# Regular Grammars and Parsing

# Set of all binary numbers divisible by 5

(101+11(01)*(1+00)1+(100 +11(01)*(1+00)0)(1+0(01)*(1+00)0)*0(01)*
(1+00)1)(0 +101+11(01)*(1+00)1+(100 +11(01)*(1+00)0)(1+0(01)*
(1+00)0)*0(01)*(1+ 00)1)* +0

$$(101 + 11U1 + (100 + 11U0)(1 + 0U0)^* 0U1)$$
$$(0 + 101 + 11U1 + (100 + 11U0)(1 + 0U0)^* 0U1)^* + 0$$

where

$$U = (01)^*(1 + 00)$$

Continuing with this process, we can further reduce the RegEx to

$$(W + XY^* Z)(0 + W + XY^* Z)^* + 0$$

where

$$\begin{aligned} W &= 101 + 11U1 \\ X &= 100 + 11U0 \\ Y &= 1 + 0U0 \\ Z &= 0U1 \end{aligned}$$

# Grammars

- Grammars express languages

- Example: the English language

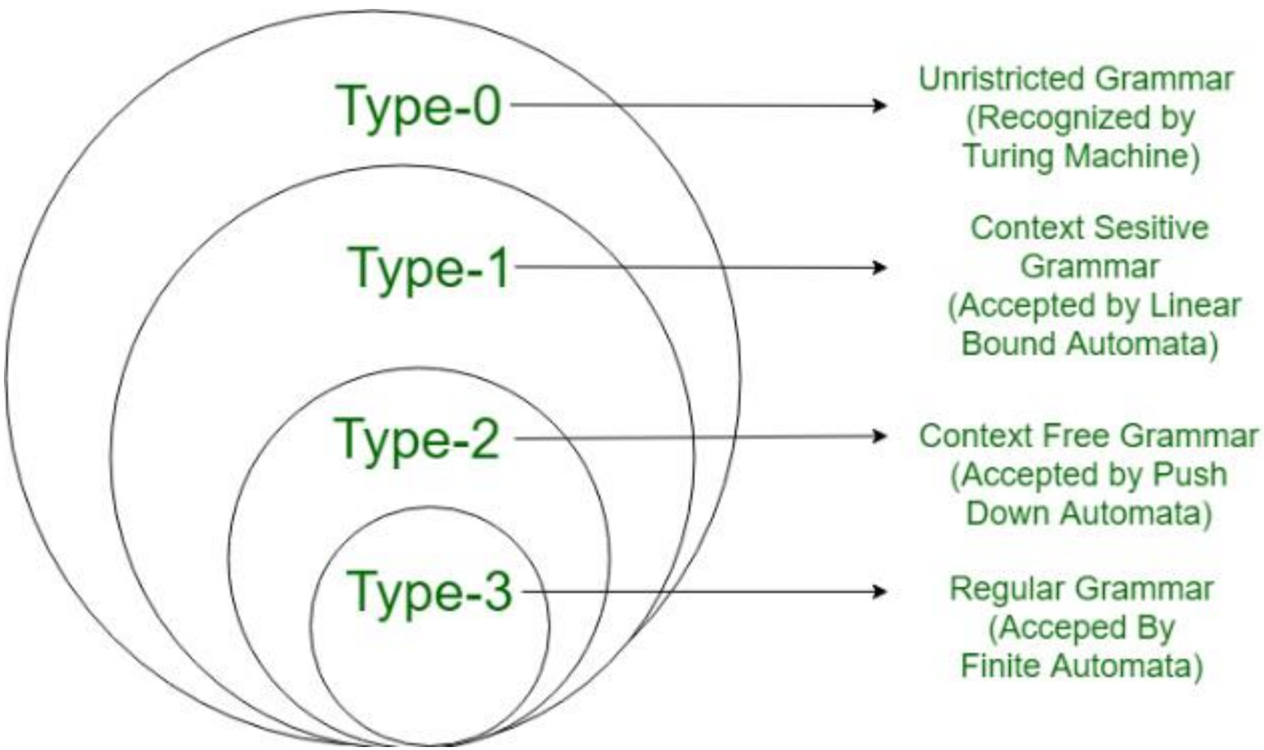
$$\langle sentence \rangle \rightarrow \langle noun\_phrase \rangle \ \langle predicate \rangle$$

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

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

# Chomsky Hierarchy in Theory of Computation

```
Type 0 known as unrestricted grammar.
Type 1 known as context sensitive grammar.
Type 2 known as context free grammar.
Type 3 Regular Grammar.
```



# Type3 - Regular Grammar

Type-3 grammars generate regular languages. These languages are exactly all languages that can be accepted by a finite state automaton.

Type 3 is most restricted form of grammar. Type 3 should be in the given form only :

**V -> VT* / T***

**(or)**

**V -> T*V /T***

Example :

S –> ab

# 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

- S is also called as sentence variable
- Variables are also called as non-terminals or phrases

# Linear Grammars

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

Examples:

$$S \to aSb \qquad\qquad S \to Ab$$

$$S \to \lambda \qquad\qquad A \to aAb$$

$$A \to \lambda$$

# A Non-Linear Grammar

Grammar $G$ :

$$S \to SS$$

$$S \to \lambda$$

$$S \to aSb$$

$$S \to bSa$$

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

Number of $a's$ in string $w$

# IS this a 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:

$$A \to xB$$

or

$$A \to x$$

string of terminals

- Example: $S \to abS$

$$S \to a$$

# Left-Linear Grammars

- All productions have form:

$$A \to Bx$$

or

$$A \to x$$

string of terminals

- Example: $S \to Aab$

$$A \to Aab \mid B$$

$$B \to a$$

V -> VT* / T*
(or)
V -> T*V /T*

# Regular Grammars

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

- Examples:

$$S \rightarrow abS$$

$$S \rightarrow a$$

$$S \rightarrow Aab$$

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

# Parsing AND Derivation

**Parsing** is used to derive a string using the production rules of a grammar.

It is used to check the acceptability of a string.

Compiler is used to check whether or not a string is syntactically correct.

A **parser** takes the inputs and builds a **parse** tree.

# Derivation

Strings $\alpha$ yields string $\beta$, written $\alpha \Rightarrow \beta$ (ON * CLOSURE), if it is possible to get from $\alpha$ to $\beta$ using the productions.

A derivation of $\beta$ is the sequence of steps that gets to $\beta$.

A leftmost derivation is where at each stage one replaces the leftmost variable.

A rightmost derivation is defined similarly.

# *Derivation Trees*

In a ***derivation tree***, the root is the start variable, all internal nodes are labeled with variables, while all leaves are labeled with terminals. The children of an internal node are labeled from left to right with the right-hand side of the production used.

$G = \{V, T, S, P\}$

$V = \{S\}$

$T = \{0, 1\}$

$S = S$

$P =$

{

S-> 0S

S->1S

S->0

}

# PARSING 1010

1010->101S->10S->1S->S

# DERIVATION OF 1010

S->1S->10S->101S->1010

PARSE TREE???

INTERMEDIATE STRINGS ARE CALLED AS ***SENTENTIAL FORMS***

***Eg.*** 101S

FINAL SENTENTIAL FORM WHICH CONTAINS NO VARIABLES IS ALSO CALLED AS A ***SENTENCE***

# PROPERTIES OF PARSE TREE

- IT IS AN ORDERED TREE
- THE ROOT IS S
- LEAVES ARE TERMINAL SYMBOLS (OR THE EMPTY SYMBOL)
- INTERIOR NODES ARE VARIABLE
- LAMDA, WHEN IT OCCURS AS A LEAF, CANNOT HAVE A SIBLING NODE

$$G = \{ V, T, S, P \}$$

$V = \{S, T\}$

$T = \{0,1\}$

$S = S$

$P =$

$\{$

$S \to T0$

$T \to T0$

$T \to T1$

$T \to \varepsilon$

$\}$

# 1010

PARSE TREE??

***Note:***

Parse tree for a right linear grammar is skewed to the right

Parse tree for a left linear grammar is skewed to the left

A regular grammar, $G_{Reg}$, is a grammar in which the production rules $x \to y$ are constrained in four ways:

1. $y$ should be at least as long as $x$, that is, the right-hand side cannot be shorter than the left-hand side; sentential forms in a derivation can only expand, they cannot shrink at any point.
2. There can be no terminal symbols on the left-hand side; there can be only one non-terminal symbol on the left-hand side (which makes the grammar *context free* as we will see in Chapter 7).
3. There can be only one non-terminal (i.e., variable) on the right-hand side (which makes the grammar *linear* as we will see in Chapter 7).
4. The single variable on the right-hand side must appear either as the rightmost symbol in every rule, giving us a *right-linear grammar*, or as the leftmost symbol in every production rule, giving us a *left-linear grammar*; the grammar cannot have some rules where the variable is the leftmost symbol on the right-hand side and some others in which the variable is the rightmost symbol on the right-hand side; no rule can have a variable in the middle of the right-hand side, that is, with terminal symbols on either side of a variable.

# Constructing Regular Grammar

Grammar for all strings over {a, b} that start with 2 or more a's followed by 3 or more b's.

$$S \to aA$$
$$A \to aB$$
$$B \to aB$$
$$B \to bC$$
$$C \to bD$$
$$D \to bE$$
$$E \to bE$$
$$E \to \lambda$$

The revised grammar is

$S \rightarrow aaB$ Two mandatory $a$s

$B \rightarrow aB$ Any number of additional $a$s

$B \rightarrow bbbE$ Three mandatory $b$s

$E \rightarrow bE$ Any number of additional $b$s

$E \rightarrow \lambda$ End generation (similar to reaching a final state)

$$
\begin{aligned}
S &\rightarrow aaB \\
B &\rightarrow aB \mid bbbE \\
E &\rightarrow bE \mid \lambda
\end{aligned}
$$

# Observation

Regular grammars generate regular languages

Examples:

$$S \to abS$$

$$S \to a$$

$$L(G_1) = ?$$

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

# Regular Grammars
# Generate
# Regular Languages

# Theorem

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} = \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

# Theorem - Part 1

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Any regular grammar generates
a regular language

# Theorem - Part 2

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Any regular language is generated by a regular grammar

# Proof - Part 1

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

The language $L(G)$ generated by any regular grammar $G$ is regular

# The case of Right-Linear Grammars

Let $G$ be a right-linear grammar

We will prove: $L(G)$ is regular

**Proof idea:** We will construct NFA $M$ with $L(M) = L(G)$

Grammar $G$ is right-linear

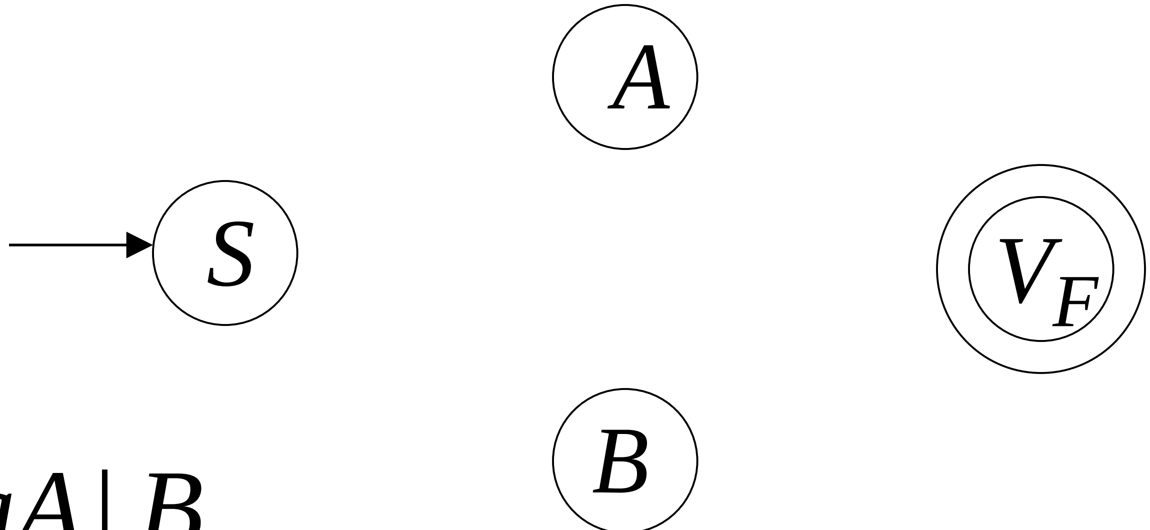
Example:

$$S \to aA \mid B$$

$$A \to aa\,B$$

$$B \to b\,B \mid a$$

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



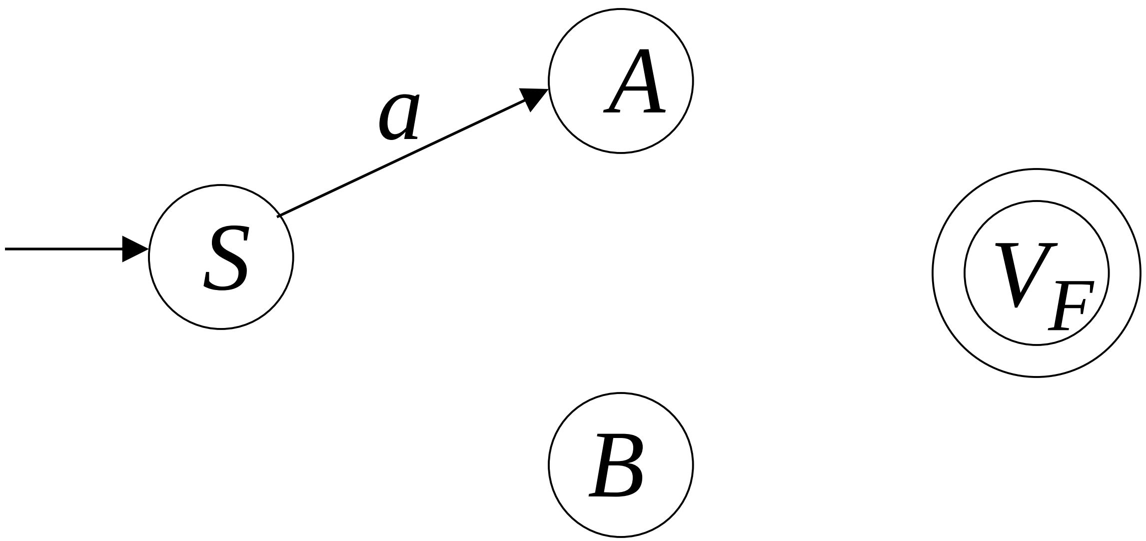

special final state

$$S \to aA \mid B$$

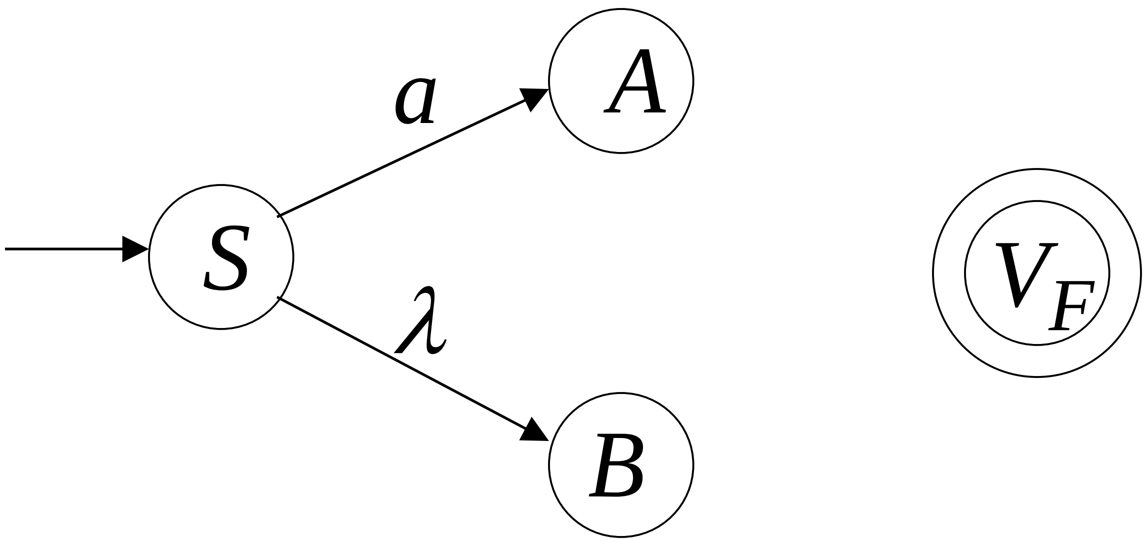
$$A \to aa\,B$$

$$B \to b\,B \mid a$$

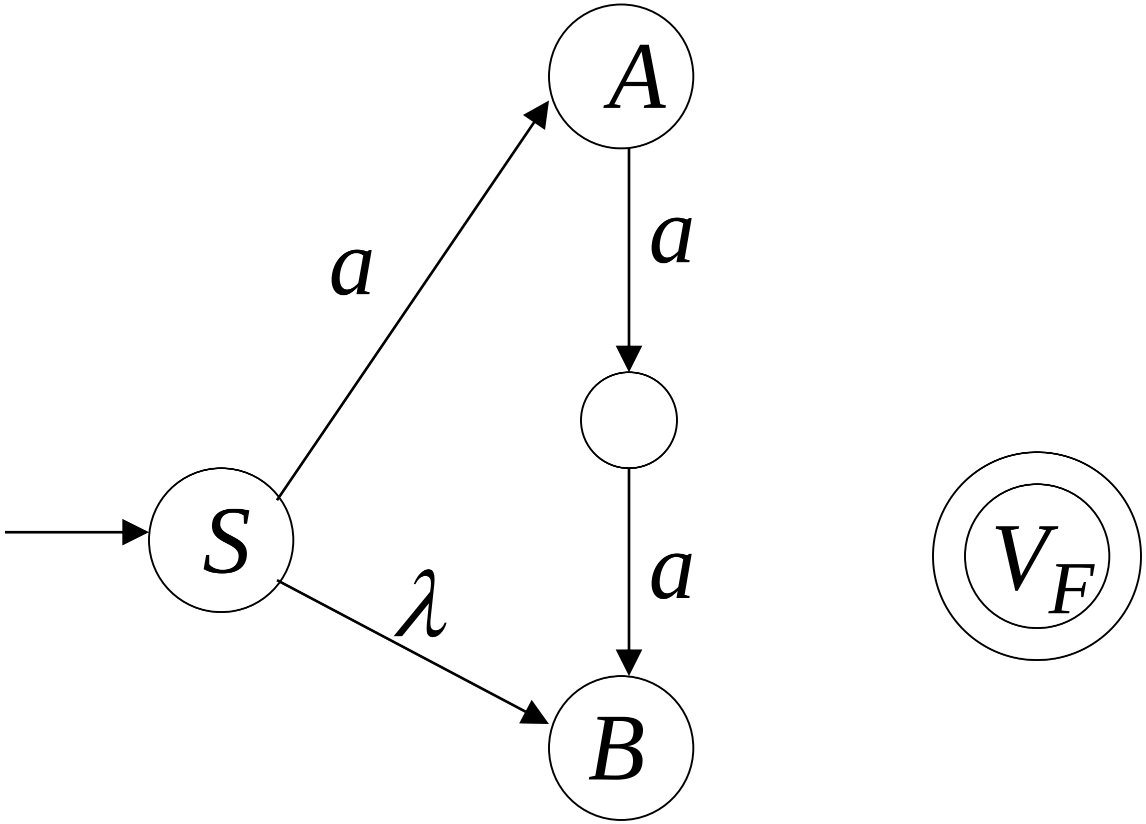
# Add edges for each production:




$$S \to aA$$

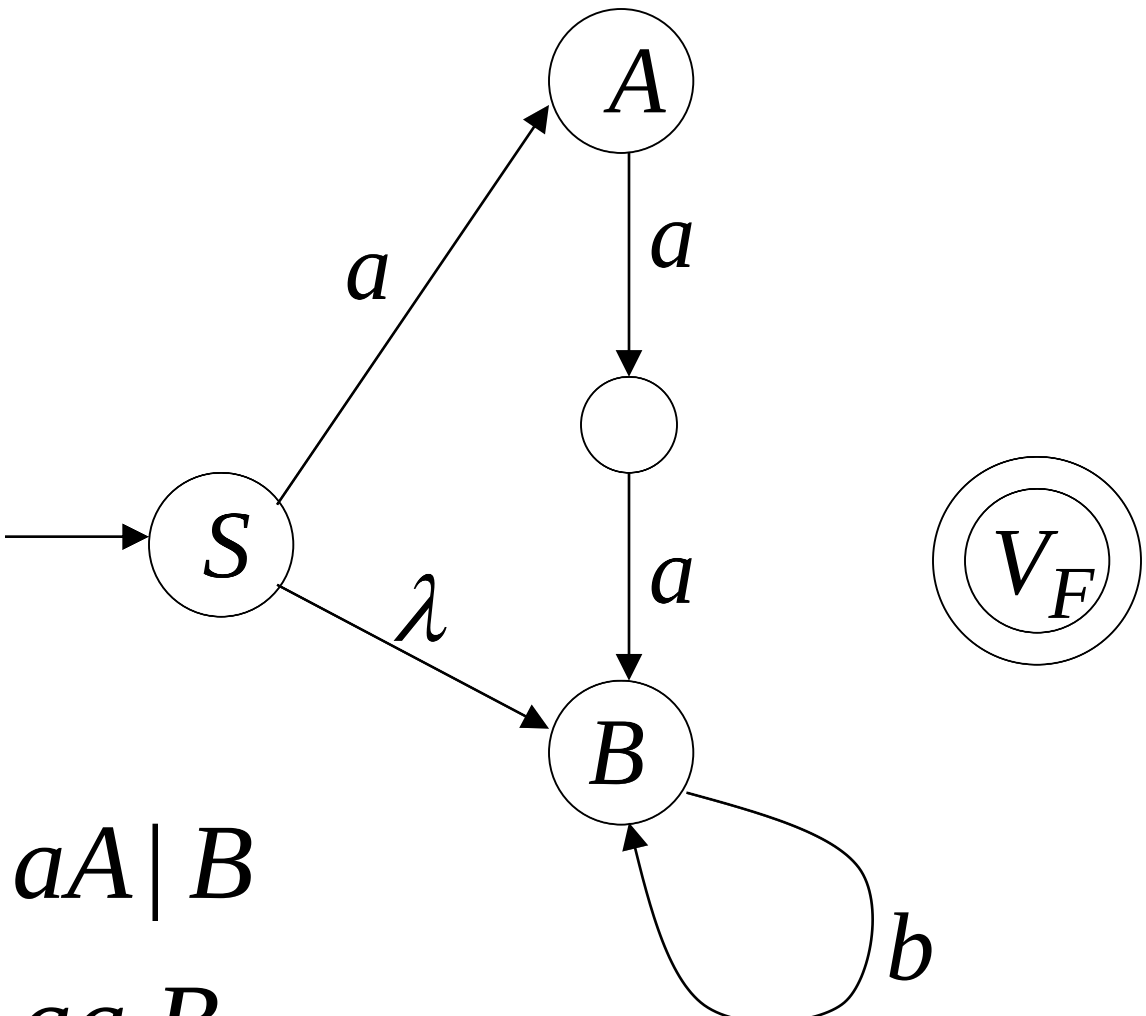



$$S \to aA \mid B$$




$$S \rightarrow aA \,|\, B$$

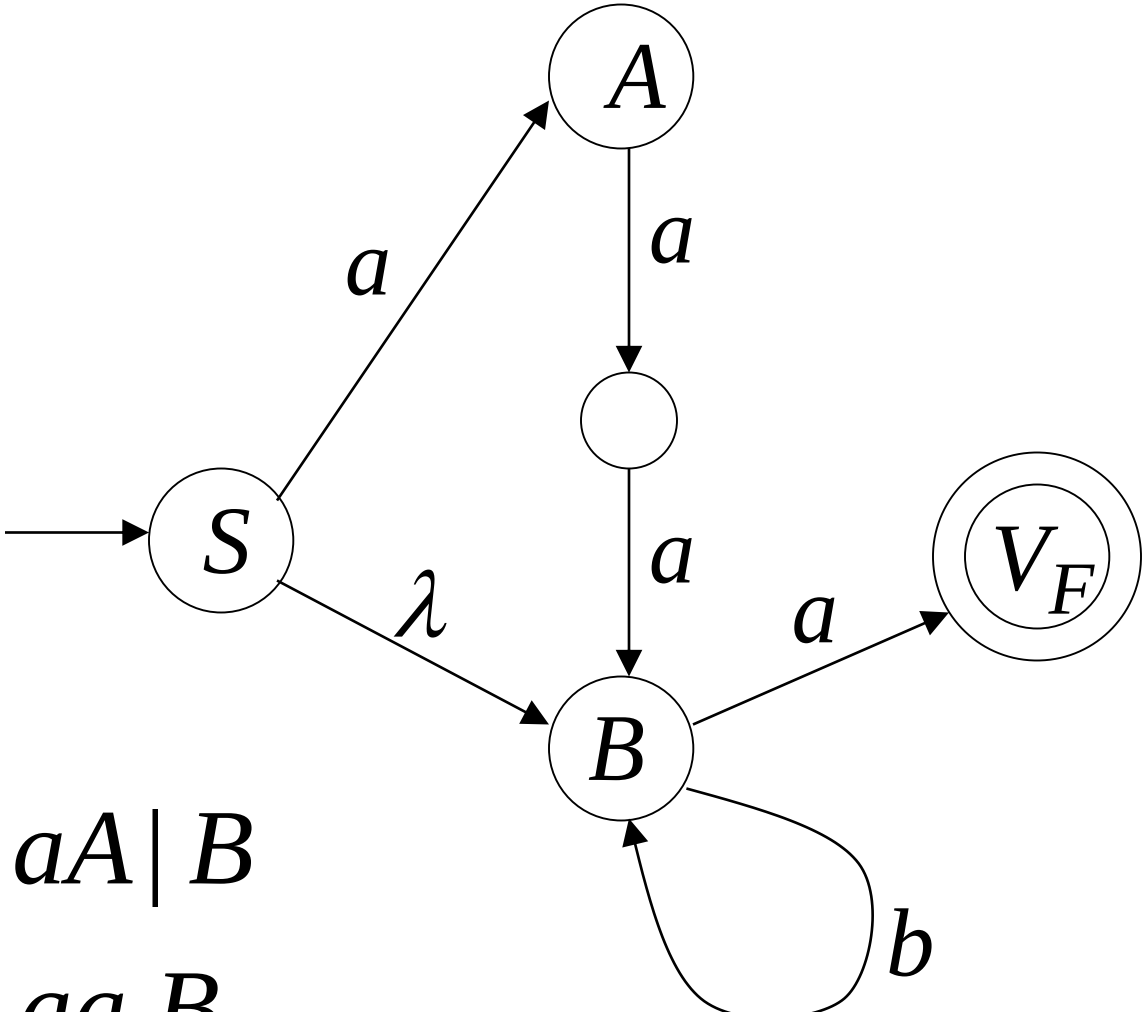
$$A \rightarrow aa\,B$$




$$S \rightarrow aA \mid B$$

$$A \rightarrow aa\, B$$

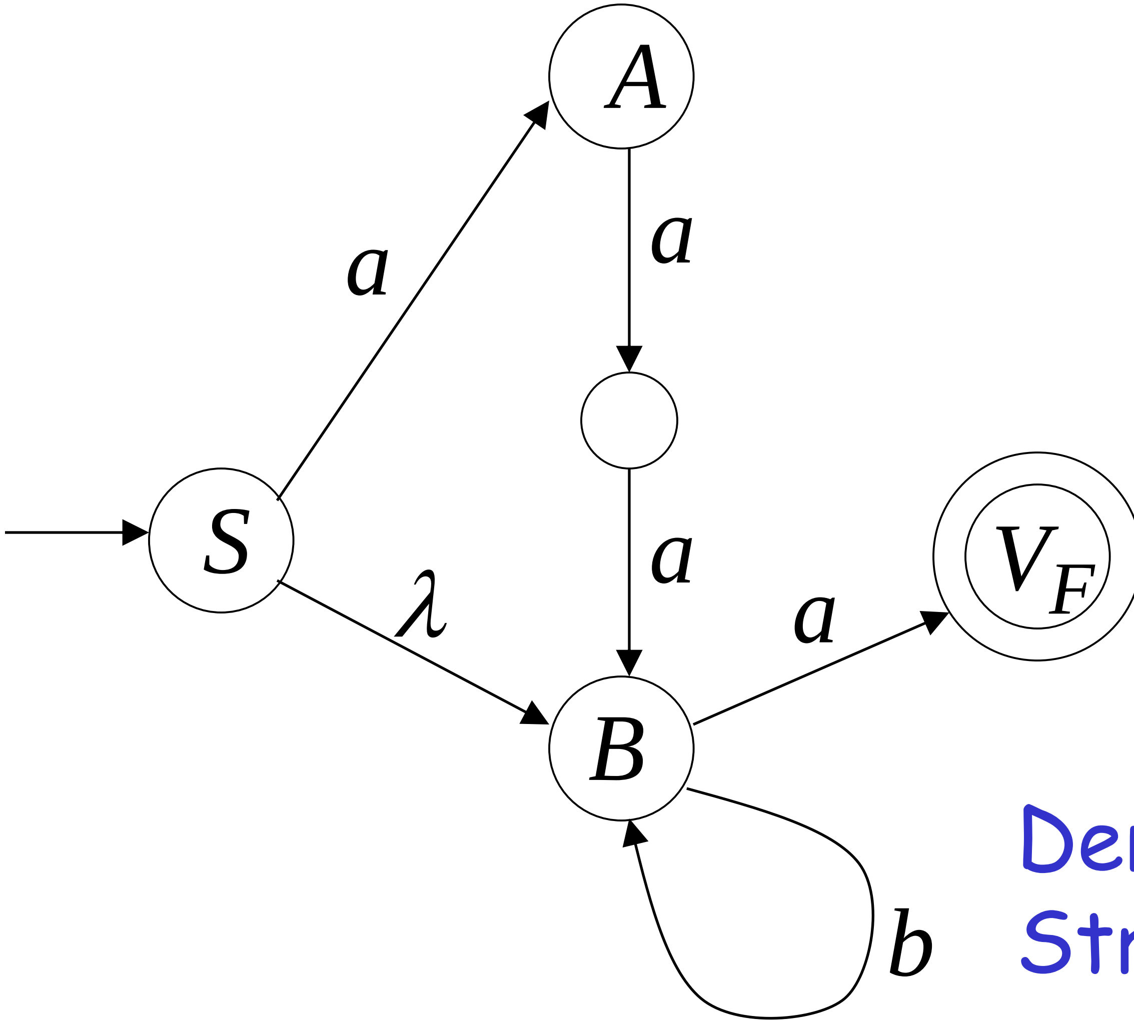
$$B \rightarrow bB$$




$$S \to aA \mid B$$

$$A \to aa\, B$$

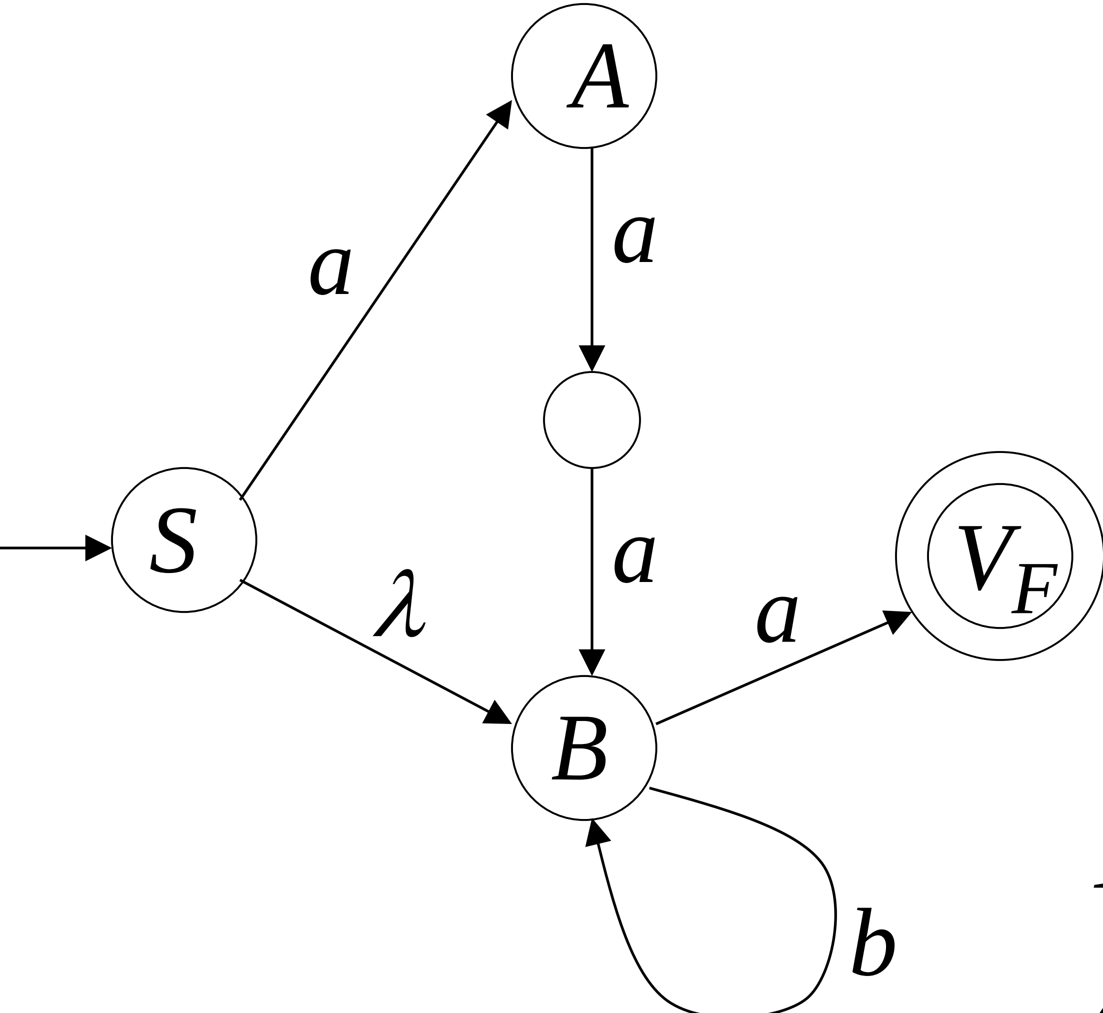
$$B \to bB \mid a$$




Derive the String aaaba

$$S \Rightarrow aA \Rightarrow aaaB \Rightarrow aaabB \Rightarrow aaaba$$

NFA $M$




Grammar

$G$

$$S \to aA \mid B$$

$$A \to aa\, B$$

$$B \to bB \mid a$$

$$L(M) = L(G) =$$
$$aaab^*a + b^*a$$

# In General

A right-linear grammar $G$

has variables: $V_0, V_1, V_2, \ldots$

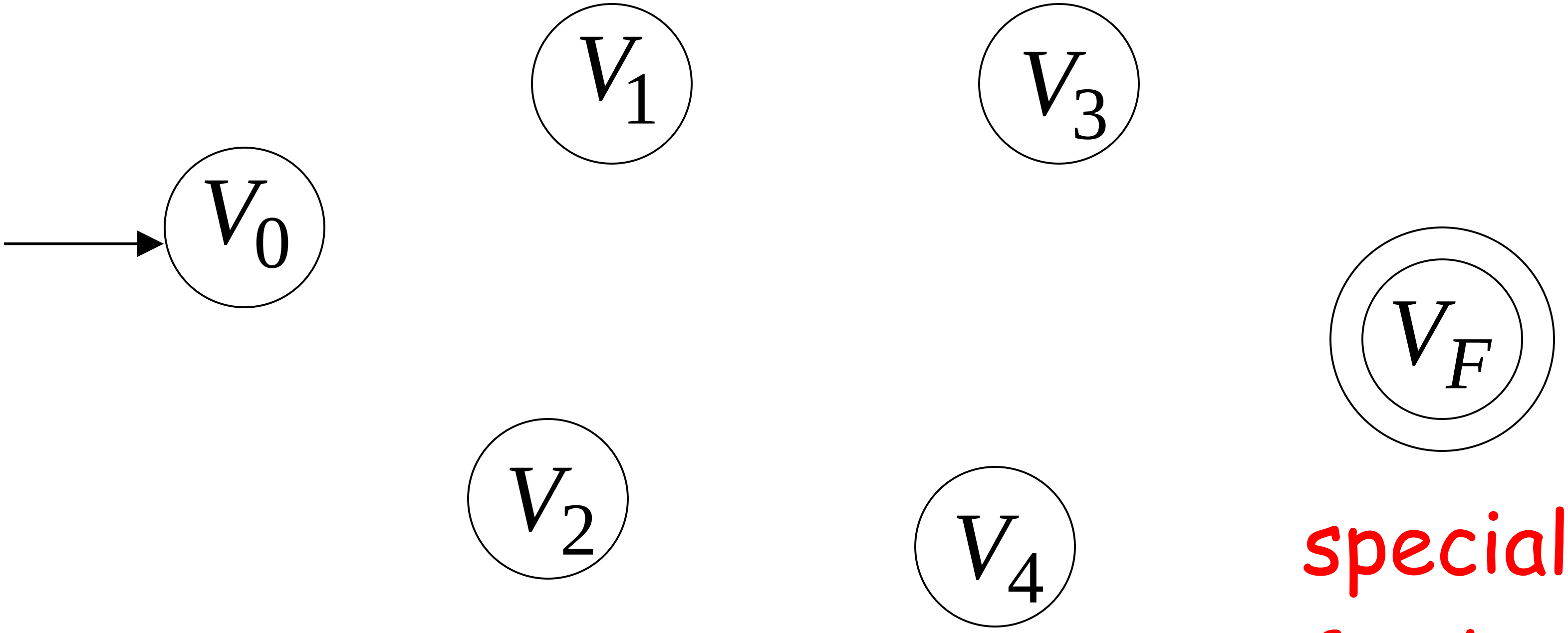
and productions: $V_i \to a_1 a_2 \cdots a_m V_j$

or

$$V_i \to a_1 a_2 \cdots a_m$$

We construct the NFA $M$ such that:

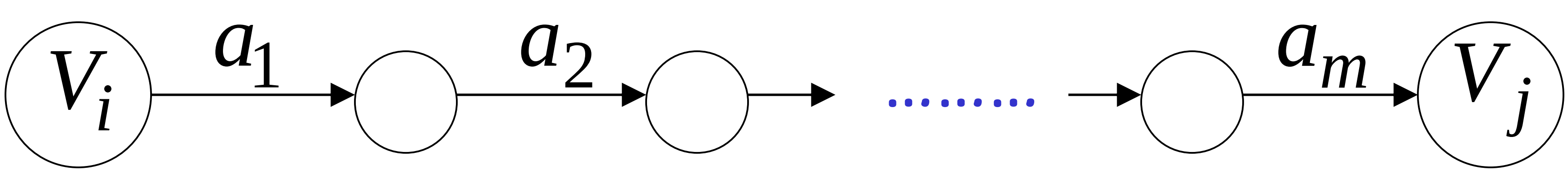
each variable $V_i$ corresponds to a node:




special final state

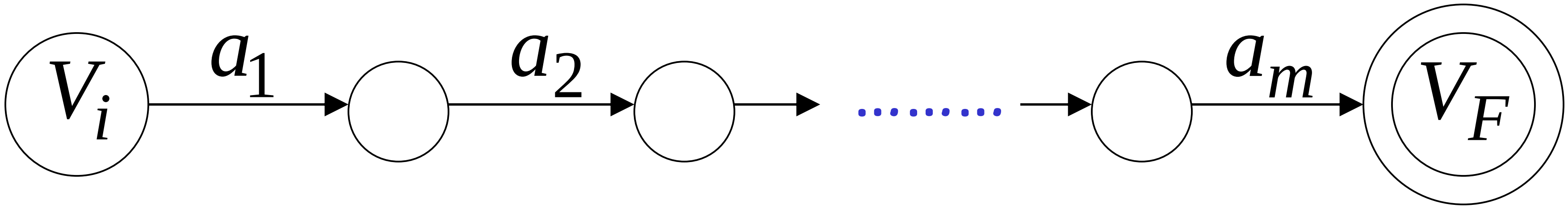
For each production: $V_i \to a_1 a_2 \cdots a_m V_j$

we add transitions and intermediate nodes

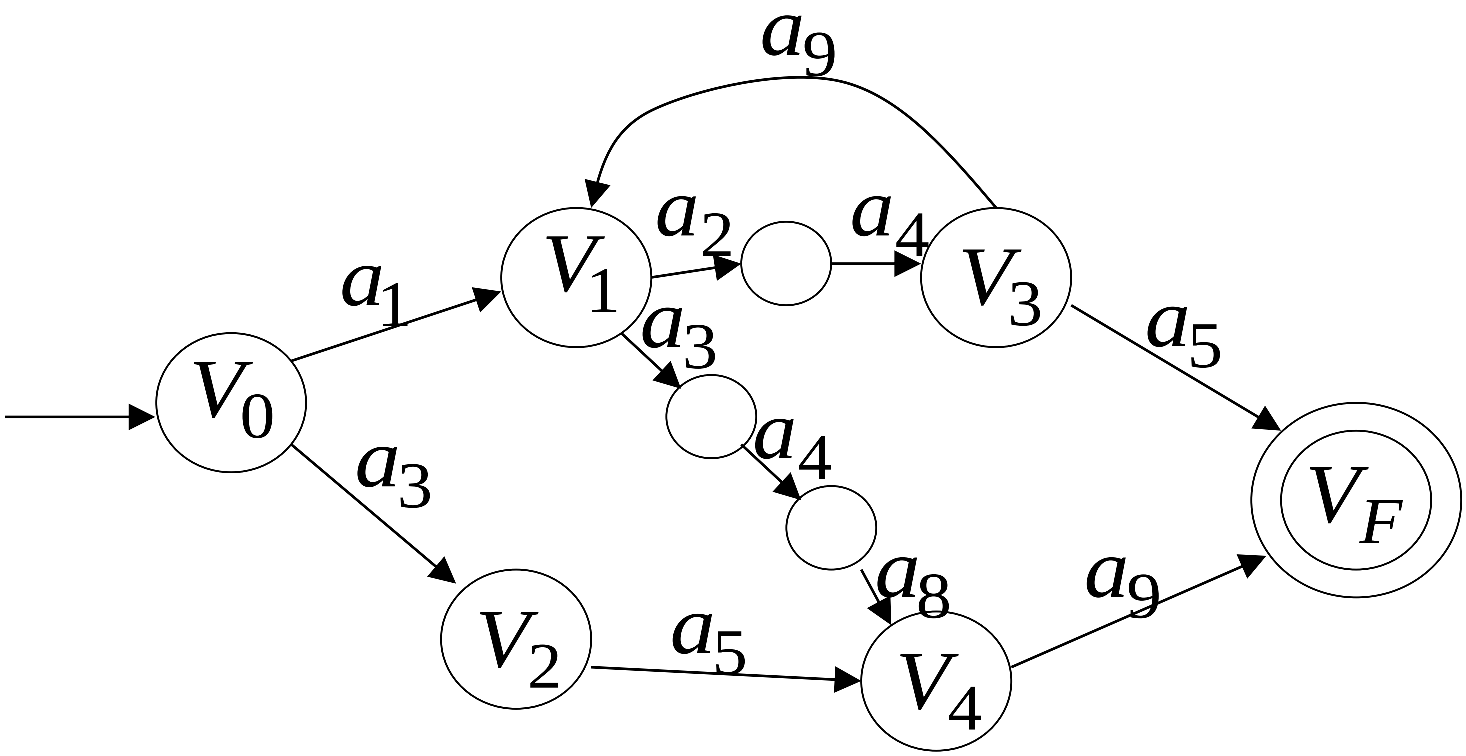


For each production: $V_i \to a_1 a_2 \cdots a_m$

we add transitions and intermediate nodes



Resulting NFA $M$ looks like this:




It holds that: $L(G) = L(M)$

# The case of Left-Linear Grammars

Let $G$ be a left-linear grammar

We will prove: $L(G)$ is regular

**Proof idea:**

We will construct a right-linear grammar $G'$ with $L(G) = L(G')^R$

Since $G$ is left-linear grammar the productions look like:

$$A \rightarrow B a_1 a_2 \cdots a_k$$

$$A \rightarrow a_1 a_2 \cdots a_k$$

Construct right-linear grammar $G'$

Left linear $G$

$$A \rightarrow B a_1 a_2 \cdots a_k$$

$$A \rightarrow Bv$$

Right linear $G'$

$$A \rightarrow a_k \cdots a_2 a_1 B$$

$$A \rightarrow v^R B$$

Construct right-linear grammar $G'$

Left linear $G$

$$A \to a_1 a_2 \cdots a_k$$

$$A \to v$$

Right linear $G'$

$$A \to a_k \cdots a_2 a_1$$

$$A \to v^R$$

It is easy to see that: $L(G) = L(G')^R$

Since $G'$ is right-linear, we have:

$L(G')$ $\Rightarrow$ $L(G')^R$ $\Rightarrow$ $L(G)$

| $L(G')$ | | $L(G')^R$ | | $L(G)$ |
|---|---|---|---|---|
| Regular Language | | Regular Language | | Regular Language |

# Proof - Part 2

$$\left\{ \begin{array}{l} \text{Languages} \\ \text{Generated by} \\ \text{Regular Grammars} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Regular} \\ \text{Languages} \end{array} \right\}$$

Any regular language $L$ is generated by some regular grammar $G$

Any regular language $L$ is generated by some regular grammar $G$

**Proof idea:**

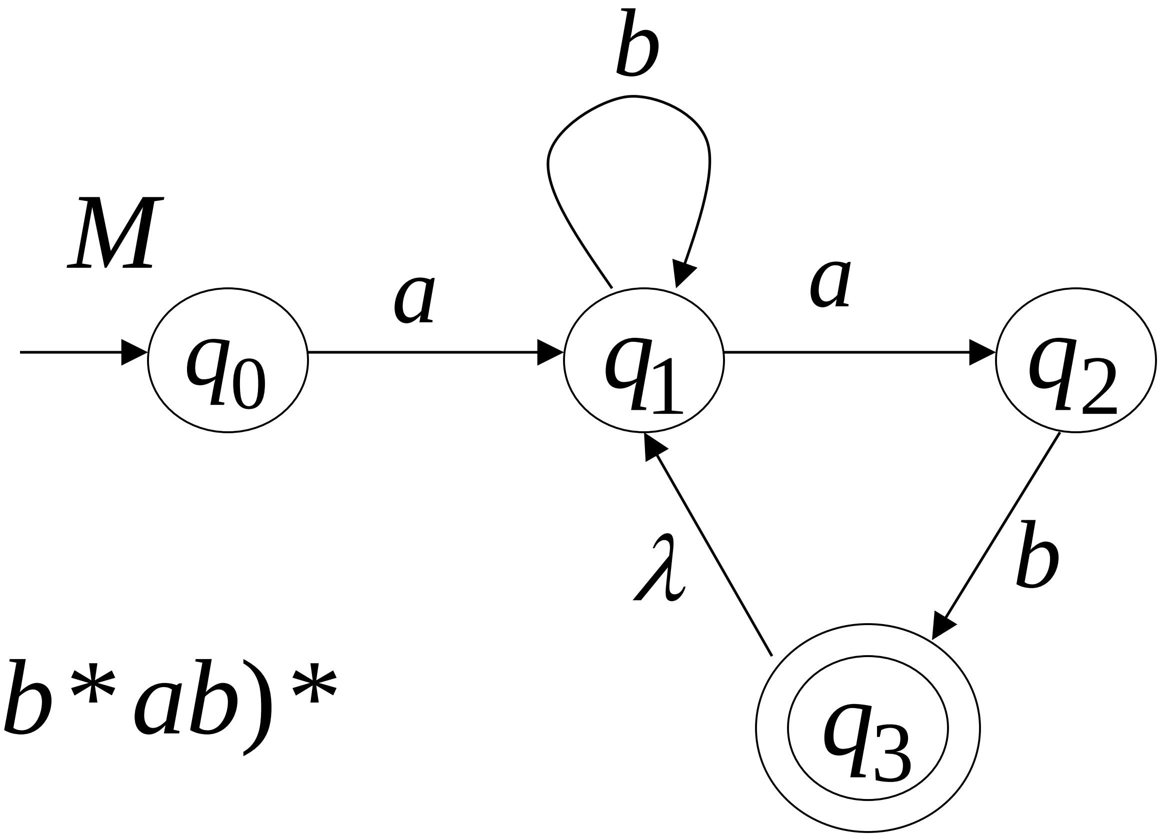
Let $M$ be the NFA with $L = L(M)$.

Construct from $M$ a regular grammar $G$ such that $L(M) = L(G)$

Since $L$ is regular

there is an NFA $M$ such that $L = L(M)$

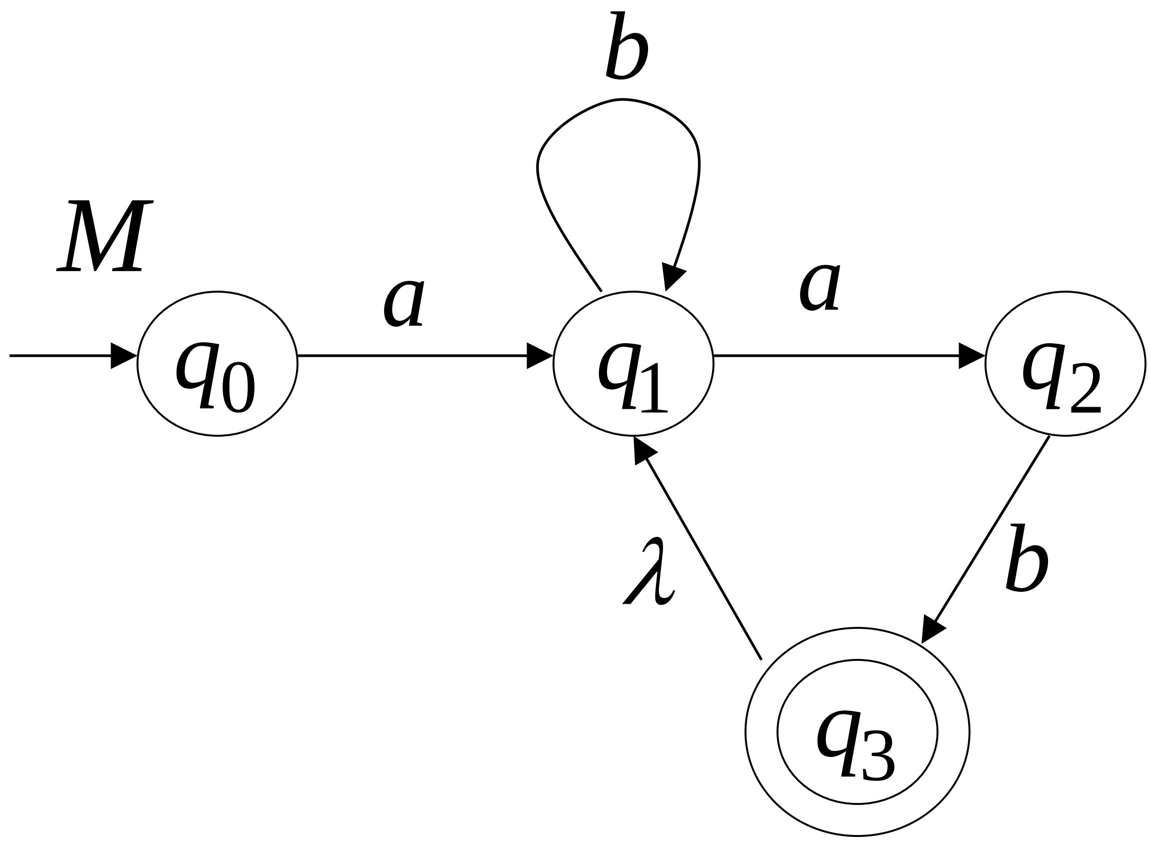
Example:



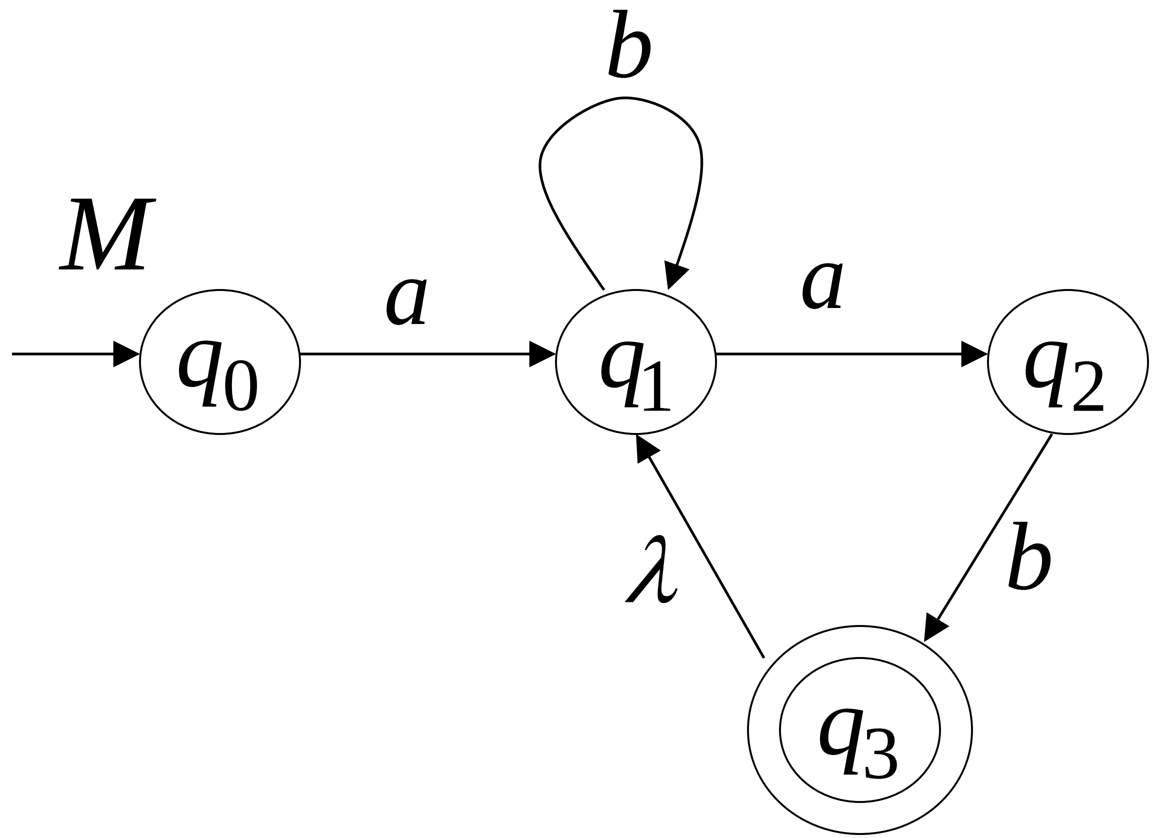

$$L = ab^*ab(b^*ab)^*$$

$$L = L(M)$$

Convert $M$ to a right-linear grammar




$$q_0 \to aq_1$$




$$q_0 \to aq_1$$

$$q_1 \to bq_1$$

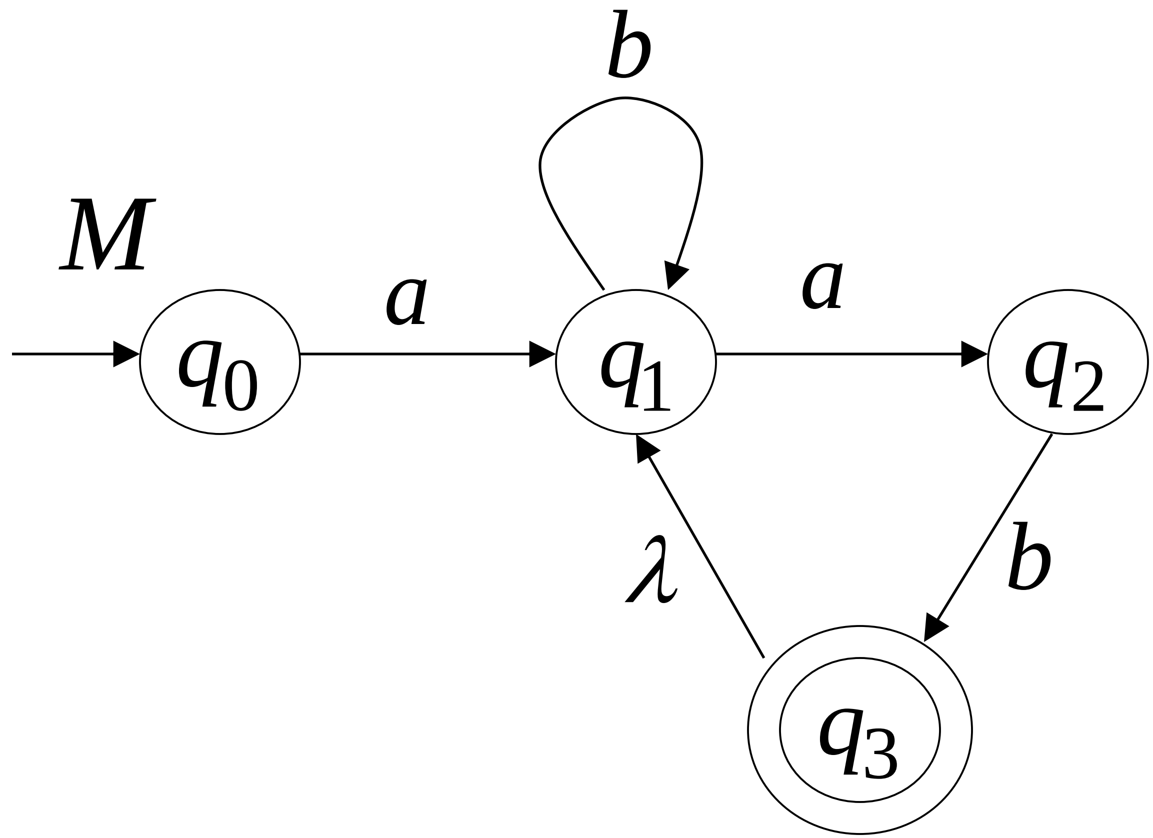
$$q_1 \to aq_2$$

$$q_0 \rightarrow aq_1$$

$$q_1 \rightarrow bq_1$$

$$q_1 \rightarrow aq_2$$

$$q_2 \rightarrow bq_3$$



$$L(G) = L(M) = L$$

$G$

$$q_0 \rightarrow a q_1$$

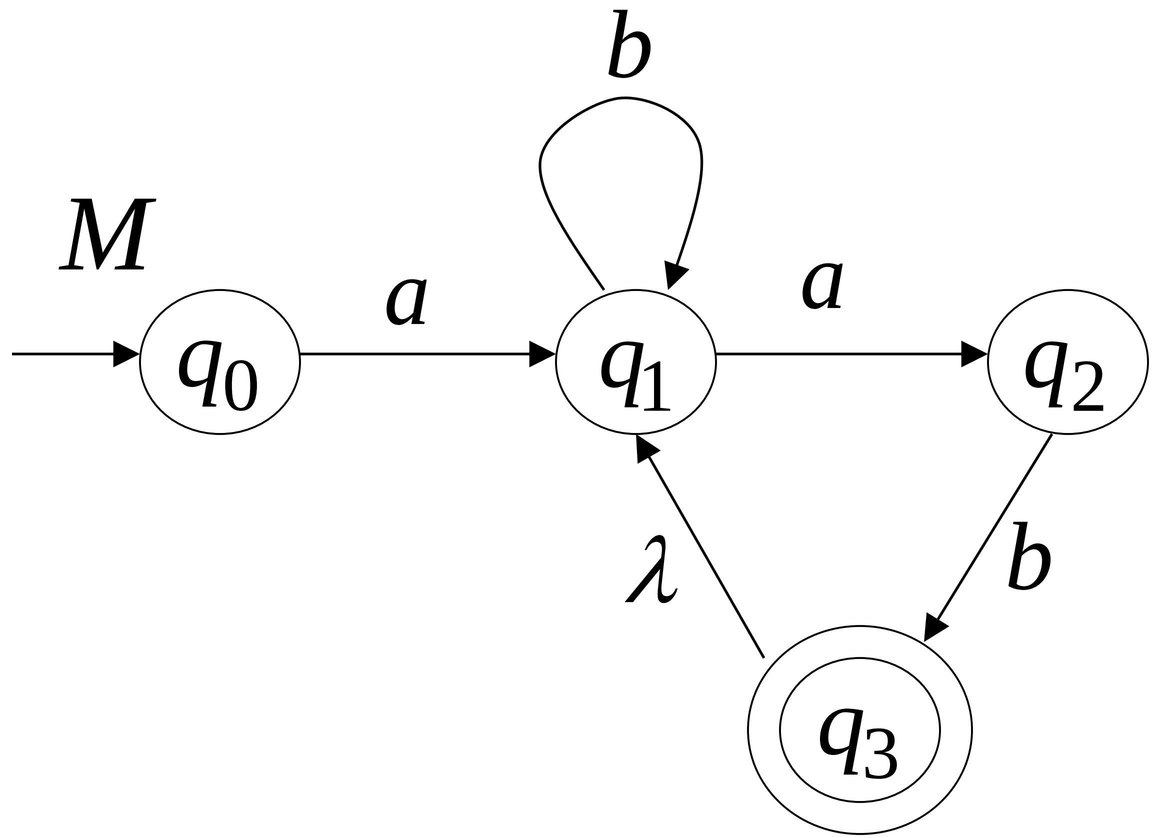
$$q_1 \rightarrow b q_1$$

$$q_1 \rightarrow a q_2$$

$$q_2 \rightarrow b q_3$$

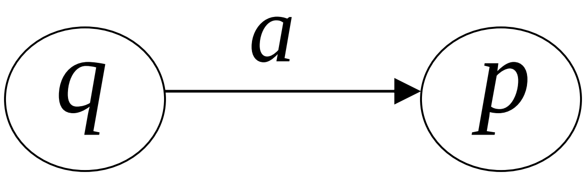
$$q_3 \rightarrow q_1$$

$$q_3 \rightarrow \lambda$$



# In General

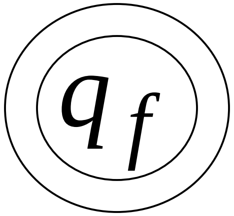
For any transition:




Add production:

$$q \to ap$$

variable terminal variable

For any final state: 


Add production: $q_f \to \lambda$

Since $G$ is right-linear grammar

$G$ is also a regular grammar

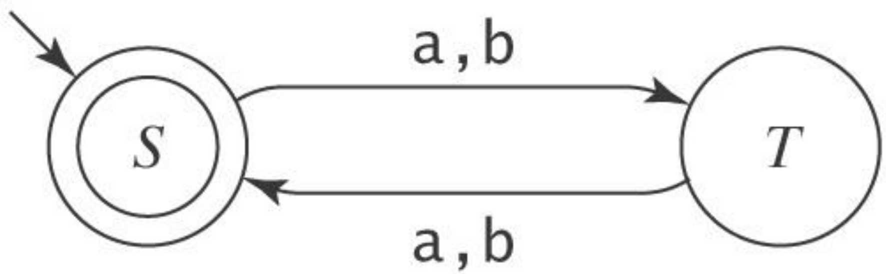
with $L(G) = L(M) = L$

**TABLE 5.1** Correspondence between Grammar Productions and Transitions in a Finite Automaton

| Transition | Production (right linear) | Remarks |
|---|---|---|
| $\delta(q_1, a) = q_2$ | $V_1 \rightarrow aV_2$ | The production generates the symbol consumed by the transition |
| $\delta(q_1, a) = q_1$ | $V_1 \rightarrow aV_1$ | Loop back to the same state or variable |
| $\delta(q_1, a) = q_f$ | $V_1 \rightarrow a$ | Final state $q_f$; no new variable introduced in the grammar |
| $\delta(q_1, \lambda) = q_2$ | $V_1 \rightarrow V_2$ | Just a change of variable (or state); such productions are called *unit productions* (see Chapter 7) |
| $\delta(q_1, \lambda) = q_1$ | $V_1 \rightarrow V_1$ | There is really no need for such a transition or production! |
| $\delta(q_1, \lambda) = q_f$ | $V_1 \rightarrow \lambda$ | Final state $q_f$; variable $V_1$ is eliminated; this is a special case and we do not consider this as the right-hand side being shorter than the left-hand side |

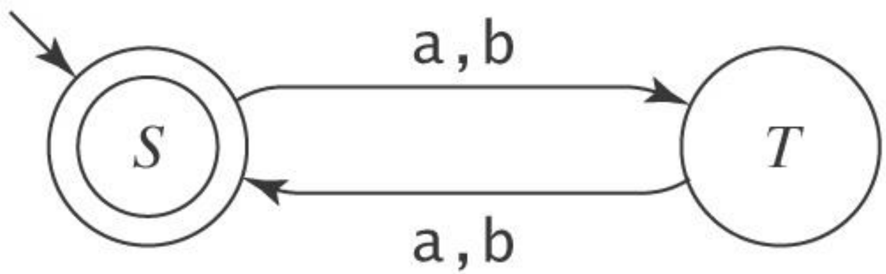
# Regular Grammar Example

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : |w| \text{ is even}\}$ $((\mathtt{aa}) \cup (\mathtt{ab}) \cup (\mathtt{ba}) \cup (\mathtt{bb}))^*$



# Regular Grammar Example

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : |w| \text{ is even}\}$ $\quad ((\mathtt{aa}) \cup (\mathtt{ab}) \cup (\mathtt{ba}) \cup (\mathtt{bb}))^*$

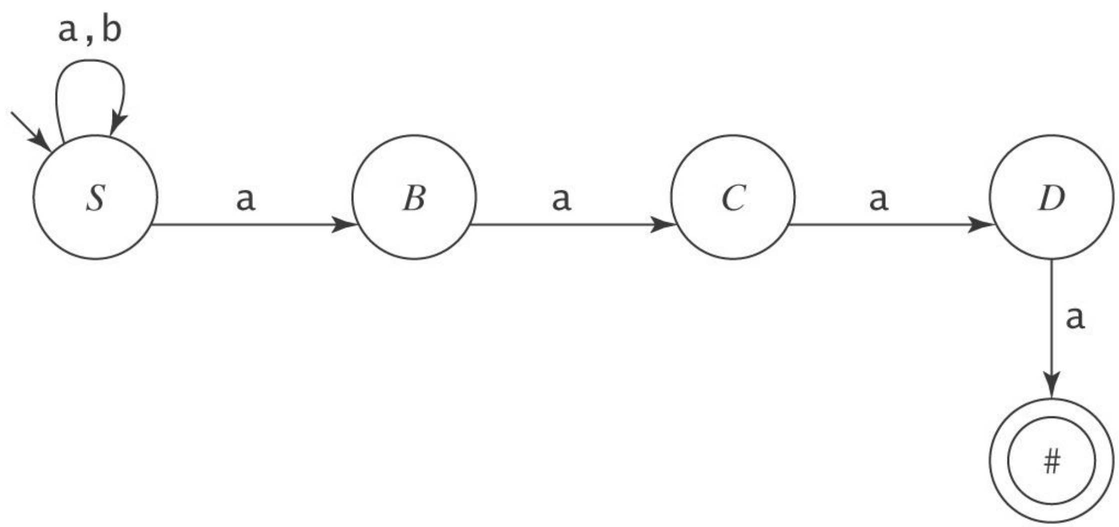



$S \to \varepsilon$
$S \to \mathtt{a}T$
$S \to \mathtt{b}T$
$T \to \mathtt{a}$
$T \to \mathtt{b}$
$T \to \mathtt{a}S$
$T \to \mathtt{b}S$

# Strings that End with aaaa

$L = \{w \in \{\texttt{a}, \texttt{b}\}^* : w$ ends with the pattern $\texttt{aaaa}\}$.

$$S \rightarrow \texttt{a}S$$
$$S \rightarrow \texttt{b}S$$
$$S \rightarrow \texttt{a}B$$
$$B \rightarrow \texttt{a}C$$
$$C \rightarrow \texttt{a}D$$
$$D \rightarrow \texttt{a}$$

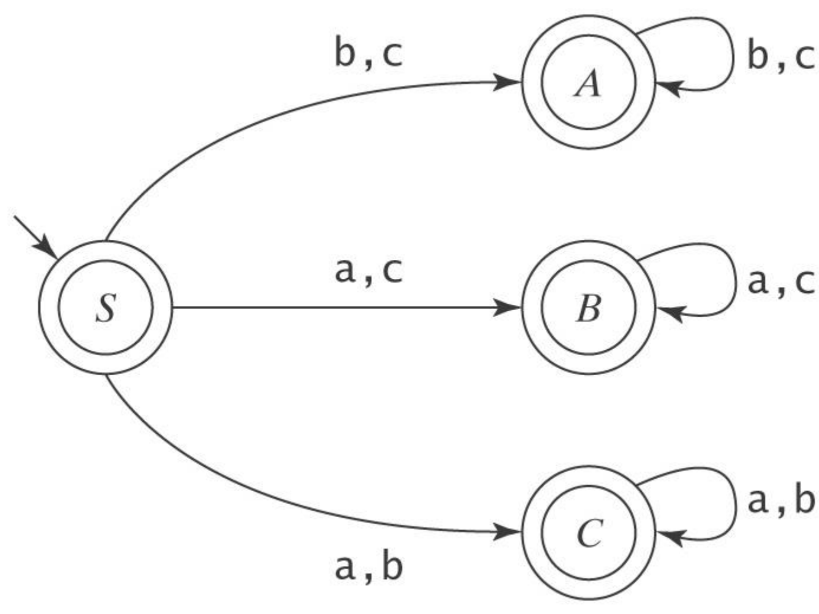


# Example 2 – One Character Missing

$S \to \varepsilon$
$S \to \mathtt{a}B$
$S \to \mathtt{a}C$
$S \to \mathtt{b}A$
$S \to \mathtt{b}C$
$S \to \mathtt{c}A$
$S \to \mathtt{c}B$

$A \to \mathtt{b}A$
$A \to \mathtt{c}A$
$A \to \varepsilon$
$B \to \mathtt{a}B$
$B \to \mathtt{c}B$
$B \to \varepsilon$

$C \to \mathtt{a}C$
$C \to \mathtt{b}C$
$C \to \varepsilon$



# Elementary Questions

# about

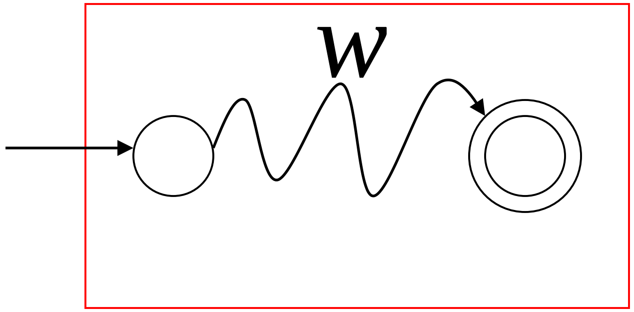
# Regular Languages

# Membership Question

**Question:** Given regular language $L$ and string $w$ how can we check if $w \in L$?

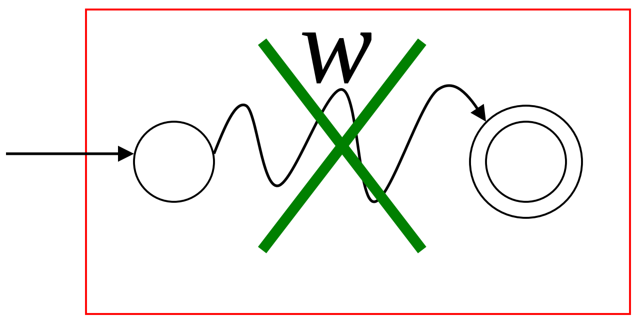
**Answer:** Take the DFA that accepts $L$ and check if $w$ is accepted

DFA




$w \in L$

DFA

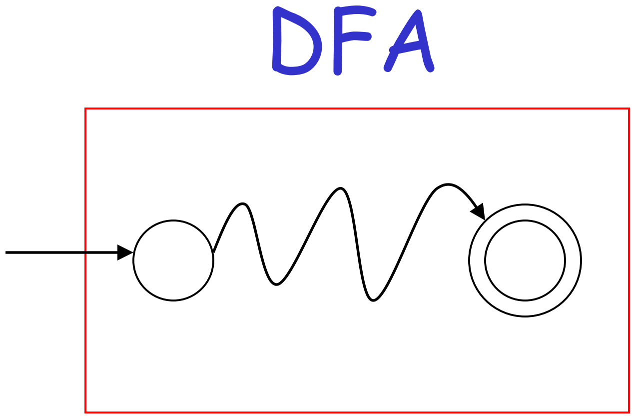



$w \notin L$

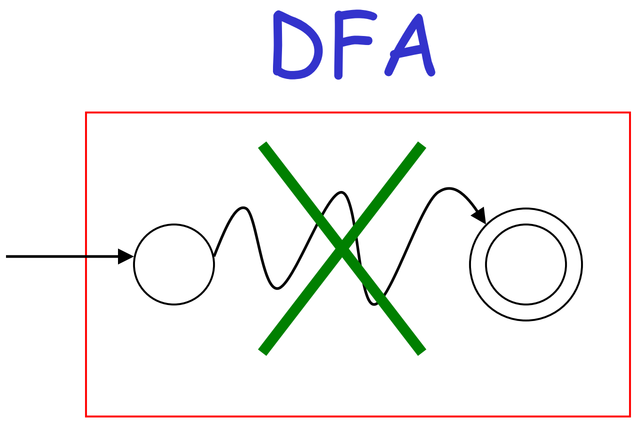
**Question:** Given regular language $L$ how can we check if $L$ is empty: $(L = \emptyset)$ ?

**Answer:** Take the DFA that accepts $L$

Check if there is any path from the initial state to a final state




$$L \neq \emptyset$$



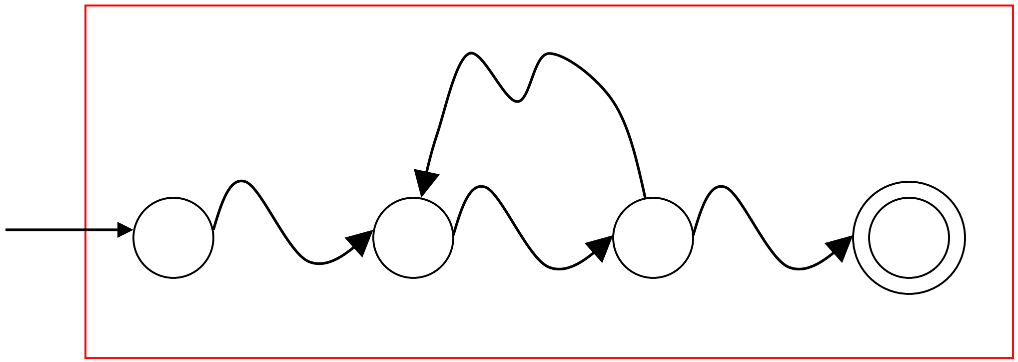

$$L = \emptyset$$

**Question:** Given regular language $L$ how can we check if $L$ is finite?

**Answer:** Take the DFA that accepts $L$

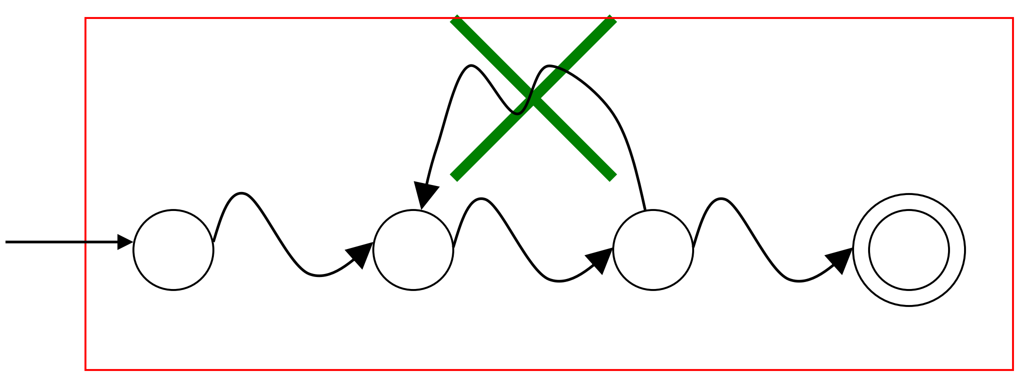
Check if there is a walk with cycle from the initial state to a final state

DFA



$L$ is infinite

DFA



$L$ is finite