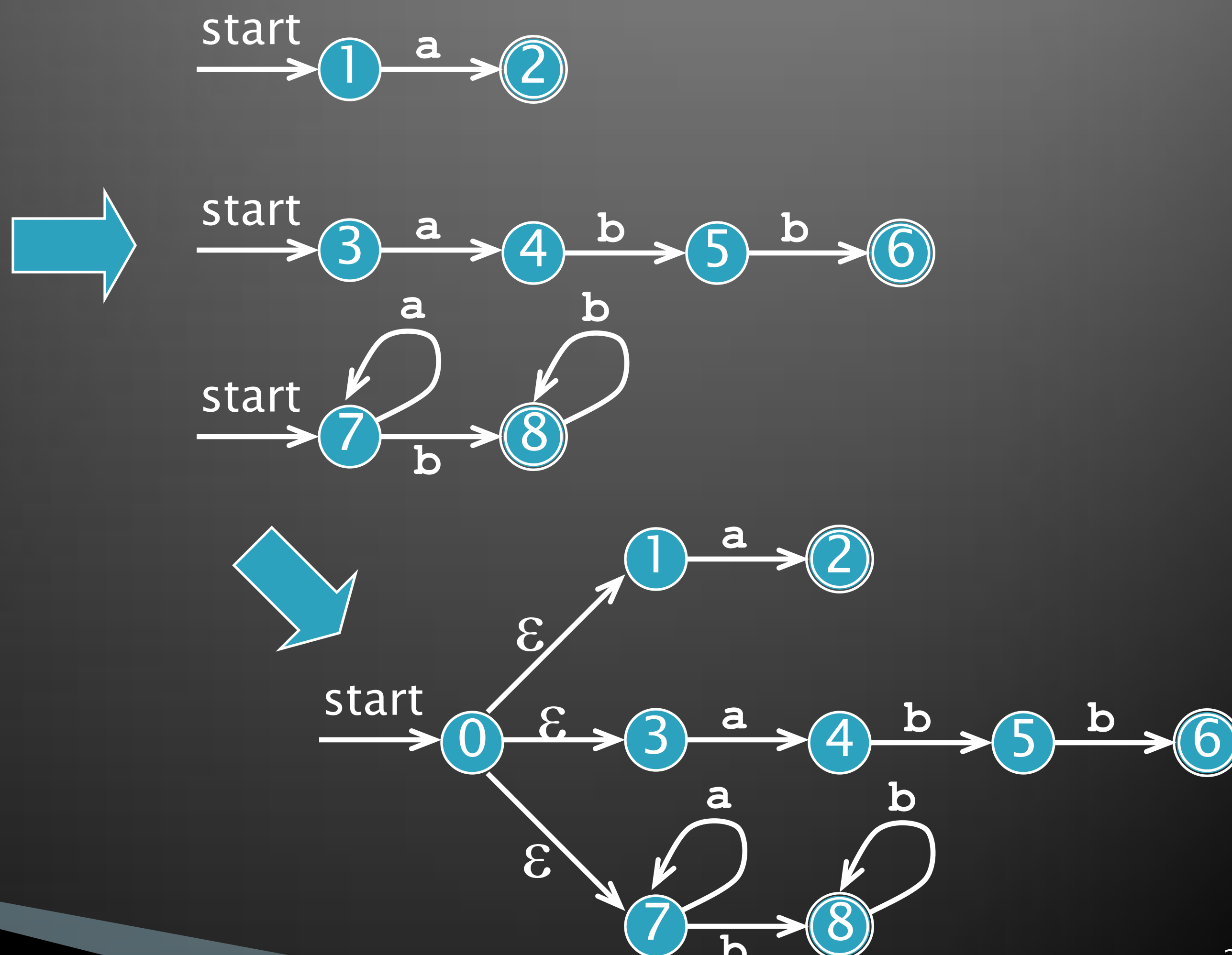
$r_1 r_2$

r^*



Combining the NFAs of a Set of Regular Expressions

a	{ *action*$_1$ }
abb	{ *action*$_2$ }
a*b+	{ *action*$_3$ }

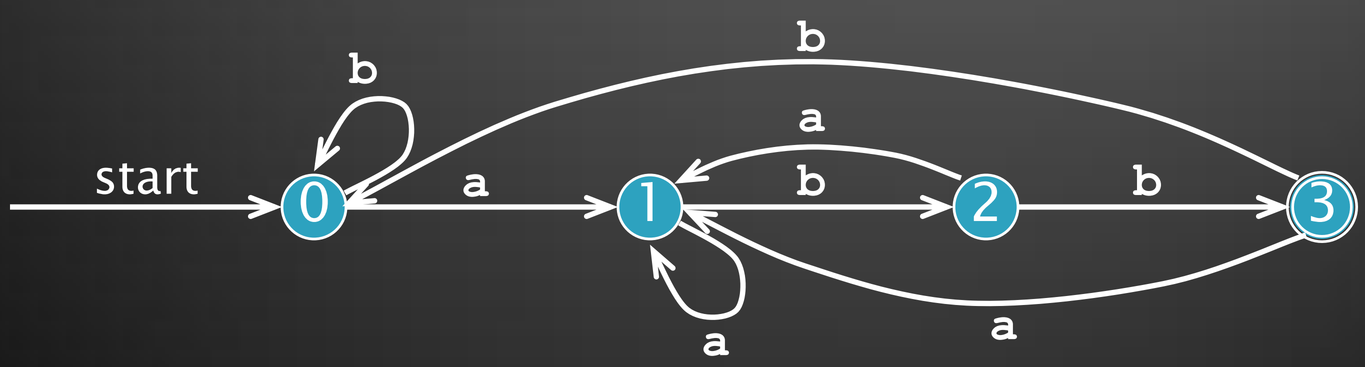


Deterministic Finite Automata

- A *deterministic finite automaton* is a special case of an NFA
  - No state has an ε–transition
  - For each state *s* and input symbol *a* there is at most one edge labeled *a* leaving *s*
- Each entry in the transition table is a single state
  - At most one path exists to accept a string
  - Simulation algorithm is simple

Example DFA

A DFA that accepts $(\mathbf{a} \mid \mathbf{b})^*\mathbf{abb}$



Conversion of an NFA into a DFA

- The *subset construction algorithm* converts an NFA into a DFA using:

  $\varepsilon\text{-}closure(s) = \{s\} \cup \{t \mid s \to_\varepsilon \ldots \to_\varepsilon t\}$

  $\varepsilon\text{-}closure(T) = \cup_{s \in T}\, \varepsilon\text{-}closure(s)$

  $move(T,a) = \{t \mid s \to_a t \text{ and } s \in T\}$

- The algorithm produces:

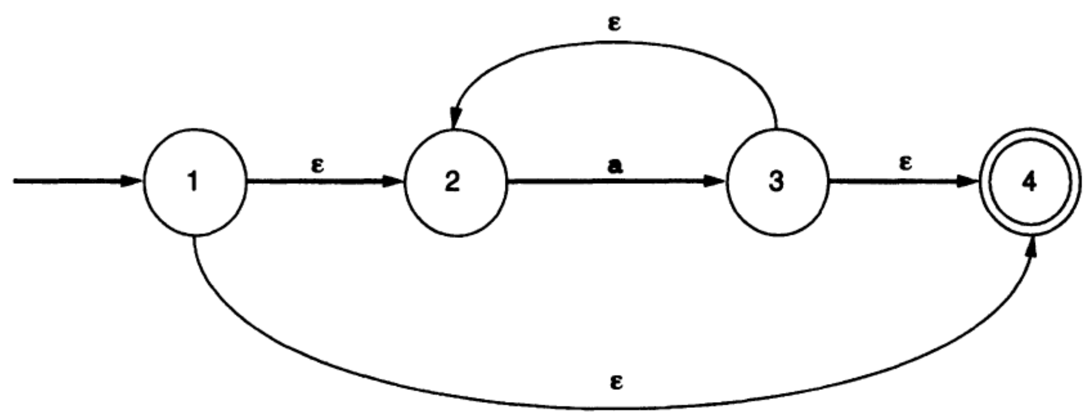
  Dstates is the set of states of the new DFA consisting of sets of states of the NFA

  Dtran is the transition table of the new DFA

Thus, this algorithm is called the **subset construction**. We first discuss the ε-closure in a little more detail and then proceed to a description of the subset construction.

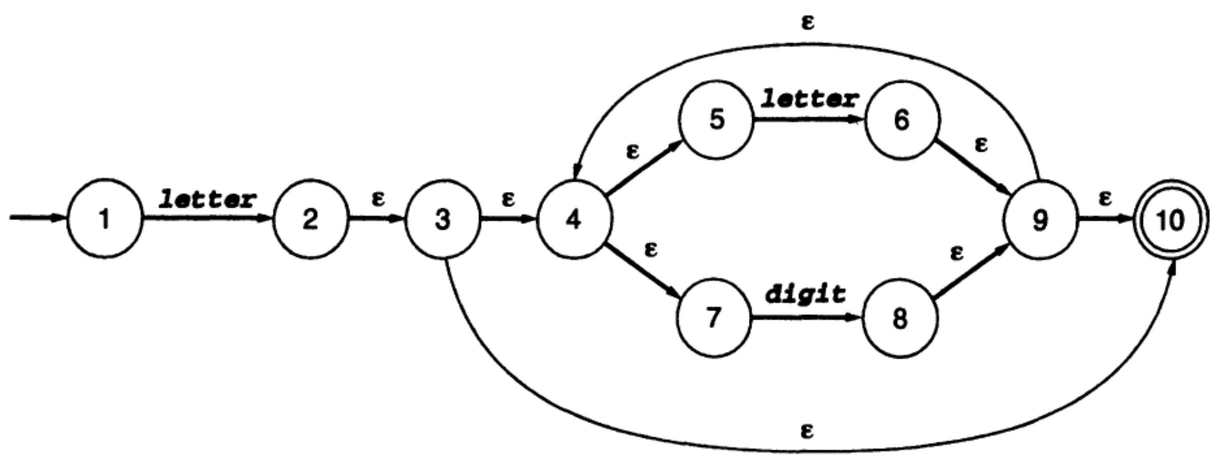
The ε-Closure of a Set of States We define the ε-closure of a single state s as the set of states reachable by a series of zero or more ε-transitions, and we write this set as $\bar{s}$. We leave a more mathematical statement of this definition to an exercise and proceed directly to an example. Note, however, that the ε-closure of a state always contains the state itself.

mple 2.14 Consider the following NFA corresponding to the regular expression **a*** under Thompson's construction:



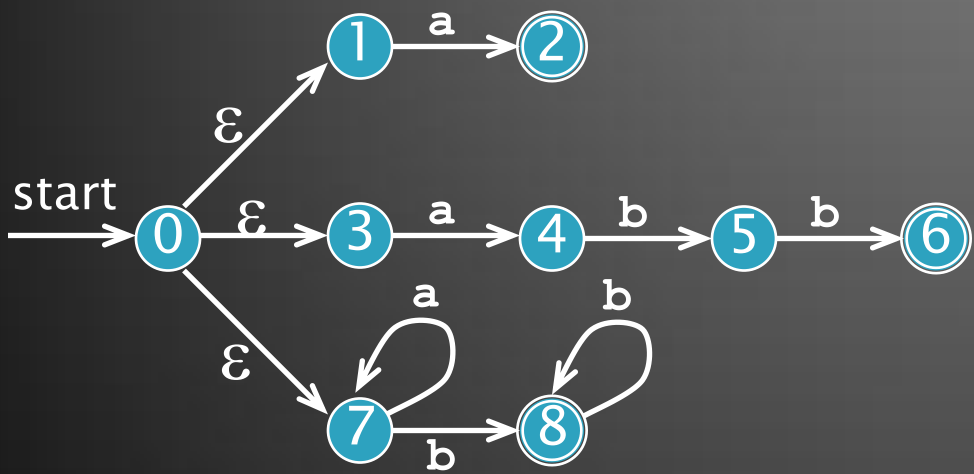
In this NFA, we have $\bar{1} = \{1, 2, 4\}$, $\bar{2} = \{2\}$, $\bar{3} = \{2, 3, 4\}$, and $\bar{4} = \{4\}$. §

mple 2.17 Consider the NFA of Figure 2.9 (Thompson's construction for the regular expression **`letter(letter|digit)*`**):

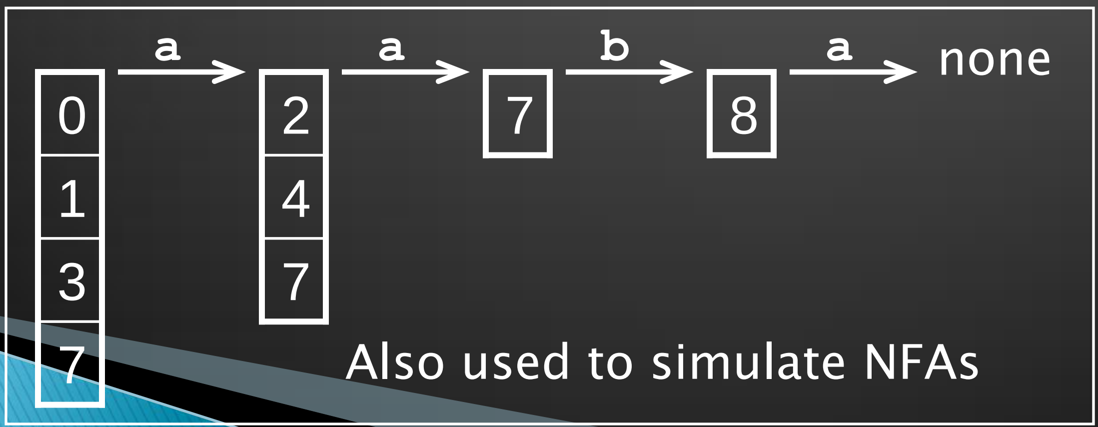


The subset construction proceeds as follows. The start state is $\overline{\{1\}} = \{1\}$. There is a transition on ***letter*** to $\overline{\{2\}} = \{2, 3, 4, 5, 7, 10\}$. From this state there is a transition on ***letter*** to $\overline{\{6\}} = \{4, 5, 6, 7, 9, 10\}$ and a transition on ***digit*** to $\overline{\{8\}} = \{4, 5, 7, 8, 9, 10\}$. Finally, each of these states also has transitions on ***letter*** and ***digit***, either to itself or to the other. The complete DFA is given in the following picture:

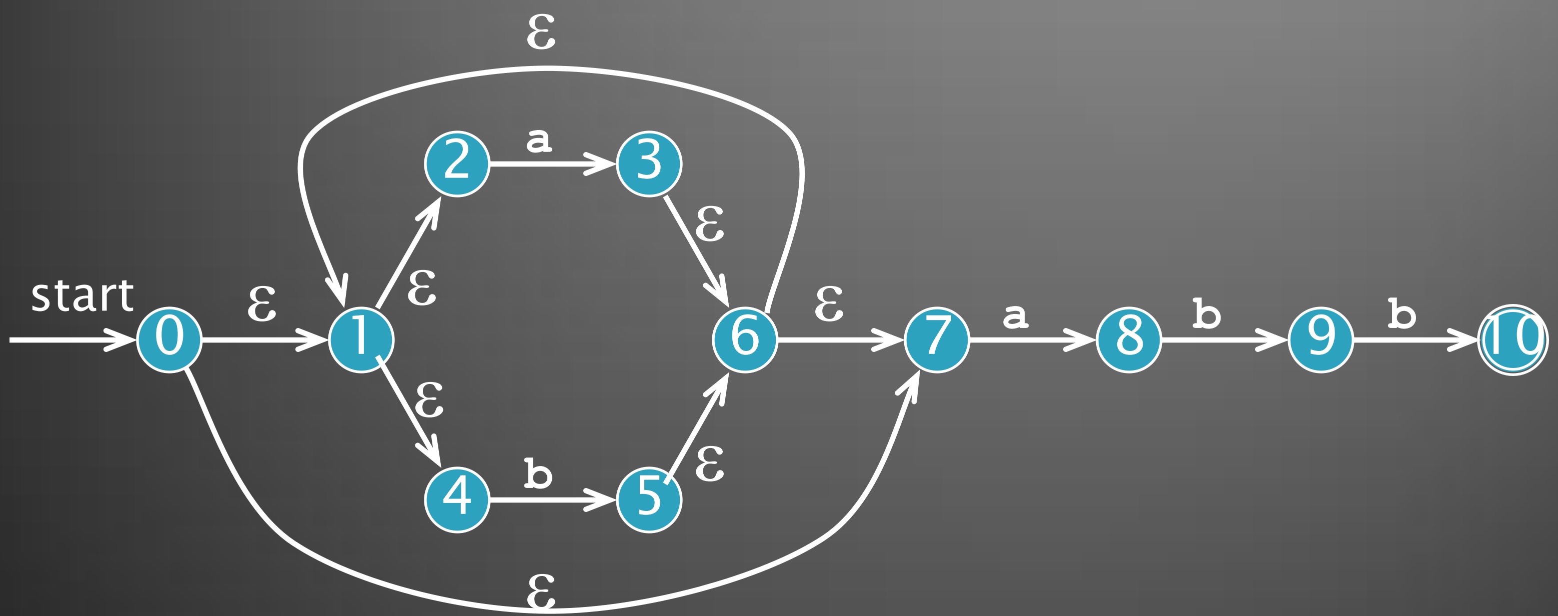
ε-*closure* and *move* Examples

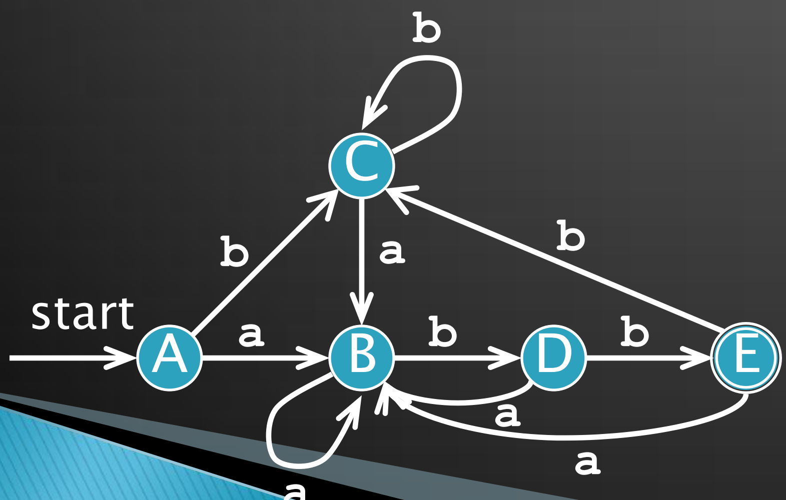


ε-*closure*({0}) = {0,1,3,7}
move({0,1,3,7},**a**) = {2,4,7}
ε-*closure*({2,4,7}) = {2,4,7}
move({2,4,7},**a**) = {7}
ε-*closure*({7}) = {7}
move({7},**b**) = {8}
ε-*closure*({8}) = {8}
move({8},**a**) = $\emptyset$



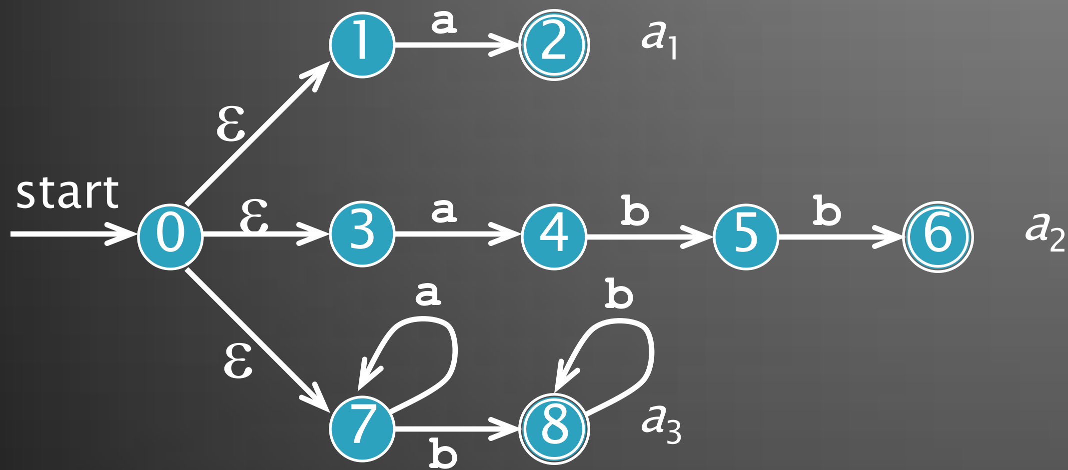
Subset Construction Example 1

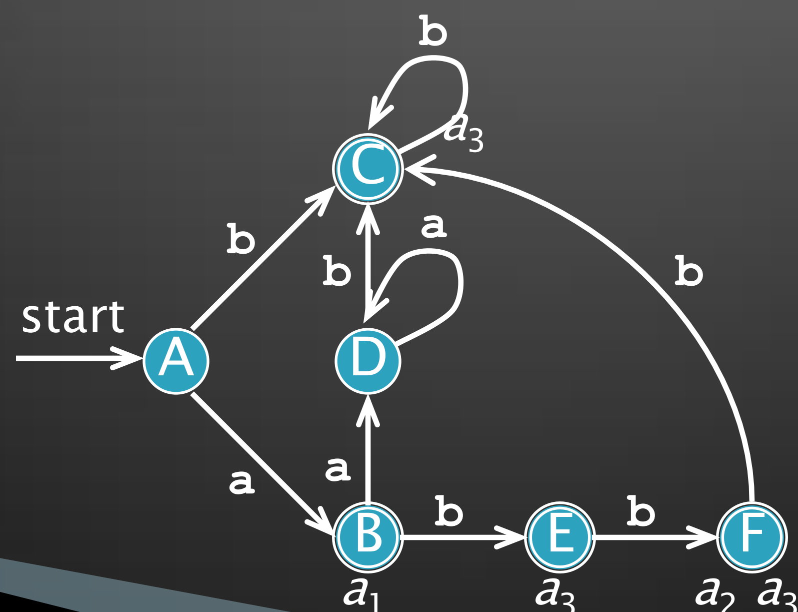




Dstates
A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}
D = {1,2,4,5,6,7,9}
E = {1,2,4,5,6,7,10}

Subset Construction Example 2





Dstates

A = {0,1,3,7}

B = {2,4,7}

C = {8}

D = {7}

E = {5,8}

F = {6,8}