1.0 Languages, Expressions, Automata

**Alphabet:** a finite set, typically a set of symbols.

**Language:** a particular subset of the strings that can be made from the alphabet.

*ex:* an alphabet of digits = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}

a language of integers = \{0, 1, 2, ..., 101, 102, 103, ..., -1, -2, etc.\}

Note that strings such as 2–20 would not be included in this language.

**Regular Expression:**
A pattern that generates (only) the strings of a desired language. It is made up of letters of the language’s alphabet, as well as of the following special characters:

- ( ) used for grouping
- * repetition
- • concatenation (usually omitted)
- + denotes a choice (“or”).
- λ a special symbol denoting the null string

**Precedence from highest to lowest: ( ) * • +

**formal (recursive) definition:**
If A is an alphabet, and a ∈ A, then a is a regular expression.

λ is a regular expression.

If r and s are regular expressions, then the following are also regular expressions: r*, r·s = rs, r + s, and ( r )

**examples:** (assume that A = \{a, b\} )

- a·b·a (or just aba ) matched only by the string aba
- ab + ba matched by exactly two strings: ab and ba
- b* matched by \{ λ, b, bb, bbb, ....\}
- b(a + ba*)*a (b + λ) matched by bbaaab, and many others

Some convenient extensions to regular expression notation:

- aa = a^2, bbbb = b^4, etc.
- a^+ = a·a* = \{ any string of a’s of positive length, i.e. excludes λ \}
- ex: (ab)^2 = abab ≠ a^2 b^2 , so don’t try to use “algebra”.
- ex: (a+b)^2 = (a+b)(a+b) = aa or ab or ba or bb.
- ex: (a+b)^* any string made up of a’s and b’s.
Examples of regular expressions over \{a, b\}:

- all strings that begin with a and end with b
  \[ a (a + b)^* b \]

- all non empty strings of even length
  \[ (aa + ab + ba + bb)^* \]

- all strings with at least one a
  \[ (a + b)^* a (a + b)^* \]

- all strings with at least two a's
  \[ (a + b)^* a (a + b)^* a (a + b)^* \]

- all strings of one or more b's with an optional single leading a
  \[ (a + \lambda) b^* \]

- the language \{ ab, ba, abaa, bbb \}
  \[ ab + ba + abaa + bbb \quad \text{or} \]
  \[ ab (\lambda + aa) + b (a + bb) \quad \text{or} \]
  \[ (a + bb) b + (b + aba) a \quad \text{or}? \]

Tips:

- Check the simplest cases
- Check for “sins of omission” (forgot some strings)
- Check for “sins of commission” (included some unwanted strings)

More examples

Find a regular expression for the following sets of strings on \{ a, b \}:

- All strings with at least two b's.
  \[ (a + b)^* b (a + b)^* b (a + b)^* \]

- All strings with exactly two b’s.
  \[ a^* b a^* b a^* \]

- All strings with at least one a and at least one b.
  \[ (a + b)^* (ab + ba) (a + b)^* \]

- All strings which end in a double letter (two a’s or two b’s).
  \[ (a + b)^* (aa + bb) \]

- All strings of even length (includes 0 length).
  \[ (aa + bb + ab + ba)^* \]
Finite Automata: a particular, simplified model of a computing machine, that is a “language recognizer”:

A finite automaton (FSA) has five pieces:

1. \( S = \) a finite number of states,
2. \( A = \) the alphabet,
3. \( S_i = \) the **start** state,
4. \( Y = \) one or more final or “accept” states, and
5. \( F = \) a transition function (mapping) between states, \( F: S \times A \rightarrow S \).

The transition function \( F \) is usually presented in one of two ways:

- as a table (called a transition table), or
- as a graph (called a transition diagram).

**Transition Table (example):**

\( A = \{ a, b \}, S = \{ s_0, s_1, s_2 \}, S_i = s_1, Y = \{ s_0, s_2 \} \)

<table>
<thead>
<tr>
<th>current state</th>
<th>current input</th>
<th>( F )</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_0 )</td>
<td>( s_0 )</td>
<td>( s_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( s_1 )</td>
<td>( s_1 )</td>
<td>( s_0 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( s_2 )</td>
<td>( s_0 )</td>
<td>( s_0 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of a finite automaton with input aaba and output yes or no]

1.0 Languages, Expressions, Automata
Note that this FSA is:

- **Complete** (no undefined transitions)
  - Not
  - Or

- **Deterministic** (no choices)
  - Not

“Skeleton Method” - a useful solution technique in limited cases:

- The “skeleton” is a sequence of states assuming legal input.
- Construct the skeleton, presume that no additional states will be needed.
- The FSA must be **complete and deterministic**: for \( A = \{ a, b \} \), every state has exactly two arcs leaving it, one labeled “a” and one labeled “b”.

*example (skeleton): All strings containing abaa*
Examples
Assume \( A = \{ a, b \} \). Construct the following automata which:

1. Accepts strings of the form \((a+b)^*\)

2. Accepts \( \lambda \) only.

3. Accepts strings which begin with \( a \)

4. Accepts strings containing ‘aa’ (skeleton method)
5. All words containing at least two a’s

4. All words containing exactly two a’s

Equivalence of Regular Expressions and Finite-State Automata

1. For every regular expression “R”, defining a language “L”, there is a FSA “M” recognizing exactly L.

2. For every FSA “M”, recognizing a language “L”, there is a regular expression “R” matching all the strings of L and no others.
   (we will prove this later)

Question: is there a FSA that can recognize \{ \lambda, ab, aabb, aaabbb, \ldots \} ??

Answer: No, because we need to “remember” how many a’s have been seen to verify that there are as many b’s. Since an FSA can only have a finite number of states there cannot be enough states to count the a’s.

We need a more powerful kind of recognizer... that is, a grammar.