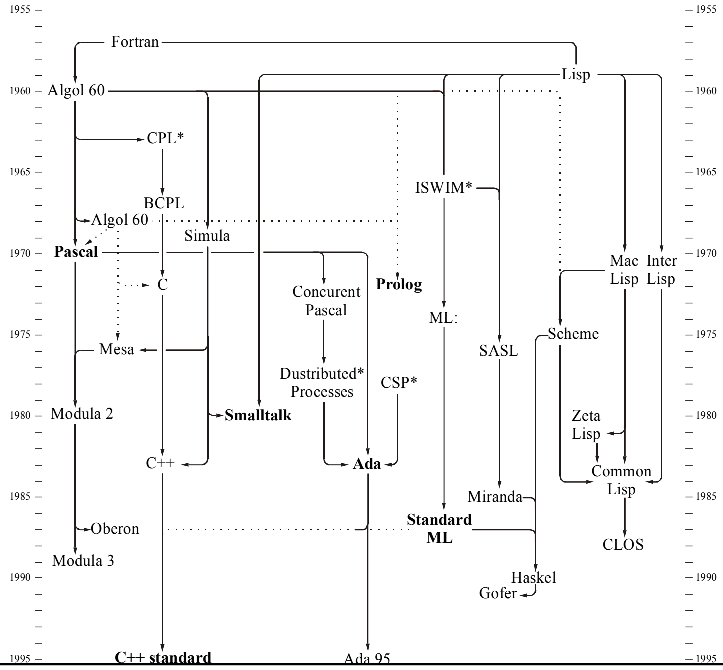
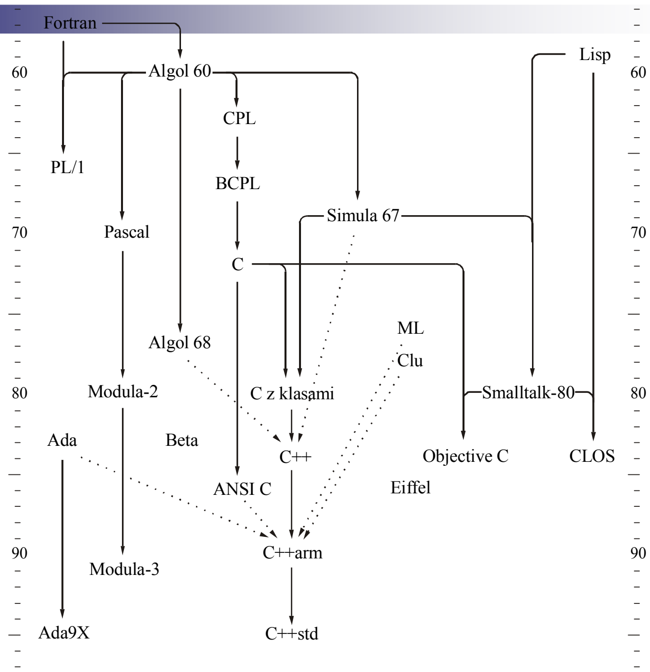
Computer Engineering Department
Technical University of Lodz

Mathematical Linguistics

Formal Languages and Grammars
Syntax Analysis

History of programming languages





Dependence between C++ and other programming languages

Language Definition

- The basis of each language is a **dictionary**. In **formal language theory,** elements of the dictionary (words) are called **symbols** (terminals).
- A sequence of words, which is built according to strict rules is called a **sentence**.
- A set of rules, which define a set of correct sentences is called a **grammar** or a **syntax**. Syntax (language structure) allows to check, if the given set of words is a sentence.
- Syntax and **semantics** (meanings) of the language are in the close relation, but in formal language theory only the purely syntactical aspects of languages are studied.

Formal Languages

- A **formal language** is a subset of finite strings of elements of the finite set which is called the alphabet.
- **A formal language** is defined by means of a **formal grammar**, and the same language can be defined by many different grammars.

EXAMPLE 1

<sentence> ::= <subject > <predicate>
<subject>::= flowers | stars
<predicate>::= bloom | shine

Sentences that can be constructed from this formal grammar:

- flowers bloom
- stars shine
- stars bloom
- flowers shine

BNF (Backus-Naur Form)

- **Start symbol**: <sentence>,
- **Non-terminal symbols** (non-terminals): <sentence>, <subject>, <predicate>;
- **Terminal symbols** (terminals): flowers, stars, bloom, shine;
- **Metasymbols** BNF notation : <, >, ::=, |
- **Production rules** – rules which allow to define a language.

EXAMPLE 1

S ::= A B
A ::= x | y
B ::= z | w

<u>Sentences generated by this grammar:</u>

xz, yz, xw, yw.

By applying rules of consecutive replacements, which is called a ***derivation,*** *language sentence can be generated from the start symbol****:***

$$S \to AB \to xB \to xw$$

$$S \to AB \to yB \to yz \qquad S \xrightarrow{*} yz$$

Chomsky Grammar
– mathematical definition of a language

1. A language ***L = L(T,N,P,S)*** is defined by:

   T – a dictionary of terminal symbols;

   N – a set of non-terminal symbols;

   P – a set of production rules (syntax rules);

   S – start symbol, which belongs to ***N***.

Chomsky Grammar, cont.

2. A language ***L(T,N,P,S)*** is a set of terminal symbol sequences ξ, which can be derived from ***S*** according to rule 3:

   $L = \{ \xi \mid S \xrightarrow{*} \xi \quad i \quad \xi \in T^* \}$,

   where: Greek letters denote symbol

   sequences,

   T* - a set of all symbol sequences over ***T***.

Chomsky Grammar, cont.

3. A sequence δ_n can be derived from a sequence δ_0 if and only if such sequences exist $\delta_1, \delta_2, \ldots, \delta_{n-1}$, such that each sequence δ_i can be directly derived from a sequence δ_{i-1} according to rule 4.

$$(\delta_0 \overset{*}{\Rightarrow} \delta_n) \leftrightarrow ((\delta_{i-1} \rightarrow \delta_i) \text{ for } i = 1, \ldots, n)$$

Chomsky Grammar, cont.

4. A sequence η can be directly derived from a sequence ξ if and only if such sequences exist α, β, ξ', η' and the following conditions are satisfied:
   a) $\xi = \alpha \xi' \beta$
   b) $\eta = \alpha \eta' \beta$
   c) ***P*** consists production rule $\xi' ::= \eta'$

EXAMPLE 1

A grammar with a recursion, that uses finite number of rules, allows to generate an infinite number of sentences.

```
S ::= xA
A ::= z | yA
```

An example of a set of sentences, which can be generated from the start symbol S:

xz
xyz
xyyz
xyyyz
.....

EXAMPLE 2

A grammar with a recursion defining integer numbers:

```
<integer number> ::= <digit>
                   |<integer number><digit>
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

Examples of sentences:

1	4321
21	54321
321	154321

EXAMPLE 3

A grammar with a recursion:

<sentence> ::= <subject> <predicate>
<subject> ::= James | Lucy
<predicate> ::= <verb> <noun phrase>
<verb> ::= eats | likes
<noun phrase> ::= <adjective><noun phrase>|<noun>
<noun> ::= nuts | almonds
<adjective>::= salted | crisp | roasted

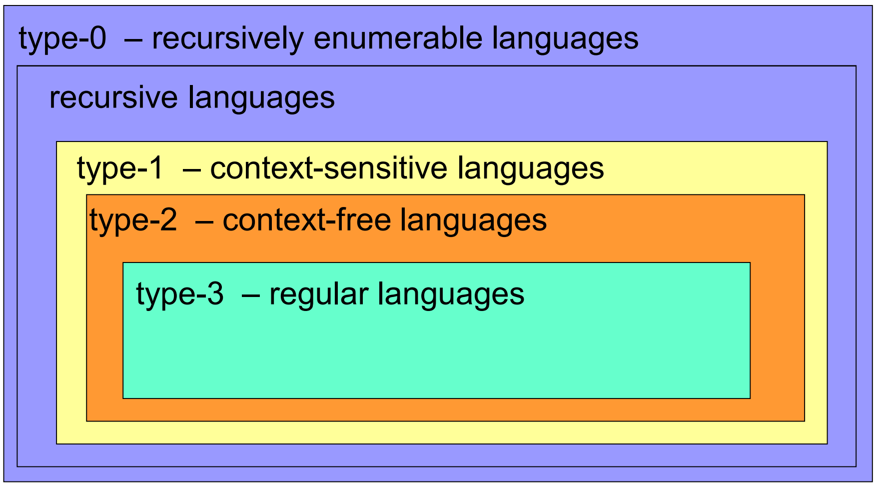
Examples of sentences:

James likes almonds
James likes roasted almonds
James likes salted roasted almonds
James likes crisp salted roasted almonds

Lucy eats nuts
Lucy eats crisp nuts

Chomsky Hierarchy

The Chomsky hierarchy consists of the following levels:



Regular Languages

Regular languages – formal languages, that are generated by regular grammars or regular expressions. A regular grammar restricts its rules to the following forms:

A ::= a

A ::= a**B**

A ::= ε

where: $A \in N, B \in N, a \in T$

Context-sensitive and Context-free Languages

Context-free language – a formal language, which is defined by the set of **context-free** rules, i.e. rules of the form:

$A ::= \xi$ and $(A \in N, \xi \in (N \cup T)^*)$

Context-sensitive language – formal language, which is defined by the rules of the form:

$\alpha A \beta ::= \alpha \xi \beta$ for non-empty ξ

$(A \in N, \alpha,\beta \in (N \cup T)^*, \xi \in (N \cup T)^+)$

Recursive and Recursively Enumerable Languages

Recursive language – a formal language, for which decisive algorithm exists, if the given string belongs to the language or not (the algorithm halts in all cases).

Recursively enumerable language – formal language, for which decisive algorithm exists, if the given string belongs to the language or not; the algorithm must halt and accept the strings belonging to the language, and for the strings do not belonging to the language, it can either halt and reject the string or do not give any answer at all (an infinite loop).

$\alpha ::= \xi$ for non-empty α ($\alpha \in (\boldsymbol{N} \cup \boldsymbol{T})^+$, $\xi \in (\boldsymbol{N} \cup \boldsymbol{T})^*$)

Syntax Analysis

- Syntax analysis deals with **derivation** of sentence structures and sentences.
- Main task of **syntax analysis** is the design of derivation algorithms for languages with complex grammatical structures.

Top-down Parsing

Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse-trees using a top-down expansion of the given formal grammar rules.

As the result of **top-down analysis,** the language sentence is generated from the start symbol.

EXAMPLE 1

<sentence> ::= < subject > <predicate>
< subject >::= flowers | stars
<predicate>::= bloom | shine

Do the sentence "stars shine" belong to the language?

<sentence>	stars shine
<subject> <predicate>	stars shine
stars < predicate >	stars shine
< predicate >	shine
shine	shine
- - -	- - -

EXAMPLE 2

S ::= xA
A ::= z | yA

Does the sentence "xyyz" belong to the language?

S	xyyz
xA	xyyz
A	yyz
yA	yyz
A	yz
yA	yz
A	z
z	z
--	--

EXAMPLE 3

S:: = A | B
A:: = xA | y
B:: = xB | z

Parsing procedure for sentence "xxxz"

S	xxxz
A	xxxz
xA	xxxz
A	xxz
xA	xxz
A	xz
xA	xz
A	z

FIRST/FIRST conflict

LL(1) Grammars

RULE 1:

For the given grammar:

$$A ::= \xi_1 \mid \xi_2 \mid \dots \mid \xi_n$$

the FIRST sets in sentences, which can be derived from ξ_i must be separated, i.e.

$$FIRST(\xi_i) \cap FIRST(\xi_j) = \varnothing \text{ for each } i \neq j.$$

LL(1) Grammars

FIRST(ξ) is a set of all the terminal symbols, which can occur at the first position in the sentences derived from ξ. This set can be determined from the following rules:

1. If the first symbol of the argument is a terminal symbol then

   $FIRST(a\xi) = \{a\}$

2. If the first symbol is a non-terminal and the production exists

   $A ::= \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$

   then

   $FIRST(A\xi) = FIRST(\alpha_1) \cup FIRST(\alpha_2) \cup \dots \cup FIRST(\alpha_n)$

EXAMPLE 3

$S ::= A \mid B$
$A ::= xA \mid y$
$B ::= xB \mid z$

⇒

$S ::= C \mid xS$
$C ::= y \mid z$

Parsing procedure for sentence "xxxz"

S	xxxz
xS	xxxz
S	xxz
xS	xxz
S	xz
xS	xz
S	z
C	z
z	z
--	--

Left-factoring

Production of a form:

$$A ::= \alpha\xi_1 \mid \alpha\xi_2 \mid \dots \mid \alpha\xi_n$$

should be rewritten as:

$$A ::= \alpha A'$$
$$A' ::= \xi_1 \mid \xi_2 \mid \dots \mid \xi_n$$

EXAMPLE 4

FIRST/FOLLOW conflict

Given a grammar:

S :: = Ax
A :: = x | ε

where: ε is an empty symbol

Parsing procedure for sentence "x"

S	x
Ax	x
xx	x
x	--

LL(1) Grammars

RULE 2:

For the each symbol $A \in N$, from which an empty symbol can be derived ($A \xrightarrow{*} \varepsilon$), a set of its FIRST symbols must be separated from the set of symbols, which can follow any sequence derived from A, i.e.

$$\text{FIRST}(A) \cap \text{FOLLOW}(A) = \varnothing$$

LL(1) Grammars

A set **FOLLOW**(A) is determined as follows:
for each production P_i of a form:

$$X :: = \xi A \eta$$

S_i means FIRST(η_i) and a set FOLLOW(A) is a sum of all sets S_i. If only an empty symbol can be derived from one η_i then a set FOLLOW(X) must be included into FOLLOW(A) also.

Recursion

Production:

$$A :: = B \mid AB$$

generates sentences: B, BB, BBB, …

According to rule 1, their usage is forbidden, because:

$$\text{FIRST}(B) \cap \text{FIRST}(AB) = \text{FIRST}(B) \neq \varnothing$$

Recursion

Production:

$$A ::= \varepsilon \mid AB$$

generates sentences: ε, B, BB, BBB, …

According to 1 rule, their usage is forbidden, because:

$$FIRST(A) = FIRST(B)$$

and therefore:

$$FIRST(A) \cap FOLLOW(A) \neq \emptyset.$$

Left recursion removal

According to 1 & 2 grammatical rules, the usage of left recursion is forbidden in LL grammars.

Problem solutions:

- exchange of left recursion into right recursion

$$A ::= \varepsilon \mid BA$$

- substitution of left recursion with an iteration.

In EBNF notation, description {B} means iteration i.e. repetition of B symbol zero, one, two, ... or infinite number of times. Production:

$$A ::= \{B\}$$

generates sentences: ε, B, BB, BBB, …

EXAMPLE 5

Given a grammar:

S :: = A | S - A
A :: = a | b | c

→ *a – b – c = ((a – b) – c)*

where: *{a, b, c, -}* ∈ T

S :: = A | A - S
A :: = a | b | c

→ *(a – (b – c))*

These two grammars are not semantically equivalent

Left recursion removal

The following production rule:

$$A ::= A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \dots \mid \beta_m$$

can be rewritten as:

$$A ::= \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_m A'$$
$$A' ::= \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$$

EXAMPLE 5

Given a grammar:

$S ::= A \mid S - A$
$A ::= a \mid b \mid c$ ⇒ $a - b - c = ((a - b) - c)$

$S ::= A \{- A\}$
$A ::= a \mid b \mid c$ ⇒ $a - b - c$

$S ::= A\ S'$
$S' ::= -AS' \mid \varepsilon$
$A ::= a \mid b \mid c$ ⇒ $a - b - c$

These grammars are semantically equivalent

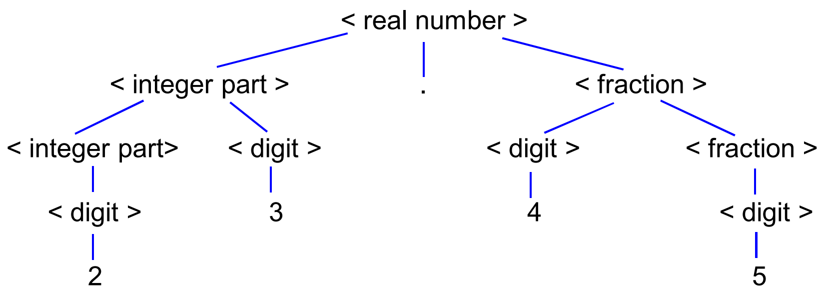
Parse tree

- A parse tree (a concrete syntax tree) is hierarchical structure, which shows graphically a sentence derivation from a formal grammar.
- A root node is labelled by the start symbol. Branch nodes are labelled by non-terminal and leaf nodes by terminal or empty symbols.
- Branch structure represents productions. Main branch is labelled by the left production side, and branches derived from it are labelled by the right production side, in the order from left to right.

EXAMPLE 6

Derive a number 23.45 from the given grammar:

<real number> :: = <integer part> . <fraction>
< integer part > :: = < digit > | < integer part > <digit>
< fraction > :: = < digit > | < digit > < fraction >
< digit > :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



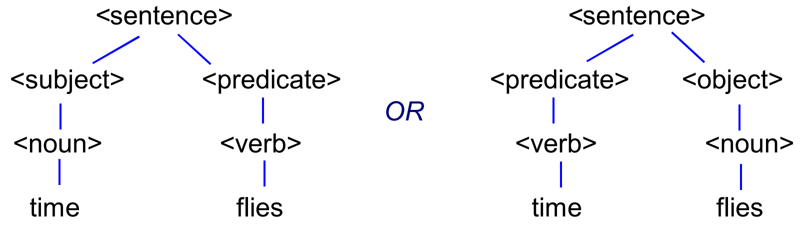
Grammar ambiguity

Problem of grammar ambiguity occurs if the grammar generates sentences, which have more than one parse tree.

EXAMPLE 1 - grammar ambiguity

<sentence> ::= <subject> <predicate>
<sentence> ::= <predicate> <object>
<subject> ::= <noun>
<predicate> ::= <verb>
<object> ::= <noun>
<verb> ::= time | flies
<noun> ::= time | flies

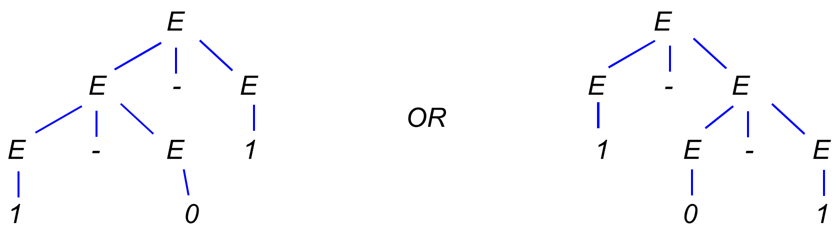
Two different parse trees can be drawn for the sentence time flies for the given grammar.



EXAMPLE 2 - grammar ambiguity

Derive the sentence 1-0-1 from the given grammar:

E:: = E – E | 0 | 1

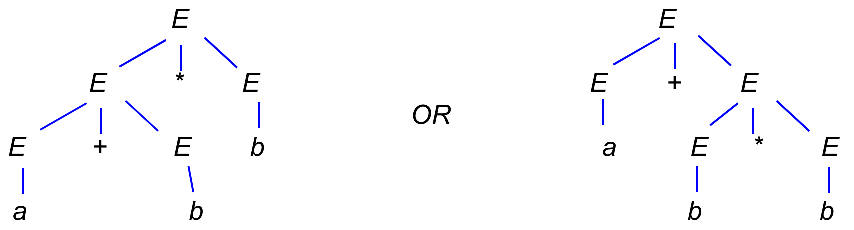


Two different parse trees can be drawn for the sentence 1-0-1.

EXAMPLE 3 - grammar ambiguity

Derive the sentence a+b*b from the given grammar:

E:: = E + E | E * E | a | b

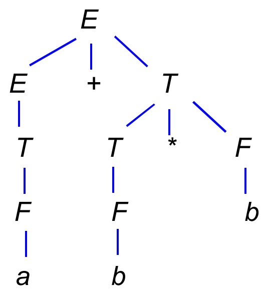


Two different parse trees can be drawn for the sentence a+b*b

EXAMPLE 3 - ambiguity removal

Derive the sentence a+b*b from the given grammar:

E:: = T |E + T | E - T
T:: = F |T * F | T / F
F:: = (E)| a | b



Only one parse trees can be drawn for the sentence a+b*b

Operators priority is correct for the given grammar!

EXAMPLE 4 - grammar ambiguity

Grammar ambiguity is present also in
C and Pascal languages for IF conditional instruction.

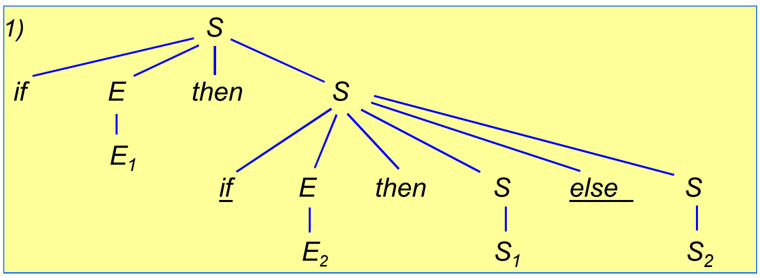
It is caused by productions:

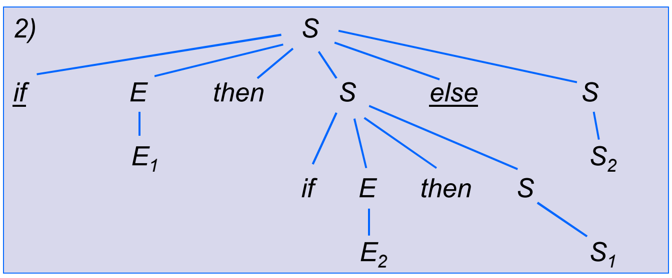
S :: = if E then S
S :: if E then S else S

Pascal

For the sentence *if E_1 then if E_2 then S_1 else S_2* two different parse trees can be derived.

if E_1 then if E_2 then S_1 else S_2





If-else ambiguity removal

If-else ambiguity problem in Pascal and C is solved in that way, that the key word else is combined with the latest key word if.

In this case the considered sentence is interpreted, as it is shown in parse tree 1.

If-else ambiguity removal in Modula-2 language

S ::= <empty>
| stmt
| if E then SL end
| if E then SL else SL end
SL ::= SL; S | S

Pascal:	*if E_1 then if E_2 then S_1 else S_2*
Modula-2:	*1º. if E_1 then if E_2 then S_1 else S_2 end end* *2º. if E_1 then if E_2 then S_1 end else S_2 end*

Pascal and Modula-2

Pascal:

if E_1 then S_1
else if E_2 then S_2
else if E_3 then S_3
else S_4

Modula-2:

if E_1 then S_1
else if E_2 then S_2
else if E_3 then S_3
else S_4 end
end
end

Modula-2

if E_1 then S_1
***elsif** E_2 then S_2*
***elsif** E_3 then S_3*
else S_4
end

S ::= if E then SL { elsif E then SL } [else SL] end
SL ::= SL; S | S

Comparison of BNF and MBNF notations

	MBNF	MEANING	BNF (EBNF)
1	=	is defined as	::=
2	|	or	|
3	.	end of formula	a terminating character is not used
4	[x]	option - zero or one repetition of string x	metasymbols [ ] are not used
5	{x}	zero or multi time repetition of string x	{x} in EBNF
6	(x | y ... | z)	any from strings: x, y,·· ,z	metasymbols ( ) are not used
7	"a"	terminal symbol (from the language alphabet)	quotation marks "..." are not used
8	small letters sequence	non-terminal symbol	Non-terminals are inside the angle brackets <...>", small and capital letters are used, hyphen is not obligatory

EXAMPLE

A grammar defining arithmetic expressions:

E::= T |E + T | E - T
T::= F |T * F | T / F
F::= (E)| a | b

BNF notation

Corrected grammar of LL(1) class:

E::= T E′
E′::= +T E′ |-T E′ |ε
T::= F T′
T′::= *F T′ |/F T′ |ε
F::= (E)| a | b

BNF notation

EXAMPLE

Corrected grammar defining arithmetic expressions of LL(1) class:

```
E = T {("+"|"–") T}.
T = F {("*"|"/") F}.
F = "(" E ")"|"a"|"b".
```

MBNF notation

In this grammar a recursion was replaced by an iteration.

Separators and terminators

Separators – separate elements
Terminators – occur after each element, i.e. sentence

Pascal:

```
S ::= <empty>
   | stmt
   | begin SL end
SL ::= SL; S | S
```

Pascal: *begin S_1; S_2; S_3 end*
begin S_1; S_2; S_3; end
begin ;S_1;;;; S_2; S_3;; end

Empty Instruction

Pascal:	*Modula-2:*
S ::= <empty>	S ::= <empty>
| stmt	| stmt
| if E then S	| if E then SL end
| if E then S else S	| if E then SL else SL end
| begin SL end	| begin SL end
| while E do S	| while E do SL end
SL ::= SL; S | S	SL ::= SL; S | S

Pascal: *if E then ; S_1;* *- semantic error*
if E then S_1; else S_2; *- syntax error*

Modula-2: *if E then S_1; S_2 end*
if E then S_1; S_2 else S_3; S_4 end

GRAMMARS WITH TRANSLATION

- A grammar with translation is a context-free grammar, in which a set of terminal symbols is extended by additional symbols called symbols of translation.
- Symbols of translation generate an extra output statement in addition to the statement generated from the grammar.

Example 1

Grammar of arithmetic expressions:

$E ::= T\ E_1$

$E_1 ::= +T\ E_1 \mid -T\ E_1 \mid \varepsilon$

$T ::= F\ T_1$

$T_1 ::= *\ F\ T_1 \mid /\ F\ T_1 \mid \varepsilon$

$F ::= -\ F \mid (E) \mid id$

$id ::= a \mid b \mid c$

Example 2

Grammar of arithmetic expressions extended with translation into RPN (Reverse Polish Notation):

$E ::= T\ E_1$

$E_1 ::= +T\ \{+\}\ E_1 \mid -T\ \{-\}\ E_1 \mid \varepsilon$

$T ::= F\ T_1$

$T_1 ::= *\ F\ \{*\}\ T_1 \mid /\ F\ \{/\}\ T_1 \mid \varepsilon$

$F ::= -\ F\ \{-\} \mid (E) \mid id\ \{id\}$

$id ::= a \mid b \mid c$