Grammer-1

COM S 319

Objectives

1. Learn formal and informal definitions of grammer.
2. Learn about types of grammer (Chomsky Hier..)
3. Learn about regular grammer
4. Learn about recognizer for regular grammer
5. Learn about LEX

Grammer is just rules to form "strings" in a language.

Examples:

- Java language has following acceptable string:

  int i = 10;

- Regular expression a*bc* has the following acceptable strings:

  b, bc, abc, ab, bccc etc

Grammer related terms -1

- Symbol: A symbol here is the smallest distinguishable element in a written language.
  - Example: a is a symbol for english language
  - Example: Θ is a symbol for greek language
  - Example: ♩is a symbol for written music.
  - Also called TERMINALS

- Alphabet: An alphabet is a FINITE set of symbols.
  - Note that it has to be FINITE set.
  - Example: { '0', '1'} is an alphabet. It just consists of two symbols.

Grammer related terms -2

- Non-Terminals: Non-terminals are variables which represent parts of a language.
  - Example: SENTENCE → NOUN_PHRASE VERB_PHRASE

- Production Rules: Production rules relate non-terminals recursively in terms of each other and terminals. They have a Left Hand Side and a Right Hand Side separated by the symbol →
  - Example: SENTENCE → NOUN_PHRASE VERB_PHRASE

Grammer related terms -3

- Language: A language is a set of strings of symbols from some alphabet.

- Example language is { a, ab, b }
- This one is finite!

- Another example is the infinite set {a, aa, aaa, ... }

Grammer

- Grammer is formally defined as follows. A grammer G is a four tuple { V, T, P, S} where V and T are finite sets of variables and terminals (or symbols). V and T are disjoint. P is a finite set of production rules. S is a special variable called the start symbol.
- Example:

  G1 = {V, T, P, S} where V = {E}, T = {+, -, (,), id},

  S = E, P = rules below

  (rule1) E → E + E, (rule2) E → E - E,

  (rule3) E → (E), (rule4) E → id

Showing that "i + (i + j * i + (i + j))" is in the grammer

- by rule 1: E + E
- by rule 4: i + E
- by rule 3: i + (E)
- by rule 1: i + (E + E)
- by rule 4: i + (i + E)
- … (after many similar steps)
- finally we will get the string "i + (i + j * i + (i + j))"

Grammer Example1

- V = {S, A, B, C}, T = {a, b, c}
- S→ A
- A → aA
- A → B
- B → bC
- C → cC
- C → ε

Q: What are some example strings in the language?

Grammer Example2

- V = {S, A, B, C}, T = {a, b, c}
- S→ aAc
- A → aAc
- A → b

Q: What are some example strings in the language?
RECAP SO FAR…

1. grammer (a 4-tuple)
2. language (sets of strings)
3. terminals (symbols)
4. non-terminals (variables)
5. production rules
6. start symbol (a special variable)

TYPES OF GRAMMERS

We know regular expressions already.

The regular expression a[bc]d will accept the language {abd, acd}.

We can express it as a grammer where

V = {start, next}

T = {a, b, c, d}

P is {

rule1: start -> a next,

rule 2: next -> b end | c end,

rule 3: end -> d

}

S is start

Consider the strings

aabb, aaabbb, aaaabbbb etc

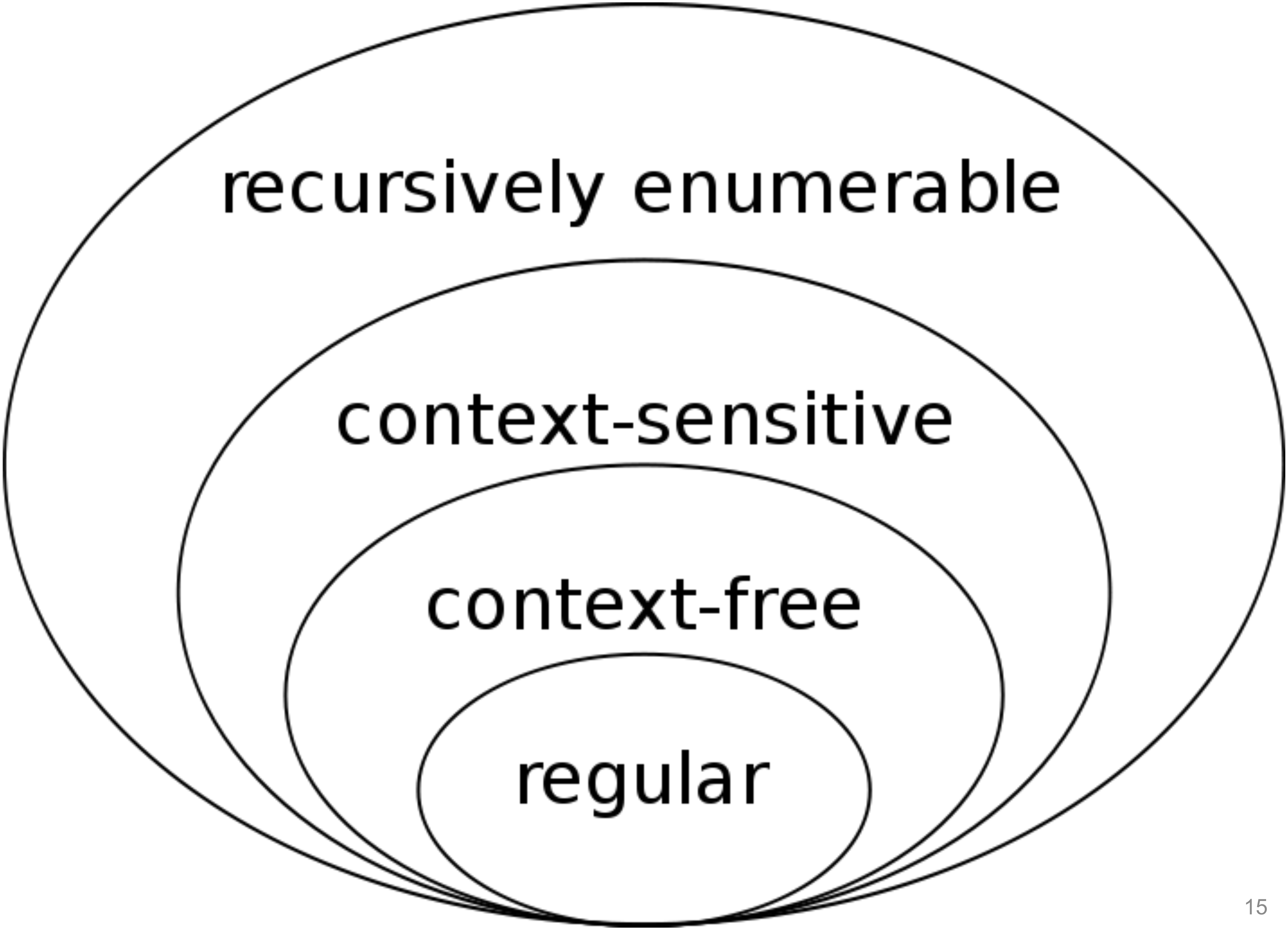
Q: Can you write a regular expression to express it?

The answer is NO.

So clearly, there are at least TWO types of languages (or sets of strings). We can write regular expressions for one type and CANNOT write regular expressions for the other type.

Q: Are there other types of languages as well?

Q: How do the grammer rules for these types of languages differ?


recursively enumerable
context-sensitive
context-free
regular

Chomsky hierarchy

- type-3 or regular grammer (regular expressions)
  - can express a^n
  - accepted by finite automaton (limited memory needs)
- type-2 or context-free grammer (progmmg langs)
  - can express $a^n b^n$ (matching parenthesis, expressions)
  - accepted by pushdown automaton (uses stack)
- type-1 or context-sensitive grammer c
  - an express $a^n b^n c^n$
  - accepted by Linear bounded Turing machines
- type-0 grammers (accepted by Turing Machines)