CSE331 Automata and Computability

Lecture 13.1: Introduction to Context-free Grammar - 1

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn about the limitations of Regular Language and introduce you to a new class of language and its corresponding description method formally called the Context-free Grammar.

A limitation of Finite Automata

We know that every regular language has an equivalent finite automaton (e.g. DFA or NFA). They are called finite because they have **a very limited memory** in the form of states.

Though many interesting problems can be solved by Finite Automata, Finite Automata has a serious limitation –

> *A Finite Automaton does not have a large (infinite) memory. As a result, a finite automaton can only "count" (that is, maintain a counter, where different states correspond to different values of the counter) a finite number of input scenarios.*

For example, there is no finite automaton that recognizes the following languages:

- $L = \{w \mid w = 0^n 1^n, n \geq 1\}$ = The set of all binary strings consisting of a number of a's followed by equal number of **b**'s.
- $L = \{w \mid w = n_0(w) = n_1(w), n \geq 1\}$ = The set of all binary strings consisting of an equal number of 1's and 0's
- The set of strings over '(' and ')' that have "balanced" parentheses.

Context-free Grammar

A context-free Grammar (**CFG**) is an entirely different formalism for describing a class of languages called Context-free Language (**CFL**). It is more powerful than Regular expression and finite automaton in the sense that it can describe more languages than finite automaton.

- In a context free grammar, we express a language by a set of rules (called **production rules**). These rules describe how a string of the language can be generated.
- Consider the language, $L = \{w \mid w = 0^m 1^{2m};\ m \geq 0\}$. This denotes the set of all binary strings consisting of a number of a's followed by twice the number of b's as a. This language can be expressed by the following CFG:

  $$\boldsymbol{S \rightarrow 1S00}$$
  $$\boldsymbol{S \rightarrow \epsilon}$$

  Each expression of the above is called a **production rule.**

In the next lecture, we shall learn more about CFG.

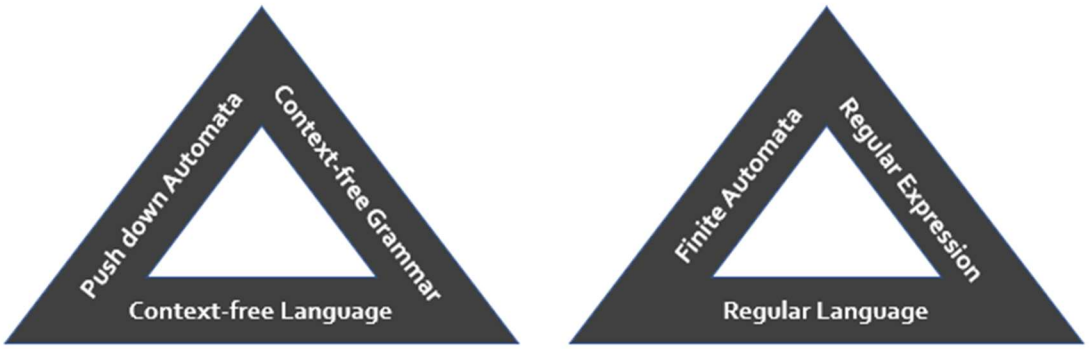
CSE331 Automata and Computability

Lecture 13.2: Introduction to Context-free Grammar - 2

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn about more about the Context-free Grammar and how they work.

- We know that any regular language can be described by a method of description called Regular Expression. Similarly, any context-free language (CFL) can be described by a method of description called Context-free Grammar (CFG).
- We know that for any regular language, there exists an abstract machine that recognizes that particular language called Finite Automaton (FA). Similarly, for any CFL, there exists an abstract machine that recognizes that particular language called Pushdown Automaton (PDA).
- The following triangles portray the above relations-



Context-free Grammar

In a context free grammar, we express a language by a set of rules (also called **productions**). These rules describe how a string of the language can be generated. This can be best illustrated using the following example. Suppose, the following production rules are given for a language:

$1. S \rightarrow 0S1$

$2. S \rightarrow \epsilon$

Let us show how to generate a string using the rules given. Start with first symbol S, then replace any symbol with the right-hand-side of a rule. Continue this until no symbol appears.

S	(Start with S)
$\rightarrow \mathbf{0S1}$	(Replace with $0S1$ using rule-1)
$\rightarrow 01$	(Replace S with ϵ using rule-2)

So, we get 01.
Let's try again.

S (Start with S)
$\rightarrow 0S1$ (Rule-1)
$\rightarrow 00S11$ (Rule-1 again)
$\rightarrow 0011$ (Rule-2)
This time we get, 0011.

Is there any pattern arising? Let's try for the last time.
S (Start with S)
$\rightarrow 0S1$ (Rule-1)
$\rightarrow 00S11$ (Rule-1 again)
$\rightarrow 000S111$ (Rule-1 once more)
$\rightarrow 000111$ (Rule-2)

Here we get 000111.

Now, it is obvious that the above grammar (CFG to be specific) is generating only the set of strings consisting of some 0's followed by equal number of 1's. So, the above two production rules i.e. the above CFG, generating the language, $L = \{w | w = 0^n 1^n, n \geq 0\}$. In previous lectures, we learnt that this language is not regular. So, it can't be described by either a regular expression or a finite automaton.

Context free grammar notations

We have already seen that a context free grammar is expressed by a set of rules. Beside the rules, there are some implicit notations that you need to understand:

- The symbols that appear on the left side of a rule are called **non-terminals** or **Variables**. The non-terminal symbol needs to be replaced. There can't be any non-terminal symbol in the final string because these symbols are not a member of the alphabet Σ.
- The symbols that appear only on the right side and never appear on the left side of a rule are called **terminal** symbols. The terminal symbols are never replaced as they are not present on the left side of a rule. The terminal symbols are members of Σ. The final string generated consists of only terminal symbols.
- The non-terminal symbol that appears on the left side of the first rule is called the **start symbol** of the grammar. The production of a string will start with this symbol (see our examples above).

CSE331 Automata and Computability

Lecture 14.1: CFG Definitions, Terminologies, Examples-1

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn the formal definition of Context-free Grammar and some examples to show you how to create CFGs for some basic problems.

Formal definition of a CFG

A CFG is formally described by a four-tuple, $G = \langle V, T, P, S \rangle$ where –

- V is the set of non-terminal symbols.
- T is the set of terminal symbols.
- P is the set of production rules
- S is the start symbol. Note, $S \in V$

Every rule is of the form $A \rightarrow (V \cup T)^*$ where $A \in V$.

Derivation: We usually say, S *derives a string* w if we can reach to w starting from S by a number of replacements or substitutions of production rules.

Shortcut notation for rule set: The rule set is usually described in a shortcut notation as follows.

General	Short Notation
$S \rightarrow 0S1$ $S \rightarrow \epsilon$	$S \rightarrow 0S1 \mid \epsilon$

In the shortcut notation, all production rules for a single non-terminal symbol are grouped and separated by a vertical bar symbol (|).

CFL as a Superset of RL

As Context-free Language is a superset of Regular Language, each Regular Expression must have an equivalent Context-free Grammar. It is sufficient to show that each of three basic regular operations can be achieved using CFG.

Basic Regular Operation	Regular Expression	Context-free Grammar	Optimized version
OR	$a \mid b$	$S \rightarrow A$ $S \rightarrow B$ $A \rightarrow a$ $B \rightarrow b$	$S \rightarrow A \mid B$ $A \rightarrow a$ $B \rightarrow b$
		$S \rightarrow a$ $S \rightarrow b$	$S \rightarrow a \mid b$

Concatenation	ab	$S \rightarrow AB$ $A \rightarrow a$ $B \rightarrow b$	-
		$S \rightarrow ab$	-
Kleene Star	a^*	$S \rightarrow AS$ $A \rightarrow a$ $S \rightarrow \epsilon$	$S \rightarrow AS \mid \epsilon$ $A \rightarrow a$
		$S \rightarrow aS$ $S \rightarrow \epsilon$	$S \rightarrow aS \mid \epsilon$
		$S \rightarrow SA$ $A \rightarrow a$ $S \rightarrow \epsilon$	$S \rightarrow SA \mid \epsilon$ $A \rightarrow a$
		$S \rightarrow SA$ $A \rightarrow a$ $S \rightarrow \epsilon$	$S \rightarrow Sa \mid \epsilon$

How $S \rightarrow aS|\epsilon$ works as a^*:

1. As a^* contains ϵ, we get $S \rightarrow \epsilon$
2. Let $S = a\boldsymbol{aaaa} \ldots \boldsymbol{aaa}$. The bold portion looks exactly the same as S. So, we can replace the the **bold** portion with an S (*See the figure*). So, we get $S \rightarrow aS$.
3. Combine these two we get $S \rightarrow aS|\epsilon$.

$$S = a\underbrace{aaaa \ldots aaa}_{S}$$
$$\therefore S = aS$$

Verify yourself to see how the above logic works after deriving some strings.

Some Examples of CFG

$$\{w|w = 0^n 1^n, n \geq 0\}$$

Language/Problem	CFG	Explanations
$\{w\|w = 0^n 1^n, n \geq 0\}$	$S \rightarrow 0S1$ $S \rightarrow \epsilon$	$S = 0\overbrace{00 \ldots 11}^{S}1$ $\therefore S = 0S1$
$\{w\|w = 0^n 1^{2n}, n \geq 0\}$	$S \rightarrow 0S11$ $S \rightarrow \epsilon$	Find out yourself

Grammar for valid mathematical Expressions

Operators to consider: $+ \; - \times \div \; (\;)$

Assume, all numbers are single digit integers.

Let E be the start symbol for the expression.

- A single digit number can be any of the 10 decimal digits yielding the grammar:

$$Num \rightarrow 0|1|2|3|4|5|6|7|8|9$$

- Note, any number is by default an expression. This yields $E \rightarrow Num$
- Any two expressions can be connected putting any binary operator between them. This yields the following-

$$E \rightarrow E + E$$
$$E \rightarrow E - E$$
$$E \rightarrow E \times E$$
$$E \rightarrow E \div E$$

- We can put parenthesis around an expression to make it a bundle to maintain precedence. This yields-

$$E \rightarrow (E)$$

- Combining,

$$E \rightarrow (E)$$
$$E \rightarrow E + E$$
$$E \rightarrow E - E$$
$$E \rightarrow E \times E$$
$$E \rightarrow E \div E$$
$$E \rightarrow Num$$
$$Num \rightarrow 0|1|2|3|4|5|6|7|8|9$$

Let's see how this grammar derives few expressions:

$4 \times (3 - 2)$	$4 \times 3 - 2$
$E \rightarrow E \times E$ $\rightarrow E \times (E)$ $\rightarrow E \times (E - E)$ $\rightarrow Num \times (Num - Num)$ $\rightarrow 4 \times (3 - 2)$	$E \rightarrow E \times E$ $\rightarrow E \times E - E$ $\rightarrow Num \times Num - Num$ $\rightarrow 4 \times 3 - 2$

Though the above grammar works for identifying correct mathematical expressions, interestingly it cannot evaluate the expressions correctly always. For example: the expression on the right $(4 \times 3 - 2)$ should evaluate to 10. But the way I derived in the above table will evaluate to 4! Can you find out the correct derivation to evaluate to 10?

CSE331 Automata and Computability

Lecture 14.2: CFG Definitions, Terminologies, Examples-2

Presenter: Azwad Anjum Islam (AAI) *Scribe: Mujtahid Al-Islam Akon (AKO)*

In this lecture, you will learn some more terminologies related to context-free grammar.

Recall, the following grammar you learned in the previous lecture for valid mathematical expressions:

$$E \to (E)$$
$$E \to E + E$$
$$E \to E - E$$
$$E \to E \times E$$
$$E \to E \div E$$
$$E \to Num$$
$$Num \to 0|1|2|3|4|5|6|7|8|9$$

Derivation, Sentential form and Sentence

You should already have a basic idea about the term ***derivation***. Recall, we usually say ***S** derives a string **w*** if we can reach to w starting from S by a number of replacements or substitutions. Let's see an alternative derivation of the expression $4 \times 3 - 2$:

$\mathbf{4 \times 3 - 2}$	**Derivation Step**
E	*Sentential Form*
$\Rightarrow E - E$	*Sentential Form*
$\Rightarrow E \times E - E$	*Sentential Form*
$\Rightarrow Num \times Num - Num$	*Sentential Form*
$\Rightarrow 4 \times 3 - 2$	*Sentential Form* *Sentence*

In the above Example we say:

- E derives the string $\mathbf{4 \times 3 - 2}$ and we express it as $E \overset{*}{\Rightarrow} 4 \times 3 - 2$.
- A ***sentential form*** is any string derivable from the start symbol. Thus, in the derivation of $4 \times 3 - 2$; each intermediate line i.e. $E - E$, $E \times E - E$, $Num \times Num - Num$ etc. are all sentential forms along with E and $4 \times 3 - 2$ themselves.
- A ***sentence*** is a sentential form consisting only of terminals such as $4 \times 3 - 2$ in the above example.

Summary

We can express the derivation of w from a start symbol S as $S \Rightarrow u_1 \Rightarrow u_2 \ldots \Rightarrow u_{n-1} \Rightarrow w$ or simply $S \overset{*}{\Rightarrow} w$, where $S, u_1, u_2 \ldots, u_{n-1}, w$ each of this is a sentential form and $\boldsymbol{w}$ is the sentence.

Now, we can define the language of a CFG formally as $L = \{w \in \Sigma^{\star} | \, S \overset{*}{\Rightarrow} w\}$.

CSE331 Automata and Computability

Lecture 15, 16: Leftmost/Rightmost Derivation And Parse Tree

Presenter: Azwad Anjum Islam *Scribe: Warida Rashid*

A derivation tree or a parse tree of a string given a Context Free Grammar (CFG) is a graphical representation of how a string can be generated from the grammar and the semantic information (More on this will be discussed in Compiler Design).

Example

E → E+E

E → E-E

E → E*E

E → E/E

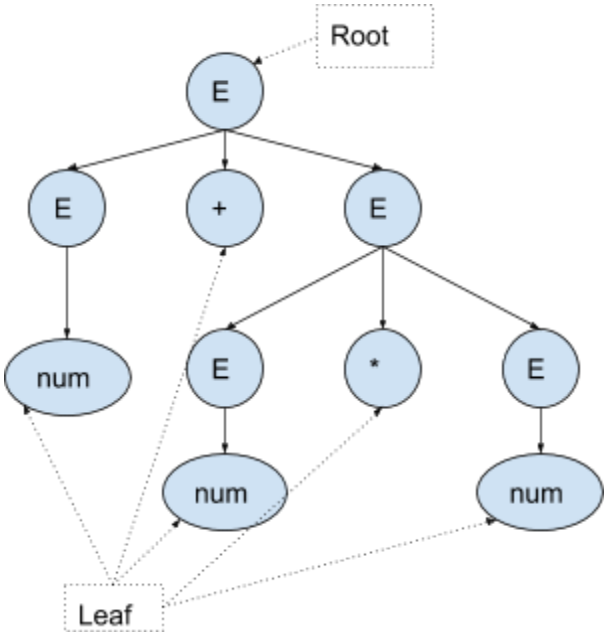
E → (E)

E → num

String derivation:

Consider the string **num +num * num**

To derive a string, we start from the designated start symbol and keep expanding it until the string is generated. The parse tree for the string would be:



Representation Technique:

Root Vertex: Must be labeled by the start symbol.

Vertex: Labeled by the non-terminal symbol/variables.

Leaves: Labeled by a terminal symbol or ε.

Leftmost Derivation:

A leftmost derivation is obtained by taking the production of the leftmost variable in each step.

The leftmost derivation of the example string is as follows:

E → E+E

E → num+E

E → num+E*E

E → num+num*E

E → num+num*num

Rightmost Derivation:

A rigthmost derivation is obtained by taking the production of the rightmost variable in each step.

The leftmost derivation of the example string is as follows:

E → E+E

E → E+E*E

E → E+E*num

E → E+num*num

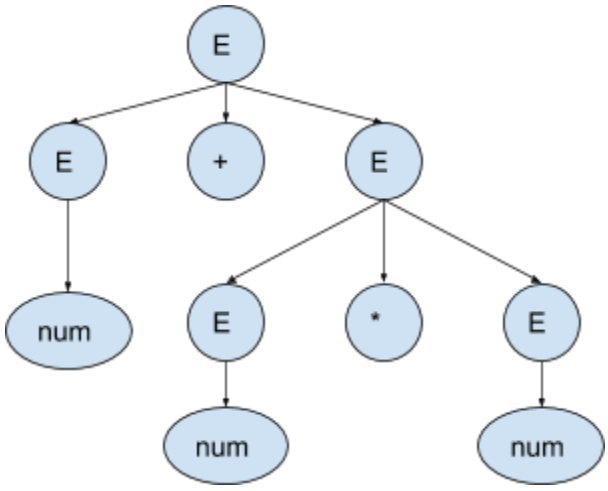
E → num+num*num

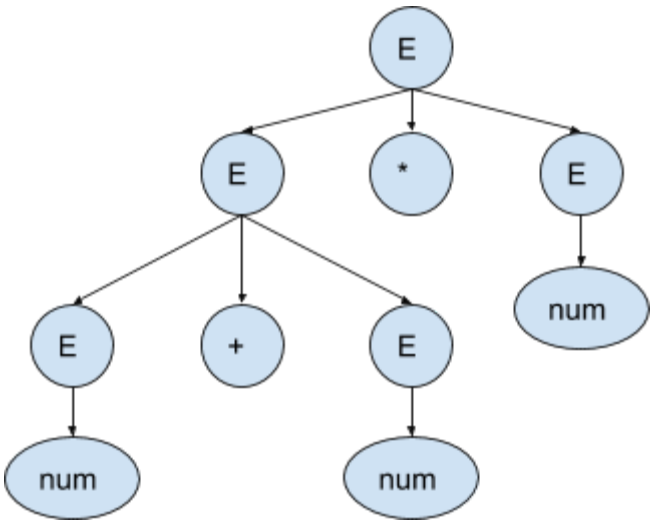
Ambiguity:

A Context Free Grammar is said to be ambiguous, if for some string w that belongs to the language of the grammar there exists multiple parse trees or multiple leftmost derivations or multiple rightmost derivations.

Example 1:

The Grammar in the example is ambiguous. The string **num+num*num** have two different leftmost derivations:

1.	E → E+E E → num+E E → num+E*E E → num+num*E E → num+num*num	 Fig. Parse tree from the leftmost derivation

2.	E → E*E E → E+E*E E → E+E*E E → num+E*E E → num+num*E E → num+num*num		 Fig. Parse tree from the rightmost derivation

Example 2:

The following grammar generates balanced parenthesis:

S → SS

S → (S)

S → ()

Two leftmost derivation of the string **()()():**

1. S → SS
   S → ()S
   S → ()SS
   S → ()()S
   S → ()()()
2. S → SS
   S → SSS
   S → ()SS
   S → ()()S
   S → ()()()

We can design a grammar for the same language (Balanced parenthesis) that is not ambiguous.

$S \rightarrow (RS$

$S \rightarrow \varepsilon$

$S \rightarrow RR$

$R \rightarrow)$

Therefore, **"Ambiguity"** is a property of the grammar, not the language itself.

Inherent Ambiguity:

inherently ambiguous language A context-free language that has no unambiguous grammar. The languages that can only be defined with an ambiguous grammar is said to be inherently ambiguous.

Example:

$L = \{0^i1^j2^k$, where i = j or j = k and i,j,k > 0$\}$

This language can only be defined with an ambiguous grammar:

$S \rightarrow AB \mid CD$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

$C \rightarrow 0C \mid 0$

$D \rightarrow 1D2 \mid 12$

For example, the string 001122, can be derived by taking either of the two productions of the start symbol S.

Limitations

1. **Unaware of the Context**

   A Context Free Grammar, as implied by the name, is unaware of the context.

   A simplified version of English language

   For example, The following image contains a CFG for a simplified English Language and the parse tree for the string "the man read this book".

S → NP VP
S → Aux NP VP
S → VP
NP → Det NOM
NOM → Noun
NOM → Noun NOM
VP → Verb
VP → Verb NP

Det → *that* | *this* | *a* | *the*
Noun → *book* | *flight* | *meal* | *man*
Verb → *book* | *include* | *read*
Aux → *does*

S → NP VP
→ Det NOM VP
→ *The* NOM VP
→ *The* Noun VP
→ *The man* VP
→ *The man* Verb NP
→ *The man read* NP
→ *The man read* Det NOM
→ *The man read this* NOM
→ *The man read this* Noun
→ *The man read this book*

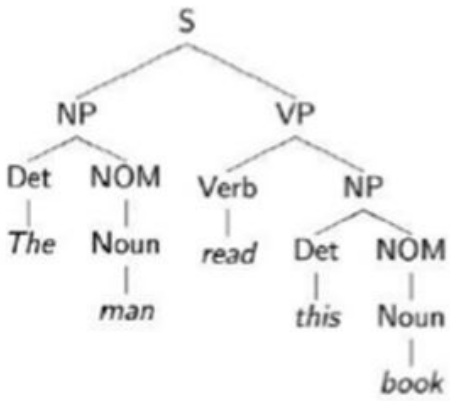


Figure: Parse Tree

However, this grammar can also generate strings like “This book man read that flight”. Because, any production rule in the grammar can be applied at any point regardless of the context and the subparts of the tree cannot communicate among themselves.

2. Limited Memory

A CFG has limited memory. It can match two things and no more than that.