Properties of CFLs

We will study three properties:

- Normal forms of CFGs. This makes other tasks on grammars easier. For instance, it might be easier to build a parser for a grammar if we can make some assumptions about the form of the grammar rules.
- Pumping Lemma for CFLs. Similar to the regular language case, allows us to show if a language is not context-free.
- Closure properties for CFLs. Operations on CFLs that produce CFLs.

Chomsky Normal Form (CNF)

- Every CFL that does not include ϵ is generated by a CFG of the form $A \to BC$, or $A \to a$.
- This is called CNF, and to get there we have to
  - First, eliminate ϵ-productions, i.e., $A \to \epsilon$.
  - Then, eliminate unit productions, i.e., $A \to B$, where A and B are variables.
  - Finally, eliminate useless symbols, those that do not appear in any derivation $S \overset{*}{\Rightarrow} w$, for start symbol S and string w.

1- Eliminating ϵ-productions

- *Basic idea*: Suppose A is nullable (i.e., $A \stackrel{*}{\Rightarrow} \epsilon$). We'll then replace a rule like like $C \to BAD$ with $C \to BAD, C \to BD$ and delete any rules with body ϵ.

Algorithm $RemoveEps(G)$, where $G = (V, T, R, S)$:

1. *Obtain the set of all nullable symbols, $n(G)$, in G*:
   - **Basis**: For all rules $A \to \epsilon \in R$, include A in $n(G)$.
   - **Induction**: For all rules $A \to C_1 C_2 \cdots C_k \in R$. If $\{C_1, C_2, \cdots, C_k\} \subseteq n(G)$, then include A in $n(G)$.
2. *Obtain the new grammar G_1*: for each rule $A \to X_1 X_2 \cdots X_k$ of R, suppose m of the k X_i's s are nullable. Then G_1 will contain 2^m versions of this rule, where the nullable X_i's in all combinations are present or absent.

Eliminating ϵ-productions: example

- Let G be $S \to AB, A \to aAA \mid \epsilon, B \to bBB \mid \epsilon$
- Now $n(G) = \{A, B, S\}$. The first rule will become: $S \to AB \mid A \mid B$, the second $A \to aAA \mid aA \mid aA \mid a$, and the third $B \to bBB \mid bB \mid bB \mid b$
- We then delete the redundant rules, and end up with grammar G_1 :
  $S \to AB \mid A \mid B, A \to aAA \mid aA \mid a, B \to bBB \mid bB \mid b$

$w8.1$

2- Eliminating unit productions $A \to B$

- Consider the grammar

$$E \to T \mid E + T$$
$$T \to F \mid T * F$$
$$F \to I \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

It has unit productions $E \to T, T \to F$, and $F \to I$.

- Such productions are there as the result of the design of a unambiguous grammar.
- Those productions can be eliminated without affecting the grammar.

Eliminating unit productions (cnt.)

The idea behind eliminating unit productions:

- We'll expand rule $E \to T$ and get rules $E \to F, E \to T * F \mid E + T$
- Then we'll expand $E \to F$ and get $E \to I \mid (E) \mid T * F \mid E + T$
- Finally we expand $E \to I$ and get

Original grammar:

$$E \to T \mid E + T$$
$$T \to F \mid T * F$$
$$F \to I \mid (E)$$
$$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

$$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T*F \mid E+T$$

The expansion method works as long as there are no cycles in the rules, as e.g. in

$$A \to B, B \to C, C \to A$$

Eliminating unit productions (cnt.)

- (A, B) is a unit pair if $A \stackrel{*}{\Rightarrow} B$ using unit productions only.
- Computing the set of unit pairs $u(G)$ for $G = (V, T, R, S)$:
  - **Basis**: Forall $A \in V$, include (A, A) in $u(G)$.
  - **Induction**: Let $B \to C \in R$. If $(A, B) \in u(G)$, then include (A, C) in $u(G)$.

$$
\begin{aligned}
I &\to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
F &\to I \mid (E) \\
T &\to F \mid T * F \\
E &\to T \mid E + T
\end{aligned}
$$

(E, E) and $E \to T$ gives (E, T)

(E, T) and $T \to F$ gives (E, F)

(E, F) and $F \to I$ gives (E, I)

(T, T) and $T \to F$ gives (T, F)

(T, F) and $F \to I$ gives (T, I)

(F, F) and $F \to I$ gives (F, I)

$u(G) = \{(I, I), (E, E), (T, T), (F, F), (E, T), (E, F), (E, I), (T, F), (T, I), (F, I)\}$

Eliminating unit productions (cnt.)

Algorithm $RemoveUnitPrds(G)$, where $G = (V, T, R, S)$:

1. Find all unit pairs of G
2. For each unit pair (A, B), add to R_{new} a new production of the form $A \rightarrow \alpha$, if $B \rightarrow \alpha \in R$ and α is not a variable.

Example: applying $RemoveUnitPrds$ on the grammar below:

$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

$F \rightarrow I \mid (E)$

$T \rightarrow F \mid T * F$

$E \rightarrow T \mid E + T$

Pair	Productions
(E, E)	$E \rightarrow E + T$
(E, T)	$E \rightarrow T * F$
(E, F)	$E \rightarrow (E)$
(E, I)	$E \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(T, T)	$T \rightarrow T * F$
(T, F)	$T \rightarrow (E)$
(T, I)	$T \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(F, F)	$F \rightarrow (E)$
(F, I)	$F \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(I, I)	$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Removing unit productions (cnt.)

Exercise: Using algorithm RemoveUnits, remove the unit production from the grammar below:

$$S \rightarrow XY$$
$$X \rightarrow A$$
$$A \rightarrow B|a$$
$$B \rightarrow b$$
$$Y \rightarrow T$$
$$T \rightarrow d|c$$

3- Eliminating useless symbols

First, some definitions: Given $G = (V, T, P, S)$

- A symbol X is generating if $X \stackrel{*}{\Rightarrow} w$, for some $w \in T^*$.
- A symbol X is reachable if $S \stackrel{*}{\Rightarrow} \alpha X \beta$, for some $\{\alpha, \beta\} \subseteq (V \cup T)^*$.
- A symbol is X useful for a grammar $G = (V, T, P, S)$ if it is generating and reachable, i.e.:

$$S \stackrel{*}{\Rightarrow} \alpha X \beta \stackrel{*}{\Rightarrow} w \text{ for a terminal string } w$$

Algorithm $RemoveUseless(G)$, where $G = (V, T, P, S)$:

1. Eliminate the non-generating symbols and all productions involving those symbols.
2. Eliminate the non-reachable symbols and all productions involving those symbols..

Computing the useful symbols

Let $G = (V, T, R, S)$:

- Generating symbols: $g(G)$

  Basis: $g(G) = T$, i.e., all terminal symbols are generating.

  Induction: For all rules $X \to C_1 \cdots C_k \in R$.

  If $C_i \in g(G)$, $i = 1 \cdots k$, then include X in $g(G)$.

- Reachable symbols: $r(G)$

  Basis: $r(G) = S$, i.e., the start symbol is reachable.

  Induction: For all rules $A \to \alpha \in R$. If variable $A \in r(G)$ then add all symbols in α to $r(G)$.

Eliminating useless symbols: example

Using $RemoveUseless(G)$, where G is the grammar below:

$$S \to AB \mid a, A \to b$$

Step 1: Generating symbols: $g(G) = \{S, A, a, b\}$, B therefore is useless. To eliminate B we have to eliminate $S \to AB$, thus obtaining G':

$$S \to a, A \to b$$

Step 2: Reachable symbols: $r(G') = \{S, a\}$. Thus, A and b are unreachable, and we should eliminate them, leaving us with G'':

$$S \to a$$

$w8.2{-}3$

Summary

To “clean up” a grammar we need to

1. Eliminate ϵ-productions
2. Eliminate unit productions
3. Eliminate useless symbols

in this order.

Chomsky Normal Form (CNF)

- A grammar is in CNF if every production is of the form:
  - $A \to BC$, where $\{A, B, C\} \subseteq V$, or
  - $A \to \alpha$, where $A \in V$, and $\alpha \in T$.
- To achieve this, start with any grammar for the CFL, and
  1. "Clean up" the grammar.
  2. Arrange that all bodies of length 2 or more consists of only variables.
  3. Break bodies of length 3 or more into a cascade of two-variable-bodied productions.

Addressing steps 2 & 3

- For step 2, for every terminal a that appears in a body of length ≥ 2, e.g., $B \rightarrow CDaE$, create a new variable, say A, and replace a by A in all bodies (e.g., $B \rightarrow CDaE$ becomes $B \rightarrow CDAE$).
  Then add a new rule $A \rightarrow a$.
- For step 3, for each rule of the form $A \rightarrow B_1 B_2 \cdots B_k, k \geq 3$, introduce new variables $C_1, C_2, \cdots C_{k-2}$, and replace the rule with

$$
\begin{aligned}
A &\rightarrow B_1 C_1 \\
C_1 &\rightarrow B_2 C_2 \\
&\cdots \\
C_{k-3} &\rightarrow B_{k-2} C_{k-2} \\
C_{k-2} &\rightarrow B_{k-1} B_k
\end{aligned}
$$

CNF: example

- Let's start with the grammar (step 1 already done):

$$
\begin{aligned}
&E \to E + T \mid T * F \mid (E) \mid a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
&T \to T * F \mid (E)a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
&F \to (E) \quad a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \\
&I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1
\end{aligned}
$$

- For step 2, we need the rules
  $A \to a, B \to b, Z \to 0, O \to 1, P \to +, M \to a, L \to (, R \to)$
  and by replacing we get the grammar

$$
\begin{aligned}
&E \to EPT \mid TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&T \to TMF \mid LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&F \to LER \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&I \to a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&A \to a, B \to b, Z \to 0, O \to 1 \\
&P \to +, M \to *, L \to (, R \to)
\end{aligned}
$$

Example (cnt.)

- For step 3, we replace

$$E \to EPT \text{ by } E \to EC_1, C_1 \to PT$$
$$E \to TMF, T \to TMF \text{ by } E \to TC_2, T \to TC_2, C_2 \to MF$$
$$E \to LER, T \to LER, F \to LER \text{ by}$$
$$E \to LC_3, T \to LC_3, F \to LC_3, C_3 \to ER$$

- The final CNF grammar is

$$\begin{aligned}
&E \to EC_1 \mid TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&T \to TC_2 \mid LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&F \to LC_3 \mid a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&I \to a \mid b \mid IA \mid IB \mid IZ \mid IO \\
&C_1 \to PT, C_2 \to MF, C_3 \to ER \\
&A \to a, B \to b, Z \to 0, O \to 1 \\
&P \to +, M \to *, L \to (, R \to)
\end{aligned}$$

Exercise

Convert the following CFG to CNF.

$$A \to BAB \mid B \mid \epsilon$$
$$B \to 00 \mid \epsilon$$

Showing that a language is context-free

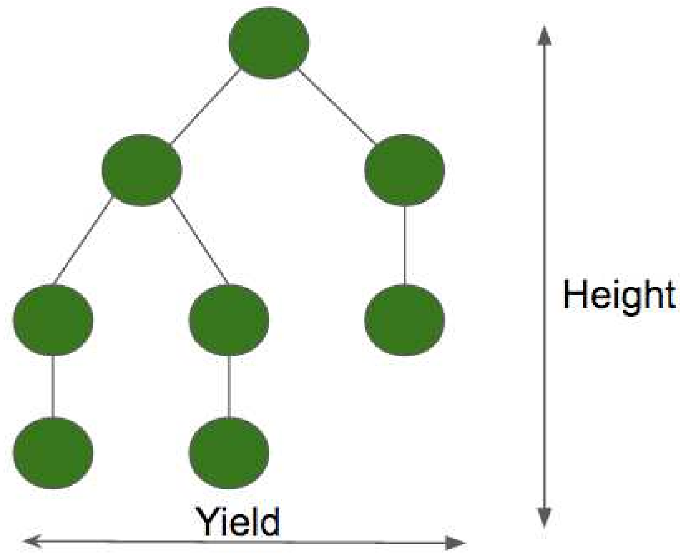
Techniques we have seen so far that can be used to show that a language L is context-free

- Provide a context-free grammar for it.
- Provide a PDA for it.

But suppose we tried to build a CFG and a PDA for L and we failed. Can we then conclude that L is not context-free?

Showing that L is NOT context-free

- The argument is based on a property that is provably true for all CFLs: the structure of the parse tree derived by a grammar in CNF.
- If we can show that L does not possess the property, then L is not context-free.



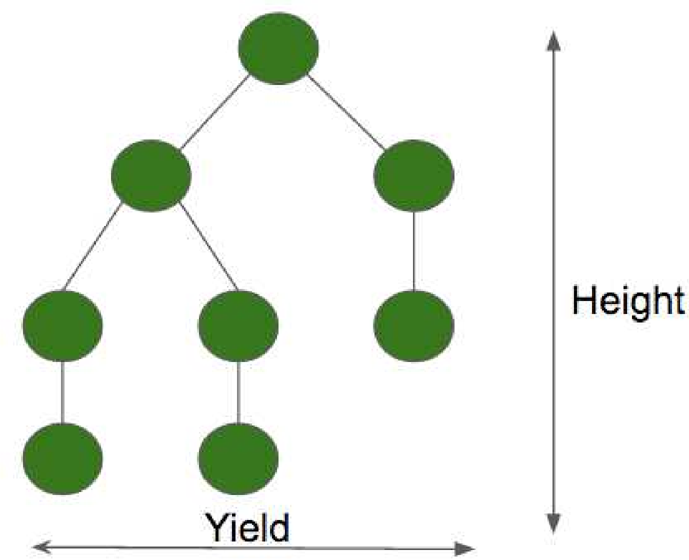
First, a helper theorem:

Theorem 1: For a grammar in CNF, suppose the yield of a parse tree is w. If the height of the tree (i.e., longest path from the root) is n, then $|w| \leq 2^{n-1}$.

Prelude to the CFL pumping lemma

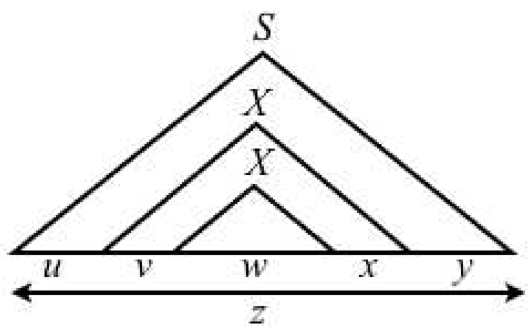
Consider the structure (height and number of leafs) of a binary parse tree produced by grammar G (in CNF).

- Suppose G has m variables.
- Suppose in a parse tree T no variable appears more than once on any path from the root of T to a terminal. Then the height of T is $\leq m$.
- From Theorem 1 (last slide) we can state that the longest string that corresponds to the yield of T has length $\leq 2^{m-1}$.



Prelude (cnt.)

- Now suppose we can find $z \in L(G)$ such that $|z| > 2^{m-1}$.
- Then, any parse tree that generates z must contain a path that contains at least one repeated variable.



We could sketch the derivation that produced the tree as:

$$S \stackrel{*}{\Rightarrow} uXy \stackrel{*}{\Rightarrow} u{\color{red}vXx}y \stackrel{*}{\Rightarrow} uvwxy$$

- If no recursive rule is used (see text in red), then the yield of the tree is uwy
- If the recursive rule is used say, i times, then the yield is uv^iwx^iy.

CFL pumping lemma (PL)

- Formally:
  - $\forall$ CFL L, $\exists$ integer n (the "pumping length")
    - $\forall z \in L, |z| \geq n$, $\exists uvwxy = z$ such that

      a) $|vwx| \leq n$

      b) $|vx| > 0$

      c) $\forall i \geq 0, uv^iwx^iy$ is in L.
- Notice that unlike the PL for RLs, we have to pump two strings, in tandem (i.e., the same number of copies of each).

Outline of proof for PL

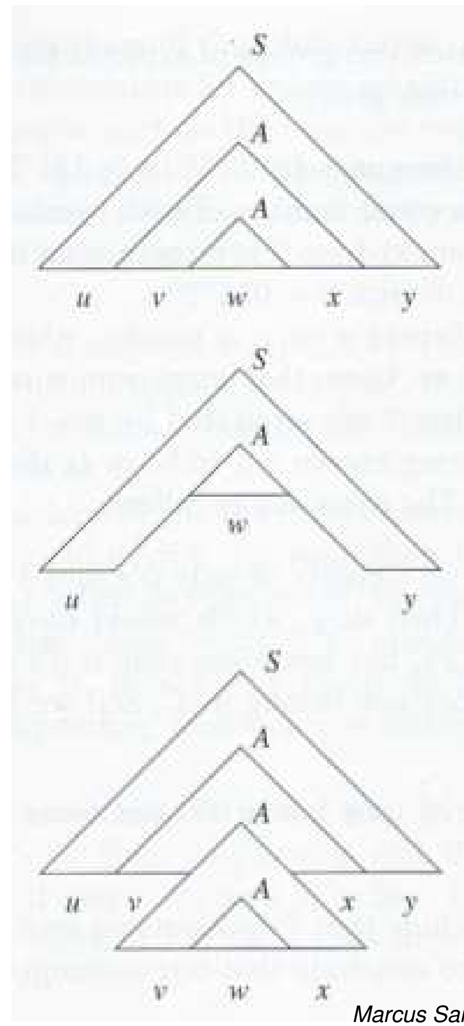
Some initial considerations:

- Let there be a Chomsky-normal-form CFG for L with m variables. Let's choose the "pumping length" $n = 2^m$.
- Because CNF grammars have binary parse trees, if the longest path in a parse tree of a string w has length p, then $|w| \leq 2^{p-1}$. (remember Theorem 1?)
- Therefore a string z of size $n = 2^m$ has some path with at least $m + 1$ variables . Why?
  - From **Theor. 1**: $2^m \leq 2^{p-1}$, hence, $p \geq m + 1$
- Therefore some variable must appear twice on the path.

Outline of proof (cnt.)

Let us focus on some sufficiently long path that has length $\geq m+1$. In this path we can find a duplication of some variable A among the variables of the path.

- Let the lower A derive w and the upper A derive vwx.
- CNF guarantees us $|vwx| \leq n$ and $vx \neq \epsilon$.[a]
- By repeatedly replacing the lower A's tree by the upper A's tree, we see uv^iwx^iy has a parse tree for all $i > 1$.
  - And replacing the upper by the lower shows the case $i = 0$, i.e., uwy is in L.



[a]The subtree rooted at the upper A has yield no greater than $2^m = n$; and there are no unit productions.

How to apply the pumping lemma

- You assume the language L in question is CF.
- You consider a certain pumping length p. (Do not bind p to a certain number. I.e., don't say "Let $p = 3$". Leave it general.)
- You find a string $|z| \geq p$ in the language. (Pick a string that you think you will not be able to pump).
- Split the string $z = uvwxy$, so that $|vx| > 0$ and $|vwx| \leq p$.
- Show that there is no possible split of z in which $\forall i \geq 0$, $uv^iwx^iy \in L$. In other words, for every split of z there will be a value of i where $uv^iwx^iy \notin L$.

Example

Using the PL to show that $L = \{ww : w \in \{0,1\}^*\}$ is not a CFL.

- Let's assume L is CF, and let p be the pumping length. Let's use the string $z = 0^p 1 0^p 1$ from L.
- z has length greater than p and appears to be a good candidate string to show that L is not a CFL.
- But $z = uv^iwx^iy$ can be pumped by dividing it as follows:

$$\overbrace{\underbrace{000\cdots000}_{u}\ \underbrace{0}_{v}\ \underbrace{1}_{w}}^{0^p1}\ \overbrace{\underbrace{0}_{x}\ \underbrace{000\cdots0001}_{y}}^{0^p1}$$

- Which does not mean L is a CFL.

Example (revisited)

Using the PL to show that $L = \{ww : w \in \{0,1\}^*\}$ is not a CFL.

- Let's use the string $z = 0^p 1^p 0^p 1^p$ from L and show that it cannot be pumped.
- According to PL, $z = uvwxy$, where $|vwx| \leq p$. There are three possible locations for vwx:
  - In the first or second halves of z: then impossible to pump and obtain a string in L.
  - Straddle the middle point: then when we pump z the resulting string has the form $0^p 1^i 0^j 1^p$, where i and j cannot both be p. Hence $z \notin L$.

Therefore, L is not context-free.