Chapter 11

The Chomsky Hierarchy

1. Show that the set of all even palindromes over {*a*, *b*} is countably infinite.

   Solution: Let us order all even palindromes first by increasing order of length and then lexicographically where *a* comes before *b*:

   $$\{\lambda, aa, bb, aaaa, abba, baab, bbbb, aaaaaa, aabbaa, \ldots\}$$

   This infinite set can be mapped one-to-one and onto the set of positive integers. Thus, we can talk about the *i*th even palindrome. Thus, the set of all even palindromes over the given alphabet is countably infinite.

2. Design an enumeration procedure for all possible names of people (without imposing any limit on the length of any part of a name: first, middle or last names).

   Solution: Let us order names of people in increasing order of total length of names (i.e., length of first name plus length of middle names plus length of last name). Then, let us order names by lexicographic (i.e., alphabetical) order of last name, then first name, then middle names (i.e., if two people have the same last name, then they are ordered by first names; and if they have the same first name as well, then they are ordered by middle names). This way, the enumeration is proper and any name of finite length is enumerated in finite time.

3. Give an example of a formal language (other than the ones shown in Figure-11.11) that is:
   a. Linear but not regular.
   b. Linear and deterministic context-free but not regular.
   c. Deterministic context-free but not linear.
   d. Linear but not deterministic context-free.
   e. Context-free but neither deterministic nor linear.

   Solution:
   a. Linear but not regular: wcw^R ($S \rightarrow aSa \mid bSb \mid c$)
   b. Linear and deterministic context-free but not regular: wcw^R
   c. Deterministic context-free but not linear: Proper nesting of parentheses ($S \rightarrow aSb \mid SS \mid \lambda$).
   d. Linear but not deterministic context-free: $a^n b^m a^m b^n$, $n, m \geq 0$.
      A linear but not deterministic grammar for this language is:
      $S \rightarrow aSb \mid T, T \rightarrow bTa \mid \lambda$
   e. Context-free but neither deterministic nor linear: $a^n b^m a^m b^n$ or $a^n b^n a^m b^m$, $n, m \geq 0$.
      A nonlinear and not deterministic grammar for this language is:
      $S \rightarrow A \mid BB, A \rightarrow aAb \mid T, T \rightarrow bTa \mid \lambda, B \rightarrow aBb \mid \lambda$

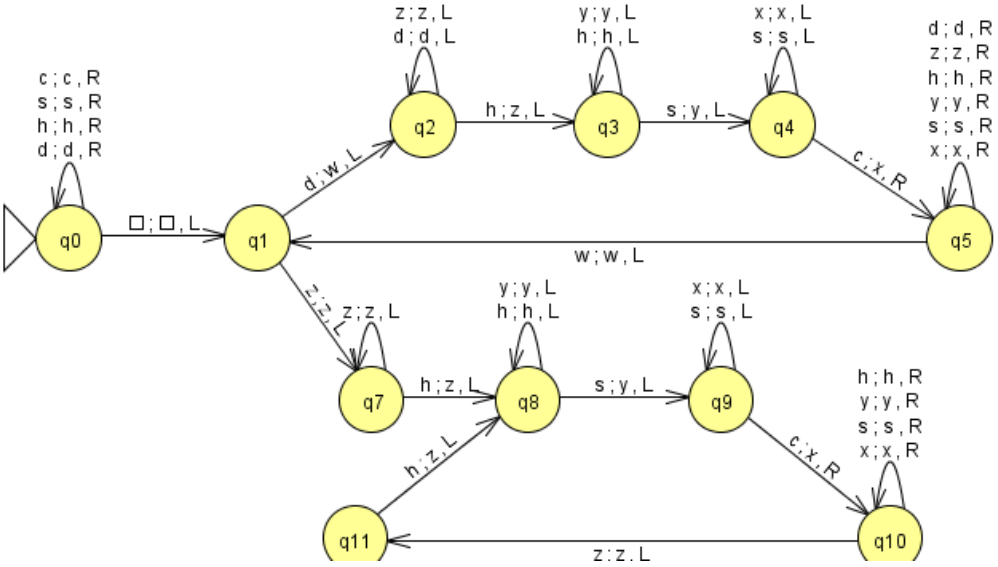
4. Old McDonald had a farm.. *E-I-E-I-O*.. and on that farm he had some chickens.. and some sheep… and some horses… and some dogs. On his farm, there must always be more chickens than sheep; more sheep than horses; and more horses than dogs. If we represent the animals on this farm using the alphabets {*c, s, h, d*} standing for chicken, sheep, horse and dog, respectively, then a string such as *ccccccsssshhhdd* represents 6 chickens, 4 sheep, 3 horses and 2 dogs (always in that order). Consider the formal language of all such strings with the above constraints.

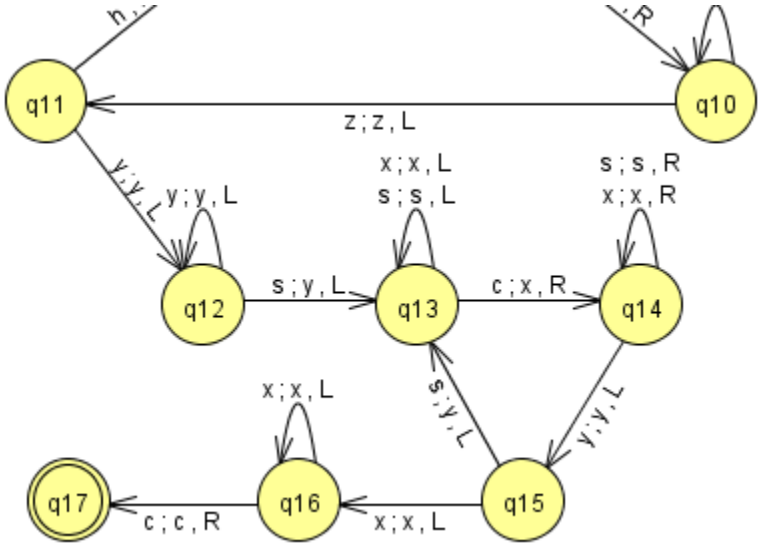
a. What class of formal languages does this language belong to if there can be at most 1000 animals on the farm? Prove your answer.

b. What class does it belong to if there is no limit on the number of animals? Prove your answer. (For example, if you claim that the language is context-free, you must prove that it is context-free AND you must prove that it is not regular.)

Solution: a. If there can be at most 1000 animals, the set of strings is finite and therefore regular (see Theorem 30 in Appendix B).

b. If there is no such limit, the language $c^n s^m h^l d^k$, $n > m > l > k$, is context-sensitive but not context-free. It is context-sensitive because we can construct a context-sensitive grammar or linear-bounded-automaton for it as shown below (i.e., just a Turing machine which anyway does not need to use any memory outside of the input strings for this language). It is not context-free (and this can be proven easily using the pumping lemma for context-free languages: choose $w = c^{m+3} s^{m+2} h^{m+1} i^m$).





5. Construct a context-sensitive grammar for the language of unary multiplication.

Solution: Strings in the language are encoded as $a^m b^n c^{mxn}$
Note that context-sensitive grammars cannot have lambda-productions since those would be shrinking rules.

LHS		RHS
S	→	a
S	→	b
S	→	abc
S	→	Aabc
A	→	B
A	→	D
A	→	AB
A	→	AD
Ba	→	aB
aBb	→	aabC
Cb	→	bCC
Cc	→	cc

LHS		RHS
Da	→	aED
Db	→	bb
Eb	→	bE
Ec	→	cc
Ea	→	aE
S	→	aG
S	→	bH
G	→	aG
H	→	bH
G	→	a
H	→	b

6. Show that (unary) division is also context-sensitive.

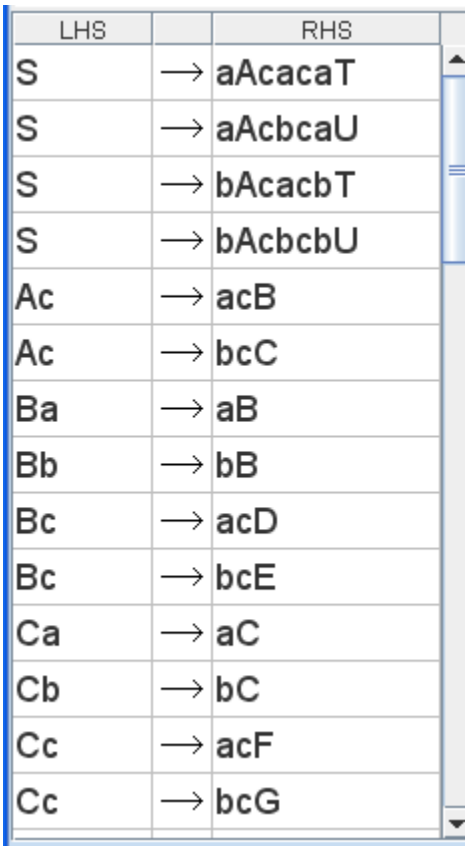
Solution: The language of division is the same as the language of multiplication since, for every string in the language of division, the number being divided is divisible by the number dividing it (i.e., the remainder is always 0). The language of multiplication $a^m b^n c^{mxn}$ can also be considered as the language of division: $a^{k/n} b^n c^k$. If we want the three parts to be re-ordered so that the quotient is the third part, that is, as $a^k b^n c^{k/n}$, the grammar for multiplication (from Exercise 5) can be modified suitably.

This is in fact one of the advantages of treating problems of formal languages!

7. Given two strings of equal length (separated by a single *c*) followed by (a single *c* and then) an interleaved string composed of the first elements of the two strings, followed by the second elements of the two strings, and so on, ending with the last elements of the two strings, in that order:
   $x_1x_2x_3...x_ncy_1y_2y_3...y_ncx_1y_1x_2y_2x_3y_3...x_ny_n$

   Construct a context-sensitive grammar that accepts such strings for the alphabet {*a, b, c*}. For example, *aabacbbabcababbaab* is in the language but not *aaacbbbcabaabb*.

   Solution: Note that CSGs cannot have lambda-productions since those would be shrinking.



LHS		RHS
S	→	aAcacaT
S	→	aAcbcaU
S	→	bAcacbT
S	→	bAcbcbU
Ac	→	acB
Ac	→	bcC
Ba	→	aB
Bb	→	bB
Bc	→	acD
Bc	→	bcE
Ca	→	aC
Cb	→	bC
Cc	→	acF
Cc	→	bcG

LHS		RHS
Ea	→	aE
Eb	→	bE
Fa	→	aF
Fb	→	bF
Da	→	aD
Db	→	bD
Ga	→	aG
Gb	→	bG
Dc	→	cD
Ec	→	cE
Fc	→	cF
Gc	→	cG
DT	→	HaaT
ET	→	HaaU

LHS		RHS
FT	→	HabT
GT	→	HabU
aH	→	Ha
bH	→	Hb
cH	→	Jc
aJ	→	Ja
bJ	→	Jb
cJ	→	Ac
T	→	a
U	→	b
DU	→	HbaT
EU	→	HbaU
FU	→	HbbT
GU	→	HbbU

LHS		RHS
S	→	acacaa
S	→	acbcab
S	→	bcacba
S	→	bcbcbb
DT	→	aaT
ET	→	aaU
FT	→	abT
GT	→	abU
DU	→	baT
EU	→	baU
FU	→	bbT
GU	→	bbU

8. Construct a PDA with two stacks for the language $a^n b^n c^n d^n e^n f^n$.

   Solution: The two-stack PDA works as follows:
   - Push *a* s onto stack 1;
   - Pop *a* s from stack 1 to match with *b* s; at the same time push *b* s onto stack 2;
   - Pop *b* s from stack 2 to match with *c* s; at the same time push *c* s onto stack 1;
   - Pop *c* s from stack 1 to match with *d* s; at the same time push *d* s onto stack 2;
   - Pop *d* s from stack 2 to match with *e* s; at the same time push *e* s onto stack 1;
   - Pop *e* s from stack 1 to match with *f* s;
   - Accept if both stacks are empty at the end of the input string.

9. Prove or disprove: Every subset of a regular language is a regular language.

   Solution: Not true. Proof by counterexample: consider the regular language a^*b^*. Its proper subset $a^n b^n$ is not a regular language as shown in Chapter 6 (see Example 6.2). It is a context-free language. Thus, subsets of regular languages need not be regular.

10. In the Chomsky Hierarchy of formal languages, why are context-sensitive languages a proper subset of recursive languages?

    Solution: Because there exist recursive languages that are not context-sensitive. There is no pumping lemma for context-sensitive languages to show this. However, a diagonalization argument can be used to show that there

exist such languages. It is very similar to how we showed the existence of languages that are not recursively enumerable.

11. Why are recursive languages a proper subset of recursively enumerable languages?

    Solution: Every recursive language is recursively enumerable because its Turing machine decider is also a Turing machine acceptor. However, there exist recursively enumerable languages that are not recursive (because their complements are not recursively enumerable). We saw an example of such a language in L_{Diag}. Therefore recursive languages are a proper subset of recursively enumerable languages.

12. Are there formal languages for which there is no Turing machine that accepts the language?

    Solution: Yes. E.g., complement of the diagonal language L_{Diag} which we called $L_{Non\text{-}RE}$ is not recursively enumerable. Therefore, it has no Turing machine acceptor.

13. How do we construct a grammar for a language for which there is no Turing machine acceptor?

    Solution: No, you cannot construct a grammar for non-recursively enumerable languages. They have no compact description and that is why they are not enumerable.

14. Is there an unrestricted grammar whose language is not context-sensitive but yet can be described concisely (without using diagonalization)? Explain.

    Solution: No. All languages that have compact descriptions can be derived by context-sensitive grammars. The difference between context-sensitive and unrestricted grammars is that context-sensitive grammars cannot have shrinking rules. The only languages that do not have context-sensitive grammars are those that cannot be described directly and concisely; their existence can only be established using indirect arguments such as diagonalization.

15. What is the smallest class of formal languages in which the complements of some of the languages do not belong to the same class?

    Solution: Context-free languages. Regular languages are closed under complementation but context-free languages are not. In fact, even deterministic context-free languages are closed under complementation.

16. Most modern programming languages have certain syntactic features that are not context-free. Yet, their syntax is specified using BNF (Backus-Naur Form) rules that are context-free. What happens to non-context-free features? How are they handled?

    Solution: They are not handled through the BNF grammars at all. Semantic rules are handled separately in the procedures of the compiler. In other words, only context-free features are described declaratively in grammars; non-context-free features are handled procedurally (i.e., by writing code in the compiler to verify semantic constraints, for example).

17. Figure 11.10 shows a Turing machine which computes and writes the product of two unary numbers onto the tape (e.g., given 1110110 as the input, i.e., 3 x 2, the output is 1110110111111, i.e., 3 x 2 = 6). Convert this to an

LBA for the language of unary multiplication whose input also includes the product of the two numbers. The LBA should accept the input string if the product is correct and reject otherwise.

Solution: The LBA is shown below: