BNF Grammars

Syntax?

- A language is defined by a syntax and a semantics.
- Syntax: Definition of what constitutes a grammatically valid program in that language.
- Semantics: Meaning of the program.
- Reference manuals: Describe the syntax and semantics of languages formally.
- Formal definition: Precise description of the syntax and semantics of a language. It is aimed at specialists.

  Formal - based on mathematics.
- How is the syntax described?

  – Using a language – **metalanguage**

  Example: Backus-Naur form (BNF)

Expressions

- Starting point for a language.
- Example: $(-b + sqrt(b * b - 4.0 * a * c))/(2.0 * a)$
- Different notations for an expression.
- **Prefix notation** (Example: + 1 2)
- **Postfix notation** (Example: 1 2 +)
- **Infix notation** (Example: 1 + 2)
- **Mixfix notation**: Operations that do not fit into the prefix, infix and postfix classifications. (Example: if-then-else)
- **Examples:**
  - Prefix:

  $*$ + 20 30 60 =

  $*$ 20 + 30 60 =

  - Postfix:

  20 30 + 60 $*$ =

  20 30 60 + $*$ =

  How to evaluate a prefix or postfix expression?

Infix Notation

- How do you compute $5 * 7 + 3 - 1$?
- Is it $(5 * 7) + (3 - 1)$?
- Is it $5 * (7 + 3) - 1$?
- To deal with the ambiguity: Use parentheses or use a precedence on operators.

  Example: $\{*, /\} > \{+, -\}$.

  $*$ and $/$ have the same precedence. $+$ and $-$ have the same precedence.

  The expression is: $(5 * 7) + 3 - 1$.

- An operator is said to be **left-associative** if subexpressions containing the same operator or operator with the same precedence are grouped from left to right to be decoded.

  Examples:

  $- -$ is left-associative. $4 - 2 - 1$ is $(4 - 2) - 1$ and not $4 - (2 - 1)$.

  $- 5 * 7 + 3 - 1$ is $((5 * 7) + 3) - 1$.

  $- +$, $*$ and $/$ are also left-associative.

- An operator is said to be **right-associative** if subexpressions containing the same operator or operator with the same precedence are grouped from right to left to be decoded.

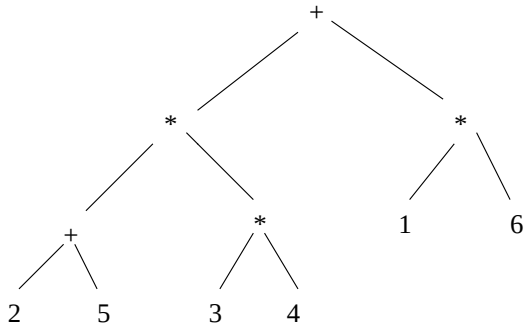
  Examples:

  – Exponentiation is right-associative. 2^{3^4} is $2^{(3^4)}$.

  – The assignment symbol is right-associative.

Expression Trees

Rooted trees can be used to represent arithmetic formulas.



The root of the tree represents the final value of the computation.

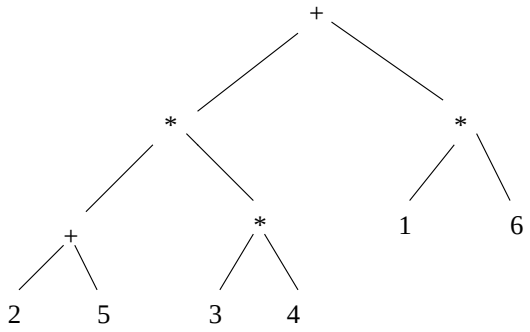
The leave represent the operands, and the intermediate nodes the operators.

This tree represents the equation $(2+5)*(3*4)+(1*6)$ using conventional arithmetic notation, or $\{2\ 5+3\ 4*\ *1\ 6*+\}$ using reverse Polish notation (as on an HP calculator) (postfix notation).

These formulas can be constructed by appropriately **traversing** the expression tree, i.e. visiting the nodes in the correct order.

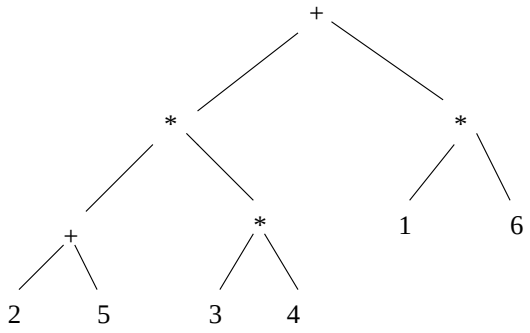
There are three natural orders to visit the nodes of the tree, each of which walks up and down the tree in a **recursive** manner:

- **In-order** – visit all the left subtree before visiting the root node, then visit the right subtree.



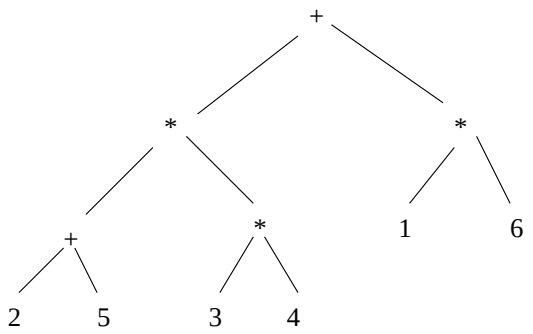
$$Infix\ notation:\ 2+5*3*4+1*6$$

- **Post-order** – visit the left subtree and right subtrees completely before visiting the root node.



$$Postfix\ notation:\ 25+34**16*+$$

- **Pre-order** – visit the root node before visiting the left subtree and right subtrees.



$Prefix\ notation:\quad + * + 25 * 34 * 16$

- A fourth natural order, **breadth-first** traversal, traverses level-by-level, $\{+\ *\ *\ +\ *\ 1\ 6\ 2\ 5\ 3\ 4\}$.

  This involves jumping around from one subtree to another, and hence is not good for evaluating expressions, although breadth-first traversal does have numerous applications in computer science.

if-then-else Tree

- How to represent $if\ a > b\ then\ a\ else\ b$ as a tree?

BNF

- The most common notation for grammars is the **BNF** (Backus-Naur form) notation.

- A BNF grammar is a set of **rewriting rules** defined on a set of **nonterminal symbols**, a set of **terminal symbols** and a **"start symbol"** .

- **Terminals** form the basic alphabet from which programs are constructed.

  Example: int, $Object$, $<$ are terminals in the BNF description of JAVA.

  Nonterminals identify grammatical categories.

  The **start symbol** identifies the principal grammatical category being defined by the grammar.

  Example: program

- The rewriting rules are written using the following meta-symbols: $\rightarrow$ and $|$.

  – Meta-symbols are symbols of the meta-language.

  – $|$ is a separator for alternative definitions (it means 'or').

  – $\rightarrow$ is used to define a rewriting rule (it means 'is')

- A **rewriting rule** is of the form:

$$left\ hand\ side\ \rightarrow\ \text{“}definition\text{”}$$

  - The left-hand-side is the name of a grammatical category. It is a nonterminal.
  - The right-hand-side is a definition that specifies the grammatical structure of the symbol appearing on the left-hand-side of the rule.
  - The right-hand-side of a rule can be any sequence of **terminals** and **nonterminals** separated by $|$.

- Designing a grammar is not easy.

  – It can lead to problems such as **ambiguities** in grammars.

Examples of Grammars

- Grammar for a digit:
  $digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
  $digit$ is a nonterminal.
  $0, ..., 9$ are terminals.

- Grammar for signed integers:
  $integer \rightarrow digit \mid digit\ integer$
  $signedinteger \rightarrow sign\ integer$
  $sign \rightarrow + \mid - \mid \sim$
  $\sim$ is the empty sign.

- Grammar for assignments:
  $assignmentstatement \rightarrow variable = expression$
  $expression \rightarrow variable \mid variable + variable$
  $variable \rightarrow x \mid y \mid z$

- Grammar for a subset of the English language!
  $sentence \rightarrow noun\ verb$
  $noun \rightarrow bees \mid dogs$
  $verb \rightarrow buzz \mid bite$

Derivation

- Given a BNF grammar called G and a grammatical category called C.

- A derivation w.r.t G is a sequence:

  $C \Rightarrow C_1 \Rightarrow C_2 \Rightarrow ...$

  where each instance of $\Rightarrow$ denotes the application of a *single* rule of G.

  – One wants a derivation to **terminate** and the **last right hand** of a derivation to be composed of **terminals only**.

- **Example:** Consider the following BNF grammar:

  Integer $\rightarrow$ Digit | Integer Digit

  Digit $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

  Integer $\Rightarrow$ Integer Digit $\Rightarrow$ Digit Digit $\Rightarrow$ 3 Digit $\Rightarrow$ 32 is a derivation.

- Each string on the right of a $\Rightarrow$ is called a **sentential form**.

  Example: Integer Digit, Digit Digit, 3 Digit and 32 are sentential forms.

- A **left-most derivation** is a derivation in which the left-most nonterminal in the sentential form is replaced at each step.

  Example: Integer $\Rightarrow$ **Integer** Digit $\Rightarrow$ **Digit** Digit $\Rightarrow$ 3 **Digit** $\Rightarrow$ 32 is a left-most derivation.

- A **right-most derivation** is a derivation in which the right-most nonterminal in the sentential form is replaced at each step.

  Example: Integer $\Rightarrow$ Integer **Digit** $\Rightarrow$ **Integer** 2 $\Rightarrow$ **Digit** 2 $\Rightarrow$ 32 is a right-most derivation.

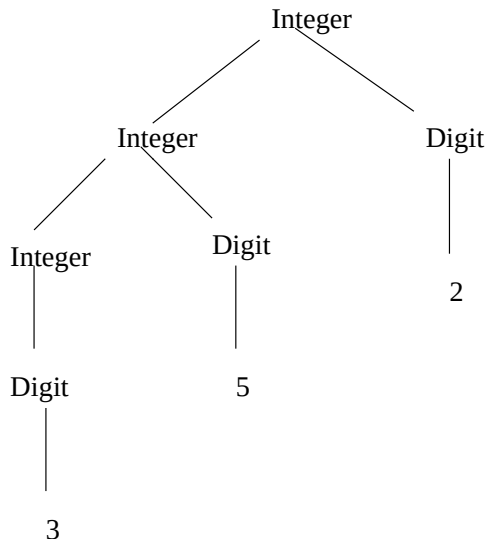
Parse Tree

- A **parse tree** is a graphical representation of a derivation. s – The root node of a parse tree is the particular grammatical category of interest (here C).

  – The internal nodes of a parse tree are grammatical categories (left hand sides of rules of G).

  – The leaves of a parse tree are terminals.

- The following tree is a parse tree:



Language

- Given a BNF grammar called G with start symbol called S.
- **Parsing** a string s to check if s is an instance of the grammatical category called C from G can be done:
  - Using a **derivation** (Is there a derivation $C \Rightarrow ... \Rightarrow s$?)
  - Using a **Parse tree** (Is there a parse tree with root C, such that reading the leaves from left to right reconstructs the string s?)
- The **Language** defined by a BNF grammar is that set of *all* strings that can be parsed or derived using the rules of the grammar (starting from S).
- **Property:** If s is a string of the language L described by G, there is a derivation:

  $S \Rightarrow ... \Rightarrow s$

  and a parse tree with root S and reading the leaves from left to right reconstructs s.

  The number of internal node of the parse tree is equal to the number steps needed to derived s from S.

Example

- BNF grammar:

  Integer $\rightarrow$ Digit | Integer Digit

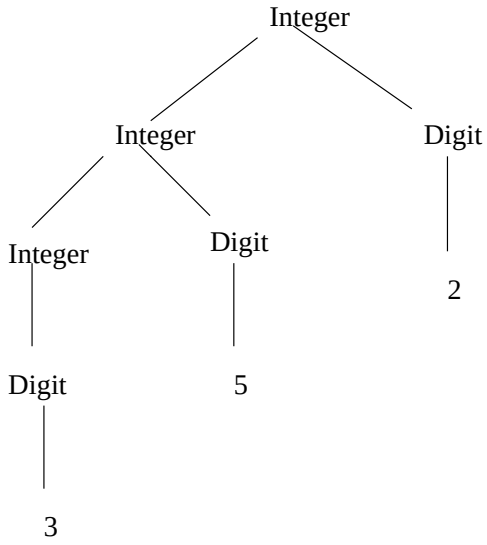
  Digit $\rightarrow$ 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

- Justify that 352 is an $Integer$.

  – First method (left-most derivation): Integer $\Rightarrow$ **Integer** Digit $\Rightarrow$ **Integer** Digit Digit $\Rightarrow$ **Digit** Digit Digit $\Rightarrow$ 3 **Digit** Digit $\Rightarrow$ 3 5 **Digit** $\Rightarrow$ 352

  – Second method (right-most derivation): Integer $\Rightarrow$ Integer **Digit** $\Rightarrow$ **Integer** 2 $\Rightarrow$ Integer **Digit** 2 $\Rightarrow$ **Integer** 5 2 $\Rightarrow$ **Digit** 5 2 $\Rightarrow$ 352

  – Third method (Parse tree):



Ambiguity

- A grammar is said **ambiguous** if it permits a string in its language to be parsed into two or more parse trees.

  – Ambiguous grammars should be avoided.

- **Example 1:** The following grammar is ambiguous.

  Expression $\rightarrow$ 0 | 1 | Expression $-$ Expression

  because $1-0-1$ can be parsed into 2 parse trees.

- **Example 2: Dangling-else ambiguity**

  The following grammar is ambiguous.

  Statement $\rightarrow$ if Expression then Statement | if Expression then Statement else Statement

  Expression $\rightarrow$ ...

  if E1 then if E2 then S1 else S2 can be parsed in 2 parse trees.

- **Example 3:** The following grammar is ambiguous.

  Assignment $\rightarrow$ Variable $=$ Expression

  Expression $\rightarrow$ variable | Expression $+$ Expression

  Variable $\rightarrow$ x | y | z

  $x = x + y + z$ can be parsed in 2 parse trees.

Solving Ambiguity

- To solve the ambiguity there are different solutions:

  – Use an explicit and formal specification outside the BNF grammar considering the properties of some symbols of the grammar.

  Example: left-associativity, right-associativity, precedence (priority) on symbols ...

  – Use an explicit and non formal specification outside the BNF syntax.

  Example: The language designer can stipulate the extra-grammatical rule that every $else$ clause will be associated with the textually closest preceding if statement. If a different attachment is desired, the programmer can always make it clear (by inserting braces for example).

  – Redesign the BNF grammar.

- **Example 1:**

  Expression $\rightarrow$ 0 | 1 | Expression $-$ Expression

  To solve ambiguity we use the fact that $-$ is left-associative.

- **Example 2: Dangling Else**

  How is it solve in programming languages?

  – C and C++ stipulate the extra-grammatical rule that every $else$ clause will be associated with the textually closest preceding if statement.

  – JAVA expands the BNF grammar with the following rules:

  IfThenStatement $\rightarrow$ if (Expression) Statement

  IfThenElseStatement $\rightarrow$ if (Expression) StatementNoShortIf else Statement

  – Assume a language JAY that has the following BNF grammar for conditionals:

  Statement $\rightarrow$; | Block | Assignment | IfStatement | WhileStatement

  Block $\rightarrow$ {Statement}

  IfStatement $\rightarrow$ if (Expression) Statement | if (Expression) Statement else Statement

- **Example 3:**

  Assignment $\rightarrow$ Variable $=$ Expression

  Expression $\rightarrow$ Variable | Expression $+$ Expression

  Variable $\rightarrow$ x | y | z

  To solve ambiguity we use the fact that $+$ is left-associative.

Regular Expressions

- An alternative to BNF for specifying a language is the use of **regular expressions**.

- Conventions for Writing Regular Expressions:

Regular Expression	Meaning
`x`	A character (stands for itself)
`"xyz"`	A literal string (stands for itself)
`M | N`	M or N
`M N`	M followed by N (concatenation)
`M*`	Zero or more occurrences of M
`M+`	One or more occurrences of M
`M?`	Zero or one occurrence of M
`[a-zA-Z]`	Any alphabetic character
`[0-9]`	Any digit
`.`	Any single character

Examples

- "true" | "false" is a regular expression to describe boolean values.

- $[a-zA-Z][a-zA-Z0-9]^*$ is a regular expression to describe Identifiers composed of letters and digits only. An Identifier begins with a letter.

- "//"$[a-zA-Z]^*(return)$ is a regular expression to describe Comments as a series of characters introduced by // and followed by a return.

- The language containing strings of the form $a^n b^n$ cannot be generated by a regular expression. Why?

  Can it be generated by a BNF grammar?

EBNF

- Extended BNF
- EBNF was introduced to simplify the specification of **recursion** in grammar rules and to introduce the idea of an optional part in a rule's right-hand side.
- EBNF uses Regular Expressions.
- **Example:** The following BNF rules:

  Expression $\rightarrow$ Term | Expression + Term | Expression − Term

  Term $\rightarrow$ Factor | Term $*$ Factor | Term / Factor

  Factor $\rightarrow$ Identifier | Literal | (Expression)

  can be written equivalently using EBNF rules the following way:

  Expression $\rightarrow$ Term $\{$[+ | -] Term$\}^*$

  Term $\rightarrow$ Factor$\{$['*' | /]Factor$\}^*$

  Factor $\rightarrow$ Identifier | Literal | (Expression)

- EBNF definitions tend to be slightly clearer and briefer than BNF definitions.
- The star notation (*) in EBNF definitions suggests a loop rather recursion.

Example

- Consider the following EBNF grammar:
  Expression → Term {[+ | -] Term}*
  Term → Factor{['*' | /]Factor}*
  Factor → Identifier | Literal | (Expression)

- EBNF-Based Parse tree for the expression $x + 2 * y$