Lecture III
CONTEXT FREE LANGUAGES

Lecture 3: Honors Theory, Spring'02.

We introduce context free languages (CFL), context free grammar (CFG) and pushdown automata (PDA). The pumping lemma for context free languages is used to show various languages to be non-CFL.

§1. Context Free Languages and Grammar

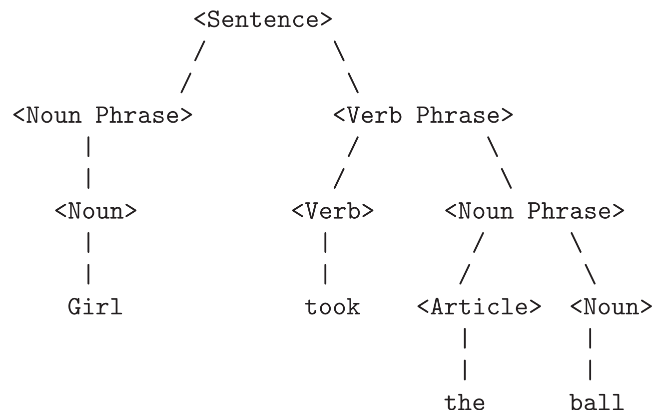
We have seen automatas (dfa, nfa) and expressions (regular) that can describe languages. We now introduce "formal grammars" as another device for describing languages. The notion of grammars is motivated by natural languages. Consider the structure of very simple English sentences. The following are "grammatical rules":

```
<Sentence>  --> <Noun Phrase> <Verb Phrase>
<Noun Phrase> --> <Article> <Noun> | <Noun>
<Verb Phrase> --> <Verb> | <Verb> <Noun Phrase>
<Article> --> a | the
<Noun> --> boy | girl | ball | dog | ...
<Verb> --> caught | saw | took | ...
```

Note the use of "|" as alternatives. Thus we can "generate" sentences such as

```
Girl took the ball
The boy saw a girl
Dog caught ball
A ball saw the boy
...
```

The proof that the first sentence can be generated by the simple grammar amounts to giving a parse tree:



The grammatical rules or **productions** has a left-hand side and a right-hand side. Each side is a string of variables (indicated by `<...>`) and terminals (such as `boy`, `girl`, etc). Using the rules, we generate sentence which has only terminals.

We now formalize this. Let V, Σ be disjoint alphabets. A **context free grammar** (CFG) is a 4-tuple

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

where V is the set of **variables**, Σ the set of **terminals**, $S \in V$ is the **start symbol**, and R is a finite set of rules. A **rule** has the form form $A \to w$ where $w \in (V \cup \Sigma)^*$. If $u, v, w \in \cup\Sigma$, define the **produce relation** as the binary relation on words

$$u \Rightarrow_G v$$

(or simply, $u \Rightarrow v$) if $u = u'Au''$, $v = v'wv''$ and $A \to w$ is a rule in G. Thus A is **replaced** by w in this produce relation. The **derivation** relation $\Rightarrow_G^*$ (or simply $\Rightarrow^*$) is the reflexive, transitive closure of $\Rightarrow$. Thus, $u \Rightarrow^* v$ iff there is a sequence

$$u_0 \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_m \quad (m \geq 0) \tag{1}$$

such that $u = u_0$ and $u_m = v$. If $m = 0$ then $u = v$. We call (1) a **derivation** of u_m from u_0. The **language generated by** G is

$$L(G) = \{w \in \Sigma^* : S \Rightarrow_G^* w\}$$

A language is **context free** if it is generated by some CFG.

The derivation (1) is called a **leftmost derivation** if the variable being replaced at each step is the leftmost variable. The order of replacing these variables is somewhat artificial in a derivation. One way to remove this artificial ordering is to consider the **parse tree** of a derivation. This is defined in the natural was to be an oriented rooted tree in which the internal nodes are variables and the leaves are terminals. The root has the start symbol S and the children of an internal node A are labeled by the symbols on the righthand side of some production, $A \to w$. An example is the parse tree of the sentence "`Girl took the ball` above.

E.g., Let the rules of G be be given by

$$S \to 0S0|1S1|\epsilon.$$

This generates the set of binary palindromes of even length. To generate all binary palindromes, add two more rules:

$$S \to 0|1.$$

Here is another example for simple arithmetic expressions:

```
<E>  --> <id> | <num> | (<E>) | - <E> | <E> <op> <E>
<id> --> a | b | c | ... x | y | z
<num> --> <dig> | <dig><num>
<dig> --> 0 | 1 | ... 8 | 9
<op> --> + | - | * | /
```

§2. Normal Form

There are two special forms for CFG's: (i) A grammar G is in **Chomsky Normal Form** if every rule has the form

$$A \to BC, \qquad A \to a$$

where A, B, C are variables, B, C is not the start symbol, and $a \in \Sigma$. In addition, we allow the special rule of the form $S \to \epsilon$. (ii) We say G is in **Greibach Normal Form** if every rule has the form

$$A \to aB_1 B_2 \cdots B_m, \qquad (m \geq 0)$$

where $a \in \Sigma$ and B_is are variables. Every CFL can be generated by grammars in both normal forms. We give the prove for Chomsky normal form:

Theorem 1 *Every context-free language is generated by a CFG in Chomsky Normal Form.*

Proof. Starting from a CFG G, we transform G into Chomsky Normal Form in five steps:

First ensure that S does not appear on the right hand side of any rule. To do this, replace all occurences of S on the RHS of any rule by a new variable S'. Also, add the rule $S \to S'$.

Second, remove each rule of the form $A \to \epsilon$ and introduce, for any rule $B \to w$ where A occurs in w, a new rule $B \to w[A/\epsilon]$ ($w[A/\epsilon]$ means we replace each occurence of A in w by ϵ). If $\epsilon \in L(G)$, we also add the special rule $S \to \epsilon$.

Third, we replace every rule of the form $A \to u_1, \ldots, u_k$ ($k \geq 3$) by the new rules: $A \to u_1 A_1$, $A_{k-2} \to u_{k-1} u_k$, and $k-3$ rules of the form

$$A_i \to u_{i+1} A_{i+1}, \qquad (i = 1, \ldots, k-3),$$

where $A_1, \ldots, A_{k-2}$ are new variables. The degree of a rule is the length of its RHS. At this point, any remaining rules have degrees 1 or 2.

Fourth, we fix rules of degree 2. For each rule of the form $A \to bC$, where $b \in T$ and $C \in V$, we replace it by $A \to BC$ and $B \to b$ (B is a new variable). We have a similar treatment for rules of the form $A \to Cb$ or $A \to bc$.

Fifth and finally, we fix rules of degree 1. If we have a derivation of the form $A \Rightarrow^* B$ and $A \neq B$ and $B \to a$ or $B \to CD$. Then introduce the rule $A \to a$ or $A \to CD$ (respectively). Moreover remove all rules of the form $A \to B$. This completes the proof. **Q.E.D.**

This is useful if we want to show that something cannot be context-free: we only have to attack grammars in this special form. As exercise, try convert the CFG example above to Chomsky Normal Form.

§3. Pumping Lemma

As for regular languages, we are interested to know if there are non-context free languages. Which of the following two languages are CFL?
$DOUBLE = \{ww : w \in 0, 1^*\}$.
$PAL = \{ww^R : w \in 0, 1^*\}$.
To show it is something is context free, we construct an appropriate CFG or PDA. What if it is not context

free? One basic technique is via a pumping lemma for CFL. Suitably formulated, this will be a generalization of the corresponding pumping lemma for regular languages.

The idea of the pumping lemma for CFL is based on the following observation. Suppose T is a derivation tree for a word $w \in L$. Let $\pi = (A_0, A_1, \ldots, A_n, a)$ be any path in T starting from the root $A_0 = S$ (the start symbol). Here, the A_i's are variables and a is a terminal. Suppose $A_i = A_j$ for some $0 \leq i < j \leq n$. Then w can be decomposed as $w = uvxyz$ where

- x is the word generated by the subtree rooted at A_j
- v (resp., y) is the word generated by the subtrees rooted at the left (resp., right) children of node in the subpath $(A_i, \ldots, A_j)$.
- u (resp., z) is the word generated by the subtrees rooted at the left (resp., right) children of node in the subpath $(A_0, \ldots, A_i)$.

This is called the $uvxyz$-decomposition of w ("$uvxyz$" is pronounced"eu-vitz"). This is illustrated by figure 1(a). Now we can modify the tree T in two ways: (a) We may delete the subpath $(A_i, \ldots, A_{j-1})$ and all

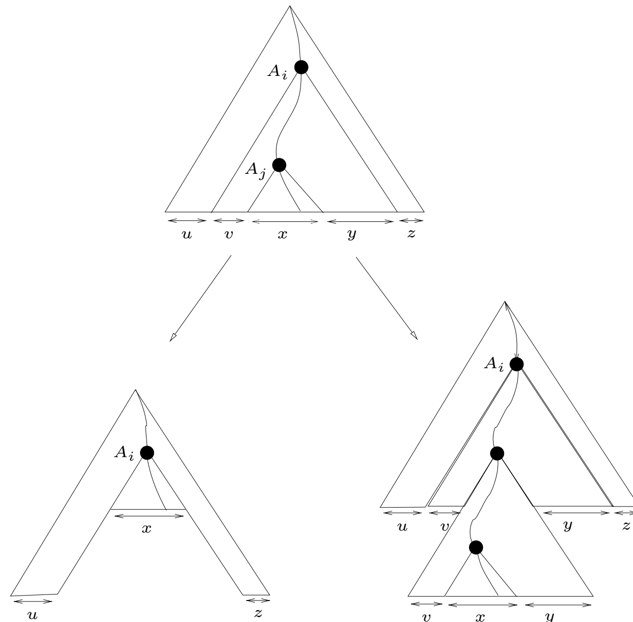


Figure 1: The $uvxyz$ decomposition of w and two transformations of the parse tree.

the associated subtrees to the left or right, and produce a derivation tree for uxz, or (b) We may duplicate the subpath $(A_i, \ldots, A_{j-1})$ and all the associated subtrees to the left or right, and produce a derivation tree for uv^2xy^2z. This shows that $uxz, uv^2xy^2z \in L$. In fact, we can repeat (b) as many times as we like to show that $uv^ixy^iz \in L$. We are now ready to prove the pumping lemma.

Theorem 2 (Pumping Lemma) *If L is a CFL then there exists $n > 0$ such that for all $w \in L$, $|w| \geq n$ implies that $w = uvxyz$ such that*
(i) $|vy| > 0$.
(ii) $uv^ixy^iz \in L$ *for all* $i \geq 0$.
(iii) $|vxy| \leq n$.

Proof. Assume L is generated by a grammar G in Chomsky Normal Form. Choose $n = 2^m$ where m is the number of variables in G. If $|w| \geq n$, then the derivation tree $T(w)$ of w must have height h that is at least $m+1$ (this is because $T(w)$ is a binary tree, and the last variable in any path that generates a leaf has degree 1). But in any path $(A_0, A_1, \ldots, A_{h-1}, a)$ of length $h \geq m+1$, there are at least $m+1$ variables (only the last node can be a terminal). Hence some variable is repeated, say $A_i = A_j$ where $0 \leq i < j \leq h-1$. The above $uvxyz$-decomposition is therefore applicable. Moreover, both (i) and (ii) are clear. To see (iii), we simply make sure that the subpath $(A_i, A_{i+1}, \ldots, A_{h-1})$ has length $\leq m$. **Q.E.D.**

We should note that there can be slight variations in the formulation of this lemma, but the underlying proof technique is the same. Hence, if this version of the pumping lemma is inadequate for a situation, you can sometimes use the underlying $uvxyz$-decomposition and argue directly.

§4. PDA

A pushdown automata (pda) is like an nfa except it has a linear storage structure called a **pushdown store** which we also call[1] a **stack** for brevity. The pda can read only the topmost symbol of the stack. Depending on the transition rules, may replace the topmost symbol by a new symbol or ϵ (this means the topmost symbol is popped); it may also place a new symbol on top of old topmost symbol (this "pushes" a new symbol). This is illustrated in figure 2.

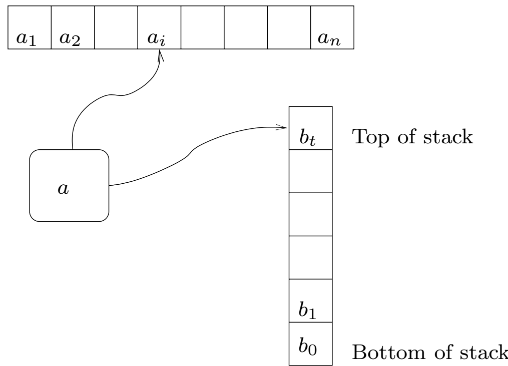


Figure 2: Illustration of a push down automata

For example, it is quite easy to design a pda to accept $L_1 = \{0^n 1^n : n \geq 0\}$.

Formally, a **pushdown automata** (pda) is a 6-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, F)$$

where Σ, Γ are **input** and **stack alphabets**, $q_0 \in Q$ is the start state, $F \subseteq Q$ the final states, and δ the transition function

$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to 2^{Q \times \Gamma_\epsilon}$$

The relation

$$(q', b') \in \delta(q, a, b) \tag{2}$$

represents a **transition rule**. Similar to grammars, we can write such a rule using the arrow notation:

$$(q, a, b \to q', b').$$

[1]A "stack" is sometimes reserved for a generalization of pushdown store in which the automata could go below the topmost symbol for the purposes of reading (but not changing) the store contents. Since we will not discuss such "stack automata", our current terminology should not be a source of confusion.

This rule says that if the current state is q, a is the current input symbol and b the top of stack, then we can next go to state q' and replace b by b'. Thus δ gives rise to a finite set of such rules, and conversely from any such set there is a unique δ. We will use whichever viewpoint that is convenient. What makes the notion of a pda transition slightly confusing is that a, b, b' can independently be ϵ.

Our goal, as usual, is to define the language $L(M)$ accepted by M. Towards this end, it is useful to formalize the concept of a **configuration** or "current state" of the computation of a pda. A configuration is simply an triple of the form

$$C = (q, w, v) \in Q \times \Sigma^* \times \Gamma^*.$$

Intuitively, q is the current state, w is the rest of the unread input (with $w[1]$ the current input symbol) and v the stack contents (with $v[1]$ the top of stack). If $C' = (q', w', v')$ is another configuration, then the binary relation

$$C \vdash_M C'$$

(or, simply, $C \vdash C'$) holds if C can reach C' by applying a single transition rule of M. More precisely, (2) is a transition rule of M and

$$\begin{aligned} w &= aw', \quad a \in \Sigma_\epsilon \\ v &= bu, \quad b \in \Gamma_\epsilon \\ v' &= b'u. \end{aligned}$$

Let $\vdash^*$ denote the reflexive transitive closure of $\vdash$. Thus $C \vdash^* C'$ iff there exists a finite sequence of the form

$$C_0 \vdash C_1 \vdash \cdots \vdash C_m, \quad m \geq 0 \tag{3}$$

where $C_0 = C$ and $C_m = C'$. We then say C **derives** C' **in** m **steps**. We define $L(M)$ to comprise all $w \in \Sigma^*$ such that

$$(q_0, w, \epsilon) \vdash^* (q, \epsilon, v)$$

for some $q \in F$ and $v \in \Gamma^*$. We say M accepts w. An alternative definition of acceptance is to insist that $v = \epsilon$, in which case we say M accepts by empty store and denote the corresponding language by $L_\epsilon(M)$.

Nondeterminism and Example. A pda in inherently nondeterministic in this definition. The nondeterminism in the above relation $C = (q, w, v) \vdash (q', w', v') = C'$ arises in three ways:

- There may be several choices of (q', b') from the set $\delta(q, a, b)$.
- If $w \neq \epsilon$ then we have a choice of $a = w[1]$ or $a = \epsilon$.
- If $v \neq \epsilon$ then we have a choice of $b = v[1]$ or $b = \epsilon$.

To illustrate this nondeterminism, let us consider a pda to accept the language $PAL = PAL_2 = \{w \in \{0,1\}^* : w = w^R\}$ of binary palindromes. We make a simple observation: if $w \in PAL$ then $w = uav$ for some $u, v \in \{0,1\}^*$ and $a \in \{0, 1, \epsilon\}$ and $u = v^R$. The idea is to operate in two phases: in the first phase, we just push u into the stack. In the second phase, we verify that the rest of the input (namely v) is equal to to the stack contents (namely u^R). But how do we know when we have reached the middle of the input, namely, the position of a? We use nondeterminism! More precisely, our pda will have 4 states: start state q_0, phase 1 state q_1, phase 2 state q_2, and accept state q_f. This pda is represented in the following state diagram:

While nondeterminism in finite automata turns out to be inessential in some sense, the nondeterminism of pda's turns out to be essential. We can define the notion of a deterministic pda (dpda) and show that the language PAL cannot be accepted by any dpda.

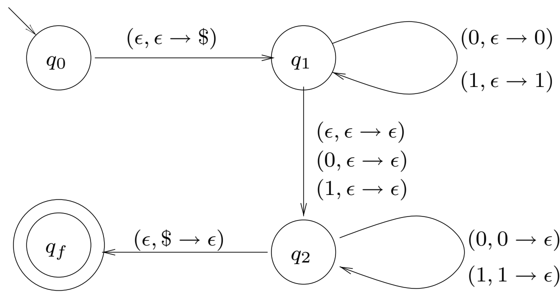


Figure 3: PDA to accept binary palindromes.

Pragmatics of pda's. We can generalize the transition function δ slightly, by allowing transition rules of the form

$$(q, a, u \to q', v) \tag{4}$$

where $q, q' \in Q$, $a \in \Sigma_\epsilon$, $u, v \in \Gamma^*$. We leave it as an exercise to show that this does enlarge the class of languages accepted by pda's. In state diagrams, the rule (4) appears as a label $(a, u \to v)$ of the edge from q to q'.

Often, a pda need to act when its stack is empty. A standard way to achieve this is to provide only one transition rule for the start state q_0, namely

$$(q_0, \epsilon, \epsilon \to q_1, \$)$$

where \$ is a special symbol that marks the bottom of the stack and q_1 some new state.

§5. PDA Characterization

The main result in this section is that pda and CFG are equivalent. We split the proof into two parts.

Lemma 3 *Every context free language is accepted by a pda.*

Proof. Assume $G = (V, \Sigma, S, R)$ is a grammar in Chomsky Normal Form. We will construct an equivalent pda $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$. Let w be generated by G using the following leftmost derivation:

$$S \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_m = w.$$

Write $u_i = w_i A_i v_i$ where w_i is a prefix of w and A_i a variable. Our pda will simulate this derivation in m steps. In the ith step, M will have finished read w_i and its stack contains $A_i v_i$ (with A_i at the top of stack). It remains to show how to proceed to the $i+1$st step.

M has only three states $Q = \{q_0, q_1, q_f\}$ and each simulation step above is performed while in state q_1. There are only two kinds of rules for A_i:
(a) $A_i \to a$. In this case, the pda has the transition rule $(q_1, a, A_i \to q_1, \epsilon)$.
(b) $A_i \to BC$. In this case, the pda has the transition rule $(q_1, \epsilon, A_i \to q_1, BC)$.

It is clear that this achieves our step by step simulation. To start off the induction, we have the transition

$$(q_0, \epsilon, \epsilon \to S\$)$$

where S is the start symbol of the grammar and \$ is a special symbol to indiate the bottom of stack. To terminate the simulation, we have the rule

$$(q_1, \epsilon, \$ \to q_f, \epsilon)$$

This is the only way to enter the final state q_f. **Q.E.D.**

The reverse direction is slightly harder but not much more.

Lemma 4 *Every language accepted by a pda is context free.*

Proof. Given a pda $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, we will construct a grammar $G = (V, \Sigma, S, R)$. We may assume that M accepts by empty stack, namely if it ever enters a state of F, then its stack is empty. In fact, we can assume $F = \{q_f\}$. Another simplification will be useful: each transition of M will either push a symbol or pop a symbol from the stack. That is each rule $(q, a, b \to q', b')$ satisfies either $b = \epsilon$ and $b' \neq \epsilon$, or $b \neq \epsilon$ and $b' = \epsilon$.

Let $p, q \in Q$. Define the language L_{pq} to comprise all words $w \in \Sigma^*$ such that $(p, w, \epsilon) \vdash^* (q, \epsilon, \epsilon)$. Our grammar G will have variable A_{pq} to generate precisely the language L_{pq}. To do this, consider a computation path starting from configuration $C_p = (p, w, \epsilon)$ and terminating in $C_q = (q, \epsilon, \epsilon)$. There are two cases.
(A) There is configuration $C_r = (r, w', \epsilon)$ such that $C_p \vdash^+ C_r \vdash^+ C_q$. In this case, $w = w''w'$ where $w'' \in L_{pr}$ and $w' \in L_{rq}$. Our grammar has the rule

$$A_{pq} \to A_{pr} A_{rq}$$

for every p, q, r. Hence, inductively we could generate w'' and w'. What is the induction hypothesis at stage n? It is that every word of length less than n in L_{pq} can be generated by our grammer, for all $p, q \in Q$.
(B) There is no such configuration C_r. We know that first and last step of this path be a push action and a pop action, respectively. If the state after the push action (resp., before the pop action) is r (resp., s), then let the corresponding transition rules be

$$(p, a, \epsilon \to r, b), \quad (s, a', b \to q, \epsilon).$$

But our grammar has, for every such pair of transitions, a rule of the form

$$A_{pq} \to a A_{rs} a'.$$

Thus $w = aw'a'$ for some $w' \in \Sigma^*$. Moreover, $w' \in L_{rs}$. Again, by induction, we know that w' can be generated from A_{rs}, and hence we are done. **Q.E.D.**

NOTES

The notion of context free grammars and pda's were introduced by Chomsky (1956) and Oettinger (1961), respectively. Context free grammars is closely related to the Backus Normal Form or Backus-Naur Form formulation, described by John Bakus and Peter Naur in the late 1950's. The dynamic programming algorithm for recognizing context free grammars (Exercise) is from T. Kasami (1965).

In the 1950s, the study of formal languages led to considerable optimism in the ability of machines to understand natural languages. In particular, machine translation (MT) was thought imminent. This turns out to be misplaced optimism. As an undergraduate, my professor (M.L. Dertouzos) told us,

``‘‘When I was a graduate student, my professor told us that MT will be possible in a few years’’``

He also gave anecdotal examples of what machines produced. Machine translating the sentence “`The spirit is willing but the flesh is weak`” into Russian, and back again, produced “`The vodka is strong but the meat is rotten`”. The sentence “`Out of sight, out of mind`” was translated into “`Blind idiot`”. Considerable progress has been made today, especially in specialized domains. For instance, in the European community (EC), official documents can be automatically translated quite accurately to all the official languages of the EC. If formal languages turns out to be inadequate for natural languages, they have great impact in the design and analysis of computer languages.

Exercises

Exercise 5.1: Recall the example of translating “`The spirit is willing but the flesh is weak`”, etc, described in the Notes. Try to outdo these machine translations from the early days of MT. [It only goes to prove that the machine translation is too hard for machines alone.] ♢

Exercise 5.2: Extend the simplistic English grammar in the introduction to generate slightly more complex sentences.
(i) Allow nouns that are either singular or plural. So you will need new variables such as `<NP Singular>`, `<NP Plural>`, etc.
(ii) Introduce simple tenses (past, present, future). ♢

Exercise 5.3: Prove that every CFL can be generated by grammar in Greibach normal form. ♢

Exercise 5.4: Show that we can allow pda's with the more general rule of the form (4). ♢

Exercise 5.5: Prove or disprove that the following are context free:
(i) $L_1 = \{w \in \{a,b,c\}^* : \#_a(w) = \#_b(w) \text{ or } \#_b(w) = \#_c(w) \text{ or } \#_b(w) = \#_c(w)\}$.
(ii) $L_2 = \{w \in \{a,b,c\}^* : \#_a(w) \neq \#_b(w) \text{ or } \#_b(w) \neq \#_c(w) \text{ or } \#_b(w) \neq \#_c(w)\}$.
(iii) $L_3 = \{w \in \{a,b,c\}^* : \#_a(w) = \#_b(w) \text{ and } \#_b(w) = \#_c(w) \text{ and } \#_b(w) = \#_c(w)\}$.
NOTE: $\#_a(w)$ counts the number of occurences of a in w. ♢

Exercise 5.6: Let $L \subseteq \{0\}^*$. Such a language is called a “sla language” (sla stands for “single letter alphabet”).
(i) Show a sla language that is not context free.
(ii) Show that any sla language L that is context free must be regular. HINT: assume G is a Chomsky Normal Form grammar for L. Try to construct an nfa to accept L. ♢

Exercise 5.7: Let $A, B \subseteq \Sigma^*$. The **right quotient** of A by B is defined to be

$$A/B := \{w \in \Sigma^* : (\exists u \in B)[wu \in A]\}.$$

(i) Show that if A is context free and B is regular, then A/B is context free.
(ii) Use part (i) to show that the language $\{0^p 1^n : p \text{ is prime}, n > p\}$ is not context free. ♢

Exercise 5.8: We say that a pda M is **nondeterministic** if it has two distinct transition rules $(q, a, b \to r, c)$ and $(q', a', b \to r', c')$ such that either (A) or (B) holds:

$$\begin{array}{ll} (A) & a = a', \text{or} \;\; a = \epsilon \text{or} \;\; a' = \epsilon, \\ (B) & b = b', \text{or} \;\; b = \epsilon \text{or} \;\; b' = \epsilon. \end{array}$$

If M is not nondeterministic, we say it is **deterministic**. Call a language **strongly deterministic context free** if it is accepted by some deterministic pda.
(i) Show that $L_1 = \{0^n 1^n : n \geq 0\}$ is strongly deterministic context free.
(ii) Show that $L_2 = \{0^n 1^m : m = n \text{or} \;\; m = 0\}$ is *not* strongly deterministic context free. ♢

Exercise 5.9: (i) Construct an efficient algorithm that, on input $\langle G, w \rangle$ where $G = (V, T, S, R)$ is a grammar in Chomsky Normal Form and w a string, decide whether $w \in L(G)$.
HINT: Use dynamic programming. For $1 \leq i \leq j \leq n$, let w_{ij} denote the substring $a_i \cdots a_j$ where $w = a_1, \ldots, a_n$. Define $V_{ij} = \{A \in V : A \Rightarrow^* w_{ij}\}$. How do you compute V_{ij} if you know the sets V_{ik}, V_{kj} for all $k = i, \ldots, j$?
(ii) What is the worst-case complexity of your algorithm, as a function of input sizes $m = |G|, n = |w|$? There is an interesting low-level issue here: $|w|$ and $|G|$ must be suitably interpreted! Note that V, T are arbitrary alphabets, but your algorithm must accept input with a fixed alphabet (say Σ). Hence use the following convention: assume Σ contains the special symbols $A, a, 0, 1$ (among others), and each symbol of V is encoded as a string of the form $A(0+1)^*$, and each symbol of T and w is encoded as a string of the form $a(0+1)^*$. The definition of $|G|$ can be taken to be the number of symbols in writing down all the rules of G plus $|V \cup T|$. Then each symbol in x has length equal to its encoding as a string in $L(a(0+1)^*)$! Similarly, you need specify your encoding G over the fixed alphabet Σ and tell us how to determine its length $|G|$. ♢

End Exercises

References