**The Compiler – Formal Languages**

* High-level language → compiler → assembly language
* Assembly language
  + Simple structure
  + Easy to parse
  + Straight-forward, unambiguous translation to machine language
* High-level language
  + Complex structure
  + Harder to recognize/parse
  + Usually no single translation to machine language
  + A formal theory of string recognition is needed to handle the complexity
* **Alphabet** – a finite set of symbols, e.g. ∑ = {a, b, c}
* **String/word** – a finite sequence of symbols (from ∑), e.g. a, ab, acb etc.
  + Length of word – # of chars in the word, e.g. |aba| = 3
  + ε = empty word – an empty sequence of symbols; i.e.|ε| = 0
* **Language** – set of words
  + E.g. {a2nb, n ≥ 0} = words made up of an even # of a’s followed by 1 b
  + {} or ∅ = empty language – contains no words
  + {ε} = singleton language – only contains ε
* How to recognize if a given string belongs to a given language?
  + Depends on how complex the language is – may be impossible
  + We can characterize languages based on the difficulty of their recognition process
* Types of languages
  + **Finite (easy)**
    - Has finitely many words
    - Can recognize a word by comparing with each word in the language set
    - E.g. L = {cat, car, cow}

If first char = c, move on, else error

If next char = a

If next char = t

If no more chars, accept, else error

Else if next char = r

… etc.

* + - As a state machine (deterministic finite automata – DFA):
      * Start
      * → seen c
      * → seen ca, → seen co
      * → seen cat^, → seen car^, → seen cow^
    - ^ means accept if the program stops here
  + **Regular**
    - Built from the union, concatenation, and repetition of finite languages
    - Union – L1 ∪ L2 = {x | x ∈ L1 or x ∈ L2}
    - Concatenation – L1 ⋅ L2 = {xy | x ∈ L1, y ∈ L2}
    - Repetition – L\* = {ε} ∪ {xy | x ∈ L\*, y ∈ L}
      * = {ε} ∪ L ∪ LL ∪ LLL …
      * = 0 or more occurrences of words in L
    - E.g. {a2nb, n ≥ 0} = ({aa})\* ⋅ {b}
    - Regular expression → set-theoretic notation
      * ∅ = {}
      * ε = {ε}
      * aaa = {aaa}
      * E1 | E2 = E1 ∪ E2
      * E1E2 = E1 ⋅ E2
      * E\* = E\*
    - Is C regular?
      * A C program is a sequence of tokens, each of which is derived from a regular language
      * C ⊆ {valid C tokens}\*
      * Therefore maybe
    - How to recognize membership in a regular language?
    - E.g. {a2nb, n ≥ 0}
    - As a DFA:
      * Start
      * → seen 2n a’s ←→ seen 2n+1 a’s (loop)
      * → seen b^
  + **Context-free**
  + **Context-sensitive**
  + **Recursive (hard)**
  + Etc. (impossible)