Context-Free Grammars and Pushdown Automata

Bill D'Alessandro—Math 503

These notes draw (very!) liberally from Lewis and Papadimitriou (1981), *Elements of the Theory of Computation*, chapter 3.

Catchup on Context-Free Grammars

A **context-free grammar** G is a quadruple (V, Σ, R, S), where

- V is an alphabet
- Σ (the set of **terminals**) is a subset of V, its elements represented by lowercase letters
- R (the set of **production rules**, or just **rules**) is a finite subset of $(V - \Sigma) \times V^*$
- S (the **start symbol**) is an element of $V - \Sigma$.

The members of $V - \Sigma$ are **nonterminals**, and are represented by uppercase letters.
Production rules look like this: $A \to AA;\ A \to aA;\ A \to \varepsilon$.

Incidentally, this is why these grammars are called "context-free". Consider the string $aAAba$. We are allowed by a CFG to operate on *any part of the string* to which the production rules apply, without caring about the *context* of that bit of the string—that is, about the other symbols in its neighborhood. So, using the above rules, $aAAba \to aaAAba \longrightarrow aaAba \longrightarrow aaba$ (e.g.) is a legal derivation. Some grammars, the *context-sensitive* ones, lack this feature. In such grammars, "replacements may be conditioned on the existence of an appropriate context" (Lewis, 97).

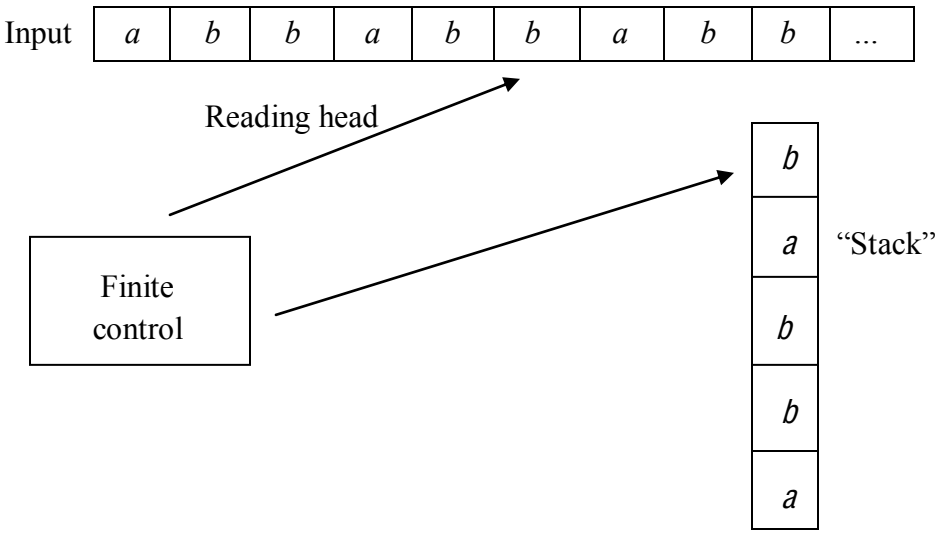
Context-free grammars do an excellent job capturing most programming languages, and a pretty nice job with most natural languages. (But see Shieber (1985) ["Evidence against the context-freeness of natural language", *Linguistics and Philosophy* 8: 333–343] for proof that the fit in the latter case is imperfect.)

Proposition
All regular languages are context-free, but not all context-free languages are regular.

Proof
Phil showed last time that all regular languages are context-free.
To see that not all CFLs are regular, consider the grammar $G = (V, \Sigma, R, S)$, where $V = \{S, a, b\}$ and $\Sigma = \{a, b\}$, and R consists of the rules $S \to aSb$ and $S \to \varepsilon$. The language generated by G looks like $\{\varepsilon, ab, aabb, aaabbb, \ldots\}$, or more succinctly $L(G) = \{a^n b^n : n \geq 0\}$. This language can't be obtained from any regular expression—one would need an expression that "counts" and "remembers" the number of *a*s one has written—and thus it is non-regular. ■

This result implies that some context-free languages (CFLs) are unrecognizable by finite automata. There is a more powerful sort of automaton, though, that can recognize arbitrary CFLs. Such machines are called pushdown automata.

Schematic of a Pushdown Automaton



Description of pushdown automata

A pushdown automaton (PDA) has, in addition to its input string, an auxiliary storage device (a stack). The stack can record the input string as it is read, and this record can be used for various purposes later. There are two ways a machine M and its stack can interact: (1) the symbol on top of the stack can affect M's transitions; and (2) M can manipulate this top symbol. The latter can, itself, happen in two ways. M can push a symbol by *adding* it to the top of the stack, or it can pop a symbol by *removing* it from the top of the stack.

Formally, a PDA is a sextuple $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where

- K is a finite set of states
- Σ is an alphabet (the input symbols)
- Γ is an alphabet (the stack symbols)
- $s \in K$ is the initial state
- $F \subseteq K$ is the set of final states
- Δ is the transition relation, a finite subset of $(K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$

Δ looks daunting, but on inspection it is seen to be rather tame. A member of Δ has the form $((p, u, \beta), (q, \gamma))$ and is called a transition of the machine M. Intuitively, the transition says that M, whenever it is in state p with β at the top of the stack, may read u from the input tape, replace

β by γ on the top of the stack, and enter state q. (Since several transitions may be legal for M in a given state, the PDA will be a nondeterministic machine.)

Example

This setup allows languages like $\{a^n b^n : n \geq 0\}$ to be recognized. To achieve this, our PDA must keep track of how many as it has read in a given string, and must compare this number to that of the subsequent bs.

Formally, the PDA recognizing $\{a^n b^n : n \geq 0\}$ may be described as follows:

Let $M = (K, \Sigma, \Gamma, \Delta, s, F)$, where

- $K = \{s, t, r\}$
- $\Sigma = \{a, b\}$
- $\Gamma = \{A, C\}$, where C is a special 'marker' stack symbol
- $s \in K$ is the initial state
- $F = \{r\} \subset K$ is the singleton final state
- Δ consists of the following seven transitions: $((s, \varepsilon, \varepsilon), (s, C))$, $((s, a, C), (s, AC))$, $((s, a, A), (s, AA))$, $((s, \varepsilon, C), (t, C))$, $((s, \varepsilon, A), (t, A))$, $((t, b, A), (t, \varepsilon))$, $((t, \varepsilon, C), (r, \varepsilon))$.

- These seven transitions, in English, are as follows:
  (1) If in state s with an empty stack, read nothing, push C, and remain in s.
  (2) If in state s with only C in its stack, [possibly] read a, push an A, and remain in s.
  (3) If in state s with an A on top of the stack, read a, push another A, and remain in s.
  (4) If in state s with only C in its stack, [possibly] read nothing, leave the stack unaltered, and move to state t.
  (5) If in state s with an A on top of the stack, [possibly] read nothing, leave the stack unaltered, and move to state t.
  (6) If in state t with an A on top of the stack, [possibly] read b, pop A, and remain in t.
  (7) If in state t with only C in its stack, read nothing, pop C, and move to state r.

Note

A machine M accepts a string w iff, by reading w, M can reach a state $(p, \varepsilon, \varepsilon)$ (where $p \in F$) by some set of transitions. We will make this notion more precise later.

Because of the properties of this nondeterministic machine, it will sometimes fail to accept a word from $\{a^n b^n : n \geq 0\}$. But that is all right: we say that a nondeterministic automaton M accepts a language L just in case it is *possible* for M to accept the words in L, which is clearly the case here. ■

Eventually, we will prove (half of) the following important Theorem: *The class of languages accepted by pushdown automata is exactly the class of context-free languages*. But first, let's get some things straight.

Definitions

- A configuration of a PDA, $M = (K, \Sigma, \Gamma, \Delta, s, F)$, is a member of the set $K \times \Sigma \times \Gamma^*$. The first component is the present state of M, the second is the portion of the input yet to be read, and the third is the contents of the stack, read top-down.
- For every transition $((p, u, \beta), (q, \gamma))$ of a PDA M, and for every $x \in \Sigma^*$ and $\alpha \in \Gamma^*$, we define $(p, ux, \beta\alpha) \vdash_M (q, x, \gamma\alpha)$. $\vdash_M$ is the relation 'yields in one step'. It holds between two configurations iff they can be represented in this form, for some transition in M.
- We denote the transitive, reflexive closure of the above relation (i.e., 'yields') by $\vdash_M^*$.
- Now we can precisify an earlier remark. We say M accepts a string $w \in \Sigma^*$ iff $(s, w, \varepsilon) \vdash_M^* (p, \varepsilon, \varepsilon)$ for some final state $p \in F$.
- Any sequence of configurations $C_0, C_1, \ldots, C_n$ such that $C_i \vdash_M C_{i+1}$ for $i = 0, \ldots, n-1$ is called a computation by M of length n, or with n steps.
- The language accepted by M, $L(M)$, is the set of all strings accepted by M.

En Route to the Theorem

Remember from before that if $G = (V, \Sigma, R, S)$ is a context-free grammar, then $w \in L(G)$ iff $w \in \Sigma^*$ and there is a derivation

$$S \Longrightarrow w_1 \Longrightarrow w_2 \Longrightarrow \cdots \Longrightarrow w_{n-1} \Longrightarrow w$$

for some strings $w_1, \ldots, w_{n-1} \in V^*$ $(n > 0)$.

Here we are dealing with derivations in a CFG again. Remember what they look like? Recall our earlier grammar G (p. 1), whose production rules were (1) $A \to AA$, (2) $A \to aA$, and (3) $A \to \varepsilon$. Now consider the string aAA, and suppose we want to apply rule (2) to this string. This can be done in two ways (since the grammar is, after all, context-free!): either $aAA \to aaAA$ or $aAA \to aAaA$. In the first derivation, we've replaced the *leftmost* occurrence of A, while in the second derivation we've replaced the *rightmost* occurrence of A.

We can classify derivations by their resemblance to one of the two derivations above. In particular, if the nonterminal symbol replaced at every step of a given derivation is the *leftmost* nonterminal symbol in the string, we call it a leftmost derivation. 'α is leftmost derivable from S' is written $S \overset{*L}{\Rightarrow} \alpha$.

Why is *leftmostness* an interesting property for a derivation to have? Well, it turns out that…

Lemma 1 (Equivalence of Derivations and Leftmost Derivations)

For any context-free grammar $G = (V, \Sigma, R, S)$ and any string $w \in \Sigma^*$, w is derivable in G via a leftmost derivation iff w is derivable in G at all.

(That is, every derivation is equivalent to some leftmost derivation, and vice versa.) I omit the proof, one direction of which is rather complex. The other direction, of course, is trivial—obviously, every leftmost derivation is a derivation *simpliciter*.

Why is this useful? Well, it allows us to restrict our attention to a subset of the derivations of a given G (those that are leftmost) without loss of generality. That will be useful in tackling the following theorem, which we will now state and partially prove:

Theorem (CFL = PDA)

The class of languages accepted by pushdown automata is exactly the class of context-free languages.

Proof

Part 1 of the proof involves demonstrating the following lemma.

Lemma 2 (CFL⊆ PDA)

Each context-free language is accepted by some pushdown automaton.

Proof of Lemma 2

Let $G = (V, \Sigma, R, S)$ be a CFG; we need to construct a PDA M such that $L(M)=L(G)$. Let this $M = (\{p, q\}, \Sigma, \Gamma = V, \Delta, p, \{q\})$. Notice:

- The machine has just two states, p and q, of which p is the start state and q is the (lone) final state.
- $\Gamma = V$. That is, M's set of stack symbols is identical to G's set of terminals and nonterminals.
- We define the transitions in Δ as follows:
  1. $\big((p, \varepsilon, \varepsilon), (q, S)\big)$
  2. $((q, \varepsilon, A), (q, x))$ for each rule $A \to x$ in R.
  3. $\big((q, a, a), (q, \varepsilon)\big)$ for each $a \in \Sigma$.

  These transitions, in English, say:
  1. If in state p with an empty stack, read nothing, push S, and move to state q.
  2. If in state q with some nonterminal symbol A on top of the stack, push any symbol x for which there is a production rule $A \to x$, and remain in state q.
  3. If in state q with some terminal symbol a on top of the stack, pop a, read a, and remain in state q.

Earlier we defined *accepting a string w* for an arbitrary PDA. Applying that definition to the present machine, we see that *M* accepts some string *w* iff $(p, w, \varepsilon) \vdash_M^* (q, \varepsilon, \varepsilon)$; that is, if the initial state of *M* with input *w* yields the configuration $(q, \varepsilon, \varepsilon)$.
To prove the lemma (i.e., that *L(M))=L(G)* for the *M* we have defined), we prove the following two claims.

- Claim 1. If $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$, where $\alpha_1 \in \Sigma^*$ and $\alpha_2 \in (V - \Sigma)V^* \cup \{\varepsilon\}$, then $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$. (That is, if the string $\alpha_1\alpha_2$ is derivable from the start symbol in *G*, then the state $(q, \varepsilon, \alpha_2)$ is computable from the start state in *M*.)
- Claim 2. If $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$, where $\alpha_1 \in \Sigma^*$ and $\alpha_2 \in V^*$, then $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$. (That is, the converse of Claim 1.)

These claims jointly imply the lemma, since Claim 1 establishes that $\alpha_1\alpha_2 \in L(M)$ if it is in *L(G)*, while Claim 2 establishes that $\alpha_1\alpha_2 \in L(G)$ if it is in *L(M)*. Hence, by taking $\alpha_2 = \varepsilon$, it follows that a given string α is in *L(G)* iff it is in *L(M)*. We prove both claims by induction.

Proof of Claim 1

Suppose that $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$, where $\alpha_1 \in \Sigma^*$ and $\alpha_2 \in (V - \Sigma)V^* \cup \{\varepsilon\}$. The proof proceeds by induction on the length of a derivation of α from *S*.

Basis Case. If the derivation is of length 0, then $S = \alpha$, and hence $\alpha_1 = \varepsilon$ and $\alpha_2 = S$. Trivially, then, $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$, since the LHS and RHS configurations are the same.

Induction Hypothesis. Assume that if $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$ by a derivation of length *n* or less, with $n \geq 0$, then $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$.

Inductive Step. Let

$$S = u_0 \overset{L}{\Rightarrow} u_1 \overset{L}{\Rightarrow} \ldots \overset{L}{\Rightarrow} u_{n+1} = \alpha$$

be a leftmost derivation of α from *S*, and let $\alpha = \alpha_1\alpha_2$ as specified. Clearly, u_n has at least one nonterminal (since u_{n+1} is derived from u_n). Hence $u_n = \beta_1 A \beta_2$ and $u_{n+1} = \beta_1\gamma\beta_2$, where $\beta_1 \in \Sigma^*$, $A \in V - \Sigma$, and $A \to \gamma$. (Although Lewis doesn't say so, it is evident from context that $\beta_2 \in (V - \Sigma)V^* \cup \{\varepsilon\}$.) By the induction hypothesis,

$$(q, \beta_1, S) \vdash_M^* (q, \varepsilon, A\beta_2), \qquad (1)$$

but since $A \to \gamma$, $((q, \varepsilon, A), (q, \gamma))$ is a type-2 transition of *M*; so

$$(q, \varepsilon, A\beta_2) \vdash_M (q, \varepsilon, \gamma\beta_2). \quad (2)$$

Now α is $\beta_1\gamma\beta_2$, and is also $\alpha_1\alpha_2$. So we can write α_1 as $\beta_1\delta$ for some $\delta \in \Sigma^*$, such that $\delta\alpha_2 = \gamma\beta_2$. Therefore

$$(q, \delta, \gamma\beta_2) \vdash_M^* (q, \varepsilon, \alpha_2) \quad (3)$$

by a series of type-3 transitions, each of which pops a symbol of δ until only the string α_2 remains. Hence, combining (1), (2), and (3), we have the sequence of computations

$$\begin{aligned}(q, \alpha_1, S) &= (q, \beta_1\delta, S) \\ &\vdash_M^* (q, \delta, A\beta_2) \\ &\vdash_M^* (q, \delta, \gamma\beta_2) \\ &\vdash_M^* (q, \varepsilon, \alpha_2),\end{aligned}$$

which completes the inductive step and the proof of Claim 1.

Proof of Claim 2

Now suppose, conversely, that $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$, where $\alpha_1 \in \Sigma^*$ and $\alpha_2 \in V^*$; we need to show that $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$. This time we proceed by induction on the length of a computation by *M*.

Basis Case. If $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$ in zero steps—that is, if $(q, \alpha_1, S) = (q, \varepsilon, \alpha_2)$— then $\alpha_1 = \varepsilon, \alpha_2 = S$, and obviously $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$.

Induction Hypothesis. If $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$ by a computation of n steps or fewer, $n \geq 0$, then $S \overset{*L}{\Rightarrow} \alpha_1\alpha_2$.

Inductive Step. Suppose that $(q, \alpha_1, S) \vdash_M^* (q, \varepsilon, \alpha_2)$ in $n+1$ steps. Then, for some $\beta \in \Sigma^*, \gamma \in \Gamma^*$, $(q, \alpha_1, S) \vdash_M^* (q, \beta, \gamma)$ in n steps, and $(q, \beta, \gamma) \vdash_M^* (q, \varepsilon, \alpha_2)$. This last move is the result of a type-2 or type-3 transition.

Suppose the last transition was type-2. Then $\beta = \varepsilon, \gamma = A\gamma_1$, and $\alpha_2 = \delta\gamma_1$ for some $A \in V - \Sigma, \gamma_1 \in V^*$, and some rule $A \to \delta$. Since $S \overset{*L}{\Rightarrow} \alpha_1 A\gamma_1$ by the induction hypothesis, $S \overset{*L}{\Rightarrow} \alpha_1\delta\gamma_1 = \alpha_1\alpha_2$.

Suppose the last transition was type-3. Then $\beta = a$, a terminal symbol, and $\gamma = a\alpha_2$. Then $\alpha_1 = \delta a$ for some $\delta \in \Sigma^*$, and $(q, \delta, S) \vdash_M^* (q, \varepsilon, a\alpha_2)$ in n steps, so by the induction hypothesis $S \overset{*L}{\Rightarrow} \delta a\alpha_2 = \alpha_1\alpha_2$.■

That concludes the proof of Lemma 2 (CFL ⊆ PDA), and the first half of the Theorem (CFL = PDA). The second half of the theorem is longer and more complicated, but as one might expect it too involves a pair of proofs by induction. Unfortunately, it is not even possible to give a general idea of the proof without going into some detail, as it requires the definition and construction of a novel type of machine (a simple PDA), and by now the hour is late.