Transforming Grammars

- Main idea to transform very "broad" CFG into some restricted forms for ease of manipulation.

- Getting L from L-\{\Lambda\}
  
  From a G for L-\{\Lambda\} as \( (V, T, S, \rightarrow) \)
  
  we get \( G' \) for L by adding a start variable \( S_0 \)
  and also \( S_0 \rightarrow S \mid \Lambda \)

- For CFGs, we can obtain CFGs \( \hat{G} \) s.t.
  
  \( L(G) = L(\hat{G}) - \{\Lambda\} \) (will show later)

Substitution Rule

- **CFG:** \((V, T, S, \rightarrow)\) contains
  
  \[ A \rightarrow x, B \rightarrow z \]  
  and \( B \) is in left-hand side of rules

- Substitution \( x \rightarrow y_1 y_2 \cdots y_n \) for rule \( A \rightarrow x \)

- \( \hat{G} = (V, T, \hat{S}, \hat{\rightarrow}) \) where \( \hat{S} \) replaces rule \( A \) by
  
  \[ \hat{A} \rightarrow y_1 y_2 \cdots y_n z \]

  then
  
  \[ L(\hat{G}) = L(C(G)) \]

- \( \hat{G} = \{(A \rightarrow B, \{a, b, c, A, B\}, \hat{\rightarrow})\} \)

Notice that the rule \( B \rightarrow \) is useless (can't be used for derivation of any string)

(\( \hat{G} \)) \( S \rightarrow ASB \mid \Lambda \A \); \( A \rightarrow AA \)

\( S \rightarrow A \) and \( A \rightarrow AA \) are useless. \( A \) is a useless variable

As it is not possible to produce a string

**Def:** A variable \( \hat{A} \in V \) is useful if \( \exists \quad \hat{C} \in \mathcal{L}(\hat{G}) \) s.t.

\( S \rightarrow \hat{A} \hat{G} \)

otherwise it is useless. A production using any useless variables is useless.
\( S \rightarrow A \); \( A \rightarrow aA \lambda \); \( B \rightarrow bA \)

\( B \) is useless; \( S \) is not reachable

\( s \rightarrow as | A | C \); \( A \rightarrow a \); \( B \rightarrow aa \); \( C \rightarrow acb \)

Remove \( C \); \( C \) can't lead to a terminal symbol.

New: \( s \rightarrow as | A \); \( A \rightarrow a \); \( B \rightarrow aa \)

**Dependency Graph** (for \( CFG \))

\[ \begin{align*}
S & \rightarrow A & B \\
& & & \text{vertices are labels} \\
& & & \text{edge indicates production} \\
& & & B \text{ is useless since no path from } S.
\end{align*} \]

**Proof:**

**Part 1:** From \( CFG = (V, T, S, P) \), create \( CFG_1 = (V_1, T_2, S_1, P_1) \)

- \( V_1 \) should contain only one variable \( A \) if \( A \in V \), then \( A \rightarrow \text{wt}^* \)
  
  a) \( V_1 = \emptyset \)
  
  b) Repeat till no more variables can be added to \( V_1 \)

  \( \exists A \in V \) s.t. \( \exists P \in (A \rightarrow x_1 x_2 \ldots x_n) \) \( \forall x_i \in (V \cup T) \)

  then add \( A \) to \( V_1 \)

  c) Take \( P \) as all productions in \( P \) whose symbols are in \( (V \cup T) \)

**Notes:**

- This procedure terminates

  - If \( A \in V_1 \) then \( A \rightarrow \text{wt}^* \)

  - Every \( A \) for which \( A \rightarrow \text{wt} \) is added to \( V_1 \)

**Part 2:** Draw the variable dependency graph for \( CFG_1 \), and

- Remove all variables and corresponding productions not reachable from \( S \).
Define: Any production \( A \rightarrow \lambda \) is called a \( \lambda \)-production. Variable \( A \) if \( A \rightarrow \lambda \) is called nullable.

A grammar may generate a \( L \) not containing \( \lambda \) yet have \( \lambda \)-productions or nullable variables. \( \lambda \)-productions can be removed for these.

Example:

1. \( G: S \rightarrow aSb, S \rightarrow \lambda \)
2. \( L(G) = \{a^n b^m : n \geq 1\} \) is a \( \lambda \)-free language
3. \( G': S \rightarrow aSb|\lambda; S \rightarrow aSb|ab \rightarrow L(G) = L(G')\)

Theorem: If \( G \) is a CFG such that \( \lambda \notin L(G) \), then \( \exists \hat{G} \) having no \( \lambda \)-production.

**Construction Step:**
1. \( V_0 = \emptyset \)
2. For all productions \( A \rightarrow \lambda \), put \( A \) into \( V_0 \).
3. Repeat till no new additions to \( V_0 \).
   - For all productions \( A \rightarrow A_1A_2...A_n \),
     - For all productions \( B \rightarrow A_1A_2...A_n \), \( B \) into \( V_0 \).
     - Where \( A_1, A_2, ..., A_n \) are in \( V_0 \), but \( B \) into \( V_0 \).

To construct \( \hat{G} \) from \( G \) for \( A \in V_0 \):
1. For all productions \( A \rightarrow x_1x_2...x_m \), where each \( x_i \in VUT \).
   - For each such production of \( G \), put into \( \hat{G} \)
     - 1) that production and
     - 2) those generated by replacing nullable variables on the right-hand side with \( \lambda \) in all possible combinations.

If all \( x_i \) are nullable, don't put \( A \rightarrow \lambda \) into \( \hat{G} \).

**Definition:** Any \( \tilde{\rho} \) in a CFG, \( A \rightarrow B \), \( A, B \in V \) is called a unit production.

We can use substitution rules to replace all unit productions.

**Procedure:** Substitution directly might not work.
1. Draw dependency graph showing edges for \( A \rightarrow B \) as \( \underline{A \rightarrow B} \).
2. Generate \( \tilde{\rho} \) by first putting into \( \hat{\rho} \) all non-unit productions of \( \rho \).
3. If \( A \rightarrow B \), then add to \( \hat{\rho} \)
   - \( A \rightarrow \beta_1 \beta_2 \ldots \beta_m \), where \( \beta_i \rightarrow y_i | y_i \ldots y_i \) is the set of rules in \( \beta_i \).
new unit productions are:

\[ S \rightarrow AA \]
\[ A \rightarrow aBc \]
\[ B \rightarrow ab \]

new rules:

\[ s \rightarrow A \]
\[ \alpha \rightarrow aBcBB \]
\[ A \rightarrow bb \]
\[ A \rightarrow abc \]

\[ \beta: \]
\[ s \rightarrow A \]
\[ aBcBB \]
\[ A \rightarrow abc \]
\[ B \rightarrow ab \]

Thus, let \( L \) be a CFL w/o unit \( \alpha \) then \( \exists \) CFG \( (L) \) w/o unit useless productions, \( \alpha \)-productions or unit productions.

Sequence:
1) Remove \( \alpha \)-productions
2) " unit - "
3) " useless "

Chomsky Normal Form (CNF)

A CFG has the Chomsky Normal Form if \( P \) are like:

\[ A \rightarrow BC \] or \[ A \rightarrow a \]

where \( A, B, C \in V, \alpha \in T \)

Any CFG with \( A \notin L(CF) \) has an equivalent CNF form.

New: All productions consist of (already done here).

**Example:**
\[ S \rightarrow ABA, A \rightarrow aab, B \rightarrow AC \]

Step (a): Remove all \( \alpha \) and unit productions (already done here).

Step (b): Introduce new variables \( B_a, B_b, B_c \) as follows:

\[ S \rightarrow ABBa, A \rightarrow B_aB_bB_c, B \rightarrow AB_c, B_a \rightarrow a, B_b \rightarrow b, B_c \rightarrow c \]

Step (c): Introduce additional variables as follows:

\[ S \rightarrow A_D, D \rightarrow B_aB_b, A \rightarrow B_cD_c \]

Greibach Normal Form (GNF)

A CFG is in GNF if all productions are \( A \rightarrow a \alpha, \alpha \in T \times \{ \epsilon \} \). (More general than S-grammar)

Greibach Normal Form (GNF)

\[ A \rightarrow \alpha \in \{ \epsilon \} \]

Therefore, every CFG with \( \alpha \notin L(CF) \), \( \exists \) a CFG in GNF.
Pushdown Automata (PDA)

Non-deterministic PDA \((\text{npda})\)

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, Z, F) \]

- \(Q\): finite set of internal states of CU.
- \(\Sigma\): input alphabet
- \(\Gamma\): finite stack alphabet
- \(q_0\): initial state \(q_0 \in Q\)
- \(F\): set of final states \(F \subseteq Q\)
- \(\delta\): transition function

Transition function \(\delta\):

\[ \delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*) \]

Example:

\[ \delta (q_1, a, b) = \{(q_2, cd), (q_2, f)\} \]

Before:

\[ \cdots \rightarrow a \rightarrow b \rightarrow f \]

After:

\[ \delta \]

Notes:

1. No entry for \(\delta(q_0, b, 0)\) (transition to \(\emptyset\) configuration)
2. Counting a’s \(\delta(q_1, a, 1) = \{(q_1, 1)\}\): add a 1 to stack for each a
3. Counting b’s \(\delta(q_2, b, 1) = \{(q_2, 1)\}\): remove a 1 from stack for each b

L = \(\{a^n b^n : n \geq 0 \} \cup \{a\}\) : accepted language.

Instantaneous Description of a PDA

\((q_1, w, u)\)

- \(q\): state of the control unit
- \(w\): unread part of the input string
- \(u\): stack contents (left most symbol indicating the top)

If \((q_1, y) \in \delta(q_1, a, b)\) then

\((q_1, a w, b x) \rightarrow (q_2, w, y z)\)

"\(\rightarrow\)" indicates a single step move. "\(*\)" can involve arbitrary number of moves.
Let $M = (Q, \Sigma, \Gamma, \delta, \delta_0, Z, F)$ be a npda.

$L(M) = \{ w \in \Sigma^*: \delta(w, Z) \rightarrow^* (p, \lambda, \lambda_1) \mid p \in F, \lambda \in \Gamma^* \}$

is the set of all strings that put $M$ in the final state.

**NPDA + CFL** ($\text{NPDA} \equiv \text{CFL}$)

**Requires** GNF (Greibach Normal Form) of grammar.

**recall:** $A \rightarrow ax, a \in \Sigma^* \text{ for all } a$.

**Given CFG** $S \rightarrow \alpha Sb\beta a \rightarrow \alpha Sb\beta a$ , convert to NPDA

**transform to GNF:**

- $S \rightarrow \alpha Sb\alpha$  $A \rightarrow bB$  $B \rightarrow b$
- Choose & start for $\Delta = \{(a_0, b, b)\}$  $\hat{F} = \{(a_0, b)\}$
- Put start symbol $s$ on stack $\delta(s, a_1, \lambda) = \{(a_0, S)\}$
- $S \rightarrow \alpha$  $\Rightarrow$  remove $S$ from stack, replace it with $\alpha$ while reading $a$ from input
- $S \rightarrow \alpha Sb\beta a$  $\Rightarrow$  remove $S$ while reading $a$
- $\delta(q_1, a, s) = \{(q_1, S\alpha)\}$
- Other productions:
  - $\delta(q_1, b, A) = \{(q_1, b)\}$
  - $\delta(q_1, b, B) = \{(q_1, b)\}$
- Termination: $\delta(q_1, a, \lambda) = \{(q_1, \lambda)\}$ (appearance of the stack start symbol indicates derivation completion.)

- Even if not in GNF form
  - for $A \rightarrow B\alpha$, remove $B$ from stack and replace with $B\alpha$ (no input)
  - for $A \rightarrow ab\alpha$, match $ab$ with similar strings in stack + even
    replace $A$ with $\alpha$.

**NPDA \rightarrow CFG**

**Requirements for NPDA**

(These are not restrictive, i.e., any NPDA \equiv NPDA w/ true reports.)

1. Single final state $p$ entered if stack is empty
2. Each move increases or decreases the stack by one symbol

$\delta(q_1, a, A) = \{ q_1, c_1, c_2, \ldots, c_n \}$

$c_i = (q_i, A)$ or $c_i = (q_i, BC)$
For transition \( \delta(q_i, a, A) = (q_j, \alpha) \),
use production \( (B_i A q_i) \rightarrow \alpha \)
\( \delta(q_i, a, A) = (q_j, BC) \)
genarate
\( (q_i A q_k) \rightarrow \alpha \), \( q_i B q_k \), \( q_i C q_k \)
production
while \( q_k \) and \( q_l \) take all possible values in \( Q \).

Thus \( \frac{L = L^{(npda)}}{\text{then } L \text{ is a CFL}} \)

**Deterministic PDA (DPDA)**

**Defn:** A PDA \( M = (Q, \Sigma, \Gamma, \delta, q_0, \delta_0, F) \) is a dpda if
it is an npda and
\( \delta(q, a, \gamma) \) contains at most one element, and
1. \( \delta(q, a, \gamma) \) contains at most one element, and
2. if \( \delta(q, a, \gamma) \) is not empty, then \( \delta(q, c, \gamma) \) must be empty
   for every \( c \in \Sigma \).

**D-CFL** \( \exists \) a dpda \( M \) so \( L = L(M) \)

**D-PDA**
- \( L = \{ a^n b^n ; n \geq 0 \} \) is a D-CFL
- \( L = \{ w w^R ; w \in \{aba^+ \} \} \) is a CFL but not a D-CFL

**PDA \neq D-PDA** (unlike DFA and NFA)

PDA are good as parsing devices for DCFLs.