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→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→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

this doesn't give extra computational power

NFA with Λ–transitions (NFA-Λ)

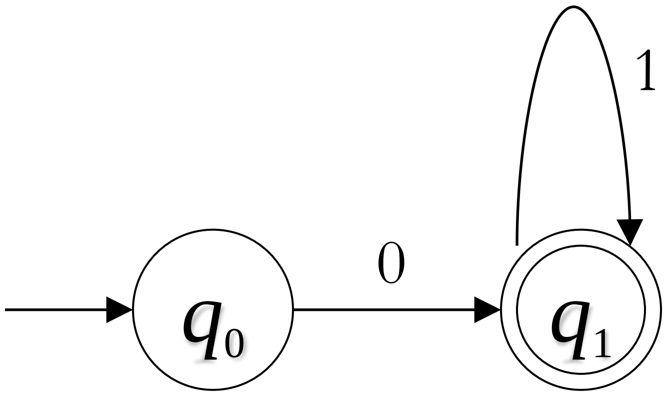
Some other books, Lamda will be epsilon

- NFA with even more freedom
  - Transitions without any input (i.e., with Λ)
  - More general than NFA
  - Denoted by NFA-Λ

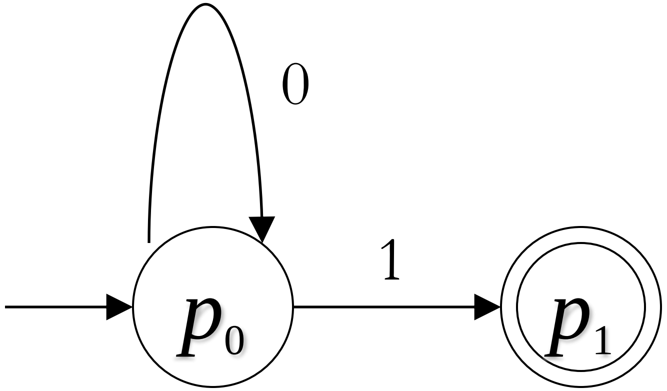
Capacity/Ability of do something based on
"is it done or not"
not on "how it does"

- But no more powerful than NFA (or DFA) with respect to languages accepted!

Example



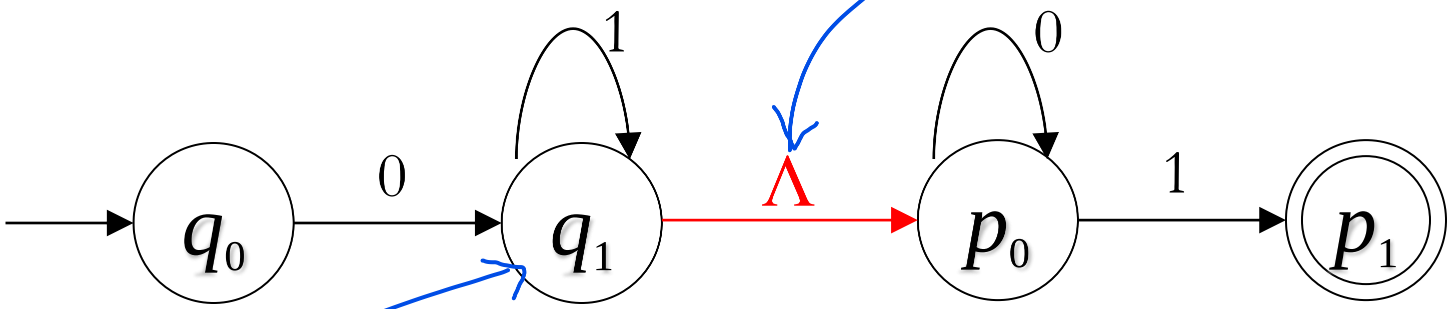
(a) NFA accepting 01*



(b) NFA accepting 0*1

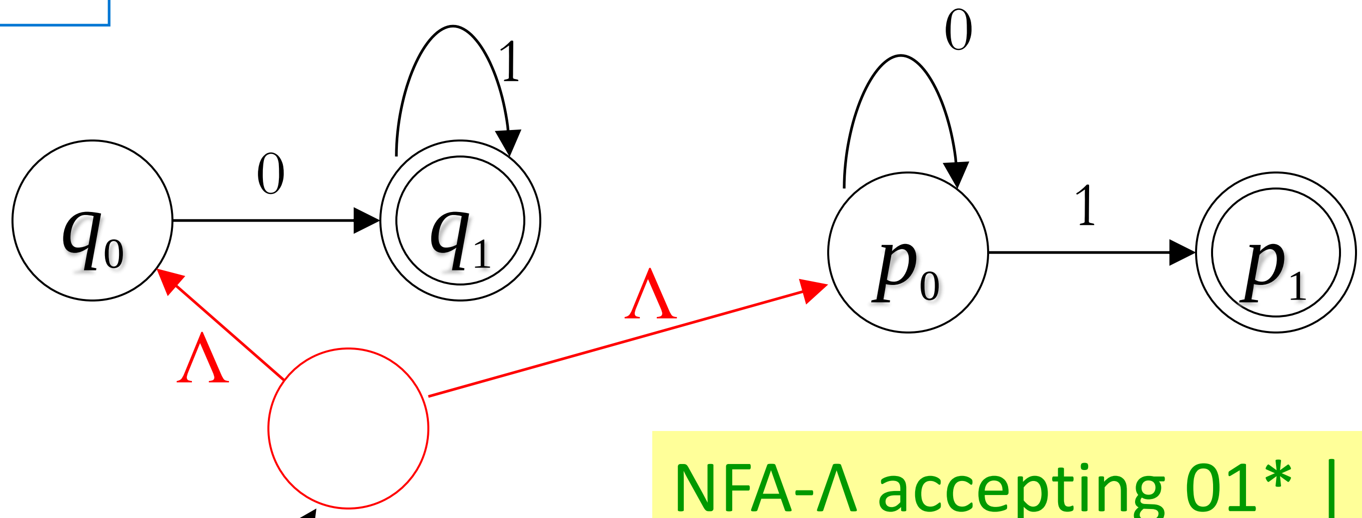
Example ...contd

You can go
(not "you must go")



NFA-Λ accepting 01*0*1

When we concat,
we remove the accepting
state here



NFA-Λ accepting 01* | 0*1

Give "OR"

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 $\boxed{Q \times (\underbrace{\Sigma \cup \{\Lambda\}}) \text{ to } 2^Q}$

set of input symbols includes {Lamda}

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

All
lamda
transitions
in lamda
closure

- **Definition**: Let M=(Q, Σ, q_0, A, δ) 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

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

DFA

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

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

All of them again inside the lamda closure

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→FA, via Thompson's Algorithm

Here we try to remove lamda transitions from the NFA-lamda to become a NFA

NFA => DFA , by removing non deterministics

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 $a \in \Sigma$, $\boxed{\delta_1(q,a) = \delta^*(q,a)}$ and

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

initial state itself an accepting state

only if the accepting states includes lamda transition in initial state

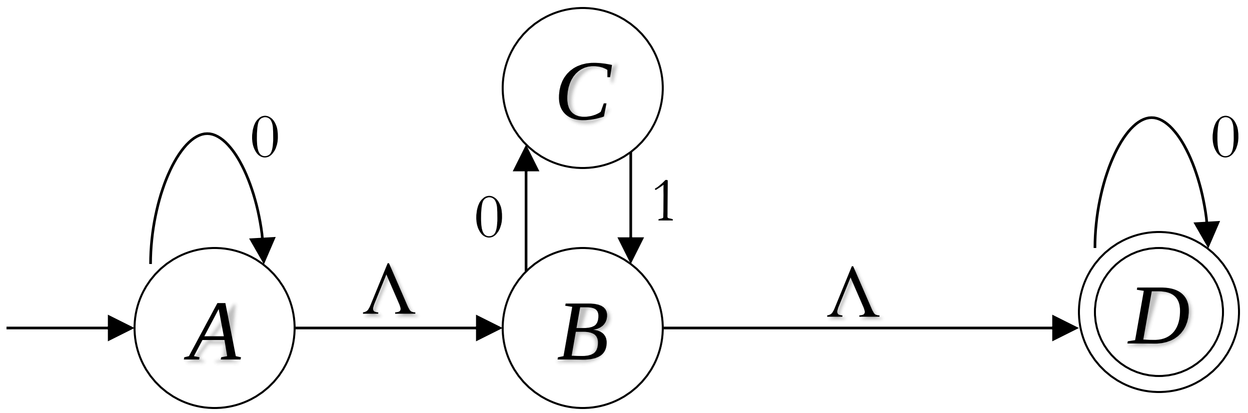
NFA Equivalent to NFA-Λ: How?

- 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→q for 0 and q→r for Λ, eliminate the Λ-transition and add p→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

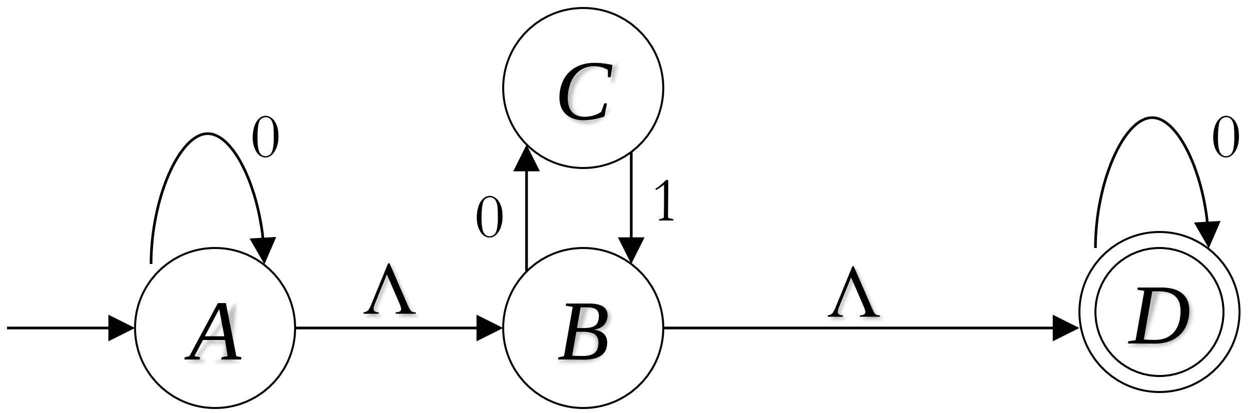
Example: NFA-Λ accepting 0*(01)*0*



Find an equivalent NFA

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

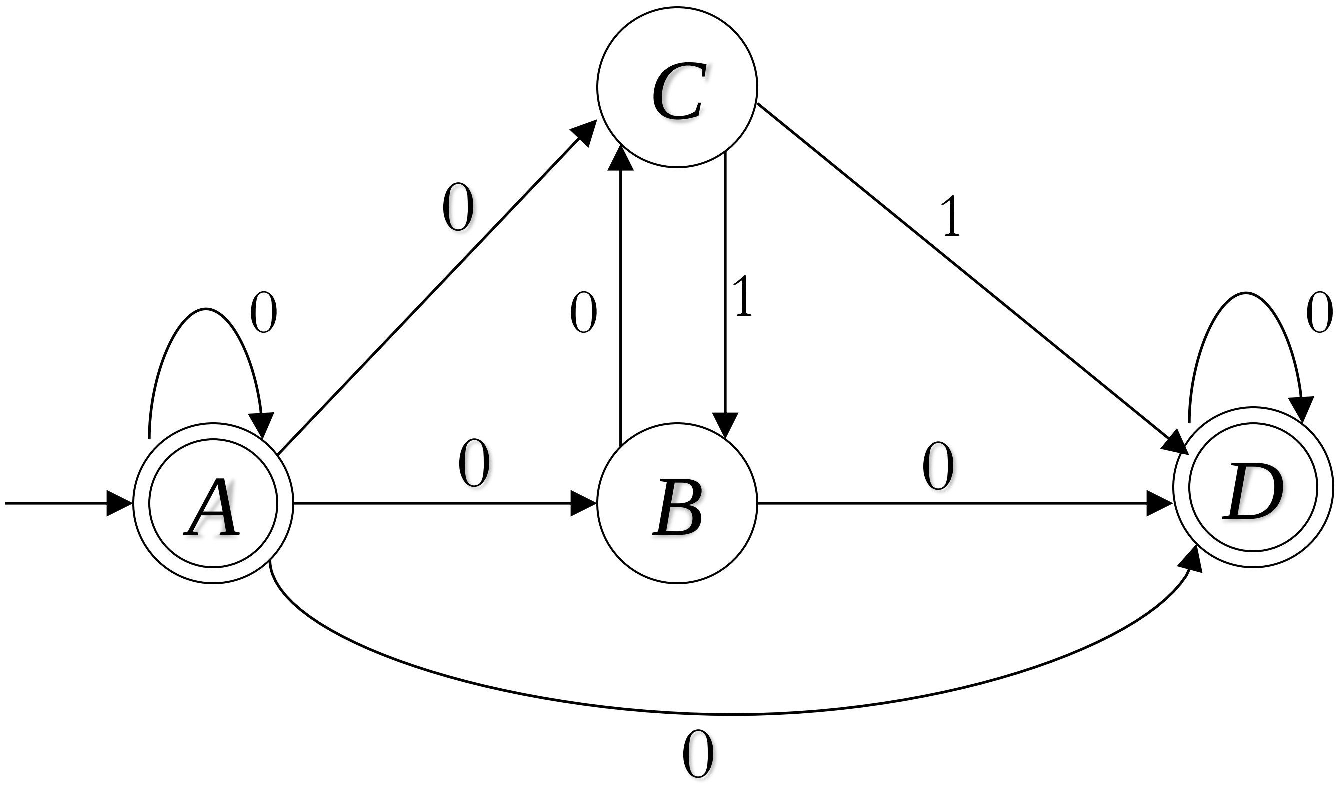
Solution



What states we can end up with
input 0 starting from A,
by looking at the diagram,

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→FA, via Thompson's Algorithm

When we have a DFA, how to minimize the number of states ???

RE→DFA : How? (Again)

- Steps
  1. Thompson's construction: RE → NFA-Λ
  2. Convert NFA-Λ → NFA → 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***

Union, Concat, Star

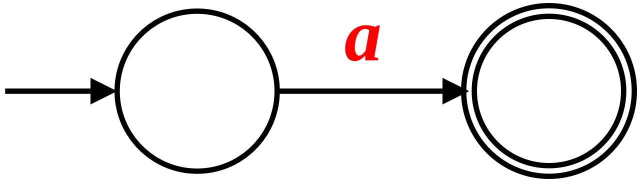
- Method
  - Break ***r*** into subexpressions, recursively until no operators are present in them
    - (get basic regular languages as elements)
  - Apply inductive rules to construct larger NFA-Λ for an expression from NFA-Λ of its subexpressions

Thompson's Construction: Basis

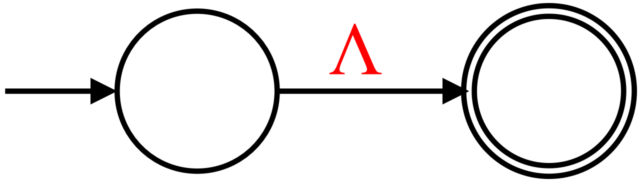
- Basic languages: $\{a\}$, $\{\Lambda\}$, $\emptyset$ (given a in Σ)

Can't represent, because there is nothing there

- Their NFA-Λ



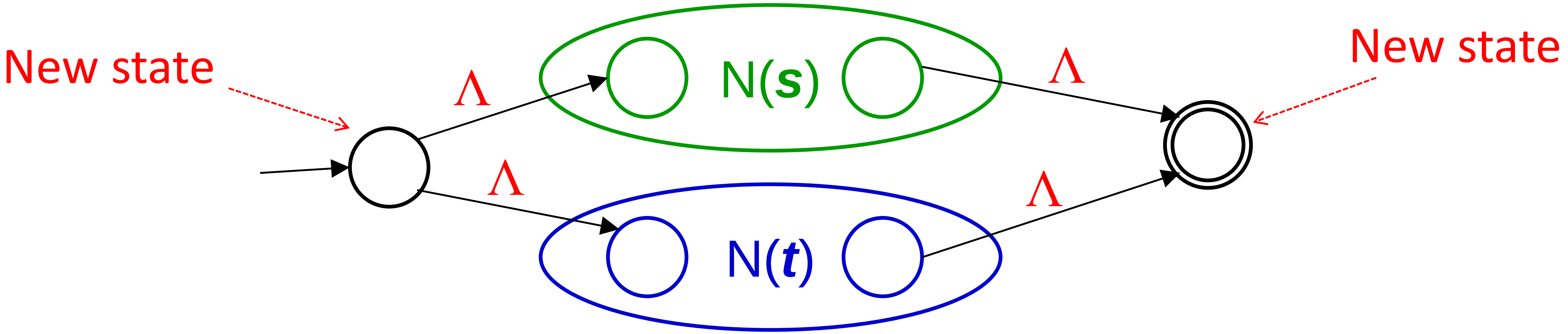
$\{a\}$



$\{\Lambda\}$

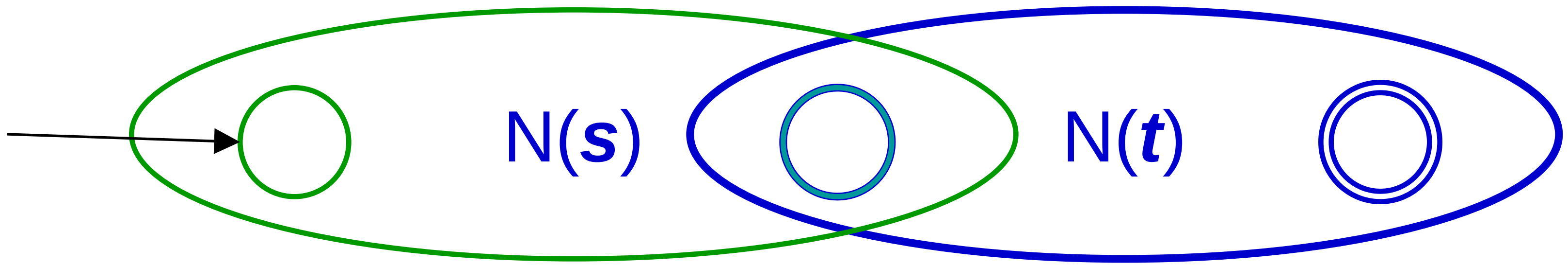
Thompson's Construction: Induction

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



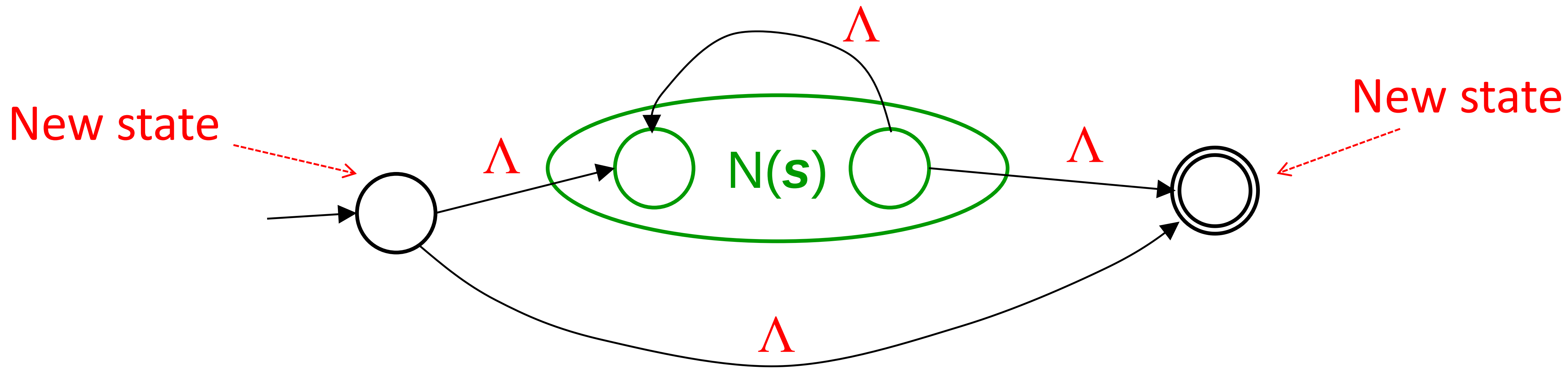
Thompson's Construction: Induction

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



Thompson's Construction: Induction

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



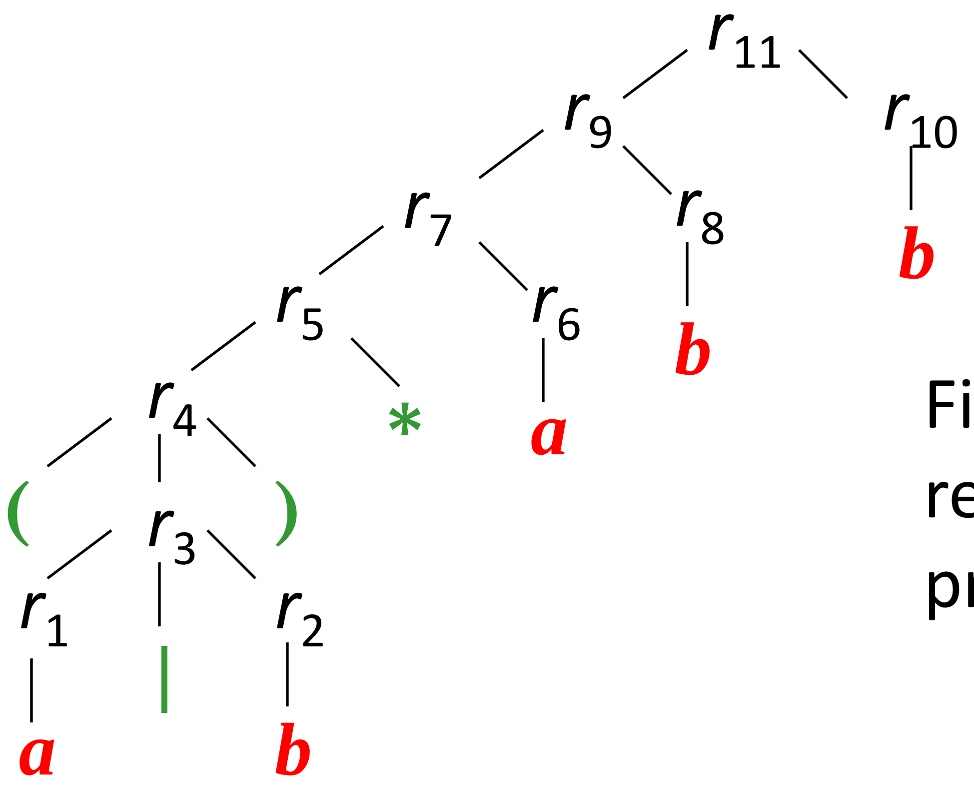
Thompson’s Construction: Induction

- Suppose N(***s***) is the NFA-Λ for regular expression ***s***
- **<u>Case (d)</u>**: ***r*** = (***s***)
  - N(***r***), the NFA-Λ for ***r*** is the same as N(***s***)

- Note:
  - # states in N(***r***) ≤ 2x # operators, operands
  - N(***r***) has 1 start state (no incoming edge), 1 accepting state (no outgoing edge)

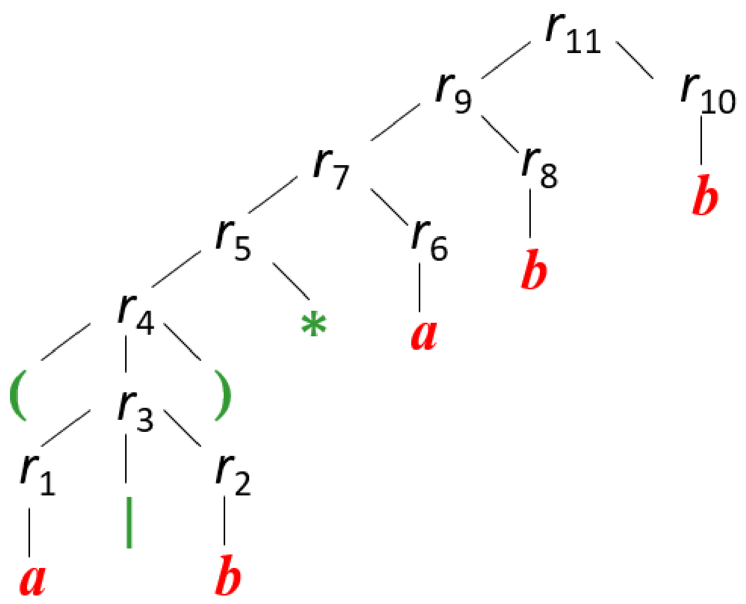
Example

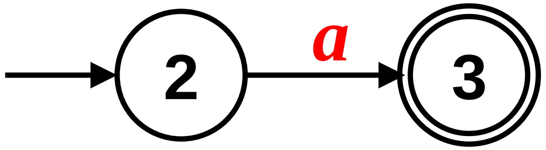
- Use Thompson's construction algorithm to construct an NFA-Λ for r = $(a|b)^*abb$



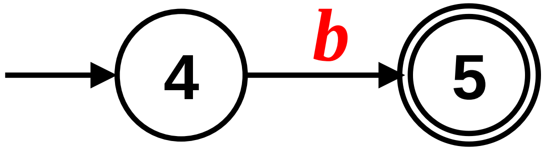
First, break r into subexpressions, recursively until no operators are present in them

Solution (for $r = (a|b)^*abb$) …

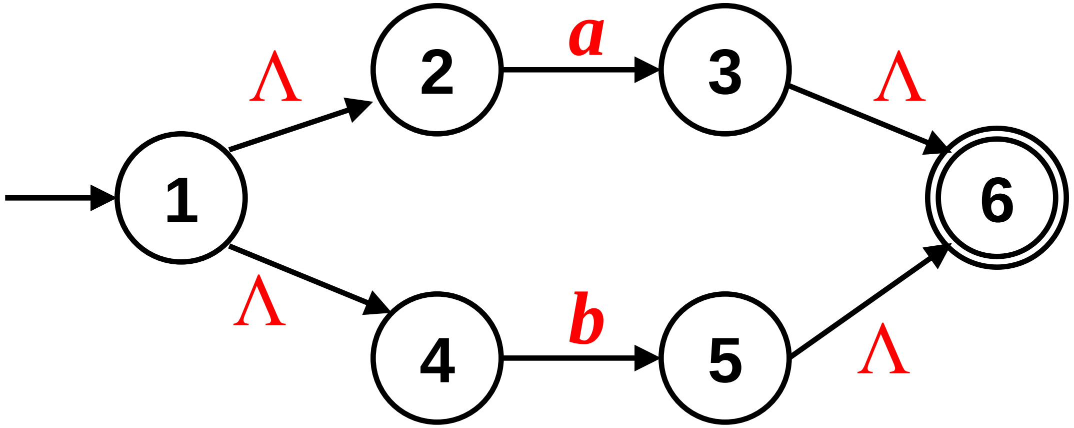




NFA-Λ for r_1

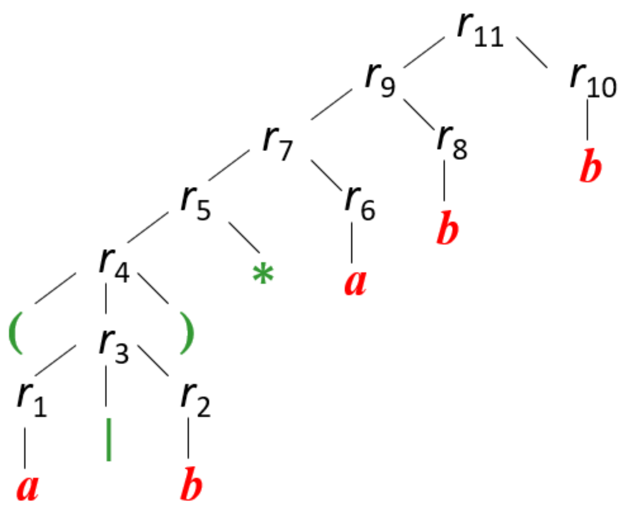


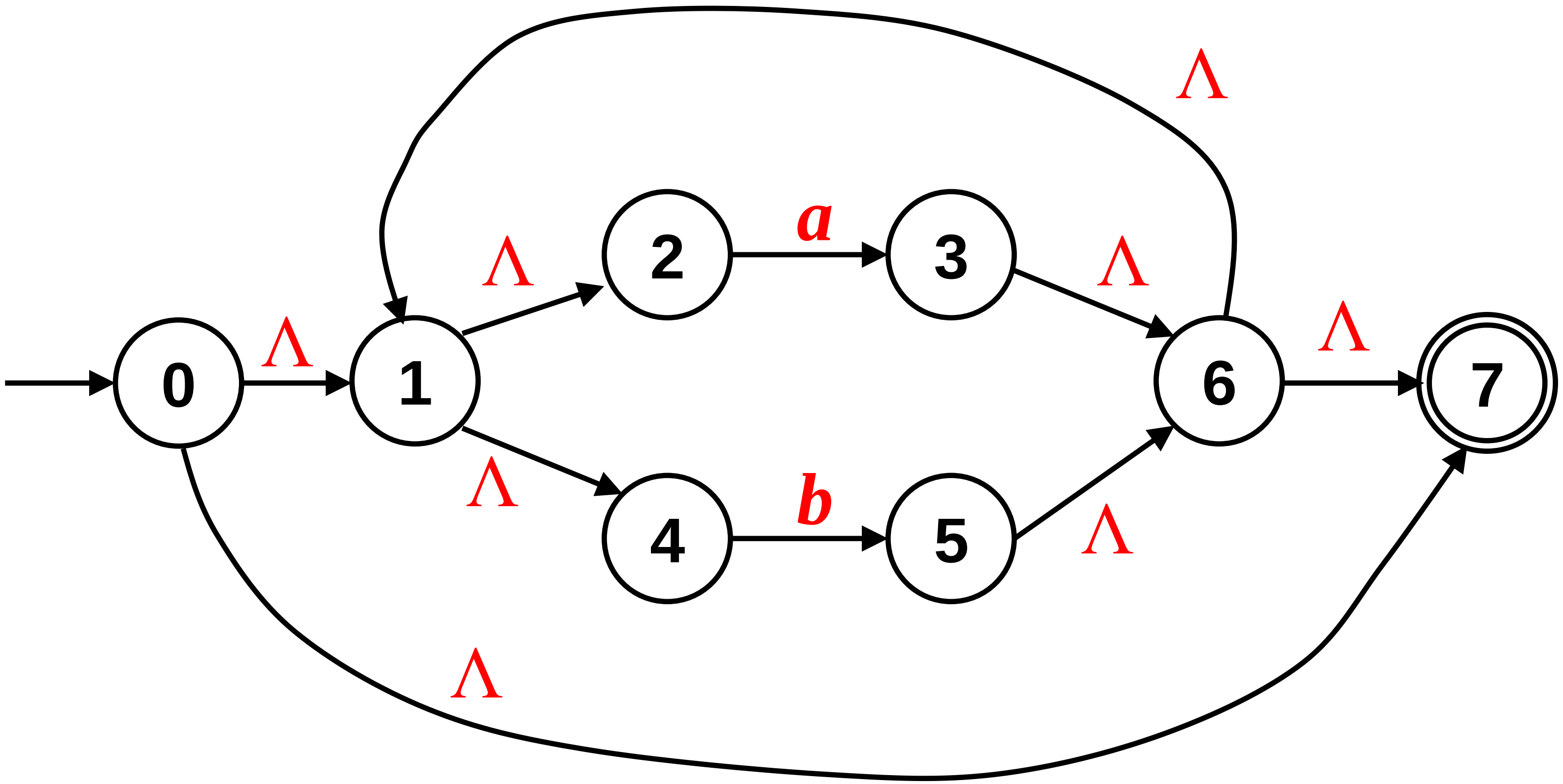
NFA-Λ for r_2



NFA-Λ for $r_3 = a|b$

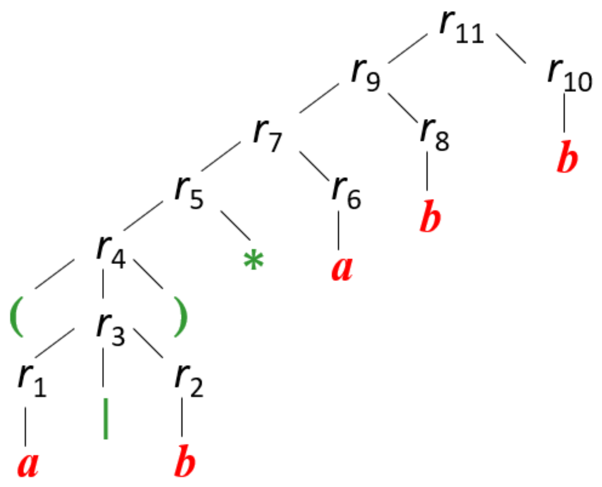
Solution (for r = $(a|b)^*abb$) …

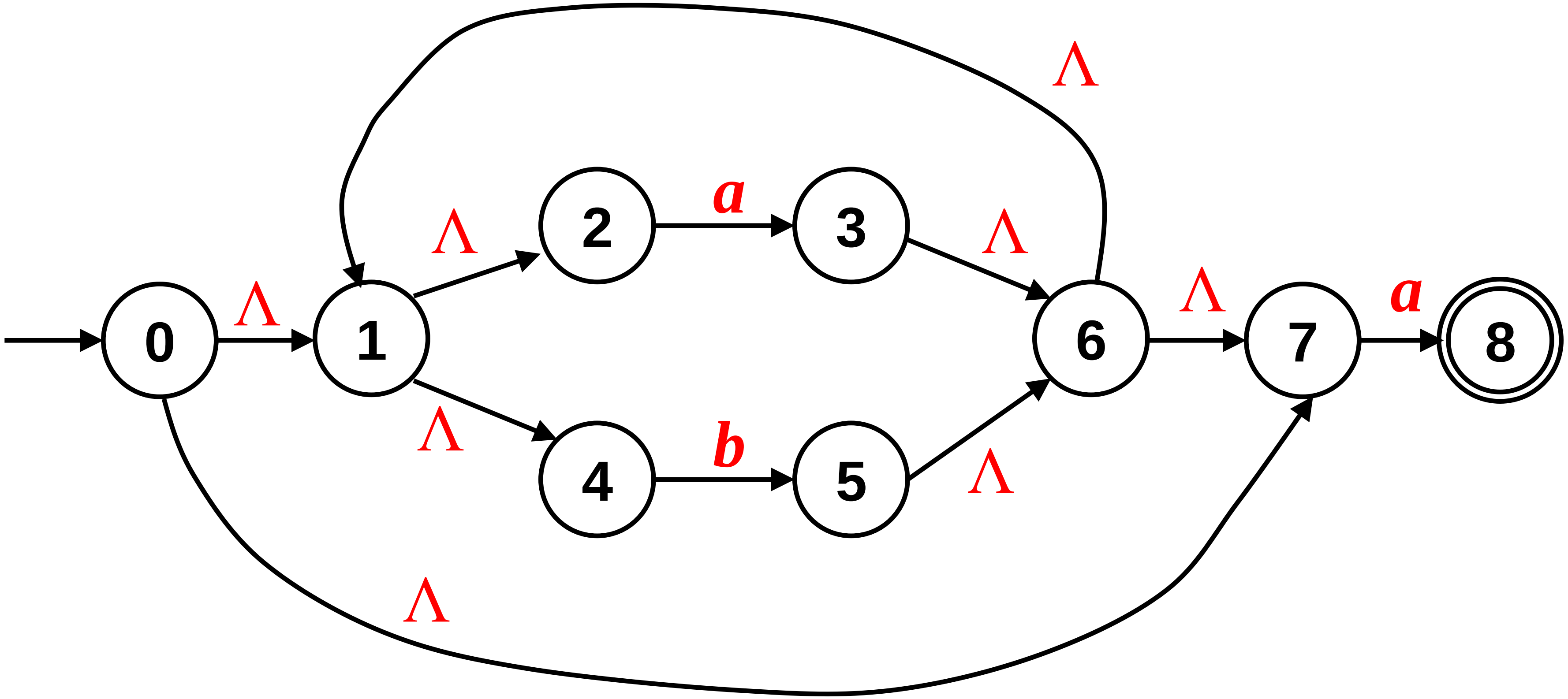




NFA-Λ for r_5 = $(a|b)^*$

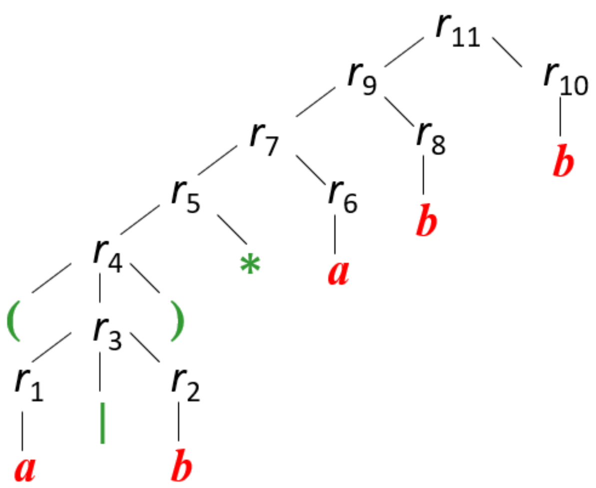
Solution (for r = ($a|b$)*abb) …

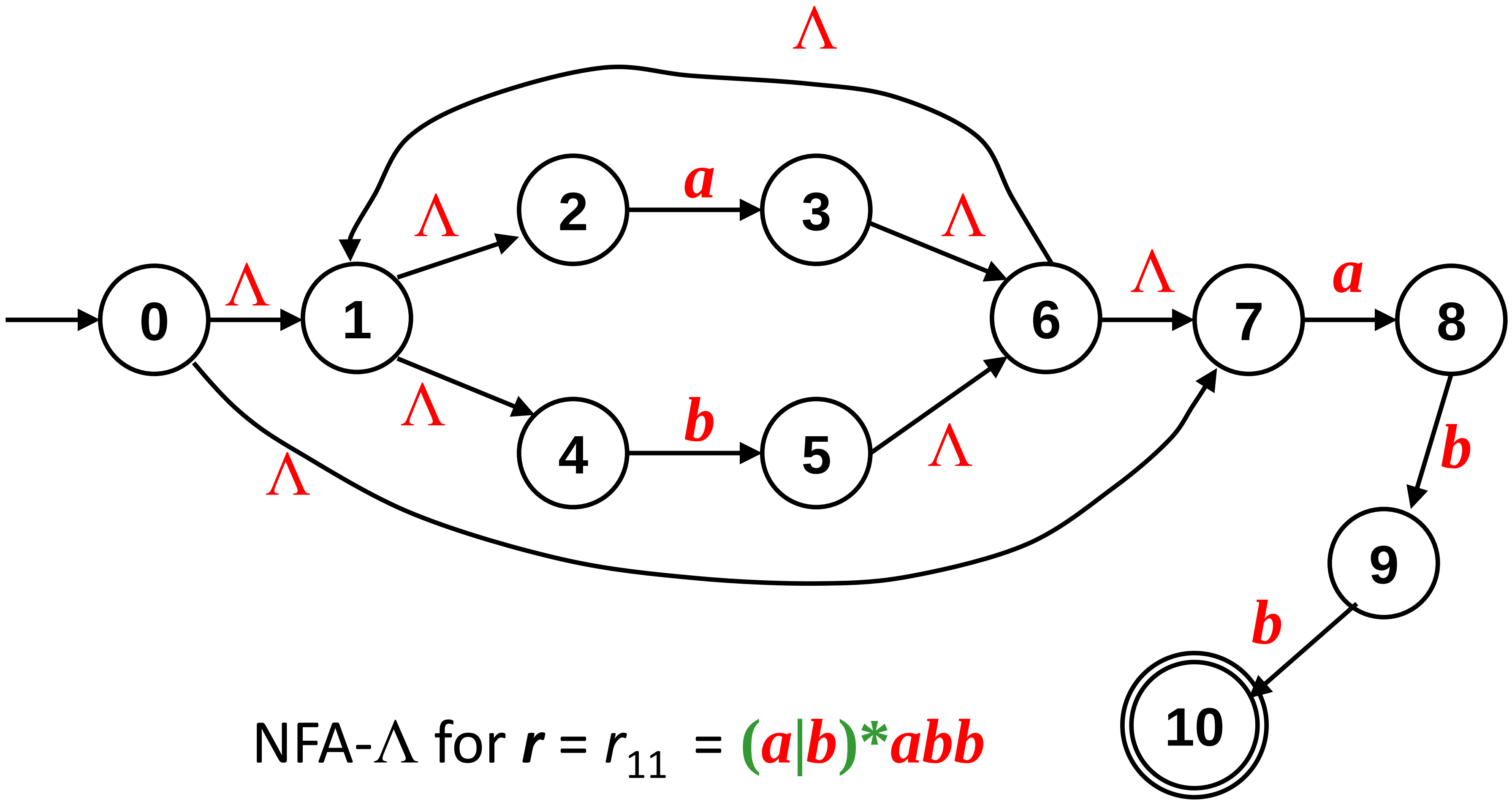




NFA-Λ for r_7 = ($a|b$)*a

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





NFA-Λ for $r = r_{11} = (a|b)^*abb$

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