Introduction to Syntax Analysis

CSE 340 FALL 2021

Rida Bazzi

Syntax in the English Language

Example text 1: The cat drank the milk.

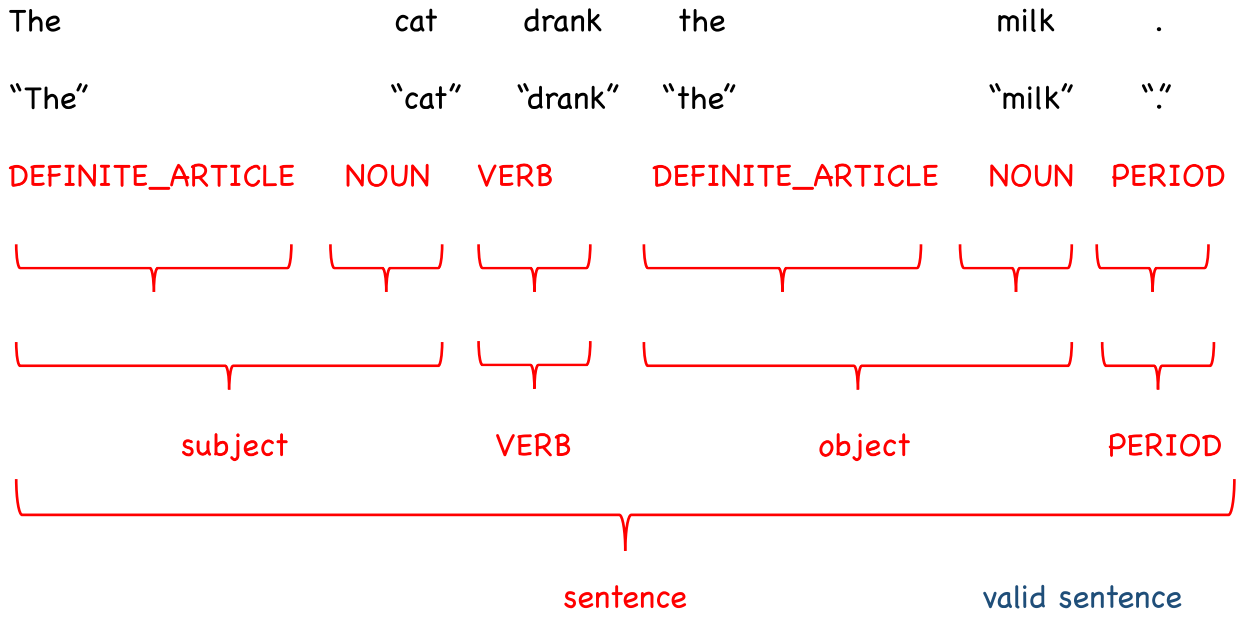
How do we make sense of this text?

1. We identify the sequence of words, and, for each word, we determine what kind or category of word it is

"The"	DEFINITE_ARTICLE
"cat"	NOUN
"drank"	VERB
"the"	DEFINITE_ARTICLE
"milk"	NOUN
"."	PERIOD

(note that I am treating the period as a word).

2. We try to determine if the sequence of words is syntactically correct as a sentence.



The cat drank the milk .

"The" "cat" "drank" "the" "milk" "."

DEFINITE_ARTICLE NOUN VERB DEFINITE_ARTICLE NOUN PERIOD

subject VERB object PERIOD

sentence

valid sentence

The process of taking some text, breaking it into words, identifying the categories of words and determining that they form a syntactically correct sequence is called PARSING!

Parsing is done according to a ***grammar***. In the example above, I did not give the grammar explicitly and relied on the shared knowledge you and I have of the grammar of the English language.

Syntax in the English Language: Grammar Rules

Example text 1: The milk drank the milk.

This is also a grammatically valid sentence because it looks the same as the previous sentence:

DEFINITE_ARTICLE NOUN VERB DEFINITE_ARTICLE NOUN PERIOD

This is the same format (syntax) as the previous sentence.

What matters for syntax are the categories of words and not the specific words. The sentence above has the correct look, even though it does not make sense semantically.

To summarize, parsing a sentence involves the following

1. identify the words
2. identify the kinds or categories of words
3. group together the "kinds" according to the English grammar to identify sentence components
4. Identify the sentence by putting the components together

Syntax in Programming Languages

The syntax of programming languages is defined in a way that is similar to the way we define syntax of natural languages

For a programming languages we need to have a well defined-alphabet and define tokens over the alphabet. We start by defining alphabets and strings.

Alphabet a finite set of symbols

Example { a , b , & , 1 , ! }

Example The alphabet of the C language =

{ a , b , … , z , A , B , … Z , 0 , 1 , … 9 , < , … }

String A string over a given alphabet is a finite sequence of symbols from the alphabet

Example "ab" , "a&b1" , and "1!aa" are strings over the the alphabet above

Note I enclose strings between double quotation marks to emphasize that they are strings

Note Strings are sequences of symbols, so order matter. "abc" is not equivalent to "bac"

Elements of Syntax

Token A token is a set of strings. We can think of a token as the name of a set of strings.

Example NUM = { "0" , "1" , "2" , … , "9" , "10" , "11", "12" , … }

Example ID = set of strings consisting of 1 or more characters and digits and starting with a character

Example DECIMAL = set of strings of the form NUM `.' NUM or `.' NUM or NUM `.'

Lexeme A lexeme of a token is a string from the set strings labeled by the token

Example "123" is a lexeme of NUM

Example "1.00" is a lexeme of DECIMAL

Example "0.00100" is a lexeme of DECIMAL

Example "0.00" is a lexeme of DECIMAL

Oftentimes we abuse notation and refer to the lexeme as the token. For example, we say "123" is a NUM token

Note that tokens correspond to word categories (for natural languages). Each token defines a category.

Lexemes correspond to words in a particular category.

Syntax vs Semantics

Syntax vs. Semantics in English

Syntax only depends on the categories of words and not on the specific word used. Semantics do not depend just on the categories of words (NOUN, ARTICLE, …), but they depend on the specific word that is used. In the example

The milk drank the cat.

the sentence does not make sense because "milk" is not a semantically meaningful subject for the verb "drank". For now, we will concentrate on syntax and not semantics.

Syntax vs. Semantics in the C language

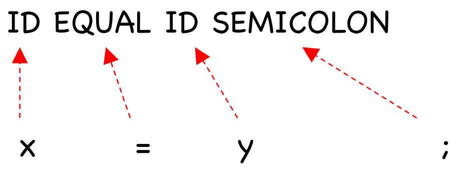
Consider the following program fragment

```
int x;
float y;

x = y;
```

This fragment is syntactically correct. From a syntax point of view, it is seen as

```
ID ID SEMICOLON
ID ID SEMICOLON
```



ID EQUAL ID SEMICOLON

x = y ;

Where ID is the category that contains all valid identifiers (valid variable names). Even though the assignment is syntactically correct, it is not valid semantically, because C does not allow a float value to be assigned to an integer variable.

The syntax is correct, but the semantics are not.

In general, syntax does not care about the specific word (or lexeme) but only about its category. When we disallow some constructs because of the specific word used, we are dealing with semantics, but the separation is not clear-cut. One can introduce categories that have only one word.

getToken() function

Given a list of tokens for a programming languages, we are interested in breaking the input into a sequence of tokens (and identify the corresponding lexemes). In this class we will use the getToken() function for this task.

getToken() takes input from standard input

returns a struct that has two fields (the struct type is token a type that I declare in the code I will provide you)

- token_type this is the category or kind (the token in our terminology)
- lexeme this is the actual part of the input that corresponds to the token_type identified by getToken()

Calling getToken() repeatedly will give us the sequence of tokes in the input. When getToken() is called, it "consumes" the part of the input that corresponds to the token and next time it is called, it starts after the token that was already consumed.

Depending on the language, getToken() might also ignore some parts of the input, such as space characters, which are typically treated as separators and are otherwise skipped over.

getToken() function

I will give two examples to make the behavior of getToken() clearer and to further specify it

Example 1 token list:

ID
EQUAL
SEMICOLON

input:

```
x = y;
```

If we call getToken() repeatedly, we get the following (the struct returned by getToken() is represented as a pair, with the first element being the token_type and the second element being the lexeme).

1. getToken() ⟶ (ID, "x")
2. getToken() ⟶ (EQUAL, "")
3. getToken() ⟶ (ID , "y")
4. getToken() ⟶ (SEMICOLON , "")
5. getToken() ⟶ (EOF , "")
6. getToken() ⟶ (EOF , "")

Some explanation of the returned values is needed:

1. getToken() starts reading the input from the very beginning. The first token it finds is ID whose lexeme is "x"
2. After the first call to getToken(), the second call starts at the space after the x. In our example, the space is ignored (as is usually the case with many programming languages) and the next token to be returned is the EQUAL. We note here that the lexeme field is given as the empty string. In reality, the lexeme is "=" but in my implementation I return the empty string for the lexeme because, for the EQUAL token, all the information about the lexeme is available in the token_type
3. the next call to getToken() starts after the =. The space is ignored and the next token returned is ID whose lexeme is "y"
4. The next call to getToken() returns SEMICOLON. Again, in the implementation, I have the lexeme as the empty string because all the information about the lexeme is in the token_type SEMICOLON.
5. The last two calls (5 and 6) are interesting. After reading the semicolon, we run out of input. The getToken() function must have a way to indicate that there is no more input. This is represented by returned a struct whose token_type is EOF which represents end of input is reached.

getToken() function

Example 2 token list:

NUM
DOT
DECIMAL which I define to be NUM.NUM

input:

1.1..1

If we call getToken() repeatedly, we get the following :

1. getToken() ⟶ (DECIMAL, "1.1")
2. getToken() ⟶ (DOT, "")
3. getToken() ⟶ (DOT, "")
4. getToken() ⟶ (NUM, "1")
5. getToken() ⟶ (EOF , "")

An explanation of the first returned value is needed. getToken() starts reading the input from the very beginning. The question is: why is DECIMAL returned and not NUM? The reason is that getToken() should always try to get the token with the longest possible lexeme. So, given a list of tokens, when getToken() is called from a given point in the input, the following two rules need to be followed:

- **Longest prefix match**: The token with the longest possible lexeme is returned
- **Priority for tokens that are listed first in the list**: If there are multiple possible tokens with longest lexeme, then the one that is listed first in the list of tokens is returned

Example reserved words and identifiers

Typically, in programming languages, lexemes for reserved words are also lexemes for identifiers (ID). For that reason, reserved words are listed before identifiers in the list of tokens. This ensures that when the lexeme matches a reserved word, the token for the reserved word is returned. For example, for the following input:

if1 ifif if 1

The list of tokens is (ID, "if1") (ID, "ifif") (IF, "") (NUM, "1")

peek()

Another functions that is useful in parsing is peek() which allows us to look ahead without consuming tokens.

To describe peek(), we assume that the input is already broken down into tokens (token and lexemes) and that the whole list of tokens is in an array, which I will call token_array:

(tt_1,lex_1) (tt_2,lex_2) (tt_3,lex_3) … (tt_i,lex_i) (tt_{i+1},lex_{i+1}) ….

The i'th token is token[i]. In the description I assume that the index i starts at 1. As we execute getToken() the index changes to reflect the fact that tokens are consumed. With this representation, the functions getToken() and peek() can be defined as follows.

getToken() this function will simply returns the next token and increments the index. It is defined as follows

```
Token getToken()            if index > number of tokens
                                      return EOF
                            else
                                      tok = token_array[index];
                                      index = index + 1;     // token is consumed
                                      return tok;
```

peek() sometimes, we want to look at the next token or the token after the next token, but without consuming them. The function peek() allows us to do so. The function peek() takes an integer argument that specifies how far ahead to peek. It is defined as follows.

```
// Argument howfar > 0.
// Behavior is not defined if howfar ≤ 0
Token peek(int howfar)      if index+howfar -1 > number of tokens
                                      return EOF
                            else
                                      return token_array[index+howfar-1]
```

Remember that index points to the next token, so peek(1) returns token_array[index+1-1] = token_array[index]. In particular getToken() and peek(1) will both return the same token but getToken() modifies the index and peek() does not modify the index.

Example of getToken() and Peek() together

We consider the following token list

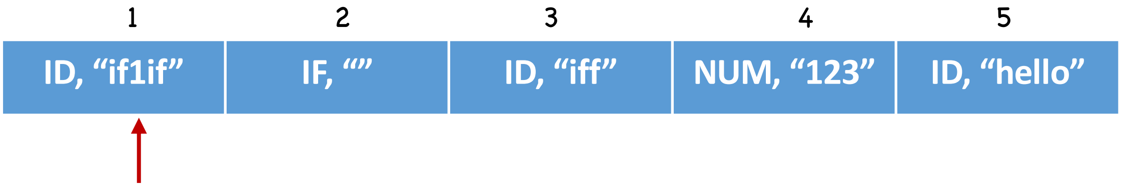
```
IF        = { "if"}
ID                              // previously defined
NUM                             // previously defined
```

We also assume that whitespace characters such as the space character, tab and newline are separators of tokens. This means that whitespace characters cannot be part of a token and that when a whitespace character is encountered by the function getToken(), it has to stop and return the longest matching prefix. The next call to getToken() will skip the whitespace characters.

Input

```
if1if  if  iff  123hello
```

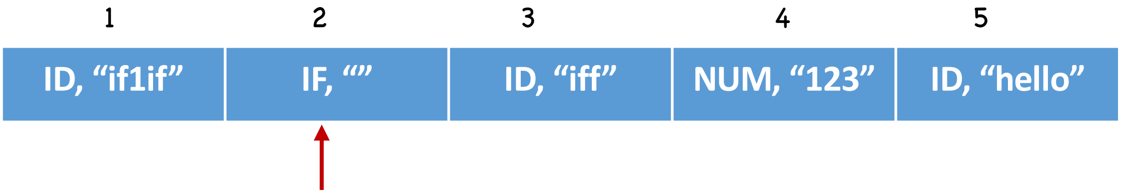
The sequence of tokens for this example is



1	2	3	4	5
ID, "if1if"	IF, ""	ID, "iff"	NUM, "123"	ID, "hello"

initially *index* is 1 which means that the first token that will be read is the first token in the array. Let us examine the behavior of getToken() and peek() through a sequence of calls.

1. getToken()   will return (ID, "if1if") and index is incremented to 2. The resulting figure is



1	2	3	4	5
ID, "if1if"	IF, ""	ID, "iff"	NUM, "123"	ID, "hello"

notice how the arrow is pointing to the second entry (index is 2). This is the next token to read. So, if we call peek(1) we get the token at index 2. If we call peek(2), we get the token after that and so on. The rest of the example follows:

2. peek(1)   will return (IF, "")   index is still 2
3. peek(2)   will return (ID, "iff")   index is still 2
4. pek(1)   will return (IF, "")   index is still 2
5. getToken()   will return (IF, "")   now index is incremented to 3
6. peek(1)   will return (ID, "iff")   index is still 3
7. peek(4)   will return (EOF, "")   index is still 3

Syntax in Programming Languages

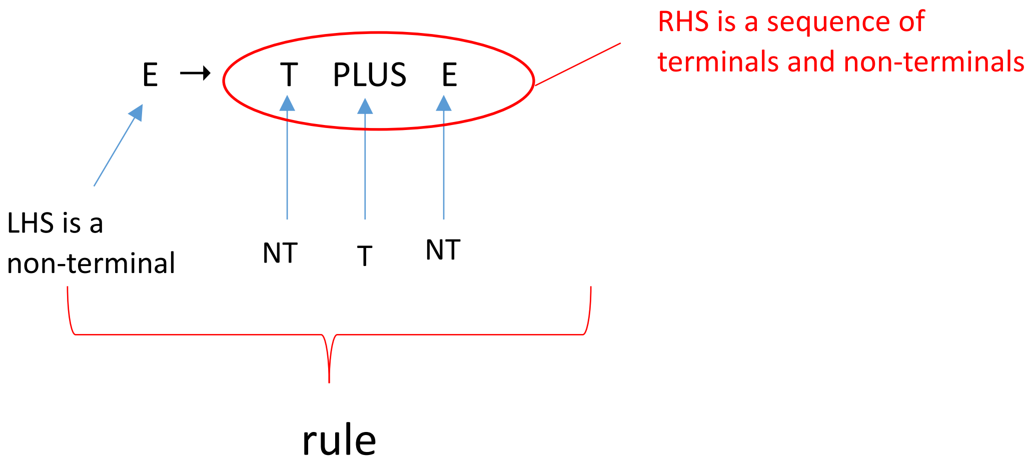
Example: Simple Expressions

```
expr      → term
expr      → term PLUS expr
term      → factor
term      → factor MULT term
factor    → ID
factor    → NUM
factor    → LPAREN expr RPAREN
```

OR

```
E  → T PLUS E                          // T + E
E  → T                                 // T
T  → F MULT T                          // F * T
T  → F                                 // F
F  → NUM | ID | LPAREN E RPAREN        // NUM | ID | (E)
```

A grammar consists of a set of rules. Each rule has a left-hand side (LHS) and a right-hand side (RHS). The left-hand sides of rules must be non-terminals (NT). Non-terminals are the symbols that appear on the LHS of rules The right-hand sides of rules are sequences of terminals and non-terminals. Terminals are simply tokens and they do not appear on the left-hand sides of rules. For example:



Parsing

- The goal of parsing is to determine if the input is syntactically correct according to the grammar

- In general, the parser (the program that does the parsing) can build a parse tree or some other representation of the program to be used by later processing stages (expression optimization for example)

- In this set of notes, I will start by showing how a simple parser can be written for a simple expression grammar.

- In another set of notes, we will introduce a general definition of grammars which are called context free grammars and show how parsing can be done efficiently for a significant subset of context free grammars

Parsing

Recursive Descent Parsing

- **One parsing function per non-terminal** We will have one parsing function for each "non-terminal". These are the symbols that appear on the left side of an arrow in a grammar rule.

- **Each parsing function consumes the part of the input that corresponds to its non-terminal**
  - Each patrse_X() is called for non-terminal X, it will either
    - succeed in parsing the non-terminal by consuming the part of the input corresponding to X (no more and no less) or
    - it throws a syntax_error() which stops the whole parsing process (in general one can attempt to recover from syntax errors but this is beyond the scope of our class)
  - A parse function does not consume any part of the input that is not part of the non-terminal it is parsing.

- **Recursive descent** A parsing function will proceed by calling other parsing functions (recursively) and the expect() and peek() functions (as we will see next)

  - To consume non-terminal X, we call parse_X()
  - To consume token ttype, we call expect(ttype)

Simple Expression Grammar

```
E → T PLUS E                                // T + E
E → T                                       // T
T → F MULT T                                // T * E
T → F                                       // F
F → NUM | ID | LPAREN E RPAREN              // NUM | ID | (E)

void parse_input()                          // top level input parsing
{
        parse_E();                          // consume E which is
                                            // the top level symbol
                                            // of the grammar

        t = lexer.getToken();
        if (t.token_type != EOF)            // input should have
                syntax_error();             // nothing after E

        return;                             // Input -> E
}
```

In the code, we call syntax_error() function whenever we encounter a syntax_error(). We assume that the syntax_error() function will simply prints a message and exits the whole program, so there is really no attempt at recovering from the error.

In the code, you will notice the following pattern which we will encounter often:

```
t = lexer.getToken();
if (t.token_type != ttype)
        syntax_error();
```

This pattern is used whenever we want to be sure that the next token is equal to a particular token type (ttype in the code). To simplify the code, we introduce the expect() function to capture this pattern.

```
expect(ttype)   ≣   t = lexer.getToken();
                    if (t.token_type != ttype)
                            syntax_error();
                    return t;
```

The expect function returns the token t if there is no syntax_error(). So, if you call expect(ID) and the next token is an ID, the returned value is the token structure for the ID.

Simple Expression Grammar

```
E → T PLUS E                        // T + E
E → T                               // T
T → F MULT T                        // T * E
T → F                               // F
F → NUM | ID | LPAREN E RPAREN      // NUM | ID | (E)
```

```
void parse_E()                      // consumes E
{
        parse_T();                  // consumes T
        t = lexer.peek(1);
        if (t.token_type == PLUS)
        {
                expect(PLUS);       // consumes +
                parse_E();          // consumes E
        }
        else if ( (t.token_type == EOF) |
                  (t.token_type == RPAREN) )
        {
                return;
        }
        else
                syntax_error();
}
```

Here you notice that I used expect(PLUS) instead of using getToken(). We could have used getToken(), but expect(PLUS) makes it clear when reading the code which token is being consumed.

Also, notice how we are using peek() when there are multiple valid possibilities for the next token. If the next token is PLUS, then we continue parsing by consuming the PLUS and calling parse_E(). If the token is EOF or RPARAN (the tokens that can follow an E), then we return, otherwise, we detect syntax error.

Parsing by Example

```
E → T PLUS E                                  // T + E
E → T                                         // T
T → F MULT T                                  // T * E
T → F                                         // F
F → NUM | ID | LPAREN E RPAREN                // NUM | ID | (E)

void parse_T()                                // consumes T
{
        parse_F();                            // consumes F
        t = lexer.peek(1);
        if (t.token_type == MULT)
        {
                expect(MULT);                 // consumes *
                parse_T();                    // consumes T
        }
        else if ( (t.token_type == EOF) |
                  (t.token_type == PLUS) |
                 (t.token_type == RPAREN))
        {
                return;
        }
        else
                syntax_error();
}

void parse_F()                                // consumes F
{
        t = lexer.peek(1);
        if (t.token_type == ID)
                expect(ID);                   // F -> ID
        else if (t.token_type == NUM) )
                expect(NUM) ;                 // F -> NUM
        else if (t.token_type == LPAREN)
        {
                expect(LPAREN);               // LPAREN
                parse_E();                    // E
                expect(RPAREN);               // RPAREN
        } else
                syntax_error();
}
```

Step by Step exection

In the following I show a step by step execution of the code we wrote in two different format

1. Illustration of how a “parse tree” is built

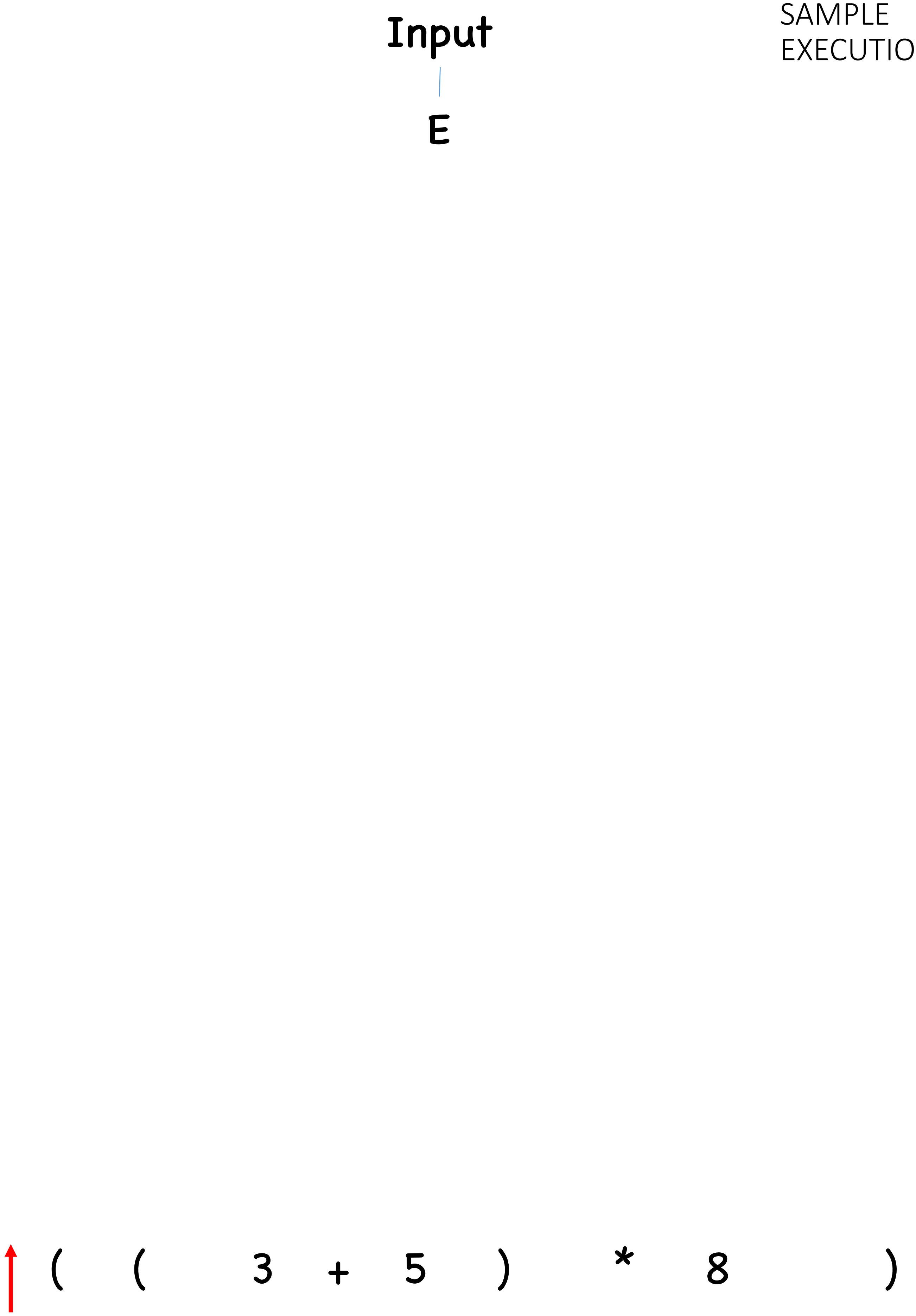
2. An equivalent illustration using call sequence

SAMPLE
EXECUTION

Input

((3 + 5) * 8)

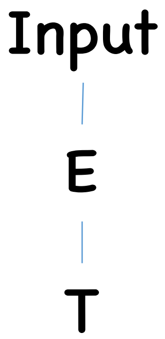
SAMPLE
EXECUTION


Input

E

((3 + 5) * 8)

SAMPLE
EXECUTION



Input

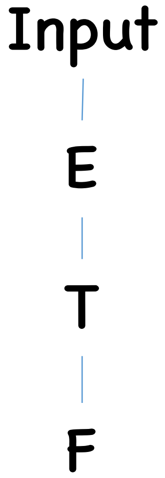
E

T

call parse_T()

↑ ((3 + 5) * 8)

SAMPLE
EXECUTION


Input

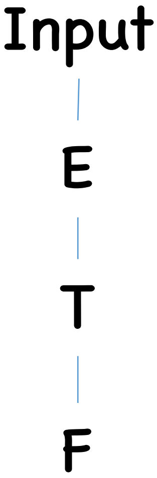
E

T

F

↑ ((3 + 5) * 8)

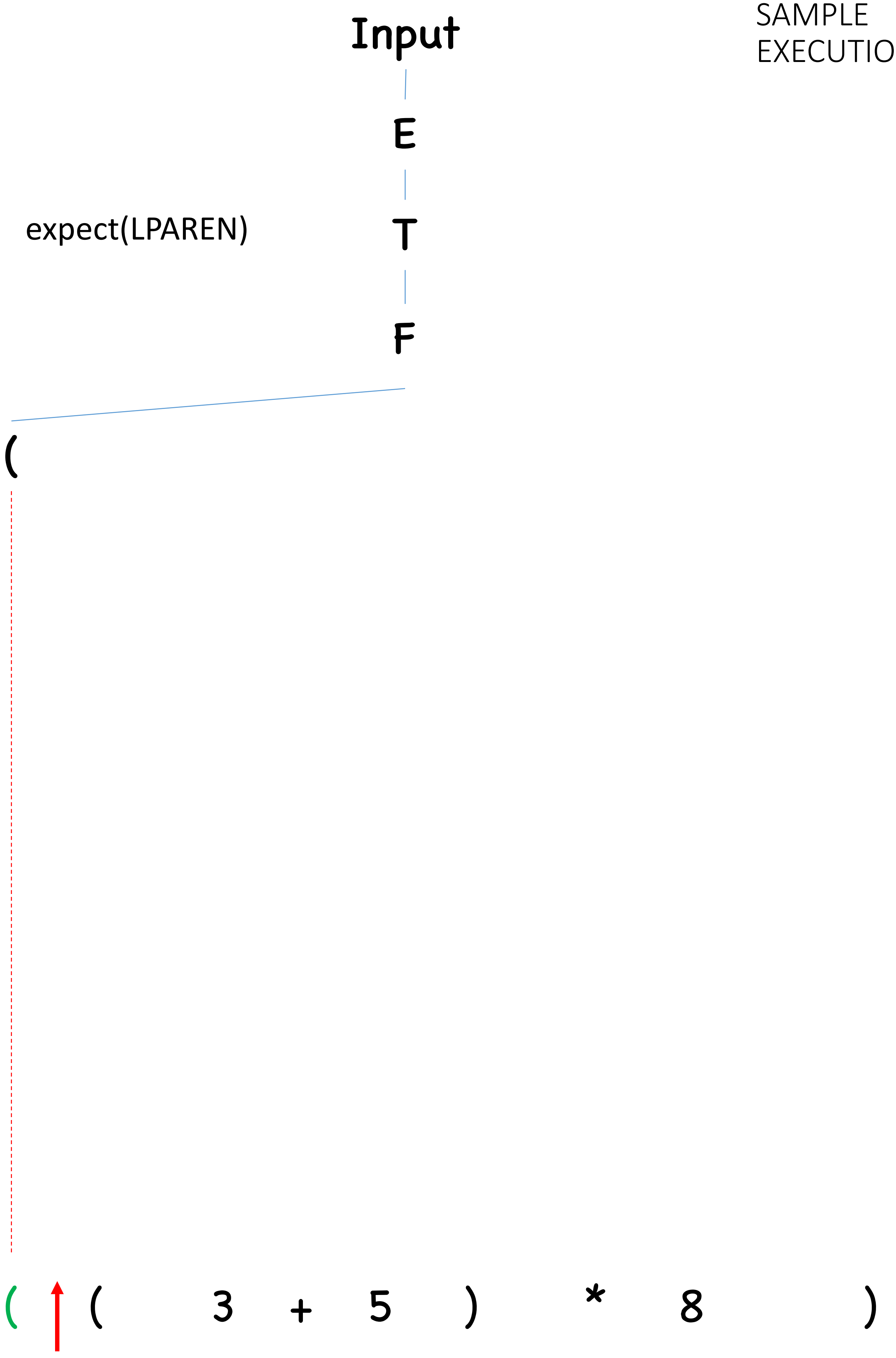
SAMPLE
EXECUTION



peek(1) returns LPAREN

((3 + 5) * 8)

SAMPLE EXECUTION



Input

E

T

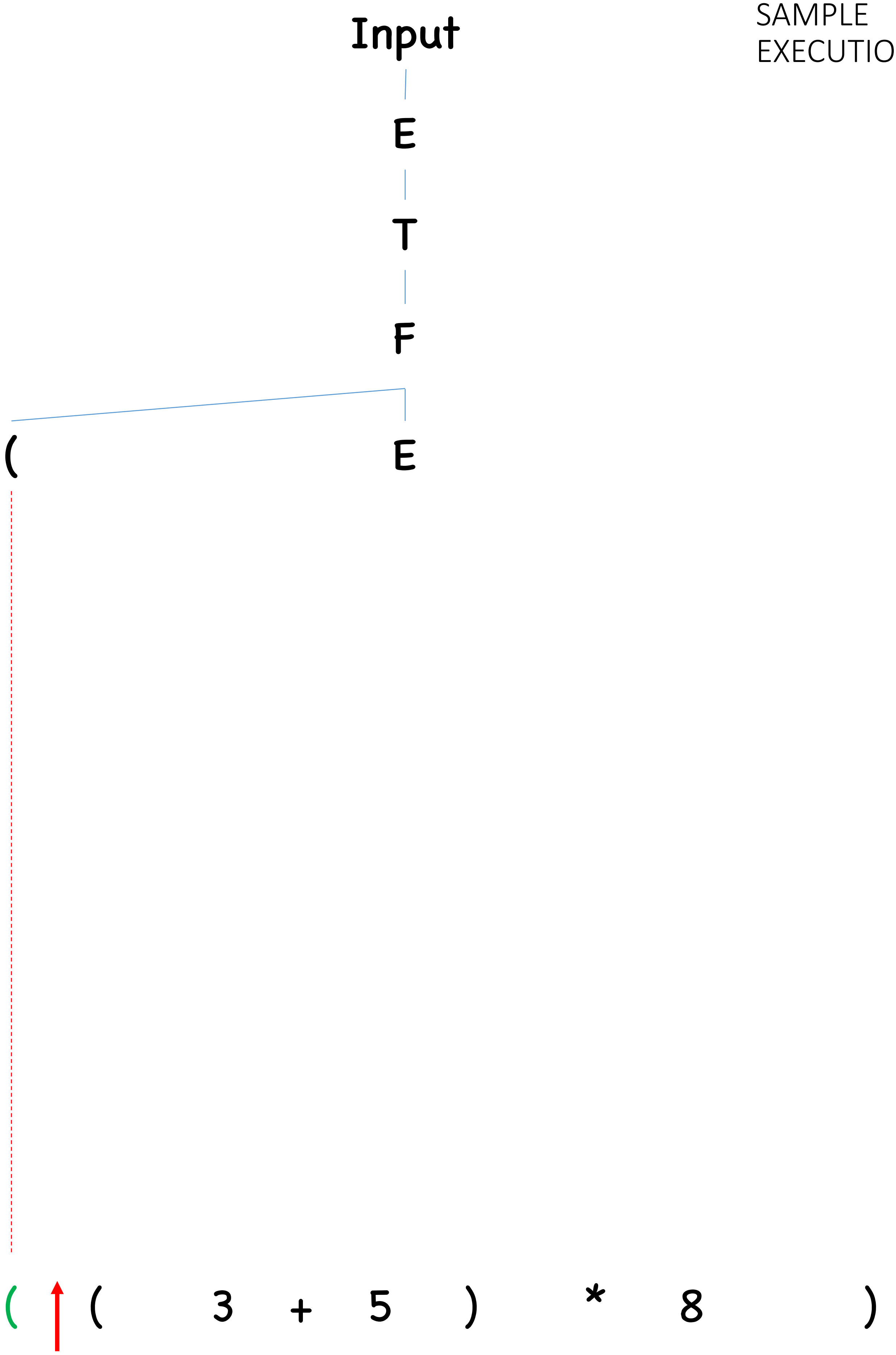
F

expect(LPAREN)

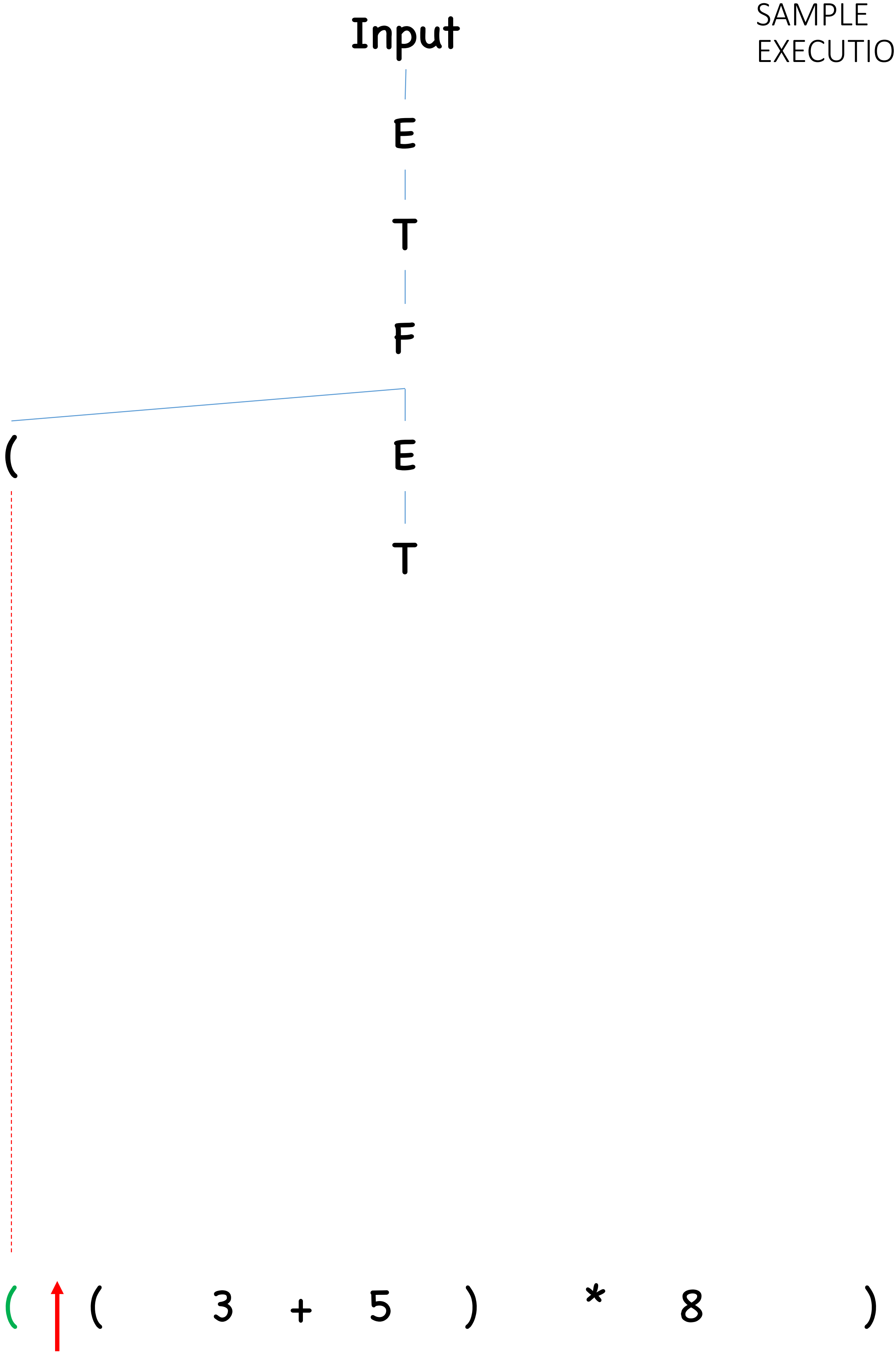
(

((3 + 5) * 8)

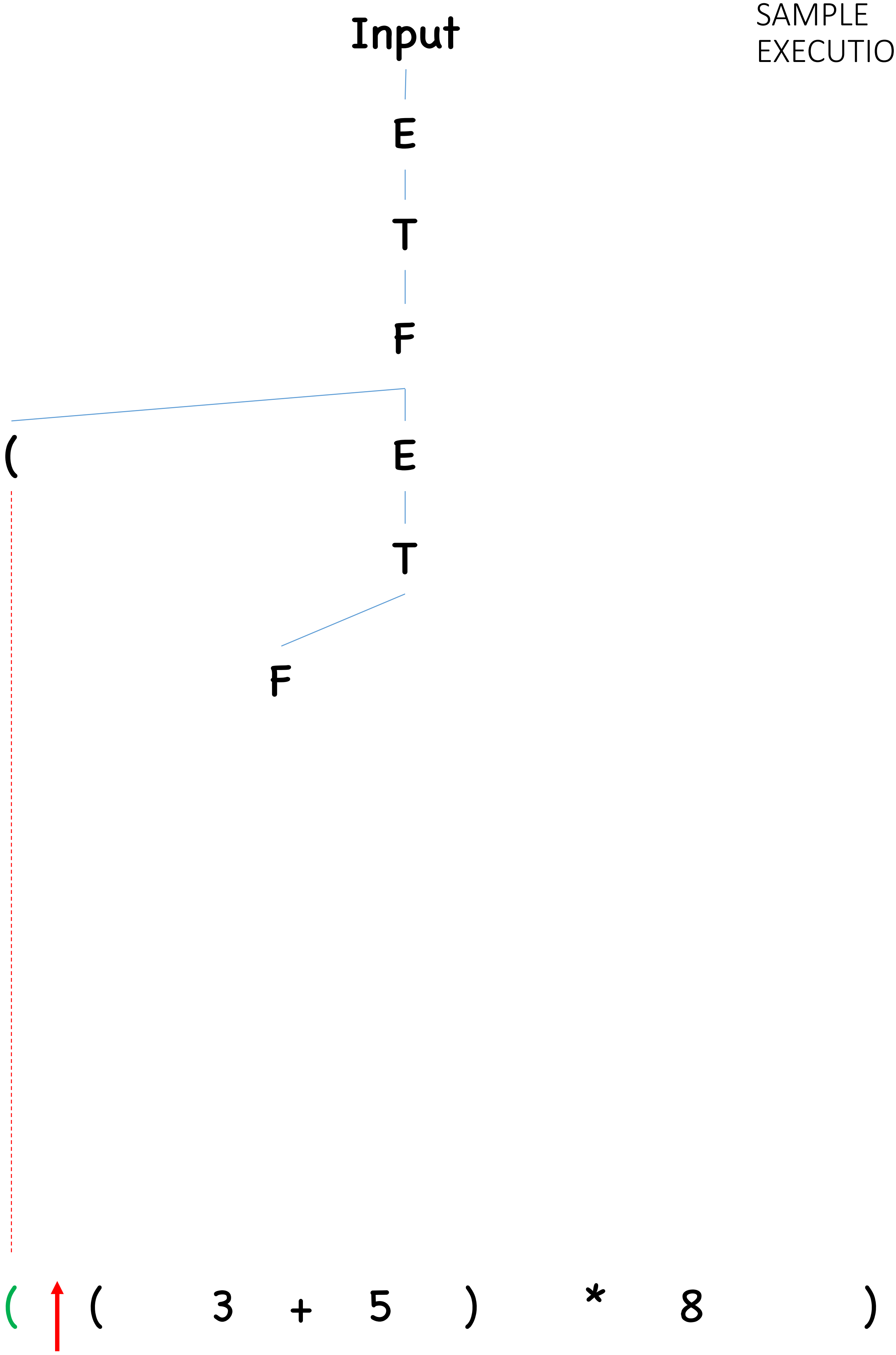
SAMPLE
EXECUTION



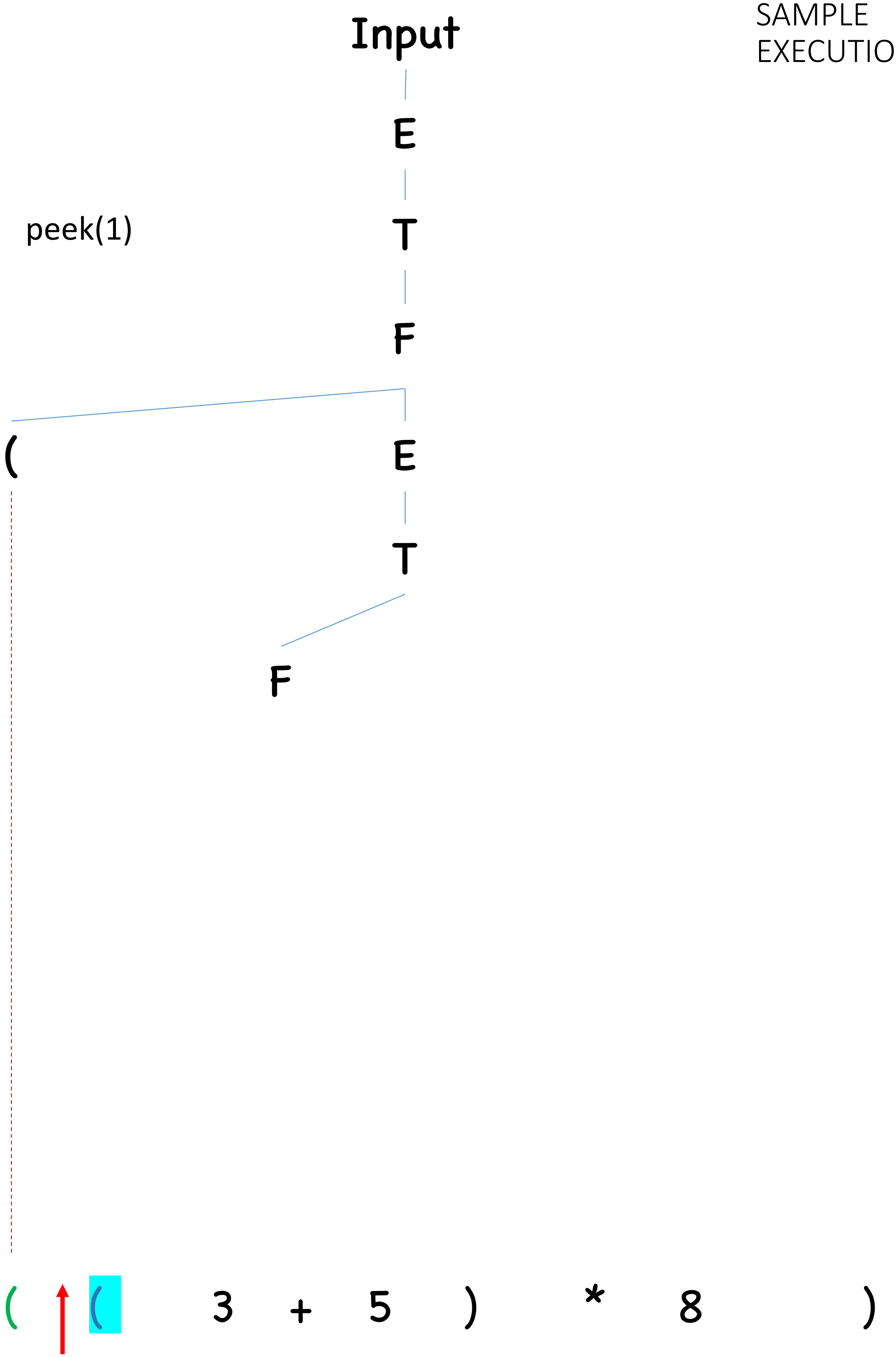
SAMPLE
EXECUTION



SAMPLE
EXECUTION



SAMPLE EXECUTION



Input

E

T

F

(E

T

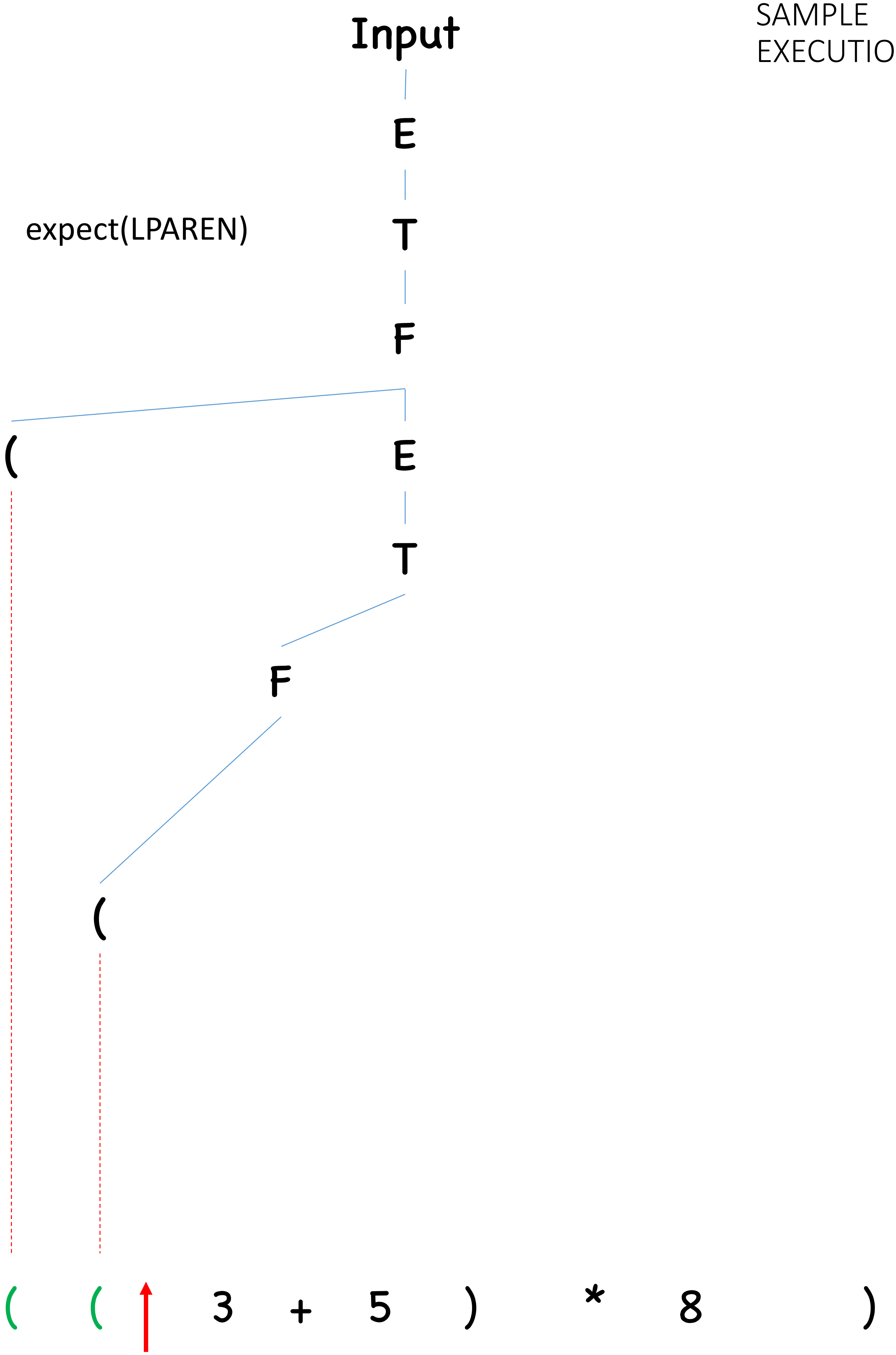
F

peek(1)

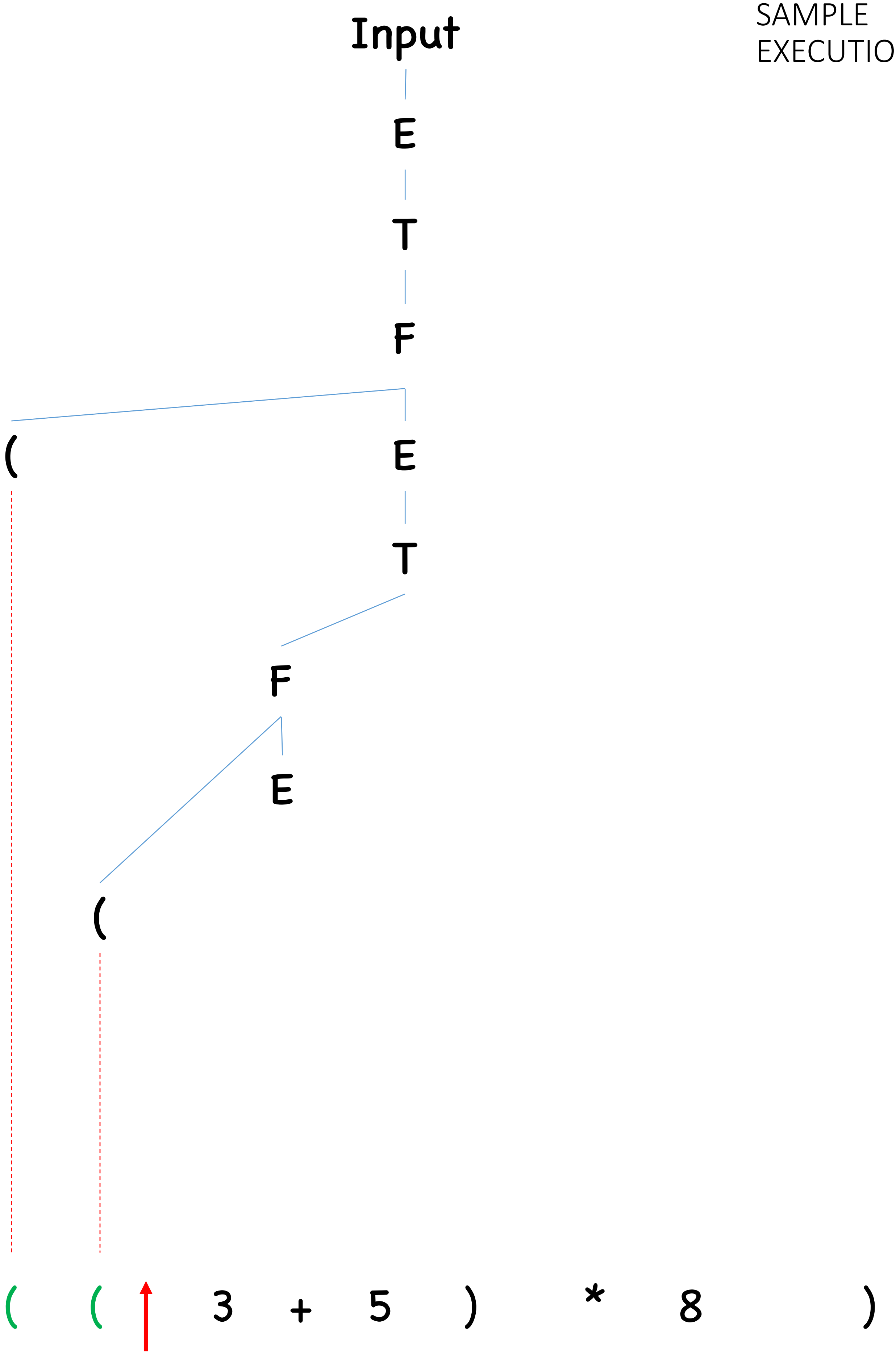
((3 + 5) * 8)

SAMPLE
EXECUTION

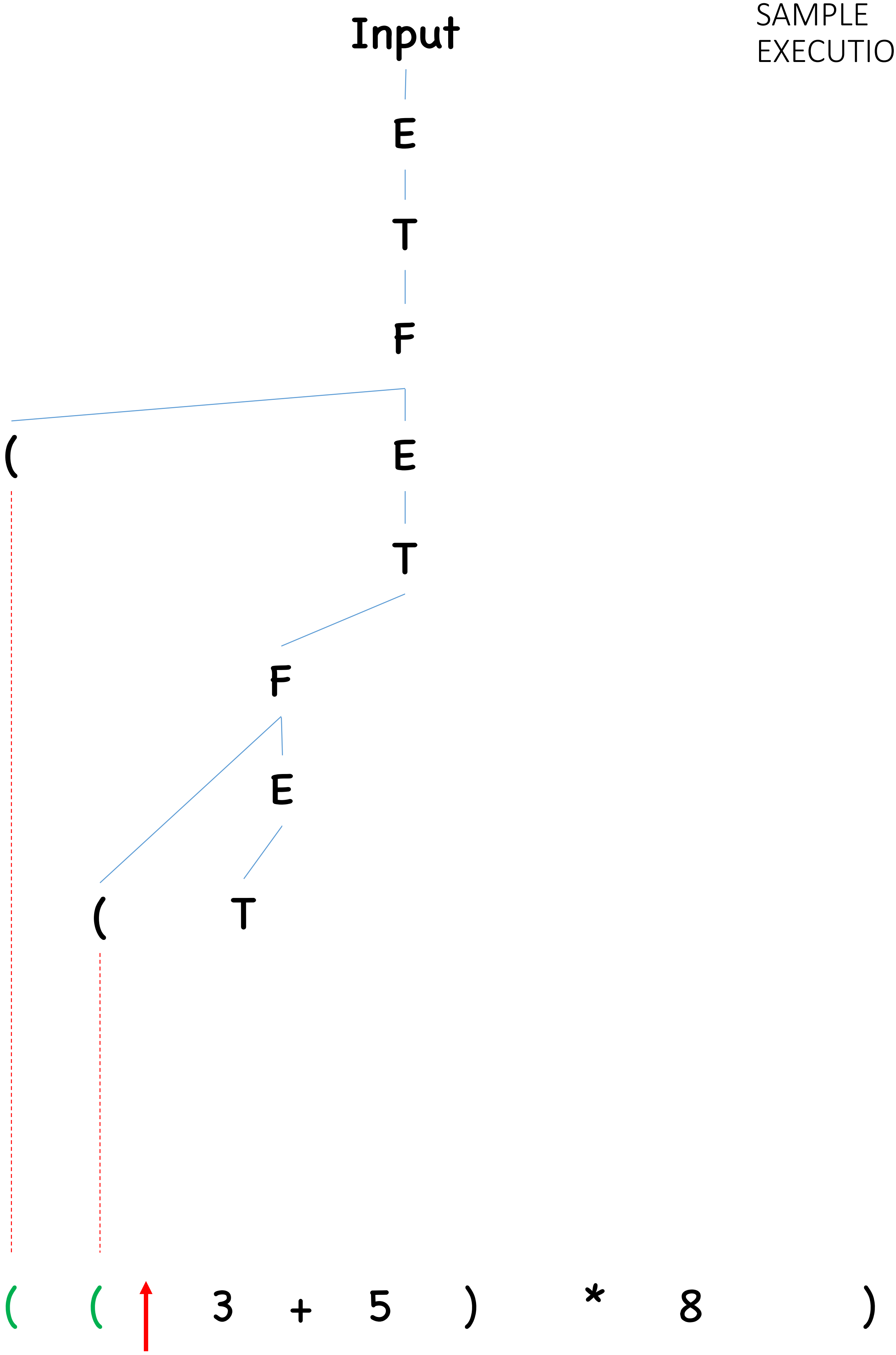
expect(LPAREN)



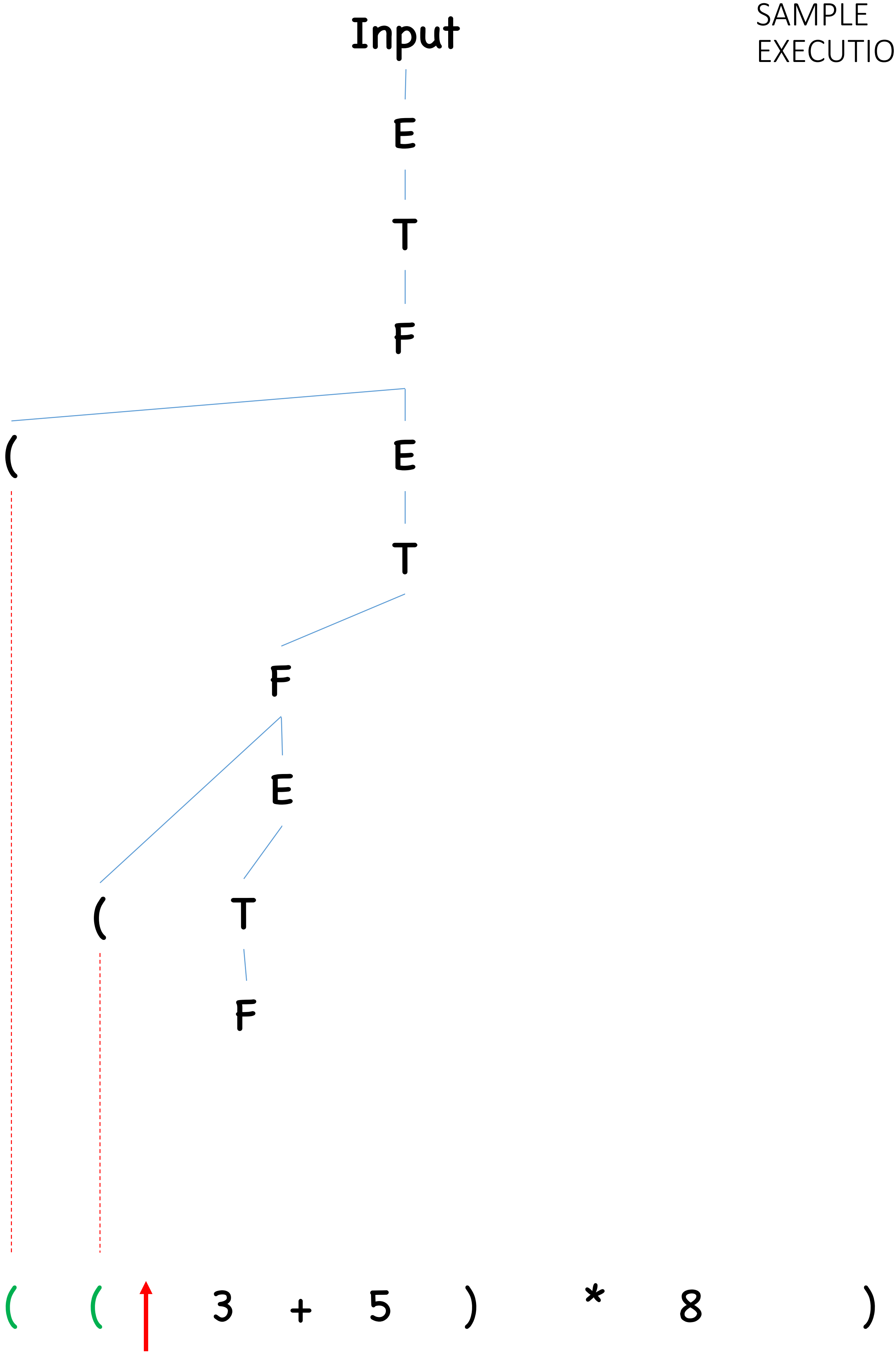
SAMPLE EXECUTION



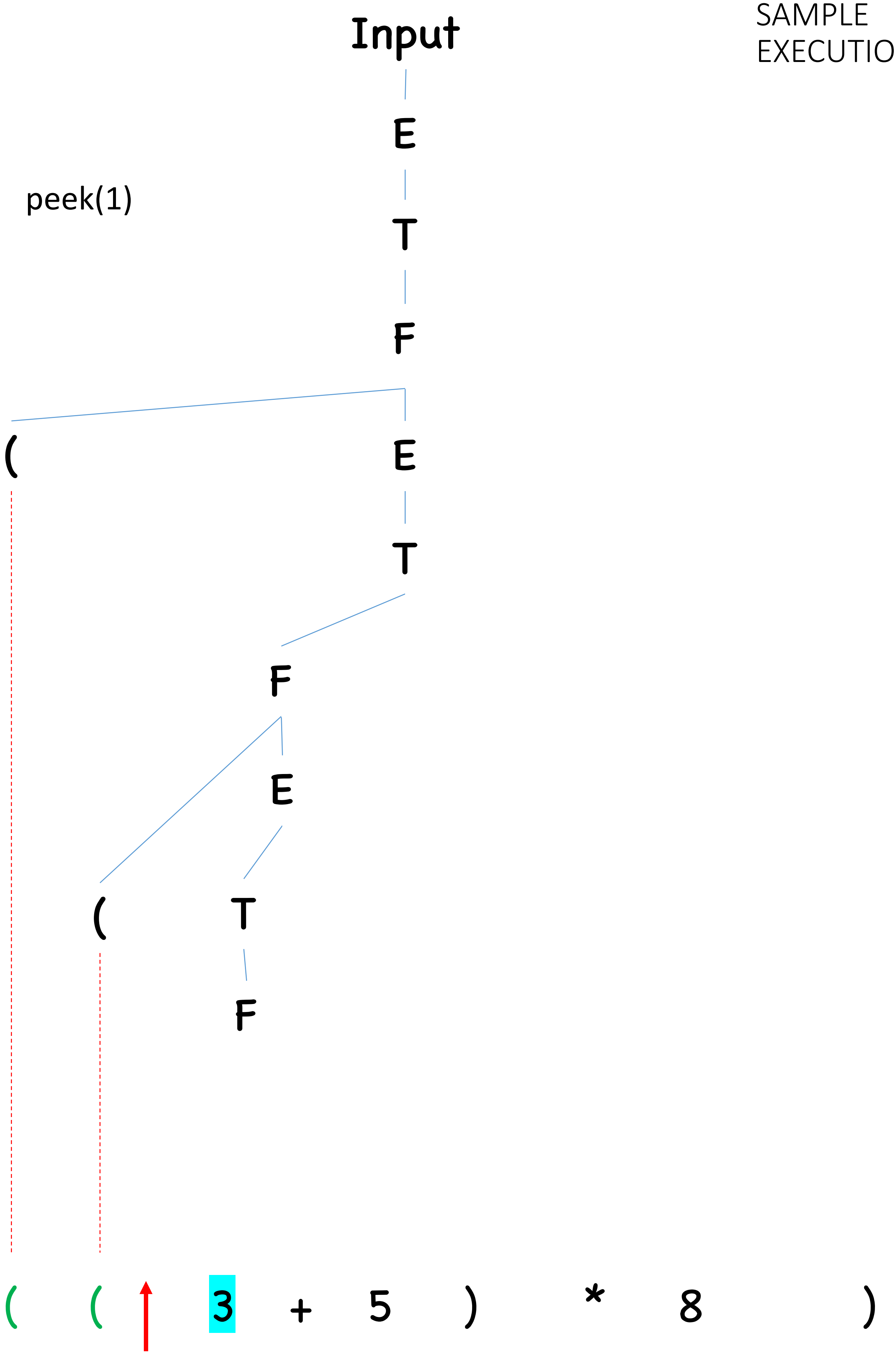
SAMPLE EXECUTION



SAMPLE EXECUTION

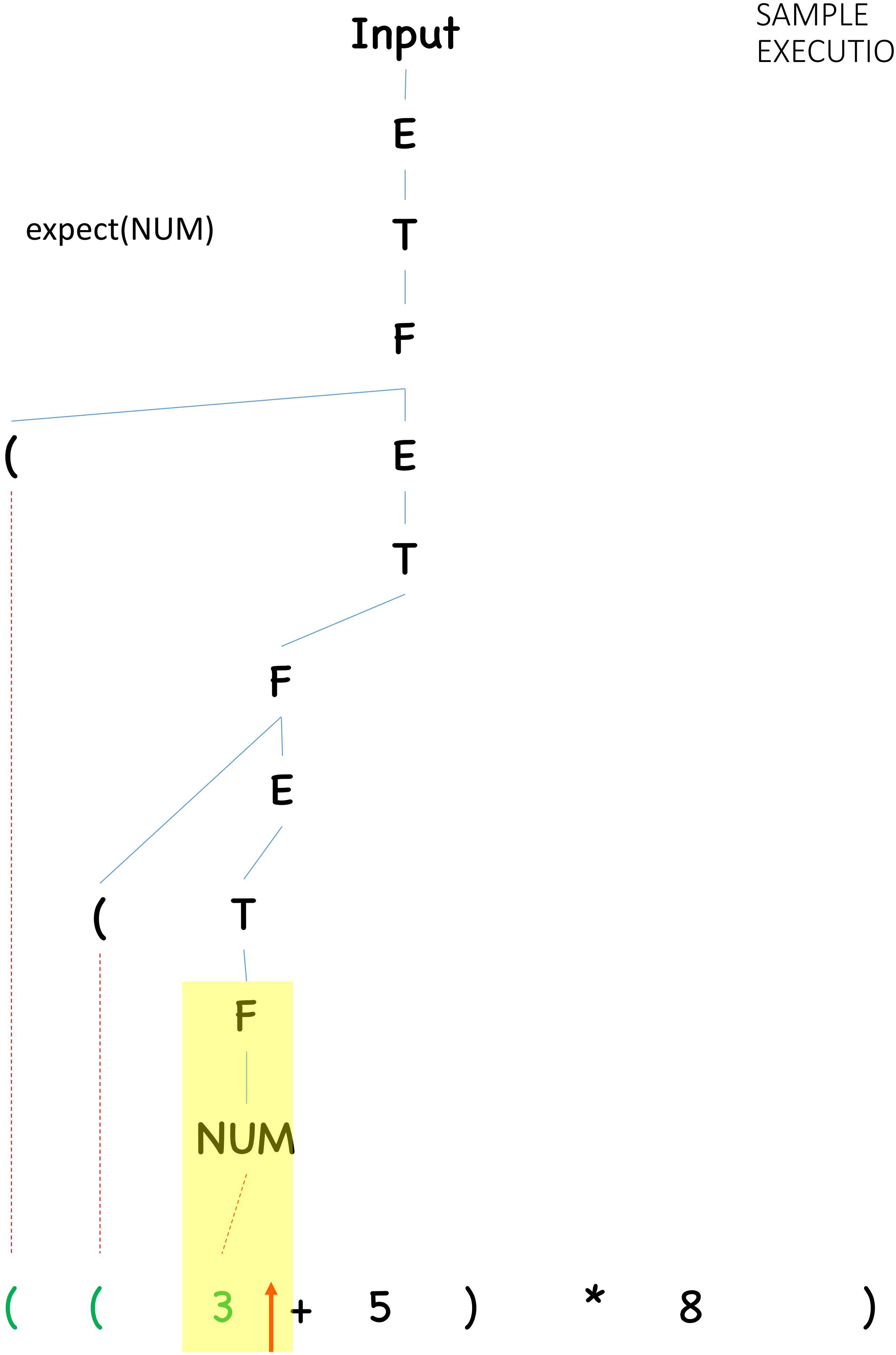


SAMPLE
EXECUTION



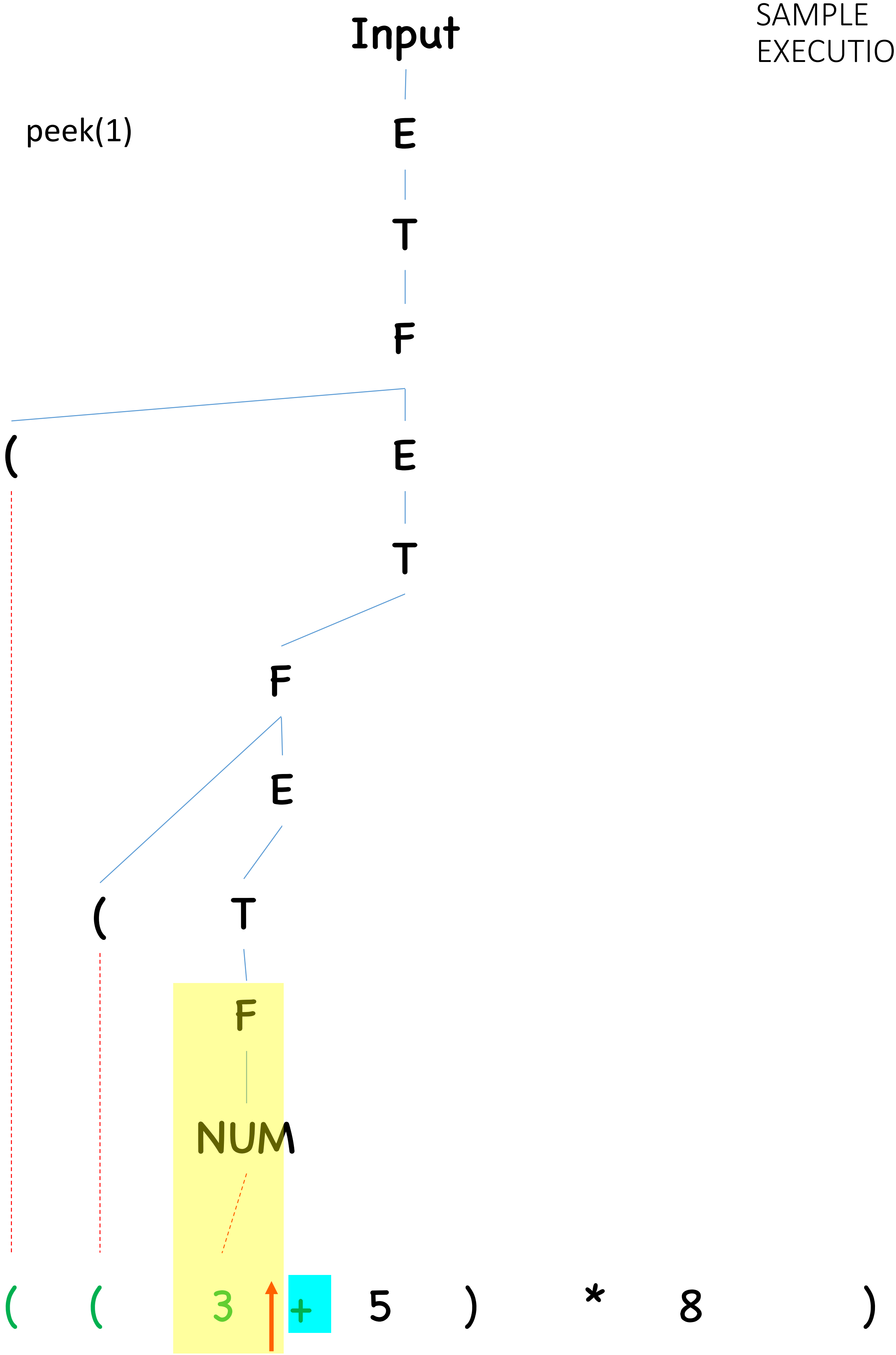
SAMPLE EXECUTION

expect(NUM)



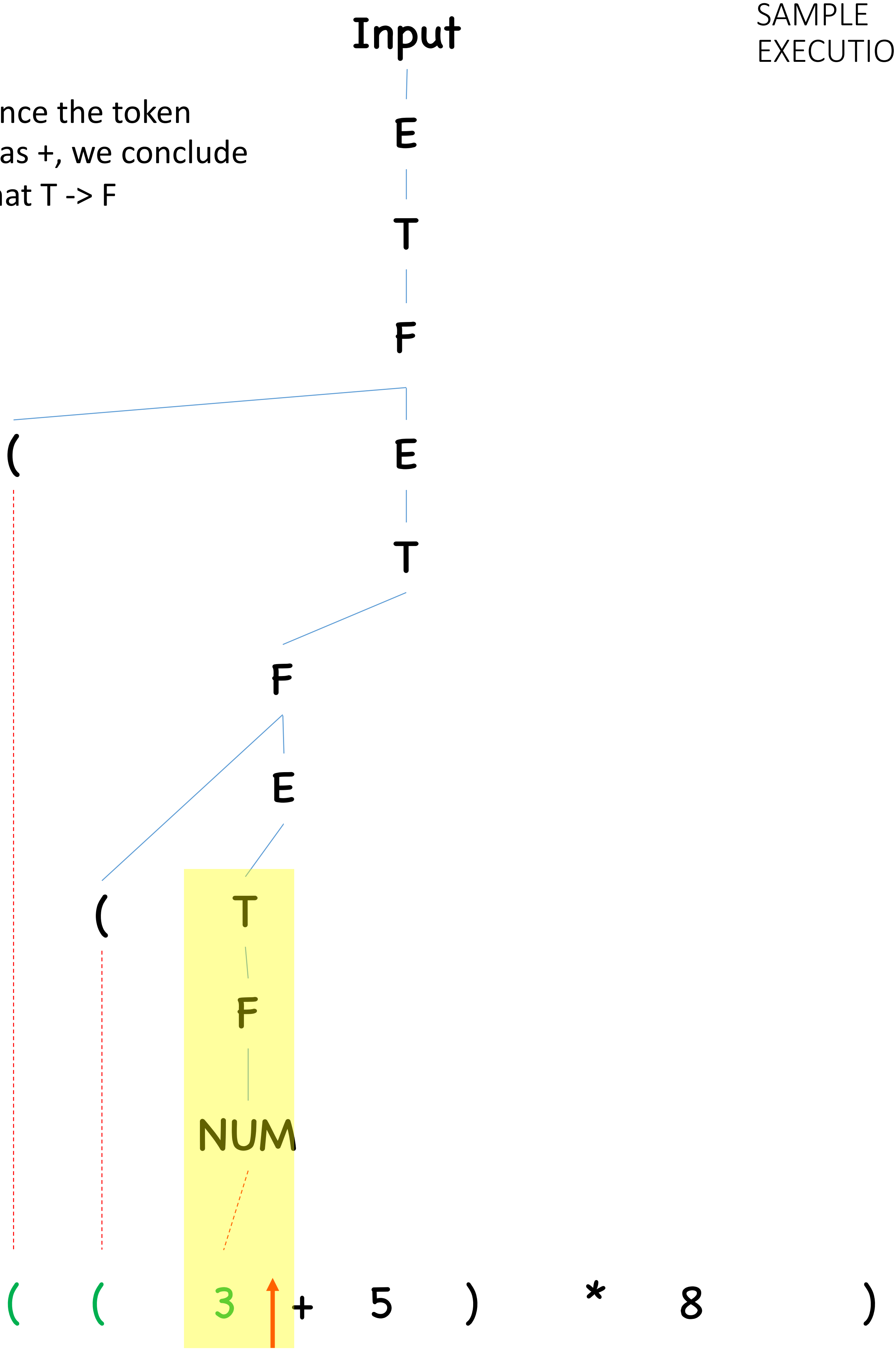
SAMPLE EXECUTION

peek(1)

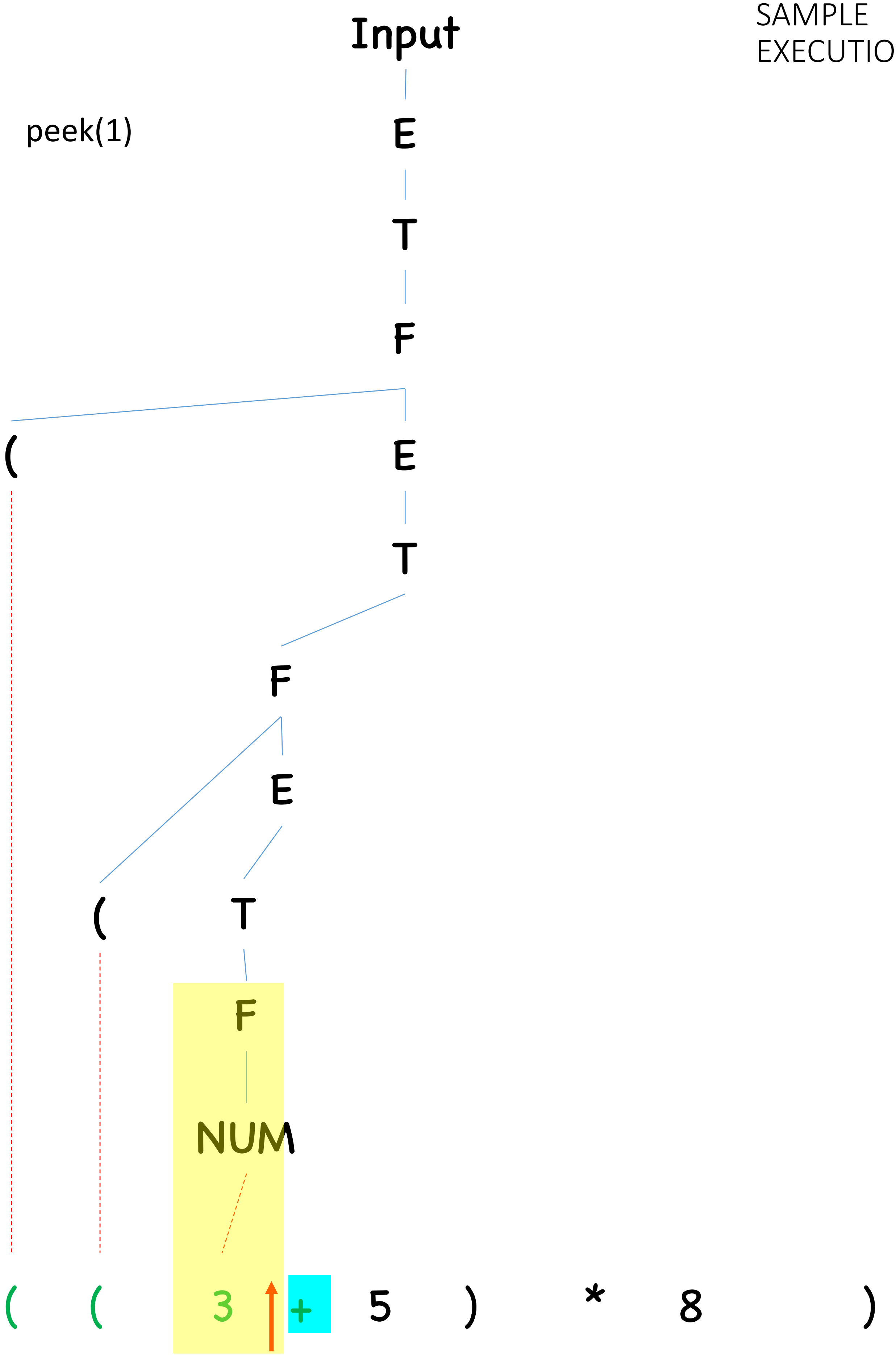


SAMPLE
EXECUTION

Since the token was +, we conclude that T -> F

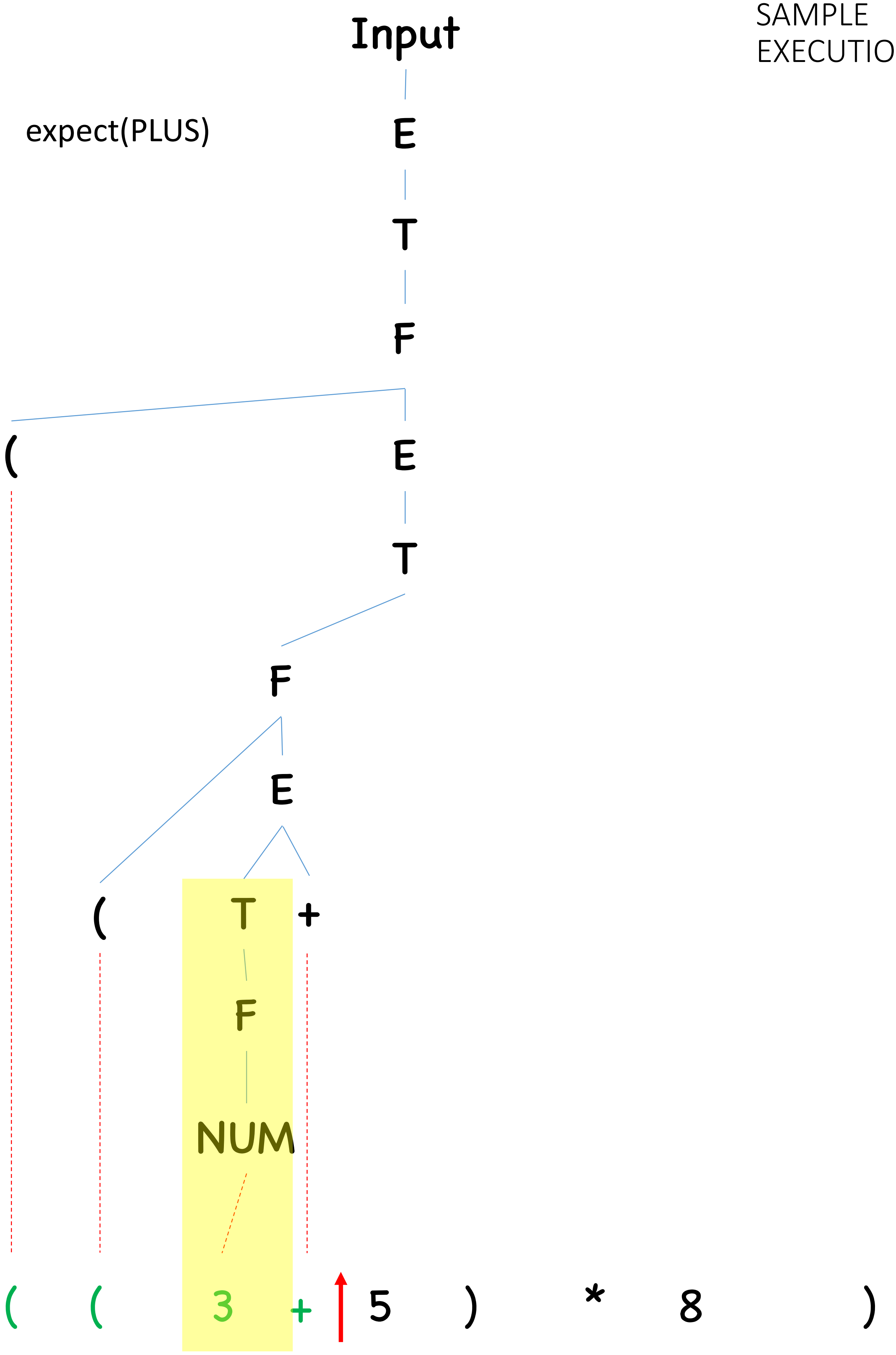


SAMPLE
EXECUTION

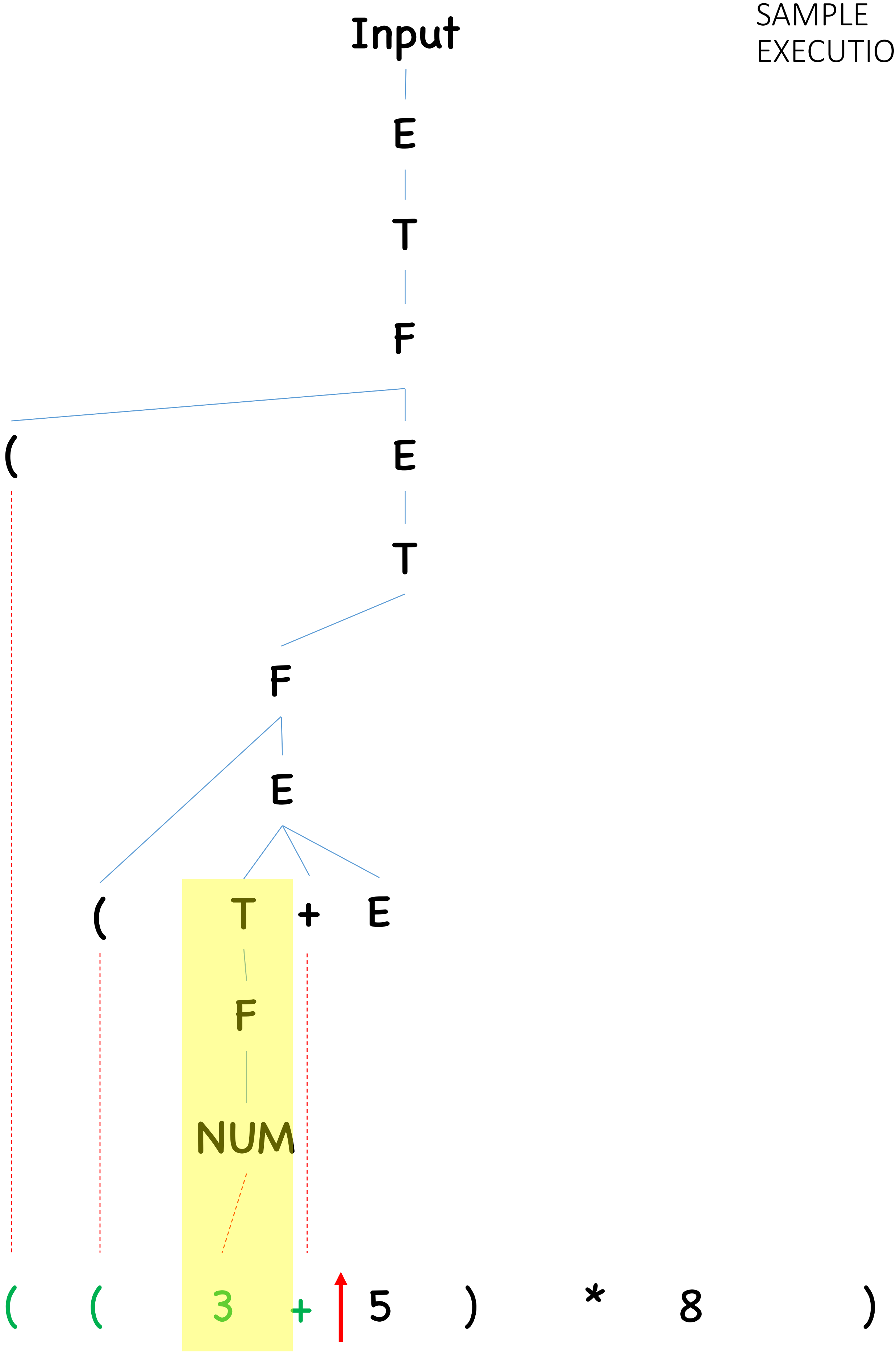


SAMPLE
EXECUTION

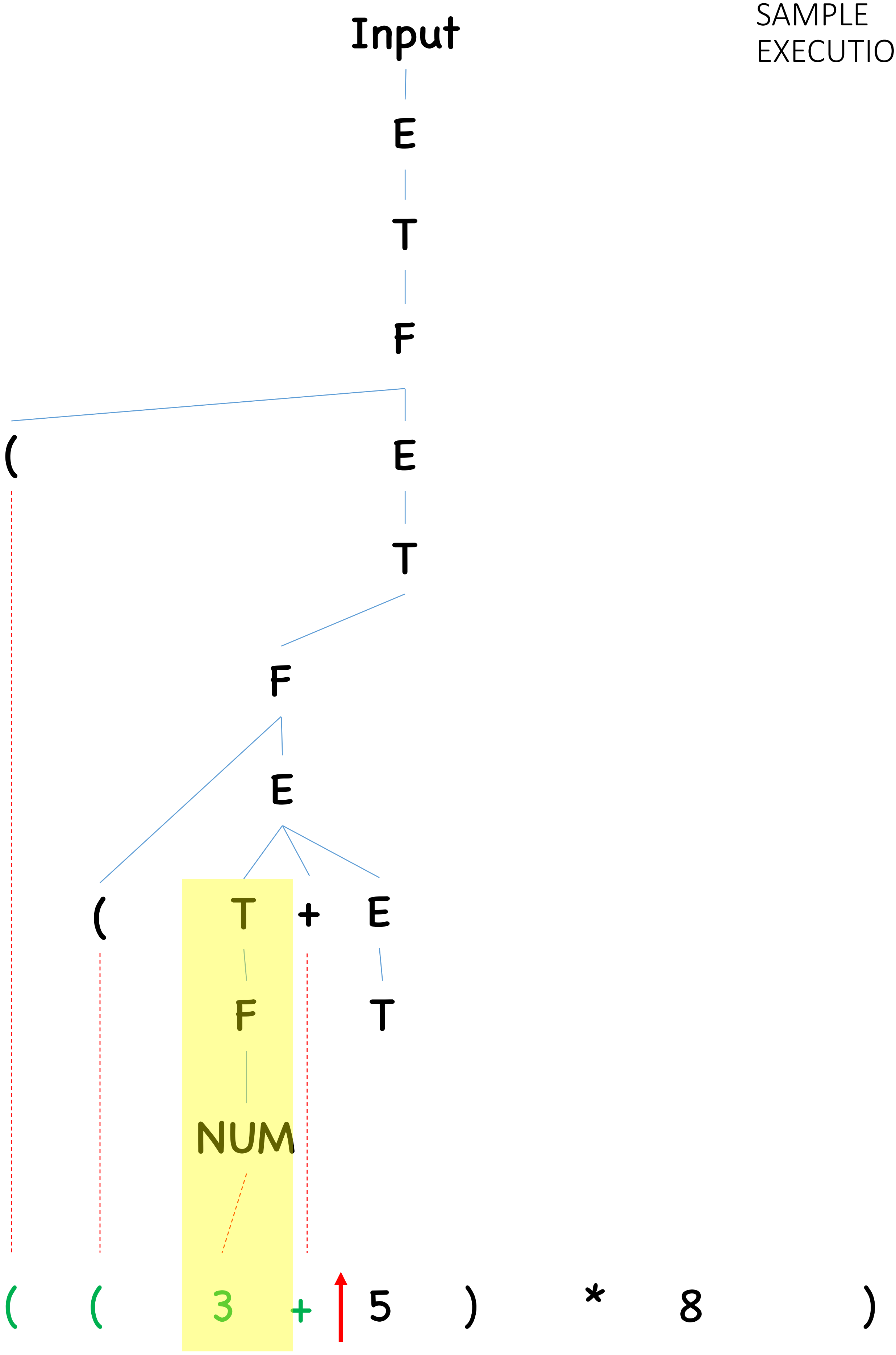
expect(PLUS)



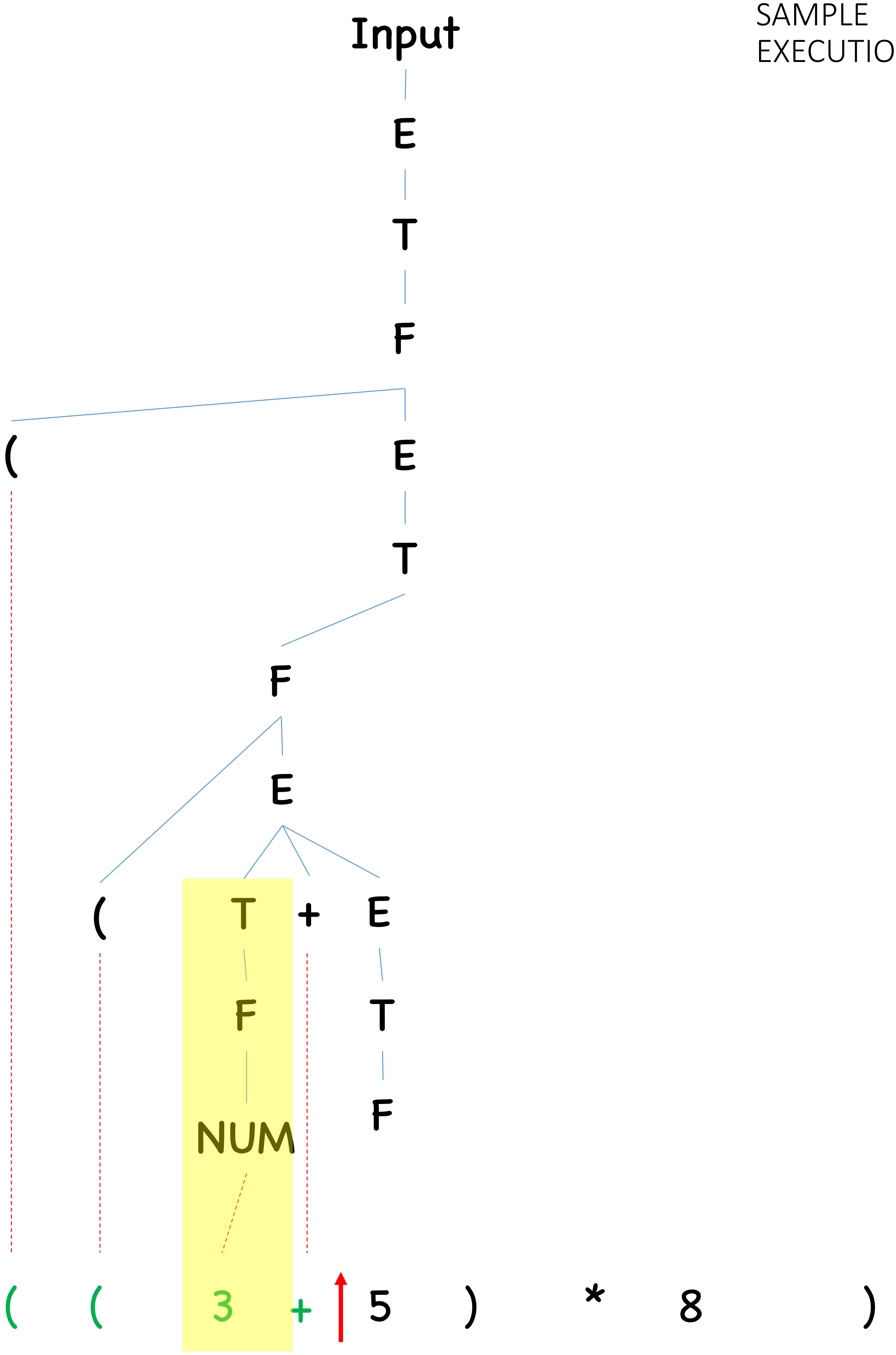
SAMPLE
EXECUTION



SAMPLE
EXECUTION

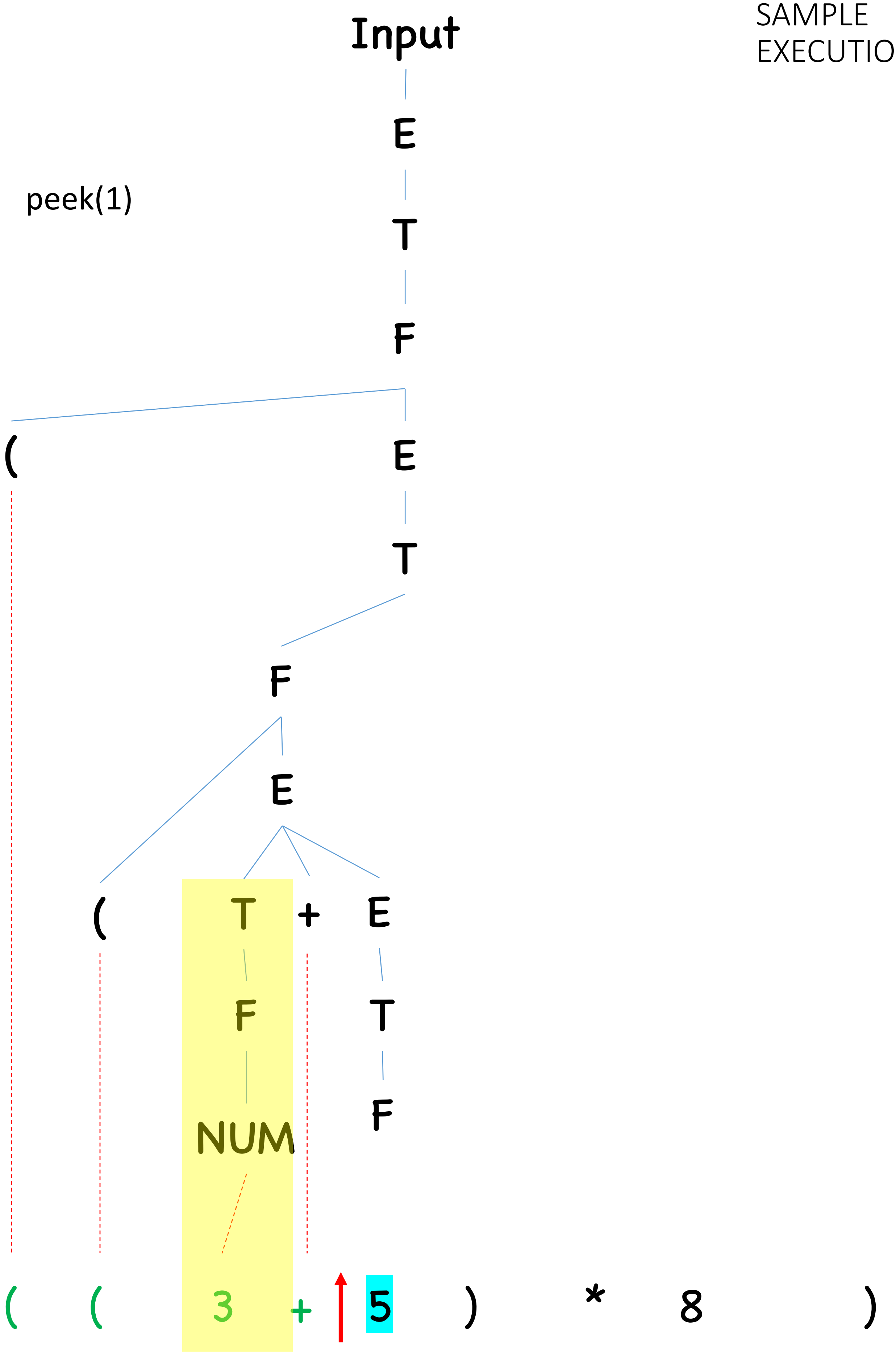


SAMPLE EXECUTION

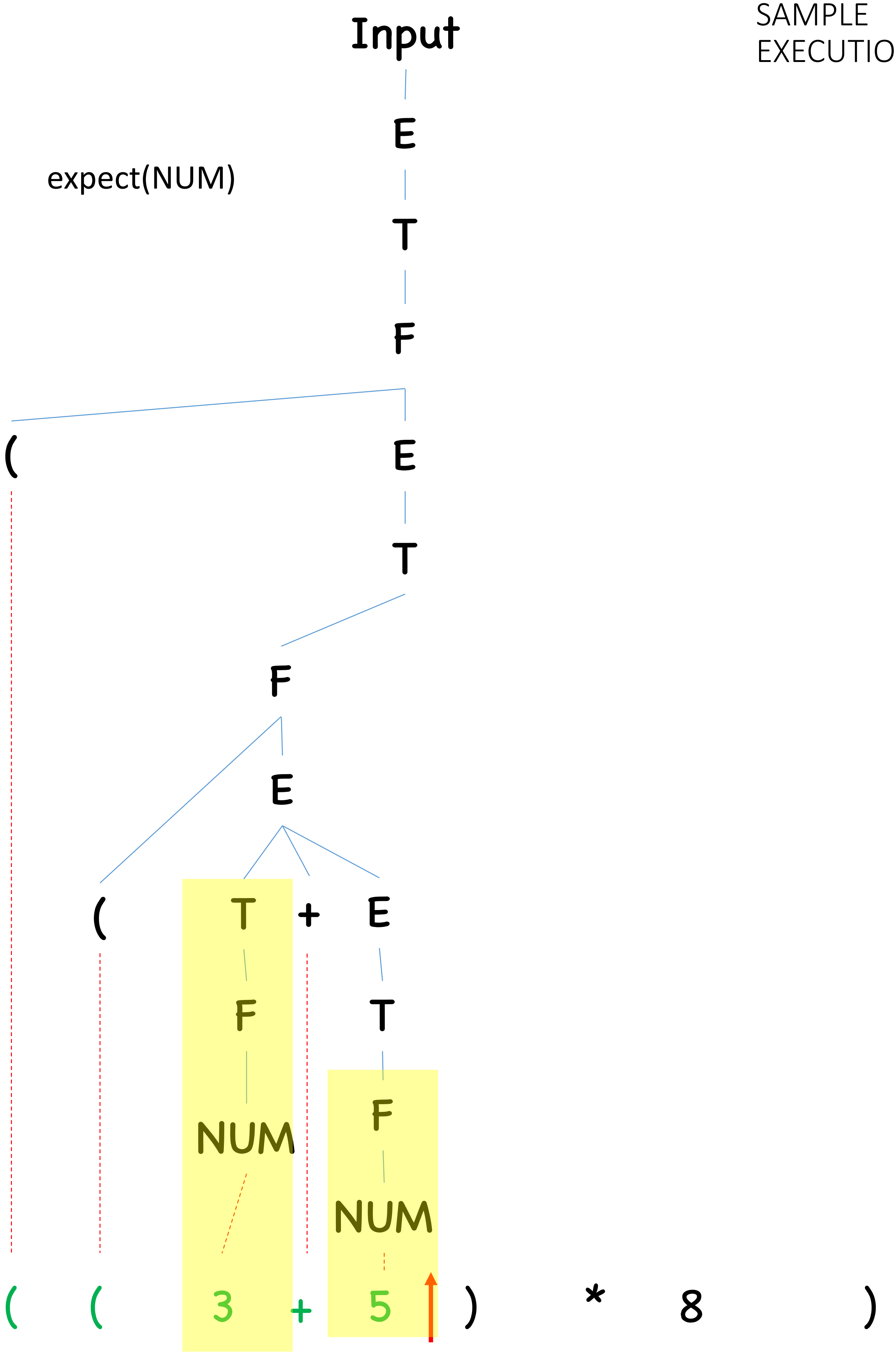


SAMPLE
EXECUTION

peek(1)

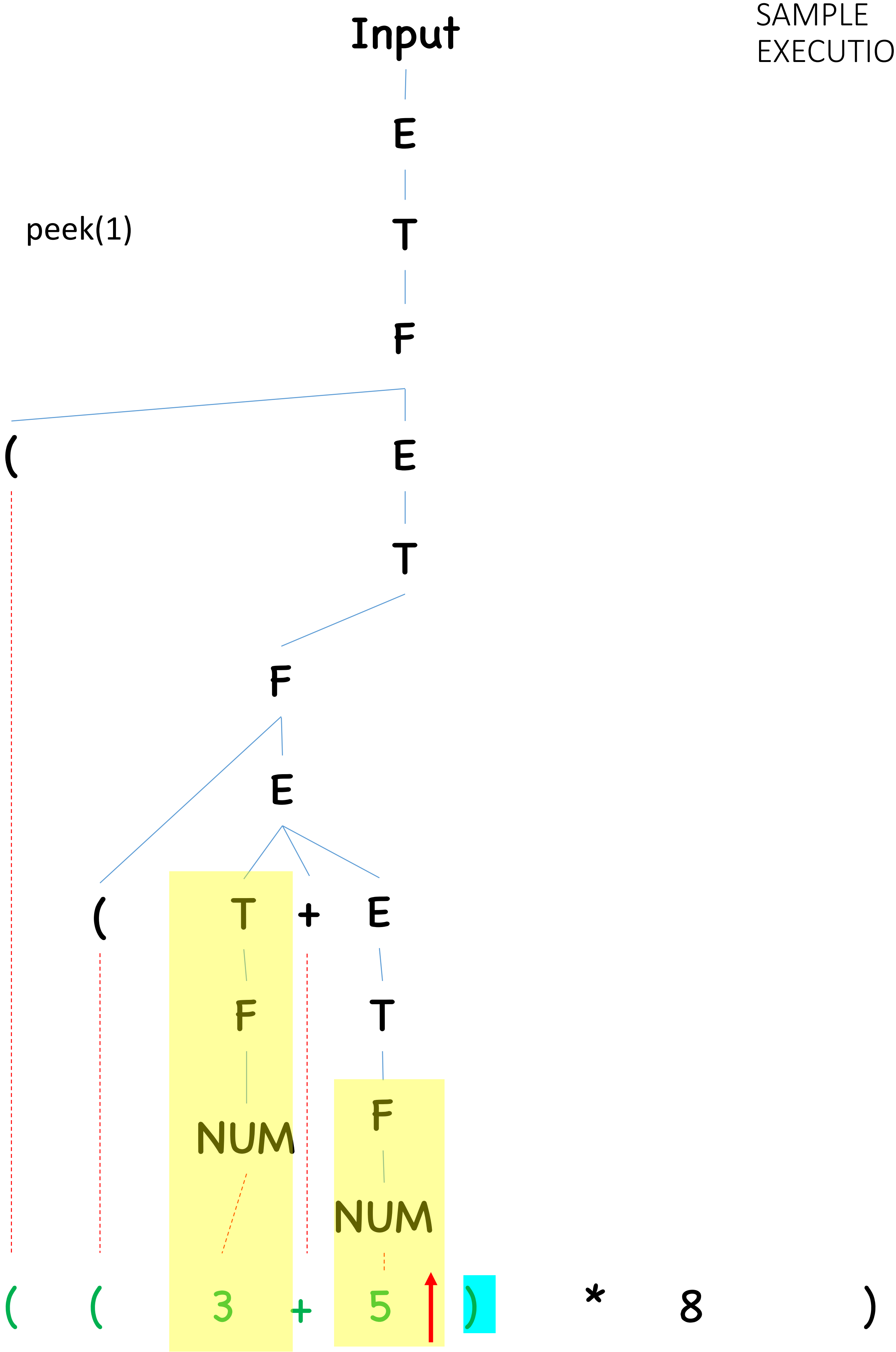


SAMPLE
EXECUTION

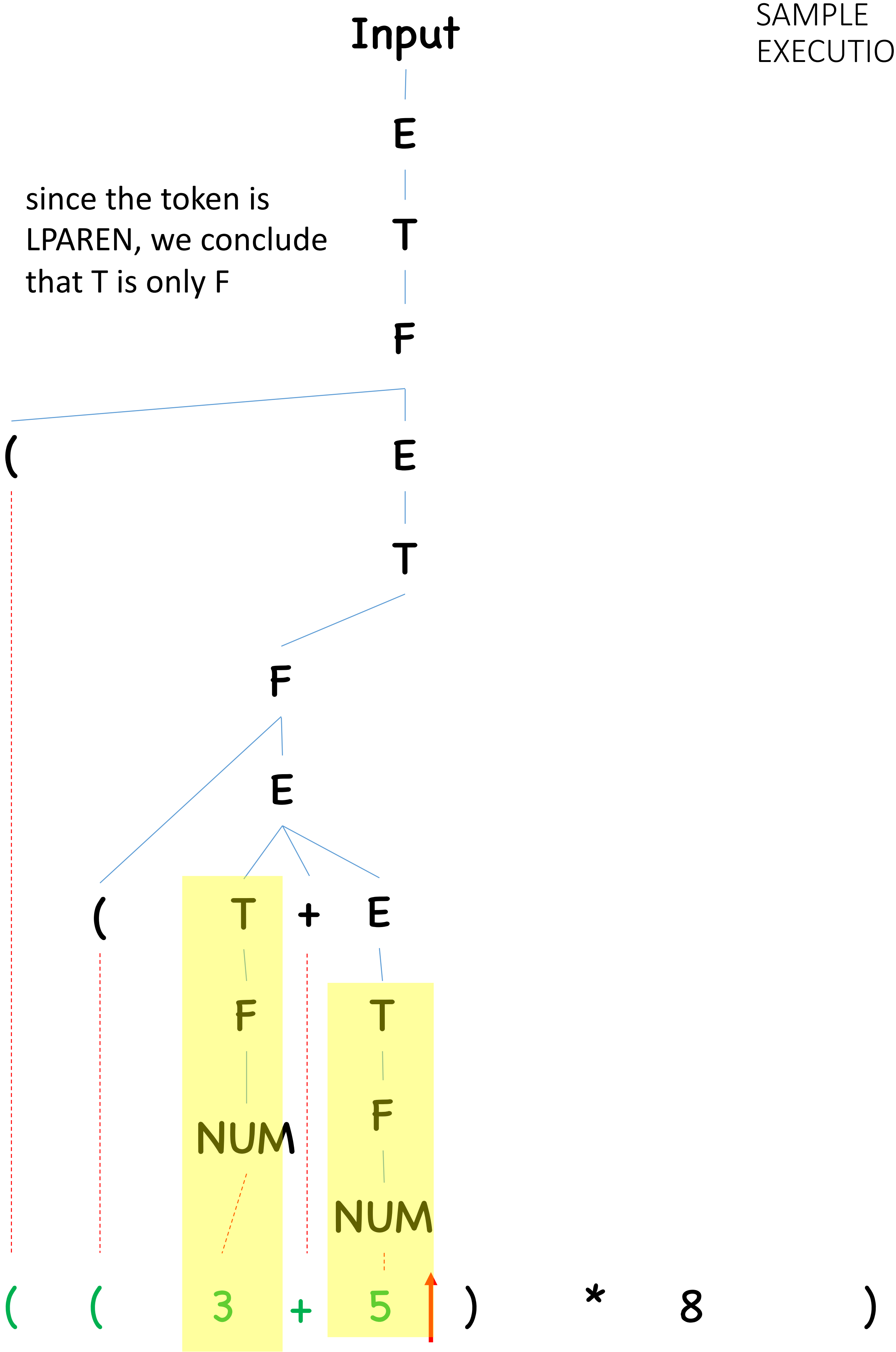


expect(NUM)

SAMPLE EXECUTION

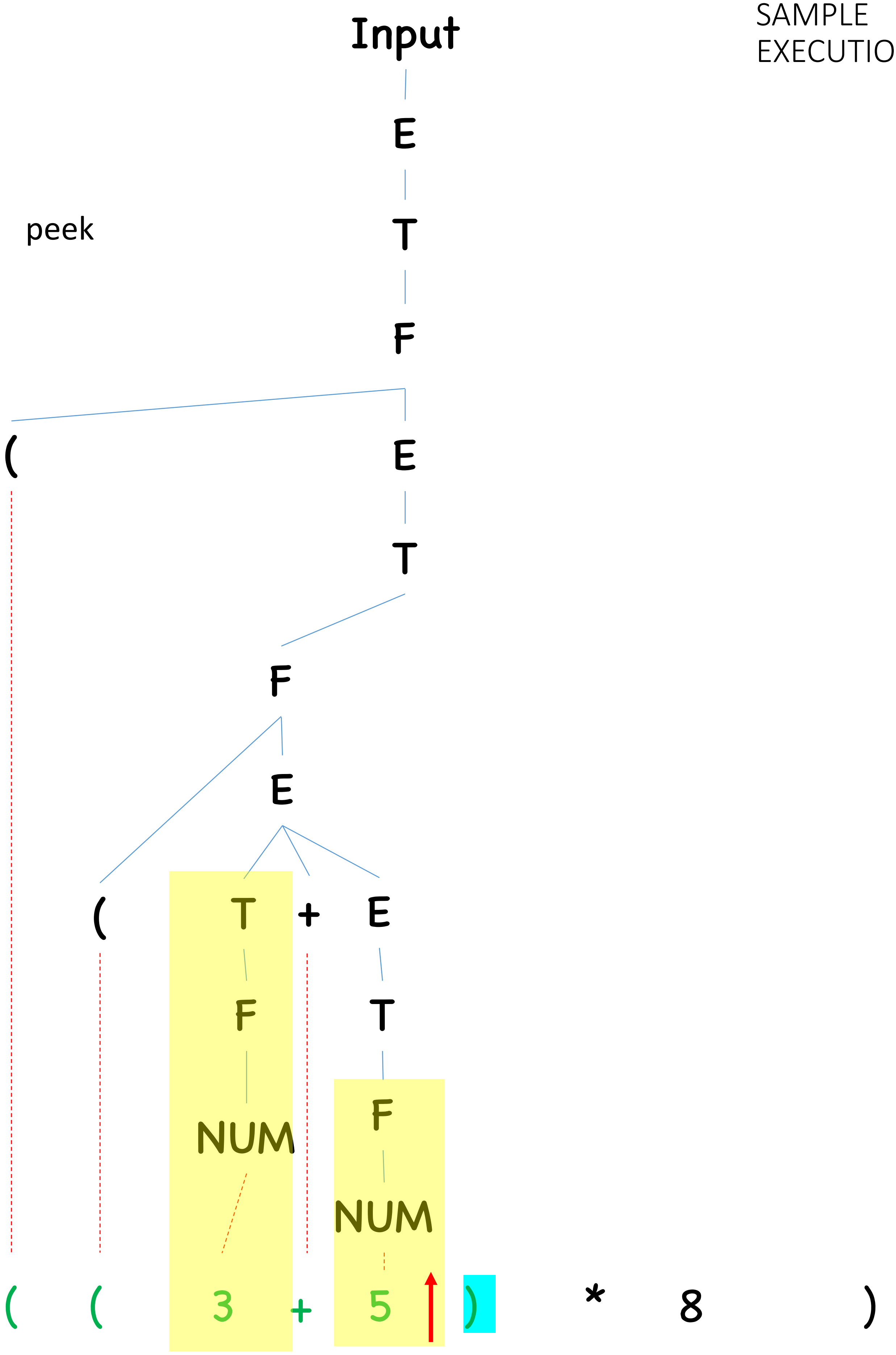


SAMPLE
EXECUTION



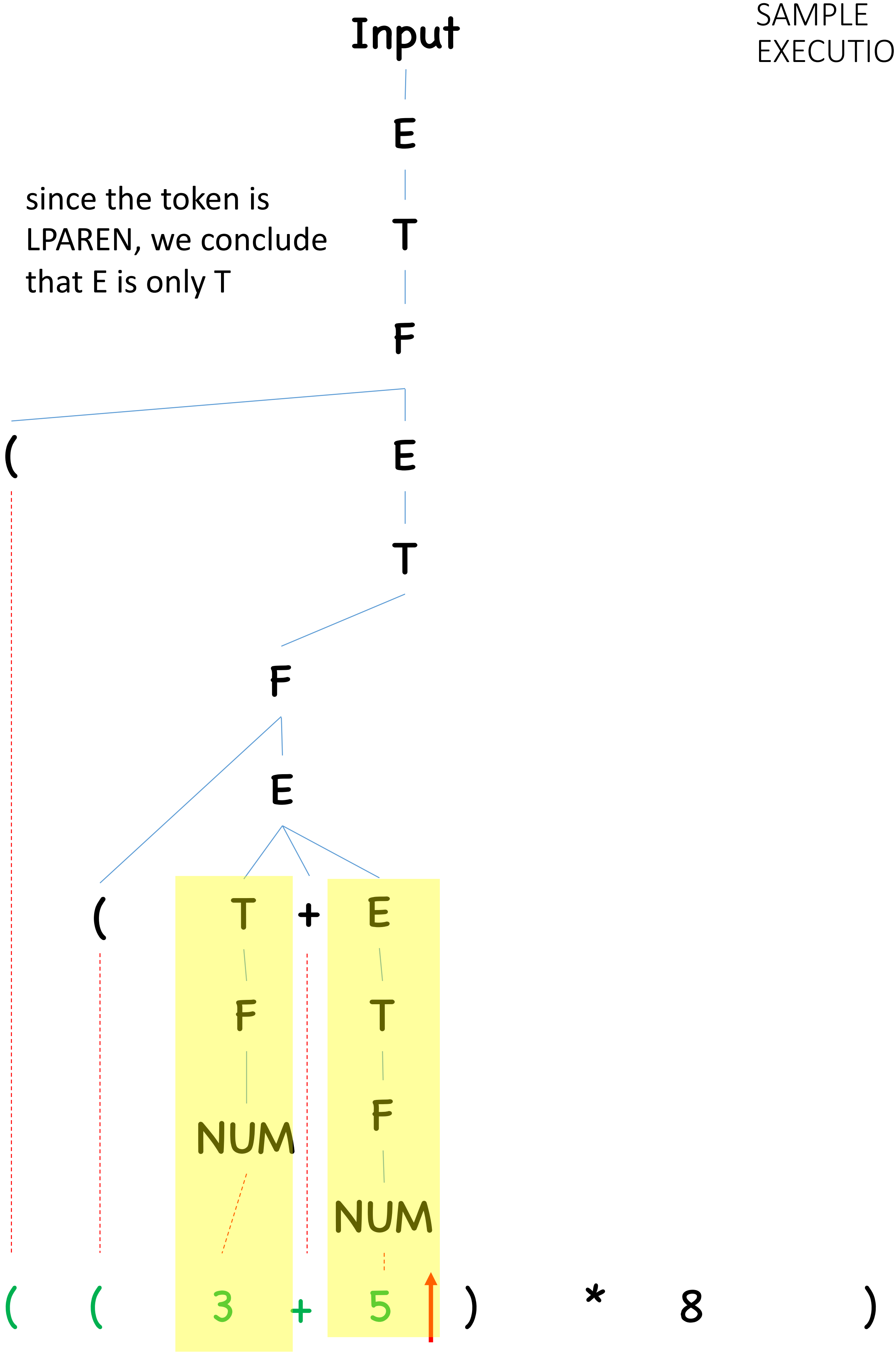
since the token is LPAREN, we conclude that T is only F

SAMPLE EXECUTION

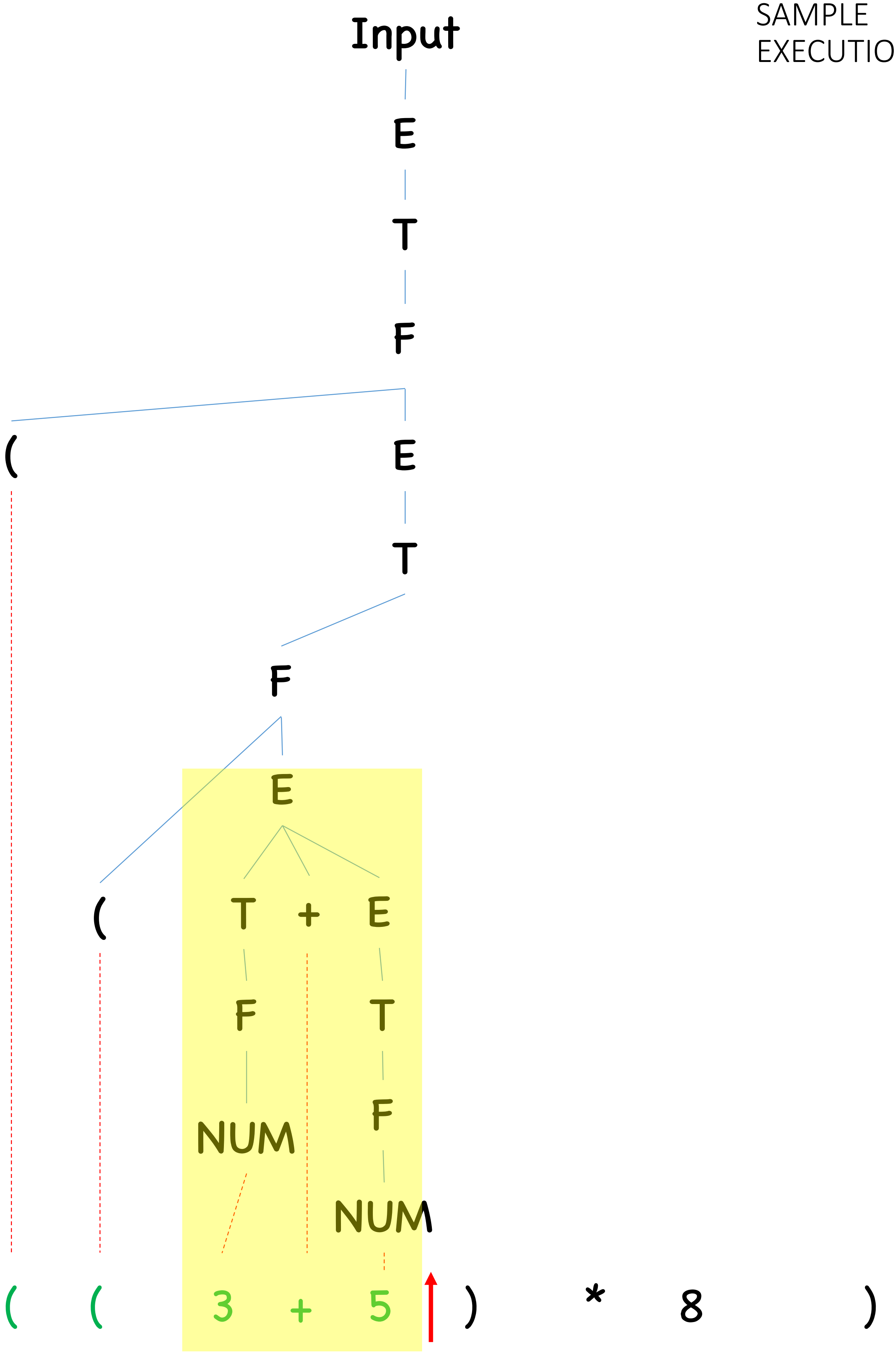


SAMPLE
EXECUTION

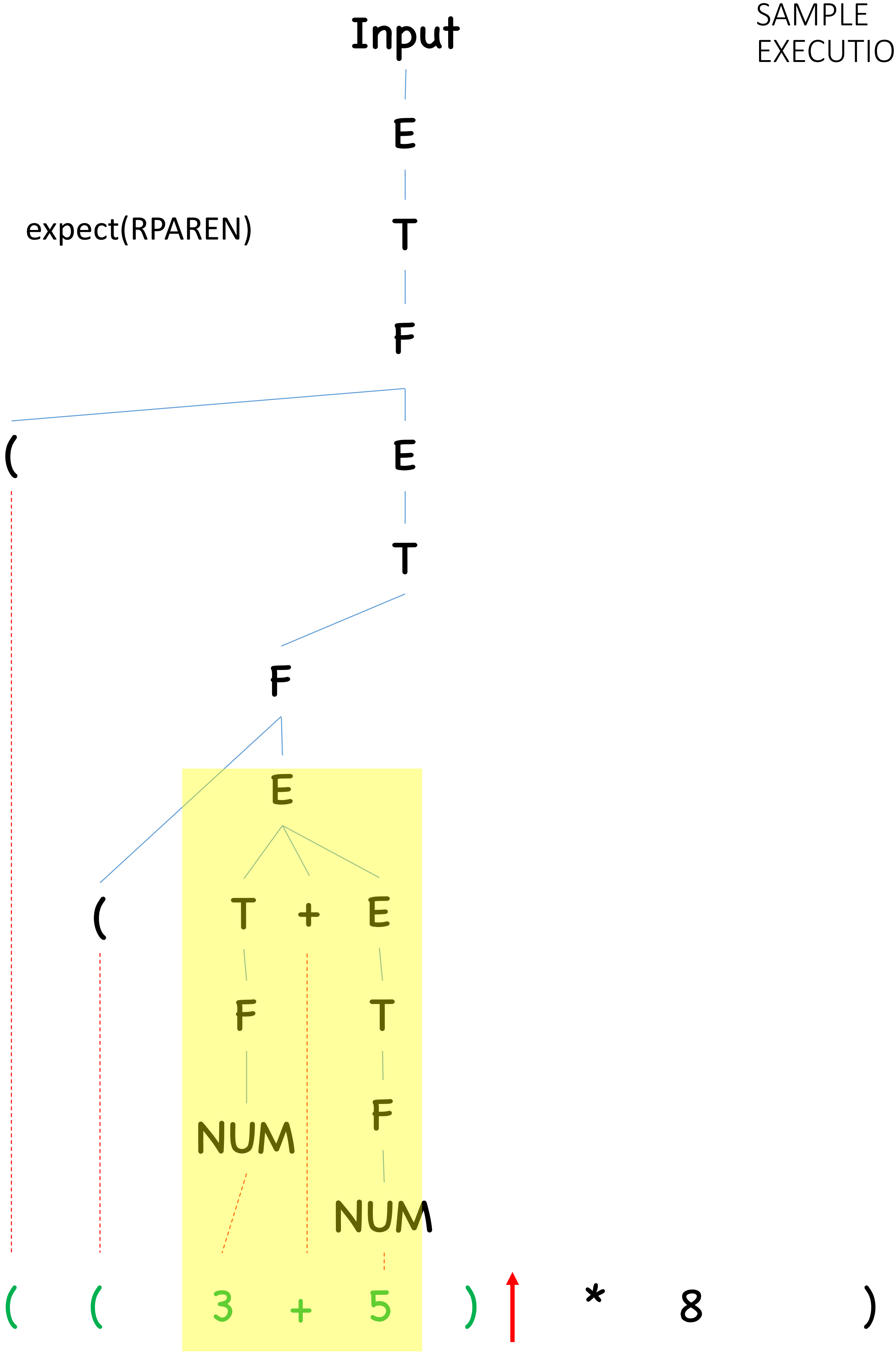
since the token is LPAREN, we conclude that E is only T



SAMPLE
EXECUTION

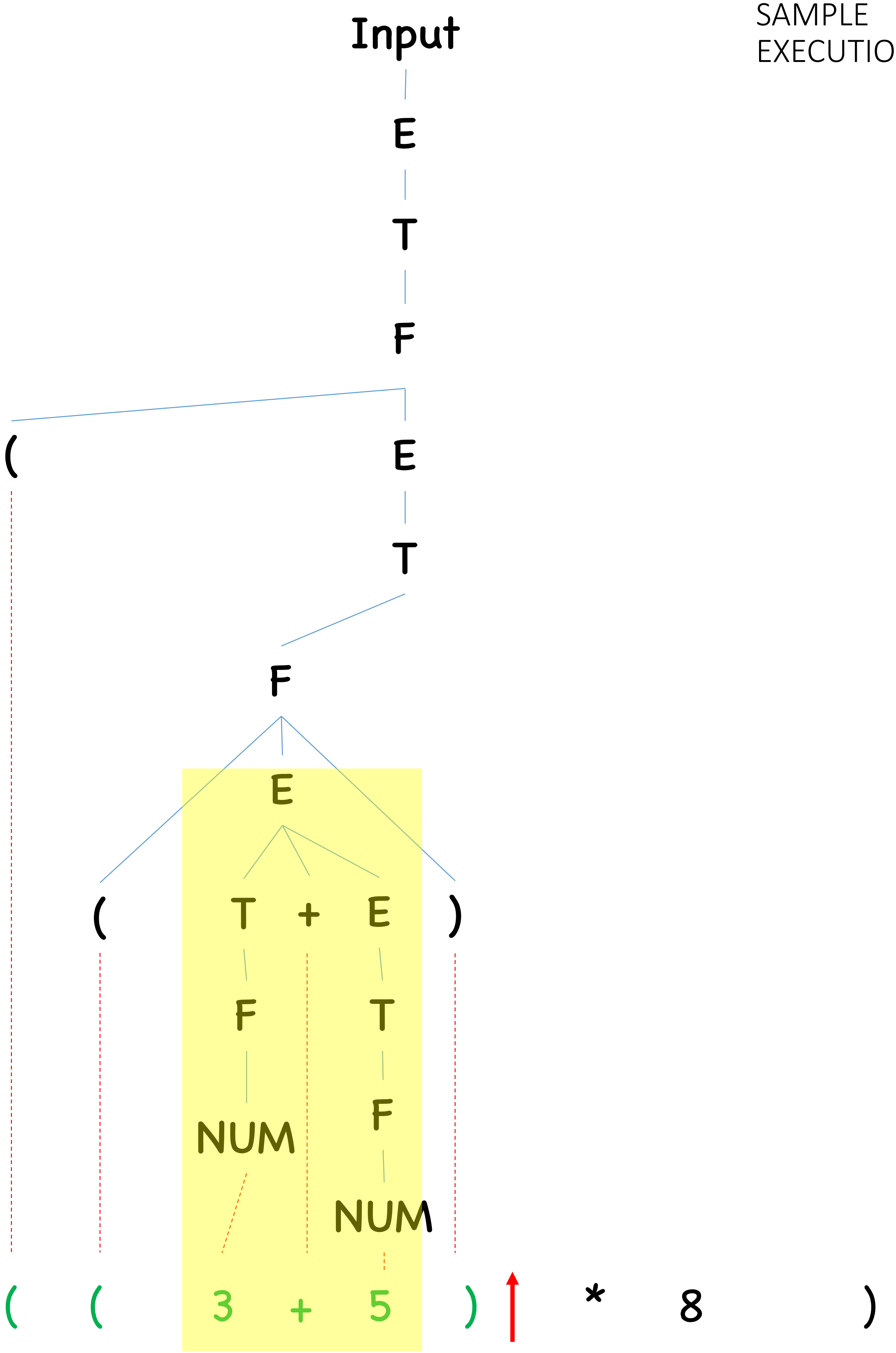


SAMPLE
EXECUTION

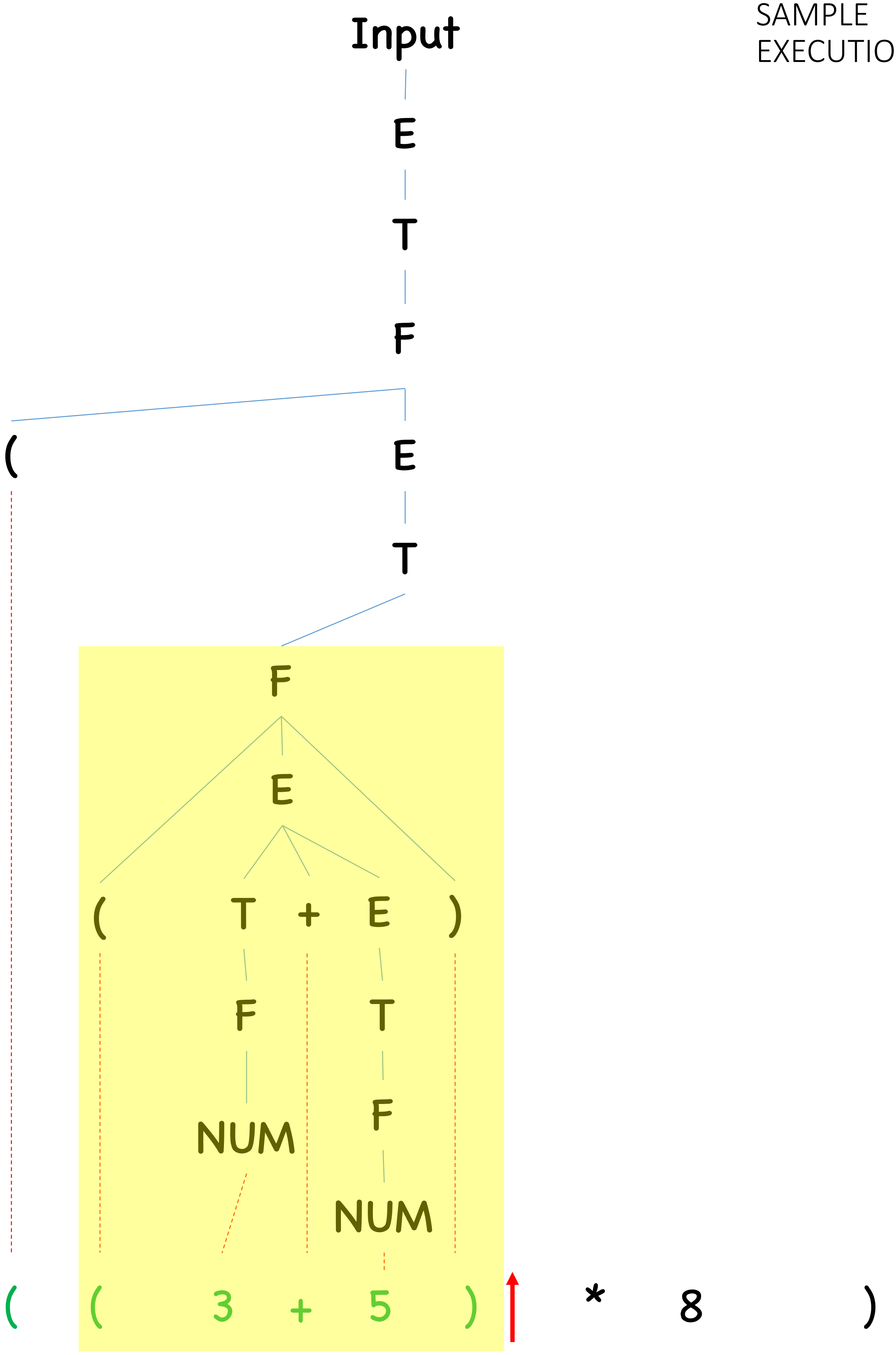


expect(RPAREN)

SAMPLE EXECUTION

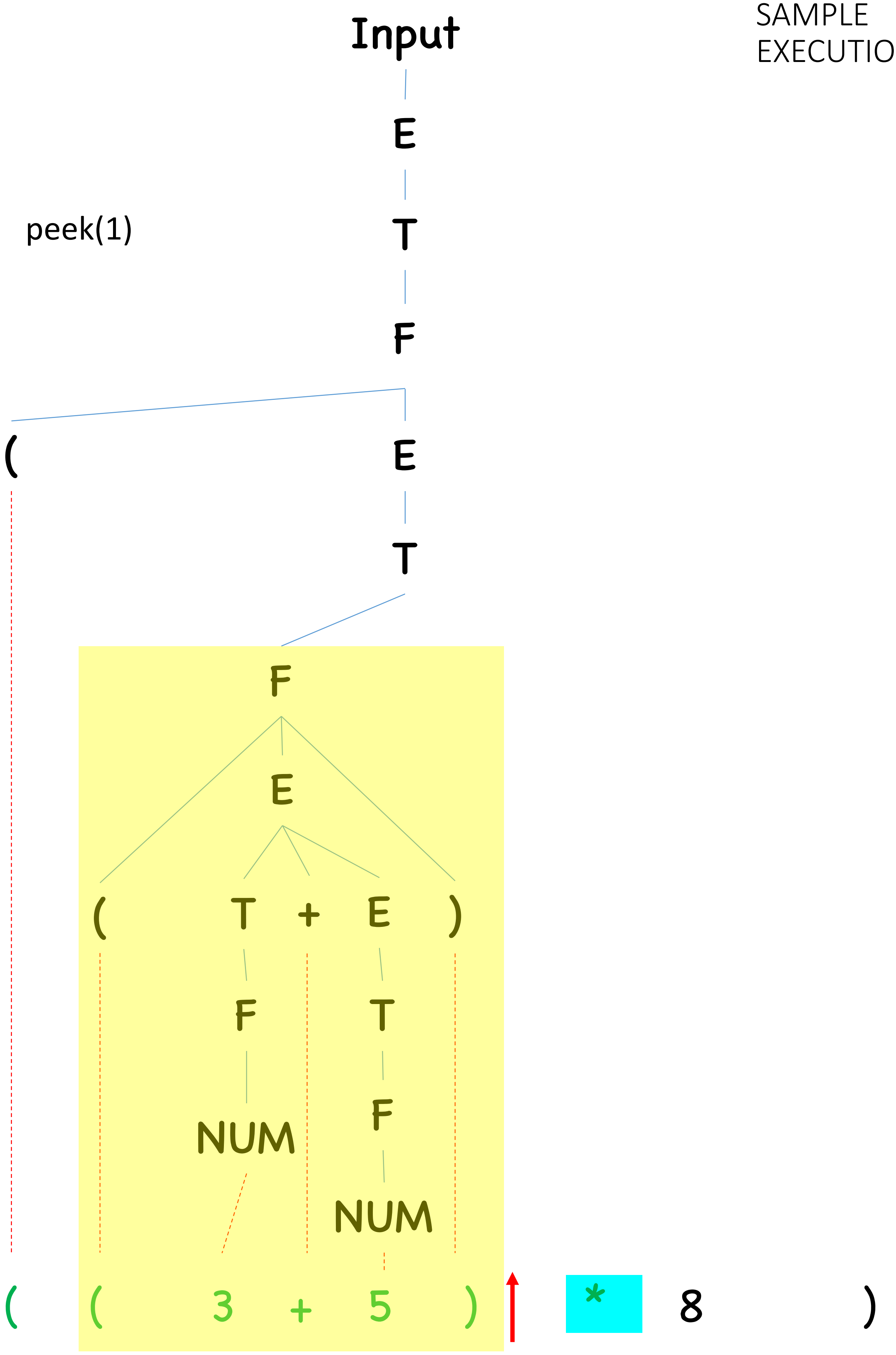


SAMPLE
EXECUTION



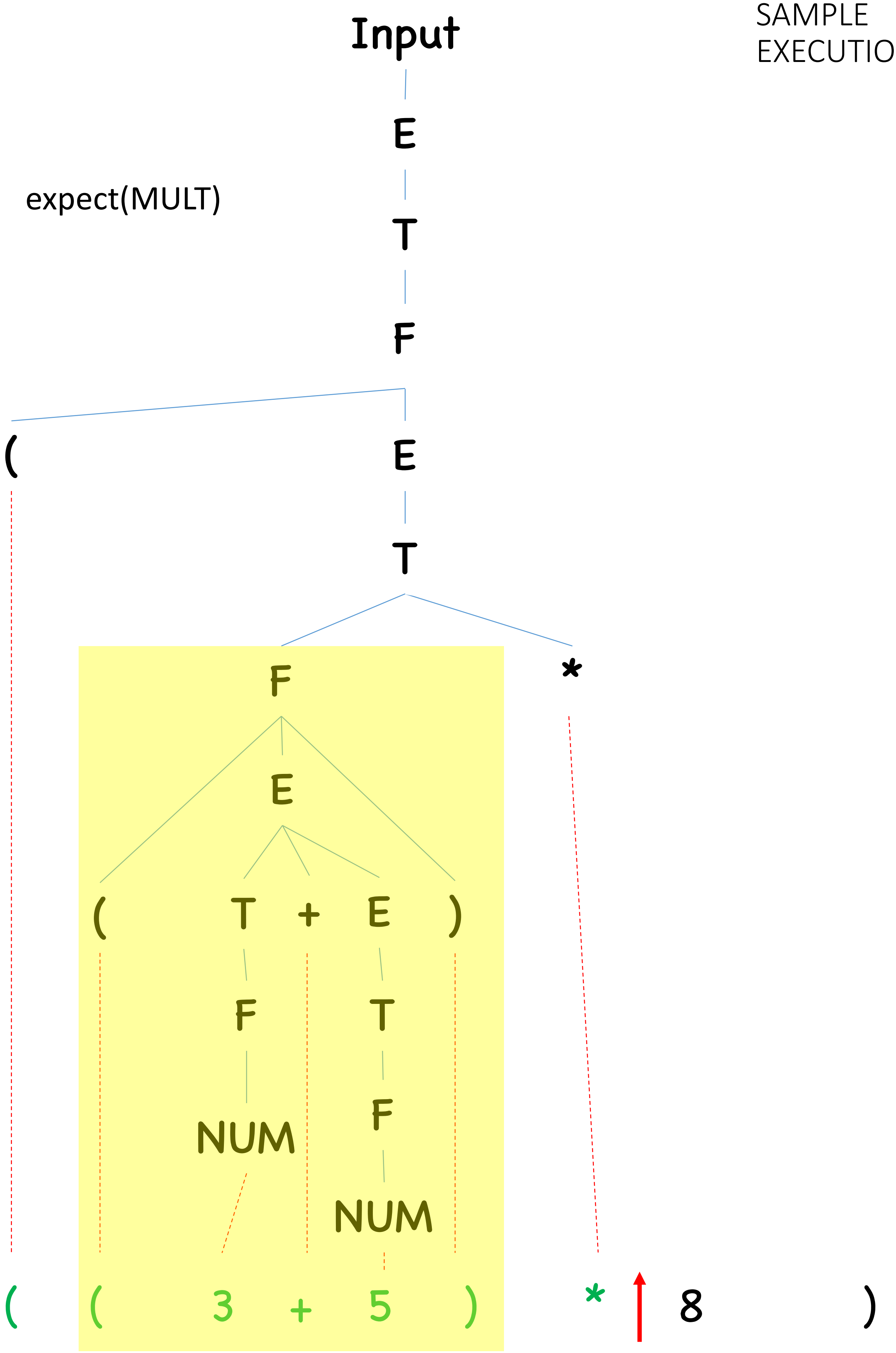
SAMPLE EXECUTION

peek(1)

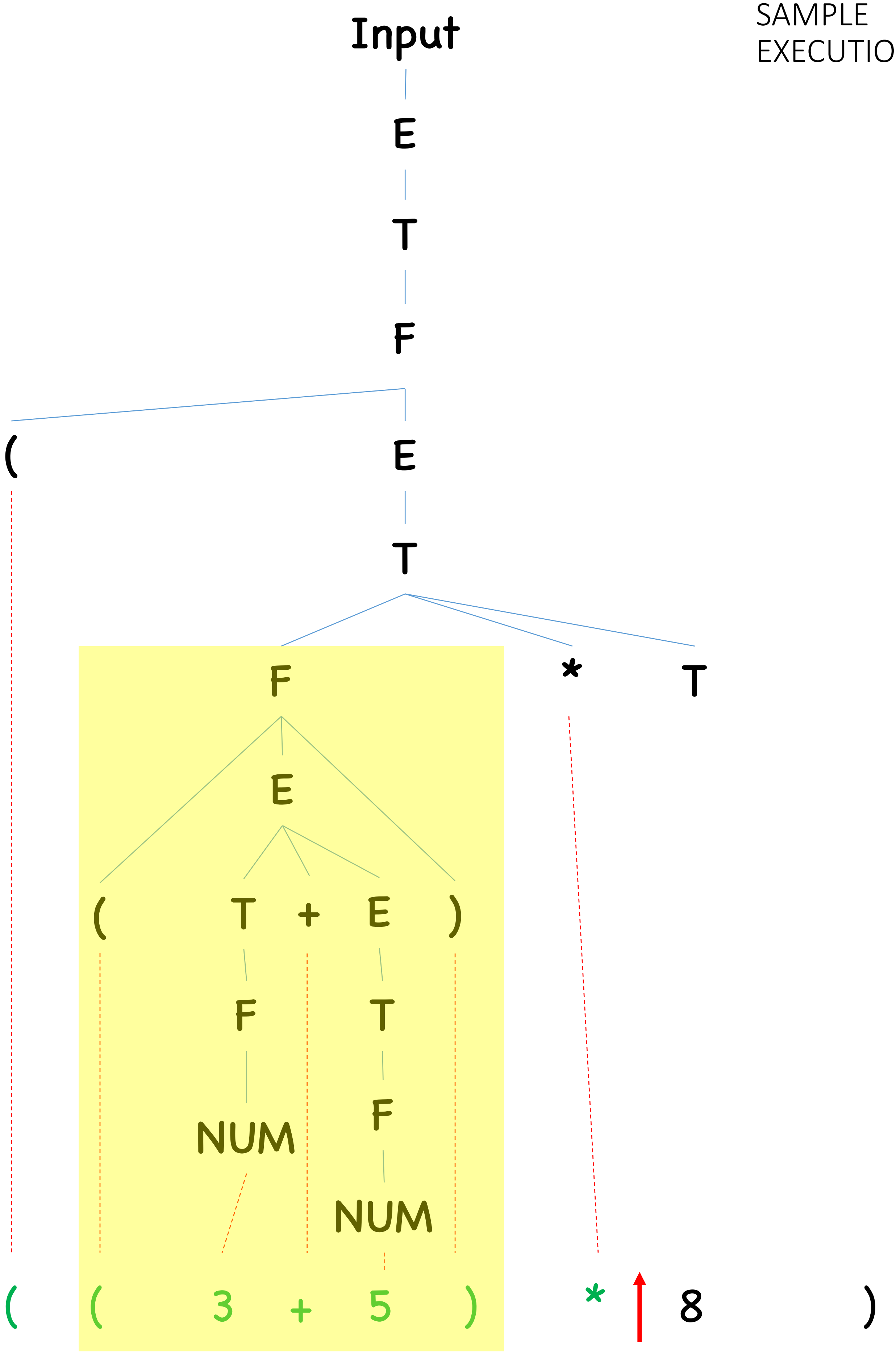


SAMPLE
EXECUTION

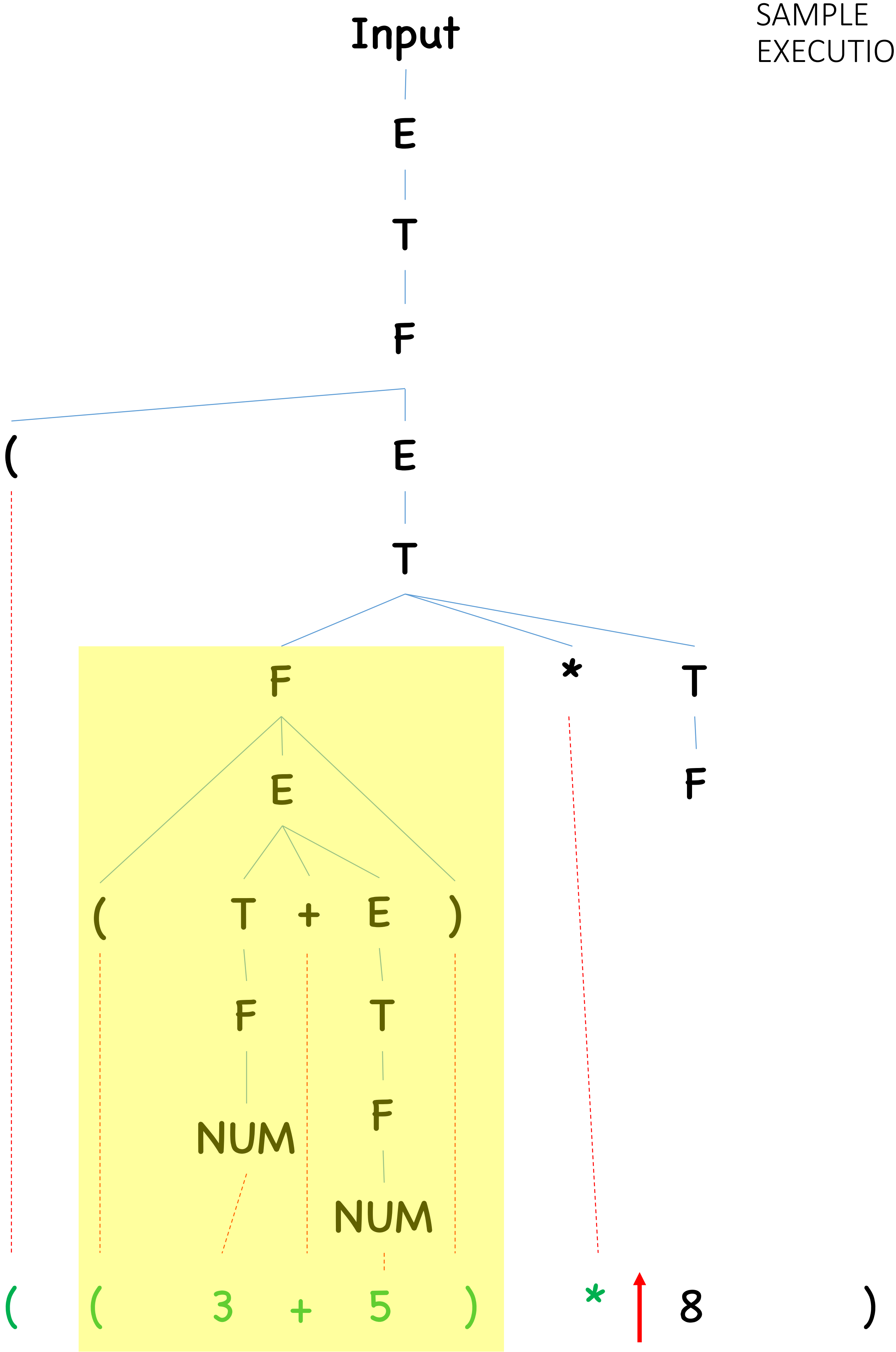
expect(MULT)



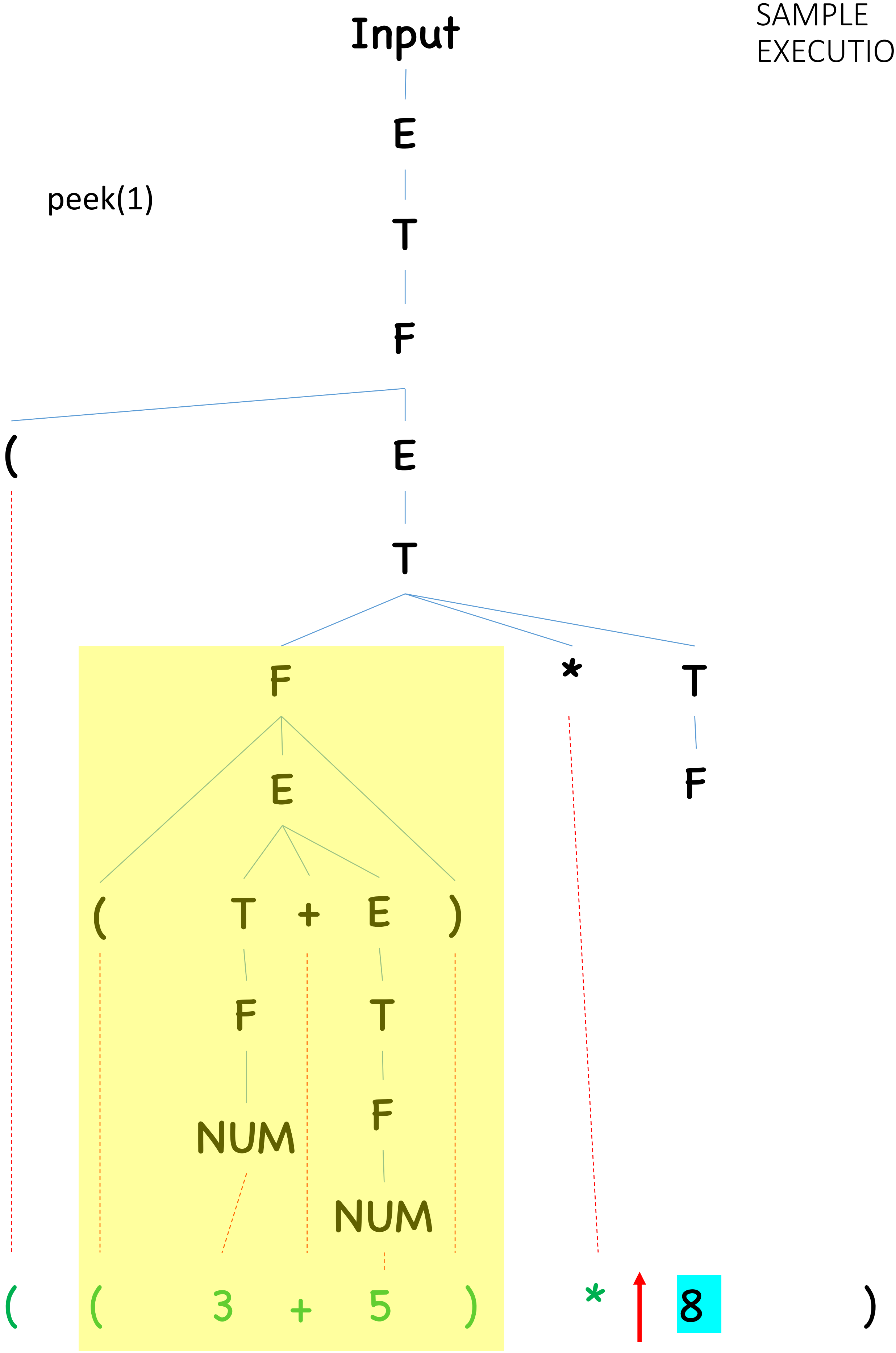
SAMPLE EXECUTION



SAMPLE EXECUTION

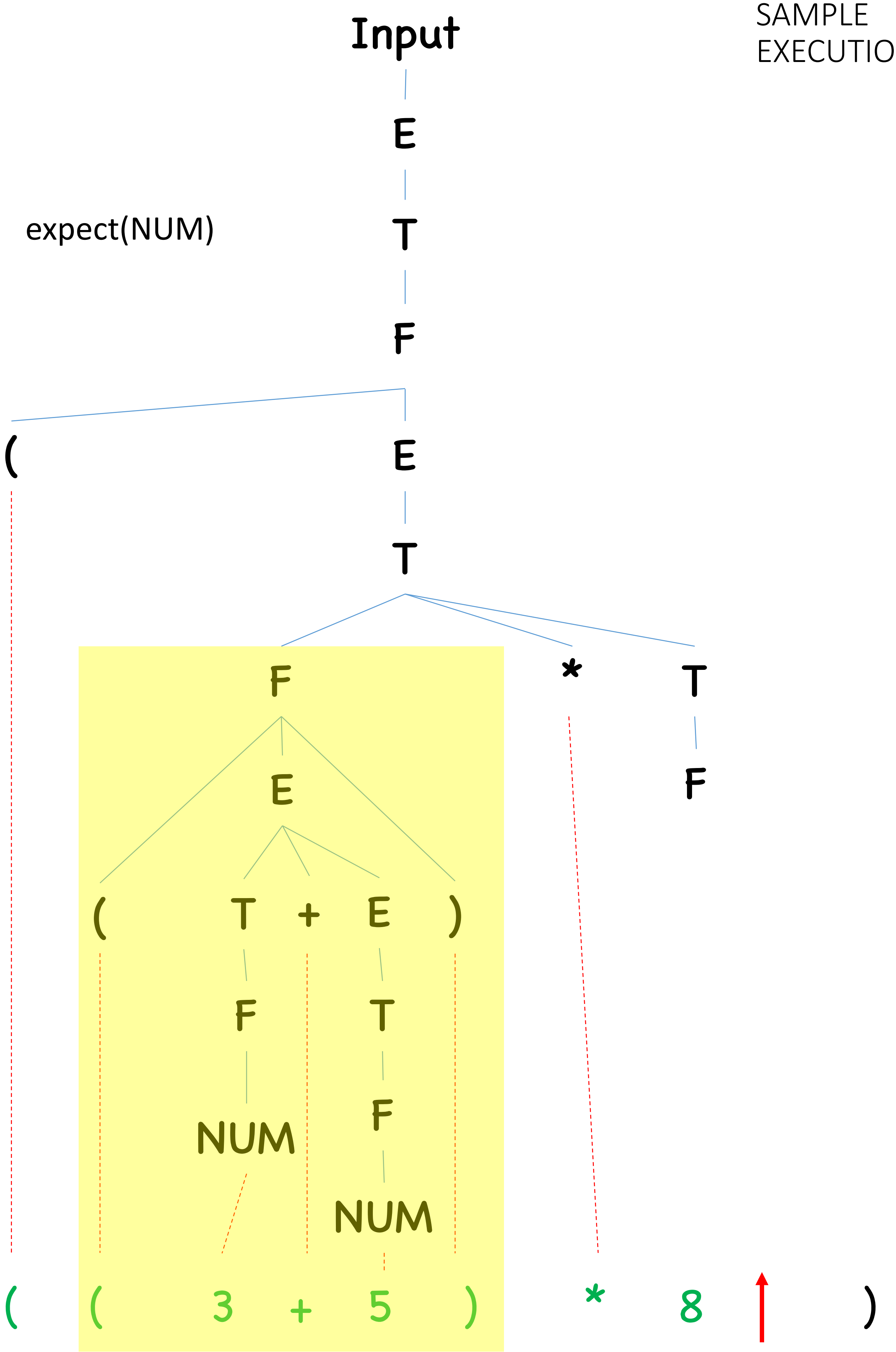


SAMPLE
EXECUTION



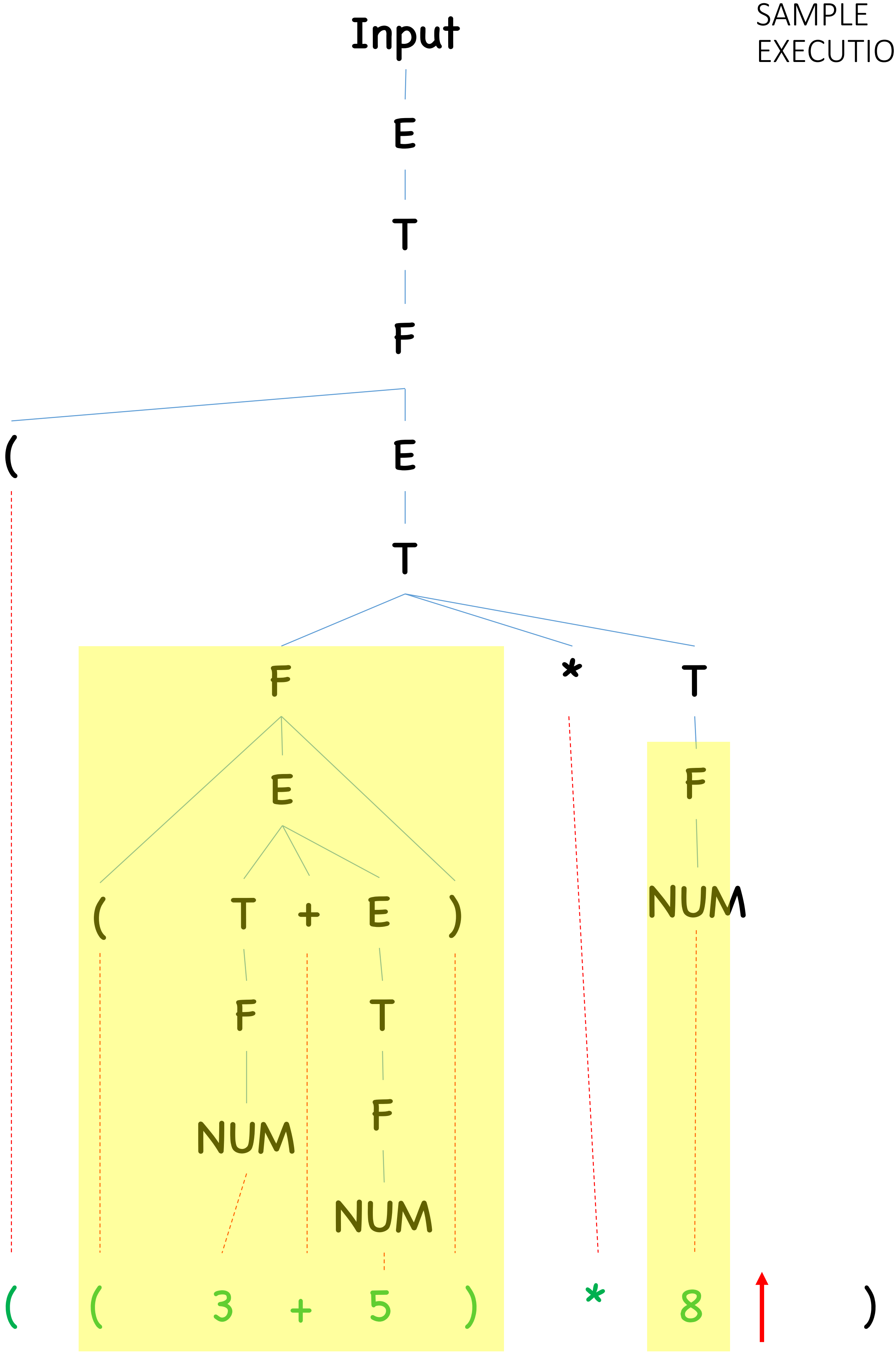
peek(1)

SAMPLE
EXECUTION



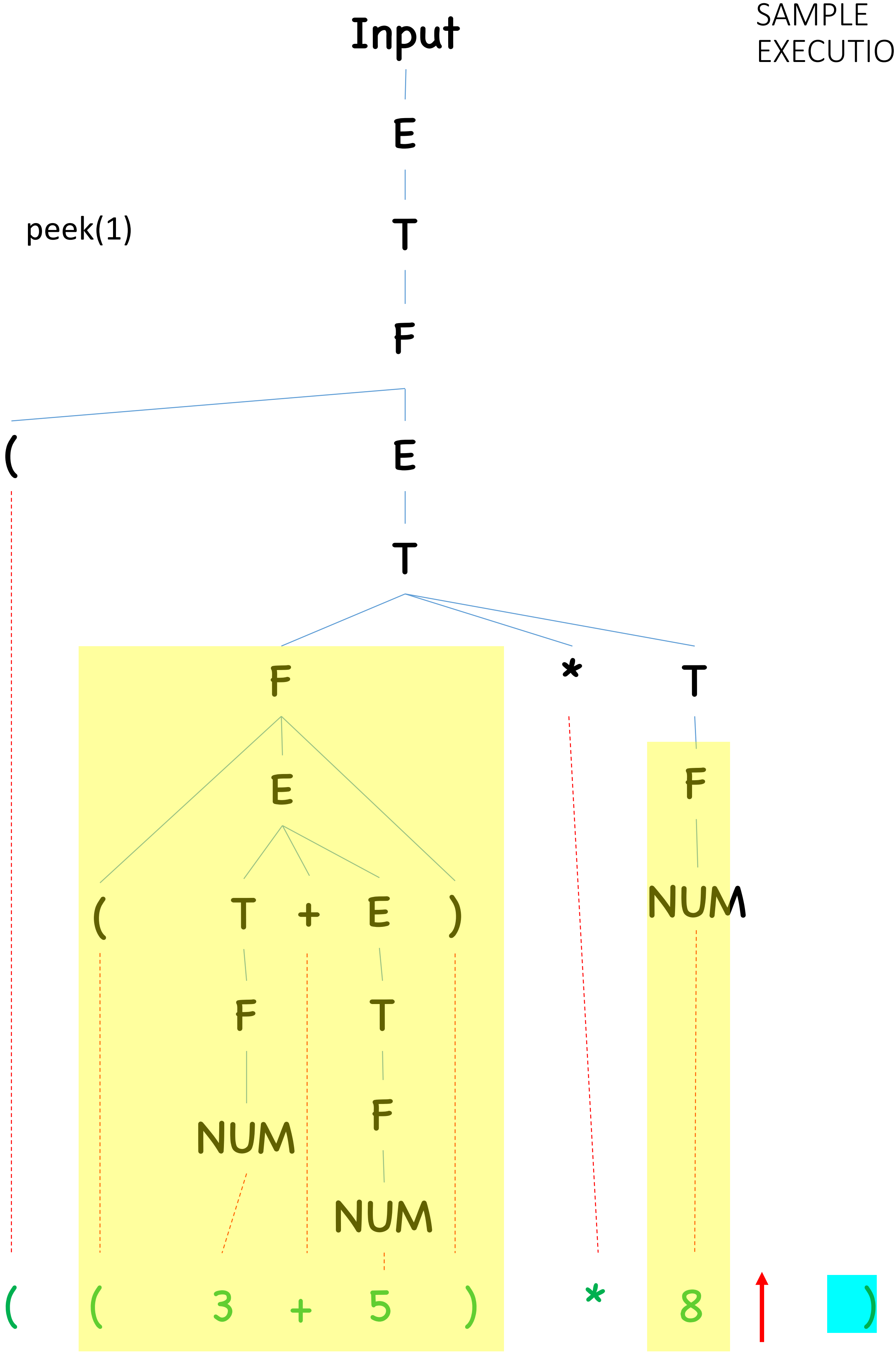
expect(NUM)

SAMPLE EXECUTION

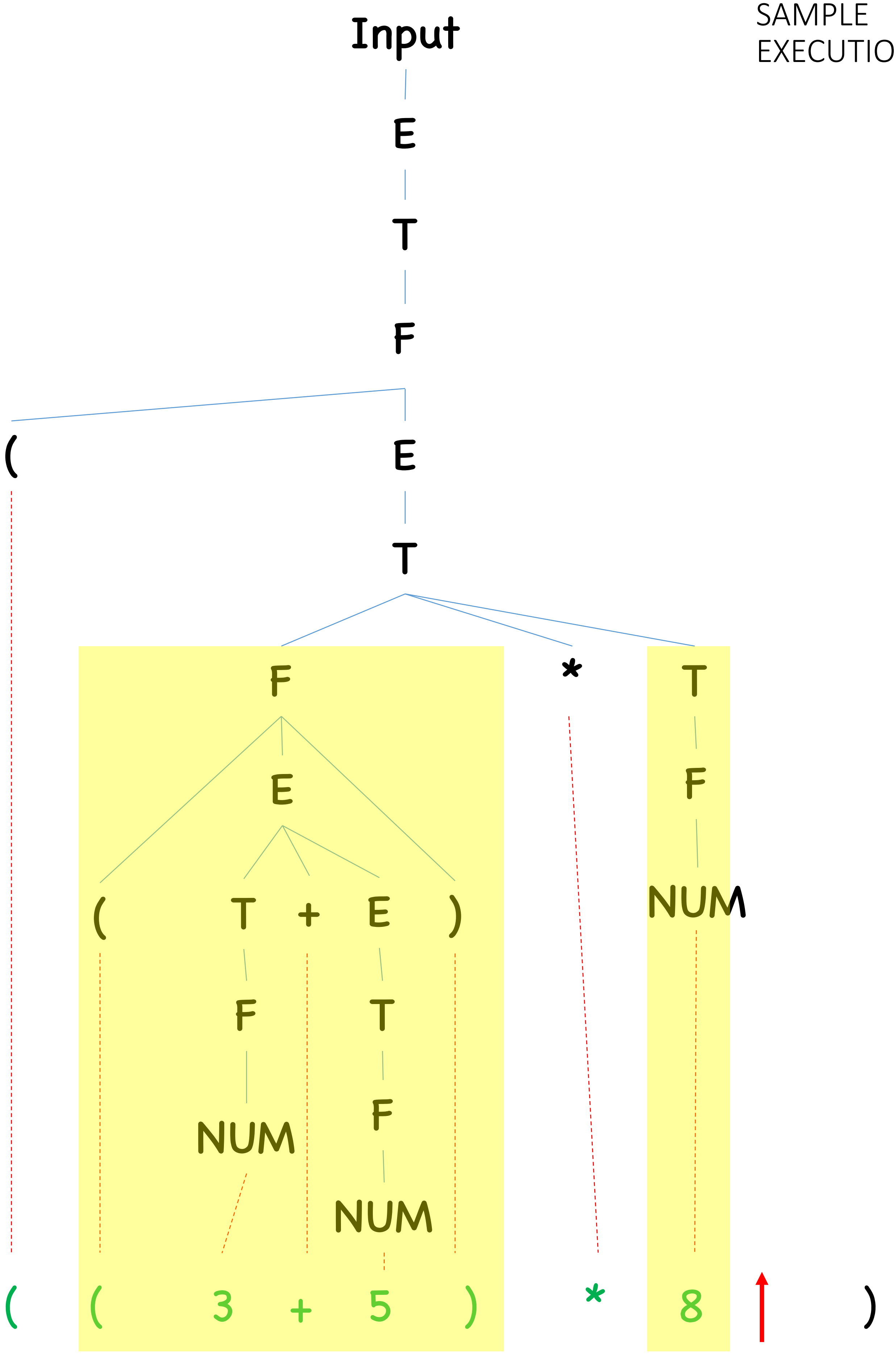


SAMPLE
EXECUTION

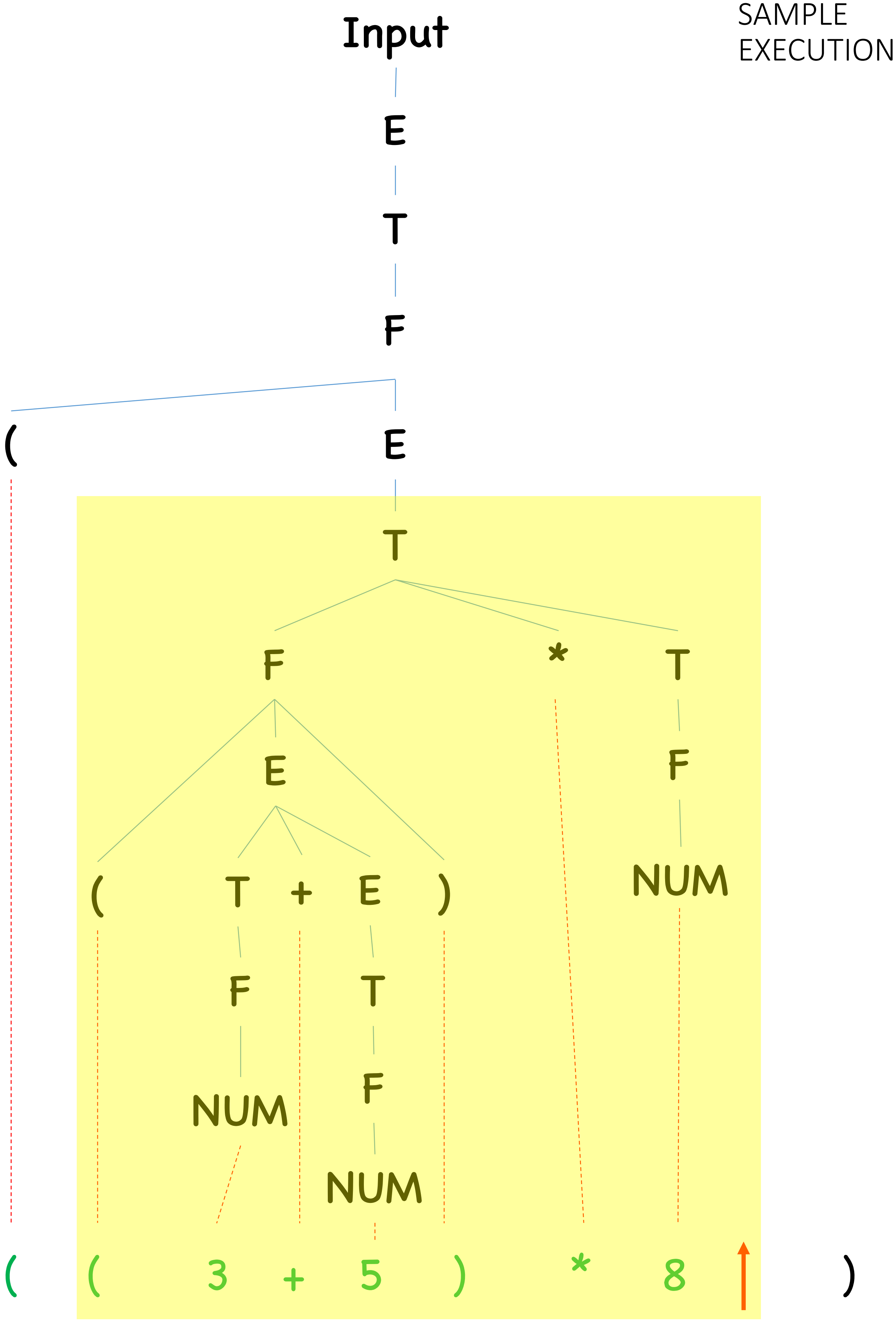
peek(1)



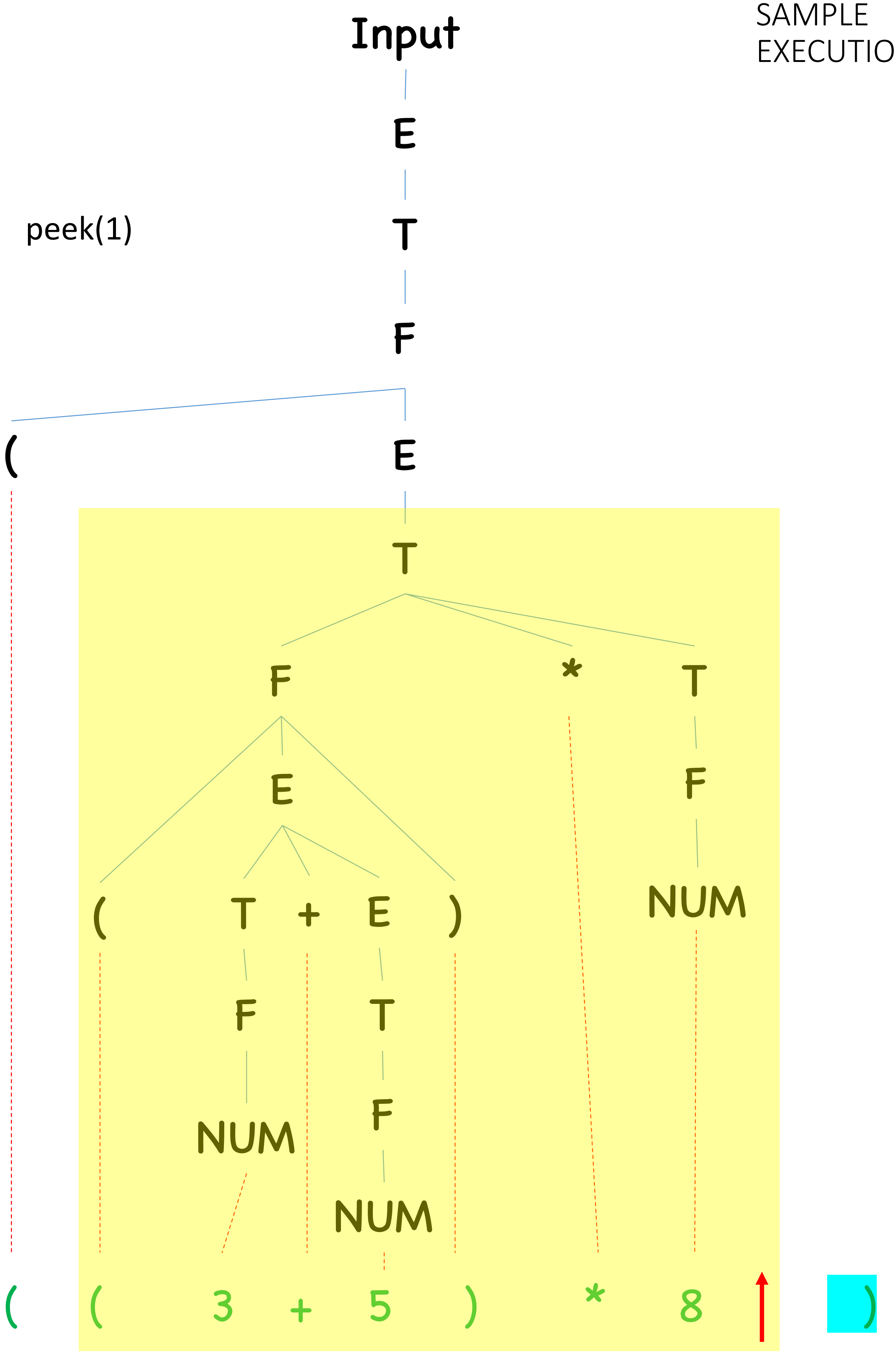
SAMPLE
EXECUTION



SAMPLE EXECUTION

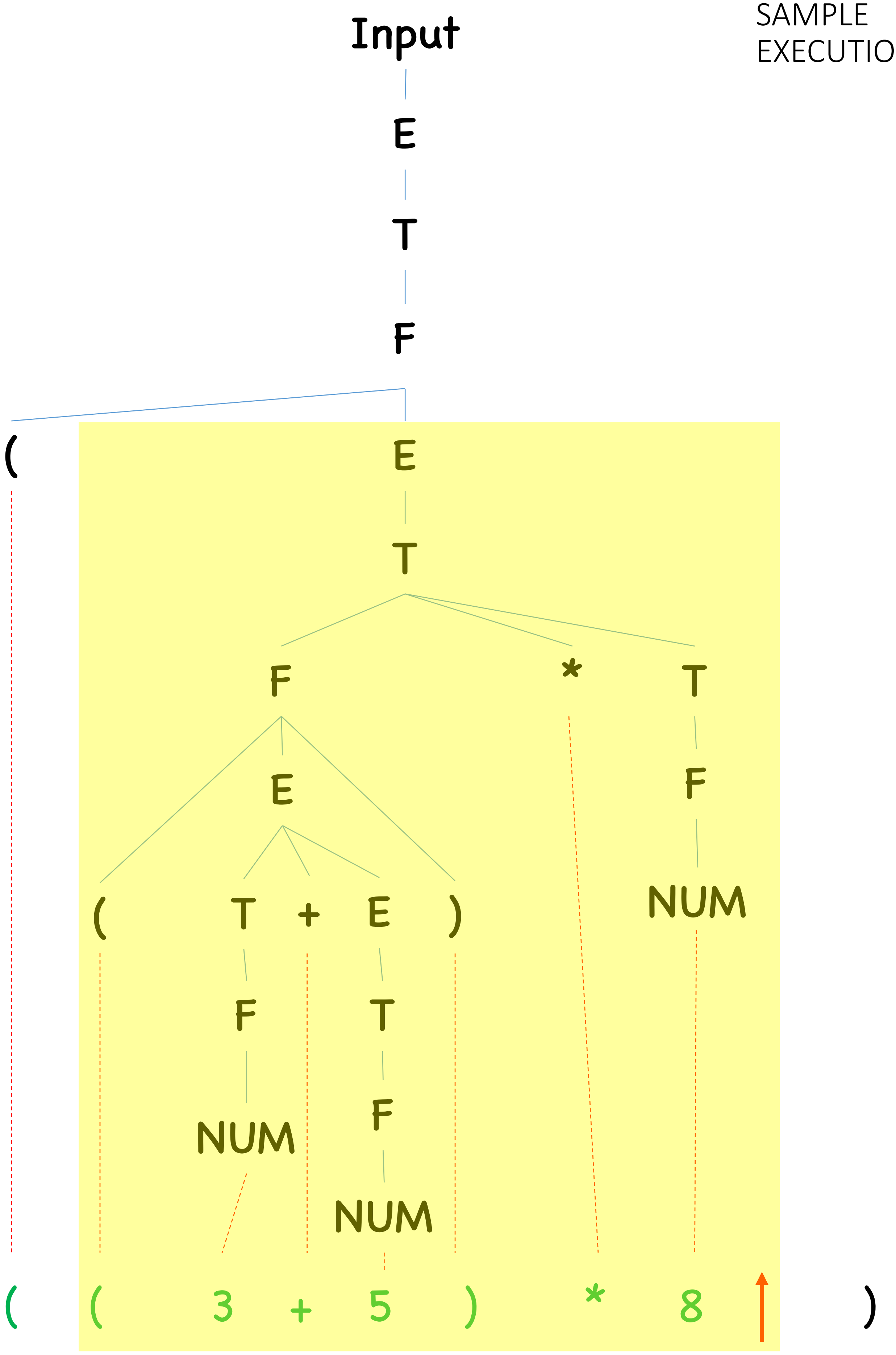


SAMPLE
EXECUTION



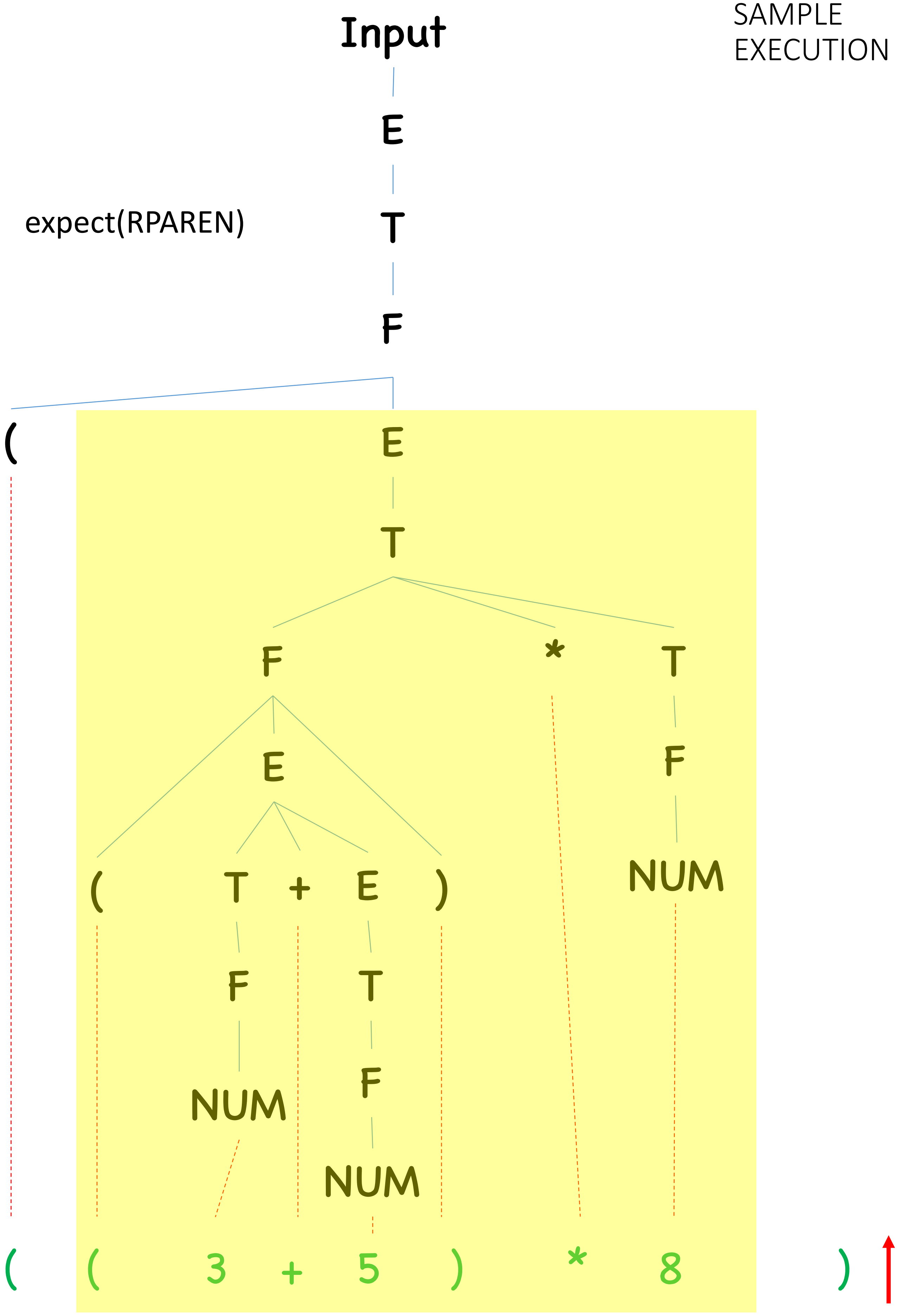
peek(1)

SAMPLE
EXECUTION

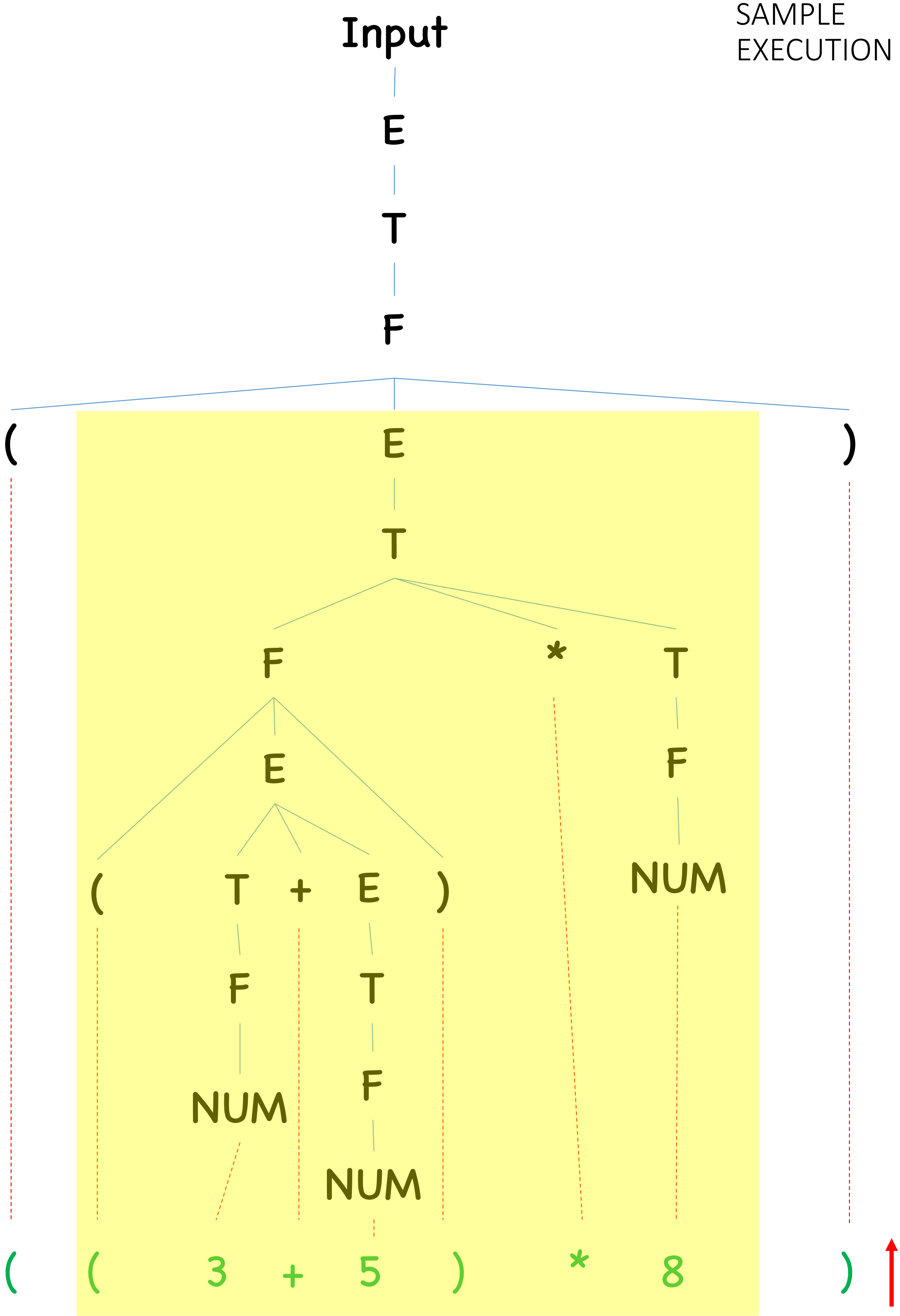


SAMPLE EXECUTION

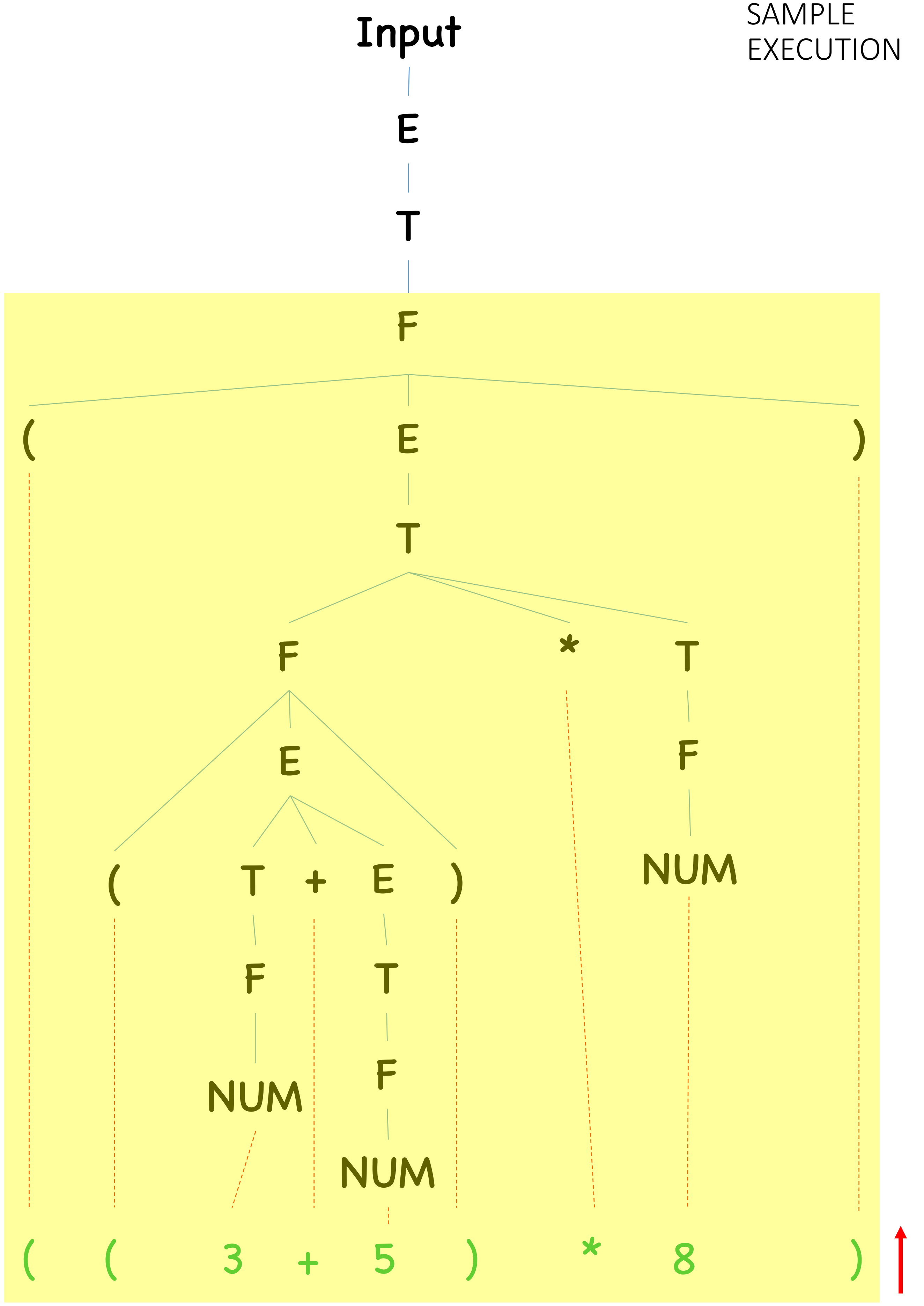
expect(RPAREN)



SAMPLE
EXECUTION

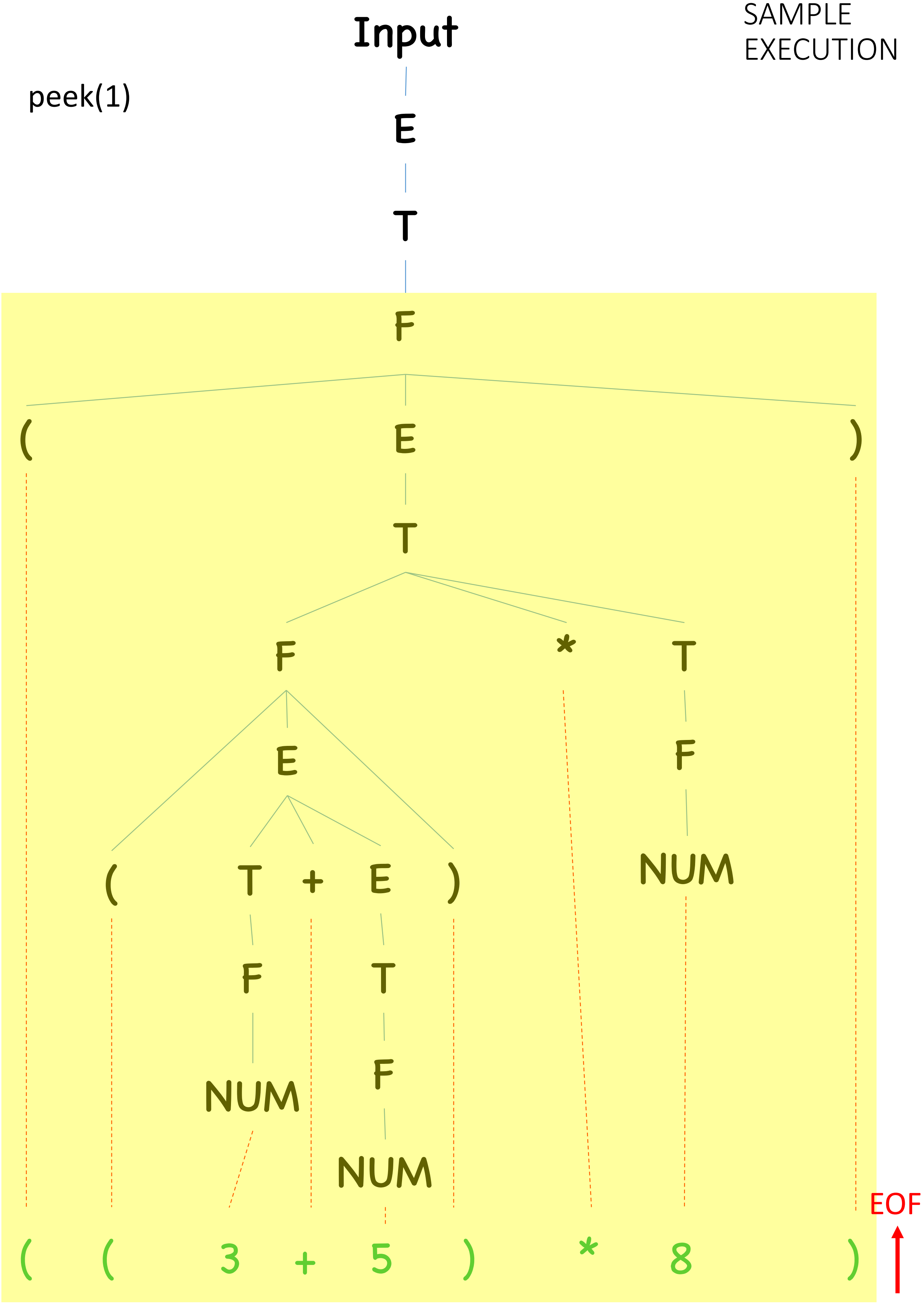


SAMPLE EXECUTION

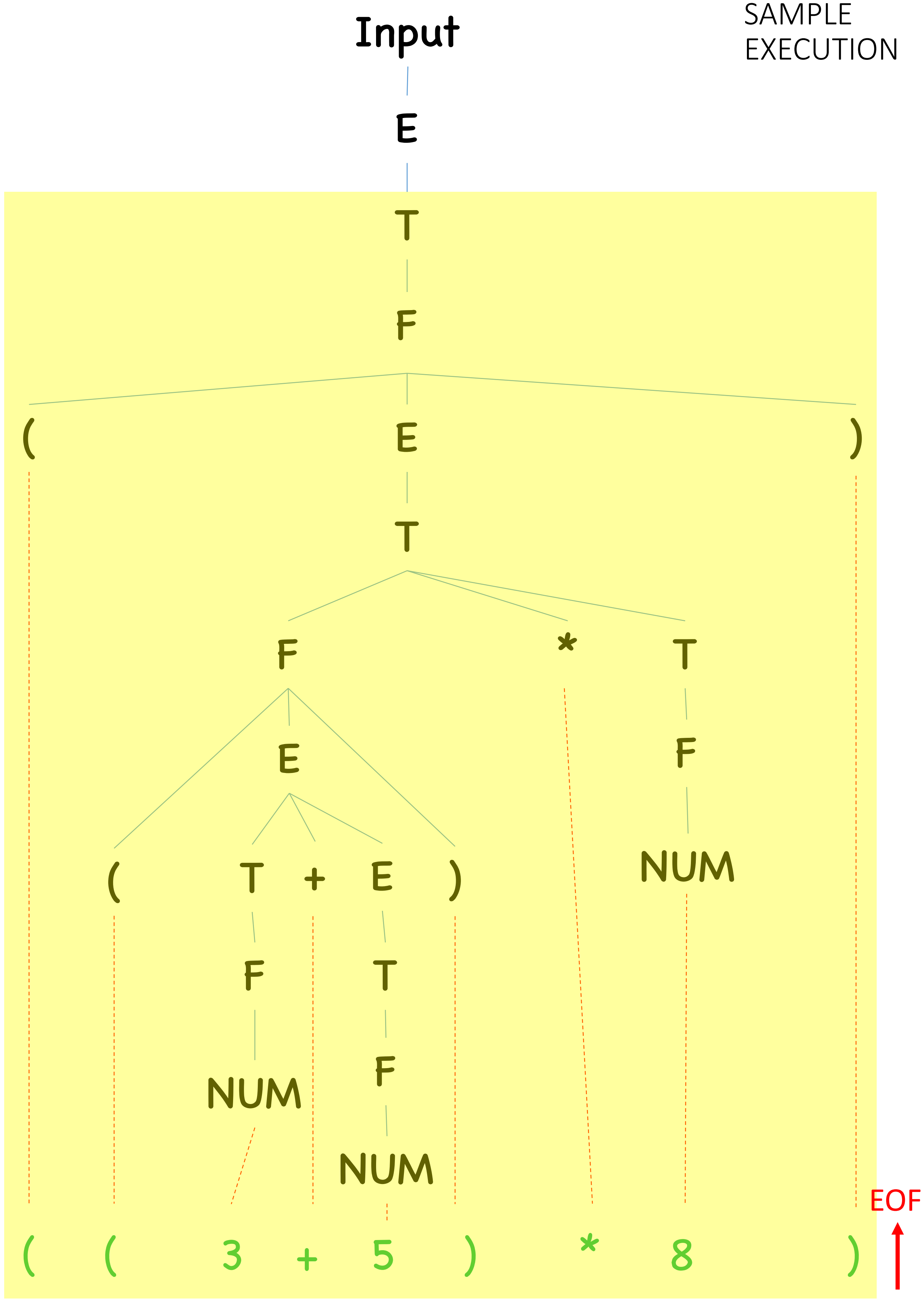


SAMPLE
EXECUTION

peek(1)

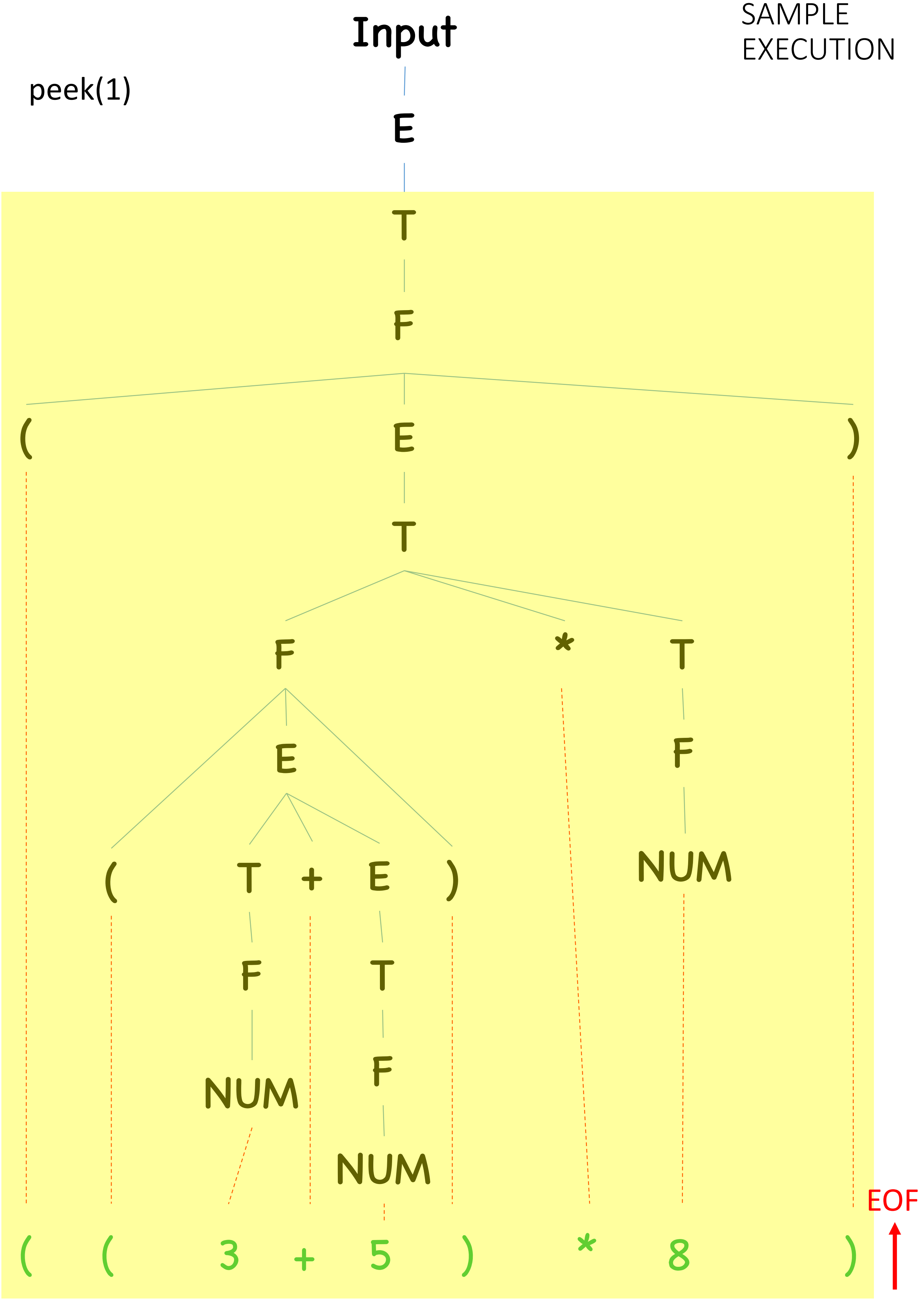


SAMPLE
EXECUTION

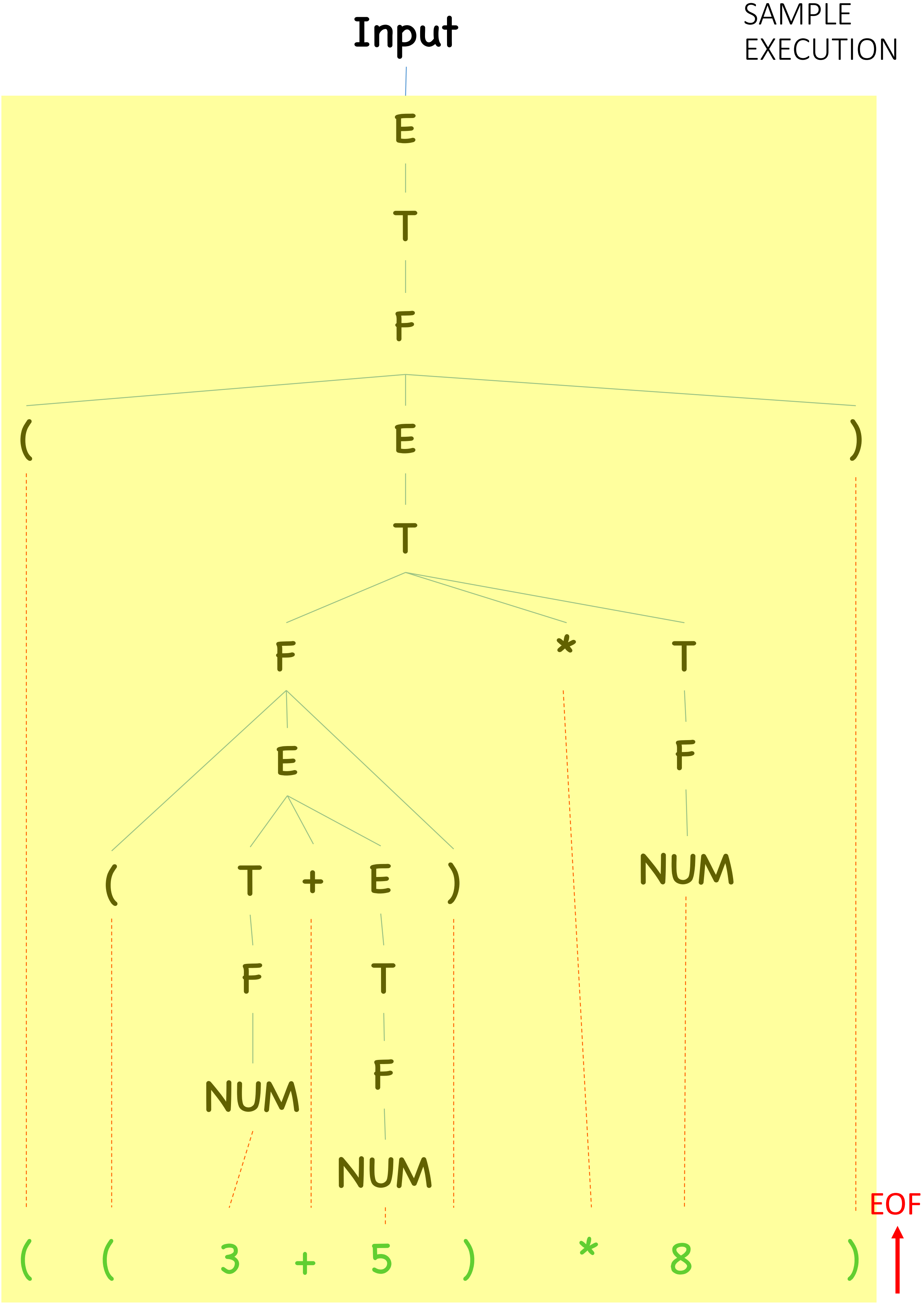


SAMPLE
EXECUTION

peek(1)



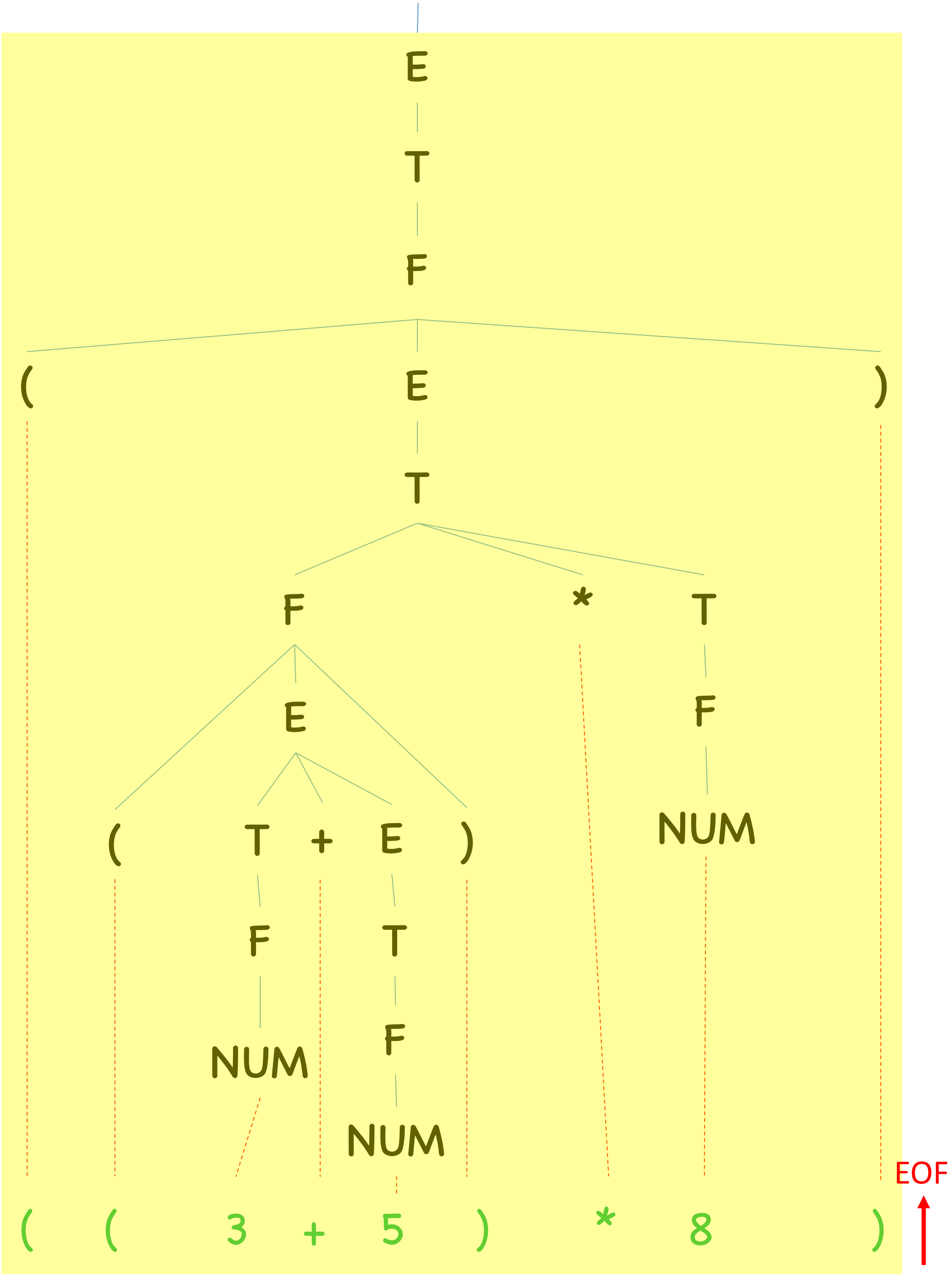
SAMPLE
EXECUTION



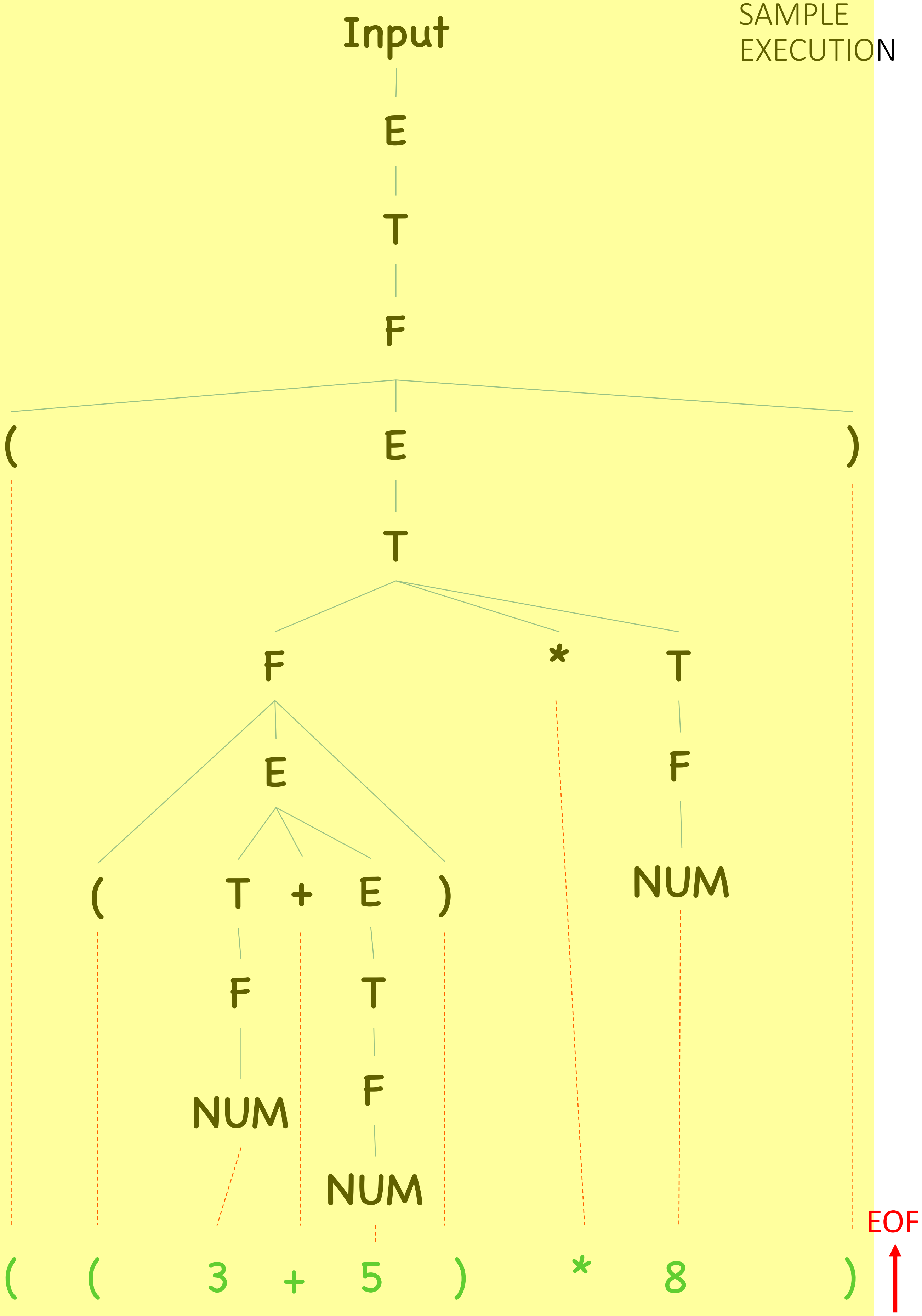
expect(EOF)

Input

SAMPLE EXECUTION



SAMPLE EXECUTION



SAMPLE EXECUTION

```
( ( 3 + 5 ) * 8 )
```

```
parse_input()
```

SAMPLE EXECUTION

```
( ( 3 + 5 ) * 8 )
```

```
parse_input()
       parse_expr()
```

SAMPLE EXECUTION

```
( ( 3 + 5 ) * 8 )
```

```
parse_input()
       parse_expr()
              parse_term()
```

SAMPLE EXECUTION

```
( ( 3 + 5 ) * 8 )
```

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
        parse_expr()
                parse_term()
                        parse_factor()
                           (    peek(1); expect(LPAREN);                                      (
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                  (
                            parse_expr()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);
                            parse_expr()
                                   parse_term()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                       (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
      parse_expr()
            parse_term()
                  parse_factor()
                      (  peek(1); expect(LPAREN);                                     (
                         parse_expr()
                               parse_term()
                                     parse_factor()
                                        (   peek(1); expect(LPAREN);                  (
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                  (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                              (   peek(1); expect(LPAREN);            (
                                                  parse_expr()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                           (    peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                              parse_factor()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (    peek(1); expect(LPAREN);                 (
                                                 parse_expr()
                                                        parse_term()
                                                               parse_factor()
                                                                      peek(1);expect(NUM); 3
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                          F   parse_factor()
                                                                     peek(1);expect(NUM); 3
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (    peek(1); expect(LPAREN);                 (
                                                 parse_expr()
                                                        parse_term()
                                                           F   parse_factor()
                                                                      peek(1);expect(NUM); 3
                                                               peek()                     +
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
            (   peek(1); expect(LPAREN);                                  (
                parse_expr()
                    parse_term()
                        parse_factor()
                        (   peek(1); expect(LPAREN);                      (
                            parse_expr()
                                parse_term()
                            T     F parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                                +
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
              (  peek(1); expect(LPAREN);                         (
                 parse_expr()
                     parse_term()
                         parse_factor()
                           (  peek(1); expect(LPAREN);            (
                              parse_expr()
                                  parse_term()
                            T       F parse_factor()
                                          peek(1);expect(NUM);    3
                                      peek()                      +
                                  peek(1);expect(PLUS);           +
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
              ( peek(1); expect(LPAREN);                (
                parse_expr()
                    parse_term()
                        parse_factor()
                          ( peek(1); expect(LPAREN);    (
                            parse_expr()
                                parse_term()
                              T   F parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()              +

                              + peek(1);expect(PLUS);   +
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                           F  parse_factor()
                                                   T                 peek(1);expect(NUM); 3
                                                              peek()                      +

                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (  peek(1); expect(LPAREN);                                (
                           parse_expr()
                                  parse_term()
                                         parse_factor()
                                           (    peek(1); expect(LPAREN);           (
                                                parse_expr()
                                                       parse_term()
                                                  T      F  parse_factor()
                                                                   peek(1);expect(NUM); 3
                                                            peek()                    +
                                                  +    peek(1);expect(PLUS);          +
                                                       parse_expr()
                                                            parse_term()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
              (  peek(1); expect(LPAREN);                        (
                 parse_expr()
                     parse_term()
                         parse_factor()
                           (  peek(1); expect(LPAREN);           (
                              parse_expr()
                                  parse_term()
                             T      F parse_factor()
                                          peek(1);expect(NUM);   3
                                      peek()                     +
                             +    peek(1);expect(PLUS);          +
                                  parse_expr()
                                      parse_term()
                                          parse_factor()
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (    peek(1); expect(LPAREN);                 (
                                                 parse_expr()
                                                        parse_term()
                                                           F   parse_factor()
                                                   T                  peek(1);expect(NUM); 3
                                                               peek()                     +
                                                   +    peek(1);expect(PLUS);             +
                                                        parse_expr()
                                                               parse_term()
                                                                      parse_factor()
                                                                        peek(1);expect(NUM); 5
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (    peek(1); expect(LPAREN);                 (
                                                 parse_expr()
                                                        parse_term()
                                                           F   parse_factor()
                                                   T                  peek(1);expect(NUM); 3
                                                               peek()                     +
                                                   +    peek(1);expect(PLUS);             +
                                                        parse_expr()
                                                               parse_term()
                                                                      parse_factor()
                                                                 F      peek(1);expect(NUM); 5
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
             (  getToken()                                          (
                peek(1); expect(LPAREN);                            (
                parse_expr()
                    parse_term()
                        parse_factor()
                         (  peek(1); expect(LPAREN);                (
                            parse_expr()
                          T     parse_term()  F
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                          +
                          +
                                peek(1);expect(PLUS);               +
                                parse_expr()
                                    parse_term()
                                  F     parse_factor()
                                         peek(1);expect(NUM); 5

                                    peek()                          )
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (  peek(1); expect(LPAREN);                                  (
                           parse_expr()
                                  parse_term()
                                         parse_factor()
                                           (    peek(1); expect(LPAREN);             (
                                                parse_expr()
                                                       parse_term()
                                                             parse_factor()
                                                  T       F         peek(1);expect(NUM); 3
                                                             peek()                  +
                                                  +    peek(1);expect(PLUS);         +
                                                       parse_expr()
                                                             parse_term()
                                                                    parse_factor()
                                                          T     F     peek(1);expect(NUM); 5
                                                             peek()                  )
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                              (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);          (
                                                parse_expr()
                                                       parse_term()
                                                   T       F  parse_factor()
                                                                     peek(1);expect(NUM); 3
                                                              peek()              +
                                                   +   peek(1);expect(PLUS);      +
                                                       parse_expr()
                                                              parse_term()
                                                                     parse_factor()
                                                           T     F     peek(1);expect(NUM); 5
                                                              peek()              )
                                                       peek()                     )
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                   T       F  parse_factor()
                                                                     peek(1);expect(NUM); 3
                                                              peek()                      +
                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
                                                              parse_term()
                                                                     parse_factor()
                                                   E       T     F     peek(1);expect(NUM); 5
                                                              peek()                      )
                                                       peek()                             )
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                  (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                           (    peek(1); expect(LPAREN);              (
                                                parse_expr()
                                                       parse_term()
                                                              parse_factor()
                                                   T       F             peek(1);expect(NUM); 3
                                                              peek()                  +

                                                   +   peek(1);expect(PLUS);          +
                                                       parse_expr()
                                                              parse_term()
                                            E                        parse_factor()
                                                   E       T     F     peek(1);expect(NUM); 5
                                                              peek()                  )
                                                       peek()                         )
```

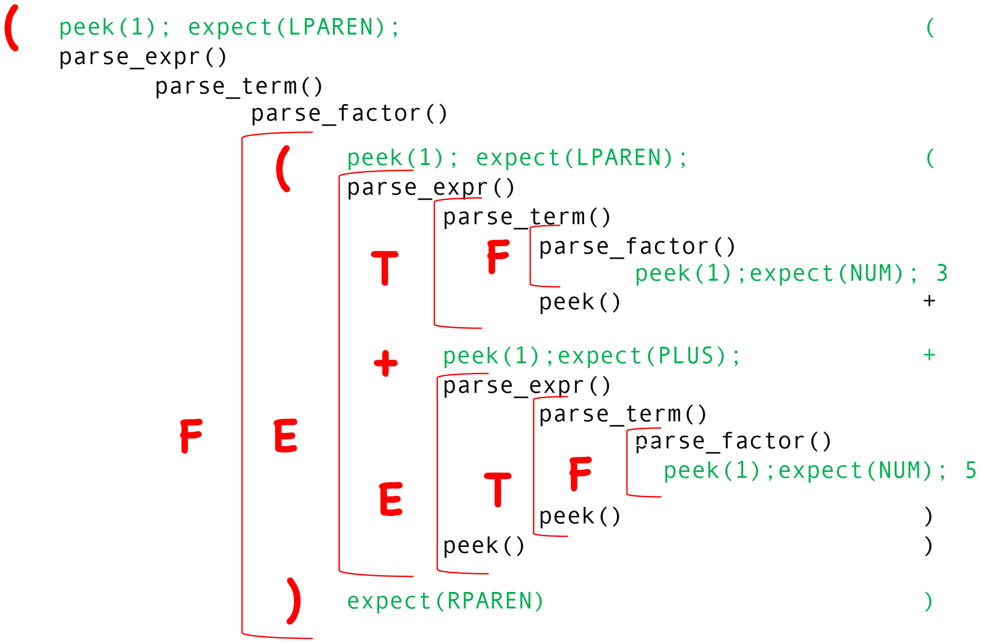
SAMPLE EXECUTION

((3+5)*8)

```
parse_input()
       parse_expr()
             parse_term()
                    parse_factor()
                       (   peek(1); expect(LPAREN);                                     (
                           parse_expr()
                                  parse_term()
                                        parse_factor()
                                          (    peek(1); expect(LPAREN);                 (
                                               parse_expr()
                                                      parse_term()
                                                  T      F  parse_factor()
                                                                   peek(1);expect(NUM); 3
                                                            peek()                      +
                                                  +   peek(1);expect(PLUS);             +
                                                      parse_expr()
                                                            parse_term()
                                           E                       parse_factor()
                                                               F     peek(1);expect(NUM); 5
                                                  E      T
                                                            peek()                      )
                                                      peek()                            )
                                          )    expect(RPAREN)                           )
```

SAMPLE EXECUTION

((3 + 5) * 8)



```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                            peek(1); expect(LPAREN);                                     (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                                 peek(1); expect(LPAREN);                (
                                                 parse_expr()
                                                        parse_term()
                                                               parse_factor()
                                                                      peek(1);expect(NUM); 3
                                                               peek()                    +
                                                        peek(1);expect(PLUS);            +
                                                        parse_expr()
                                                               parse_term()
                                                                      parse_factor()
                                                                        peek(1);expect(NUM); 5
                                                               peek()                    )
                                                        peek()                           )
                                                 expect(RPAREN)                          )
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                           F  parse_factor()
                                                   T                 peek(1);expect(NUM); 3
                                                              peek()                      +
                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
                                                              parse_term()
                                     F      E                        parse_factor()
                                                                 F     peek(1);expect(NUM); 5
                                                    E      T
                                                              peek()                      )
                                                       peek()                             )
                                             )  expect(RPAREN)                            )
                                         peek(1)                                          *
```

SAMPLE EXECUTION

((3+5)*8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                              parse_factor()
                                                   T       F         peek(1);expect(NUM); 3
                                                              peek()                      +
                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
                                                              parse_term()
                                    F       E                        parse_factor()
                                                    E      T     F     peek(1);expect(NUM); 5
                                                              peek()                      )
                                                       peek()                             )
                                             )  expect(RPAREN)                            )
                                         peek()                                           *
                                     *   gexpect(MULT)                                    *
```

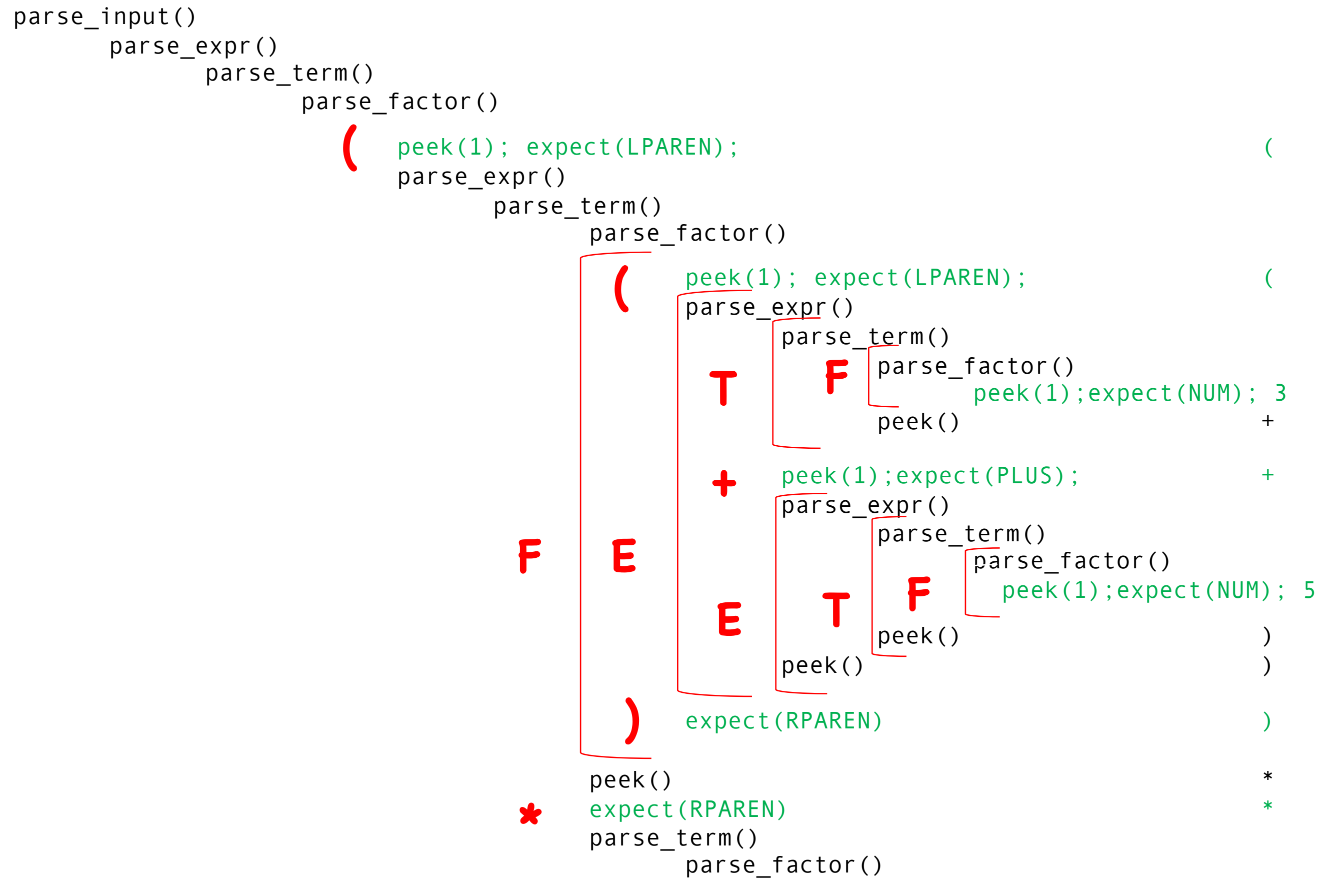
SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
             parse_term()
                    parse_factor()
                       (   peek(1); expect(LPAREN);                                    (
                           parse_expr()
                                 parse_term()
                                        parse_factor()
                                          (    peek(1); expect(LPAREN);                (
                                               parse_expr()
                                                      parse_term()
                                                         F  parse_factor()
                                                 T                 peek(1);expect(NUM); 3
                                                            peek()                     +
                                                 +    peek(1);expect(PLUS);            +
                                                      parse_expr()
                                                            parse_term()
                                    F     E                        parse_factor()
                                                               F     peek(1);expect(NUM); 5
                                                  E      T  peek()                     )
                                                      peek()                           )
                                           )   expect(RPAREN)                          )
                                        peek()                                         *
                                    *   expect(RPAREN)                                 *
                                        parse_term()
```

SAMPLE EXECUTION

((3 + 5) * 8)



```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                            peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                                 peek(1); expect(LPAREN);                 (
                                                 parse_expr()
                                                        parse_term()
                                                               parse_factor()
                                                                      peek(1);expect(NUM); 3
                                                               peek()                     +
                                                        peek(1);expect(PLUS);             +
                                                        parse_expr()
                                                               parse_term()
                                                                      parse_factor()
                                                                        peek(1);expect(NUM); 5
                                                               peek()                     )
                                                        peek()                            )
                                                 expect(RPAREN)                           )
                                          peek()                                          *
                                          expect(RPAREN)                                  *
                                          parse_term()
                                                 parse_factor()
```

SAMPLE EXECUTION

((3+5)*8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                   T       F  parse_factor()
                                                                     peek(1);expect(NUM); 3
                                                              peek()                      +
                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
                                                              parse_term()
                                    F       E                        parse_factor()
                                                   E       T     F     peek(1);expect(NUM); 5
                                                              peek()                      )
                                                       peek()                             )
                                            )   expect(RPAREN)                            )
                                         peek()                                           *
                                    *    expect(RPAREN)                                   *
                                         parse_term()
                                                parse_factor()
                                                       peek(1);expect(NUM)                8
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                   T       F  parse_factor()
                                                                     peek(1);expect(NUM); 3
                                                              peek()                      +
                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
                                                              parse_term()
                                     F      E                        parse_factor()
                                                   E       T     F     peek(1);expect(NUM); 5
                                                              peek()                      )
                                                       peek()                             )
                                            )   expect(RPAREN)                            )
                                         peek()                                           *
                                     *   expect(RPAREN)                                   *
                                         parse_term()
                                              F parse_factor()
                                                       peek(1);expect(NUM)                8
```

SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                            (    peek(1); expect(LPAREN);                 (
                                                 parse_expr()
                                                        parse_term()
                                                           F   parse_factor()
                                                   T                  peek(1);expect(NUM); 3
                                                               peek()                     +
                                                   +    peek(1);expect(PLUS);             +
                                                        parse_expr()
                                                               parse_term()
                                     F      E                         parse_factor()
                                                   E       T     F      peek(1);expect(NUM); 5
                                                               peek()                     )
                                                        peek()                            )
                                            )    expect(RPAREN)                           )
                                          peek()                                          *
                                     *    expect(RPAREN)                                  *
                                          parse_term()
                                              F  parse_factor()
                                                        peek(1);expect(NUM)               8
                                          peek(1)                                         )
```

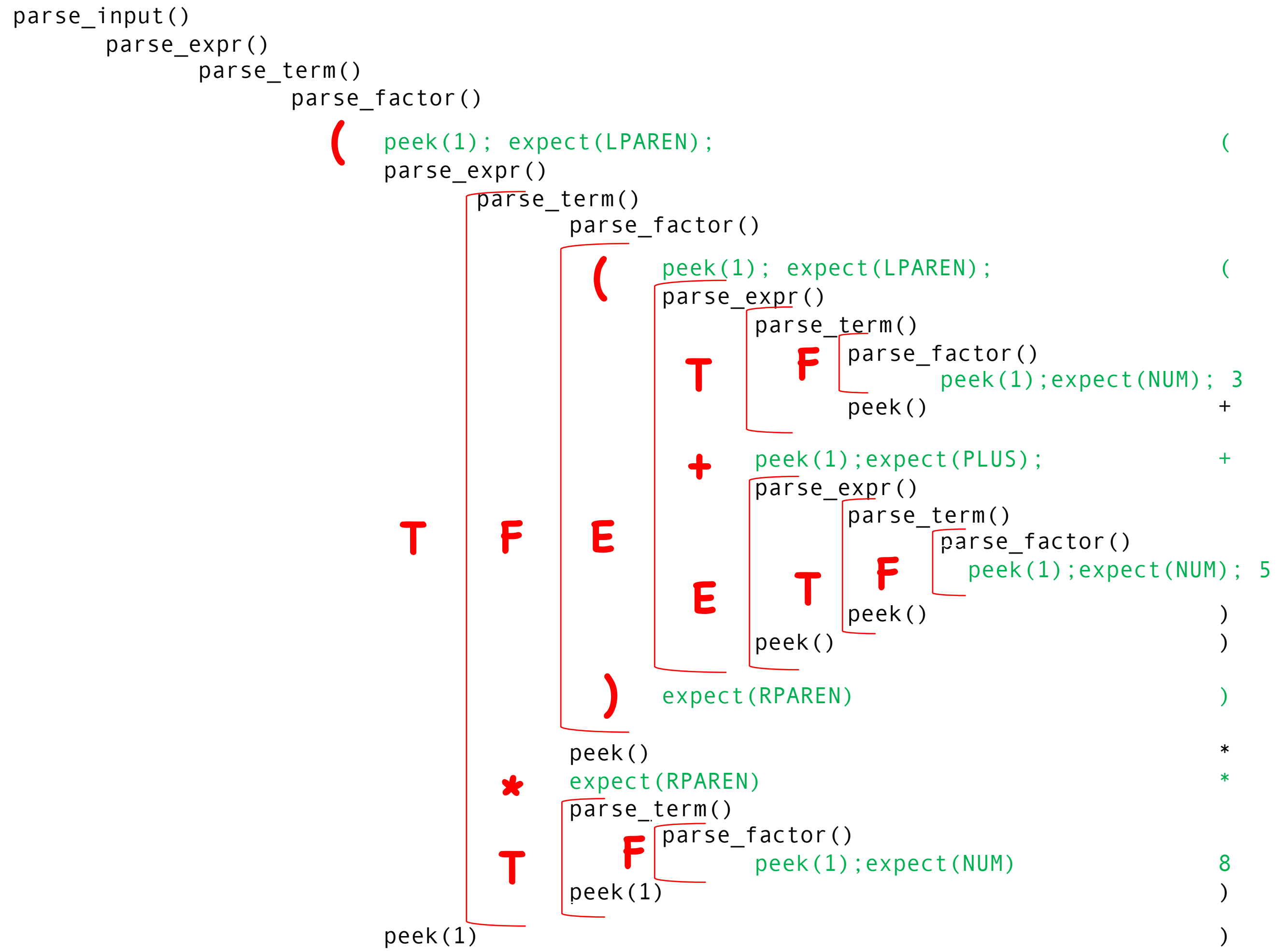
SAMPLE EXECUTION

((3 + 5) * 8)

```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                        (   peek(1); expect(LPAREN);                                      (
                            parse_expr()
                                   parse_term()
                                         parse_factor()
                                            (   peek(1); expect(LPAREN);                  (
                                                parse_expr()
                                                       parse_term()
                                                           F  parse_factor()
                                                   T                 peek(1);expect(NUM);  3
                                                              peek()                      +
                                                   +   peek(1);expect(PLUS);              +
                                                       parse_expr()
                                                              parse_term()
                                     F      E                        parse_factor()
                                                                 F     peek(1);expect(NUM);  5
                                                    E      T  peek()                      )
                                                       peek()                             )
                                            )   expect(RPAREN)                            )
                                         peek()                                           *
                                     *   expect(RPAREN)                                   *
                                         parse_term()
                                              F parse_factor()
                                     T                 peek(1);expect(NUM)                8
                                         peek(1)                                          )
```

SAMPLE EXECUTION

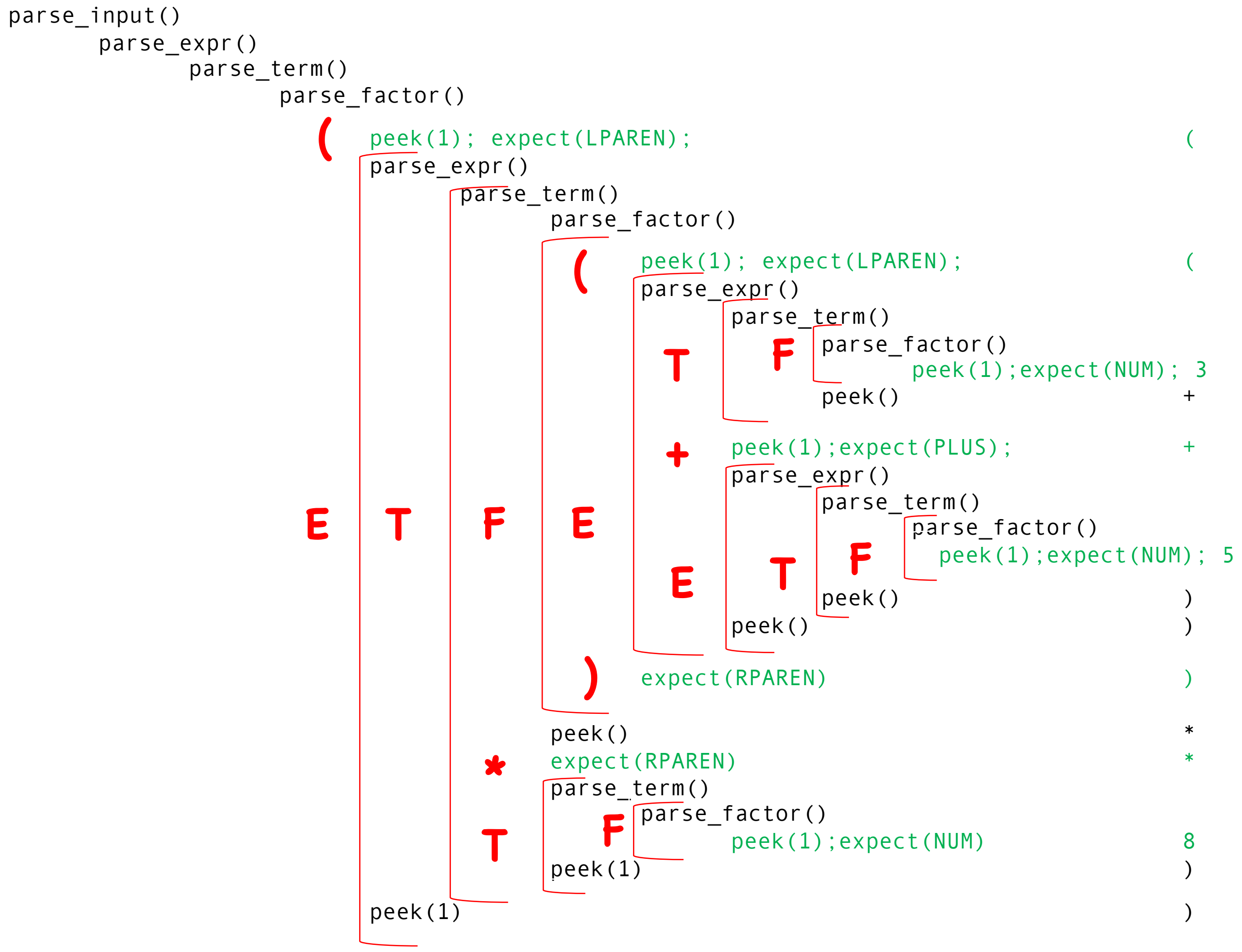
((3 + 5) * 8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                        (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);            (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                      +
                                peek(1);expect(PLUS);           +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                         peek(1);expect(NUM); 5
                                    peek()                      )
                                peek()                          )
                            expect(RPAREN)                      )
                        peek()                                  *
                        expect(RPAREN)                          *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)             8
                        peek(1)                                 )
                peek(1)                                         )
```

SAMPLE EXECUTION

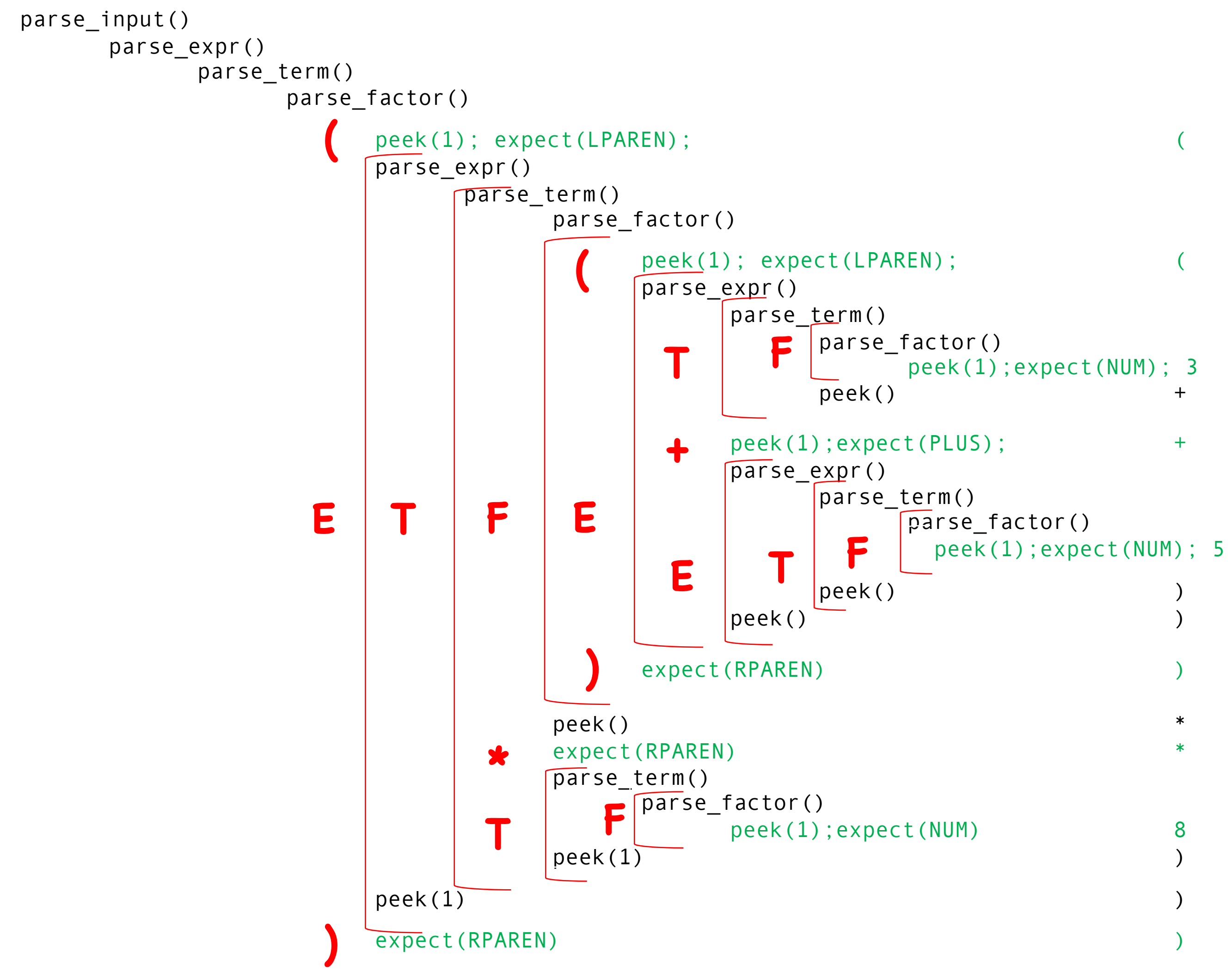
((3 + 5) * 8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);              (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);  (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()            +
                                peek(1);expect(PLUS); +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                         peek(1);expect(NUM); 5
                                    peek()            )
                                peek()                )
                            expect(RPAREN)            )
                        peek()                        *
                        expect(RPAREN)                *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)   8
                        peek(1)                       )
                peek(1)                               )
```

SAMPLE EXECUTION

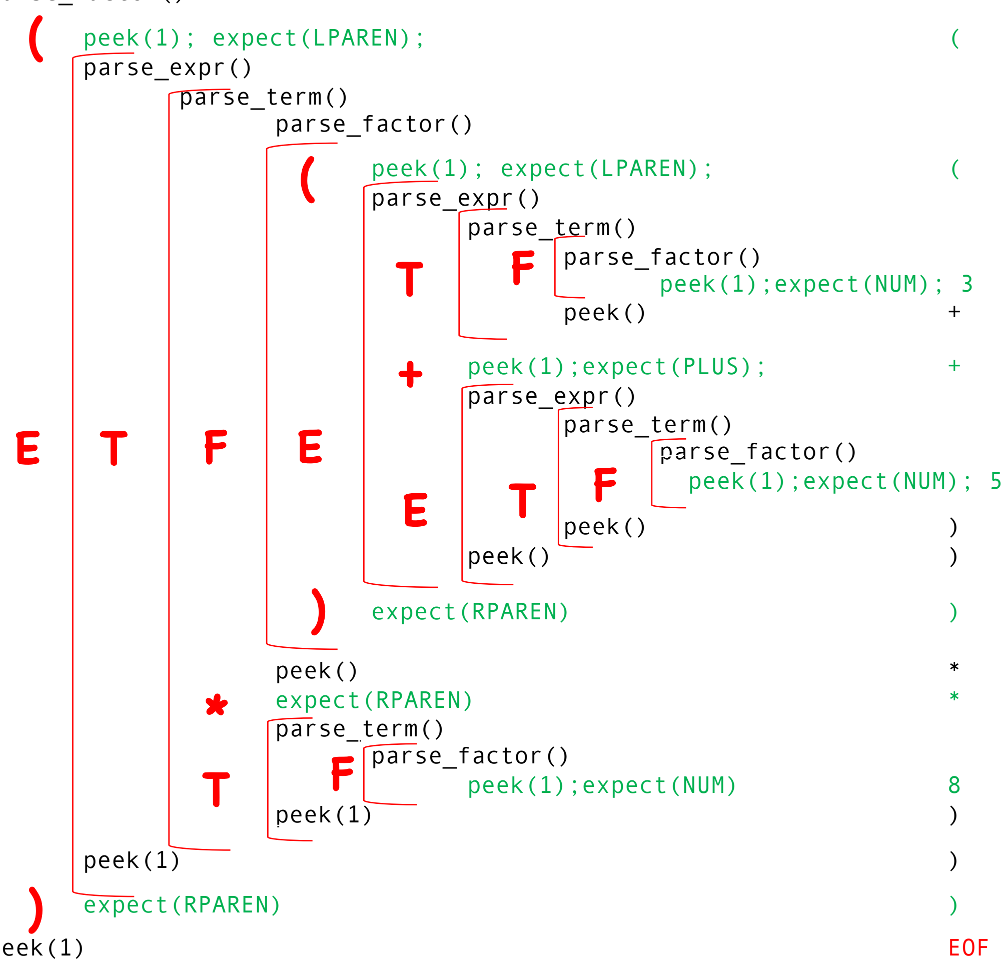
((3 + 5) * 8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                              (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);                  (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM);  3
                                    peek()                            +
                                peek(1);expect(PLUS);                 +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                         peek(1);expect(NUM);  5
                                    peek()                            )
                                peek()                                )
                            expect(RPAREN)                            )
                        peek()                                        *
                        expect(RPAREN)                                *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)                   8
                        peek(1)                                       )
                peek(1)                                               )
                expect(RPAREN)                                        )
```

SAMPLE EXECUTION

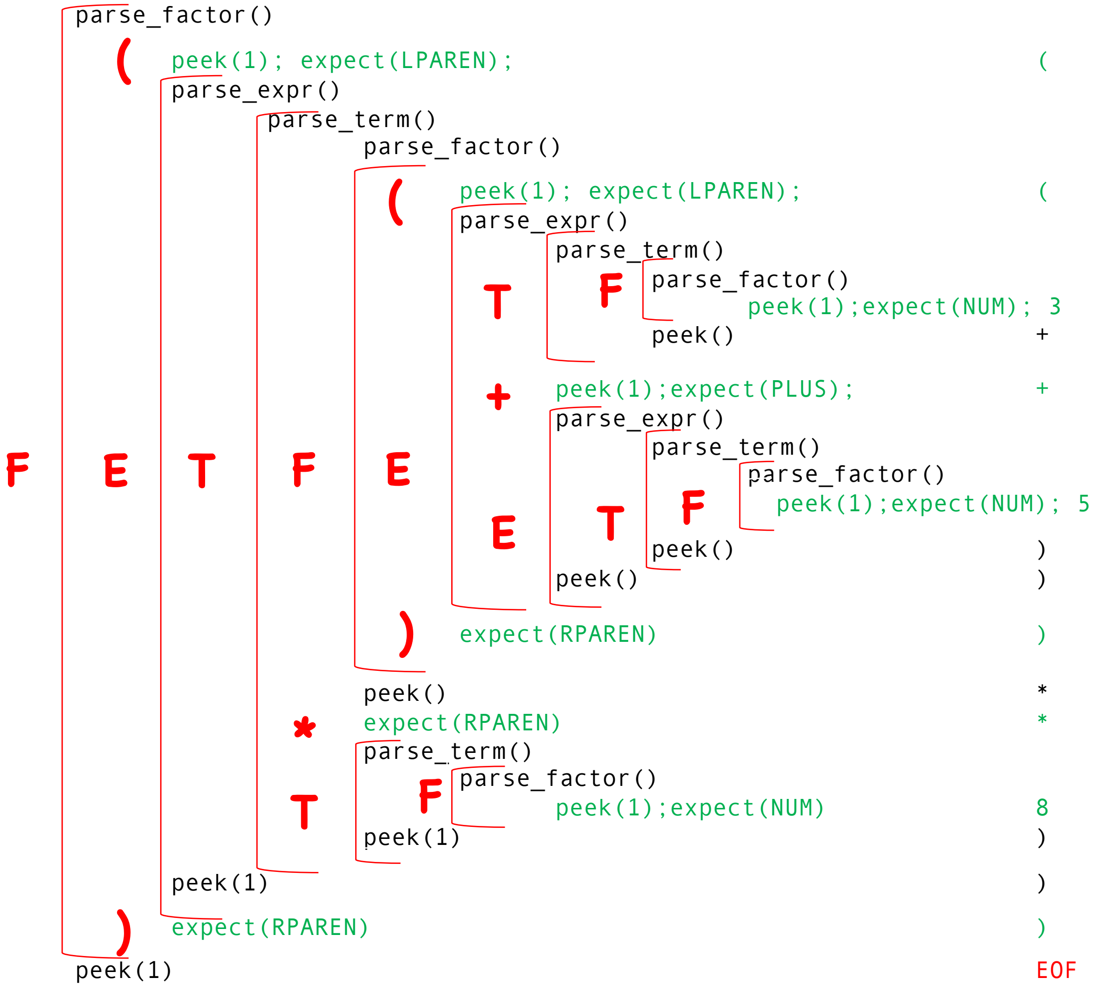
((3+5)*8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                                      (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);                          (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                                    +
                                peek(1);expect(PLUS);                         +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                          peek(1);expect(NUM); 5
                                    peek()                                    )
                                peek()                                        )
                            expect(RPAREN)                                    )
                        peek()                                                *
                        expect(RPAREN)                                        *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)                           8
                        peek(1)                                               )
                peek(1)                                                       )
                expect(RPAREN)                                                )
            peek(1)                                                           EOF
```

SAMPLE EXECUTION

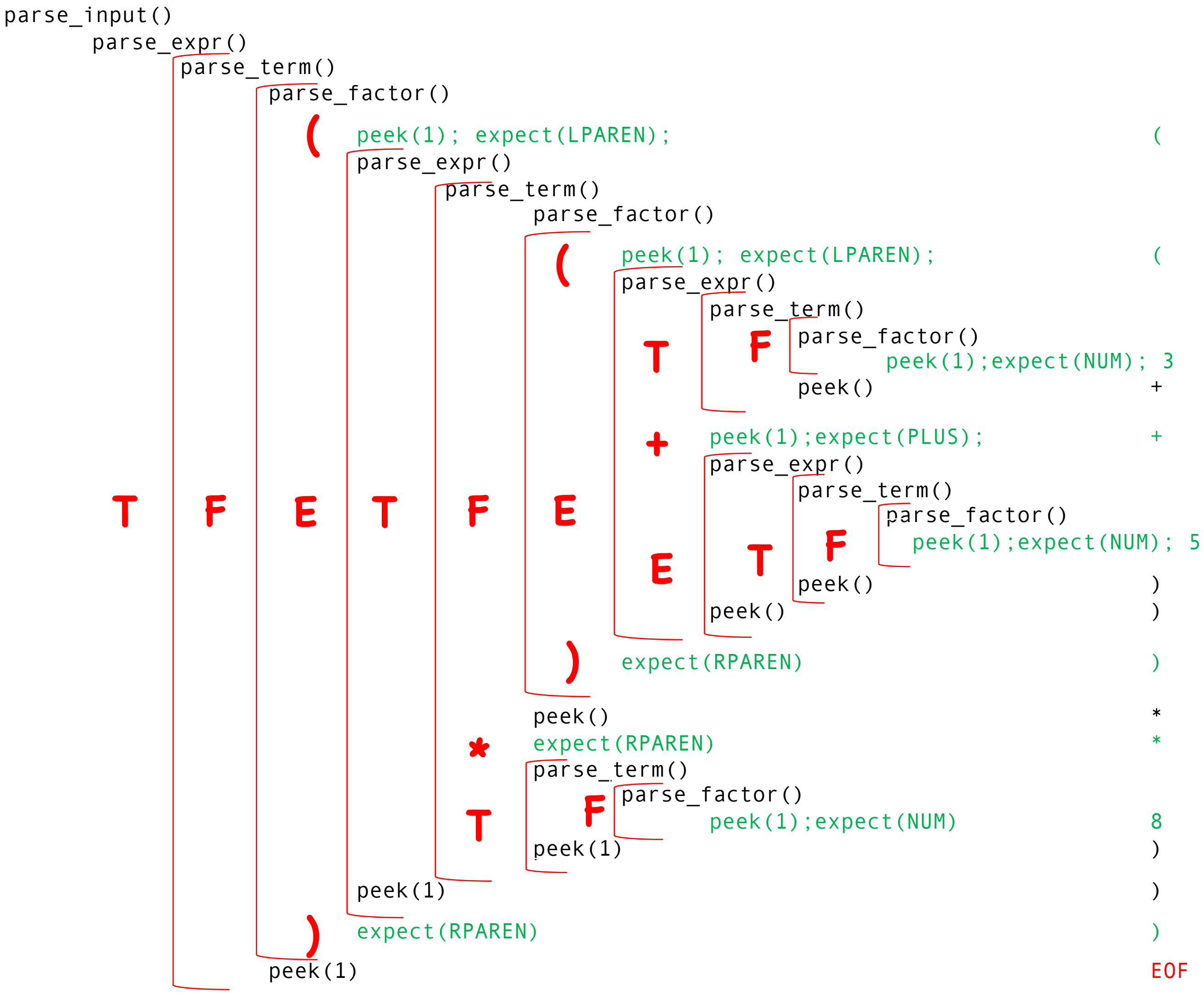
((3 + 5) * 8)



```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                            peek(1); expect(LPAREN);                                  (
                            parse_expr()
                                   parse_term()
                                          parse_factor()
                                                 peek(1); expect(LPAREN);             (
                                                 parse_expr()
                                                        parse_term()
                                                               parse_factor()
                                                                      peek(1);expect(NUM); 3
                                                               peek()                 +
                                                        peek(1);expect(PLUS);         +
                                                        parse_expr()
                                                               parse_term()
                                                                      parse_factor()
                                                                        peek(1);expect(NUM); 5
                                                               peek()                 )
                                                        peek()                        )
                                                 expect(RPAREN)                       )
                                          peek()                                      *
                                          expect(RPAREN)                              *
                                          parse_term()
                                                 parse_factor()
                                                        peek(1);expect(NUM)           8
                                          peek(1)                                     )
                            peek(1)                                                   )
                            expect(RPAREN)                                            )
                     peek(1)                                                          EOF
```

SAMPLE EXECUTION

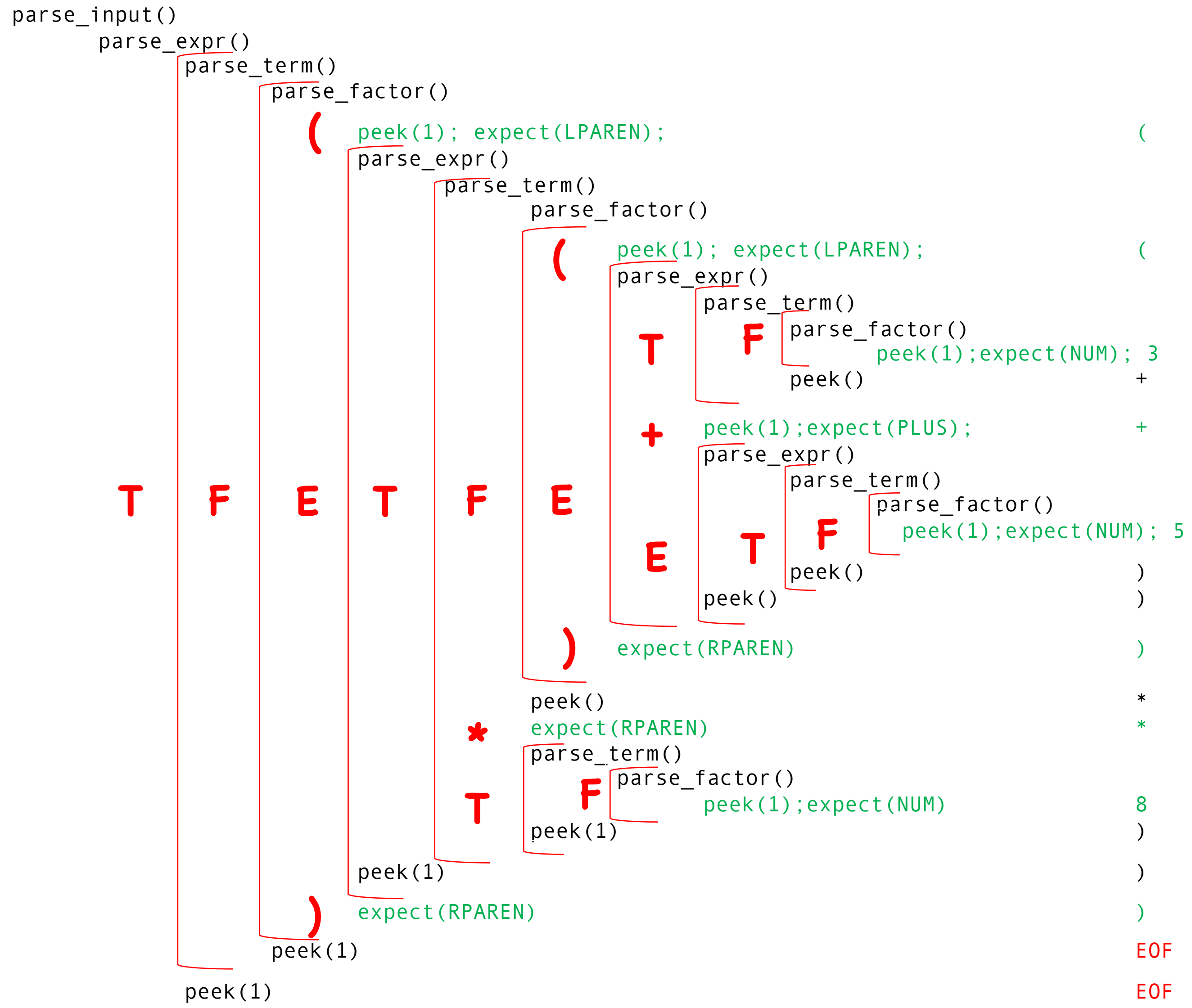
((3+5)*8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                                  (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);                      (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                                +
                                peek(1);expect(PLUS);                     +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                         peek(1);expect(NUM); 5
                                    peek()                                )
                                peek()                                    )
                            expect(RPAREN)                                )
                        peek()                                            *
                        expect(RPAREN)                                    *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)                       8
                        peek(1)                                           )
                peek(1)                                                   )
                expect(RPAREN)                                            )
            peek(1)                                                       EOF
```

SAMPLE EXECUTION

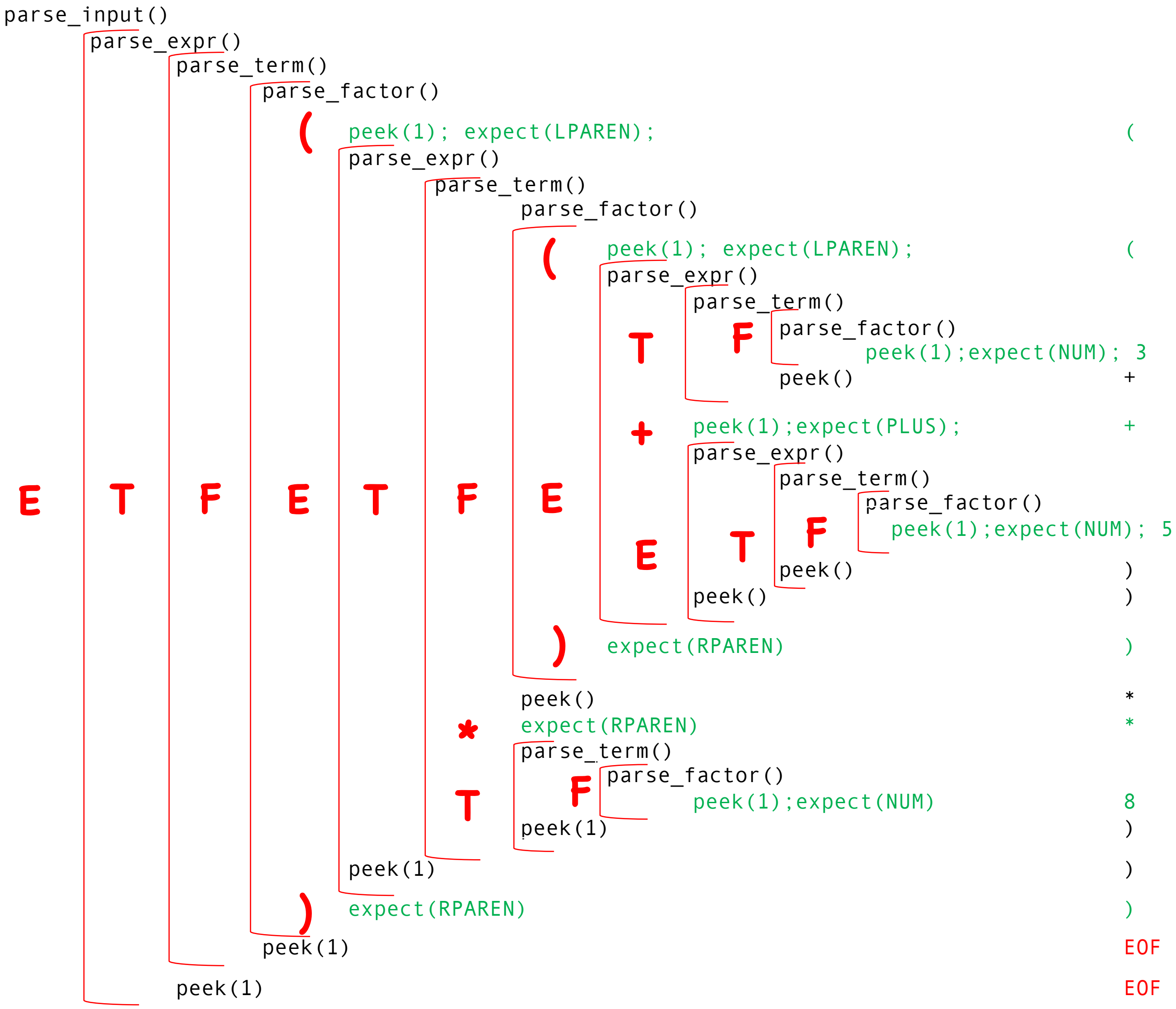
((3 + 5) * 8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                         (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);             (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                       +
                                peek(1);expect(PLUS);            +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                         peek(1);expect(NUM); 5
                                    peek()                       )
                                peek()                           )
                            expect(RPAREN)                       )
                        peek()                                   *
                        expect(RPAREN)                           *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)              8
                        peek(1)                                  )
                peek(1)                                          )
                expect(RPAREN)                                   )
            peek(1)                                              EOF
        peek(1)                                                  EOF
```

SAMPLE EXECUTION

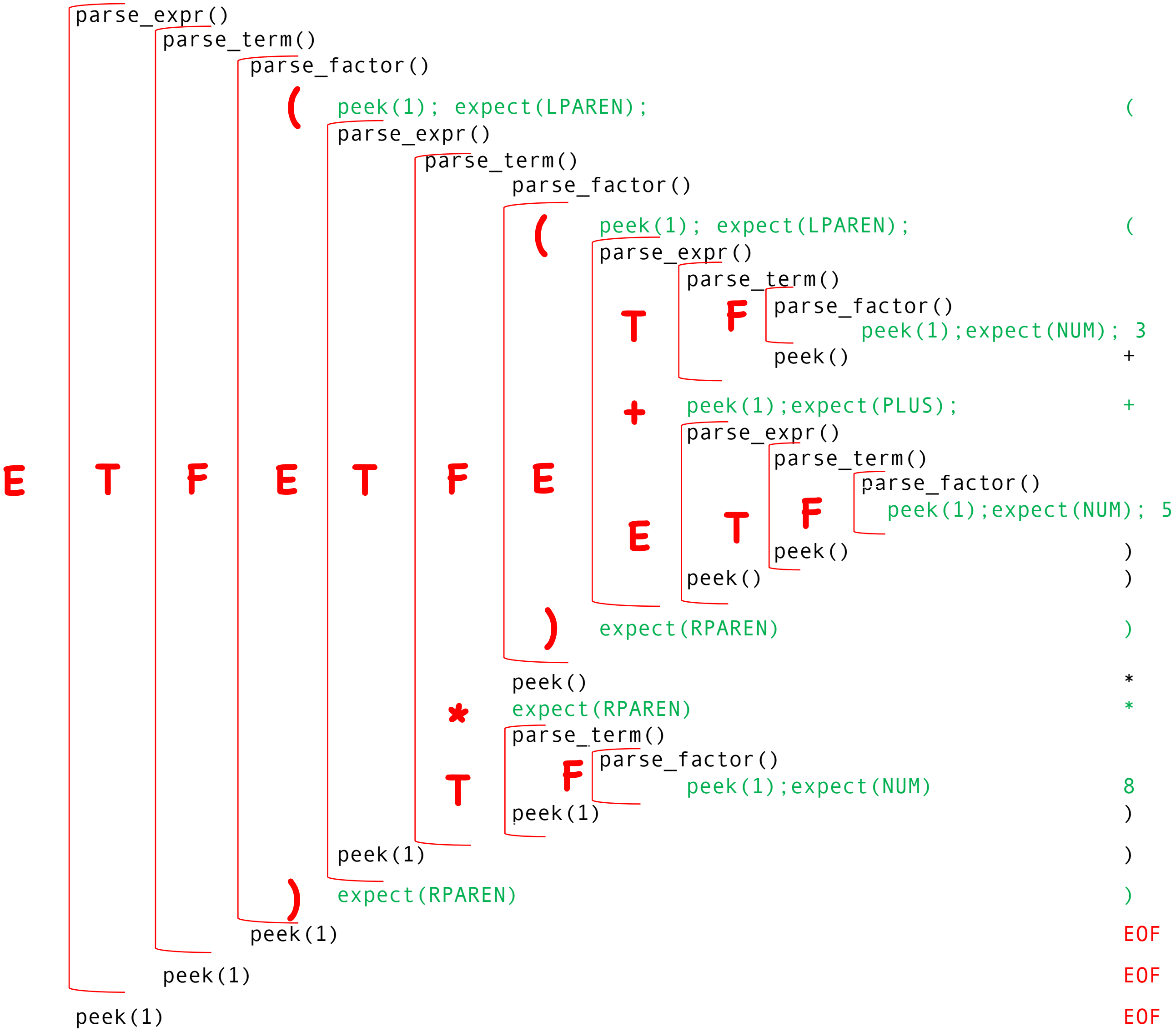
((3 + 5) * 8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                          (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);              (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                        +
                                peek(1);expect(PLUS);             +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                         peek(1);expect(NUM); 5
                                    peek()                        )
                                peek()                            )
                            expect(RPAREN)                        )
                        peek()                                    *
                        expect(RPAREN)                            *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)               8
                        peek(1)                                   )
                peek(1)                                           )
                expect(RPAREN)                                    )
            peek(1)                                               EOF
        peek(1)                                                   EOF
```

SAMPLE EXECUTION

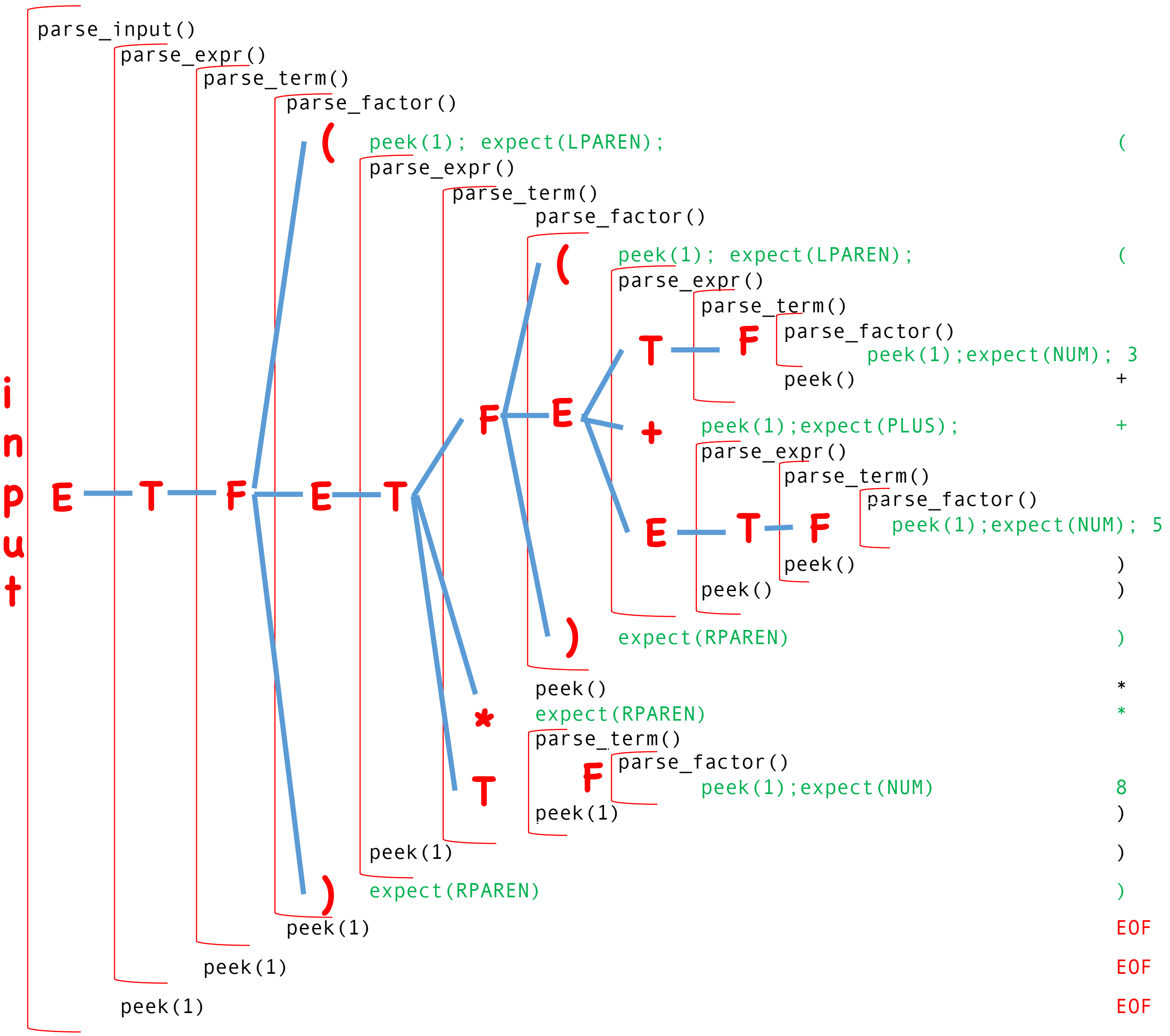
((3 + 5) * 8)



```
parse_input()
    parse_expr()
        parse_term()
            parse_factor()
                peek(1); expect(LPAREN);                                        (
                parse_expr()
                    parse_term()
                        parse_factor()
                            peek(1); expect(LPAREN);                            (
                            parse_expr()
                                parse_term()
                                    parse_factor()
                                        peek(1);expect(NUM); 3
                                    peek()                                      +
                                peek(1);expect(PLUS);                           +
                                parse_expr()
                                    parse_term()
                                        parse_factor()
                                          peek(1);expect(NUM); 5
                                    peek()                                      )
                                peek()                                          )
                            expect(RPAREN)                                      )
                        peek()                                                  *
                        expect(RPAREN)                                          *
                        parse_term()
                            parse_factor()
                                peek(1);expect(NUM)                             8
                        peek(1)                                                 )
                peek(1)                                                         )
                expect(RPAREN)                                                  )
            peek(1)                                                             EOF
        peek(1)                                                                 EOF
    peek(1)                                                                     EOF
```

SAMPLE EXECUTION

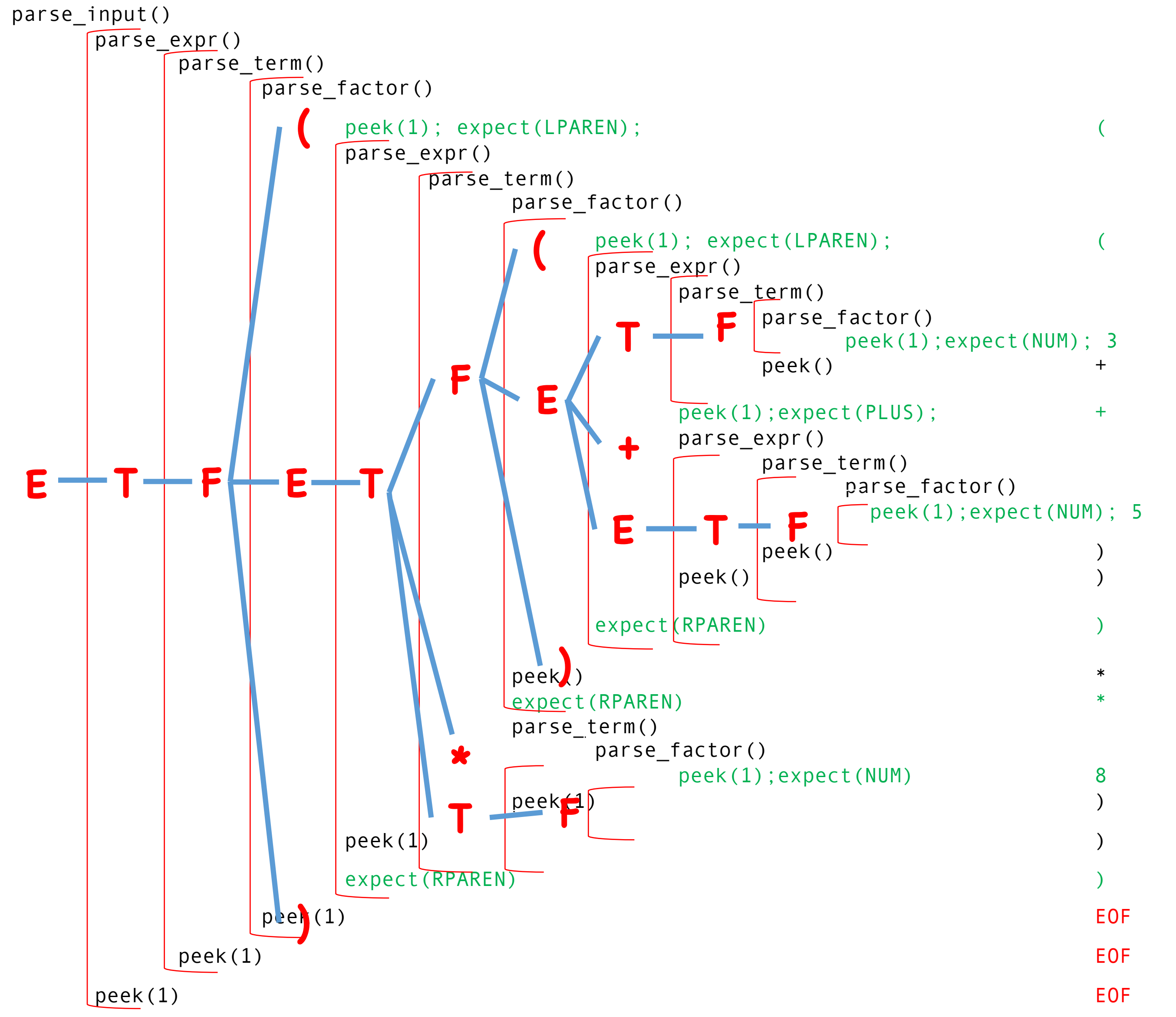
((3 + 5) * 8)



```
parse_input()
     parse_expr()
          parse_term()
               parse_factor()
                    peek(1); expect(LPAREN);                                (
                    parse_expr()
                         parse_term()
                              parse_factor()
                                   peek(1); expect(LPAREN);                 (
                                   parse_expr()
                                        parse_term()
                                             parse_factor()
                                                  peek(1);expect(NUM); 3
                                             peek()                         +
                                        peek(1);expect(PLUS);               +
                                        parse_expr()
                                             parse_term()
                                                  parse_factor()
                                                    peek(1);expect(NUM); 5
                                             peek()                         )
                                        peek()                              )
                                   expect(RPAREN)                           )
                              peek()                                        *
                              expect(RPAREN)                                *
                              parse_term()
                                   parse_factor()
                                        peek(1);expect(NUM)                 8
                              peek(1)                                       )
                    peek(1)                                                 )
                    expect(RPAREN)                                          )
               peek(1)                                                      EOF
          peek(1)                                                           EOF
     peek(1)                                                                EOF
```

SAMPLE EXECUTION

((3 + 5) * 8)



```
parse_input()
  parse_expr()
    parse_term()
      parse_factor()
        peek(1); expect(LPAREN);                       (
        parse_expr()
          parse_term()
            parse_factor()
              peek(1); expect(LPAREN);                 (
              parse_expr()
                parse_term()
                  parse_factor()
                      peek(1);expect(NUM); 3
                    peek()                             +
                peek(1);expect(PLUS);                  +
                parse_expr()
                  parse_term()
                    parse_factor()
                      peek(1);expect(NUM); 5
                    peek()                             )
                  peek()                               )
              expect(RPAREN)                           )
            peek()                                     *
            expect(RPAREN)                             *
            parse_term()
              parse_factor()
                peek(1);expect(NUM)                    8
            peek(1)                                    )
        peek(1)                                        )
        expect(RPAREN)                                 )
      peek(1)                                          EOF
    peek(1)                                            EOF
  peek(1)                                              EOF
```