Figure 1.1

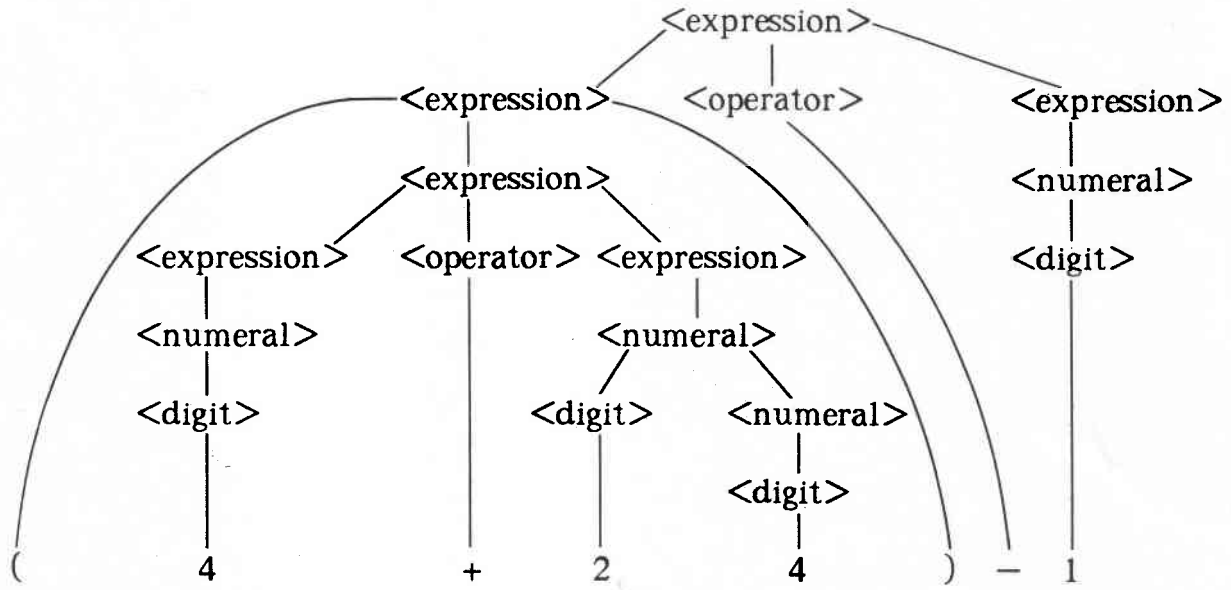


Figure 1.2

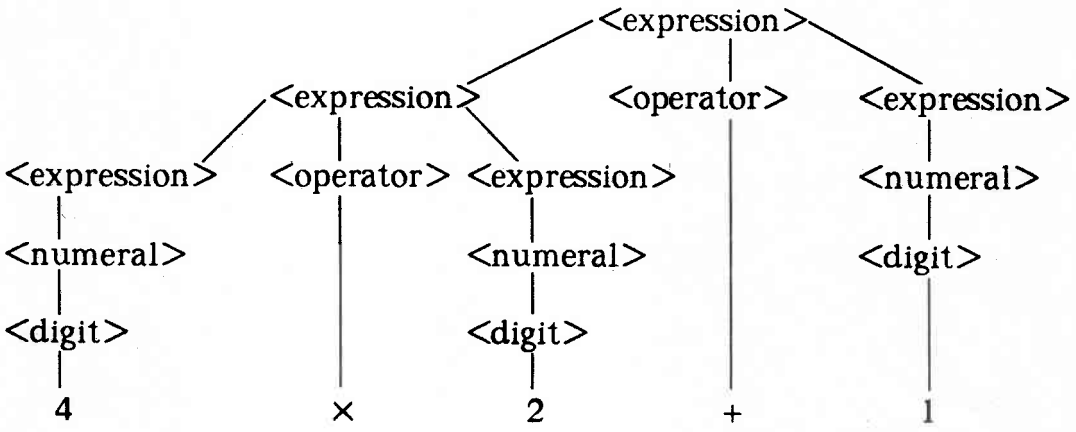
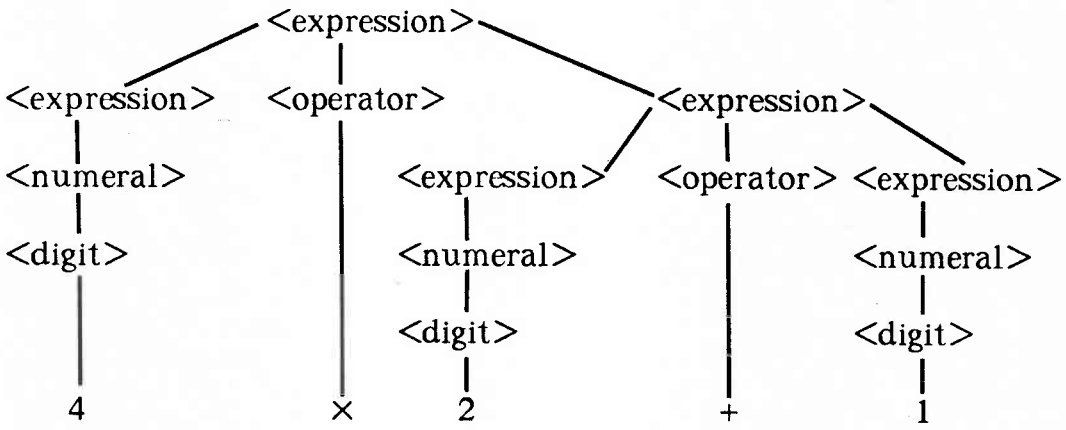


Figure 1.3



(The rules for <numeral> and <digit> remain the same.) This definition solves the ambiguity problem, and now there is only one derivation tree for $4 \times 2 + 1$, given in Figure 1.4. The tree is more complex than the one in Figure 1.2 (or 1.3) and the intuitive structure of the expression is obscured. Compiler writers

Figure 1.4

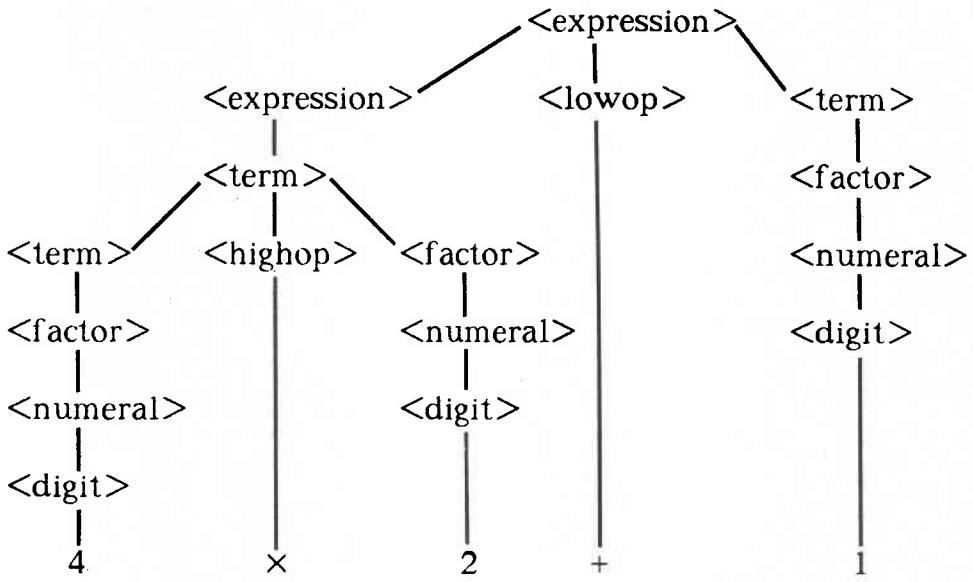


Figure 1.5

Sets:

Expression
Op
Numeral

Operations:

make-numeral-into-expression: $Numeral \rightarrow Expression$
make-compound-expression: $Expression \times Op \times Expression \rightarrow Expression$
make-bracketed-expression: $Expression \rightarrow Expression$

plus: *Op*
minus: *Op*
mult: *Op*
div: *Op*

zero: *Numeral*
one: *Numeral*
two: *Numeral*
. . .
ninety-nine: *Numeral*
one-hundred: *Numeral*
. . .

Figure 1.6

Abstract syntax:

P∈ Program
B∈ Block
D∈ Declaration
C∈ Command
E∈ Expression
O∈ Operator
I ∈ Identifier
N∈ Numeral

P ::= B.

B ::= D;C

D ::= **var** I | **procedure** I; C | D_1; D_2

C ::= I:=E | **if** E **then** C | **while** E **do** C | C_1;C_2 | **begin** B **end**

E ::= I | N | E_1 O E_2 | (E)

O ::= + | − | * | **div**

Figure 1.7

Abstract syntax:

P∈ Program-session
S∈ Command-sequence
C∈ Command
R∈ Record
I ∈ Identifier

P ::= S **cr**

S ::= C **cr** S | **quit**

C ::= **newfile** | **open** I | **moveup** | **moveback** |
insert R | **delete** | **close**

Figure 1.8

Abstract syntax:

P∈ Program
E∈ Expression
L∈ List
A∈ Atom

P ::= E,P | **end**

E ::= A | L | **head** E | **tail** E | **let** A = E_1 **in** E_2

L ::= (A L) | ()