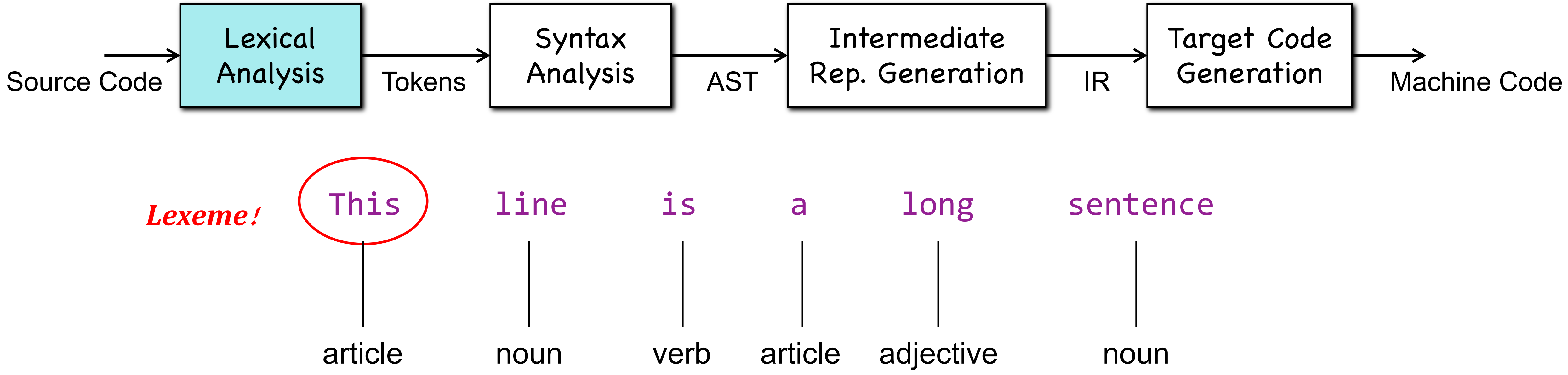
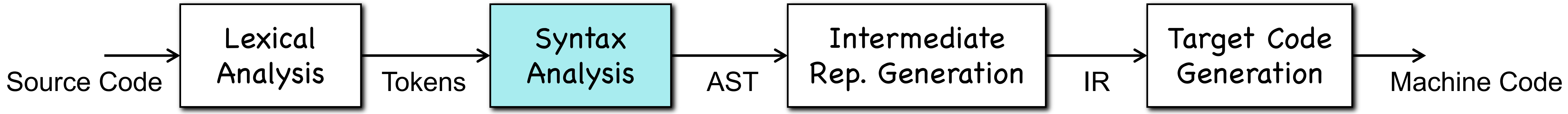
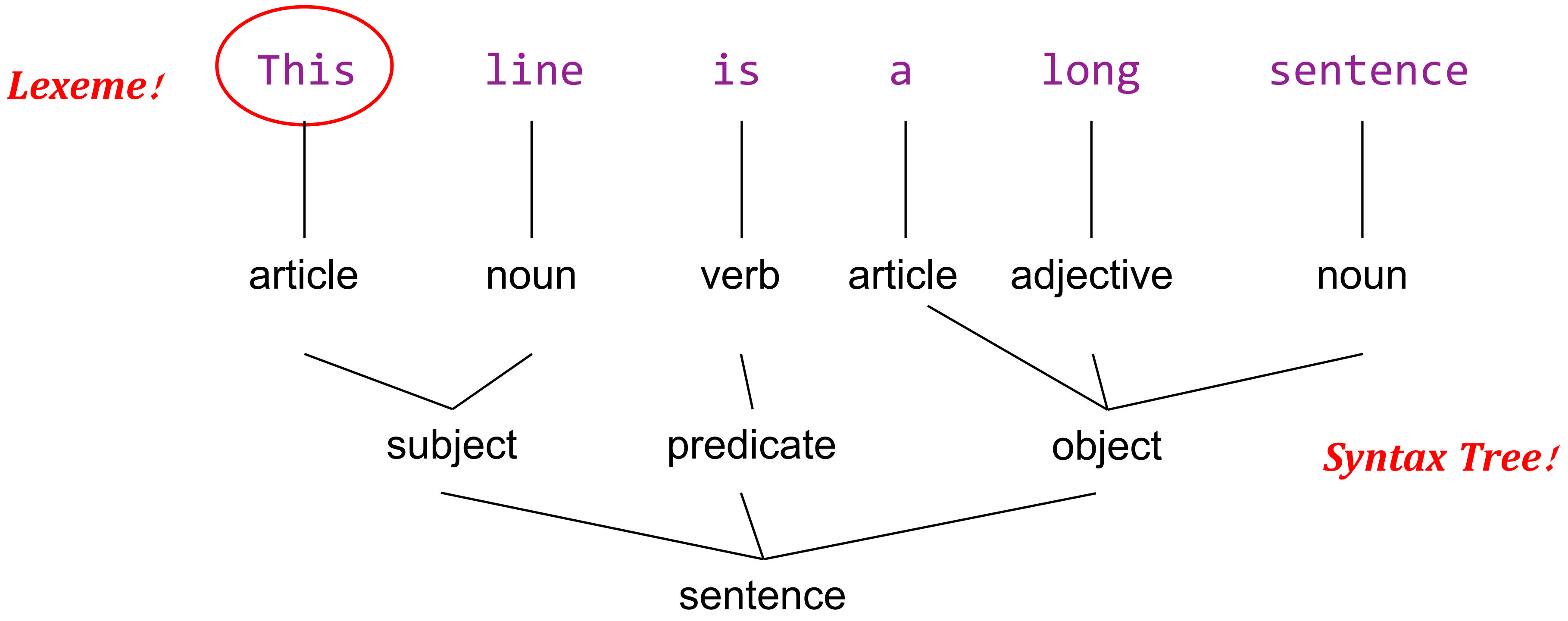
Chapter 4-1
Context-Free Language

Natural Language Perspective

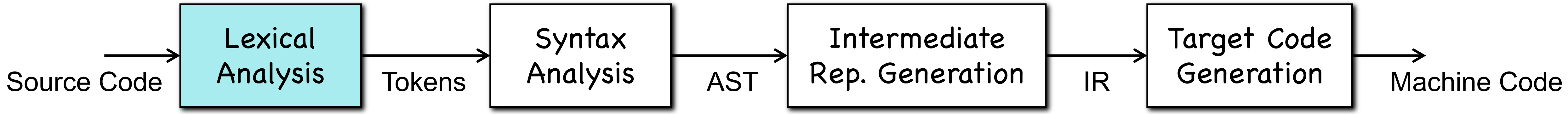


Natural Language Perspective



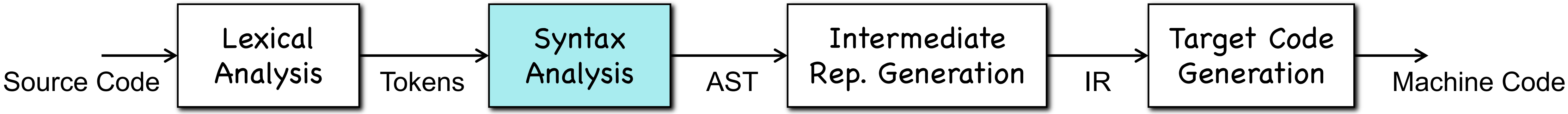


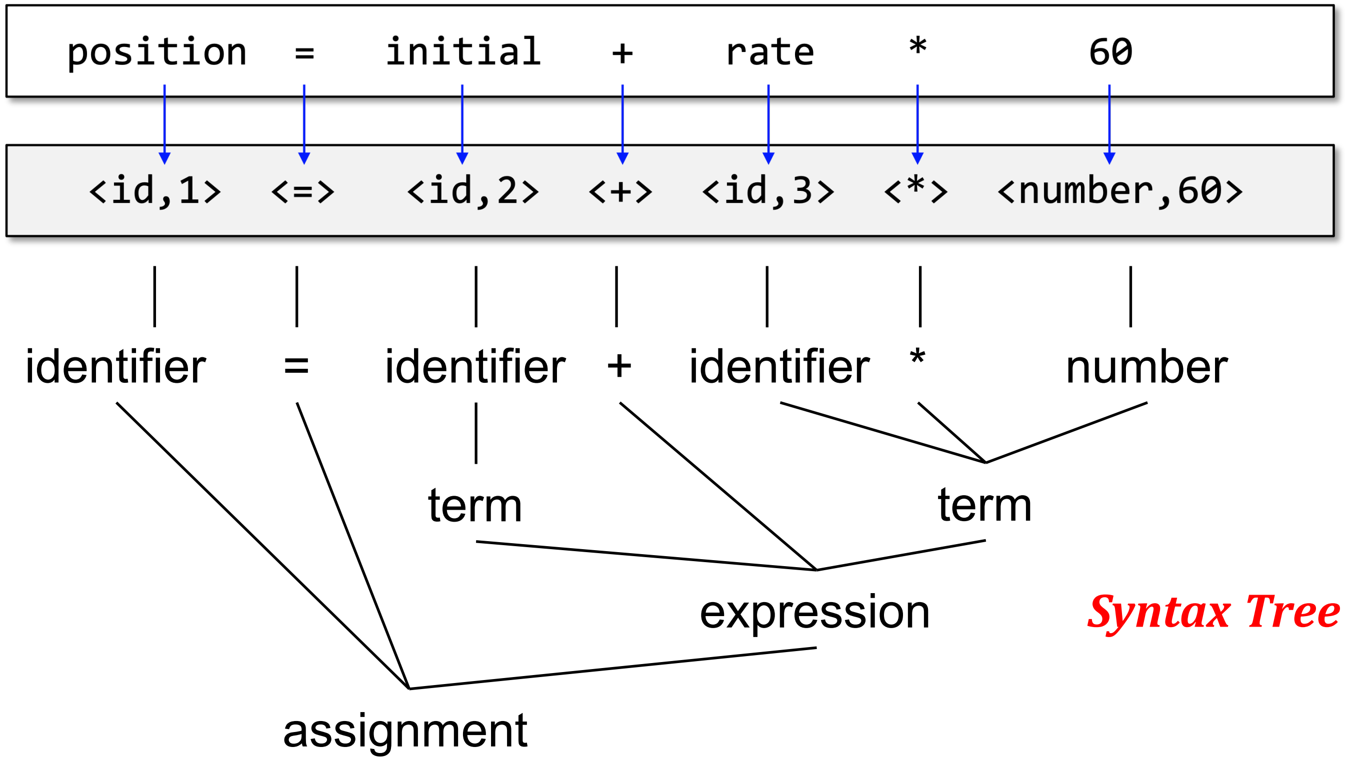
Programming Lang Perspective




```
position   =   initial   +   rate   *   60
<id,1>   <=>   <id,2>   <+>   <id,3>   <*>   <number,60>
```

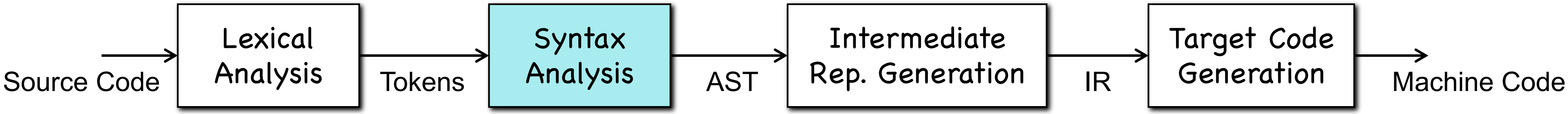
Programming Lang Perspective

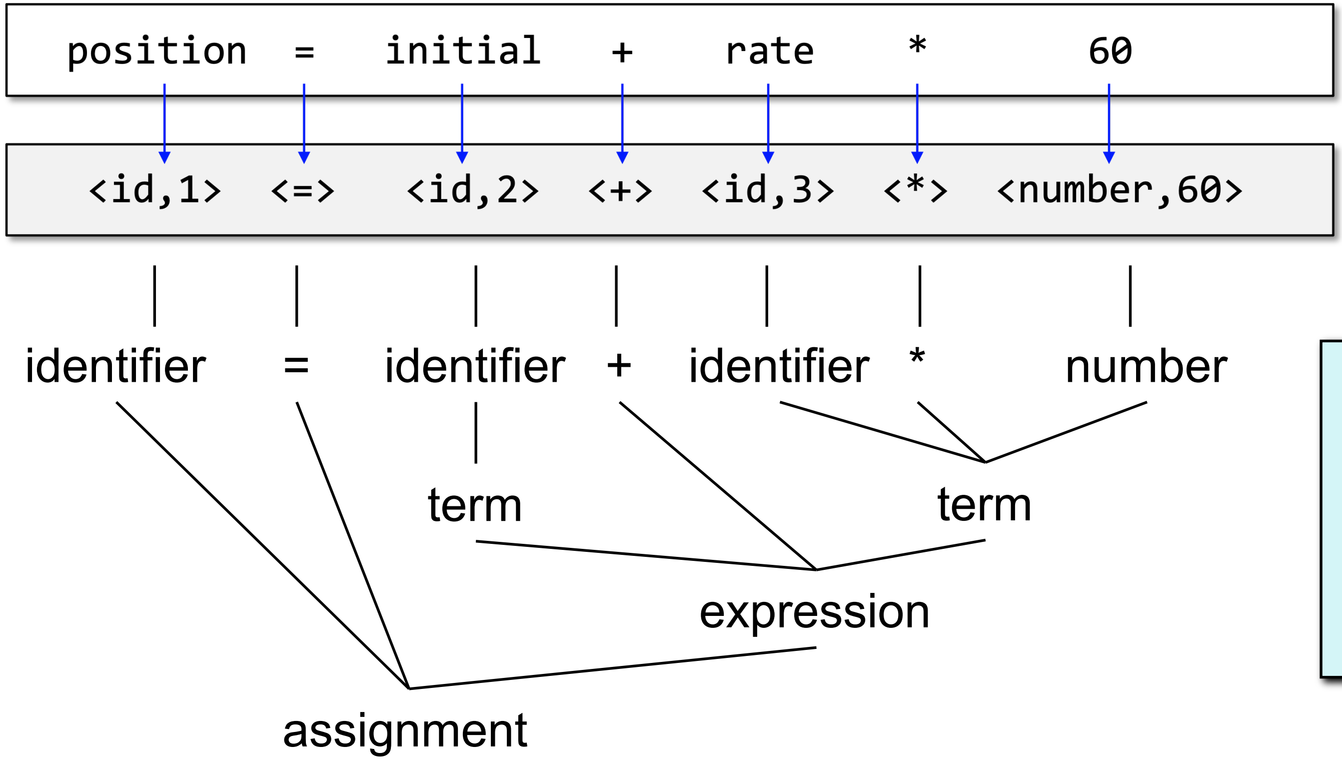




Syntax Tree!

Programming Lang Perspective





assignment → identifier = expression

expression → term + term

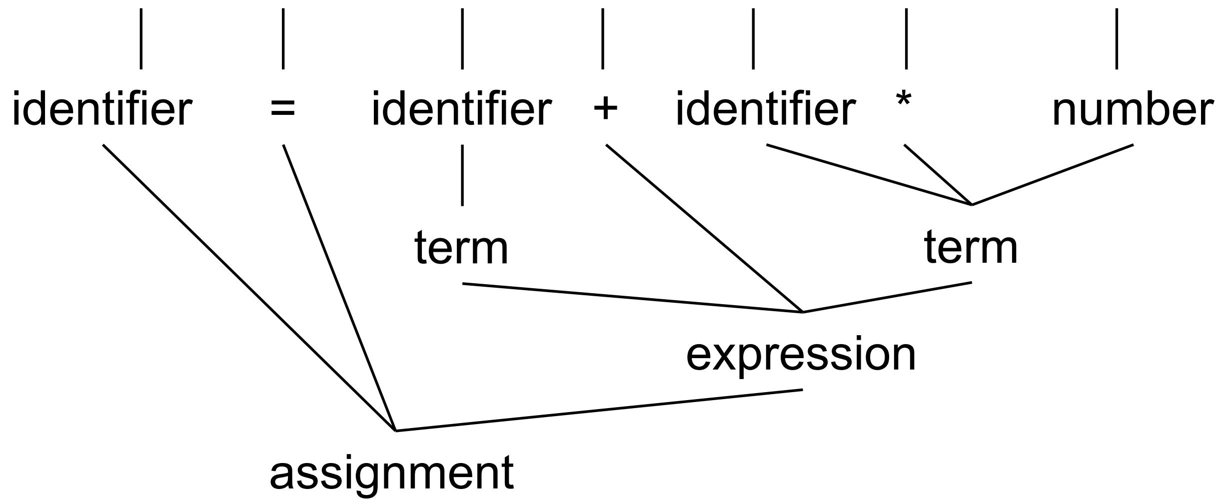
term → identifier

| identifier * number

Programming Lang Perspective

Source Code → Lexical Analysis → Tokens → Syntax Analysis → AST → Intermediate Rep. Generation → IR → Target Code Generation → Machine Code

```
position  =  initial  +  rate  *  60
<id,1> <=> <id,2> <+> <id,3> <*> <number,60>
```

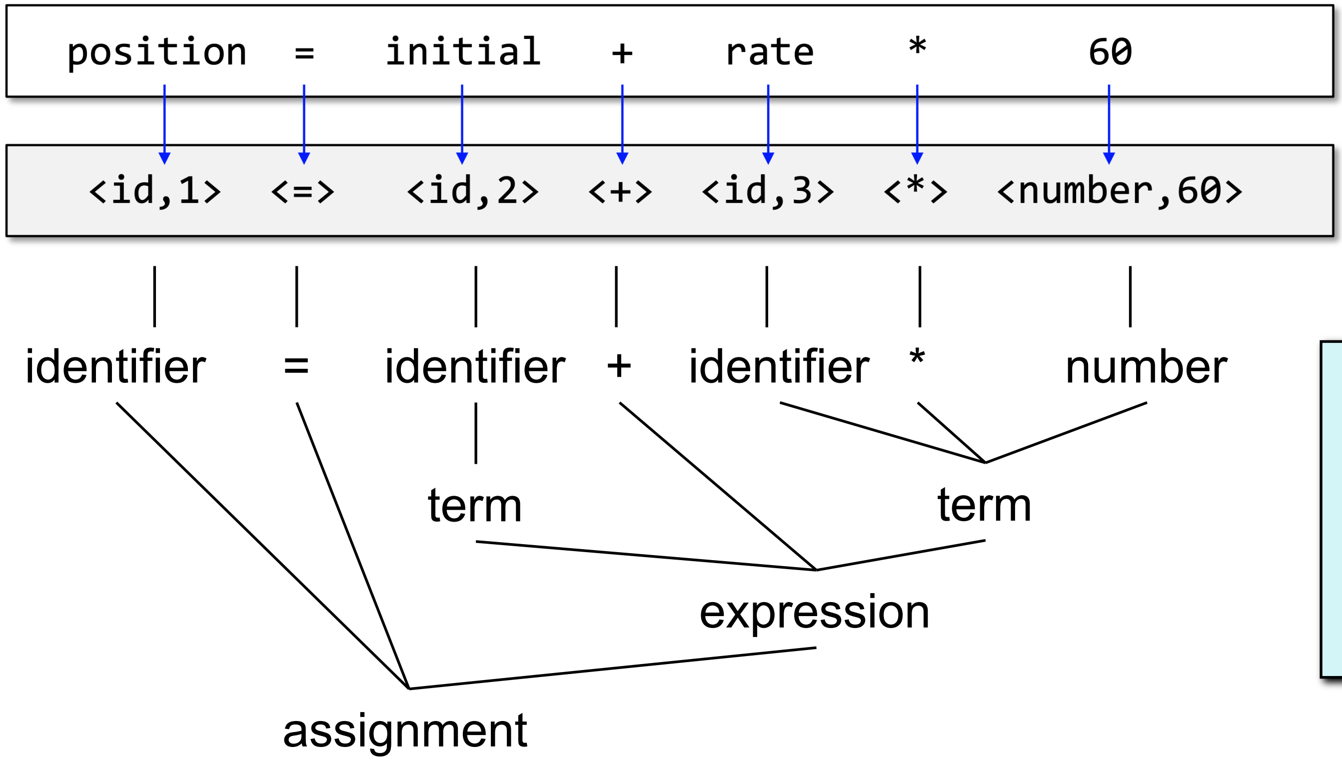


Context-Free Grammar!

assignment → identifier = expression
expression → term + term
term → identifier
| identifier * number

Programming Lang Perspective

Source Code → Lexical Analysis → Tokens → Syntax Analysis → AST → Intermediate Rep. Generation → IR → Target Code Generation → Machine Code



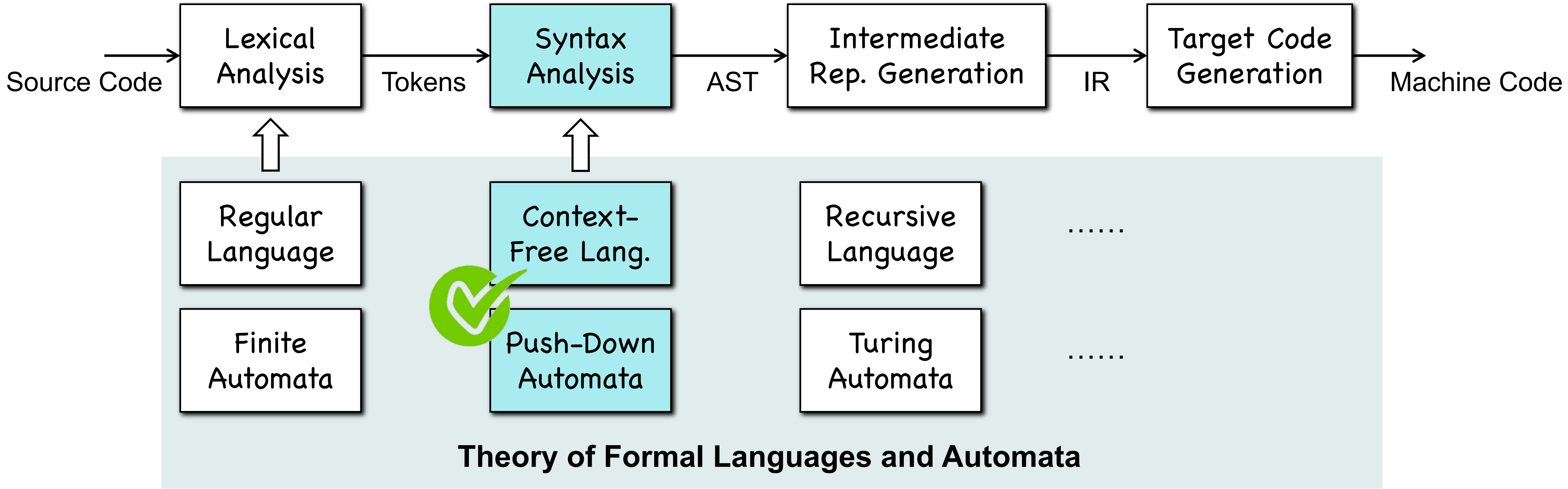
Context-Free Grammar!
Context-Free Language!

assignment → identifier = expression

expression → term + term

term → identifier
| identifier * number

Syntax Analysis



PART I: Context-Free Language

Context Free Grammar (CFG)

- A context-free grammar is a tuple $G = (N, T, S, P)$
  - N: a finite set of non-terminals
  - T: a finite set of terminals, such that $N \cap T = \emptyset$

assignment → identifier = expression
expression → term + term
term → identifier
term → identifier * number

Context Free Grammar (CFG)

- A context-free grammar is a tuple $G = (N, T, S, P)$
  - N: a finite set of non-terminals
  - T: a finite set of terminals, such that $N \cap T = \emptyset$
  - $S \in N$: start non-terminals

assignment → identifier = expression
expression → term + term
term → identifier
term → identifier * number

Context Free Grammar (CFG)

- A context-free grammar is a tuple $G = (N, T, S, P)$
  - N: a finite set of non-terminals
  - T: a finite set of terminals, such that $N \cap T = \emptyset$
  - $S \in N$: start non-terminals
  - P: production rules in the form of $A \rightarrow a$, where $A \in N$ and $a \in (N \cup T)^*$

assignment → identifier = expression
expression → term + term
term → identifier
term → identifier * number

Context Free Grammar (CFG)

- A context-free grammar is a tuple $G = (N, T, S, P)$
  - N: a finite set of non-terminals
  - T: a finite set of terminals, such that $N \cap T = \emptyset$
  - $S \in N$: start non-terminals
  - P: production rules in the form of $A \rightarrow a$, where $A \in N$ and $a \in (N \cup T)^*$

```
assignment → identifier = expression
expression → term + term
      term → identifier
      term → identifier * number
```

```
assignment → identifier = expression
expression → term + term
      term → identifier
           | identifier * number
```

Derivation ($\Rightarrow_G$)

- Assume: $A \rightarrow \gamma$
- Derivation: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$

Derivation ($\Rightarrow_G$)

- Assume: $A \to \gamma$
- Derivation: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$, written as $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if G is clear

Derivation ($\Rightarrow_G$)

- Assume: $A \to \gamma$
- Derivation: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$, written as $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if G is clear
- Consecutive derivation: $\Rightarrow^*$

Derivation ($\Rightarrow_G$)

- Assume: $A \rightarrow \gamma$
- Derivation: $\alpha A\beta \Rightarrow_G \alpha\gamma\beta$, written as $\alpha A\beta \Rightarrow \alpha\gamma\beta$ if G is clear
- Consecutive derivation: $\Rightarrow^*$

- **Example**: $S \rightarrow aSb \mid \epsilon$
  - $S \Rightarrow^* aaabbb$

Derivation ($\Rightarrow_G$)

- Assume: $A \to \gamma$
- Derivation: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$, written as $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if G is clear
- Consecutive derivation: $\Rightarrow^*$

- **Example**: $S \to aSb \mid \epsilon$
  - $S \Rightarrow^* aaabbb$
  - $\underbrace{S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb}_{S \to aSb} \underbrace{\Rightarrow aaabbb}_{S \to \epsilon}$

Derivation ($\Rightarrow_G$)

- Assume: $A \to \gamma$
- Derivation: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$, written as $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if G is clear
- Consecutive derivation: $\Rightarrow^*$

$a^n b^n$ (not a regular language)

- **Example**: $S \to aSb \mid \epsilon$
  - $S \Rightarrow^* aaabbb$
  - $\underbrace{S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb}_{S \to aSb} \underbrace{\Rightarrow aaabbb}_{S \to \epsilon}$

Derivation ($\Rightarrow_G$)

- Assume: $A \to \gamma$
- Derivation: $\alpha A \beta \Rightarrow_G \alpha \gamma \beta$, written as $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if G is clear
- Consecutive derivation: $\Rightarrow^*$

- **Example**: $S \to aSb \mid \epsilon$
  - $S \Rightarrow^* aaabbb$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

$a^n b^n$ (not a regular language)

- **Context-Free Language**: $L(G) = \{w \in T^* \colon S \Rightarrow^* w\}$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal

- Left-most derivation for $v * (v + d)$

$$\boxed{\begin{aligned} &E \rightarrow EOE \\ &E \rightarrow (E) \\ &E \rightarrow v \mid d \\ &O \rightarrow + \mid \ * \end{aligned}}$$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal
- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE$$

$E \rightarrow EOE$
$E \rightarrow (E)$
$E \rightarrow v \mid d$
$O \rightarrow + \mid *$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal
- Left-most derivation for $v * (v + d)$

$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE$

$E \to EOE$
$E \to (E)$
$E \to v \mid d$
$O \to + \mid *$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal
- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E$$

$$\boxed{\begin{array}{l} E \rightarrow EOE \\ E \rightarrow (E) \\ E \rightarrow v \mid d \\ O \rightarrow + \mid * \end{array}}$$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal
- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E)$$

$$\boxed{\begin{array}{l} E \rightarrow EOE \\ E \rightarrow (E) \\ E \rightarrow v \mid d \\ O \rightarrow + \mid \ * \end{array}}$$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal

- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E) \Rightarrow_{lm} v * (EOE)$$

$E \rightarrow EOE$
$E \rightarrow (E)$
$E \rightarrow v \mid d$
$O \rightarrow + \mid *$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal

- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E) \Rightarrow_{lm} v * (EOE)$$
$$\Rightarrow_{lm} v * (vOE)$$

$$E \rightarrow EOE$$
$$E \rightarrow (E)$$
$$E \rightarrow v \mid d$$
$$O \rightarrow + \mid *$$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal

- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E) \Rightarrow_{lm} v * (EOE)$$
$$\Rightarrow_{lm} v * (vOE) \Rightarrow_{lm} v * (v + E) \Rightarrow_{lm} v * (v + d)$$

$$\boxed{\begin{array}{l} E \rightarrow EOE \\ E \rightarrow (E) \\ E \rightarrow v \mid d \\ O \rightarrow + \mid * \end{array}}$$

Left-Most Derivation ($\Rightarrow_{lm}$)

- In every step derivation, we replace the left-most non-terminal

- Left-most derivation for $v * (v + d)$

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E) \Rightarrow_{lm} v * (EOE)$$
$$\Rightarrow_{lm} v * (vOE) \Rightarrow_{lm} v * (v + E) \Rightarrow_{lm} v * (v + d)$$

$$\boxed{\begin{array}{l} E \to EOE \\ E \to (E) \\ E \to v \mid d \\ O \to + \mid * \end{array}}$$

- Written as $E \Rightarrow_{lm}^{*} v * (v + d)$

Right-Most Derivation ($\Rightarrow_{rm}$)

- In every step derivation, we replace the right-most non-terminal

Right-Most Derivation ($\Rightarrow_{rm}$)

- In every step derivation, we replace the right-most non-terminal
- Right-most derivation for $v * (v + d)$

$$
\boxed{\begin{array}{l} E \to EOE \\ E \to (E) \\ E \to v \mid d \\ O \to + \mid * \end{array}}
$$

Right-Most Derivation ($\Rightarrow_{rm}$)

- In every step derivation, we replace the right-most non-terminal

- Right-most derivation for $v * (v + d)$

$$E \Rightarrow_{rm} EOE \Rightarrow_{rm} EO(E) \Rightarrow_{rm} EO(EOE) \Rightarrow_{rm} EO(EOd) \Rightarrow_{rm} EO(E + d)$$
$$\Rightarrow_{rm} EO(v + d) \Rightarrow_{rm} E * (v + d) \Rightarrow_{rm} v * (v + d)$$

$$E \to EOE$$
$$E \to (E)$$
$$E \to v \mid d$$
$$O \to + \mid *$$

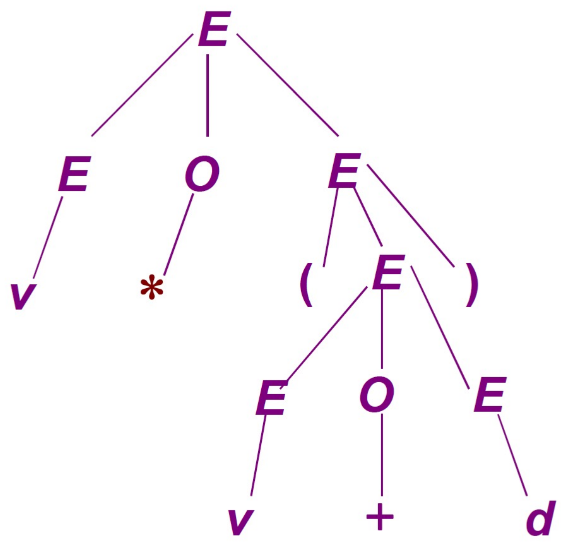
- Written as $E \Rightarrow_{rm}^{*} v * (v + d)$

Parse/Syntax Trees

- Derivation is a procedure of building a parse tree

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E) \Rightarrow_{lm} v * (EOE)$$
$$\Rightarrow_{lm} v * (vOE) \Rightarrow_{lm} v * (v + E) \Rightarrow_{lm} v * (v + d)$$

$E \to EOE$
$E \to (E)$
$E \to v \mid d$
$O \to + \mid *$

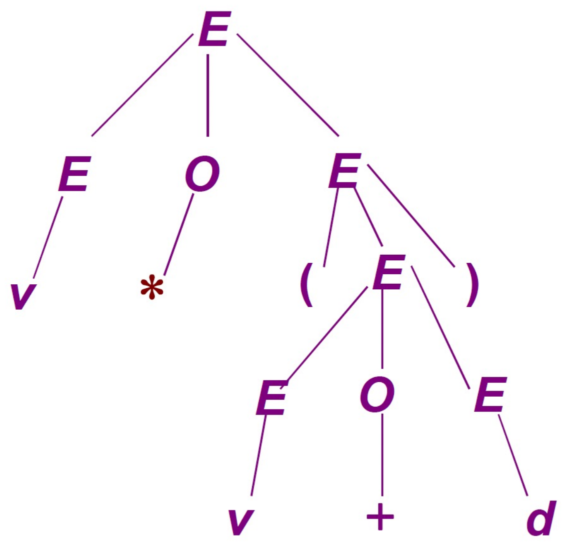


Parse/Syntax Trees

- Derivation is a procedure of building a parse tree

$$E \Rightarrow_{lm} EOE \Rightarrow_{lm} vOE \Rightarrow_{lm} v * E \Rightarrow_{lm} v * (E) \Rightarrow_{lm} v * (EOE)$$
$$\Rightarrow_{lm} v * (vOE) \Rightarrow_{lm} v * (v + E) \Rightarrow_{lm} v * (v + d)$$

$$\begin{aligned} &E \to EOE \\ &E \to (E) \\ &E \to v \mid d \\ &O \to + \mid * \end{aligned}$$



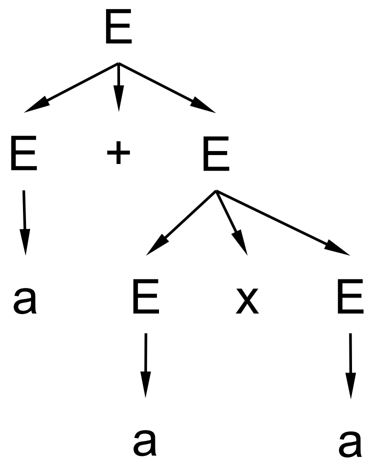
- Parse tree:
  - Every leaf is a terminal
  - Every non-leaf node is a non-terminal

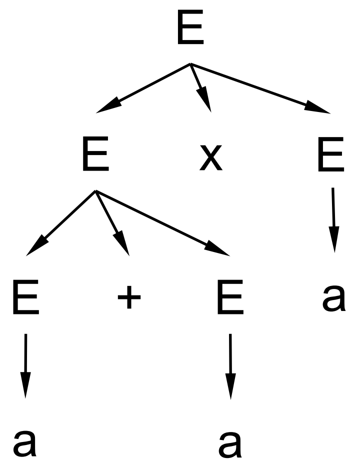
Ambiguity

- $E \rightarrow E + E \mid E \times E \mid a$
- $w = a + a \times a$

$$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$

$$E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$



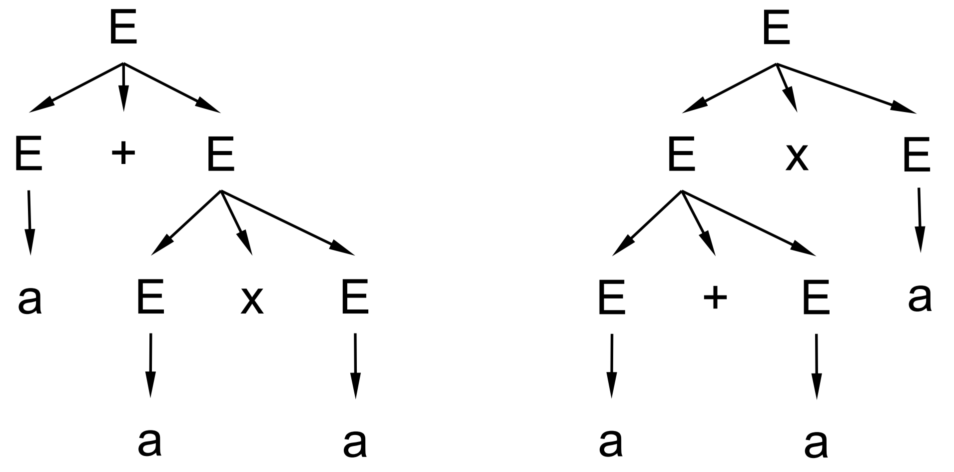


Ambiguity

- $E \to E + E \mid E \times E \mid a$
- $w = a + a \times a$

$$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$

$$E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$



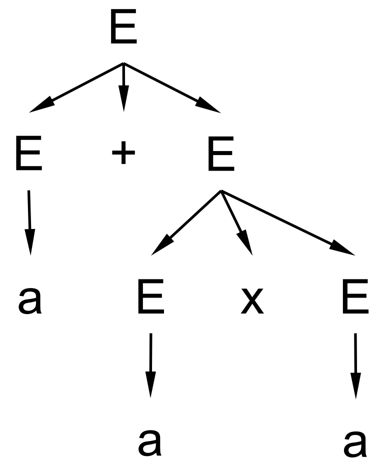
- A grammar is ambiguous if there is a string has two different derivation trees

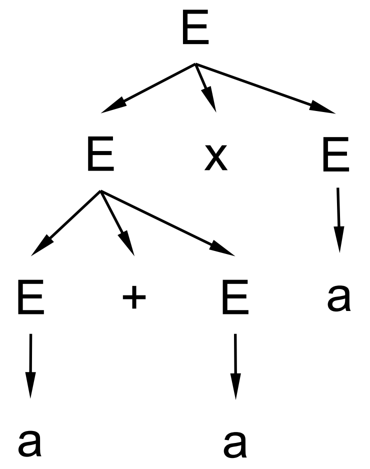
Ambiguity

- $E \rightarrow E + E \mid E \times E \mid a$
- $w = a + a \times a$

$$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$

$$E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$





- A grammar is ambiguous if there is a string has two different derivation trees
- A grammar is not ambiguous if for any string, it has only one derivation tree

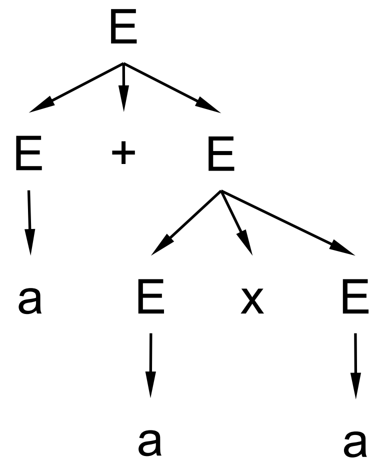
Ambiguity

- $E \rightarrow E + E \mid E \times E \mid a$
- $w = a + a \times a$

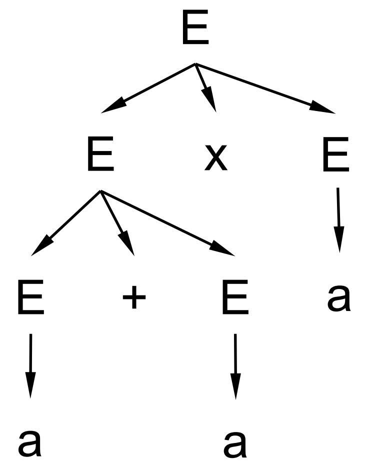
$$E \Rightarrow E + E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$

$$E \Rightarrow E \times E \Rightarrow E + E \times E \Rightarrow \cdots \Rightarrow w$$

2 + 2 x 2 = 6



2 + 2 x 2 = 8



- A grammar is ambiguous if there is a string has two different derivation trees
- A grammar is not ambiguous if for any string, it has only one derivation tree

Inherent Ambiguity

- Some context-free languages have only ambiguous grammars
- We cannot eliminate the ambiguity

Inherent Ambiguity

- Some context-free languages have only ambiguous grammars
- We cannot eliminate the ambiguity

- There is not a general technique to eliminate the ambiguity!

Inherent Ambiguity

- Some context-free languages have only ambiguous grammars
- We cannot eliminate the ambiguity

- There is not a general technique to eliminate the ambiguity!
- There is not an algorithm that can decide if a grammar is ambiguous (This is an undecidable problem!)

Eliminating Ambiguity

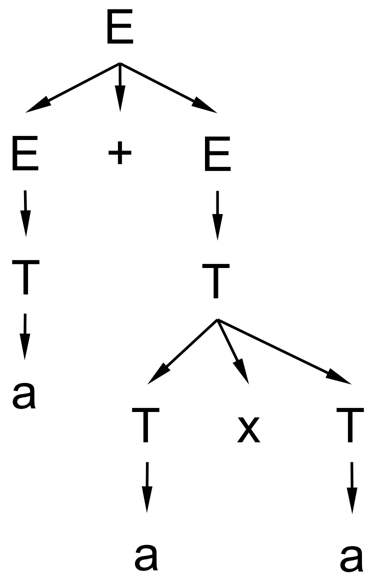
- Lifting operators with higher priority
- $E \rightarrow E + E \mid E \times E \mid a$

Eliminating Ambiguity

- Lifting operators with higher priority
- $E \rightarrow E + E \mid E \times E \mid a$
- ----------------
- $E \rightarrow E + E \mid T$
- $T \rightarrow T \times T \mid a$

Eliminating Ambiguity

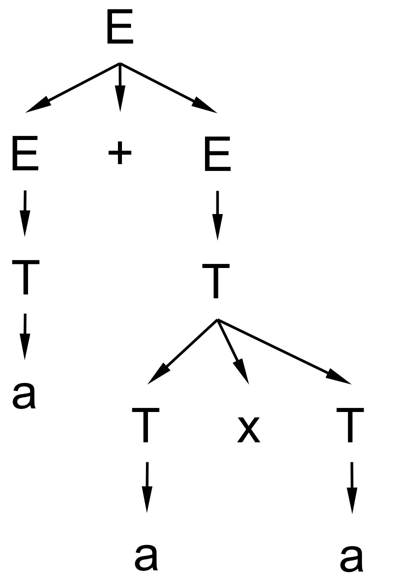
- Lifting operators with higher priority
- $E \rightarrow E + E \mid E \times E \mid a$
- ----------------
- $E \rightarrow E + E \mid T$
- $T \rightarrow T \times T \mid a$
- ----------------
- $w = a + a \times a$

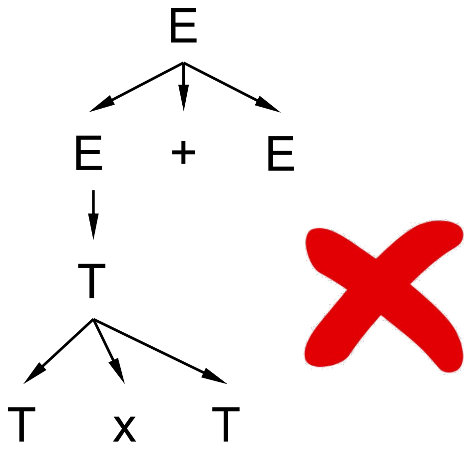


$E \Rightarrow E + E \Rightarrow E + T \times T \Rightarrow T + T \times T \Rightarrow a + a \times a$

Eliminating Ambiguity

- Lifting operators with higher priority
- $E \rightarrow E + E \mid E \times E \mid a$
- ---------------
- $E \rightarrow E + E \mid T$
- $T \rightarrow T \times T \mid a$
- ---------------
- $w = a + a \times a$





$E \Rightarrow E + E \Rightarrow E + T \times T \Rightarrow T + T \times T \Rightarrow a + a \times a$

$E \Rightarrow E + E \Rightarrow T \times T + E \Rightarrow \cdots$ (does not work)

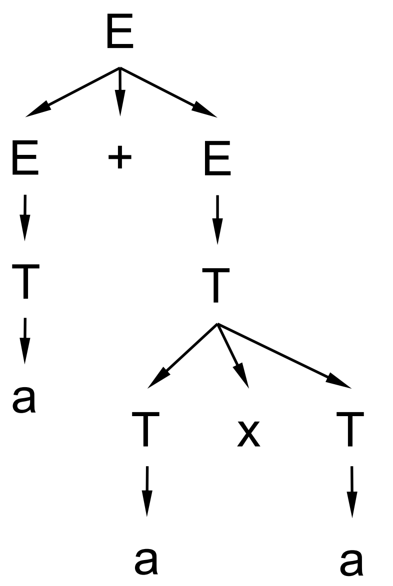
Eliminating Ambiguity

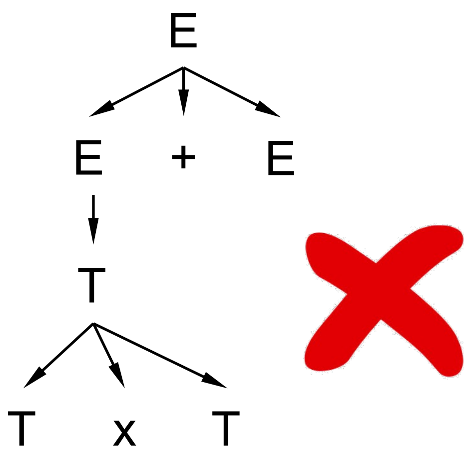
- Lifting operators with higher priority
- $E \rightarrow E + E \mid E \times E \mid a$
- ---------------
- $E \rightarrow E + E \mid T$
- $T \rightarrow T \times T \mid a$
- ---------------
- $w = a + a \times a$

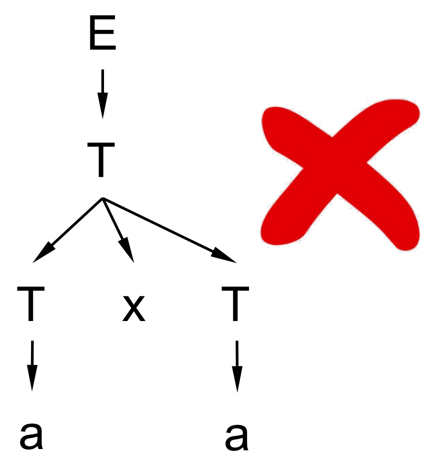
$E \Rightarrow E + E \Rightarrow E + T \times T \Rightarrow T + T \times T \Rightarrow a + a \times a$

$E \Rightarrow E + E \Rightarrow T \times T + E \Rightarrow \cdots$ (does not work)

$E \Rightarrow T \Rightarrow T \times T \Rightarrow \cdots$ (does not work)

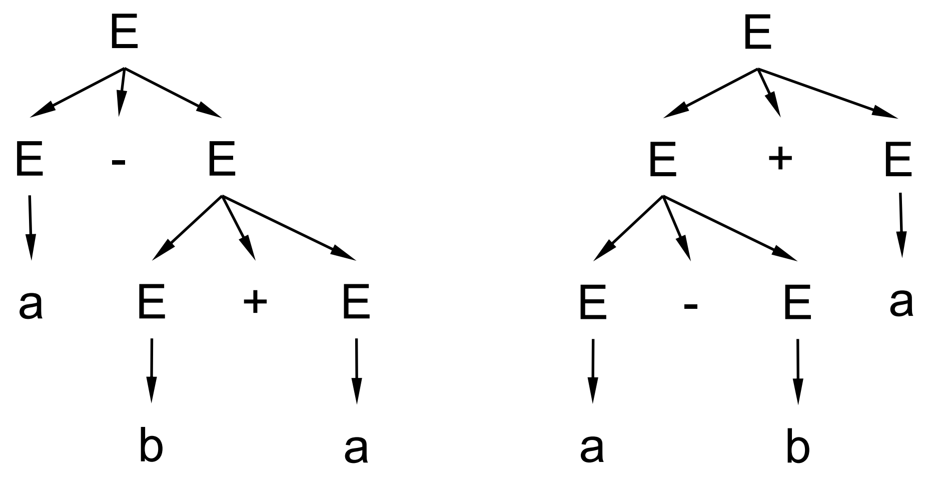






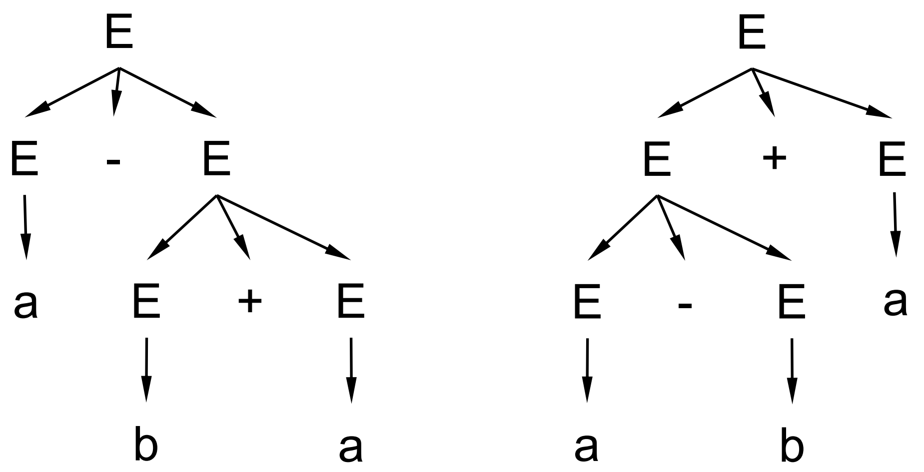
Eliminating Ambiguity

- $E \to E + E \mid E - E \mid T;$
- $T \to T \times T \mid a \mid b$
- ----------------
- $w = a - b + a$



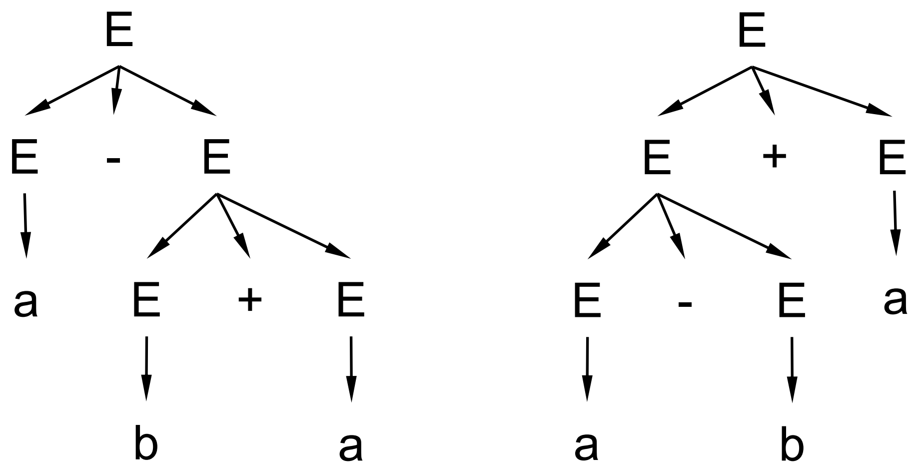
Eliminating Ambiguity

- $E \to E + E \mid E - E \mid T;$
- $T \to T \times T \mid a \mid b$
- ----------------
- $w = a - b + a$
- -----------------
- Left associativity for most operators of the same priority



Eliminating Ambiguity

- $E \rightarrow E + E \mid E - E \mid T;$
- $T \rightarrow T \times T \mid a \mid b$
- ----------------
- $w = a - b + a$
- -----------------
- Left associativity for most operators of the same priority
- -----------------
- $E \rightarrow E + T \mid E - T \mid T; \quad T \rightarrow T \times T \mid a \mid b$



Eliminating Ambiguity

$$
\begin{array}{lcl}
\mathit{stmt} & \rightarrow & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \\
& | & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \textbf{ else } \mathit{stmt} \\
& | & \textbf{other}
\end{array}
$$

Eliminating Ambiguity

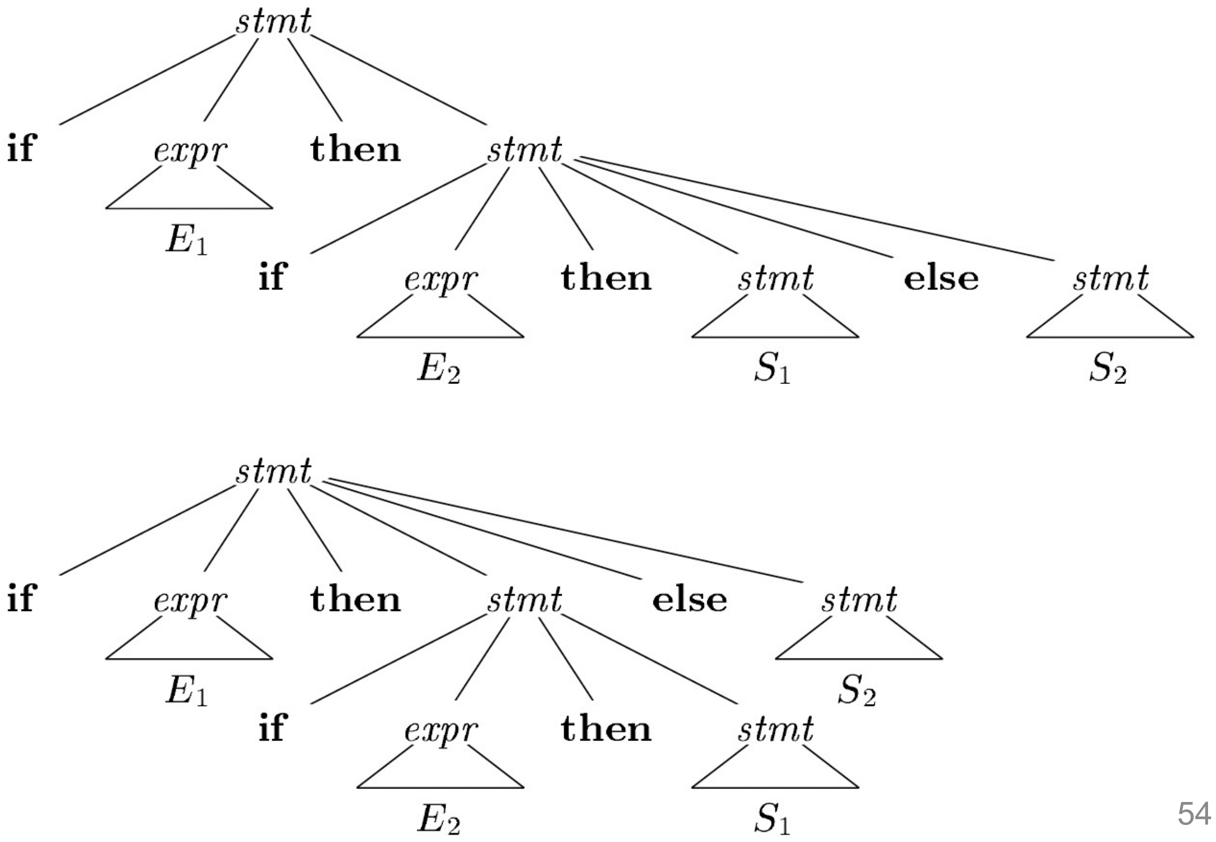
$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{other}
\end{array}
$$

if E_1 **then if** E_2 **then** S_1 **else** S_2

Eliminating Ambiguity

$$
\begin{array}{lcl}
\mathit{stmt} & \rightarrow & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \\
 & | & \textbf{if } \mathit{expr} \textbf{ then } \mathit{stmt} \textbf{ else } \mathit{stmt} \\
 & | & \textbf{other}
\end{array}
$$

if E_1 **then** **if** E_2 **then** S_1 **else** S_2

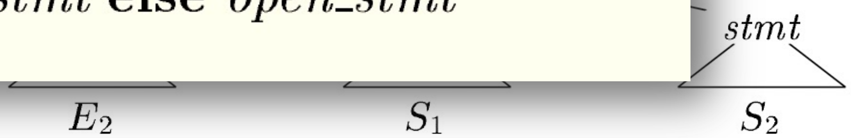


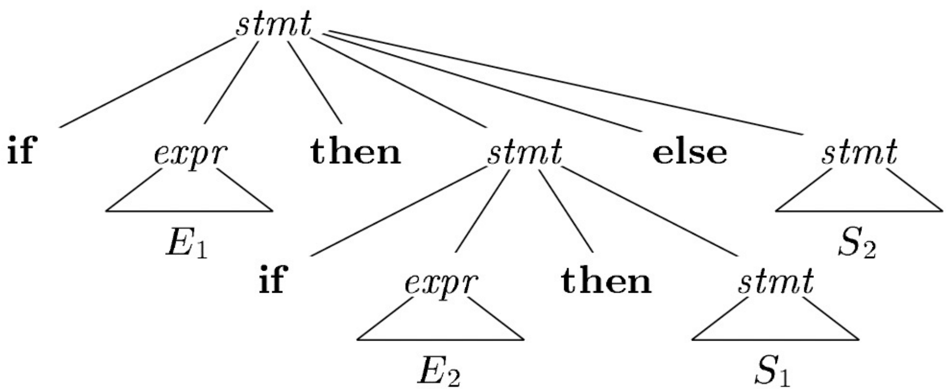
Eliminating Ambiguity

stmt →

$$
\begin{array}{rcl}
stmt & \rightarrow & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ matched\_stmt\ \textbf{else}\ matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \rightarrow & \textbf{if}\ expr\ \textbf{then}\ stmt \\
 & | & \textbf{if}\ expr\ \textbf{then}\ matched\_stmt\ \textbf{else}\ open\_stmt
\end{array}
$$

if E_1 **th**



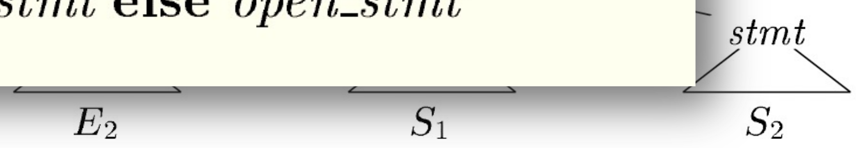


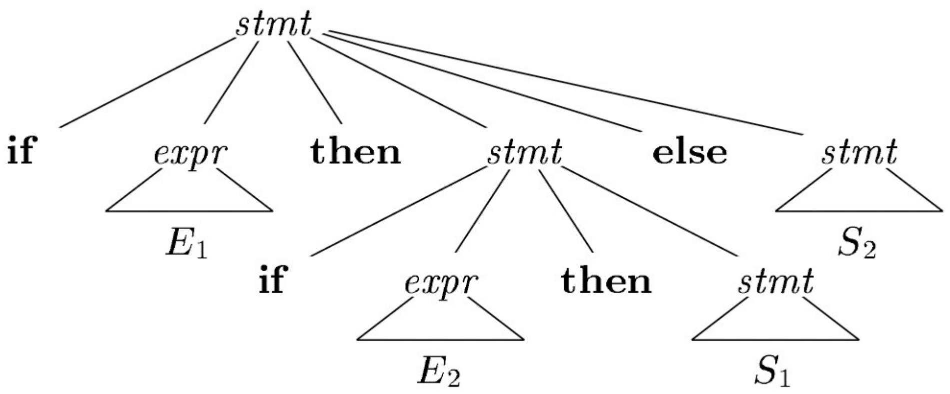
Eliminating Ambiguity

$stmt$ →

if E_1 the

$$
\begin{array}{rcl}
stmt & \rightarrow & matched\_stmt \\
 & | & open\_stmt \\
matched\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } matched\_stmt \textbf{ else } matched\_stmt \\
 & | & \textbf{other} \\
open\_stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \\
 & | & \textbf{if } expr \textbf{ then } matched\_stmt \textbf{ else } open\_stmt
\end{array}
$$





Using Ambiguous Grammar

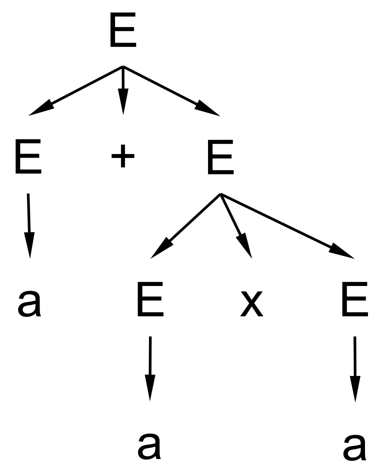
- Given that we cannot always eliminate ambiguity, just use it!
- We will discuss how we use ambiguous grammar in the next lecture when introducing specific parsing techniques

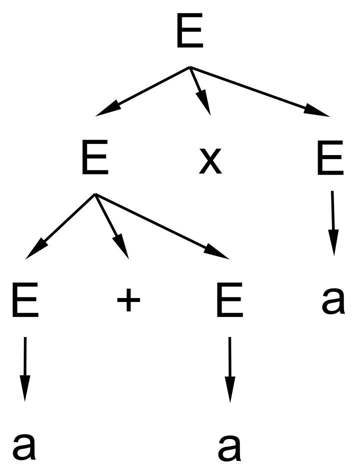
Using Ambiguous Grammar

- Given that we cannot always eliminate ambiguity, just use it!

- We will discuss how we use ambiguous grammar in the next lecture when introducing specific parsing techniques, e.g.,

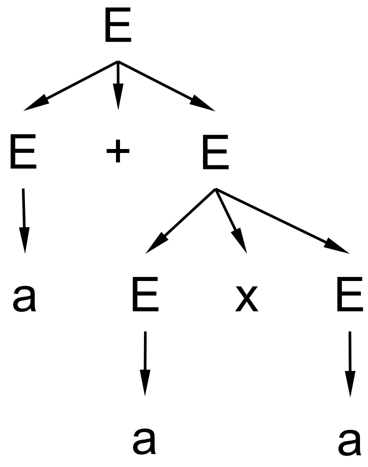
- $E \rightarrow E + E \mid E \times E \mid a$
- $w = a + a \times a$

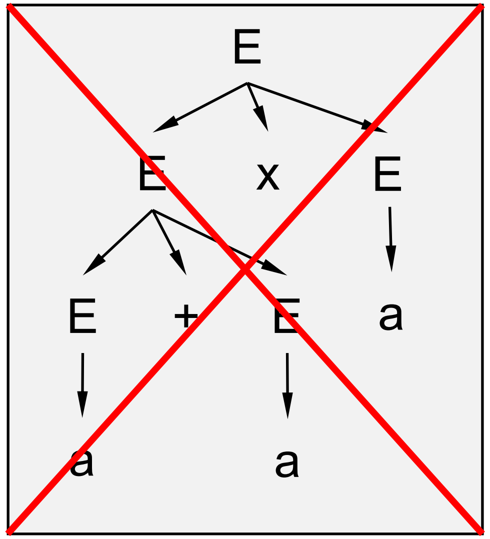




Using Ambiguous Grammar

- Given that we cannot always eliminate ambiguity, just use it!
- We will discuss how we use ambiguous grammar in the next lecture when introducing specific parsing techniques
- $E \rightarrow E + E \mid E \times E \mid a$
- $w = a + a \times a$

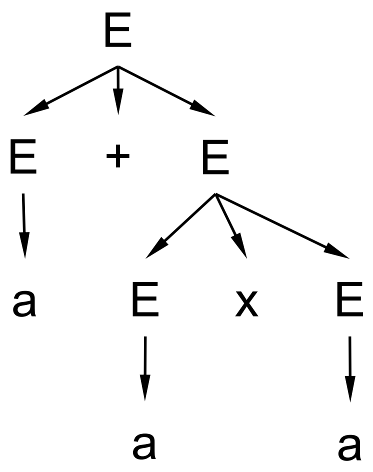


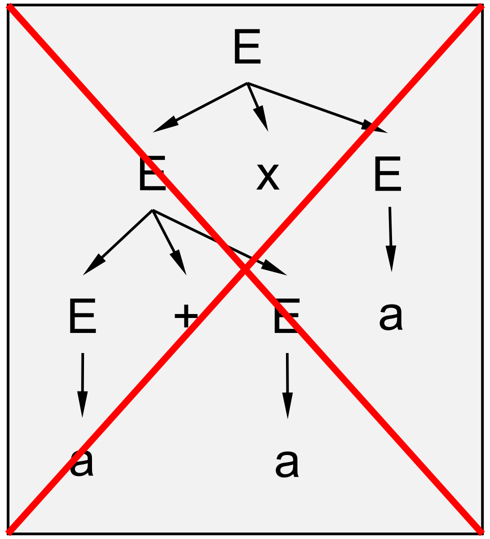


Using Ambiguous Grammar

- Given that we cannot always eliminate ambiguity, just use it!
- We will discuss how we use ambiguous grammar in the next lecture when introducing specific parsing techniques
- $E \rightarrow E + E \mid E \times E \mid a$
- $w = a + a \times a$

Let us always try production rules in order!





Exercises

$S \to aSb \mid \epsilon$ is the grammar for $a^n b^n$

- Write a context-free grammar for the following languages
  - 0^*1^*

Exercises

$S \to aSb \mid \epsilon$ is the grammar for $a^n b^n$

- Write a context-free grammar for the following languages
  - $0^n 1^{2n}$

Exercises

$S \to aSb \mid \epsilon$ is the grammar for $a^n b^n$

- Write a context-free grammar for the following languages
  - L = { w | w is a string of balanced parentheses }

Exercises

$S \to aSb \mid \epsilon$ is the grammar for $a^n b^n$

- Write a context-free grammar for the following languages
  - $a^i b^j c^k$ where $i = j$ or $j = k$

Exercises

$S \to aSb \mid \epsilon$ is the grammar for $a^n b^n$

- Write a context-free grammar for the following languages
  - $a^i b^j c^k$ where $i \neq j$ or $j \neq k$

Exercises

- Prove all regular languages are context-free languages

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Given two regex: r_1, r_2, the following are regex
- $r_1|r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Primitive regex
-
-
-

Given two regex: r_1, r_2, the following are regex
- $r_1 | r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Given two regex: r_1, r_2, the following are regex
- $r_1 | r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Primitive regex
- $S \rightarrow S$
-
-

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Primitive regex
- $S \to S$
- $S \to \epsilon$
- $S \to a$

Given two regex: r_1, r_2, the following are regex
- $r_1|r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Primitive regex
- $S \to S$
- $S \to \epsilon$
- $S \to a$

Given two regex: r_1, r_2, the following are regex
- $r_1 | r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Given the CFG start symbols S_1, S_2, we have
-
-
-
-

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Primitive regex
- $S \rightarrow S$
- $S \rightarrow \epsilon$
- $S \rightarrow a$

Given two regex: r_1, r_2, the following are regex
- $r_1|r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Given the CFG start symbols S_1, S_2, we have
- $S \rightarrow S_1|S_2$
- $S \rightarrow S_1 S_2$
-
-

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Primitive regex
- $S \rightarrow S$
- $S \rightarrow \epsilon$
- $S \rightarrow a$

Given two regex: r_1, r_2, the following are regex
- $r_1|r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Given the CFG start symbols S_1, S_2, we have
- $S \rightarrow S_1|S_2$
- $S \rightarrow S_1 S_2$
- $S \rightarrow S_1 S|\epsilon$
-

Exercises

- Prove all regular languages are context-free languages

Primitive regex
- $\emptyset$,
- ϵ
- a

Primitive regex
- $S \rightarrow S$
- $S \rightarrow \epsilon$
- $S \rightarrow a$

Given two regex: r_1, r_2, the following are regex
- $r_1|r_2$
- $r_1 r_2$
- r_1^*
- (r_1)

Given the CFG start symbols S_1, S_2, we have
- $S \rightarrow S_1|S_2$
- $S \rightarrow S_1 S_2$
- $S \rightarrow S_1 S|\epsilon$
- $S \rightarrow (S_1)$

PART II: Push-Down Automata

Recap: Context-Free Language

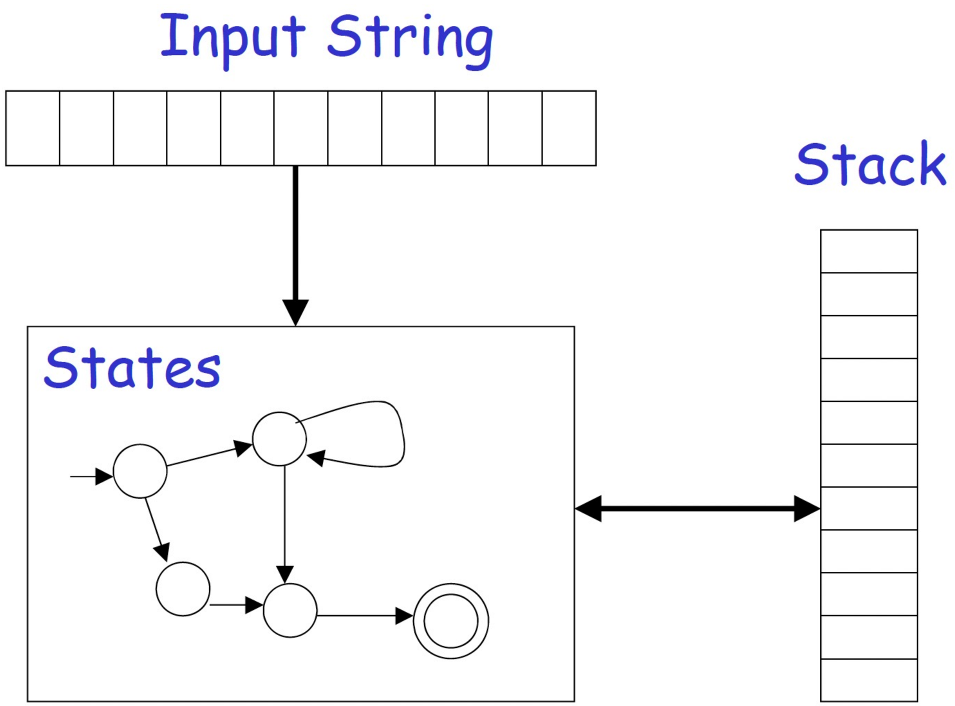
- $L(G) = \{w \in T^*: S \Rightarrow^* w\}$ is a context-free language

Push-Down Automata (PDA)

- Regular language = DFA/NFA
- Context-free language = PDA = NFA + Stack (z_0)

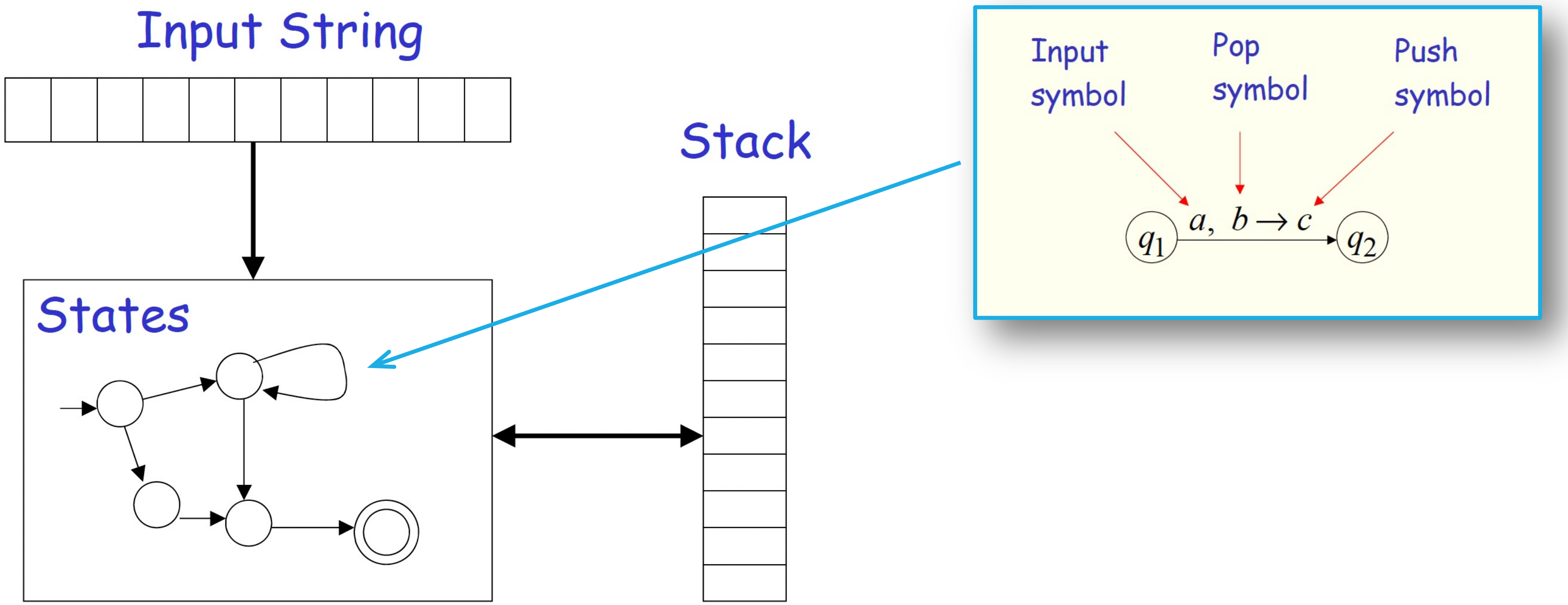
Push-Down Automata (PDA)

- Regular language = DFA/NFA
- Context-free language = PDA = NFA + Stack (z_0)



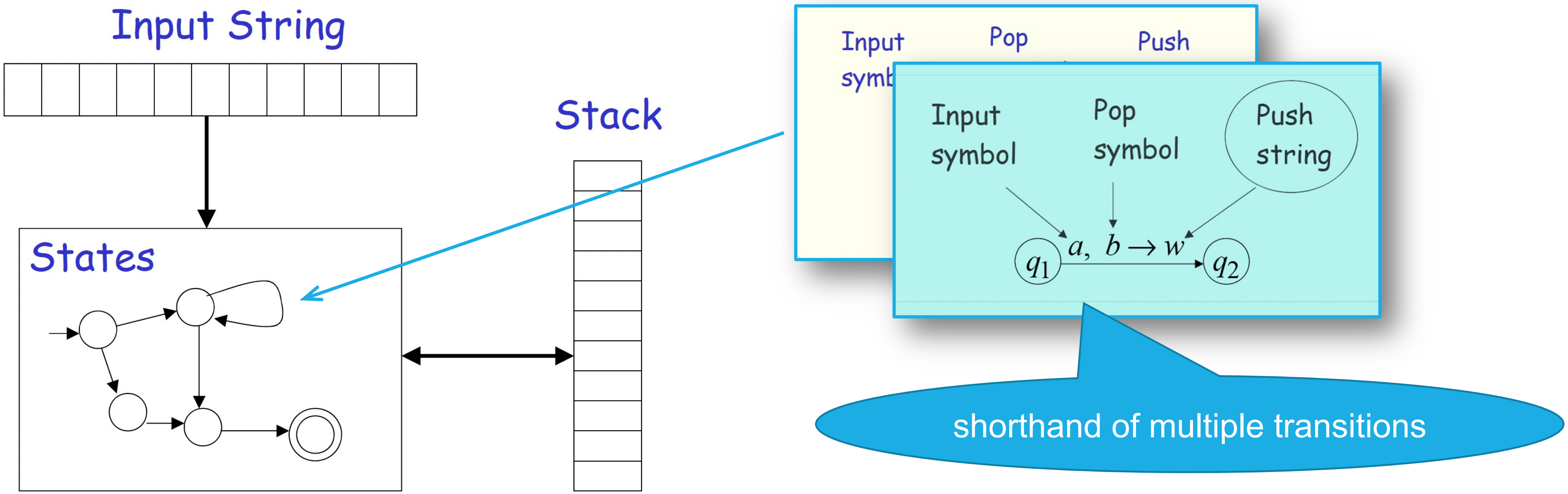
Push-Down Automata (PDA)

- Regular language = DFA/NFA
- Context-free language = PDA = NFA + Stack (z_0)



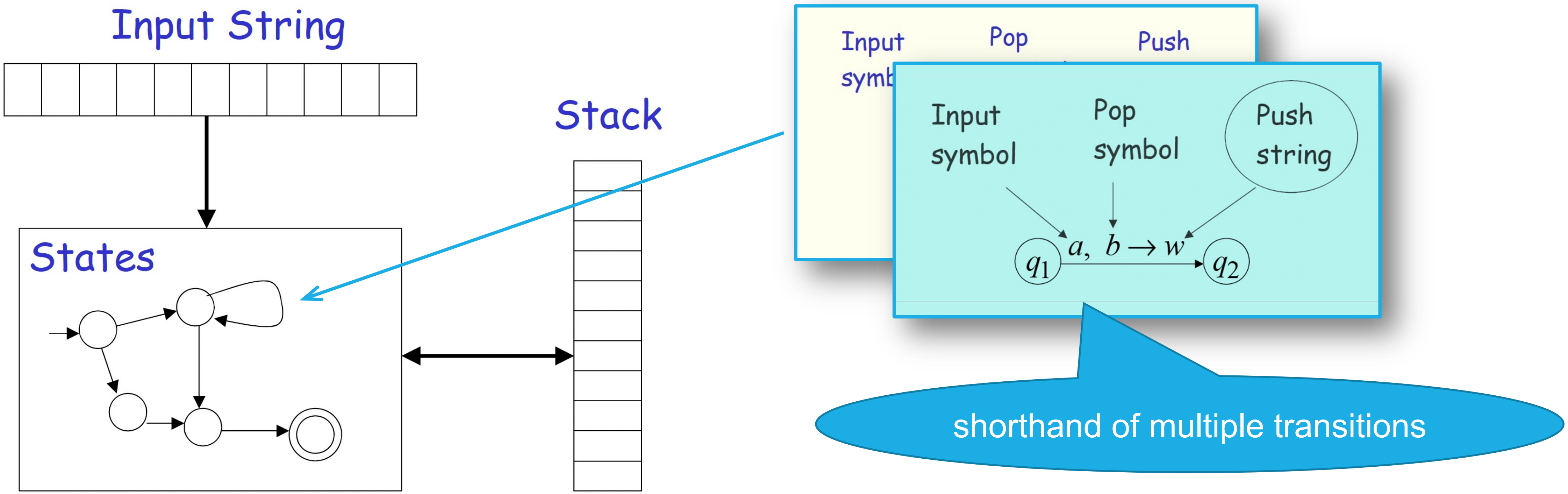
Push-Down Automata (PDA)

- Regular language = DFA/NFA
- Context-free language = PDA = NFA + Stack (z_0)



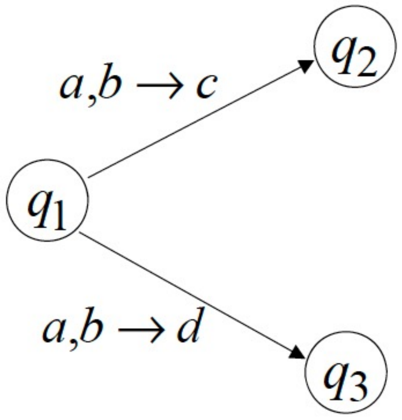
Push-Down Automata (PDA)

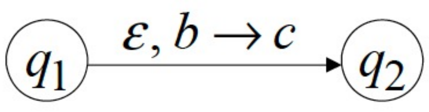
- Regular language = DFA/NFA
- Context-free language = PDA = NFA + Stack (z_0) = NPDA



Non-Deterministic PDA (NPDA)

- Regular language = DFA/NFA
- Context-free language = PDA = NFA + Stack (z_0) = NPDA





Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states

Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states
  - Σ: finite input alphabet

Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states
  - Σ: finite input alphabet
  - Γ: finite stack alphabet

Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states
  - Σ: finite input alphabet
  - Γ: finite stack alphabet
  - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longmapsto 2^{Q \times \Gamma^*}$: transition

Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states
  - Σ: finite input alphabet
  - Γ: finite stack alphabet
  - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longmapsto 2^{Q \times \Gamma^*}$: transition
  - $q_0 \in Q$: initial state

Non-Deterministic PDA (NPDA)

- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states
  - Σ: finite input alphabet
  - Γ: finite stack alphabet
  - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longmapsto 2^{Q \times \Gamma^*}$: transition
  - $q_0 \in Q$: initial state
  - $z_0 \in \Gamma$: stack start symbol

Non-Deterministic PDA (NPDA)

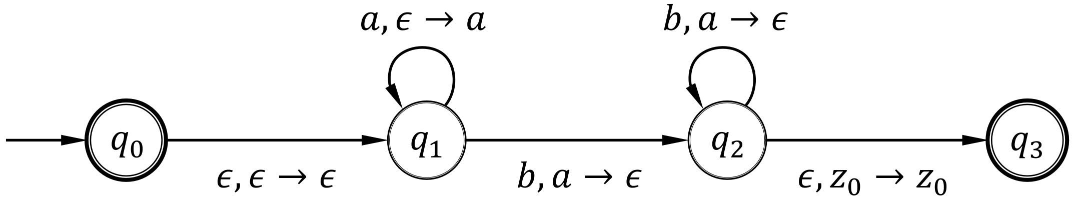
- NPDA/PDA is defined by the septuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$
  - Q: a finite set of states
  - Σ: finite input alphabet
  - Γ: finite stack alphabet
  - $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \longmapsto 2^{Q \times \Gamma^*}$: transition
  - $q_0 \in Q$: initial state
  - $z_0 \in \Gamma$: stack start symbol
  - $F \subseteq Q$: set of final states

Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA

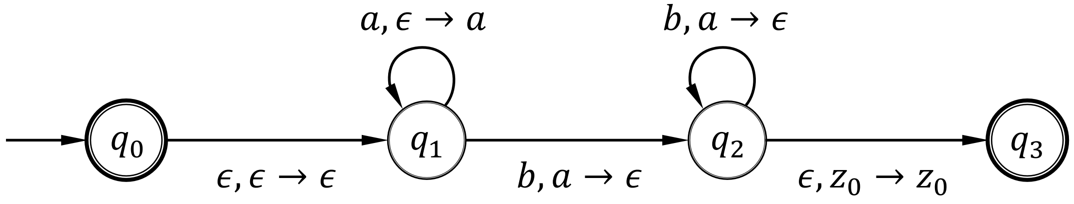
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



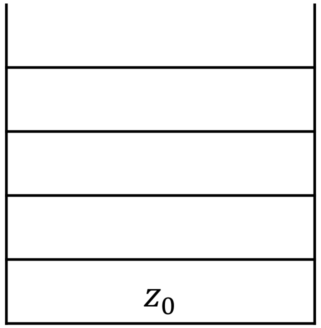
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



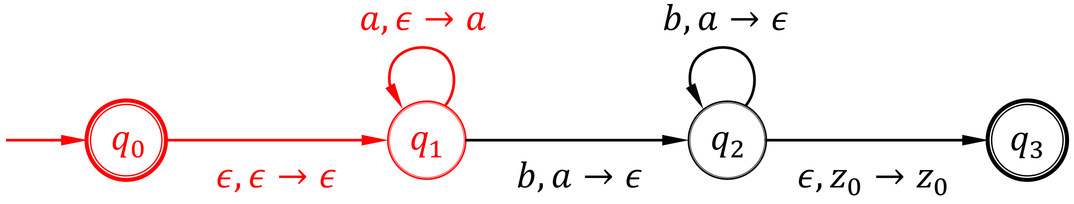
a	a	a	b	b	b

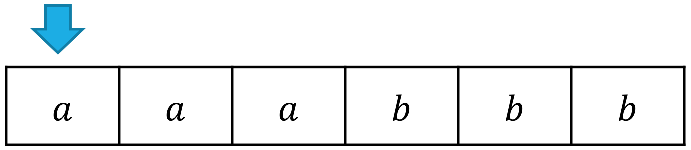
- Let's check how it accepts the string $aaabbb$



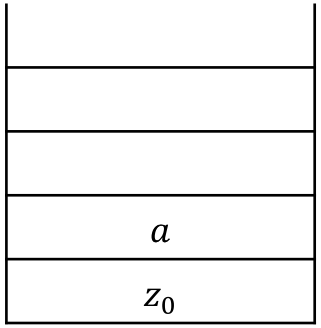
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



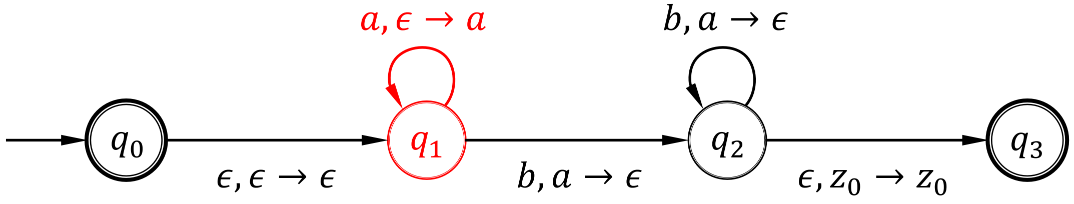


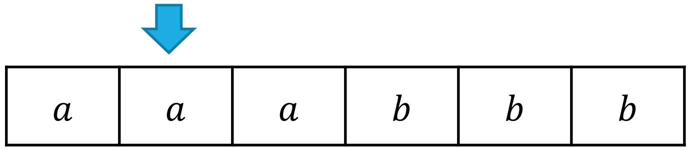
- Let's check how it accepts the string $aaabbb$



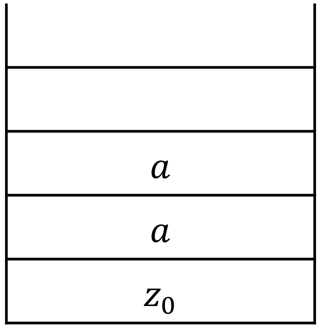
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



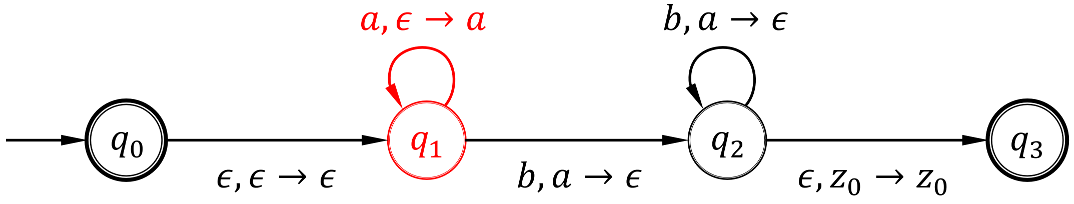


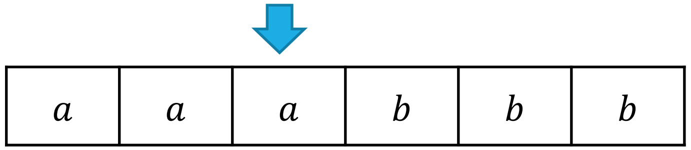
- Let's check how it accepts the string $aaabbb$



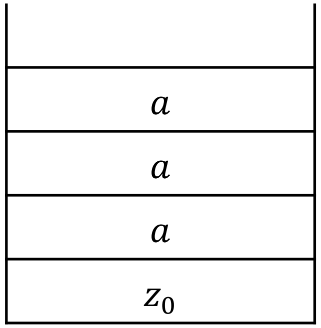
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



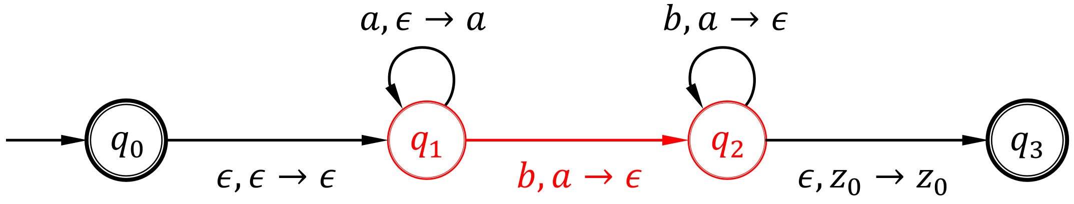


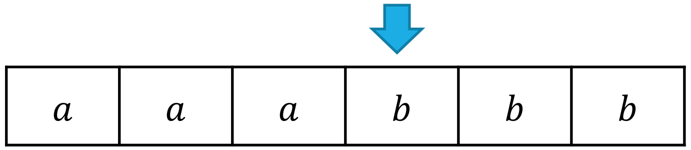
- Let's check how it accepts the string $aaabbb$



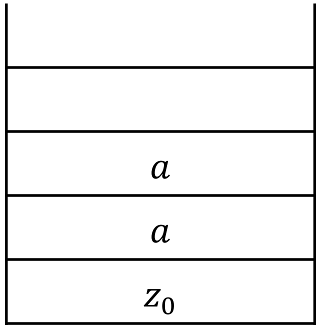
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



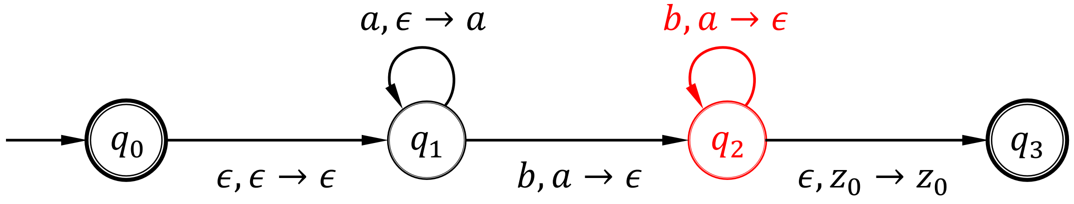


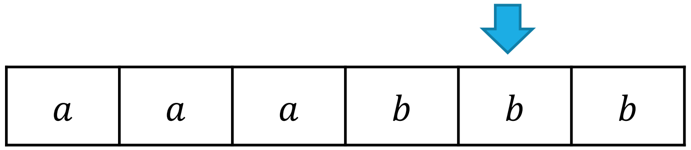
- Let's check how it accepts the string $aaabbb$



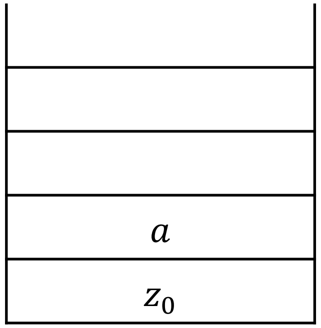
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



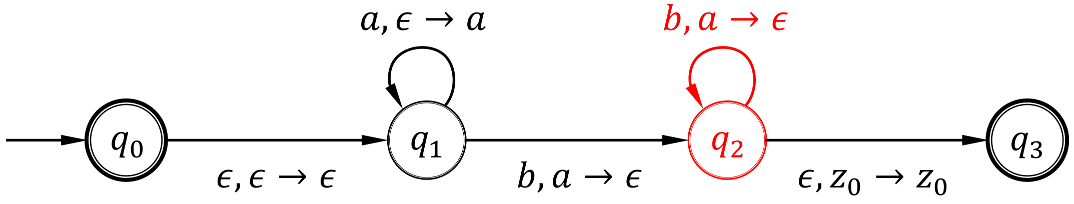


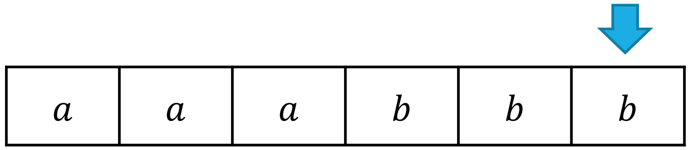
- Let's check how it accepts the string $aaabbb$



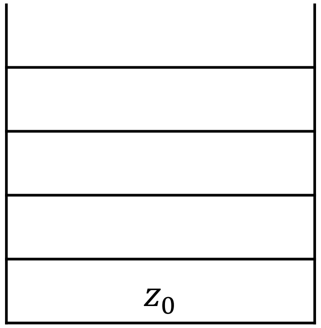
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



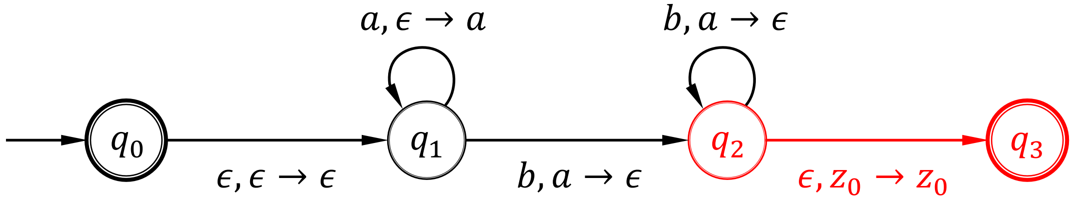


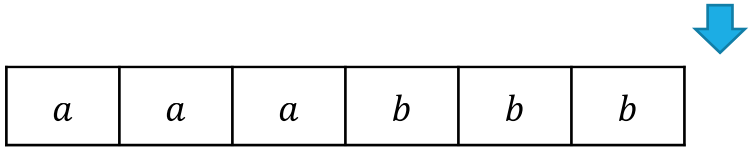
- Let's check how it accepts the string $aaabbb$



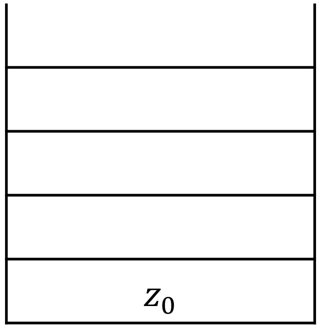
Non-Deterministic PDA (NPDA)

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA





- Let's check how it accepts the string $aaabbb$



Instantaneous Description

- An instantaneous description of a PDA is a triple

$$(q,\ w,\ \gamma)$$

Current state

Remaining unread input

Remaining stack contents

Instantaneous Description

- An instantaneous description of a PDA is a triple

$$(q,\ w,\ \gamma)$$

Current state

Remaining unread input

Remaining stack contents

- If $(q, \alpha) \in \delta(p, a, X)$ then

Instantaneous Description

- An instantaneous description of a PDA is a triple

$$(q,\ w,\ \gamma)$$

Current state

Remaining
unread input

Remaining
stack contents

- If $(q, \alpha) \in \delta(p, a, X)$ then $(p, aw, X\beta) \vdash_M (q, w, \alpha\beta)$

Instantaneous Description

- An instantaneous description of a PDA is a triple

$$(q,\ w,\ \gamma)$$

Current state

Remaining unread input

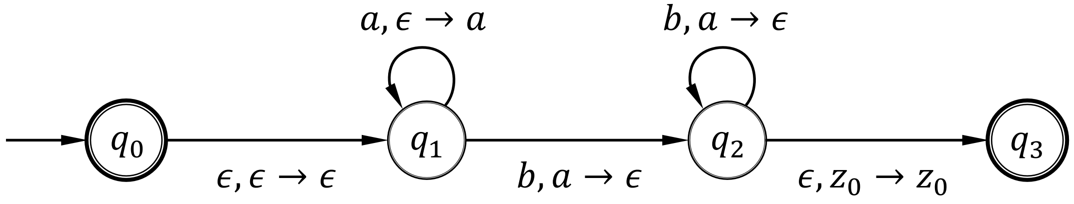
Remaining stack contents

- If $(q, \alpha) \in \delta(p, a, X)$ then $(p, aw, X\beta) \vdash_M (q, w, \alpha\beta)$

Write $\vdash$ when M is clear

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

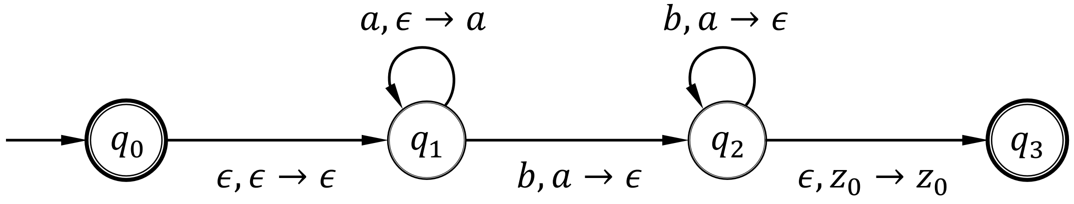


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

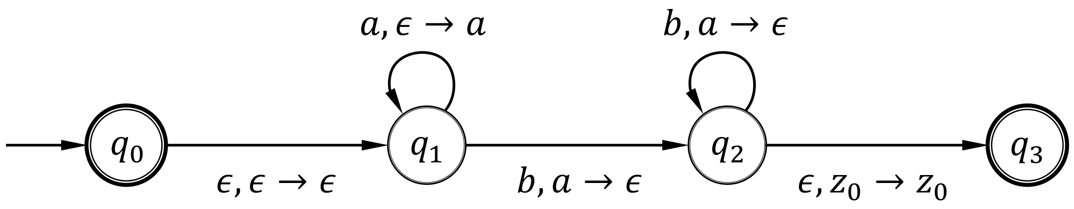


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

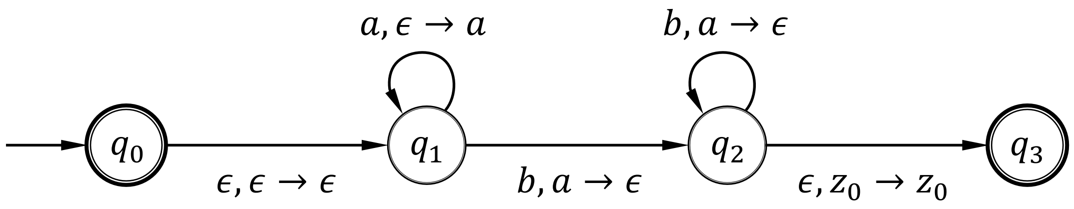


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0) \vdash (q_1, abbb, aaz_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

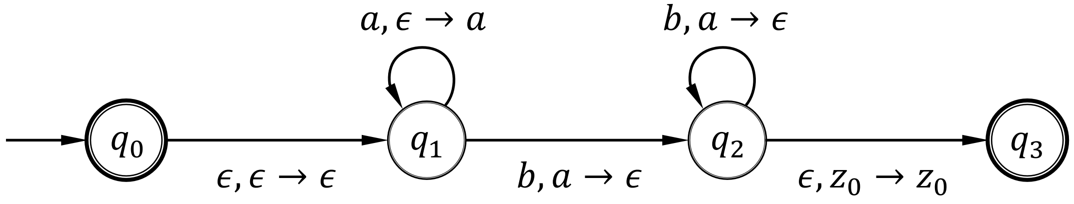


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0) \vdash (q_1, abbb, aaz_0) \vdash (q_1, bbb, aaaz_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

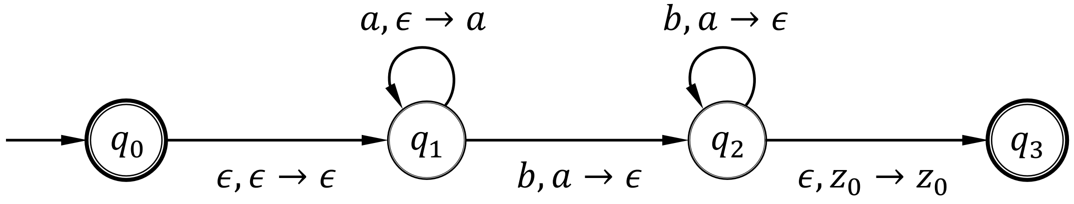


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0) \vdash (q_1, abbb, aaz_0) \vdash (q_1, bbb, aaaz_0)$$
$$\vdash (q_2, bb, aaz_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

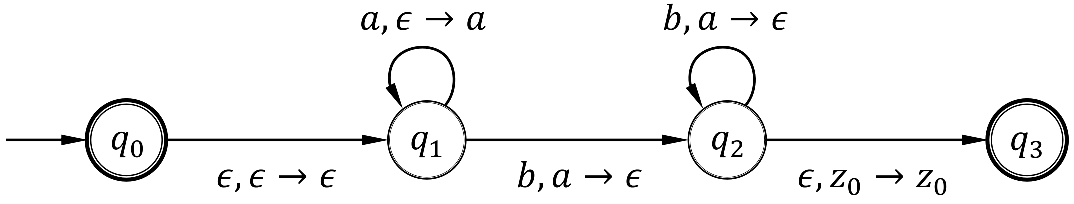


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0) \vdash (q_1, abbb, aaz_0) \vdash (q_1, bbb, aaaz_0)$$
$$\vdash (q_2, bb, aaz_0) \vdash (q_2, b, az_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA

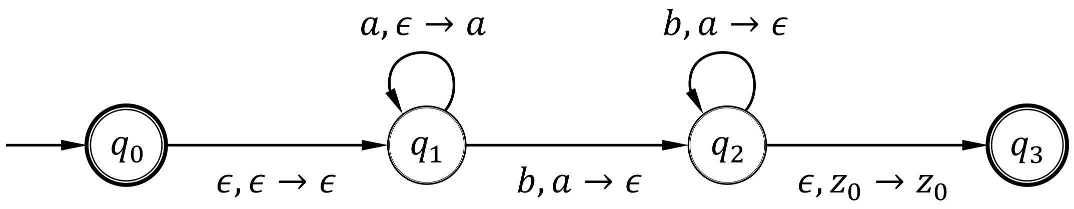


- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0) \vdash (q_1, abbb, aaz_0) \vdash (q_1, bbb, aaaz_0)$$
$$\vdash (q_2, bb, aaz_0) \vdash (q_2, b, az_0) \vdash (q_2, \epsilon, z_0)$$

Instantaneous Description

- Recall that $a^n b^n$ is not regular and cannot be accepted by DFA
- Strings in the language can be accepted by the following NPDA



- Let's check how it accepts the string $aaabbb$

$$(q_0, aaabbb, z_0) \vdash (q_1, aaabbb, z_0) \vdash (q_1, aabbb, az_0) \vdash (q_1, abbb, aaz_0) \vdash (q_1, bbb, aaaz_0)$$
$$\vdash (q_2, bb, aaz_0) \vdash (q_2, b, az_0) \vdash (q_2, \epsilon, z_0) \vdash (q_3, \epsilon, z_0)$$

Language of PDA

- A string is accepted if there is a computation such that
- (1) all the input symbols are consumed
- (2) the last state is a final state

Language of PDA

- A string is accepted if there is a computation such that
- (1) all the input symbols are consumed
- (2) the last state is a final state

- At the end of computation, we don’t care about stack contents

Language of PDA

- A string is accepted if there is a computation such that
- (1) all the input symbols are consumed
- (2) the last state is a final state

- At the end of computation, we don’t care about stack contents

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$

Language of PDA

- A string is accepted if there is a computation such that
- (1) all the input symbols are consumed
- (2) the last state is a final state

- At the end of computation, we don't care about stack contents

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

Language of PDA

- A string is accepted if there is a computation such that
- (1) all the input symbols are consumed
- (2) the last state is a final state

- At the end of computation, we don't care about stack contents

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

Equivalent definitions
(1) acceptance by final states
(2) acceptance by empty stack

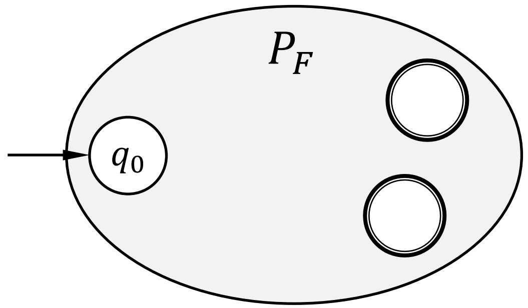
Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

- If there is a PDA that accepts strings by final states, there is a PDA accepting the strings by empty stack, vice versa.

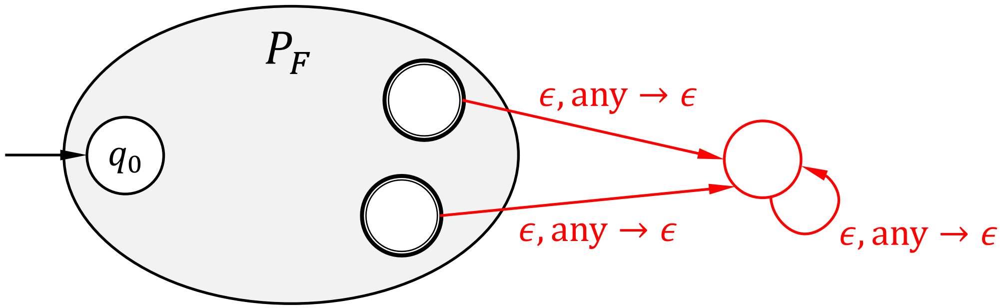
Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$
- From final states (P_F) to empty stack ($P_\emptyset$)



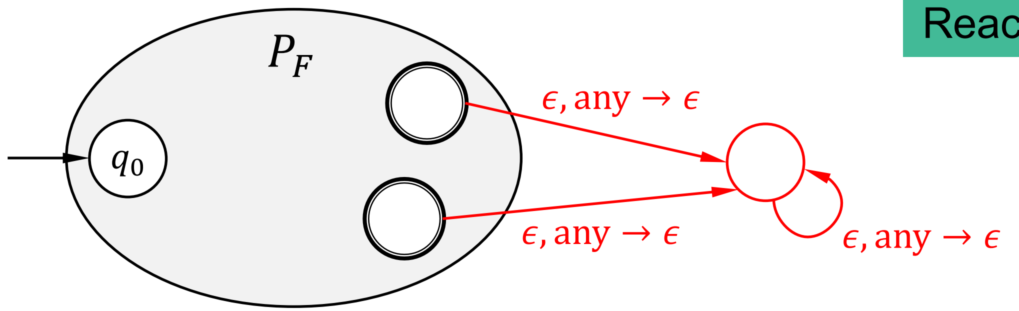
Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$
- From final states (P_F) to empty stack ($P_\emptyset$)



Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$
- From final states (P_F) to empty stack ($P_\emptyset$)



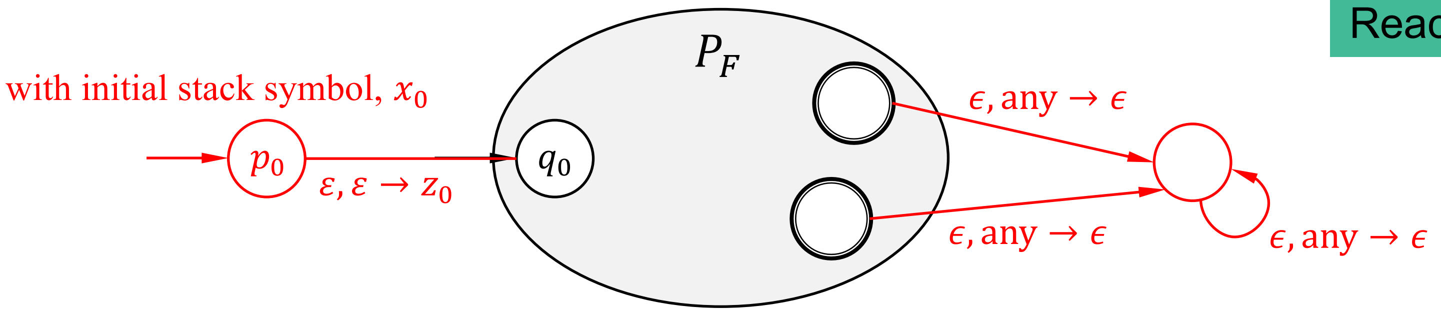
Reaching Final States => Reaching Empty Stack

Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

- From final states (P_F) to empty stack ($P_\emptyset$)

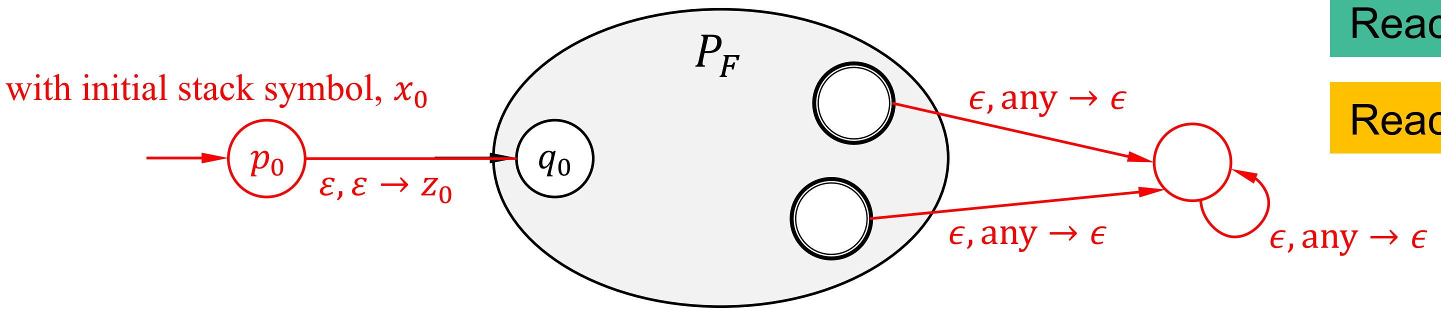
Reaching Final States => Reaching Empty Stack



Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

- From final states (P_F) to empty stack ($P_\emptyset$)



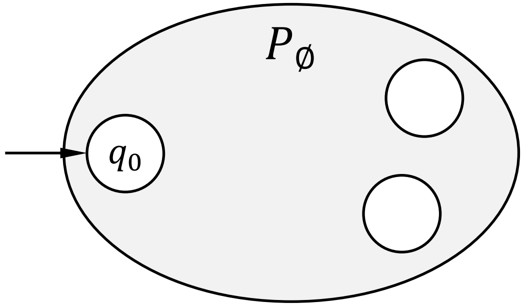
Reaching Final States => Reaching Empty Stack

Reaching Final States <= Reaching Empty Stack

Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

- From empty stack ($P_\emptyset$) to final states (P_F)

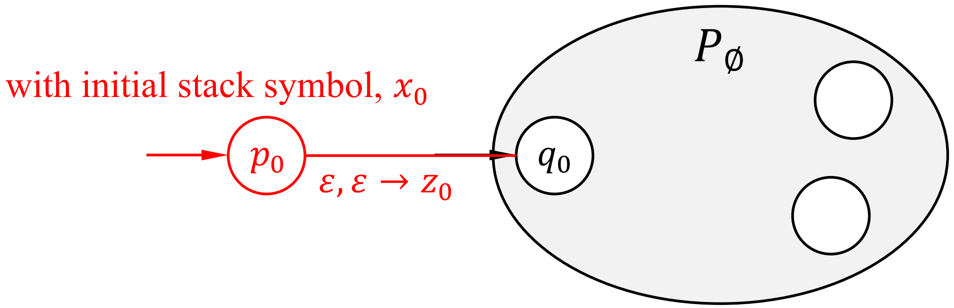


Reaching Final States => Reaching Empty Stack

Reaching Final States <= Reaching Empty Stack

Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$
- From empty stack ($P_\emptyset$) to final states (P_F)



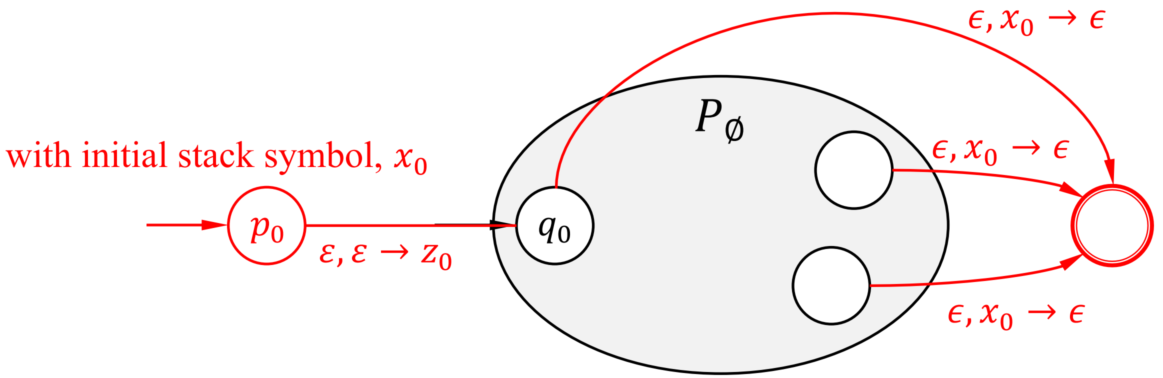
Reaching Final States => Reaching Empty Stack

Reaching Final States <= Reaching Empty Stack

Language of PDA

- The language of a PDA is the set of strings accepted by it
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q_f, \epsilon, u), q_f \in F, u \in \Gamma^*\}$
  - $L(M) = \{w \in \Sigma^*: (q_0, w, z_0) \vdash_M^* (q, \epsilon, \epsilon), q \in Q\}$

- From empty stack ($P_\emptyset$) to final states (P_F)



Reaching Final States => Reaching Empty Stack

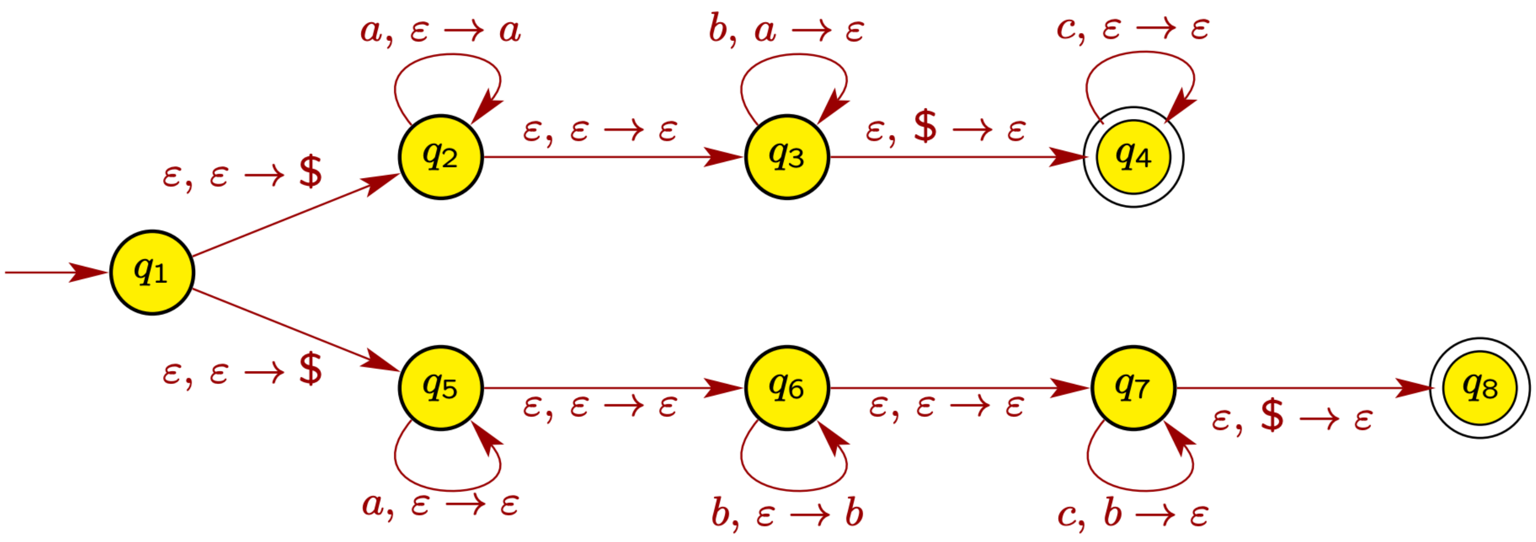
Reaching Final States <= Reaching Empty Stack

Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i = j$ or $j = k$, and write the formal definition of the PDA

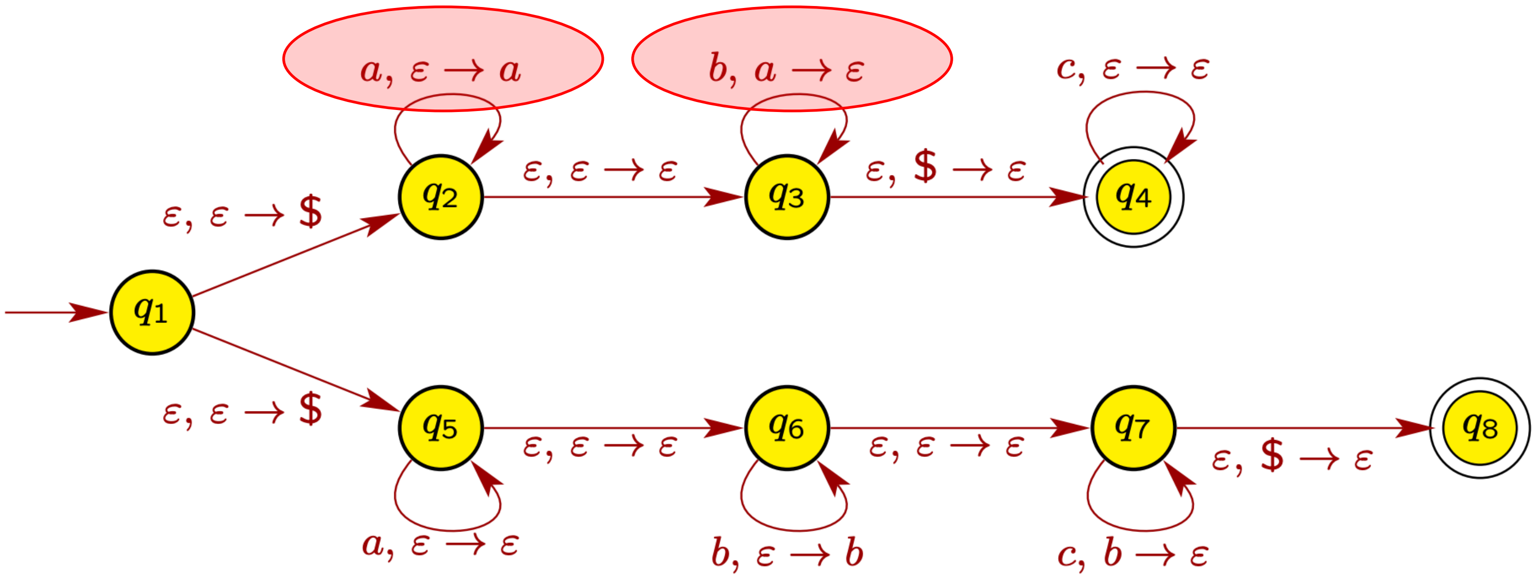
Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i = j$ or $j = k$, and write the formal definition of the PDA



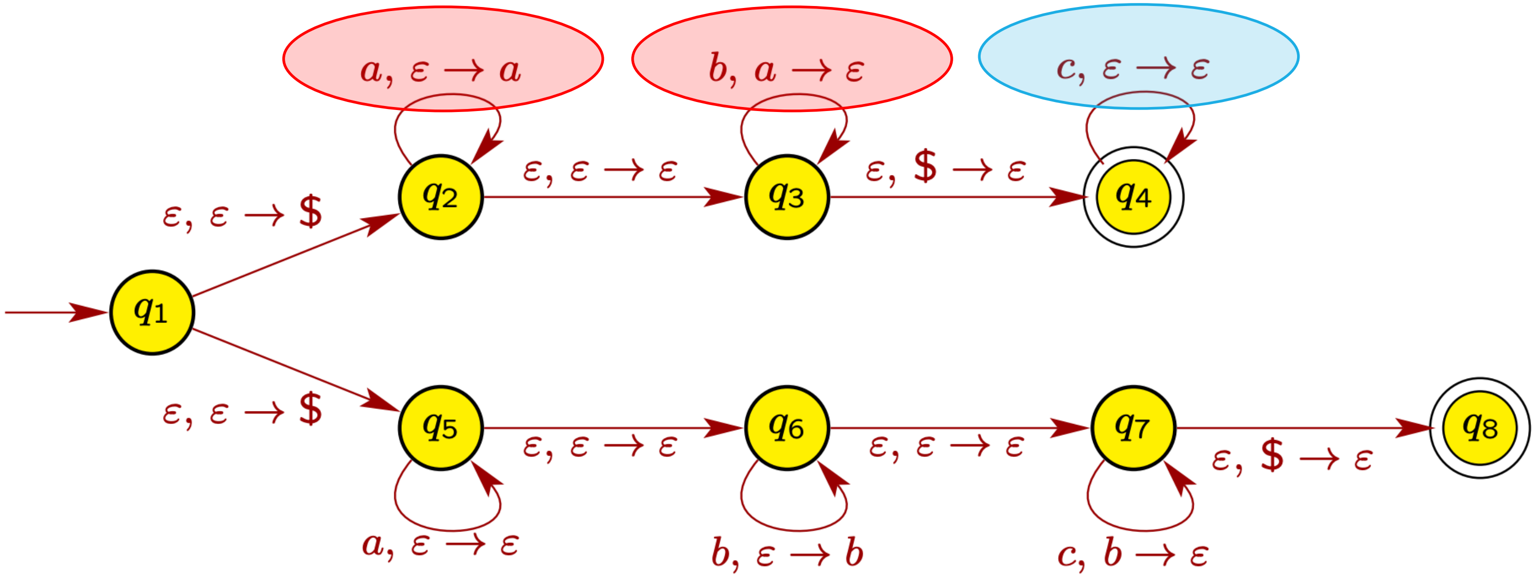
Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i = j$ or $j = k$, and write the formal definition of the PDA



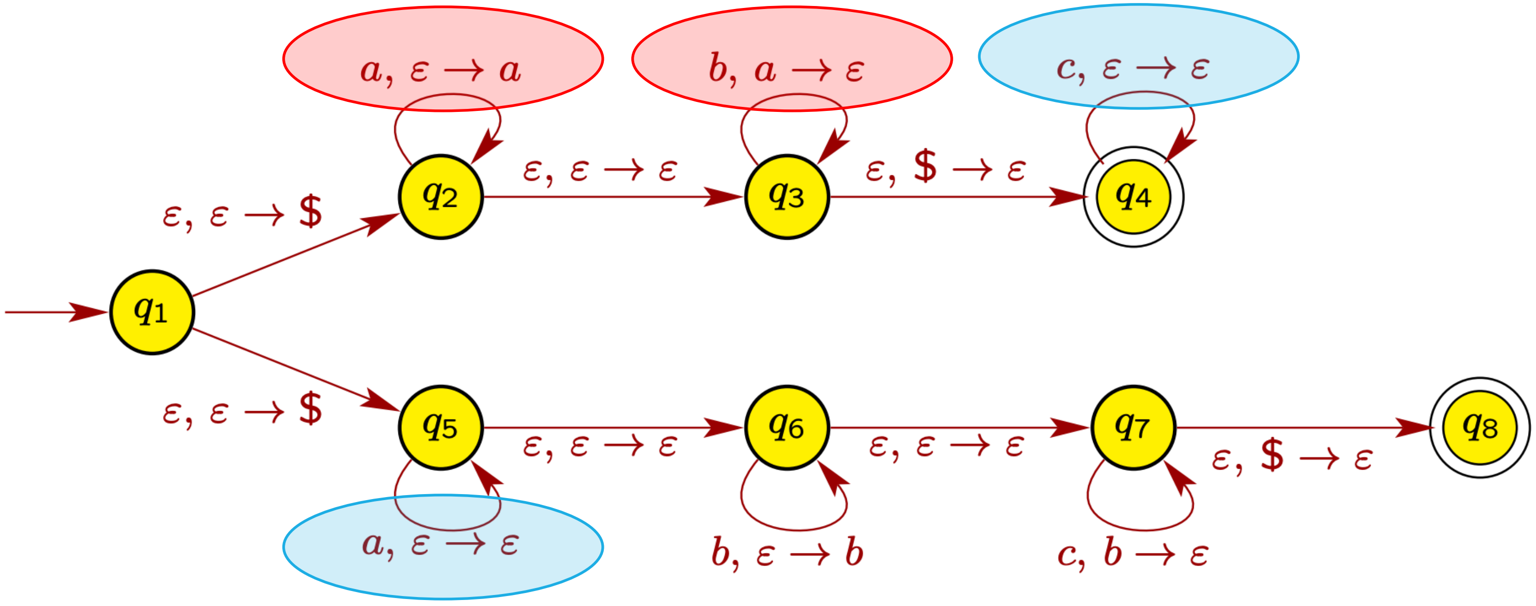
Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i = j$ or $j = k$, and write the formal definition of the PDA



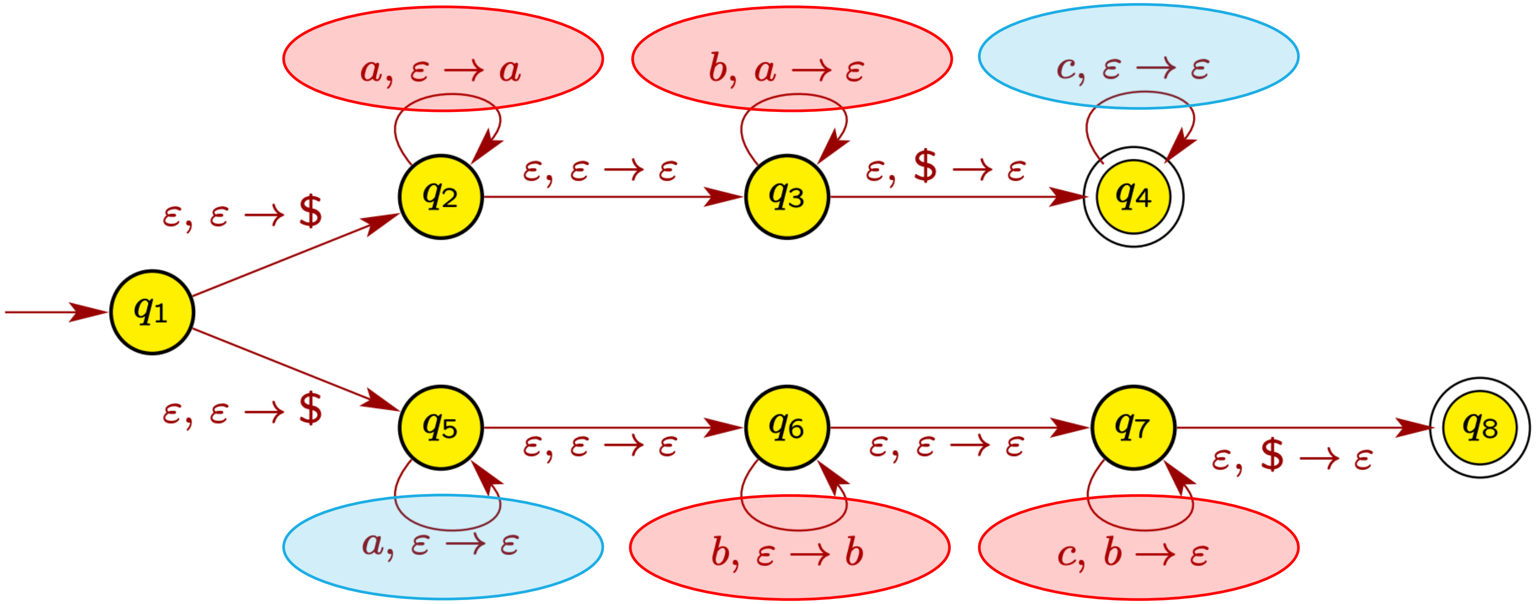
Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i = j$ or $j = k$, and write the formal definition of the PDA



Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i = j$ or $j = k$, and write the formal definition of the PDA

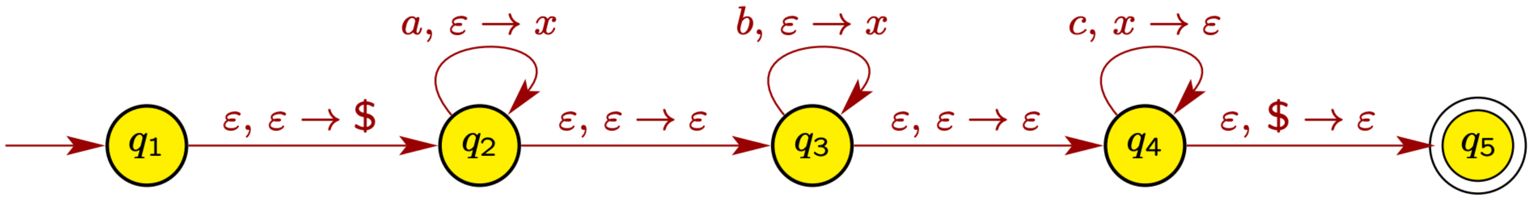


Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i + j = k$, and write the formal definition of the PDA

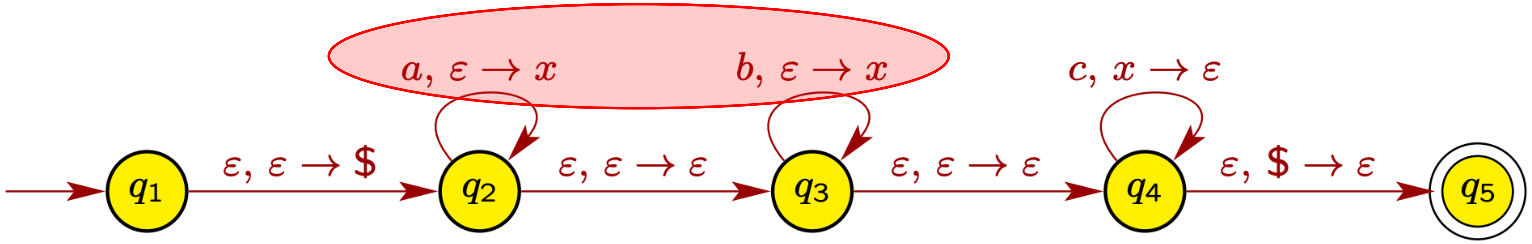
Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i + j = k$, and write the formal definition of the PDA



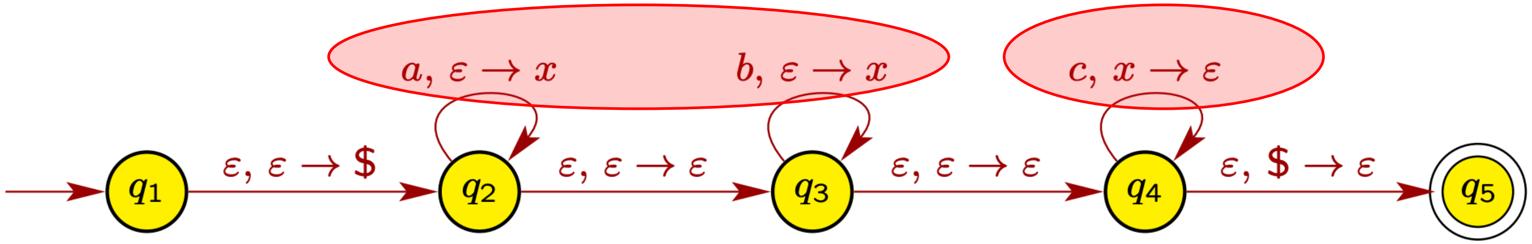
Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i + j = k$, and write the formal definition of the PDA



Exercises

- Design a PDA to accept $a^i b^j c^k$ where $i + j = k$, and write the formal definition of the PDA

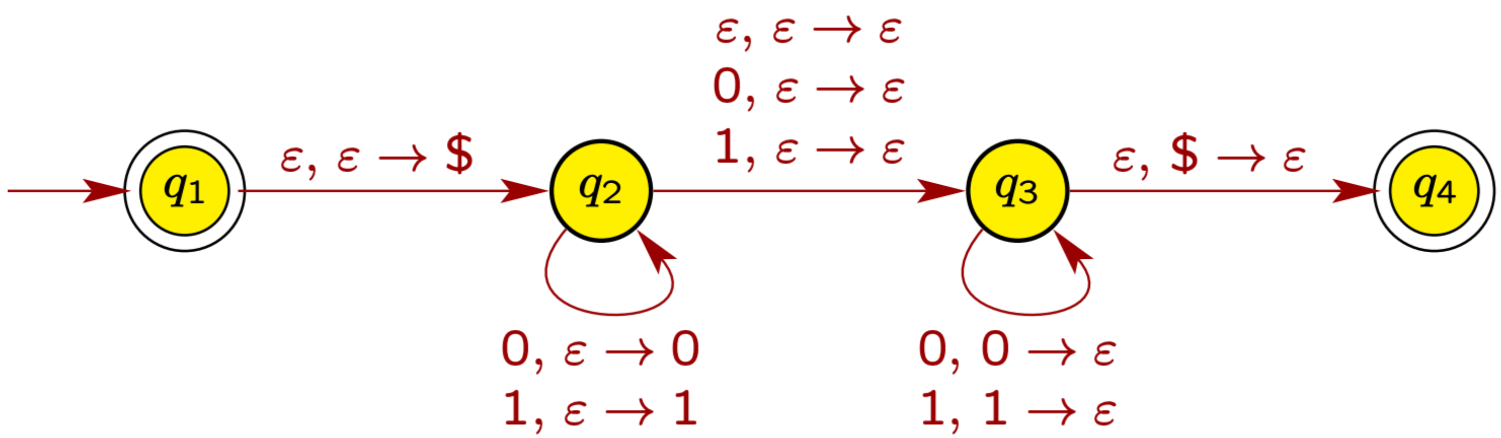


Exercises

- Design a PDA to accept $w \in \{0,1\}^*$ where $w = w^R$. Write the formal definition of the PDA

Exercises

- Design a PDA to accept $w \in \{0,1\}^*$ where $w = w^R$. Write the formal definition of the PDA

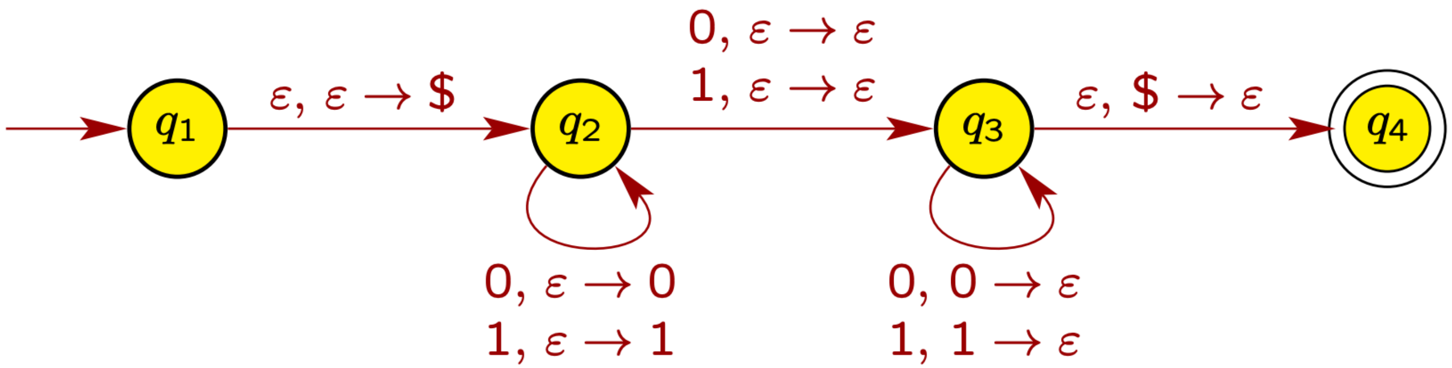


Exercises

- Design a PDA to accept $w \in \{0,1\}^*$ where $w = w^R$ and $|w|$ is odd. Write the formal definition of the PDA

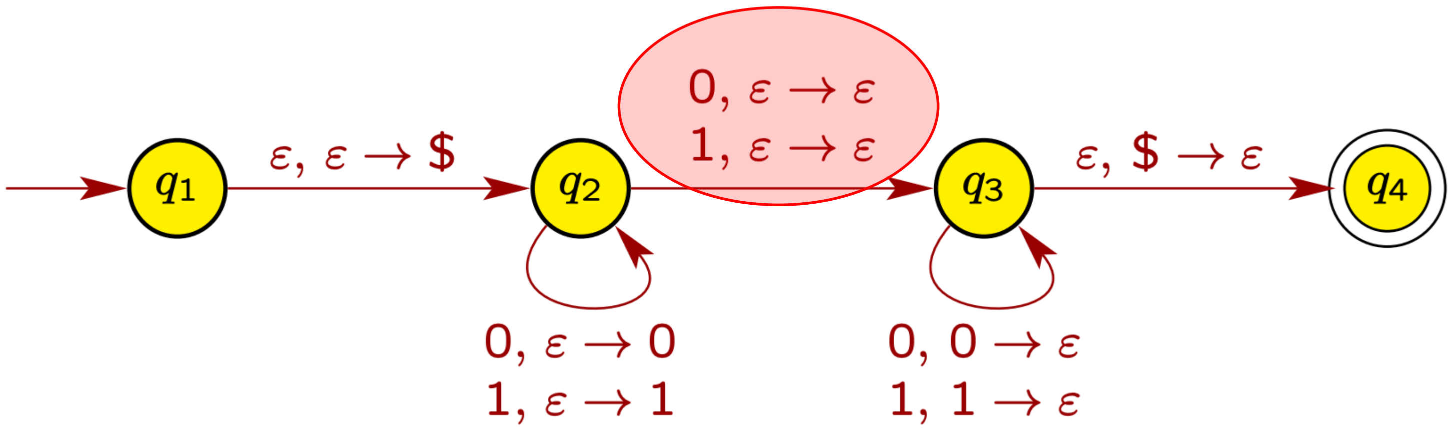
Exercises

- Design a PDA to accept $w \in \{0,1\}^*$ where $w = w^R$ and $|w|$ is odd. Write the formal definition of the PDA



Exercises

- Design a PDA to accept $w \in \{0,1\}^*$ where $w = w^R$ and $|w|$ is odd. Write the formal definition of the PDA



Exercises

- Prove that, for any PDA, there is an equivalent PDA consisting of only two stack symbols

PART III: CFG = PDA

Equivalence of PDA and CFG

- CFG $\subseteq$ PDA
- PDA $\subseteq$ CFG

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

Keep the Top-of-Stack value as the symbol to consume or derive

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$$

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$$

The current Top-of-Stack value

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$$

The current Top-of-Stack value

Replace it with β to simulate the derivation

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \rightarrow \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, \epsilon, A) = \{(q, \beta) : A \rightarrow \beta \in P\}$$

Derivation does not read any input symbol

The current Top-of-Stack value

Replace it with β to simulate the derivation

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

Read and Pop the Top-of-Stack value *a*

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

Read and Pop the Top-of-Stack value *a*

CFG $\subseteq$ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

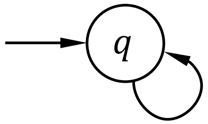
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$

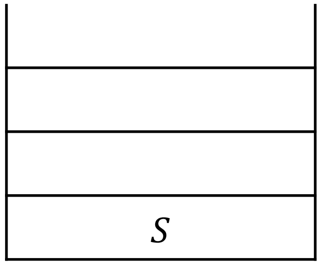
Exercise: Try to draw the PDA!

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) \colon A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$

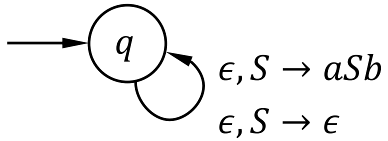


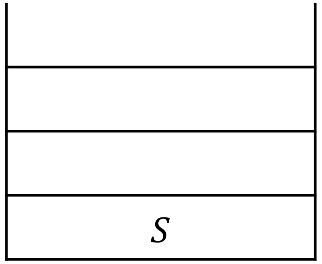


CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$

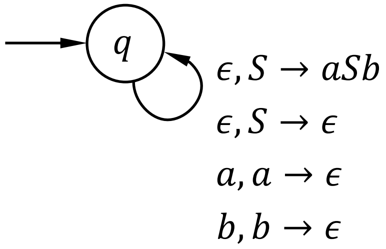


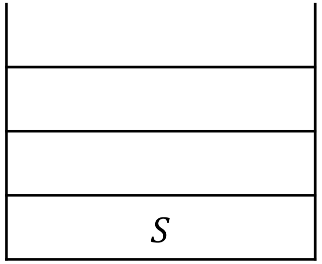


CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$

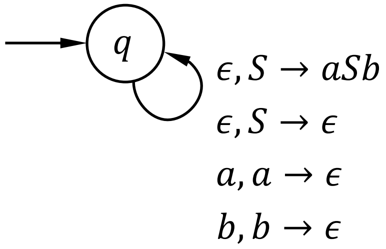


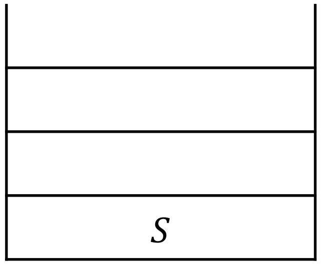


CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with** $\boldsymbol{aabb}$



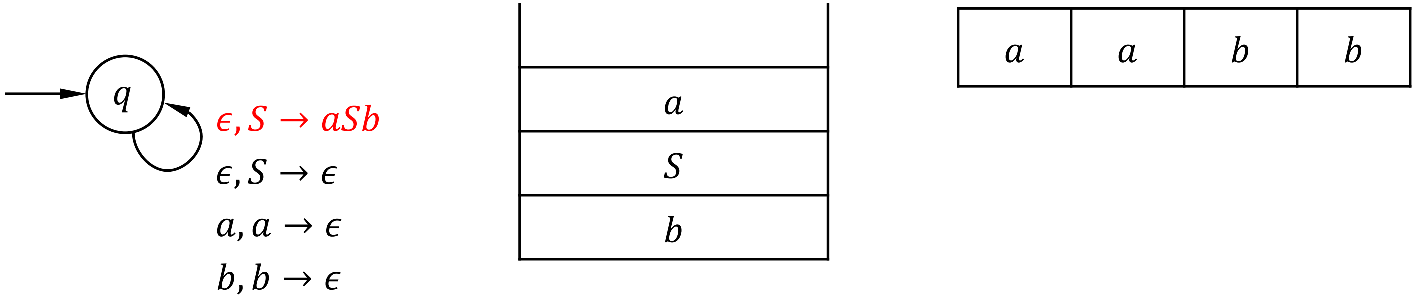


a	a	b	b

CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

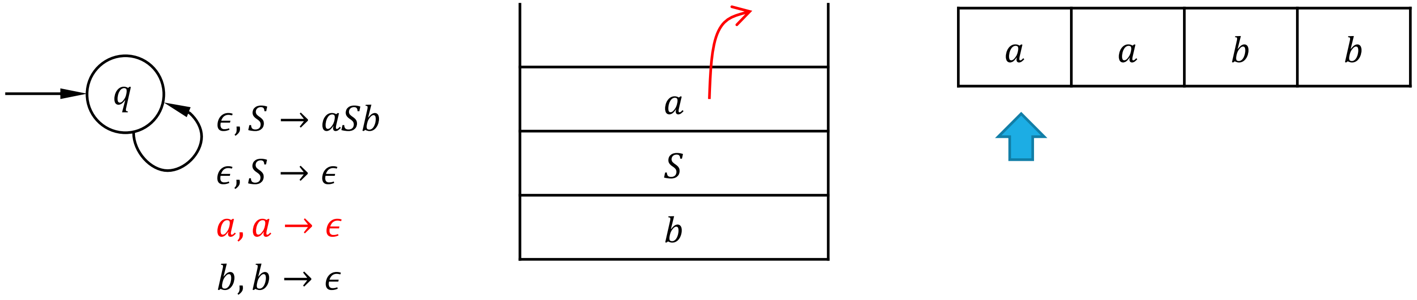
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with *aabb***



CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

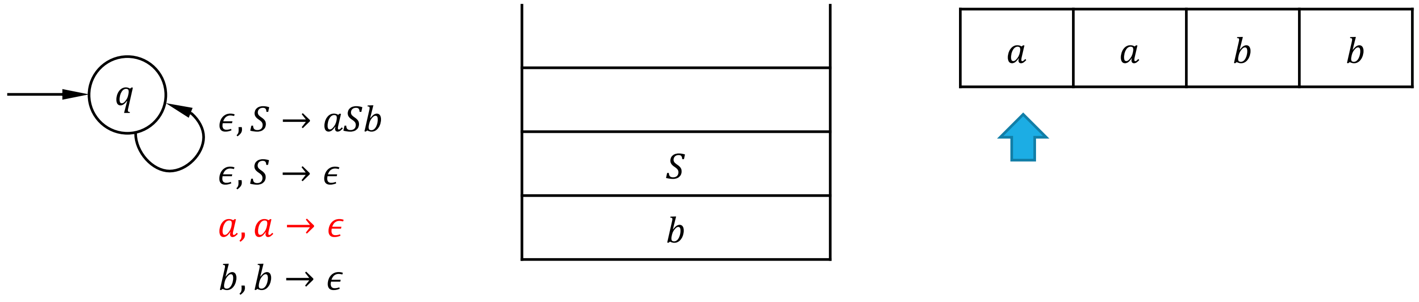
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with** $\boldsymbol{aabb}$



CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

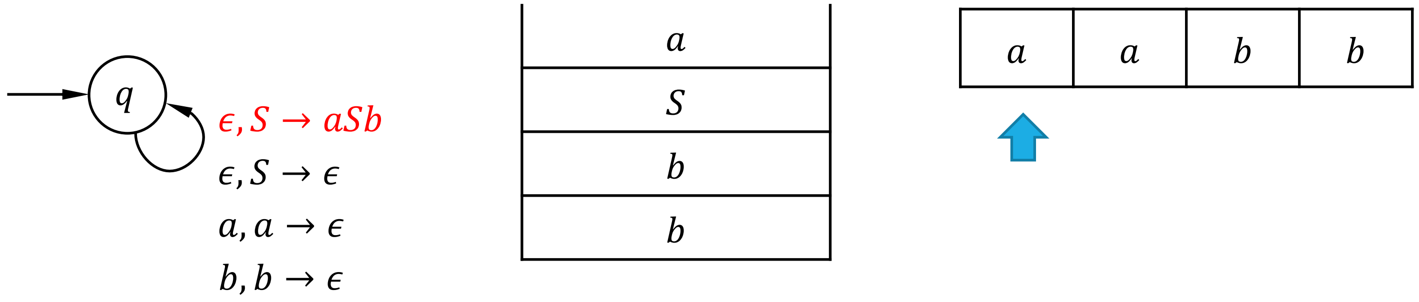
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with *aabb***



CFG $\subseteq$ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \rightarrow \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

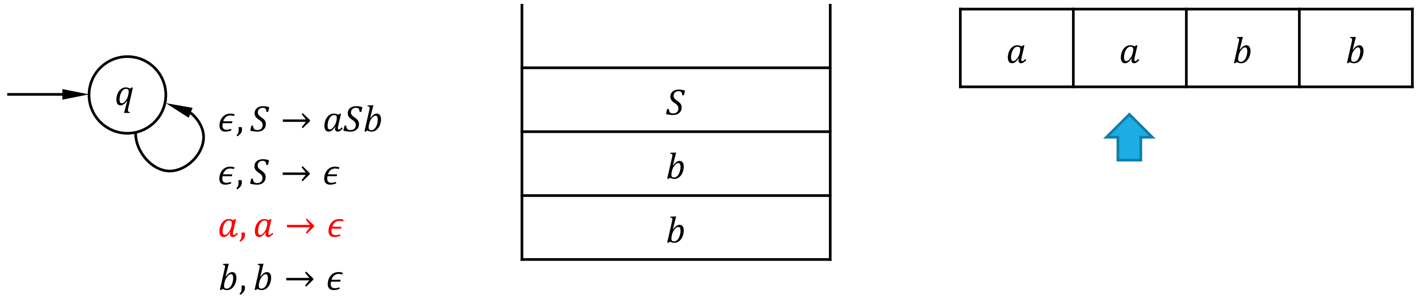
- **Example**: $S \rightarrow aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with** $\boldsymbol{aabb}$



CFG $\subseteq$ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta): A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

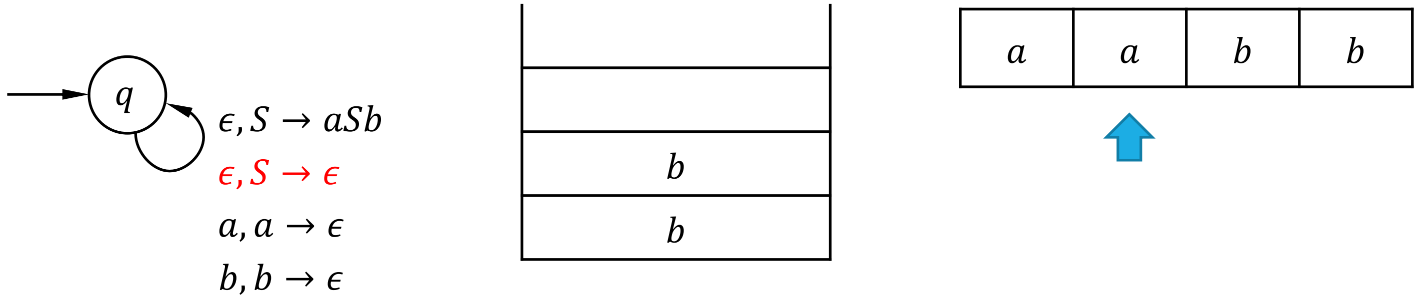
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with *aabb***



CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

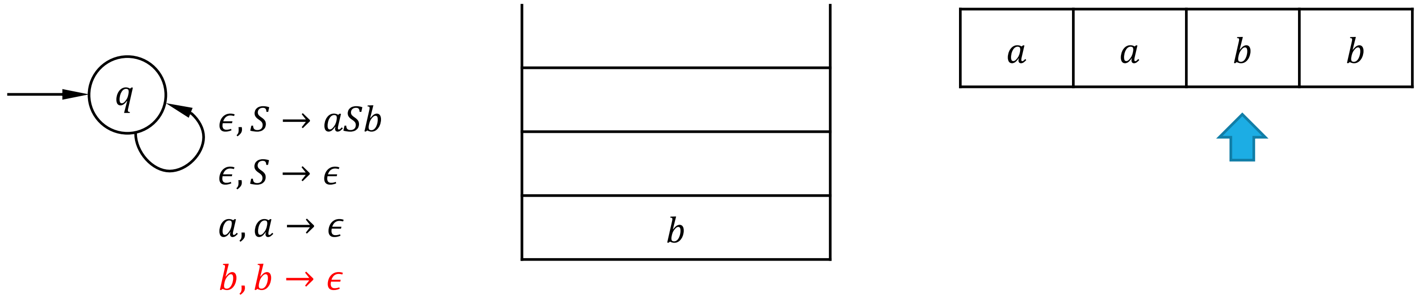
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with *aabb***



CFG $\subseteq$ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

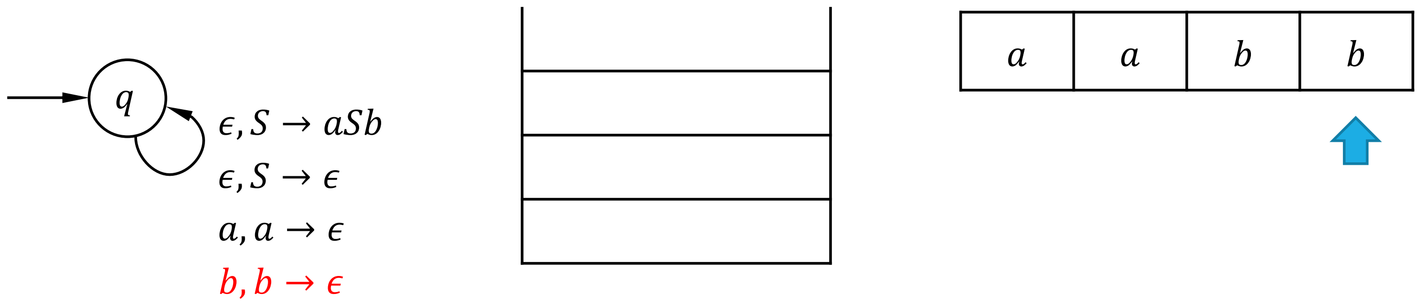
- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with** $\boldsymbol{aabb}$



CFG ⊆ PDA

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \to \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

- **Example**: $S \to aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with** $\boldsymbol{aabb}$

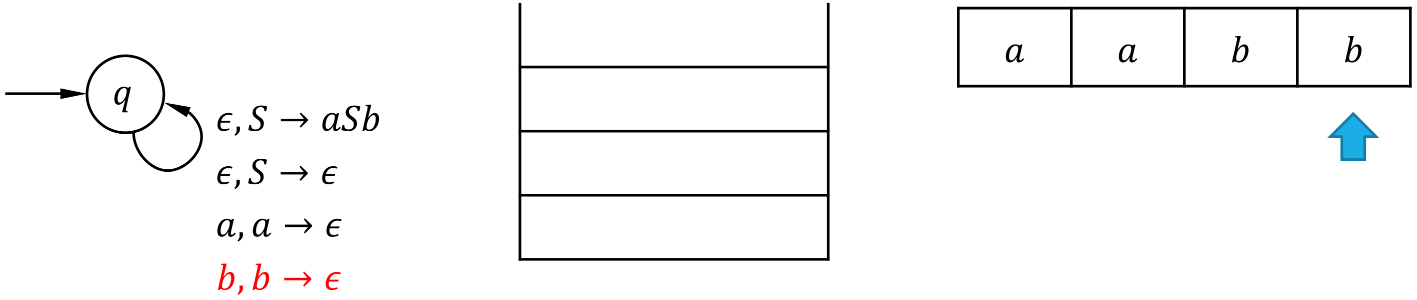


CFG ⊆ PDA

Exercise: Try to prove the equivalence!

- Given any CFG, $G = (N, T, P, S)$, we can build an equivalent PDA that accepts string by empty stack, $(\{q\}, T, N \cup T, \delta, q, S, \emptyset)$, where
  - (1) for any non-terminal $A \in N$, $\delta(q, \epsilon, A) = \{(q, \beta) : A \rightarrow \beta \in P\}$
  - (2) for any terminal $a \in T$, $\delta(q, a, a) = \{(q, \epsilon)\}$

- **Example**: $S \rightarrow aSb \mid \epsilon$, the grammar for $a^n b^n$. **Check with** $\boldsymbol{aabb}$



PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$

From p to q, pop X

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \rightarrow N_{q_0 z_0 p} \in P$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \rightarrow N_{q_0 z_0 p} \in P$

Pop z$_0$, leading to empty stack

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$

Pop z_0, leading to empty stack

Strings accepted by empty stack = Strings derived from S

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \rightarrow N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \rightarrow a \in P$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$

Pop X, consuming *a*

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$

Pop X, consuming *a*

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \rightarrow N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \rightarrow a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \rightarrow aN_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$

南京大学编译技术
及软件安全研究组

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$

Pop X, consuming *a*, pushing $X_1X_2 \ldots X_k$

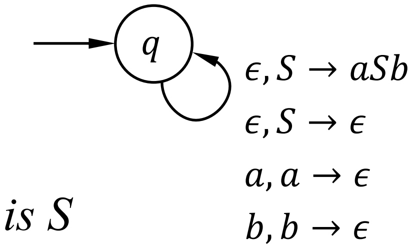
PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$

Pop X, consuming *a*, pushing $X_1X_2 \ldots X_k$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \rightarrow N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \rightarrow a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \rightarrow a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**

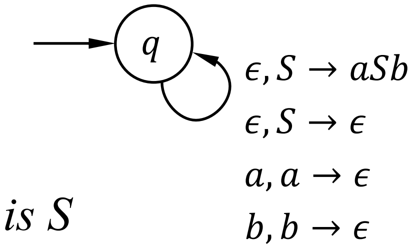


initial stack symbol is S

Exercise: Try to write the CFG!

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \rightarrow N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \rightarrow a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \rightarrow a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**

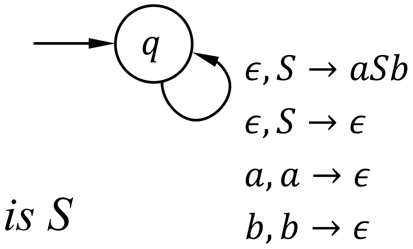


initial stack symbol is S

$S \rightarrow N_{qSq}$

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q : S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**

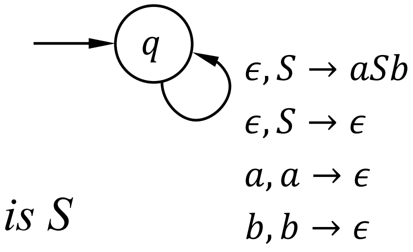


initial stack symbol is S

$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qaq} \to a$
$N_{qbq} \to b$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq}: p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**

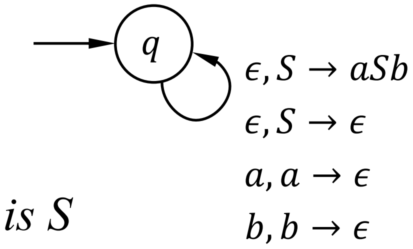


initial stack symbol is S

$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qaq} \to a$
$N_{qbq} \to b$
$N_{qSq} \to \epsilon N_{qaq} N_{qSq} N_{qbq}$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q : S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**

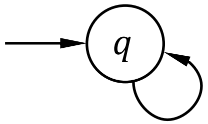


initial stack symbol is S

$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qaq} \to a$
$N_{qbq} \to b$
$N_{qSq} \to \epsilon N_{qaq} N_{qSq} N_{qbq} \to a N_{qsq} b$

PDA ⊆ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**



$\epsilon, S \to aSb$
$\epsilon, S \to \epsilon$
$a, a \to \epsilon$
$b, b \to \epsilon$

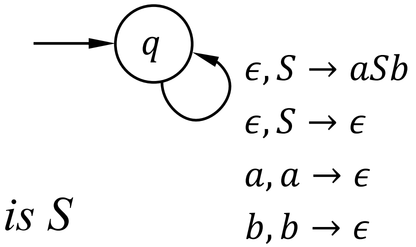
initial stack symbol is S

$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qaq} \to a$
$N_{qbq} \to b$
$N_{qSq} \to \epsilon N_{qaq} N_{qSq} N_{qbq} \to a N_{qsq} b$

$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qSq} \to a N_{qsq} b$

PDA $\subseteq$ CFG

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q: S \to N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \to a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \to a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**



initial stack symbol is S

$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qaq} \to a$
$N_{qbq} \to b$
$N_{qSq} \to \epsilon N_{qaq} N_{qSq} N_{qbq} \to a N_{qsq} b$

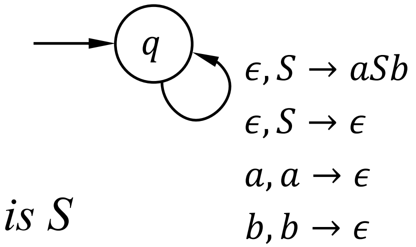
$S \to N_{qSq}$
$N_{qSq} \to \epsilon$
$N_{qSq} \to a N_{qsq} b$

$S \to \epsilon$
$S \to aSb$

PDA $\subseteq$ CFG

Exercise: Try to prove the equivalence!

- Given any PDA, $(Q, \Sigma, \Gamma, \delta, q_0, z_0, \emptyset)$, we can build an equivalent CFG (N, Σ, P, S), where $N = \{S\} \cup \{N_{pXq} : p, q \in Q, X \in \Gamma\}$
- The production rules are defined below, where $a \in \Sigma \cup \{\epsilon\}$
  - (1) $\forall p \in Q : S \rightarrow N_{q_0 z_0 p} \in P$
  - (2) $(q, \epsilon) \in \delta(p, a, X) \Rightarrow N_{pXq} \rightarrow a \in P$
  - (3) $(q, X_1 X_2 \cdots X_k) \in \delta(p, a, X) \Rightarrow N_{pXp_k} \rightarrow a N_{qX_1p_1} N_{p_1X_2p_2} \cdots N_{p_{k-1}X_kp_k} \in P$
- **Example:**



initial stack symbol is S

$S \rightarrow N_{qSq}$
$N_{qSq} \rightarrow \epsilon$
$N_{qaq} \rightarrow a$
$N_{qbq} \rightarrow b$
$N_{qSq} \rightarrow \epsilon N_{qaq} N_{qSq} N_{qbq} \rightarrow a N_{qsq} b$

$S \rightarrow N_{qSq}$
$N_{qSq} \rightarrow \epsilon$
$N_{qSq} \rightarrow a N_{qsq} b$

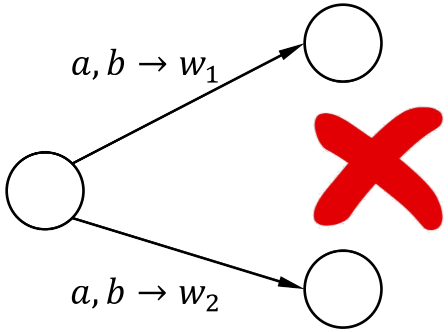
$S \rightarrow \epsilon$
$S \rightarrow aSb$

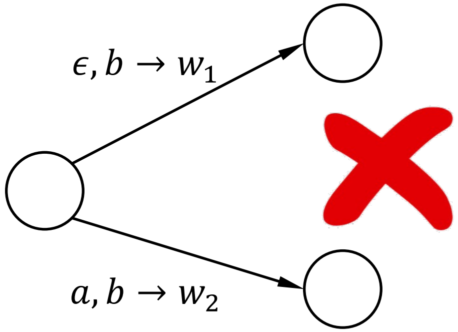
Deterministic PDA (DPDA)

- DFA/NFA $\subseteq$ DPDA $\subseteq$ NPDA

Deterministic PDA (DPDA)

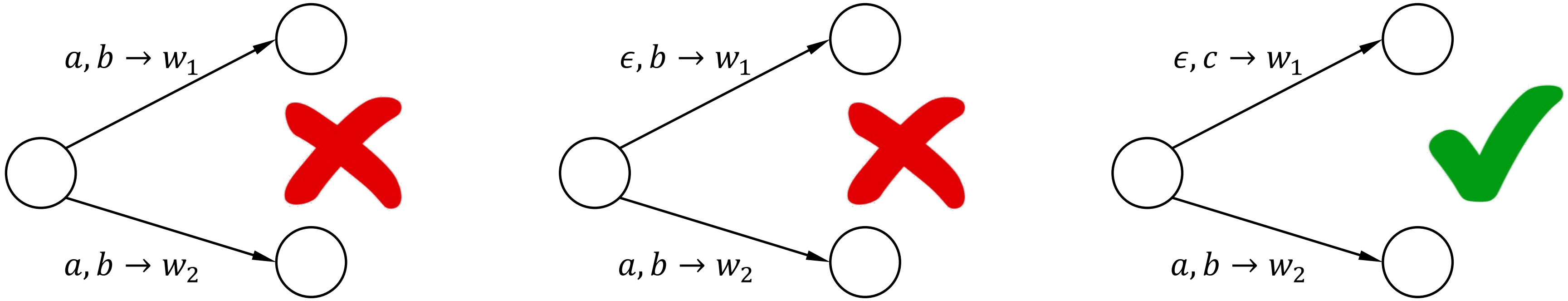
- DFA/NFA ⊆ DPDA ⊆ NPDA





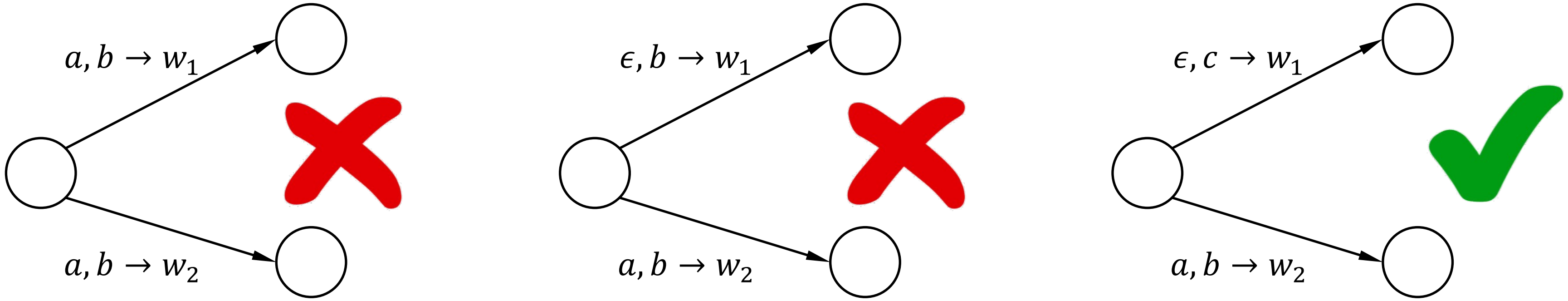
Deterministic PDA (DPDA)

- DFA/NFA ⊆ DPDA ⊆ NPDA



Deterministic PDA (DPDA)

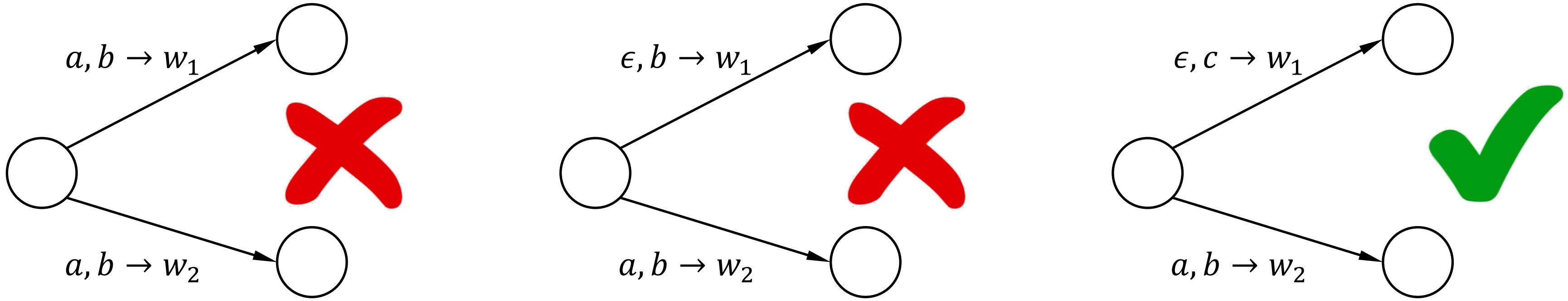
- DFA/NFA ⊆ DPDA ⊆ NPDA



- A CFL must have a NPDA, but may not have a DPDA

Deterministic PDA (DPDA)

- DFA/NFA ⊆ DPDA ⊆ NPDA



- A CFL must have a NPDA, but may not have a DPDA
- A DPDA language has a CFG without ambiguity

PART IV: Properties of CFL

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

南京大学编译技术
及软件安全研究组

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

$S_1 \to \cdots ; \cdots$

$S_2 \to \cdots ; \cdots$

$S \to S_1 \mid S_2$

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

$S_1 \to \cdots ; \cdots$

$S_2 \to \cdots ; \cdots$

$S \to S_1 S_2$

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - L_1^R

$S_1 \to \cdots ; \cdots$

$S \to S_1 S \mid \epsilon$

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - $L_1^R = \{w^R : w \in L_1\}$

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - $L_1^R = \{w^R : w \in L_1\}$

Given the CFG of L_1, (N, T, P, S), the grammar of L_1^R is (N, T, P^R, S), where

$$P^R = \{A \to \alpha^R : A \to \alpha \in P\}$$

南京大学编译技术及软件安全研究组

Closure Properties

- Given two CFL, L_1 and L_2, the following are CFL
  - $L_1 \cup L_2$
  - $L_1 L_2$
  - L_1^*
  - $L_1^R = \{w^R : w \in L_1\}$

  - $L_1 \cap L_2$????

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2$
  - $\overline{L_1}$
  - $L_1 - L_2$

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2$

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2$

Example: $L_1 = \{a^n b^n c^m\}$, $L_2 = \{a^n b^m c^m\}$

$L_1 \cap L_2 = \{a^n b^n c^n\}$ *(not CFL, let's prove later)*

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2$
  - $\overline{L_1}$

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

    if CFL is closed under complement, it is also closed under ∩
  - $\overline{L_1}$

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
  - $\overline{L_1}$
  - $L_1 - L_2$

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$
  - $\overline{L_1} = \Sigma^* - L_1$ if CFL is closed under $-$, it is also closed under complement
  - $L_1 - L_2$

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given a CFL L_1 and RL L_2, $L_1 \cap L_2$ is CFL
- **Proof:** Construct a new NPDA that simulates the PDA of L_1 and the DFA of L_2 in parallel

Intersection of CFL

Assume $\{a^n b^n c^n\}$ is not a CFL

- Given a CFL L_1 and RL L_2, $L_1 \cap L_2$ is CFL
- **Proof:** Construct a new NPDA that simulates the PDA of L_1 and the DFA of L_2 in parallel
- This is an exercise!

Application of the Properties

- Prove a language is a CFL or not

Application of the Properties

- Prove that $L = \{a^n b^n : n \neq 100, n \geq 0\}$ is a CFL

Application of the Properties

- Prove that $L = \{a^n b^n : n \neq 100, n \geq 0\}$ is a CFL
- **Proof:**
  - $L_1 = \{a^{100} b^{100}\}$ is regular

Application of the Properties

- Prove that $L = \{a^n b^n : n \neq 100, n \geq 0\}$ is a CFL
- **Proof:**
  - $L_1 = \{a^{100} b^{100}\}$ is regular
  - $\overline{L_1} = \{a, b\}^* - \{a^{100} b^{100}\}$ is regular

Application of the Properties

- Prove that $L = \{a^n b^n : n \neq 100, n \geq 0\}$ is a CFL
- **Proof:**
  - $L_1 = \{a^{100} b^{100}\}$ is regular
  - $\overline{L_1} = \{a, b\}^* - \{a^{100} b^{100}\}$ is regular
  - $L = \{a^n b^n\} \cap \overline{L_1}$ is context-free as we know $\{a^n b^n\}$ is context-free

Application of the Properties

- Prove that $L = \{w \in \{a,b,c\}^*: n_a = n_b = n_c\}$ is not a CFL

Application of the Properties

- Prove that $L = \{w \in \{a, b, c\}^*: n_a = n_b = n_c\}$ is not a CFL
- **Proof:**
  - Assume L is context-free
  - $L \cap \{a^* b^* c^*\}$ is context free as the former is CFL, the latter is RL

Application of the Properties

- Prove that $L = \{w \in \{a, b, c\}^* : n_a = n_b = n_c\}$ is not a CFL
- **Proof:**
  - Assume L is context-free
  - $L \cap \{a^*b^*c^*\}$ is context free as the former is CFL, the latter is RL
  - However, $L \cap \{a^*b^*c^*\} = \{a^n b^n c^n\}$, which, as we know, is not a CFL

Application of the Properties

- Prove that $L = \{w \in \{a, b, c\}^* : n_a = n_b = n_c\}$ is not a CFL
- **Proof:**
  - Assume L is context-free
  - $L \cap \{a^*b^*c^*\}$ is context free as the former is CFL, the latter is RL
  - However, $L \cap \{a^*b^*c^*\} = \{a^n b^n c^n\}$, which, as we know, is not a CFL
  - The assumption is wrong

Decidable Properties

- **Empty Language Question:**
  - Given a CFG, is the CFL empty?
- **Infinite Language Question:**
  - Given a CFG, is the CFL infinite?
- **Membership Question:**
  - Given a CFG, is a string belongs to the CFL?

Decidable Properties

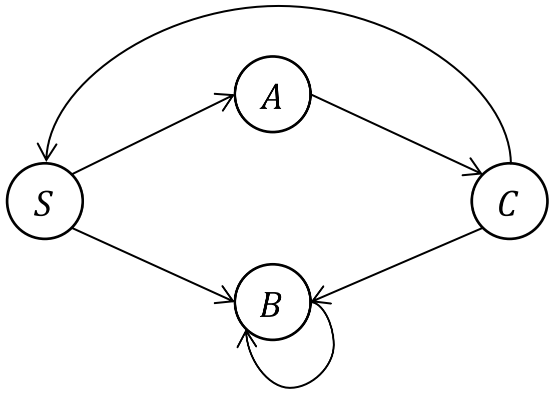
- **Empty Language Question:**
  - Given a CFG, is the CFL empty?

- **Algorithm**:
  - Check if the start non-terminal is not used, or useless, e.g., $S \to S$

Decidable Properties

- **Infinite Language Question:**
  - Given a CFG, is the CFL infinite?

$$S \to AB$$
$$A \to aCb \mid a$$
$$B \to bB \mid bb$$
$$C \to cBS$$



- **Algorithm**:
  - Remove useless non-terminals
  - Remove unit and epsilon productions
  - Create a dependency graph for remaining non-terminals
  - Check if the graph has a circle

Decidable Properties

- **Membership Question:**
  - Given a CFG, is a string belongs to the CFL?
- **Algorithm 1:**
  - Create an NPDA of the CFG
  - Check if the NPDA can accept the string
- **Algorithm 2:**
  - the CYK algorithm ($O(n^3)$)
  - https://en.wikipedia.org/wiki/CYK_algorithm

Decidable Properties

- **Membership Question:**
  - Given a CFG, is a string belongs to the CFL?
- **Algorithm 1:**
  - Create an NPDA of the CFG
  - Check if the NPDA can accept the string
- **Algorithm 2:**
  - the CYK algorithm ($O(n^3)$)
  - https://en.wikipedia.org/wiki/CYK_algorithm

We will come back to this (CFL-reachability) question when introducing the middle end of compilers

Undecidable Properties

- Check if a CFG is not ambiguous
- Check if a CFG has inherent ambiguity
- Check if the intersection of two CFLs is empty
- Check if two CFLs are equivalent
- Check if a CFL is equivalent to Σ^*

PART V: Pumping Lemma for CFL

Recap: Intersection of CFL

- Given two CFL, L_1 and L_2, the following may not be CFL
  - $L_1 \cap L_2$

Example: $L_1 = \{a^n b^n c^m\}$, $L_2 = \{a^n b^m c^m\}$

$L_1 \cap L_2 = \{a^n b^n c^n\}$ *(let's prove it)*

Chomsky Normal Form

- We can rewrite a CFG in many standard forms, e.g., CNF
- A CFG is in CNF if all its production rules are of the form
  - $A \rightarrow BC;\ \ A \rightarrow a;\ \ S \rightarrow \epsilon$

Chomsky Normal Form

- We can rewrite a CFG in many standard forms, e.g., CNF
- A CFG is in CNF if all its production rules are of the form
  - $A \to BC;\ \ A \to a;\ \ S \to \epsilon$
- A, B, and C are non-terminals and a is a terminal

Chomsky Normal Form

- We can rewrite a CFG in many standard forms, e.g., CNF
- A CFG is in CNF if all its production rules are of the form
  - $A \to BC;\ \ A \to a;\ \ S \to \epsilon$

- A, B, and C are non-terminals and a is a terminal
- S is the start symbol, B and C cannot be start symbol

Chomsky Normal Form

- We can rewrite a CFG in many standard forms, e.g., CNF
- A CFG is in CNF if all its production rules are of the form
  - $A \rightarrow BC;\ \ A \rightarrow a;\ \ S \rightarrow \epsilon$

- A, B, and C are non-terminals and a is a terminal
- S is the start symbol, B and C cannot be start symbol
- $S \rightarrow \epsilon$ can appear if ϵ is in the language

Chomsky Normal Form

- We can rewrite a CFG in many standard forms, e.g., CNF
- A CFG is in CNF if all its production rules are of the form
  - $A \rightarrow BC;\ \ A \rightarrow a;\ \ S \rightarrow \epsilon$

- A, B, and C are non-terminals and a is a terminal
- S is the start symbol, B and C cannot be start symbol
- $S \rightarrow \epsilon$ can appear if ϵ is in the language

- https://en.wikipedia.org/wiki/Chomsky_normal_form

Chomsky Normal Form

- We can rewrite a CFG in many standard forms, e.g., CNF
- A CFG is in CNF if all its production rules are of the form
  - $A \to BC;\ \ A \to a;\ \ S \to \epsilon$

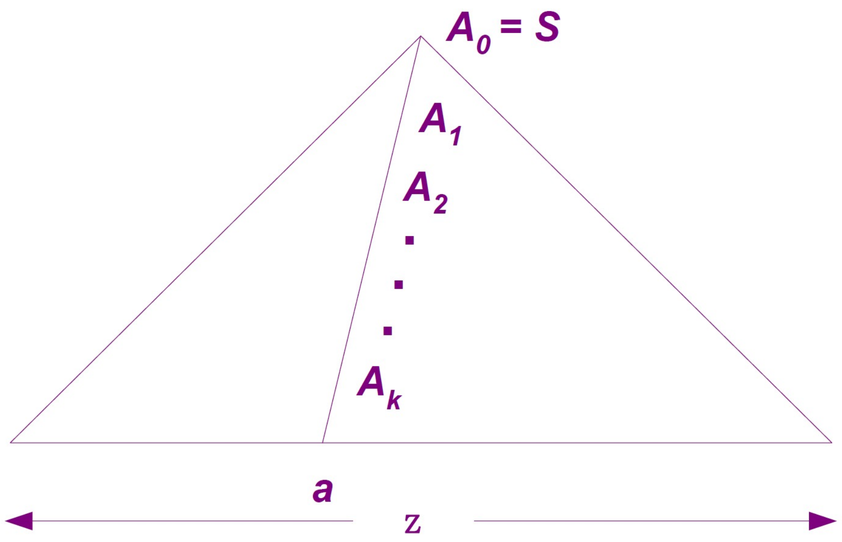
- ⇒ The parse tree is a binary tree (a tree node has ≤ 2 children)

Observation

- Assume L is a CFL without ϵ, its CFG (N, T, S, P) is in CNF
- Assume $|N| = m, n = 2^m$

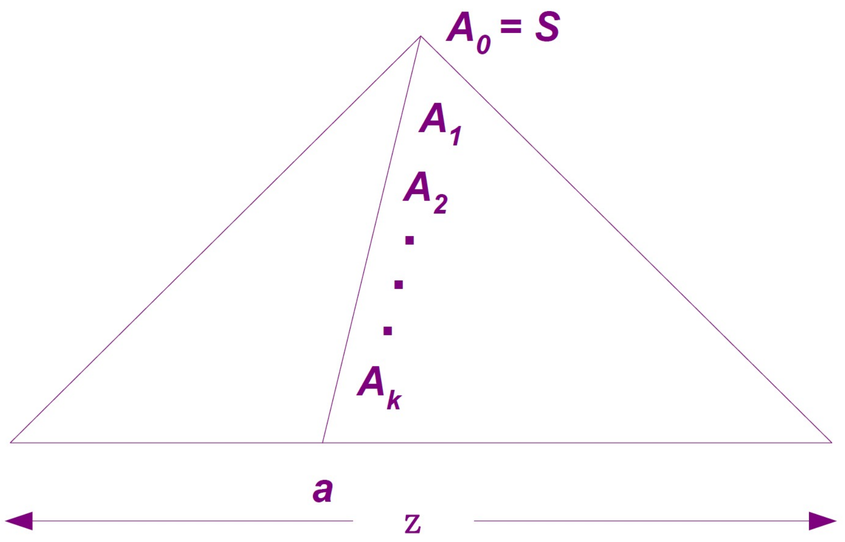
Observation

- Assume L is a CFL without ϵ, its CFG (N, T, S, P) is in CNF
- Assume $|N| = m, n = 2^m$
- For any string z ($|z| \geq n$), its parsing tree must be a binary tree and the number of leaves is $|z|$



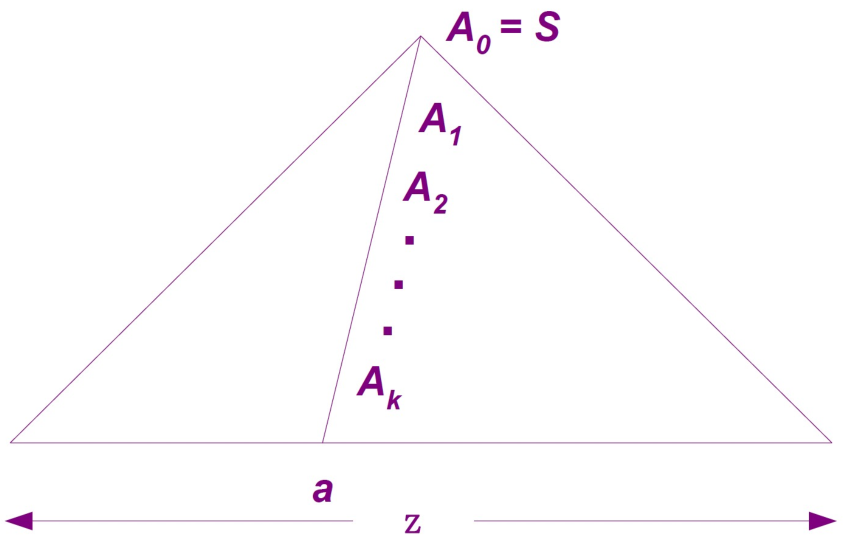
Observation

- Assume L is a CFL without ϵ, its CFG (N, T, S, P) is in CNF
- Assume $|N| = m, n = 2^m$
- For any string z $(|z| \geq n)$, its parsing tree must be a binary tree and the number of leaves is $|z|$
- Let the longest path be $A_0 A_1 \cdots A_k a$



Observation

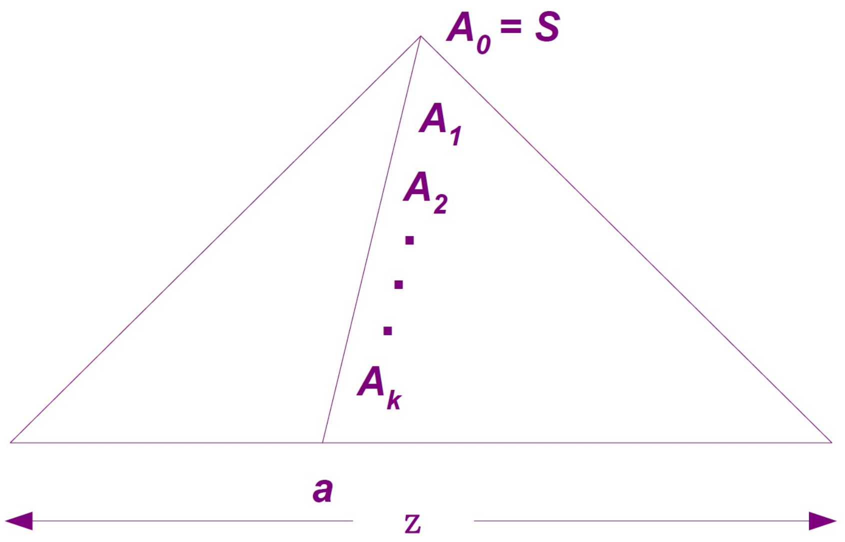
- Assume L is a CFL without ϵ, its CFG (N, T, S, P) is in CNF
- Assume $|N| = m, n = 2^m$
- For any string z ($|z| \geq n$), its parsing tree must be a binary tree and the number of leaves is $|z|$
- Let the longest path be $A_0 A_1 \cdots A_k a$
- $|z| \geq n = 2^m \Rightarrow k \geq m = |N|$



Observation

- Assume L is a CFL without ϵ, its CFG (N, T, S, P) is in CNF
- Assume $|N| = m, n = 2^m$
- For any string z ($|z| \geq n$), its parsing tree must be a binary tree and the number of leaves is $|z|$

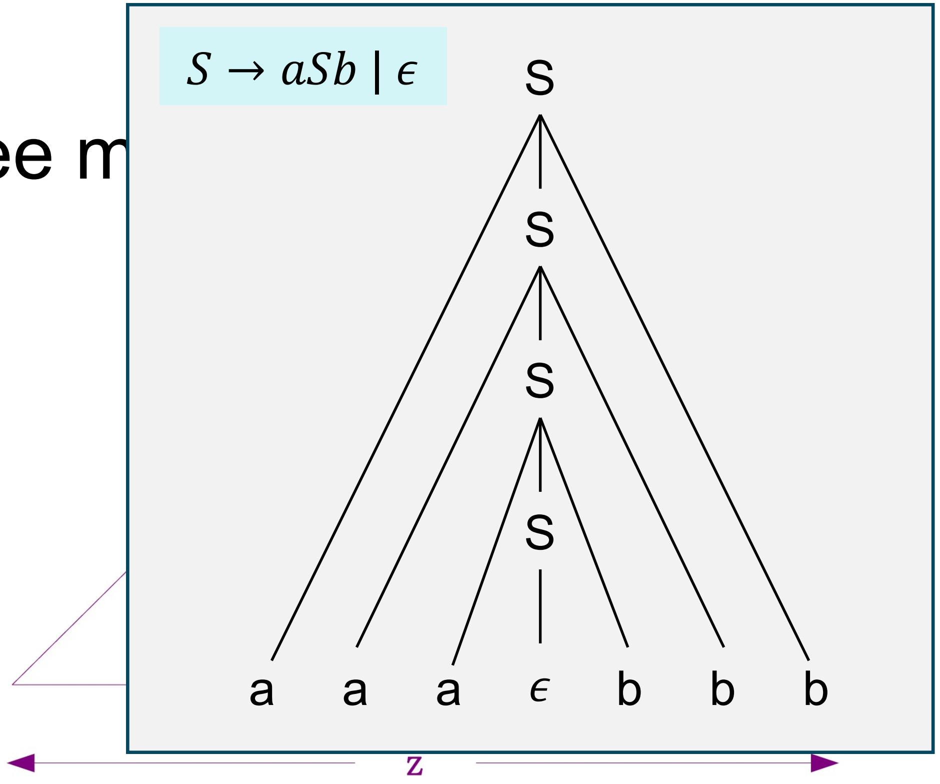
- Let the longest path be $A_0 A_1 \cdots A_k a$
- $|z| \geq n = 2^m \Rightarrow k \geq m = |N|$
- The path has repetitive non-terminals



Observation

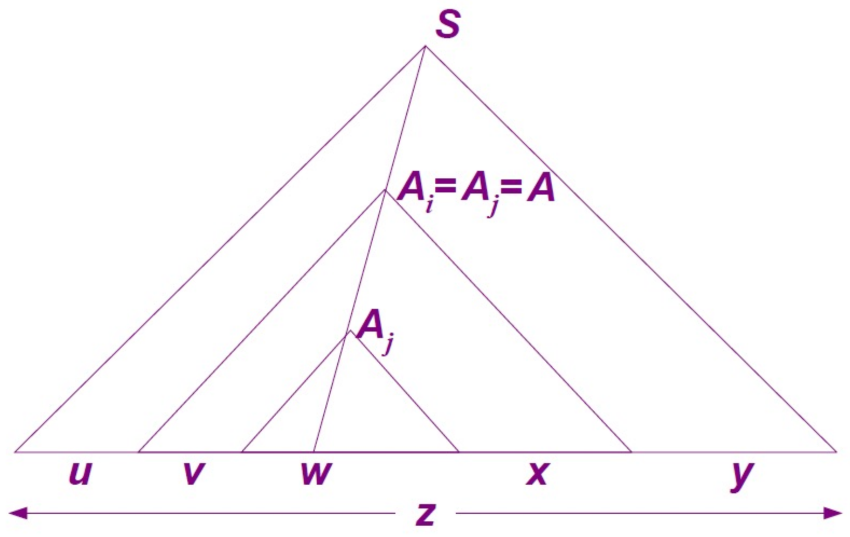
- Assume L is a CFL without ϵ, its CFG (N, T, S, P) is in CNF
- Assume $|N| = m, n = 2^m$
- For any string z ($|z| \geq n$), its parsing tree m and the number of leaves is $|z|$

- Let the longest path be $A_0 A_1 \cdots A_k a$
- $|z| \geq n = 2^m \Rightarrow k \geq m = |N|$
- The path has repetitive non-terminals



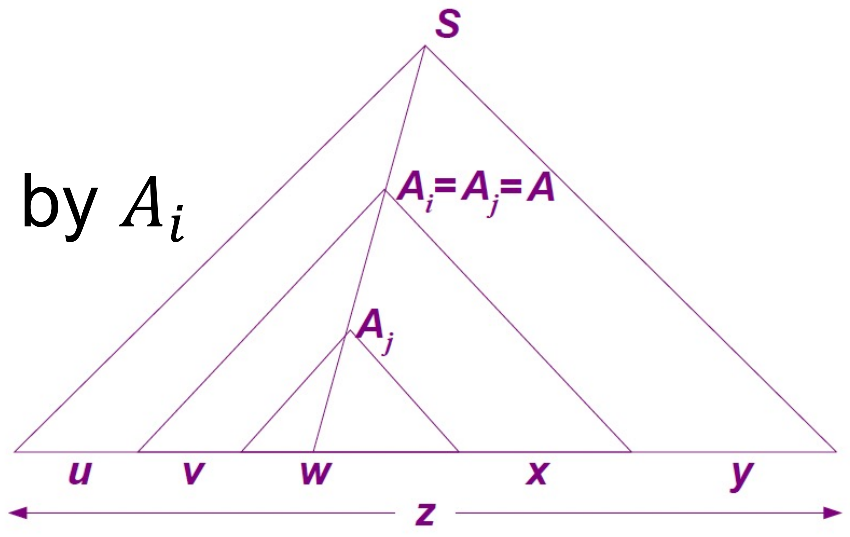
Observation

- Let the longest path be $A_0 A_1 \cdots A_k a$
- $|z| \geq n = 2^m \Rightarrow k \geq m = |N|$
- The path contains repetitive non-terminals:
  - Assume $A_i = A_j$ such that $k - m \leq i < j \leq k$



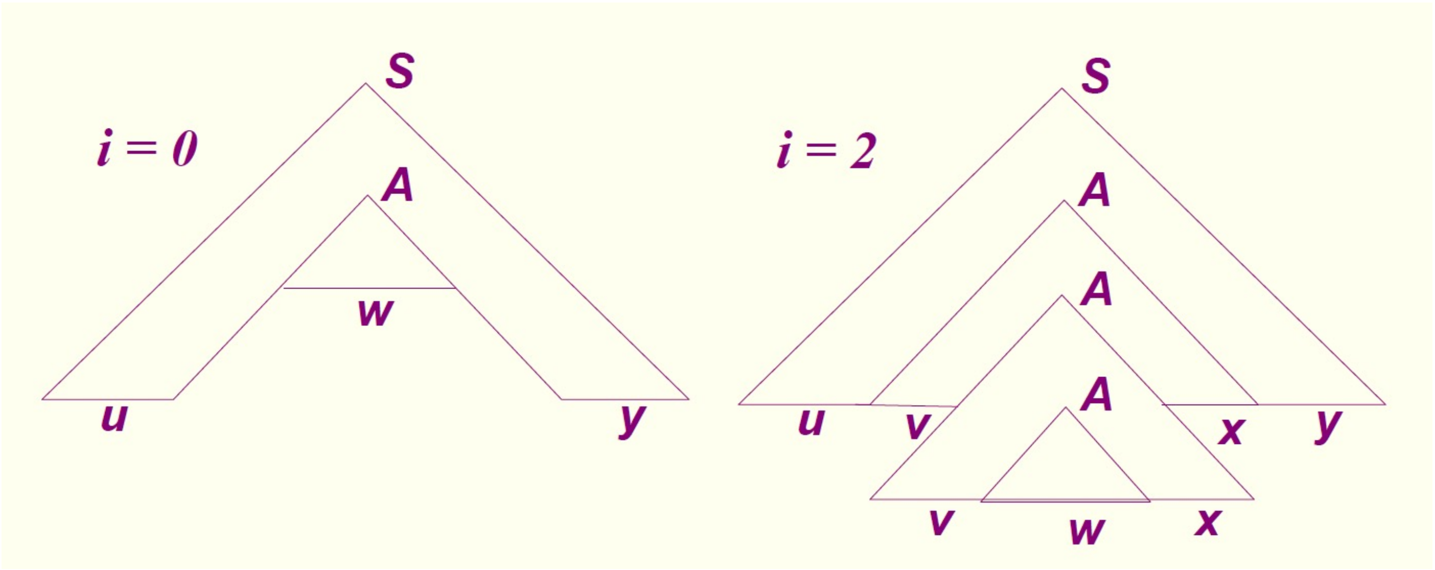
Observation

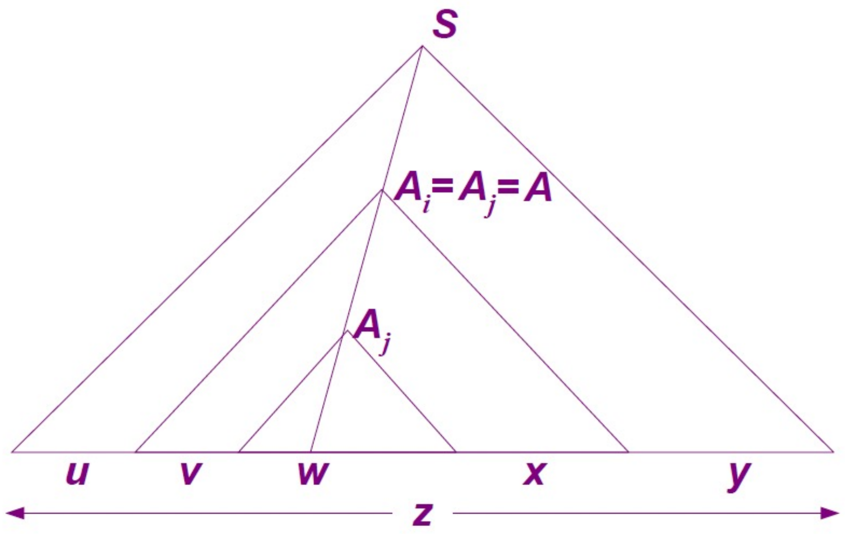
- Let the longest path be $A_0 A_1 \cdots A_k a$
- $|z| \geq n = 2^m \Rightarrow k \geq m = |N|$
- The path contains repetitive non-terminals:
  - Assume $A_i = A_j$ such that $k - m \leq i < j \leq k$
- Write $z = uvwxy$, where vwx is produced by A_i



Observation

- Write $z = uvwxy$, where vwx is produced by A_i
- Check $z_i = uv^iwx^iy$, which belongs to the CFL for any $i \geq 0$





Pumping Lemma for CFL

- Given a CFL L, there exists a constant n such that

Pumping Lemma for CFL

- Given a CFL L, there exists a constant n such that
- we have $z = uvwxy \in L, |z| \geq n$ satisfying the conditions below

Pumping Lemma for CFL

- Given a CFL L, there exists a constant n such that
- we have $z = uvwxy \in L, |z| \geq n$ satisfying the conditions below
  - $vx \neq \epsilon$

Pumping Lemma for CFL

- Given a CFL L, there exists a constant n such that
- we have $z = uvwxy \in L, |z| \geq n$ satisfying the conditions below
  - $vx \neq \epsilon$
  - $|vwx| \leq n$

Pumping Lemma for CFL

- Given a CFL L, there exists a constant n such that
- we have $z = uvwxy \in L, |z| \geq n$ satisfying the conditions below
  - $vx \neq \epsilon$
  - $|vwx| \leq n$
  - $\forall i \geq 0, z_i = uv^i wx^i y \in L$

Pumping Lemma for CFL

- Given a CFL L, there exists a constant n such that
- we have $z = uvwxy \in L, |z| \geq n$ satisfying the conditions below
  - $vx \neq \epsilon$
  - $|vwx| \leq n$
  - $\forall i \geq 0, z_i = uv^iwx^iy \in L$

- To prove a language is not a CFL, show that we cannot split a string in such a manner.

Proving $L = \{a^n b^n c^n\}$ is not a CFL

Proving $L = \{a^n b^n c^n\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^n b^n c^n = uvwxy$, where $|z| \geq n \land |vwx| \leq n$

Proving $L = \{a^n b^n c^n\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^n b^n c^n = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n$

- **Case 1**: vwx consists of only a, pumping v, x will increase $\#a$

Proving $L = \{a^n b^n c^n\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^n b^n c^n = uvwxy$, where $|z| \geq n \land |vwx| \leq n$

- **Case 1**: vwx consists of only a, pumping v, x will increase $\#a$
- **Case 2**: vwx consists of only b, pumping v, x will increase $\#b$

Proving $L = \{a^n b^n c^n\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^n b^n c^n = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n$

- **Case 1**: vwx consists of only a, pumping v, x will increase $\#a$
- **Case 2**: vwx consists of only b, pumping v, x will increase $\#b$
- **Case 3**: vwx consists of only c, pumping v, x will increase $\#c$

Proving $L = \{a^n b^n c^n\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^n b^n c^n = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n$

- **Case 1**: vwx consists of only a, pumping v, x will increase $\#a$
- **Case 2**: vwx consists of only b, pumping v, x will increase $\#b$
- **Case 3**: vwx consists of only c, pumping v, x will increase $\#c$
- **Case 4**: vwx consists of a, b, pumping v, x will increase $\#a, b$

Proving $L = \{a^n b^n c^n\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^n b^n c^n = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n$

- **Case 1**: vwx consists of only a, pumping v, x will increase $\#a$
- **Case 2**: vwx consists of only b, pumping v, x will increase $\#b$
- **Case 3**: vwx consists of only c, pumping v, x will increase $\#c$
- **Case 4**: vwx consists of a, b, pumping v, x will increase $\#a, b$
- **Case 5**: vwx consists of b, c, pumping v, x will increase $\#b, c$

Proving $L = \{a^{n^2}\}$ is not a CFL

Proving $L = \{a^{n^2}\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^{n^2} = uvwxy$, where $|z| \geq n \land |vwx| \leq n$

Proving $L = \{a^{n^2}\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^{n^2} = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n \wedge \mathbf{1} \leq |\boldsymbol{vx}| \leq \boldsymbol{n}$

Proving $L = \{a^{n^2}\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^{n^2} = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n \wedge \mathbf{1 \leq |vx| \leq n}$

- Let us pump vx twice, we have $z_2 = uv^2wx^2y$, where

Proving $L = \{a^{n^2}\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^{n^2} = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n \wedge \mathbf{1} \leq |\boldsymbol{vx}| \leq \boldsymbol{n}$

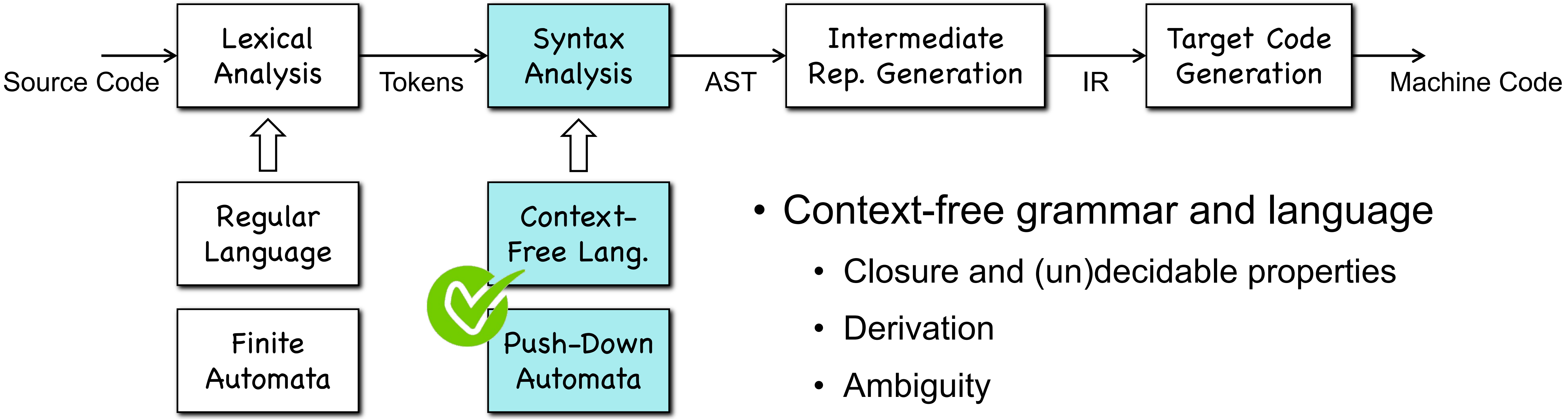
- Let us pump vx twice, we have $z_2 = uv^2wx^2y$, where
  - $|z_2| = n^2 + |vx|$

Proving $L = \{a^{n^2}\}$ is not a CFL

- Assume it is a CFL, we then can split is as
- $z = a^{n^2} = uvwxy$, where $|z| \geq n \wedge |vwx| \leq n \wedge \mathbf{1} \leq |\boldsymbol{vx}| \leq \boldsymbol{n}$

- Let us pump vx twice, we have $z_2 = uv^2wx^2y$, where
  - $|z_2| = n^2 + |vx|$
  - $n^2 + 1 \leq |z_2| \leq n^2 + n < (n+1)^2$

Summary



- Context-free grammar and language
  - Closure and (un)decidable properties
  - Derivation
  - Ambiguity
- PDA vs. NPDA vs. DPDA vs. CFL
- The Pumping Lemma for CFL

THANKS!