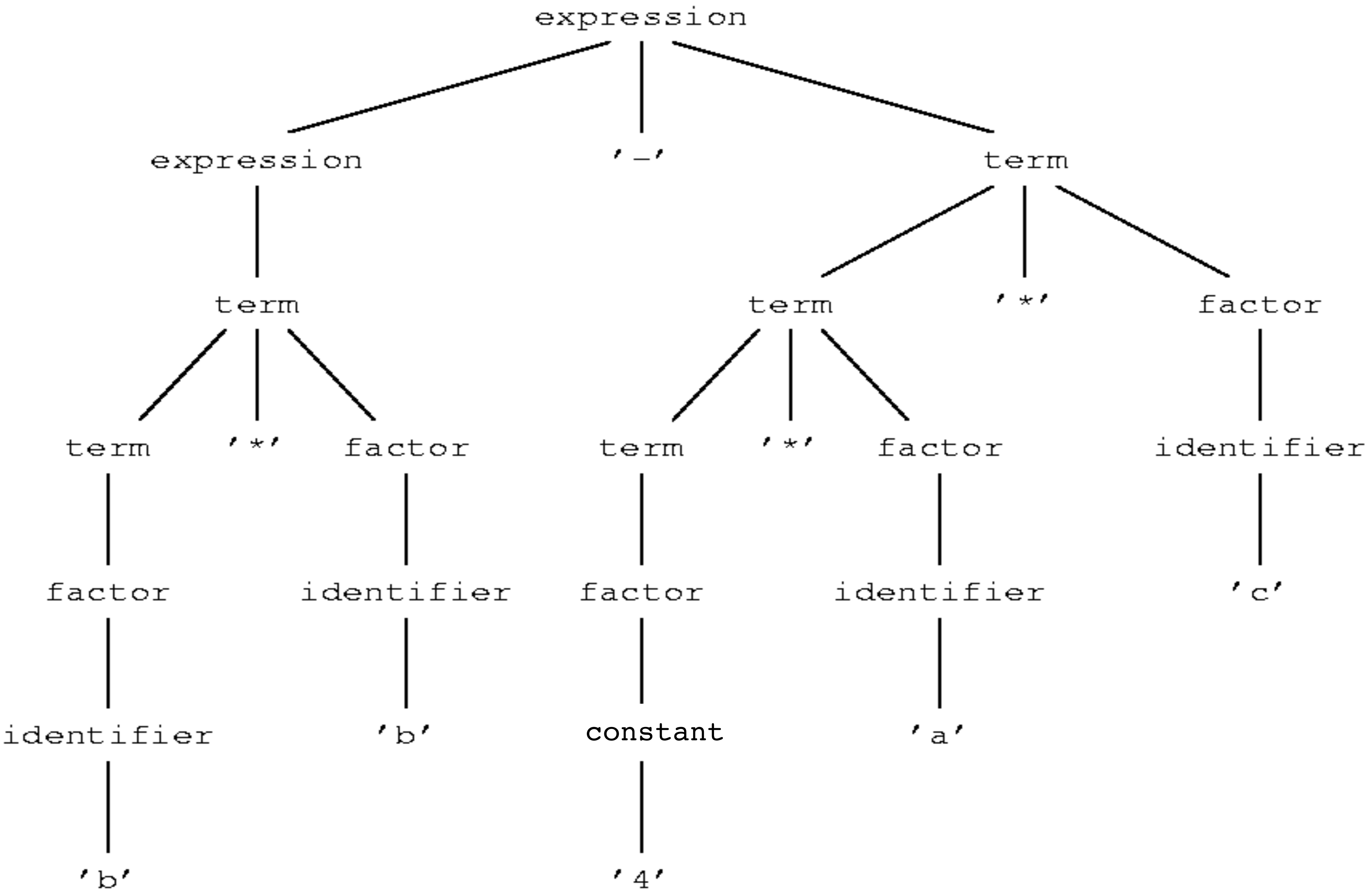
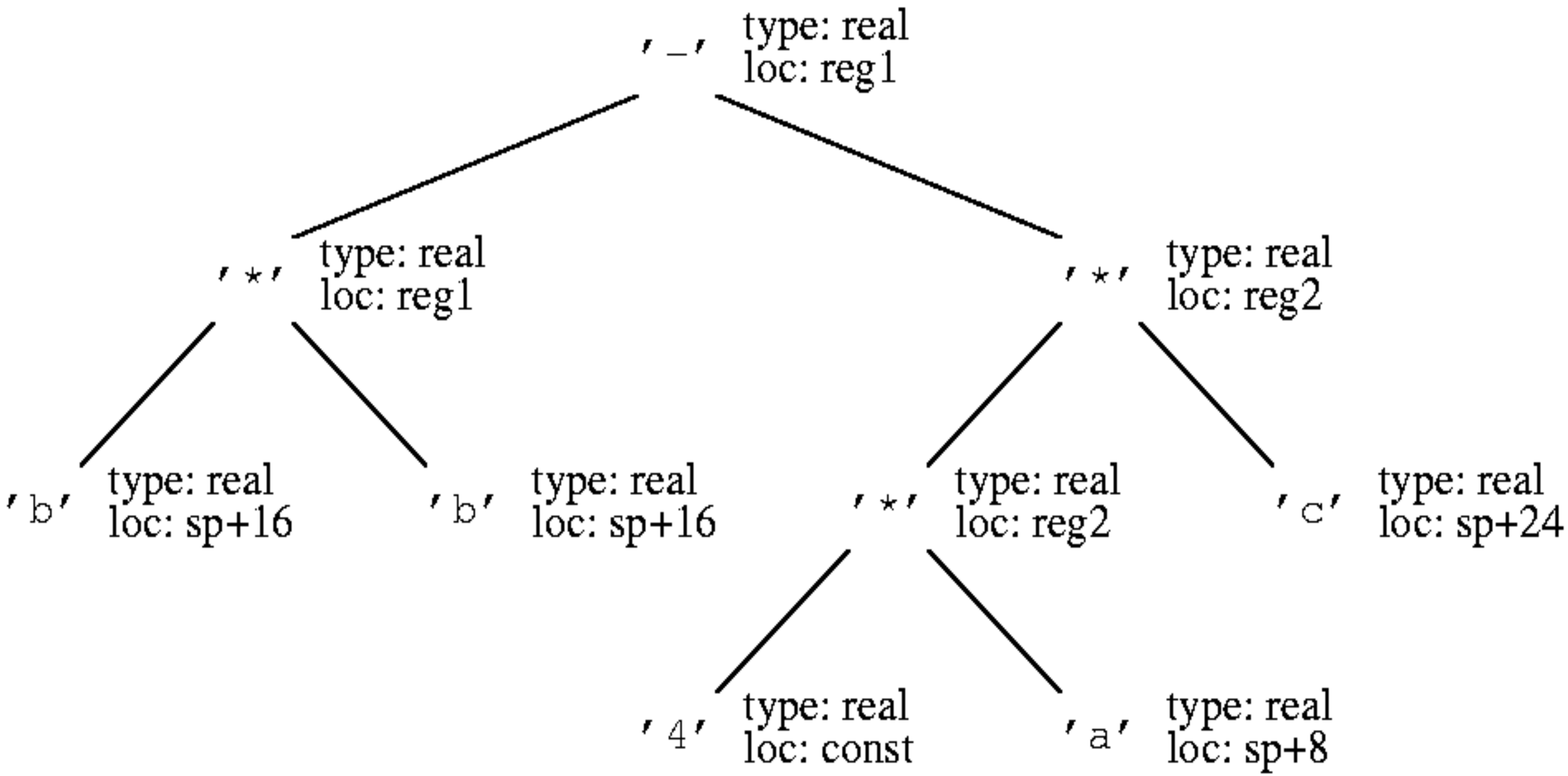
Parse tree for b*b-4*a*c



Annotated abstract syntax tree for b*b-4*a*c



```
<exp>       ::= <exp> + <term>
              | <exp> - <term>
              | <term>
<term>      ::= <term> * <factor>
```

Phrase Structure and Arithmetic Expressions

```
| <term> / <factor>
              | <factor>
<factor>    ::= ( <exp> )
              | <identifer>
```

- There are four operators (+, -, * and /), with two levels of precedence.
- The grammar imposes a ***phrase structure*** on expressions. In `a * b + c` the subexpression `a * b` is a phrase because it corresponds to a subtree of the derivation tree. This phrase structure gives effect to the precedence of the operators.
- The derivation of `a * (b + c)` the parentheses indicate a <factor>, so its derivation tree would be different.

Backus-Naur Form

Here is an example of a grammar:

```
<identifier> ::=  <letter>
                  | <identifier> <digit>
                  | <identifier> <letter>
<letter>     ::= a|b|c|d| ... x|y|z
<digit>      ::= 0|1|2|3|4|5|6|7|8|9
```

The essential features of the BNF formalism are:

1. Angle brackets. These signify non-terminal symbols.
2. The symbol ::= which is read `is defined as'.
3. The symbol | which means ‘or'.
4. The idea of a production rule.
5. A terminal symbol : anything not enclosed in angle brackets.

Ambiguous Grammar Example

```
<statement>          ::=   <conditional statement>
                     | . . .
                     | . . .

<conditional statement> ::=
    if <condition> then <statement>
  | if <condition> then <statement> else <statement>
```

Ambiguity

(from the exercises)

```
<exp>         ::= <exp> + <exp>
                | <exp> * <exp>
                | <ident>
<ident>       ::= x | y | z
```

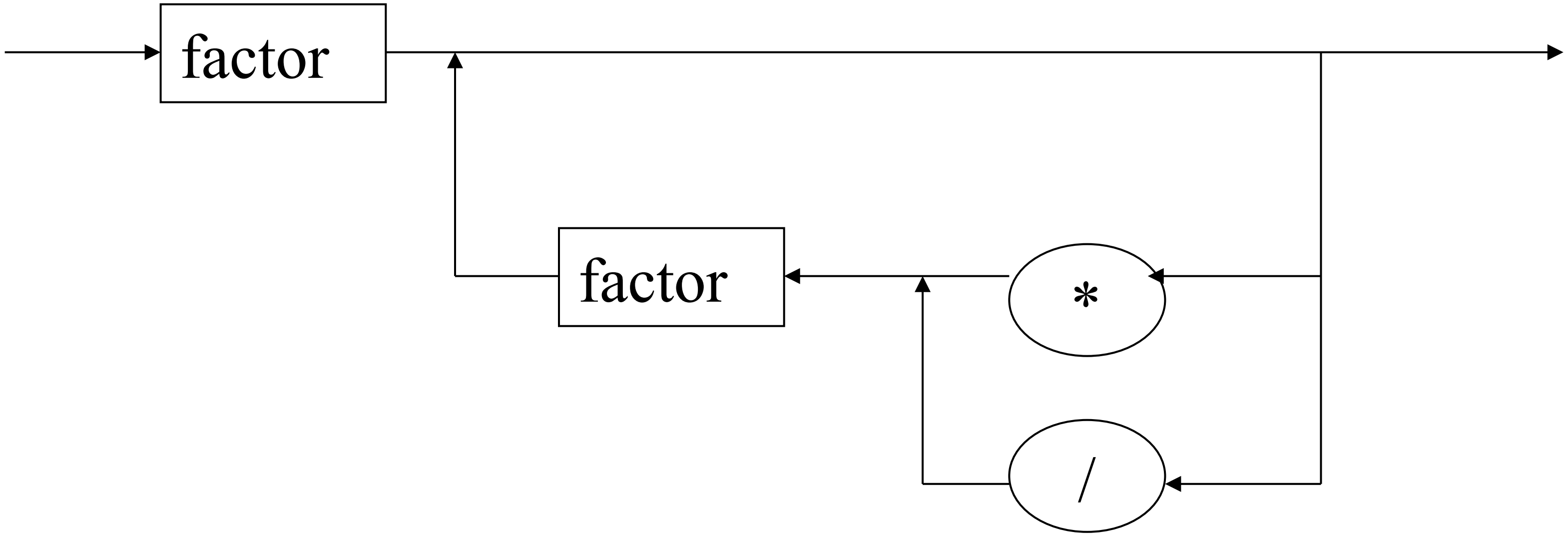
Legality and EBNF conversion

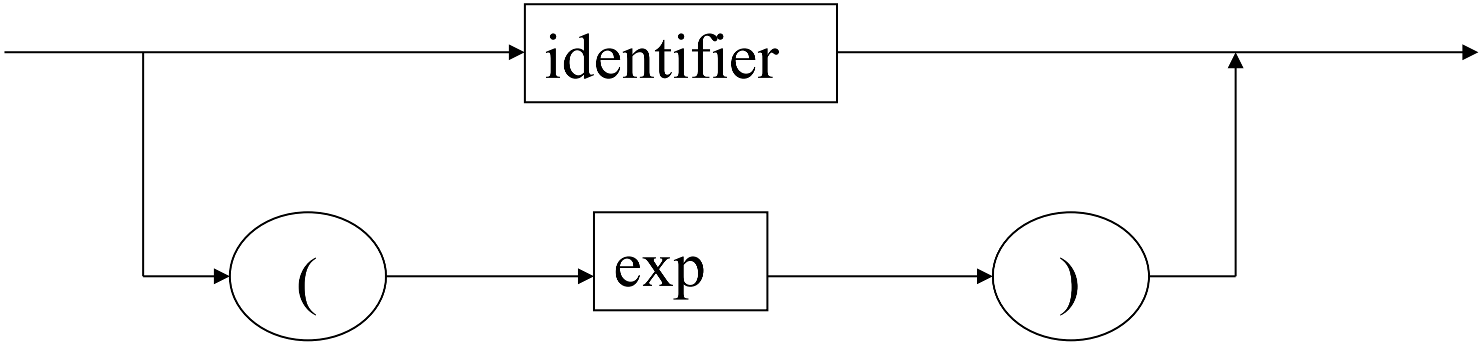
(from the exercises)

```
<rule1> ::= <rule1> Y    | <rule2>
<rule2> ::= Z <rule3>    | Z
<rule3> ::= X
```

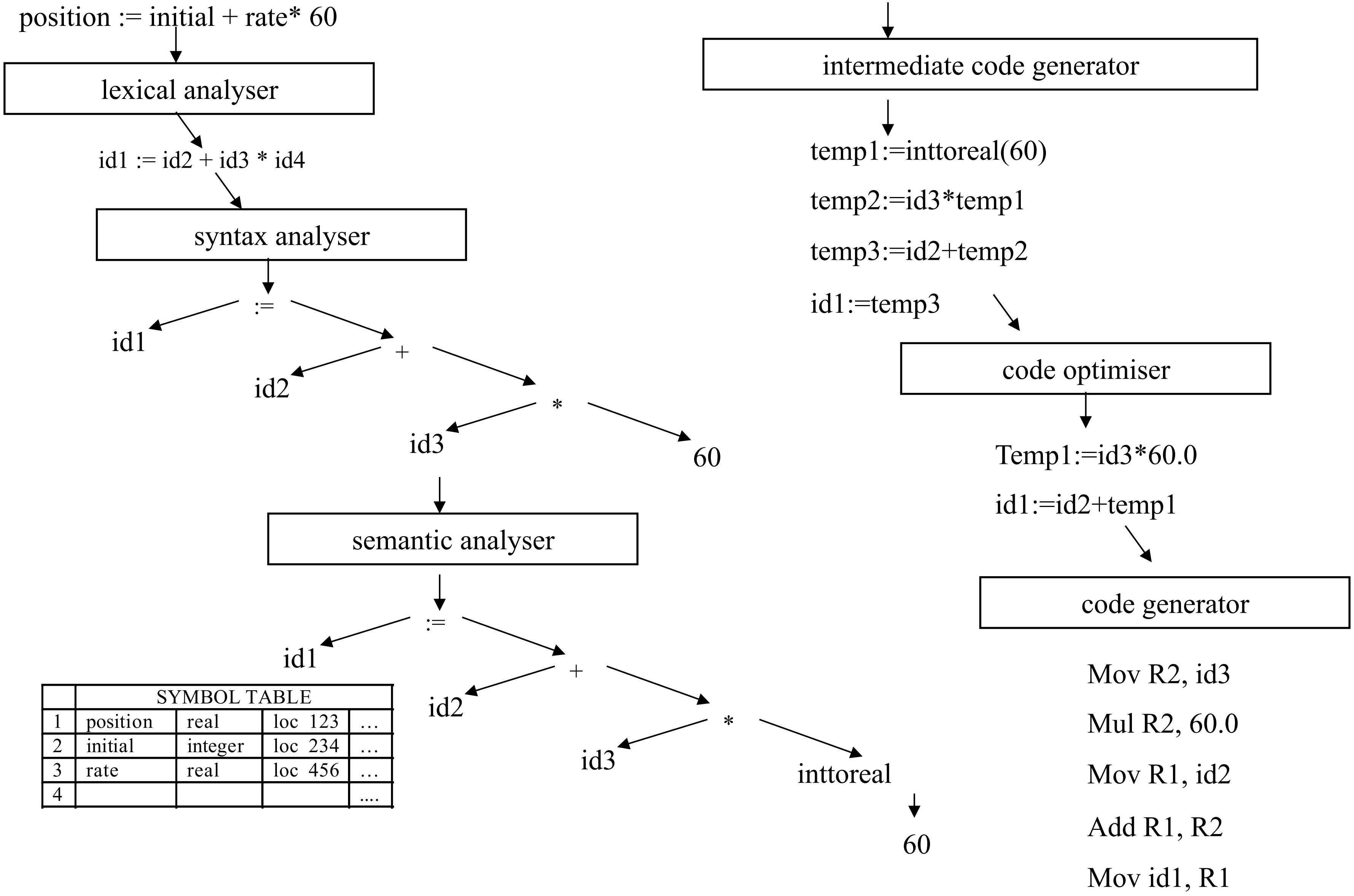
Syntax Diagrams and Extended BNF (EBNF)

```
term -> factor { (‘*’ | ‘/’) factor }
factor -> ‘(‘exp ‘) ‘ | identifier
```





position := initial + rate * 60



position := initial + rate* 60

lexical analyser

id1 := id2 + id3 * id4

syntax analyser

:=
id1
+
id2
*
id3
60

semantic analyser

:=
id1
+
id2
*
id3
inttoreal
60

	SYMBOL TABLE			
1	position	real	loc 123	…
2	initial	integer	loc 234	…
3	rate	real	loc 456	…
4				....

intermediate code generator

temp1:=inttoreal(60)

temp2:=id3*temp1

temp3:=id2+temp2

id1:=temp3

code optimiser

Temp1:=id3*60.0

id1:=id2+temp1

code generator

Mov R2, id3

Mul R2, 60.0

Mov R1, id2

Add R1, R2

Mov id1, R1

Operational Semantics

- Assignment Statements

$$\frac{\text{E |- <exp> => v}}{\text{E |- <identifer> = <exp> => E[<identifer> |-> v]}}$$

$$\frac{\text{E |- <statement> => E'}\quad\text{E' |- <prog> => E''}}{\text{E |- <statement> ; <prog> => E''}}$$

Axiomatic Semantics

$$\{P\}\ S\ \{Q\}$$

- assignment statements (axiom)

$$\{R(e)\}\ x := e\ \{R(x)\}$$

- sequencing program statements (rule of inference)

$$\frac{\{P\}\ S1\ \{R\}\ \{R\}\ S2\ \{Q\}}{\{P\}\ S1;S2\ \{Q\}}$$