CS481F01 Solutions 4 – CFGs

A. Demers

10 Oct – due 17 Oct

1. For each of the following languages, give a context-free grammar that generates the language.

(**a**)	$\{\ 0^i1^i \mid i \geq 0\ \}^*$
(**b**)	$\{\ 0^i1^j0^k \mid i = j \vee j = k\ \}$
(**c**)	$\{\ 0^i1^j \mid i \leq j \leq 2i\ \}$
(**d**)	$\{\ w \in \{0,1\}^* \mid \sharp 1(w) = \sharp 0(w)\ \}$
(**e**)	$\{\ x \in \{0,1\}^* \mid \neg(\ (\exists i,j)(x = (0^i1^i)^j)\ )\ \}$

(answer a) We start with the productions

$$S \rightarrow AS \mid \epsilon$$

It is straightforward to argue that (in the absence of any other productions for S) $L(S) = L(A)^*$. Then add

$$A \rightarrow 0A1 \mid \epsilon$$

to generate the strings 0^i1^i and we're done.

(answer b) This is the union of two sets:

$$\{\ 0^i1^i0^k \mid i,k \geq 0\ \} \cup \{\ 0^i1^k0^k \mid i,k \geq 0\ \}$$

and can be generated by

$$\begin{aligned} S &\rightarrow AC \mid CB \\ A &\rightarrow 0A1 \mid \epsilon \\ B &\rightarrow 1B0 \mid \epsilon \\ C &\rightarrow 0C \mid \epsilon \end{aligned}$$

Note this grammar is ambiguous: any string of the form $0^i1^i0^i$ is generated in two different ways, one way using A and the other using B. It can be shown that this property is necessary. The language is *inherently ambiguous* – that is, *every* grammar generating the language is ambiguous.

(answer c) Here is an ambiguous grammar:

$$\begin{array}{rcl} S & \rightarrow & 0SA \mid \epsilon \\ A & \rightarrow & 1 \mid 11 \end{array}$$

This language is not inherently ambiguous, however. The following unambiguous language also generates it:

$$\begin{array}{rcl} S & \rightarrow & 0S11 \mid A \\ A & \rightarrow & 0A1 \mid \epsilon \end{array}$$

To argue correctness, argue that either grammar generates a subset of 0^*1^* and every production that generates a 0 generates either one or two $1s$.

(answer d) There are many similar answers. Suppose G has productions

$$S \rightarrow 0S1S \mid 1S0S \mid \epsilon$$

Clearly every generated string has equal numbers of $0s$ and $1s$, since every rule that generates a 0 also generates a 1 and *vice versa*. Can all such strings be generated by G? We can argue by induction on the string length. Clearly G generates ϵ. For nonempty strings, suppose the string starts with 0. Write it $0x$. Then

$$\sharp 1(x) = \sharp 0(x) + 1$$

Consider the shortest prefix of x for which this property holds. At least one such prefix must exist, since the property holds of x, which is a prefix of itself. Moreover, the shortest such prefix necessarily ends with 1, and the part preceding the 1 has equal numbers of $0s$ and $1s$; that is:

$$x = y1z \qquad \text{where } \sharp 1(y) = \sharp 0(y)$$

Now, by subtraction

$$\sharp 1(y1z) = \sharp 0(y1x) + 1 \quad \Rightarrow \quad \sharp 1(z) = \sharp 0(z)$$

The induction hypothesis applies to both y and z, allowing us to conclude that

$$S \rightarrow 0S1S \rightarrow^* 0y1S \rightarrow^* 0y1z$$

which is what we wanted. The case where the string starts with 1 is completely symmetric.

(answer e) This one is a bit tricky. Let

$$\bar{L} = \{ x \mid (\exists i, j)(x = (0^i 1^i)^j) \ ) \}$$

That is, $\bar{L}$ is the set of “bad” strings. How can a string *fail* to be bad? Note first that *every* $w \in \{0,1\}^*$ trivially consists of alternating sequences of $0s$ and $1s$. To be bad, a string must

1. start with a 0,
2. end with a 1,
3. have $m = n$ in each $0^m 1^n$ sequence,
4. have $m = n$ in each $1^m 0^n$ sequence.

Claim these four properties completely characterize the bad strings; the good strings (the strings in L) are strings that violate any one of the properties. Generating strings that violate the properties by a CFG is comparatively easy. First, introduce productions

$$A \rightarrow 0A \mid 1A \mid \epsilon$$

Clearly A generates any string in $\{0,1\}^*$.

To violate property (1), just add

$$S \rightarrow 1A$$

to generate all strings that do not begin with 0.

Similarly, to violate property (2) we add

$$S \rightarrow A0$$

which generates all strings that do not end with 1.

To violate property (3), we first add productions

$$\begin{aligned} B &\rightarrow 0B1 \mid 0C1 \mid 0D1 \\ C &\rightarrow 0C \mid 0 \\ D &\rightarrow D1 \mid 1 \end{aligned}$$

B generates

$$\{\ 0^m1^n \mid m, n > 0\ \wedge\ m \neq n\ \}$$

Now we can violate property (3) with the productions

$$\begin{aligned} E &\rightarrow A1 \mid \epsilon \\ F &\rightarrow 0A \mid \epsilon \\ S &\rightarrow EBF \end{aligned}$$

where E and F are used to construct left and right context for the (maximal) sequence from 0^*1^* generated by B.

Violating property (4) is completely analogous. We add productions

$$\begin{aligned} G &\rightarrow 1G0 \mid 1H0 \mid 1J0 \\ H &\rightarrow 1H \mid 1 \\ J &\rightarrow J0 \mid 0 \end{aligned}$$

so G generates

$$\{\ 1^m0^n \mid m, n > 0\ \wedge\ m \neq n\ \}$$

Now we can violate property (4) with the single production

$$S \rightarrow A0G1A$$

Note that the left and right context for G are somewhat simpler than the context for B. We assume to violate (4) an instance of G must be surrounded by $0G1$; the case where G appears at either end of the string is already covered by properties (1) and (2).

2. Recall from lecture that a *right linear* grammar is one in which the productions are of the form

$$\begin{aligned} S &\rightarrow \varepsilon \\ A &\rightarrow aB \qquad \text{or} \\ A &\rightarrow a \end{aligned}$$

These grammars are often called *regular* grammars. In this question you show why.

(a) Suppose you are given an arbitrary union-dot-star regular expression $\mathcal{E}$. Show (by induction on the structure of $\mathcal{E}$) how to construct a right linear grammar G such that $L(G) = L(\mathcal{E})$. Explain your answer.

(answer a) As announced in lecture, you may answer this problem using ϵ rules in your right linear grammars for "nearly full credit." For full credit, you can observe that the construction from the text and lectures for eliminating chain rules and ϵ rules converts a right-linear grammar with ϵ rules to a right-linear grammar without ϵ rules. Specifically, suppose L is generated by grammar G whose productions have the form

$$A \rightarrow aB \qquad \text{or} \qquad A \rightarrow \epsilon$$

We close the set of productions under the rule

$$A \rightarrow aB \;\wedge\; B \rightarrow \epsilon \qquad \Rightarrow \qquad A \rightarrow a$$

We then eliminate all ϵ rules. The resulting grammar is right-linear and generates $L - \{\epsilon\}$. To restore ϵ to the language if necessary, we add a new start symbol S' with the productions

$$\begin{array}{lll} S' \rightarrow \alpha & & \text{if } S \rightarrow \alpha \\ S' \rightarrow \epsilon & & \text{if } \epsilon \in L \end{array}$$

Thus, to show that the strict right linear grammars can generate any regular set, it is sufficient to show that right linear grammars with ϵ rules can generate any regular set. That's what we'll do here, since it's easy.

We give an inductive construction on the regular expression $\mathcal{E}$.

Case 1 ($\mathcal{E} = a$): The productions

$$S \rightarrow aA \qquad\qquad A \rightarrow \epsilon$$

are sufficient, and give us a right-linear grammar with ϵ rules as described above.

Case 2 ($\mathcal{E}_1 + \mathcal{E}_2$): Assume we have constructed grammars

$$G_i = (\ N_i, \Sigma, P_i, S_i\ ) \qquad \text{for } \mathcal{E}_i$$

We merge the sets of productions and add

$$S \rightarrow \alpha \qquad \text{for } \alpha \text{ where } S_1 \rightarrow \alpha \vee S_2 \rightarrow \alpha$$

This grammar correctly generates the union, and all productions are of the proper form.

Case 3 ($\mathcal{E}_1 \cdot \mathcal{E}_2$): Here we can rely on the assumption that the grammar uses epsilon rules. We merge the grammars G_1 and G_2; then let

$$S \;\rightarrow\; \alpha \qquad \text{for } \alpha \text{ where } S_1 \;\rightarrow\; \alpha$$

Then for every production $A \;\rightarrow\; \epsilon$ in G_1 we remove that production and replace it by the set

$$A \;\rightarrow\; \alpha \qquad \text{for } \alpha \text{ where } S_2 \;\rightarrow\; \alpha$$

The fact that this grammar generates the concatenation of $L(\mathcal{E}_1)$ and $L(\mathcal{E}_1)$ follows from the fact that every derivation in G_1 ends by replacing the rightmost symbol (a nonterminal) by ϵ.

Case 4 ($\mathcal{E}_1^*$): This case looks a lot like concatenation. We add a new start symbol S with the productions

$$\begin{array}{lll} S & \rightarrow & \epsilon \\ S & \rightarrow & \alpha \qquad \text{for } \alpha \text{ where } S_1 \;\rightarrow\; \alpha \end{array}$$

Now, for every production $A \;\rightarrow\; \epsilon$ in G_1 we add the set

$$A \;\rightarrow\; \alpha \qquad \text{for } \alpha \text{ where } S_1 \;\rightarrow\; \alpha$$

Note we do *not* remove the production $A \;\rightarrow\; \epsilon$.

This constructs a right-linear grammar with ϵ rules for any regular expression. The conversion procedure given above can be used to construct a right-linear grammar without ϵ rules if required.

(b) Suppose you are given an arbitrary right linear grammar G. Show how to construct an NFA M such that $L(M) = L(G)$. Argue that your solution is correct.

(answer b) Given a right-linear grammar without ϵ rules

$$G \;=\; (\ N, \Sigma, P, S\ )$$

We construct NDFA

$$M \;=\; (\; Q, \Sigma, \Delta, S_M, F \;)$$

as follows. First, the states are

$$Q \;=\; N \,\cup\, \{\, f \,\}$$

where f is a new state name different from any symbol of N. The transition function is

$$\Delta(A, a) \;=\; \{\, B \mid A \to aB \;\in\; P \,\} \,\cup\, \{\, f \mid A \to a \;\in\; P \,\}$$

The start and final states are

$$\begin{aligned} S_M &= \{\, S \,\} \\ F &= \{\, f \,\} \,\cup\, \{\, S \mid S \to \epsilon \;\in\; P \,\} \end{aligned}$$

This requires a (fairly routine) inductive proof that the states of the machine correspond to the nonterminals used in a derivation. Specifically,

$$\begin{array}{l} B \in \hat{\Delta}(A, w) \quad \text{iff} \quad A \to^* wB \\ f \in \hat{\Delta}(A, w) \quad \text{iff} \quad A \to^* w \quad (w \neq \epsilon) \\ S \in F \quad \text{iff} \quad S \to \epsilon \quad \text{iff} \quad \hat{\Delta}(S, \epsilon) \in F \end{array}$$

from which the equivalence of the languages follows.

3. Say grammar G is a *symmetric linear* grammar if its productions are of the form

$$\begin{array}{l} A \to aBc \\ A \to a \qquad \text{or} \\ A \to \epsilon \end{array}$$

A language L is a *symmetric linear language* if $L = L(G)$ for some symmetric linear grammar G.

(a) Give an example of a symmetric linear language that is not regular.

(answer a) Our old friend $\{0^i 1^i \mid i \geq 0\}$ will do.

(b) Prove every regular language is a symmetric linear language. For example, given a DFA M, show how to construct a symmetric linear grammar G generating $L(M)$.

(answer b) Okay, this is a bit subtle, but similar to some of the "first half" constructions we did for finite automata, and very similar to the proof given in lecture that CFLs are closed under intersection with regular sets. Given a DFA

$$M \quad = \quad (\ Q, \Sigma, \delta, s, F \ )$$

we will construct an equivalent symmetric linear grammar

$$G \quad = \quad (\ N, \Sigma, P, S \ \}$$

where

$$N \quad = \quad \{ \ S \ \} \ \cup \ (Q \times Q)$$

We denote nonterminals from $Q \times Q$ using square brackets, e.g. $[pq]$. Now the productions of G are

$$\begin{array}{lll}
S \rightarrow \epsilon & \text{if } s \in F \\
S \rightarrow a & \text{if } \delta(s,a) \in F \\
S \rightarrow a[pq]b & \text{if } \delta(s,a) = p \ \wedge \ \delta(q,b) \in F
\end{array}$$

$$\begin{array}{lll}
[pq] \rightarrow \epsilon & \text{if } p = q \\
[pq] \rightarrow a & \text{if } \delta(p,a) = q \\
[pq] \rightarrow a[p'q']b & \text{if } \delta(p,a) = p' \ \wedge \ \delta(q',b) = q
\end{array}$$

Correctness follows from a routine inductive proof of the claims

$$\begin{array}{l}
[pq] \ \rightarrow^* \ w \ \Leftrightarrow \ \hat{\delta}(p,w) \ = \ q \\
S \ \rightarrow^* \ w \ \Leftrightarrow \ (\exists f \in F)\hat{\delta}(s,w) \ = \ f
\end{array}$$

(c) Show that every symmetric linear language over a single-letter alphabet is regular.

(answer c) Gosh, over a single-letter alphabet, *the order of the symbols doesn't matter*. Given a symmetric linear grammar over the alphabet $\{0\}$, we transform the productions as follows

$A \rightarrow 0B0$	becomes	$A \rightarrow 00B$
$A \rightarrow 0$	unchanged	
$A \rightarrow \epsilon$	unchanged	

Over a single letter alphabet the generated language is unchanged. But the new grammar is (or is trivially convertible to) a right-linear grammar, and so generates a regular set.