COMPUTATION: DAY 4

BURTON ROSENBERG
UNIVERSITY OF MIAMI

Contents

1. PDA: the push down automata 1
1.1. Some notes on the notation 2
1.2. The language $a^i b^i$ 3
1.3. The language $w\,w^R$ 3
2. Context Free Grammars 4
2.1. The language $a^i b^i$ 4
2.2. The language $w\,w^R$ 5
2.3. The language of balanced parenthesis 5
2.4. Same number a's and b's 6
2.5. Unequal a's and b's 6
3. Context Free Languages 7
3.1. All regular languages are context free 7
3.2. The class of CFL's is a algebra with Kleene star 7
3.3. The class of CFL's is not closed by Intersection 8
3.4. The class of CFL's is not closed by Complementation 9
4. Chomsky Normal Form and the CYK algorithm 10
Appendix A. The Pumping Lemma for CFL's 10
A.1. The language $a^i b^i c^i$ is not context free 10
A.2. The language $w\,w$ is not context free 11
Appendix B. CNF grammar for $a^i b^i c^j$ 11

1. PDA: the push down automata

We are considering the language,

$$L_o = \{\, a^i\, b^i \mid i \geq 0 \,\}$$

which has been shown to be not regular, but can be recognized by a DFA enriched with a single counter. The counter contains an integer, and upon which the finite automata can selectively act with `increment` and `decrement` actions, and the finite

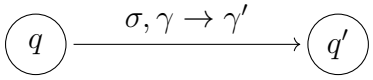

FIGURE 1. A transition of a PDA

automata can select the next state based on a `zero` predicate returning true only if the counter is zero.

Another data structure that, if added to a DFA can recognize L_o, is a stack. Familiar to programmers, we will represent a stack as an element of Γ^*, where Γ is a finite set of symbols. The actions on this stack are `push`, `pop`, and the topmost element of the stack can be used to determine the state transition.

Definition 1.1. A *push down automata* (PDA) is a 6-tuple $\langle\, Q, \Sigma, \Gamma, \delta, q_0, F \,\}$, a finite-automata including a stack with stack symbols from the finite set Γ. The transition function is

$$\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$$

That describes the state transition and stack update based on the input symbol $\sigma \in \Sigma$ and symbol at the top of the stacks $\gamma \in \Gamma$.

A PDA *computation* on word $w \in \Sigma_\varepsilon^*, w = w_1 w_2 \cdots w_n$ is a sequence of states and stacks,

$$\begin{aligned} \langle\, Q \,\rangle &= q_1, q_2, \ldots, q_n \\ \langle\, G \,\rangle &= G_1, G_2, \ldots, G_n \text{ where } G_i \in \Gamma_\varepsilon^* \end{aligned}$$

where,

$$(q_{i+1}, \gamma_{i+1}) \in \delta(q_i, w_i, \gamma_i)$$

where $G_i = \gamma_i\, G'$ and $G_{i+1} = \gamma_{i+1}\, G'$.

1.1. **Some notes on the notation.** The push and pop are modeled using the ε character. Consider the transition,

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

If $\gamma = \varepsilon$, then since $G = \varepsilon\, G$ for any G, the transition can be taken for any $\gamma \in \Gamma$.

If $\gamma \neq \varepsilon$ and $\gamma' = \varepsilon$, the transition requires a stack as $\gamma\, G$ resulting in G, so it is a pop. If $\gamma = \varepsilon$ and $\gamma' \neq \varepsilon$, the transition is from a stack G resulting in a stack $\gamma'\, G$, so it is a push.

The graphical notation for a finite automata can be used for a push down automata with a new convention for the labeling of edges, see Figure 1.

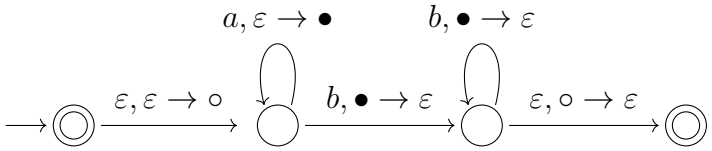

FIGURE 2. PDA accepting $a^i b^i$

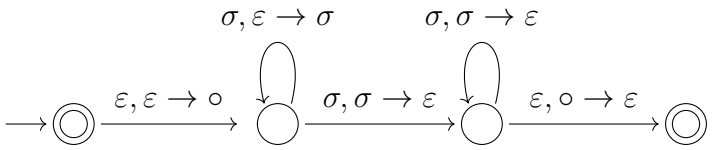

FIGURE 3. PDA accepting $w\, w^R$

1.2. **The language $a^i b^i$.** The diagram for the language $\{\, a^i b^i \mid i \geq 0 \,\}$ is given in Figure 2. The machines first step is to push a bottom–of–stack marker. Then it pushes one stack symbol for every a seen. The machine pops one symbol for every b seen. The states require that the form be $a^* b^*$. The mysterious last transition on empty stack occurs as an ε-move, predicting that there are not more characters in the input string.

One must verify that,

(1) If the string is of the form $a^i b^i$, the PDA ends in a final state.
(2) If the string is not of this form, the PDA does not end in a final state.

1.3. **The language $w\, w^R$.** The language $\{\, w\, w^R \mid w \in \Sigma \,\}$ is a CFL. It can be accepted by the PDA described in Figure 3. The diagram uses so shortcuts to describe the transitions. The the arrow,

$$\sigma, \varepsilon \to \sigma$$

can be explained as "push the input symbol".. The arrow

$$\sigma, \sigma \to \varepsilon$$

can be explained as "match the input symbol against the stack symbol and pop the statck symbol".

Note the essential role of non-determinism in the machine of Figure 3. The machine guesses the mid-point of the string, changing from pushing to match-and-pop. It also guesses the end of the string, transitioning to the final state on view of the bottom-on-stack marker.

What is essential, is that any string accepted by non-deterministic guesses fit the criteria for being in the language. The paradigm is "guess and check". If the guess

was correct about the mid-point of the string is correct, and if the guess about the last symbol in the input is correct, then the word is of the correct form to be in the language. If any guess is incorrect, the computation does not lead to a final state.

2. Context Free Grammars

A *Context Free Grammar* is a mathematical complex $\langle V, T, R, S\rangle$ where,

- V is a finite set of *variables*, also called *non-terminals*,
- T is a finite set of *terminals*,
- R is a set of *rules*, and is a subset of $V \times (V \cup T)^*$,
- and $S \in V$ is the start symbol.

Computation in this system is to repeatedly apply a rule to a string. The computation proceeds with a sequence of intermediate strings

$$\langle\, R\,\rangle = r_1, r_2, \ldots, r_n, \quad r_i \in (\Sigma \cup \Gamma)^*$$

and r_1 is the start symbol and r_n is the only r_i that is composed of terminals, with a transition rule,

$$r_i = wXv,\ r_{i+1} = wuv,\ w, v \in (\Sigma \cup \Gamma)^* \text{ and } (X, u) \text{ is a rule.}$$

All strings of terminals that can be generated in this way, starting from the start symbol, is the language described by the grammar.

The rules are described in the definition as a relation, since for a variable X there can be many substitutions $U_1, U_2, \ldots, U_k$. As a relation, this means there are elements,

$$(X, U_1), (X, U_2), \ldots, (X, U_k).$$

We write this in function notation as,

$$X \to U_1 \,|\, U_2 \,|\, \ldots \,|\, U_k.$$

2.1. The language $a^i b^i$.

We have been interested in the language L_o of strings of the form $a^i b^i$. Here is a context free grammar for this language,

$$S \quad \longrightarrow \quad a\,S\,b \,|\, \varepsilon$$

For instance, the string *aaabbb* is given by this grammar by the derivation,

$$S \to aSb \to aaSbb \to aaaSbbb \to aaa\varepsilon bbb = aaabbb$$

A slight change to the grammar give the language $a^i b^j$ for $i \geq j$,

$$S \quad \longrightarrow \quad a\,S\,b \,|\, a\,S \,|\, \varepsilon$$

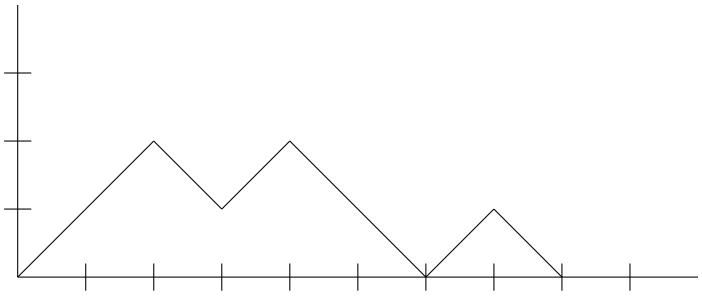

FIGURE 4. Graph of excess opening parenthesis by string position

2.2. **The language $w\,w^R$.** The PDA for $w\,w^R$ would push the $\sigma \in \Sigma$ on the stack and then non-deterministically begin a matching phase, popping the σ and matching them with the input. Here is this language described as a grammar,

$$S \longrightarrow a\,S\,a \,|\, b\,S\,b \,|\, \varepsilon$$

2.3. **The language of balanced parenthesis.** By *balanced parenthesis* I mean that the number of a's equal the number of b's, and in addition, at any point in the string, the number of a's is at least as large as the number of b's. The notion of balanced parenthesis is intuitive from our experience writing formulas with parenthesis, in which every closing parenthesis must exclusively match an opening parenthesis that appears prior in the string.

For instance, *aababbab* is a string in this language. It might be more familiar in the form (()())(). Here is the grammar,

$$S \longrightarrow a\,S\,b \,|\, S\,S \,|\, \varepsilon$$

To prove this grammar is what we want, two directions of suitability must be considered. First, does the grammar generate only balanced strings. Second, does any string of balanced a's and b's get generated by the grammar. The first requirement is easy enough, by inspection of the grammar. The second requirement might need some structure to ascertain.

Lemma 2.1. Any balanced string in $\{ a, b \}$ can be generated by the above grammar.

Proof: Consider a graph of with the horizontal axis the character index in the string, and the vertical axis the excess of a's over b's seen up the the processing of that index, see Figure 4 for an example with the string *aababbab*.

Let s be a string of balanced parenthesis. If $|s| = 0$ the grammar generates the string.

Suppose S is a string of length $2i$, and for all shorter strings the grammar generates such strings. There are two cases,

(1) The graph drawn for this string shows a return to zero in the middle of the graph,
(2) or it does not.

In the first case, the rule $S \to SS$ applies at the point of return to zero, separating s into $s's''$ each shorter than s and balanced.

In the second case, the rule $S \to aSb$ applies, separating s in $a\,s'b$, with s' shorter than s and balanced.

In both cases, the induction hypothesis applies. □

2.4. **Same number a's and b's.** The grammar for equal number of a's and b's is given by this gramma,

$$S \longrightarrow a\,S\,b \,|\, b\,S\,a \,|\, S\,S \,|\, \varepsilon$$

The proof uses the graph above. The graph begins and ends at zero but can take negative values. If the graph has some portion negative and some portion positive, it most be zero somewhere between a negative and positive value, The rule $S \to SS$ applies to drive an induction. Else it remains positive or remains negative, and in the first case the rule $S \to aSb$ applies, in the second the rules $S \to bSa$ applies, and that drives the induction.

2.5. **Unequal a's and b's.** In general, the complement of a context free language is not a context free language. However, in this case it is. We work in steps. First we need a grammar for exactly one more a's than b's,

$$\begin{aligned} S_a &\longrightarrow T\,a\,T \\ T &\longrightarrow a\,T\,b \,|\, b\,T\,a \,|\, T\,T \,|\, \varepsilon \end{aligned}$$

To prove this, the graph begins at zero and ends at one. Draw a line at one, and find the rightmost point the graph goes below this line. Such a point must exist, and is happens when an a is countered, and divides the string s into $s'a\,s''$ where s' and s'' are both balanced.

Likewise a grammar with the roles a and b reversed,

$$\begin{aligned} S_b &\longrightarrow U\,b\,U \\ U &\longrightarrow a\,U\,b \,|\, b\,U\,a \,|\, U\,U \,|\, \varepsilon \end{aligned}$$

The the (almost) union of the two languages (rules for U and T above are included in the grammar), and finally the union of the two languages,

$$\begin{aligned} S &\longrightarrow S_a \,|\, S_b \\ T &\longrightarrow a\,T \\ U &\longrightarrow b\,U \end{aligned}$$

3. Context Free Languages

A subset $S \subseteq \Sigma^*$, some finite alphabet, is *context free* if either,

- There exists a PDA that accepts exactly the set S or,
- There exists a CFG that generates exactly the set S.

Theorem 3.1. The two conditions above are equivalent. If there is a PDA accepting a language, there is a CFG generating that language. If there is a CFG generating a language, there is a PDA accepting that language.

Proof: The proof is in the book and omitted from the content of this course.

3.1. **All regular languages are context free.** An NFA is a PDA where all stack operations are $\varepsilon \to \varepsilon$. Hence that can be accepted by an NFA can be accepted by a PDA. Assuming PDA's are the same as context free languages, the same as languages generated by context free grammars, then we have our proof.

Here is a way to directly create a DFA into a CFG that has the identical language.

(1) For each each state $r \in Q$, in the state set of M, define a variable $R \in V$ in the CFG.
(2) For each transition $\delta(r, \sigma) = r'$, define a grammar rule $R \to \sigma R'$.
(3) For each final state $f \in F$, define a grammar rule $F \to \varepsilon$.
(4) For the start state r_o of M, add the rule $S \to R_o$, with S a fresh variable.

3.2. **The class of CFL's is a algebra with Kleene star.** The class of regular languages formed an algebra with the operations union and concatenation. It also had the Kleene start operation. This is also true for context free languages, although context free languages are not closed by complementation and intersection, where regular languages are.

Lemma 3.1. Context languages are closed by union.

Proof: Given CFG A,

$$\langle V_A, T, R_A, S_A \rangle$$

and B,

$$\langle V_B, T, R_B, S_B \rangle$$

We assume V_A and V_B contain no symbols in common and the symbol S is in neither. The grammar for the union $A \cup B$ is,

$$\langle V, T, R, S \rangle$$

where,

$$\begin{aligned} V &= V_A \cup V_B \cup \{ S \} \\ R &= R_A \cup R_B \cup \{ S \to S_A \,|\, S_B \} \end{aligned}$$

□

Lemma 3.2. Context languages are closed by concatenation.

Proof: Notation as above the languages A and B, the grammar for the concatenation $A \circ B$ is,

$$\begin{aligned} V &= V_A \cup V_B \cup \{ S \} \\ R &= R_A \cup R_B \cup \{ S \to S_A \, S_B \} \end{aligned}$$

□

Lemma 3.3. Context languages are closed by Kleene star.

Proof: Notation as above for language A, the language for the Kleene star A^* is,

$$\begin{aligned} V &= V_A \cup \{ S \} \\ R &= R_A \cup \{ S \to S_A \, S \,|\, \varepsilon \} \end{aligned}$$

□

3.3. **The class of CFL's is not closed by Intersection.** Given CFL's A and B, it is not always true that $A \cap B$ is a CFL.
Consider the languages,

$$\begin{aligned} A &= \{ a^i b^i c^j \,|\, i, j \geq 0 \} \\ B &= \{ a^i b^j c^j \,|\, i, j \geq 0 \} \end{aligned}$$

then

$$A \cap B = \{ a^i b^i c^i \,|\, j \geq 0 \}$$

No CFG or PDA can accept exactly this language (see the appendix of the pumping lemma for CFL's).

However, each term of the intersection is context free. For instance, the language for A is generated by,

$$\begin{aligned} S_A &\to V \, R \\ V &\to \varepsilon \,|\, a \, V \, b \\ R &\to \varepsilon \,|\, c \, R \end{aligned}$$

See the appendix for this grammar in Chomsky Normal Form.

3.4. **The class of CFL's is not closed by Complementation.** Given CFL A, its complement is not always a CFL.
Consider the grammar,

$$\begin{aligned} S &\to U\,T \,|\, T\,U \\ T &\to V\,T\,V \,|\, a \\ U &\to V\,U\,V \,|\, b \\ V &\to a \,|\, b \end{aligned}$$

Lemma 3.4. The above grammar generates the language,

$$\{w\,v \in \{\,a, b\,\}^* \,|\, w \neq v,\,\}$$

Proof: To reduce notational clutter, and string denoted by x, x' or x'' has length i, and any string denoted by y, y' or y'' as length j.

Any string generated is in the language: Any string generated by the grammar can be written as,

$$s = x\,a\,x'\,y\,b\,y' \text{ or } x\,b\,x'\,y\,a\,y'$$

The string $x'y$ can be re-parsed as $y''x''$ because both strings are of length $i + j$, so,

$$s = x\,a\,y''x''\,b\,y' \text{ or } x\,b\,y''x''\,a\,y'.$$

These strings are of the form $w\,v$ with $w \neq v$. For instance, in the first case, $w = x\,a\,y''$ and $v = x''\,b\,y'$.

This is a picture of what is happening. The black dot ($\bullet$) is any a or b. Here is two 4 length strings re-parsed as a 5 length string followed by an 3 length string,

$$(\bullet \quad \bullet \quad a \quad \bullet \,|\, \bullet) \quad (\bullet \quad b \quad \bullet)$$

Here is two 5 length strings re-parsed as a 3 length string followed by a 7 length string,

$$(\bullet \quad a \quad \bullet) \quad (\bullet \quad \bullet \,|\, \bullet \quad b \quad \bullet \quad \bullet \quad \bullet)$$

Any string in the language is generated: Consider $s = v\,w$ where v and w are equal length and unequal. v and w must be unequal in at least one place, so either

$$v = x\,a\,y \text{ and } w = x'\,b\,y'$$

or

$$v = x\,b\,y \text{ and } w = x'\,a\,y'.$$

Write $y\,x' = x''\,y''$ and so,

$$s = x\,a\,x''\,y''\,b\,y'$$

or

$$s = x\,b\,x''\,y''\,a\,y'$$

which are of the form of a string generated by the language. □

Theorem 3.2. The complement of this language,

$$\{\, s \in \{\, a, b \,\}^* \,|\, |s| \text{ is odd or } s = v\,w \text{ and } v \neq w \,\}$$

is not a CFL.

Proof: See appendix.

Hence by example, CFL's are not closed by complement.

4. Chomsky Normal Form and the CYK algorithm

Neither the CFG generation model nor the PDA machine model are simple to compute with. An algorithm to determine membership in a language, given the grammar for the language, exists if the grammar is properly prepared. if rules never, with a single exception, decrease the length of an intermediary product during generation, and those rules keeping the length constant introduce new terminals, the number of rules applied can never exceed by much the length of the final string that is generated. The Chomsky Normal Form (CNF) of a grammar will act as desired.

There are algorithms to place a general grammar in CNF. Then, given a string of terminals, the derivations can be searched fro a one that gives that string.

Naively this search would be exponential. However, the search can be described recursively in a way that allows a solution by dynamic programming. The Cocke-Younger-Kasami algorithm (CYK) does this, and provided a solution to whether a string of length n can be generated by the grammar in CNF in time $O(n^3)$.

Appendix A. The Pumping Lemma for CFL's

Theorem A.1. Given a CFL $\mathcal{L}$, there exists a p such that for all $s \in \mathcal{L}$, $|s| \geq p$, then $s = uvxyz$ such that,

(1) $|vxy| < p$,
(2) $vy \neq \varepsilon$
(3) $\forall i, uv^i xy^i z \in \mathcal{L}$.

The proof is by considering the parse tree. Any path of depth exceeding the number of variable in a grammar for the language will have a repeated variable on the path. The section between occurrences of this variable can be removed or repeated, giving the pumped versions of the string.

A.1. **The language $a^i b^i c^i$ is not context free.** Select $s = a^p b^p c^p$, If the language were a CFL the pumping lemma would find a vxy that can be pumped. However vxy cannot contain all three of the characters a, b and c. Therefore pumping will make the frequency of at least one of these letters different than one of the others. Hence the language is not context free.

A.2. **The language $w\,w$ is not context free.** Select $s = a^p b^p a^p b^p$. If the language were a CFL the pumping lemma would find a vxy that can be pumped. Investigating cases, every such vxy cannot be pumped, so the language is not a CFL.

Appendix B. CNF grammar for $a^i b^i c^j$

A grammar for,

$$A \;=\; \{\, a^i b^i c^j \mid i, j \geq 0 \,\}$$

was, given,

$$\begin{aligned} S_A &\to V\,R \\ V &\to \varepsilon \mid a\,V\,b \\ R &\to \varepsilon \mid c\,R \end{aligned}$$

To change this to CNF, step one is to introduce variables for each terminal and reduce rules with more than three on the right hand side,

$$\begin{aligned} S_A &\to V\,R \\ V &\to \varepsilon \mid A\,V_1 \\ V_1 &\to V\,B \\ R &\to \varepsilon \mid C\,R \\ A, B, C &\to a, b, c \end{aligned}$$

Then to absorb ε productions into the rule calling the variable that can produce a ε,

$$\begin{aligned} S_A &\to \varepsilon \mid V \mid R \mid V\,R \\ V &\to A\,V_1 \\ V_1 &\to V\,B \mid B \\ R &\to C\,R \mid C \\ A, B, C &\to a, b, c \end{aligned}$$

Then remove unit productions,

$$\begin{aligned} S_A &\rightarrow \varepsilon \mid A\,V_1 \mid c \mid C\,R \mid V\,R \\ V &\rightarrow A\,V_1 \\ V_1 &\rightarrow V\,B \mid b \\ R &\rightarrow C\,R \mid c \\ A, B, C &\rightarrow a, b, c \end{aligned}$$