Formal Definition of Regular Expression:

Example:

- In the above example, the language of all words that contain both an a and ab is defined by the expression

  $(a + b)^*a(a + b)^*b(a + b)^* + bb^*aa^*$

- The only words that do not contain both an a and ab are the words of all a's, all b's, or Λ.
- When these are included, we get everything. Hence, the expression
  $(a + b)^*a(a + b)^*b(a + b)^* + bb^*aa^* + a^* + b^*$
  defines all possible strings of a's and b's, including (accounted for in both a and b).
- Thus: $(a + b)^* = (a + b)^*a(a + b)^*b(a + b)^* + bb^*aa^* + a^* + b^*$

Example:

- The following equivalences show that we should not treat expressions as algebraic polynomials:
  $(a + b)^* = (a + b)^* + (a + b)^*$
  $(a + b)^* = (a + b)^* + a^*$
  $(a + b)^* = (a + b)^*(a + b)^*$
  $(a + b)^* = a(a + b)^* + b(a + b)^* + \Lambda$
  $(a + b)^* = (a + b)^*ab(a + b)^* + b^*a^*$
- The last equivalence may need some explanation:
  - The first term in the right hand side, $(a + b)^*ab(a + b)^*$, describes all the words that contain the substring ab.
  - The second term, b^*a^* describes all the words that do not contain the substring ab (i.e., all a's, all b's, Λ, or some b's followed by some a's).

Example:

- Let V be the language of all strings of a's and b's in which either the strings are all b's, or else an a followed by some b's. Let V also contain the word Λ. Hence,
  $V = \{\Lambda, a, b, ab, bb, abb, bbb, abbb, bbbb, \ldots\}$
- We can define V by the expression: $b^* + ab^*$ where Λ is included in b^*.
- Alternatively, we could define V by $(\Lambda + a)b^*$

which means that in front of the string of some b's, we have either an a or nothing.

- Hence, $(\Lambda + a)b^* = b^* + ab^*$
- Since $b^* = \Lambda b^*$, we have $(\Lambda + a)b^* = b^* + ab^*$
  which appears to be distributive law at work.
- However, we must be extremely careful in applying distributive law. Sometimes, it is difficult to determine if the law is applicable.

Product Set:

- If S and T are sets of strings of letters (whether they are finite or infinite sets), we define the **product set** of strings of letters to be

ST = {all combinations of a string from S concatenated with a string from T in that order}.

Example:

- If S = {a, aa, aaa} and T = {bb, bbb} then
  ST = {abb, abbb, aabb, aabbb, aaabb, aaabbb}
- Note that the words are not listed in lexicographic order.
- Using regular expression, we can write this example as
  (a + aa + aaa)(bb + bbb) = abb + abbb + aabb + aabbb + aaabb + aaabbb

Example:

- If M = {Λ, x, xx} and N = {Λ, y, yy, yyy, yyyy, …} then
- MN ={Λ, y, yy, yyy, yyyy,…x, xy, xyy, xyyy, xyyyy, …xx, xxy, xxyy, xxyyy, xxyyyy, …}
- Using regular expression (Λ + x + xx)(y*) = y* + xy* + xxy*

Languages Associated with Regular Expressions:

Definition:

- The following rules define the **language associated** with any regular expression:
- Rule 1: The language associated with the regular expression that is just a single letter is that one-letter word alone, and the language associated with Λ is just {Λ}, a one-word language.
- Rule 2: If r_1 is a regular expression associated with the language L_1 and r_2 is a regular expression associated with the language L_2, then:

(i) The regular expression $(r_1)(r_2)$ is associated with the product L_1L_2, that is the language L_1 times the language L_2: language$(r_1r_2) = L_1L_2$

(ii) The regular expression $r_1 + r_2$ is associated with the language formed by the union of L_1 and L_2: language$(r_1 + r_2) = L_1 + L_2$

(iii) The language associated with the regular expression $(r_1)^*$ is L_1^*, the Kleene closure of the set L_1 as a set of words: language$(r_1^*) = L_1^*$

Finite Languages Are Regular:

Theorem 5:

- **If** L **is a finite language (a language with only finitely many words), then** L **can be defined by a regular expression. In other words, all finite languages are regular.**

Proof

- Let L be a finite language. To make one regular expression that defines L, we turn all the words in L into boldface type and insert plus signs between them.

- For example, the regular expression that defines the language
  L = {baa, abbba, bababa} is baa + abbba + bababa
- This algorithm only works for finite languages because an infinite language would become a regular expression that is infinitely long, which is forbidden.

How Hard It Is To Understand A Regular Expression:

Let us examine some regular expressions and see if we could understand something about the languages they represent.

Example:

- Consider the expression
  (a + b)*(aa + bb)(a + b)* =(arbitrary)(double letter)(arbitrary)
- This is the set of strings of a's and b's that at some point contain a double letter.

Let us ask, "What strings do not contain a double letter?" Some examples are
Λ; a; b; ab; ba; aba; bab; abab; baba; …

- The expression (ab)* covers all of these except those that begin with b or end with a. Adding these choices gives us the expression: (Λ + b)(ab)*(Λ + a)
- Combining the two expressions gives us the one that defines the set of all strings
  (a + b)*(aa + bb)(a + b)* + (Λ + b)(ab)*(Λ + a)

Examples:

- Note that (a + b*)* = (a + b)*

since the internal * adds nothing to the language. However,

(aa + ab*)* ≠ (aa + ab)*

since the language on the left includes the word *abbabb*, whereas the language on the right does not. (The language on the right cannot contain any word with a double b.)

Example

- Consider the regular expression: (a*b*)*.
- The language defined by this expression is all strings that can be made up of factors of the form a*b*.
- Since both the single letter a and the single letter b are words of the form a*b*, this language contains all strings of a's and b's. That is,
  (a*b*) = (a + b)*
- This equation gives a big doubt on the possibility of finding a set of algebraic rules to reduce one regular expression to another equivalent one.

Introducing EVEN-EVEN:

- Consider the regular expression E = [aa + bb + (ab + ba)(aa + bb)*(ab + ba)]*
- This expression represents all the words that are made up of *syllables* of three types: $type_1$ = aa
  $type_2$ = bb
  $type_3$ = (ab + ba)(aa + bb)*(ab + ba)

- Every word of the language defined by E contains an **even number of** a**'s and an even number of** b**'s**.
- All strings with an **even number of** a**'s and an even number of** b**'s** belong to the language defined by E.

Algorithms for EVEN-EVEN:

- We want to determine whether a long string of a's and b's has the property that the number of a's is even and the number of b's is even.

Algorithm 1: Keep two binary flags, the a-flag and the b-flag. Every time an a is read, the a-flag is reversed (0 to 1, or 1 to 0); and every time a b is read, the b-flag is reversed. We start both flags at 0 and check to be sure they are both 0 at the end.

Algorithm 2: Keep only one binary flag, called the $type_3$-flag. We read letter in two at a time. If they are the same, then we do not touch the $type_3$-flag, since we have a factor of $type_1$ or $type_2$. If, however, the two letters do not match, we reverse the $type_3$-flag. If the flag starts at 0 and if it is also 0 at the end, then the input string contains an even number of a's and an even number of b's.

- If the input string is

(aa)(ab)(bb)(ba)(ab)(bb)(bb)(bb)(ab)(ab)(bb)(ba)(aa) then, by Algorithm 2, the $type_3$-flag is reversed 6 times and ends at 0.

- We give this language the name EVEN-EV EN. so, EVEN-EV EN ={Λ, aa, bb, aaaa, aabb, abab, abba, baab, baba, bbaa, bbbb, aaaaaa, aaaabb, aaabab, …}