Regular Expressions

* A regular expression (RE) describes a language.
  + Example:
    - (2 + (1 1)\*) 0
      1. + means inclusive or
      2. \* means repeat arbitrarily many times, also including no times.
      3. Brackets () are used for grouping.
      4. 20 is in the language of this expression.
      5. 110 and 11110 and 1111110 are all in the language of this expression.
      6. 1110 is NOT in the language of this expression.
         1. (11)\* means you repeat this substring many times, you cannot talk half of the substring.
      7. 0 is also in the language of this expression.
      8. We are using three operations:
         1. + (or)
         2. \* (repeat)
         3. Concatenation
  + Example:
    - (0+1)(0+1)
      1. This is the concatenation of two pieces.
      2. You can take a 0 or a 1.
      3. This means that the resulting answer would be all strings with the length of 2.
      4. 00, 01, 11, 10
  + Example:
    - (0+1)\*
      1. This generates all binary strings.
      2. Asterisk allows you to repeat the piece as many times as you choose.
  + Example:
    - 2+11\*0
      1. The asterisk will apply to the smallest piece it can since there are no brackets. Therefore it will only apply to the rightmost one.
      2. Either you get 2, or 110, or 1110, or 10, or 111110.
* A regular expression has a recursive definition.
  + If R & S are RE
    - R+S
      1. Has language as the union of languages of R and S.
         1. In order to get an empty string, both strings need to have an empty string.
    - RS
      1. Has language as the concatenation of the languages of R and S.
         1. Any string that can be built by concatenating a string of the first language and a string of the second language.
    - R\*
      1. Has language which is the star of the language of R.
         1. Any string that can be built by concatenating the same number of strings from R.
         2. You can also choose to take NO strings if you so choose.
      2. The star applies to the smallest piece of it.
      3. The R applies to the largest piece of it.
  + Example:
    - R has language of {bob, pat}
    - S has language {ti, ger}
    - R+S -> {bob, pat, ti, ger}
    - RS -> {bobti, bobger, patti, patger, ….}
    - R\*->{bob, bobbob, patpatbob, ….}
  + Example:
    - Give RE for all binary strings that
      1. Start with 11
         1. 11(0+1)\*
      2. End with 11
         1. (0+1)\*11 or Σ\*11
      3. Alternate with 0 & 1
         1. (01)\* + (01)\*0 + (10)\* + (10)\*1

This is every possible string in our language.

* + - 1. Have even number of 0’s
         1. (01\*0+1)\*

| 0110 | 1 | 1 | 0110 | 1

* + - * 1. (1\*01\*01\*)\*

01101 | 101101

* + Example (From in-class practice 3):
    - For the alphabet {0, 1}, give REs for each language:
      1. All Strings Containing exactly two 0’s
         1. 1\*01\*01\*

REMEMBER the STARS only apply to the 1’s

* + - 1. All strings containing at least two 0’s
         1. (0+1)\*0(0+1)\*0(0+1)\*

Same as Σ\*0Σ\*0Σ\*

* + - 1. All strings containing 00 as substring
         1. (0+1)\*00(0+1)\*
      2. All strings NOT containing 00 as substring

Can break such a string before each instance of 1.

Each resultant piece is either 1 or 10, apart from the first piece which might be 0.

* + - * 1. (0+ε)(1+10)\*
* Kleene’s Theorem
  + There is a Finite Automata (FA) for a language
    - Regular Language
  + There is a Regular Expression (RE) for a language.
    - Regular Language
  + The proof is an algorithm that converts an FA into an RE and vice versa.