CS 3313
Foundations of Computing:

Simplification and Conversion to CNF

http://gw-cs3313-2021.github.io

The Membership Problem

- Does a string belong to a CFG; or equivalently, does a CFG generate a string?
  - In programming: if not, then there are syntax errors (w.r.t the grammar) in the program (the string)

- We can check manually
  - Programs with thousands of lines; with various levels of nested structures; etc.

- Or through some automation procedures, e.g., **Parsing**
  - How an IDE tells you where an error/warning occurs.

Simplification

- However, CFGs do not impose restrictions on the RHS of the production rules.
  - Redundancies, useless productions, rules complicate parsing tree;
  - $O(|P|^n)$ complexity to determine exhaustively for a string with length n.

- **Simplify** the rules: three-step procedure
  - Removing λ-productions
    - If $A \to \lambda$ and $B \to \lambda$, then so is AB.
    - If $A \to B$ and $B \to \lambda$, then $A \to \lambda$
  - Removing unit-productions
  - Removing non-terminating or non-reachable variables
- **Next step**: converting the rules to a certain “form”.
- **Before** we apply our automation procedures.

Chomsky Normal Form

- **Def**: A CFG $G = (V, T, P, S)$ is in Chomsky Normal Form (CNF) if all productions are of the form
  - $A \rightarrow BC$, or
  - $A \rightarrow a$,
- where $A, B, C \in V$, and $a \in T$.

- **Benefit**: Parsing tree for $w \in G$ becomes a binary tree.

CNF

- G_1 with production rules:
  - $S \rightarrow AS \mid a$
  - $A \rightarrow SA \mid b$
- Is G_1 in CNF?

- G_2 with production rules:
  - $S \rightarrow AS \mid AAS$
  - $A \rightarrow SA \mid aa$
- Is G_2 in CNF?

CNF Construction-1

- **Theorem 6.6**: Any CFG $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ in CNF.
- *Proof* by constructing $\hat{G}$ for arbitrary G that has no λ or unit productions [form the simplification algorithms].

❖ **Step 1**: Constructing $G_1 = (V_1, T, S, P_1)$ from G by considering all productions P in the form

$$A \to x_1 x_2 \dots x_n$$

where each x_i is either in V or T.

CNF Construction-2

- $A \rightarrow x_1 x_2 \dots x_n$

- If $n = 1$, then x_1 must already be a terminal, since we do not have unit productions.
  - In the case, let P be P_1.

- Otherwise, in V_1, we introduce new variables B_a for each $a \in T$, and $B_a \rightarrow a$ is put into P_1.

- Then, for each A, we put into P_1 the production

$$A \rightarrow C_1 C_2 \dots C_n$$

where $C_i = x_i$ if $x_i \in V$, and $C_i = B_a$ if $x_i = a$.

CNF Construction-3

- This part of the algorithm removes all terminals from productions whose RHS has length greater than one, replacing them with newly introduced variables.
- At the end of this step, we have a grammar G_1 with all its productions in the form of either
  - $A \rightarrow a$ The B_a's
  - or $A \rightarrow C_1 C_2 \dots C_n$, where $C_i \in V_1$.

✓ It is easy to see that $L(G_1) = L(G)$.

CNF Construction-4

- **Step 2**: Constructing $\hat{G}$ by reducing lengths of the RHS of rules in G_1 when necessary.

- First, from P_1, we put all productions in the form of $A \to a$ or $A \to C_1 C_2$ into $\hat{P}$.

- For rules with $A \to C_1 \ldots C_n, n > 2$, we introduce new variables $D_1, D_2, \ldots$ and put into $\hat{P}$ the productions
  - $A \to C_1 D_1$
  - $D_1 \to C_2 D_2 \ \ldots \ \ldots$
  - $D_{n-1} \to C_{n-1} C_n$, where each $A, D_1, \ldots, D_{n-1}$ is in CNF.

✓ It is easy to see that $\hat{G}$ is in CNF, and $L(\hat{G}) = L(G)$.

CNF Construction-Example

- Consider G with production rules:

$$S \to ABa \quad A \to aab \quad B \to Ac$$

- First of all, no λ or unit or useless productions.
- **Step 1**: For G_1, we add $S \to ABB_a \quad A \to B_aB_aB_b \quad B \to AB_c$ and $B_a \to a \quad B_b \to b \quad B_c \to c$ into P_1.

- **Step 2**: For $\hat{G}$, we add $S \to AD_1 \quad D_1 \to BB_a \quad A \to B_aD_2 \quad D_2 \to B_aB_b \quad B \to AB_c$ and $B_a \to a \quad B_b \to b \quad B_c \to c$ into $\hat{P}$.

Scratch

- $P: S \rightarrow ABa \quad A \rightarrow aab \quad B \rightarrow Ac$
- **Step 1**:

Scratch

- P_1: $S \to ABB_a \quad A \to B_aB_aB_b \quad B \to AB_c \quad B_a \to a \quad B_b \to b \quad B_c \to c$
- **Step 2**:

In-Class Exercise

- Convert the following grammar to CNF:
  - $S \to PSQ$
  - $P \to aPS \mid a \mid \lambda$
  - $Q \to SbS \mid P \mid bb$

HW4 Hints

- P1. Regular Grammar to DFA/NFA: **Theorem 3.3** (last lab)
- P2. Find a left-linear grammar: **Example 3.13**

- P3. (f). $\{w \in \{a,b\}^* \mid n_a(w) = 3n_b(w),\ n_a(w), n_b(w) \geq 0\}$: **Example 5.4**., $\{a^*b^* \mid n_a(w) = n_b(w)\}$
  - $S \rightarrow aSb \mid SS \mid \lambda$
  - How do we modify the rules?
  - Recall $\{w \in \{a,b\}^* \mid n_a(w)\ mod\ 3 = 0\}$ from HW2.

HW4 Hints

- P3. (e). Consider the matching symbols on two sides of the equation and construct these matching rules. Then integrate these rules together.
  - <u>Example</u>: For every d, we need ________ to balance the constraint equation.
  - $S \rightarrow$
  - … …
  - $P \rightarrow$
  - $Q \rightarrow$
  - $R \rightarrow$

HW4 Hints

- P3. (g). $\{w \in \{a, b, c\}^* \mid n_a(w) + n_b(w) \neq 3n_b(w), n_a(w), n_b(w), n_c(w) \geq 0\}$:
  - Establish equality cases; then add either a/b or c.
  - How to add? Add before, or after, or in-between: $TS \mid ST \mid STS$, where T can either add a/b or c, and S holds the equality but can also have $SS \mid STS$.
  - T can add any number of a/b's; the other direction, any number of c's
  - T can also add say 1 c and 2 a's: two c's and six a/b's → three c's and eight a/b's.