Chapter 2
Languages
Languages

- **Defn.** A language is a set of *strings* over an *alphabet*.
  
  - A more restricted definition requires some forms of restrictions on the strings, i.e., strings that satisfy certain properties

- **Defn.** The *syntax* of a language restricts the set of strings that satisfy certain *properties*.
Defn. A string over an alphabet $X$, denoted $\Sigma$, is a finite sequence of elements from $X$, which are indivisible objects

- e.g., Strings can be words in English

- The set of strings over an alphabet is defined recursively (as given below)
Languages

- Defn. 2.1.1. Let $\Sigma$ be an alphabet. $\Sigma^*$, the set of strings over $\Sigma$, is defined recursively as follows:
  
  (i) **Basis:** $\lambda \in \Sigma^*$, the null string
  
  (ii) **Recursion:** $w \in \Sigma^*$, $a \in \Sigma \Rightarrow wa \in \Sigma^*$
  
  (iii) **Closure:** $w \in \Sigma^*$ is obtained by step (i) and a finite # of step (ii)

- The length of a string $w$ is denoted $\text{length}(w)$

- Q: If $\Sigma$ contains $n$ elements, how many possible strings over $\Sigma$ are of length $k$ ($\in \Sigma^*$)?
Languages

Example: Given $\Sigma = \{a, b\}$, $\Sigma^*$ includes $\lambda$, $a$, $b$, $aa$, $ab$, $ba$, $bb$, $aaa$, ...

Defn 2.1.2. A language over an alphabet $\Sigma$ is a subset of $\Sigma^*$.

Defn 2.1.3. Concatenation, is the fundamental binary operation in the generation of strings, which is associative, but not commutative, is defined as

i. Basis: If $\text{length}(v) = 0$, then $v = \lambda$ and $uv = u$

ii. Recursion: Let $v$ be a string with $\text{length}(v) = n > 0)$. Then $v = wa$, for string $w$ with length $n-1$ and $a \in \Sigma$, and $uv = (uw)a$
Languages

- **Example:** Let $\alpha = ab$, $\beta = cd$, and $\gamma = e$
  - $\alpha(\beta\gamma) = (\alpha\beta)\gamma$, but
  - $\alpha\beta \neq \beta\alpha$, unless $\alpha = \lambda$, $\beta = \lambda$, or $\alpha = \beta$.

- Exponents are used to abbreviate the *concatenation* of a string with itself, denoted $u^n$ ($n \geq 0$)

- **Defn 2.1.5.** Reversal, which is a unary operation, rewrites a string *backward*, is defined as
  i) Basis: If $\text{length}(u) = 0$, then $u = \lambda$ and $\lambda^R = \lambda$.
  ii) Recursion: If $\text{length}(u) = n (> 0)$, then $u = wa$ for some string $w$ with length $n - 1$ and some $a \in \Sigma$, and $u^R = aw^R$

- **Theorem 2.1.6.** Let $u, v \in \Sigma^*$. Then, $(uv)^R = v^R u^R$. 
Languages

- Finite language specification
  - **Example 2.2.1.** The language \( L \) of string over \( \{a, b\} \) in which each string begins with an ‘a’ and has even length.
    - i) Basis: \( aa, ab \in L \).
    - ii) Recursion: If \( u \in L \), then \( uaa, uab, uba, ubb \in L \).
    - iii) Closure: \( u \in L \) only if \( u \) is obtained from the basis elements by a finite number of applications of the recursive step.

- Use *set operations* to construct complex sets of strings.
  - **Defn 2.2.1.** The *concatenation* of languages \( X \) and \( Y \), denoted \( XY \), is the language
    \[
    XY = \{ uv \mid u \in X \text{ and } v \in Y \}
    \]
  - Given a set \( X \), \( X^* \) denotes the set of strings that can be defined with \( \cdot \) and \( \cup \).
Languages

- **Defn 2.2.2.** Let $X$ be a set. Then

\[ X^* = \bigcup_{i=0}^{\infty} X^i \quad \text{and} \quad X^+ = \bigcup_{i=1}^{\infty} X^i \]

- $X^+ = XX^*$ or $X^+ = X^* - \{\lambda\}$

- **Observation:** Formal (i) *recursive* definitions, (ii) *concatenation*, and (iii) *set operations* precisely define *languages*, which require the *unambiguous specification* of the strings that belong to the language.
Defn 2.3.1 Let \( \Sigma \) be an alphabet. The regular sets over \( \Sigma \) are defined recursively as follows:

(i) **Basis**: \( \emptyset, \{ \lambda \}, \text{ and } \{ a \}, \forall a \in \Sigma \), are regular sets over \( \Sigma \).

(ii) **Recursion**: Let \( X \) and \( Y \) be regular sets over \( \Sigma \). The sets \( X \cup Y, XY \) and \( X^* \) are regular sets over \( \Sigma \).

(iii) **Closure**: Any regular set over \( \Sigma \) is obtained from (i) and by a finite number of applications of (ii).

**Example**: Describe the content of each of the following regular sets:

(i) \( \{ aa \}^* \), (ii) \( \{ a \}^* \cup \{ b \}^* \), (iii) \( \{ \{ a \} \cup \{ b \} \}^* \), (iv) \( \{ a \} (\{ b \} \{ c \})^* \)

- Regular expressions are used to abbreviate the descriptions of regular sets, e.g., replacing \( \{ b \} \) by \( b \), union (\( \cup \)) by (,), etc.
Languages

Examples.

(a) The set of strings over \{a, b\} that contains the substrings aa or bb

\[ L = \{(a} \cup \{b\}\)*\{a\}\{a\} \cup \{b\}\)* \cup \{(a} \cup \{b\}\)*\{b\}\{b\}\{(a} \cup \{b\}\)* \]

(b) The set of strings over \{a, b\} that do not contain the substrings aa and bb

\[ L = (a, b)* - ((a, b)*aa(a, b)* \cup (a, b)*bb(a, b)*) \text{ [non-regular set]} \]

(c) The set of strings over \{a, b\} that contain exactly two b's

\[ L = \{a\}\{b\}\{a\}\{b\}\{a\}\* \]
Defn 2.3.2. Let $\Sigma$ be an alphabet. The regular expressions over $\Sigma$ are defined recursively as follows:

(i) **Basis**: $\emptyset$, $\lambda$, and $a$, $\forall a \in \Sigma$, are regular expressions over $\Sigma$.

(ii) **Recursion**: Let $u$ and $v$ be regular expressions over $\Sigma$. Then $(u, v)$, $(uv)$ and $(u)^*$ are regular expressions over $\Sigma$.

(iii) **Closure**: Any regular expression over $\Sigma$ is obtained from (i) and by a finite number of applications of (ii).

It is assumed that the following precedence is assigned to the operators to reduce the number of parentheses:

$\ast$, $\bullet$, $\cup$
Regular Sets and Expressions

- **Example:** Give a regular expression for each of the following over the alphabet \( \{ 0, 1 \} \):
  
  - \( \{ w \mid w \text{ begins with a } '1' \text{ and ends with a } '0' \} \)
  - \( \{ w \mid w \text{ contains at least three } 1's \} \)
  - \( \{ w \mid w \text{ is any string without the substring } '11' \} \)
  - \( \{ w \mid w \text{ is a string that begin with a } '1' \text{ and contain exactly two } 0's \} \)
  - \( \{ w \mid w \text{ contains an even number of } 0's, \text{ or contains exactly two } 1's \text{ and nothing else } \} \)

- Regular expression definition of a language is **not** unique.
## Regular Expression Identities

### TABLE 2.1

<table>
<thead>
<tr>
<th></th>
<th>Regular Expression Identities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$\emptyset u = u \emptyset = \emptyset$</td>
</tr>
<tr>
<td>2.</td>
<td>$\lambda u = u \lambda = u$</td>
</tr>
<tr>
<td>3.</td>
<td>$\emptyset^* = \lambda$</td>
</tr>
<tr>
<td>4.</td>
<td>$\lambda^* = \lambda$</td>
</tr>
<tr>
<td>5.</td>
<td>$u \cup v = v \cup u$</td>
</tr>
<tr>
<td>6.</td>
<td>$u \cup \emptyset = u$</td>
</tr>
<tr>
<td>7.</td>
<td>$u \cup u = u$</td>
</tr>
<tr>
<td>8.</td>
<td>$u^* = (u^<em>)^</em>$</td>
</tr>
<tr>
<td>9.</td>
<td>$u (v \cup w) = uv \cup uw$</td>
</tr>
<tr>
<td>10.</td>
<td>$(u \cup v) w = uw \cup vw$</td>
</tr>
<tr>
<td>11.</td>
<td>$(uv)^* u = u (vu)^*$</td>
</tr>
</tbody>
</table>
| 12. | $(u \cup v)^* = (u^* \cup v)^*$  
|   | $= u^* (u \cup v)^* = (u \cup vu^*)^*$  
|   | $= (u^* v^*)^* = u^* (vu^*)^*$  
|   | $= (u^* v)^* u^*$ |
Regular Expressions

There exist non-regular expressions such as

- \{a^n b^n \mid n \geq 0\}
- \{(0, 1)^n (0, 1)(10)^n (0, 1)^* 1 \mid n \geq 0\}

Table 2.1 Regular Expression Identities

- \phi^* = \lambda;  The * operation puts together any number of strings from the language to get a (new) string in the result. If the language is empty, the * operation can put together 0 strings, giving only the null string (\lambda).
- \phi u = u \phi = \phi;  Concatenating \phi to any set yields \phi.
- (a, \lambda)(b, \lambda) = \{\lambda, a, b, ab\}.  How about \(c^*(b, ac^*)^*\)?
- The regular expression \(c^*(b, ac^*)^*\) yields all strings that do not contain the substring bc.
<table>
<thead>
<tr>
<th>Grammars</th>
<th>Languages</th>
<th>Accepting Machines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 0 grammars,</td>
<td>Recursively enumerable</td>
<td>TM</td>
</tr>
<tr>
<td>Phrase-structure grammars,</td>
<td>Unrestricted</td>
<td>NDTM</td>
</tr>
<tr>
<td>Unrestricted grammars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type 1 grammars,</td>
<td>Contest-sensitive</td>
<td>Linear-bounded</td>
</tr>
<tr>
<td>Context-sensitive grammars,</td>
<td>languages</td>
<td>Automata</td>
</tr>
<tr>
<td>Monotonic grammars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type 2 grammars,</td>
<td>Context-free</td>
<td>PDA</td>
</tr>
<tr>
<td>Context-free grammars</td>
<td>languages</td>
<td></td>
</tr>
<tr>
<td>Type 3 grammars,</td>
<td>Regular</td>
<td>FSA</td>
</tr>
<tr>
<td>Regular grammars,</td>
<td>languages</td>
<td>NDFA</td>
</tr>
<tr>
<td>Left-linear grammars,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right-linear grammars</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>