# Closure Properties of CFL's

The class of context-free languages is closed under these three operations: Union, Concatenation, Kleene Star

Assumptions:

Let $G_1=(V_1,T_1,P_1,S_1)$ and $G_2=(V_2,T_2,P_2,S_2)$

be two CF grammars. Assume the sets of variables, $V_1$ and $V_2$ are disjoint.

## Union

A grammar for the union $L(G_1) \cup L(G_2)$ is

$G=(\{S\} \cup V_1 \cup V_2, T_1 \cup T_2, P, S)$

where P consists of productions in $P_1$ and $P_2$ together with $S \rightarrow S_1 \mid S_2$

## Concatenation

A grammar for the concatenation $L(G_1)L(G_2)$ is

$G=(\{S\} \cup V_1 \cup V_2, T_1 \cup T_2, P, S)$

where P consists of productions in

$P_1$ and $P_2$ together with $S \rightarrow S_1S_2$.

## Kleene Star

A grammar for $L(G_1)^*$ is

$G=(\{S\} \cup V_1, T_1,P,S)$

where P consists of productions in $P_1$ together with $S \to \Lambda \mid SS_1$

qed

## Negative result for Complement, Intersection

The class of context-free languages is *not* closed under these two operations: Complement, Intersection

**Proof.** The language

$L_1 = \{a^i b^i c^j \mid i,j \geq 0\} = \{ a^i b^i \mid i \geq 0\} \bullet c^*$

being the concatenation of two CFL's is CFL itself.

Similarly, $L_2 = \{ a^j b^i c^i \mid i,j \geq 0 \}$ is a CFL.

However, $L_1 \cap L_2 = \{a^i b^i c^i \mid i \geq 0\}$ is not a CFL, as we saw last time.

Since the intersection can be expressed in terms of union and complementation $A \cap B = \text{Comp}(\text{Comp}(A) \cup \text{Comp}(B))$ , it follows that the class of CFL's is not closed under complementation.

## Mixtures of CFL and RE

**Theorem**. Intersection of any context-free language with any regular language is context-free.

*Proof Idea*. Product construction. Take a PDA for the first language and a DFA for the second. Construct a PDA for the intersection by taking for its states the set of all pairs of states of the first two automata. Etc.

qed

Note that there is no sensible definition of the product of two PDA's: we cannot combine two stacks into one.

## Chomsky Normal Form

There are many CFG's for any given CFL. When reasoning about CFL's, it often helps to assume that a grammar for it has some particularly simple form.

Here are some ideas how CFG's can be simplified.

## Useless Symbols

A *useful* symbol (terminal or variable) X must be

1. *generating*: X $\Rightarrow^*$ w for some w $\in$ $T^*$ (I.e. w is all terminal symbols)
2. *reachable from S*: S $\Rightarrow^*$ $\alpha X \beta$ for some $\alpha,\beta \in (V \cup T)^*$

An algorithm for elimination of useless symbols first eliminates non-generating ones, then eliminates those not reachable from S.

The order is important, because, for example, when S $\Rightarrow^*$ $\alpha X \beta$ and $\alpha$ contains a non-generating symbol, then X is reachable, but will become unreachable after elimination of non-generators.

## Algorithm: Part 1

We describe the algorithm on an example grammar:

$S \rightarrow AB \mid C$
$A \rightarrow 0B \mid C$
$B \rightarrow 1 \mid A0$
$C \rightarrow AC \mid C1$

*1. Elimination of non-generators*

0 and 1 are in. (because 0 and 1 are terminal)
$B \rightarrow 1$, says B is in.
$A \rightarrow 0B$, says A is in.
$S \rightarrow AB$, says S is in.
Nothing more can be added.

Thus, C can be eliminated, along with any productions containing it. The result is this grammar:

$S \rightarrow AB$
$A \rightarrow 0B$
$B \rightarrow 1 \mid A0$

## Algorithm: Part 2

*2. Elimination of non-reachables*

S is in. (since it is the start symbol)

A and B are in.

0 and 1 are in.

Nothing more can be added.

There is nothing left to eliminate.

$$\boxed{\begin{array}{l} S \rightarrow AB \\ A \rightarrow 0B \\ B \rightarrow 1 \mid A0 \end{array}}$$

In this case, the end result is the same grammar we used as input to this part of algorithm.

## $\Lambda$-Productions

A variable A is *nullable* if $A \Rightarrow^* \Lambda$. We can modify a given grammar G and obtain a grammar G' in which there are no nullable variables and which satisfies L(G') = L(G) - $\{\Lambda\}$.

Find nullable symbols iteratively, using these facts:

1. If $A \rightarrow \Lambda$ is a production, then A is nullable.
2. If $A \rightarrow B_1B_2 \ldots B_k$ is a production and $B_1, B_2, \ldots, B_k$ are all nullable, then A is nullable.

Once nullable symbols are known, we get G' as follows:

1. For every production $A \rightarrow \alpha$, add new productions $A \rightarrow \alpha'$ , where $\alpha'$ is obtained by deleting some (or all) nullable symbols from $\alpha$.
2. Remove all productions $A \rightarrow \Lambda$

**Example**. If G contains a production $A \rightarrow BC$ and both B and C are nullable, then we add

$A \rightarrow B \mid C$

to G'.

## Unit Productions

These are of the form $A \rightarrow B$, where A,B are variables.

Assuming the grammar has no $\Lambda$–productions, we can eliminate unit productions as follows.

1. Find all pairs of variables such that $A \Rightarrow^* B$. (This happens iff B can be obtained from A by a chain of unit productions.)
2. Add new production $A \rightarrow \alpha$ whenever $A \Rightarrow^* B \Rightarrow \alpha$.
3. Remove all unit productions.

## Chomsky Normal Form defined

A grammar is in *Chomsky normal form* (CNF) if it has no useless symbols and all its productions have one of these two forms:

1. $A \rightarrow BC$, where B,C are variables
2. $A \rightarrow a$, where a is a terminal

**Theorem**. For every CFG G, there exists a CFG G' in CNF such that L(G')=L(G) - {$\varepsilon$}

The first three steps of getting G' are elimination of $\Lambda$-productions, elimination of unit productions, and elimination of useless symbols (in that order). There remain two steps:

1. Arrange that all productions are of the form A $\rightarrow \alpha$, where $\alpha$ is a terminal, or contains only variables.
2. Break up every production A $\rightarrow \alpha$ with | $\alpha$ |>2 into productions whose rhs has length two.

For the first part, introduce a new variable C for each terminal c that occurs in the rhs of some production, add the production $C \rightarrow c$ (unless such a production already exists), and replace c with C in all other productions.

For example, the production $A \rightarrow 0B1$ would be replaced with $A_0 \rightarrow 0$, $A_1 \rightarrow 1$, $A \rightarrow A_0BA_1$.

An example explains the second part. The production $A \rightarrow BCDE$ is replaced by three others,

1. $A \rightarrow BA_1$,
2. $A_1 \rightarrow CA_2$,
3. $A_2 \rightarrow DE$,

using two new variables $A_1$, $A_2$.

# Example

To bring the grammar: $S \to SS \mid (S) \mid \Lambda$

into CNF, we first eliminate the only $\Lambda$-production and get

$S \to SS \mid (S) \mid ()$

There are no unit productions and no useless symbols. We need to introduce new variables for both terminals, so we get the grammar

$S \to SS \mid LSR \mid LR$
$L \to ($
$R \to )$

Finally, we need to take care of the (only) long production $S \to LSR$, and the result is

$S \to SS \mid LA \mid LR$
$L \to ($
$R \to )$
$A \to SR$