Unit 4

GRAMMARS

- Grammars express languages

- Example: the English language

$$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \langle predicate \rangle$$

$$\langle noun\_phrase \rangle \rightarrow \langle article \rangle \langle noun \rangle$$

$$\langle predicate \rangle \rightarrow \langle verb \rangle$$

Grammar Notation

•

$$\overbrace{\begin{array}{l}\langle noun \rangle \rightarrow cat \\ \langle noun \rangle \rightarrow dog\end{array}}^{\text{Production Rules}}$$

Variable

Terminal

Some Terminal Rules

$$\langle ARTICLE \rangle \rightarrow A$$

$$\langle ARTICLE \rangle \rightarrow THE$$

$$\langle noun \rangle \rightarrow cat$$

$$\langle noun \rangle \rightarrow dog$$

$$\langle verb \rangle \rightarrow runs$$

$$\langle verb \rangle \rightarrow walks$$

A Resulting Sentence

$$
\begin{aligned}
\langle sentence \rangle &\Rightarrow \langle noun\_phrase \rangle \langle predicate \rangle \\
&\Rightarrow \langle noun\_phrase \rangle \langle verb \rangle \\
&\Rightarrow \langle article \rangle \langle noun \rangle \langle verb \rangle \\
&\Rightarrow the\ \langle noun \rangle \langle verb \rangle \\
&\Rightarrow the\ dog\ \langle verb \rangle \\
&\Rightarrow the\ dog\ walks
\end{aligned}
$$

The Resulting Language

L = { "a cat runs",
"a cat walks",
"the cat runs",
"the cat walks",
"a dog runs",
"a dog walks",
"the dog runs",
"the dog walks" }

Definition of a Grammar

$$G = (V, T, S, P)$$

$V:$ Set of variables

$T:$ Set of terminal symbols

$S:$ Start variable

$P:$ Set of Production rules

A Simple Grammar

- Grammar: $S \rightarrow aSb$

  $S \rightarrow \lambda$

- Derivation of sentence : ab

$$S \Rightarrow aSb \Rightarrow ab$$

$S \rightarrow aSb$ $\quad S \rightarrow \lambda$

Example Grammar Notation

$$S \to aSb$$

$$S \to \lambda$$

$$\begin{tikzcd}
 & G = (V, & T, & S, P) \\
V = \{S\} \arrow[ur] & & T = \{a,b\} \arrow[u] & \\
 & & & P = \{S \to aSb,\ S \to \lambda\} \arrow[uu]
\end{tikzcd}$$

Deriving Strings in the Grammar

- Grammar:

$$S \to aSb$$

$$S \to \lambda$$

- Derivation of sentence $aabb$:

$$\begin{tikzcd}
S & \Rightarrow & aSb & \Rightarrow & aaSbb & \Rightarrow & aabb \\
& & S \to aSb \arrow[ul] \arrow[ur] & & & S \to \lambda \arrow[u] &
\end{tikzcd}$$

Sentential Form

- A sentence that contains variables and terminals

$$S \Rightarrow \underbrace{aSb \Rightarrow aaSbb \Rightarrow aaaSbbb}_{\text{Sentential Forms}} \Rightarrow \underbrace{aaabbb}_{\text{sentence}}$$

General Notation for Derivations

- In general we write: $w_1 \overset{*}{\Rightarrow} w_n$
- If: $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \square \Rightarrow w_n$
- It is always the case that: $w \overset{*}{\Rightarrow} w$

Why Notation Is Useful

- We can now write:

$$S \overset{*}{\Rightarrow} aaabbb$$

- Instead of:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Language of a Grammar

Grammar can produce some set of strings

Set of strings over an alphabet is a language

Language of a grammar is all strings produced by the grammar

$$L(G) = \{w : \quad S \overset{*}{\Rightarrow} w\}$$

$$\uparrow$$

String of terminals

Example Language

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Consider the set of all strings that can derived from this grammar…..

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb$$

$$\Rightarrow aaaaSbbbb \Rightarrow aaaabbbb$$

What language is being described?

The Resulting Language

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

Always add on a and b on each side resulting in:

a's at the left

b's at the right

equal number of a's and b's

Linear Grammars

- Grammars with at most one variable at the right side of a production

- Examples:

$$S \rightarrow aSb$$

$$S \rightarrow \lambda$$

A Non-Linear Grammar

Grammar G :

$$S \rightarrow SS$$
$$S \rightarrow \lambda$$
$$S \rightarrow aSb$$
$$S \rightarrow bSa$$

$L(G) = \{w$: $n_a(w) = n_b(w)\}$

Number of $a's$ string w

Another Linear Grammar

Grammar G :

$$S \to A$$

$$A \to aB \mid \lambda$$

$$B \to Ab$$

$$L(G) = \{a^n b^n : n \geq 0\}$$

Right-Linear Grammars

- All productions have form:

$$A \rightarrow xB$$

or

$$A \rightarrow x$$

string of terminals

- Example: $S \rightarrow abS$

  $S \rightarrow a$

Left-Linear Grammars

- All productions have form:

$$A \to Bx$$

or

$$A \to x$$

string of terminals

- Example:

$$S \to Aab$$
$$A \to Aab \mid B$$
$$B \to a$$

Regular Grammars

Regular Grammars

- A regular grammar is any right-linear or left-linear grammar

- Examples:

$$S \to abS$$
$$S \to a$$

$$S \to Aab$$
$$A \to Aab \mid B$$
$$B \to a$$

What languages are generated by these grammars?

Languages and Grammars

$$S \rightarrow abS$$

$$S \rightarrow a$$

$$S \rightarrow Aab$$

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

$$L(G_1) = (ab)^* a$$

$$L(G_2) = aab(ab)^*$$

Note both these languages are regular

- we have regular expressions for these languages (above)
- we can convert a regular expression into an NFA (how?)
- we can convert an NFA into a DFA (how?)
- we can convert a DFA into a regular expression (how?)

Do regular grammars also describe regular languages??

Example

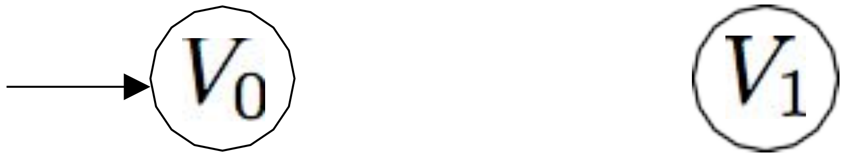
Given right linear grammar:

$$V_0 \rightarrow aV_1$$

$$V_1 \rightarrow abV_0|b$$

Step 1: Create States for Each Variable

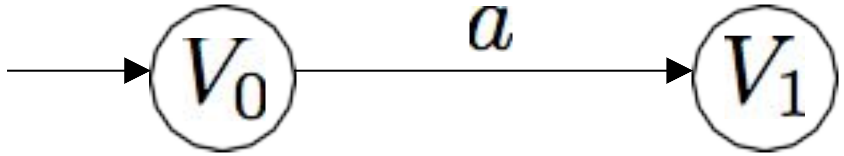
- Construct NFA M such that every state is a grammar variable:



$$V_0 \rightarrow aV_1$$
$$V_1 \rightarrow abV_0 | b$$

Step 2.1: Edges for Productions

- Productions of the form $V_i \rightarrow aV_j$ result in $\delta(V_i, a) = V_j$



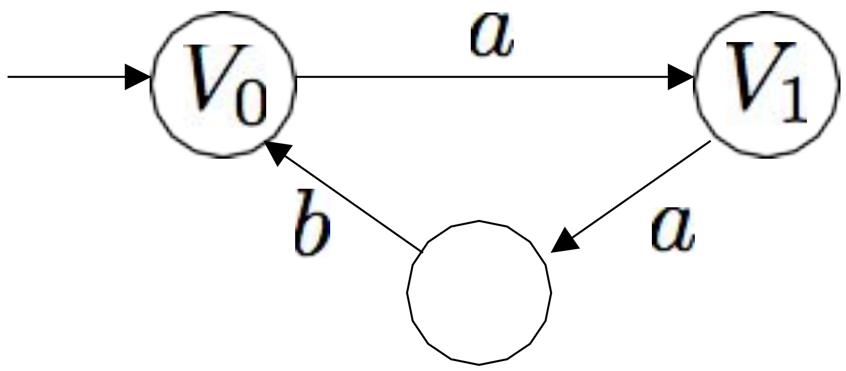
$$\begin{tikzcd}
{} \arrow[r] & V_0 \arrow[r, "a"] & V_1
\end{tikzcd}$$

$V_0 \rightarrow aV_1$

$V_1 \rightarrow abV_0|b$

Step 2.2: Edges for Productions

- Productions of the form $V_i \rightarrow wV_j$ are only slightly harder…. Create row of states that derive w and end in V_j

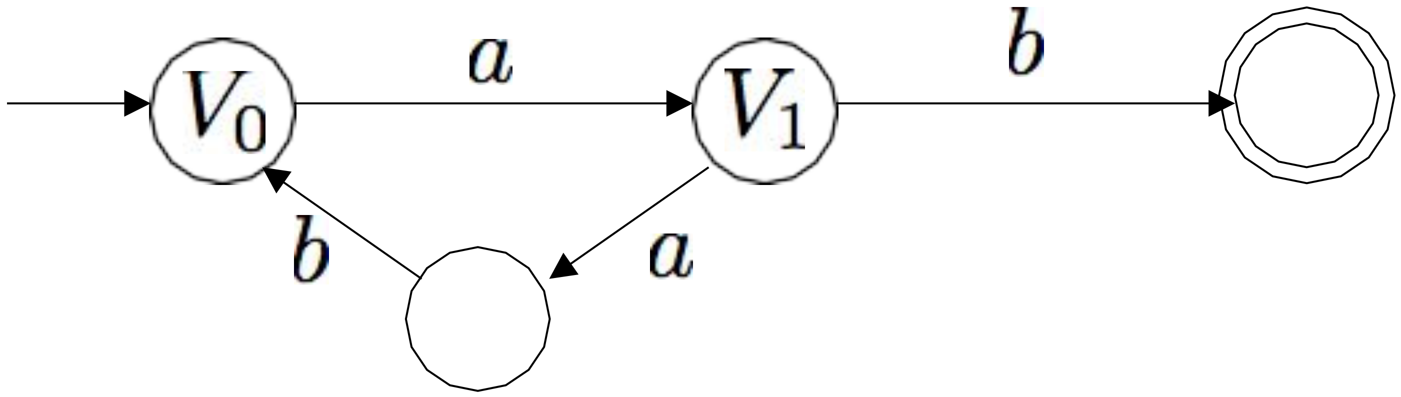


$$V_0 \rightarrow aV_1$$

$$V_1 \rightarrow abV_0 | b$$

Step 2.3: Edges for Productions

- Productions of the form $V_i \rightarrow w$ Create row of states that derive w and end in a final state



$V_0 \rightarrow aV_1$

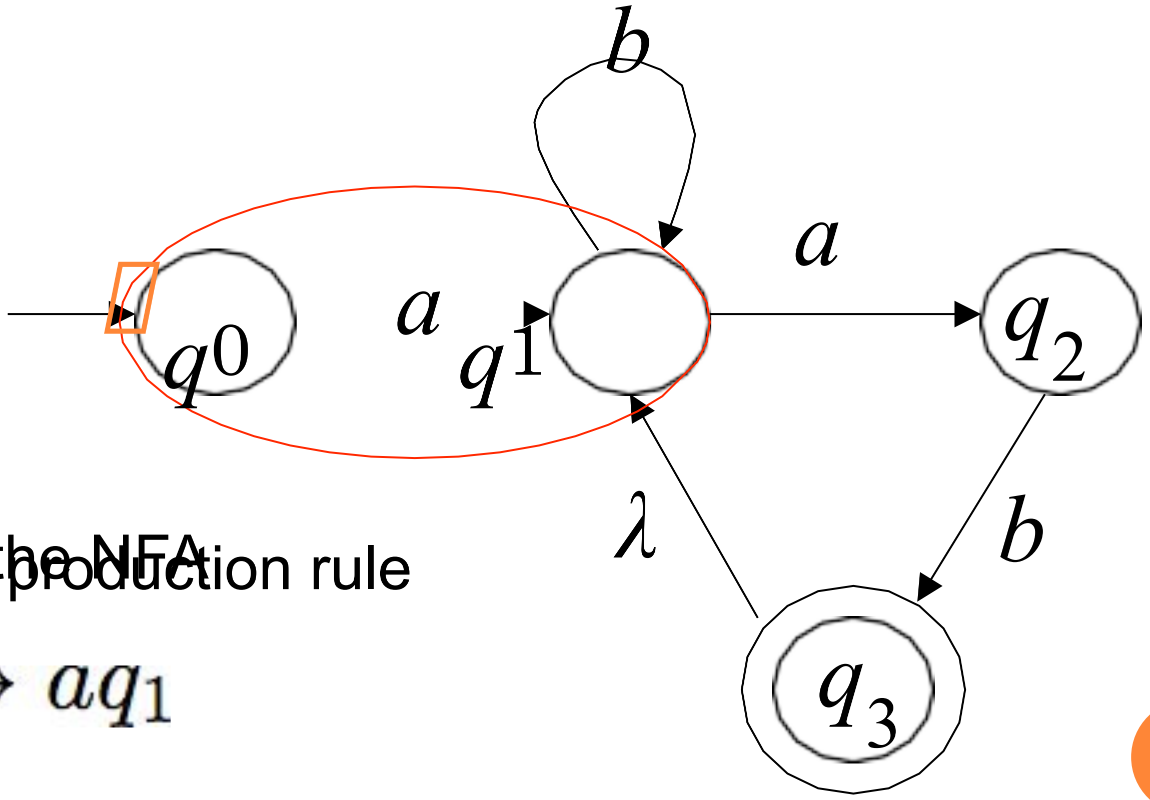
$V_1 \rightarrow abV_0|b$

In General

- Given any right-linear grammar, the previous procedure produces an NFA
  - We sketched a proof by construction
  - Result is both a proof and an algorithm
  - *Why doesn't this work for a non linear grammar?*
- Since we have an NFA for the language, the right-linear grammar produces a regular language

NFA to Grammar Example

- Since L is regular there is an NFA



This transition in the NFA

Looks a lot like a production rule

$$q_0 \rightarrow aq_1$$

Step 1:
Convert Edges to Productions

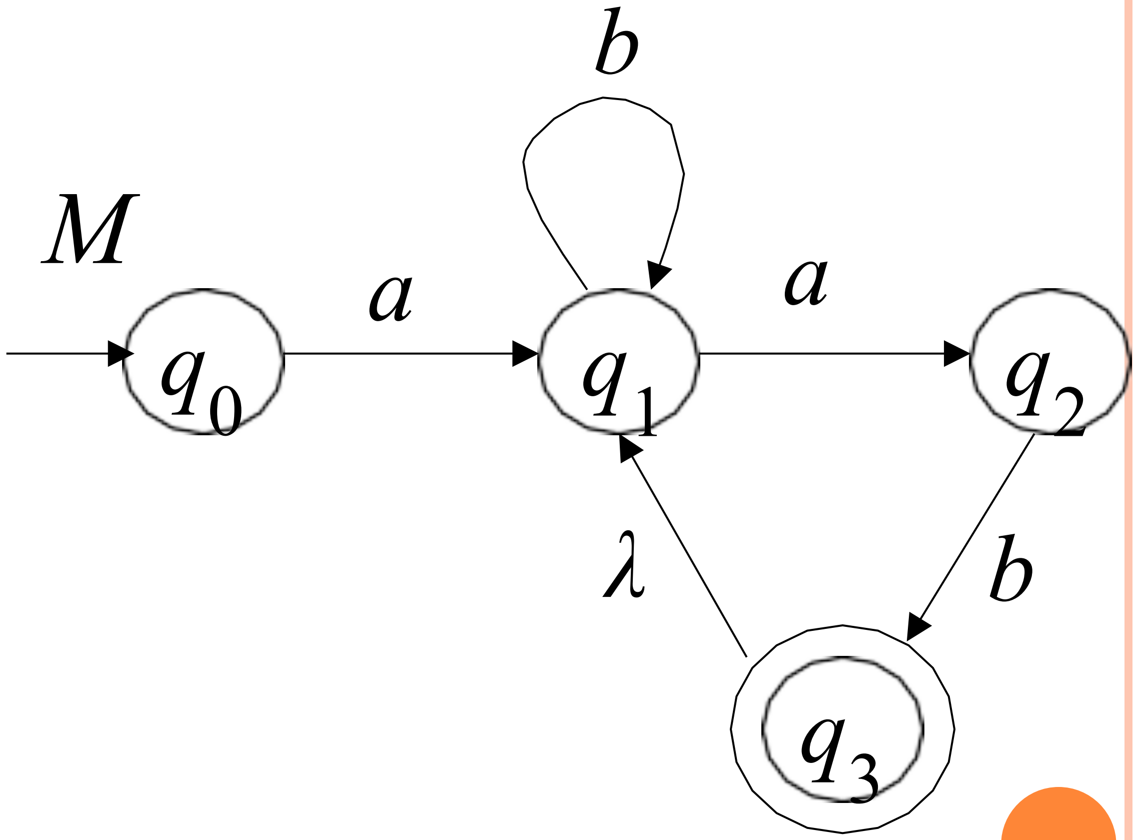
$q_0 \rightarrow aq_1$

$q_1 \rightarrow bq_1$

$q_1 \rightarrow aq_2$

$q_2 \rightarrow bq_3$

M



Step 2:

λ Edges and Final States

$q_0 \rightarrow aq_1$

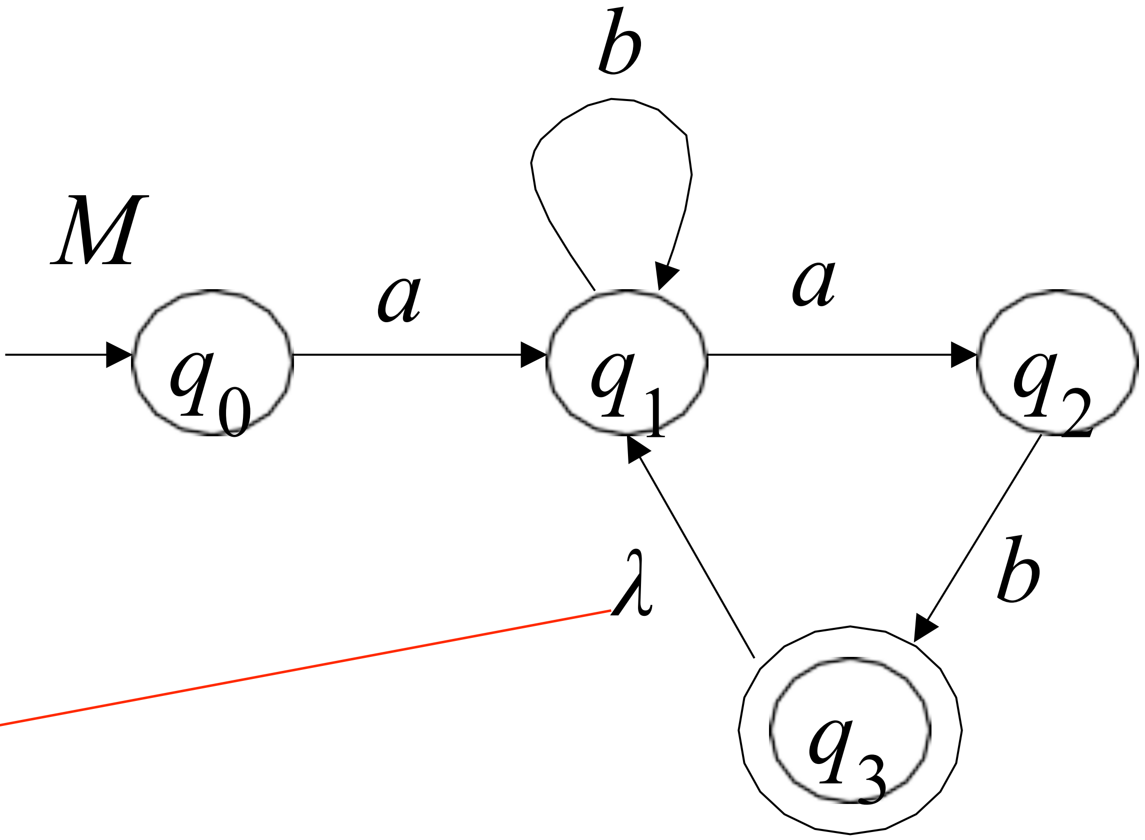
$q_1 \rightarrow bq_1$

$q_1 \rightarrow_2 aq$

$q_2 \rightarrow bq_3$

$q_3 \rightarrow q_1$

$q_3 \rightarrow \lambda$



Step 2:

λ Edges and Final States

$q_0 \rightarrow aq_1$

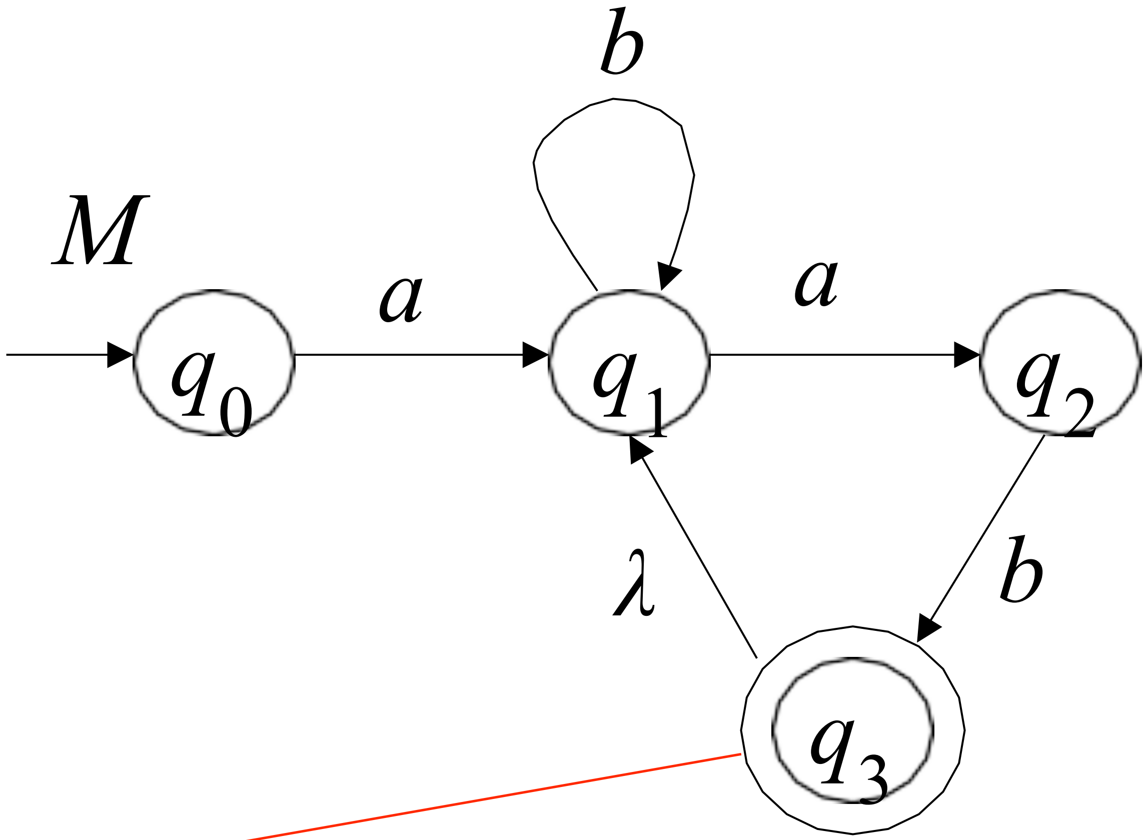
$q_1 \rightarrow bq_1$

$q_1 \rightarrow aq_2$

$q_2 \rightarrow bq_3$

$q_3 \rightarrow q_1$

$q_3 \rightarrow \lambda$



If q_i is a final state, add $q_i \rightarrow \lambda$

In General

- Given any NFA, the previous procedure produces a right linear grammar
  - We sketched a proof by construction
  - Result is both a proof and an algorithm
- Every regular language has an NFA
  - Can convert that NFA into a right linear grammar
  - Thus every regular language has a right linear grammar
- Combined with Part 1, we have shown right linear grammars are yet another way to describe regular languages

But What About Left-Linear Grammars

- What happens if we reverse a left linear grammar as follows:

$V_i \to V_j w$ Reverses to $V_i \to w^R V_j$

$V_i \to w$ Reverses to $V_i \to w^R$

- The result is a right linear grammar.
  - If the left linear grammar produced L, then what does the resulting right linear grammar produce?

But What About Left-Linear Grammars

- The previous slide reversed the language!

Reverses to

$V_i \rightarrow V_j w$ $\quad$ $V_i \rightarrow w^R V_j$

Reverses to

$V_i \rightarrow w$ $\quad$ $V_i \rightarrow w^R$

- If the left linear grammar produced language , L
  then the resulting right linear grammar produces L^R

Claim we just proved left linear grammars produce regular languages? Why?

Left-Linear Grammars Produce Regular Languages

- Start with a Left Linear grammar that produces L want to show L is regular
- Can produce a right linear grammar that produces L^R
- All right linear grammars produce regular languages so L^R is a regular language
- The reverse of a regular language is regular so $(L^R)^R = L$ is a regular language!

For regular languages L_1 and L_2

we will prove that:

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1{}^*$

Reversal: L_1^R

Complement: $\overline{L_1}$

Intersection: $L1 \cap L2$

Are regular Languages

We say: Regular languages are **closed under**

Union: $L_1 \cup L_2$

Concatenation: $L_1 L_2$

Star: $L_1 *$

Reversal: L_1^R

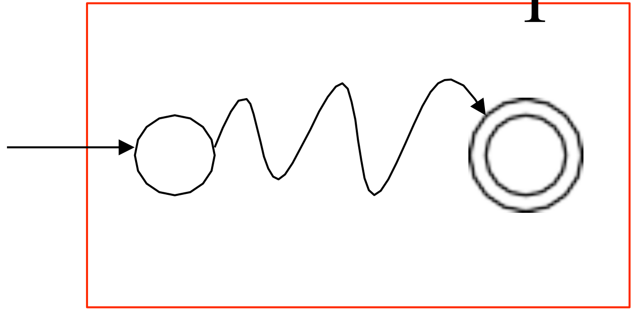
Complement: $\overline{L_1}$

Intersection: $L1 \cap L2$

Regular language L_1

$L(M_1) = L_1$

NFA M_1

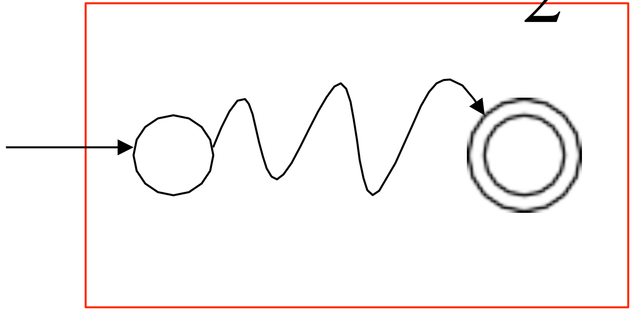


Single final state

Regular language L_2

$L(M_2) = L_2$

NFA M_2

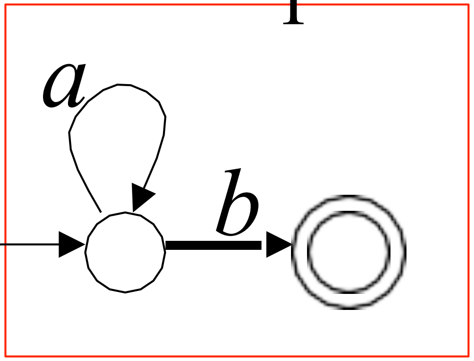


Single final state

Example

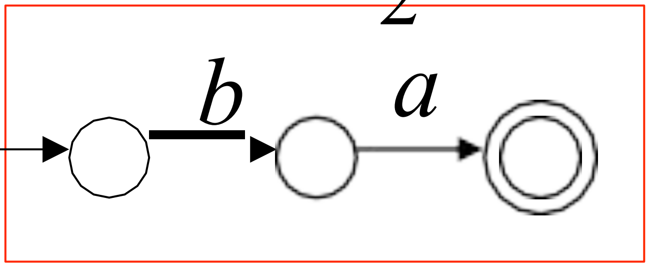
$$L_1 = \{a^n b\}$$
$$n \geq 0$$

M_1



$$L_2 = \{ba\}$$

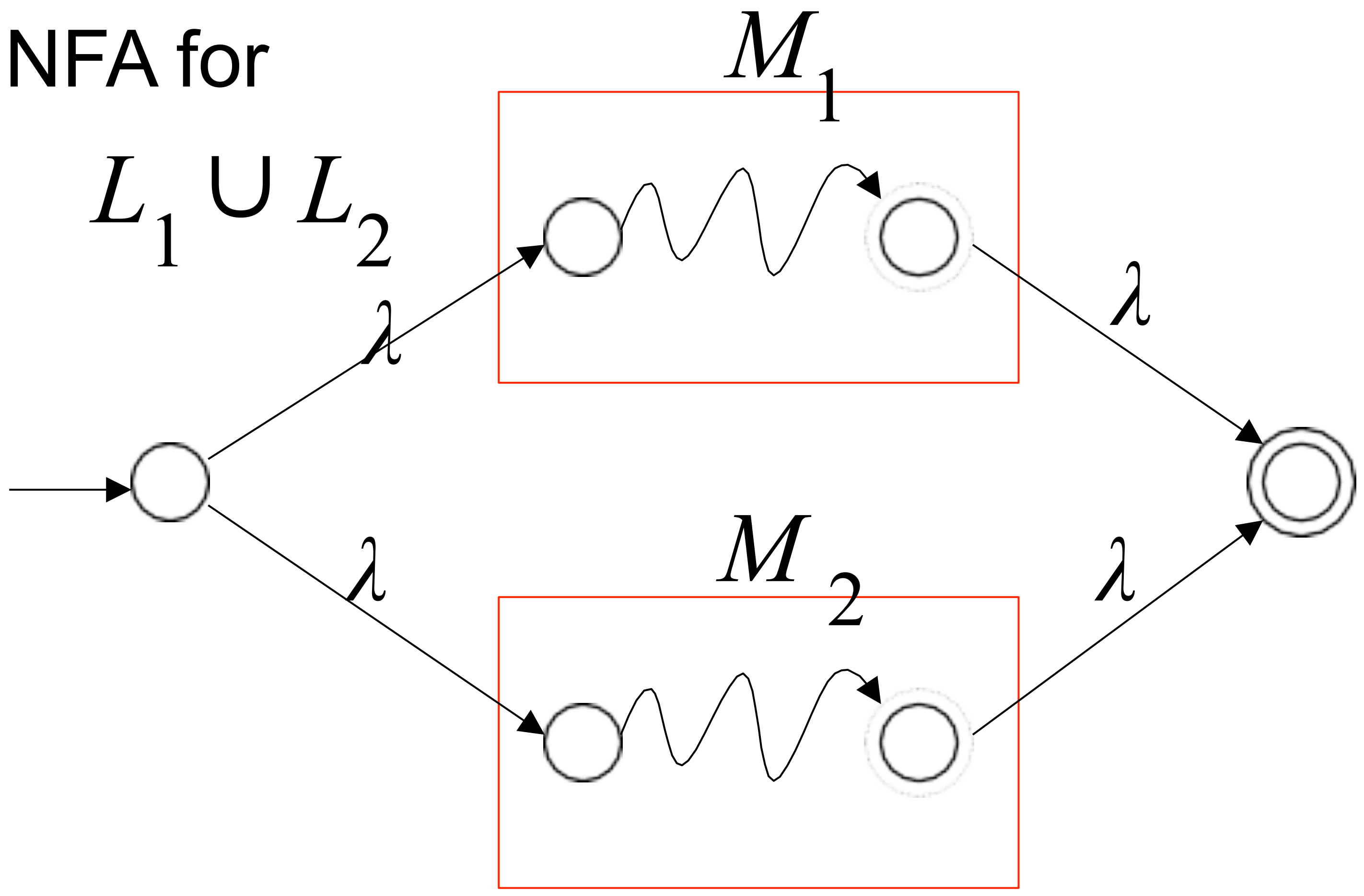
M_2



Union

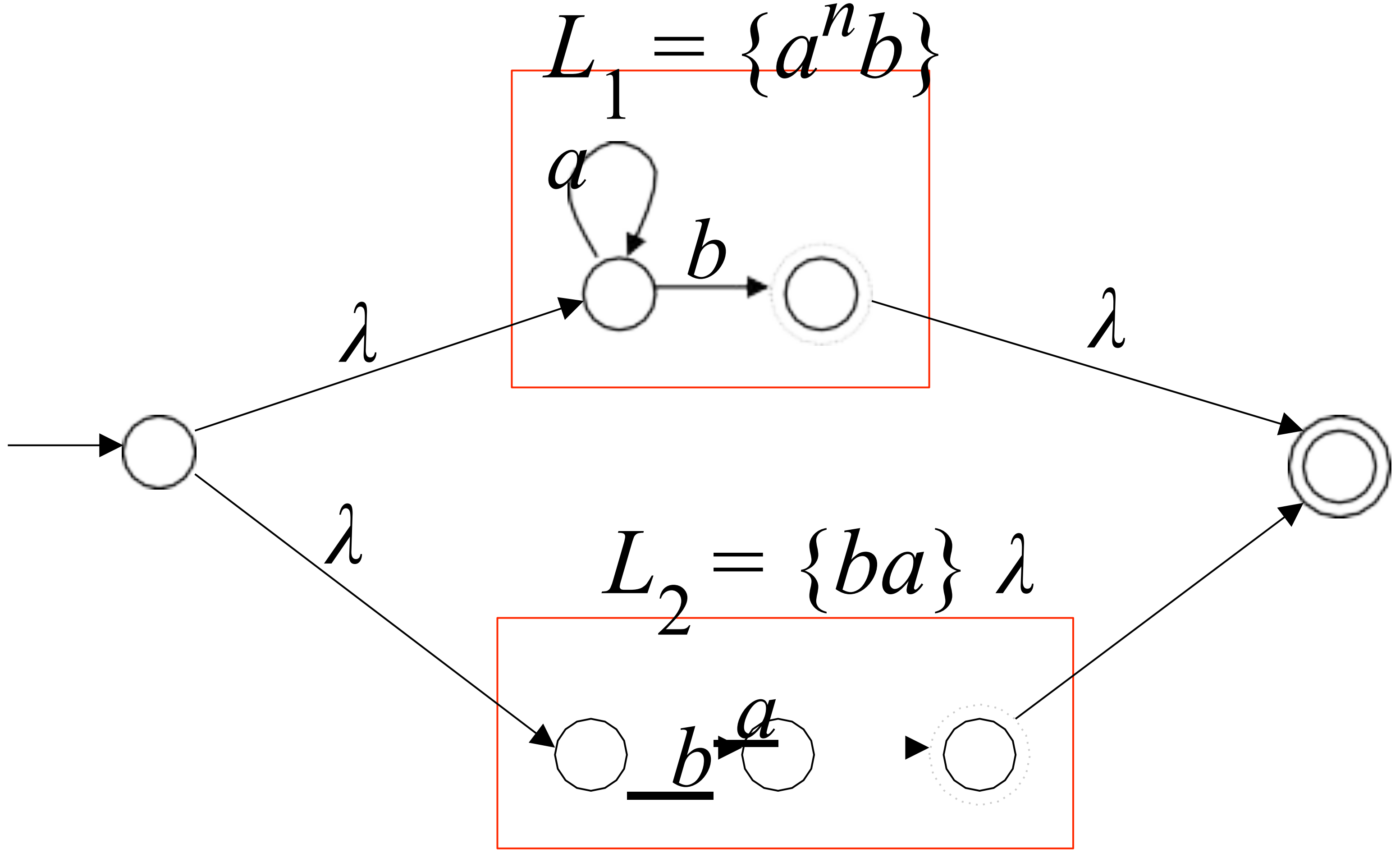
- NFA for

$L_1 \cup L_2$



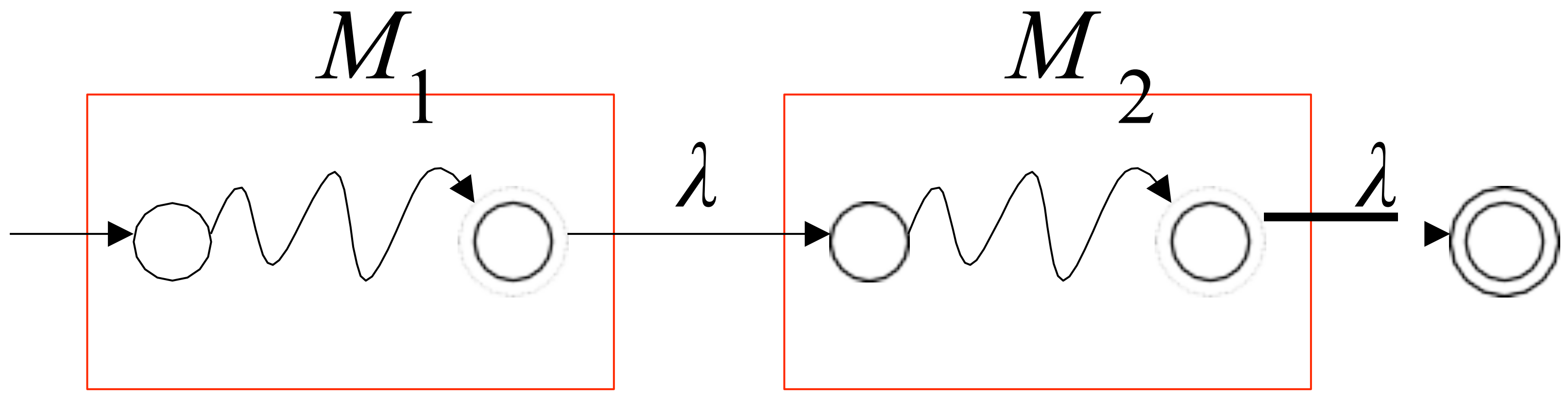
Example

NFA for $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$



Concatenation

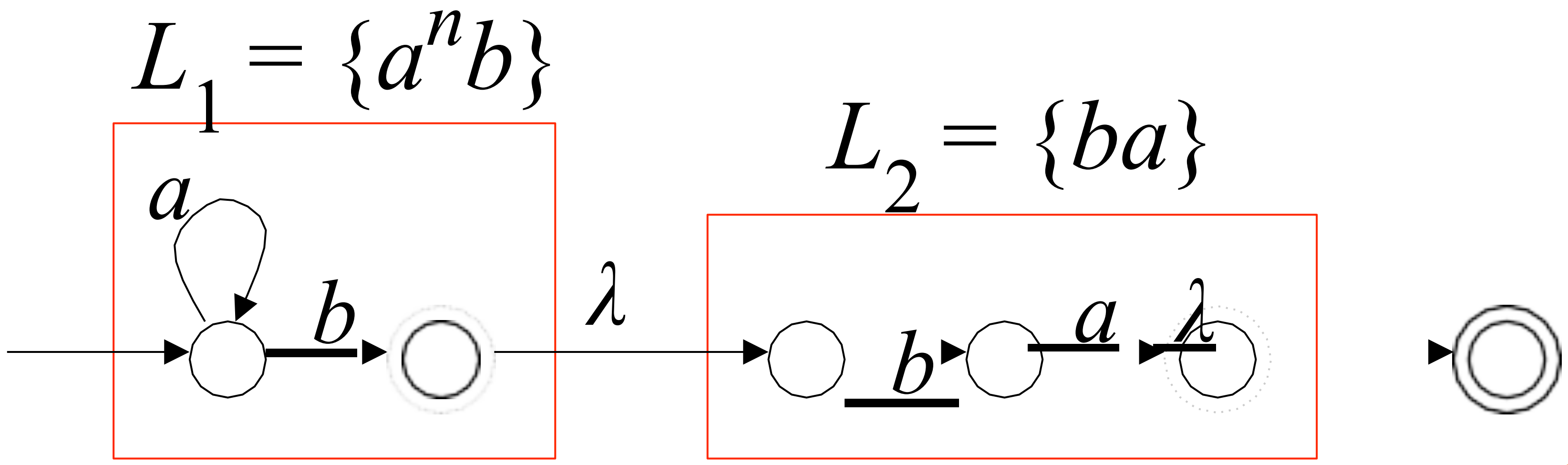
NFA for $L_1 L_2$



Example

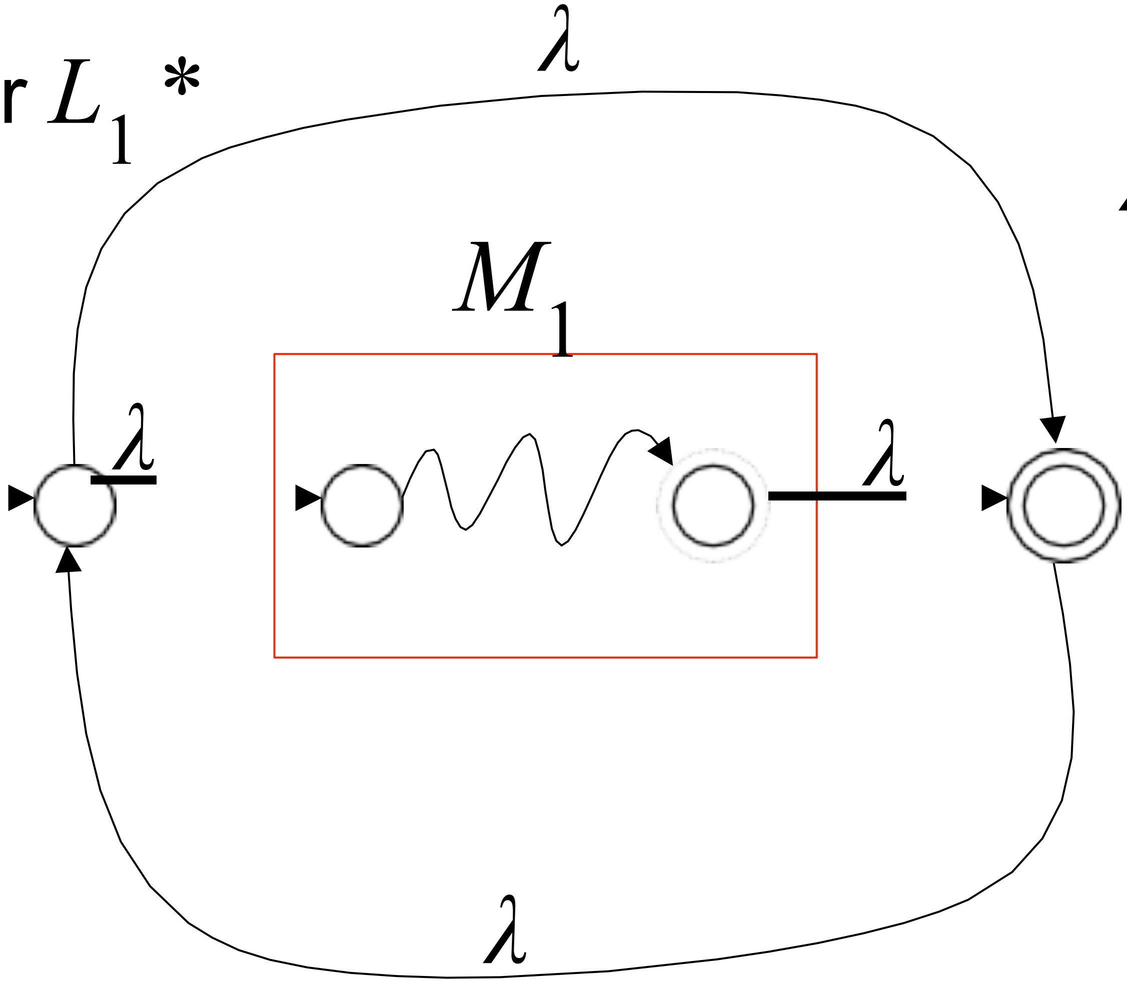
$$L_1 L_2 = \{a^n b\}\{ba\} = \{a^n bba\}$$

NFA for



Star Operation

NFA for $L_1{}^*$



$\lambda \in L_1{}^*$

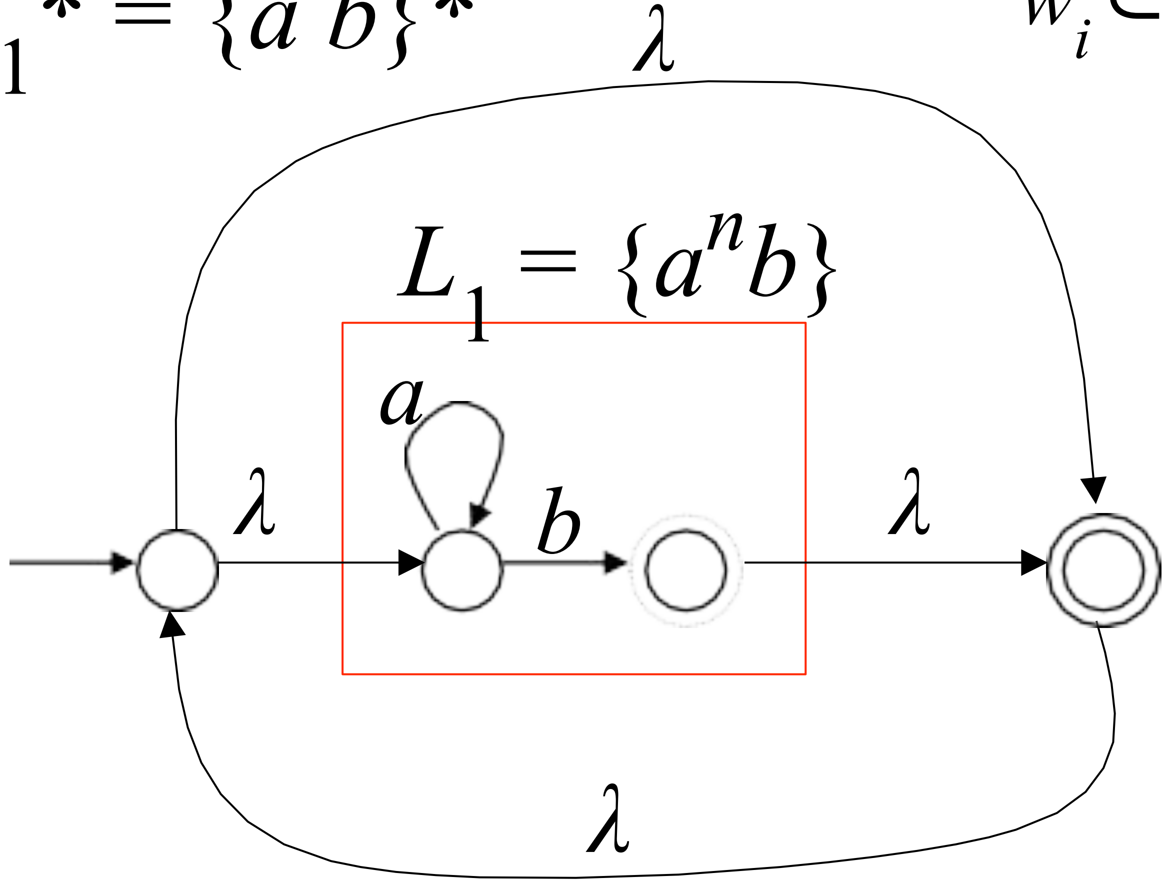
Example

NFA for

$$L_1{}^* = \{a^n b\}^*$$

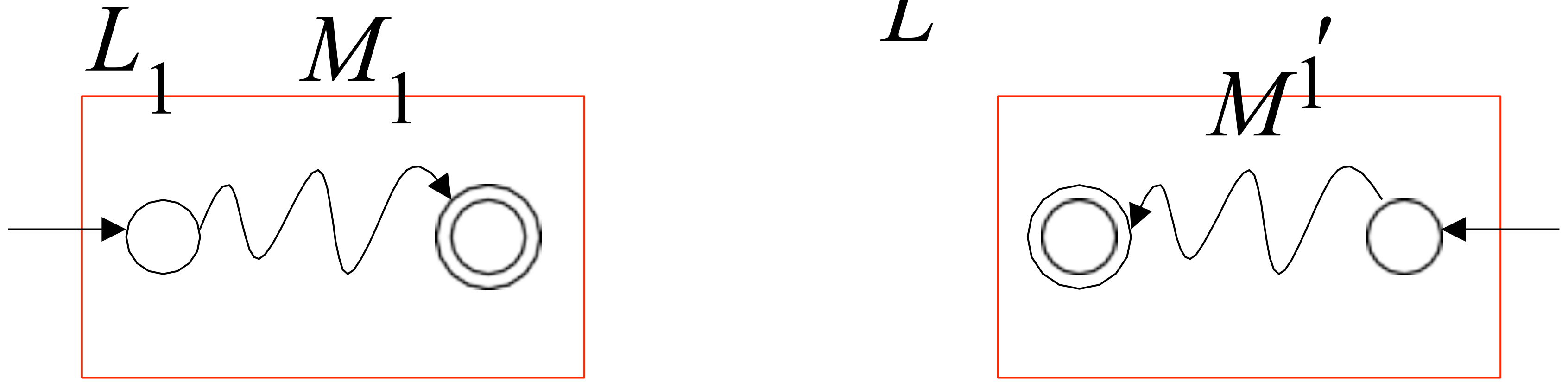
$$w = w_1 w_2 \cdots w_k$$

$$w_i \in L_1$$



REVERSE

NFA for L_1^R

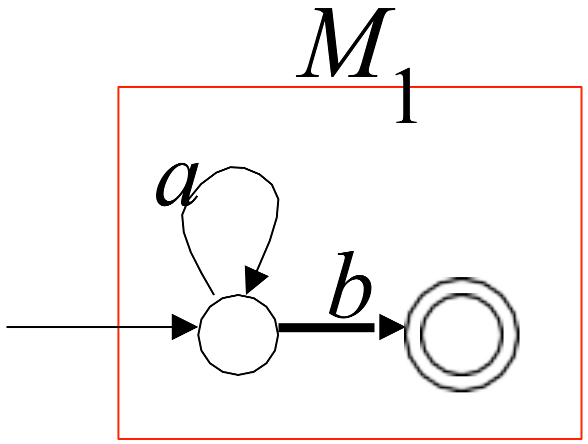


1. Reverse all transitions

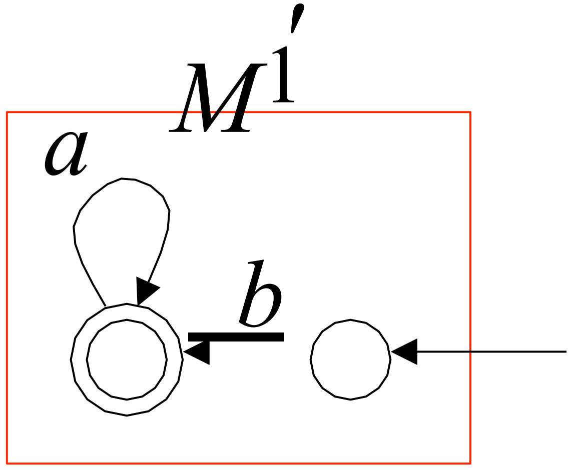
2. Make initial state final state and vice versa

Example

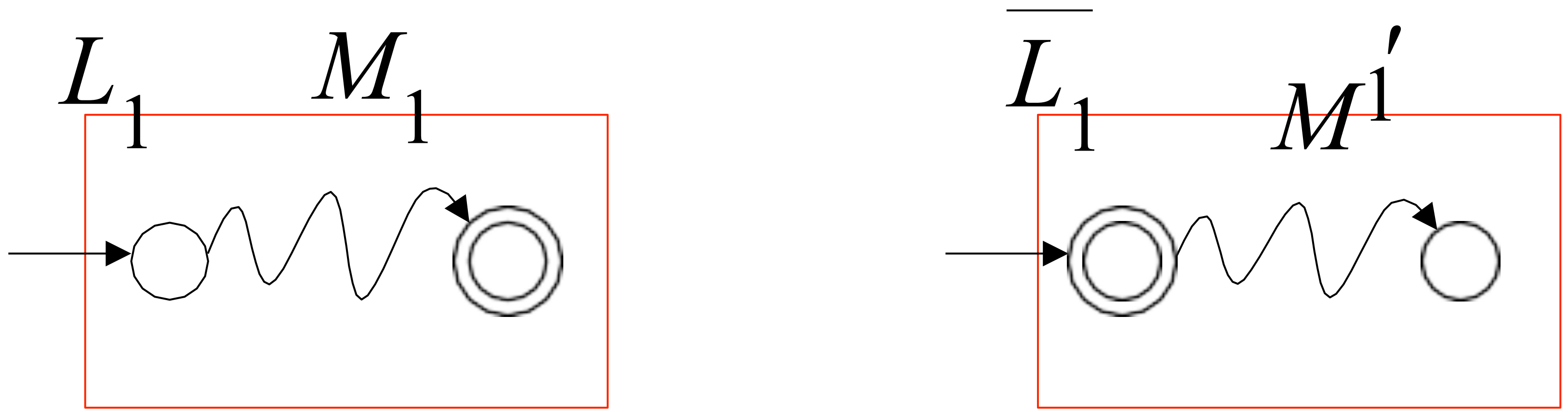
$$L_1 = \{a^n b\}$$



$$L_1^R = \{ba^n\}$$



Complement



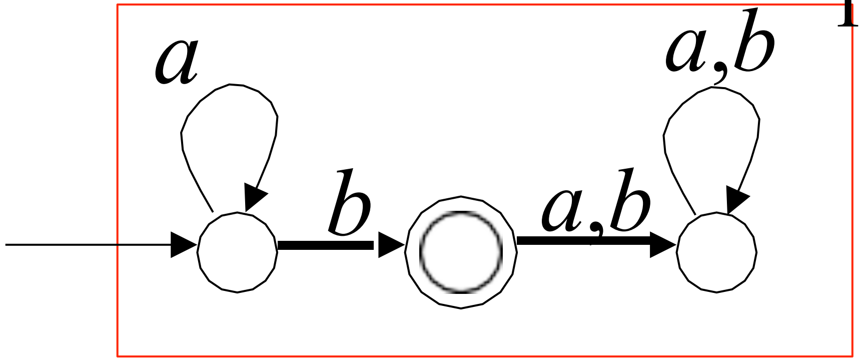
1. Take the **DFA** that accepts L_1

2. Make final states non-final, and vice-versa

Example

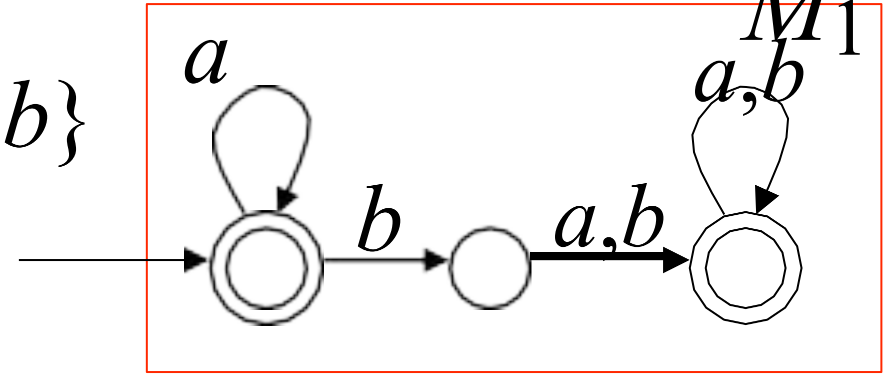
$$L_1 = \{a^n b\}$$

M_1



$$\overline{L_1} = \{a,b\}^* - \{a^n b\}$$

M_1'



Intersection

DeMorgan's Law: $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

L_1, L_2 regular

$\Rightarrow \overline{L_1}, \overline{L_2}$ regular

$\Rightarrow \overline{L_1} \cup \overline{L_2}$ regular

$\Rightarrow \overline{\overline{L_1} \cup \overline{L_2}}$ regular

$\Rightarrow L_1 \cap L_2$ regular

Context Sensitive Grammar(CSG)

- Context Sensitive Grammar(CSG) is a quadruple G=(N,Σ,P,S), where
  - N is set of non-terminal symbols
    Σ is set of terminal symbols
  - S is set of start symbol
  - P's are of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ where $\gamma \neq s$
    where $(\alpha, \beta, \gamma) \in (N \cup \Sigma)^*$ and $(A \in N)$
- Why Context Sensitive??
  - Given a production : $\alpha A\beta \rightarrow \alpha\gamma\beta$ where $\gamma \neq s$. During derivation non-terminal A will be changed to γ only when it is present in context of α and β
- As a consequence of $\gamma \neq s$ we have $\alpha \rightarrow \beta \Rightarrow |\alpha| \leq |\beta|$
  (Noncontracting grammar)

Context Sensitive Languages

- The language generated by the Context Sensitive Grammar is called context sensitive language.
- If G is a Context Sensitive Grammar then
  - $L(G) = \{w \mid w \in \Sigma^* \text{ and } S \overset{+}{\Rightarrow} w\}$
- CSG for L = $\{ a^n b^n c^n \mid n \geq 1 \}$
  - $N : \{S, B\}$ *and* $\Sigma = \{a, b, c\}$
  - P : S → aSBc | abc   cB → Bc   bB → bb
- Derivation of *aabbcc* :
  - $S \Rightarrow aSBc \Rightarrow aabcBc \Rightarrow aabBcc \Rightarrow aabbcc$

Linear Bounded Automata - Definition

Linear Bounded Automata is a single tape Turing Machine with two special tape symbols call them left marker < and right marker >The transitions should satisfy these conditions:

- It should not replace the marker symbols by any other symbol.
- It should not write on cells beyond the marker symbols.

Thus the initial configuration will be:

< q0a1a2a3a4a5.......an >

- A linear bounded automaton can be defined as an 8-tuple (Q, X, ∑, q_0, ML, MR, δ, F) where −
- **Q** is a finite set of states
- **X** is the tape alphabet
- ∑ is the input alphabet
- $\mathbf{q_0}$ is the initial state
- $\mathbf{M_L}$ is the left end marker
- $\mathbf{M_R}$ is the right end marker where $M_R \neq M_L$
- **δ** is a transition function which maps each pair (state, tape symbol) to (state, tape symbol, Constant 'c') where c can be 0 or +1 or -1
- **F** is the set of final states

END