

CS3063 Theory of Computing

Semester 4 (20 Intake), Feb – Jun 2023

Lecture 4

Regular Languages & Finite Automata
– Part 3

Sanath Jayasena

Announcements

- **Assignment 1**: on Moodle, due on 24th April

Today's Outline

Lecture 4

- NFA- Λ (NFA with Λ -transitions)
- Equivalence among FA, NFA, NFA- Λ
- **RE** \rightarrow **FA**, via Thompson's Algorithm

PART 1

Today's Outline

Lecture 4

- **NFA- Λ (NFA with Λ -transitions)**
- Equivalence among FA, NFA, NFA- Λ
- $RE \rightarrow FA$, via Thompson's Algorithm

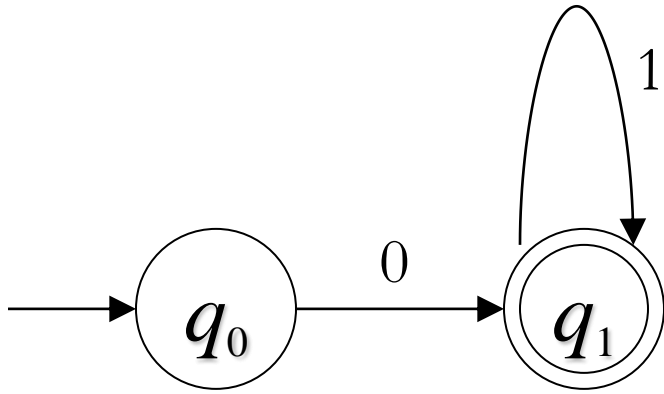
Review

- Regular expressions/languages
- Finite automata (FA): deterministic version, DFA
- A language is regular iff it is the set of strings accepted by some FA
- $FA \leftrightarrow RE$
- Set operations on languages
- NFA
- Equivalence between NFA and DFA

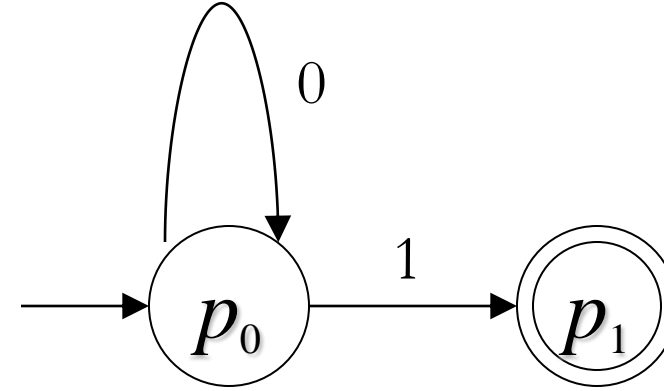
NFA with Λ -transitions (NFA- Λ)

- NFA with even more freedom
 - Transitions without any input (i.e., with Λ)
 - More general than NFA
 - Denoted by **NFA- Λ**
- But **no more powerful than NFA (or DFA) with respect to languages accepted!**

Example

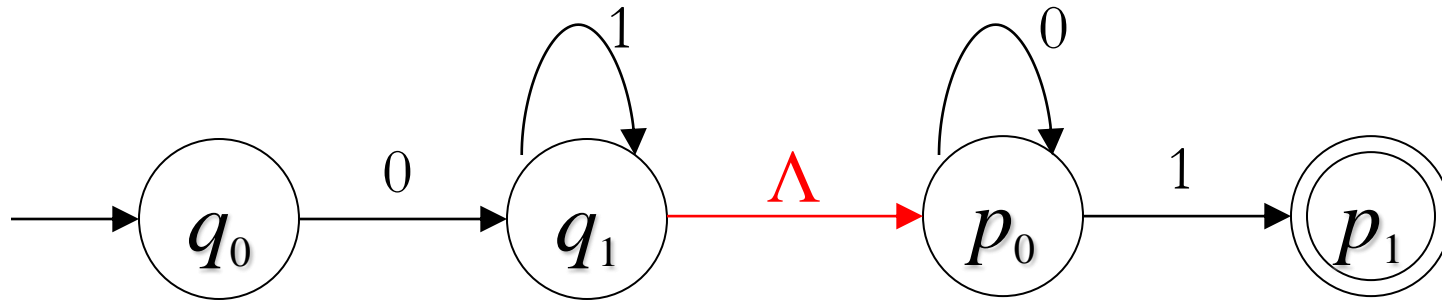


(a) NFA accepting 01^*

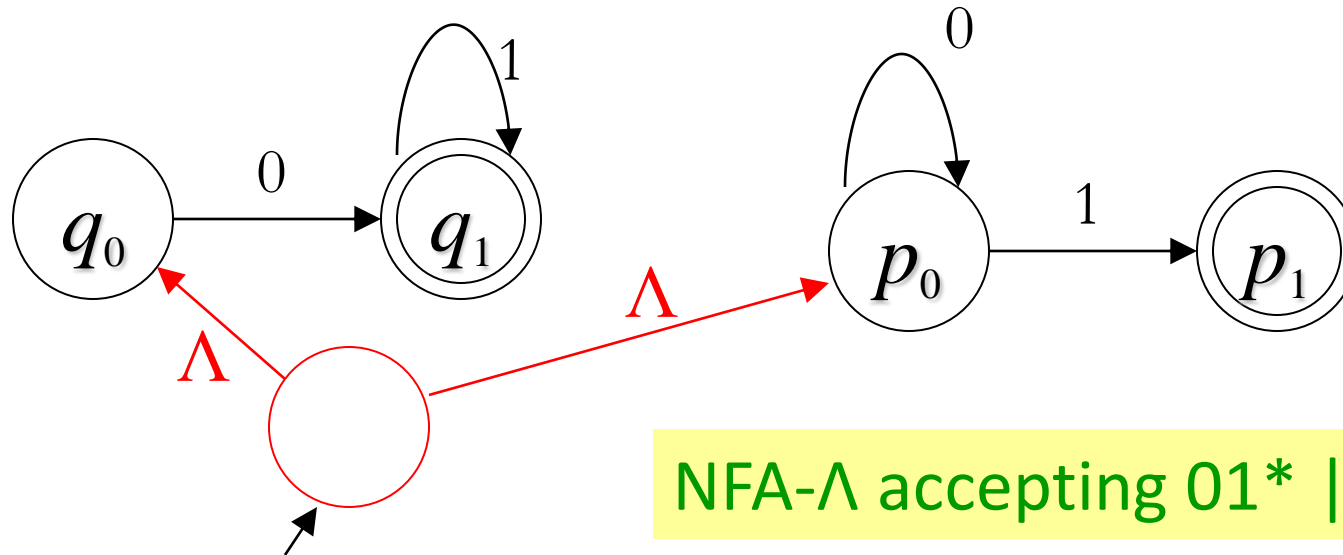


(b) NFA accepting 0^*1

Example ...contd



NFA- Λ accepting 01^*0^*1



NFA- Λ accepting $01^* \mid 0^*1$

NFA- Λ : Definition

- NFA- Λ is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where:
 - Q, Σ, q_0 and A are the same as for an NFA
 - But ...
 - δ , the transition function, maps $Q \times (\Sigma \cup \{\Lambda\})$ to 2^Q

Extended Transition Function δ^*

- Can we extend δ , as with NFA, to obtain δ^* that describes the status of an NFA- Λ on an input string x ?
- Not as easy to define recursively, because Λ -transitions are involved
- Let us define the notion of **Λ -closure** first

Λ -Closure

- **Definition:** Let $M=(Q, \Sigma, q_0, A, \delta)$ be an NFA- Λ and S be a subset of Q . The **Λ -closure** of S is the set $\Lambda(S)$ defined as follows:
 - Every element of S is an element of $\Lambda(S)$
 - For any q in $\Lambda(S)$, every element of $\delta(q, \Lambda)$ is in $\Lambda(S)$
 - No other element of Q is in $\Lambda(S)$
- For a state q , $\Lambda(q)$ is the set of all states that can be reached from q using Λ -transitions
 - There is a path labeled Λ from q to all such states

Λ -Closure ...contd

- If $\delta^*(q, y)$ is the set of all the states that can be reached from q using the symbols of y and Λ -transitions, then

$$R = \bigcup_{r \in \delta^*(q, y)} \delta(r, a)$$

is the set of states R we can reach in one more step by using the symbol a

- The Λ -closure of this R includes any additional states that we can reach with Λ -transitions subsequently

Defining δ^* for an NFA- Λ

- **Definition:** Let $M=(Q, \Sigma, q_0, A, \delta)$ be an NFA- Λ ; the function $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ is such that:
 - For any $q \in Q$, $\delta^*(q, \Lambda) = \Lambda(\{q\})$
 - For any $q \in Q$, $y \in \Sigma^*$ and, $a \in \Sigma$

$$\delta^*(q, ya) = \Lambda \left(\bigcup_{r \in \delta^*(q, y)} \delta(r, a) \right)$$

A string x is **accepted** by M if

$$\delta^*(q_0, x) \cap A \neq \emptyset$$

PART 2

Today's Outline

Lecture 4

- NFA- Λ (NFA with Λ -transitions)
- **Equivalence among FA, NFA, NFA- Λ**
- RE \rightarrow FA, via Thompson's Algorithm

Equivalence of NFA- Λ & NFA

- **Theorem:** For an NFA- Λ , $M=(Q, \Sigma, q_0, A, \delta)$ accepting a language $L \subseteq \Sigma^*$, there is an NFA, $M_1=(Q_1, \Sigma, q_1, A_1, \delta_1)$ that accepts L
- Proof involves showing M_1 will be the NFA $(Q, \Sigma, q_0, A_1, \delta_1)$ where, for $q \in Q$ and $a \in \Sigma$, $\delta_1(q, a) = \delta^*(q, a)$ and

$$A_1 = \begin{cases} A \cup \{q_0\} & \text{if } \Lambda(\{q_0\}) \cap A \neq \emptyset \text{ in } M \\ A & \text{otherwise} \end{cases}$$

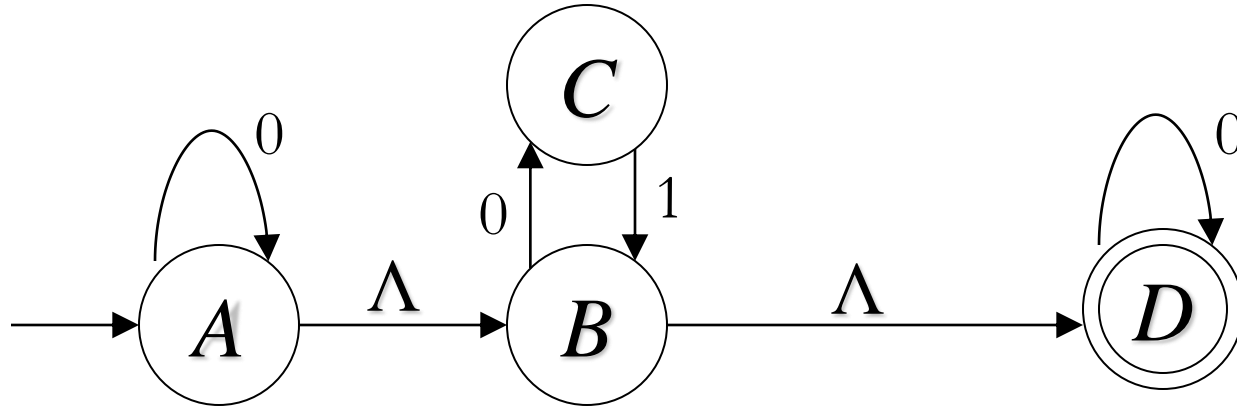
NFA Equivalent to NFA- Λ : How?

- From the theorem on equivalency
 - Proof gives a method to obtain equivalent NFA
 - Proof by induction (on length of input string)
- Method based on eliminating Λ -transitions without changing states
 - E.g., if we have $p \rightarrow q$ for 0 and $q \rightarrow r$ for Λ , eliminate the Λ -transition and add $p \rightarrow r$ for 0
 - The transition function δ_1 shows how this is done

Constructing an NFA Equivalent to an NFA- Λ ...contd

- For a state q and symbol a , $\delta^*(q, a)$ is the set of states that can be reached from q , using a and Λ -transitions before and after
- Can say similarly for a string x
- Specifying the accepting states A_1 is to be done with care
 - Check whether it is possible to go to one of A from q_0 using only Λ -transitions
 - If yes, make q_0 an element of A_1

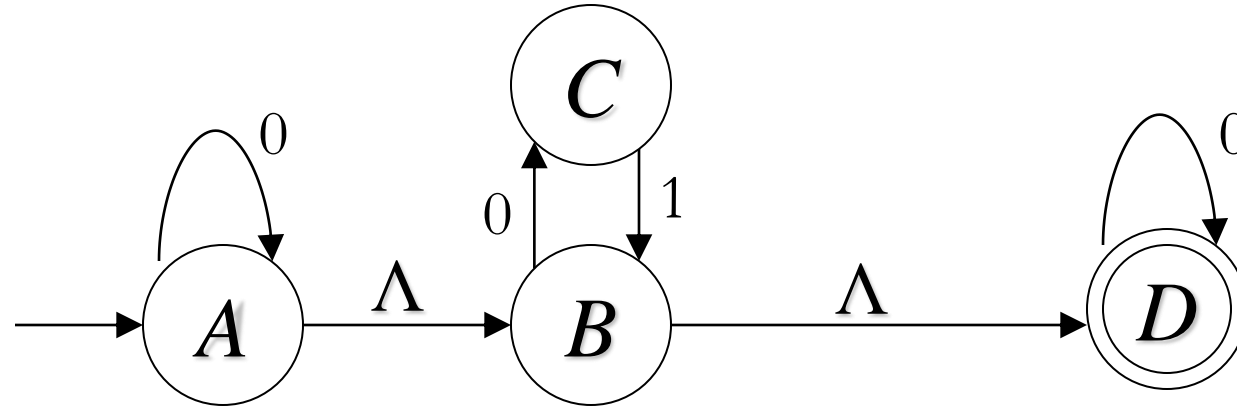
Example: NFA- Λ accepting $0^*(01)^*0^*$



Find an
equivalent
NFA

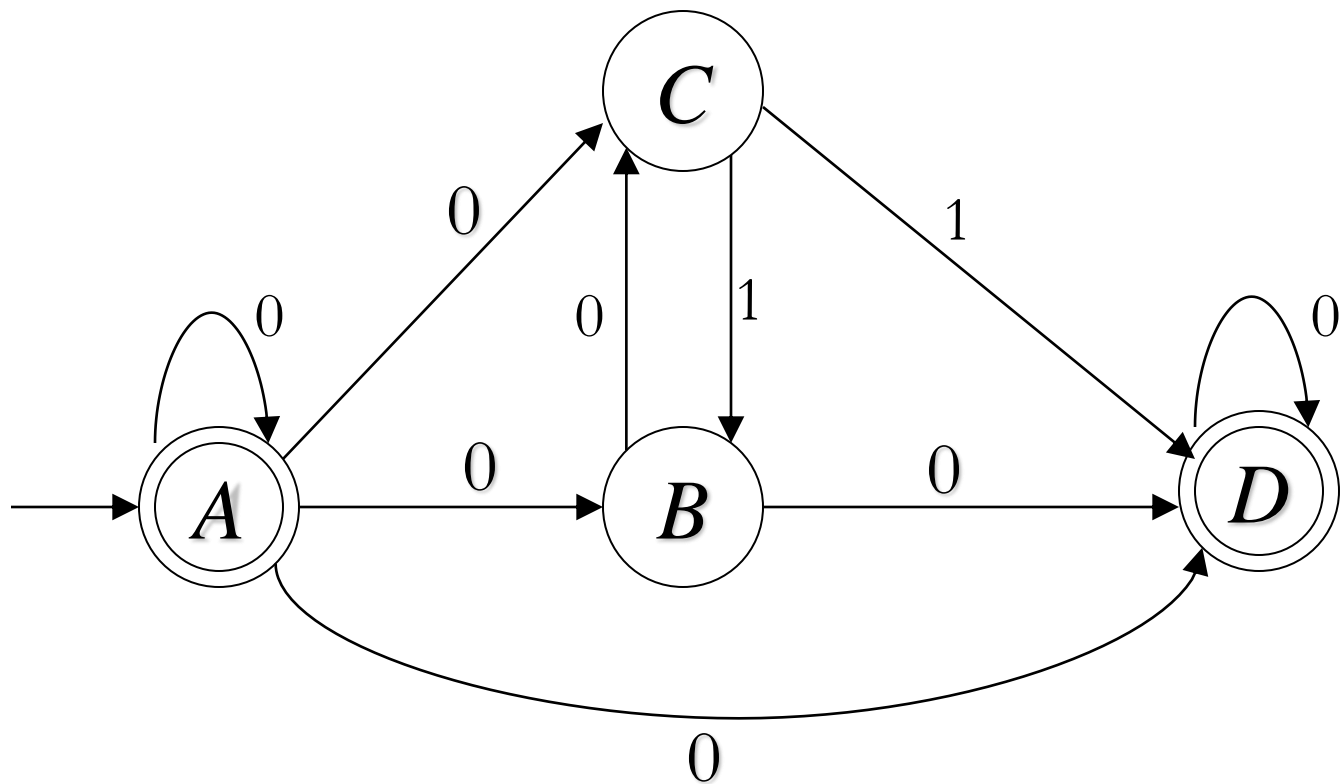
q	$\delta(q, \Lambda)$	$\delta(q, 0)$	$\delta(q, 1)$
A	$\{B\}$	$\{A\}$	\emptyset
B	$\{D\}$	$\{C\}$	\emptyset
C	\emptyset	\emptyset	$\{B\}$
D	\emptyset	$\{D\}$	\emptyset

Solution



q	$\delta(q, \Lambda)$	$\delta(q, 0)$	$\delta(q, 1)$	$\delta^*(q, 0)$	$\delta^*(q, 1)$
A	$\{B\}$	$\{A\}$	\emptyset	$\{A, B, C, D\}$	\emptyset
B	$\{D\}$	$\{C\}$	\emptyset	$\{C, D\}$	\emptyset
C	\emptyset	\emptyset	$\{B\}$	\emptyset	$\{B, D\}$
D	\emptyset	$\{D\}$	\emptyset	$\{D\}$	\emptyset

Solution ...contd



Summary on FA

- Theorem
 - For any alphabet Σ , and a language $L \subset \Sigma^*$, the following statements are equivalent:
 1. L can be recognized by a DFA
 2. L can be recognized by an NFA
 3. L can be recognized by an NFA- Λ

Kleene's Theorem (again)

- *A language L is regular if and only if there is an FA that accepts L*
- Part 1
 - Any regular language can be recognized by an FA
- Part 2
 - The language accepted by any FA is regular

PART 3

Today's Outline

Lecture 4

- NFA- Λ (NFA with Λ -transitions)
- Equivalence among FA, NFA, NFA- Λ
- **RE \rightarrow FA, via Thompson's Algorithm**

RE \rightarrow DFA : How? (Again)

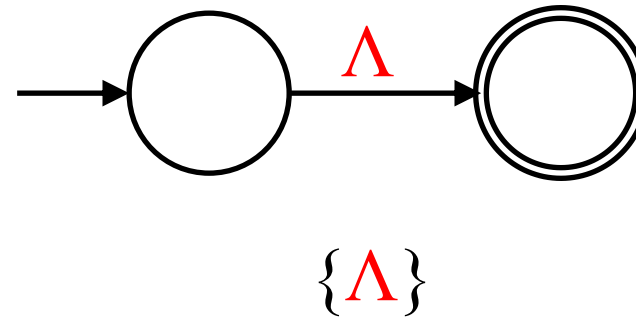
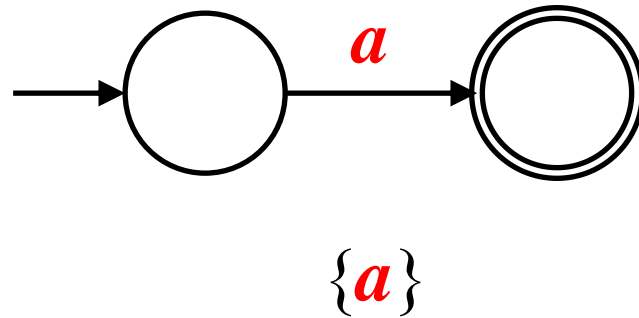
- Steps
 1. Thompson's construction: RE \rightarrow NFA- Λ
 2. Convert NFA- $\Lambda \rightarrow$ NFA \rightarrow DFA
- Thompson's construction
 - Also known as the McNaughton-Yamada-Thompson algorithm

Thompson's Construction

- Input: regular expression r over alphabet Σ
- Output: NFA- Λ for r
- Method
 - Break r into subexpressions, recursively until no operators are present in them
 - (get basic regular languages as elements)
 - Apply inductive rules to construct larger NFA- Λ for an expression from NFA- Λ of its subexpressions

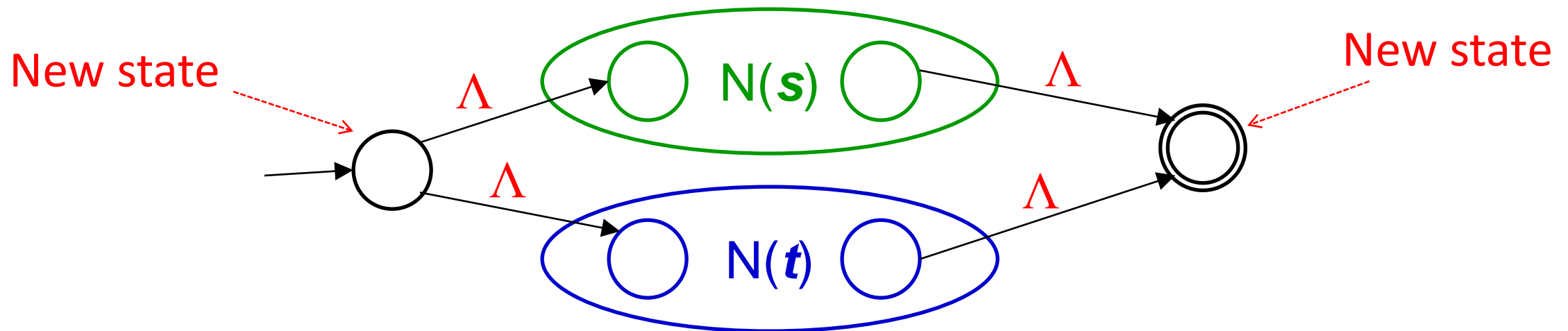
Thompson's Construction: Basis

- Basic languages: $\{a\}$, $\{\Lambda\}$, \emptyset (given a in Σ)
- Their NFA- Λ



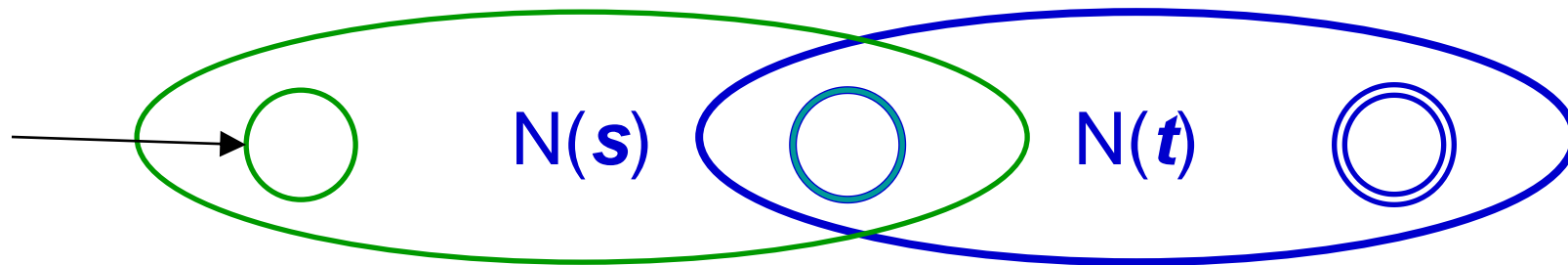
Thompson's Construction: Induction

- Suppose $N(\mathbf{s})$ and $N(\mathbf{t})$ are NFA- Λ for regular expressions \mathbf{s} and \mathbf{t} , respectively
- **Case (a)**: $\mathbf{r} = \mathbf{s} \mid \mathbf{t}$ (Union)
 - $N(\mathbf{r})$, the NFA- Λ for \mathbf{r} is constructed as follows



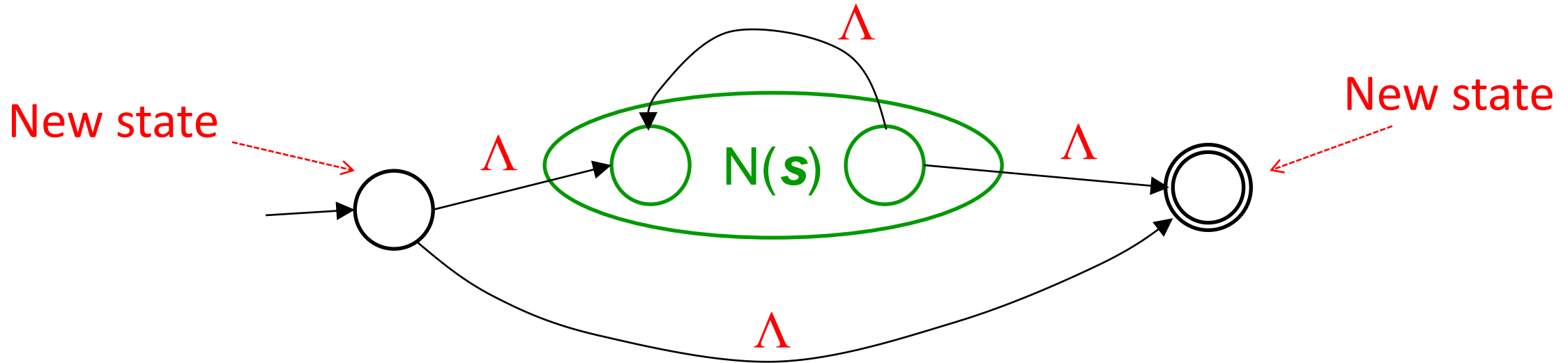
Thompson's Construction: Induction

- Suppose $N(\mathbf{s})$ and $N(\mathbf{t})$ are NFA- Λ for regular expressions \mathbf{s} and \mathbf{t} , respectively
- **Case (b):** $\mathbf{r} = \mathbf{st}$ (Concatenation)
 - $N(\mathbf{r})$, the NFA- Λ for \mathbf{r} is constructed as follows



Thompson's Construction: Induction

- Suppose $N(s)$ is the NFA- Λ for regular expression s
- **Case (c):** $r = s^*$ (Closure)
 - $N(r)$, the NFA- Λ for r is constructed as follows

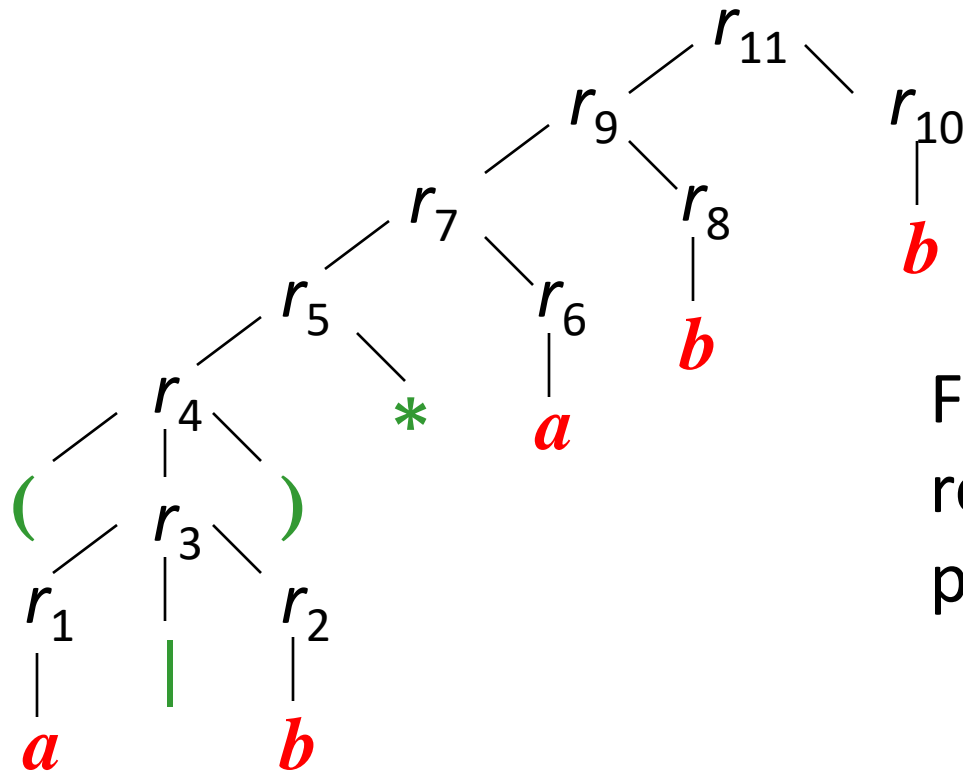


Thompson's Construction: Induction

- Suppose $N(\mathbf{s})$ is the NFA- Λ for regular expression \mathbf{s}
- **Case (d):** $\mathbf{r} = (\mathbf{s})$
 - $N(\mathbf{r})$, the NFA- Λ for \mathbf{r} is the same as $N(\mathbf{s})$
- Note:
 - # states in $N(\mathbf{r}) \leq 2 \times \text{\# operators, operands}$
 - $N(\mathbf{r})$ has 1 start state (no incoming edge), 1 accepting state (no outgoing edge)

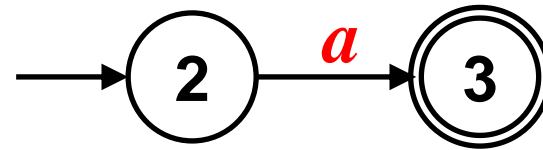
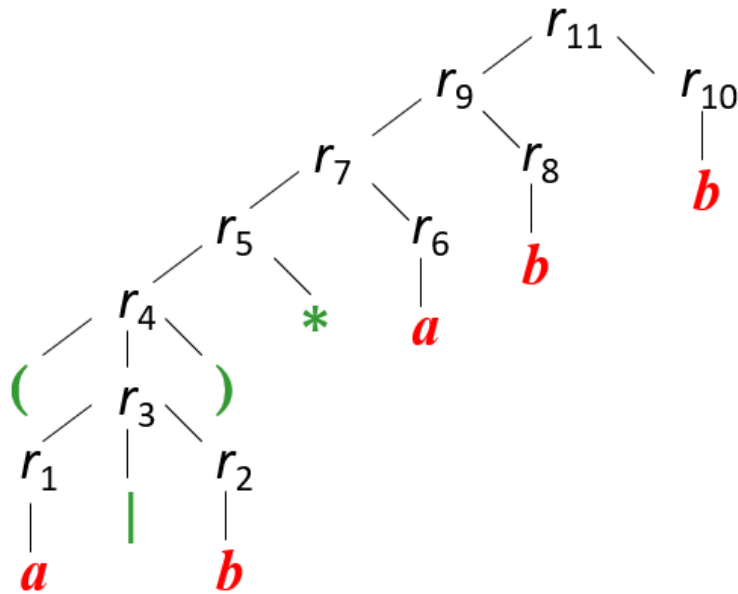
Example

- Use Thompson's construction algorithm to construct an NFA- Λ for $r = (a|b)^*abb$

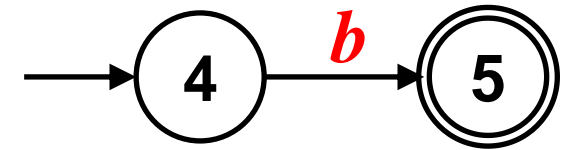


First, break r into subexpressions, recursively until no operators are present in them

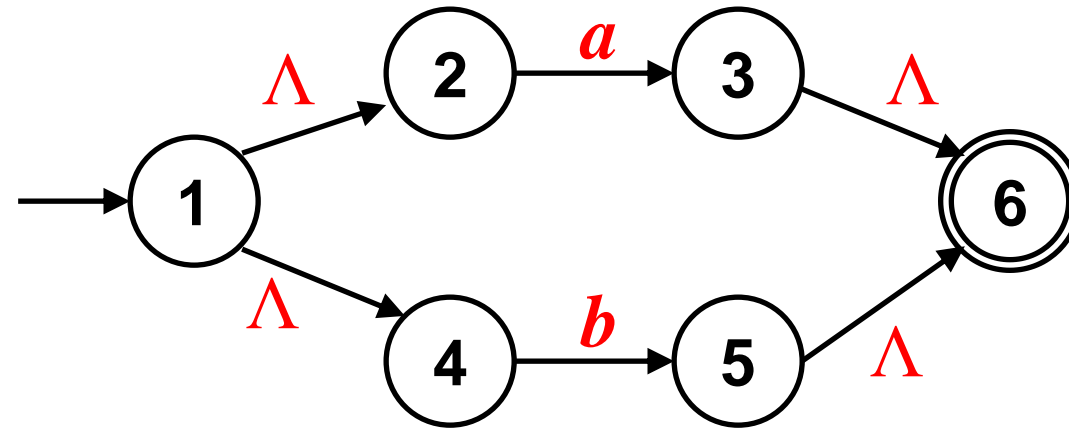
Solution (for $r = (a|b)^*abb$) ...



NFA- Λ for r_1

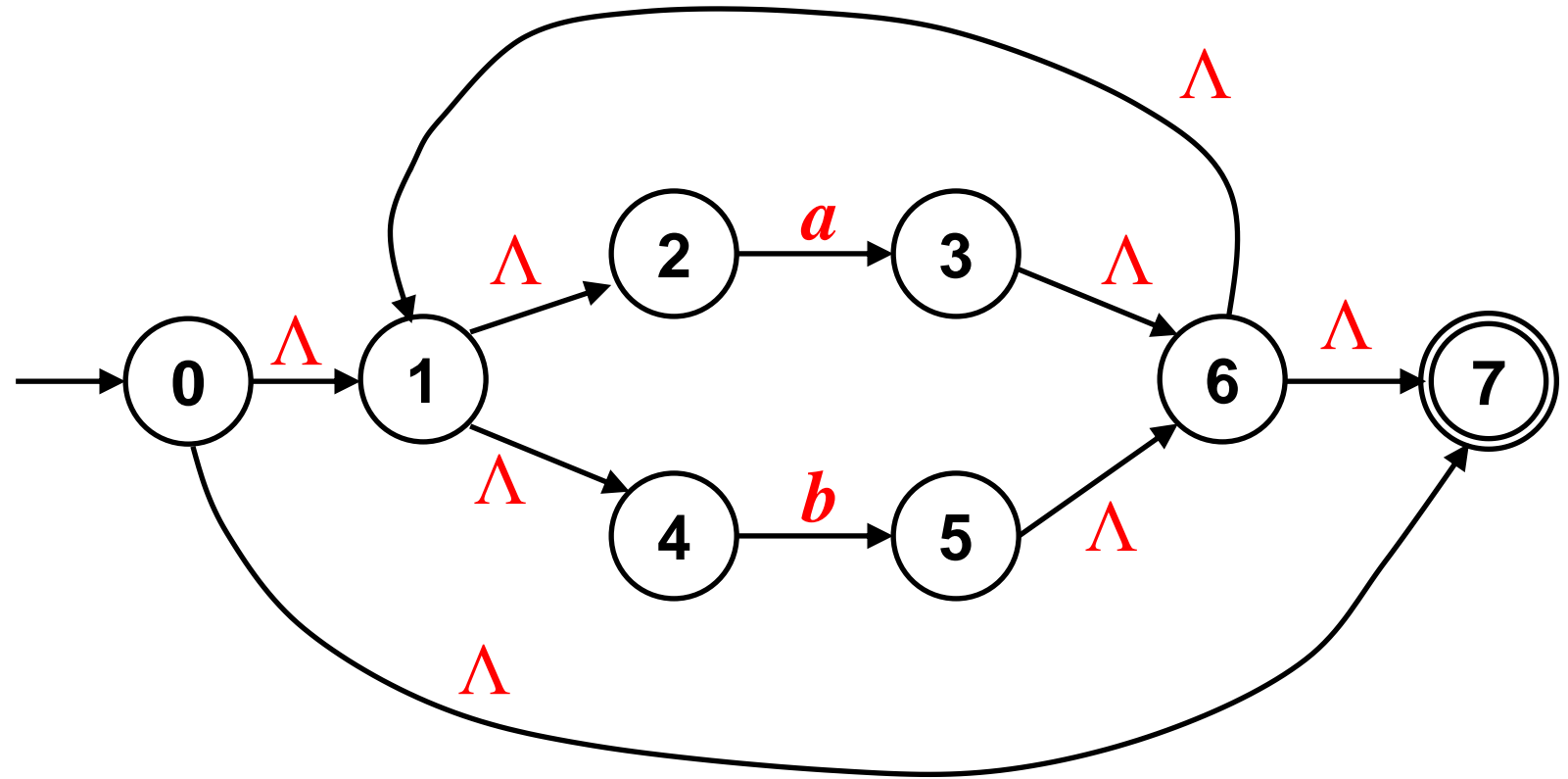
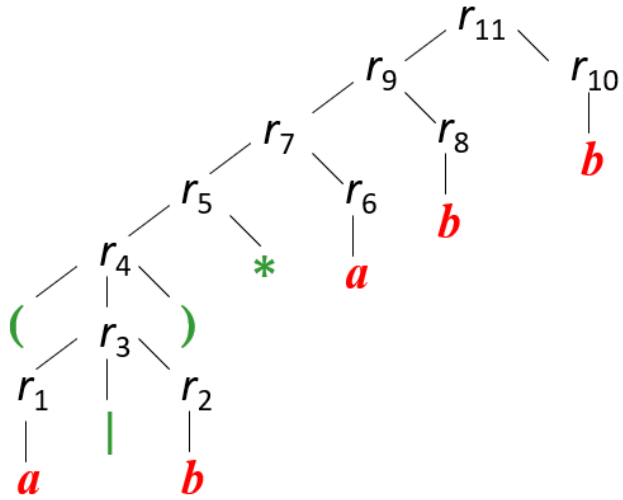


NFA- Λ for r_2



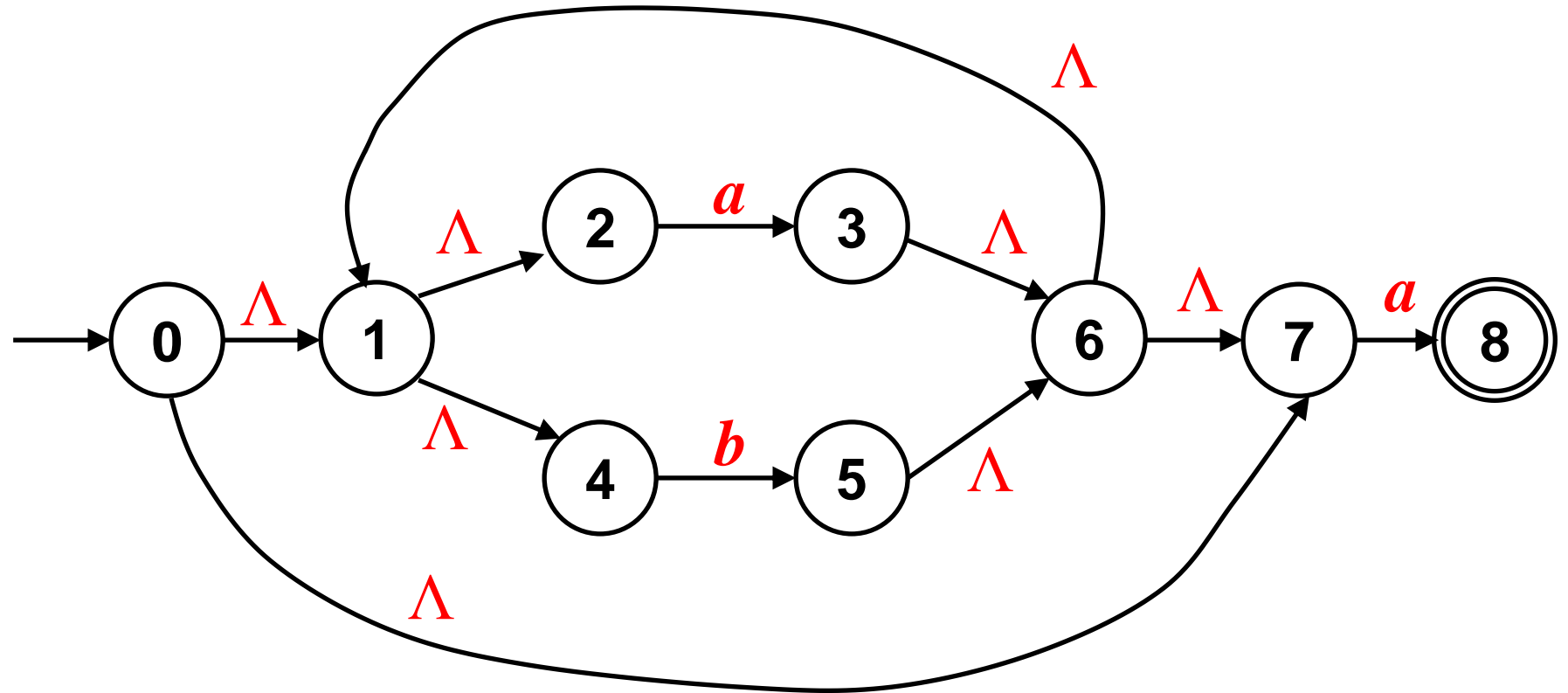
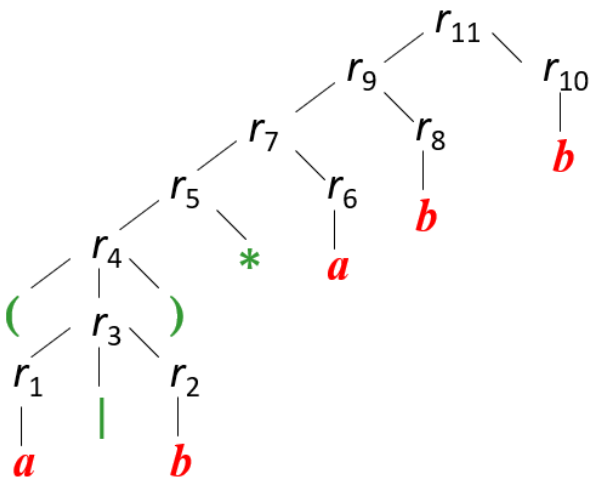
NFA- Λ for $r_3 = a|b$

Solution (for $r = (a|b)^*abb$) ...



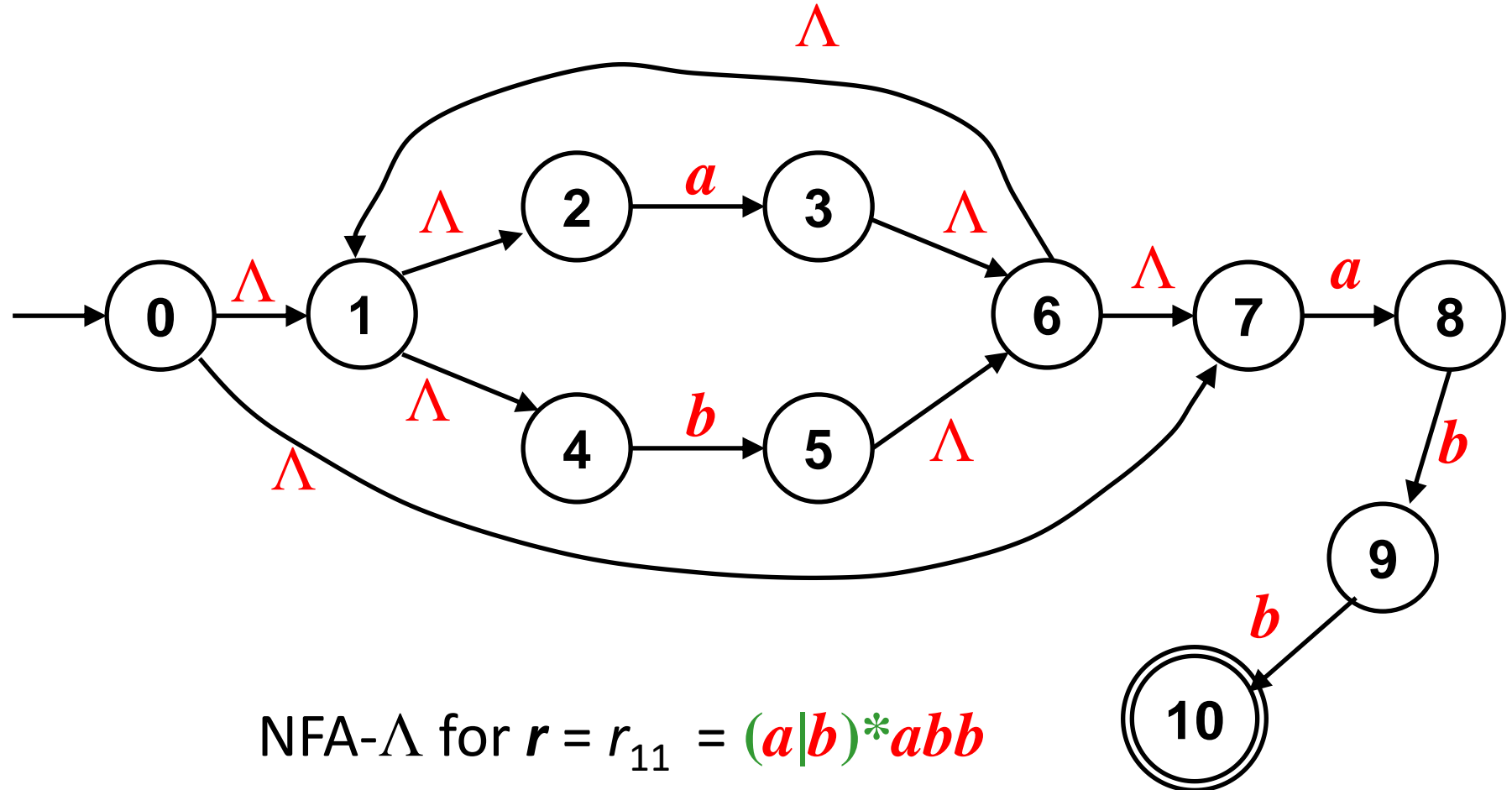
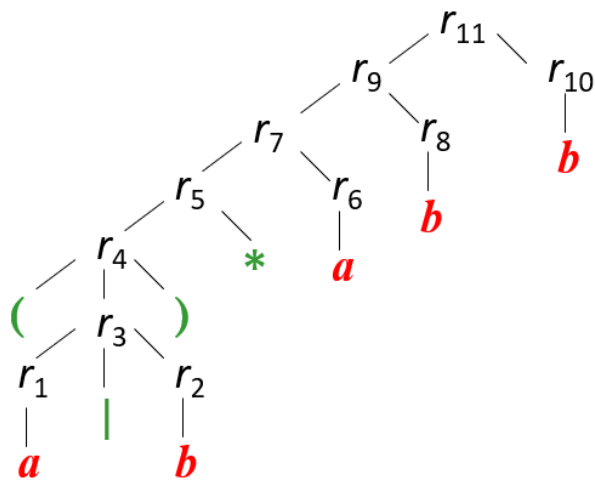
NFA- Λ for $r_5 = (a|b)^*$

Solution (for $r = (a|b)^*abb$) ...



NFA- Λ for $r_7 = (a|b)^*a$

Final Solution (for $r = (a|b)^*abb$)



Conclusion

- Today we discussed
 - NFA- Λ (NFA with Λ -transitions)
 - Converting NFA- Λ to equivalent NFA
 - Equivalence among DFA, NFA, NFA- Λ
 - Thompson's construction