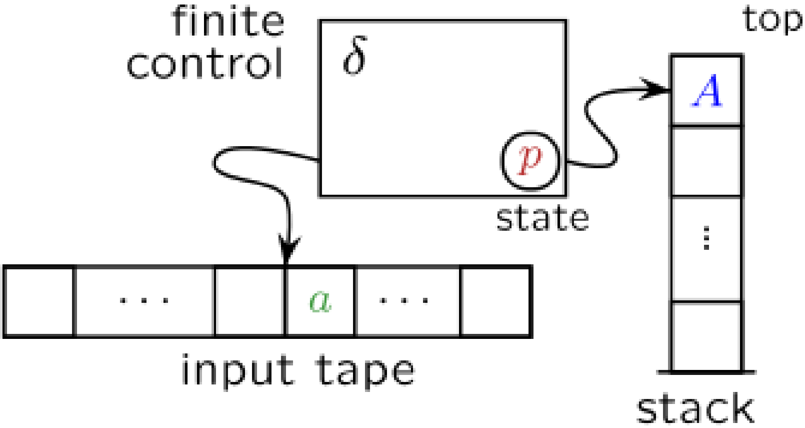
PDA, the Chomsky hierarchy, and the pumping lemma for context-free languages



Textbook: 2.2 and 2.3

PushDown Automata (PDA)

- We have been looking at *finite* state machines
- What if we need to remember an **infinite** amount information?
- Would allow us to match the set of strings containing matching parenthesis
- We can give out state machine an additional memory store in the form of an infinitely large **stack**
- Allow it to push or pop from that stack
- **Not** random access: Last in, first out

nPDA

- Unlike DFA/NFA, **dPDA and nPDA are not equivalent in power**
- Unlike other models, **when we say PDA we mean nPDA not dPDA**
- We will be ignoring deterministic PDA because they are not equivalent to context-free grammars: nondeterministic PDA are

Formal definition

Def: nPDA. A **nondeterministic pushdown automaton (nPDA)** is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, F are all finite sets and:

1. Q is the set of states
2. Σ is the **input alphabet**
3. Γ is the **stack alphabet**
4. $\delta : (Q \times \Sigma_\epsilon \times \Gamma_\epsilon) \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the **nondeterministic transition function**
5. $q_0 \in Q$ is the start state
6. $F \subseteq Q$ is the set of acceptance states

We let \$ be the special symbol indicating that the stack is empty.

Interpreting an nPDA transition

For some mapping

$$\delta(q, \sigma, \gamma) = (q', \gamma')$$

- The current state is q
- The transition is taken when the input is σ (or ϵ for nonconsumptive transitions)
- γ is the item to pop off the stack (or ϵ if we don't want to)
- q' is the new state
- γ' is the item to push to the top of the stack (or ϵ if we don't want to)

Note: ϵ means "don't pop from the stack", \$ means "empty stack". We disallow the removal of \$.

Equivalence with context-free grammars pt. 1

Thm: A language is context free iff some nPDA recognizes it.

Lemma 1: A language being context free implies some nPDA recognizes it. (pg 115)

- By construction
- We show how to use an nPDA to determine if the CFG derives any input string

Lemma 2: Any language recognized by a nPDA is context free. (pg 119)

- By construction
- We show how to create grammar rules from an nPDA in CFG form

Equivalence with context-free grammars pt. 2

Lemma 1: All CFG have equivalent nPDA.

- Each step of a CFG A yields an **intermediate string** of terminals and variables. We design P to show whether some series of substitutions on the starting variable of A can lead to the input string w
- We will keep the current intermediate string partially on the stack, eliminating any literals off the top as if we were a NFA
- We will branch nondeterministically when multiple rules could apply

Note: Pushing any string of finite length to the stack is equivalent to pushing each character at a time.

Equivalence with context-free grammars pt. 3

The following steps create P.

1. Push \$ and A's starting variable onto the stack.
2. Repeat these steps forever:
   a. If the top of the stack is some variable V, nondeterministically select each of the applicable rules $V \to \cdots$ and replace V on the stack
   b. For as long as the top of the stack is a terminal, move through the state machine as if it were an NFA, popping from the stack each time a character matches the input. If a character doesn't match, **reject this branch of the nondeterminism.**
   c. If the stack top is \$, enter the accept state. Doing so accepts the input for the entire nondeterminism if it has all been read.

P is a valid nPDA such that $\mathcal{L}(P) = \mathcal{L}(A)$. End of lemma 1.

Equivalence with context-free grammars pt. 4

Lemma 2: All languages recognized by nPDA are context-free.

First, we assume that our input nPDA has the following properties. If it does not, an equivalent one can be constructed (pf. exclude).

1. It has only one accept state, $q_{\texttt{accept}}$
2. It empties its stack before accepting
3. Each transition either pushes or pops, but not both and not neither

Let $A_{p,q}$ be a variable representing any valid nPDA path from state p to q. Let the entry variable be $A_{q_0,q_{\texttt{accept}}}$. $A_{p,q}$ **will assume the stack is empty and leave it so.** If it pushes anything in its operation, it must pop before completion.

Equivalence with context-free grammars pt. 5

Add the following rules to our CFG G:

1. For every possible combination of $p, q, r \in Q$, add the rule $A_{p,q} \to A_{p,r}A_{r,q}$
2. For every $p \in Q$, add the rule $A_{p,p} \to \epsilon$
3. For every possible combination of states $p, q, r, s \in Q$, stack symbol $t \in \Gamma$, and (possibly ϵ) input characters $a, b, \in \Sigma_\epsilon$:
   - **If** $\delta(p, a, \epsilon)$ contains (r, t) (state p can move to state r **pushing** t on input a) **and** $\delta(s, b, t)$ contains (q, ϵ) (new state s can move to other new state q **popping the same** t on new input b) **then** add the rule $A_{p,q} \to aA_{r,s}b$

- The first condition allows us to concatenate valid rules
- The second condition means that it is free to get from a state to itself (base case)
- The third condition lets us handle stack pushing and popping, as well as input consumption
  - It also ensures that any consumptive rule **maintains the stack size**, ensuring the stack is kept empty at the end

Equivalence with context-free grammars pt. 6

Lemma 2.1: $A_{p,q}$ generates x iff x can bring P from p with empty stack to q with empty stack. If this holds, lemma 2 holds.

By design, rule $A_{p,q}$ brings the machine from state p to state q. Condition 3 of the construction ensured that any time anything is pushed by a rule, that rule must pop it. Thus, the input contents of the stack to a rule are the output. Therefore, $A_{p,q}$ will bring itself from p with an empty stack to q with an empty stack.

A similar proof holds for the other direction of the iff (excluded here).

Equivalence with context-free grammars pt. 7

Since $A_{p,q}$ generates x iff x can bring P from p with empty stack to q with empty stack, $A_{q_0,q_{\texttt{accept}}}$ generates x iff x is recognized by the nPDA. Thus, all nPDA languages are context-free. End of lemma 2.

Since both directions hold, **nPDA and CFG are equivalent.** End of proof.

Corollary: Since NFA are nPDA that ignore their stack, all regular languages are context-free. Since there exists at least one context-free language that is not regular, regular languages are a proper subset of context-free ones.

The Chomsky hierarchy

- The **Chomsky hierarchy** states the power of different models of computation
- Each entry is a linguistic class and corresponds to a type of automata

In increasing power:

1. Finite-state automata / regular languages
2. Pushdown automata / context-free grammars
3. Linear bounded automata / context-sensitive grammars
4. Turing machines / unrestricted grammars

Chomsky hierarchy grammar rules

Grammar	Languages	Recognizing Automaton	Production rules (constraints)[a]	Examples[5][6]
Type-3	Regular	Finite-state automaton	$A \rightarrow \mathrm{a}$ $A \rightarrow \mathrm{a}B$ (right regular) or $A \rightarrow \mathrm{a}$ $A \rightarrow B\mathrm{a}$ (left regular)	$L = \{a^n \mid n > 0\}$
Type-2	Context-free	Non-deterministic pushdown automaton	$A \rightarrow \alpha$	$L = \{a^n b^n \mid n > 0\}$
Type-1	Context-sensitive	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$	$L = \{a^n b^n c^n \mid n > 0\}$
Type-0	Recursively enumerable	Turing machine	$\gamma \rightarrow \alpha$ (γ non-empty)	$L = \{w \mid w$ describes a terminating Turing machine$\}$

Via Wikipedia

The pumping lemma for context-free languages

- Just like we have a pumping lemma to prove that a given language is nonregular, we have another to prove a given language is not context-free
- Very similar to the pumping lemma for regular languages

Def: Pumping lemma for context-free languages. If A is a context-free language, then there is a number p (the pumping length) where, if s is any string $\in A$ such that $|s| \geq p$, then s may be divided into five pieces $s = uvxyz$ such that:

1. For each $i \geq 0, uv^ixy^iz \in A$
2. $|vy| > 0$
3. $|vxy| \leq p$

CFG Pumping lemma proof

Pf: By construction. We will derive a minimal bound for the pumping length p, then show that any string longer than this must have a derivation where some variable R derives itself. This implies that the lemma holds, ending the proof.

Lemma 1: String sizes. Let b be the maximum number of variables on the RHS of any rule in our CFG. Then at most b leaves are within 1 step from the starting variable, b^2 within 2, and b^h within h. Thus, any parse tree of height h produces a string of size at most b^h. Conversely, if a generates string is at least $b^h + 1$ long, each of its parse trees must be at least $h + 1$ high.

If $|V|$ is the number of variables in our CFG, we let $p = b^{|V|+1}$. Thus, any string longer than the pumping length must have parse trees of height at least $|V| + 1$, since $b^{|V|+1} \geq b^{|V|} + 1$.

CFG Pumping lemma proof pt. 2

Since the height of the parse tree is the number of variable replacements, there must be $|V| + 1$ replacements of $|V|$ variables. Thus, there must be a variable which is repeated. This variable derives another instance of itself. Thus, we can replace its first occurrence with its final occurrence or infinitely pump the in-between region while remaining in the language.

Conditions 2 and 3 are proven in the book (pg 125).

CFG Pumping lemma proof pt. 3

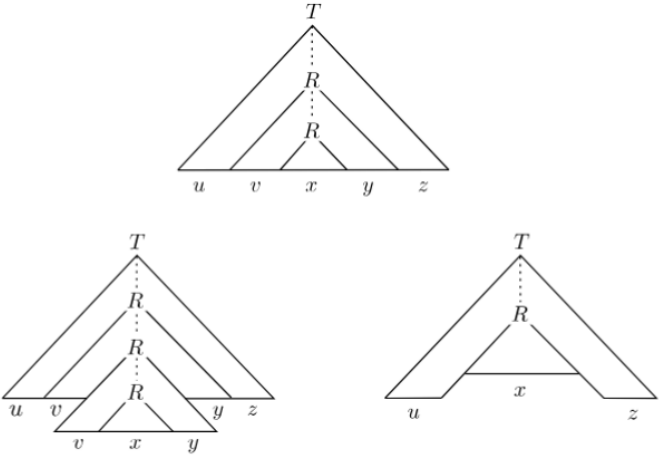


FIGURE 2.35
Surgery on parse trees

Non-context-free languages

Practice: Let $A = \{a^n b^n c^n : n \geq 0\}$. Prove that A is not context free.

- See pg 126 of textbook

Next up: The Church-Turing thesis, and Turing machines

An assignment on part 1 of the textbook should come soon