Lecture 6

Ben Rosenberg

June 15, 2021

Context-Free Grammars

Regular expressions contain strings; context-free grammars *generate* strings.

Definition: A **context-free grammar** is a structure:

$$G = (\Sigma, V, P, S)$$

We already know Σ, the alphabet. But similarly to regular expressions, we really have two alphabets – one for the letters used in the grammar, and one for the programming keyboard of symbols used to represent a grammar.

We define each of the components as follows:

- Σ: input/output alphabet; *alphabet of terminals*
- V: alphabet of variables (*non*-terminals)
  - Note: It is required that V and Σ be disjoint; that is, $V \cap \Sigma = \emptyset$
  - Note also that V can never be empty, as it always must contain S
- S: designated start symbol. $S \in V$
- P: *finite* set of productions (*rules*). $P \subseteq V \times (\Sigma \cup V)^*$
  - Note that despite the fact that the right-hand side above is infinite on the level of $\aleph_0$, P itself is finite.

Notation: When we have $(A, w) \in P$, this means that $A \in V$ and $w \in (V \cup \Sigma)^*$. Then, we write $A \to w$, pronounced "A to w" or "A derives w" (*not* A implies w).

Notation: If $A \to w_1, A \to w_2, \ldots, A \to w_k$ then we may write $A \to w_1|w_2|\ldots|w_k$.

Ex.

- $G = (V, \Sigma, P, S)$
- $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$
- $V = \{S, A, B, D\}$ (capital C is omitted because it looks too similar to lowercase $\mathtt{c}$)
- $P : S \to AB|BD$, $A \to \mathtt{a}A|\mathtt{c}$, $B \to B\mathtt{b}|\mathtt{ca}$, $D \to \mathtt{ab}D|\lambda$

Def (informal): Start with a start symbol. Then, replace the start symbol with the RHS of some of its rules (those that have the start symbol on the left side) and replace S with one of the possibilities separated by $|$s on the right. In the above example, either AB or BD can be substituted for S. We then continue doing this.

Definition (continued): We say that a variable $E \in V$ **derives** a **sentence** $w \in (\Sigma \cup V)^*$ *in one step* if $[E \to w] \in P$ ($E \to w$ is a rule).

In our G, for instance, B derives $\mathtt{ca}$ in one step.

Variable $E \in V$ *derives a sentence* $x \in (\Sigma \cup V)^*$ in $n + 1$ steps if all of the following hold:

1. w derives $y \in (\Sigma \cup V)^*$ in n steps

2. $y = y_1 F y_2$ where y_1 and y_2 are strings $\in (\Sigma \cup V)^*$ and $F \subset V$ is a variable
3. $[F \to y_0] \in P$ is a rule, and $y_0 \in (\Sigma \cup V)^*$ is a string
4. $y_1 y_0 y_2 = x$

In other words:

$$E \xrightarrow{n+1 \text{ steps}} y \iff E \xrightarrow{n \text{ steps}} \boxed{y_1}\boxed{F}\boxed{y_2} = \boxed{y_1}\boxed{y_0}\boxed{y_2}$$

The rule here is $F \to y_0$.

We are done substituting when we are out of variables in the sentence.

Definition: We say that **language** $L(G)$ is *generated* by grammar G if it is the set of exactly those terminal ($\in \Sigma$) strings that are derivable from the start symbol in any number of steps.

Example, using our previous G:

$S \to AB \to \texttt{a}AB$

$A \to \texttt{c}$

$A \to \texttt{a}A \to \texttt{ac}$

From the above two, we can see that $A \to \texttt{a}^*\texttt{c}$ by using the rule $A \to \texttt{a}A$ zero or more times, and then applying $A \to \texttt{c}$.

Similarly, $B \to \texttt{cab}^*$, and $D \to \texttt{ab}^*$.

So, we have, for our starting string:

$S \to AB|BD \to \texttt{a}^*\texttt{ccab}^* \cup \texttt{cab}^*(\texttt{ab})^*$

Note that grammars can make languages that *do not* have regular expressions, though.

Consider a language $L_2 = \{\texttt{a}^n\texttt{b}^n | n \geq 0\}$. There is no regular expression for this. If we tried to do this with $\texttt{a}^*\texttt{b}^*$, we would have aab in the regular expression but not in L_2. So, $L_2 \subset \texttt{a}^*\texttt{b}^*$. We will prove this later in the course.

Let's write a grammar for L_2:

- $G = (V, \Sigma, P, S)$
- $V = \{S\}$
- $\Sigma = \{\texttt{a}, \texttt{b}\}$
- $P : S \to \lambda | \texttt{a}S\texttt{b}$

So, we have (from above) $\boxed{\quad \texttt{a}^n \quad}\ \lambda\ \boxed{\quad \texttt{b}^n \quad}$.

Example: Consider $L = \{\texttt{a}^n\texttt{ccb}^{2n} | n \geq 0\}$. Then we want the left side to have a telescope of a, and the right side to have bb.

Let's write a grammar for L:

- $G = (V, \Sigma, P, S)$
- $V = \{S\}$
- $\Sigma = \{\texttt{a}, \texttt{b}\}$
- $P : S \to \texttt{a}S\texttt{bb}|\texttt{cc}$

Theorem: Algorithm ①:

Input: context-free grammars G_1 and G_2

Output: context-free grammar G that generates:

1. $L(G_1) \cup L(G_2)$
2. $L(G_1) \circ L(G_2)$
3. $L(G_1)^*$

Construction: Let

- $G_1 = (V_1, \Sigma_1, P_1, S_1)$
- $G_2 = (V_2, \Sigma_2, P_2, S_2)$
- $V_1 \cap V_2 = \emptyset$

We need to construct $G = (V, \Sigma, P, S)$.

1. $\cup$ operation
   1. $V = V_1 \cup V_2 \cup \{S\}, S \notin V_1 \cup V_2$
   2. S is a new variable
   3. $P = P_1 \cup P_2 \cup \underbrace{\{S \to S_1 | S_2\}}_{\text{new rules}}$
2. $\circ$ operation
   1. $V = V_1 \cup V_2 \cup \{S\}, S \notin V_1 \cup V_2$
   2. $P = P_1 \cup P_2 \cup \underbrace{\{S \to S_1 \circ S_2\}}_{\text{new rules}}$
3. $*$ operation
   1. $V = V_1 \cup V_2 \cup \{S\}, S \notin V_1 \cup V_2$
   2. $P = P_1 \cup P_2 \cup \underbrace{\{S \to \lambda | SS | S_1\}}_{\text{new rules}}$
      1. S can make zero or more copies of itself

So, the class of context-free languages is **closed** under regular operations. That is, applying regular operations to a context-free language yields itself a context-free language.

Algorithm (2):

Input: regular expression e

Output: equivalent context-free grammar; G such that $L(G) = L(e)$

Construction:

Recursion on the number of operators in e

Base case: e has zero operators, and $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$

- any alphabet letter $\in \Sigma$ (ex: a):
  - $G = (V, \Sigma, P, S); V = \{S\}, P : S \to \texttt{a}$
- λ
  - $G = (V, \Sigma, P, S); V = \{S\}, P : S \to \lambda$
- $\emptyset$
  - $G = (V, \Sigma, P, S); V = \{S\}, P = \emptyset$ (no rules)

Recursively: 3 cases for outermost operator in e (applied *last*)

1. $e = e_1 \cup e_2$
2. $e = e_1 \circ e_2$
3. $e = e_1^*$

Then, we apply Algorithm (1).

Example: $e = \texttt{ab}^*(\texttt{a} \cup \texttt{b}) \cup (\texttt{bc} \cup a)^*$.

Let's write a grammar for e:

- $G = (V, \Sigma, P, S)$

- $V = \{S\}$
- $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$
- $P : S \to A|B$; $A \to \texttt{a}DE$; $D \to \texttt{b}D|\lambda$; $E \to \texttt{a}|\texttt{b}$; $B \to \lambda|BB|F$; $F \to \texttt{bc}|\texttt{a}$

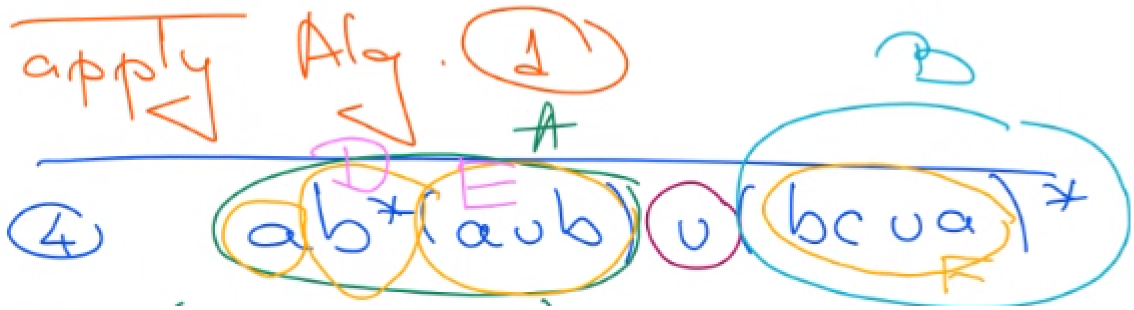

Figure 1: Diagram of regular expression

Example: Set of strings that are matched parentheses

In other words, $\Sigma = \{(,)\}$, and () is good, but)()() or)(or (() are not good.

Let's write a grammar for this set:

- $G = (V, \Sigma, P, S)$
- $V = \{S\}$
- $\Sigma = \{(,)\}$
- $P : S \to \lambda|(S)|SS$

Example: Set of palindromes over $\Sigma = \{\texttt{a,b,c}\}$

In other words, strings that are equal to their reversal.

Let's write a grammar for this set:

- $G = (V, \Sigma, P, S)$
- $V = \{S\}$
- $\Sigma = \{\texttt{a,b,c}\}$
- $P : S \to \lambda|\texttt{a}S\texttt{a}|\texttt{b}S\texttt{b}|\texttt{c}S\texttt{c}|\texttt{a}|\texttt{b}|\texttt{c}$

This is similar to the previous parenthesis matching, but with a's, b's, and c's as their own sets of parentheses (minus nesting).

Example: Set of strings that are *not* palindromes over $\Sigma = \{\texttt{a,b,c}\}$

In other words, strings that are *not* equal to their reversal. Using our previous analogy, we are looking for a *lack* of parentheses.

Let's write a grammar for this set:

- $G = (V, \Sigma, P, S)$
- $V = \{S\}$
- $\Sigma = \{\texttt{a,b,c}\}$
- $P : S \to \texttt{a}S\texttt{a}|\texttt{b}S\texttt{b}|\texttt{c}S\texttt{c}|\texttt{a}A\texttt{b}|\texttt{a}A\texttt{c}|\texttt{b}A\texttt{a}|\texttt{b}A\texttt{c}|\texttt{c}A\texttt{a}|\texttt{c}A\texttt{b}$; $A \to \lambda|AA|\texttt{a}|\texttt{b}|\texttt{c}$

We want to start from the outside of a string, and work inwards until we find a pair of letters equidistant from the ends that are not equal (good string) or reach the middle (bad string).

Example: $L : \texttt{a}^{3n+1}\texttt{b}^{k+2}\texttt{g}^{j}\texttt{c}^{2k+1}\texttt{a}^{p+2}\texttt{b}^{2p}\texttt{g}^{n+3}$, with $n, k, j, p \geq 0$.

This telescopes as follows (matching exponents):

Let's write a grammar for this set:

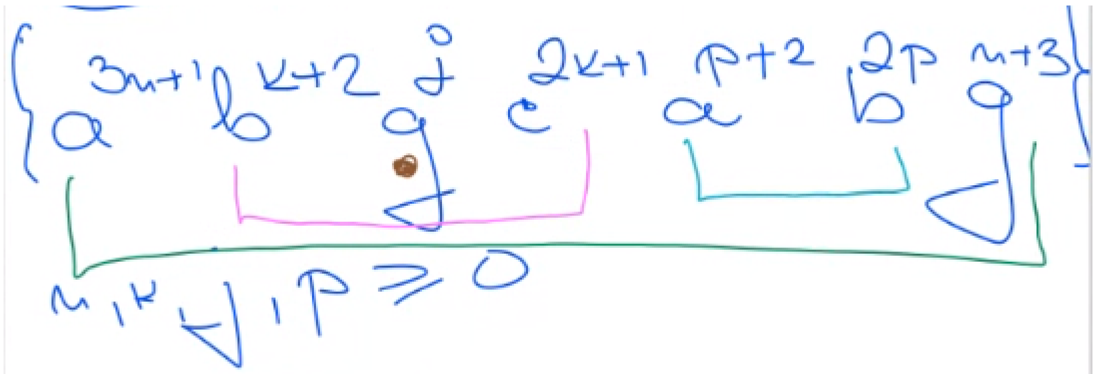

Figure 2: Grouping the telescoping

- $G = (V, \Sigma, P, S)$
- $V = \{S, A, B, D\}$
- $\Sigma = \{\texttt{a,b,c,g}\}$
- $P : S \to \texttt{aaa}S\texttt{g}|\texttt{a}AB\texttt{ggg};\ A \to \texttt{b}A\texttt{cc}|\texttt{bb}D\texttt{c};\ D \to D\texttt{g}|\lambda;\ B \to \texttt{a}B\texttt{bb}|\texttt{aa}$