Translation Models

- **Grammars**
  - Terminal and non terminal symbols
  - Grammar rules
  - BNF notation
  - Derivations, Parse trees and Ambiguity
- **Grammars for programming languages**
  - Types of grammars
  - Regular expressions and Context-free grammars

Grammars

Syntax is concerned with the structure of programs. The formal description of the syntax of a language is called **grammar**

Grammars consist of **rewriting rules** and may be used for both recognition and generation of sentences (statements).

Grammars are independent of the syntactic analysis.

More about grammars

Word categories, constituents

The boy **is reading a book.**

The boy **is reading an interesting book**.

The boy **is reading a book by Mark Twain**.

Terminal and non-terminal symbols, grammar rules

Terminal - to represent the words

Non-terminal - to represent categories of words and constituents.

Starting symbol **S** represents "sentence"

These symbols are used in **grammar rules**

Example

Rule	**Meaning**
N $\rightarrow$ boy	**N** is the non-terminal symbol for "noun", "boy" is a terminal "symbol"
D $\rightarrow$ the | a | an	**D** is the non-terminal symbol for definite or indefinite articles.
NP $\rightarrow$ D N	this rule says that a noun phrase **NP** may consist of an article followed by a noun

BNF notation

Grammars for programming languages use a special notation called **BNF** (Backus-Naur form)

The non-terminal symbols are enclosed in < >
Instead of $\rightarrow$ the symbol **::=** is used
The vertical bar is used in the same way - meaning choice.
[] are used to represent optional constituents.

BNF notation is equivalent to the first notation in the examples above.

BNF Example

The rule

<assignment statement> ::=

<variable> = <arithmetic expression>

says that an assignment statement has

a variable name on its left-hand side

followed by the symbol "=",

followed by an arithmetic expression

Derivations, Parse trees and Ambiguity

Using a grammar, we can generate sentences.
The process is called **derivation**

Example :

$$S \rightarrow SS \mid (S) \mid (\ )$$

generates all sequences of paired parentheses.

Derivation example

The rules of the grammar can be written separately:

Rule1: $S \rightarrow SS$

Rule2: $S \rightarrow (S)$

Rule3: $S \rightarrow (\ )$

Derivation of (( )( ))

$S \Rightarrow (S)$	by Rule2
$\Rightarrow (SS)$	by Rule1
$\Rightarrow ((\ )S)$	by Rule3
$\Rightarrow ((\ )(\ ))$	by Rule3

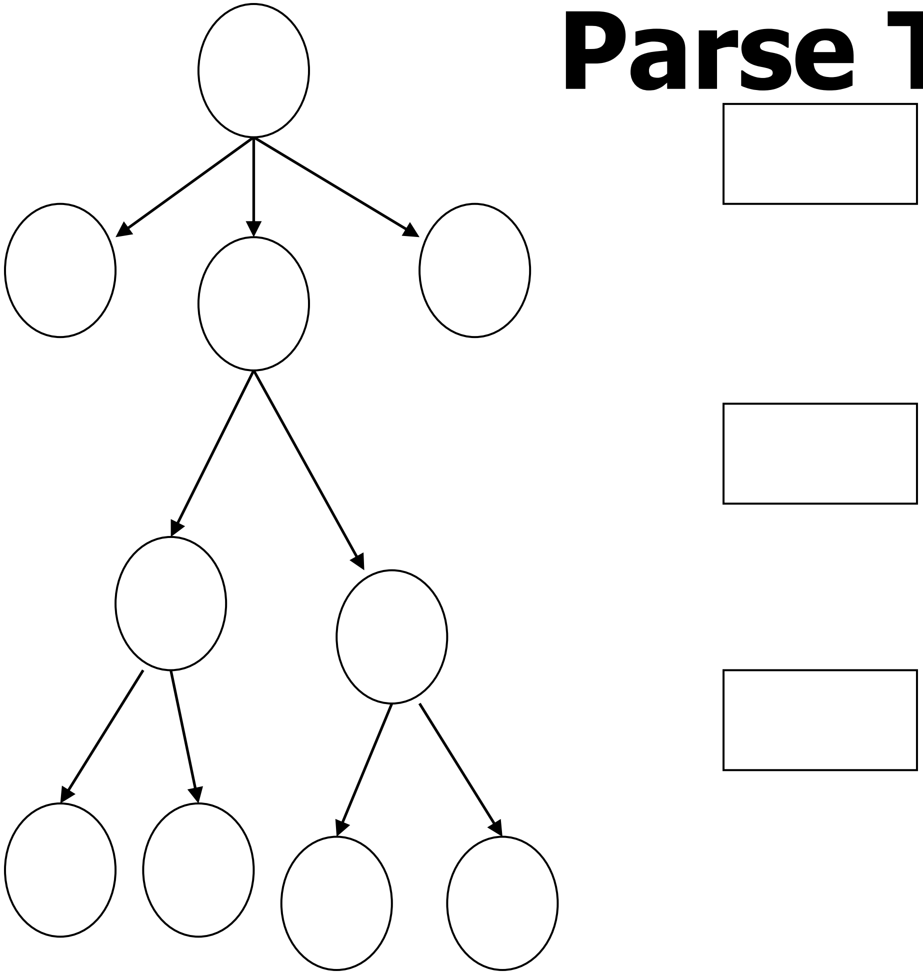
Parsing

Sentential forms - strings obtained at each derivation step. May contain both terminal and non-terminal symbols.

Sentence - the last string obtained in the derivation, contains only terminal symbols.

Parsing - determines whether a string is a correct sentence or not. It can be displayed in the form of ***a parse tree***

Parse Trees



A non-terminal symbol is expanded by applying a grammar rule that contains the symbol in its left- hand side.

Its children are the symbols in the right-hand side of the rule.

Ambiguity

Grammar rules can generate two possible parse trees for one and the same sequence of terminal symbols.

Example

Rule1: If_statement $\rightarrow$ if Exp then S else S
Rule2: If_statement $\rightarrow$ if Exp then S

if a < b then if c < y then write(yes) else write(no);

If a < b then
    if c < y then write(yes)
    else write(no);

Rule 2

If a < b then **if c < y then write(yes)**
    else write(no);

Rule 1

Grammars for programming languages

Four types of grammars

Regular grammars: (Type 3)

Rule format:

$A \rightarrow a$

$A \rightarrow aB$

Context-free grammars (Type 2)

Rule format:

$A \rightarrow$ any string consisting of terminals and non-terminals

Types of grammars (cont)

Context-sensitive grammars (Type 1)

Rule format:

String1 → String2

|String1| ≤ |String2|, terminals and non-terminals

General (unrestricted) grammars (Type 0)

Rule format:

String1 → String2, no restrictions.

Regular grammars and regular expressions

Operations on strings of symbols:

concatenation - appending two strings

Kleene star operation - any repetition of the string. e.g. a* can be a, or aa, or aaaaaa, etc

Regular expressions

Regular expressions:

a form of representing regular grammars

Regular expressions on alphabet **Σ**

string concatenations
combined with the symbols U and *,
possibly using '(' and ')'.

the empty expression: Ø

Examples

Let $\boldsymbol{\Sigma}$ **=** {0,1}.

Examples of regular expressions are**:**

0,1, 010101, any combination of 0s and 1s

	generated strings:
0 U 1	0, 1
(0 U 1)1*	0, 01, 011, 0111,…, 1, 11, 111..
(0 U 1)*01	01, 001, 0001,… 1101, 1001,
	(any strings that end in 01)

Regular languages
Context-free languages

Regular languages are languages whose sentences can be described by a regular expression.

Regular expressions are used to describe identifiers in programming languages and arithmetic expressions.

Context-free grammars generate **context-free languages.** They are used to describe programming languages.