ECS 20 **Chapter 12, Languages, Automata, Grammars**

1. Introduction
   1.1. Computer programming languages must be defined in a manner that allows programmers to write compilers to translate the languages into machine code.
   1.2. People use regular expressions for pattern matching for searching in text as well as in web searching.
   1.3. Finite state automata can be used as mathematical models of languages to aid in the of design of programs and logic circuits.
2. Alphabet, Words, Free Semigroup
   2.1. "Alphabet" Σ = a finite set of characters, e.g. $\{a, b\}$
   2.2. "Word" or "String" over Σ = 1) a finite sequence of elements of Σ, or 2) the "null string" ε. We will use u, v, and w as symbols for strings in this whole chapter, e.g. u = aaaabab = $a^3(ab)^2$
   2.3. "Length" of a string, u, over Σ = $|u|$ = $l(u)$ = The number of characters that made up the string, with the null string having length 0. For example, let $u = aab$, then $|u| = 3$.
   2.4. "Concatenation" of u and v = uv = appending v to u. For example, let $u = aab$, and $v = bbb$, then $uv = aabbbb$, and $vu = bbbaab$.
   2.5. "Subword" or "substring" of u = a sequence of successive characters within u. For example, let $u = aab$, then the substrings of u are a, aa, ab, b, aab, and ε.
   2.6. "Reversal" of a string w^R = 1) $\varepsilon^R = \varepsilon$, 2) for a non-empty string $w = x_1 x_2 \ldots x_n$, $w^R = x_n x_{n-1} \ldots x_2 x_1$. For example , let $u = aab$, then $u^R = baa$.
   2.7. Free semigroup, and free monoid (skipped)
3. Languages
   3.1. A "language," or "formal language" L over an alphabet Σ is a set of strings over Σ. Note that the empty set, $\varnothing$, satisfies the criteria for a language.
   3.2. Some languages based on length.
      3.2.1. Σ^n = the set of all strings over Σ that have length n, where n is a nonnegative integer, e.g. $\Sigma^0 = \{\varepsilon\}$, $\Sigma^1 = \{a, b\}$, $\Sigma^2 = \{aa, ab, ba, bb\}$, and $\Sigma^3 = \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$.
      3.2.2. Σ^+ = the set of all strings over Σ that have a length of at least 1.
      3.2.3. Σ^* = the set of all strings over Σ. Thus, $\Sigma^+ = \Sigma^* - \varepsilon$, and all languages over Σ are subsets of Σ^*.
   3.3. Examples with $\Sigma = \{a, b\}$
      3.3.1. $L = \{a, aa, aaa\}$
      3.3.2. $L = \{a, b\}$
      3.3.3. $L = \varnothing$
      3.3.4. $L = \{bab, ba^2b, ba^3b, \ldots ba^{11}b\}$
      3.3.5. $L = \{aab^m a^n b \mid 4 > m > n \geq 0\}$ Provide L.
      $\{aabb, aabbb, aabbbb, aabbab, aabbbab, aabbbaab\}$
      3.3.6. L_1 is the set of all strings that begin with the character a, and have a length less than four. Provide L.
      $\{a, aa, aaa, ab, aba, abb\}$
      3.3.7. A "palindrome" is a string that looks the same if the order of its characters is reversed. For instance, aba and $baab$ are palindromes. L_2 is the set of all palindromes that have length 3 or 4. Provide L
      $\{aaa, aba, bbb, bab, aaaa, abba, bbbb, baab\}$
   3.4. Operations on Languages that are over the same alphabet. Let $L_1 = \{ab, abb, abbb, b\}$, and $L_2 = \{b, ba\}$
      3.4.1. "Concatenation" $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\} = \{abb, abbb, abbbb, bb, abba, abbba, abbbba, bba\}$
         3.4.1.1. "Powers" $L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = LL$, $L^m = L^{m-1}L$ for $m > 1$.
      3.4.2. "Union" $L_1 \cup L_2 = \{ab, abb, abbb, b, ba\} = \{u \mid u \in L_1 \vee u \in L_2\}$ = all strings that are contained in either languages.
         3.4.2.1. "Kleene closure" = $L^* = L^0 \cup L^1 \cup L^2 \ldots = \bigcup_{k=0}^{\infty} L^k$
         3.4.2.2. $L^+ = L^* - \varepsilon$
      3.4.3. "Intersection" $L_1 \cap L_2 = \{b\} = \{u \mid u \in L_1 \wedge u \in L_2\}$ = all strings that are contained in both languages.
      3.4.4. "Complement" $\neg L_1 = \Sigma^* - L_1$ = all strings over the alphabet that are not in the language.
      3.4.5. "Reversal" $L_1^R = \{ba, bba, bbba, b\} = \{u^R \mid u \in L_1\}$
4. Regular Expressions, Regular Languages
   4.1. A recursive definition of a "regular expression" over Σ:
      4.1.1. Base: $\varnothing$, ε, (), and each individual symbol in Σ are regular expressions over Σ.

4.1.2. Recursion: If *r* and *s* are regular expressions over Σ, then the following are also regular expressions over Σ:
4.1.2.1. (*rs*) where *rs* denotes the concatenation of *r* and *s*.
4.1.2.2. (*r* | *s*) or (*r* ∨ *s)* which denote either one of the regular expressions *r* or *s*.
4.1.2.3. (*r**) where *r** is called the "Kleene closure" of *r*, which allows any number (including zero) of repeated concatenations of *r*.
4.1.2.4. The order of precedence has parentheses highest, then Kleene closure, concatenation is next, and "|" is the lowest.
4.1.3. Nothing is a regular expression over Σ except objects defined in 4.1.1 and 4.1.2.
4.1.4. Note that if the alphabet contains a regular expression metacharacter, i.e. (,), |, or *, then to indicate the alphabet symbol instead of its metacharacter meaning we preface it with a backslash, e.g. \(or \|.

4.2. Add parentheses to make the order of precedence clear in the given expressions:
4.2.1. (*a* | *b***b)(a** | *ab)*
4.2.2. *0*1* | *0(0*1)**
4.2.3. *(x* | *yz**)**(yx* | *(yz)**z)*

4.3. Regular expression examples based on Σ = {*a*, *b*}
4.3.1. *a***b** = All strings starting with any number of *a*'s, followed by any number of *b*'s.
4.3.2. *a***b** | *b***a** = All strings starting with *a*'s followed by *b*'s, or starting with *b*'s followed by *a*'s.
4.3.3. *a*(a^2 | *ab)** = All strings starting with an *a*, followed by an any finite combination of *aa*'s and *ab*'s.
4.3.4. (*a* | *b*)* = All strings of *a*'s and *b*'s = Σ*

4.4. Shorthands for common regular expressions
4.4.1. [*abc*] = (*a* | *b* | *c*)
4.4.2. A "character class" uses the notation [*beginning character – ending character*], when the delimiting characters are in an understood order, e.g [*a-z*] = any lower case letter, [0-9] = any digit, [*A-D 1-5*] = (*A* | *B* | *C* | *D* | *1* | *2* | *3* | *4* | *5*).
4.4.3. A ^ at the beginning of a character class indicates that a character of the same type as those in the range of the class is to occur at that point in the string, expect for one of the specific characters indicated after the ^ sign, e.g. [^D-Z][^3-8] = (A | B | C)(0 | 1 | 2 | 9). When a character class is not given, a beginning ^ indicates any character not inside the brackets, e.g. [^ac2] = any character except a, c, or 2.
4.4.4. A single period stands for an arbitrary character. For example with Σ = {*a*, *b*, *c*}, (*a.c)* = (*aac* | *abc* | *acc*)
4.4.5. If *r* is a regular expression, then *r*+ denotes the concatenation of *r* with itself any positive finite number of times, i.e. *r*+ = *rr**, e.g. (*ab)*+ = *ab(ab)**
4.4.6. If *r* is a regular expression, then *r*{*n*} denotes the concatenation of *r* with itself exactly *n* times, e.g. (*ab)*{3} = *ababab*

4.5. Use the shorthands to write a regular expression to define the given set of strings.
4.5.1. All strings that are written in lower-case letters, and start with the letters *pre*, but do not consist of *pre* all by itself.
4.5.2. All UC Davis SIDs (which consist of three digits, a hyphen, two digits, another hyphen, and finally four more digits), where the first two digits are always 99.
4.5.3. All words that are written in upper-case letters, and do not start with one of the vowels, A, E, I, O, or U, but contain exactly two of these vowels next to each other.

4.6. A recursive definition of a language, *L(r)*, defined by a regular expression, *r* over the alphabet Σ.
4.6.1. Base: *L*(∅) = ∅, *L*(ε) = {ε}, *L*(*x*) = {*x*}, where $x \in \Sigma$.
4.6.2. Recursion: If *L*(*r*) and *L*(*s*) are the languages defined by regular expressions *r* and *s* over Σ:
4.6.2.1. *L(rs)* = *L(r)L(s)*, the concatenation of the languages
4.6.2.2. *L(r* | *s)* = *L(r)* ∪ *L(s)*, the union of the languages
4.6.2.3. *L(r**) = (*L(r)*)* the Kleene closure of *L(r)*.

4.7. *L* is called a "regular language" over Σ if there exists a regular expression *r* over Σ such that *L* = *L(r)*.
4.7.1. All finite languages are regular.
4.7.2. {ε} is regular.
4.7.3. All of the strings over the alphabet {*a*, *b*} which contain an even number of *a*'s is a regular language.
4.7.4. $L = \{a^n b^n \mid n \geq 0\}$ is not regular because the number of *a*'s controls the number of *b*'s, which is not allowed by any of the rules of regular expressions.

4.8. Examples of regular languages
4.8.1. Write five strings that belong to the language defined by the given regular expressions.
4.8.1.1. 0*1(0*1*)*
4.8.1.2. *b***|***b***ab***

4.8.1.3. *x*(yxxy|x)**

4.8.2. Use words to describe the language defined by the language defined by the regular expression *b*ab*ab*a*
The language consists of all strings of *a*'s and *b*'s that contain exactly three *a*'s, and end with an *a.*

4.8.3. Do the following strings belong to the languages defined by the given regular expressions?

4.8.3.1. Expression: (*b* | ε)*a(a|b)*a(b* | ε) Strings: *aaaba. baabb*

4.8.3.2. Expression: (01*2)* Strings: 120, 01202

5. Finite State Automata

5.1. A finite state automaton (FSA), *M*, consists of five parts:

5.1.1. A finite set (alphabet) of input symbols, Σ.

5.1.2. A finite set of (internal) states that the automaton can be in, *S.*

5.1.3. An initial state s_0, with $s_0 \in S$.

5.1.4. A designated set of states called the accepting (final) states, *Y*, where $Y \in S$.

5.1.5. A next-state function, $N: S \times \Sigma$, that associates a "next-state" to each ordered pair consisting of a "current state" and a "current input." For each state *s* in *S*, and input symbol *m* in Σ, *N(s,m)* is the state to which *M* goes if *m* is input to *M* when *M* is in state *s*.

5.1.5.1. A "next-state table" shows the values of the next-state function for all possible states *s* and input symbols *i.*

5.2. An automaton can be specified in a quintuple, (Σ, *S*, s_0, *N, Y).*

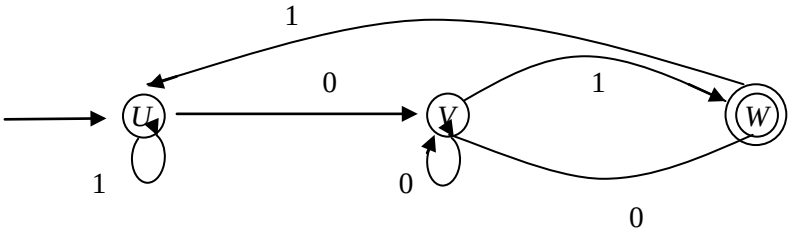
5.3. The operation of a FSA is commonly described with a "state transition diagram."

5.3.1. States are represented by circles, with accepting states represented by double circles.

5.3.2. There is one arrow that points to the initial state, and other arrows that are labeled with input symbols and point from each state to other states to indicate the action of the nest-state function.

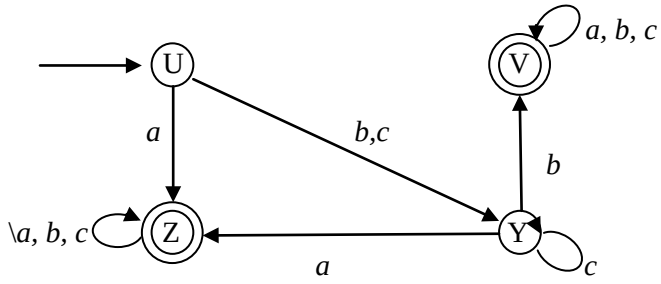
5.4. Example of an FSA, ({0,1}, {*U, V, W*}, {*U}, N,* {W}), where *N* is specified in the following next-state table.

State	Input	
	0	**1**
U	*V*	*U*
V	*V*	*W*
W	*V*	*U*



5.5. Provide the state transition diagram of the FSA based on ({*a, b, c}*, {*U, V, Y, Z*}, *U, N,* {*V, Z*}), and the following next-state table.

State	Input		
	a	***b***	***c***
U	*Z*	*Y*	*Y*
V	*V*	*V*	*V*
Y	*Z*	*V*	*Y*
Z	*Z*	*Z*	*Z*



5.6. Each automaton *M* with input alphabet Σ defines a language over Σ, denoted by *L(M)* as follows. Let Σ* be the set of all strings over Σ, and let *w* be a string in Σ*. Then "*w* is accepted by *M"* if, and only if *M* goes to an accepting

state when the symbols of *w* are input to *M* in sequence from left to right, start when *M* is in its initial state. The "language accepted by *M*," denoted *L(M)*, is the set of all strings that are accepted by *M*.

5.6.1. What is the language accepted by the automaton in 5.4? The set of all strings of 0's and 1's that end in 01.

5.6.2. What is the regular expression that defines the language of the automaton in 5.4? (0 | 1)*01.

5.7. A language *L* is regular if and only if there is a finite state automaton *M* such that *L = L(M)*.

5.8. The "Pumping Lemma" states that if an automaton *M* with *k* states accepts a word *w* from Σ where $|w| > k$, then there must be a section, or sections in the string, that are repeated any number of times, before reaching an accepting state. This simply means that there must be a repeatable cycle(s) of states in the next-state function of the automaton. Note that this can be proved using the pigeonhole principle.

5.8.1. The Pumping Lemma or the pigeonhole principle can be used to show that $L = \{a^m b^m \mid m \text{ is positive}\}$ is not regular.

6. Grammar

6.1. A "formal grammar" (sometimes simply called a grammar) is a set of formation rules for strings in a formal language. A grammar, *G*, consists of four parts, denoted by the quadruple *G = (V, T, S, P)* :

6.1.1. A finite set of symbols, *V*, for vocabulary. This is the alphabet of the grammar.

6.1.2. A finite set of terminal symbols, *T*. The rest of the symbols in *V* are nonterminals, *N*.

6.1.3. A distinguished symbol $S \in N$ that is the "start symbol."

6.1.4. A finite set *P* of production rules, each rule of the form $(V)^* N (V)^* \rightarrow V^*$ where * is the Kleene star operator. That is, each production rule maps from one string of symbols (the "head") to another (the "body"), where the first string contains an arbitrary number of symbols provided at least one of them is a nonterminal.

6.2. We will denote terminals with *italic* lower case Latin letters, and nonterminals with *italic* capital Latin letters, with *S* as the start symbol. Since we have used all of the Latin letters, we will use Greek letters to denote words in *V*.

6.3. Often a grammar is specified by just listing its production rules, with *S* assumed to be the start symbol, and its vocabulary limited to the symbols used in the production rules.

6.3.1. Example grammar *G* = (*V, T, S, P*), where *V* = {*a, b, A, B, S*}, *T* = {*a, b*}, *S* is the start symbol, and *P* = (*S* →*ABa, A*→*BB, B*→*ab, AB*→*b*). This could be represented with simply: *G = S* →*ABa, A*→*BB, B*→*ab, AB*→*b*. Two words that would satisfy this grammar are *ba*, and *abababa*. Are there others?

6.4. We will denote that a word, α can be transformed into another word β by the application of a production rule using a "⇒", so $\alpha \Rightarrow \beta$. If more than one production rule must be applied then $\alpha \Rightarrow \Rightarrow \beta$, or $\alpha \overset{*}{\Rightarrow} \beta$.

6.5. The language of grammar *G*, denoted by *L(G)*, consists of all words in *T** that can be obtained from the start symbol of *S* by applying the production rules: $L(G) = \{\alpha \in T^* \mid S \Rightarrow\Rightarrow \alpha\}$

6.6. Four Types of Grammar.

6.6.1. In the 1950s the linguist Noam Chomsky attempted to understand the underlying principles of human speech by developing a theory of formal languages, including a classification of formal grammars based on restrictions to the forms of the production rules.

6.6.2. Type 0 has no restrictions on the productions.

6.6.3. Type 1 has every production in the form $\alpha \rightarrow \beta$, where $|\alpha| \leq |\beta|$, or of the form $\alpha \rightarrow \varepsilon$. That is, no production can produce a longer string. Such grammars are called "context sensitive" grammars since production rules such as *aAb* → *aBb* are allowed which make the replacement of a nonterminal dependent on other symbols.

6.6.4. Type 2 has every production in the form $\alpha \rightarrow \beta$, where α is a nonterminal. Such grammars are called a "context free" grammars since a nonterminal symbol that is on the left side of a production can be replaced in a string whenever it occurs, no matter what else is in the string..

6.6.5. Type 3 has every production in the form $\alpha \rightarrow \beta$, where α is a nonterminal, and β is a terminal, a terminal followed by a nonterminal, or ε. Such grammars are called "regular" grammars, and produce regular languages.

6.6.6. Note that each type of grammar is a more restrictive case of the previous grammars. Thus, all regular grammars are context free grammars, but the reverse is not necessarily the case.

6.7. All context free grammars can be represented graphically by means of an ordered, rooted tree *T*, called a "derivation tree" or a "parse tree."

6.7.1. Example for *G = S* →*ABa, A*→*BB, B*→*ab, AB*→*b*.

6.8. Backus-Naur Form is used for describing the productions of context free grammars.

6.8.1. "::=" is used instead of "→"

6.8.2. Every nonterminal is enclosed in brackets ⟨⟩

6.8.3. All productions with same nonterminal head are combined into one statement with the bodies listed on the right of ::= separated by vertical bars, e.g. *A* → *aB, A* → *b, A* →*BC* becomes *A* ::= *a*⟨*B*⟩ | *b* | ⟨*B*⟩⟨*C*⟩

6.9. Machine and Grammars (skipped until next chapter).