# Properties of context-free languages



We will study

1) Normal forms for CFGs (useful for proving properties of CFLs)
2) Expressive power ⇒ pumping lemma for CFLs
3) Closure and decision properties

## Normal forms for CFGs

We look at how to simplify CFGs, while preserving the generated language.

- gain efficiency in parsing
- simplify proving properties

1) Eliminate useless symbols:

We say that $X \in V$ is useful if

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* w \qquad \text{with } w \in V_T^* \quad \alpha, \beta \in V^*$$

Thus, a symbol is useless (not useful) if it does not participate in any derivation of strings of the language.
⇒ can be eliminated

Definition: $X \in V$ is generating if $X \Rightarrow^* w$, for $w \in V_T^*$

$X \in V$ is reachable if $S \Rightarrow^* \alpha X \beta$, for $\alpha, \beta \in V^*$

Hence, $X$ is useful, if it is both generating and reachable.

We identify useless symbols by

1a) eliminating non-generating symbols and all their productions

1b) ... unreachable —"— 

Note: it is important to do these two steps in the above order

Example: $\begin{cases} S \to AB \mid b \\ A \to a \end{cases}$ Let us consider what happens if we do first (1b) and then (1a)

- we eliminate unreachable symbols: all are reachable
- ... non-generating —"— : we eliminate B and $S \to AB$

$\Rightarrow$ we obtain: $S \to b$
$A \to a$

But if we do it in the right order:

1a) Eliminate non-generating symbols: B and $S \to AB$

1b) —"— unreachable —"— : A and $A \to a$

$\Rightarrow$ We obtain: $S \to b$

1a) Eliminating non-generating symbols:

We construct the set H of generating symbols, and then eliminate the symbols not in H and the productions containing them.

Algorithm to construct the set H of generating symbols

Input: grammar $G = (V_N, V_T, P, S)$

Output: set H of generating symbols

$H \leftarrow V_T$

while there is a change in H do

for each production $A \to X_1 \dots X_k$ in P do

if $\{X_1, \dots, X_k\} \subseteq H$ (i.e., all of $X_1, \dots, X_k$ are generating)

then $H \leftarrow H \cup \{A\}$

return H

Example: $G_1 = (V_N, V_T, P, S)$

with $V_N = \{S, A, B, C, D\}$, $V_T = \{a, b, c, d\}$,

P:
$$\begin{cases} S \to AB \mid AC \mid CD \\ A \to BB \\ B \to AC \mid ab \\ C \to Ca \mid CC \\ D \to Bc \mid b \mid d \end{cases}$$

initialization: $H = \{a, b, c, d\}$

iteration 1: $H \leftarrow H \cup \{B, D\}$

-"- 2: $H \leftarrow H \cup \{A\}$

-"- 3: $H \leftarrow H \cup \{S\}$

-"- 4: $H$ does not change

$C$ is not generating $\Rightarrow$ Remove $C$ and all productions containing $C$

1b) Eliminating unreachable symbols

We construct the set $R$ of reachable symbols, and then eliminate the symbols not in $R$ and the productions containing them.

Algorithm to construct the set $R$ of reachable symbols

Input: grammar $G = (V_N, V_T, P, S)$

Output: set $R$ of reachable symbols

$R \leftarrow \{S\}$

while there is a change in $R$ do

for each production $A \to X_1 \dots X_k$ in $P$ do

if $A \in R$ (i.e., $A$ is reachable)

then $H \leftarrow H \cup \{X_1, \dots, X_k\}$

return $R$

Example: $G_2 = (V_N, V_T, P, S)$ with $V_N = \{S, A, B, D\}$, $V_T = \{a, b, c, d\}$

P:
$$\begin{cases} S \to AB \\ A \to BB \\ B \to ab \\ D \to b \mid d \end{cases}$$

initialization: $R = \{S\}$

iteration 1: $R \leftarrow R \cup \{A, B\}$

-"- 2: $R \leftarrow R \cup \{a, b\}$

-"- 3: $R$ does not change

$D, c, d$ are unreachable

$\Rightarrow$ Remove $D, c, d$ and all productions containing them.

## 2) Eliminate $\varepsilon$-productions

$\varepsilon$-production: $A \to \varepsilon$ slows down parsing

Definition: $A \in V_N$ is nullable if $A \Rightarrow^* \varepsilon$

We first need to find all nullable symbols

Algorithm to construct the set $N$ of nullable symbols

Input: grammar $G = (V_N, V_T, P, S)$
Output: set $N$ of nullable symbols

$N = \emptyset$
for each production $A \to \varepsilon$ in $P$ do $N \leftarrow N \cup \{A\}$
while there is a change in $N$ do
  for each production $A \to X_1 \dots X_k$ in $P$ do
    if $\{X_1, \dots, X_k\} \subseteq N$ (i.e., all of $X_1, \dots, X_k$ are nullable)
      then $N \leftarrow N \cup \{A\}$
return $N$

Example: $G_3 = (V_N, V_T, P, S)$ with $V_N = \{S, A, B, C\}$, $V_T = \{a, b\}$

$P$:
$S \to ABC \mid BCB$
$A \to aB \mid a$
$B \to CC \mid b$
$C \to S \mid \varepsilon$

initialization: $N = \{C\}$
iteration 1: $N \leftarrow N \cup \{B\}$
-"- 2: $N \leftarrow N \cup \{S\}$
-"- 3: $N$ does not change

Knowing the nullable symbols, allows us to compensate for the elimination of $\varepsilon$-productions.

Example: in $G_3$, since B and C are nullable, we can derive:

$S \Rightarrow^* BCB$, $S \Rightarrow^* CB$, $S \Rightarrow^* BC$, $S \Rightarrow^* BB$
$S \Rightarrow^* B$, $S \Rightarrow^* C$, $S \Rightarrow^* \varepsilon$

Hence, if we eliminate $C \to \varepsilon$ (and B, C are not nullable anymore), we have to add direct productions for the above derivations.

Algorithm to eliminate $\varepsilon$-productions

1) Identify all nullable symbols

2) Replace each production $A \to X_1 \dots X_n$
by the set of all productions of the form $A \to \alpha_1 \dots \alpha_n$
where $\alpha_i = X_i$, if $X_i$ is not nullable
$\alpha_i = X_i$ or $\varepsilon$, if $X_i$ is nullable

3) If the resulting grammar contains $S \to \varepsilon$, introduce a new start symbol $S'$ and add the productions $S' \to S \mid \varepsilon$

4) Remove all $\varepsilon$-productions, except possibly the one for $S'$.

Example: by applying steps (1) and (2) to $G_3$, we get

$S \to ABC \mid AB \mid AC \mid A \mid$
$\quad BCB \mid BC \mid BB \mid CB \mid B \mid C \mid \varepsilon$

$A \to aB \mid a$

$B \to CC \mid C \mid \varepsilon \mid b$

$C \to S \mid \varepsilon$

Since we have $S \to \varepsilon$, we add
$S' \to S \mid \varepsilon$
and remove the remaining $\varepsilon$-productions.

Eliminate unit productions

Unit-production: $A \to B$ slows down parsing

Algorithm to eliminate unit-productions

1) Remove $\varepsilon$-productions

2) For all $A, B \in V_N$
if $A \Rightarrow^* B$ and $B \to \alpha$ is not unit
then add $A \to \alpha$

3) Eliminate all unit-productions

How do we determine whether $A \Rightarrow^* B$ holds?

Since we have no $\varepsilon$-productions, we have that

$A \Rightarrow^* B$ if and only if

$A \Rightarrow B_1 \Rightarrow B_2 \Rightarrow \cdots \Rightarrow B_{k-1} \Rightarrow B_k \Rightarrow B$

where all $B_i$'s are pairwise distinct. Hence $k \leq |V_N|$.

(if we had a sequence where two $B_i$'s are the same, we could eliminate all steps inbetween, and get a new sequence where all $B_i$'s are pairwise distinct)

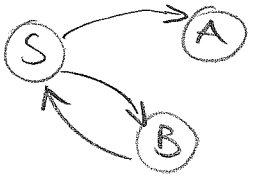
Each single derivation step $B_i \Rightarrow B_{i+1}$ must correspond to a unit production $B_i \to B_{i+1}$ of $G_1$.

Hence, we can detect whether $A \Rightarrow^* B$ by checking whether $B$ is reachable from $A$ in the graph of the unit productions:

- nodes: non-terminals
- edges: one edge (A) → (B) for each unit prod. $A \to B$

Example: $G_1$:
$$\begin{cases} S \to A \mid B \\ A \to Sa \mid a \\ B \to S \mid b \end{cases}$$

Graph of unit productions




Reachability: $S \Rightarrow^* A$ $S \Rightarrow^* B$ $B \Rightarrow^* S$ $B \Rightarrow^* A$

We get:
$$\begin{cases} S \to A \mid B \mid Sa \mid a \mid S \mid b \\ A \to Sa \mid a \\ B \to S \mid b \mid A \mid B \mid Sa \mid a \end{cases}$$

Removing unit productions, we get
$$\begin{cases} S \to Sa \mid a \mid b \\ A \to Sa \mid a \\ B \to Sa \mid a \mid b \end{cases}$$

Note: A and B have become unreachable

We have seen: removal of: useless symbols, $\varepsilon$-prod, unit-prod

Does the <u>order</u> of the steps matter?

Observation:

- removing useless symbols: does not add productions at all (and therefore not $\varepsilon$-prod. or unit-prod)
- removing $\varepsilon$-prod: could add unit-prod
- removing unit-prod: - needs removing $\varepsilon$-prod first
  - could create useless symbols
  - cannot create $\varepsilon$-prod.

$\Rightarrow$ The right order for removal is

1) $\varepsilon$-productions
2) unit-productions
3) useless symbols: first non-generating
   then unreachable

## Chomsky Normal Form

<u>Definition</u>: A CFG $G$ is in Chomsky Normal (CNF) if all its productions are of the form

$$A \to a \qquad \text{with} \quad a \in V_T$$
$$A \to BC \qquad A, B, C \in V_N$$

Given a CFG $G$, we can always construct a CFG $G_C$ that is in CNF and such that $\mathcal{L}(G_C) = \mathcal{L}(G) \setminus \{\varepsilon\}$.

Note: since a CFG in CNF cannot generate $\varepsilon$, if $G$ generates $\varepsilon$, then we cannot have that $\mathcal{L}(G_C) = \mathcal{L}(G)$. However, apart from $\varepsilon$, the two languages are equal.