2. Lexical Analysis

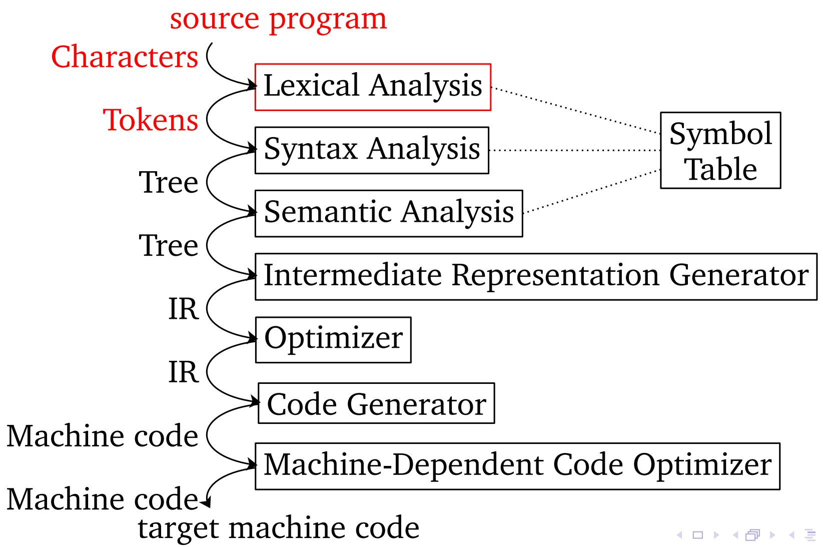
Eva Rose Kristoffer Rose

NYU Courant Institute

Compiler Construction (CSCI-GA.2130-001)
http://cs.nyu.edu/courses/fall14/CSCI-GA.2130-001/lecture-2.pdf

September 11, 2014

First compilation phase

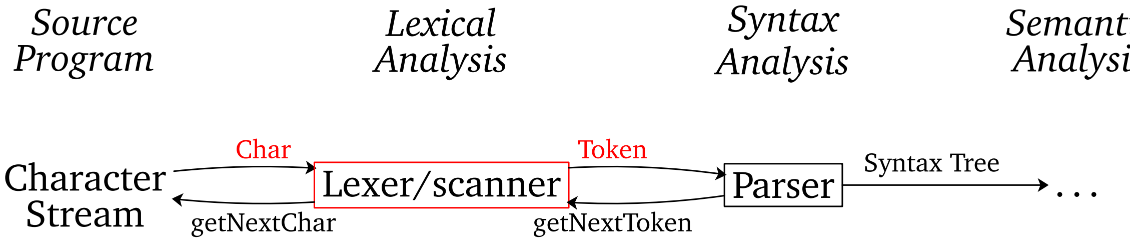


Purpose

Primary roles of the lexer/scanner:

- read input characters and expand macros,
- identify lexeme patterns for tokens,
- identify token attributes,
- construct the symbol table,
- error handling and reporting,
- strip whitespace and comments.

Software modules involved



Question

Why separating lexical and syntactic analysis?

- Simplifies overall compiler design.
- Improves compiler efficiency (allows specialized techniques for lexer issues vs parser issues).
- Better portability (input-device specific peculiarities restricted to lexer).

Example

The stream of *characters*:

```
position = initial + rate * 60
```

Scanned into list of *tokens*, one for each *lexeme*:

$\langle \mathbf{id}, 1\rangle \ \langle = \rangle \ \langle \mathbf{id}, 2\rangle \ \langle + \rangle \ \langle \mathbf{id}, 3\rangle \ \langle * \rangle \ \langle \mathbf{num}, 60\rangle$

1	`position`
2	`initial`
3	`rate`

Example

Lexeme	Token	Lexeme pattern (informal)	Attribute value
`position`	$\langle \mathbf{id}, 1\rangle$	identifier string	1
`=`	$\langle = \rangle$	equality symbol	
`initial`	$\langle \mathbf{id}, 2\rangle$	identifier string	2
`+`	$\langle + \rangle$	addition symbol	
`rate`	$\langle \mathbf{id}, 3\rangle$	identifier string	3
`*`	$\langle * \rangle$	multiplication symbol	
`60`	$\langle \mathbf{num}, 60\rangle$	numeric constant	60

Tokens

Typical classes of tokens in a programming language:

- keywords,
- operators,
- one token for identifiers,
- tokens for constants (numbers, literal strings,..),
- tokens for punctuation symbols (comma, semicolon,..).

Token attributes

Typical attribute values for the identifier token (**id**):

- the precise lexeme,
- its type/storage requirements,
- location where first found (error reporting),

Example

```
float limitedSquare(float x) {
  /* returns x, squared, but never more than 100 */
  return (x<=-10.0||x>=10.0) ? 100 : x*x;
}
```

What tokens will likely be generated from the above C program?

Dealing with errors

Lexical analyser unable to proceed when no pattern matches

- Panic mode recovery: delete characters from input until a matching pattern is found.
- Insert missing character.
- Replace a character with another.
- Transpose two adjacent characters.

Input buffering

Buffering issues

- Lexer may need to look at least a character ahead to make a token decision.
- Example: Assume two tokens are defined as ‘*’ and ‘***’. Now consider the input character sequence ‘**a’. We need to scan and buffer all three characters to decide the tokens are two ‘*’.

Specifying tokens: lexeme patterns

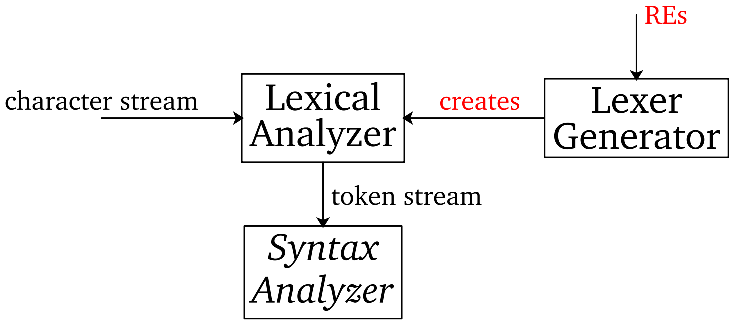
Some lexeme patterns: `+`, `6-0`, `=`, `=-=`, `p-o-s-i-t-i-o-n`, ...

"The world" of lexeme patterns include finite, infinite, cyclic, and balanced patterns ...

partition into

- Regular expressions (RE) (formal token descriptions),
- other patterns.

Lexer Generator



- Lex, Flex
- HACS has its own lexical-analyser *generator*

Token specification

Identifiers as RE:

$$letter(letter|digit)^*$$

where

- *letter* – any letter in *the alphabet*,
- *digit* – any single digit from the 0 to 9 range.

Token specification

Some lexeme patterns/strings as RE:

- operator: +
- equality: (=|==)
- number: $\{ d^+ \mid d \in Digit \}$
- identifiers: $\{ l(d|l)^* \mid d \in Digit,\ l \in Letter \}$
- empty string: ϵ (epsilon)

Digit and *Letter* stands for *alphabets*.

Definition

Regular Expression concepts:

Alphabet	any finite set of symbols.
String	finite sequence of symbols drawn from an alphabet.
Language	countable set of strings over some fixed alphabet.

Definition

Basic RE taxonomy:

Concept	Syntax	Pattern	Matches
Character	e	b	b
Concatenation	$e_1\, e_2$	bc	bc
Choice	$e_1 \mid e_2$	a | bc	a, bc
Repetition($\geq$0)	e^*	$\mathtt{a}^*$	ϵ, a, aa, ...
Grouping	(e)	(a|b)c	ac, bc

where $*$ denotes the *Kleene Closure* of the generated language.

Examples

Describe the languages (patterns/strings) generated by REs:

- $(\epsilon|a)b^*$
- $a(a|b)^*a$
- $a^*ba^*ba^*ba^*$

Definition

Extended RE taxonomy:

Concept	Syntax	Pattern	Matches
Optional	$e?$	$($a $\mid$ b$)?$	ϵ, a, b
Repetition(≥ 1)	e^{+}	a^{+}	a, aa, $\ldots$
Character class	$[\ldots]$	[0-9 _ a-z]	7, _, c

where $+$ denotes the *Positive Closure* of the generated language.

Examples

State REs for the following languages:

- all non-empty strings over the alphabet of ‘0’s and ‘1’s,
- all strings beginning with a digit, followed by either a or b, and ending with an uppercase letter.

Answers

- all non-empty strings over the alphabet of ‘0’s and ‘1’s,
- **(0|1)+**

- all strings beginning with a digit, followed by either a or b, and ending with an uppercase letter.
- **[0-9](a|b)[A-Z]**

Simulation

Extended RE features as basic RE:

Extended syntax	Basic syntax
$e?$	$e \mid \epsilon$
e^+	$e\, e^*$
$[e_1 - e_2 \ldots e_i - e_{i+1}]$	$e^1 \mid \ldots \mid e^i$

where $e^1 \in \{e_1, \ldots, e_2\}, \ldots, e^i \in \{e_i, \ldots, e_{i+1}\}, i \geq 1$

HACS

Token specification in HACS as REs:

```
space [ \t\n] ;

token Int   | ⟨Digit⟩+ ;
token Float | ⟨Int⟩ "." ⟨Int⟩ ;
token Id    | ⟨Lower⟩+ ('_'? ⟨Int⟩)? ;

token fragment Digit | [0−9] ;
token fragment Lower | [a−z] ;
```

Token specification

Some common computer alphabets:

- Ascii code (*e.g.*, `\177`)
- Unicode (*e.g.*, `\u12f`)

Example REs from the real world (in HACS):

- `[\000-\037]` ascii control codes,
- `[\\]u[0-9a-fA-F]+` unicode escape characters.

Additional string vocabulary

Prefix	a string obtained by removing zero or more symbols from the end.
Suffix	a string obtained by removing zero or more symbols from the beginning.
Substring	a string obtained by removing any prefix and suffix.
subsequence	a string formed by removing zero or more arbitrary positions.

Consider: ‘banana’

Formal Languages

Sets of strings

Approaches for defining formal languages (strings):

- basic RE taxonomy
- extended RE taxonomy
- context-free grammar (CFG) <—
- nondeterministic finite automaton
- determinstic finite automaton

Formal grammars

Grammar for branching statements

$$\begin{aligned}
\mathit{stmt} &\rightarrow \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt} \\
&\mid \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stmt}\ \textbf{else}\ \mathit{stmt} \\
&\mid \epsilon \\
\mathit{expr} &\rightarrow \mathit{term}\ \textbf{relop}\ \mathit{term} \\
&\mid \mathit{term} \\
\mathit{term} &\rightarrow \textbf{id} \\
&\mid \textbf{number}
\end{aligned}$$

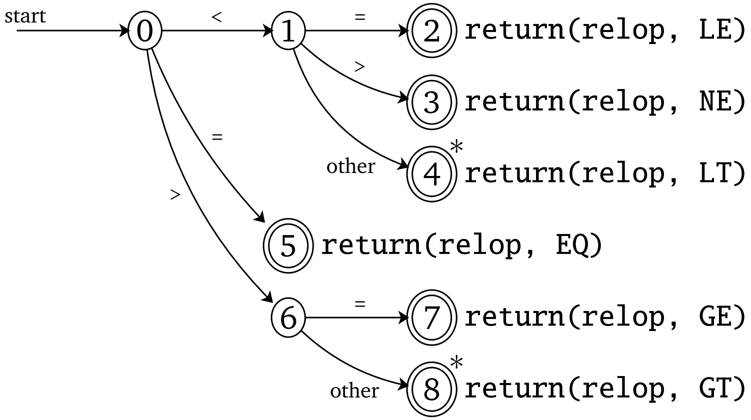
Formal grammars

Patterns for tokens

$$\begin{aligned}
id &\rightarrow letter(letter \mid digits)^* \\
letter &\rightarrow [A-Za-z] \\
number &\rightarrow digits(.digits)?([+-]?digits)? \\
digits &\rightarrow digit^+ \\
digit &\rightarrow [0-9] \\
if &\rightarrow if \\
then &\rightarrow then \\
else &\rightarrow else \\
relop &\rightarrow \; < \mid > \mid <= \mid >= \mid = \mid <>
\end{aligned}$$

Transition diagram

Diagram recognizes the **relop** token.



Expressive Language Power

“Language power” given by the possibly generated strings

Comparisons:

- extended and basic RE has *same power*
- CFG more powerful than RE …
- finite automata and RE has *same power*

Comparing CFG and RE

CFG at least as expressive as RE:
Can emulate RE with CFG:

RE:	CFG:
$(\mathsf{a} \mid \mathsf{b})^* \, \mathsf{abb}$	$X \to Y\,\mathsf{abb}$ $Y \to YZ \mid \epsilon$ $Z \to \mathsf{a} \mid \mathsf{b}$

Comparing CFG and RE

CFG more expressive than RE:

CFG:	RE:
$X \to$ [X] | a	?

Captures a, [a], [[a]], ..., $[^n \mathtt{a}\,]^n$.

Definition

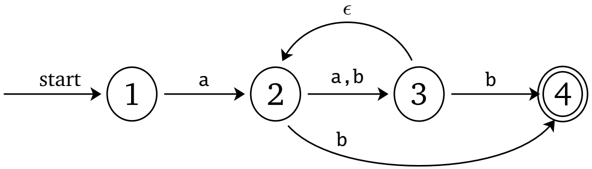
Finite state automata are string recognizers: answering "yes" or "no" to each input string.

Nondeterministic finite automata (NFA) transition diagrams with no restrictions on the labels of their edges.

Deterministic automata transition diagrams where for each state and each symbol of the input alphabet there is exactly one edge leaving that state.

NFA: Nondeterministic Finite (State) Automata

NFA *accepting* $a(a|b)^*b$:



NFA accepting $a(a|b)^*b$

- $States = \{1, 2, 3, 4\}$
- $\Sigma = \{\mathtt{a}, \mathtt{b}\}$
-

	a	b	ϵ
1	$\{2\}$	$\{\}$	$\{\}$
2	$\{3\}$	$\{3, 4\}$	$\{\}$
3	$\{\}$	$\{4\}$	$\{2\}$
4	$\{\}$	$\{\}$	$\{\}$

- $Start = 1$
- $Finals = \{4\}$

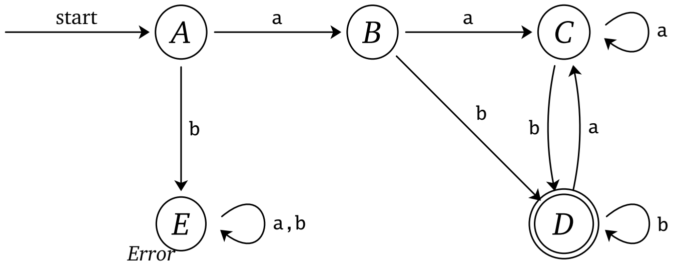
NFA core definition

- finite set of *states* S (circles),
- set of *input symbols* Σ (labels),
- next-state *transition function* $\tau : S \times \Sigma \to \Sigma$ (edges,table),

- dedicated *start state* $s_0 \in S$ (start arrow),
- set of *accepting states* $\subseteq S$ (double circles).

DFA: Deterministic Finite (State) Automata

DFA *accepting* $a(a|b)^*b$:



DFA accepting $a(a|b)^*b$

- $States = \{A, B, C, D, E\}$
- $\Sigma = \{\mathsf{a}, \mathsf{b}\}$
-

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

- $Start = A$
- $Finals = \{D\}$

DFA core definition

Special case of NFA:

- no moves on ϵ,
- for each $(s, a) \in S \times \Sigma : \exists\, \tau(s, a)$ unique!
- Transition graph: there is exactly *one edge* out of each state s and each label a.
- Transition table: each entry is a *single state*.

Some definitions

Operation	Description
ϵ-$\mathit{closure}(s)$	Set of NFA states reachable from set s on ϵ-transitions alone.
ϵ-$\mathit{closure}(T)$	Set of NFA states reachable from some set s in T on ϵ- transitions alone.
$\mathit{move}(T, a)$	Set of NFA states to which there is a transition on input symbol a from some state s in T

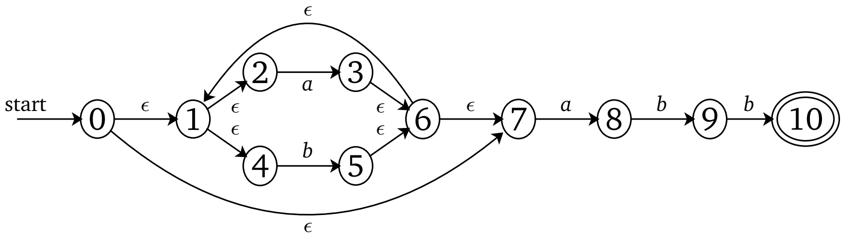
Subset construction

NFA to DFA (Algorithm 3.20.):

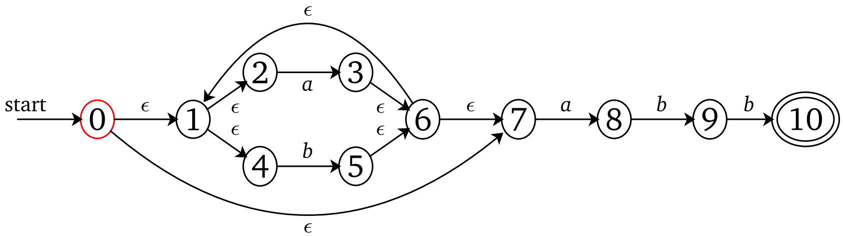
```
// Initially, $\epsilon$-closure($s_0$) is the only state in $Dstates$,
    and it is unmarked;
while (there is an unmarked state $T$ in $Dstates$) {
    mark $T$;
    for (each input symbol $a$) {
            $\bigcup = \epsilon\text{-}closure(move(T, a))$;
             if ($\bigcup$ is not in $Dstates$)
                  add $\bigcup$ as an unmarked state to $Dstates$
            $Dtran[T, a] = \bigcup$
    }
}
```

Dstates states of the DFA we are constructing.

NFA → DFA Translation for $(a|b)^*abb$

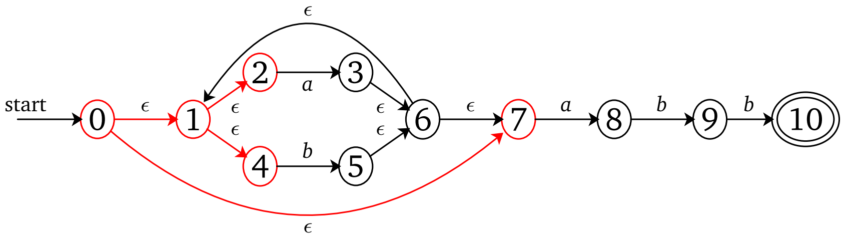

$$\begin{tikzcd}
 & & 2 \arrow[r, "a"] & 3 \arrow[dr, "\epsilon"] & & & & & \\
\text{start} \arrow[r] & 0 \arrow[r, "\epsilon"] \arrow[rrrrr, bend right, "\epsilon"'] & 1 \arrow[u, "\epsilon"] \arrow[d, "\epsilon"] & & 6 \arrow[r, "\epsilon"] \arrow[lll, bend right, "\epsilon"'] & 7 \arrow[r, "a"] & 8 \arrow[r, "b"] & 9 \arrow[r, "b"] & 10 \\
 & & 4 \arrow[r, "b"] & 5 \arrow[ur, "\epsilon"] & & & & &
\end{tikzcd}$$

NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



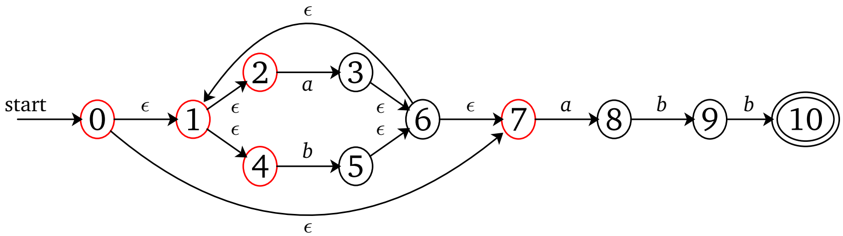
$$\epsilon\text{-}closure(0) = \{0, 1, 2, 4, 7\} = A$$

NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$

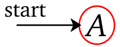


$$\epsilon\text{-}closure(0) = \{0, 1, 2, 4, 7\} = A$$

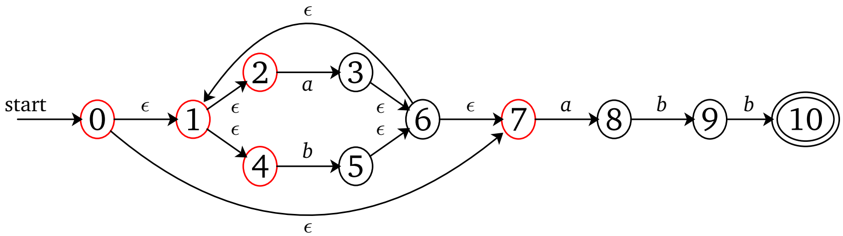
NFA → DFA Translation for $(a|b)^*abb$



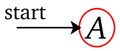
$$\epsilon\text{-}closure(0) = \{0, 1, 2, 4, 7\} = A$$



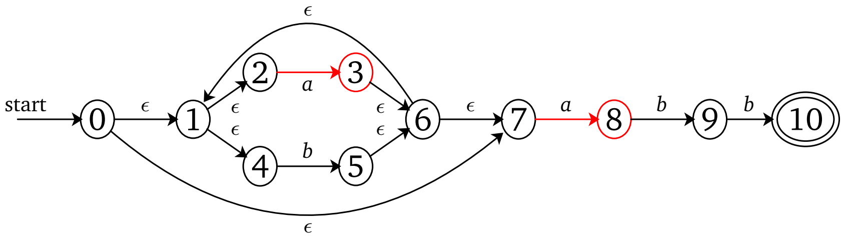
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



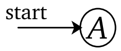
$$
\begin{aligned}
DTran[A, a] &= \epsilon\text{-}closure(move(A, a)) \\
&= \epsilon\text{-}closure(\{3, 8\}) \\
&= \{1, 2, 3, 4, 6, 7, 8\} = B
\end{aligned}
$$



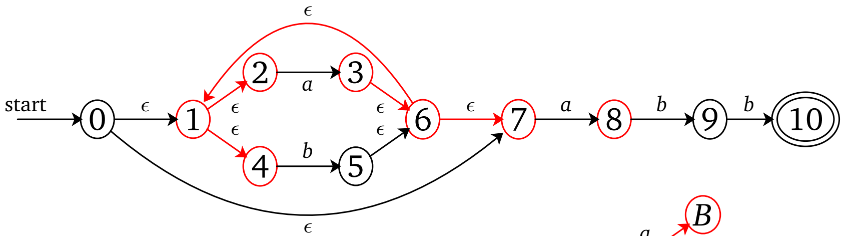
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



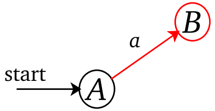
$$
\begin{aligned}
DTran[A, a] &= \epsilon\text{-}closure(move(A, a)) \\
&= \epsilon\text{-}closure(\{3, 8\}) \\
&= \{1, 2, 3, 4, 6, 7, 8\} = B
\end{aligned}
$$



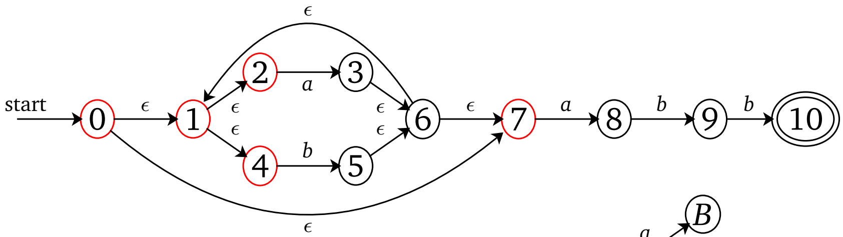
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



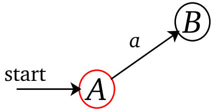
$$
\begin{aligned}
DTran[A, a] &= \epsilon\text{-}closure(move(A, a)) \\
&= \epsilon\text{-}closure(\{3, 8\}) \\
&= \{1, 2, 3, 4, 6, 7, 8\} = B
\end{aligned}
$$



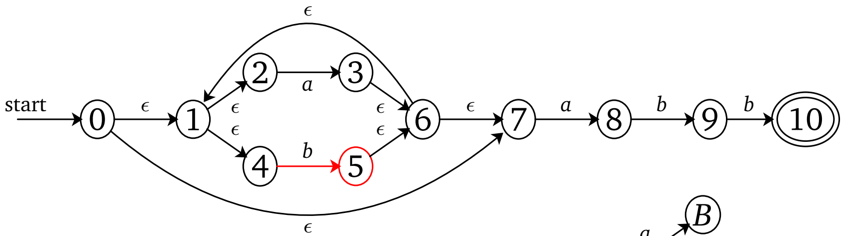
NFA → DFA Translation for $(a|b)^*abb$

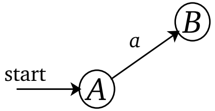


$$
\begin{aligned}
DTran[A, b] &= \epsilon\text{-}closure(move(A, b)) \\
&= \epsilon\text{-}closure(\{5\}) \\
&= \{1, 2, 4, 5, 6, 7\} = C
\end{aligned}
$$



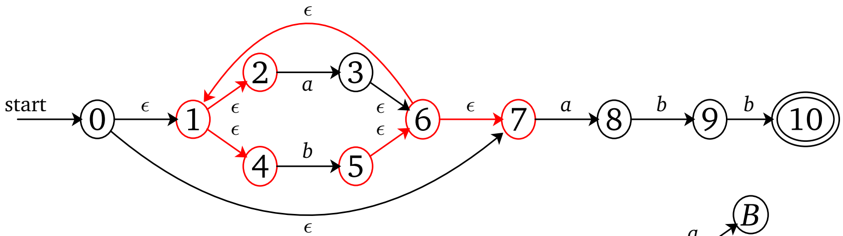
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



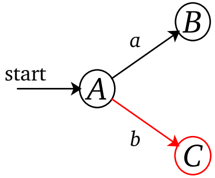


$$\begin{aligned} DTran[A, b] &= \epsilon\text{-}closure(move(A, b)) \\ &= \epsilon\text{-}closure(\{5\}) \\ &= \{1, 2, 4, 5, 6, 7\} = C \end{aligned}$$

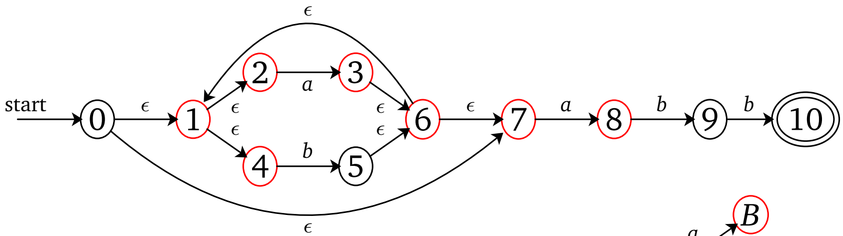
NFA → DFA Translation for $(a|b)^*abb$



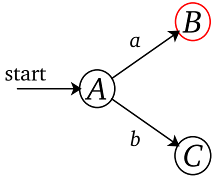
$$
\begin{aligned}
DTran[A,b] &= \epsilon\text{-}closure(move(A,b))\\
&= \epsilon\text{-}closure(\{5\})\\
&= \{1,2,4,5,6,7\} = C
\end{aligned}
$$



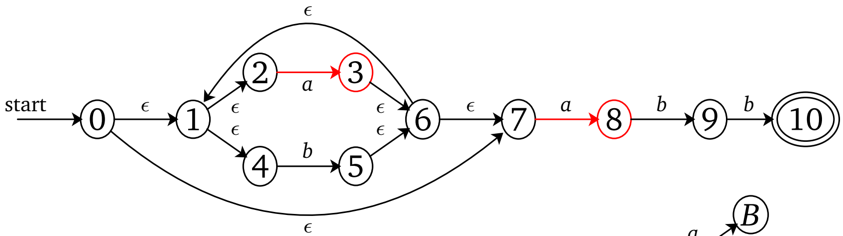
NFA → DFA Translation for $(a|b)^*abb$



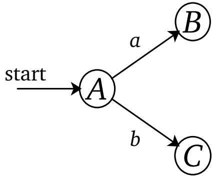
$$\begin{aligned} DTran[B, a] &= \epsilon\text{-}closure(move(B, a)) \\ &= \epsilon\text{-}closure(\{3, 8\}) \\ &= \{1, 2, 3, 4, 6, 7, 8\} = B \end{aligned}$$



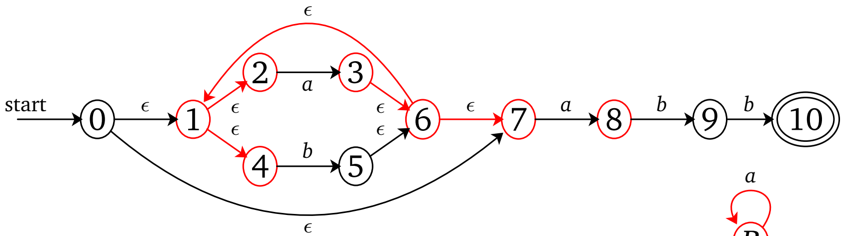
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



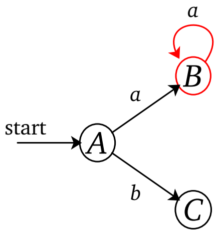
$$DTran[B,a] = \epsilon\text{-}closure(move(B,a))$$
$$= \epsilon\text{-}closure(\{3,8\})$$
$$= \{1,2,3,4,6,7,8\} = B$$



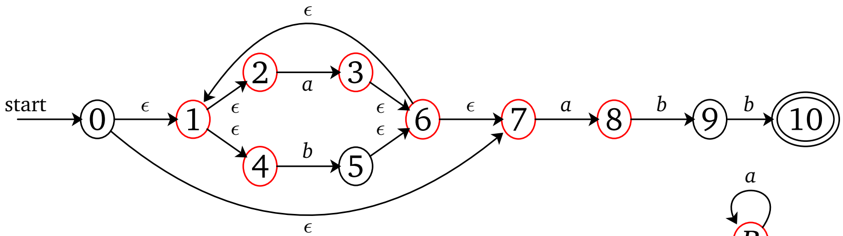
NFA → DFA Translation for $(a|b)^*abb$



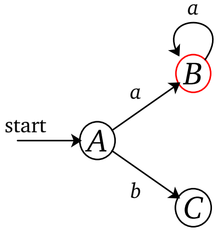
$$
\begin{aligned}
DTran[B,a] &= \epsilon\text{-}closure(move(B,a))\\
&= \epsilon\text{-}closure(\{3,8\})\\
&= \{1,2,3,4,6,7,8\} = B
\end{aligned}
$$



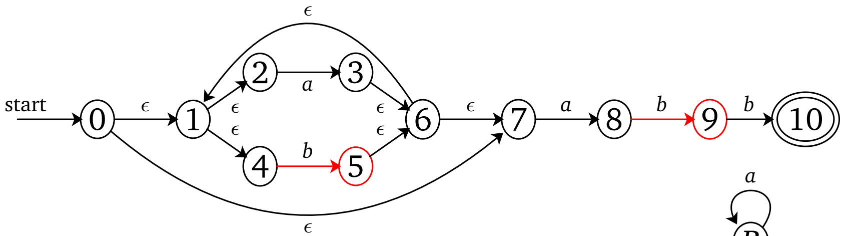
NFA → DFA Translation for $(a|b)^*abb$



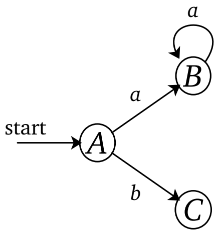
$$
\begin{aligned}
DTran[B, b] &= \epsilon\text{-}closure(move(B, b)) \\
&= \epsilon\text{-}closure(\{5, 9\}) \\
&= \{1, 2, 4, 5, 6, 7, 9\} = D
\end{aligned}
$$



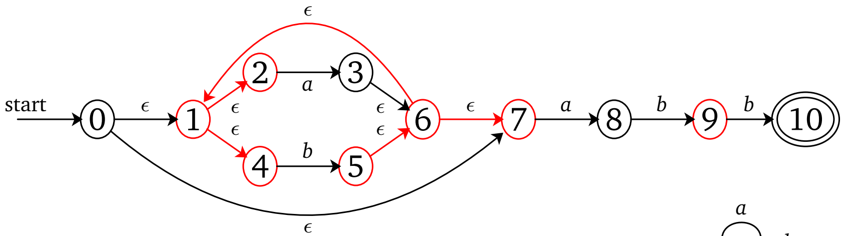
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



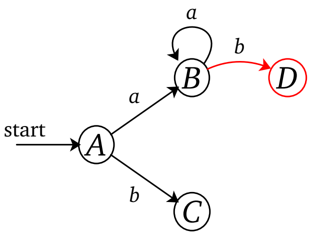
$$
\begin{aligned}
DTran[B, b] &= \epsilon\text{-}closure(move(B, b)) \\
&= \epsilon\text{-}closure(\{5, 9\}) \\
&= \{1, 2, 4, 5, 6, 7, 9\} = D
\end{aligned}
$$



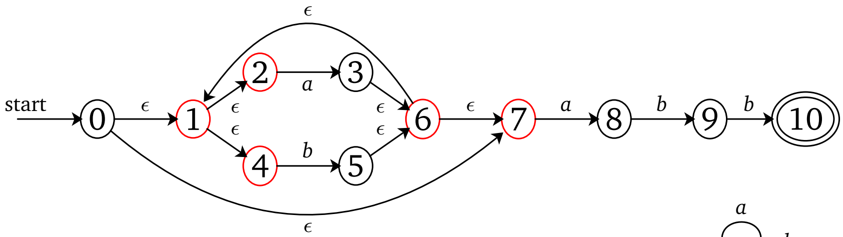
NFA → DFA Translation for $(a|b)^*abb$



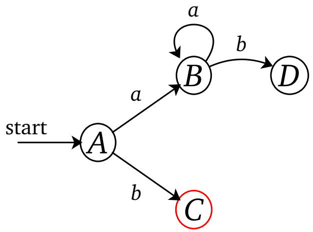
$$
\begin{aligned}
DTran[B,b] &= \epsilon\text{-}closure(move(B,b))\\
&= \epsilon\text{-}closure(\{5,9\})\\
&= \{1,2,4,5,6,7,9\} = D
\end{aligned}
$$



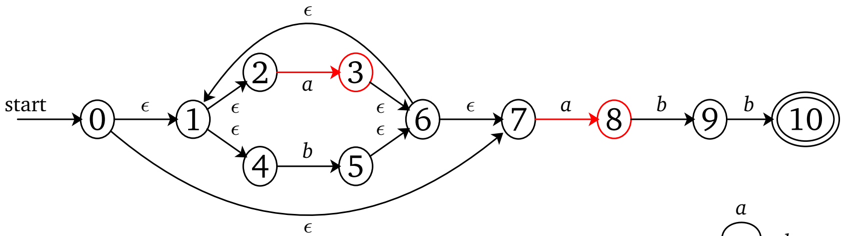
NFA → DFA Translation for $(a|b)^*abb$



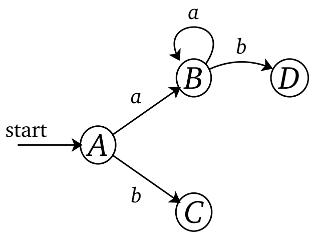
$$
\begin{aligned}
DTran[C, a] &= \epsilon\text{-}closure(move(C, a)) \\
&= \epsilon\text{-}closure(\{3, 8\}) \\
&= \{1, 2, 3, 4, 6, 7, 8\} = B
\end{aligned}
$$



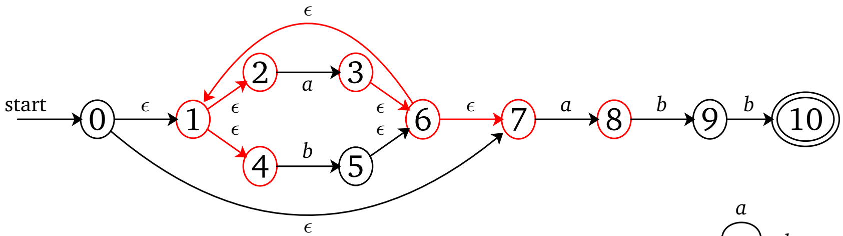
NFA → DFA Translation for $(a|b)^*abb$



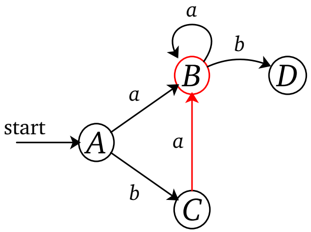
$$
\begin{aligned}
DTran[C, a] &= \epsilon\text{-}closure(move(C, a)) \\
&= \epsilon\text{-}closure(\{3, 8\}) \\
&= \{1, 2, 3, 4, 6, 7, 8\} = B
\end{aligned}
$$



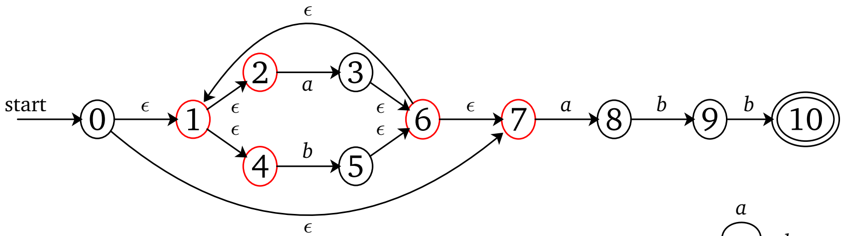
NFA → DFA Translation for $(a|b)^*abb$



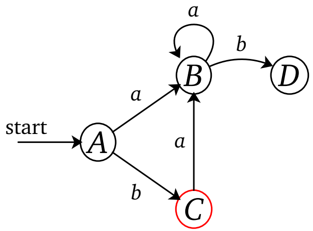
$$\begin{aligned} DTran[C,a] &= \epsilon\text{-}closure(move(C,a)) \\ &= \epsilon\text{-}closure(\{3,8\}) \\ &= \{1,2,3,4,6,7,8\} = B \end{aligned}$$



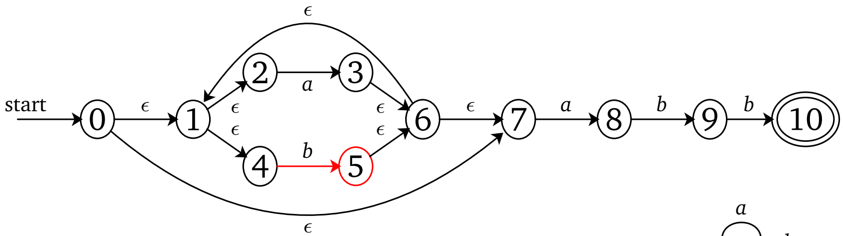
NFA → DFA Translation for $(a|b)^*abb$



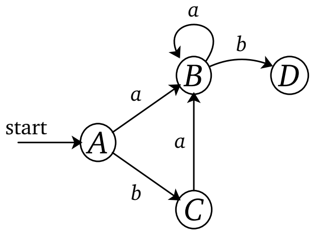
$$DTran[C, b] = \epsilon\text{-}closure(move(C, b))$$
$$= \epsilon\text{-}closure(\{5\})$$
$$= \{1, 2, 4, 5, 6, 7\} = C$$



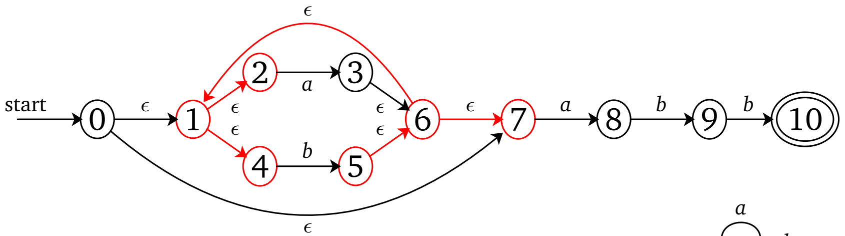
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



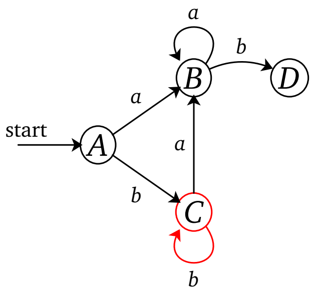
$$DTran[C,b] = \epsilon\text{-}closure(move(C,b))$$
$$= \epsilon\text{-}closure(\{5\})$$
$$= \{1,2,4,5,6,7\} = C$$



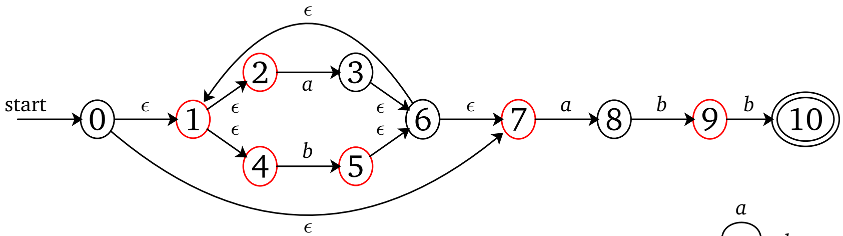
NFA → DFA Translation for $(a|b)^*abb$



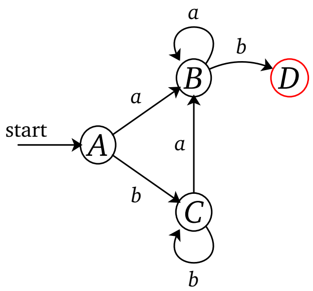
$$
\begin{aligned}
DTran[C,b] &= \epsilon\text{-}closure(move(C,b)) \\
&= \epsilon\text{-}closure(\{5\}) \\
&= \{1,2,4,5,6,7\} = C
\end{aligned}
$$



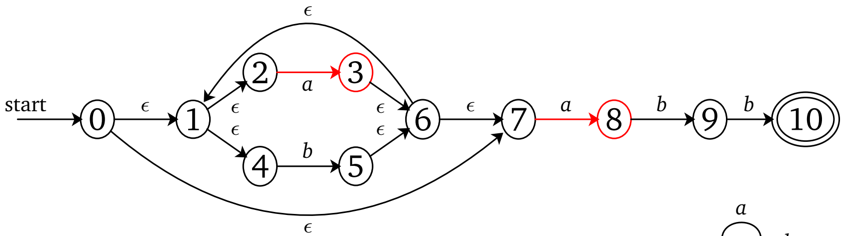
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



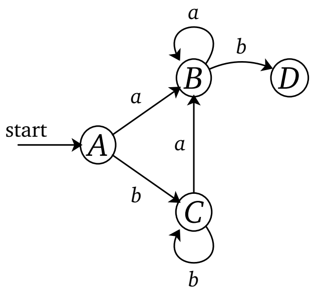
$$
\begin{aligned}
DTran[D, a] &= \epsilon\text{-}closure(move(D, a)) \\
&= \epsilon\text{-}closure(\{3, 8\}) \\
&= \{1, 2, 3, 4, 6, 7, 8\} = B
\end{aligned}
$$



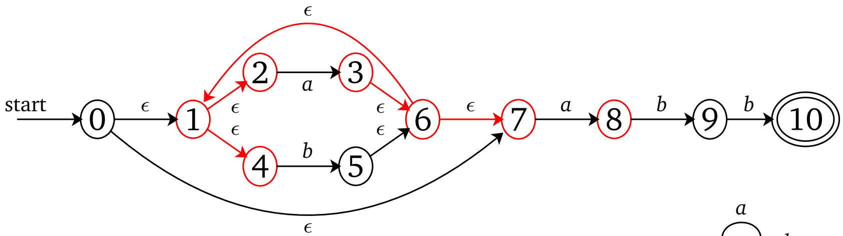
NFA → DFA Translation for $(a|b)^*abb$



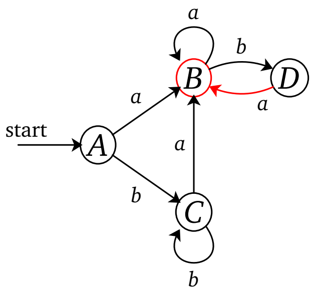
$$DTran[D,a] = \epsilon\text{-}closure(move(D,a))$$
$$= \epsilon\text{-}closure(\{3,8\})$$
$$= \{1,2,3,4,6,7,8\} = B$$



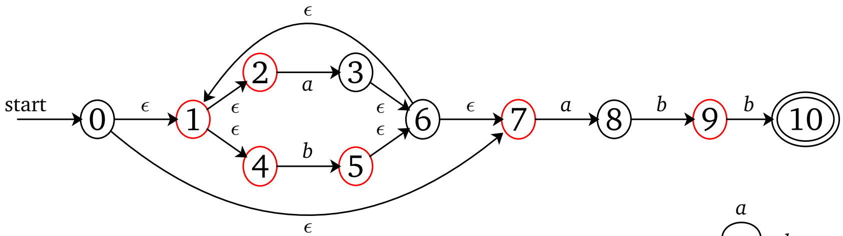
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



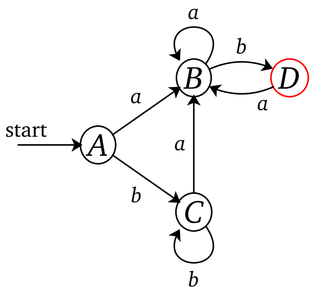
$$\begin{aligned} DTran[D,a] &= \epsilon\text{-}closure(move(D,a)) \\ &= \epsilon\text{-}closure(\{3,8\}) \\ &= \{1,2,3,4,6,7,8\} = B \end{aligned}$$



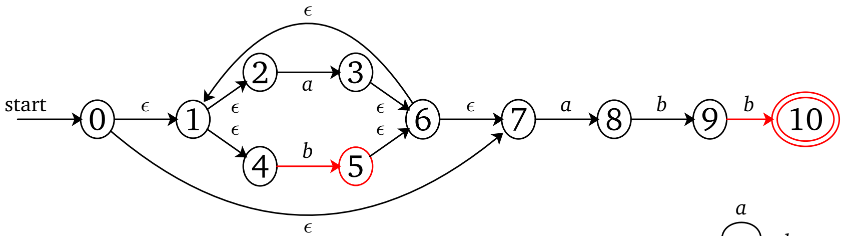
NFA → DFA Translation for $(a|b)^*abb$



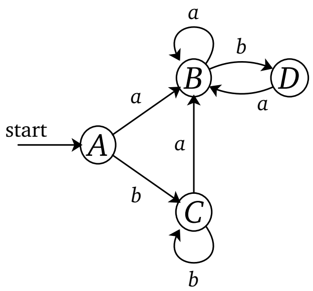
$$
\begin{aligned}
DTran[D, b] &= \epsilon\text{-}closure(move(D, b)) \\
&= \epsilon\text{-}closure(\{5, 10\}) \\
&= \{1, 2, 4, 5, 6, 7, 10\} = E
\end{aligned}
$$



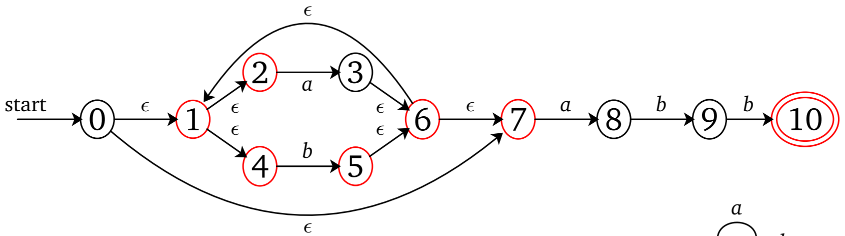
NFA → DFA Translation for $(a|b)^*abb$



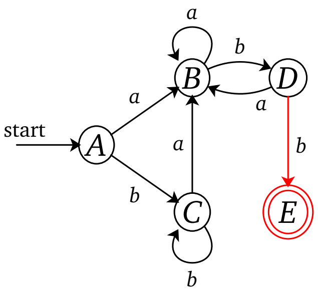
$$DTran[D, b] = \epsilon\text{-}closure(move(D, b))$$
$$= \epsilon\text{-}closure(\{5, 10\})$$
$$= \{1, 2, 4, 5, 6, 7, 10\} = E$$



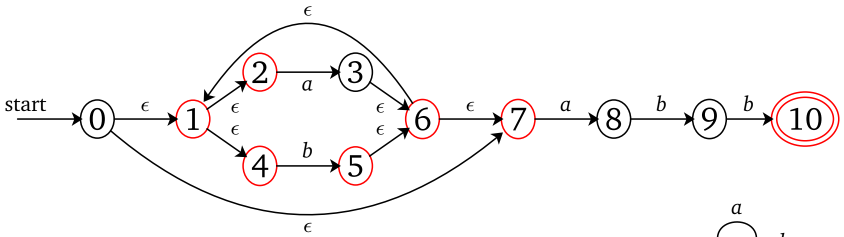
NFA → DFA Translation for $(a|b)^*abb$



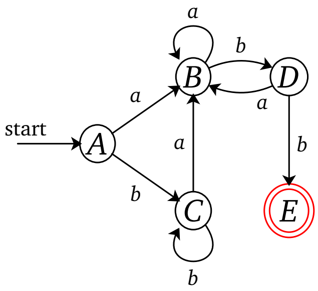
$$\begin{aligned} DTran[D,b] &= \epsilon\text{-}closure(move(D,b)) \\ &= \epsilon\text{-}closure(\{5,10\}) \\ &= \{1,2,4,5,6,7,10\} = E \end{aligned}$$



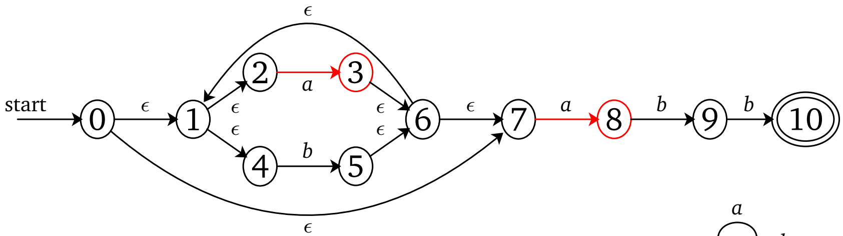
NFA → DFA Translation for $(a|b)^*abb$



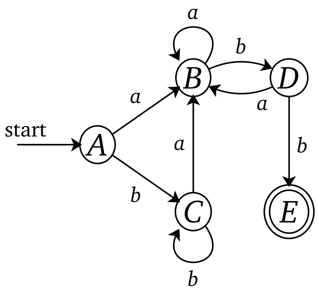
$$\begin{aligned} DTran[E, a] &= \epsilon\text{-}closure(move(E, a)) \\ &= \epsilon\text{-}closure(\{3, 8\}) \\ &= \{1, 2, 3, 4, 6, 7, 8\} = B \end{aligned}$$



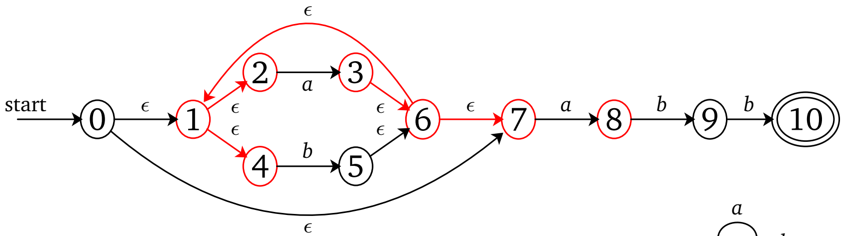
NFA → DFA Translation for $(a|b)^*abb$



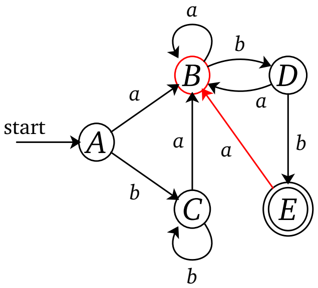
$$DTran[E,a] = \epsilon\text{-}closure(move(E,a))$$
$$= \epsilon\text{-}closure(\{3,8\})$$
$$= \{1,2,3,4,6,7,8\} = B$$



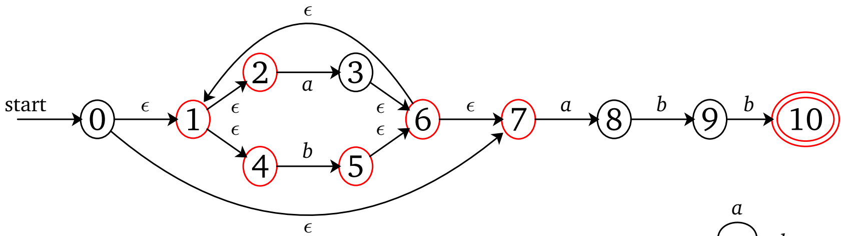
NFA → DFA Translation for $(a|b)^*abb$



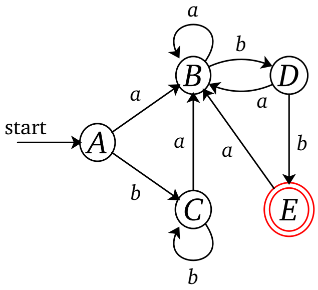
$$\begin{aligned} DTran[E,a] &= \epsilon\text{-}closure(move(E,a)) \\ &= \epsilon\text{-}closure(\{3,8\}) \\ &= \{1,2,3,4,6,7,8\} = B \end{aligned}$$



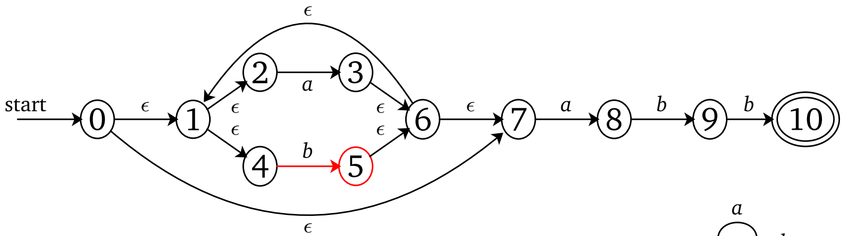
NFA $\rightarrow$ DFA Translation for $(a|b)^*abb$



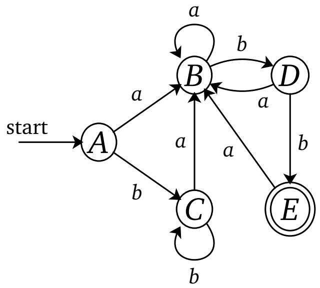
$$
\begin{aligned}
DTran[E, b] &= \epsilon\text{-}closure(move(E, b))\\
&= \epsilon\text{-}closure(\{5\})\\
&= \{1, 2, 4, 5, 6, 7\} = C
\end{aligned}
$$



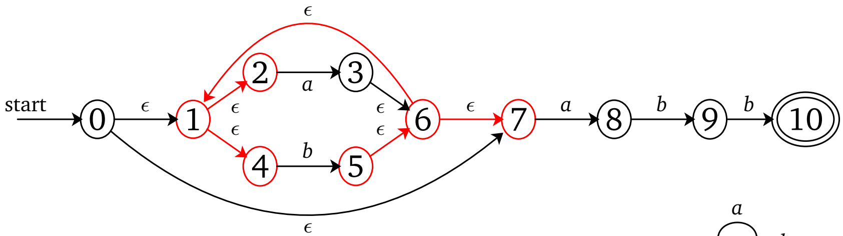
NFA → DFA Translation for $(a|b)^*abb$



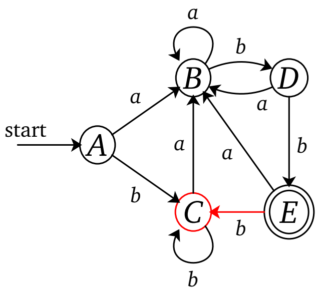
$$DTran[E,b] = \epsilon\text{-}closure(move(E,b))$$
$$= \epsilon\text{-}closure(\{5\})$$
$$= \{1,2,4,5,6,7\} = C$$



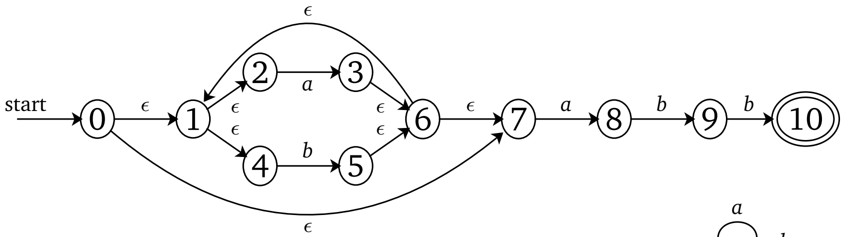
NFA → DFA Translation for $(a|b)^*abb$



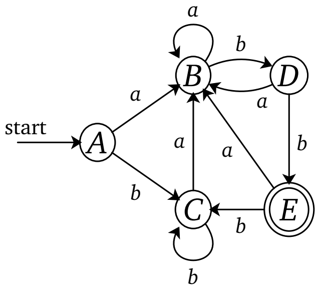
$$\begin{aligned} DTran[E,b] &= \epsilon\text{-}closure(move(E,b)) \\ &= \epsilon\text{-}closure(\{5\}) \\ &= \{1,2,4,5,6,7\} = C \end{aligned}$$



NFA → DFA Translation for $(a|b)^*abb$

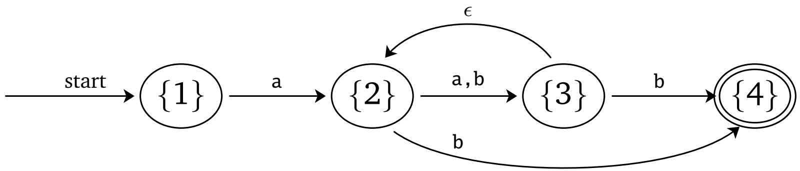


$$\Rightarrow$$



Exercise for NFA accepting $a(a|b)^*b$

Create DFA by subset construction:



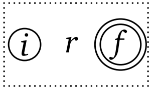
Comparing DFA and NFA properties

	DFA	*NFA*
labels from same state	unique complete	may be incomplete
ϵ-labels	no	yes
transition table	states	state sets
expressive power	same as RE	same as RE
trade-offs	easy to run	easy to design

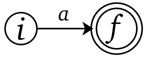
Definition: RE $\rightarrow$ NFA translation

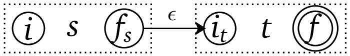
Constructing NFA from RE (Algorithm 3.23): Regular expression r to NFA with initial/final states i, f:

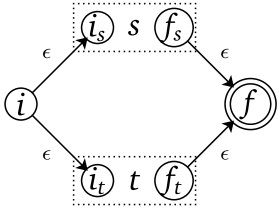
$$[r]_{i,f} \equiv$$

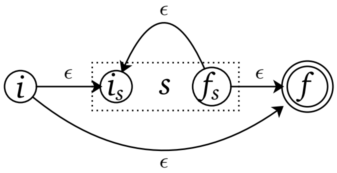


Definition: RE → NFA translation

$[a]_{i,f} \equiv$


$[st]_{i,f} \equiv$


$[s|t]_{i,f} \equiv$


$[s^*]_{i,f} \equiv$


Exercise: RE $\rightarrow$ NFA translation

Construct an NFA that accepts the same language as **(a|b)*abb** using Algorithm 3.23.