DPDA

Deterministic PDA

Deterministic PDA: DPDA

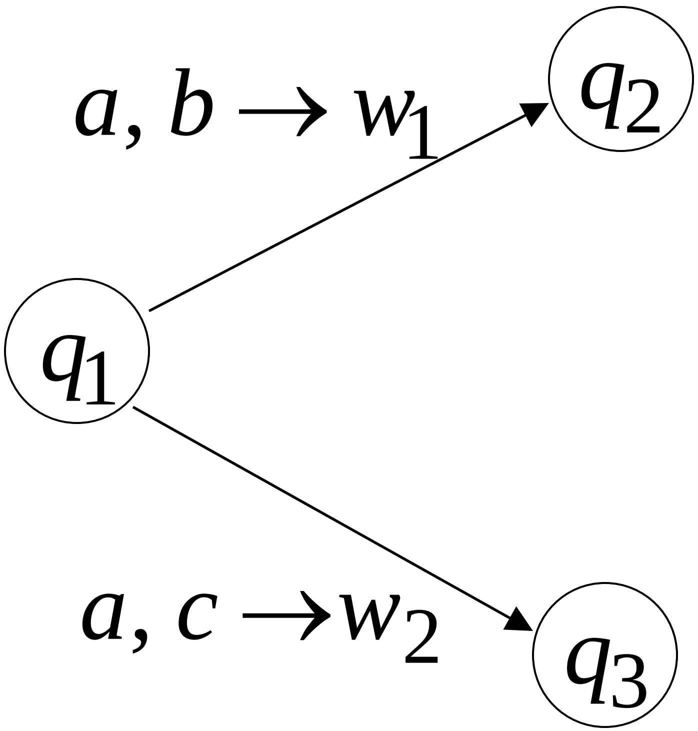
Allowed transitions:

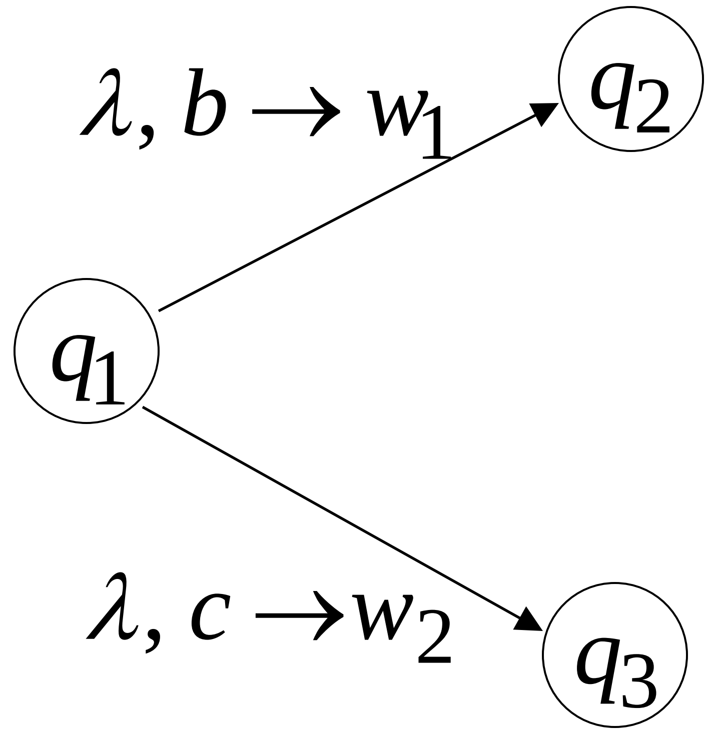
$$q_1 \xrightarrow{a,\, b \to w} q_2$$

$$q_1 \xrightarrow{\lambda,\, b \to w} q_2$$

(deterministic choices)

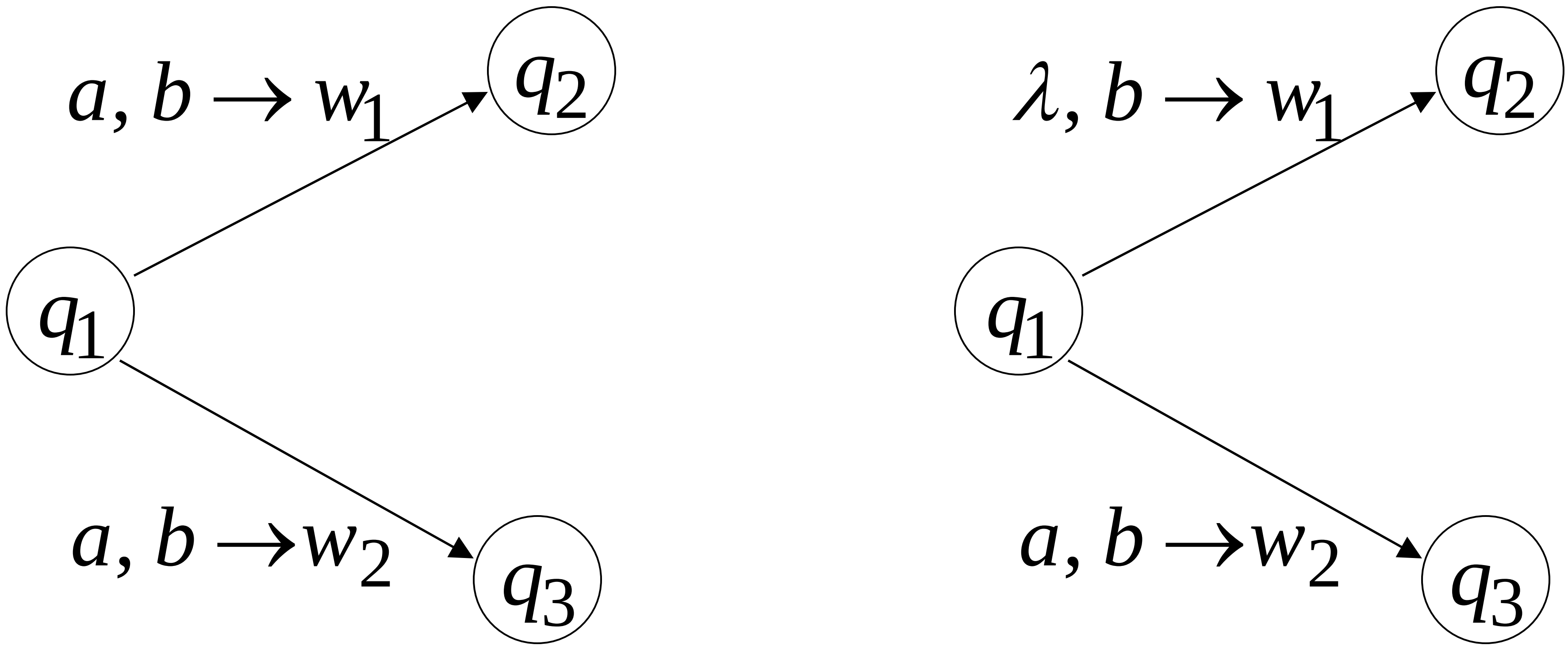
Allowed transitions:





(deterministic choices)

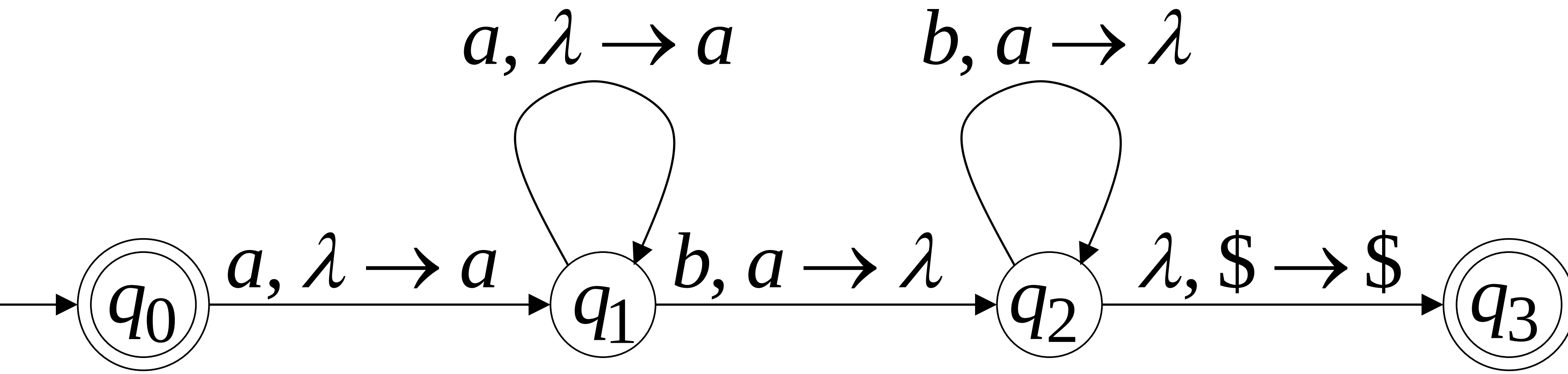
Not allowed:



(non deterministic choices)

DPDA example

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



$w c w^R, w \in \{a, b\}^*$

Definition:

A language L is **deterministic context-free** if there exists some DPDA that accepts it

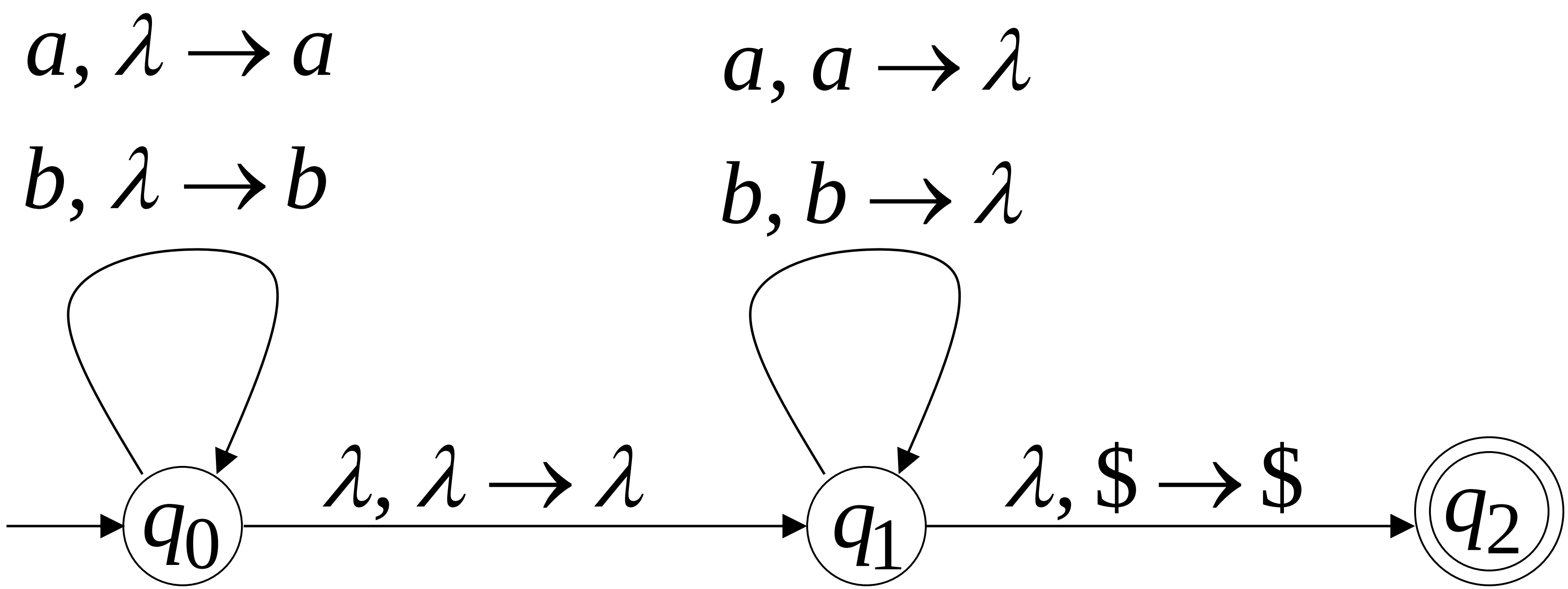
Example:

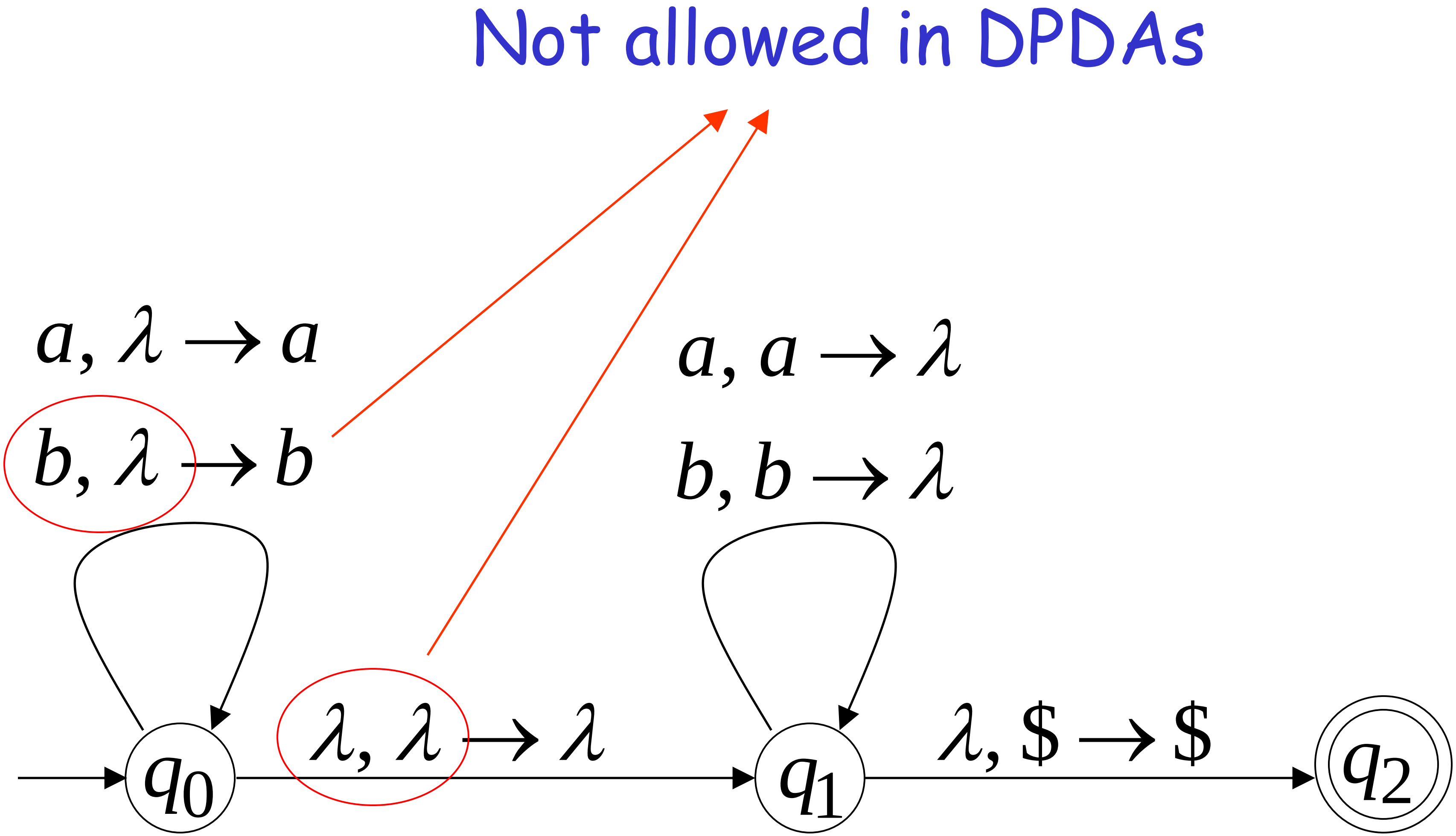
The language $L(M) = \{a^n b^n : n \geq 0\}$

is **deterministic context-free**

Example of Non-DPDA (PDA)

$$L(M) = \{vv^R : v \in \{a,b\}^*\}$$





Not allowed in DPDAs
$a, \lambda \rightarrow a$
$b, \lambda \rightarrow b$
$a, a \rightarrow \lambda$
$b, b \rightarrow \lambda$
$\lambda, \lambda \rightarrow \lambda$
$\lambda, \$ \rightarrow \$$
q_0
q_1
q_2

PDAs

Have More Power than

DPDAs

It holds that:

$$\left\{\begin{array}{c}\text{Deterministic}\\ \text{Context-Free}\\ \text{Languages}\\ \text{(DPDA)}\end{array}\right\} \subseteq \left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{PDAs}\end{array}\right\}$$

Since every DPDA is also a PDA

We will actually show:

$$\left\{\begin{array}{c}\text{Deterministic}\\ \text{Context-Free}\\ \text{Languages}\\ \text{(DPDA)}\end{array}\right\} \subset \left\{\begin{array}{c}\text{Context-Free}\\ \text{Languages}\\ \text{(PDA)}\end{array}\right\}$$

$L \notin$ $\qquad\qquad$ $L \in$

CFL

R.L

DCFL

We will show that there exists a context-free language L which is not accepted by any DPDA

The language is:

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\} \qquad n \geq 0$$

We will show:

- L is context-free
- L is **not** deterministic context-free

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

Language L is context-free

Context-free grammar for L :

$$S \to S_1 \mid S_2 \qquad \{a^n b^n\} \cup \{a^n b^{2n}\}$$

$$S_1 \to a S_1 b \mid \lambda \qquad \{a^n b^n\}$$

$$S_2 \to a S_2 bb \mid \lambda \qquad \{a^n b^{2n}\}$$

Theorem:

The language $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$

is **not** deterministic context-free

(there is **no** DPDA that accepts L)

Proof: Assume for contradiction that

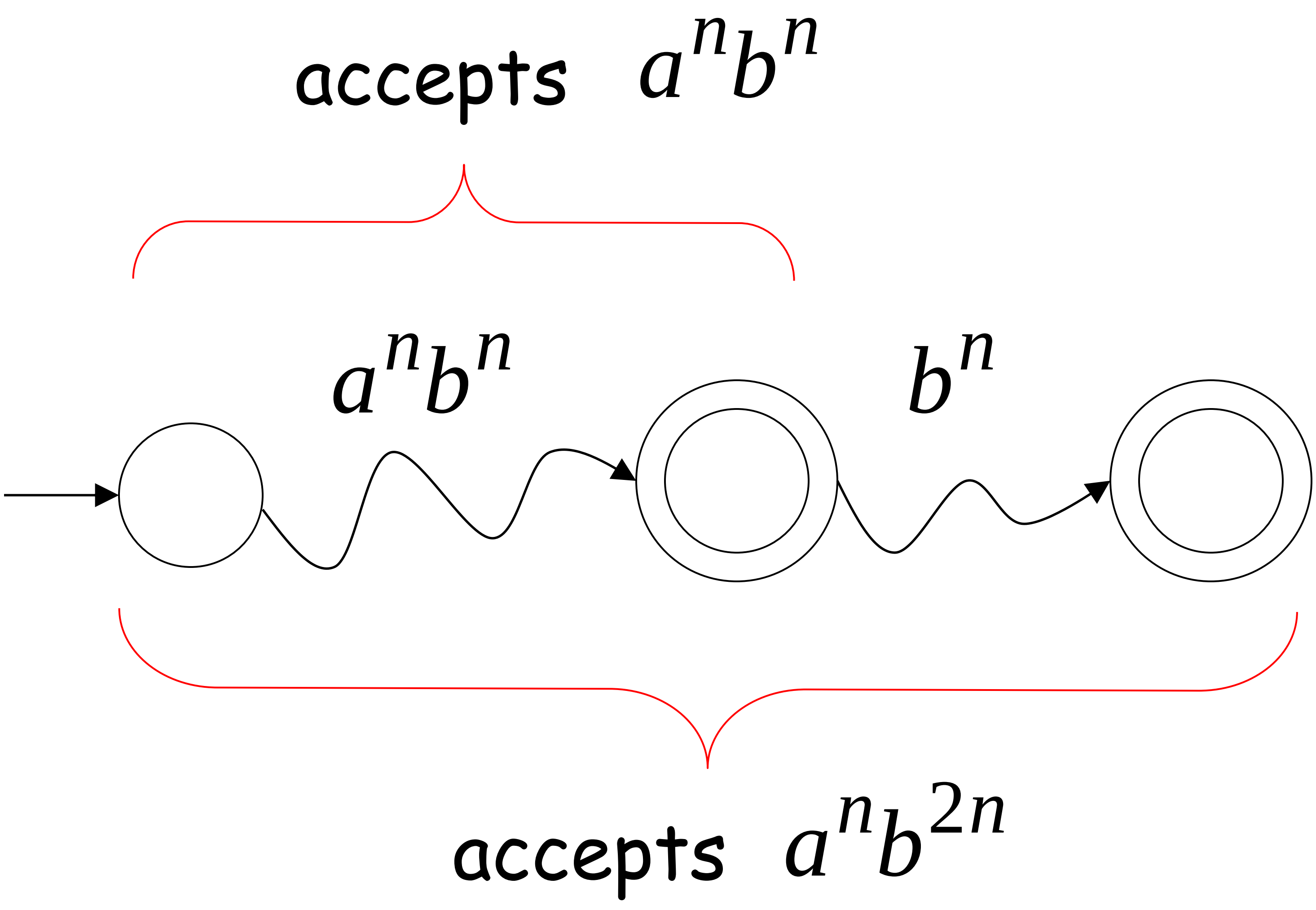
$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

is deterministic context free

Therefore:

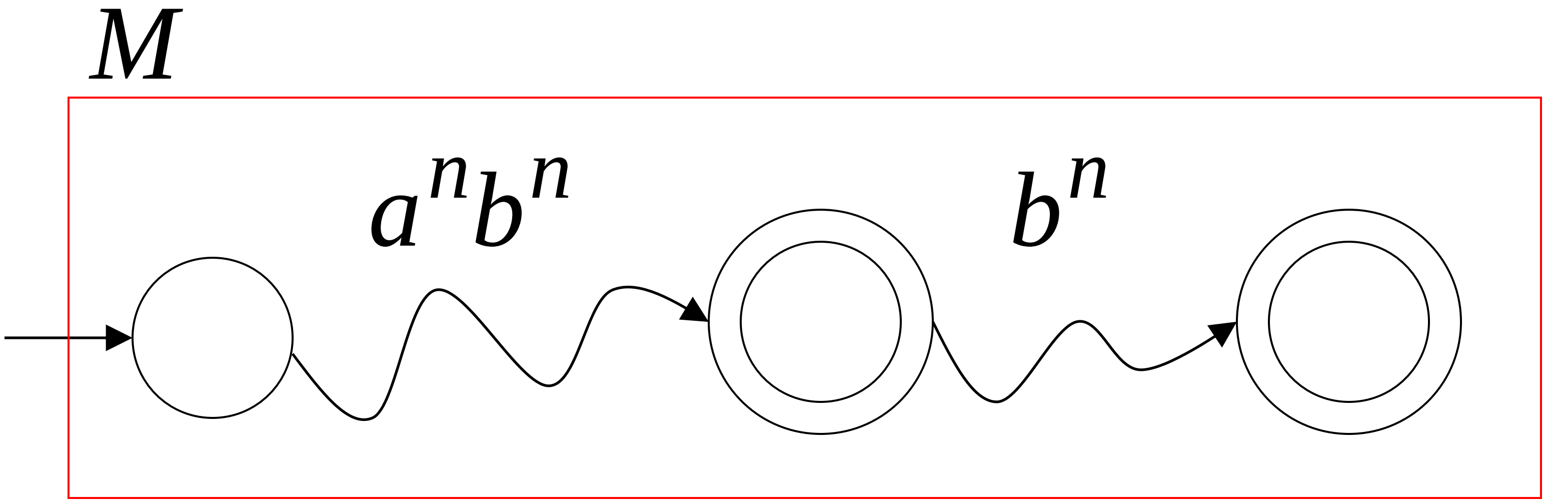
there is a DPDA M that accepts L

DPDA M with $L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$



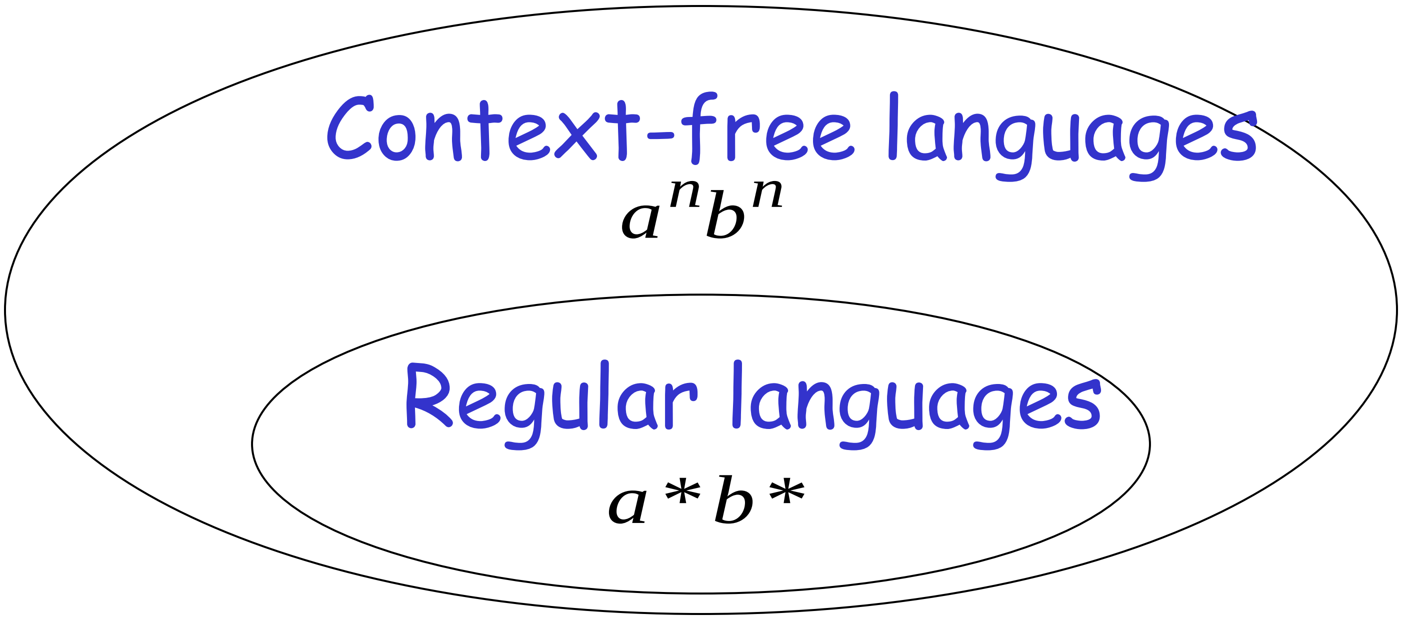
DPDA M with $L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$

Such a path exists due to determinism



Fact 1: The language $\{a^n b^n c^n\}$

is **not** context-free



(we will prove this at a later class using pumping lemma for context-free languages)

Fact 2: The language $L \cup \{a^n b^n c^n\}$ is **not** context-free

$$(L = \{a^n b^n\} \cup \{a^n b^{2n}\})$$

(we can prove this using pumping lemma for context-free languages)

We will construct a PDA that accepts:

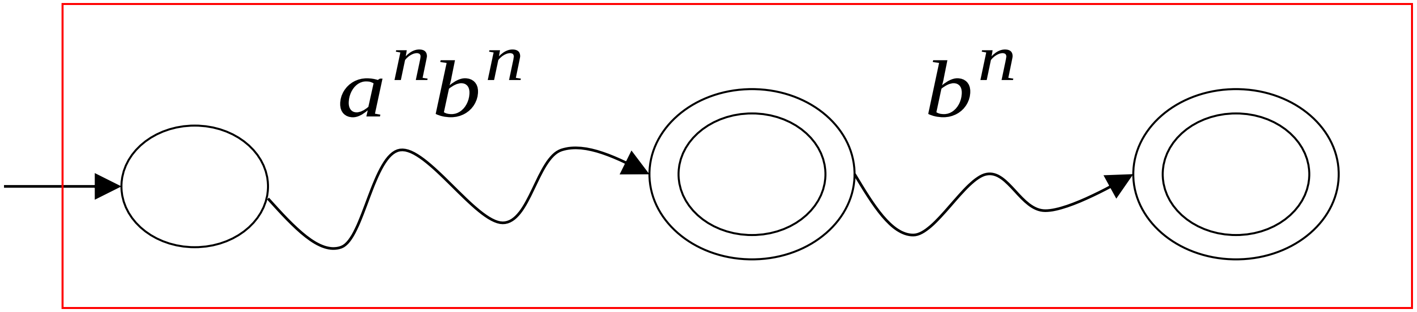
$$L \cup \{a^n b^n c^n\}$$

$a^n b^n b^n$

$$(L = \{a^n b^n\} \cup \{a^n b^{2n}\})$$

which is a contradiction!

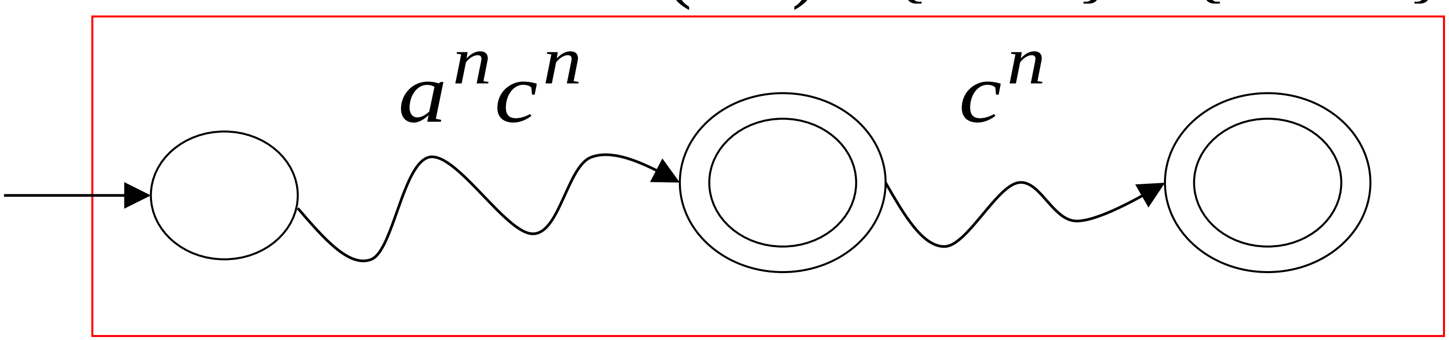
DPDA M $L(M)=\{a^n b^n\}\cup\{a^n b^{2n}\}$



Modify M

Replace b
with c

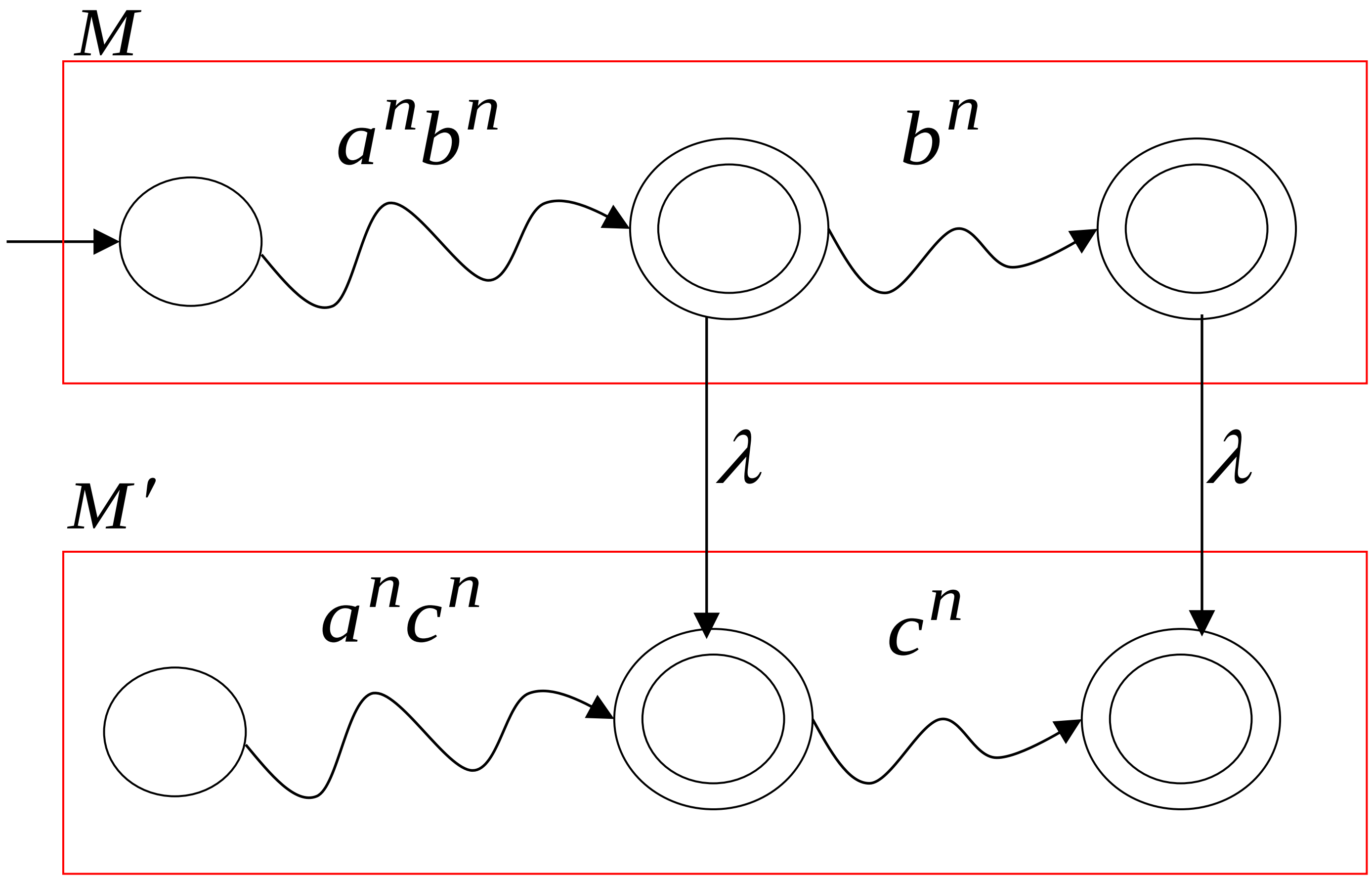
DPDA M' $L(M')=\{a^n c^n\}\cup\{a^n c^{2n}\}$



A PDA that accepts $L \cup \{a^n b^n c^n\}$

Connect the final states of M

with the final states of M'



Since $L \cup \{a^n b^n c^n\}$ is accepted by a PDA

it is context-free

Contradiction!

(since $L \cup \{a^n b^n c^n\}$ is not context-free)

Therefore:

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

Is not deterministic context free

There is **no** DPDA that accepts it

End of Proof

Normal Forms for Context-free Grammars

$L = \{a^n b^n\} \cup \{a\}$

Chomsky Normal Form

Each productions has form:

$$A \rightarrow BC \qquad \text{or} \qquad A \rightarrow a$$

variable variable terminal

Examples:

$$S \to AS$$
$$S \to a$$
$$A \to SA$$
$$A \to b$$

Chomsky
Normal Form

$$S \to AS$$
$$S \to AAS$$
$$A \to SA$$
$$A \to aa$$

Not Chomsky
Normal Form

Convertion to Chomsky Normal Form

- Example:

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

Not Chomsky
Normal Form

Introduce variables for terminals: T_a, T_b, T_c

$$S \to ABa$$
$$A \to aab$$
$$B \to Ac$$

$$S \to ABT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_1

$$S \to ABT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$\Longrightarrow$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Introduce intermediate variable: V_2

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aT_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

$$S \to AV_1$$
$$V_1 \to BT_a$$
$$A \to T_aV_2$$
$$V_2 \to T_aT_b$$
$$B \to AT_c$$
$$T_a \to a$$
$$T_b \to b$$
$$T_c \to c$$

Final grammar in Chomsky Normal Form:

Initial grammar

$$S \to ABa$$

$$A \to aab$$

$$B \to Ac$$

$$S \to AV_1$$

$$V_1 \to BT_a$$

$$A \to T_aV_2$$

$$V_2 \to T_aT_b$$

$$B \to AT_c$$

$$T_a \to a$$

$$T_b \to b$$

$$T_c \to c$$

In general:

From any context-free grammar
(which doesn't produce λ)
not in Chomsky Normal Form

we can obtain:

An equivalent grammar
in Chomsky Normal Form

The Procedure

First remove:

Nullable variables

Unit productions

Then, for every symbol a:

Add production $T_a \rightarrow a$

In productions: replace a with T_a

New variable: T_a

Replace any production $A \to C_1 C_2 \cdots C_n$

with
$$A \to C_1 V_1$$
$$V_1 \to C_2 V_2$$
$$\cdots$$
$$V_{n-2} \to C_{n-1} C_n$$

New intermediate variables: $V_1, V_2, \ldots, V_{n-2}$

Theorem: For any context-free grammar (which doesn't produce λ) there is an equivalent grammar in Chomsky Normal Form

Observations

- Chomsky normal forms are good for parsing and proving theorems

- It is very easy to find the Chomsky normal form for any context-free grammar

Greinbach Normal Form

All productions have form:

$$A \to a V_1 V_2 \cdots V_k \qquad k \geq 0$$

symbol

variables

Examples:

$$S \to cAB$$

$$A \to aA \mid bB \mid b$$

$$B \to b$$

Greinbach
Normal Form

$$S \to abSb$$

$$S \to aa$$

Not Greinbach
Normal Form

Conversion to Greinbach Normal Form:

$$S \to abSb$$

$$S \to aa$$

$$S \to aT_bST_b$$

$$S \to aT_a$$

$$T_a \to a$$

$$T_b \to b$$

Greinbach Normal Form

Theorem: For any context-free grammar (which doesn't produce λ) there is an equivalent grammar in Greinbach Normal Form

Observations

- Greinbach normal forms are very good for parsing

- It is hard to find the Greinbach normal form of any context-free grammar

The CYK Parser

The CYK Membership Algorithm

Input:

- Grammar G in Chomsky Normal Form
- String w

Output:

find if $w \in L(G)$

this claim. The algorithm we will describe here is called the CYK algorithm, after its originators J. Cocke, D. H. Younger, and T. Kasami. The algorithm works only if the grammar is in Chomsky normal form and succeeds by breaking one problem into a sequence of smaller ones in the following way. Assume that we have a grammar $G = (V, T, S, P)$ in Chomsky normal form and a string

$$w = a_1 a_2 \cdots a_n.$$

We define substrings

$$w_{ij} = a_i \cdots a_j,$$

and subsets of V

$$V_{ij} = \left\{ A \in V : A \stackrel{*}{\Rightarrow} w_{ij} \right\}.$$

Clearly, $w \in L(G)$ if and only if $S \in V_{1n}$.

To compute V_{ij}, observe that $A \in V_{ii}$ if and only if G contains a production $A \rightarrow a_i$. Therefore, V_{ii} can be computed for all $1 \leq i \leq n$ by inspection of w and the productions of the grammar. To continue, notice that for $j > i$, A derives w_{ij} if and only if there is a production $A \rightarrow BC$, with $B \stackrel{*}{\Rightarrow} w_{ik}$ and $C \stackrel{*}{\Rightarrow} w_{k+1j}$ for some k with $i \leq k, k < j$. In other words,

$$V_{ij} = \bigcup_{k \in \{i, i+1, \ldots, j-1\}} \{A : A \rightarrow BC, \text{ with } B \in V_{ik}, C \in V_{k+1,j}\}. \quad (6.8)$$

An inspection of the indices in (6.8) shows that it can be used to compute all the V_{ij} if we proceed in the sequence

1. Compute $V_{11}, V_{22}, \ldots, V_{nn}$

2. Compute $V_{12}, V_{23}, \ldots, V_{n-1,n}$

3. Compute $V_{13}, V_{24}, \ldots, V_{n-2,n}$

The Algorithm

Input example:

- Grammar G :
$$S \rightarrow AB$$
$$A \rightarrow BB$$
$$A \rightarrow a$$
$$B \rightarrow AB$$
$$B \rightarrow b$$

- String w : $aabbb$

aabbb

1 2 3 4 5

a V_{11}	a V_{22}	b V_{33}	b V_{44}	b V_{55}
aa V_{12}	ab V_{23}	bb V_{34}	bb V_{45}	
aab V_{13}	abb V_{24}	bbb V_{35}		
aabb V_{14}	abbb V_{25}			
aabbb V_{15}				

$S \to AB$

$A \to BB$

$A \to a$

$B \to AB$

$B \to b$

w_{11}				
a	a	b	b	b
A	A	B	B	B
w_{12} aa	~~ab~~	bb	bb	
$\emptyset$	S, B			
aab	abb	bbb		
w_{13} S, B	S			
aabb	abbb			
w_{14}				
aabbb				

1 | 2 | 3

$(w_{11})(w_{23})$ — S, B $\cup$

$(w_{12})(w_{33})$ — $\emptyset$

w_{11} w_{24}

w_{12} w_{34}

w_{13} w_{44}

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

a	a	b	b	b
A	A	B	B	B
aa	ab	bb	bb	
	S,B	A	A	
aab	abb	bbb		
aabb	abbb			
aabbb				

$$S \to AB$$

$$A \to BB$$

$$A \to a$$

$$B \to AB$$

$$B \to b$$

a	a	b	b	b
A	A	B	B	B
aa	ab	bb	bb	
	S,B	A	A	
aab	abb	bbb		
S,B	A	S,B		
aabb	abbb			
A	S,B			
aabbb				
S,B				

Therefore: $aabbb \in L(G)$

Time Complexity: $|w|^3$

Observation: The CYK algorithm can be easily converted to a parser (bottom up parser)