Theory of Computational

Arithmetic Expressions:

- Suppose we ask ourselves what constitutes a valid arithmetic expression or AE for short.
- The alphabet for this language is
- Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (,)}

Arithmetic Expression AE:

- Obviously, the following expressions are not valid:

  (3 + 5) + 6) 2(/8 + 9) (3 + (4-)8)
- The first contains unbalanced parentheses; the second contains the forbidden substring /; the third contains the forbidden substring -).
- Are there more rules? The substrings // and */ are also forbidden.
- Are there still more?
- The most natural way of defining a valid AE is by using a **recursive definition**, rather than a long list of forbidden substrings.

Recursive Definition of AE:

- **Rule 1:** Any number (positive, negative, or zero) is in AE.
- **Rule 2:** If x is in AE, then so are
  (i) (x)
  (ii) -x (provided that x does not already start with a minus sign)
- **Rule 3:** If x and y are in AE, then so are
  (i) x + y (if the first symbol in y is not + or -)
  (ii) x - y (if the first symbol in y is not + or -)
  (iii) x * y
  (iv) x / y
  (v) x ** y (our notation for exponentiation)
- The above definition is the most natural, because it is the method we use to recognize valid arithmetic expressions in real life.
- For instance, we wish to determine if the following expression is valid:

  (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) - 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:

  Is (2 + 4) OK? Yes Is (9 - 3) OK? Yes Is 7 * (9 - 3)/4 OK? Yes, and so on.

- Note that the recursive definition of the set AE gives us the possibility of writing 8/4/2, which is ambiguous, because it could mean 8/(4/2) = 4 or (8/4)/2 = 1.
- However, the ambiguity of 8/4/2 is a problem of *meaning*. There is no doubt that this string is a word in AE, only doubt about what it means.
- By applying Rule 2, we could always put enough parentheses to avoid such confusion.
- The recursive definition of the set AE is useful for proving many theorems about arithmetic expressions, as we shall see in the next few slides.

Defining Languages by Another New Method:

Regular Expressions:

Defining Languages by Another New Method
Formal Definition of Regular Expressions
Languages Associated with Regular Expressions
Finite Languages Are Regular
How Hard It Is to Understand a Regular Expression
Introducing EVEN-EVEN

Language-Defining Symbols:

- We now introduce the use of the Kleene star, applied not to a set, but directly to the letter x and written as a superscript: x*.
- This simple expression indicates some sequence of x's (may be none at all):

  x* =Λ or x or x^2 or x^3…

  = x^n for some n = 0, 1, 2, 3, …
- Letter **x** is intentionally written in boldface type to distinguish it from an alphabet character.
- We can think of the star as an unknown power. That is, **x*** stands for a string of x's, but we do not specify how many, and it may be the null string .
- The notation x* can be used to define languages by writing, say L_4 = language (x*)
- Since x* is any string of x's, L_4 is then the language of all possible strings of x's of any length (including Λ).
- We should not confuse x* (which is a **language-defining symbol**) with L_4 (which is the **name** we have given to a certain language).
- Given the alphabet = {a, b}, suppose we wish to define the language L that contains all words of the form one *a* followed by some number of *b*'s (maybe no *b*'s at all); that is

  L = {a, ab, abb, abbb, abbbb, …}

- Using the language-defining symbol, we may write
  L = language (ab*)
- This equation obviously means that L is the language in which the words are the concatenation of an initial a with some or no b's.
- From now on, for convenience, we will simply say **some** b**'s** to mean **some or no** b**'s**. When we want to mean **some positive number of** b**'s**, we will explicitly say so.
- We can apply the Kleene star to the whole string ab if we want:
  (ab)* = Λ or ab or abab or ababab…
- Observe that
  (ab)* ≠ a*b*
- Because the language defined by the expression on the left contains the word abab, whereas the language defined by the expression on the right does not.
- If we want to define the language L1 = {x; xx; xxx; …} using the language-defining symbol, we can write
  L1 = language (xx*)
  which means that each word of L1 must start with an x followed by some (or no) x's.
- Note that we can also define L1 using the notation + (as an exponent) introduced in Chapter 2:
  L1 = language(x^+)
- which means that each word of L1 is a string of some positive number of x's.

Plus Sign:

- Let us introduce another use of the plus sign. By the expression
  x + y
  where x and y are strings of characters from an alphabet, we mean **either** x **or** y.
- Care should be taken so as not to confuse this notation with the notation + (as an exponent).

Example:

- Consider the language T over the alphabet
  Σ = {a; b; c}:
- T = {a; c; ab; cb; abb; cbb; abbb; cbbb; abbbb; cbbbb; …}
- In other words, all the words in T begin with either an a or a c and then are followed by some number of b's.
- Using the above plus sign notation, we may write this as
  T = language ((a+ c)b*)

Example:

- Consider a finite language L that contains all the strings of a's and b's of length three exactly:
  L = {aaa, aab, aba, abb, baa, bab, bba, bbb}
- Note that the first letter of each word in L is either an a or a b; so are the second letter and third letter of each word in L.
  - Thus, we may write: L = language((a+ b)(a + b)(a + b))
  - or for short, L = language((a+ b)^3)

Example:

- In general, if we want to refer to the set of all possible strings of a's and b's of any length whatsoever, we could write language ((a+ b)*)
- This is the set of **all possible strings** of letters from the alphabet Σ = {a, b}, **including the null string**.
- This is powerful notation. For instance, we can describe all the words that begin with first an a, followed by anything (i.e., as many choices as we want of either a or b) as: a(a + b)*

Formal Definition of Regular Expressions:

- The set of **regular expressions** is defined by the following rules:
- Rule 1: Every letter of the alphabet Σ can be made into a regular expression by writing it in **boldface, Λ** itself is a regular expression.
- Rule 2: If r_1 and r_2 are regular expressions, then so are:
  (i) (r_1)
  (ii) r_1r_2
  (iii) $r_1 + r_2$
  (iv) r_1*
- Rule 3: Nothing else is a regular expression.

Note: If r_1 = aa + b then when we write r_1* , we really mean (r_1)*, that is r_1* = (r_1)* = (aa + b)*

Example:

- Consider the language defined by the expression: (a + b)*a(a + b)*
- At the beginning of any word in this language we have *(a + b)**, which is any string of *a*'s and *b*'s, then comes an *a*, then another any string.
- For example, the word abbaab can be considered to come from this expression by 3 different choices:
  (Λ)a(bbaab) or (abb)a(ab) or (abba)a(b)

- This language is the set of all words over the alphabet Σ = {a, b} that have at least one a.
- The only words left out are those that have only b's and the word Λ.
  These left out words are exactly the language defined by the expression b*.
- If we combine this language, we should provide a language of all strings over the alphabet Σ = {a, b}. That is, (a + b)* = (a + b)*a(a + b)* + b*

Example:

- The language of all words that have at least two a's can be defined by the expression: (a + b)*a(a + b)*a(a + b)*
- Another expression that defines all the words with at least two a's is
  b*ab*a(a + b)*
- Hence, we can write: (a + b)*a(a + b)*a(a + b)* = b*ab*a(a + b)*

where by the equal sign we mean that these two expressions are **equivalent** in the sense that they describe the same language.

Example:

- The language of all words that have at least one a and at least one b is somewhat trickier. If we write
  (a + b)*a(a + b)*b(a + b)*

then we are requiring that an a must precede a b in the word. Such words as ba and bbaaaa are not included in this language.

- Since we know that either the a comes before the b or the b comes before the a, we can define the language by the expression
  (a + b)a(a + b)b(a + b) + (a + b)b(a + b)a(a + b)

Note that the only words that are omitted by the first term (a + b)*a(a + b)*b(a + b)* are the words of the form some b's followed by some a's. They are defined by the expression bb*aa*

Example:

- We can add these specific exceptions. So, the language of all words over the alphabet Σ = {a, b} that contain at least one a and at least one b is defined by the expression: (a + b)a(a + b)b(a + b) + bb*aa*
- Thus, we have proved that
  (a + b)*a(a + b)*b(a + b)* + (a + b)*b(a + b)*a(a + b)*
  = (a + b)*a(a + b)*b(a + b)* + bb*aa*