POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
Anonas, Sta.Mesa, Maynila,
Kalakhang Maynila

Snek Mini Programming Language Documentation

BSCS 3-4
Department of Computer Science

Chacon, Alissa Carla
Miranda, Allyza
Obias, Jovelyn
Cepe, Carl Joseph
Flordeliza, Jim Schandler
Mojado, Mathew
Parilla, Aaron

Mr. Montaigne Molejon
Professor

I. Introduction

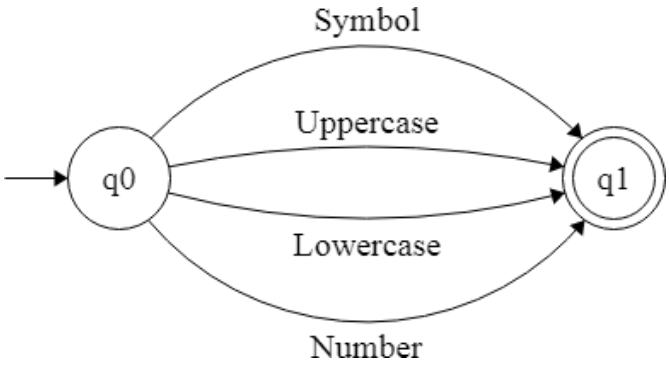
Snek is a newly conceptualized mini programming language designed to perform basic programming functions. Snek mini programming language is heavily inspired by famous programming languages such as C and Python. Its syntax is designed to be simple and light, like the C syntax. The two main feature of Snek programming language is its simple set of keywords and its ability to distinguish reserved words from user-generated entities with the help of apostrophe (‘). The features of Snek are intended to make the programming language simple and suitable for beginners or other programmers who wants to learn a new language. It emphasizes code readability and minimal learning curve for newcomers in the field of computer programming.

Snek mini programming language is a structured programming language that aims to improve clarity and quality of a computer program. It can perform fundamental instructions namely repetition, structured control flow of selection and sequence are at most tasks that can be performed.

II. Synactic Elements of a Language

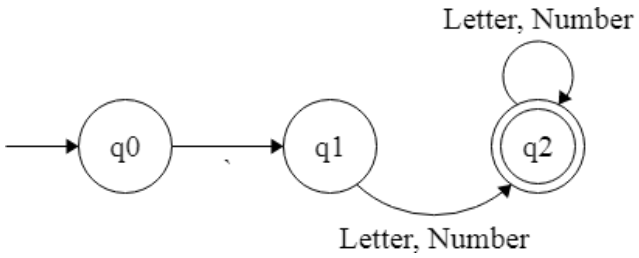
1. Character Set

<Character> → {Letter, Number, Symbol}
<Letter> → {uppercase, lower-case}
<Uppercase> → {A…Z}
<Lowercase> →{a…z}
<Number> →{0,1,2,3,4,5,6,7,8,9}
<Symbol> → { (,), {, }, !, =, >, <, |, /, ”, ,, + , - , %, *, ; ,. , :}



2. Identifiers

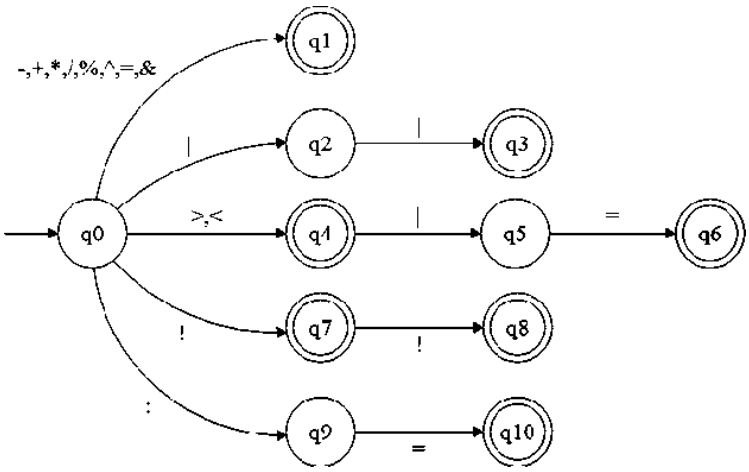
- Always starts with the symbol " ` ".
- Followed by series of letters or numbers.
- It must be case sensitive.



3. Operation Symbols

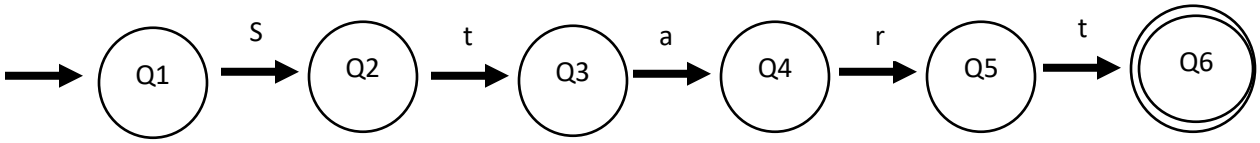
<Arithmetic> → {+, -, /, *, %, ^}
<Logic> → {||, &, !}
<Relational> → {>, <, =, !!, >|=, <|=}
<Assignment> → {:=}

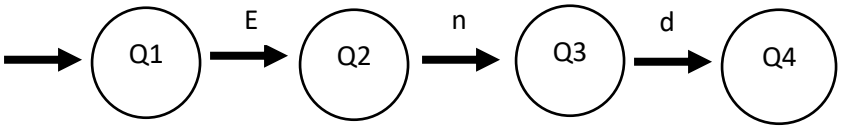
ARITHMETIC		LOGIC		RELATIONAL		ASSIGNMENT	
+	Addition	||	Logic or	<	Less than	:=	Takes
-	Subtraction	&	Logic and	>	Greater than		
/	Division	!	Logic not	=	Equal		
*	Multiplication			<|=	Less than or equal		
^	Exponential			>|=	Greater than or equal		
				!!	Not equal		



4. Noise Words

NOISE WORDS	DESCRIPTION
Start Syntax: Main (Start)	Indicates the start of the program
End Syntax: Main (End)	Indicates the start of the program





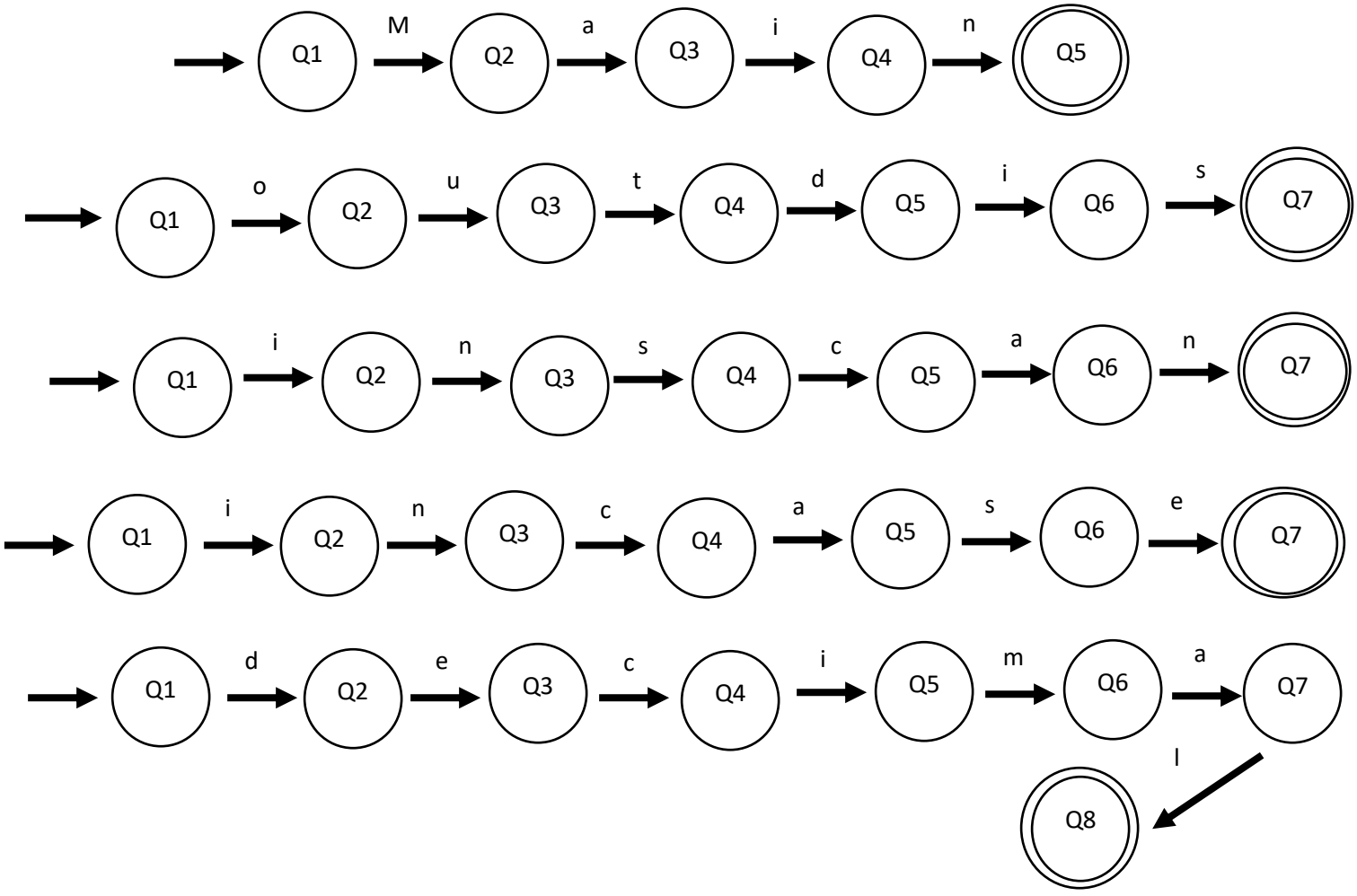
5. Keywords/Reserved Words

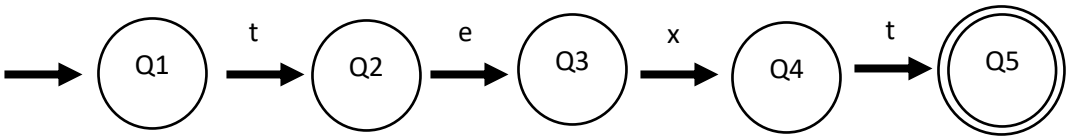
Keywords	Syntax	Description
Main	Main(start)	Starting point for program execution.
outdis	outdis (“text”); outdis (“text”, identifier);	Is used to print string literals and values of the identifiers onto the output screen.
inscan	inscan identifier := value	Is used to read character, string, numeric data from keyboard.
integral	Integral	Define numeric variables holding both positive and negative numbers.
decimal	Decimal	Define numeric variables holding numbers with decimal points.
text	Text	A character or sequence of characters
incase	incase(condition)	Statement is responsible for modifying the flow of execution of a program. incase statement is always used with a condition. The condition is evaluated first before executing any statement inside the body of incase. Statement evaluates the test expression inside the parenthesis ()
then	then {statements}	If the condition in incase will be evaluated as true, statements under “then” will be performed.
else	else {statements}	Will be executed if the condition/s in the incase block is/are not met.
amid	amid (condition)	Repeatedly executes a target statement as long as a given condition is true.
act	act {statements}	Similar to amid, except that it is guaranteed to execute at least one time.

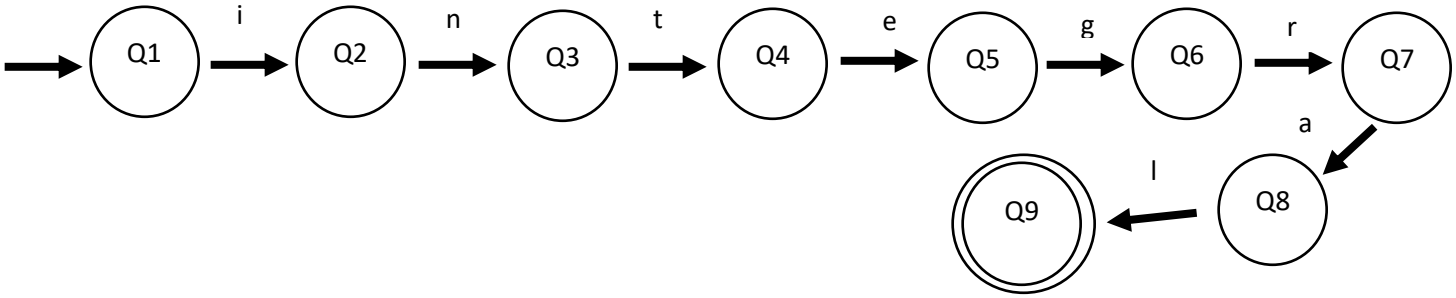
forloop	forloop (initialization; condition; arithmetic expression)	Used for executing a block of statements repeatedly until a given condition returns false

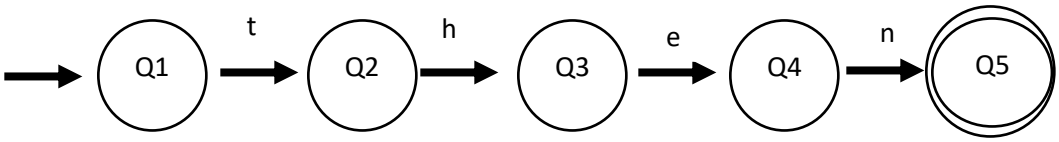
boolean	boolean identifier = true;
	
	boolean identifier = false;
		It is a data type that has one of two possible values (true or false).

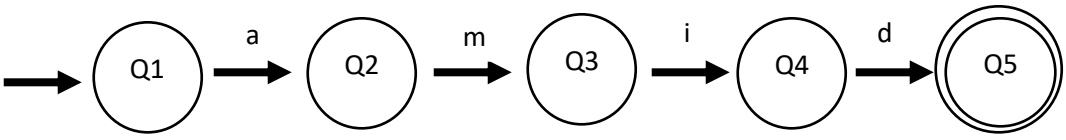
true	boolean identifier = true;	It a value that a boolean data type holds; it is a result of a boolean expression
false	boolean identifier = false;	It a value that a boolean data type holds; it is a result of a boolean expression

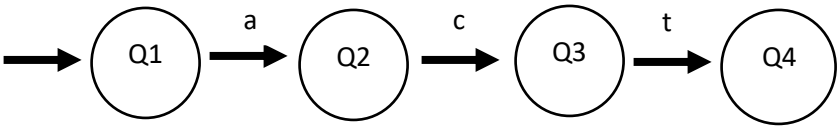


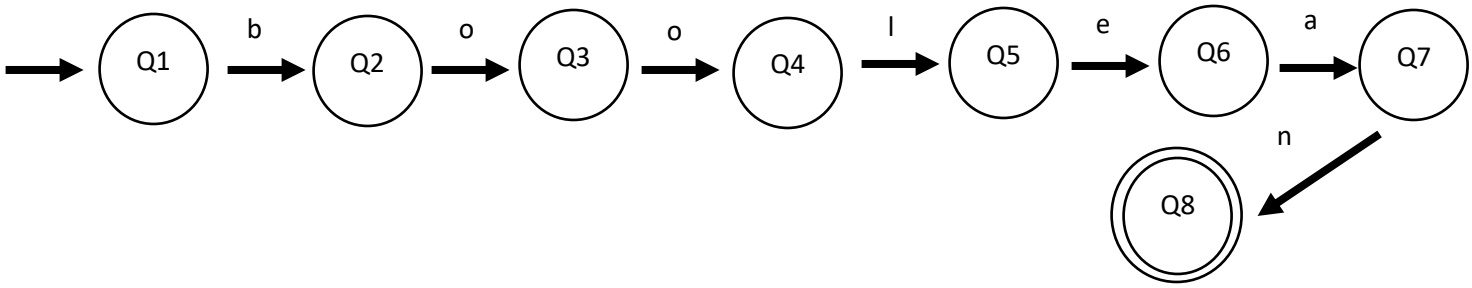

Q1
t
Q2
e
Q3
x
Q4
t
Q5

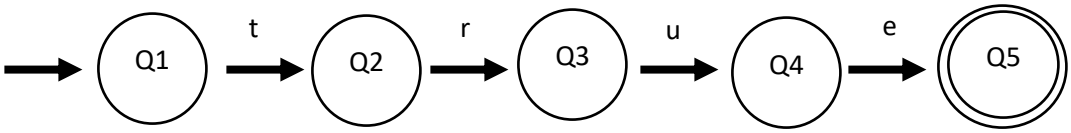

Q1
i
Q2
n
Q3
t
Q4
e
Q5
g
Q6
r
Q7
a
Q8
l
Q9

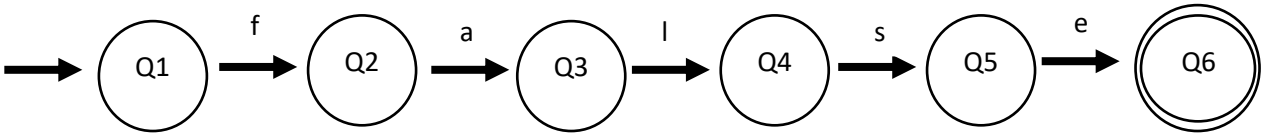

Q1
t
Q2
h
Q3
e
Q4
n
Q5


Q1
a
Q2
m
Q3
i
Q4
d
Q5


Q1
a
Q2
c
Q3
t
Q4


Q1
b
Q2
o
Q3
o
Q4
l
Q5
e
Q6
a
Q7
n
Q8


Q1
t
Q2
r
Q3
u
Q4
e
Q5


Q1
f
Q2
a
Q3
l
Q4
s
Q5
e
Q6

6. Comments

- Comments are used to provide supplementary information making it easier to understand the source code of the computer program. The comments are generally ignored by the interpreter.
- One line comment should start with the symbol “>>”, followed by any combinations of letters and numbers, ends in next line.
- Multiple line comment should start with the symbol “>/” and end with “/<”
- All the strings after this symbol “>/” would be ignored by the compiler and starts to be recognized again after “/<” for multiple lines and end of line for single line comment.

Comment Style	Syntax	Example
Line comments – delimit a region of source code for a single line only. The symbol “>>” indicate the beginning of a comment. A newline character indicates the end of a line comment.	>>	>> This is a line comment
Block comments – delimit a region of source code which may span multiple lines. The symbol “ >/” indicate the beginning and the symbol “/< “ for the end of the block comment.	>/ /<	>/ This is a Block comment /<

7. Blanks

- The use of blank spaces can improve the style of a program. It improves the readability of a program.
- Blank space does not correspond to a visible mark, but it actually occupies an area on a page. Blanks are white spaces such as \t,\n and space which is ignored by the compiler and only significant for text literals.

8. Delimiters and Braces

DELIMITERS	DESCRIPTION	SAMPLE
;	Semicolons are used to identify the end of line in a line of code	Integral ‘a;
{}	Curly Brackets are used to signify a block of code	Main(START){ …… }Main(END)
“”	Double quotes are used to identify String Literals	outdis(“Random”);
,	Commas are used to separate data data fields	outdis(“text” ~ var ~ “text, ‘var);
()	Open and Closed Parenthesis are used as a field for parameters, String Literals, etc.	outdis(“Text”);
>//< >>	Angle Brackets and back slash are used for representing comments	>/Multiple-line comment/< >>One line comment

9. Expressions

9.1 Arithmetic Expression

Precedence	Operators	Order of Evaluation	Example	Result
4	()	If the parentheses are nested, expression in the innermost pair is evaluated first. If there is several pair of parentheses on the same level, they are evaluated from left to right.	((3+2)*(8/2)) 5 * 4 20	20
3	^	If there are no parentheses in the expression, this will always be evaluated first	6/3+2^2 6/3+ 4 6/3+ 4 2 + 4 6	6
2	*, /, %	If there are several on the same level, they are evaluated from left to right.	10%7*6/9 3 *6/9 18/9 2	2
1	+, -	If there are several on the same level, they are evaluated from left to right.	2+3-1-2+10 5 -1-2+10 4 -2+10 2 +10 12	12

Example combination of all arithmetic operators

((45-12*2) + (20/5%1) * 3^2) → Inner parentheses first, left to right
((45- 24)) + (20/5%1) * 3^2) → * has a higher precedence
(21 + (20/5%1) * 3^2) → First inner parentheses has been evaluated
(21 + (4 %1) * 3^2) → All operators on the second inner parentheses is on the same level of precedence, left to right

9.2 Relational Expression

Precedence	Operators	Order of Evaluation	Example	Result
2	< > >|= <|=	This will always be evaluated first.	2 > 4 == 5 <|= 3 false == false true	true
1	= !!	Evaluated last.		

9.3 Logical Expression

Precedence	Operators	Order of Evaluation	Example	Result
4	!	Highest precedence among all operators.	(2>1) & !(6=2) || (5<10) (2>1) & !(false)|| (5<10) (2>1) & true || (5<10) true & true || (5<10) true || (5<10) true || true true	true
3	()	If the parentheses are nested, expression in the innermost pair is evaluated first. If there is several pair of parentheses on the same level, they are evaluated from left to right.	(((9 < 2) || (6 >|= 2)) & (4=1)) ((false || (6 >|= 2)) & (4=1)) ((false || true) & (4=1)) ((false || true) & (4=1)) (true & (4=1)) (true & false) false	false
2	&	After all the expressions inside parentheses have been evaluated, this will always be evaluated first if there are other logical operators on the same level pair of parentheses. If all are the same operators on the same level, it will be evaluated from left to right.	((6 > 4) || (1 < 10) & (2 >|=5)) (true || true & false) (true || false) true	true
1	||	Last to be evaluated. Left to right evaluation also if same operators on the same level pair of parentheses.	((6 > 4) || (1 < 10) & (2 >|=5)) (true || true & false) (true || false) true	true

9.4 All Expressions

Precedence	Operators	Order of Evaluation	Example
6	!	Highest precedence on all operators.	X = 5 Y = 2 Z = 3
5	()	If the parentheses are nested, expression in the innermost pair is evaluated first. If there is several pair of parentheses on the same level, they are evaluated from left to right.	(y = x + 3) > 10 || 2= Z & (2+Y <|= X & !(Z = Y + X*2 > 20)) (y = x + 3) > 10 || 2= Z & (2+Y <|= X & !(Z = Y + 10 > 20)) (y = x + 3) > 10 || 2= Z & (2+Y <|= X & !(Z = 12 > 20)) X = 5 Y = 2 Z =12
4	Arithmetic Operators	Precedence of arithmetic operators are also applied.	(y = x + 3) > 10 || 2= Z & (2+Y <|= X & !(12 > 20)) (y = x + 3) > 10 || 2= Z & (2+Y <|= X & !false)
3	Relational Operators	Precedence of relational operators is also applied.	(y = x + 3) > 10 || 2= Z & (2+Y <|= X & true) (y = 8) > 10 || 2= Z & (2+Y <|= X & true) X = 5 Y = 8 Z =12 8 > 10 || 2= Z & (2+Y <|= X & true) 8 > 10 || 2= Z & (10 <|= X & true)
2	Logical Operators	Precedence of logical operators is also applied.	8 > 10 || 2= Z & (false & true) 8 > 10 || 2= Z & false false || 2= Z & false
1	=	Lowest precedence among all operator hence, last to be evaluated	false || false & false false || false **false**

- In an expression, operator with the highest precedence is grouped with its operands first, then the next highest operator will be grouped with its operands and so on. If there are several operators of the same precedence, they will be examined left to right.
- All expression should always start with either a brace (parenthesis only), symbol " ' " (for variable/identifier), numbers (for constant values)
  Example:
  - ('x + 3)
  - 'x * 3
  - 3 / 'x
- Every opening brace should have its corresponding close brace.
  - (('Y + 'X – 3) > 5)

- First operand which can be letters or numbers will be followed by an operator and ends with another operand composed of letters or numbers or can also end with braces to partner its corresponding open brace.
  - 'Y % 4
  - ('Y > 'X) && ('Z <|= 'X)
- It can only start with an operator in logic expression using the operator "!" followed by a relational expression or logical expression which inverts their resulting logical value – true or false.
  - !(4 < 5)
- Arithmetic operators cannot perform operation between relational expressions and logic expressions. Only for letters/numbers
  - (3 > 4) +(6 < 4) → wrong
  - (3 > 4 && 6 =2) - ('X < 4 || 3 >|= 'Y) → wrong
- Relational operator cannot perform operation between logic expressions
  (3 > 4 && 6 =2) >|= ('X < 4 || 3 >|= 'Y) → wrong
- Logic operator can only perform logic and relational expressions
  - (Y > X) && (Z <|=X)
  - (Y > X) || (Z <|=X && 3 > 4)
  - ! ('X = 2)
  - !('X + 3)

10. Statements

Syntax	Example
Comment	
>> <character> one line	>> uno is life
>/ <character> two or more lines /<	>/ Multiple Line /<
Declaration	
<data_type> <identifier>;	integral `a;
Data type	
<data_type> <identifier>;	integral `catNumber;
<data_type> <identifier>;	decimal `average;
Declaration plus initialization/assignment	
<data_type> <identifier> <assignment_operator> literal_value;	decimal `dogNumber := 32.3;
<data_type> <identifier> <assignment_operator> “string_literal_value”;	text `dogName := “Manny”;
Initialization/assignment (assigning of initial value separated from the declaration)	
<identifier> <assignment_operator> literal_value;	`dogNumber := 3;
<identifier> <assignment_operator> “string_literal_value”;	`catName := “Chico”;
Assignment (assigning of new value to a variable with a value already)	
<identifier> <assignment_operator> literal_value;	`deerNumber := 13;
<identifier> <assignment_operator> “string_literal_value”;	`catName := “elChoco”;
<identifier> <assignment_operator> <identifier>;	`birdNumber := `eagleNumber;
<identifier> <assignment_operator> <arithmetic_expression>;	`average := 2+10/5;
Input	
inscan <identifier> <assignment_operator> literal_value;	inscan a := 28;
inscan <identifier> <assignment_operator> “string_literal_value”;	inscan b := “Argentina”;

Output		
Syntax	**Example**	**Output**
outdis (“string_literal_value”);	outdis (“text”);	text
outdis (“string_literal_value”, <identifier>,“string_literal_value”, `<identifier>);	‘x = "This"; outdis("that", ‘x , "there", 'x);	thatThisthere
outdis(<identifier>);	‘x=1; outdis(‘x);	1
Conditional		
incase, then, else		
incase(condition) then{statements}	incase(‘x=3) then { ‘y:= “cat”;}	
Incase(condition) then{statements} else{statements}	incase(‘x=3) then { ‘y:= “cat”;} else { ‘y:= “dog”; z:= 4; ‘x:= z+2; }	
Loop/iteration		
amid, act		
amid(condition){statements}	amid(x=z){ ‘ y:= zebra; ‘total := 1+2+3+4; ‘x := ‘x+1;}	

act{statements}amid(condition);	act {
	‘y:= zebra ;
	integral total := 1+2+3+4;
	amid(‘x = ’z);	

forloop		
forloop (initialization; condition; arithmetic expression){statements}	forloop (integral ‘x :=0; ‘x<|=10; ‘x := ‘x+1){ ‘y := zebra; }	

III. Design Issues

Upon constructing Snek programming language, drawbacks and potential downsides to the implemented structural design have been anticipated by Snek developers. These drawbacks are:

- Readability for identifiers with extensive use of numbers or only use numbers.
- Lacks capital letters for keywords/reserved words and basic syntax.
- Case Sensitive, same name with different capital letters would result in different meanings.
- Identifiers do not support the usage of symbols.
- Has no support for switches.
- Does not support terminating keywords and relies on symbols and brackets.
- Indentations don't affect code structure, could potentially result in messy code blocks.
- Very few syntactic sugar and syntactic salt.
- Built to avoid overly verbose code.
- Potentially too terse.
- Has no candy grammar feature.
- Lacks support for abstraction.
- Lacks support for extensibility.

IV. List of Data Types for Snek Programming Language

1. Text

Text data type primarily holds alphanumeric data, can be single or group of characters like letters, numbers, punctuations and spaces that are stored together as one. It should be enclosed in double quotes (“”) for it to be accepted. The quote marks aren’t printed when the string is displayed. If the programmer prefers printing the quotes, just place it right after the outside quotes.

Example:

Input	Output
“Snek is a mini programming language”	Snek is a mini programming language
“”Snek””	”Snek”

2. Integral

Integral data type accepts values that are whole numbers, which are numbers not having any fractional component. It can be negative or positive. For positive, there is no need to specify its sign or else it will cause an error. But in the case of negative integers, a minus (-) sign should be placed immediately before the digit. For larger values, commas should not be used as it functions as a separator of values.

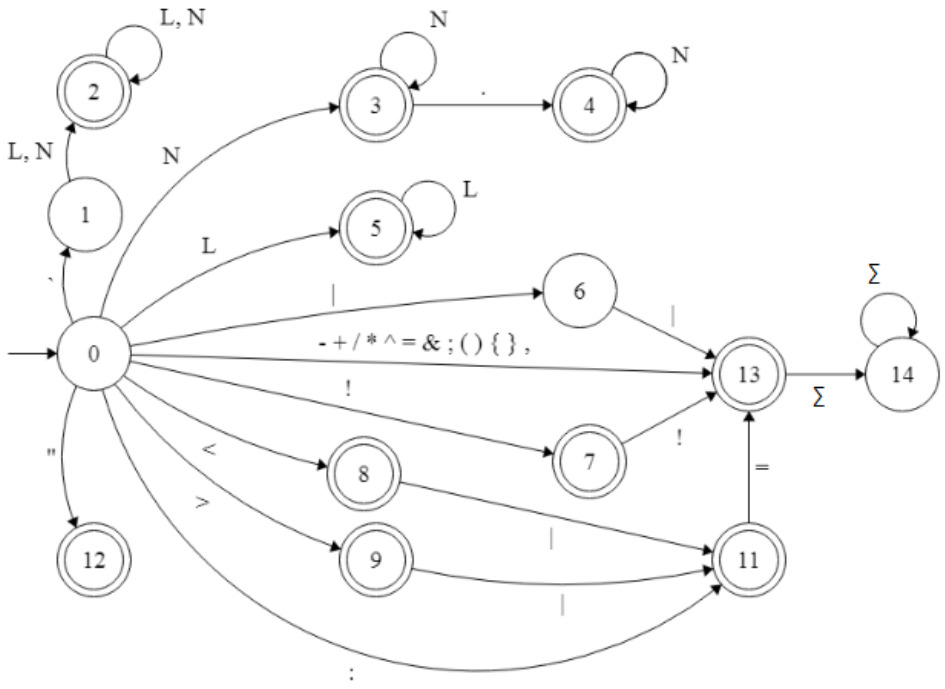
3. Decimal

Decimal data type, unlike integral, it contains fractional component. Set of numbers should have period (.) in between them. Strictly, there are no spaces after or before the decimal point or it will display error.

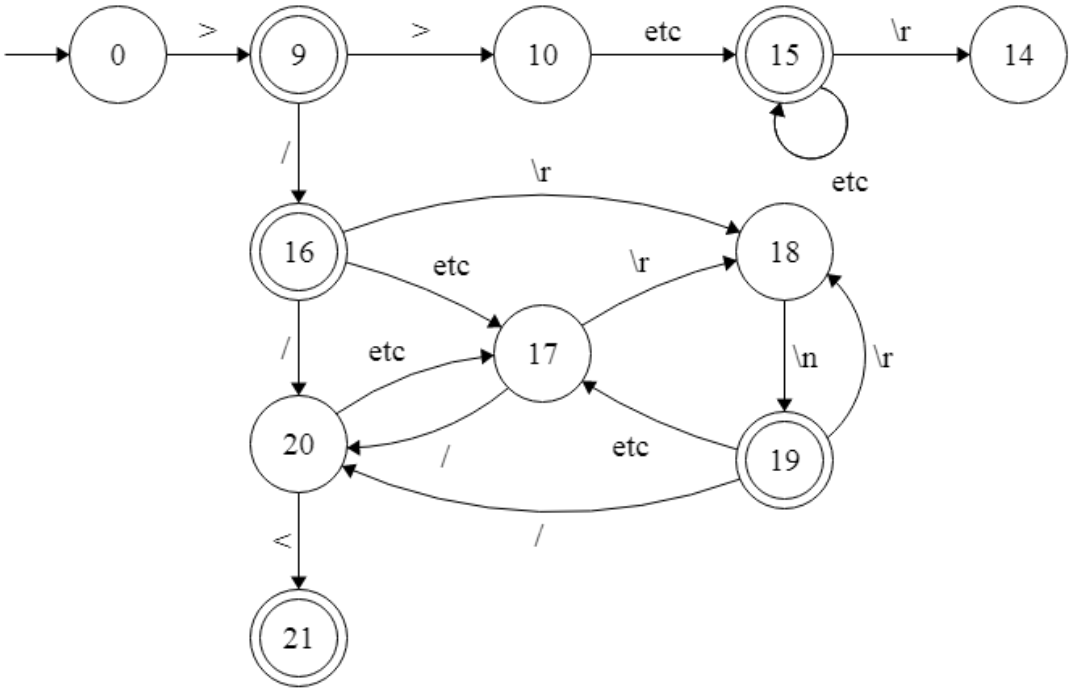
4. Boolean

Boolean data type stores only two possible values such as “true” or “false” which is useful in conditional statements. Boolean type is primarily the result of conditional statements, which are used to control workflow in program

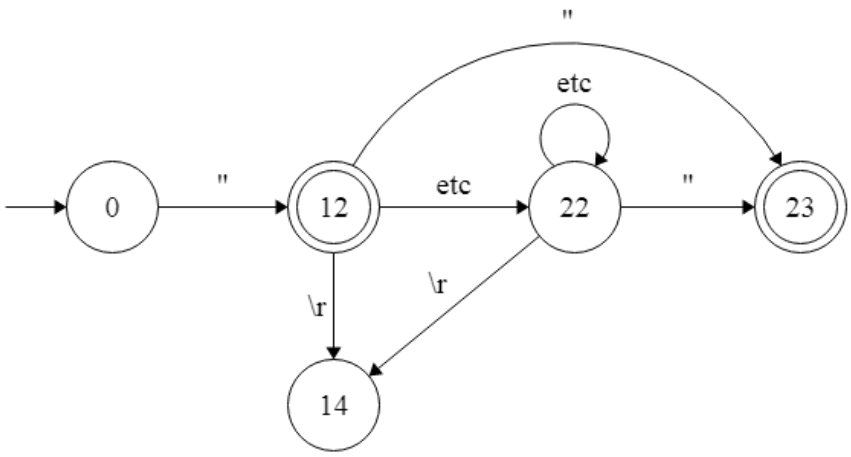
V. Snek Automaton



States	Meaning
{0,1,2}	Identifier
{0,3,4}	Integer values
{0,5}	Keywords/Reserved words/Invalid Words
{0,6,13}	OR operator
{0,13}	Single character symbols
{0,7,13}	NOT EQUAL operator
{0,8,11}	LESS THAN OR EQUAL operator
{0,9,11}	GREATER THAN OR EQUAL operator
{0,11,13}	ASSIGNMENT operator
{0,12}	Double Quotation



States	Meaning
{0,9}	Single Comment Symbol
{0,15}	Single Line Comment
{9,16}	Open Multiple Line Comment Symbol
{17,18,19,20}	Multiple Line Comment
{20,21}	Close Multiple Line Comment Symbol



States	Meaning
{0,12}	Open Quotation
{12,22}	Text Literal
{12,13} and {22,13}	Close Quotation

Note:

- /r and /n represents enter
- etc represents all other possible inputs
- State 14 is a dead state
- Each state is connected to the dead state for all other possible inputs that is not specified in the automaton

VI. SYNTAX

Backus-Naur Form of Snek Mini Programming Language

START // Program should start and end with Main()

<snekPL> ::= Main(Start) {<statement>} Main(End)

STATEMENTS // Program can have varieties of statements.

<statement> ::= <add_statement> <statement>

| Є

<add_statement> ::= <declaration statement>;

| <i/o statement>

| <loop statement>

| <conditional statement>

| <assignment>

DECLARATION // For declaration, it requires data type and type of declaration

<declaration statement> ::= <data type> <declaration type>

<data type> ::= integral

| decimal

| text

| boolean

// Variables can be declared with or without initialzation

<declaration type> ::= <assignment> <list>

| <variable> <list>

// there can be a list of variables to be declared in a certain data type

<list> ::= , <assignment> <list>

| , <variable> <list>

| Є

// can assign literals, arithmetic expression or even variable to a variable

<assignment> ::= <variable> := <literals>

| <variable> := <arithmetic expression>

| <variable> := <variable>

<literals> ::= <constant> | “<text literal>”| <boolean literal>

<constant> ::= <integral literal> | <decimal literal>

<boolean literal> ::= true | false

INPUT/ OUTPUT STATEMENT

<i/o statement> ::= <output> | <input>

<output> ::= outdis (<print>);

//can print literals and variables with concatenation symbol comma (,)

<print> ::= “<text literal>”<print>

| <variable> <print>

| <constant> <print>

| , <print>

| Є

// only one inscan for every input

<input> ::= inscan (<variable>);

LOOP STATEMENT

<loop statement> ::= <forloop> | <amid loop> | <act_amid loop>

<forloop> ::= forloop (<assignment> ; <relational expression>; <arithmetic expression>){<statement>}

<amidloop> ::= amid (<logical expression>) {statement}

<act_amidloop> ::= act {<statement>} amid (<logical expression>)

CONDITIONAL STATEMENT //curly braces are required even with one statement

<conditional statement> ::= if (<logical expression>) then { <statement>}

| if (<logical expression>) then {<statement>} else {<statement>}

EXPRESSIONS

//expressions are hierarchical
//arithmetic expression is under relational expression, relational expression is under logical expression
//precedence and associativity are considered

<logical expression> ::= <logical expression> || <logical term>

| <logical term>

<logical term> ::= <logical term> & <relational expression>

| <relational expression>

<relational expression>::= <relational expression> = <relational term>

| <relational expression> !! <relational term>

| <relational term>

<relational term> ::= <relational term> > <arithmetic expression>

| <relational term> < < arithmetic expression >

| <relational term> <|= < arithmetic expression >

| <relational term> >|= < arithmetic expression >

| <arithmetic expression>

<arithmetic expression> ::= <arithmetic expression> + <arithmetic term>

| <arithmetic expression> - < arithmetic term>

| <arithmetic term>

<arithmetic term> ::= < arithmetic term > * < arithmetic factor>

| <arithmetic term> / < arithmetic factor>

| <arithmetic term> % < arithmetic factor>

| <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<exponent> ::= ^ <base> <exponent> | Є

<base> ::= !base | <variable> | <constant> |<boolean literal>| (<logical expression>)

Example 1:

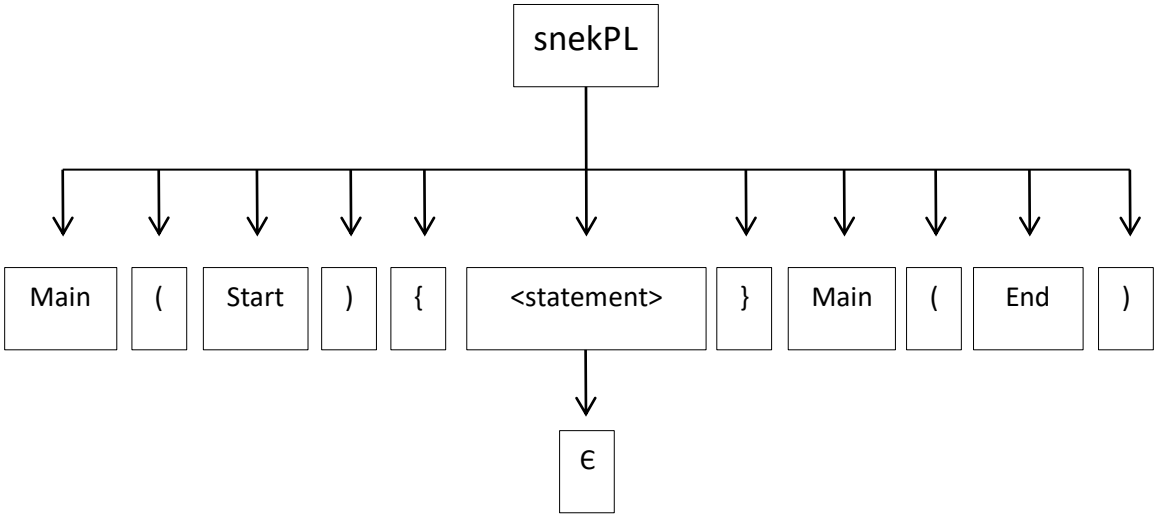
```
Main(Start){

}Main(End)
```

G(V) = {<snekPL>, <statement>}
G(T) = {Є}
G(S) = {snekPL}
G(P)

<snekPL> ::= Main(Start) {<statement>} Main(End)
<statement> ::= Є

LEFTMOST and RIGHTMOST DERIVATION/TREE

<snekPL> → Main(Start) {<**statement**>} Main(End)
→ Main(Start) { Є } Main(End)



Example 2:

```
Main(Start){
Integral ‘age := 12;
}Main(End)
```

G(V) = {<statement>, <add_statement>, <declaration statement>, <data type>, <declaration type>,<assignment>,<list>,<variable>, <literals>, <constant>, <integral literal>}

G(T) = {integral, ‘age, 12,Main, Start, End, (,),{,},:=,;, Є}

G(S) = {snekPL}

G(P)

<snekPL> ::= Main(Start) {<statement>} Main(End)
<statement> ::= <add_statement> <statement>| Є
<add_statement> ::= <declaration statement>;
<declaration statement> ::= <data type> <declaration type>
<data type> ::= integral
<declaration type> ::= <assignment> <list>
<list> ::= Є
<assignment> ::= <variable> := <literals>
<literals> ::= <constant>
<constant> ::= <integral literal>
<integral literal> ::= 12
<variable> ::= ‘age

LEFTMOST DERIVATION

<snekPL> → Main(Start) { **<statement>** } Main(End)
→ Main(Start) { **<add_statement>** <statement> } Main(End)
→ Main(Start) { **<declaration statement>** <statement> } Main(End)
→ Main(Start) { **<data type>** <declaration type> <statement> } Main(End)
→ Main(Start) { integral **<declaration type>** <statement> } Main(End)
→ Main(Start) { integral **<assignment>**<list> <statement> } Main(End)
→ Main(Start) { integral **<variable>** := <literals><list>; <statement> } Main(End)
→ Main(Start) { integral ‘age := **<literals>**<list>; <statement> } Main(End)
→ Main(Start) { integral ‘age := **<constant>**<list> ; <statement> } Main(End)
→ Main(Start) { integral ‘age := **<integral literal>**<list> ; <statement> } Main(End)
→ Main(Start) { integral ‘age := 12 <list>; <statement> } Main(End)
→ Main(Start) { integral ‘age := 12 Є; Є } Main(End)

RIGHTMOST DERIVATION

<snekPL> → Main(Start) { **<statement>** } Main(End)

→ Main(Start) { <add_statement> **<statement>** } Main(End)

→ Main(Start) { **<add_statement>** Є} Main(End)

→ Main(Start) { **<declaration statement>** Є} Main(End)

→ Main(Start) { <data type> **<declaration type>** Є} Main(End)

→ Main(Start) { <data type> **<assignment>** <ist>Є} Main(End)

→ Main(Start) { <data type> <variable> := **<literals>** <ist>; Є} Main(End)

→ Main(Start) { <data type> <variable> := **<constant>** <ist>; Є} Main(End)

→ Main(Start) { <data type> <variable> := **<integral literal>** <ist>; Є} Main(End)

→ Main(Start) { <data type> **<variable>** := 12 <list>; Є} Main(End)

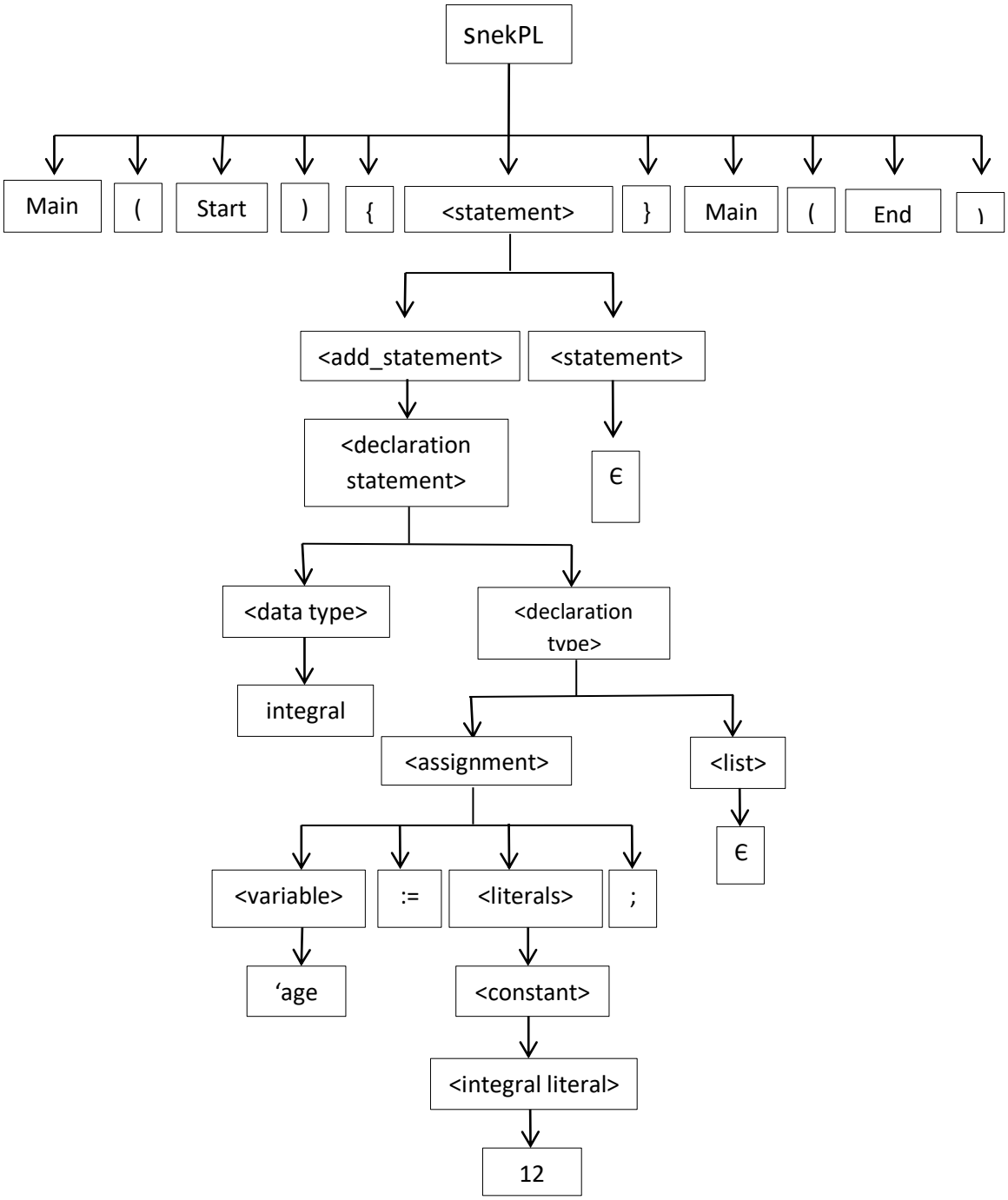
→ Main(Start) { <data type> **<variable>** := 12 Є; Є} Main(End)

→ Main(Start) { **<data type>** ‘age := 12 Є; Є} Main(End)

→ Main(Start) { integral ‘age := 12 Є; Є} Main(End)

(TREE ON NEXT PAGE)

LEFTMOST and RIGHTMOST TREE



Example 3:

```
decimal ‘gwa := 1.00, ‘price := 25.50;

text ‘name, ‘birthdate;
```

G(V) = {<statement>, <add_statement>, <declaration statement>, <data type>, <declaration type>,<assignment>,<list>,<variable>, <literals>, <constant>, <decimal literal>}

G(T) = {decimal, text, 1.00,25.50, ‘gwa, ‘price, ‘name, ‘birthdate,:=,;, Є}

G(S) = {<statement>}

G(P)

<statement> ::= <add_statement> <statement>| Є

<add_statement> ::= <declaration statement>;

<declaration statement> ::= <data type> <declaration type>

<data type> ::= decimal | text

<declaration type> ::= <assignment> <list> | <variable> <list>

<list> ::= , <assignment> <list>| , <variable> <list>| Є

<assignment> ::= <variable> := <literals>

<literals> ::= <constant>

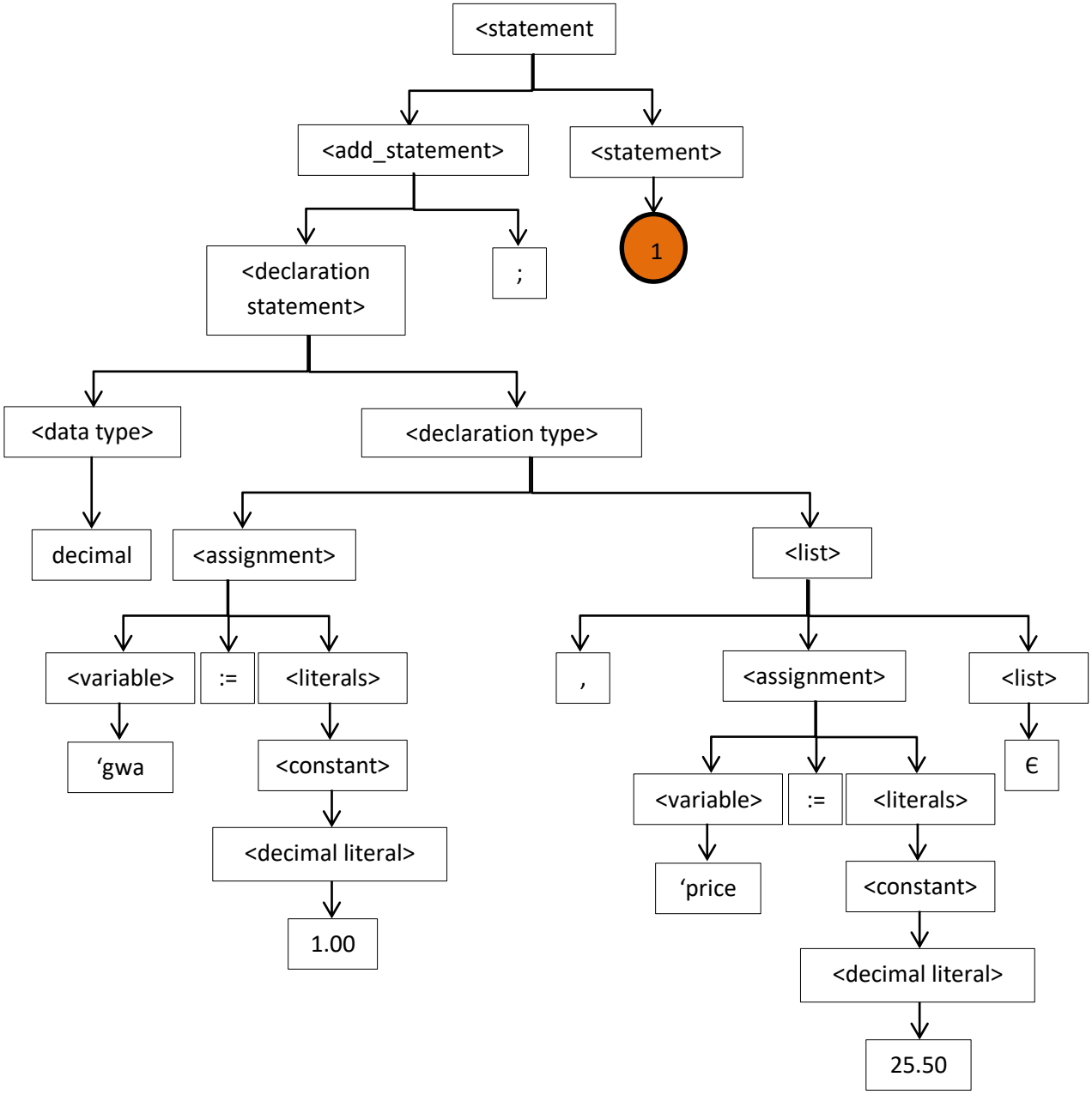
<constant> ::= <decimal literal>

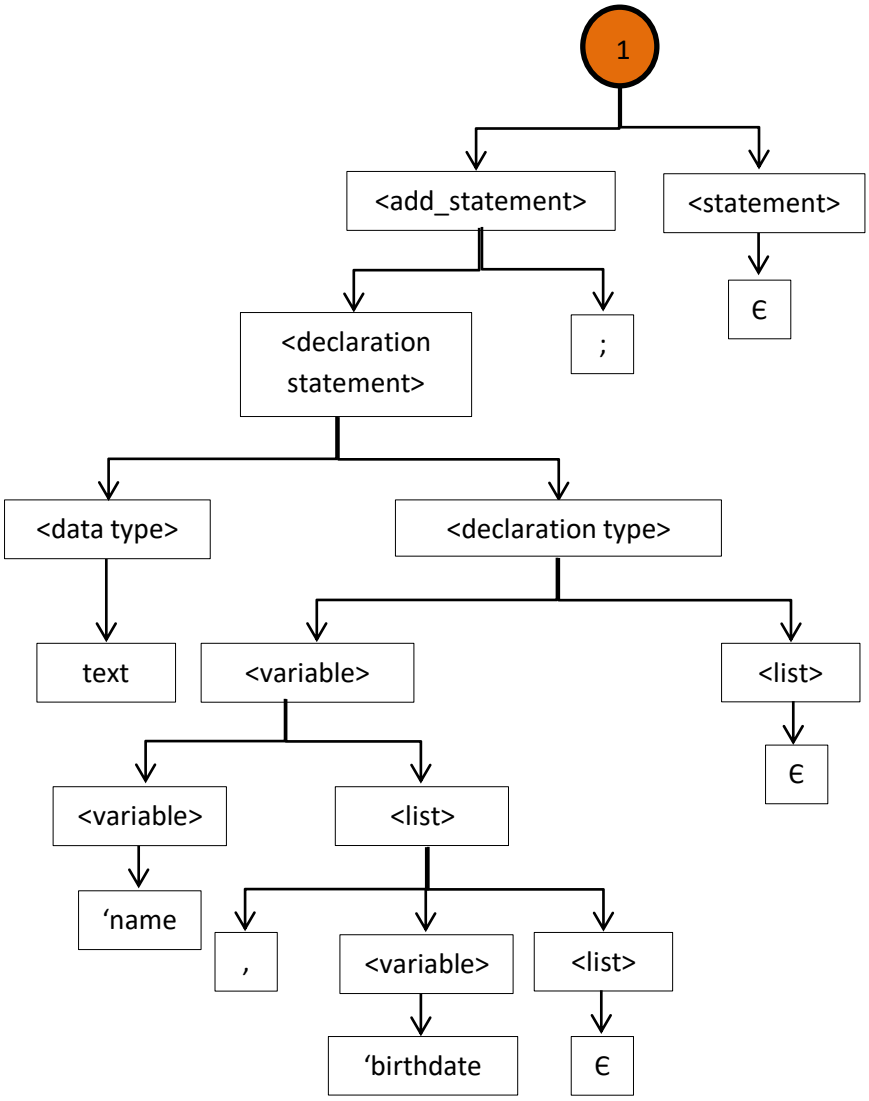
<decimal literal> ::= 1.00 | 25.50

<variable> ::= ‘gwa | ‘price | ‘name | ‘birthdate

LEFTMOST DERIVATION

<statement> → **<add_statement>** <statement>
→ **<declaration statement>**; <statement>
→ **<data type>** <declaration type>; <statement>
→ decimal **<declaration type>**; <statement>
→ decimal **<assignment>** <list>; <statement>
→ decimal **<variable>** := <literals> <list>; <statement>
→ decimal ‘gwa := **<literals>** <list>; <statement>
→ decimal ‘gwa := **<constant>** <list>; <statement>
→ decimal ‘gwa := **<decimal literal>** <list>; <statement>
→ decimal ‘gwa := 1.00 **<list>**; <statement>
→ decimal ‘gwa := 1.00, **<assignment>** <list>; <statement>
→ decimal ‘gwa := 1.00, **<variable>** := <literals> <list>; <statement>
→ decimal ‘gwa := 1.00, ‘price := **<literals>** <list>; <statement>
→ decimal ‘gwa := 1.00, ‘price := **<constant>** <list>; <statement>
→ decimal ‘gwa := 1.00, ‘price := **<decimal literal>** <list>; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 **<list>**; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; **<statement>**
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; **<add_statement>** <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; **<declaration statement>**; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; **<data type>** <declaration type>; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text **<declaration type>**; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text **<variable>** <list>; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text ‘name **<list>**; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text ‘name , **<variable>** <list>; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text ‘name , ‘birthdate **<list>**; <statement>
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text ‘name , ‘birthdate Є; **<statement>**
→ decimal ‘gwa := 1.00, ‘price := 25.50 Є ; text ‘name , ‘birthdate Є; Є


<statement
<add_statement>
<statement>
<declaration statement>
;
1
<data type>
<declaration type>
decimal
<assignment>
<list>
<variable>
:=
<literals>
,
<assignment>
<list>
‘gwa
<constant>
<variable>
:=
<literals>
Є
<decimal literal>
‘price
<constant>
1.00
<decimal literal>
25.50


1
<add_statement>
<statement>
ϵ
<declaration statement>
;
<data type>
<declaration type>
text
<variable>
<list>
ϵ
<variable>
<list>
‘name
,
<variable>
<list>
‘birthdate
ϵ

RIGHTMOST DERIVATION

<statement> → <add_statement> **<statement>**

→ < add_statement> <add_statement> **<statement>**

→ < add_statement> **<add_statement>** Є

→ < add_statement> **<declaration statement>**; Є

→ < add_statement> <data type> **<declaration type>**; Є

→ < add_statement> <data type> <variable> **<list>**; Є

→ < add_statement> <data type> <variable> , <variable> **<list>**; Є

→ < add_statement> <data type> **<variable>** , **<variable>** Є; Є

→ < add_statement> **<data type>** <variable> , 'birthdate Є; Є

→ **<add_statement>** text 'name , 'birthdate Є; Є

→ **<declaration statement>**; text 'name , 'birthdate Є; Є

→ <data type> **<declaration type>**; text 'name , 'birthdate Є; Є

→ <data type> <assignment> **<list>**; text 'name , 'birthdate Є; Є

→ <data type> <assignment> , <assignment> **<list>**; text 'name , 'birthdate Є; Є

→ <data type> <assignment> , **<assignment>** Є; text 'name , 'birthdate Є; Є

→ <data type> <assignment> , <variable> := **<literals>** Є; text 'name , 'birthdate Є; Є

→ <data type> <assignment> , <variable> := **<constant>** Є; text 'name , 'birthdate Є; Є

→ <data type> <assignment> , <variable> := **<decimal literal>** Є; text 'name 'birthdate Є; Є

→ <data type> <assignment> , **<variable>** := 25.50 Є; text 'name , 'birthdate Є; Є

→ <data type> **<assignment>** , 'price := 25.50 Є; text 'name , 'birthdate Є; Є

→ <data type> <variable> := **<literals>** , 'price := 25.50 Є; text 'name , 'birthdate Є; Є

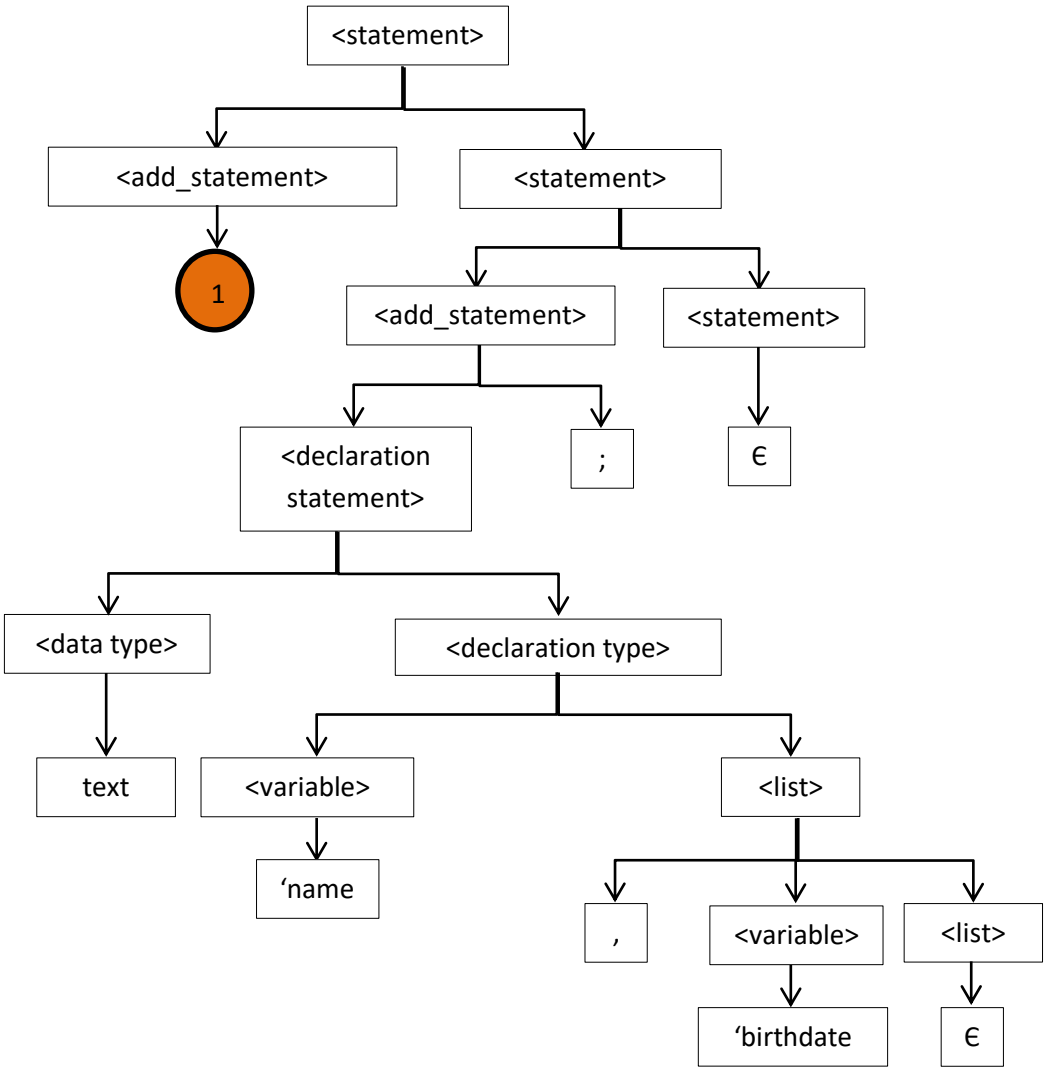
→ <data type> <variable> := **<constant>** , 'price := 25.50 Є; text 'name , 'birthdate Є; Є

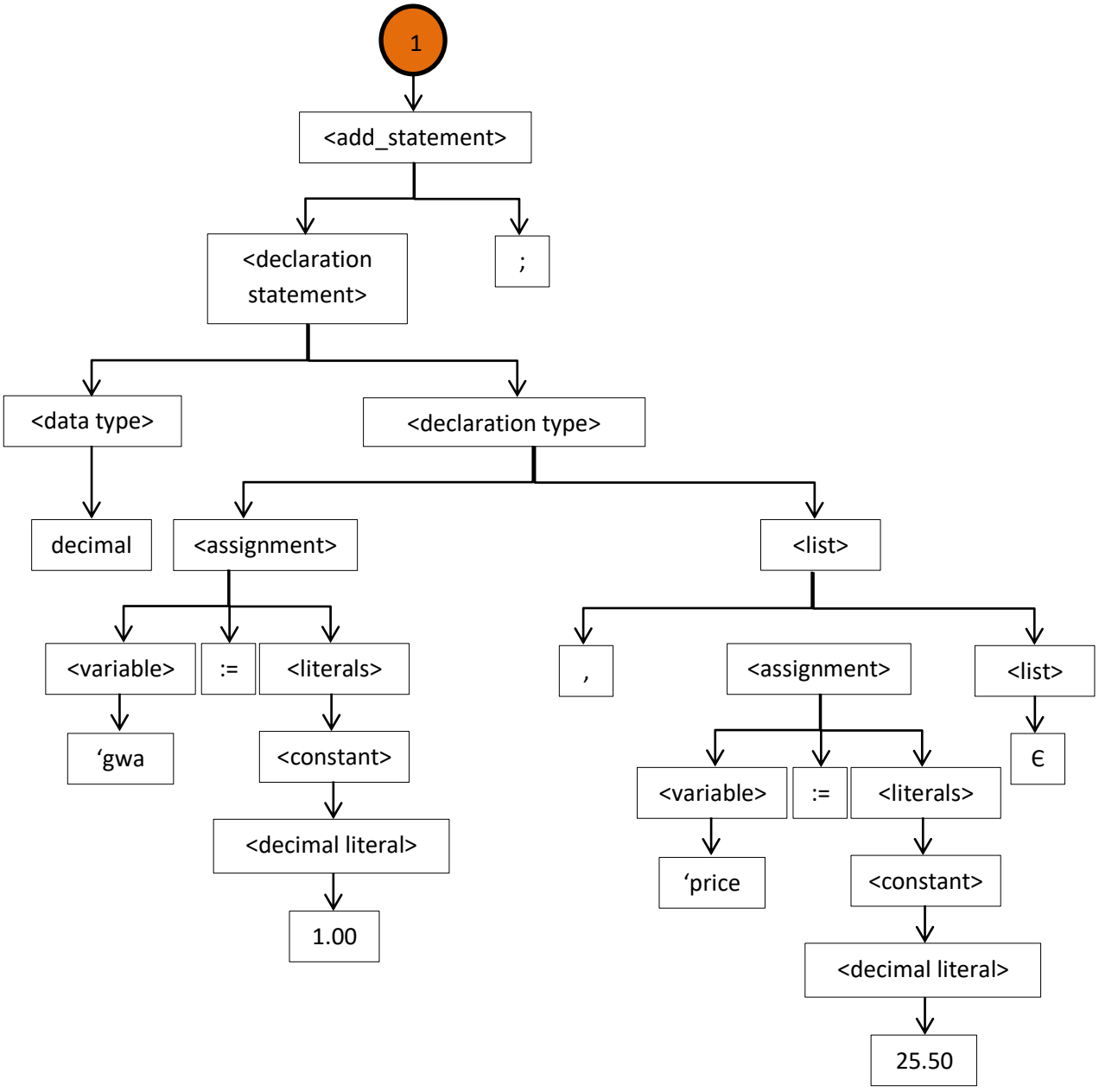
→ <data type> <variable> := **<decimal literal>** , 'price := 25.50 Є; text 'name , 'birthdate Є; Є

→ <data type> **<variable>** := 1.00 , 'price := 25.50 Є; text 'name , 'birthdate Є; Є

→ **<data type>** 'gwa := 1.00 , 'price := 25.50 Є; text 'name , 'birthdate Є; Є

→ decimal 'gwa := 1.00 , 'price := 25.50 Є; text 'name , 'birthdate Є; Є


<statement>
<add_statement>
<statement>
1
<add_statement>
<statement>
<declaration statement>
;
Є
<data type>
<declaration type>
text
<variable>
<list>
‘name
,
<variable>
<list>
‘birthdate
Є


1
<add_statement>
<declaration statement>
;
<data type>
<declaration type>
decimal
<assignment>
<list>
<variable>
:=
<literals>
,
<assignment>
<list>
‘gwa
<constant>
<variable>
:=
<literals>
Є
<decimal literal>
‘price
<constant>
1.00
<decimal literal>
25.50

Example 4:

boolean ‘option := ‘x, ‘bool;

G(V) = {<statement>, <add_statement>, <declaration statement>, <data type>, <declaration type>,<assignment>,<list>,<variable>}

G(T) = {boolean, ‘option, ‘bool,:=,;, , ,Є}

G(S) = {<statement>}

G(P)

<statement> ::= <add_statement> <statement>| Є

<add_statement> ::= <declaration statement>;

<declaration statement> ::= <data type> <declaration type>

<data type> ::= boolean

<declaration type> ::= <assignment> <list>

<list> ::= , <variable> <list>| Є

<assignment> ::= <variable> := <variable>

<variable> ::= ‘option | ‘bool

LEFTMOST DERIVATION

<statement> → **<add_statement>** <statement>

→**<declaration statement>;** <statement>

→**<data type>** <declaration type>; <statement>

→boolean **<declaration type>** ;<statement>

→boolean **<assignment>** <list>; <statement>

→boolean **<variable>** := <variable> <list>; <statement>

→boolean ‘option := **<variable>** <list>; <statement>

→boolean ‘option := ‘x **<list>**; <statement>

→boolean ‘option := ‘x , **<variable>** <list>; <statement>

→boolean ‘option := ‘x , ‘bool **<list>**; <statement>

→boolean ‘option := ‘x , ‘bool Є; **<statement>**

→boolean ‘option := ‘x ‘bool Є; Є

RIGHTMOST DERIVATION

<statement> → <add_statement> **<statement>**

→ **<add_statement>** Є

→**<declaration statement>**; Є

→<data type> **<declaration type>**; Є

→<data type> <assignment> **<list>**; Є

→<data type> <assignment> , <variable> **<list>**; Є

→<data type> <assignment> , **<variable>** Є; Є

→<data type> **<assignment>** , 'bool Є; Є

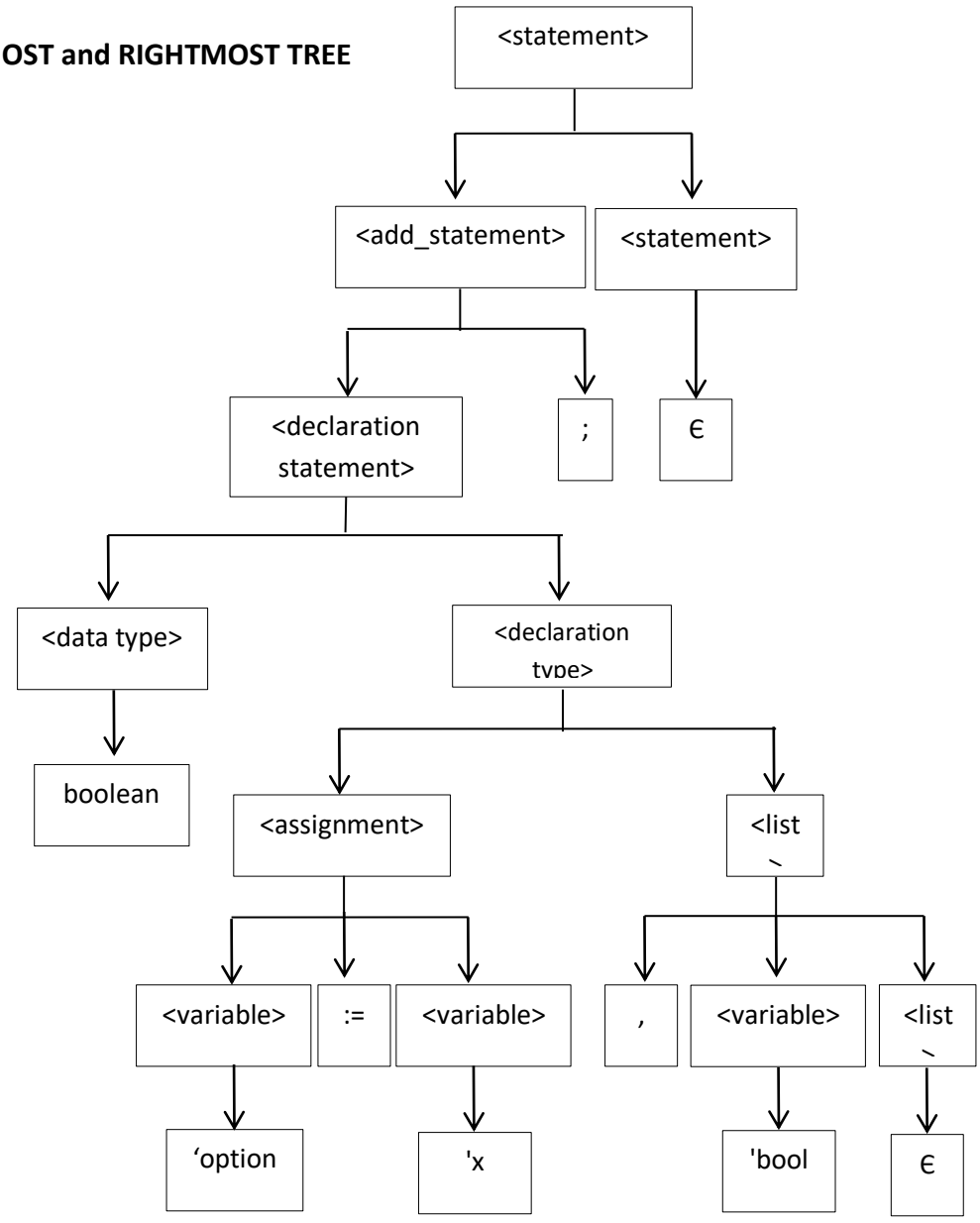
→<data type> <variable> := **<variable>** , 'bool Є; Є

→<data type> **<variable>** := 'x , 'bool Є; Є

→**<data type>** 'option := 'x , 'bool Є; Є

→boolean 'option := 'x, 'bool Є; Є

LEFTMOST and RIGHTMOST TREE



Example 5: integral ‘age := 2 + ’current;

G(V) = {<statement>, <add_statement>, <declaration statement>, <data type>, <declaration type>,<assignment>,<list>,<variable>, <literals>, <constant>, <integral literal>, <arithmetic expression>}

G(T) = {integral, 2, ‘age, ‘current,:=,;, +, Є}

G(S) = {<statement>}

G(P)

<statement> ::= <add_statement> <statement>| Є

<add_statement> ::= <declaration statement>;

<declaration statement> ::= <data type> <declaration type>

<data type> ::= integral

<declaration type> ::= <assignment> <list>

<list> ::= Є

<assignment> ::= <variable> := <arithmetic expression >

<literals> ::= <constant>

<constant> ::= <integral literal>

<integral literal> ::= 2

<variable> ::= ‘age | ‘current

LEFTMOST DERIVATION

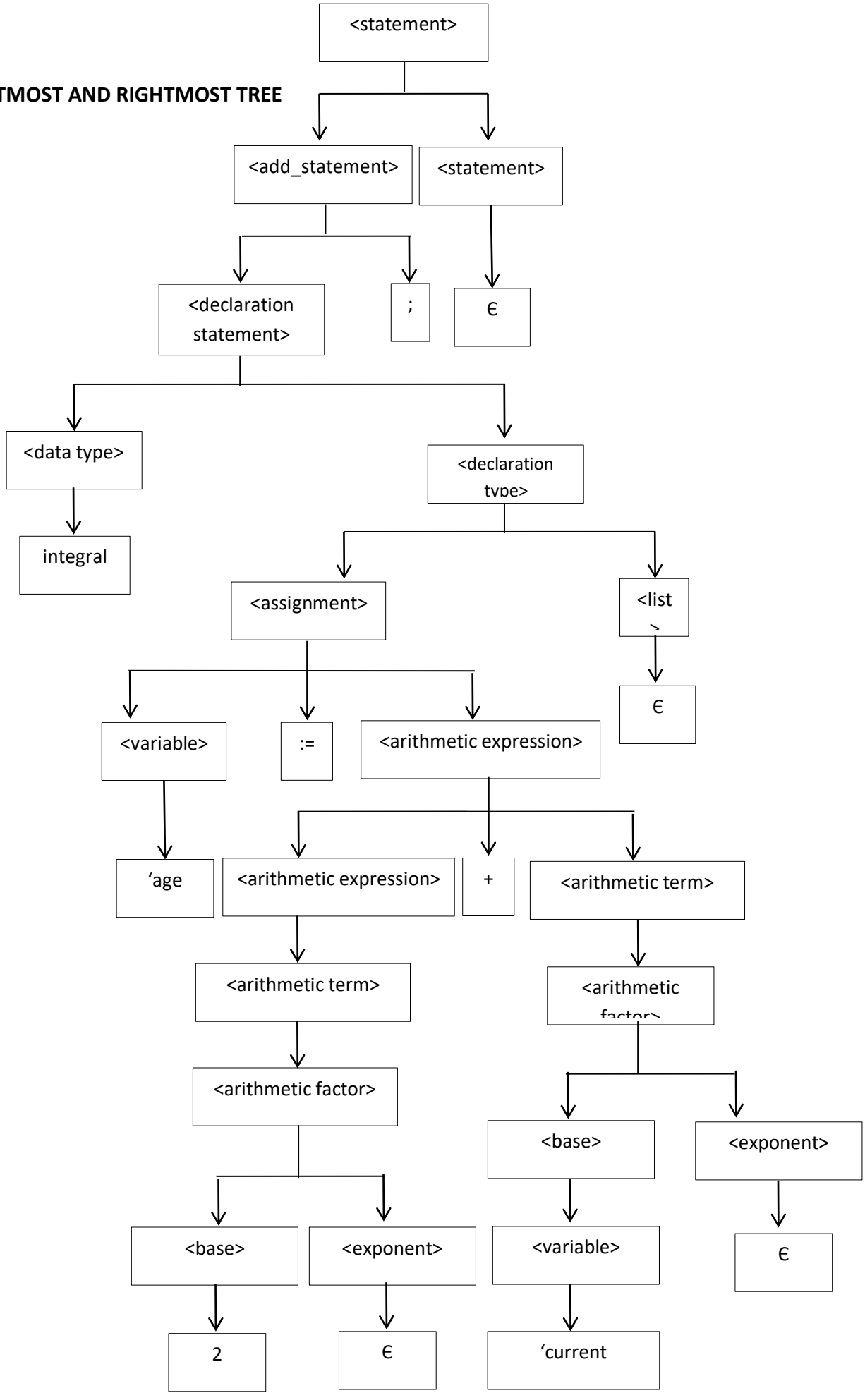
<statement> →**<add_statement>** <statement>
→**<declaration statement>;** <statement>
→**<data type>** <declaration type>; <statement>
→integral **<declaration type>;** <statement>
→integral **<assignment>** <list>; <statement>
→integral **<variable>** := <arithmetic expression> <list>; <statement>
→integral ‘age ::= **<arithmetic expression>**<list>; <statement>
→integral ‘age ::= **<arithmetic expression>** + <arithmetic term><list>; <statement>
→integral ‘age ::= **<arithmetic expression>** + <arithmetic term><list>; <statement>
→integral ‘age ::= **<arithmetic term>** + <arithmetic term><list>; <statement>
→integral ‘age ::= **<arithmetic factor>** + <arithmetic term><list>; <statement>
→integral ‘age ::= **<base>** <exponent> + <arithmetic term><list>; <statement>
→integral ‘age ::= **<constant>** <exponent> + <arithmetic term><list>; <statement>
→integral ‘age ::= **<integral literal>** <exponent> + <arithmetic term><list>;

<statement>
→integral ‘age ::= 2 **<exponent>** + <arithmetic term><list>; <statement>
→integral ‘age ::= 2 Є + **<arithmetic term>**<list>; <statement>
→integral ‘age ::= 2 Є + **<arithmetic factor>**<list>; <statement>
→integral ‘age ::= 2 Є + **<base>** <exponent><list>; <statement>
→integral ‘age ::= 2 Є + **<variable>** <exponent><list>; <statement>
→integral ‘age ::= 2 Є + ‘current **<exponent>**<list>; <statement>
→integral ‘age ::= 2 Є + ‘current Є **<list>**; <statement>
→integral ‘age ::= 2 Є + ‘current Є Є; **<statement>**
→integral ‘age ::= 2 Є + ‘current Є Є; Є

RIGHTMOST DERIVATION

<statement> →<add_statement> **<statement>**
→**<add_statement>** Є
→**<declaration statement>**; Є
→<data type> **<declaration type>**; Є
→<data type> <assignment> **<list>**; Є
→<data type> **<assignment>** Є; Є
→<data type> <variable> := **<arithmetic expression>** Є; Є
→<data type> <variable> := <arithmetic expression> + **<arithmetic term>** Є; Є
→<data type> <variable> := <arithmetic expression> + **<arithmetic factor>** Є; Є
→<data type> <variable> := <arithmetic expression> + <base> **<exponent>** Є; Є
→<data type> <variable> := <arithmetic expression> + **<base>** Є Є; Є
→<data type> <variable> := <arithmetic expression> + **<variable>** Є Є; Є
→<data type> <variable> := **<arithmetic expression>** + ‘current Є Є; Є
→<data type> <variable> := **<arithmetic term>** + ‘current Є Є; Є
→<data type> <variable> := **<arithmetic factor>** + ‘current Є Є; Є
→<data type> <variable> := <base> **<exponent>** + ‘current Є Є; Є
→<data type> <variable> := **<base>** Є + ‘current Є Є; Є
→<data type> <variable> := **<consant>** Є + ‘current Є Є; Є
→<data type> <variable> := **<integral literal>** Є + ‘current Є Є; Є
→<data type> **<variable>** := 2 Є + ‘current Є Є; Є
→**<data type>** ‘age := 2 Є + ‘current Є Є; Є
→integral ‘age := 2 Є + ‘current Є Є; Є

LEFTMOST AND RIGHTMOST TREE



Example 6:

boolean ‘bool := false;

G(V) = {<statement>, <add_statement>, <declaration statement>, <data type>, <declaration type>,<assignment>,<list>,<variable>, <literals>, <boolean literal>}

G(T) = {boolean, false, ‘bool,:=,;, Є}

G(S) = {<statement>}

G(P)

<statement> ::= <add_statement> <statement>| Є

<add_statement> ::= <declaration statement>;

<declaration statement> ::= <data type> <declaration type>

<data type> ::= boolean

<declaration type> ::= <assignment> <list>

<list> ::= Є

<assignment> ::= <variable> := <literals>

<literals> ::= <boolean literal>

<boolean literal> ::= false

<variable> ::= ‘bool

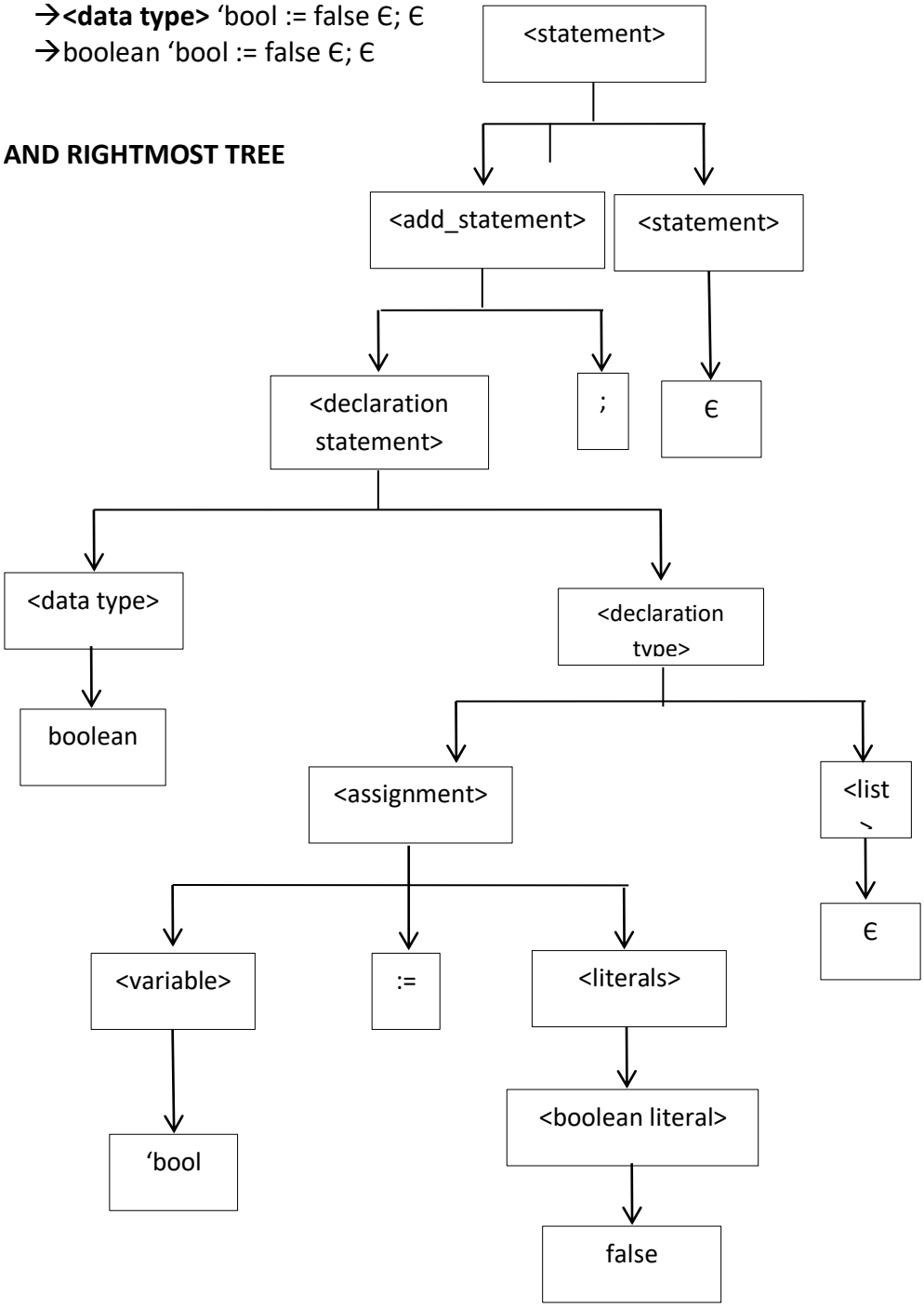
LEFTMOST DERIVATION

<statement> →**<add_statement>** <statement>
→**<declaration statement>**; <statement>
→**<data type>** <declaration type>; <statement>
→boolean **<declaration type>**; <statement>
→boolean **<assignment>**<list>; <statement>
→boolean **<variable>** := <literals><list>; <statement>
→boolean ‘bool := **<literals>**<list>; <statement>
→boolean ‘bool := **<boolean literal>** <list>; <statement>
→boolean ‘bool := false **<list>**; <statement>
→boolean ‘bool := false Є; **<statement>**
→boolean ‘bool := false Є; Є

RIGHTMOST DERIVATION

<statement> →<add_statement> **<statement>**
→**<add_statement>** Є
→**<declaration statement>**; Є
→<data type> **<declaration type>**; Є
→<data type> <assignment> **<list>**; Є
→<data type> **<assignment>** Є; Є
→<data type> <variable> := **<literals>** Є; Є
→<data type> <variable> := **<boolean literal>** Є; Є
→<data type> **<variable>** := false Є; Є
→**<data type>** 'bool := false Є; Є
→boolean 'bool := false Є; Є

LEFTMOST AND RIGHTMOST TREE



Example 7:

```
outdis ("Display");
```

G(V) = {<statement>, <add_statement>, <i/o statement>,<output> ,<print> ,<text literal>, <variable>, <literals>, <boolean literal>}
G(T) = {outdis, (,),",",;, Є}
G(S) = {<statement>}
G(P)

<statement> ::= <add_statement> <statement>| Є
<add_statement> ::= <i/o statement>
<i/o statement> ::= <output>
<output> ::= outdis (<print>);
<print> ::= "<text literal>"<print> | Є
<text literal> ::= Display

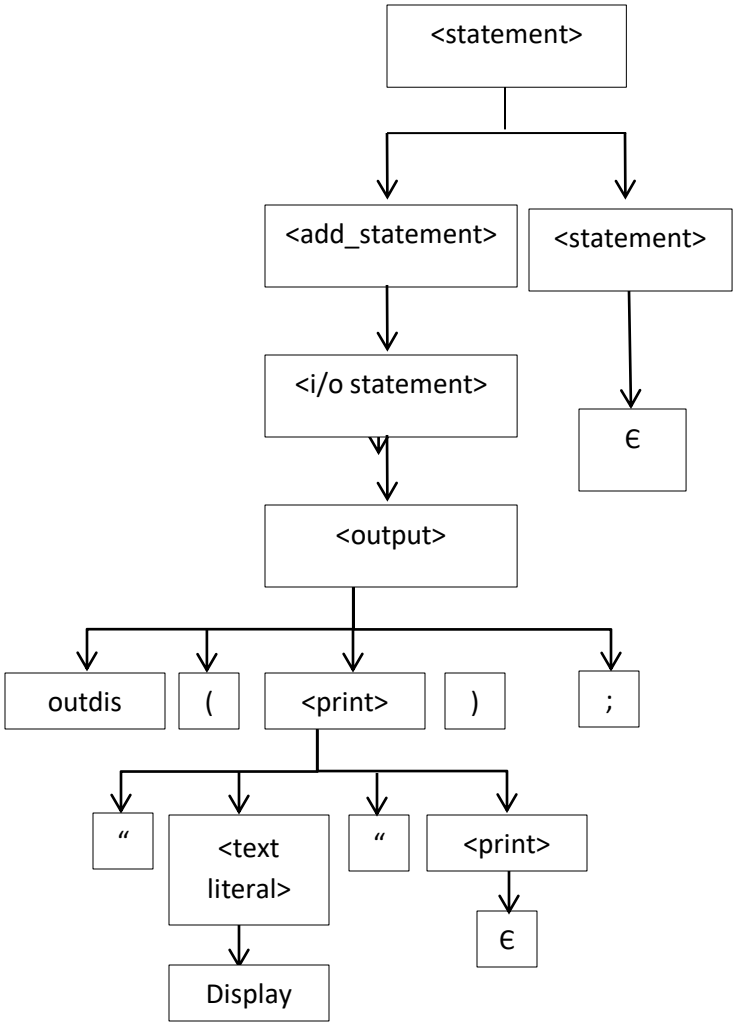
LEFTMOST DERIVATION

<statement> →<add_statement> <statement>
→<i/o statement> <statement>
→<output> <statement>
→outdis (<print>); <statement>
→outdis ("<text literal>"<print>); <statement>
→outdis ("Display" <print>); <statement>
→outdis ("Display" Є); <statement>
→outdis ("Display" Є); Є

RIGHTMOST DERIVATION

<statement> →<add_statement> **<statement>**
→**<add_statement>** Є
→**<i/o statement>** Є
→**<output>** Є
→ outdis (**<print>**); Є
→ outdis ("<text literal>"**<print>**); Є
→ outdis ("**<text literal>**"Є); Є
→ outdis ("Display"Є); Є

PARSE TREE (LEFTMOST & RIGHTMOST)



Example 8: outdis (“Age: ”, 13);

G(V) = {<i/o statement>, <output>, <print>, <text literal>, <constant>, <integral literal>}
G(T) = {outdis, (,), ;, “, ”, Age, 13, Є}
G(S) ={<i/o statement>}
G(P)

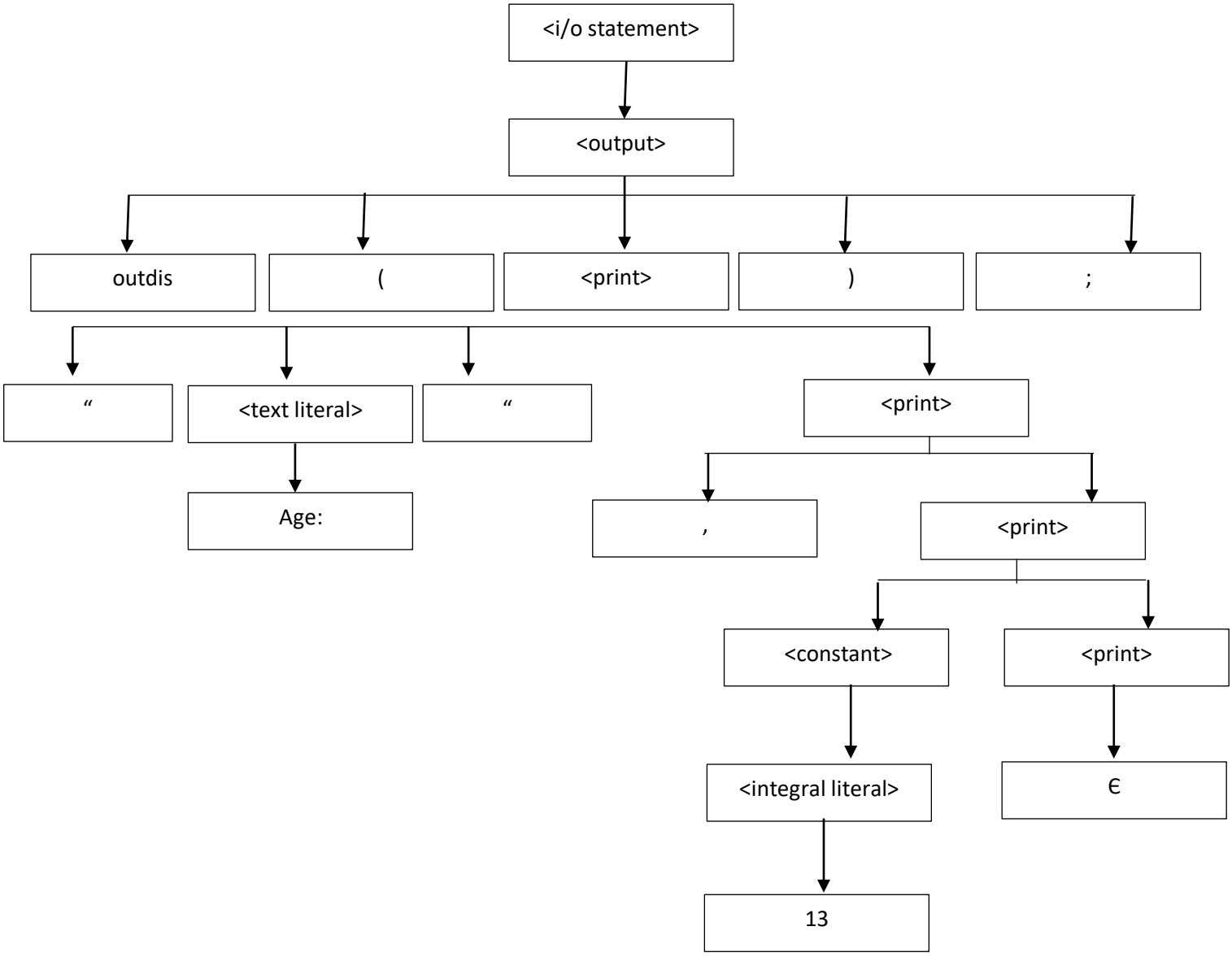
<i/o statement> ::= <output>
<output> ::= outdis (<print>);
<print> ::= “<text literal>”<print> | , <print> | <constant> <print> | Є
<text literal> ::= Age:
<constant> ::= <integral literal>
<integral literal ::= 13

LEFTMOST DERIVATION

<i/o statement>→ **<output>**
→outdis(**<print>**);
→outdis(”**<text literal>**”<print>);
→outdis(”Age:”**<print>**);
→outdis(“Age:”, **<print>**);
→outdis(“Age:”, **<constant>**<print>);
→outdis(“Age:”, **<integral literal>**<print>);
→outdis(“Age:”, 13**<print>**);
→outdis(“Age:”, 13 Є);

RIGHTMOST DERIVATION

<i/o statement>→ **<output>**
→outdis(**<print>**);
→outdis(”<text literal>”**<print>**);
→outdis(”<text literal”,**<print>**);
→outdis(“<text literal>”, <constant>**<print>**);
→outdis(“<text literal>”, **<constant>** Є);
→outdis(“<text literal>”, **<integral literal>** Є);
→outdis(“**<text literal>**”, 13 Є);
→outdis(“Age:”, 13 Є);


<i/o statement>
<output>
outdis
(
<print>
)
;
“
<text literal>
“
Age:
<print>
,
<print>
<constant>
<print>
<integral literal>
Є
13

Example 9:

```
outdis (“Birthday: ”, ‘birthdate);
```

G(V) = {<i/o statement>, <output>, <print>, <text literal>, <variable>}
G(T) = {outdis, Birthday, :, ‘birthdate, Є, “, ”, ;, (,)}
G(S) = {<i/o statement>}
G(P) =

<i/o statement> ::= <output>
<output> ::= outdis (<print>);
<print> ::= “<text literal>”<print> | , <print> | <variable> <print> | Є
<text literal> ::= Birthday:
<variable> ::= ‘birthdate

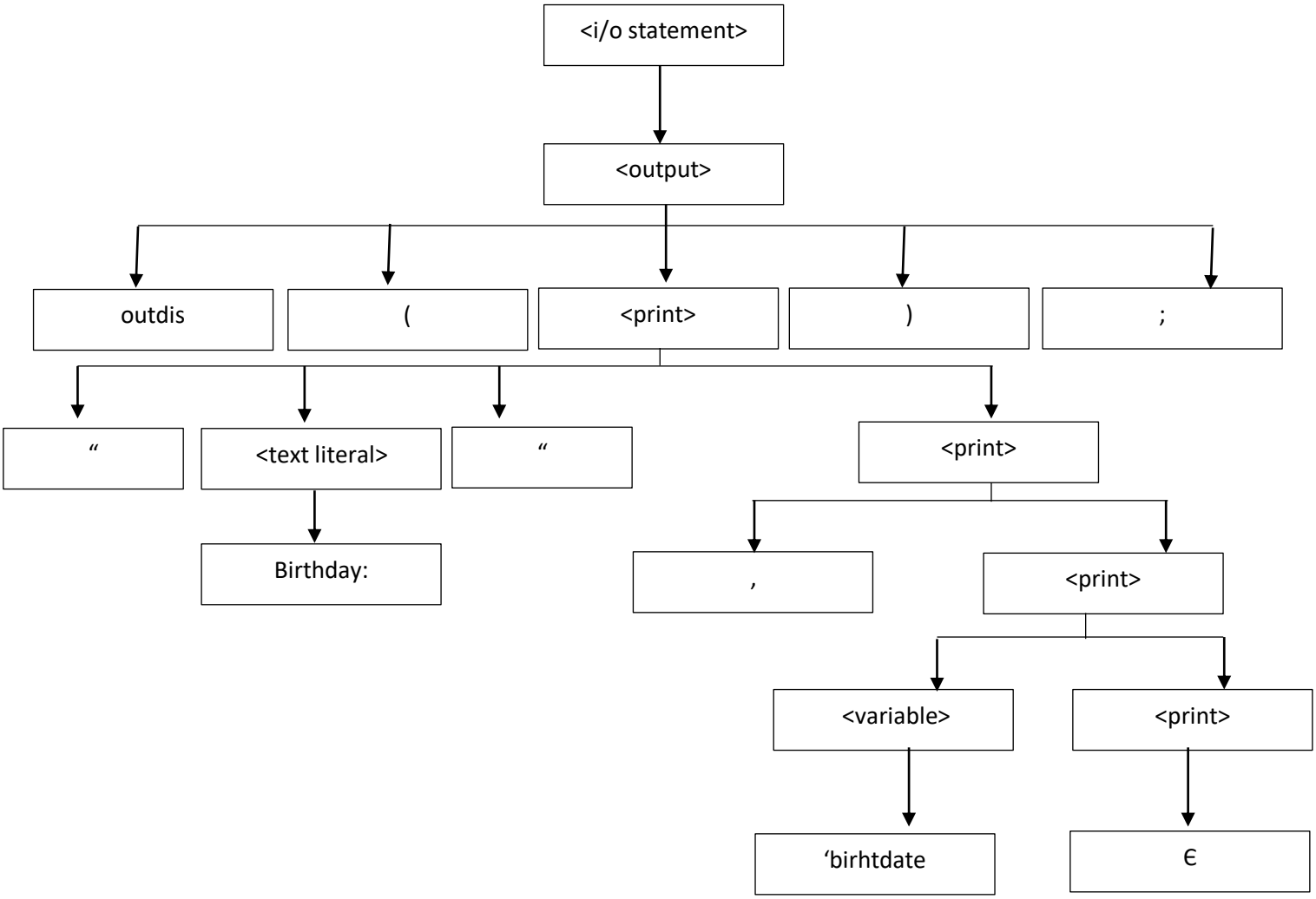
LEFTMOST DERIVATION

<i/o statement>→ **<output>**
→outdis(**<print>**);
→outdis(”**<text literal>**”<print>);
→outdis(”Birthday: ”**<print>**);
→outdis(“Birthday: ”, **<print>**);
→outdis(“Birthday: ”, **<variable>**<print>);
→outdis(“Birthday: ”, ‘birthdate **<print>**);
→outdis(“Birthday: ”, ‘birthdate Є);

RIGHTMOST DERIVATION

<i/o statement>→ **<output>**
→outdis(**<print>**);
→outdis(”<text literal>”**<print>**);
→outdis(”<text literal>”, **<print>**);
→outdis(“<text literal>”, <variable>**<print>**);
→outdis(“<text literal>”, **<variable>** Є);
→outdis(“**<text literal>**”, ‘birthdate Є);
→outdis(“Birthday: ”, ‘birthdate Є);

LEFTMOST and RIGHTMOST TREE



Example 10:

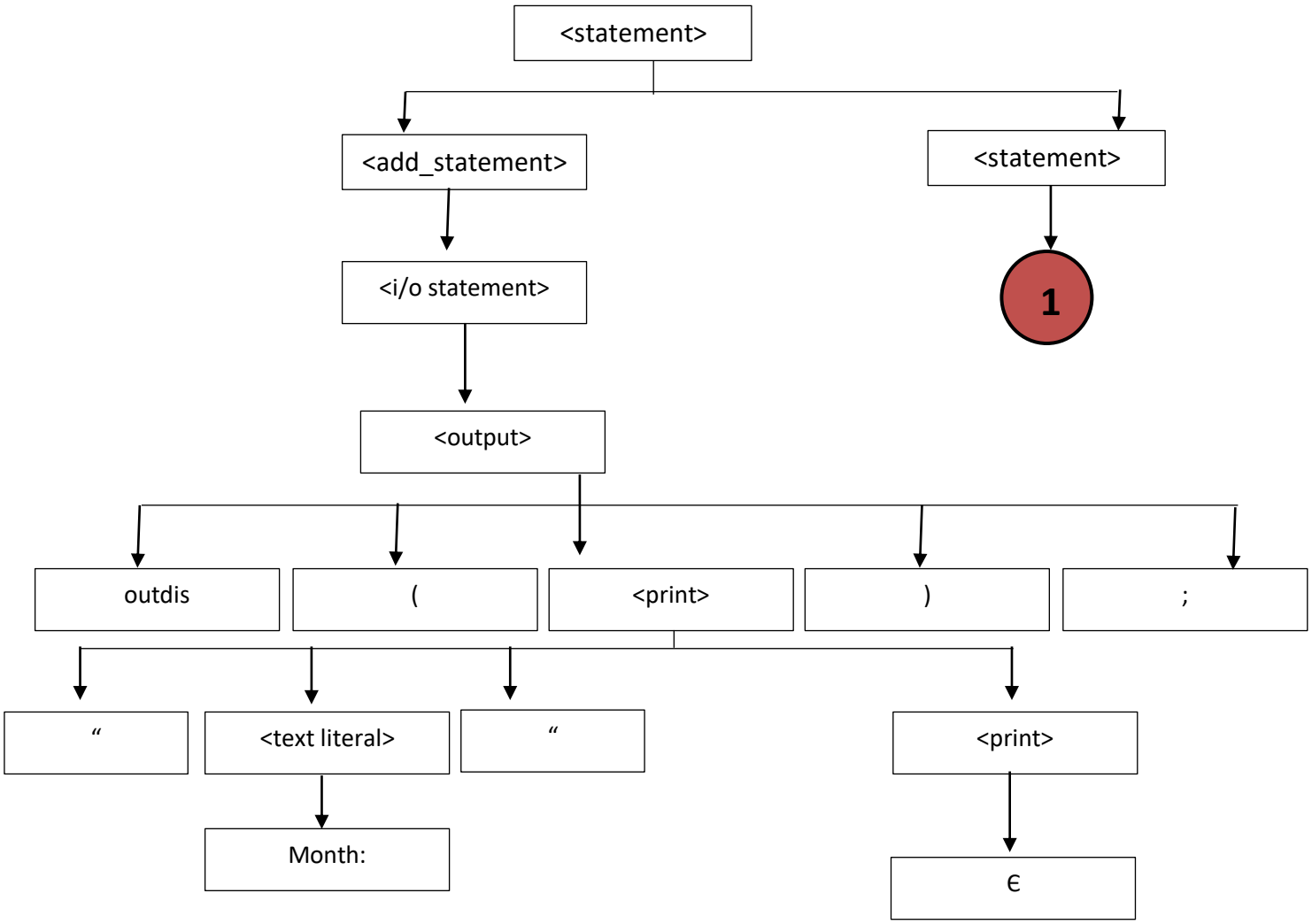
```
outdis (“Month: ”);
inscan (‘month);
```

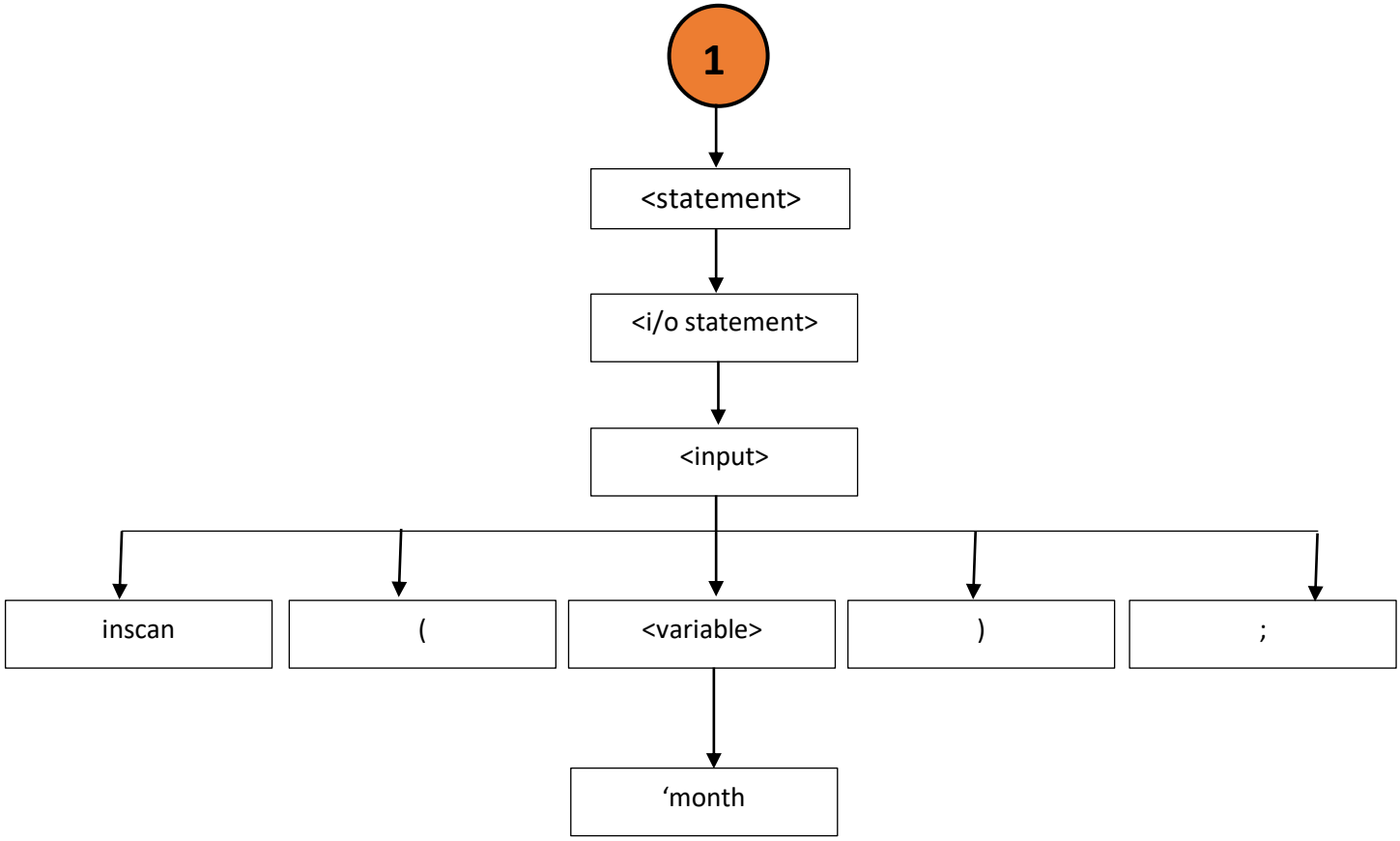
G(V) = {<statement>, <add_statement>, <i/o statement>, <output>, <print>, <text literal>, <input>, <variable>, }
G(T) = {outdis, inscan, Є, ‘month, Month, :, “, ”, (,), ;}
G(S) = {<statement>}
G(P) =

<i/o statement> ::= <output> | <input>
<output> ::= outdis (<print>);
<input> ::= inscan (<variable>);
<print> ::= “<text literal>”<print> | , <print> | <variable> <print> | Є
<text literal> ::= Month:
<variable> ::= ‘month

LEFTMOST DERIVATION

<statement>→ **<add_statement>** <statement>
→**<i/o statement>** <statement>
→**<output>** <statement>
→outdis(“**<print>**”); <statement>
→outdis(“**<text literal>**” <print>); <statement>
→outdis(“Month: ” **<print>**); <statement>
→outdis(“Month: ” Є); **<statement>**
→outdis(“Month: ” Є); **<i/o statement>**
→outdis(“Month: ” Є); **<input>**
→outdis(“Month: ” Є); inscan(**<variable>**);
→outdis(“Month: ” Є); inscan(‘month);


<statement>
<add_statement>
<statement>
<i/o statement>
1
<output>
outdis
(
<print>
)
;
“
<text literal>
“
<print>
Month:
Є


1
<statement>
<i/o statement>
<input>
inscan
(
<variable>
)
;
‘month

RIGHTMOST DERIVATION

<statement>→ <add_statement> **<statement>**

→<add statement> **<i/o statement>**

→<add_statement> **<input>**

→<add_statement> inscan(**<variable>**);

→**<add_statement>** inscan('month);

→**<i/o statement>** inscan('month);

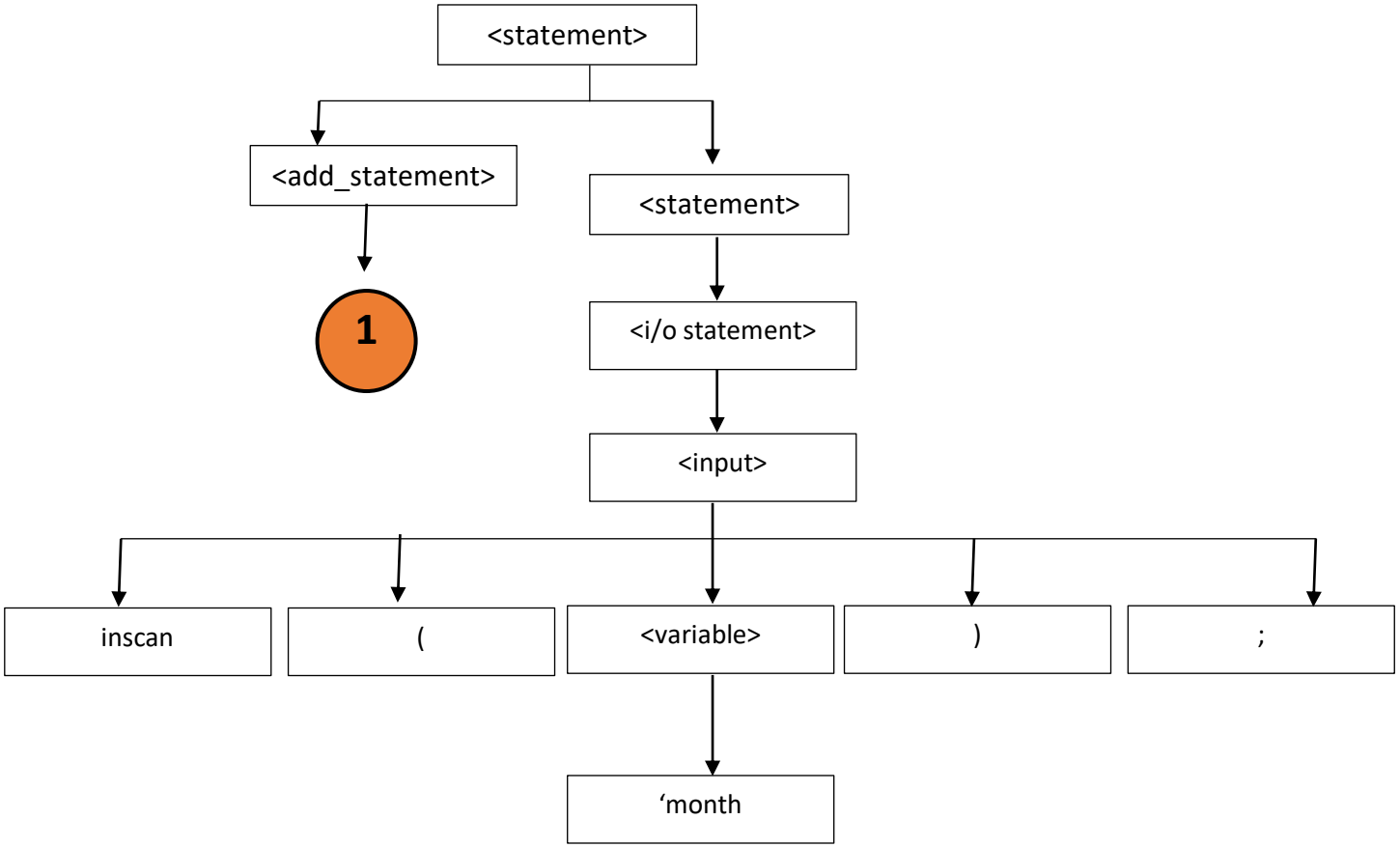
→**<output>** inscan('month);

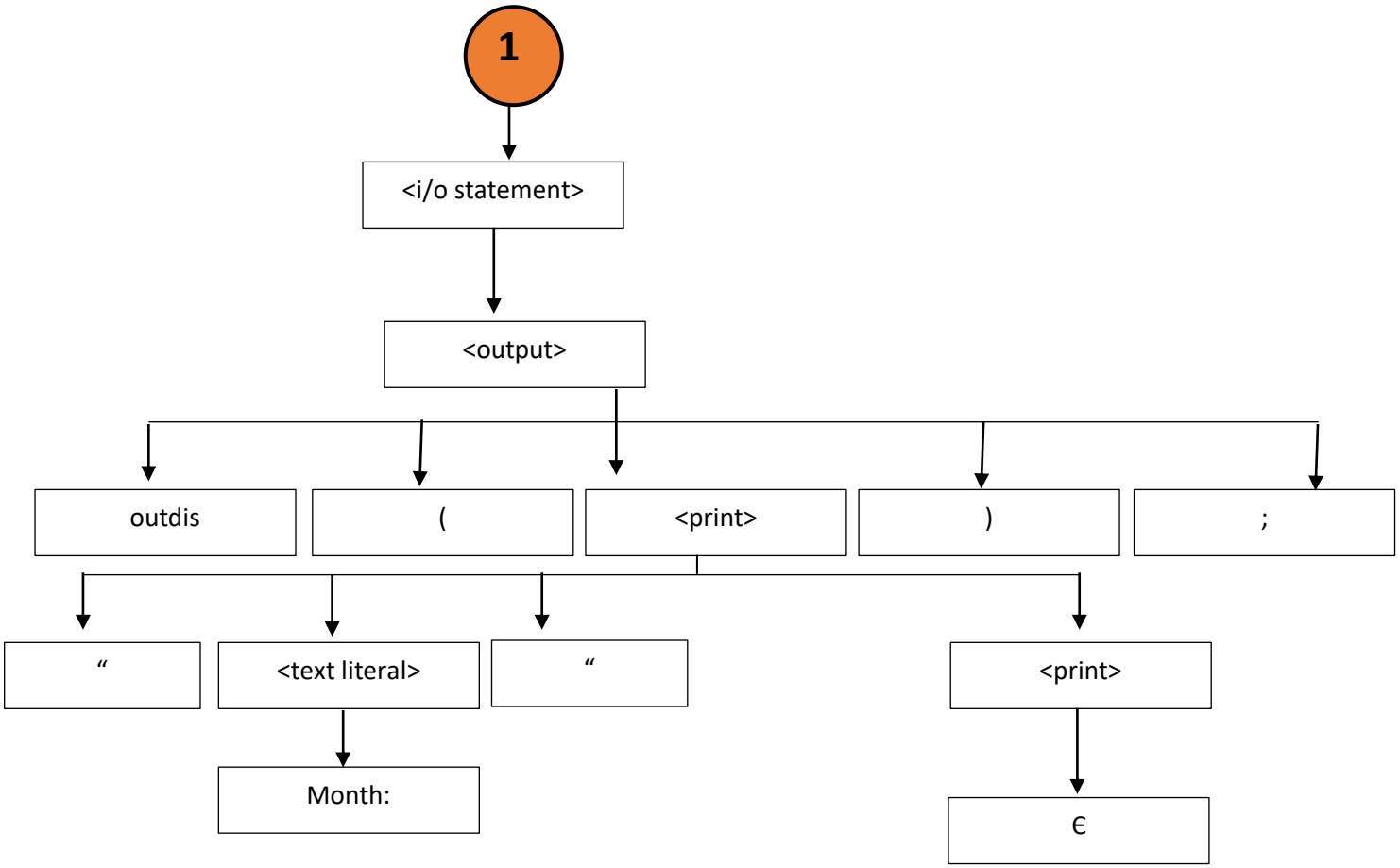
→outdis(**<print>**); inscan('month);

→outdis("<text literal>" **<print>**); inscan('month);

→outdis("**<text literal>**" Є); inscan('month);

→outdis("Month: " Є); inscan('month);




1
<i/o statement>
<output>
outdis
(
<print>
)
;
“
<text literal>
“
<print>
Month:
Є

Example 11:

```
forloop (‘x := 1; ‘x<5 ; ‘x := ’x+1) {
    integral ‘count := ‘counter; }
```

G(V) = {<loop statement>, <statement>, <forloop>, <assignment>, <variable>, <literals>, <constant>, <relational expression>, <relational term>, <arithmetic expressions>, <arithmetic term>, <arithmeric factor>, <base>, <exponent>, <data type>, <declaration type>, <list>}
G(T) ={forloop, ‘x, :=, <, ;, (,), +, 1, 5, Є, ‘count, ‘counter, integral, {, }}
G(S) = {<loop statement>}
G(P) =

<statement> ::= <add_statement> <statement>
<loop statement> ::= <forloop>
<forloop> ::= forloop (<assignment> ; <relational expression>; <arithmetic expression>){<statement>}
<assignment> ::= <variable> := <variable>
<relational expression>::= <relational term>
<relational term> ::= <relational term> < < arithmetic expression >
| <arithmetic expression>
<arithmetic expression> ::= <arithmetic expression> + <arithmetic term> | <arithmetic term>
<arithmetic term> ::= <arithmetic factor>
<arithmetic factor> ::= <base> <exponent>
<base> ::= <variable> | <constant>
<exponent> ::= Є
<data type> ::= integral
<declaration type> ::= <assignment> <list>
<literals> ::= <constant>
<variables> ::= ‘x, ‘count, ‘counter
<constant> ::= 1, 5
<list> ::= Є

LEFTMOST DERIVATION

<loop statement>→**<forloop>**

→forloop(**<assignment>** ; <relational expression> ; <arithmetic expression>){<statement>}

→forloop(**<variable>** := <literals> ; <relational expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := **<literals>** ; <relational expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := **<constant>** ; <relational expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<relational expression>** ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<relational term>** ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<relational term>** < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<arithmetic expression>** < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<arithmetic term>** < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<arithmetic factor>** < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<base>**<exponent> < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; **<variable>**<exponent> < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x**<exponent>** < <arithmetic expression> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < **<arithmetic expression>** ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < **<arithmetic term>** ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < **<arithmetic factor>** ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < **<base>**<exponent> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < **<constant>**<exponent> ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5**<exponent>** ; <arithmetic expression>){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; **<arithmetic expression>**){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; **<arithmetic expression>** + <arithmetic term>){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; **<arithmetic term>** + <arithmetic term>){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; **<arithmetic factor>** + <arithmetic term>){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; **<base>**<exponent> + <arithmetic term>){<statement>}

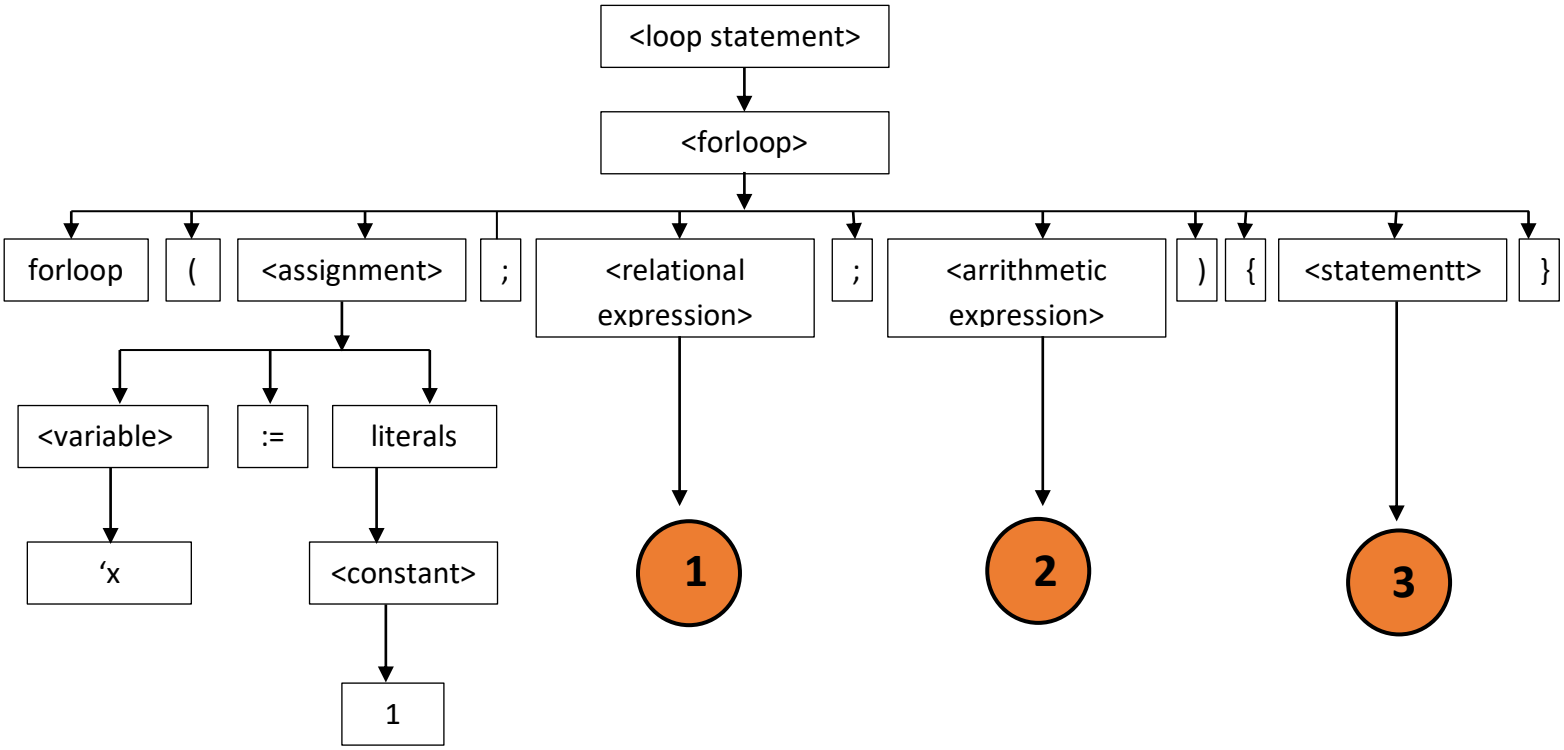
→forloop(‘x := 1 ; ‘x Є < 5 Є ; **<variable>**<exponent> + <arithmetic term>){<statement>}

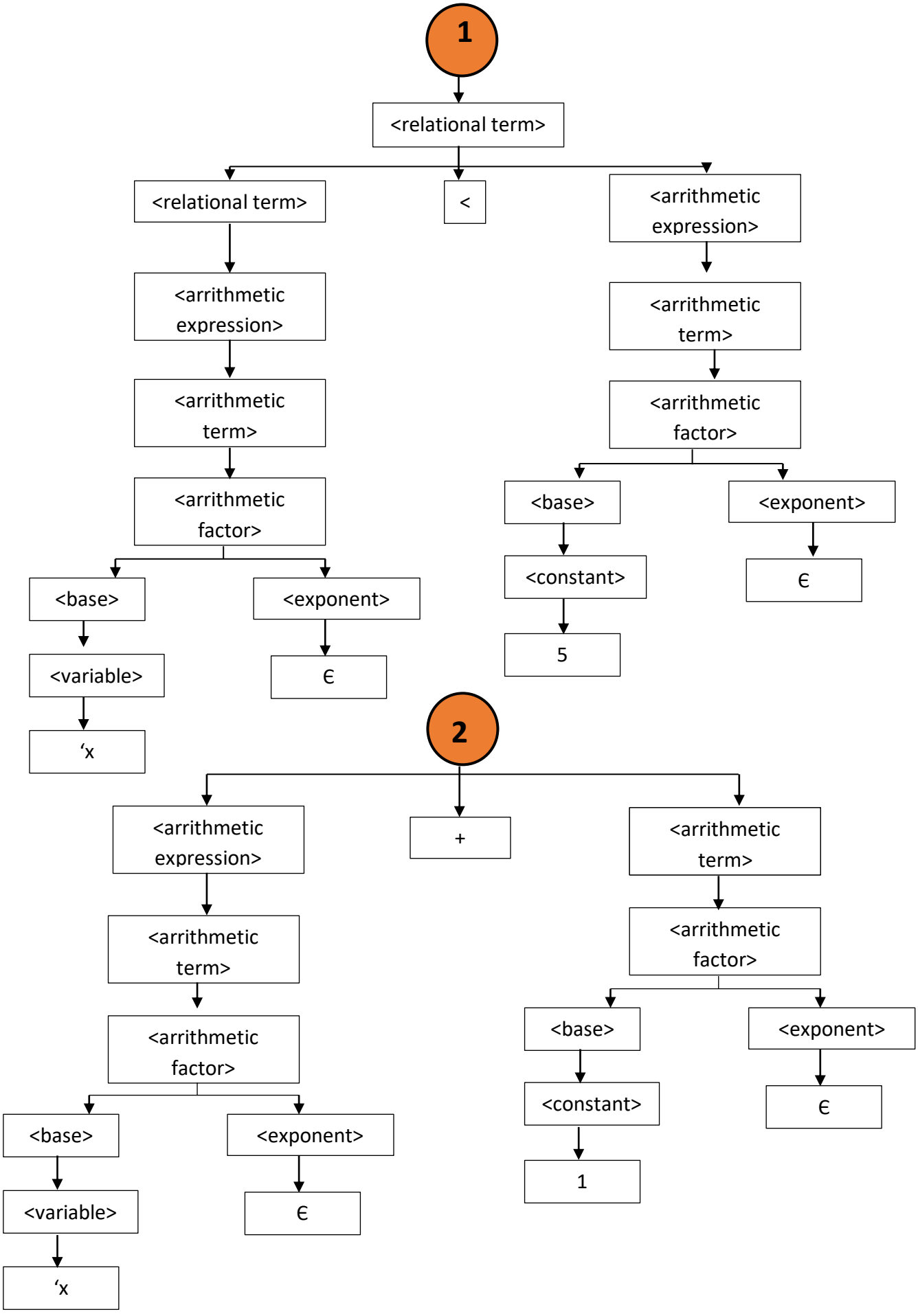
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x**<exponent>** + <arithmetic term>){<statement>}

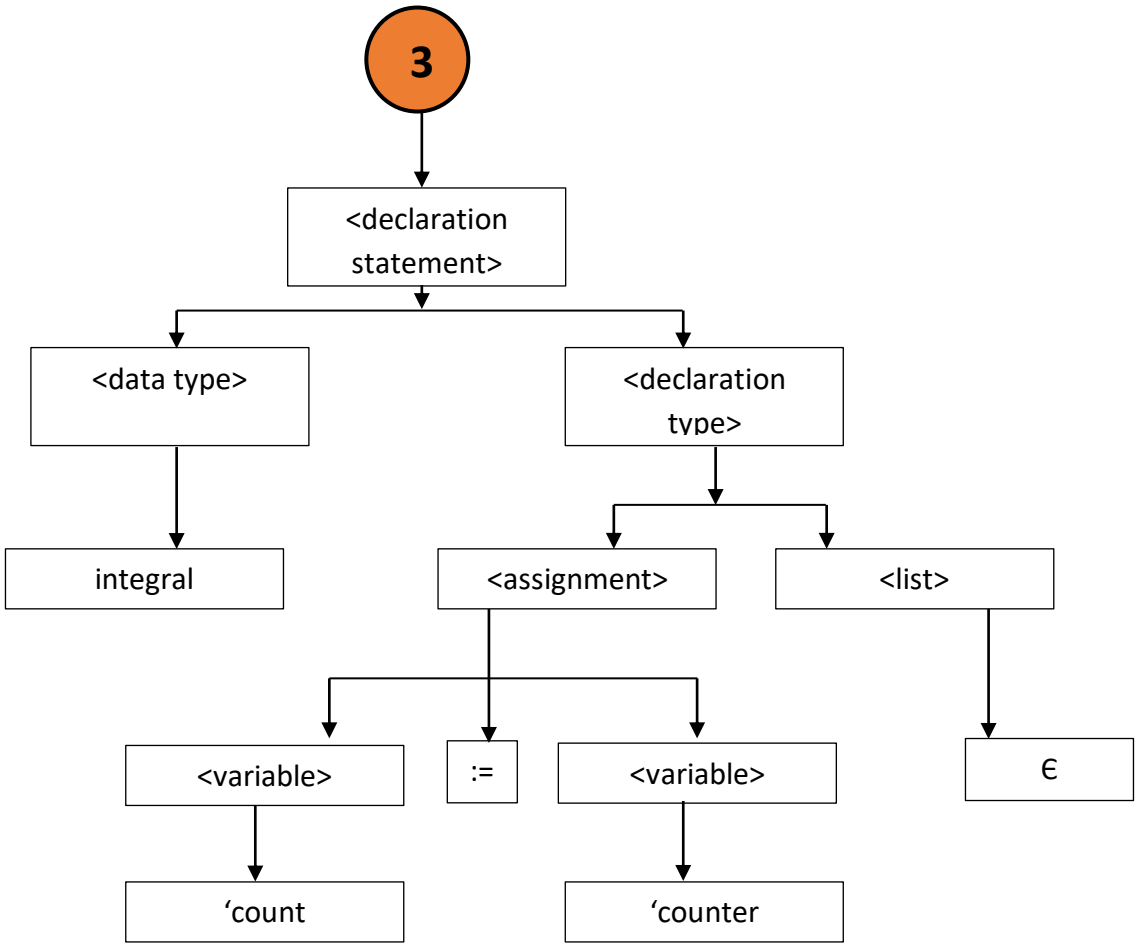
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + **<arithmetic term>**){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + **<arithmetic factor>**){<statement>}

→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + **<base>**<exponent>){<statement>}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + **<constant>**<exponent>){<statement>}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1**<exponent>**){<statement>}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){**<statement>**}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){**<declaration statement>**;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){**<data type>** <declaration type>;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){integral **<declaration type>**;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){integral **<assignment>** <list>;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){integral **<variable>** := <variable> <list>;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){integral ‘count := **<variable>** <list>;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){integral ‘count := ‘counter **<list>**;}
→forloop(‘x := 1 ; ‘x Є < 5 Є ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}




1
<relational term>
<relational term>
<
<arrithmetic expression>
<arrithmetic expression>
<arrithmetic term>
<arrithmetic term>
<arrithmetic factor>
<arrithmetic factor>
<base>
<exponent>
<base>
<exponent>
<constant>
Є
<variable>
Є
5
‘x
2
<arrithmetic expression>
+
<arrithmetic term>
<arrithmetic term>
<arrithmetic factor>
<arrithmetic factor>
<base>
<exponent>
<base>
<exponent>
<constant>
Є
<variable>
Є
1
‘x


3
<declaration statement>
<data type>
<declaration type>
integral
<assignment>
<list>
<variable>
:=
<variable>
ϵ
‘count
‘counter

RIGHTMOST DERIVATION

<loop statement>→**<forloop>**

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){**<statement>}**

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){**<declaration statement>**;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){<data type> **<declaration type>**;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){<data type><assignment> **<list>** ;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){<data type> **<assignment>** Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){<data type> <variable> := **<variable>** Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){<data type> **<variable>** := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression>){**<data type>** ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; **<arithmetic expression>)**{integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression> + **<arithmetic term>**){integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression> + **<arithmetic factor>**){integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression> + <base> **<exponent>**){integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression> + **<base>** Є){integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <arithmetic expression> + **<constant>** Є){integral ‘count := ‘counter Є;}

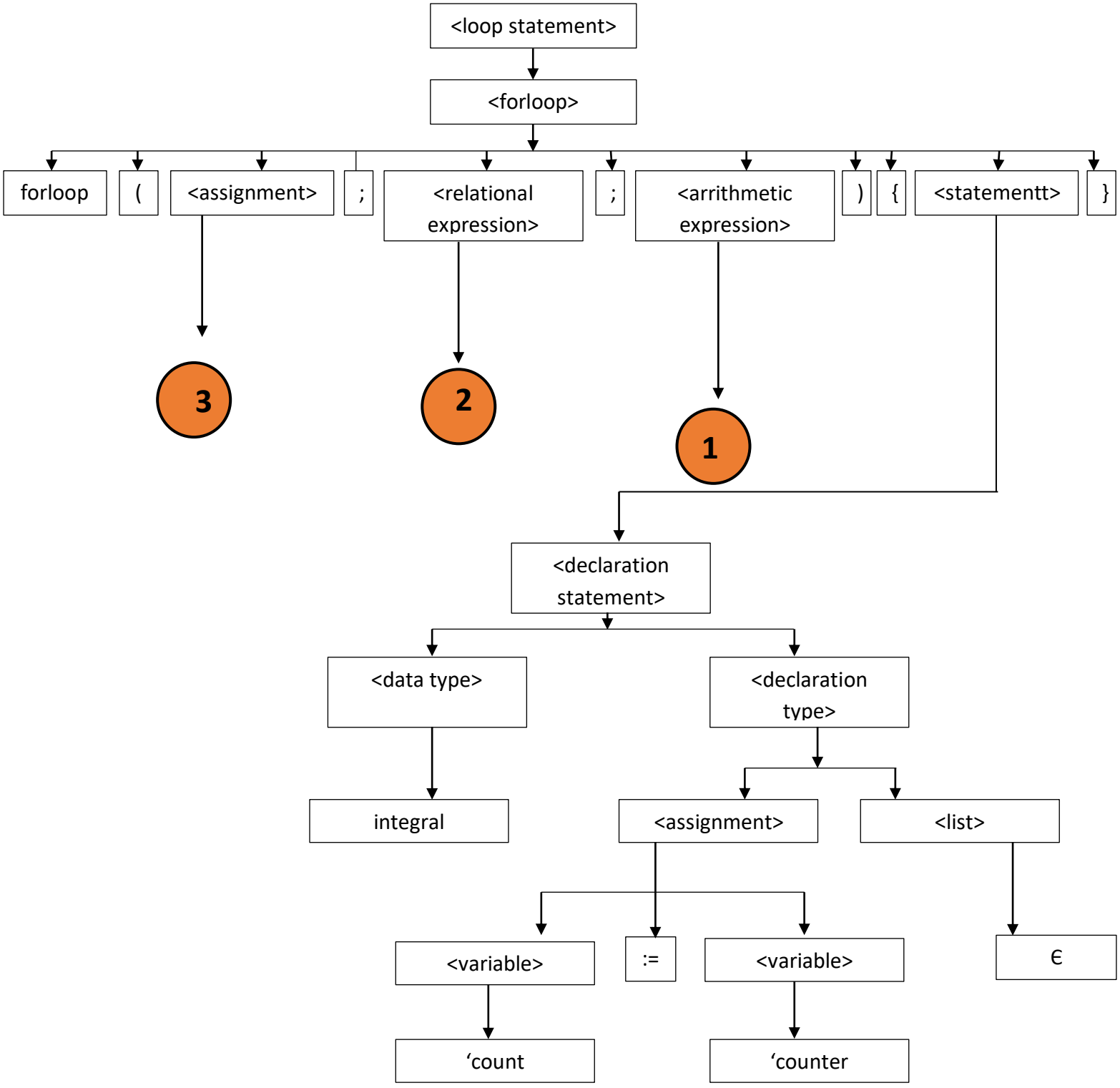
→forloop(<assignment> ; <relational expression> ; **<arithmetic expression>** + 1 Є){integral ‘count := ‘counter Є;}

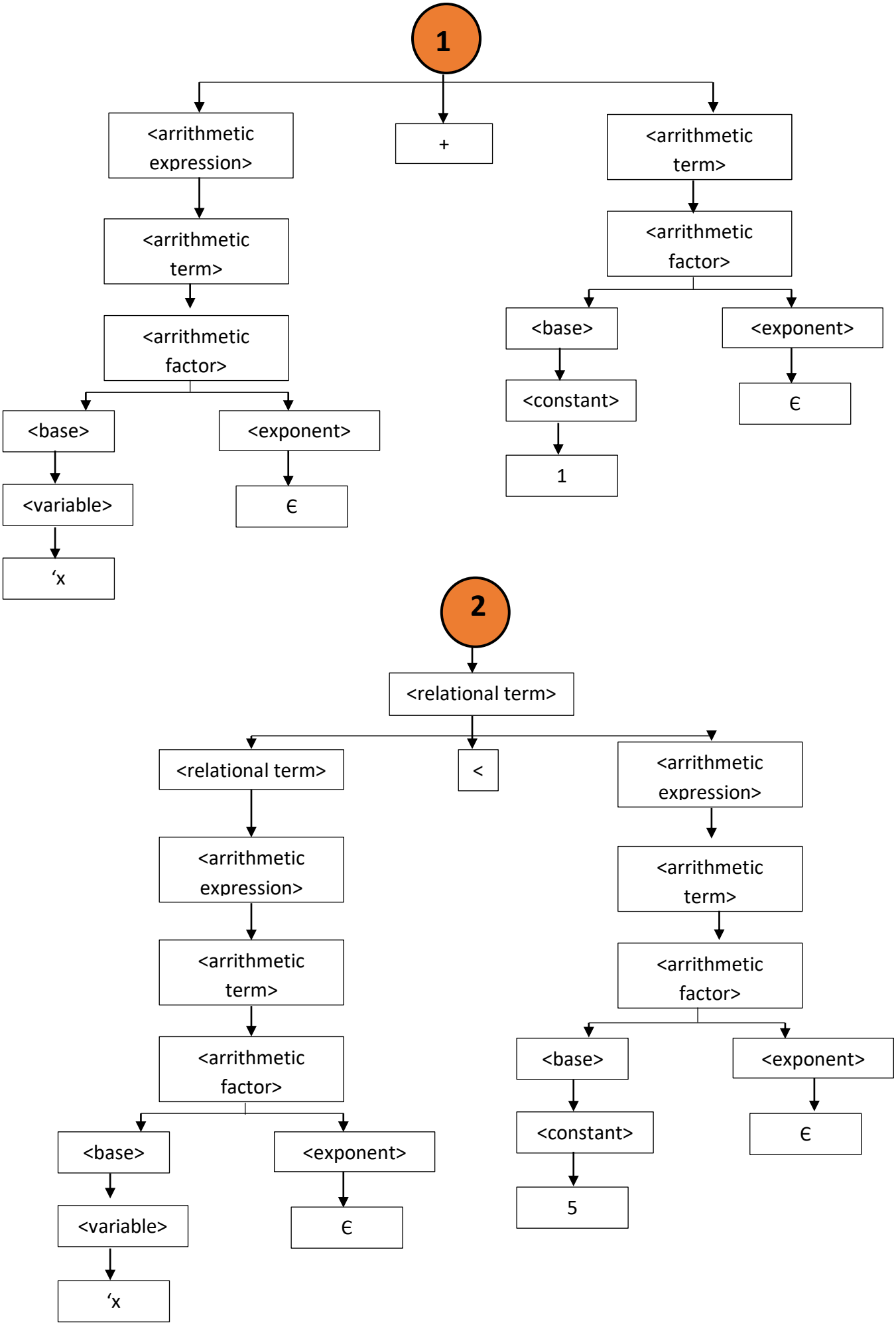
→forloop(<assignment> ; <relational expression> ; **<arithmetic term>** + 1 Є){integral ‘count := ‘counter Є;}

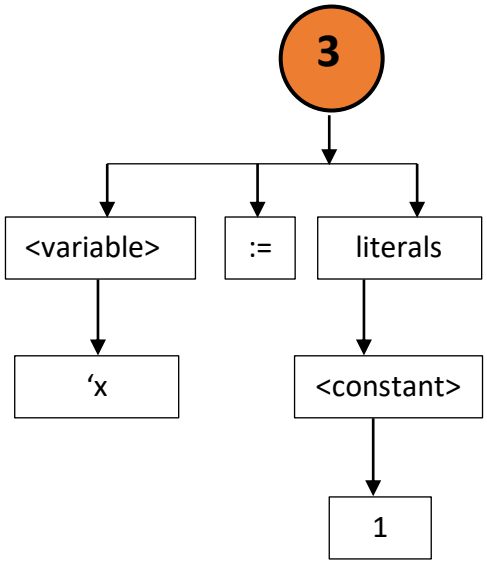
→forloop(<assignment> ; <relational expression> ; **<arithmetic factor>** + 1 Є){integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; <base>**<exponent>** + 1 Є){integral ‘count := ‘counter Є;}

→forloop(<assignment> ; <relational expression> ; **<base>** Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <relational expression> ; **<variable>** Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<relational expression>** ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<relational term>** ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <realtional term> < **<arithmetic expression>** ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <realtional term> < **<arithmetic term>** ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <realtional term> < **<arithmetic factor>** ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <realtional term> < <base>**<exponent>** ; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <realtional term> < **<base>** Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <realtional term> < **<constant>** Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<realtional term>** < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<arithmetic expression>** < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<arithmetic term>** < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; <base>**<exponent>** < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<base>** Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<assignment> ; **<variable>** Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(**<assignment>** ; x’ Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<variable> := **<literals>** ; x’ Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(<variable> := **<constant>** ; x’ Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(**<variable>** := 1 ; x’ Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}
→forloop(‘x := 1 ; x’ Є < 5 Є; ‘x Є + 1 Є){integral ‘count := ‘counter Є;}


<loop statement>
<forloop>
forloop
(
<assignment>
;
<relational expression>
;
<arrithmetic expression>
)
{
<statementt>
}
3
2
1
<declaration statement>
<data type>
integral
<declaration type>
<assignment>
<list>
<variable>
:=
<variable>
ϵ
‘count
‘counter


1
<arrithmetic expression>
+
<arrithmetic term>
<arrithmetic term>
<arrithmetic factor>
<arrithmetic factor>
<base>
<exponent>
<base>
<exponent>
<variable>
Є
<constant>
Є
‘x
1
2
<relational term>
<relational term>
<
<arrithmetic expression>
<arrithmetic expression>
<arrithmetic term>
<arrithmetic term>
<arrithmetic factor>
<arrithmetic factor>
<base>
<exponent>
<base>
<exponent>
<variable>
Є
<constant>
Є
‘x
5


3
<variable>
:=
literals
‘x
<constant>
1

Example 12:

```
amid (‘ctr < 10) {
        ‘ctr = ‘ctr + 2; }
```

G(V) ={<logical expression>,<logical term>, <relational expression>, <relational term>, <statement>, <add statement>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <base>,<variable>, <exponent>, <constant>, <integral literal>}
G(T) = {amid, {, `ctr, }, 10,:=, +,2,;,(,), <,”, ϵ}
G(S) = {<amidloop>}

G(P) :
<amidloop> ::= amid (<logical expression>){statement}
<logical expression> ::= <logical term>
<logical term> ::= <relational expression>
<relational expression> ::= <relational term>
<relational term> ::= <relational term> < <arithmetic expression> | <arithmetic expression>
<statement> ::= <add statement> <statement> | ϵ
<add statement> ::= <assignment>
<assignment> ::= <variable> := <arithmetic expression>
<arithmetic expression> ::= <arithmetic term>
<arithmetic term> ::= <arithmetic factor>
<arithmetic factor> ::= <base><exponent>
<base> ::= <variable>|<constant>
<constant> ::= <integral literal>
<integral literal> ::= 10 | 2
<variable> ::= ‘ctr
<exponent> ::= Є

LEFTMOST DERIVATION

<amidloop> → amid(**<logical expression>**) { <statement> }

→ amid(**<logical term>**) { <statement> }

→ amid(**<relational expression>**) { <statement> }

→ amid(**<relational term>**) { <statement> }

→ amid(**<relational term>** < <arithmetic expression>) { <statement> }

→ amid(**<arithmetic expression>** < <arithmetic expression>) { <statement> }

→ amid(**<arithmetic term>** < <arithmetic expression>) { <statement> }

→ amid(**<arithmetic factor>** < <arithmetic expression>) { <statement> }

→ amid(**<base>**<exponent> < <arithmetic expression>) { <statement> }

→ amid(**<variable>**<exponent> < <arithmetic expression>) { <statement> }

→ amid(`ctr**<exponent>** < <arithmetic expression>) { <statement> }

→ amid(`ctr ϵ < **<arithmetic expression>**) { <statement> }

→ amid(`ctr ϵ < **<arithmetic term>**) { <statement> }

→ amid(`ctr ϵ < **<arithmetic factor>**) { <statement> }

→ amid(`ctr ϵ < **<base>**<exponent>) { <statement> }

→ amid(`ctr ϵ < **<constant>**<exponent>) { <statement> }

→ amid(`ctr ϵ < **<integral literal>**<exponent>) { <statement> }

→ amid(`ctr ϵ < 10 **<exponent>**) { <statement> }

→ amid(`ctr ϵ < 10 ϵ) { **<statement>** }

→ amid(`ctr ϵ < 10 ϵ) { **<add_statement>**<statement> }

→ amid(`ctr ϵ < 10 ϵ) { **<assignment>**;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { **<variable>** := <arithmetic expression>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := **<arithmetic expression>**<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := **<arithmetic expression>** + <arithmetic term>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := **<arithmetic term>** + <arithmetic term>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := **<arithmetic factor>** + <arithmetic term>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := **<base>**<exponent> + <arithmetic term>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := **<variable>**<exponent> + <arithmetic term>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr **<exponent>** + <arithmetic term>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + **<arithmetic term>**;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + **<arithmetic factor>**;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + **<base>**<exponent>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + **<constant>**<exponent>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + **<integral literal>**<exponent>;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + 2 **<exponent>**;<statement> }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ; **<statement>** }

→ amid(`ctr ϵ < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

RIGHTMOST DERIVATION

<amidloop> → amid(<logical expression>) { **<statement>** }

→amid(<logical expression>) { <add statement>; **<statement>** }

→amid(<logical expression>) { **<add statement>** ϵ }

→amid(<logical expression>) { **<assignment>**; ϵ }

→amid(<logical expression>) { <variable> := **<arithmetic expression>**; ϵ }

→amid(<logical expression>) { <variable> := <arithmetic expression>+<**arithmetic term>**; ϵ }

→amid(<logical expression>) { <variable> := <arithmetic expression>+**<arithmetic factor>**; ϵ }

→ amid(<logical expression>) { <variable> := <arithmetic expression> + <base> **<exponent>**; ϵ }

→ amid(<logical expression>) { <variable> := <arithmetic expression>+**<base>** ϵ ; ϵ }

→ amid(<logical expression>) { <variable> := <arithmetic expression>+**<constant>** ϵ ; ϵ }

→amid(<logical expression>) { <variable> := <arithmetic expression>+<**integral literal**> ϵ ; ϵ }

→amid(<logical expression>) { <variable> := **<arithmetic expression>** + 2 ϵ ; ϵ }

→amid(<logical expression>) { <variable> := **<arithmetic term>** + 2 ϵ ; ϵ }

→amid(<logical expression>) { <variable> := **<arithmetic factor>** + 2 ϵ ; ϵ }

→amid(<logical expression>) { <variable> := <base>**<exponent>** + 2 ϵ ; ϵ }

→amid(<logical expression>) { <variable> := **<base>** ϵ + 2 ϵ ; ϵ }

→amid(<logical expression>) { <variable> := **<variable>** ϵ + 2 ϵ ; ϵ }

→amid(<logical expression>) { **<variable>** := `ctr ϵ + 2 ϵ ; ϵ }

→amid(**<logical expression>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(**<logical term>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(**<relational expression>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(**<relational term>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(<relational term> < **<arithmetic expression>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(<relational term> < **<arithmetic term>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

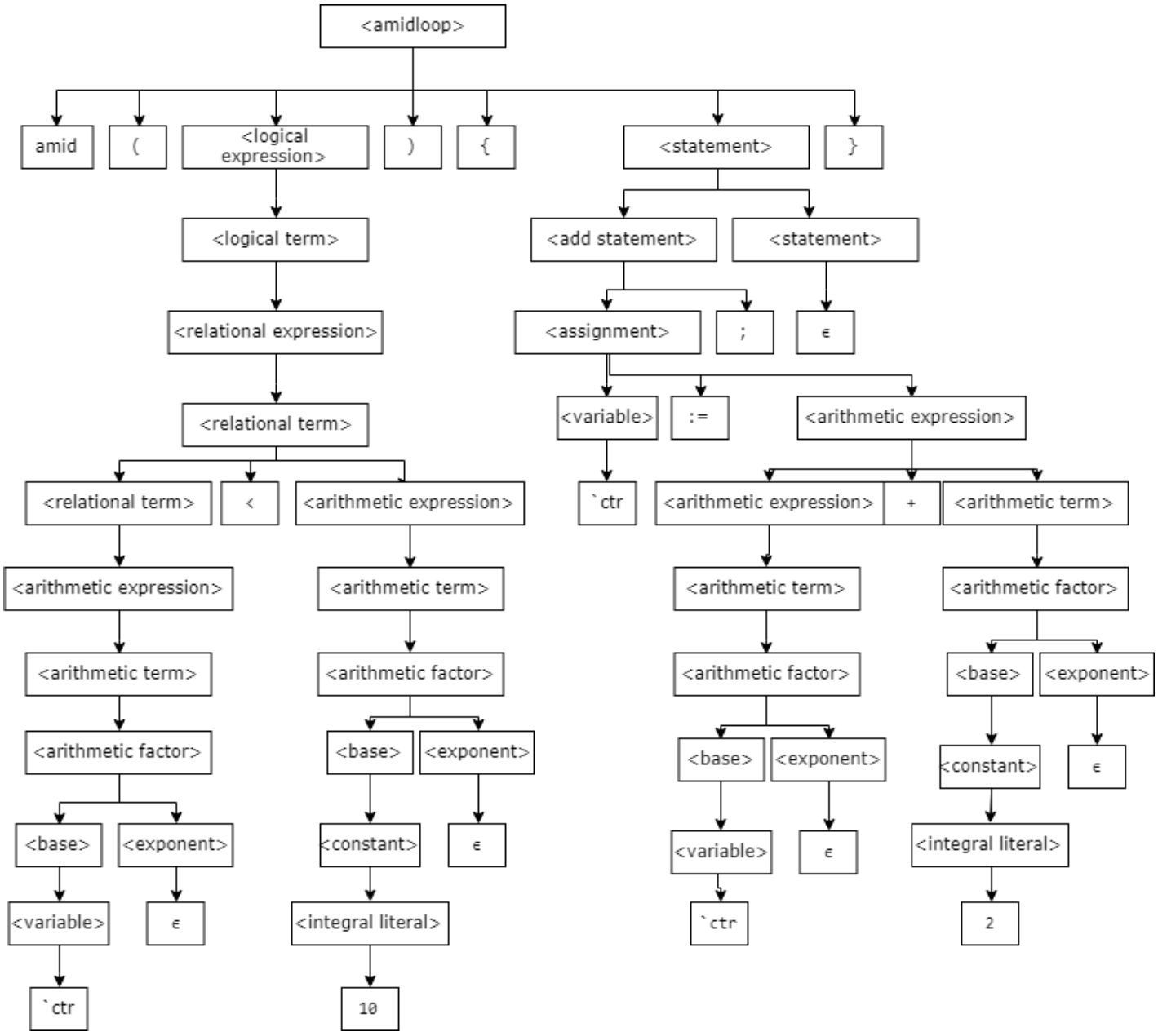
→amid(<relational term> < **<arithmetic factor>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(<relational term> < <base>**<exponent>**) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(<relational term> < **<base>** ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

→amid(<relational term> < **<constant>** ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(<relational term> < **<integer literal>** ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(**<relational term>** < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(**<arithmetic expression>** < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(**<arithmetic term>** < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(**<arithmetic factor>** < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(<base>**<exponent>** < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(**<base>** ϵ < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(**<variable>** ϵ < 10 ϵ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }
→amid(`ctr ϵ < 10 ϵ ) { `ctr := `ctr ϵ + 2 ϵ ; ϵ }

LEFTMOST and RIGHTMOST TREE



Example 13:

```
act {
        outdis("*");
}amid('ctr < 10)
```

G(V) ={<i/o statement>, <output>, <logical expression>,<logical term>, <relational expression>, <relational term>, <statement>, <add statement>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <base>,<variable>, <exponent>, <constant>, <integral literal>}
G(T) = {amid, act, outdis, {, `ctr, }, 10,,;,(,), <,", ϵ}
G(S) = {<amidloop>}

G(P) :
<statement> ::= <add statement> <statement> | ϵ
<add statement> ::= <i/o statement>
<i/o statement> ::= <output>
<output> ::= outdis (<print>);
<print> ::= "<text literal>"<print>
<print> ::= Є
<amidloop> ::= amid (<logical expression>){statement}
<logical expression> ::= <logical term>
<logical term> ::= <relational expression>
<relational expression> ::= <relational term>
<relational term> ::= <relational term> < <arithmetic expression> | <arithmetic expression>
<arithmetic expression> ::= <arithmetic term>
<arithmetic term> ::= <arithmetic factor>
<arithmetic factor> ::= <base><exponent>
<base> ::= <variable>|<constant>
<constant> ::= <integral literal>
<integral literal> ::= 10
<variable> ::= 'ctr
<exponent> ::= Є

LEFTMOST DERIVATION

<act_amidloop> → act{**<statement>**} amid (<logical expression>)

→act{<**add_ statement>**<statement> } amid (<logical expression>)

→act{**<i/o statement>**<statement> } amid (<logical expression>)

→act{**<output>**;<statement> } amid (<logical expression>)

→act{outdis(**<print>**);<statement> } amid (<logical expression>)

→act{outdis("**<text literal>**"<print>);<statement> } amid (<logical expression>)

→ act{outdis("*"**<print>**);<statement> } amid (<logical expression>)

→ act{outdis("*"ϵ);**<statement**> } amid (<logical expression>)

→act{outdis("*"ϵ); ϵ} amid (**<logical expression>**)

→act{outdis("*"ϵ); ϵ} amid (**<logical term>**)

→act{outdis("*"ϵ); ϵ} amid (**<relational expression>**)

→act{outdis("*"ϵ); ϵ} amid (**<relational term>**)

→act{outdis("*"ϵ); ϵ} amid (**<relational term>** < <arithmetic expression>)

→act{outdis("*"ϵ); ϵ} amid (**<arithmetic expression>** < <arithmetic expression>

→ act{outdis("*"ϵ); ϵ} amid (**<arithmetic term>** < <arithmetic expression>)

→act{outdis("*"ϵ); ϵ} amid (**<arithmetic factor>** < <arithmetic expression>)

→act{outdis("*"ϵ); ϵ} amid (**<base>**<exponent> < <arithmetic expression>)

→act{outdis("*"); ϵ} amid (**<variable>**<exponent> < <arithmetic expression>)

→act{outdis("*"); ϵ} amid (`ctr**<exponent>** < <arithmetic expression>)

→act{outdis("*"); ϵ} amid (`ctr ϵ < **<arithmetic expression>**)

→act{outdis("*"); ϵ} amid (`ctr ϵ < **<arithmetic term>**)

→act{outdis("*"); ϵ} amid (`ctr ϵ < **<arithmetic factor>**

→act{outdis("*"); ϵ} amid (`ctr ϵ < **<base>**<exponent>)

→ act{outdis("*"); ϵ} amid (`ctr ϵ < **<constant>**<exponent>)

→act{outdis("*"); ϵ} amid (`ctr ϵ < **<integral literal>**<exponent>)

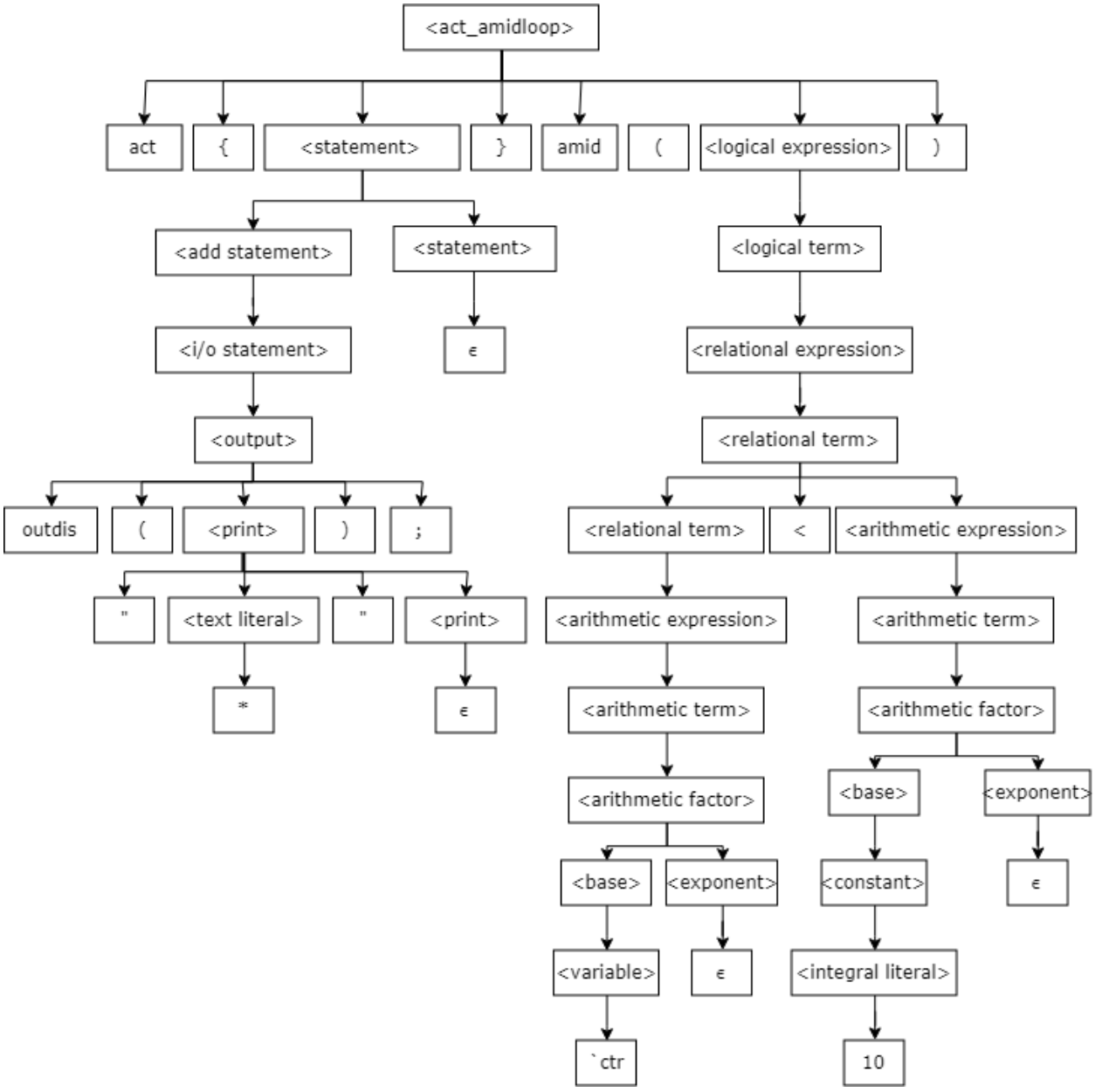
→act{outdis("*"); ϵ} amid (`ctr ϵ < 10 **<exponent>**)

→act{outdis("*"); ϵ} amid (`ctr ϵ < 10 ϵ)

RIGHTMOST DERIVATION

<act_amidloop> →act{<statement>} amid **(<logical expression>**)
→act{<statement>} amid (**<logical term>**)
→act{<statement>} amid (**<relational expression>**)
→act{<statement>} amid (**<relational term>**)
→act{<statement>} amid (<relational term> < **<arithmetic expression>**)
→act{<statement>} amid (<relational term> < **<arithmetic term>**)
→act{<statement>} amid (<relational term> < **<arithmetic factor>**)
→act{<statement>} amid (<relational term> < <base>**<exponent>**)
→act{<statement>} amid (<relational term> < **<base>** ϵ)
→act{<statement>} amid (<relational term> < **<constant>** ϵ)
→act{<statement>} amid (<relational term> < **<integral literal>** ϵ)
→act{<statement>} amid (**<relational term>** < 10 ϵ)
→act{<statement>} amid (**<arithmetic expression>** < 10 ϵ)
→act{<statement>} amid (**<arithmetic term>** < 10 ϵ)
→act{<statement>} amid (**<arithmetic factor>** < 10 ϵ)
→act{<statement>} amid (<base>**<exponent>** < 10 ϵ)
→act{<statement>} amid (**<base>** ϵ < 10 ϵ)
→act{<statement>} amid (**<variable>** ϵ < 10 ϵ)
→act{**<statement>}** amid (`ctr ϵ < 10 ϵ)
→act{<add_statement> **<statement>**} amid (`ctr ϵ < 10 ϵ)
→act{**<add statement>** ϵ } amid (`ctr ϵ < 10 ϵ)
→act{**<i/o statement>** ϵ } amid (`ctr ϵ < 10 ϵ)
→act{**<output>** ϵ } amid (`ctr ϵ < 10 ϵ)
→act{outdis(**<print>**); ϵ } amid (`ctr ϵ < 10 ϵ)
→act{outdis("<text literal>"**<print>**); ϵ } amid (`ctr ϵ < 10 ϵ)
→act{outdis("**<text literal>**" ϵ); ϵ } amid (`ctr ϵ < 10 ϵ)
→act{outdis("*"ϵ);} amid (`ctr < 10)

LEFTMOST and RIGHTMOST TREE



Example 14:

```
If(‘gwa = 1.00) then {
        outdis (“Outstanding!”);
}
```

G(V) = { <conditional statement>, <logical expression>, <relational expression>, <relational term >, <arithmetic expression> , <arithmetic term> , <arithmetic factor >, < base >, <constant>, <decimal literal>, < add_statement>, < i/o statement >, <output>, < outdis (<print>); > ,“<text literal>”}

G(T) ={ ‘gwa , 1.00, = , ; , Є}

G(S) = {<statement>}

G(P) =

<conditional statement> ::= if (<logical expression>) then { <statement>}
<logical expression> ::= <logical expression> || <logical term>
<logical term> ::= <relational expression>
<relational expression>::= <relational expression> = <relational term>
<relational term> ::= <arithmetic expression>
<arithmetic expression> ::= <arithmetic term>
<arithmetic term> ::= <arithmetic factor>
<arithmetic factor> ::= <base> <exponent>
<base> ::= <variable> | <constant>
<exponent> ::= Є
<variable> ::= ‘gwa
<constant> := <decimal literal>
<decimal literal> := 1.00
<statement> ::= <add_statement> <statement>
<add_statement> ::= <i/o statement>
<i/o statement> ::= <output>
<output> ::= outdis (<print>);
<print> ::= “<text literal>”<print> | Є
“<text literal>” : = “Outstanding!”

LEFTMOST DERIVATION

<conditional statement> → if (**<logical expression>**) then { <statement>}

→ if (**<relational expression>**) then { <statement>}

→ if (**<relational term >**) then { <statement>}

→ if (**<relational term >** = **< arithmetic expression >**) then { <statement>}

→ if (**<arithmetic expression >** = < arithmetic expression >) then { <statement>}

→ if (**<arithmetic term >** = < arithmetic expression >) then { <statement>}

→ if (**<arithmetic factor >** = < arithmetic expression >) then { <statement>}

→ if (**<base>**<exponent> = < arithmetic expression >) then { <statement>}

→ if (**<variable>**<exponent> = < arithmetic expression >) then { <statement>}

→ if ('gwa **<exponent>** = < arithmetic expression >) then { <statement>}

→ if ('gwa Є = **< arithmetic term >**) then { <statement>}

→ if ('gwa Є = **< arithmetic factor >**) then { <statement>}

→ if ('gwa Є = **< base >**<exponent>) then { <statement>}

→ if ('gwa Є = **< constant >**<exponent>) then { <statement>}

→ if ('gwa Є = **< decimal literal >** <exponent>) then { <statement>}

→ if ('gwa Є = 1.00 **<exponent>**) then { <statement>}

→ if ('gwa Є = 1.00 Є) then { **< add_statement>** <statement t>}

→ if ('gwa Є = 1.00 Є) then { **< i/o statement >** <statement t>}

→ if ('gwa Є = 1.00 Є) then { **<output>** <statement >}

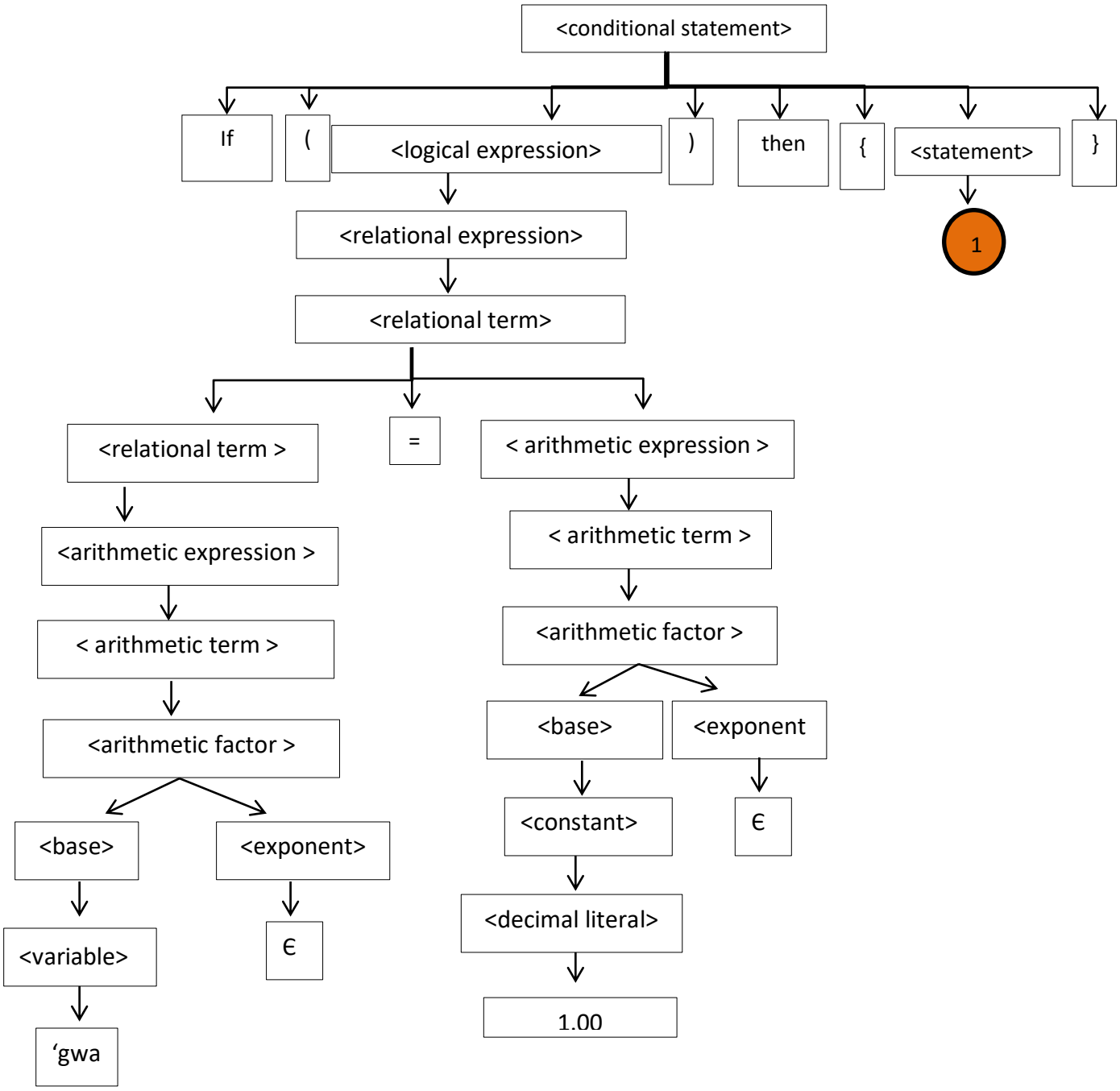
→ if ('gwa Є = 1.00 Є) then { outdis (<**print>**); > <statement >}

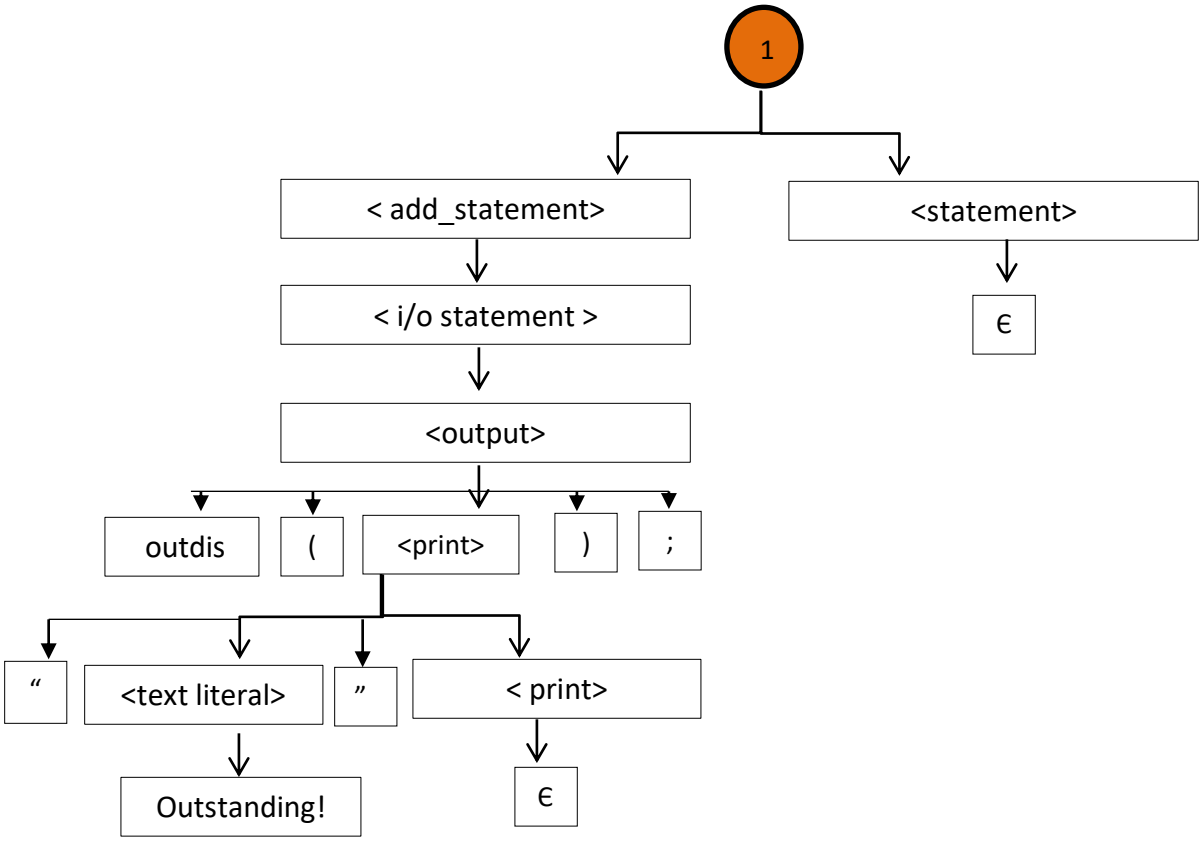
→ if ('gwa Є = 1.00 Є) then { outdis ("**<text literal>**"<print>); > <statement>}

→ if ('gwa Є = 1.00 Є) then { outdis ("Outstanding"**<print>**); > <statement >}

→ if ('gwa Є = 1.00 Є) then { outdis ("Outstanding"Є); > <statement >}

→ if ('gwa Є = 1.00 Є) then { outdis ("Outstanding" Є); Є }

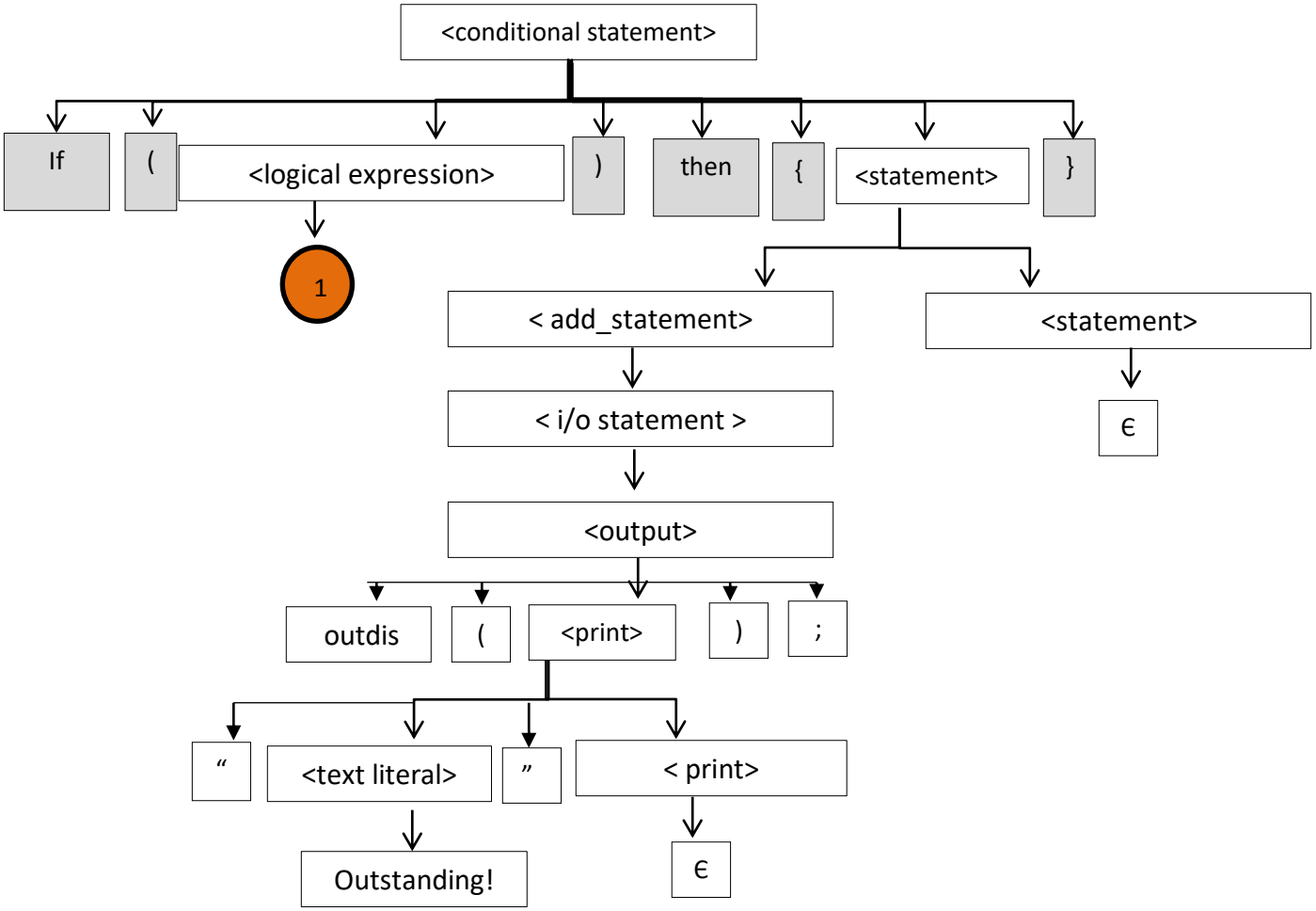

<conditional statement>
If
(
<logical expression>
)
then
{
<statement>
}
1
<relational expression>
<relational term>
<relational term >
=
< arithmetic expression >
<arithmetic expression >
< arithmetic term >
<arithmetic factor >
<base>
<exponent>
<variable>
Є
'gwa
< arithmetic term >
<arithmetic factor >
<base>
<exponent
<constant>
Є
<decimal literal>
1.00

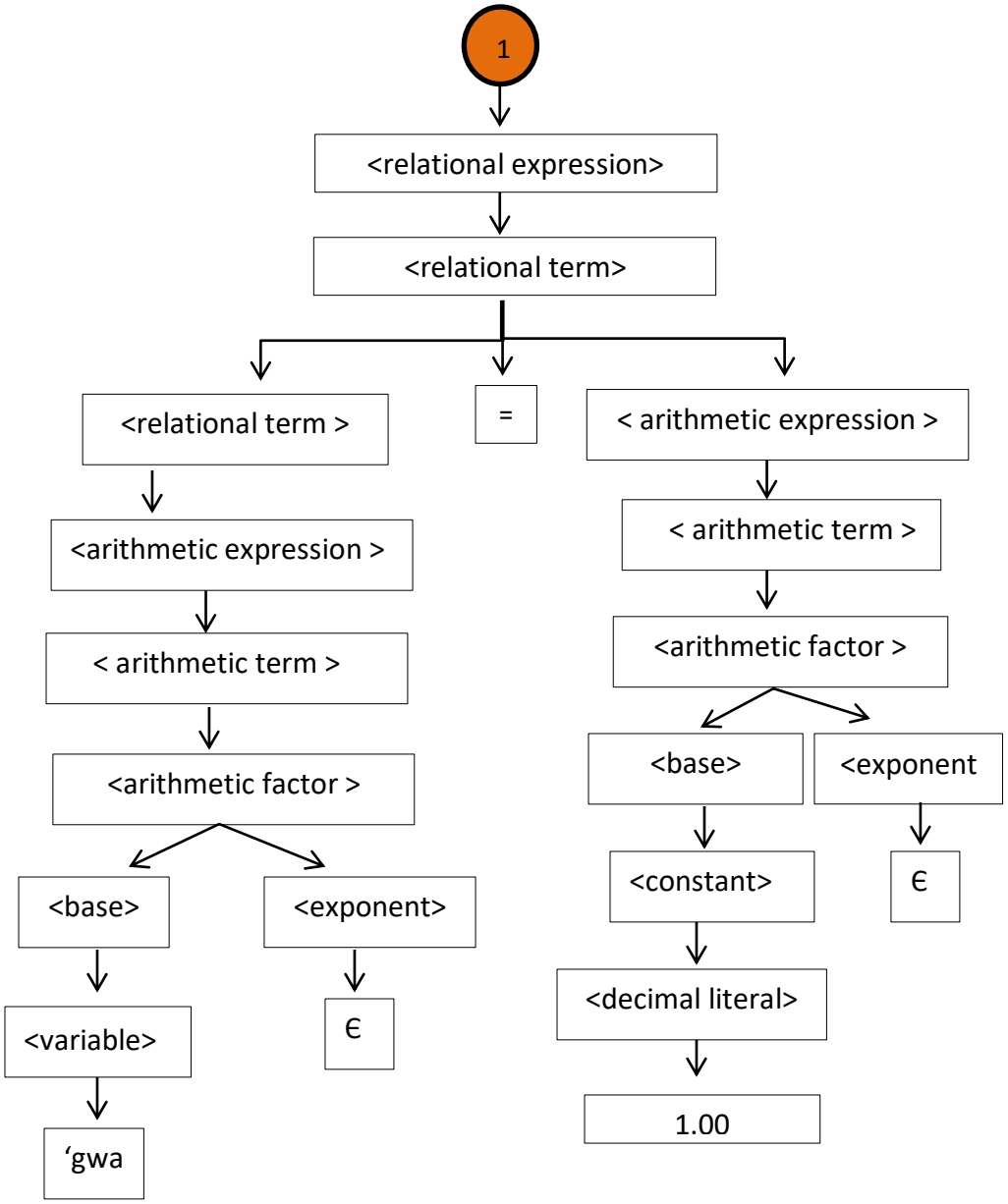

1
< add_statement>
<statement>
< i/o statement >
ϵ
<output>
outdis
(
<print>
)
;
“
<text literal>
”
< print>
Outstanding!
ϵ

RIGHTMOST DERIVATION

<conditional statement> → if (<logical expression>) then { **<statement>**}

→ if (<logical expression>) then { <add_statement> **<statement>** }
→ if (<logical expression>) then { <add_statement> Є }
→ if (<logical expression>) then { **<i/o statement>** Є }
→ if (<logical expression>) then { **<output>** Є }
→ if (<logical expression>) then { outdis (**<print>**); Є }
→ if (<logical expression>)then { outdis ("<text literal>"**<print>**); Є }
→ if(<logical expression>) then { outdis ("**<text literal>**" Є); Є }
→ if(**<logical expression>**) then { outdis ("Outstanding" Є); Є }
→ if(**<relational expression>**) then { outdis ("Outstanding" Є); Є }
→ if (**<relational term>**) then { outdis ("Outstanding" Є); Є }
→ if (<relational term> = **<arithmetic expression>**)then{ outdis ("Outstanding" Є); Є }
→ if (<relational term> = **<arithmetic term>**) then { outdis ("Outstanding" Є); Є }
→ if (<relational term> = **<arithmetic factor>**) then { outdis ("Outstanding" Є); Є }
→ if (<relational term> = <base> **<exponent>**) then { outdis ("Outstanding" Є); Є }
→ if (<relational term> = **<base>** Є) then { outdis ("Outstanding" Є); Є }
→ if (<relational term> = **<constant>** Є) then { outdis ("Outstanding" Є); Є }
→ if (<relational term> = **<decimal literal>** Є) then { outdis ("Outstanding" Є); Є }
→ if (**<relational term>** = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if (**<arithmetic expression>** = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if (**<arithmetic term>** = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if (**<arithmetic factor>** = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if (<base> **<exponent>** = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if (**<base>** Є = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if (**<variable>** Є = 1.00 Є) then { outdis ("Outstanding" Є); Є }
→ if ('gwa Є = 1.00) then { outdis ("Outstanding" Є); Є }


<conditional statement>
If
(
<logical expression>
)
then
{
<statement>
}
1
< add_statement>
<statement>
< i/o statement >
Є
<output>
outdis
(
<print>
)
;
“
<text literal>
”
< print>
Outstanding!
Є


1
<relational expression>
<relational term>
<relational term >
=
< arithmetic expression >
<arithmetic expression >
< arithmetic term >
<arithmetic factor >
<base>
<exponent>
<variable>
Є
'gwa
< arithmetic term >
<arithmetic factor >
<base>
<exponent
<constant>
Є
<decimal literal>
1.00

Example 15:

```
If(‘age = 2) then {
outdis(“The age is two years old.”);
}
If (‘age = 3) then {
outdis(“The age is three years old.”);
}
```

G(V) = { <conditional statement>, <logical expression>, <logic term>, <relational expression>, <relational term >, <arithmetic expression>, <arithmetic term>, <arithmetic factor >, <base>, <exponent>, <constant>, <decimal literal>, <add_statement>, <i/o statement>, <output>, <print> , <text literal>}

G(T) ={ outdis, ‘age ,2, 3 , = , ; ,(,), “, The age is two years old., The age is three years old. ,Є}

G(S) = {<condtional statement>}

G(P) =

<statement> ::= <add_statement> <statement>
<add_statement> ::= <conditional statement> | <i/o statement>
<conditional statement> ::= if (<logical expression>) then { <statement>}
<logical expression> ::= <logical term>
<logical term> ::= <relational expression>
<relational expression>::= <relational expression> = <relational term>
<relational term> ::= <arithmetic expression>
<arithmetic expression> ::= <arithmetic term>
<arithmetic term> ::= <arithmetic factor>
<arithmetic factor> ::= <base> <exponent>
<base> ::= <variable> | <constant>
<exponent> := Є
<variable> ::= ‘age
<constant> := <integral literal >
<integral literal> := 2 | 3
<i/o statement> ::= <output>
<output> ::= outdis (<print>);
<print> ::= “<text literal>”<print> | Є
“<text literal>” : = The age is two years old | The age is three years old

LEFTMOST DERIVATION

<statement> → **<add_statement>** <statement>
→ **<conditional statement>** <statement>
→if (**<logical expression>**) then { <statement>} <statement>
→ if (**<relational expression>**) then { <statement>} <statement>
→ if (**<relational term>**) then { <statement>} <statement>
→ if (**<relational term>** = < arithmetic expression >) then { <statement>} <statement>
→ if (**<arithmetic expression>** = <arithmetic expression>) then { <statement>} <statement>
→ if (**<arithmetic term>** = <arithmetic expression>) then { <statement>} <statement>
→ if (**<arithmetic factor>** = <arithmetic expression>) then { <statement>} <statement>
→ if (**<base>** <exponent> = < arithmetic expression >) then { <statement>} <statement>
→ if (**<variable>** <exponent>= < arithmetic expression >) then { <statement>} <statement>
→ if ('age **<exponent>** = < arithmetic expression>) then { <statement>} <statement>
→ if ('age Є = **<arithmetic expression>**) then { <statement>} <statement>
→ if ('age = **<arithmetic term>**) then { <statement>} <statement>
→ if ('age = **<arithmetic factor>**) then { <statement>} <statement>
→ if ('age = **<base>** <exponent>) then { <statement>} <statement>
→ if ('age = **<constant>** <exponent>) then { <statement>} <statement>
→ if ('age = **<integral literal>** <exponent>) then { <statement>} <statement>
→ if ('age = 2 **<exponent>**) then { <statement>} <statement>
→ if ('age = 2 Є) then { **<statement>**} <statement>
→ if ('age = 2 Є) then { **<add_statement> <statement>**} <statement>
→ if ('age = 2 Є) then { **<i/o statement>** <statement>} <statement>
→ if ('age = 2 Є) then { **<output>** <statement>} <statement>
→ if ('age = 2 Є) then { outdis (**<print>**); <statement>} <statement>
→ if ('age = 2 Є) then { outdis ("**<text literal>**"<print>); <statement t>} <statement>
→ if ('age = 2 Є) then { outdis ("The age is two years old" **<print>**); <statement>} <statement>
→ if ('age = 2 Є) then { outdis ("The age is two years old " Є); <statement>} <statement>
→ if ('age = 2 Є) then { outdis ("The age is two years old" Є); Є } <statement>
→ if ('age = 2 Є) then { outdis ("The age is two years old" Є); Є }
<add_statement> <statement>
→ if ('age = 2 Є) then { outdis ("The age is two years old" Є); Є }
<conditional statement> <statement>
→ if ('age = 2 Є) then { outdis ("The age is two years old" Є); Є }
if (**<logical expression>**) then { <statement>} <statement>
→ if ('age = 2) then { outdis ("The age is two years old" Є); Є }
if (**<logical term>**) then { <statement>} <statement>

→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<relational expression>**) then { <statement>} <statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<relational expression>** = <relational term>) then { <statement>} <statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<arithmetic expression>** = <relational term>) then { <statement>} <statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<arithmetic term>** = <relational term>) then { <statement>} <statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<arithmetic factor>** = <relational term>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<base>**<exponent> = <relational term>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (**<variable>**<exponent> = <relational term>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age **<exponent>** = <relational term>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age **<exponent>** = <relational term>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є= **<relational term>**) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є= **<arithmetic expression>**) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є= **<arithmetic term>**) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = **<arithmetic factor>**) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = **<base>** <exponent>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = **<constant>**<exponent>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = **<integral literal>**<exponent>) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = 3 **<exponent>**) then { <statement>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = 3 Є) then { **<statement>**}<statement>

→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = 3 Є) then { **<add_statement>** <statement >}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = 3 Є) then { **<i/o statement>** <statement >}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age Є = 3 Є) then { **<output>** <statement >}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age = 3) then { outdis (**<print>**); <statement t>}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age = 3) then { outdis (“**<text literal>**”<print>); <statement >}<statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age = 3) then { outdis (“The age is three years old”**<print>**); <statement >} <statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age = 3) then { < outdis (“The age is three years old“ Є); **<statement >**} <statement>
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age = 3) then { < outdis (“The age is three years old“ Є); Є } **<statement>**
→ if (‘age = 2) then { outdis (“The age is two years old” Є); Є }
if (‘age = 3) then { < outdis (“The age is three years old“ Є); Є } Є

RIGHTMOST DERIVATION

<statement> → <add_statement>**<statement>**
→<add_statement><add_statement>**<statement>**
→<add_statement>**<add_statement>**Є
→<add_statement>**<conditional statement>**Є
→<add_statement> if (<logical expression>) then { **<statement>**} Є
→<add_statement> if (<logical expression>) then { <add_statement>**<statement>**} Є
→<add_statement> if (<logical expression>) then { **<i/o statement>**Є} Є
→<add_statement> if (<logical expression>) then { **<output>**Є} Є
→<add_statement> if (<logical expression>) then { outdis (**<print>**); Є} Є
→<add_statement> if (<logical expression>) then { outdis ("<text literal>"**<print>**); Є} Є
→<add_statement> if (<logical expression>) then { outdis ("**<text literal>**" Є); Є} Є
→<add_statement> if (**<logical expression>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<logical term>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<relational expression>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<relational term>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<arithmetic expression>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<arithmetic term>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<arithmetic factor>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = <base>**<exponent>**) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<base>**Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<constant>**Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<relational expression> = **<integral literal>**Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<relational expression>** = 3 Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<relational term>** = 3 Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<arithmetic expression>** = 3 Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<arithmetic term>** = 3 Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<arithmetic factor>** = 3 Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (<base>**<exponent>**= 3 Є) then { outdis ("The age is three" Є); Є} Є
→<add_statement> if (**<base>**Є = 3 Є) then { outdis ("The age is three" Є); Є} Є

→<add_statement> if (**<variable>** Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→**<add_statement>** if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→**<conditional statement>** if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { **<statement>**}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { <add_statement>**<statement>**}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { **<add_statement>** Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { **<i/o statement>** Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { **<output>** Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { outdis(**<print>**); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { outdis(“<text literal>” **<print>**); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<logical expression>) then { outdis(“**<text literal>**” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (**<logical expression>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (**<logical term>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (**<relational expression>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = **<relational term>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = **<arithmetic expression>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = **<arithmetic term>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = **<arithmetic factor>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = <base>**<exponent>**) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = **<base>**Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є
→ if (<relational expression> = **<constant>**Є) then { outdis(“The age is two” Є); Є}

if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (<relational expression> = **<integral literal>** Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<relational expression>** = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<relational term>** = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<arithmetic expression>** = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<arithmetic term>** = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<arithmetic factor>** = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

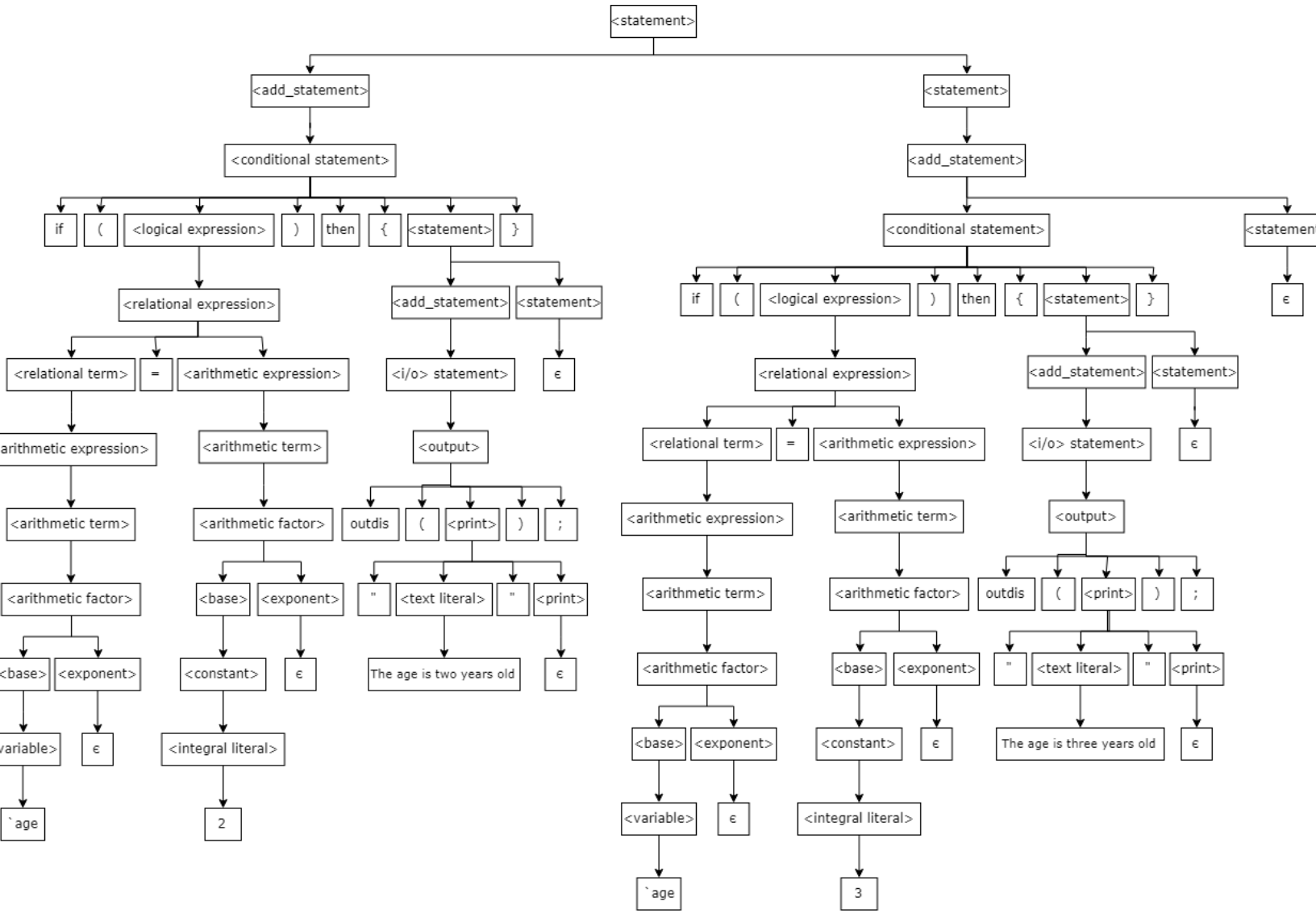
→ if (<base>**<exponent>** = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<base>**Є = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (**<variable>**Є = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

→ if (‘age Є = 2 Є) then { outdis(“The age is two” Є); Є}
if (‘age Є = 3 Є) then { outdis (“The age is three” Є); Є} Є

LEFTMOST and RIGHTMOST TREE



Example 16:

```
if(‘bool = true) then {
        If (‘age = 3) then {
                ‘age := ‘age + 3;
        }
}
```

G(V) = { <conditional statement>, <add statement>,<assignment>, <logical expression>, <logic term>, <relational expression>, <relational term >, <arithmetic expression>, <arithmetic term>, <arithmetic factor >, <base>, <exponent>, <constant>, <integral literal>, <variable>}

G(T) ={ if, then, ‘bool, ‘age, true, 3, +,=,:=,(,),{,},;,Є}

G(S) = {<condtional statement>}

G(P) =

<statement> ::= <add_statement> <statement> | Є

<add_statement> ::= <conditional statement> | <assignment>

<conditional statement> ::= if (<logical expression>) then { <statement>}

<assignment> ::= <variable> := <arithmetic expression>

<logical expression> ::= <logical term>

<logical term> ::= <relational expression>

<relational expression>::= <relational expression> = <relational term>

<relational term> ::= <arithmetic expression>

<arithmetic expression> ::= <arithmetic term>

<arithmetic term> ::= <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<base> ::= <variable> | <constant>

<exponent> := Є

<variable> ::= ‘age

<constant> := <integral literal >

<integral literal> := 3

LEFTMOST DERIVATION:

<conditional statement> → if (**<logical expression>**) then {<statement>}
→ if (**<logical term>**) then {<statement>}
→ if (**<relational expression>**) then {<statement>}
→ if (**<relational expression>** = <relational term>) then {<statement>}
→ if (**<relational term>** = <relational term>) then {<statement>}
→ if (**<arithmetic expression>** = <relational term>) then {<statement>}
→ if (**<arithmetic term>** = <relational term>) then {<statement>}
→ if (**<arithmetic factor>** = <relational term>) then {<statement>}
→ if (**<base>** <exponent> = <relational term>) then {<statement>}
→ if (**<variable>** <exponent> = <relational term>) then {<statement>}
→ if (`bool **<exponent>** = <relational term>) then {<statement>}
→ if (`bool Є = **<relational term>**) then {<statement>}
→ if (`bool Є = **<arithmetic expression>**) then {<statement>}
→ if (`bool Є = **<arithmetic term>**) then {<statement>}
→ if (`bool Є = **<arithmetic factor>**) then {<statement>}
→ if (`bool Є = **<base>** <exponent>) then {<statement>}
→ if (`bool Є = **<boolean literal>** <exponent>) then {<statement>}
→ if (`bool Є = true **<exponent>**) then {<statement>}
→ if (`bool Є = true Є) then {**<statement>**}
→ if (`bool Є = true Є) then {**<add_statement>**<statement>}
→ if (`bool Є = true Є) then {**<conditional statement>** <statement>}
→ if (`bool Є = true Є) then { if (**<logical expression>**) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<logical term>**) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<relational expression>**) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<relational expression>** = <relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<relational term>** =<relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<arithmetic expression>** = <relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<arithmetic term>** =<relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<arithmetic factor>** =<relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є) then { if (**<base>**<exponent> =<relational term>) then { <statement>} <statement>}

→ if (`bool Є = true Є) then { if (**<variable>** <exponent>= <relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age **<exponent>** =<relational term>) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є =**<relational term>**) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = **<arithmetic expression>**) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = **<arithmetic term>**) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = **<arithmetic factor>**) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = **<base>** <exponent>) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = **<constant>** <exponent>) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = **<integral literal>** <exponent>) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 **<exponent>**) then { <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { **<statement>**} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { **<add statement>** <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { **<assignment>** <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { **<variable>** := <arithmetic expression> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<arithmetic expression>** <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<arithmetic expression>** + <arithmetic term> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<arithmetic term>** + <arithmetic term> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<arithmetic term>** + <arithmetic term> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<arithmetic factor>** + <arithmetic term> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<base>** <exponent> + <arithmetic term> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := **<variable>** <exponent> + <arithmetic term> <statement>} <statement>}

→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age **<exponent>** + <arithmetic term> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + **<arithmetic term>** <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + **<arithmetic factor>** <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + **<base>** <exponent> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + **<constant>** <exponent> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + **<integral literal>** <exponent> <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + **3 <exponent>** <statement>} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + 3 Є **<statement>**} <statement>}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + 3 Є **Є** } **<statement>**}
→ if (`bool Є = true Є ) then { if (`age Є = 3 Є) then { `age := `age Є + 3 Є Є } **Є**}

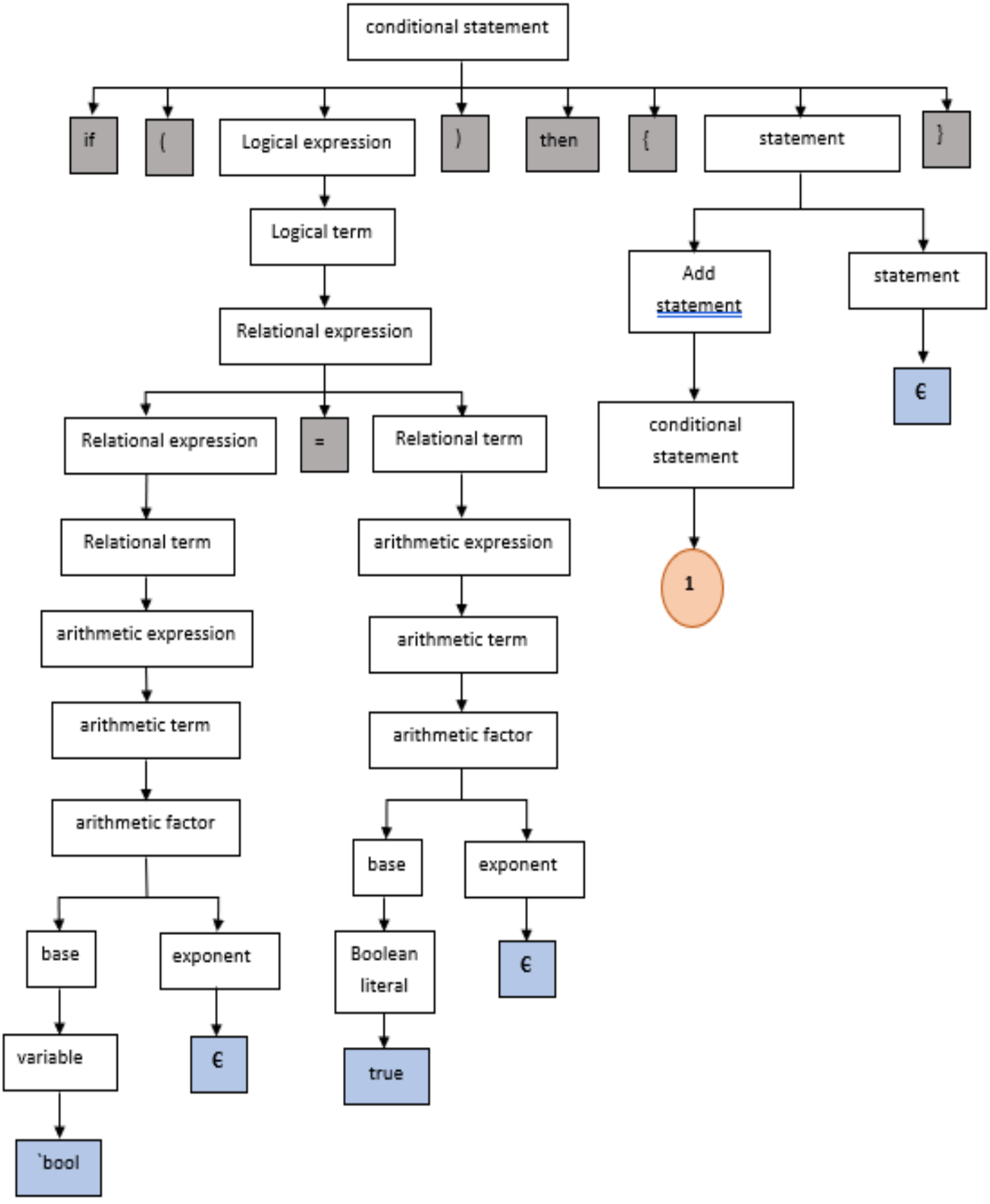
RIGHTMOST DERIVATION:

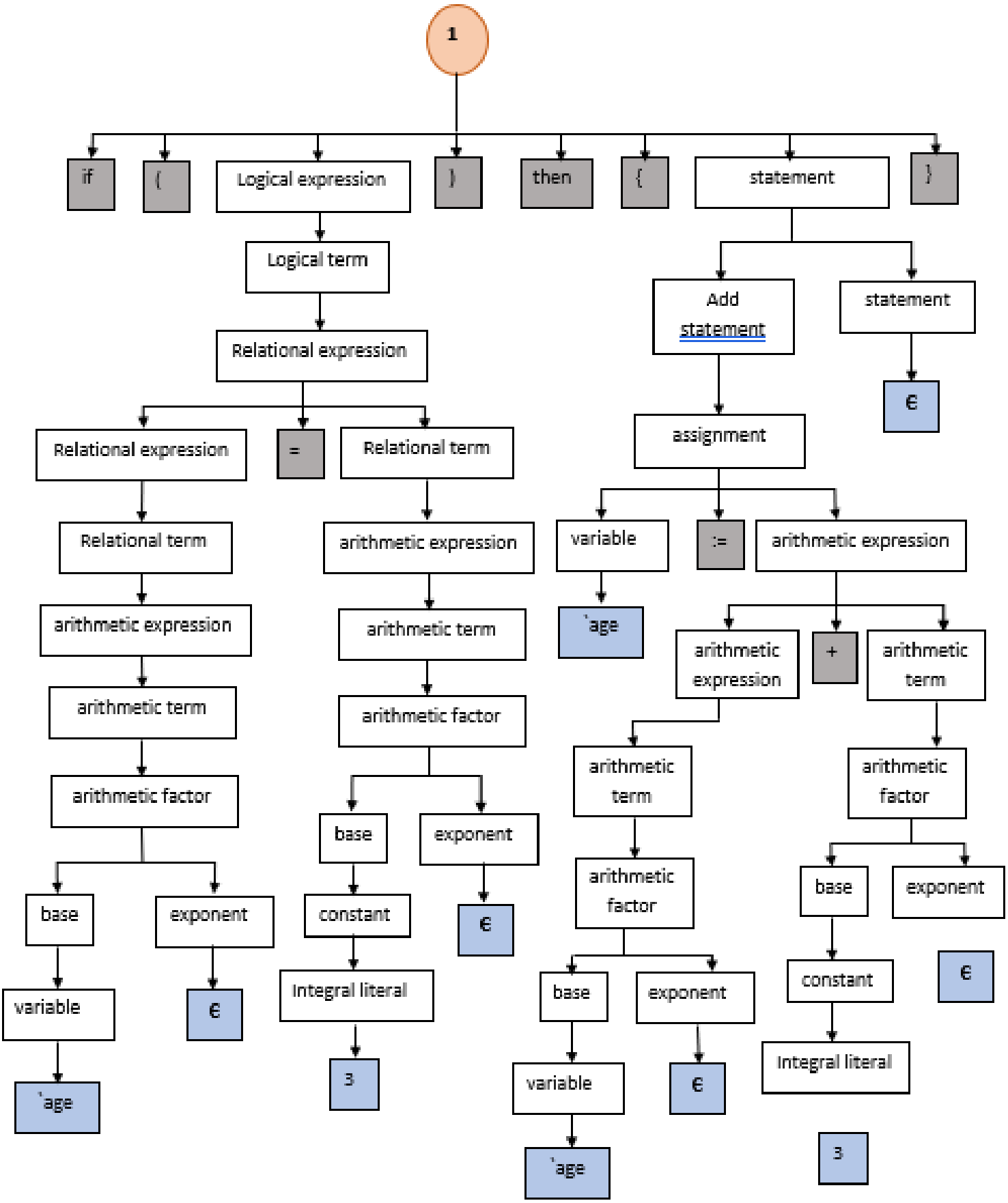
<conditional statement> → if (<logical expression>) then **{<statement>**}
→ if (<logical expression>) then {<add statement> **<statement>**}
→ if (<logical expression>) then {<add statement> Є}
→ if (<logical expression>) then {<conditional statement> Є}
→ if (<logical expression>) then { if (<logical expression>) then {**<statement>**} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<add statement>**<statement>**} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<add statement> Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> **:= <arithmetic expression>** Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <arithmetic expression> + **<arithmetic term>** Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <arithmetic expression> + **<arithmetic factor>** Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <arithmetic expression> + <base> **<exponent>** Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <arithmetic expression> + **<base>** Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <arithmetic expression> + **<constant>** Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <arithmetic expression> + **<integral literal>** Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := **<arithmetic expression>** + 3 Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := **<arithmetic term>** + 3 Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := **<arithmetic factor>** + 3 Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := <base> **<exponent>** + 3 Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := **<base>** Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {<variable> := **<variable>** Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<logical expression>) then {**<variable>** := ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<logical expression>)** then {‘age:= ‘age Є+ 3 Є Є} Є}

→ if (<logical expression>) then { if (**<logical term>)** then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<relational expression>**) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<relational term>**) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<arithmetic expression>**) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<arithmetic term>**) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<arithmetic factor>**) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = <base> **<exponent>**) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<base> Є** then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<constant>** Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<relational expression> = **<integral literal>** Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<relational expression>** = **3** Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<relational term>** = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<arithmetic expression>** = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<arithmetic term>** = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<arithmetic factor>** = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (<base>**<exponent**> = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<base>** Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (**<variable>** Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<logical expression>) then { if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<relational expression>)** then { if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = **<relational term>**) then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = **<arithmetic expression>**) then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = **<arithmetic term>**) then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = **<arithmetic factor>**) then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}

→ if (<relational expression> = <base> **<exponent>**) then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = **<base>** Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = **<boolean literal>** Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<relational expression> = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<relational term>** = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<arithmetic expression>** = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<arithmetic term>** = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<arithmetic factor>** = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (<base> **<exponent>** = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<base>** Є = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (**<variable>** Є = true Є)then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}
→ if (`bool Є = true Є )then {if (`age Є = 3 Є) then {‘age:= ‘age Є+ 3 Є Є} Є}

LEFTMOST and RIGHTMOST TREE




1
if
(
Logical expression
)
then
{
statement
}
Logical term
Relational expression
Relational expression
=
Relational term
Relational term
arithmetic expression
arithmetic term
arithmetic factor
base
exponent
variable
ϵ
`age
arithmetic expression
arithmetic term
arithmetic factor
base
exponent
constant
ϵ
Integral literal
3
Add statement
statement
assignment
ϵ
variable
:=
arithmetic expression
`age
arithmetic expression
+
arithmetic term
arithmetic term
arithmetic factor
arithmetic factor
base
exponent
base
exponent
constant
ϵ
variable
ϵ
Integral literal
`age
3

Example 17:

12> 4 || 5 = ‘var & 3 < 5

G(V) = {<logical expression>, <logical term>, <relational expression>, <relational term>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <base>, <exponent>, <variable>, <constant>}

G(T) = {12, 4, 5, `var, 3 ,5, ||, =, <,>, &, Є}

G(P) =

<logical expression> ::= <logical expression> || <logical term>| <logical term>
<logical term> ::= <logical term> & <relational expression> | <relational expression>
<relational expression>::= <relational expression> = <relational term>| <relational term>
<relational term> ::= <relational term> > <arithmetic expression>
| <relational term> < < arithmetic expression >
<arithmetic expression> ::= <arithmetic term>
<arithmetic term> ::= <arithmetic factor>
<arithmetic factor> ::= <base> <exponent>
<exponent> ::= Є
<base> ::= <variable> | <constant>
G(S) = {<logical expression>}

LEFTMOST DERIVATION:

<logical expression> → **<logical expression>** || <logical term>
→ **<logical term>** || <logical term>
→ **<relational expression>** || <logical term>
→ **<relational term>** || <logical term>
→ **<relational term>** > <arithmetic expression> || <logical term>
→ **<arithmetic expression>** > <arithmetic expression> || <logical term>
→ **<arithmetic term>** > <arithmetic expression> || <logical term>
→ **<arithmetic factor>** > <arithmetic expression> || <logical term>
→ **<base>** <exponent> > <arithmetic expression> || <logical term>
→ **<constant>** <exponent> > <arithmetic expression> || <logical term>
→ **<integral literal>** <exponent> > <arithmetic expression> || <logical term>
→ **12 <exponent>** > <arithmetic expression> || <logical term>
→ **12** Є > **<arithmetic expression>** || <logical term>
→ 12 Є > **<arithmetic term>** || <logical term>
→ 12 Є > **<arithmetic factor>** || <logical term>
→ 12 Є > **<base>**<exponent> || <logical term>
→ 12 Є > **<constant>**<exponent> || <logical term>
→ 12 Є > **<integral literal>**<exponent> || <logical term>
→ 12 Є > **4** <exponent>|| **<logical term>**
→ 12 Є > **4** Є || **<logical term>**
→ 12 Є > 4 Є || <logical term> & <relational expression>
→ 12 Є > 4 Є || **<relational expression>** & <relational expression>
→ 12 Є > 4 Є || **<relational expression>** = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<relational term>** = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<arithmetic expression>** = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<arithmetic term>** = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<arithmetic factor>** = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<base>** <exponent> = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<constant>** <exponent> = <relational term>& <relational expression>
→ 12 Є > 4 Є || **<integral literal>** <exponent> = <relational term>& <relational expression>
→ 12 Є > 4 Є || 5 **<exponent>** = <relational term>& <relational expression>
→ 12 Є > 4 Є || 5 Є = **<relational term>** & <relational expression>
→ 12 Є > 4 Є || 5 Є = **<arithmetic expression>** & <relational expression>
→ 12 Є > 4 Є || 5 Є = **<arithmetic term>**& <relational expression>
→ 12 Є > 4 Є || 5 Є = **<arithmetic factor>**& <relational expression>
→ 12 Є > 4 Є || 5 Є = **<base>** <exponent>& <relational expression>

→ 12 Є > 4 Є || 5 Є = **<variable>** <exponent>& <relational expression>
→ 12 Є > 4 Є || 5 Є = ‘var **<exponent>** & <relational expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <relational expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <relational term>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <relational term> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <arithmetic expression> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <arithmetic term> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <arithmetic factor> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <base><exponent> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <constant><exponent> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & <integral literal><exponent> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 <exponent> < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < <arithmetic expression>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < <arithmetic term>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < <arithmetic factor>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < <base><exponent>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < <constant><exponent>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < 5 <exponent>
→ 12 Є > 4 Є || 5 Є = ‘var Є & 3 Є < 5 Є

RIGHTMOST DERIVATION

<logical expression> → <logical expression> || **<logical term>**
→ <logical expression> || <logical term> & **<relational expression>**
→ <logical expression> || <logical term> & **<relational term>**
→ <logical expression> || <logical term> & <relational term> < **<arithmetic expression>**
→ <logical expression> || <logical term> & <relational term> < **<arithmetic term>**
→ <logical expression> || <logical term> & <relational term> < **<arithmetic factor>**
→ <logical expression> || <logical term> & <relational term> < <base>**<exponent>**
→ <logical expression> || <logical term> & <relational term> < **<base>** Є
→ <logical expression> || <logical term> & <relational term> < **<constant>** Є
→ <logical expression> || <logical term> & <relational term> < **<integral literal>** Є
→ <logical expression> || <logical term> & **<relational term>** < 5 Є
→ <logical expression> || <logical term> & **<arithmetic expression>** < 5 Є
→ <logical expression> || <logical term> & **<arithmetic term>** < 5 Є
→ <logical expression> || <logical term> & **<arithmetic factor>** < 5 Є
→ <logical expression> || <logical term> & <base>**<exponent>** < 5 Є
→ <logical expression> || <logical term> & **<base>**Є < 5 Є
→ <logical expression> || <logical term> & **<constant>** Є < 5 Є
→ <logical expression> || <logical term> & **<integral literal>** Є < 5 Є
→ <logical expression> || **<logical term>** & 3 Є < 5 Є
→ <logical expression> || **<relational expression>** & 3 Є < 5 Є
→ <logical expression> || <relational expression> = **<relational term>** & 3 Є < 5 Є
→ <logical expression> || <relational expression> = **<arithmetic expression>** & 3 Є < 5 Є
→ <logical expression> || <relational expression> = **<arithmetic term>** & 3 Є < 5 Є
→ <logical expression> || <relational expression> = **<arithmetic factor>** & 3 Є < 5 Є
→ <logical expression> || <relational expression> = <base>**<exponent>** & 3 Є < 5 Є
→ <logical expression> || <relational expression> = **<base>** Є & 3 Є < 5 Є
→ <logical expression> || <relational expression> = **<variable>** Є & 3 Є < 5 Є
→ <logical expression> || **<relational expression>** = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<relational term>** = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<arithmetic expression>** = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<arithmetic term>** = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<arithmetic factor>** = ‘var Є & 3 Є < 5 Є
→ <logical expression> || <base> **<exponent>** = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<base>** Є = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<constant>** Є = ‘var Є & 3 Є < 5 Є
→ <logical expression> || **<integral literal>** Є = ‘var Є & 3 Є < 5 Є

→ **<logical expression>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ **<logical term>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ **<relational expression>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ **<relational term>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > **<arithmetic expression>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > **<arithmetic term>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > **<arithmetic factor>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > <base>**<exponent>** || 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > **<base>** Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > **<constant>** Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ <relational term> > **<integral literal>** Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<relational term>** > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<arithmetic expression>** > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<arithmetic term>** > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<arithmetic factor>** > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ <base> **<exponent>** > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<base>** Є > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<constant>** Є > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ **<integral literal>** Є > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є
→ 12 Є > 4 Є|| 5 Є = ‘var Є & 3 Є < 5 Є

Example 18: ‘var ^ ‘var ^ 3

G(V) = {<logical expression>, <logical term>, <relational expression>, <relational term>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <exponent>, <base>, <variable>, <constant>, <integral literal> }

G(T) = {‘var, ^, 3, Є }

G(S) = {<logical expression>}

G(P)

<logical expression> ::= <logical term>

<logical term> ::= <relational expression>

<relational expression>::= <relational term>

<relational term> ::= <arithmetic expression>

<arithmetic expression> ::= <arithmetic term>

<arithmetic term> ::= <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<exponent> ::= ^ <base> <exponent> | Є

<base> ::= <variable> | <constant>

<variable> ::= ‘var

<constant> ::= <integral literal>

<integral literal> ::= 3

LEFTMOST DERIVATION

<logical expression> → **<logical term>**

→ **<relational expression>**

→ **<relational term>**

→ **<arithmetic expression>**

→ **<arithmetic term>**

→ **<arithmetic factor>**

→ **<base> <exponent>**

→ **<variable>** <exponent>

→ 'var **<exponent>**

→ 'var ^ **<base>** <exponent>

→ 'var ^ **<variable>** <exponent>

→ 'var ^ 'var **<exponent>**

→ 'var ^ 'var ^ **<base>** <exponent>

→ 'var ^ 'var ^ **<constant>** <exponent>

→ 'var ^ 'var ^ **<integral literal>** <exponent>

→ 'var ^ 'var ^ 3 **<exponent>**

→ 'var ^ 'var ^ 3 Є

RIGHTMOST DERIVATION

<logical expression> → **<logical term>**

→ **<relational expression>**

→ **<relational term>**

→ **<arithmetic expression>**

→ **<arithmetic term>**

→ **<arithmetic factor>**

→ <base> **<exponent>**

→ <base> ^ <base> **<exponent>**

→ <base> ^ <base> ^ <base> **<exponent>**

→ <base> ^ <base> ^ **<base>** Є

→ <base> ^ <base> ^ **<constant>** Є

→ <base> ^ <base> ^ **<integral literal>** Є

→ <base> ^ **<base>** ^ 3 Є

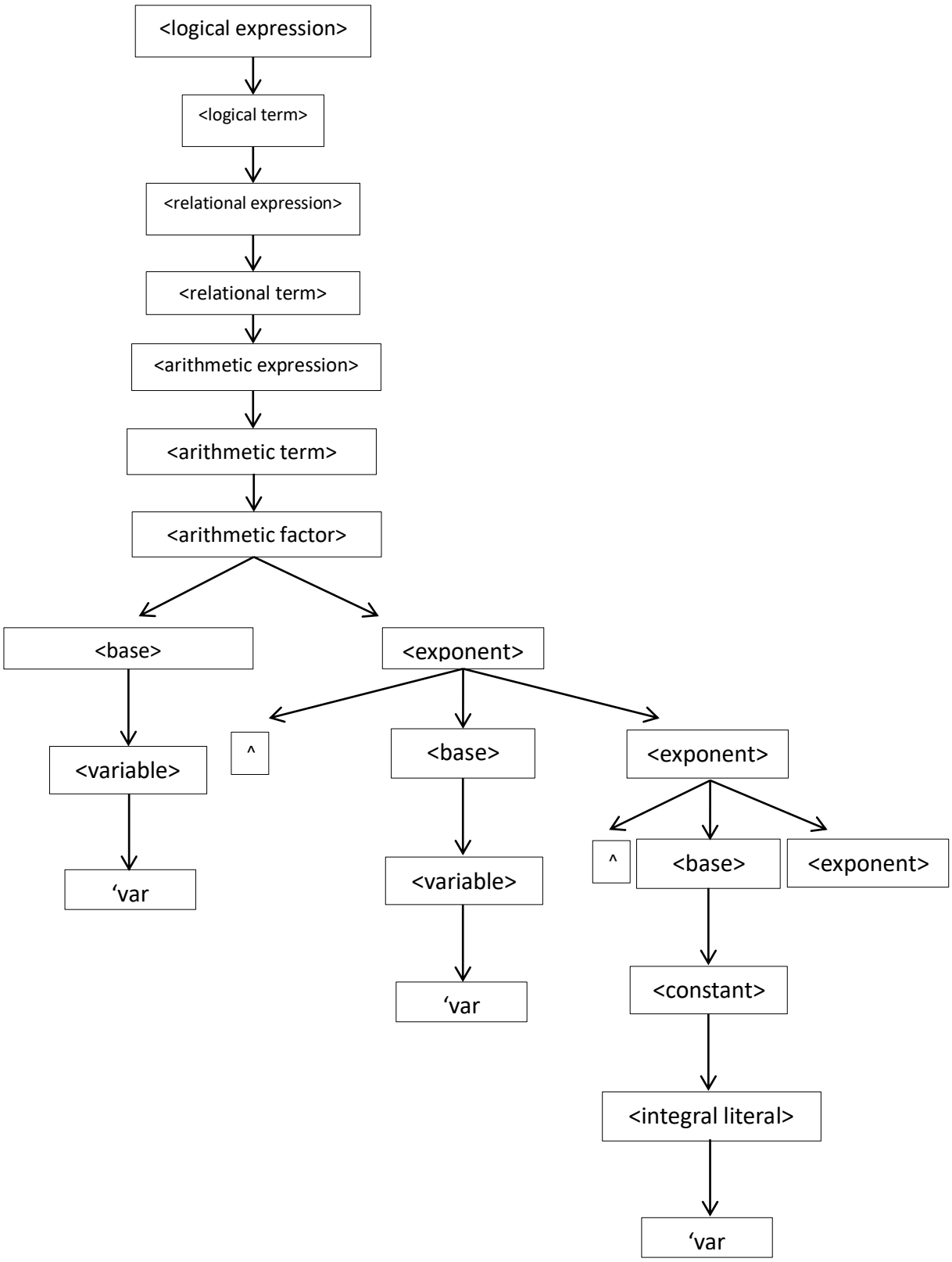
→ <base> ^ **<variable>** ^ 3 Є

→ **<base>** ^ 'var ^ 3 Є

→ **<variable>** ^ 'var ^ 3 Є

→ 'var ^ 'var ^ 3 Є

LEFTMOST and RIGHTMOST TREE



Example 19:

2 > 5 = 5 >|= 10

G(V) = {<logical expression>, <logical term>, <relational expression>, <relational term>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <exponent>, <base>, <constant>, <integral literal> }

G(T) = { 2, >, 5, =, 5, >|=, 10, Є}

G(S) = {<logical expression>}

G(P)

<logical expression> ::= <logical term>

<logical term> ::= <relational expression>

<relational expression>::= <relational expression> = <relational term>

| <relational term>

<relational term> ::= <relational term> > <arithmetic expression>

| <relational term> >|= < arithmetic expression >

| <arithmetic expression>

<arithmetic expression> ::= <arithmetic term>

<arithmetic term> ::= <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<exponent> ::= Є

<base> ::= <constant>

<constant> ::= <integral literal>

LEFTMOST DERIVATION

<logical expression> → **<logical term>**

→ **<relational expression>**

→ **<relational expression>** = <relational term>

→ **<relational term>** > <arithmetic expression> = <relational term>

→ **<arithmetic expression>** > <arithmetic expression> = <relational term>

→ **<arithmetic term>** > <arithmetic expression> = <relational term>

→ **<arithmetic factor>** > <arithmetic expression> = <relational term>

→ **<base>** <exponent> > <arithmetic expression> = <relational term>

→ **<constant>** <exponent> > <arithmetic expression> = <relational term>

→ **<integral literal>** <exponent> > <arithmetic expression> = <relational term>

→ 2 **<exponent>** > <arithmetic expression> = <relational term>

→ 2 Є > **<arithmetic expression>** = <relational term>

→ 2 Є > **<arithmetic term>** = <relational term>

→ 2 Є > **<arithmetic factor>** = <relational term>

→ 2 Є > **<base>** <exponent> = <relational term>

→ 2 Є > **<constant>** <exponent> = <relational term>

→ 2 Є > **<integral literal>** <exponent> = <relational term>

→ 2 Є > 5 **<exponent>** = <relational term>

→ 2 Є > 5 Є = **<relational term>**

→ 2 Є > 5 Є = **<relational term>** >|= < arithmetic expression >

→ 2 Є > 5 Є = **<arithmetic expression>** >|= < arithmetic expression >

→ 2 Є > 5 Є = **<arithmetic term>** >|= < arithmetic expression >

→ 2 Є > 5 Є = **<arithmetic factor>** >|= < arithmetic expression >

→ 2 Є > 5 Є = **<base>** <exponent> >|= < arithmetic expression >

→ 2 Є > 5 Є = **<constant>** <exponent> >|= < arithmetic expression >

→ 2 Є > 5 Є = **<integral literal>** <exponent> >|= < arithmetic expression >

→ 2 Є > 5 Є = 5 **<exponent>** >|= < arithmetic expression >

→ 2 Є > 5 Є = 5 Є >|= **<arithmetic expression>**

→ 2 Є > 5 Є = 5 Є >|= **<arithmetic term>**

→ 2 Є > 5 Є = 5 Є >|= **<arithmetic factor>**

→ 2 Є > 5 Є = 5 Є >|= **<base>** <exponent>

→ 2 Є > 5 Є = 5 Є >|= **<constant>** <exponent>

→ 2 Є > 5 Є = 5 Є >|= **<integral literal>** <exponent>

→ 2 Є > 5 Є = 5 Є >|= 10 **<exponent>**

→ 2 Є > 5 Є = 5 Є >|= 10 Є

RIGHTMOST DERIVATION

<logical expression> → **<logical term>**

→ **<relational expression>**

→ <relational expression> = **<relational term>**

→ <relational expression> = <relational term> >|= **<arithmetic expression >**

→ <relational expression> = <relational term> >|= **<arithmetic term>**

→ <relational expression> = <relational term> >|= **<arithmetic factor>**

→ <relational expression> = <relational term> >|= <base> **<exponent>**

→ <relational expression> = <relational term> >|= **<base>** Є

→ <relational expression> = <relational term> >|= **<constant>** Є

→ <relational expression> = <relational term> >|= **<integral literal>** Є

→ <relational expression> = **<relational term>** >|= 10 Є

→ <relational expression> = **<arithmetic expression>** >|= 10 Є

→ <relational expression> = **<arithmetic term>** >|= 10 Є

→ <relational expression> = **<arithmetic factor>** >|= 10 Є

→ <relational expression> = <base> **<exponent>** >|= 10 Є

→ <relational expression> = **<base>** Є >|= 10 Є

→ <relational expression> = **<constant>** Є >|= 10 Є

→ <relational expression> = **<integral literal>** Є >|= 10 Є

→ **<relational expression>** = 5 Є >|= 10 Є

→ **<relational term>** = 5 Є >|= 10 Є

→ <relational term> > **<arithmetic expression>** = 5 Є >|= 10 Є

→ <relational term> > **<arithmetic term>** = 5 Є >|= 10 Є

→ <relational term> > **<arithmetic factor>** = 5 Є >|= 10 Є

→ <relational term> > <base> **<exponent>** = 5 Є >|= 10 Є

→ <relational term> > **<base>** Є = 5 Є >|= 10 Є

→ <relational term> > **<constant>** Є = 5 Є >|= 10 Є

→ <relational term> > **<integral literal>** Є = 5 Є >|= 10 Є

→ **<relational term>** > 5 Є = 5 Є >|= 10 Є

→ **<arithmetic expression>** > 5 Є = 5 Є >|= 10 Є

→ **<arithmetic term>** > 5 Є = 5 Є >|= 10 Є

→ **<arithmetic factor>** > 5 Є = 5 Є >|= 10 Є

→ <base> **<exponent>** > 5 Є = 5 Є >|= 10 Є

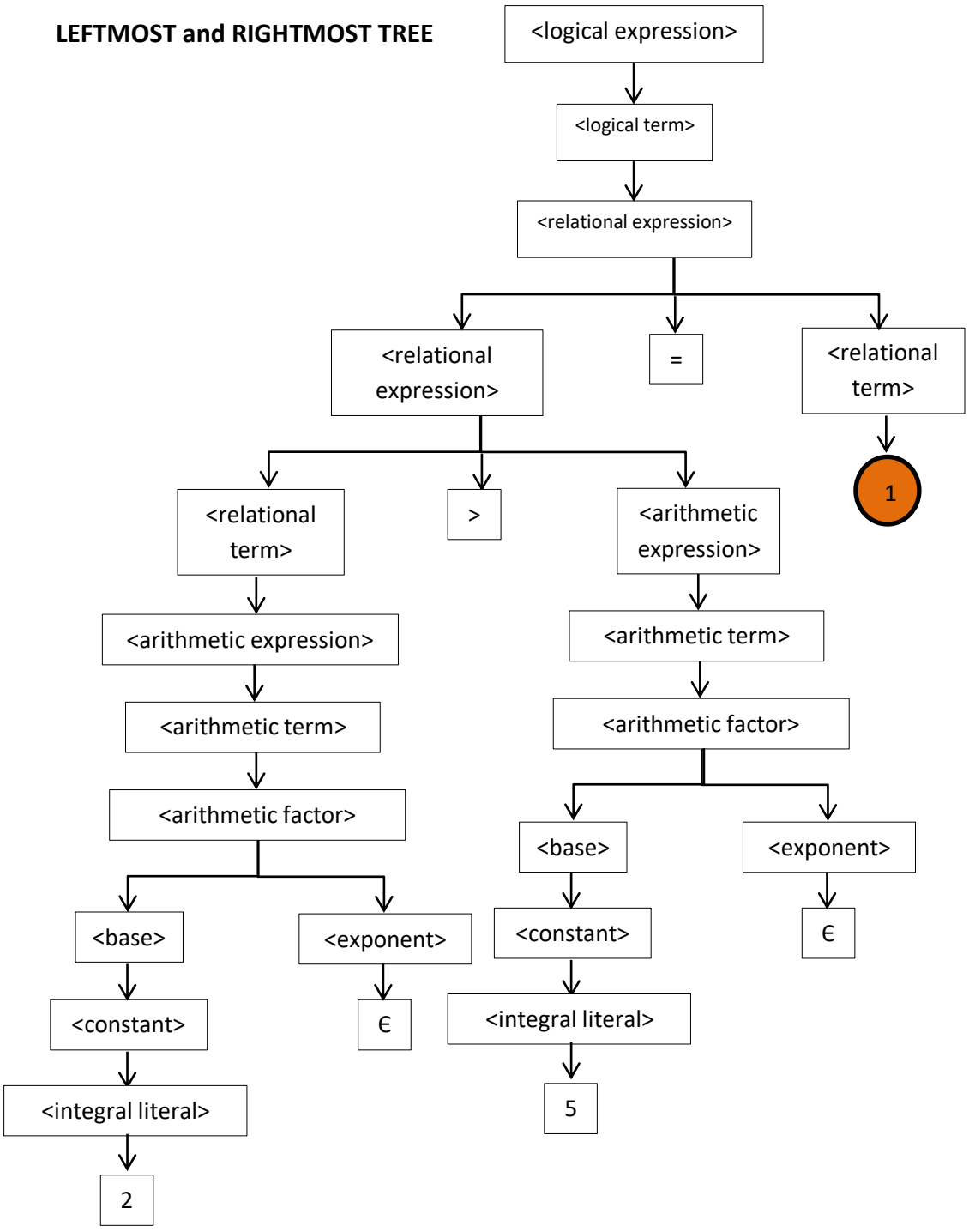
→ **<base>** Є > 5 Є = 5 Є >|= 10 Є

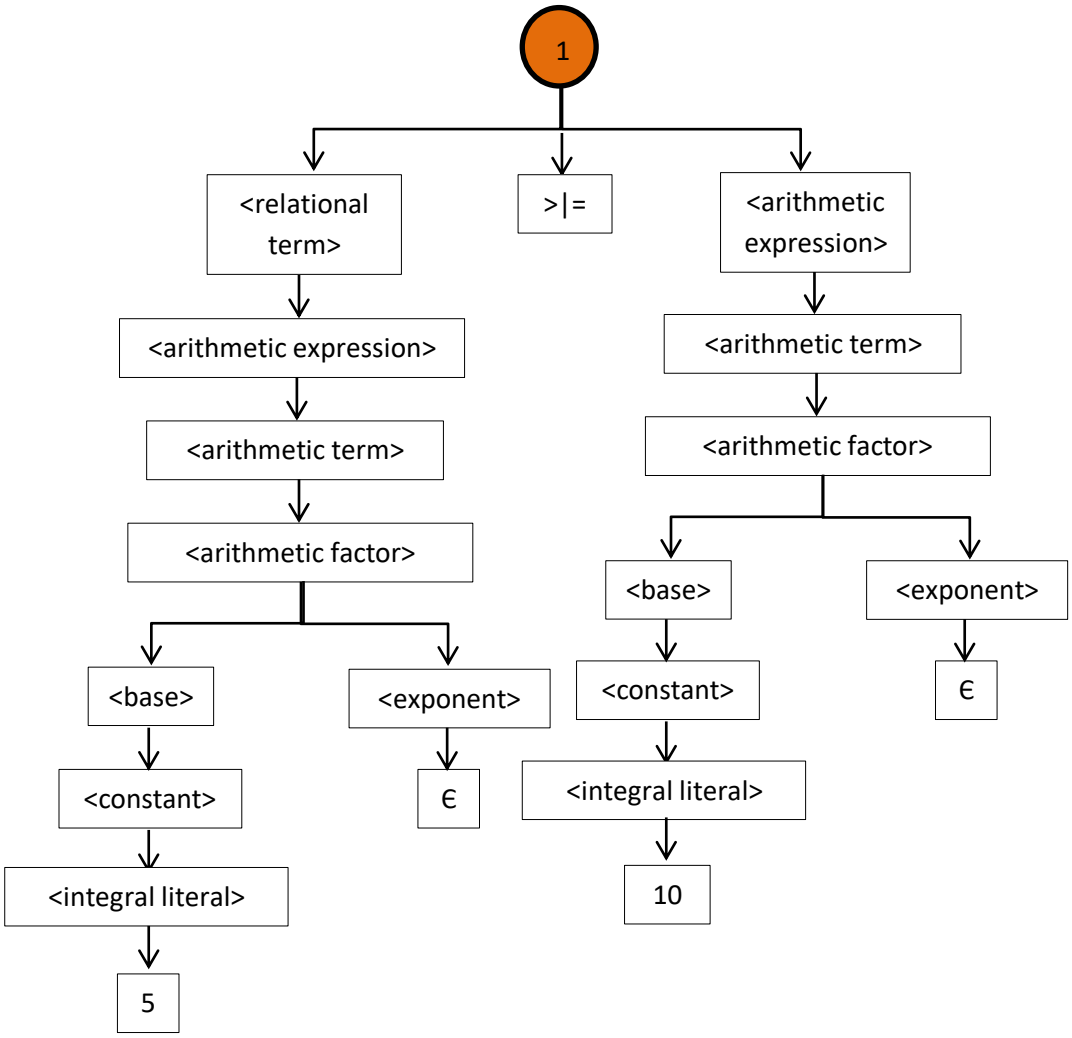
→ **<constant>** Є > 5 Є = 5 Є >|= 10 Є

→ **<integral literal>** Є > 5 Є = 5 Є >|= 10 Є

→ 2 Є > 5 Є = 5 Є >|= 10 Є

LEFTMOST and RIGHTMOST TREE




1
<relational term>
>|=
<arithmetic expression>
<arithmetic expression>
<arithmetic term>
<arithmetic term>
<arithmetic factor>
<arithmetic factor>
<base>
<exponent>
<base>
<exponent>
<constant>
Є
<constant>
Є
<integral literal>
<integral literal>
10
5

Example 20: 2+3*5/5+3

G(V) = {<logical expression>, <logical term>, <relational expression>, <relational term>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <exponent>, <base>, <constant>, <integral literal>}

G(T) = { 2, +, 3, *, 5, /, 5, +, 3, Є}

G(S) = {<logical expression>}

G(P)

<logical expression> ::= <logical term>

<logical term> ::= <relational expression>

<relational expression>::= <relational term>

<relational term> ::= <arithmetic expression>

<arithmetic expression> ::= <arithmetic expression> + <arithmetic term>
| <arithmetic term>

<arithmetic term> ::= < arithmetic term > * < arithmetic factor>
| <arithmetic term> / < arithmetic factor>
| <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<exponent> ::= Є

<base> ::= <constant>

<constant> ::= <integral literal>

<integral literal> ::= 2, 3, 5

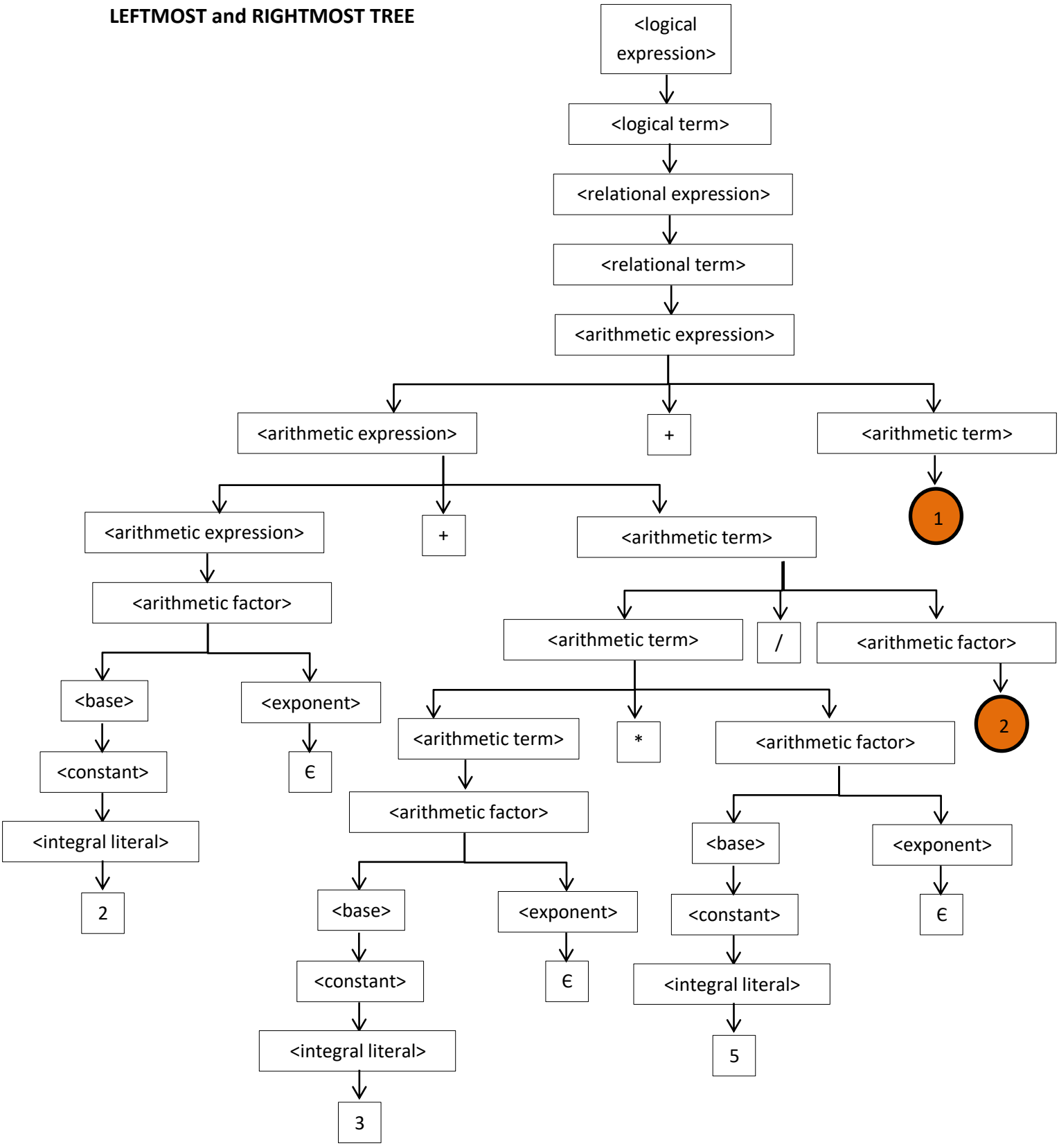
LEFTMOST DERIVATION

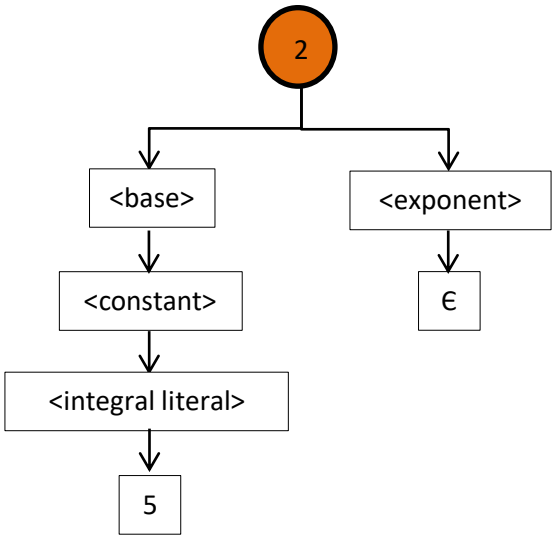
<logical expression> → **<logical term>**
→ **<relational expression>**
→ **<relational term>**
→ **<arithmetic expression>**
→ **<arithmetic expression>** + <arithmetic term>
→ **<arithmetic expression>** + <arithmetic term> + <arithmetic term>
→ **<arithmetic term>** + <arithmetic term> + <arithmetic term>
→ **<arithmetic factor>** + <arithmetic term> + <arithmetic term>
→ **<base>** <exponent> + <arithmetic term> + <arithmetic term>
→ **<constant>** <exponent> + <arithmetic term> + <arithmetic term>
→ **<integral literal>** <exponent> + <arithmetic term> + <arithmetic term>
→ 2 **<exponent>** + <arithmetic term> + <arithmetic term>
→ 2 Є + **<arithmetic term>** + <arithmetic term>
→ 2 Є + **<arithmetic term>** / < arithmetic factor> + <arithmetic term>
→ 2 Є + **<arithmetic term>** * < arithmetic factor> / < arithmetic factor> + <arithmetic term>
→ 2 Є + **<arithmetic factor>** * < arithmetic factor> / < arithmetic factor> + <arithmetic term>
→ 2 Є + **<base>** <exponent> * < arithmetic factor> / < arithmetic factor> + <arithmetic term>
→ 2 Є + **<integral literal>** <exponent> * < arithmetic factor> / < arithmetic factor> +
<arithmetic term>
→ 2 Є + 3 **<exponent>** * < arithmetic factor> / < arithmetic factor> + <arithmetic term>
→ 2 Є + 3 Є * **<arithmetic factor>** / < arithmetic factor> + <arithmetic term>
→ 2 Є + 3 Є * **<base>** <exponent> / < arithmetic factor> + <arithmetic term>
→ 2 Є + 3 Є * **<constant>** <exponent> / < arithmetic factor> + <arithmetic term>
→ 2 Є + 3 Є * **<integral literal>** <exponent> / < arithmetic factor> + <arithmetic term>
→ 2 Є + 3 Є * 5 **<exponent>** / < arithmetic factor> + <arithmetic term>
→ 2 Є + 3 Є * 5 Є / **<arithmetic factor>** + <arithmetic term>
→ 2 Є + 3 Є * 5 Є / **<base>** <exponent>+ <arithmetic term>
→ 2 Є + 3 Є * 5 Є / **<constant>** <exponent>+ <arithmetic term>
→ 2 Є + 3 Є * 5 Є / **<integral literal>** <exponent>+ <arithmetic term>
→ 2 Є + 3 Є * 5 Є / 5 **<exponent>** + <arithmetic term>
→ 2 Є + 3 Є * 5 Є / 5 Є + **<arithmetic term>**
→ 2 Є + 3 Є * 5 Є / 5 Є + **<arithmetic factor>**
→ 2 Є + 3 Є * 5 Є / 5 Є + **<base>** <exponent>
→ 2 Є + 3 Є * 5 Є / 5 Є + **<constant>** <exponent>
→ 2 Є + 3 Є * 5 Є / 5 Є + **<integral literal>** <exponent>
→ 2 Є + 3 Є * 5 Є / 5 Є + 3 **<exponent>**
→ 2 Є + 3 Є * 5 Є / 5 Є + 3 Є

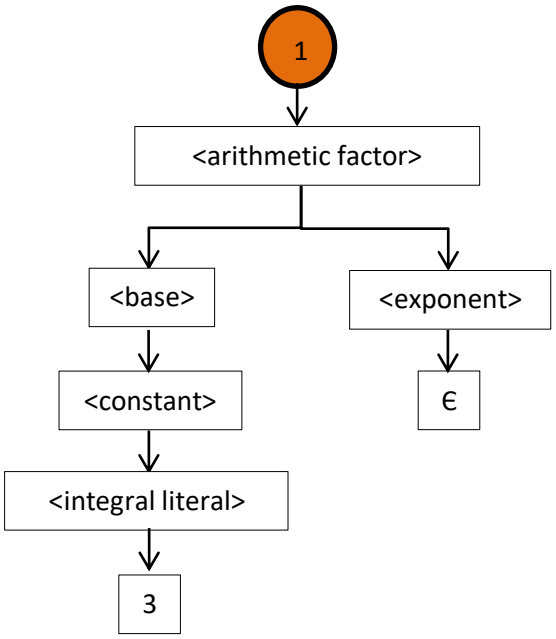
RIGHTMOST DERIVATION

<logical expression> → **<logical term>**
→ **<relational expression>**
→ **<relational term>**
→ **<arithmetic expression>**
→ <arithmetic expression> + **<arithmetic term>**
→ <arithmetic expression> + **<arithmetic factor>**
→ <arithmetic expression> + <base> **<exponent>**
→ <arithmetic expression> + **<base>** Є
→ <arithmetic expression> + **<constant>** Є
→ <arithmetic expression> + **<integral literal>** Є
→ **<arithmetic expression>** + 3 Є
→ <arithmetic expression> + **<arithmetic term>** + 3 Є
→ <arithmetic expression> + <arithmetic term> / **<arithmetic factor>** + 3 Є
→ <arithmetic expression> + <arithmetic term> / <base> **<exponent>** + 3 Є
→ <arithmetic expression> + <arithmetic term> / **<base>** Є + 3 Є
→ <arithmetic expression> + <arithmetic term> / **<constant>** Є + 3 Є
→ <arithmetic expression> + <arithmetic term> / **<integral literal>** Є + 3 Є
→ <arithmetic expression> + **<arithmetic term>** / 5 Є + 3 Є
→ <arithmetic expression> + < arithmetic term > * **<arithmetic factor>** / 5 Є + 3 Є
→ <arithmetic expression> + < arithmetic term > * <base> **<exponent>** / 5 Є + 3 Є
→ <arithmetic expression> + < arithmetic term > * **<base>** Є / 5 Є + 3 Є
→ <arithmetic expression> + < arithmetic term > * **<constant>** Є / 5 Є + 3 Є
→ <arithmetic expression> + < arithmetic term > * **<integral literal>** Є / 5 Є + 3 Є
→ <arithmetic expression> + **< arithmetic term >** * 5 Є / 5 Є + 3 Є
→ <arithmetic expression> + **<arithmetic factor>** * 5 Є / 5 Є + 3 Є
→ <arithmetic expression> + <base> **<exponent>** * 5 Є / 5 Є + 3 Є
→ <arithmetic expression> + **<base>** Є * 5 Є / 5 Є + 3 Є
→ <arithmetic expression> + **<constant>** Є * 5 Є / 5 Є + 3 Є
→ <arithmetic expression> + **<integral literal>** Є * 5 Є / 5 Є + 3 Є
→ **<arithmetic expression>** + 3 Є * 5 Є / 5 Є + 3 Є
→ **<arithmetic term>** + 3 Є * 5 Є / 5 Є + 3 Є
→ **<arithmetic factor>** + 3 Є * 5 Є / 5 Є + 3 Є
→ <base> **<exponent>** + 3 Є * 5 Є / 5 Є + 3 Є
→ **<base>** Є + 3 Є * 5 Є / 5 Є + 3 Є
→ **<constant>** Є + 3 Є * 5 Є / 5 Є + 3 Є
→ **<integral literal>** Є + 3 Є * 5 Є / 5 Є + 3 Є
→ 2 Є + 3 Є * 5 Є / 5 Є + 3 Є

LEFTMOST and RIGHTMOST TREE




2
<base>
<exponent>
<constant>
Є
<integral literal>
5


1
<arithmetic factor>
<base>
<exponent>
<constant>
Є
<integral literal>
3

Example 21: (‘x + 2) ^ ‘Y

G(V) = {<logical expression>, <logical term>, <relational expression>, <relational term>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <base>, <exponent>, <variable>, <literals>, <constant>}

G(T) = {‘x, ‘Y, 2, +, ^, (,), Є}

G(S) = {<logical expression>}

G(P)

<logical expression> ::= <logical term>

<logical term> ::= <relational expression>

<relational expression> ::= <relational term>

<relational term> ::= <arithmetic expression>

<arithmetic expression>::= <arithmetic term> |<arithmetic expression> + <arithmetic term>

<arithmetic term> ::= <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<base> ::= <variable> | <constant>| (<logical expression>)

<exponent> ::= ^ <base> <exponent> | Є

<literals> ::= <constant>

<constant> ::= <integral literal>

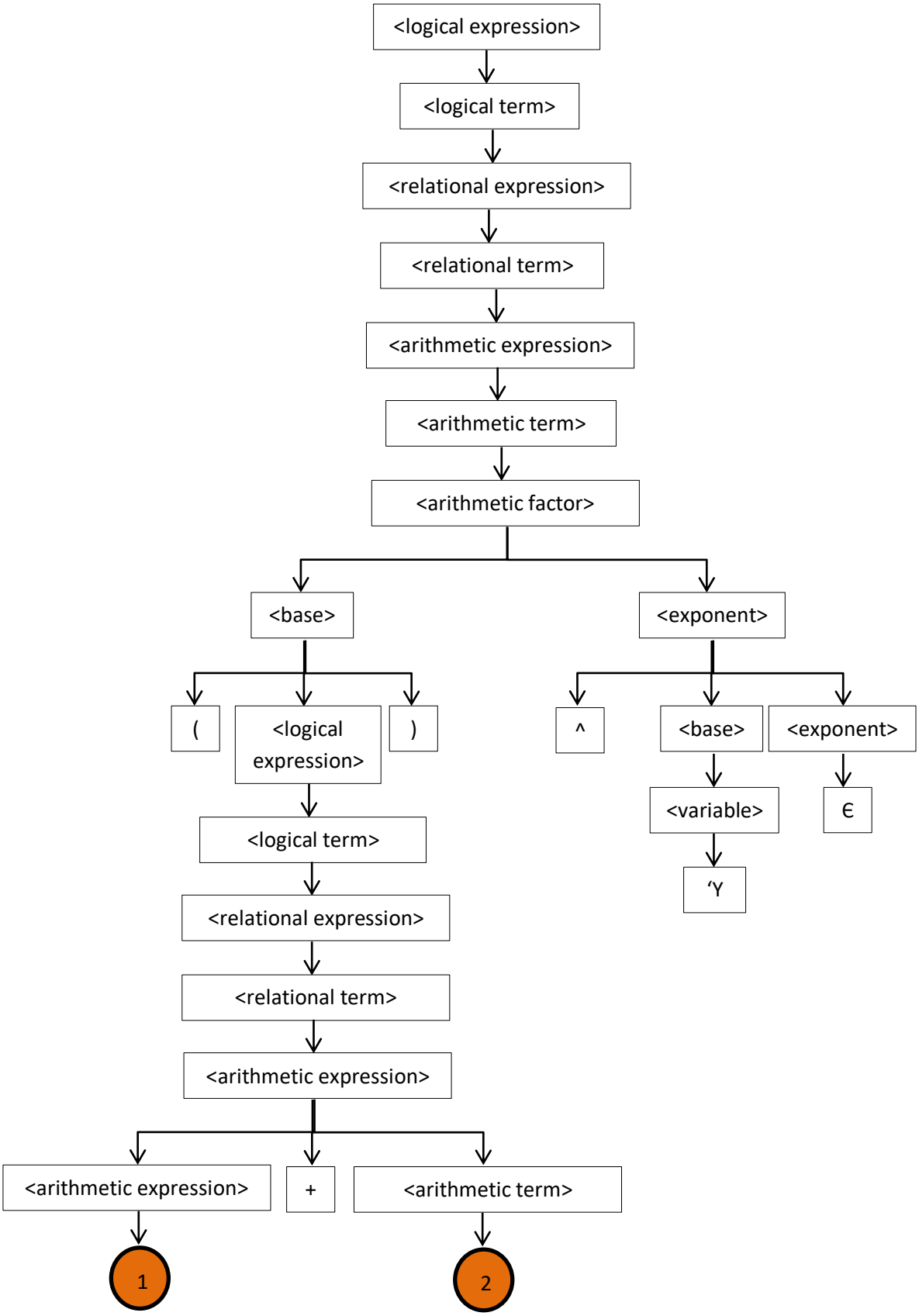
<integral literal> ::= 2

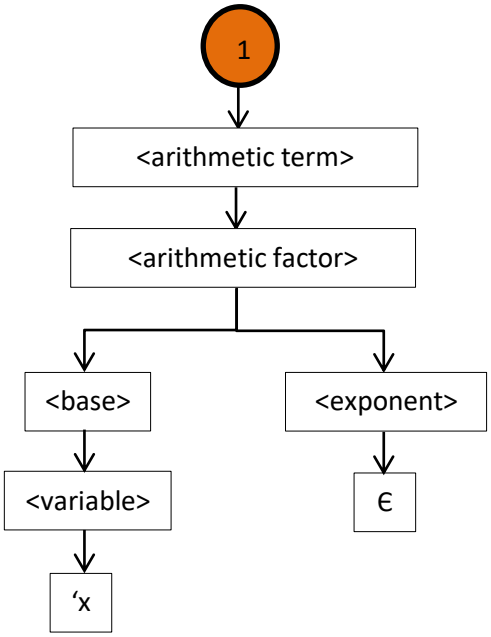
<variable> ::= ‘x, ‘Y

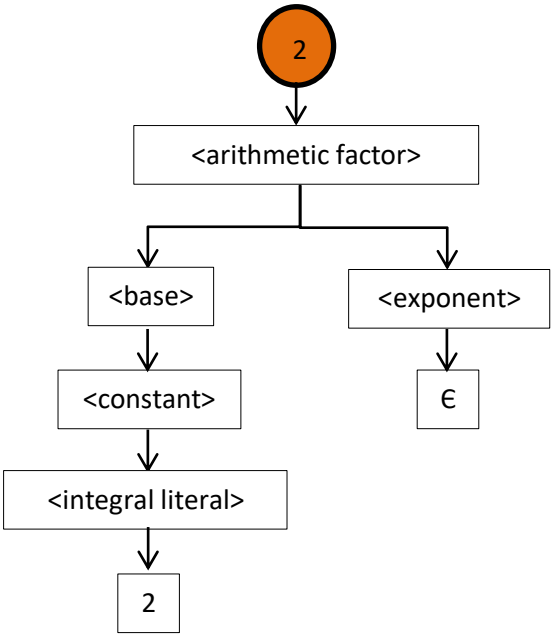
LEFTMOST DERIVATION

<logical expression> → **<logical term>**
→ **<relational expression>**
→ **<relational term>**
→ **<arithmetic expression>**
→ **<arithmetic term>**
→ **<arithmetic factor>**
→ **<base>** <exponent>
→ (**<logical expression>**) <exponent>
→ (**<logical term>**) <exponent>
→ (**<relational expression>**)<exponent>
→ (**<relational term>**)<exponent>
→ (**<arithmetic expression>**)<exponent>
→ (**<arithmetic expression>** + <arithmetic term>) <exponent>
→ (**<arithmetic term>** + <arithmetic term>) <exponent>
→ (**<arithmetic factor>** + <arithmetic term>) <exponent>
→ (**<base>** <exponent> + <arithmetic term>) <exponent>
→ (**<variable>** <exponent> + <arithmetic term>) <exponent>
→ (‘x **<exponent>** + <arithmetic term>) <exponent>
→ (‘x Є + **<arithmetic term>**) <exponent>
→ (‘x Є + **<arithmetic factor>**) <exponent>
→ (‘x Є + **<base>** <exponent>) <exponent>
→ (‘x Є + **<constant>** <exponent>) <exponent>
→ (‘x Є + **<integral literal>** <exponent>) <exponent>
→ (‘x Є + 2 **<exponent>**) <exponent>
→ (‘x Є + 2 Є) **<exponent>**
→ (‘x Є + 2 Є) ^ **<base>** <exponent>
→ (‘x Є + 2 Є) ^ **<variable>** <exponent>
→ (‘x Є + 2 Є) ^ ‘Y **<exponent>**
→ (‘x Є + 2 Є) ^ ‘Y Є

LEFTMOST TREE



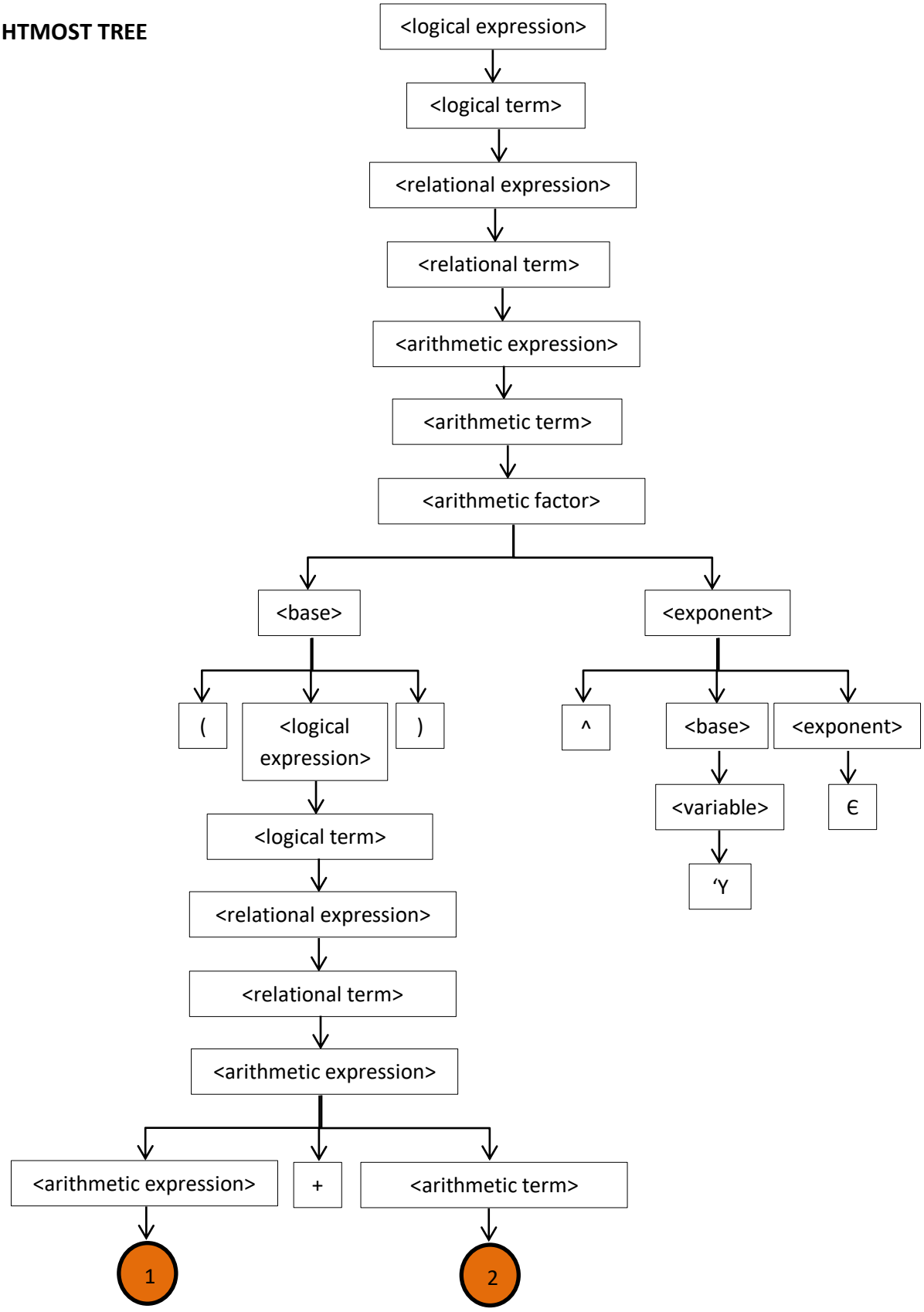

1
<arithmetic term>
<arithmetic factor>
<base>
<exponent>
<variable>
Є
‘x

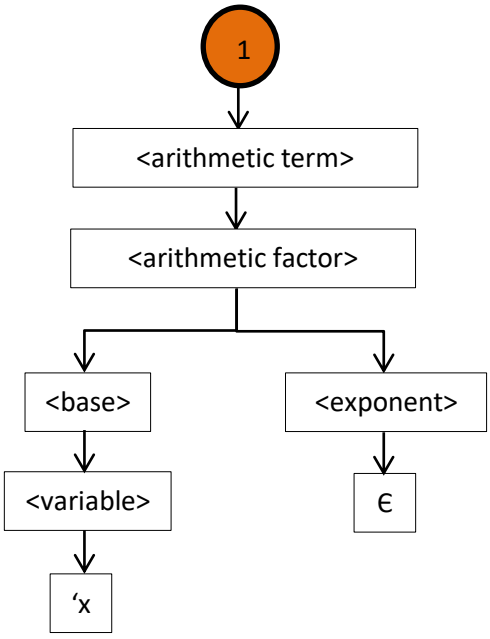

2
<arithmetic factor>
<base>
<exponent>
<constant>
Є
<integral literal>
2

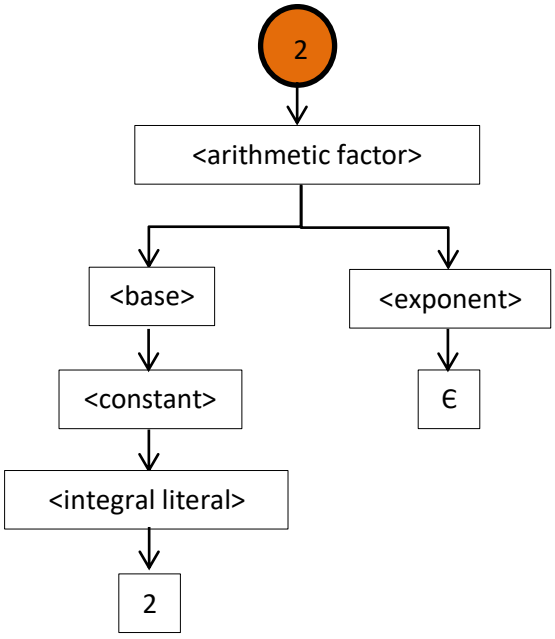
RIGHTMOST DERIVATION

<logical expression> → **<logical term>**
→ **<relational expression>**
→ **<relational term>**
→ **<arithmetic expression>**
→ **<arithmetic term>**
→ **<arithmetic factor>**
→ <base> **<exponent>**
→ <base> ^ <base> **<exponent>**
→ <base> ^ **<base>** Є
→ <base> ^ **<variable>** Є
→ **<base>** ^ ‘Y Є
→ (**<logical expression>**) ^ ‘Y Є
→ (**<logical term>**) ^ ‘Y Є
→ (**<relational expression>**) ^ ‘Y Є
→ (**<relational term>**) ^ ‘Y Є
→ (**<arithmetic expression>**) ^ ‘Y Є
→ (<arithmetic expression> + **<arithmetic term>**) ^ ‘Y Є
→ (<arithmetic expression> + **<arithmetic factor>**) ^ ‘Y Є
→ (<arithmetic expression> + <base> **<exponent>**) ^ ‘Y Є
→ (<arithmetic expression> + **<base>** Є) ^ ‘Y Є
→ (<arithmetic expression> + **<constant>** Є) ^ ‘Y Є
→ (<arithmetic expression> + **<integral literal>** Є) ^ ‘Y Є
→ (**<arithmetic expression>** + 2 Є) ^ ‘Y Є
→ (**<arithmetic term>** + 2 Є) ^ ‘Y Є
→ (**<arithmetic factor>** + 2 Є) ^ ‘Y Є
→ (<base> **<exponent>** + 2 Є) ^ ‘Y Є
→ (**<base>** Є+ 2 Є) ^ ‘Y Є
→ (**<variable>** Є+ 2 Є) ^ ‘Y Є
→ (‘x Є+ 2 Є) ^ ‘Y Є

RIGHTMOST TREE




1
<arithmetic term>
<arithmetic factor>
<base>
<exponent>
<variable>
Є
‘x


2
<arithmetic factor>
<base>
<exponent>
<constant>
Є
<integral literal>
2

Example 22: 10/'Y* ('x-'z)

G(V) = {<logical expression>, <logical term>, <relational expression>, <relational term>, <arithmetic expression>, <arithmetic term>, <arithmetic factor>, <base>, <exponent>, <variable>, <literals>, <constant>}

G(T) = {'x, 'Y, 2, +, ^, (,), Є}

G(S) = {<logical expression>}

G(P)

<logical expression> ::= <logical term>

<logical term> ::= <relational expression>

<relational expression> ::= <relational term>

<relational term> ::= <arithmetic expression>

<arithmetic expression>::= <arithmetic expression> + <arithmetic term>

| <arithmetic term>

<arithmetic term> ::= < arithmetic term > * <arithmetic factor>

| <arithmetic term> / <arithmetic factor>

| <arithmetic factor>

<arithmetic factor> ::= <base> <exponent>

<base> ::= <variable> | <constant>| (<logical expression>)

<exponent> ::= ^ <base> <exponent> | Є

<literals> ::= <constant>

<constant> ::= <integral literal>

<integral literal> ::= 10

<variable> ::= 'x,'Y, 'z

LEFTMOST DERIVATION

<logical expression> → **<logical term>**

→ **<relational expression>**

→ **<relational term>**

→ **<arithmetic expression>**

→ **<arithmetic term>**

→ **<arithmetic term>** * <arithmetic factor>

→ **<arithmetic term>** / <arithmetic factor> * <arithmetic factor>

→ **<arithmetic factor>** / <arithmetic factor> * <arithmetic factor>

→ **<base>** <exponent> / <arithmetic factor> * <arithmetic factor>

→ **<constant>** <exponent> / <arithmetic factor> * <arithmetic factor>

→ **<integral literal>**<exponent> / <arithmetic factor> * <arithmetic factor>

→ 10 **<exponent>** / <arithmetic factor> * <arithmetic factor>

→ 10 Є / **<arithmetic factor>** * <arithmetic factor>

→ 10 Є / **<base>** <exponent> * <arithmetic factor>

→ 10 Є / **<variable>** <exponent> * <arithmetic factor>

→ 10 Є / ‘Y **<exponent>** * <arithmetic factor>

→ 10 Є / ‘Y Є* **<arithmetic factor>**

→ 10 Є / ‘Y Є* **<base> <exponent>**

→ 10 Є / ‘Y Є* (**<logical expression>**) <exponent>

→ 10 Є / ‘Y Є* (**<logical term>**) <exponent>

→ 10 Є / ‘Y Є* (**<relational expression>**) <exponent>

→ 10 Є / ‘Y Є* (**<relational term>**) <exponent>

→ 10 Є / ‘Y Є* (**<arithmetic expression>**) <exponent>

→ 10 Є / ‘Y Є* (**<arithmetic expression>** - <arithmetic term>) <exponent>

→ 10 Є / ‘Y Є* (**<arithmetic term>** - <arithmetic term>) <exponent>

→ 10 Є / ‘Y Є* (**<arithmetic factor>** - <arithmetic term>) <exponent>

→ 10 Є / ‘Y Є* (**<base>** <exponent> - <arithmetic term>) <exponent>

→ 10 Є / ‘Y Є* (**<variable>** <exponent> - <arithmetic term>) <exponent>

→ 10 Є / ‘Y Є* (‘x **<exponent>** - <arithmetic term>) <exponent>

→ 10 Є / ‘Y Є* (‘x Є - **<arithmetic term>**) <exponent>

→ 10 Є / ‘Y Є* (‘x Є - **<arithmetic factor>**) <exponent>

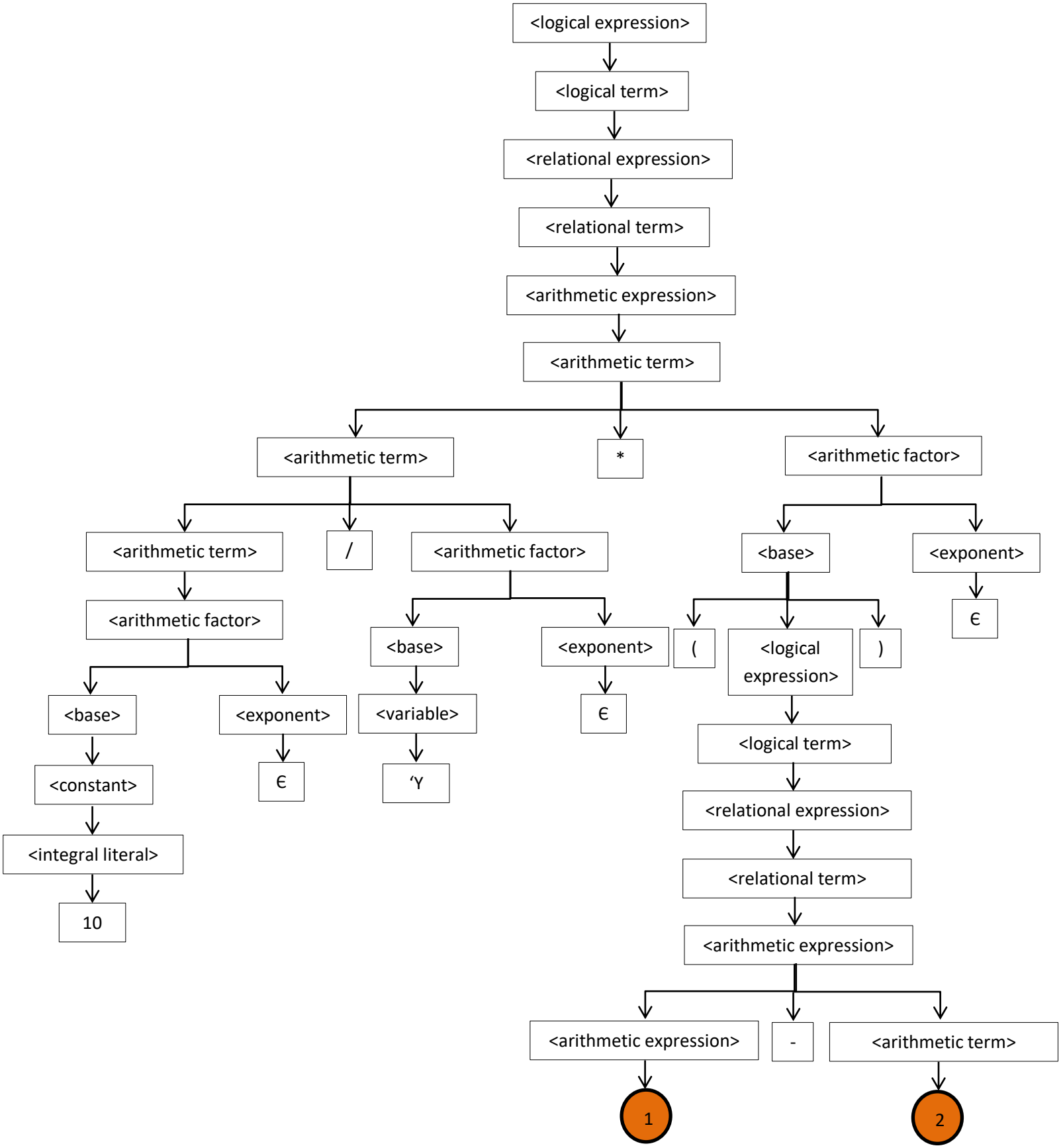
→ 10 Є / ‘Y Є* (‘x Є - **<base>** <exponent>) <exponent>

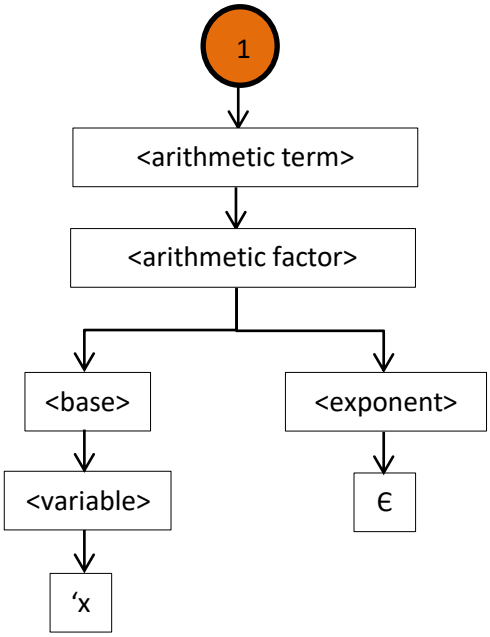
→ 10 Є / ‘Y Є* (‘x Є - **<variable>** <exponent>) <exponent>

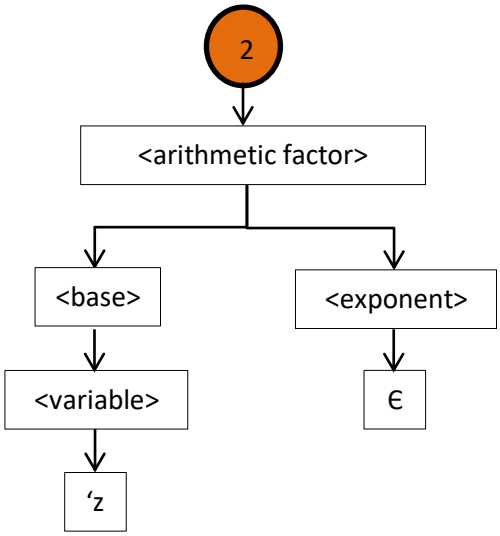
→ 10 Є / ‘Y Є* (‘x Є – ‘z **<exponent>**) <exponent>

→ 10 Є / ‘Y Є* (‘x Є – ‘z Є) **<exponent>**

→ 10 Є / ‘Y Є* (‘x Є – ‘z Є) Є


<logical expression>
<logical term>
<relational expression>
<relational term>
<arithmetic expression>
<arithmetic term>
<arithmetic term>
*
<arithmetic factor>
<arithmetic term>
/
<arithmetic factor>
<base>
<exponent>
<arithmetic factor>
<base>
<exponent>
(
<logical expression>
)
ϵ
<base>
<exponent>
<variable>
ϵ
<constant>
ϵ
'y
<integral literal>
10
<logical term>
<relational expression>
<relational term>
<arithmetic expression>
<arithmetic expression>
-
<arithmetic term>
1
2


1
<arithmetic term>
<arithmetic factor>
<base>
<exponent>
<variable>
Є
‘x


2
<arithmetic factor>
<base>
<exponent>
<variable>
Є
‘z

RIGHTMOST DERIVATION

<logical expression> → **<logical term>**

→ **<relational expression>**

→ **<relational term>**

→ **<arithmetic expression>**

→ **<arithmetic term>**

→ <arithmetic term> * **<arithmetic factor>**

→ <arithmetic term> * <base> **<exponent>**

→ <arithmetic term> * (**<logical expression>**) Є

→ <arithmetic term> * (**<logical term>**) Є

→ <arithmetic term> * (**<relational expression>**)Є

→ <arithmetic term> * (**<relational term>**) Є

→ <arithmetic term> * (**<arithmetic expression>**) Є

→ <arithmetic term> * (<arithmetic expression> - **<arithmetic term>**)Є

→ <arithmetic term> * (<arithmetic expression> - **<arithmetic factor>**)Є

→ <arithmetic term> * (<arithmetic expression> - <base> **<exponent>**)Є

→ <arithmetic term> * (<arithmetic expression> - **<base>** Є)Є

→ <arithmetic term> * (<arithmetic expression> - **<variable>** Є)Є

→ <arithmetic term> * (**<arithmetic expression>** - ‘z Є)Є

→ <arithmetic term> * (**<arithmetic term>** - ‘z Є)Є

→ <arithmetic term> * (**<arithmetic factor>** - ‘z Є)Є

→ <arithmetic term> * (<base> **<exponent>** - ‘z Є)Є

→ <arithmetic term> * (**<base>** Є- ‘z Є)Є

→ <arithmetic term> * (**<variable>** Є- ‘z Є)Є

→ **<arithmetic term>** * (‘x Є- ‘z Є)Є

→ <arithmetic term> / **<arithmetic factor>*** (‘x Є- ‘z Є)Є

→ <arithmetic term> / <base> **<exponent>** * (‘x Є- ‘z Є)Є

→ <arithmetic term> / **<base>** Є * (‘x Є- ‘z Є)Є

→ <arithmetic term> / **<variable>** Є * (‘x Є- ‘z Є)Є

→ **<arithmetic term>** / ‘Y Є * (‘x Є- ‘z Є)Є

→ **<arithmetic factor>** / ‘Y Є * (‘x Є- ‘z Є)Є

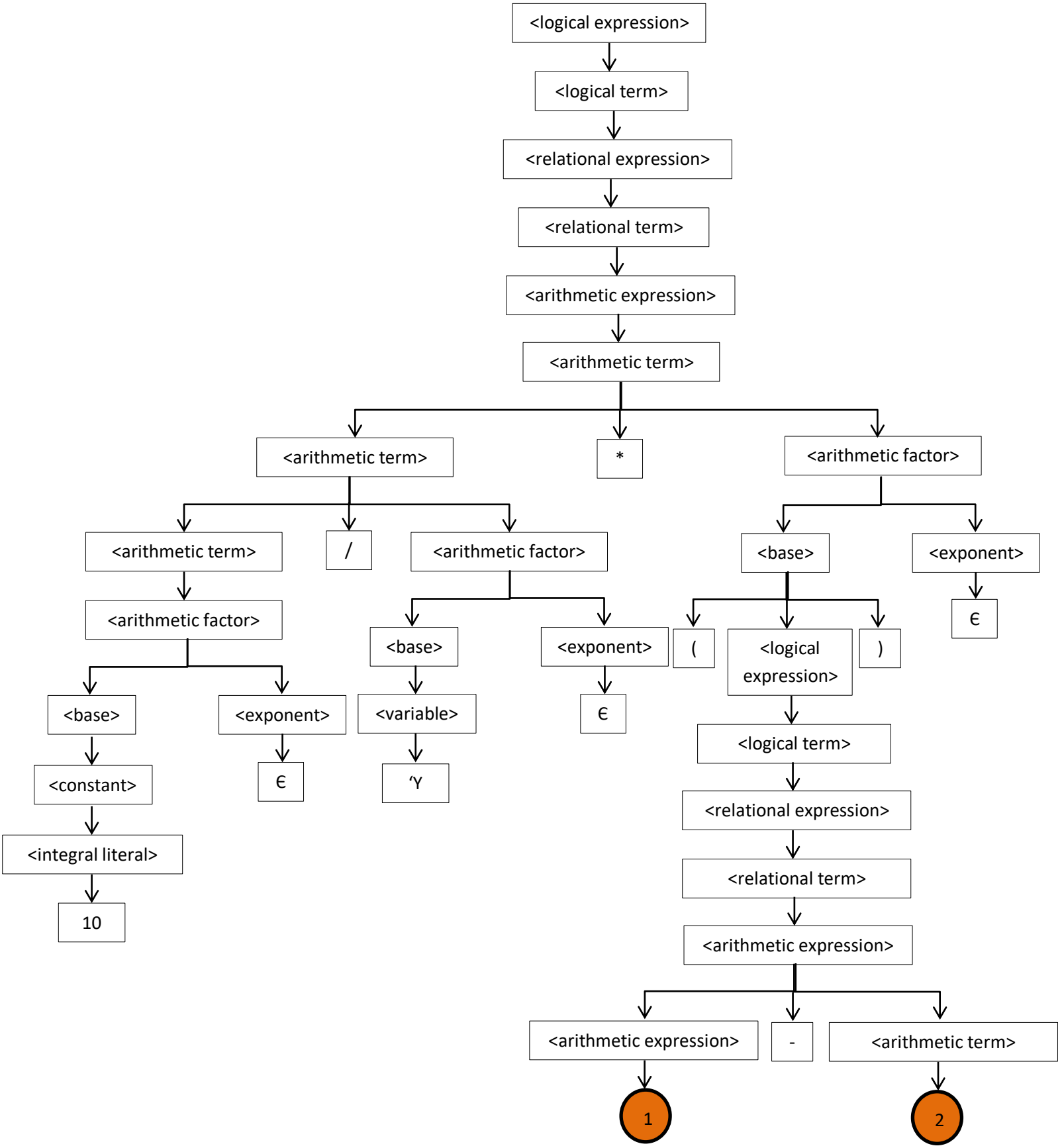
→ <base> **<exponent>** / ‘Y Є * (‘x Є- ‘z Є)Є

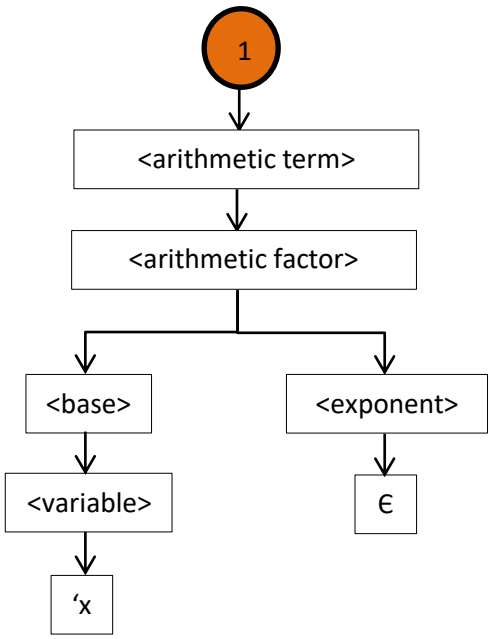
→ **<base>** Є/ ‘Y Є * (‘x Є- ‘z Є)Є

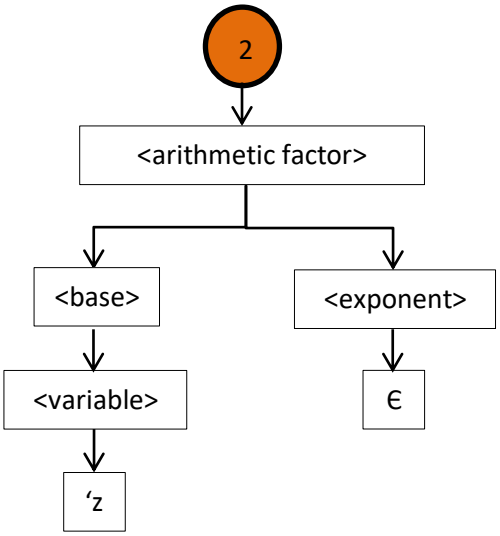
→ **<constant>** Є/ ‘Y Є * (‘x Є- ‘z Є)Є

→ **<integral literal>** Є/ ‘Y Є * (‘x Є- ‘z Є)Є

→ 10 Є/ ‘Y Є * (‘x Є- ‘z Є)Є


<logical expression>
<logical term>
<relational expression>
<relational term>
<arithmetic expression>
<arithmetic term>
<arithmetic term>
*
<arithmetic factor>
<arithmetic term>
/
<arithmetic factor>
<base>
<exponent>
<arithmetic factor>
<base>
<exponent>
(
<logical expression>
)
ϵ
<base>
<exponent>
<variable>
ϵ
<constant>
ϵ
'y
<integral literal>
10
<logical term>
<relational expression>
<relational term>
<arithmetic expression>
<arithmetic expression>
-
<arithmetic term>
1
2


1
<arithmetic term>
<arithmetic factor>
<base>
<exponent>
<variable>
Є
‘x


2
<arithmetic factor>
<base>
<exponent>
<variable>
Є
‘z