# Theory of Computation

*Unit 3: Context Free Grammars and NPDAs*

***Syedur Rahman***

***Lecturer, CSE Department***
***North South University***
***syedur.rahman@wolfson.oxon.org***

# *Theory of Computation Lecture Notes*

**Acknowledgements**

These lecture notes contain material from the following sources:

[Plump] D.J. Plump: *Theory of Computation*, Dept of Computer Science, University of York, 2003

[MOC] *Models of Computation*, Oxford University Computing Laboratory, University of Oxford, 2005.

[MCS] *Mathematics for Computer Scientists*, Dept of Computer Science, University of York, 2003

## Context Free Grammars

The ingredients of a CFG are ($V$, $\Sigma$, $S$, $P$) where:

- $V$ is a set of variables
- $\Sigma$ is a set of terminals or symbols
- $S$ is the start symbol which is a special variable i.e. $S \in V$
- $P$ is a set of productions/rules where each production is of the form:

$$\boldsymbol{A \rightarrow \beta}$$

where $A \in V$ and $\beta \in (V \cup \Sigma)^*$

## *An example of a CFG and derivations*

The CFG $(\{A, B\}, \{a, b\}, A, P)$ with P consisting of the following productions:

| | |
|---|---|
| $A \rightarrow aB$ | Which can also be written as |
| $B \rightarrow bB$ | $A \rightarrow aB$ |
| $B \rightarrow \varepsilon$ | $B \rightarrow bB \mid \varepsilon$ |

generates the language ab* through the following sequences called derivations:

$$A \Rightarrow aB \Rightarrow a\varepsilon \Rightarrow a$$

$$A \Rightarrow aB \Rightarrow abB \Rightarrow ab\varepsilon \Rightarrow ab$$

$$A \Rightarrow aB \Rightarrow abB \Rightarrow abbB \Rightarrow abb\varepsilon \Rightarrow abb$$

## *How to generate strings with a CFG*

1. Set $W$ to be the start symbol
2. Choose an occurrence of a variable $X$ in $W$ if any, otherwise stop.
3. Pick a production whose left hand side is $X$ and replace the chosen occurrence of $X$ in $W$ by the right hand side.
4. Go to step 2

We write $\Rightarrow^*$ as the reflexive transitive enclosure of $\Rightarrow$.
The language of the grammar, written $L(G)$ is:

$$\{ w \mid w \in \Sigma^* \land S \Rightarrow^* w \}$$

## *An example of a CFG*

The CFG ($\{S\}$, {a, b}, $S$, $P$) with P consisting of the following productions:

$S \rightarrow \varepsilon$

$S \rightarrow \mathrm{a}S\mathrm{b}$

generates the language $\mathrm{a}^n\mathrm{b}^n$ where $n \geq 0$

This is not a regular language but it can be generated by a context free grammar is therefore a context free language.

Regular languages can be considered as special types of context free languages, i.e. all regular languages are CF languages but not all CF languages are regular.

## *Another Example of a CFG*

Well-balanced parentheses are generated by the CFG:

$$(\{S\}, \{(, )\}, S, \{S\rightarrow(S), S\rightarrow SS, S\rightarrow\varepsilon\})$$

That is:

*S* is the start symbol and the only variable.

The terminals (or alphabet) include only ( and ).

The production rules include:

$$S\rightarrow(S)$$

$$S\rightarrow SS$$

$$S\rightarrow\varepsilon$$

$$\text{OR } S\rightarrow(S) \mid SS \mid \varepsilon$$

Examples: (), ()(()), ((())) but not ((), (())), ()(

# Regular Languages are Context Free

A language is a **context free language** (**CFL**) if it can be generated by a context free grammar. All regular languages are context free (e.g. a*b*) but not all context free languages are regular (e.g. $a^nb^n$)

A CFG is **right linear** if every rule is either of the form $R \rightarrow aT$ or of the form $R \rightarrow a$, where a is a terminal.

A language is regular if and only if it is generated by a right linear CFG.

This can be proved by showing how a DFA/NFA can be constructed for every right linear CFG, proving that languages generated by such CFGs are regular.

# *Derivations and Parse Trees*

Each derivation determines a parse tree, which is an ordered tree (i.e. the children at each node are ordered). The parse tree of a derivation shows how variables are replaced in the sequence while abstracting away from the order in which the replacements occur.

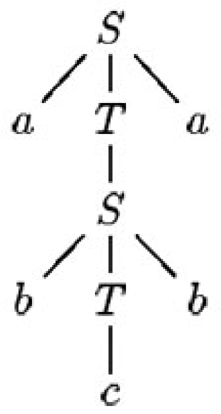
Palindromes over $\{a, b, c\}$: generated by $(\{S, T\}, \{a, b, c\}, \mathcal{R}, S)$ where $\mathcal{R}$ consists of eight rules:

$$\begin{array}{lcl} S & \rightarrow & aTa \mid bTb \mid cTc \mid T \\ T & \rightarrow & S \mid a \mid b \mid c \mid \epsilon \end{array}$$

derivation

$$\begin{array}{lcl} S & \Rightarrow & aTa \\ & \Rightarrow & aSa \\ & \Rightarrow & abTba \\ & \Rightarrow & abcba \end{array}$$

parse tree



From [MCS]

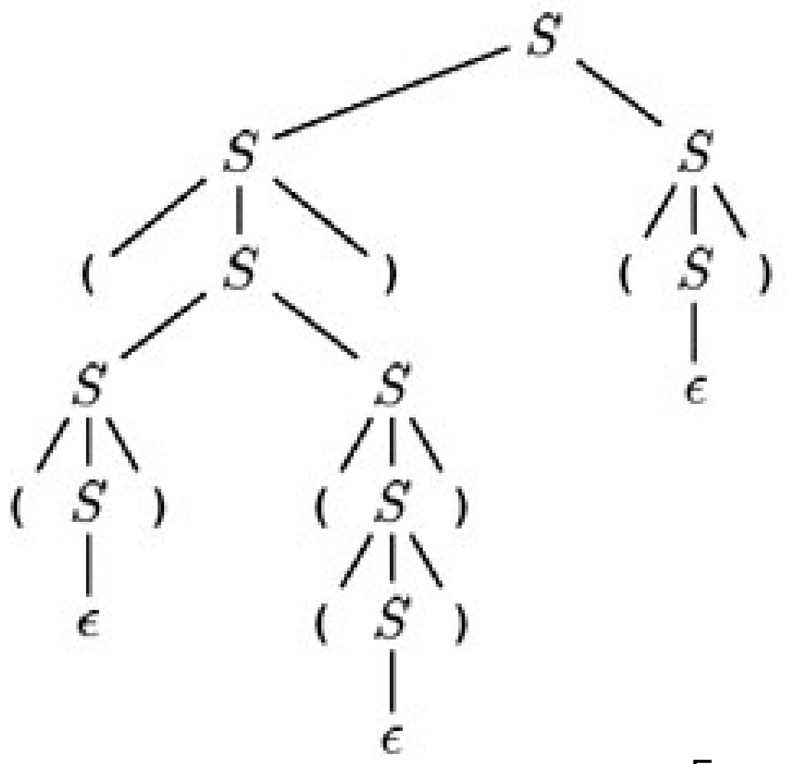
# More examples of Parse Trees

$$S \rightarrow (S) \mid SS \mid \varepsilon$$

derivation

$$\begin{aligned}
S &\Rightarrow \underline{S}\,S \\
&\Rightarrow \overline{(\underline{S})}\,S \\
&\Rightarrow (\overline{\underline{S}\,\underline{S}})\,S \\
&\Rightarrow^{2} (\overline{(\underline{S})}\,\overline{(\underline{S})})\,\underline{S} \\
&\Rightarrow^{3} ((\overline{\epsilon})\overline{((\underline{S}))})\overline{(\underline{S})} \\
&\Rightarrow^{2} ((\epsilon)((\overline{\epsilon})))(\overline{\epsilon}) \\
&= (()(()))()
\end{aligned}$$

parse tree



From [MCS]

# A small English language

| | | |
|---|---|---|
| ⟨sentence⟩ | → | ⟨noun-phrase⟩ ⟨verb-phrase⟩ |
| ⟨noun-phrase⟩ | → | ⟨cmplx-noun⟩ \| ⟨cmplx-noun⟩ ⟨prep-phrase⟩ |
| ⟨verb-phrase⟩ | → | ⟨cmplx-verb⟩ \| ⟨cmplx-verb⟩ ⟨prep-phrase⟩ |
| ⟨prep-phrase⟩ | → | ⟨prep⟩ ⟨cmplx-noun⟩ |
| ⟨cmplx-noun⟩ | → | ⟨article⟩ ⟨noun⟩ |
| ⟨cmplx-verb⟩ | → | ⟨verb⟩ \| ⟨verb⟩ ⟨noun-phrase⟩ |
| ⟨article⟩ | → | a \| the |
| ⟨noun⟩ | → | boy \| girl \| flower |
| ⟨verb⟩ | → | touches \| like \| see |
| ⟨prep⟩ | → | with |

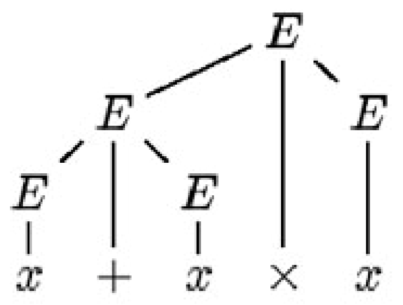
From [MCS]

## *Ambiguity in Derivations*

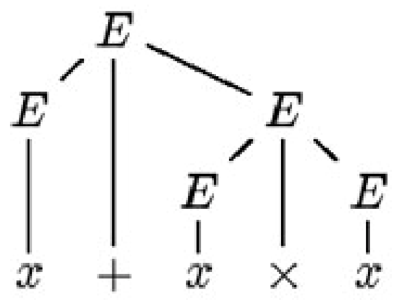
Two derivations are essentially different if they determine different parse trees. CFGs that contain strings with essentially different derivations are called ambiguous CFGs. An example:

$$E \rightarrow E + E \mid E \times E \mid x$$

$$\begin{aligned} E &\Rightarrow \underline{E} \times E \\ &\Rightarrow \underline{E} + E \times E \\ &\Rightarrow x + \underline{E} \times E \\ &\Rightarrow x + x \times \underline{E} \\ &\Rightarrow x + x \times x \end{aligned}$$



$$\begin{aligned} E &\Rightarrow \underline{E} + E \\ &\Rightarrow x + \underline{E} \\ &\Rightarrow x + \underline{E} \times E \\ &\Rightarrow x + x \times \underline{E} \\ &\Rightarrow x + x \times x \end{aligned}$$



From [MCS]

# *Leftmost derivations and Ambiguity*

A **leftmost derivation** is one in which at every step the leftmost occurring variable is chosen for replacement. E.g. Both the derivations for $x+x\text{x}x$ are leftmost derivations.

A CFG *G* is **ambiguous** if there is at least one word in L(*G*) which has two (or more) leftmost derivations (or parse trees). There is a 1-1 correspondence between parse trees and leftmost derivations.

Sometimes the language generated by an ambiguous grammar has an equivalent unambiguous grammar. Languages that can only be generated by ambiguous grammars are called **inherently ambiguous**.

## Chomsky Normal Form

A CFG is in Chomsky Normal Form if every rule in it is of the form:

$$A \rightarrow BC$$

$$A \rightarrow \mathrm{a}$$

where a is any terminal, and *A*, *B* and *C* are any variables except that *B* and *C* can not be the start variable. In addition we permit $S \rightarrow \varepsilon$ where *S* is the start symbol.

Theorem: Any CFL can be expressed by a CFG in Chomsky Normal Form.

# *Nondeterministic Pushdown Automata (NPDA)*

A *non-deterministic pushdown automaton* (NPDA) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ where $Q, \Sigma, \Gamma, \delta$ and $F$ are all finite sets, and

- $Q$ is the set of states
- $\Sigma$ is the *input alphabet*
- $\Gamma$ is the *stack alphabet*
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$ is the transition function
- $q_0 \in Q$ is the start state
- $\perp \in \Gamma$ is the initial stack symbol
- $F \subseteq Q$ is the set of accept states.

From [MOC]

A configuration of $M$ is an element of $Q \times \Sigma^* \times \Gamma^*$ describing (1) the current state, (2) the portion of the input yet unread (i.e. under and to the right of the input head) and (3) the current stack contents.

The *start configuration* is $(q_0, w, \perp)$. I.e. $M$ always starts in the start state with its input head scanning the leftmost input symbol and the stack containing only $\perp$.

The *next-configuration relation* $\rightarrow$ describes how $M$ moves from one configuration to another in one step. Formally

Formally we say that $M$ *accepts an input* $x$ *by final state* if for some $q \in F$ and $\gamma \in \Gamma^*$, we have $(q_0, x, \perp) \overset{*}{\rightarrow} (q, \epsilon, \gamma)$. Configurations of the form $(q, \epsilon, \gamma)$ where $q \in F$ and $\gamma \in \Gamma^*$ are called *accepting*.

There is another accepting convention:
$M$ *accepts an input* $x$ *by empty stack* if for some $q \in Q$,

$$(q_0, x, \perp) \overset{*}{\rightarrow} (q, \epsilon, \epsilon).$$

N.B. $F$ is irrelevant in the definition of acceptance by empty stack.

From [MOC]

# NPDA accepting balanced parentheses

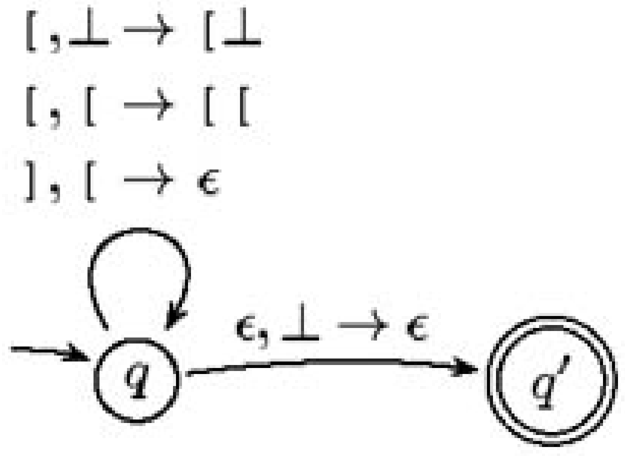
**Implementation-level description:**

$(\{q, q'\}, \{[, ]\}, \{\perp, [\}, \delta, q, \perp, \{q'\})$

where

$$\delta \begin{cases} (q, [, \perp) & \mapsto & \{(q, [\perp)\} \\ (q, [, [) & \mapsto & \{(q, [[)\} \\ (q, ], [) & \mapsto & \{(q, \epsilon)\} \\ (q, \epsilon, \perp) & \mapsto & \{(q', \epsilon)\} \end{cases}$$

**Transition diagram:**



From [MOC]

# *NPDA accepting Palindromes i.e. $ww^R$ where $w \in \{0, 1\}^*$ $w^R$ is the reverse of $w$*

**Implementation-level description:**

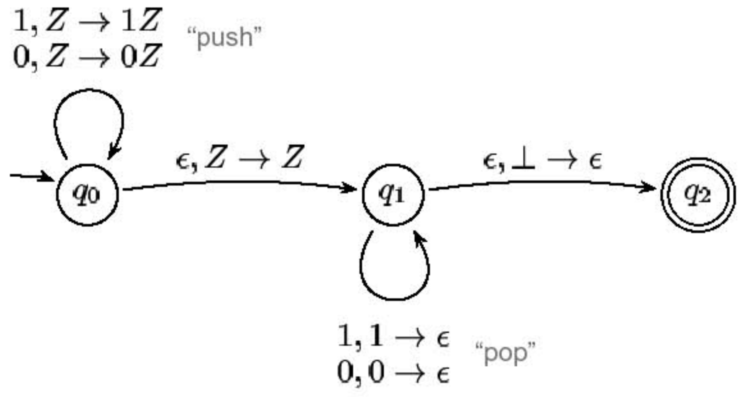
$$(\{q_1, q_0, q_2\}, \underbrace{\{0, 1\}}_{\Sigma}, \underbrace{\{0, 1, \perp\}}_{\Gamma}, \delta, q_0, \perp, \{q_2\})$$

where

$$\delta : \begin{cases} (q_0, 0, Z) & \mapsto & \{(q_0, 0Z)\} \\ (q_0, 1, Z) & \mapsto & \{(q_0, 1Z)\} \\ (q_0, \epsilon, Z) & \mapsto & \{(q_1, Z)\} \\ (q_1, 0, 0) & \mapsto & \{(q_1, \epsilon)\} \\ (q_1, 1, 1) & \mapsto & \{(q_1, \epsilon)\} \\ (q_1, \epsilon, \perp) & \mapsto & \{(q_2, \epsilon)\} \end{cases}$$

where $Z = 0, 1$.

**Transition graph:**



**A run accepting 011110:**

$$\begin{array}{lllll} & (q_0, & 011110, & \perp) & \\ \to & (q_0, & 11110, & 0\perp) & \\ \to & (q_0, & 1110, & 10\perp) & \\ \to & (q_0, & 110, & 110\perp) & \\ \to & (q_1, & 110, & 110\perp) & \\ \to & (q_1, & 10, & 10\perp) & \\ \to & (q_1, & 0, & 0\perp) & \\ \to & (q_1, & \epsilon, & \perp) & \\ \to & (q_2, & \epsilon, & \epsilon) & \end{array}$$

From [MCS]