Definition 2.8: A CFG is in Chomsky normal form if every rule is of the form $A \rightarrow BC$ or $A \rightarrow a$. Let $S$ be the start symbol. $S$ can only appear on the left-hand side, we allow the rule $S \rightarrow \epsilon$. Note that our previous definition did not have $\epsilon$ but did allow the right-hand side to have any number of terminals and variables (including zero of them). CNF is a more restrictive grammar, which will make some proofs easier.

Chomsky Normal Form

Overview

- CNF
- Equivalence
- Definition
- Pushdown Automata
- CNF
**Proof**

- Step 1: add a new start variable $S_0$ and the rule $S_0 \rightarrow S$

- Start state is just on left-hand-side

- \[ \text{Example: } R \rightarrow uAv ] (u a string of terminals and variables without } A) \]

- Add extra versions of rules where $A$ is on right-hand-side, without the $A$

- Remove rules $A \rightarrow \epsilon$ where $A$ is not $S_0$

- Remove extra versions of rules where $A$ is on right-hand-side

- Add a new start variable $S_0$ and the rule $S_0 \rightarrow S$

**Theorem 2.9:** Any CFL is generated by a CFG in Chomsky normal form.
Example 2.10

Step 1: Add new start state: 

\[ \epsilon \rightarrow B \]

\[ B \rightarrow A \]

\[ B \rightarrow b \] or \[ \epsilon \]

Proof (continued)

- Step 3: Unit rules:
  - Remove each rule \[ A \rightarrow B \]
  - For each rule \[ B \rightarrow u \] (where \[ u \] is a string of terminals and variables)
    - Add \[ A \rightarrow u \] unless this is a unit rule that was already removed
    - Remove each rule \[ B \rightarrow \] (where \[ B \] is a string of terminals and variables)

- Step 4: Other rules:
  - A \[ \rightarrow u_1 u_2 \ldots u_k \] where \[ k \geq 3 \]
    - Where each \[ u_i \] is a variable or terminal
    - Replace with binary rules
      - A \[ \rightarrow u_1 A_1 \]
      - A_1 \[ \rightarrow u_2 A_2 \]
      - \ldots
      - A_k \[ \rightarrow u_{k-1} u_k \]
    - Where \[ A_i \]’s are new variables

- Step 4: Other rules:
  - A \[ \rightarrow u_1 u_2 \] where \[ u_1 \] or \[ u_2 \] is a terminal
    - Replace terminal \[ u_i \] with a new variable, say \[ U_i \]
    - Add rule \[ U_i \rightarrow u_i \]
Step 3: Remove unit rules

- If S goes to something, add a rule with L goes to it.
  \[ S \rightarrow L \]
  Remove rule •

- If S goes to something, add a rule with \( \epsilon \) goes to it.
  \[ S \rightarrow \epsilon \]
  Remove rule •

- If B goes to something, add a rule with \( \epsilon \) goes to it.
  \[ B \rightarrow \epsilon \]
  Remove rule •

- Anything that \( \epsilon \) goes to, S already goes to it, so just remove it.

Step 2: Remove epsilon rules

- Get rid of A \[ \epsilon \]
- Get rid of B \[ \epsilon \]

Step 2: Remove epsilon rules
Step 4: Rules with 2 on right side & terminals

• Remove $S \rightarrow aB$

• Remove $T \rightarrow aB$

Step 4: Rules with more than 2 on right side & terminals

• Replace $T \rightarrow ASA$

• Replace $S \rightarrow ASA$
Pushdown Automata

- Can push onto the top of the stack
- Transitions to another state (and read it)
- Can pop off the top of the stack
- Has an input head, which it can read
- Decides what to do based on:
  - Functions in the control
  - Beyond finite memory
  - Provides extra memory

Like a NFA, but has a stack as well.

Overview

- CNF
- Definition
- Equivalence

Pushdown Automata $\equiv$ CNF
Non-determinism

We will focus on non-deterministic push-down automata, which are not so with push-down automata. NFA and DFAs have the same power, they recognize the same classes of languages.

Example:

0^n 1^n

- Pushdown automata can read the 0's and push them onto the stack.
- Can pop off the 0's from the stack as it reads the 1's.
Formal Definition

A pushdown automaton is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where $Q$, $\Sigma$, and $F$ are all finite sets, and

1. $Q$ is the set of states
2. $\Sigma$ is the input alphabet
3. $\Gamma$ is the stack alphabet
4. $\delta$ is the transition function
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of accept states

$\delta$ is the transition function, which is a 6-tuple $(Q, \Sigma, \Gamma, F, q_0, F)$, where $Q$, $\Sigma$, $\Gamma$, $F$, and $q_0$ are all finite sets.

Overview

- CNF
- Pushdown Automata
- Equivalence
- Definition
- Pushdown Automata
- CNF
A pushdown automaton \( M = (Q, \Sigma, \Gamma, \delta, q_0, F) \) accepts \( w \) if \( w \) can be written as \( w = w_1w_2...w_m \), where each \( w_i \in \Sigma \) and there is a sequence of states \( r_0, r_1, ..., r_m \in Q \) and a sequence of stack contents \( s_0, s_1, ..., s_m \in \Gamma^* \) such that

1. \( r_0 = q_0 \) (\( M \) starts in start state)
2. \( s_0 = \epsilon \) (\( M \) starts with an empty stack)
3. For \( i = 0, ..., m-1 \), we have \((r_i, w_{i+1}, a) \in \delta(r_i, w_i, \epsilon), s_i = at\) and \( s_{i+1} = bt \), for some \( a, b \in \Gamma \) and \( t \in \Gamma^* \).
4. \( r_m \in F \) - Stack formalization is a bit tricky

Transition Function \( \delta \):

- Transition function decides what to do based on (inputs)
- What state it is in
- Can look at the input (and consume it) or not (\( \epsilon \))
- Can look at the stack (in which case it pops it)
- Can ignore the stack (not pop anything off)
- Does this by specifying what to do on a certain stack value
- Can look at the stack (in which case it pops it)
- Can ignore the stack (not pop anything off)

Transition function outcomes:

- New state
- Can push onto stack or not (\( \epsilon \))
- Specified by reading \( \epsilon \) on the stack
- Does this by specifying what to do on a certain stack value
- Can look at the stack (in which case it pops it)
- Can ignore the stack (not pop anything off)
- What state it is in

To prepare for next slide, draw this figure.
Go over special cases where some parts might be $\epsilon$.

Talk about if stack has $s$ on it beforehand.

After transition, stack is $\beta$

For $w + t_d$, $q \leftarrow q' n \Rightarrow \beta' t_b$.

Give formal definition.

Make sure initial ends properly.

Give state diagram (with just two states). Don't worry about how it ends.

Example
How is the nondeterminism captured?

- Does it know when it is at the end of the input?
- Does it know when the stack is empty?
- Can it pop on an empty stack?
- Can it not look at the stack at all during a transition?
- Can it pop and push in the same transition?
- Must stack be empty at acceptance?
- Can it pop on an empty stack?

Example: \( w^m v^n \) where \( m, n \in \mathbb{N} \)

Give state diagram:
Running out of Stack or Input

- Formal definition of PDA does not have a mechanism to allow a PDA to test for an empty stack
  - Can add ‘$’ to stack alphabet
  - Have an initial state push this on stack
  - Add extra states to test for this

- Formal definition of PDA does not allow it to check for end of input
  - Not needed as accept state only takes effect when machine is at end of input

Equivalence with CFG

**Theorem:** A language is context free if and only if some pushdown automaton recognizes it.
Overview

• CNF
• Pushdown Automata
• Definition ⇒ Equivalence

Example

• Give a derivation of \( \text{aabbb} \) with
  \[ S \rightarrow \text{a}S\text{b} | \text{c} | \varepsilon \]

  • Trace stack operations
  \[ S \rightarrow \varepsilon \text{shovel} \text{a} \text{a} \text{a} \text{b} \text{b} | \varepsilon \]
Lemma: If language is context free, some PDA recognizes it.

• Let $A$ be a CFL, so there is a CFG $G$ that generates it.

• Idea: Construct PDA $P$ that simulates doing a derivation in $G$.

Rules

- If top of the stack is a variable $A$, pop it, and non-deterministically choose a rule with $A$ on left-hand side, and push the right-hand side onto the stack.

- If top of the stack is a terminal, pop it, and read that character from input.

- Use stack to hold the current derivation, starting with $S$.

- Final transition returns to the main looping state.

- For each rule $A \rightarrow x_1x_2\ldots x_k$, where $x_i$ are terminals or symbols

  - Have a sequence of states in which the first state is transitioned to if the top of the stack is $A$ (and pops it) and pushes $z$ onto the stack.

  - If top of the stack is a variable $A$, pop it, and push the right-hand side onto the stack.

- Final transition returns to the main looping state.
So far

Make a PDA that does the following for CFG $G$

- Place $\$'$ on top of stack
- Place start variable on top of stack
- Repeat the following forever
  - If the top of stack is a $\text{variable}$, pop it, and non-deterministically choose a rule with $\text{variable}$ on left-hand side, and push the right-hand side onto the stack
  - If top of the stack is a $\text{terminal}$, pop it and read that character from input
    - If top of the stack is $\$'$, enter the accept state (doing so accepts the input if they don't match, then this branch of nondeterminism dies)