Parsing and Ambiguity

Raju Hazari

Department of Computer Science and Engineering
National Institute of Technology Calicut

November 18-19, 2020

Parsing

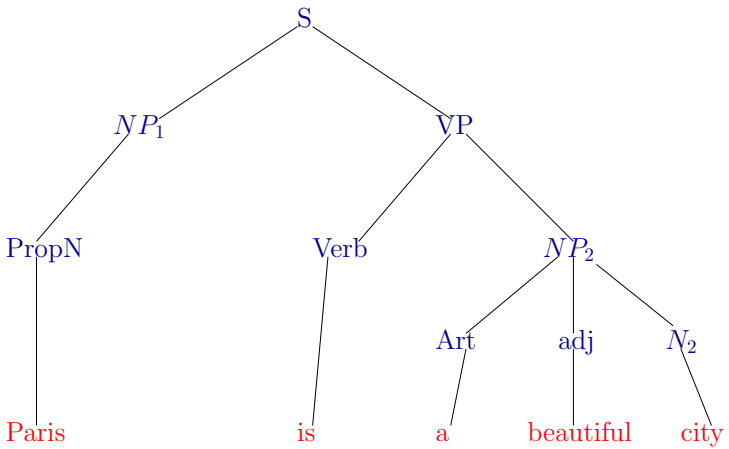
- Explaining a sentence through its grammatical derivation is familiar to most of us from a study of natural languages and is called **parsing**.
- Parsing is a way of describing sentence structure.
- It is important whenever we need to understand the meaning of a sentence, as we do for instance in translating from one language to another.
- In computer science, this is relevant in interpreters, compilers, and other translating programs.

Parsing

- Let us take a sentence " **Paris is a beautiful city** " and see how they can be parsed.

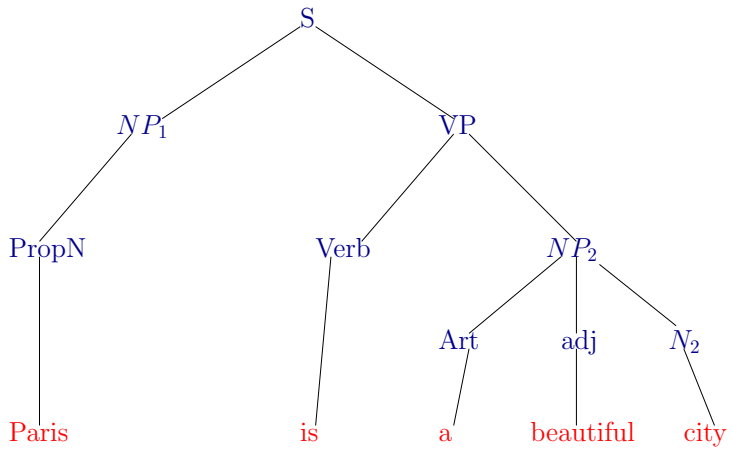
Parsing

- Let us take a sentence " **Paris is a beautiful city** " and see how they can be parsed.



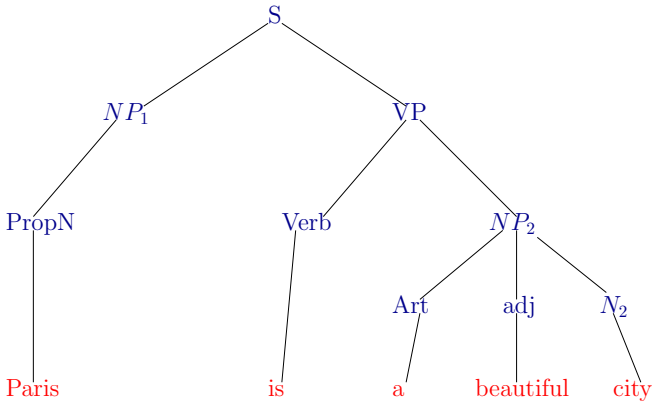
Parsing

- Let us take a sentence " **Paris is a beautiful city** " and see how they can be parsed.



- The parse tree are the syntax tree tree for the grammar.
- We also call it as a derivation tree.
  - If we go from sentence symbol to the sentence, it is a derivation.
  - If we go from the sentence to the sentence symbol, trying to construct the tree in the bottom up manner, it is parsing.

Parsing



- Let us look at the rules here:
  - $\langle S \rangle \rightarrow \langle NP_1 \rangle \langle VP \rangle$
  - $\langle NP_1 \rangle \rightarrow \langle PropN \rangle$
  - $\langle PropN \rangle \rightarrow$ Paris
  - $\langle VP \rangle \rightarrow \langle Verb \rangle \langle NP_2 \rangle$
  - $\langle Verb \rangle \rightarrow$ is
  - $\langle NP_2 \rangle \rightarrow \langle Art \rangle \langle adj \rangle \langle N_2 \rangle$
  - $\langle Art \rangle \rightarrow$ a
  - $\langle adj \rangle \rightarrow$ beautiful
  - $\langle N_2 \rangle \rightarrow$ city

Parsing

- We have so far concentrated on the generative aspect of grammars.
  - Given a grammar G, we studied the set of strings that can be derived using G.
- In cases of practical applications, we are also concerned with the analytical side of the grammar :
  - Given a string w of terminals, we want to know whether or not w is in $L(G)$.

Parsing

- We have so far concentrated on the generative aspect of grammars.
  - Given a grammar G, we studied the set of strings that can be derived using G.
- In cases of practical applications, we are also concerned with the analytical side of the grammar :
  - Given a string w of terminals, we want to know whether or not w is in $L(G)$.
    - ★ If so, we may want to find a derivation of w.

Parsing

- We have so far concentrated on the generative aspect of grammars.
  - Given a grammar G, we studied the set of strings that can be derived using G.
- In cases of practical applications, we are also concerned with the analytical side of the grammar :
  - Given a string w of terminals, we want to know whether or not w is in $L(G)$.
    - ⋆ If so, we may want to find a derivation of w.
  - An algorithm that can tell us whether w is in $L(G)$ is a membership algorithm.
  - The term **parsing** describes finding a sequence of productions by which a $w \in L(G)$ is derived.

Parsing and Membership

Given a string w, we want to know whether or not w is in L(G)

- We systematically construct all possible (say, leftmost) derivations and see whether any of them match w.
- Specifically, we start at round one by looking at all productions of the form $S \to x$, finding all x that can be derived from S in one step.
- If none of these results in a match with w, we go to the next round, in which we apply all applicable productions to the leftmost variable of every x.
- This gives us a set of sentential forms, some of them possibly leading to w.
- On each subsequent round, we again take all leftmost variables and apply all possible productions.

Parsing and Membership

- It may be that some of these sentential forms can be rejected on the grounds that w can never be derived from them, but in general, we will have on each round a set of possible sentential forms.
- After the first round, we have sentential forms that can be derived by applying a single production, after the second round we have the sentential forms that can be derived in two steps, and so on.
- If $w \in L(G)$, then it must have a leftmost derivation of finite length.
- Thus, the method will eventually give a leftmost derivation of w.
- We will call this **exhaustive search parsing** or **brute force parsing**.
- It is a form of **top-down parsing**, which we can view as the construction of a derivation tree from the root down.

Parsing and Membership

- **Example:** Consider the grammar $G : S \rightarrow SS|aSb|bSa|\epsilon$ and the string $w = aabb$.

Parsing and Membership

- **Example:** Consider the grammar $G : S \to SS|aSb|bSa|\epsilon$ and the string $w = aabb$.

  Round one gives us

  1. $S \Rightarrow SS$,
  2. $S \Rightarrow aSb$,
  3. $S \Rightarrow bSa$,
  4. $S \Rightarrow \epsilon$

Parsing and Membership

- **Example:** Consider the grammar $G : S \to SS|aSb|bSa|\epsilon$ and the string $w = aabb$.

  Round one gives us

  1. $S \Rightarrow SS$,
  2. $S \Rightarrow aSb$,
  3. $S \Rightarrow bSa$,
  4. $S \Rightarrow \epsilon$

  The last two of these can be removed from further consideration for obvious reasons. Round two the yields sentential forms

  1. $S \Rightarrow SS \Rightarrow SSS$,
  2. $S \Rightarrow SS \Rightarrow aSbS$,
  3. $S \Rightarrow SS \Rightarrow bSaS$,
  4. $S \Rightarrow SS \Rightarrow S$

  which are obtained by replacing the leftmost S in sentenmtial form **1** with all applicable substitutes.

Parsing and Membership

Similarly, from sentential form **2** we get the additional sentential forms.

1. $S \Rightarrow aSb \Rightarrow aSSb$,
2. $S \Rightarrow aSb \Rightarrow aaSbb$,
3. $S \Rightarrow aSb \Rightarrow abSab$,
4. $S \Rightarrow aSb \Rightarrow ab$

Parsing and Membership

Similarly, from sentential form **2** we get the additional sentential forms.

1. $S \Rightarrow aSb \Rightarrow aSSb$,
2. $S \Rightarrow aSb \Rightarrow aaSbb$,
3. $S \Rightarrow aSb \Rightarrow abSab$,
4. $S \Rightarrow aSb \Rightarrow ab$

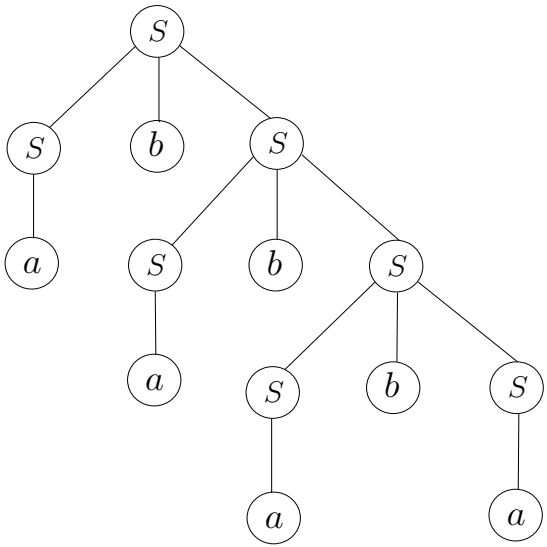
Again, several of these can be removed from contention. On the next round, we find the actual target string from the sequence

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

Therefore, $aabb$ is in the language generated by the grammar under consideration.

Ambiguity in CFG

- Consider the grammar G : 1. $S \to SbS$.
  2. $S \to a$
- Let us consider the string $abababa$. Now, a derivation tree we can have like this–



Ambiguity in CFG

- Now, We can write a derivation which is leftmost, and another one which is not leftmost.

Leftmost

$S \overset{1}{\Rightarrow} SbS$
$S \overset{2}{\Rightarrow} abS$
$S \overset{1}{\Rightarrow} abSbS$
$S \overset{2}{\Rightarrow} ababS$
$S \overset{1}{\Rightarrow} ababSbS$
$S \overset{2}{\Rightarrow} abababS$
$S \overset{2}{\Rightarrow} abababa$

Not Leftmost

$S \overset{1}{\Rightarrow} SbS$
$S \overset{1}{\Rightarrow} SbSbS$
$S \overset{1}{\Rightarrow} SbSbSbS$
$S \overset{2}{\Rightarrow} abSbSbS$
$S \overset{2}{\Rightarrow} abSbabS$
$S \overset{2}{\Rightarrow} abababS$
$S \overset{2}{\Rightarrow} abababa$

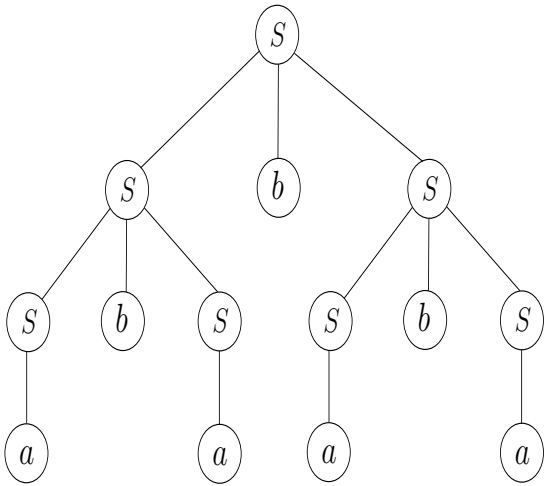
- So, you find that for this derivation tree, there is a leftmost derivation and you have many derivation which are not leftmost.
- So, the correspondence between derivation trees and derivation is not one to one.

Ambiguity in CFG

- But, for one derivation tree there will be only one leftmost derivation, and for one leftmost derivation there will be only one derivation tree.
- The correspondence between derivation trees and leftmost derivation is a bijection. But, the correspondence between derivation trees and all derivations it is not bijection.

Ambiguity in CFG

- But, for one derivation tree there will be only one leftmost derivation, and for one leftmost derivation there will be only one derivation tree.
- The correspondence between derivation trees and leftmost derivation is a bijection. But, the correspondence between derivation trees and all derivations it is not bijection.
- Now, let draw another derivation tree



Ambiguity in CFG

- The same string *abababa* is derived., but the derivation trees are different. The leftmost derivation for this tree–

$$
\begin{aligned}
S &\overset{1}{\Rightarrow} SbS \\
S &\overset{1}{\Rightarrow} SbSbS \\
S &\overset{2}{\Rightarrow} abSbS \\
S &\overset{2}{\Rightarrow} ababS \\
S &\overset{1}{\Rightarrow} ababSbS \\
S &\overset{2}{\Rightarrow} abababS \\
S &\overset{2}{\Rightarrow} abababa
\end{aligned}
$$

- Two derivation trees generates the same string *abababa*.
- So, for a derivation tree you may have one leftmost derivation, one rightmost derivation and several derivations which are neither leftmost nor rightmost.

Ambiguity in CFG

- Sometimes a grammar can generates the same string in several different ways.
- Such a string will have several different parse trees and thus several different meaning.
- This result may be undesirable for certain application, such as programming languages, where a program should have a unique interpretation.
- If a grammar generates the same string in several different ways, we say that the string is derived *ambiguously* in that grammar.
- If a grammar generates some string ambiguously, we say that the grammar is *ambiguous*.

Ambiguity in CFG

- **Example :** Consider the grammar $G = (N, T, E, P)$ with $N = \{E, I\}$, $T = \{a, b, c, +, *\}$, and productions

$$E \to E + E,$$
$$E \to E * E,$$
$$E \to I,$$
$$I \to a|b|c$$

Ambiguity in CFG

- **Example :** Consider the grammar $G = (N, T, E, P)$ with $N = \{E, I\}$, $T = \{a, b, c, +, *\}$, and productions

$$E \to E + E,$$
$$E \to E * E,$$
$$E \to I,$$
$$I \to a|b|c$$

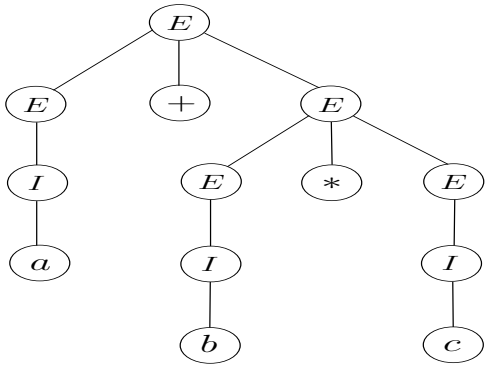
- The string $a + b * c$ is in $L(G)$.

Ambiguity in CFG

- **Example :** Consider the grammar $G = (N, T, E, P)$ with $N = \{E, I\}$, $T = \{a, b, c, +, *\}$, and productions

$$E \to E + E,$$
$$E \to E * E,$$
$$E \to I,$$
$$I \to a|b|c$$

- The string $a + b * c$ is in $L(G)$.

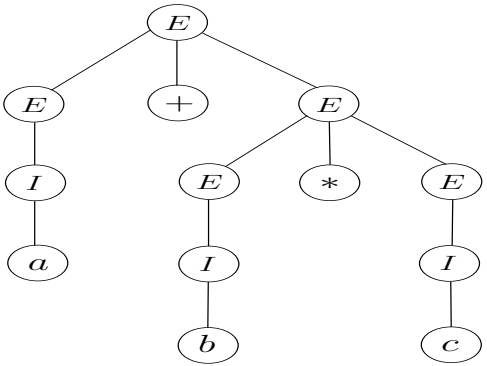


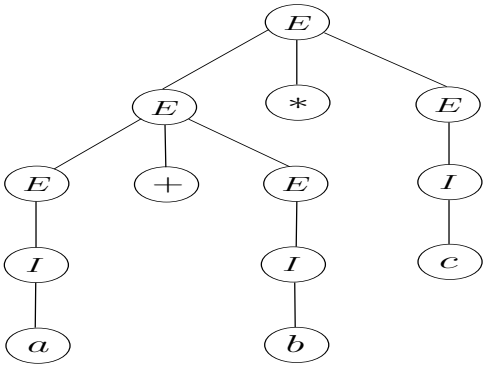
Ambiguity in CFG

- **Example :** Consider the grammar $G = (N, T, E, P)$ with $N = \{E, I\}$, $T = \{a, b, c, +, *\}$, and productions

$$E \to E + E,$$
$$E \to E * E,$$
$$E \to I,$$
$$I \to a|b|c$$

- The string $a + b * c$ is in $L(G)$.



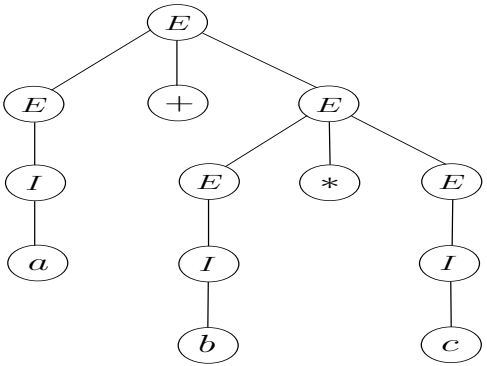


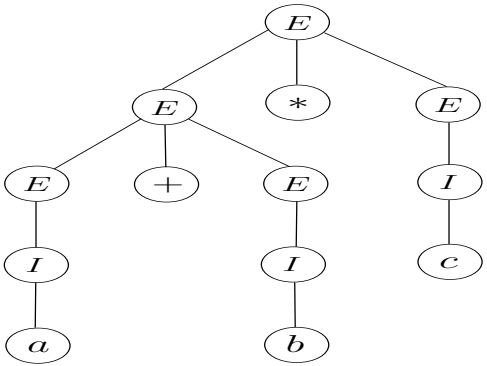
Ambiguity in CFG

- **Example :** Consider the grammar $G = (N, T, E, P)$ with $N = \{E, I\}$, $T = \{a, b, c, +, *\}$, and productions

$$E \to E + E,$$
$$E \to E * E,$$
$$E \to I,$$
$$I \to a|b|c$$

- The string $a + b * c$ is in $L(G)$.





- It is easy to see that this grammar generates a restricted subset of arithmetic expressions for C-like programming languages.

Ambiguity in CFG

- The grammar is ambiguous, because the string $a + b * c$ has two different derivation trees.
- Now, if we evaluate this, suppose $a = 2$, $b = 3$, $c = 4$ then from first one we get $2 + 3 * 4 = 2 + 12 = 14$ and from second one we get $2 + 3 * 4 = 5 * 4 = 20$.
- So, you can not write grammar like this, it is very essential that the grammar should be unambiguous.
- But we can have this grammar and usually while evaluating we say that * has higher priority than + and then the evaluation will be from left to right. If we put some more restriction then this can be used.
- But without those restriction, two type of evaluation becomes possible, that is two types of code can be generated, which has to be avoided.

Ambiguity in CFG

- So, the ambiguity or meaning is different because you are having two different derivation trees, and because of the correspondence between leftmost derivation and derivation trees, ambiguity can be derived in terms of derivation trees or in terms of leftmost derivation. Both are equivalent.

Ambiguity in CFG

- So, the ambiguity or meaning is different because you are having two different derivation trees, and because of the correspondence between leftmost derivation and derivation trees, ambiguity can be derived in terms of derivation trees or in terms of leftmost derivation. Both are equivalent.

- **Definition :** Let $G = (N, T, P, S)$ be a CFG.
  - A word $w \in L(G)$ is said to be **ambiguously derivable** if there exists more than one derivation tree for w in G.

Ambiguity in CFG

- So, the ambiguity or meaning is different because you are having two different derivation trees, and because of the correspondence between leftmost derivation and derivation trees, ambiguity can be derived in terms of derivation trees or in terms of leftmost derivation. Both are equivalent.

- **Definition :** Let $G = (N, T, P, S)$ be a CFG.
  - A word $w \in L(G)$ is said to be **ambiguously derivable** if there exists more than one derivation tree for w in G.
  - A context-free grammar G is said to be **ambiguous** if there exists some $w \in L(G)$ that has at least two distinct derivation trees. Alternatively, **ambiguity** implies the existence of two or more leftmost derivations.
  - Otherwise, it is said to be **unambiguous**.

Ambiguity in CFG

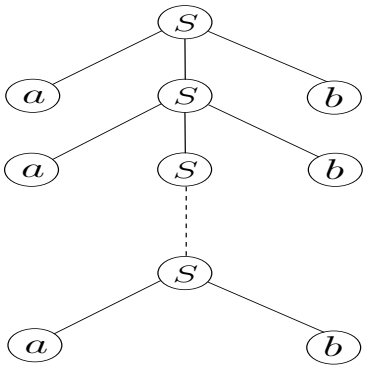
- **Example :** Consider the grammar G_1 : 1. $S \rightarrow aSb$
  2. $S \rightarrow ab$
  - The language generate by this grammar is :

Ambiguity in CFG

- **Example :** Consider the grammar G_1 : 1. $S \to aSb$
  2. $S \to ab$
  - The language generate by this grammar is :$L = \{a^n b^n : n \geq 1\}$

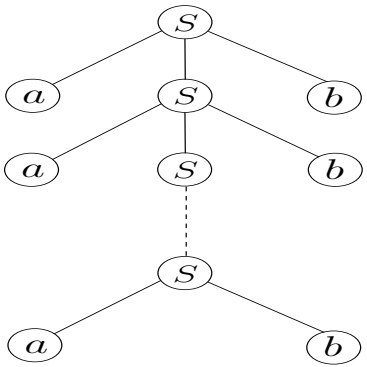
Ambiguity in CFG

- **Example :** Consider the grammar G_1 : 1. $S \to aSb$
  2. $S \to ab$
  - The language generate by this grammar is :$L = \{a^n b^n : n \geq 1\}$
  - Now, if you take any word, the derivation tree will be like this–



Ambiguity in CFG

- **Example :** Consider the grammar G_1 : 1. $S \to aSb$
  2. $S \to ab$
  - The language generate by this grammar is :$L = \{a^n b^n : n \geq 1\}$
  - Now, if you take any word, the derivation tree will be like this–

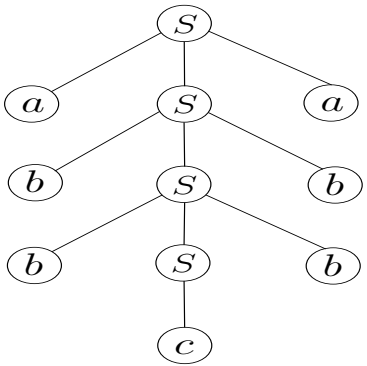


  - So, if you take $a^n b^n$, there will be only one derivation tree for that.
  - So, this grammar is **unambiguous**.

Ambiguity in CFG

- **Example :** Consider the grammar G_2 : 1. $S \to aSa$
  2. $S \to bSb$
  3. $S \to c$

  - The language generated consists of strings of the form wcw^R where $w \in \{a, b\}^*$
  - For example, if we take the string $abbcbba$, the derivation tree will be like this–



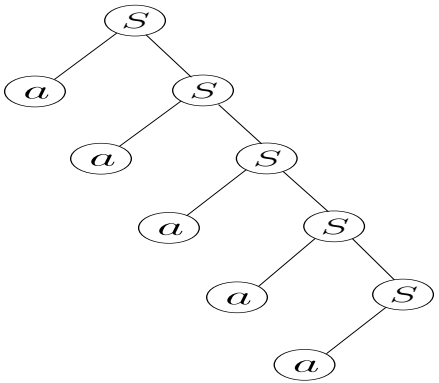
  - There will be only one derivation tree for each word. So, this grammar is **unambiguous**.

Ambiguity in CFG

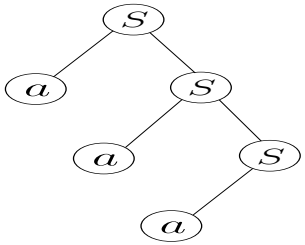
- **Example :** Consider the grammar G_3 : 1. $S \to aS$
  2. $S \to a$
  - The language generate by this grammar is :

Ambiguity in CFG

- **Example :** Consider the grammar G_3 : 1. $S \to aS$
  2. $S \to a$
  - The language generate by this grammar is :$L = \{a^n : n \geq 1\}$
  - If we want to have the derivation tree for a^5 and a^3, it will be like this–



For a^5



For a^3

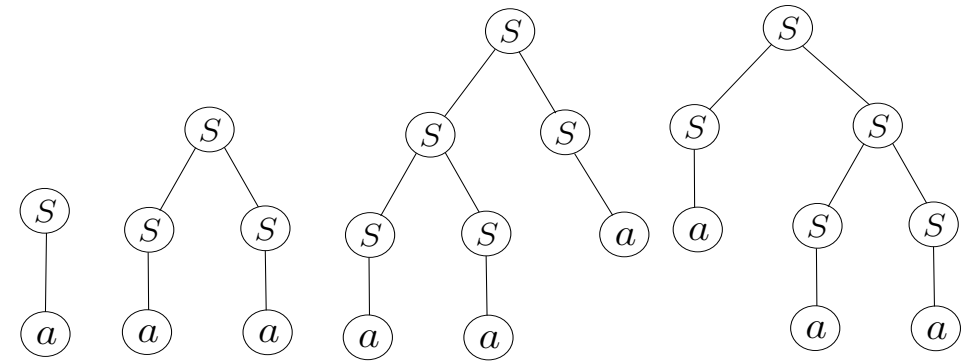
  - So, any string a^n if you take, there will be only one derivation tree.
  - So, this grammar is **unambiguous**.

Ambiguity in CFG

- **Example :** Consider the grammar G_4 : 1. $S \to SS$
  2. $S \to a$
  - The language generate by this grammar is :

Ambiguity in CFG

- **Example :** Consider the grammar G_4 : 1. $S \to SS$
  2. $S \to a$
  - The language generate by this grammar is :$L = \{a^n : n \geq 1\}$
  - If we want to have the derivation tree for a, a^2 and a^3, it will be like this–



  - For a^3 you can have two different derivation trees.
  - So, this grammar is **ambiguous**.

Ambiguity in CFG

- Both the grammar G_3 and G_4 generate the same language $L = \{a^n : n \geq 1\}$, but G_3 is a unambiguous grammar and G_4 is a ambiguous grammar.
- So, many grammar may generate the same language L. Some of them may be ambiguous and some of them may be unambiguous.

Ambiguity in CFG

- Both the grammar G_3 and G_4 generate the same language $L = \{a^n : n \geq 1\}$, but G_3 is a unambiguous grammar and G_4 is a ambiguous grammar.
- So, many grammar may generate the same language L. Some of them may be ambiguous and some of them may be unambiguous.
- **Definition :** A Context-free language L can be generated by many grammar $G_1, G_2, G_3, \cdots$
  - L is said to be **unambiguous**, if there is an unambiguous grammar generating it.
  - If all the grammar generating L are ambiguous, i.e., there is no unambiguous grammar generating L, then L is said to be an **inherently ambiguous** CFL.

- For the language $L = \{a^n : n \geq 1\}$, there is a one grammar which is unambiguous, there is a one grammar which is ambiguous. So, it is unambiguous language.

- There are inherently ambiguous context-free language:
  1. $L_1 = \{a^i b^j c^k : i, j, k \geq 1,\ i = j \text{ or } j = k\}$
  2. $L_2 = \{a^n b^m c^p d^q : n, m, p, q \geq 1,\ n = p \text{ or } m = q\}$

- To show some context-free language is inherently ambiguous is not easy, the proof is quite involved. You have to show that any grammar which generates has to be ambiguous that is not a very easy thing to prove.

- The language L_2 was shown to be inherently ambiguous by Parikh, and L_1 was shown to be inherently ambiguous by Chomsky and Schutzenberger.