Regular Languages

- A language L is *regular* if it is the language accepted by some DFA.
  - Note: the DFA must accept only the strings in L, no others.
- Some languages are not regular.
  - Intuitively, regular languages "cannot count" to arbitrarily high integers.

Example: A Nonregular Language

$L_1 = \{0^n1^n \mid n \geq 1\}$

- Note: a^i is conventional for i a's.
  - Thus, $0^4 = 0000$, e.g.
- Read: "The set of strings consisting of n 0's followed by n 1's, such that n is at least 1.
- Thus, $L_1 = \{01, 0011, 000111, \ldots\}$

$L_2 = \{w \mid w \text{ in } \{(,)\}^* \text{ and } w \text{ is } balanced\}$

- Balanced parentheses are those sequences of parentheses that can appear in an arithmetic expression.
- E.g.: (), ()(), (()), (()()),...

The Pigeonhole Principle

In Example 2.13 we used an important reasoning technique called the *pigeonhole principle*. Colloquially, if you have more pigeons than pigeonholes, and each pigeon flies into some pigeonhole, then there must be at least one hole that has more than one pigeon. In our example, the "pigeons" are the sequences of n bits, and the "pigeonholes" are the states. Since there are fewer states than sequences, one state must be assigned two sequences.

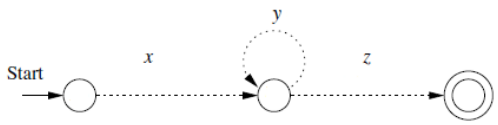
The pigeonhole principle may appear obvious, but it actually depends on the number of pigeonholes being finite. Thus, it works for finite-state automata, with the states as pigeonholes, but does not apply to other kinds of automata that have an infinite number of states.

To see why the finiteness of the number of pigeonholes is essential, consider the infinite situation where the pigeonholes correspond to integers $1, 2, \ldots$. Number the pigeons $0, 1, 2, \ldots$, so there is one more pigeon than there are pigeonholes. However, we can send pigeon i to hole $i + 1$ for all $i \geq 0$. Then each of the infinite number of pigeons gets a pigeonhole, and no two pigeons have to share a pigeonhole.

Theorem 4.1: (The *pumping lemma for regular languages*) Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|w| \geq n$, we can break w into three strings, $w = xyz$, such that:

1. $y \neq \epsilon$.
2. $|xy| \leq n$.
3. For all $k \geq 0$, the string xy^kz is also in L.

That is, we can always find a nonempty string y not too far from the beginning of w that can be "pumped"; that is, repeating y any number of times, or deleting it (the case $k = 0$), keeps the resulting string in the language L.



Informal Comments

- A *context-free grammar* is a notation for describing languages.
- It is more powerful than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

- Basic idea is to use "variables" to stand for sets of strings (i.e., languages).
- These variables are defined recursively, in terms of one another.
- Recursive rules ("productions") involve only concatenation.
- Alternative rules for a variable allow union.

5.1.2 Definition of Context-Free Grammars

There are four important components in a grammatical description of a language:

1. There is a finite set of symbols that form the strings of the language being defined. This set was $\{0,1\}$ in the palindrome example we just saw. We call this alphabet the *terminals*, or *terminal symbols*.

2. There is a finite set of *variables*, also called sometimes *nonterminals* or *syntactic categories*. Each variable represents a language; i.e., a set of strings. In our example above, there was only one variable, P, which we used to represent the class of palindromes over alphabet $\{0,1\}$.

3. One of the variables represents the language being defined; it is called the *start symbol*. Other variables represent auxiliary classes of strings that are used to help define the language of the start symbol. In our example, P, the only variable, is the start symbol.

4. There is a finite set of *productions* or *rules* that represent the recursive definition of a language. Each production consists of:

   (a) A variable that is being (partially) defined by the production. This variable is often called the *head* of the production.

   (b) The production symbol $\rightarrow$.

   (c) A string of zero or more terminals and variables. This string, called the *body* of the production, represents one way to form strings in the language of the variable of the head. In so doing, we leave terminals unchanged and substitute for each variable of the body any string that is known to be in the language of that variable.

   We saw an example of productions in Fig. 5.1.

The four components just described form a *context-free grammar*, or just *grammar*, or *CFG*. We shall represent a CFG G by its four components, that is, $G = (V, T, P, S)$, where V is the set of variables, T the terminals, P the set of productions, and S the start symbol.

- *Terminals* = symbols of the alphabet of the language being defined.
- *Variables* = *nonterminals* = a finite set of other symbols, each of which represents a language.
- *Start symbol* = the variable whose language is the one being defined.

- A *production* has the form variable (*head*) -> string of variables and terminals (*body*).
- Convention:
  - A, B, C,... and also S are variables.
  - a, b, c,... are terminals.
  - ..., X, Y, Z are either terminals or variables.
  - ..., w, x, y, z are strings of terminals only.
  - α, β, γ,... are strings of terminals and/or variables.

CFG for $\{ 0^n1^n \mid n \geq 1\}$

- Productions:

  S -> 01

  S -> 0S1
- Basis: 01 is in the language.
- Induction: if w is in the language, then so is 0w1.

- Here is a formal CFG for $\{ 0^n1^n \mid n \geq 1\}$.
- Terminals = {0, 1}.
- Variables = {S}.
- Start symbol = S.
- Productions =

  S -> 01

  S -> 0S1