Context Free Grammars: Given by production rules:

$A \longrightarrow 0A1$

$A \longrightarrow B$

$B \longrightarrow \#$

Variables, terminals, and rules: $G = (V, \Sigma, R, S)$.

The language of a grammar G, $L(G)$, is $\{w \in \Sigma^* | S \overset{*}{\Rightarrow} w\}$.

A parse tree is a graphical representation of a derivation. A CFG is ambiguous if it has a string with two different parse-tree derivations.

Chomsky Normal Form: A CFG is in CNF if every rule is of the form $A \longrightarrow BC$ or $A \longrightarrow a$; B, C may not be the start variable. In addition, we permit $S \longrightarrow \varepsilon$.

Theorem: Any CFG is generated by a CFG in CNF.

Proof: Translate a general CFG into CNF:

1. Add $S_0 \longrightarrow S$.

2. If $A \neq S_0$, we eliminate $A \longrightarrow \varepsilon$: replace $R \longrightarrow \alpha A \beta$ by $R \longrightarrow \alpha\beta$ (if there is more than one A, replace all combinations). If there was $R \longrightarrow A$, replace it by $R \longrightarrow \varepsilon$, unless R was *already* dealt with.

3. Eliminate unit rules: remove $A \longrightarrow B$, and whenever $B \longrightarrow u$ appears, replace it by $A \longrightarrow u$, unless it was a unit rule previously removed.

4. Finally: given $A \longrightarrow u_1 u_2 \ldots u_k$, with $k \geq 3$, replace it by a chain: $A \longrightarrow u_1 A_1, A_1 \longrightarrow u_2 A_2, \ldots$. If some u_i is a terminal, replace it in the preceding rules by U_i, and add $U_i \longrightarrow u_i$.

Pushdown Automaton: Like an NFA, but with a stack attached. It has an input alphabet Σ, and also a stack alphabet Γ, and the transition function is:

$$\delta : Q \times \Sigma \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma)$$

Accept by final state or empty stack (initially stack is empty).

Theorem: A language is CF iff some PDA accepts it.

Proof: Suppose that L is CF, so it is generated by a CFG G. We build a PDA P that simulates this generation:

On input w, P pushes $S, \$$ onto the stack. If by end of input the top stack symbol is \$, accept.

Repeat: if top symbol of stack is a variable, then (nondeterministically) select a rule to replace it. If top symbol is a terminal, match it with the next terminal of input & delete both.

For the other direction, suppose that we have a PDA P that accepts a language L. We construct a CFG G for L.

First modify the PDA to be as follows:

1. Single accept state
2. Empties stack before accepting
3. Each transition either pops or pushes the stack, but not both.

Design G so that A_{pq} generates all strings that take P from state p to state q, starting and ending with an empty stack.

On input x, P must first push, and the last operation must be a pop. The stack may or may not be empty during processing x.

If it is not, then use rules $A_{pq} \longrightarrow aA_{rs}b$. If it is, $A_{pq} \longrightarrow A_{pr}A_{rq}$.

Here are the rules of G in detail: suppose that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{\text{accept}}\})$.

Variables of G are: $\{A_{pq} | p, q \in Q\}$; the start variable is $A_{q_0, q_{\text{accept}}}$.

Rules: If $(r, t) \in \delta(p, a, \varepsilon)$ and $(q, \varepsilon) \in \delta(s, b, t)$, then add:

$$A_{pq} \longrightarrow aA_{rs}b$$

For all $p, q, r \in Q$ add the rule:

$$A_{pq} \longrightarrow A_{pr}A_{rq}$$

and for all p, add the rule $A_{pp} \longrightarrow \varepsilon$.

Correctness: $A_{pq} \stackrel{*}{\Rightarrow} x$ iff x takes P from p with empty stack to q with empty stack.

We prove the correctness by induction on length of derivation and length of computation, respectively.

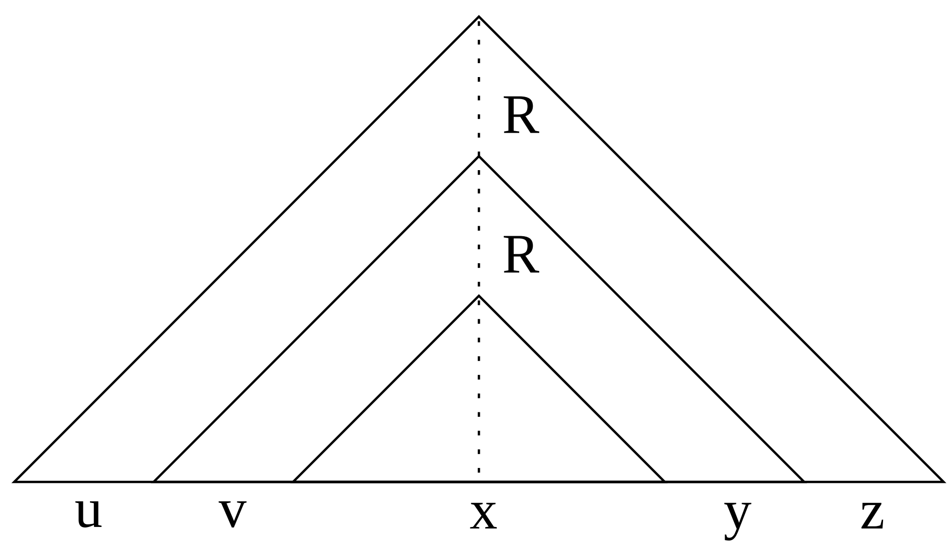
Pumping Lemma for CFLs: There exists a p so that any s, $|s| \geq p$, can be written as $s = uvxyz$, and:

1. uv^ixy^iz is in the language, for all $i \geq 0$,

2. $|vy| > 0$,

3. $|vxy| \leq p$

Proof:



Turing machine (TM): $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$

1. Q, set of states
2. Σ, Γ, input and tape alphabets, $\Sigma \subseteq \Gamma$, and $\sqcup \in \Gamma, \sqcup \notin \Sigma$
3. $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$
4. $q_0, q_{\text{accept}}, q_{\text{reject}}$ the start, accept, reject states

Configuration uqv: current state is q, the tape contains uv, and the head is on top of the first symbol of v.

$$uaq_ibv \qquad \textbf{yields} \qquad \begin{cases} uq_jacv & \text{if } \delta(q_i, b) = (q_j, c, L) \\ uacq_jv & \text{if } \delta(q_i, b) = (q_j, c, R) \end{cases}$$

start/accepting/rejecting/halting configuration.

M accepts w if $\exists$ sequence of configurations $C_1, C_2, \ldots, C_k$ such that $C_1 = q_0w$, C_i yields C_{i+1}, and C_k is accepting.

$L(M)$ is the language of strings accepted by M.

L is **Turing-recognizable (recursively enumerable)** if $\exists$ a TM M such that $L = L(M)$

L is **Turing-decidable (decidable, recursive)** if $\exists$ a TM M such that $L = L(M)$ & M *halts on all inputs.*

A TM that halts on all inputs is also called a **decider**.

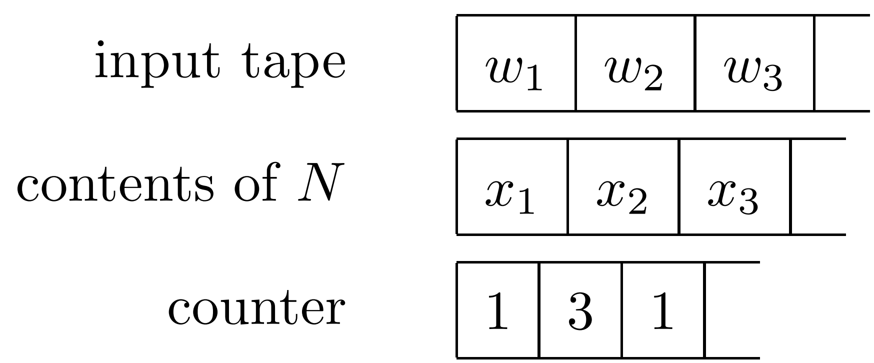
Robustness:

1. Multi-tape TMs
2. Tape infinite in both directions
3. Non-deterministic TMs $\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$
   A string is accepted if *some* branch of the computation accepts (other branches may reject or never halt).

Theorem: Every nondet TM has an equivalent det TM.

Proof: Breadth first search; *not* depth first!

TM with three tapes:



Let b be the *degree of nondet of* N; add $\Sigma_b = \{1, 2, \ldots, b\}$. The 3rd tape generates the lexicographically next string (indicating which branch we are following).

First tape contains input, and it is never altered. The second tape contains the contents of the tape of N resulting from the sequence of choices given on the 3rd tape.

Theorem: L is Turing-recognizable/decidable iff some nondet TM N recognizes/decides it.

Enumerator: Starts with a blank tape, and a work tape, and prints strings on the blank tape (possibly with repetition).

Theorem: L is Turing-recognizable iff some enumerator enumerates it.

Proof: $[\Longrightarrow]$ uses the idea of "dove-tailing."

Church-Turing Thesis: Intuitive notion of algorithm = Turing machines.