$= \delta_A(q_0, h(w))$ by defn of Homomorphism

Context Free Grammars (CFG) - Informal Comments

* A context free grammar (CFG) is a notation for describing languages
* It is more powerful than finite automata or RE's but still cannot define all possible languages
* Useful for nested structures, eg. : parentheses in programming languages

Some languages which cannot be described by finite automata

$\{0^n 1^n \mid n \geq 1\}$ can be described by a CFG.

$\{0^n 1^n 2^n \mid n \geq 1\}$ cannot be " " " CFG.

Basic idea is to use "variables" to stand for sets of strings (i.e, languages)

- These variables are defined recursively in terms of one another

• Recursive rules ("productions") involve only concatenation

• Alternative rules for a variable allow union

- Example:

CFG for $\{0^n 1^n \mid n \geq 1\}$

$S \longrightarrow 01$ "S produces 01" $\boxed{S \rightarrow 0S1}$

0 01 1

again.

$S \rightarrow$

0 0011 1

Any ~~~~ RL can be expressed in terms of CFG

Proof:

Basis: 01 is in the language

Induction: If w is in the language, then so is 0w1

CFG Formalism

- Terminals:
  = symbols of the alphabet of the language being defined
- Variables. non-terminals
  = a finite set of other symbols, each of which represent a language
- Start symbol
  = the variables whose language is the one being defined
- Productions
  A production has the form:
  variable $\rightarrow$ string of variables & terminals

Example : Define a CFG for a language consisting of all strings of a and b, in which the number of a's and the number of b's are not equal

L = {abb, bba, aab, ababa, a, b, aa}

X	$S \rightarrow aSb$	$A \rightarrow a$
	$S \rightarrow bSa$	$A \rightarrow Aa$
starts & ends with a →	$S \rightarrow A$	$B \rightarrow B$
starts and ends with b →	$S \rightarrow B.$	$B \rightarrow Bb$

Example : Formal CFG

- Here is the formal CFG for $\{0^n 1^n \mid n \geq 1\}$

Terminals = {0,1} G = (V, S, T, P)
Variables = {S}
Start symbol : = S
Productions : $S \rightarrow 0S1$
$S \rightarrow 01$

Derivations – Intuition.

- We derive strings in the language of CFG by
  - starting with the start symbol

and repeatedly replacing some variable A by the right side of one of its productions

  - That is the "productions for A" are those that have A on the left side of the →

$$A \longrightarrow \alpha$$

Derivations — Formation

- We say $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if $A \rightarrow \gamma$ is a production
- Example.

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$$

$W = 000111$

Iterated - Derivation

- $\overset{*}{\Rightarrow}$ means "zero or more derivation steps"
- Basis :

  $\alpha \overset{*}{\Rightarrow} \alpha$ for any string α
- Induction :

  if $\alpha \overset{*}{\Rightarrow} \beta$ and $\beta \Rightarrow \gamma$ then $\alpha \overset{*}{\Rightarrow} \gamma$
- Example : Iterated Derivation

  $S \longrightarrow 01$; $S \longrightarrow 0S1$

  $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$

So $S \overset{*}{\Rightarrow} S$; $S \overset{*}{\Rightarrow} 0S1$; $S \overset{*}{\Rightarrow} 00S11$;

$S \overset{*}{\Rightarrow} 000111$

* Sentential Forms :

Any string of variables and/or terminals derived from the start symbol is called a sentential form.

Left most Derivations

- Say $wA\alpha \underset{lm}{\Rightarrow} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.

  (string of terminals / string of variables & terminals)

- Also $\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$ if α becomes β by a sequence of 0 or more $\underset{lm}{\Rightarrow}$ steps

  ← left most derivation

Example : Left most derivations replacing left most variable

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

Derive $w = (())()$

$S \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S$

$\Rightarrow (())()$

* Right most Derivations :

✓ Say $\alpha A w \underset{rm}{\Rightarrow} \alpha\beta w$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.

✓ Also $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$ if α becomes β by a sequence of 0 or more $\underset{rm}{\Rightarrow}$ steps.

✓ Example : Right most Derivations

✓ Balanced - parentheses grammar:

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

$w = (())()$

$S \underset{rm}{\Rightarrow} S(S) \Rightarrow S() \underset{rm}{\Rightarrow} (S)()$

$\underset{rm}{\Rightarrow} (())()$

$S \overset{*}{\underset{rm}{\Rightarrow}} (())()$

Formally α is a sentential from iff $S \stackrel{*}{\Rightarrow} \alpha$

* <u>Language of a Grammar</u>

- If G is a CFG, then $L(G)$, the language of G, is $\{ w \mid S \stackrel{*}{\Rightarrow} w \}$
  - Note w must be a terminal string, S is the start symbol

- <u>Example</u>:
  G has ~~an~~ productions $S \rightarrow \epsilon$ and $S \rightarrow 0S1$

  $L(G) = \{ 0^n 1^n \mid n \geq 0 \}$ Note: ϵ is the legitimate right side

  $\{\epsilon, 01, 0011, 000111, \ldots\}$

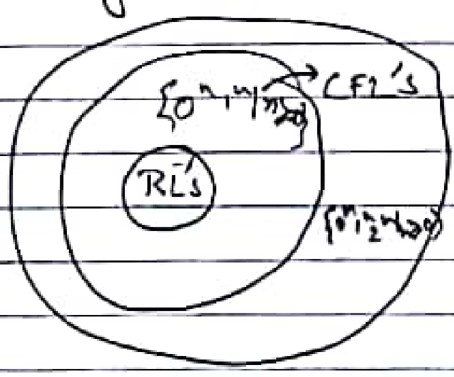
* <u>Context - Free languages</u>

  - A language that is defined by some CFG is called a context free language.

- There are CFL's that are not regular languages such as $\{ 0^n 1^n \mid n \geq 0 \}$

- But not all languages are CFL's, eg. $\{0^n 1^n 2^n \mid n \geq 1\}$

- Intuitively, CFL's can count two things not three



$\{ 0^n 1 \mid n \geq 1 \}$

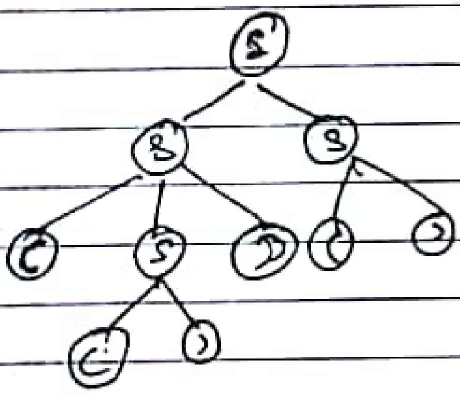
$S \rightarrow S_1 1$

$S_1 \rightarrow 0 S_1$

$S_1 \rightarrow 0$

* Parse Trees

- are trees named by symbol of a particular CFG
- Leaves: labeled by terminal or ϵ
- Interior nodes: labeled by a variable
  - Children are labelled by the right side of a production for the parent
- Root: must be labelled by the start symbol
- Example: Parse Trees



* Yeild of a Parse Tree.
  - The concatenation of the labels of the leaves in left to right order is called the yeild of the parse tree.

  Yeild = (())()

(( )) ( )

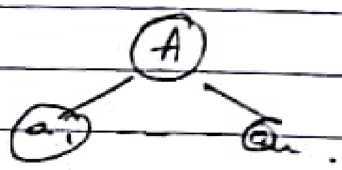
- For every parse tree, there is a unique left most and a unique right most derivation.
- We will prove
  1. If there is a parse tree with root labelled A and yeild w then.
     $A \Rightarrow^{*}_{lm} w$
  2. If $A \Rightarrow^{*}_{lm} w$, then there is a parse tree with root A and yeild w.

• Proof - Part 1

✓ Induction on the height (length of the longest path from the root) of the tree

✓ Basic: height 1

Tree looks like

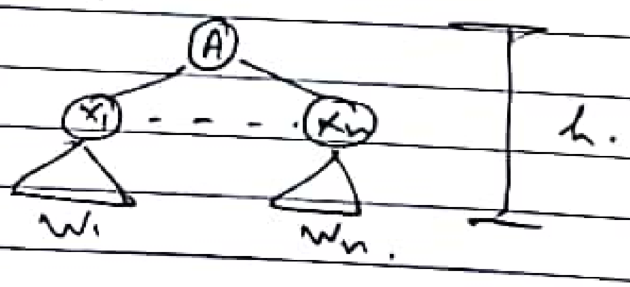


$\therefore$ $A \rightarrow a_1 \ldots a_n$ must be a production.

Thus $A \underset{lm}{\Rightarrow} a_1 \ldots a_n$

<u>Induction:</u>

Assume (1) for trees of height $< h$ and let this tree have height h.



By IH,

$x_i \overset{*}{\underset{lm}{\Rightarrow}} w_i$ Note: If x_i is a terminal then $x_i = w_i$

Thus $A \underset{lm}{\Rightarrow} X_1 \ldots X_n$

$\overset{*}{\underset{lm}{\Rightarrow}} w_1 X_2 \, X_n$

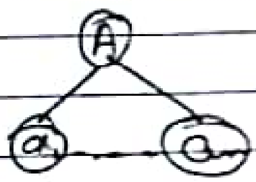
$\overset{*}{\underset{lm}{\Rightarrow}} w_1 w_2 \ldots X_n \Rightarrow_{lm} \ldots$

$\overset{*}{\underset{lm}{\Rightarrow}} w_1 \ldots w_n$

Proof : Part 2:

Given a left most derivation of a terminal string, we need to prove the existence of a parse tree.
The proof is an induction on the length of the derivation

Basis:
If $A \underset{lm}{\Rightarrow} a_1 \ldots a_n$ by a one step derivation then there must be a parse tree.



Induction:
Assume (2) for derivation of fewer than $k > 1$ steps, and let $A \overset{*}{\underset{lm}{\Rightarrow}} w$ be a k-step derivation

- First step is $A \underset{lm}{\Rightarrow} X_1 \ldots X_n$

Now w can be divided so that first partition is derived from X_1, the next is derived from X_2 and so on.
If X_i is a terminal the $w_i = X_i$

Induction:

That is $X_i \overset{*}{\underset{lm}{\Rightarrow}} w_i$ for all i such that X_i is a variable
And the derivation takes fewer than k-steps

By the IH, if X_i is a variable then there is a parse tree with root X_i and yeild w_i
Thus there is a parse tree.

Ambiguous Grammar

* A CFG is ambiguous if there is a string in the language that is the yield of two or more parse trees

Example : $S \rightarrow SS \mid (S) \mid ( )$

~~Cong~~ Consider a string ( ) ( ) ( ) and the corresponding parse trees

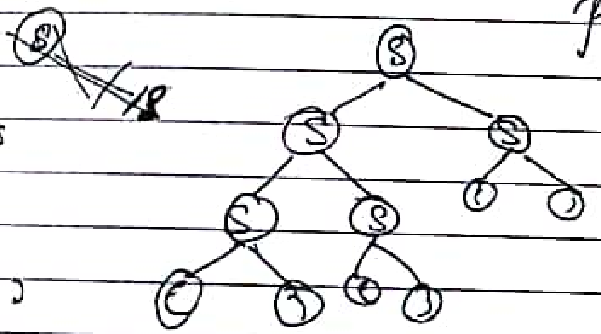
Left

$S \Rightarrow SS \Rightarrow SSS$

$\Rightarrow ( )\, SS$

$\Rightarrow ( )\, ( )\, S$

$\Rightarrow ( )\, ( )\, ( )$

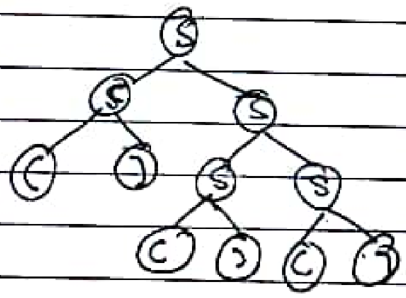


$S \Rightarrow SS$

$\Rightarrow ( )\, SS$

$\Rightarrow ( )\, ( )\, S$

$\Rightarrow ( )\, ( )\, ( )$



* Removing Ambiguity of a Grammar

For the balanced - parantheses language here is another CFG, which is unambiguous

Start symbol $B \rightarrow (RB \mid \epsilon$

$R \rightarrow)\mid (RR$

Let w = (()) ()

$B \Rightarrow (RB \Rightarrow ((RRB \Rightarrow (()RB$

$\Rightarrow (())B$

$\Rightarrow (())(RB$

$\Rightarrow (())()B$

$\Rightarrow (())()$

Inherent Ambiguity

- Certain CFL's ~~are~~ are inherently ambiguous, meaning that every grammar for the language is ambiguous

- Example:

$$\{0^i 1^j 2^k \mid i = j \text{ or } j = k\}$$

$S \rightarrow AB \mid CD$

$A \rightarrow 0A1 \mid 01$

$B \rightarrow 2B \mid 2$

$C \rightarrow 0C \mid 0$

$D \rightarrow 1D2 \mid 12$

There are two derivations of every string with equal numbers of 0's, 1's and 2's,

eg: consider 012

$S \Rightarrow AB \Rightarrow 01B \Rightarrow 012$

$S \Rightarrow CD \Rightarrow 0D \Rightarrow 012$

Variables that derive Nothing

- Consider,

$S \rightarrow AB$

$A \rightarrow aA|a$

$B \rightarrow AB$

Although A derives all string of a's. B derives no terminal string.

Thus S derive nothing and the language is empty

✓ Testing whether a variable derives some terminal string.

Basis: If there is a production $A \rightarrow w$, where w has no variables, then A derives a terminal string

Induction:

If there is a production $A \rightarrow \alpha$, ~~ub~~ where α consists only of terminals and variables known to derive a terminal string, then A derives a terminal string

Eventually we can find no more variables.

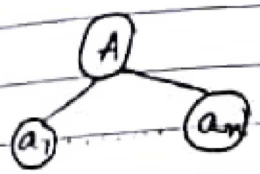
* An easy induction on the order in which variables are ~~discovde~~ discovered shows that

* Conversely any variable that derives a terminal string will be discovered by this algorithm

Proof: This proof is an induction on the height of the ~~last~~ least height parse tree by which a variable A derives a terminal string

Basis:

Height = 1 Tree looks like

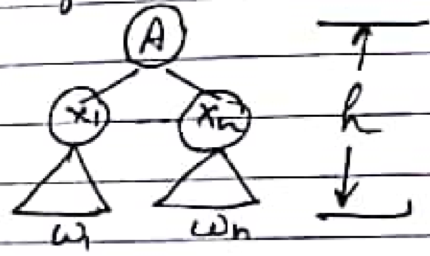


Then the Basis of the algorithm tells us that A will be discovered.

Assume Induction hypothesis for parse trees of height < h and suppose A derives a terminal string via a parse tree of height h

By IH, those X's that are variables are discovered

Thus A will also be discovered because it has a right side of terminals and/or discovered variables



Algorithm to Eliminate.

(A) Variables that derive Nothing

Step:

1. Discover all variables that derive terminal strings
2. For all other variables, remove all productions in which they appear either on the left or the right

Example:

$S \to AB \mid a$

$A \to aA$

$B \to bB$

$C \to c$

Basis: A and C are identified because of $A \to a$ and $C \to c$.

Induction: S is identified ~~bec~~ because of $S \to C$.

Nothing else can be identified.

Result: $S \to C$
$A \to aA \mid a$
$C \to c$.

* Unreachable Symbol
✓ Another way a terminal or variable deserves to be eliminated is if it cannot appear in any derivation from the start symbol

✓ Basis: We can reach S (the start symbol)

✓ Induction: If we can reach A and there is a production $A \to \alpha$, then we can reach all symbols of α

When we can discover no more symbols, we have all and only the symbols that appear in derivation from S.

✓ Algorithm:
Remove from the grammar all symbols not discovered / reachable from S and all productions that involve these symbols.

Eliminating Useless Symbols

- A symbol is useful if it appears in some derivation of some terminal string from the start symbol.

- Otherwise it is useless
  Eliminate all useless symbols by :
  - Eliminate symbols that derive no terminal string
  - Eliminate unreachable symbol

Example :

$S \rightarrow \cancel{AB}$

$A \rightarrow c$

$C \rightarrow c.$

$B \rightarrow \cancel{bB}$

- ✓ A & C both derive terminal string
- ✓ B & S do not derive any terminal string.
- ✓ A, C and c are all unreachable symbols

Epsilon Productions (ϵ-productions)

We can almost avoid using productions of the form $A \rightarrow \epsilon$ (called ϵ-productions)

✓ The problem is that ϵ cannot be in the language of any grammar that has no ϵ-productions

✓ Theorem :

If L is a CFL then $L - \{\epsilon\}$ has a CFG with no ϵ-productions

Nullable Symbols

• Nullable Symbols

To ~~dismiss~~ eliminate ϵ-productions, we first need to discover the nullable variables

For every nullable variable A, there is a derivation
$A \overset{*}{\Rightarrow} \epsilon$

Basis:
If there is a production $A \rightarrow \epsilon$, then A is nullable.

Induction:
If there is a production $A \rightarrow \alpha$ and all symbols of α are nullable then A is nullable.

Example: Nullable Symbols
$S \rightarrow AB$
$A \rightarrow aA \mid \epsilon$
$B \rightarrow bB \mid A$

Basis: A is nullable because $A \rightarrow \epsilon$

Induction: B is nullable because of $B \rightarrow A$

then S is nullable because of $S \rightarrow AB$

Eliminating e-productions

• Key idea:
Turn each production
$A \rightarrow X_1 \ldots X_n$.
into a family of productions.

- For each subset of nullable X_i's there is one production with those eliminated from the right side "in advance".
  - Except if all X_i's are nullable do not make a production with ϵ as the right side.

- Example: Eliminating ϵ-productions

$S \rightarrow ABC$	$B \rightarrow bB$	$S \cancel{\rightarrow} BC$	$S \rightarrow \cancel{ABC}$
$A \rightarrow aA$	$\cancel{B \rightarrow \epsilon}$	$S \cancel{\rightarrow} AC$	
$\cancel{A \rightarrow \epsilon}$	$\cancel{C \rightarrow \epsilon}$	$S \rightarrow AB$	
		$S \cancel{\rightarrow} C$	$A \rightarrow aA$
		$S \rightarrow A$	$B \rightarrow bB$
		$S \rightarrow B$	

- A, B, C, and S are nullables.

Unit Productions:

- A unit production is one whose right side consists of exactly one variable.
- These productions can be eliminated.

  Key idea:

  If $A \stackrel{*}{\Rightarrow} B$ by a series of unit production and $B \rightarrow \alpha$ is a non-unit production, then add production $A \rightarrow \alpha$.

  Then drop all unit production

✓ Cleaning up a Grammar:

- Theorem:

  If L is a CFL then there is a CFG for $L - \{\epsilon\}$ that has:
  - No useless symbols
  - No ϵ-productions
  - No unit production.

  i.e every right side is either a single terminal or has length ≥ 2.

Proof : Start with a ~~CFG~~ CFG for L
Perform the following steps in order.

1. Eliminate ϵ production
2. Eliminate unit productions.
3. Eliminate variables that derive no terminal string
4. Eliminate variables that derive no terminal String
5. Eliminate variables not reachable from start symbol.

* Chomsky Normal Form (CNF)
  - A CFG is said to be in CNF if every production is one of these two forms

1. $A \to BC$ (right side has two Variables
2. $A \to a$ (right side has a single terminal)

Theorem:

If L is a CFL then $L - \{\epsilon\}$ has a CFG in CNF

Proof Of CNF Theorem:

Step 1: "Clean" the grammar, so every production right side is either a single terminal or of length at least 2.

Step 2: For each right side $\neq$ a single terminal, make the right side all variables

- For each terminal, create a new variable A_a and production $A_a \to a$
- Replace a by A_a in right side of length ≥ 2

Step 3:

Consider production $A \to BcDe$.

✓ We need variables A_c and A_e with productions

$A_c \to c$.

$A_e \to e$

✓ Replace $A \to BcDe$ by $A \to BA_cDA_e$.

Step 4:

Break right sides longer than 2 into a ~~de~~ ~~ds~~ ~~chain~~ ~~of~~ chain of productions with right sides of two variables

Example: $A \to BCDE$ is replaced by

$A \to BF$

$F \to CG$.

$G \to DE$

✓ Recall $A \to BCDE$ is replaced by

$A \to BF$, $F \to CG$ and $G \to DE$

In the new grammar

$A \Rightarrow BF \Rightarrow BCG \Rightarrow BCDE$