Strings

- Strings of characters are fundamental building blocks in computer science.
- The alphabet over which the strings are defined may vary with the application. For our purposes, we define an alphabet to be any nonempty finite set.
- The members of the alphabet are the symbols of the alphabet.
- We generally use capital Greek letters $\sum$ and τ to designate alphabets and a typewriter font for symbols from an alphabet.

input alphabet

Strings (Cont.)

The following are a few examples of alphabets.

- $\sum_{binary} = \{0, 1\}$ Binary alphabet
- $\sum_{eng}$ = { a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z }
- τ = { 0, 1, x, y, z }

1. $\sum$ = Alphabet = Set of Symbols.
2. $\tau = \{a, b, c, d\}$ (Always a finite set!)

String or words

Length = 9 String = $\overset{1}{a}\overset{2}{a}\overset{3}{a}\overset{4}{a}\overset{5}{a}\overset{6}{a}\overset{7}{a}\overset{8}{a}\overset{9}{a}$

- A string over an alphabet is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas. $\{a, b, c, \cdots, z\}$
- STRING = A finite sequence of symbols. (baccadda). length = 8 $\Sigma = \{a, b, c, d\}$
- The empty string is a string with zero occurances of the symbol (no symbol). $\Sigma = \{0, 1\}$ Empty String (ϵ) → Epsilon
- Length of the string is the number of symbol in the string. If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$.
- If w is a string over Σ, the length of w, written $|w|$, is the number of symbols that it contains. $w = \overset{1}{C}\overset{2}{S}\overset{3}{E}\overset{4}{'}\overset{5}{S}\overset{6}{'}$ $|w| = 6$ $w_1 = CSE$ $|w_1| = 3$
- The reverse of w, written w^R, is the string obtained by writing w in the opposite order (i.e., $w_n w_{(n-1)} \cdots w_1$). Reverse of $w_1 = w_1^R = ESC$

String $V =$ Computer $= xUy$ $U = \overset{x=\epsilon}{}$Computer$^{y=\epsilon}$

where x and y are any string including ϵ. $x=\epsilon$ $U=$ Com $y=$puter

- **Substring:** U is a substring of V if $\exists xUy = V$, where, U,x,y,V $\in \Sigma^*$
  - String U is a substring of V if U appears consecutively within w. For example, cad is a substring of abracadabra.

$\underbrace{\text{abra}}_{x}\underbrace{\text{cad}}_{U}\underbrace{\text{abra}}_{y}$

$y=\epsilon$ $U=$ ter $x=$compu $y=$ter $U=$ pu $x=$ com

- **Prefix:** U is a prefix of V if $\exists Ux = V$, where, U,x,V $\in \Sigma^*$
- **Concatenation:** If we have string x of length m and string y of length n, the concatenation of x and y, written xy, is the string obtained by appending y to the end of x, as in $x_1 \cdots x_m \ y_1 \cdots y_n$. To concatenate a string with itself many times, we use the superscript notation x^k to mean

Concatenation of ω_2 and $\omega_1 = \omega_2 \cdot \omega_1$

$\omega=$ HELLO

Prefix of $\omega=$ HELLO, HELL, HEL, HE, H $=$ CSEHELLO

Suffix of $\omega=$ HELLO, ELLO, LLO, LO, O

Let $\omega_1=$ HELLO $\omega_2=$ CSE Concatenation of ω_1 and $\omega_2 = \omega_1\omega_2$

$=$ HELLOCSE

Operations on Alphabets, Strings

Let $\omega = CSE$ $\omega\omega = \omega^2 = CSECSE$ $\omega\omega\omega = \omega^3 = CSECSECSE$

$\omega.\omega.\omega. \cdots \omega$ k times $= \omega^k$

Strings of Length

- **Power of an alphabet:** $\Sigma = \{0, 1\}$ $3 \to \Sigma^3 = \{0,1\}.\{0,1\}.\{0,1\}$

  0 $\Sigma_1^0 = \in$ empty string $= \{00,01,10,11\}.\{0,1\}$

  $1 \to \Sigma_1^1 = \Sigma = \{0,1\}$ $= \{000,001,010,011,$

  $2 \to \Sigma_1^2 = \Sigma.\Sigma = \{00,01,10,11\} \Leftarrow \{0,1\} \bullet \{0,1\}.$ $100,101,110,111\}$

  $k \to \Sigma_1^k = w| \; |w| = k$ and w are words or strings formed using Σ

- **Kleen Closure:** $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \; ....$

  $\Sigma^* = \{\in, 0,1,00,10,01,11.......\}$, *where*, $\Sigma = \{0, 1\}$

  $\Sigma^* = \cup_{i \geq 0} \Sigma^i$ $\to$ Any string using 0 or 1. includes $\in$

  $\Sigma^* = \{w||w| \geq 0\}$

- **Positive Closure:** $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \; ....$

  $\Sigma^+ = \{0, 1, 00, 10, 01, 11.......\}$, where, $\Sigma = \{0, 1\}$

No Epsilon

If $|\Sigma| = m$ then $|\Sigma^k| = m^k$ $\Sigma^+ = \Sigma^* - \{\in\}$

- A **language** is a finite set of non empty strings.
- $L_1 = \subseteq \Sigma^*$
- $L_2 = \{\} = \phi$ this language doesnt even contain $\in$. It is called empty language.
- $L_3 = \{w||w| \leq 2\}$ It is called finite language. ⇒ length of string ≤ 2 i.e. length = 0, 1, 2
- $L_4 = \{0^n1^n|n \geq 1\}$ It is called infinite language.

For Ex : $\Sigma = \{0, 1\}$

7 element

finite Lang $L_3 = \{\epsilon, 0, 1, 00, 01, 10, 11\}$

$n \geq 1$

i.e $n = 1, 2, 3, \ldots$

$L_4 = 0^n 1^n = \{01, 0011, 000111, \ldots\}$

n=1 n=2 n=3

Infinite Lang.

Boolean logic

- Boolean logic is a mathematical system built around the two values TRUE and FALSE.
- The values TRUE and FALSE are called the Boolean values and are often represented by the values 1 and 0.
- We use Boolean values in situations with two possibilities, such as:
  - A wire that may have a high or a low voltage, a proposition that may be true or false,
  - Or a question that may be answered yes or no.

Boolean logic (cont.)

- We can manipulate Boolean values with the Boolean operations.
- The simplest Boolean operation is the negation or NOT operation, designated with the symbol $\neg$.
- The negation of a Boolean value is the opposite value. Thus $\neg\, 0 = 1$ and $\neg\, 1 = 0$.
- We designate the conjunction or AND operation with the symbol $\wedge$.
- The conjunction of two Boolean values is 1 if both of those values are 1.
- The disjunction or OR operation is designated with the symbol $\vee$.
- The disjunction of two Boolean values is 1 if either of those values is 1.

Boolean logic (cont.)

AND $\wedge$ OR $\vee$ NOT $\neg$

We summarize this information as follows.

$$
\begin{array}{lll}
0 \wedge 0 = 0 & 0 \vee 0 = 0 & \neg 0 = 1 \\
0 \wedge 1 = 0 & 0 \vee 1 = 1 & \neg 1 = 0 \\
1 \wedge 0 = 0 & 1 \vee 0 = 1 & \\
1 \wedge 1 = 1 & 1 \vee 1 = 1 &
\end{array}
$$

- We use Boolean operations for combining simple statements into more complex Boolean expressions, just as we use the arithmetic operations $+$ and $\times$ to construct complex arithmetic expressions.

For example, if P is the Boolean value representing the truth of the statement: "the sun is shining" and Q represents the truth of the statement: "today is Tuesday",

- We may write $P \wedge Q$ to represent the truth value of the statement: "the sun is shining and today is Tuesday"
- And similarly for $P \vee Q$ with and replaced by or.
- The values P and Q are called the operands of the operation.

the sun is shining or today is Tuesday.

Boolean logic (cont.)

- The exclusive or, or XOR, operation is designated by the $\oplus$ symbol and is 1 if either but not both of its two operands is 1.
- The equality operation, written with the symbol $\leftrightarrow$, is 1 if both of its operands have the same value.
- The implication operation is designated by the symbol $\rightarrow$ and is 0 if its first operand is 1 and its second operand is 0; otherwise, is 1.

Boolean logic (cont.)

XOR Equality Implication

We summarize this information as follows.

$0 \oplus 0 = 0 \quad 0 \leftrightarrow 0 = 1 \quad 0 \rightarrow 0 = 1$

$0 \oplus 1 = 1 \quad 0 \leftrightarrow 1 = 0 \quad 0 \rightarrow 1 = 1$

$1 \oplus 0 = 1 \quad 1 \leftrightarrow 0 = 0 \quad 1 \rightarrow 0 = 0$

$1 \oplus 1 = 0 \quad 1 \leftrightarrow 1 = 1 \quad 1 \rightarrow 1 = 1$

Boolean logic (cont.)

In fact, we can express all Boolean operations in terms of the AND and NOT operations, as the following identities show.

- $P \vee Q$ $\quad\quad \neg(\neg P \wedge \neg Q)$
- $P \rightarrow Q$ $\quad\quad \neg P \vee Q$
- $P \leftrightarrow Q$ $\quad\quad (P \rightarrow Q) \wedge (Q \rightarrow P)$
- $P \oplus Q$ $\quad\quad \neg(P \leftrightarrow Q)$

The two expressions in each row are equivalent.

Boolean logic (cont.)

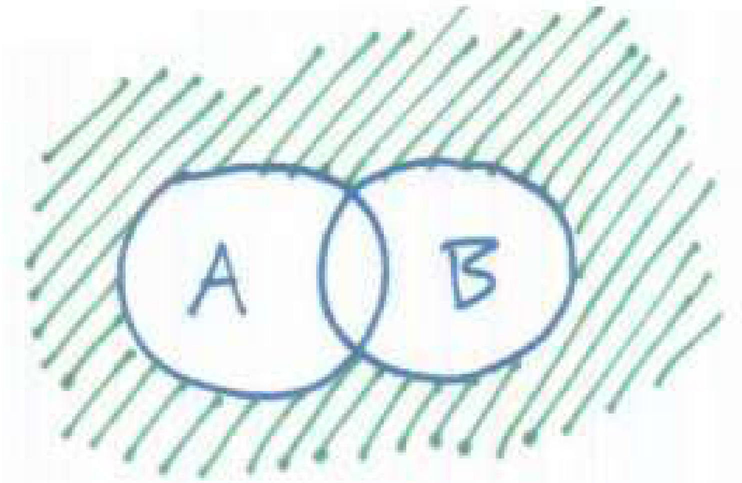
The distributive law for AND and OR comes in handy when we manipulate Boolean expressions.

- It is similar to the distributive law for addition and multiplication, which states that:
  - $a \times (b + c) = (a \times b) + (a \times c)$.
- The Boolean version comes in two forms:
  - $P \wedge (Q \vee R)$ equals $(P \wedge Q) \vee (P \wedge R)$, and its dual
  - $P \vee (Q \wedge R)$ equals $(P \vee Q) \wedge (P \vee R)$.

Boolean logic (cont.)

DeMorgan's laws:

- $\neg(A \vee B) = (\neg A) \wedge (\neg B)$
- $\overline{\mathrm{A} \cup \mathrm{B}} = \overline{\mathrm{A}} \cap \overline{\mathrm{B}}$
- Using Venn Diagrams:



- $\neg(A \wedge B) = (\neg A) \vee (\neg B)$
- $\overline{\mathrm{A} \cap \mathrm{B}} = \overline{\mathrm{A}} \cup \overline{\mathrm{B}}$
- Using Venn Diagrams: