- Defining Languages Recursively - previous examples

- Consider the Arithmetic Expressions (denoted by AE)

- Alphabet $= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, *, -, \}$

- Valid and Invalid expressions

- Rules to determine valid arithmetic expressions might look like the following

  1. If $x \in \{0, \ldots, 9\}$ then $x \in AE$

  2. If $A, B \in AE$ then $A + B$, $A - B$, $A * B$, AB are $\in AE$

- How to determine if a given string is valid or not

  1. Construct a parse tree

- Rules like above describe the structure of arithmetic expressions.

- Their meaning (value in this case) might be ambiguous.

- The rules are like a grammar for a language

- Grammar for (a subset of) the English Language consists of:

  1. Grammatical categories: e.g.noun phrase, verb phrase, article, noun, verb etc...

  2. words (elements of alphabet)

  3. rules for describing the order in which the elements of the grammatical categories must appear in a sentence.

- Grammar for a fragment of English

  1. Grammatical categories: SEN, NP, VP, A, N, V

  2. Words: the, cat, mouse, caught

  3. Rules:
     SEN $\rightarrow$ NP VP
     NP $\rightarrow$ N
     NP $\rightarrow$ A N
     VP $\rightarrow$ V
     VP $\rightarrow$ V NP
     V $\rightarrow$ caught
     A $\rightarrow$ the
     N $\rightarrow$ cat | mouse

- Backus-Naur Format (BNF) format:
  <SEN> ::= <NP> <VP>
  <NP> ::= <N>
  | <NP> <A> <N>
  <VP> ::= <V>
  | <V> <NP>
  <V> ::= "caught"
  <A> ::= "the"
  <N> ::= "cat" | "mouse"

- Grammar described Syntax - structure not Semantics (meaning)

- Grammars describe the syntax of programming languages

- Rules for recognition and generation of programs from programming languages

- Alphabets and Languages

- A language is defined by its words and sentence structure

- The collection of words is the alphabet(in computing terms)

- token - smallest component for description of programming language

- alphabet for programming language: set of lexical tokens
  e.g. keywords, identifiers, constants, other symbols

```
void main()
{
  cout<<''Hello World'';
}
```

- What are the tokens in the example above?

- Lexical tokens form a language: alphabet of the language

- In the English language: tokens are the words

- A program in programming language is equivalent to a sentence in the English language

- can break tokens down further into regular expressions...(the letters a-z in English)

- Context-Free Grammar(CFG): describes rules for the ordering of symbols within a program

- cannot specify context sensitive aspects, e.g.

  1. variable must be declared before being referenced

  2. order and number of actual parameters in a procedure call must match the order and number of formal arguments in procedure declaration

  3. compatibility of types in assignment statement

- Terminology for Context-Free Grammar(CFG): Non-terminal, Terminal, Productions, Start symbol

- 4-Tuple: $G = (V_N, V_T, P, S)$

- V_N:
  - finite set of non-terminal symbols
  - correspond to: SEN, NP, VP, A, N, V

- V_T
  - Finite set of symbols
  - correspond to: 'the', 'cat', 'mouse', 'caught'

- $V = V_N \cup V_T$

- $V_N \cap V_T = \{\}$

- P: Set of Relations of the form: $C \rightarrow x$ where $C \in V_N$ and $x \in (V_N \cup V_T)^*$

- S: $\in V_N$, Start state (SEN)

- Generate sentences of a language

- Consider the string $x = aBc$ and a production of the form $B \rightarrow d$

- Replace B with d to obtain $y = adc$

- x derives y. $x \Rightarrow y$

- $w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$

- w_1 derives w_n; $w_1 \Rightarrow^* w_n$

- Generation of a language from a grammar

- Let G be the grammar

- A language L is context free iff there is a CFG G such that each element of l can be generated by applying the rules of the grammar

- Example sentence: the cat caught the mouse

- SEN $\Rightarrow$ NP VP
  $\Rightarrow$ A N NP
  $\Rightarrow$ the N VP
  $\Rightarrow$ the cat VP
  $\Rightarrow$ the cat V NP
  $\Rightarrow$ the cat caught NP
  $\Rightarrow$ the cat caught A N
  $\Rightarrow$ the cat caught the N
  $\Rightarrow$ the cat caught the mouse

- Derivation tree/ Parse tree

- Consider the language of Propositional Logic with only propositional symbols p, q, r

- the alphabet for the language is $V_T = \{p, q, r, (,), \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow\}$

- $V_N = \{W\}$

- Productions

  1. $W \rightarrow p$

  2. $W \rightarrow q$

  3. $W \rightarrow r$

  4. $W \rightarrow (\neg W)$

  5. $W \rightarrow (W \wedge W)$

  6. $W \rightarrow (W \vee W)$

  7. $W \rightarrow (W \Rightarrow W)$

  8. $W \rightarrow (W \Leftrightarrow W)$

- Each production corresponds to a generator(rule) of an inductively defined set.

- Generating a Well-formed formula from the grammar of the Propositional Logic:

- for example: $(((\neg p) \wedge q) \Rightarrow r)$

- $W \rightarrow (W \Rightarrow W)$
  $\rightarrow (W \Rightarrow r)$
  $\rightarrow ((W \wedge W) \Rightarrow r$
  $\rightarrow ((W \wedge q) \Rightarrow r$
  $\rightarrow ((\neg W \wedge q) \Rightarrow r$
  $\rightarrow ((\neg p \wedge q) \Rightarrow r$

- A parse tree: a graphical representation of the separation of a sentence into its components.

- Let $G = (V_N, V_T, P, S)$ be a CFG. A Parse Tree has the following properties:

  1. The root corresponds to the Start Symbol.

  2. Every interior vertex corresponds to a Non-Terminal symbol.

  3. If a vertex has a label $A \in V_N$ and its children are labelled (from left to right) $a_1, \ldots, a_n$ then P must contain a production of the form $A \rightarrow a_1 \ldots a_n$

  4. Every Leaf has a label from V_T.

- Replace the leftmost or rightmost non-terminal first?

- Can you construct a different parse tree for these sentences?

- Consider the CFG: $V_N = \{Exp\}$ $S = Exp$
  $V_T = \{id, +, *\}$
  Productions:
  $Exp \rightarrow Exp + Exp$
  $Exp \rightarrow Exp * Exp$
  $Exp \rightarrow id$

- Construct a Parse Tree for $id + id * id$?

- Can you parse this string in a number of different ways?

- A CFG is ambiguous if there exists some sentence in $L(G)$ which has two distinct parse trees.

- An unambiguous grammar for the Expressions
  $Exp \rightarrow term$
  $Exp \rightarrow Exp + term$
  $term \rightarrow term * term$
  $term \rightarrow id$

- What are the non-terminals and start symbol?

- What are the terminal symbols?