CSC A890. Parser.

Program file $\xrightarrow{\text{Input String}}$ [Lexical analyzer $\longrightarrow$ Parser] $\xrightarrow{\text{output}}$ machine code

Compiler

Lexical analyzer: Recognizes the lexemes of the input program file: keywords, integers, identifiers. It is built with DFAs.

Parser: knows the grammar of the programming language to be compiled. $\rightarrow$ constructs derivation for input program file $\rightarrow$ converts derivation to machine code.

1. Exhaustive Parser: at most $2|w|$ derivation steps are required to produce w.
   The exhaustive search requires at most $2|w|$ phases.
   Suppose the grammar has k production.
   Then the total exploration choices for string w: $k + k^2 + \cdots + k^{2|w|} = O(k^{2|w|})$ bad!

2. Faster Parsers: S-grammar: $A \rightarrow aV$ (a → symbol, V → string of variables.)
   Each string has a unique derivation, total steps for parsing string w : $|w|$.

The CYK Parsing Algorithm: $O(|w|^3)$. Dynamic Programming.

i). Input: Arbitrary Grammar G in Chomsky Normal Form.
   string w.

ii). Output: Determine if $w \in L(G)$. $O(\underbrace{|w|^2}_{\text{number of substrings}} \cdot \underbrace{|w|}_{\text{number of prefix-suffix decompositions for a string}}) = O(|w|^3)$

E.g.: Given grammar G: $S \rightarrow AB$, $A \rightarrow BB \mid a$, $B \rightarrow AB \mid b$.
Determine if $w = aabbb \in L(G)$.

1. $\underset{a}{\{A\}}$ $\underset{a}{\{A\}}$ $\underset{b}{\{B\}}$ $\underset{b}{\{B\}}$ $\underset{b}{\{B\}}$ $\Leftarrow A \rightarrow a$, $B \rightarrow b$.

2. $\underset{aa}{\text{No}\{AA\}x = \{\}}$ $\underset{ab}{\{S,B\}}$, $\underset{bb}{\{A\}}$ $\underset{bb}{\{A\}}$ $\Leftarrow aa = \{AA\}$, $ab = \{AB\}$, $bb = \{BB\}$. $\Leftarrow aa = \{\ \}$, $ab = \{S, B\}$, $bb = \{A\}$.

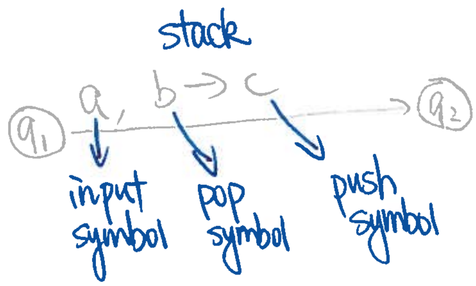
3. $\underset{aab}{\{S,B\}}$. $\underset{abb}{\{A\}}$ $\underset{bbb}{\{S,B\}}$ $\Leftarrow aab = a + ab = \{A\} \cdot \{S,B\} = \{AS, AB\} = \{S,B\}$.

4. $\underset{aabb}{\{A\}}$ $\underset{abbb}{\{S,B\}}$

5. $\underset{aabbb}{\{S,B\}}$

Pushdown Automata (PDAs).

The states:



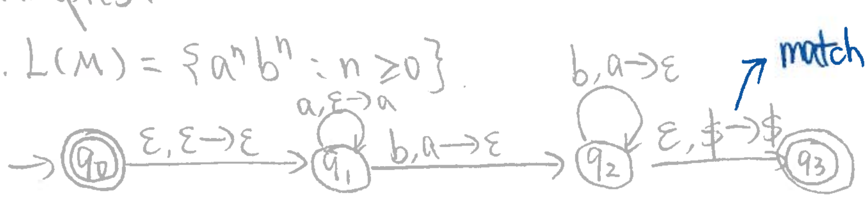
A string is accepted if there is a computation such that: all the input is consumed and the last state is an accepting state.

We do not care about the stack contents at the end of the accepting computation.

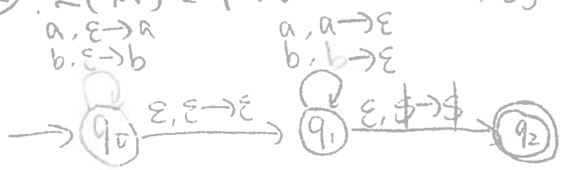
PDAs are non-deterministic, which allow non-deterministic transitions.

Examples:

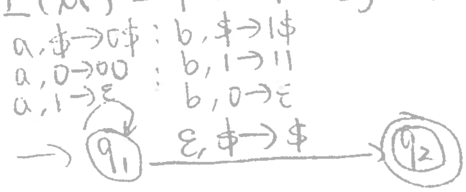
①. $L(M) = \{a^n b^n : n \geq 0\}$.



②. $L(M) = \{vv^R : v \in \{a, b\}^*\}$



③. $L(M) = \{w \in \{a, b\}^* : n_a(w) = n_b(w)\}$



Language $L(M)$ accepted by PDA 为:

$$L(M) = \{w : (q_0, w, z) \xrightarrow{*} (q_f, \varepsilon, s)\}.$$

q_0: initial state

q_f: Accept state

PDAs Accept Context-Free Languages.

theorem: $\left\{\begin{matrix}\text{context-Free} \\ \text{Languages}\end{matrix}\right\} = \left\{\begin{matrix}\text{Languages} \\ \text{Accepted by PDAs}\end{matrix}\right\}$

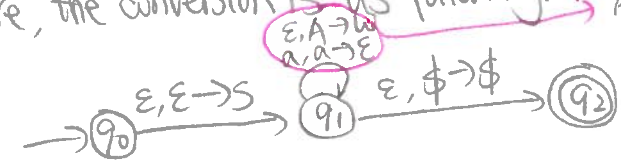
Proof:

step 1: convert context-Free Grammars to PDAs.

Conversion Procedure: For each production in G: $A \to w \Rightarrow \varepsilon, A \to w$.

For each terminal in G: $a, a \to \varepsilon$.

Therefore, the conversion is as following: → All all transitions here



Grammar G generates string w: $S \stackrel{*}{\Rightarrow} w$ if and only if   PDA M accepts w: $(q_0, w, \$) \stackrel{*}{\to} (q_2, \varepsilon, \$)$.

Therefore, $L(G) = L(M)$.

step 2: convert PDAs to Context-Free Grammars.

Pumping Lemma for Context-free Languages

For any infinite context-free language L, there exists an integer p such that for any string $w \in L$, $|w| \geq p$.

we can write $w = uvx\underline{y}z$ with lengths $\underline{|vxy| \leq p}$, and $\underline{|vy| \geq 1}$.

and it must be that:

$$uv^i x y^i z \in L, \text{ for all } i \geq 0.$$

Theorem: The language $L = \{a^n b^n c^n : n \geq 0\}$ is not context free.

Proof: Assume L is context-free, since it is infinite, we can apply the pumping Lemma.

Let p be the critical length of the pumping lemma.

Pick any string $w \in L$ with length $|w| \geq p$.

we pick $w = a^p b^p c^p$.

From pumping Lemma, we can write: $w = uvxyz$. with length $|vxy| \leq p$, $|vy| \geq$

we examine all the possible locations of string vxy in w:

case 1: vxy is in a^p. $\Rightarrow v = a^{k_1}$, $y = a^{k_2}$, with $k_1 + k_2 \geq 1$.

From pumping Lemma, $uv^2xy^2z \in L$, $k_1 + k_2 \geq 1$. $\Rightarrow a^{p+k_1+k_2} b^p c^p \in L$.

However, we got a contradiction.

Case 2: vxy is in b^p, this is similar to case 1.

case 3: vxy is in c^p, this is similar to case 1.

case 4: vxy overlaps a^p and b^p:

i). subcase1: v contains only a. y contains only b. $\Rightarrow$ similar to case 1.

ii). subcase2: v contains a and b, y contains only b. $\Rightarrow v = a^{k_1} b^{k_2}$, $y = b^{k_3}$, we have $k_1 k_2 \geq 1$

From pumping Lemma: $uv^2xy^2z \in L \Rightarrow a^p a^{k_1} b^{k_2} b^{p+k_3} c^p \in L$.

we got contradiction here.

[illegible] contains a and b $\Rightarrow$ similar to subcase 2.

Case 5: vxy overlaps b^p and c^p $\Rightarrow$ similar to case 4.

Case 6: overlaps a^p. b^p, c^p. $\Rightarrow$ impossible since $|vxy| \leq p$.

Therefore, L is not context-free.

Turing Machines.

States & Transitions:

Read, write, move left

$$q_1 \xrightarrow{a \to b,\ L} q_2$$

move right.

$$q_1 \xrightarrow{a \to b,\ R} q_2$$

The head at each transition (time step):

1. Reads a symbol
2. Writes a symbol
3. Moves left or right.

Turing Machines are deterministic. (no ε transitions allowed).

The machine halts in a state if there is no transition to follow.

Accepting states have no outgoing transitions. $q_1 \longrightarrow q_2$ X not allowed.

If machine halts in an accept state: accept input string.

If machine halts in a non-accept state or if machine enters an infinite loop:

Reject input string.

In order to accept an input string, it is not necessary to scan all the symbols of the input string.

The accepted language: for any Turing Machine M:

$$L(M) = \{w : \underbrace{q_0}_{\text{initial state}} w \xrightarrow{*} \underbrace{x_1 q_f x_2}_{\text{Accept state}} \}.$$

If a language L is accepted by a Turing machine M, then we say that L is: Turing recognizable or Turing Acceptable or Recursively Enumerable.