Introduction to Syntax Analysis

Rida A. Bazzi

Syntax Analysis in the English Language

Before introducing syntax analysis for programming languages, we start with a motivating example from the English language. The example will illustrate various elements of syntax analysis in the English language that are also relevant for programming. After going over the English language example, we will explore syntax for programming languages.

Example text 1: The cat drank the milk.

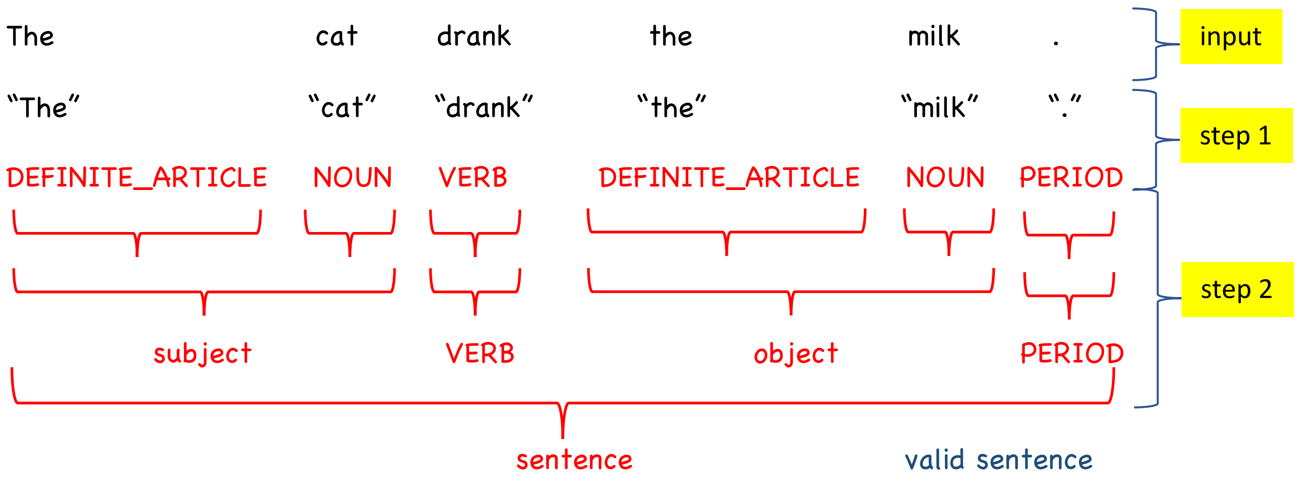
How do we determine if this text is grammatically correct?

1. First, we need to determine if the text is made up of English words and symbols. So, we start by identifying 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 (punctuation symbol) as a word and the category is PERIOD and I don't consider a larger category of PUNCTUATION_SYMBOLS. When we analyze an input, we should be given the various categories of words ahead of time, so that we can properly break the input into a sequence of words of the right categories (or kinds). For the English language, that is available in the dictionary.

2. The second step consists of taking the sequence of word categories and determining if the sequence is syntactically correct.



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
input
step 1
step 2

The first step which consists of breaking the text into words and determining the category or kind of each word is called **Lexical analysis**.

The second step which consists of grouping the "categories" (or kinds) together to determine if they form a grammatically correct sequence is called **parsing**.

Step 1 and Step 2 together are called **syntax analysis** (they might also be referred to as parsing, but in this course, we will restrict the term parsing to step 2 and lexical analysis to step 1).

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

In the example above, parsing considered the sequence of categories to determine if the input is syntactically correct. Parsing does not look at the specific words. The next example illustrates this point.

Example text 2: The milk drank the cat.

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

DEFINITE_ARTICLE NOUN VERB DEFINITE_ARTICLE NOUN PERIOD

As far as parsing is concerned, this is the exact same format (syntax) as the previous sentence.

What matters for parsing 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

For a natural language like the English language, assuming language is static, (which it is not) we can list all words and all categories in one large unabridged dictionary.

The situation is different for programming languages as we will see next.

Syntax of Programming Languages

The elements of syntax for of programming languages are similar to those of natural languages.

For programming languages, we need to have a well-defined alphabet and a well-defined list of categories. The categories of words in programming languages are called tokens and the words are called lexemes (we also call them tokens for convenience). We start by defining alphabet and strings.

Alphabet An alphabet is a finite set of symbols.

Example 1 { a , b , & , 1 , ! }
Example 2 The alphabet of the C language contains { a , b , … , z , A , B , … Z , 0 , 1 , … 9 , < , … }

The alphabets of widely used programming languages are significantly larger than what you might expect. I uploaded on canvas the specifications of Ocaml, C99 and the C# languages. I encourage you to go through them looking for their alphabets (the set of characters that can appear in a valid program).

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"
Note: Sometimes I enclose single characters in single quotation marks to indicate that they are alphabet characters and not meta characters used in the description. For example, I write { ',' , ';' } for a set containing two symbols ',' (comma) and ';' (semicolon). The middle red comma is used a separator in the description of the set and is not part of the symbols of the set.

Token A token is a set of strings. We can think of a token as the name of a set of strings.
Every programming language has a definition of the list of tokens of the language. The following are some examples adopted from Ocaml.

Example 1 integer-literal = [-] (0…9) { 0…9 | _ }

we read this as:

1. optional - sign ([-] indicate that - can appear at the beginning or can be omitted. In fact, in this notation anything that appears within square brackets is optional) followed by
2. a digit (0 through 9 denoted with 0…9) followed by
3. a sequence of zero or more digits and/or underscore ({ 0…9 | _ } denotes a repetition of 0 or more elements of 0…9 | _ . | is used to denote or and we have already seen that 0…9 is used to denote any digit. So, 0…9 | _ denotes any digit or underscore and { 0…9 | _ } denotes a sequence of zero or more digits or underscores.

The underscores in integer-literal are ignored as far as the value is concerned but are provided for clarity. They allow us to write 1_000_000 for example.

Note: The full definition of integer literals in Ocaml also define the form of hexadecimal, octal and binary literals. I did not include them for simplicity.

Example 2 identifier = (letter | _) { letter | 0…9 | _ | ' }

We read this as letter or underscore followed by an optional sequence of letters, digits, underscore or single quote mark ('). In the definition letter = A…Z|a…z (anything that is part of A through Z or a through z).

Example 3 float-literal = [-] (0…9) { 0…9 | _ } [. { 0…9 | _ }] [(e | E) [+ | -] (0…9) { 0…9 | _ }]

Again, this is not the full definition as it appears in Ocaml.

Note: In what follows, for conciseness, I will use NUM, ID and DECIMAL to denote integer-literal, identifier and float-literal respectively.

Elements of Syntax

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

Example 1	"123" is a lexeme of NUM "123a" is not a lexeme of NUM "-123_123" is a lexeme of NUM
Example 2	"1.00" is a lexeme of DECIMAL "1." is a lexeme of DECIMAL "0.00100" is a lexeme of DECIMAL "0.00" is a lexeme of DECIMAL "0.1c" is not a lexeme of DECIMAL
Example 3	"a123" is a lexeme of ID "123a" is not a lexeme of ID

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

As we stated earlier, 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

As we pointed out earlier, 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 also depend on the specific word that is used. The example

The milk drank the cat.

is syntactically correct, but it does not make sense (it is not semantically correct). The sentence does not make sense because "milk" is not a semantically meaningful subject for the verb "drank".

Syntax vs. Semantics in programming languages

The situation is similar for programming languages. Consider the following two program fragment in the C language

```
int x;                          int x;
float y;        1                int y;          2

x = y;                           x = y;
```

The two fragments are syntactically correct. From a syntax point of view, they look exactly the same:

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

$$\begin{tikzcd}
\text{ID} & \text{EQUAL} & \text{ID} & \text{SEMICOLON} \\
x \arrow[u] & = \arrow[u] & y \arrow[u] & ; \arrow[u]
\end{tikzcd}$$

Where ID is the category that contains all valid identifiers (valid variable names as well as type names). Even though the two fragments are syntactically correct, only 2 is valid semantically because C does not allow a float value to be assigned to an integer variable.

The syntax of fragment 1 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.

For now, we will concentrate on syntax and not semantics.

getToken() function

In discussing the examples above (the cat and the milk or the program fragments), we assumed that we could take a list of categories (or tokens) and an input string and break that string into a sequence of tokens/lexemes. When we write a parser, we need to have a function that can provide this functionality. In this class, we use a function that I call getToken() to help us with this task. For each project, you will be provided with a getToken() function.

A getToken() function works for a specific token list. We will assume that input is always from standard input (the keyboard). When getToken() is called, it will *consult* the list of tokens and returns the next token in the input (this will become clearer as we describe it in more details).

getToken()   takes input from standard input

returns a struct that has three 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 corresponds to the token_type identified by getToken()[1]
- line_no   this is the line on which the token appears. This can be useful for error messages

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

When we call getToken(), a token and its lexeme are identified. Next time getToken() is called, we start after after the previously identified lexeme. Every call to getToken() effectively *consumes* part of the input which corresponds to the identified lexeme.

How getToken() returns a token if there are multiple possibilities: A given call to getToken() will try every token in the list to find a lexeme that matches that token. Th search for a lexeme starts from the next character of the input that was not consumed by a previous call to getToken(). In general, it is possible that there are multiple matches. getToken() returns the token with the longest matching lexeme. If there is more than one token with longest matching lexeme, then the token that appears first in the list is returned. The rules for getToken() are also repeated at the bottom of the next page.

Separators and Space Characters.

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.

In general, we are going to assume in the examples that white space characters are ignored and only serve as separators, but that need not be the case always. For example, in Python, the indentation level is determined by white space.

[1] If the token (category) has only one possible lexeme, the implementation I will provide will have the lexeme field equal to the empty string. The reason is that, for such tokens, all the information you need about the token is already in the token_type field.

getToken() function

Example 1 DOT
DEC which I define to be NUM.NUM (this is different from DECIMAL that we saw earlier

input: 1.1..1

If we call getToken() repeatedly, we get the following (note how every call tries to match all tokens):

1. getToken() ⟶ { DEC, "1.1" }

		match	remaining input
			1.1..1
1.	NUM	"1"	1.1..1
2.	DOT	no match	
3.	DEC	"1.1"	1.1..1

There are two tokens that match a prefix of the remaining input. We return the longer match

2. getToken() ⟶ { DOT, "" }

			..1
1.	NUM	no match	
2.	DOT	"."	..1
3.	DEC	no match	

There is only one token that matches a prefix of the remaining input, and we return it

3. getToken() ⟶ { DOT, "" }

			.1
1.	NUM	no match	
2.	DOT	"."	.1
3.	DEC	no match	

There is only one token that matches a prefix of the remaining input, and we return it

1. getToken() ⟶ { NUM, "1" }

			1
1.	NUM	"1"	1
2.	DOT	no match	
3.	DEC	no match	

There is only one token that matches a prefix of the remaining input, and we return it

5. getToken() ⟶ (EOF , "") no input remaining

Since there is no input remaining, we return an indication that there is no more input. EOF (end of file) indicates that the end of input is reached and that there are no more tokens. The EOF itself is not a token (or a category). It is just an indication that the call of getToken() did not find more input.

5. getToken() ⟶ (EOF , "") no remaining input

Since there is no input remaining, we return an indication that there is no more input. EOF (end of file) indicates that the end of input is reached and that there are no more tokens.

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 that matches a prefix of remainder of the input is returned (this statements needs to be read carefully!)
- **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

getToken() function

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

Example 2	token list:	ID EQUAL SEMICOLON
	input:	x = & y;

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

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

Each line above corresponds to one call of getToken(). The struct that is returned by getToken() is represented as a pair, with the first element being the token_type and the second element being the lexeme).

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 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. In the third call to getToken(), we start by skipping space and we start at the & which is not a valid symbol for any of the three tokens on our list. So, the call returns ERROR to indicate that getToken() failed because there is still input but no prefix of the remaining input can match a token. Note how is this is different from EOF which indicates that no token can be returned because there is no more input.
4. Since we encountered an error on the third call, we need to specify where the next call should resume from. We are going to go with the convention that if an error is encountered, getToken() will advance one non-space character, so the next call to getToken() will start after the character that is skipped by the call that failed. In our case, the next call will start after the & after the &, which is a space character. The space is ignored, and the next token returned is ID whose lexeme is "y"
5. The next call to getToken() returns SEMICOLON. Again, in the implementation, I have the lexeme for SEMICOLON equal to the empty string because all the information about the lexeme is captured by the token_type SEMICOLON.
6. The last two calls (5 and 6) return EOF because there is no more input (see example on previous page for explanation).

Example 3 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. So, in the following example,

token list	IF
	ID
	NUM
input	if1 ifif if 1

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

To understand why this the case, let us consider the first four calls. The first call starts at the beginning of the input. The possibilities for the first call are

IF	if1 ifif if 1
ID	if1 ifif if 1
NUM	X

Since if1 is longer than if, the first call to getToken() returns (ID, "if1"). Note that getToken() does not keep on going past the space, because the space cannot be part of a token, but it is a separator.

For the second call, the remaining input is

 ifif if 1

The second call starts at the space after before ifif. Since the space character is ignored, we skip ahead until the i at the beginning of ifif. Then, the second call to getToken() will try all three possible tokens (in the list):

IF	ifif if 1
ID	ifif if 1
NUM	X

and returns the longest matching prefix which is (ID, "ifif").

For the third call, the remaining input is

 if 1

The third call at the space before if. Since the space character is ignored, we skip ahead until the i at the beginning of if. Then, the third call to getToken() will try all possible tokens:

IF	if 1
ID	if 1
NUM	X

So, the third call to getToken() return (IF, "") because IF is listed before ID in the list of tokens.

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:

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

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. After each call, if index is not larger than the number of tokens, then index points to the next unconsumed token With this representation, the functions getToken() and peek() can be defined as follows.

getToken() this function will simply return 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 howfar > 0
                                if index+howfar -1 > number of tokens
                                        return EOF
                                else
                                        return token_array[index+howfar-1]
                         else
```

Remember that index points to the next token that has not been yet consumed, 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

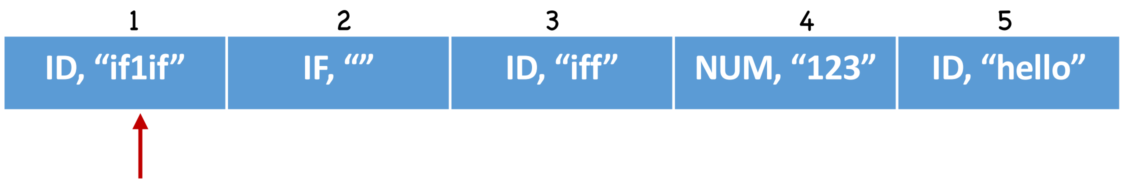
IF	= { "if"}	
ID		// previously defined
NUM		// previously defined

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 stops and returns the longest matching prefix. If the next character before a call to getToken() is a space character, the next call to getToken() will first skip the whitespace characters before attempting to identify the next token.

Input

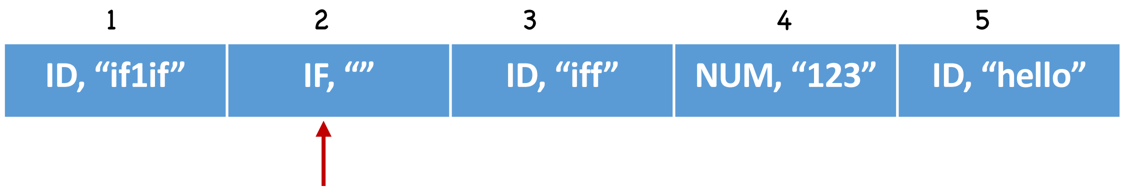
```
if1if  if  iff  123hello
```

The sequence of tokens for this example is



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



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

Now that we have covered the getToken() and peek() functions, we are ready to start with parsing which consists of determining if the sequence of tokens (categories) is valid according to a grammar. I will start by introducing the parsing concepts through an example grammar for simple expressions.

Example: Simple Expressions grammar

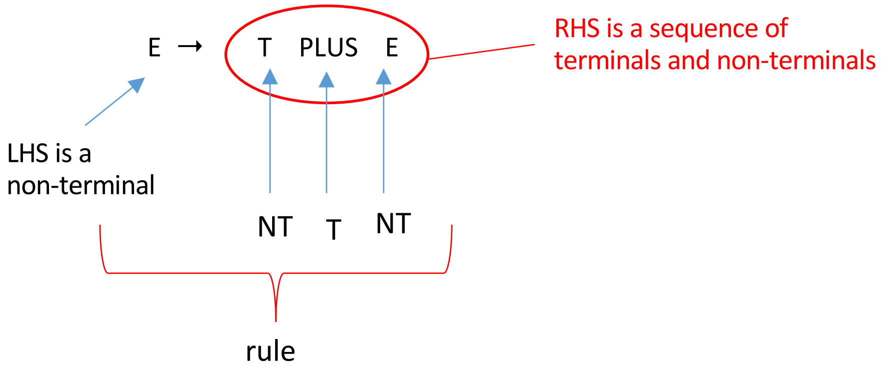
```
expr        → term
expr        → term PLUS expr
term        → factor
term        → factor MULT term
factor      → ID
factor      → NUM
factor      → LPAREN expr RPAREN
```

Instead of being overly verbose, I am going to rewrite the grammar as follows

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

Note. The concepts of expr, term and factor are typically introduced in Algebra 1 when you learn about operators and their precedence!

Grammar Format. 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:



For the grammar above, the **terminals** are the tokens PLUS, MULT, LPAREN, RPAREN, NUM and ID. The **non-terminals** are E, T and F.

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 The parsers we will write are called recursive descent parsers. These parsers are used in practice. For example, the gcc compiler is a recursive descent parser. The parsers we will write in this class have the following properties:

- **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**
    - If parse_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. This behavior is appropriate for grammars for which no backtracking is needed, but if backtracking is needed, the function would need to return with an indication that it could not correctly parse.
    - A parse function does not consume any part of the input that is not *part* of the non-terminal it is parsing.

- **Recursive:** 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)

- **Descent:** Parsing is done from top to bottom. To parse a program, we start by calling parse_program() which calls other parse functions to parse components of the program which in turn call other functions to parse subcomponents and so on.

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 determine that there is a syntax_error(). We assume that the syntax_error() function will simply print 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 is more than one valid possibility 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 execution

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

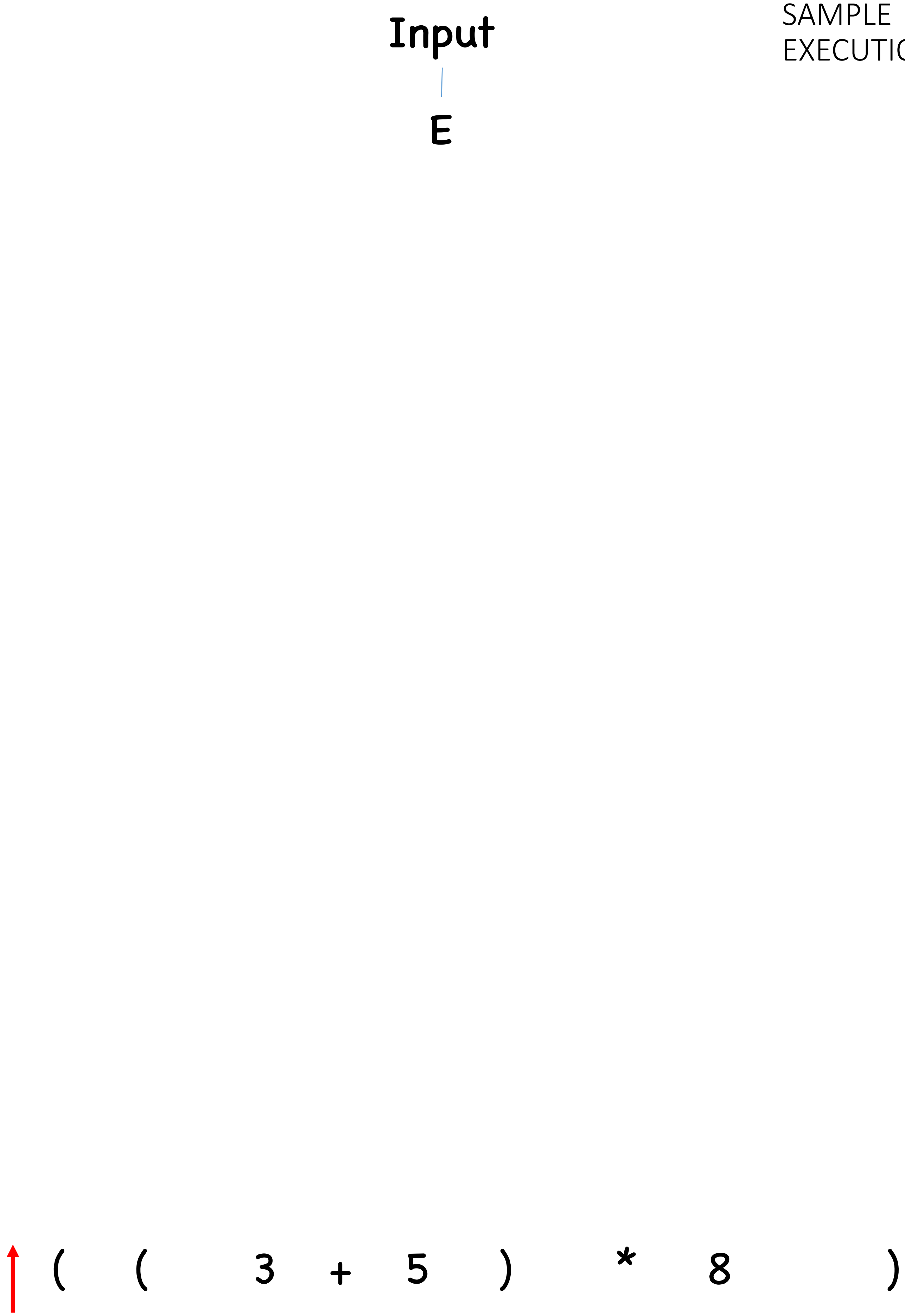
2. An equivalent illustration using call sequence

SAMPLE
EXECUTION

Input

((3 + 5) * 8)

SAMPLE EXECUTION



SAMPLE
EXECUTION

$$\begin{tikzcd}
\text{Input} \arrow[d, dash] \\
E \arrow[d, dash] \\
T
\end{tikzcd}$$

call parse_T()

$\uparrow$ ((3 + 5) * 8)

SAMPLE
EXECUTION

$$\begin{tikzcd}
\text{Input} \arrow[d, dash] \\
E \arrow[d, dash] \\
T \arrow[d, dash] \\
F
\end{tikzcd}$$

$\uparrow$ ((3 + 5) * 8)

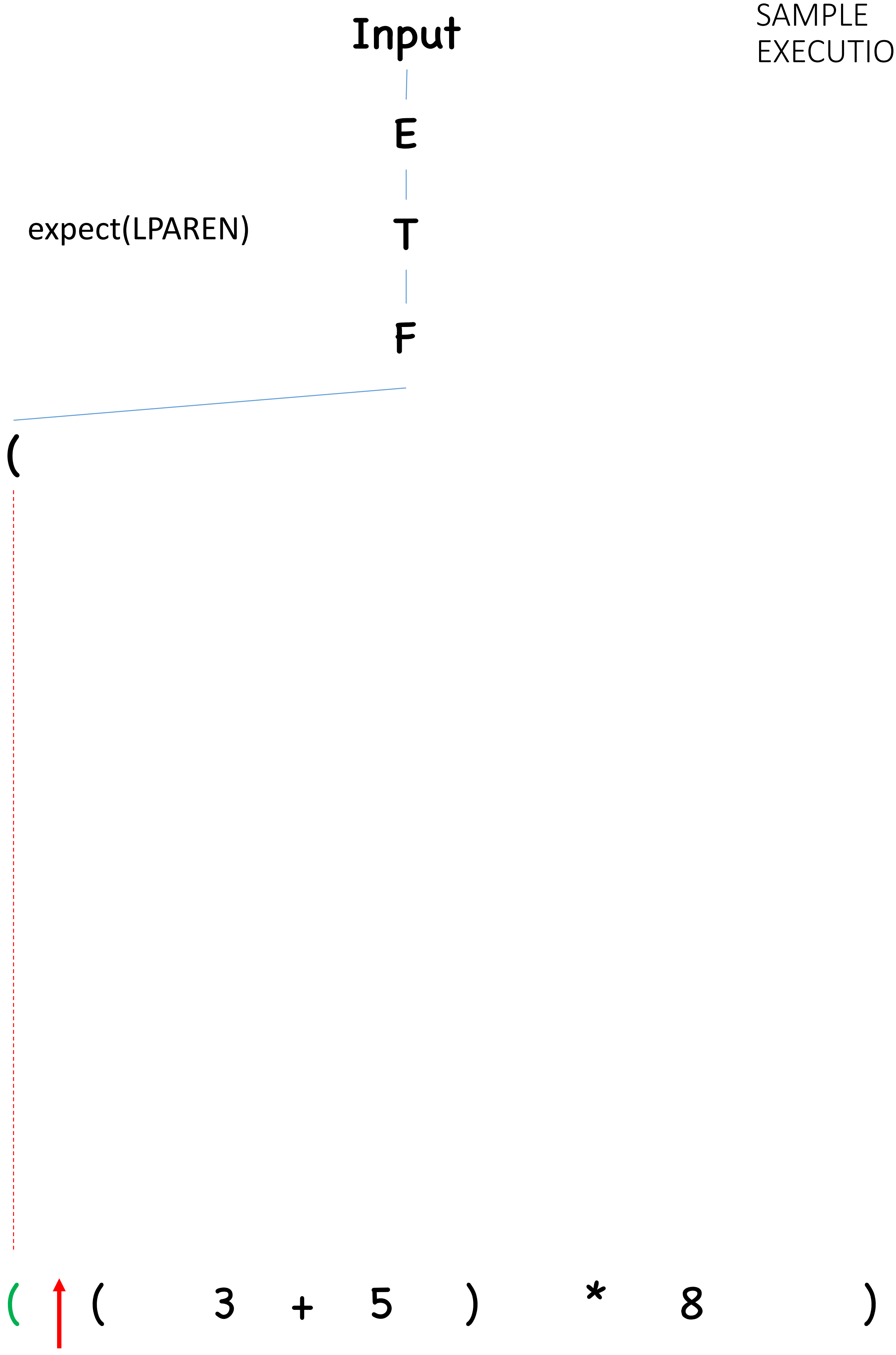
SAMPLE
EXECUTION

$$\begin{tikzcd}
\text{Input} \arrow[d, no head] \\
E \arrow[d, no head] \\
T \arrow[d, no head] \\
F
\end{tikzcd}$$

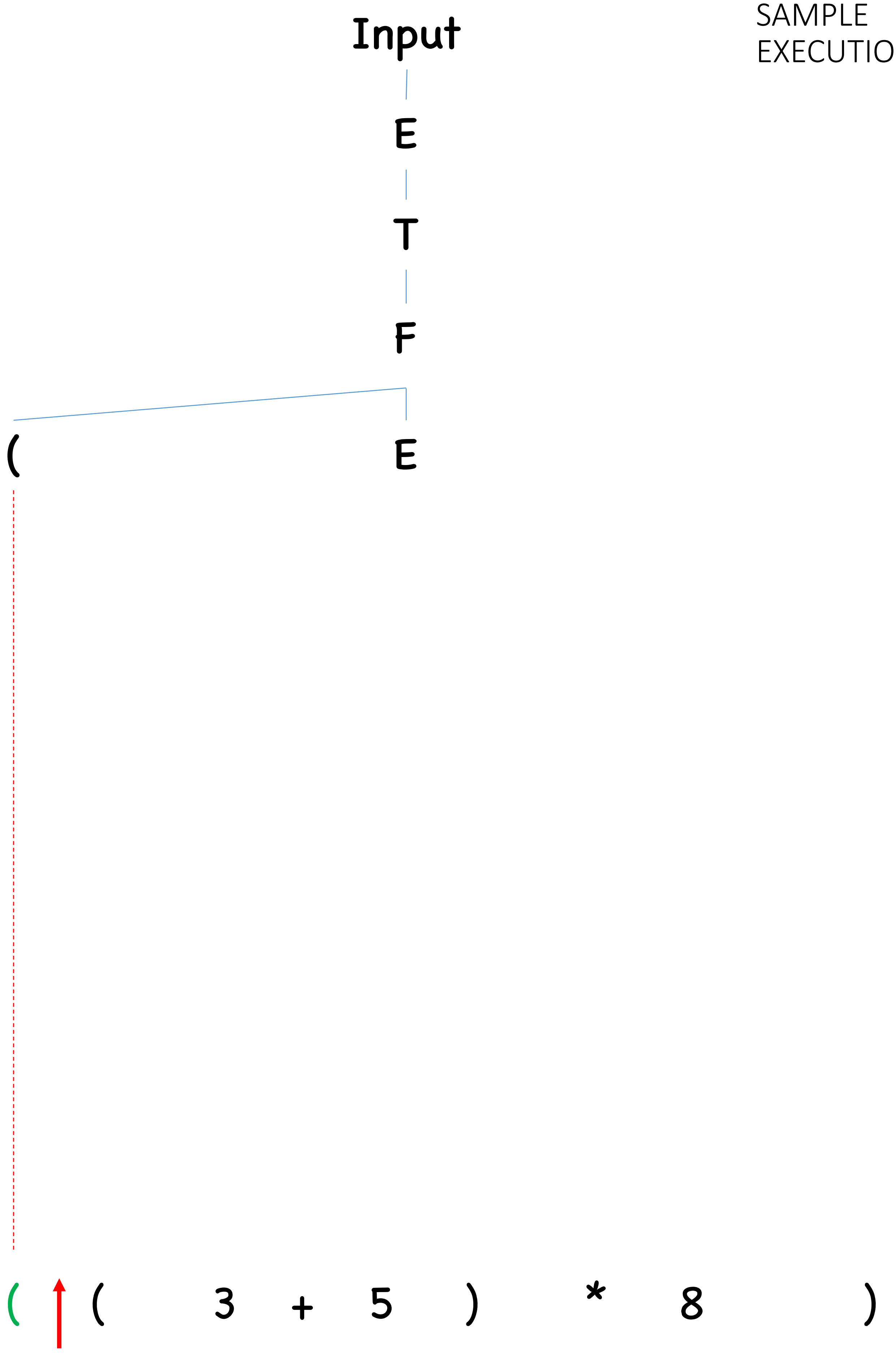
peek(1) returns LPAREN

↑ ((3 + 5) * 8)

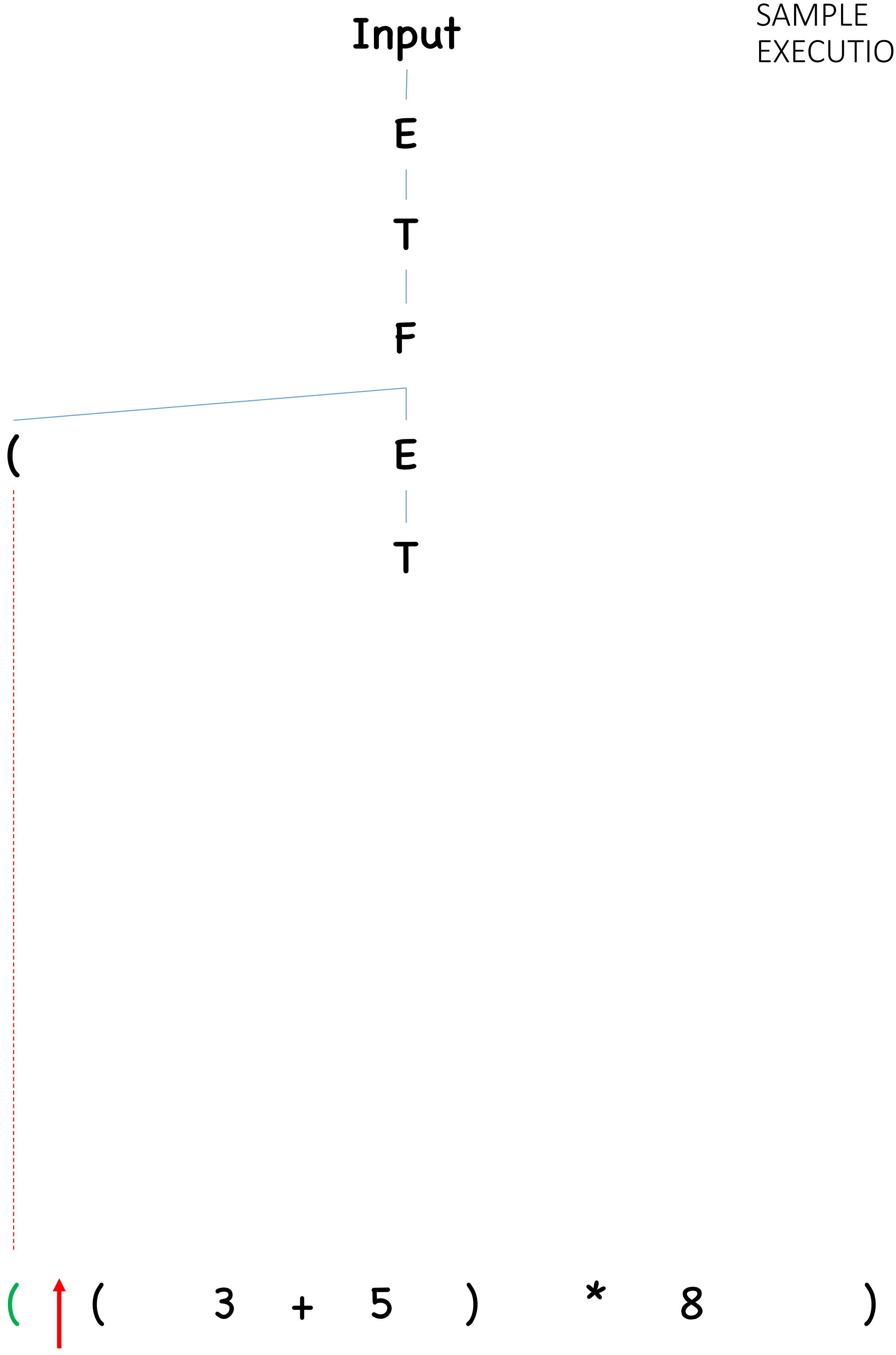
SAMPLE EXECUTION



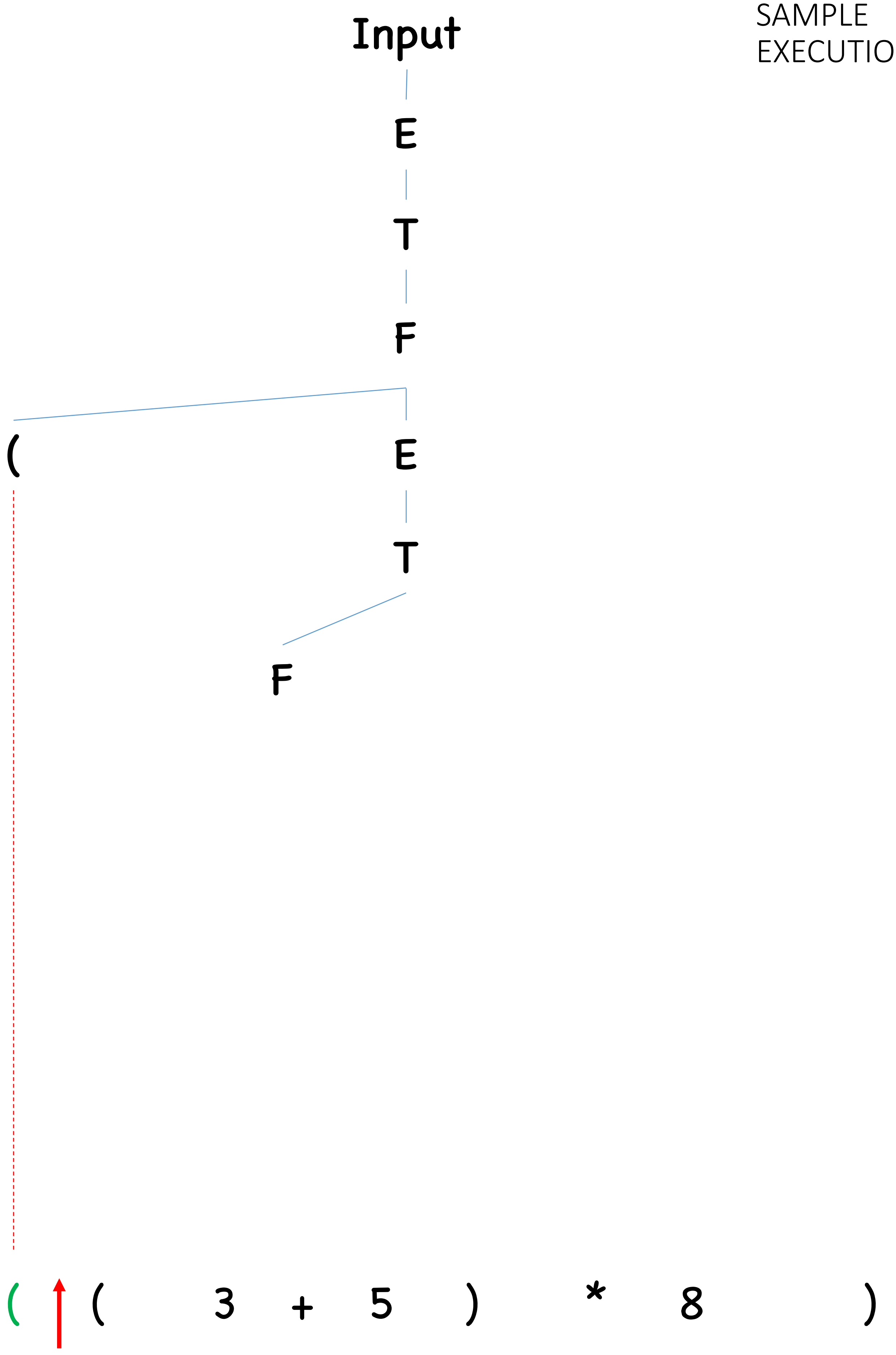
SAMPLE EXECUTION



SAMPLE
EXECUTION

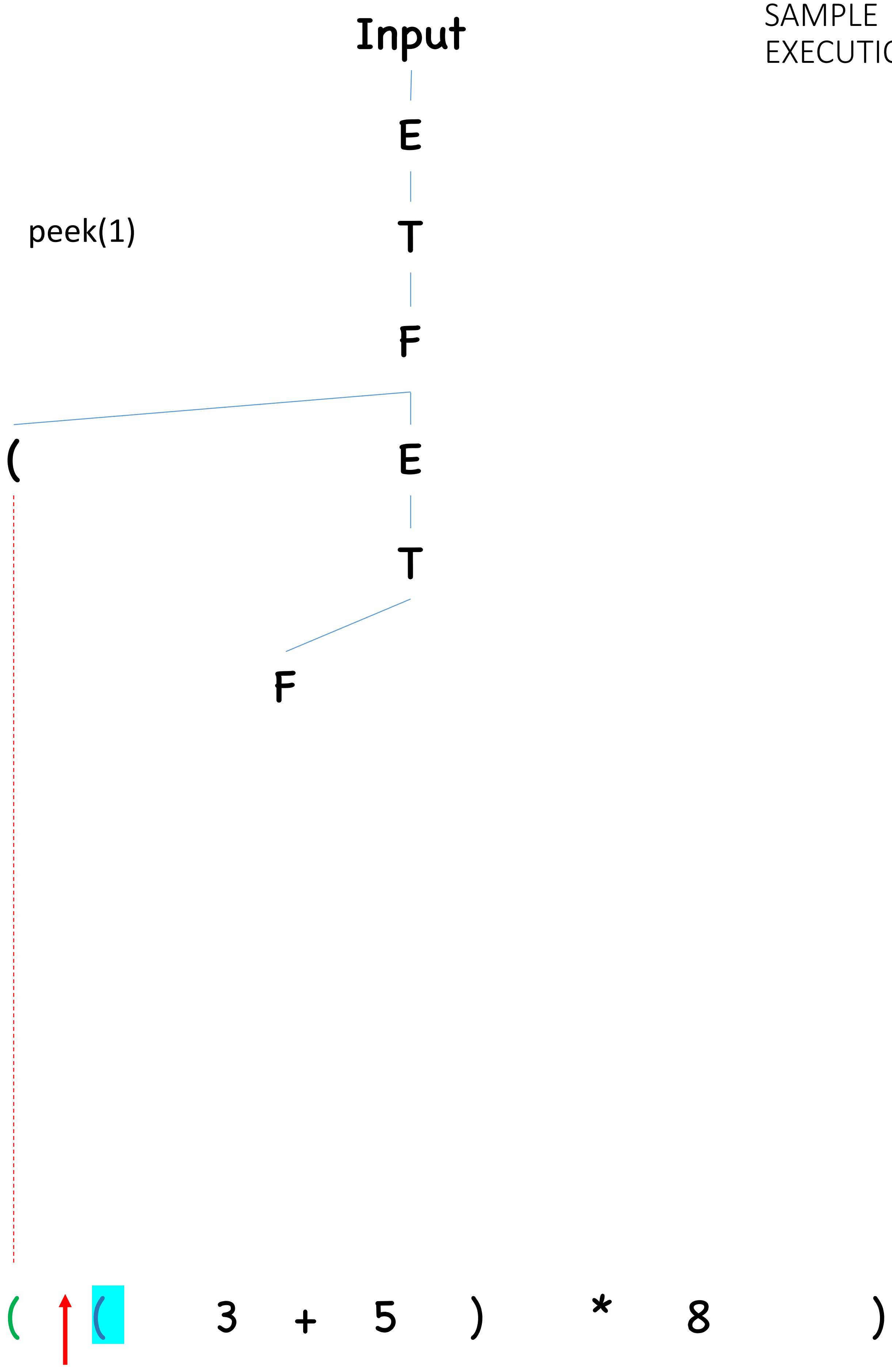


SAMPLE
EXECUTION



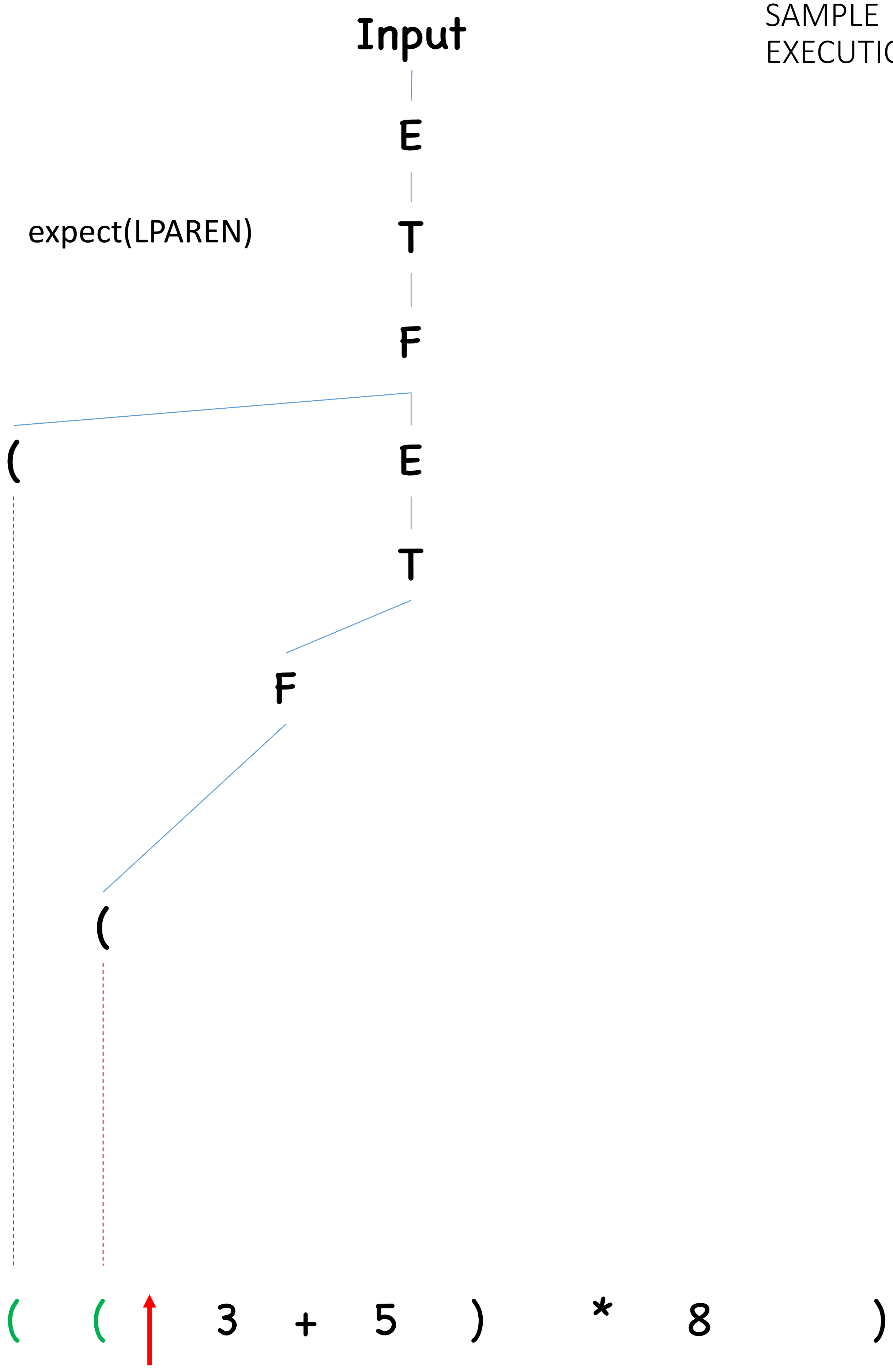
SAMPLE
EXECUTION

peek(1)

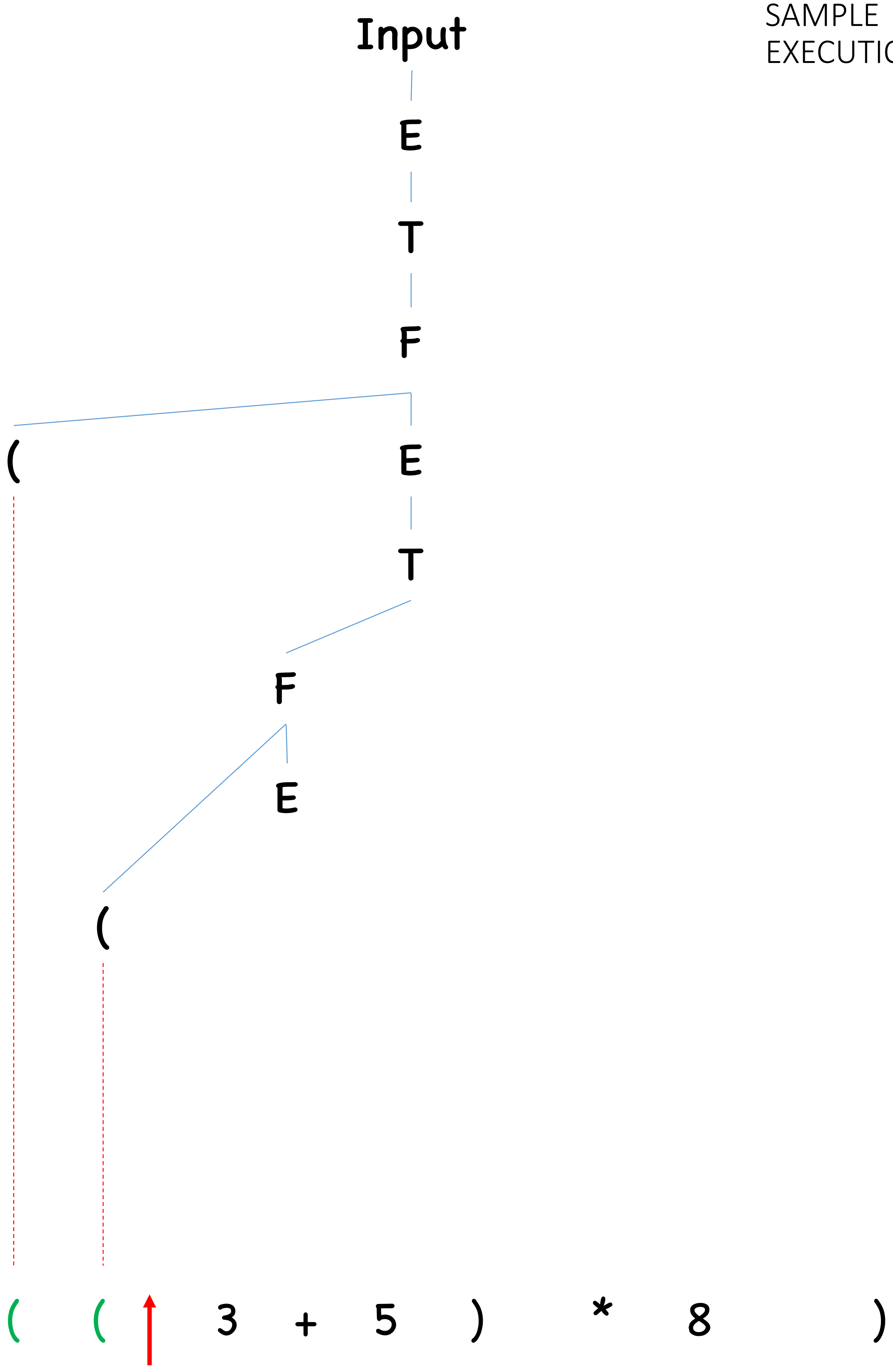


SAMPLE
EXECUTION

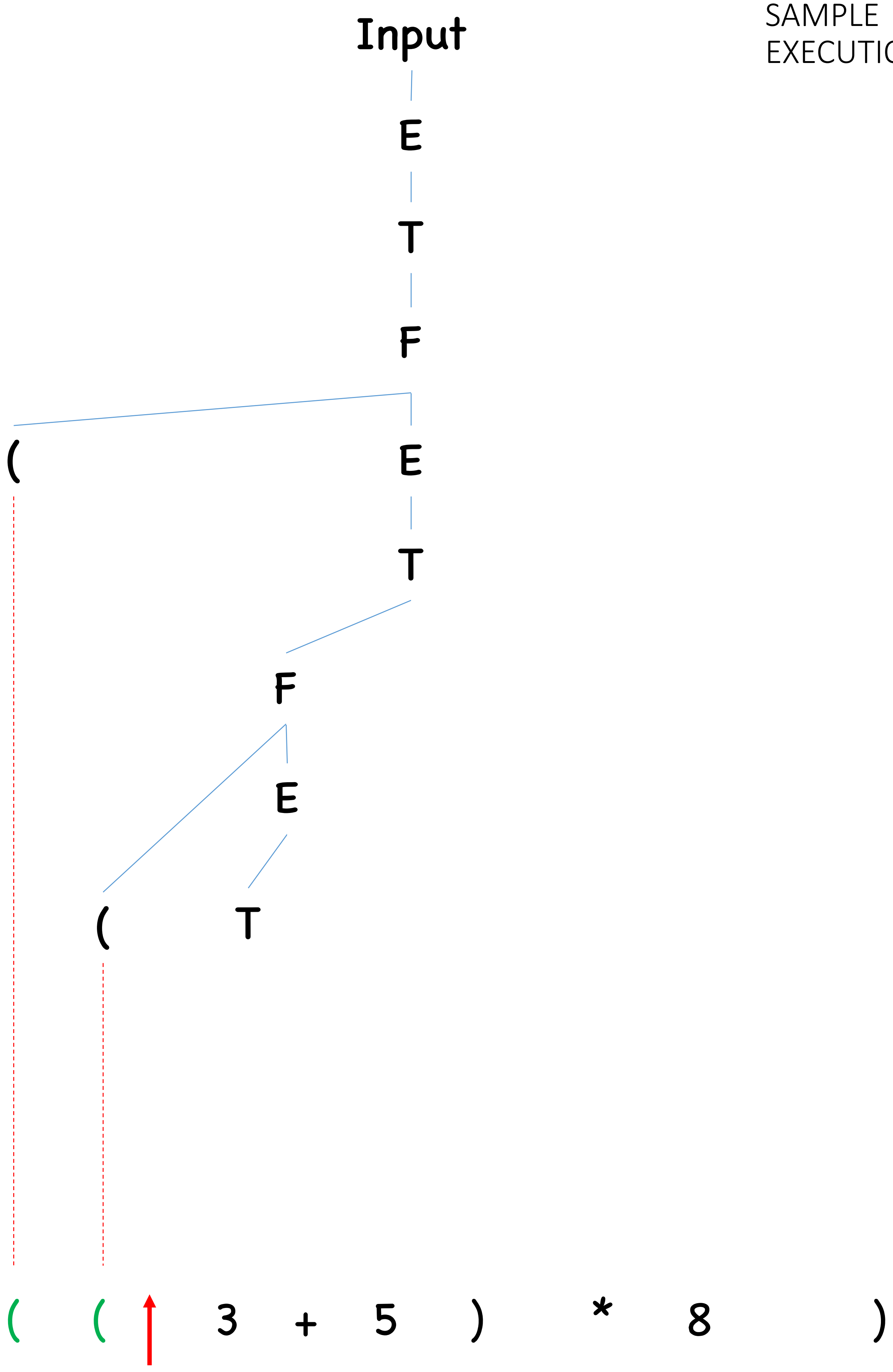
expect(LPAREN)



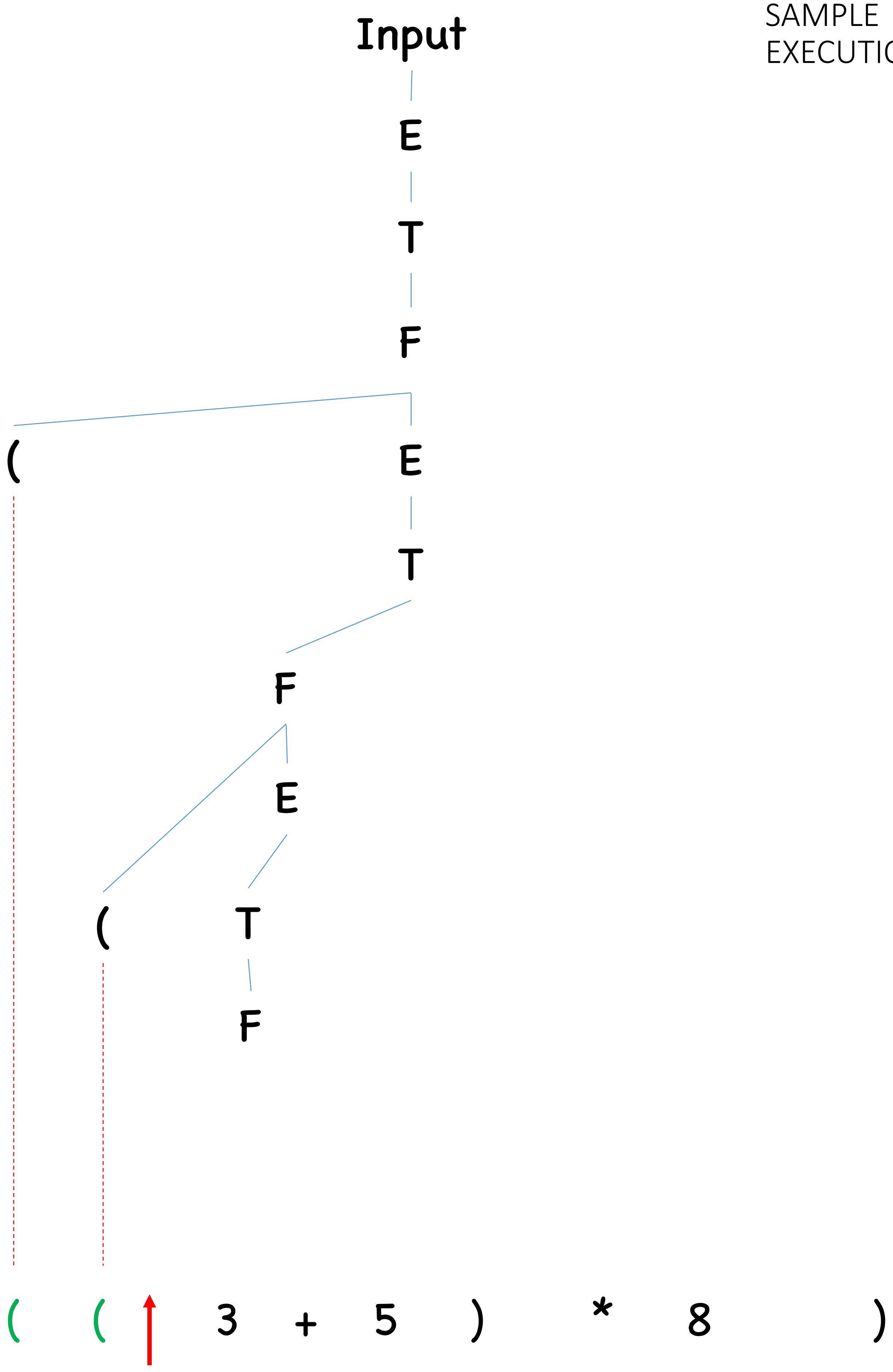
SAMPLE EXECUTION



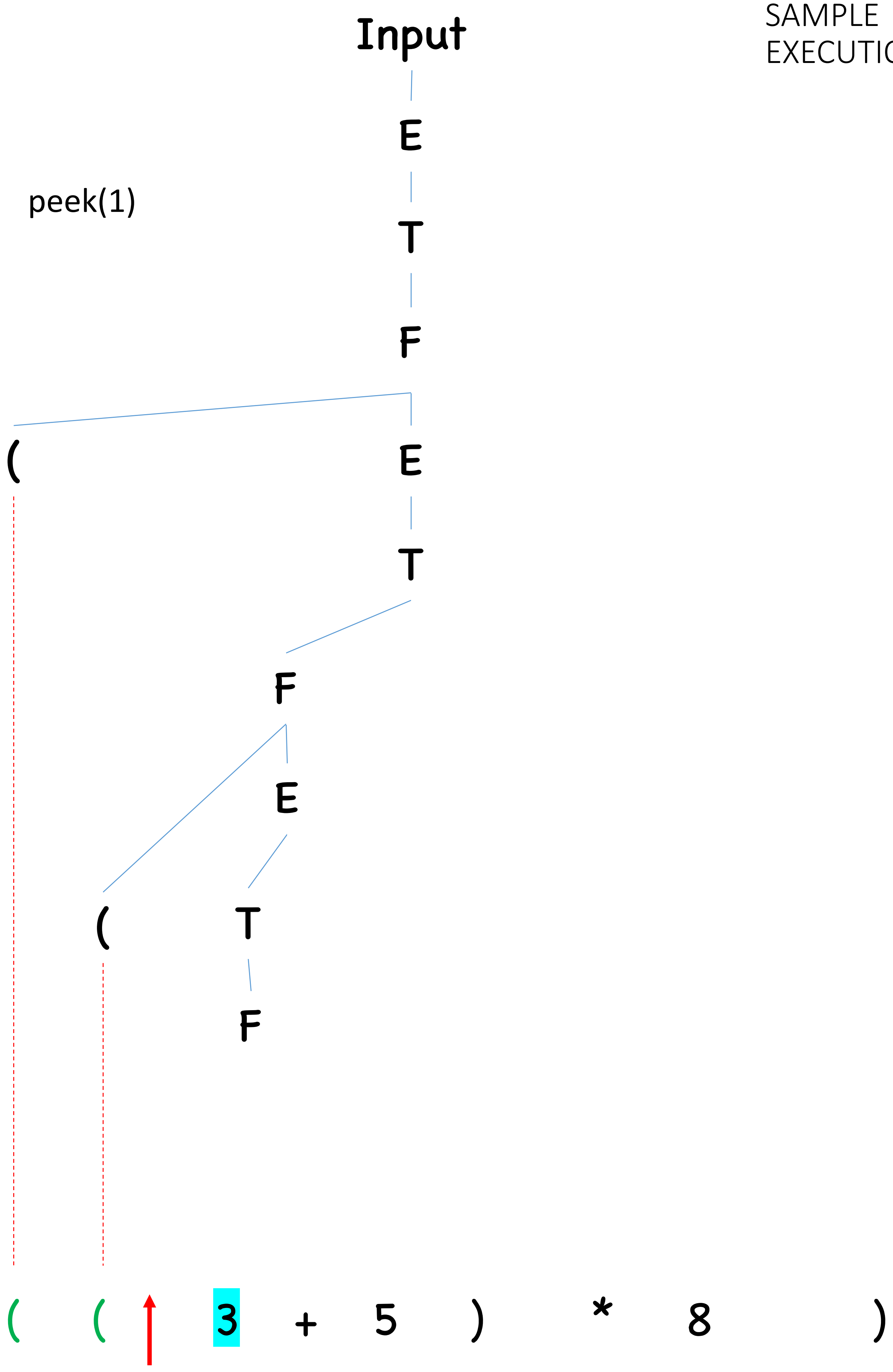
SAMPLE EXECUTION



SAMPLE EXECUTION

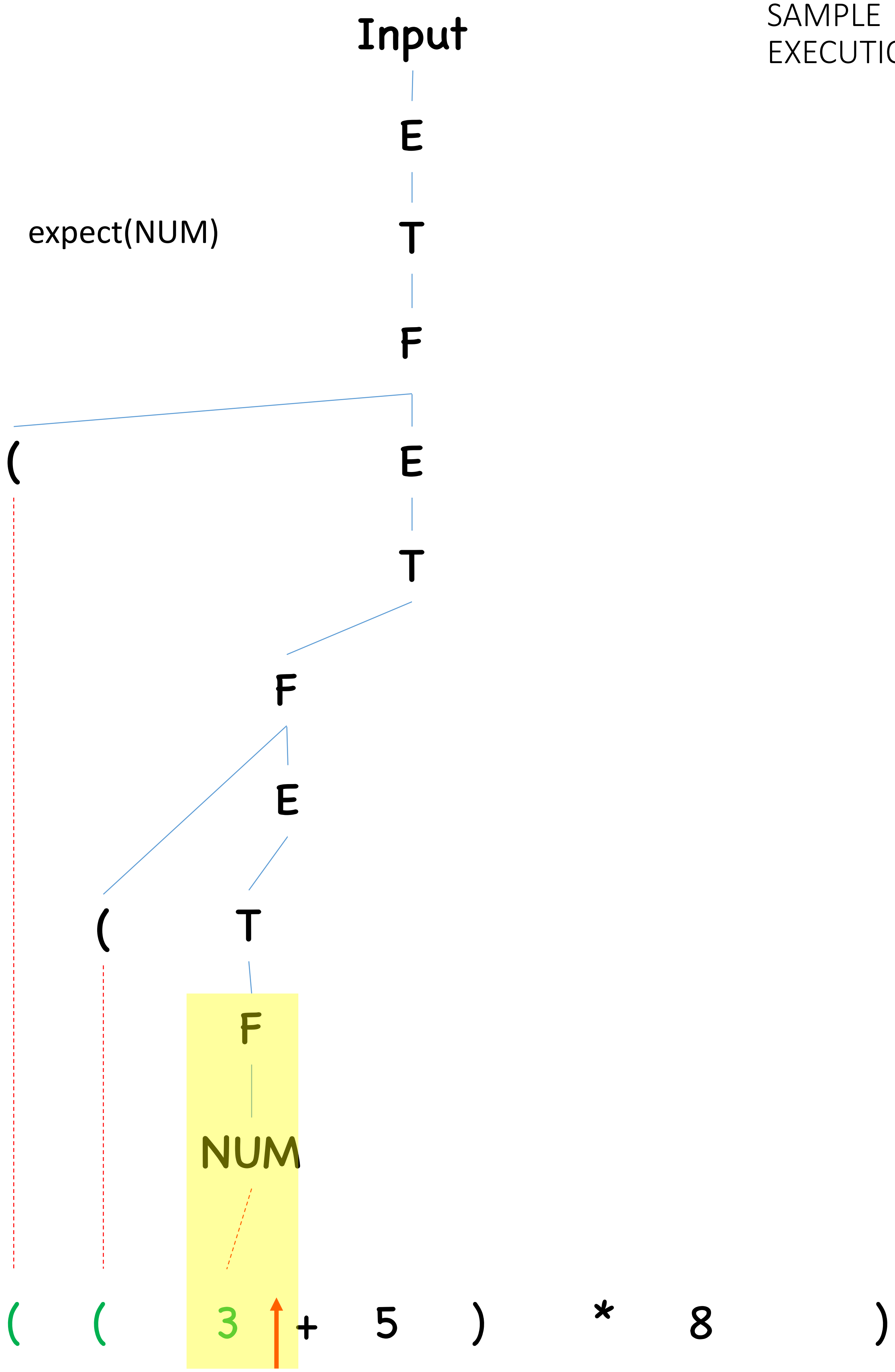


SAMPLE
EXECUTION



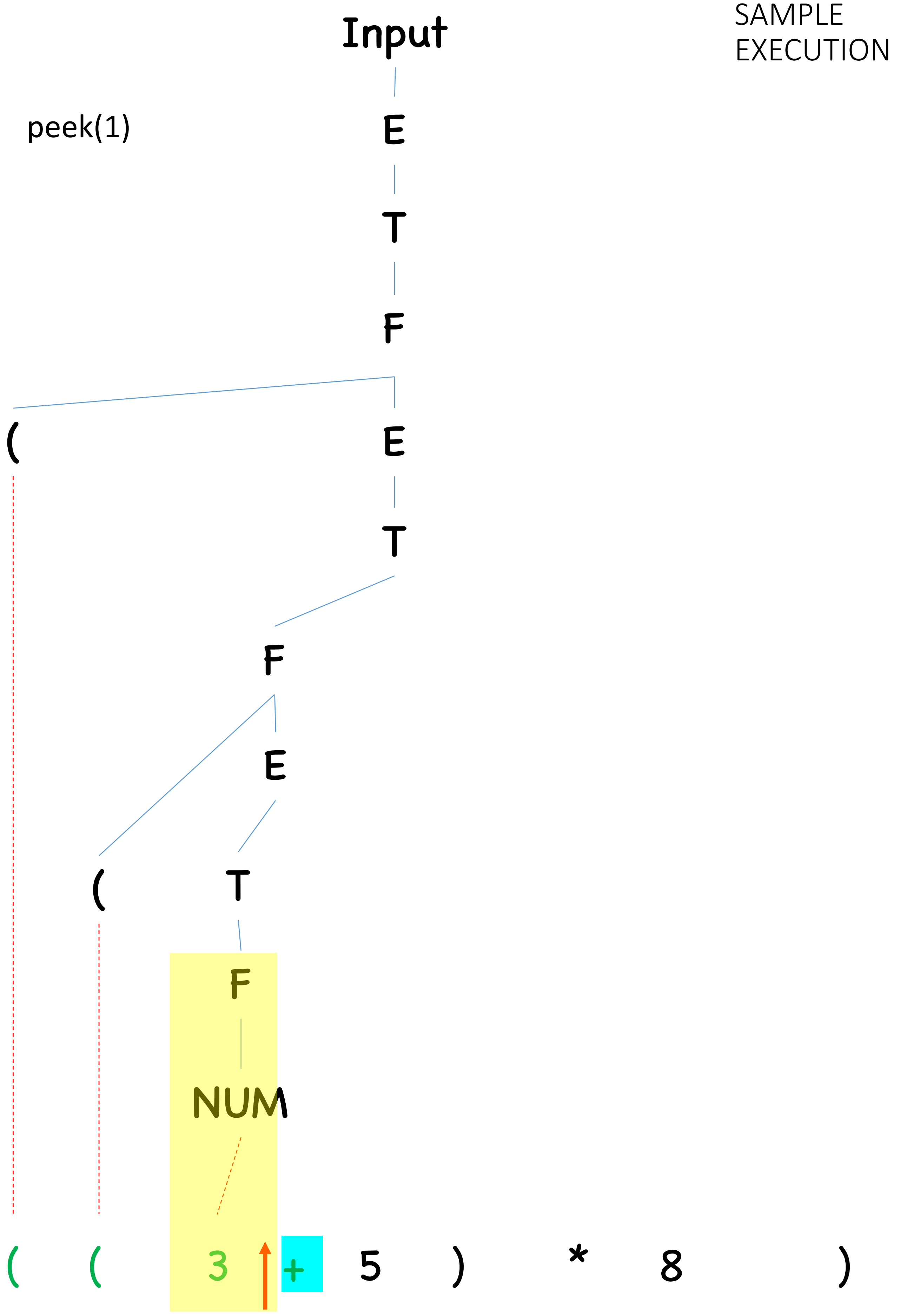
SAMPLE
EXECUTION

expect(NUM)



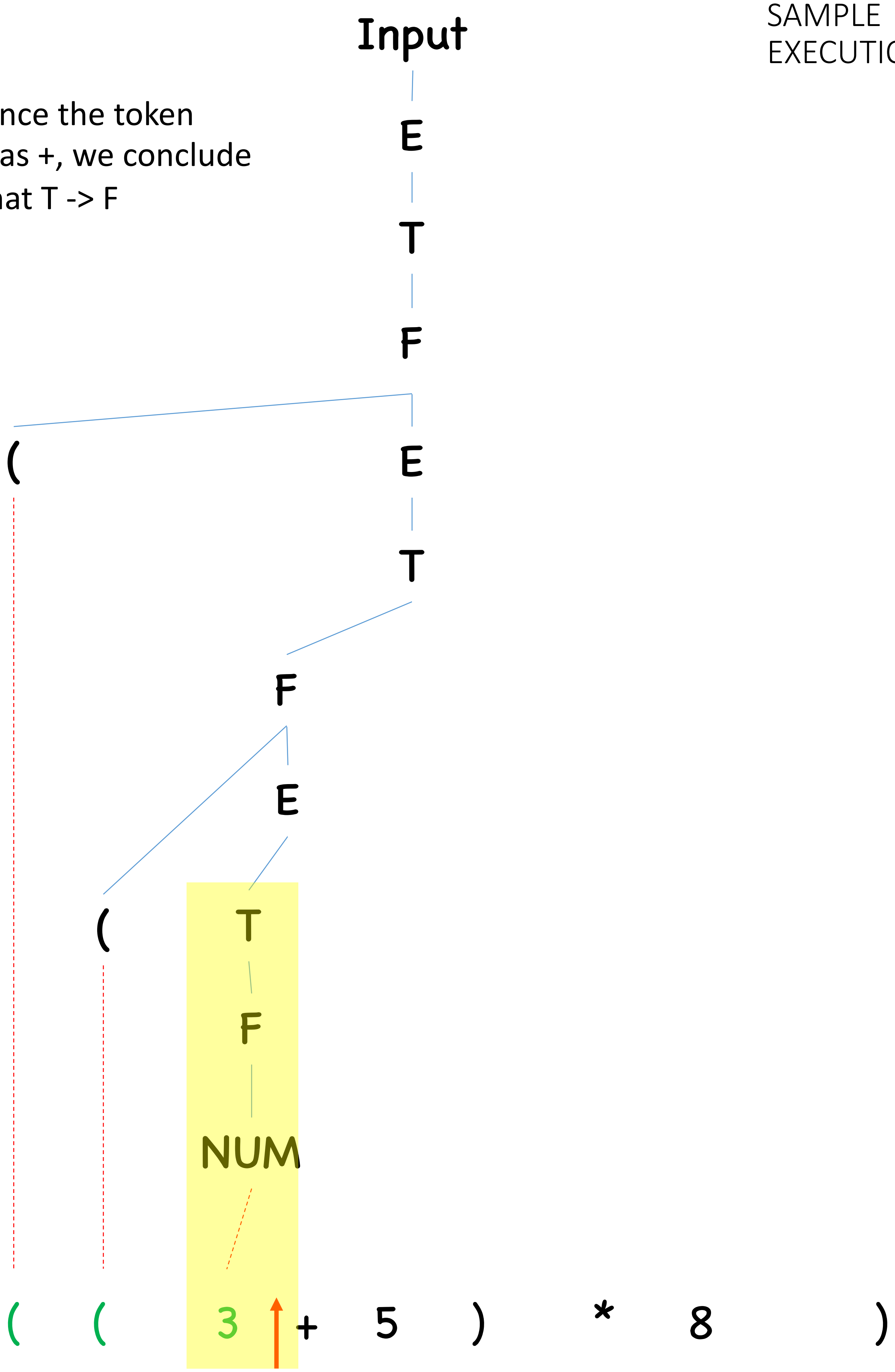
SAMPLE
EXECUTION

peek(1)

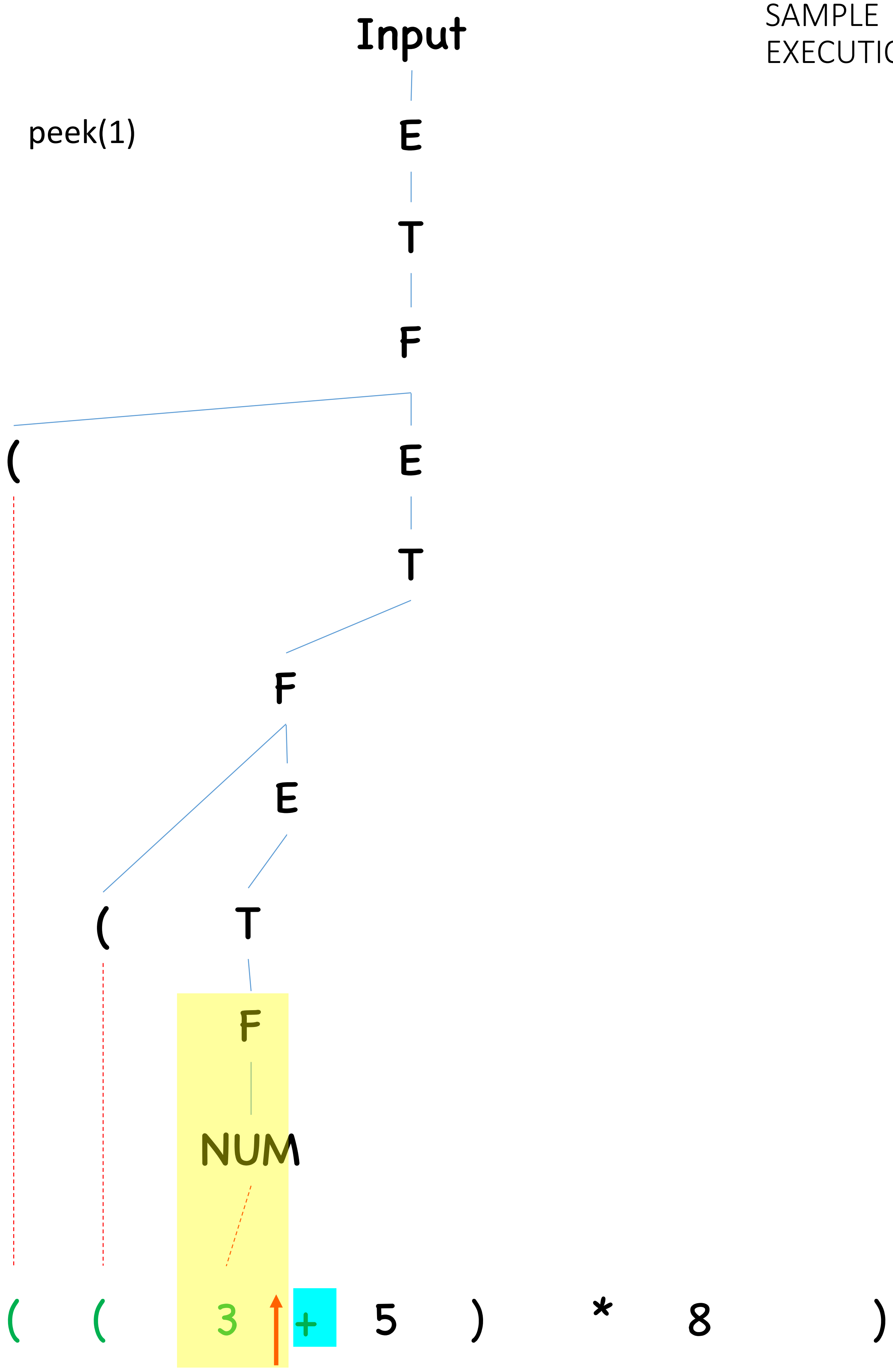


SAMPLE EXECUTION

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

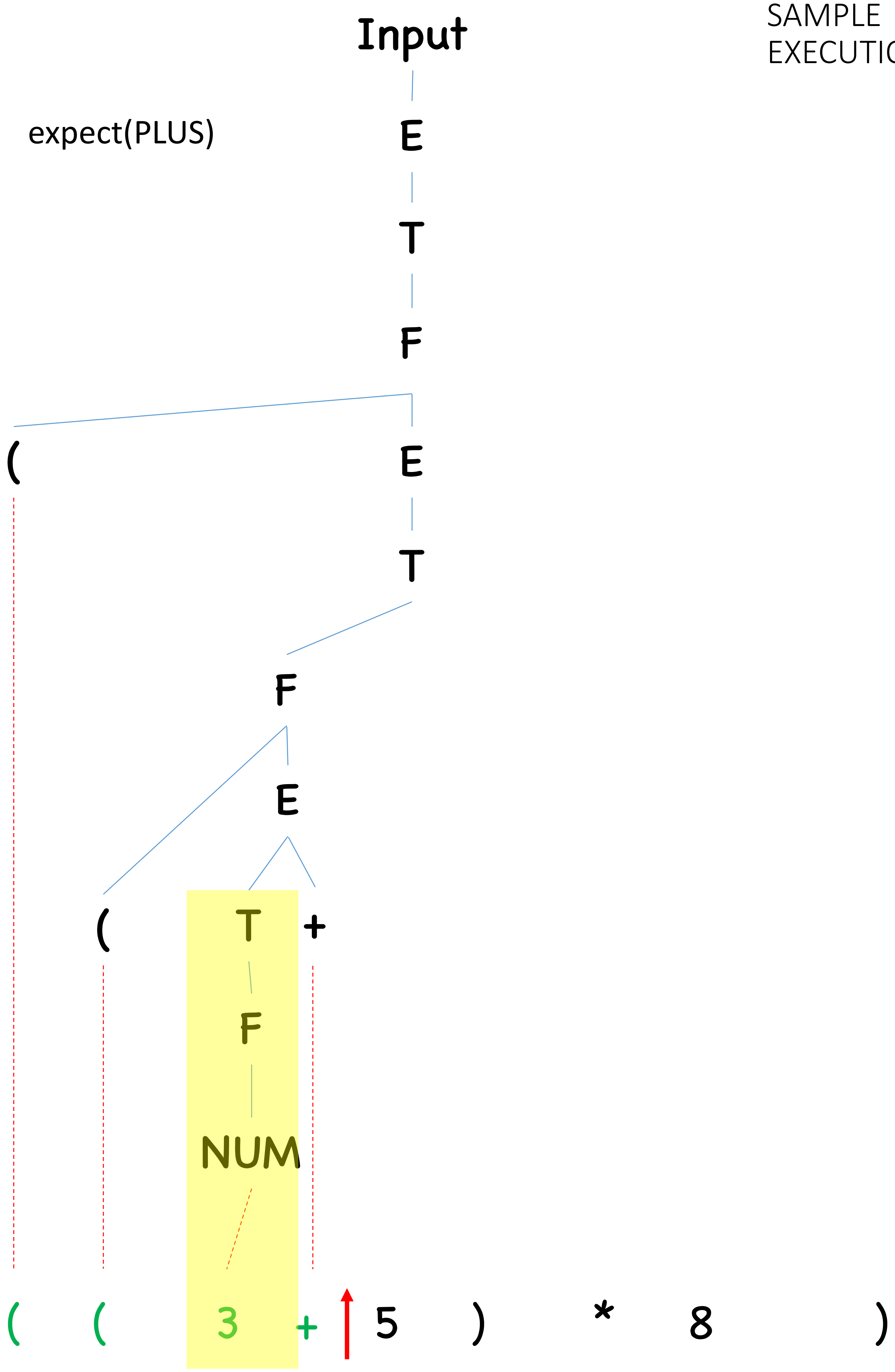


SAMPLE EXECUTION

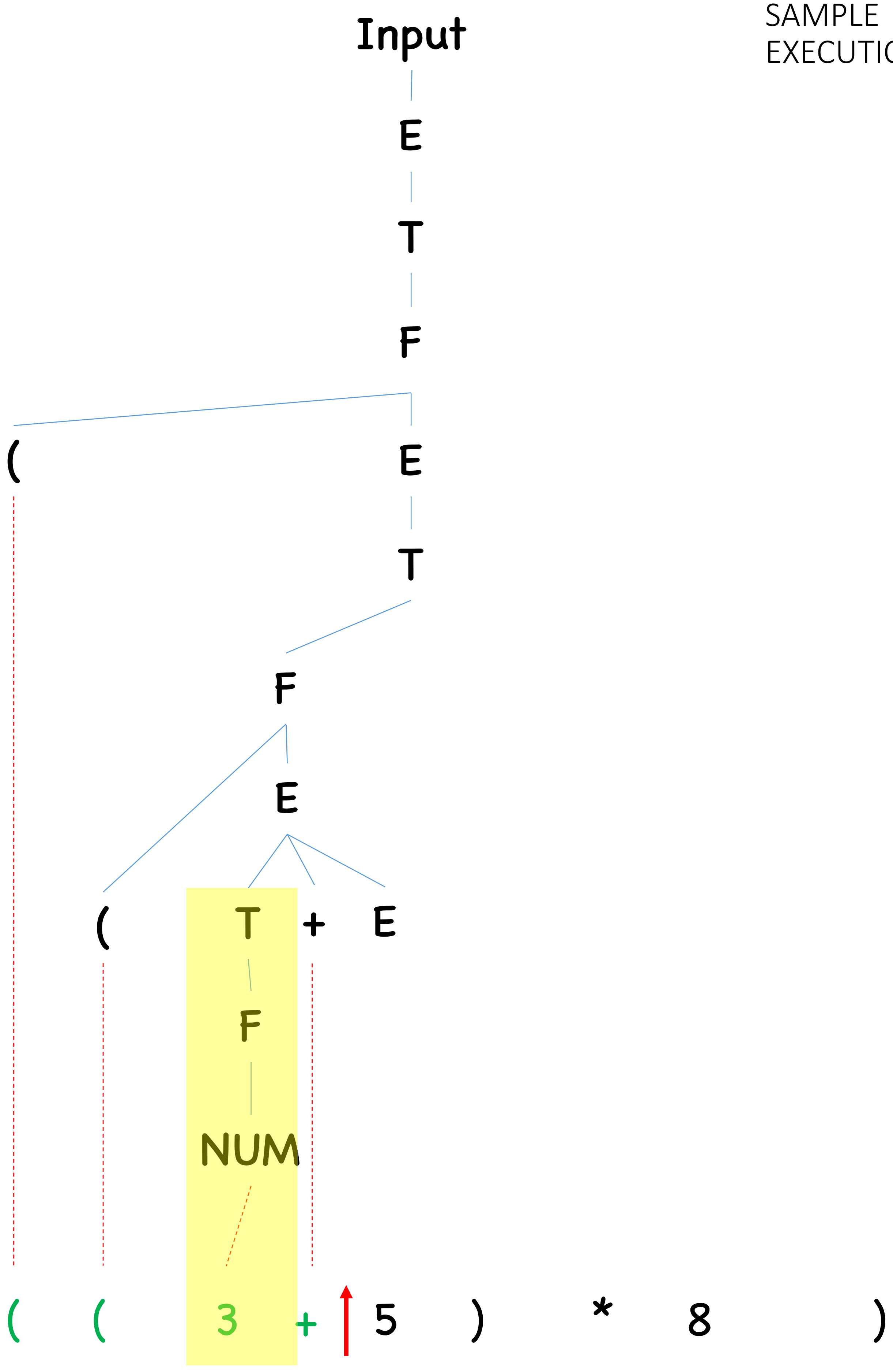


SAMPLE
EXECUTION

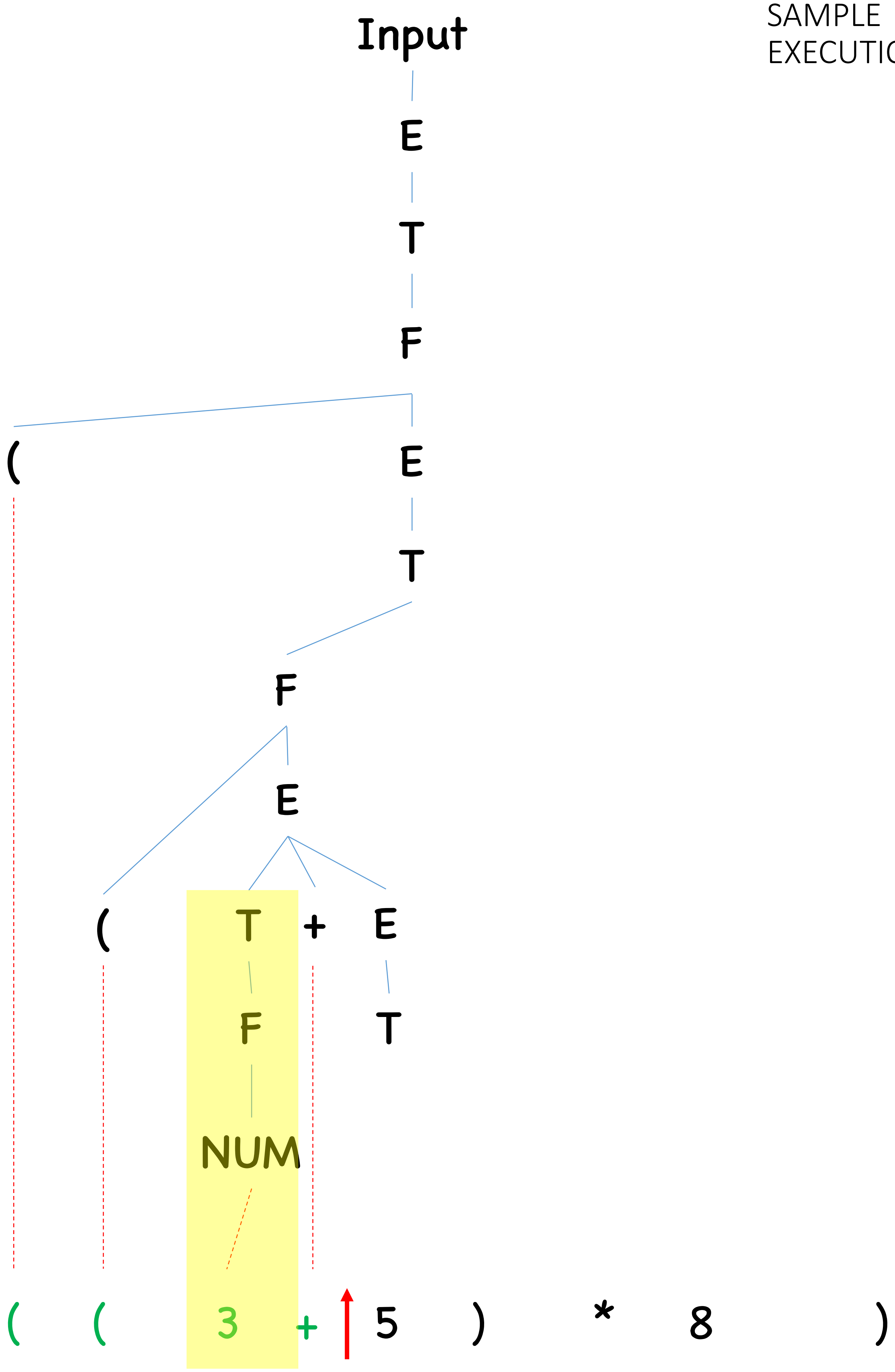
expect(PLUS)



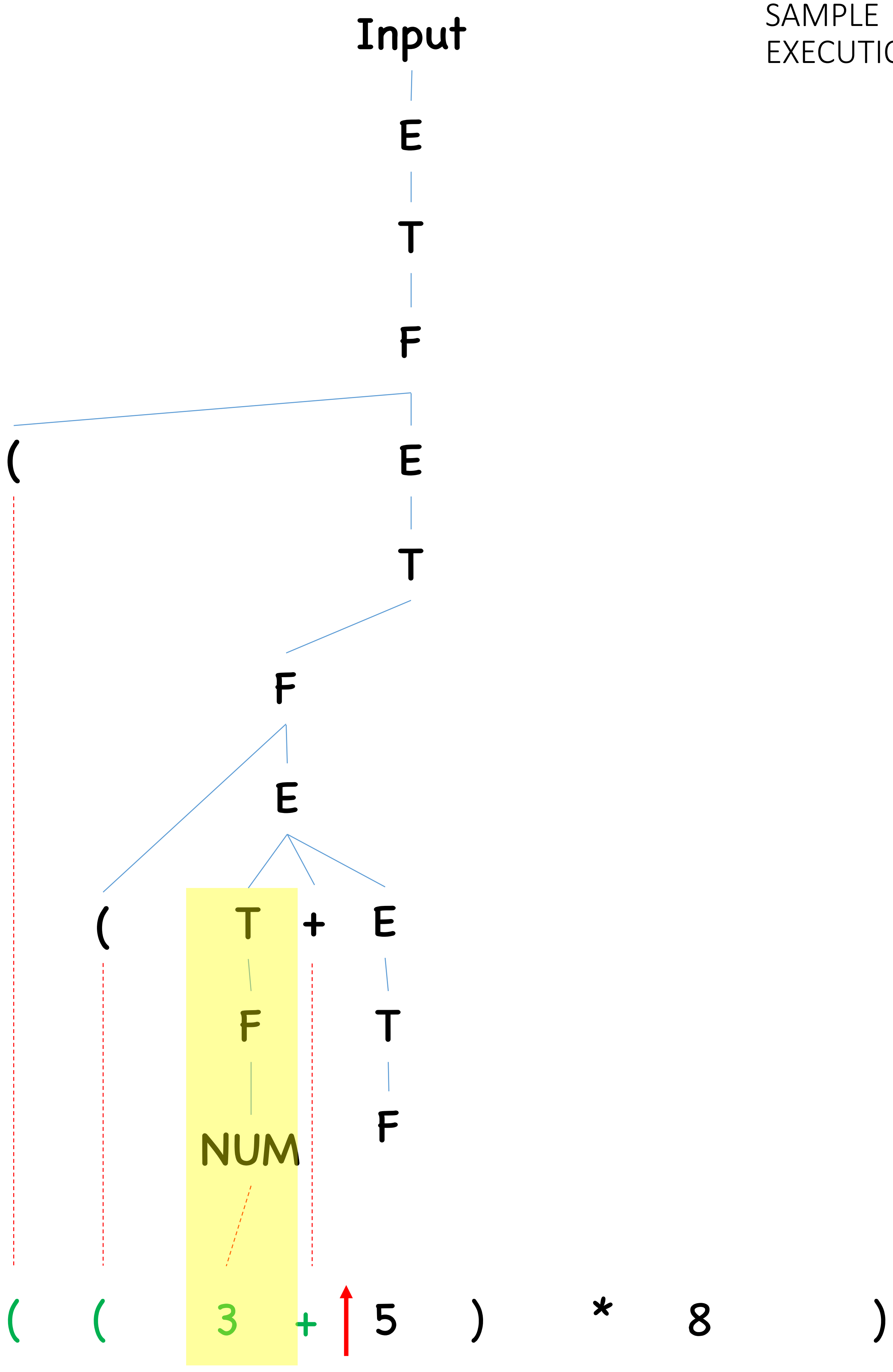
SAMPLE EXECUTION



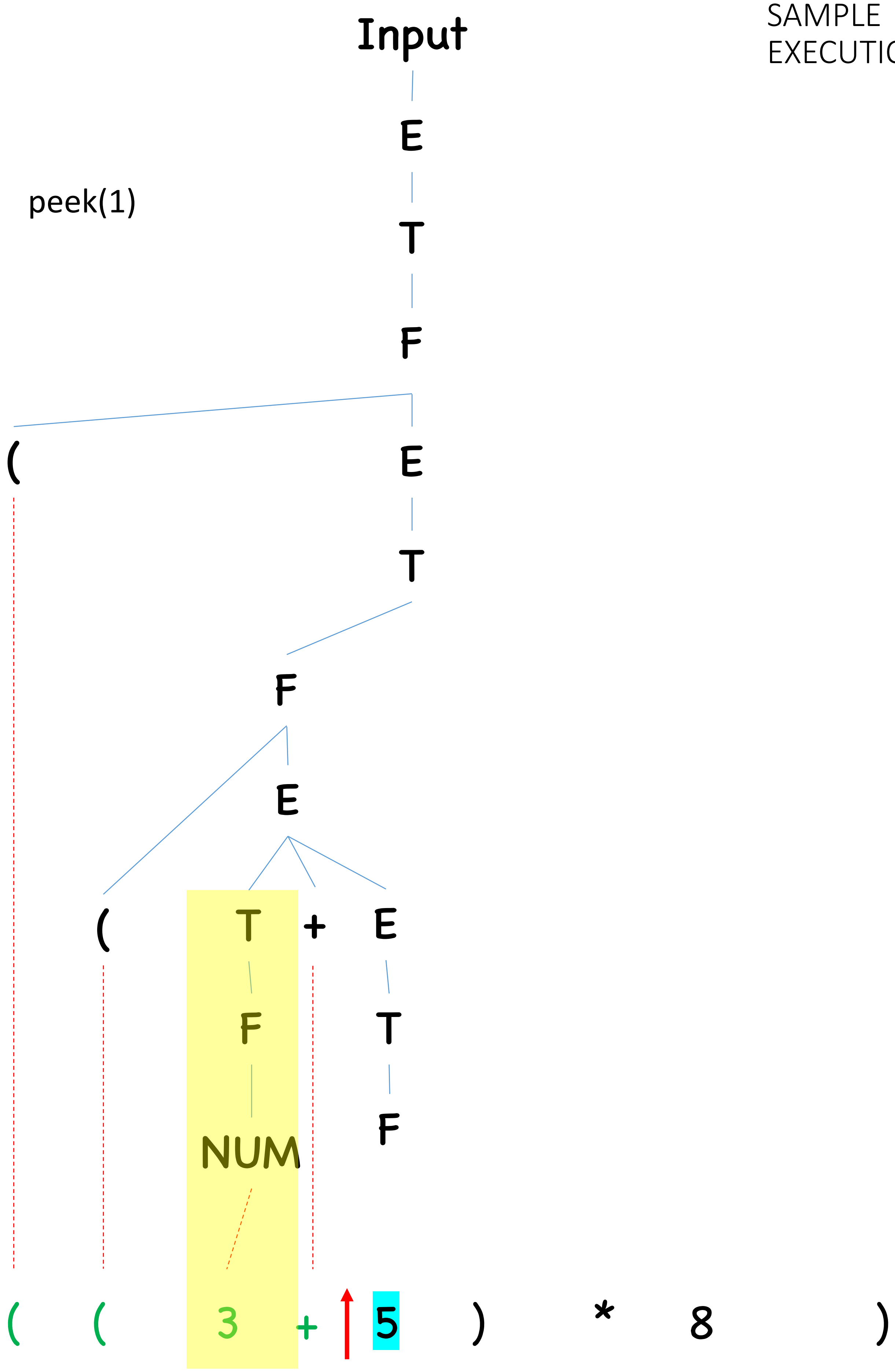
SAMPLE EXECUTION



SAMPLE
EXECUTION

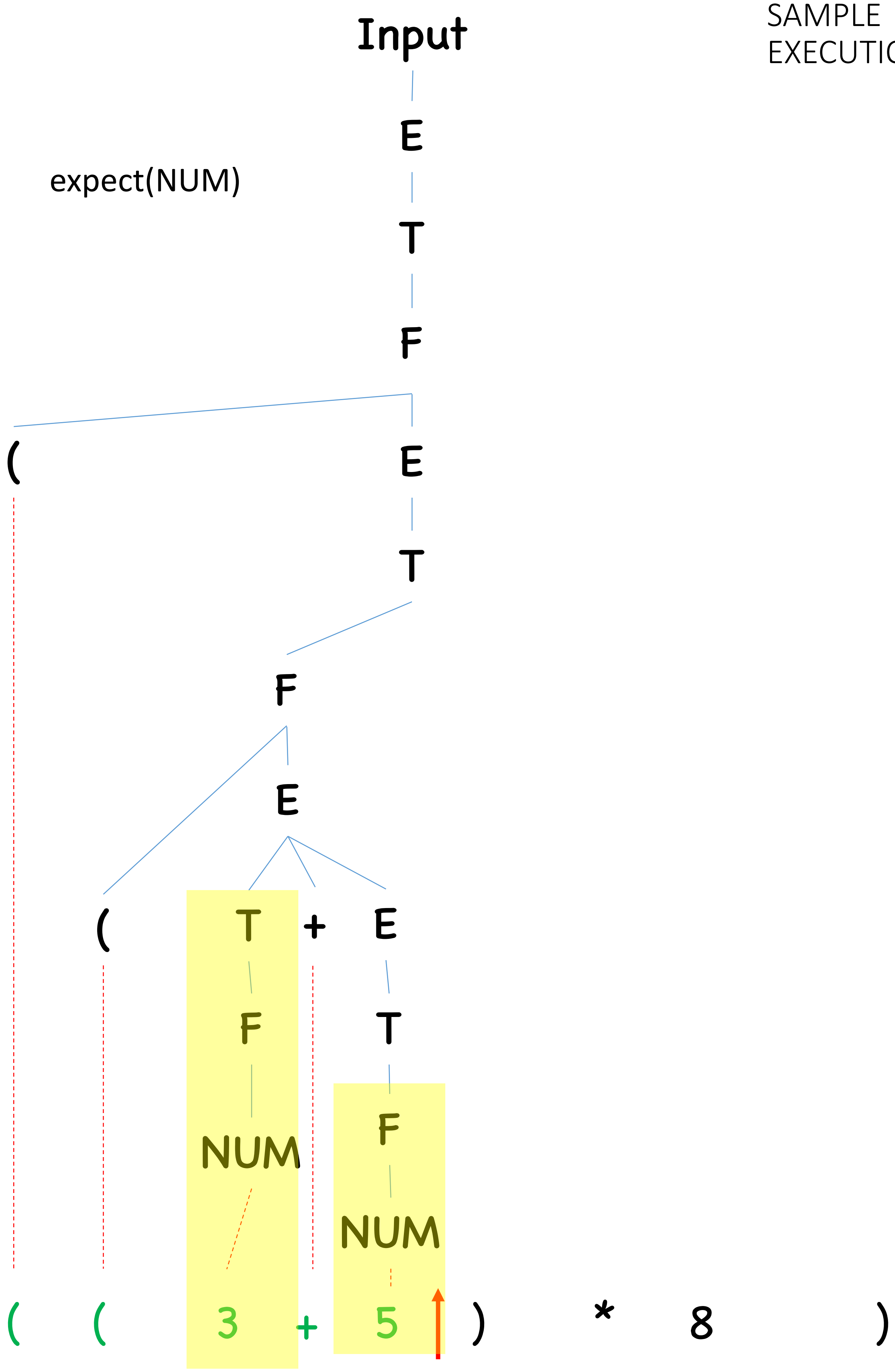


SAMPLE
EXECUTION



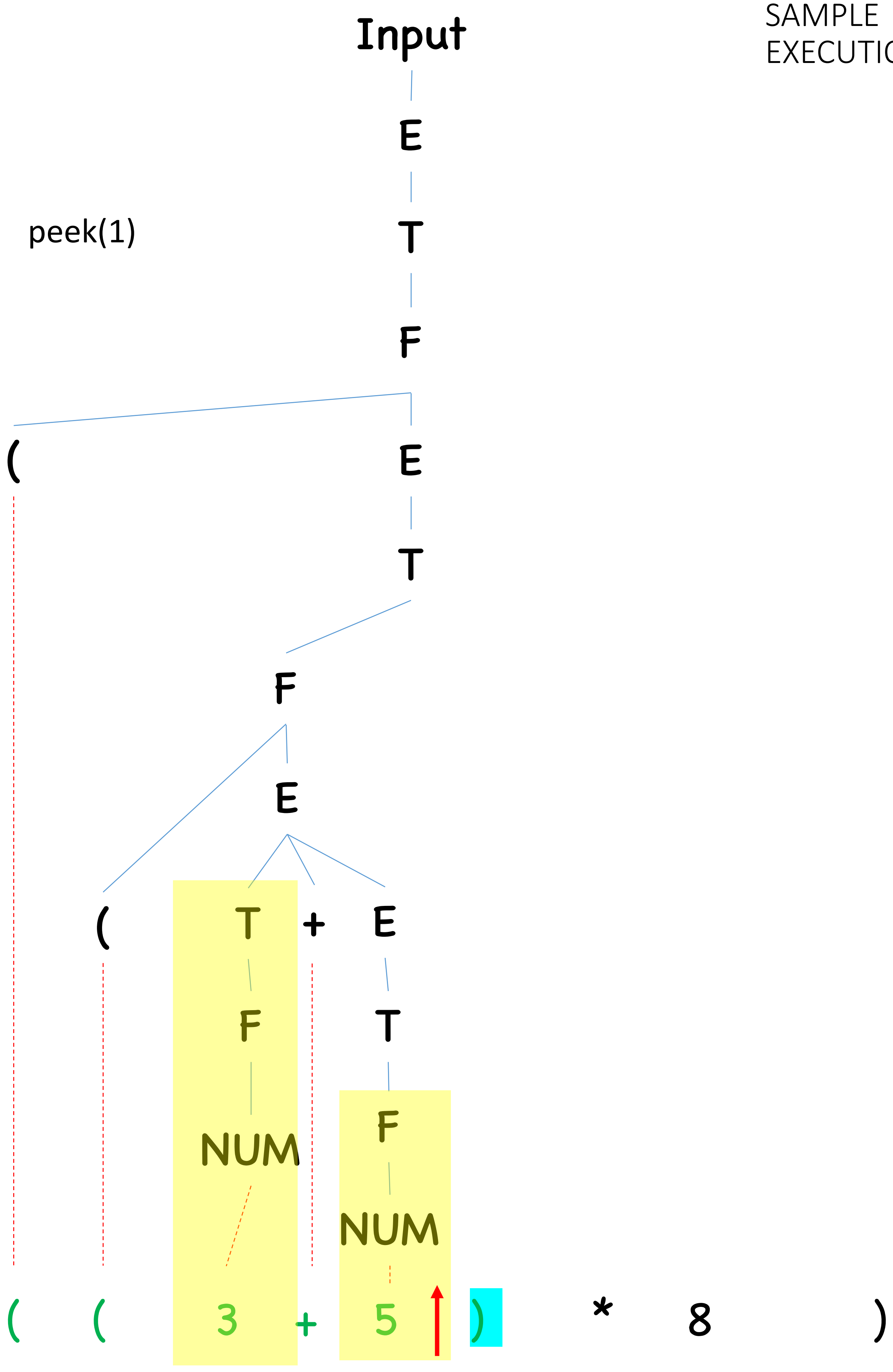
SAMPLE EXECUTION

expect(NUM)



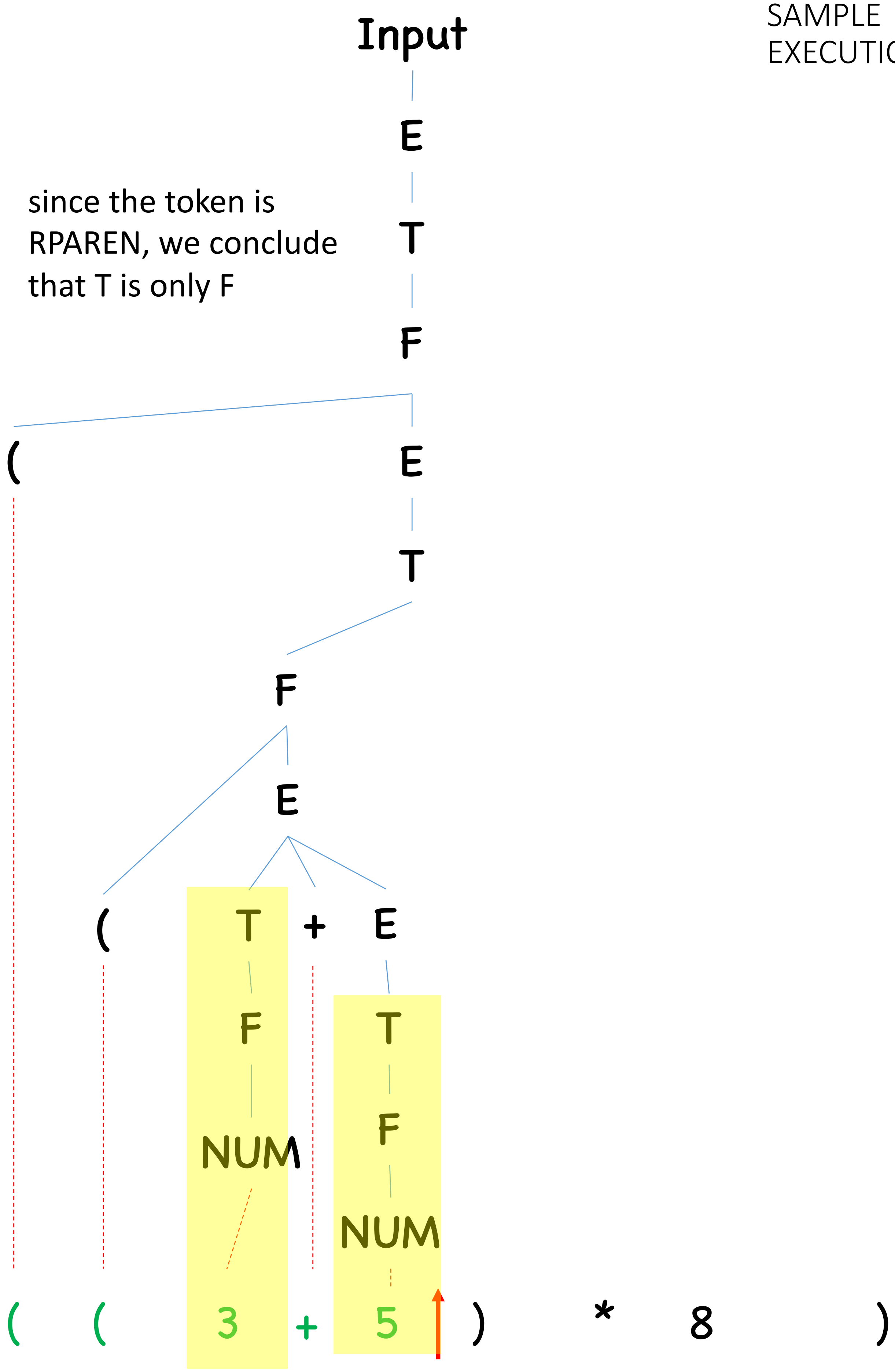
SAMPLE EXECUTION

peek(1)

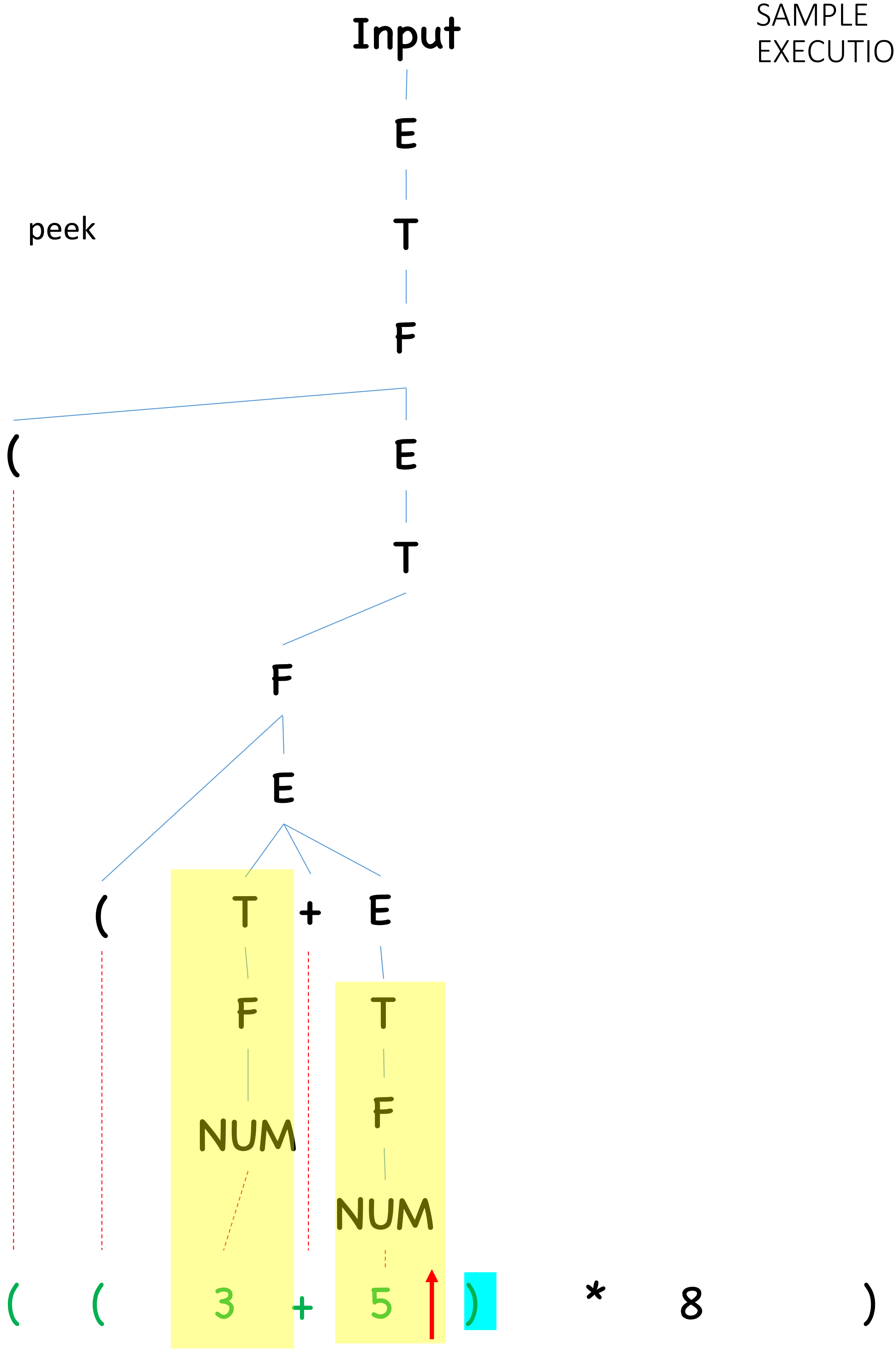


SAMPLE EXECUTION

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

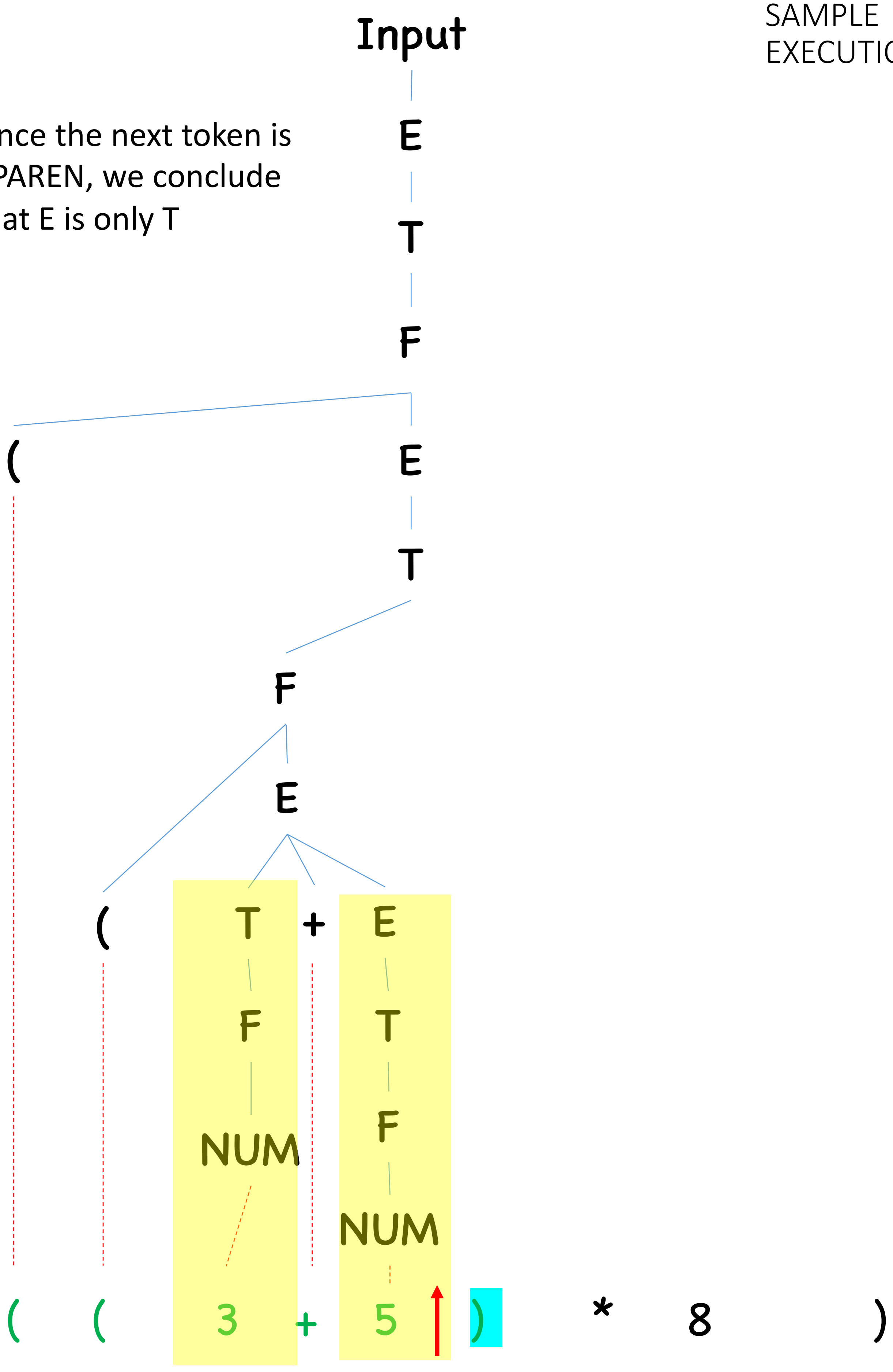


SAMPLE EXECUTION



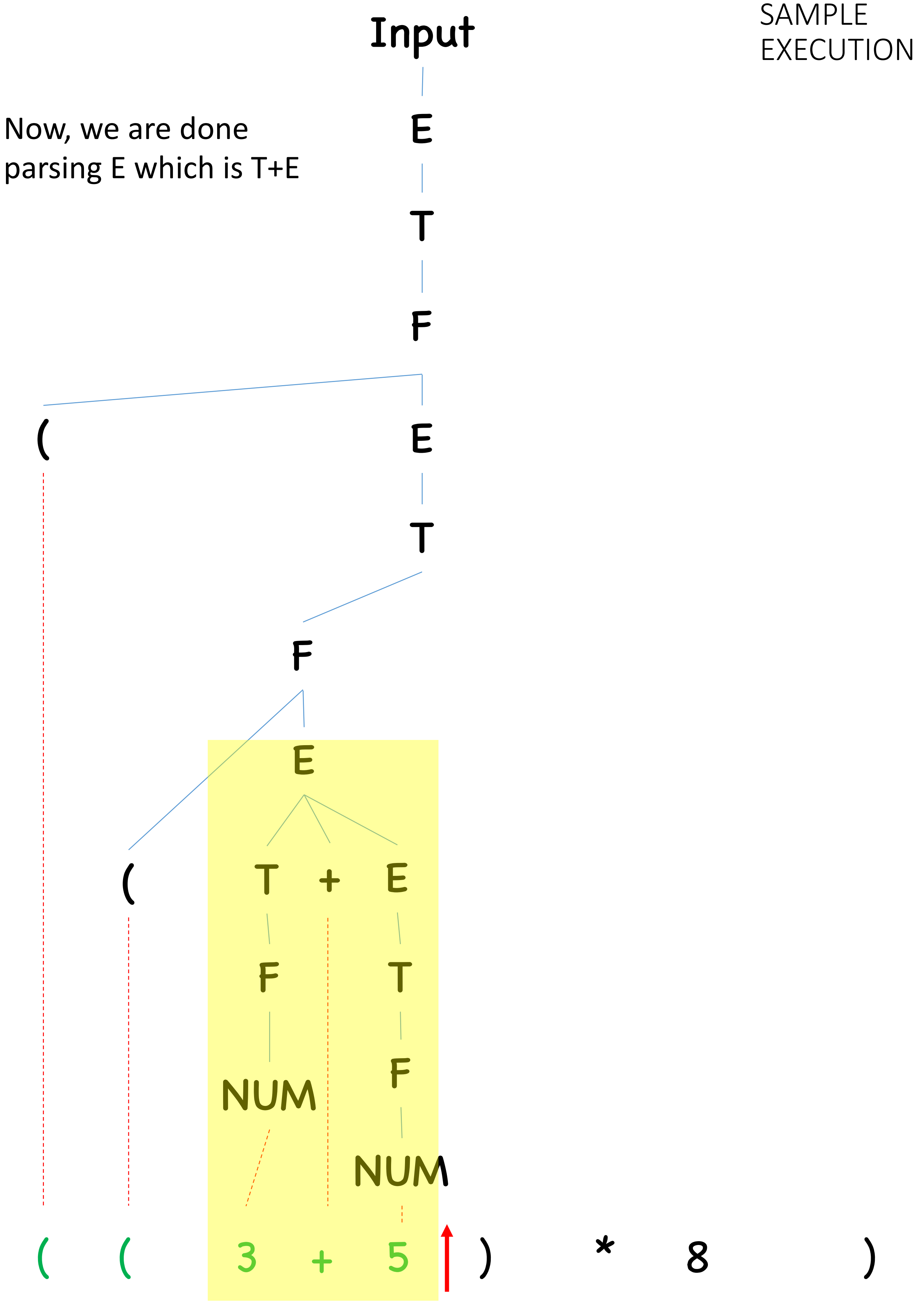
SAMPLE EXECUTION

Since the next token is RPAREN, we conclude that E is only T



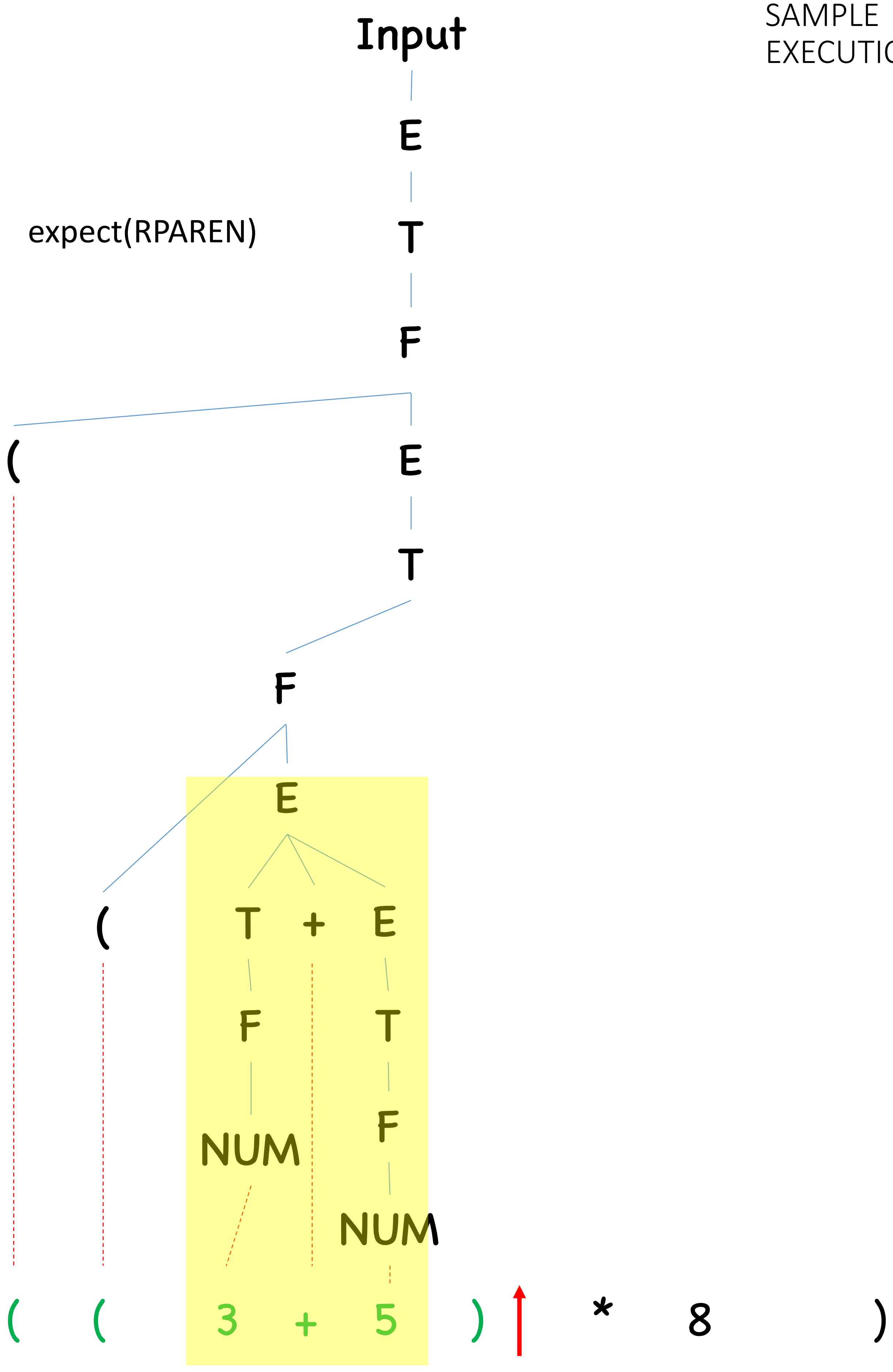
SAMPLE EXECUTION

Now, we are done parsing E which is T+E

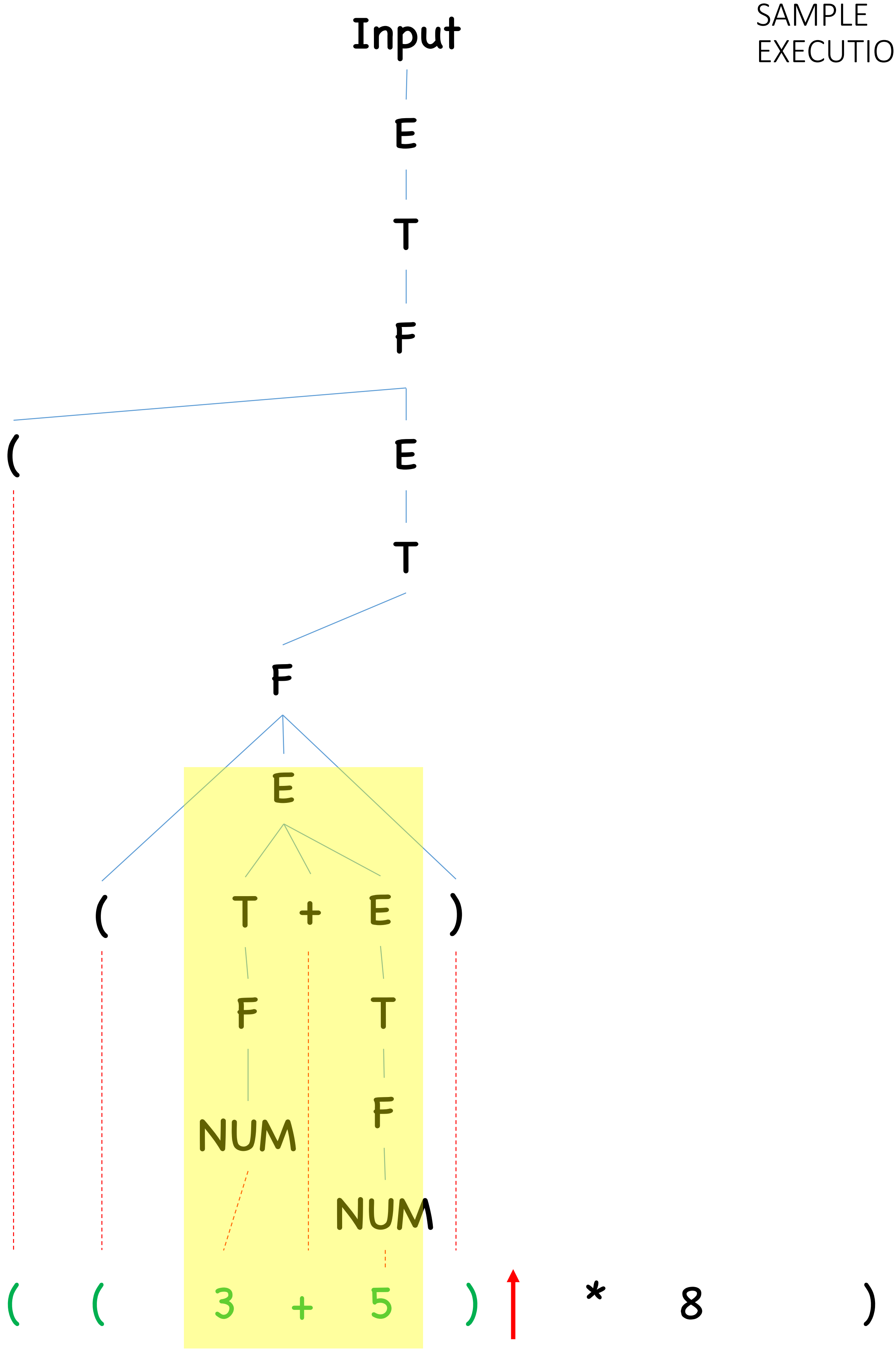


SAMPLE
EXECUTION

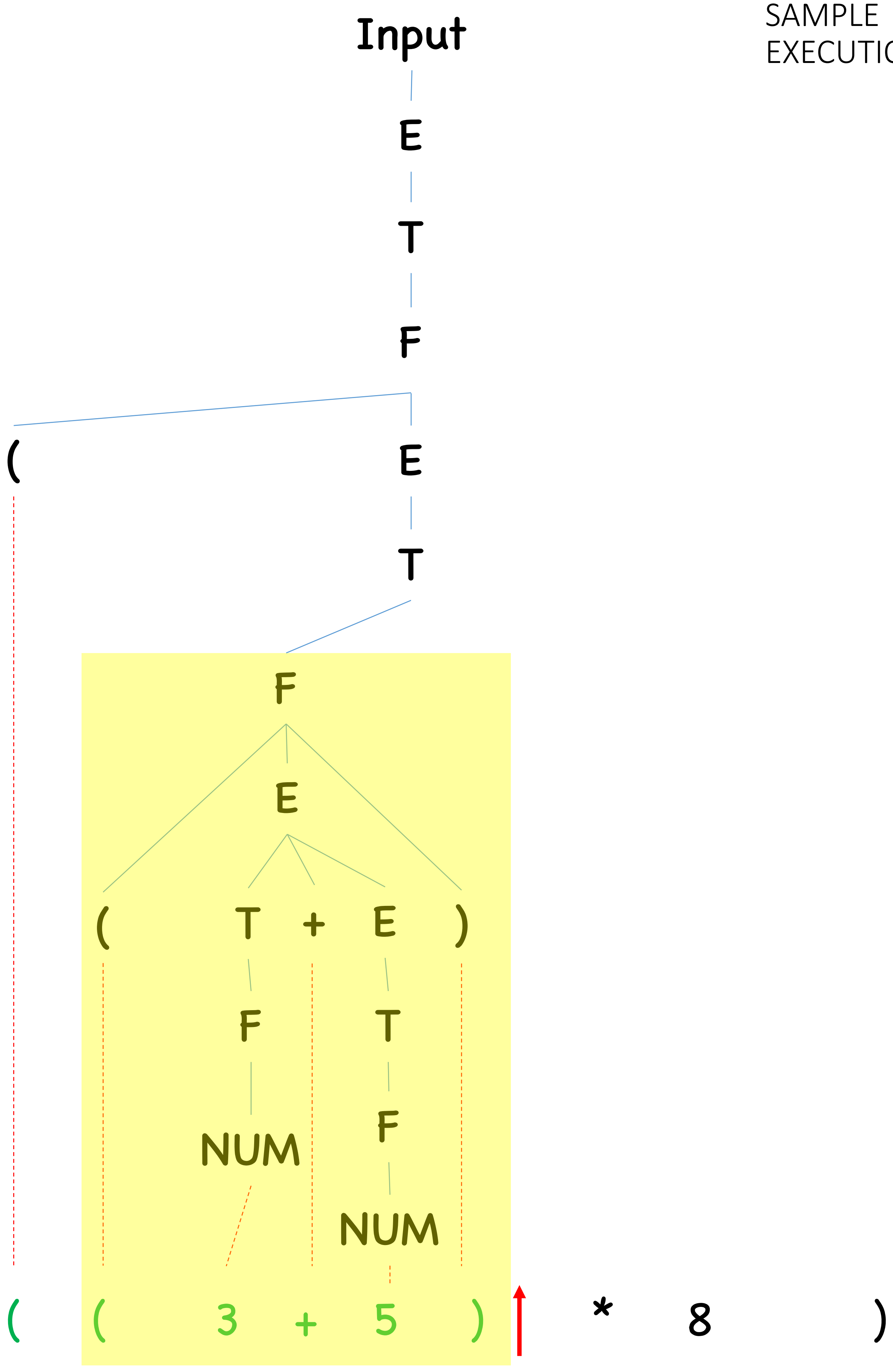
expect(RPAREN)



SAMPLE EXECUTION

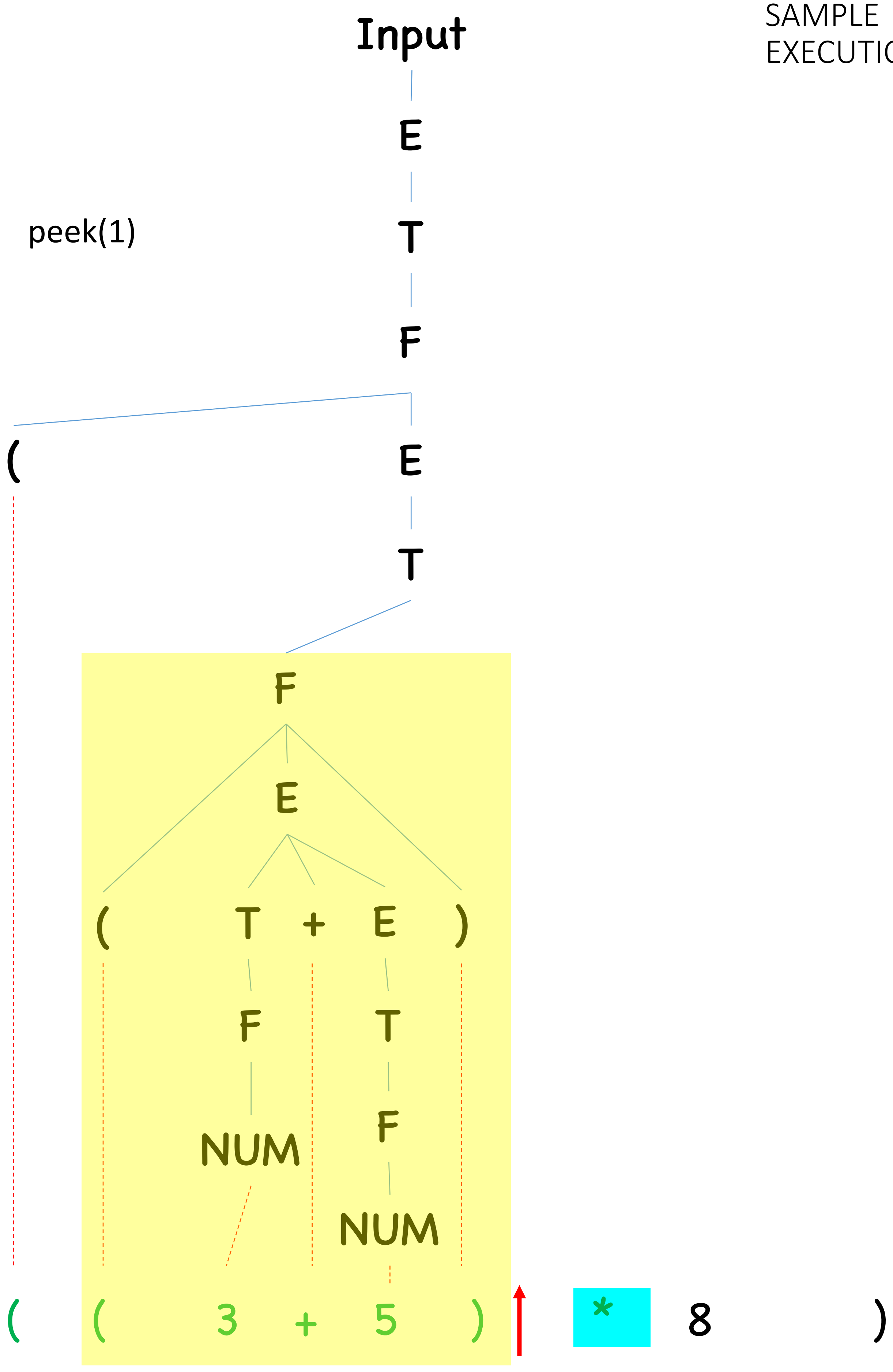


SAMPLE EXECUTION



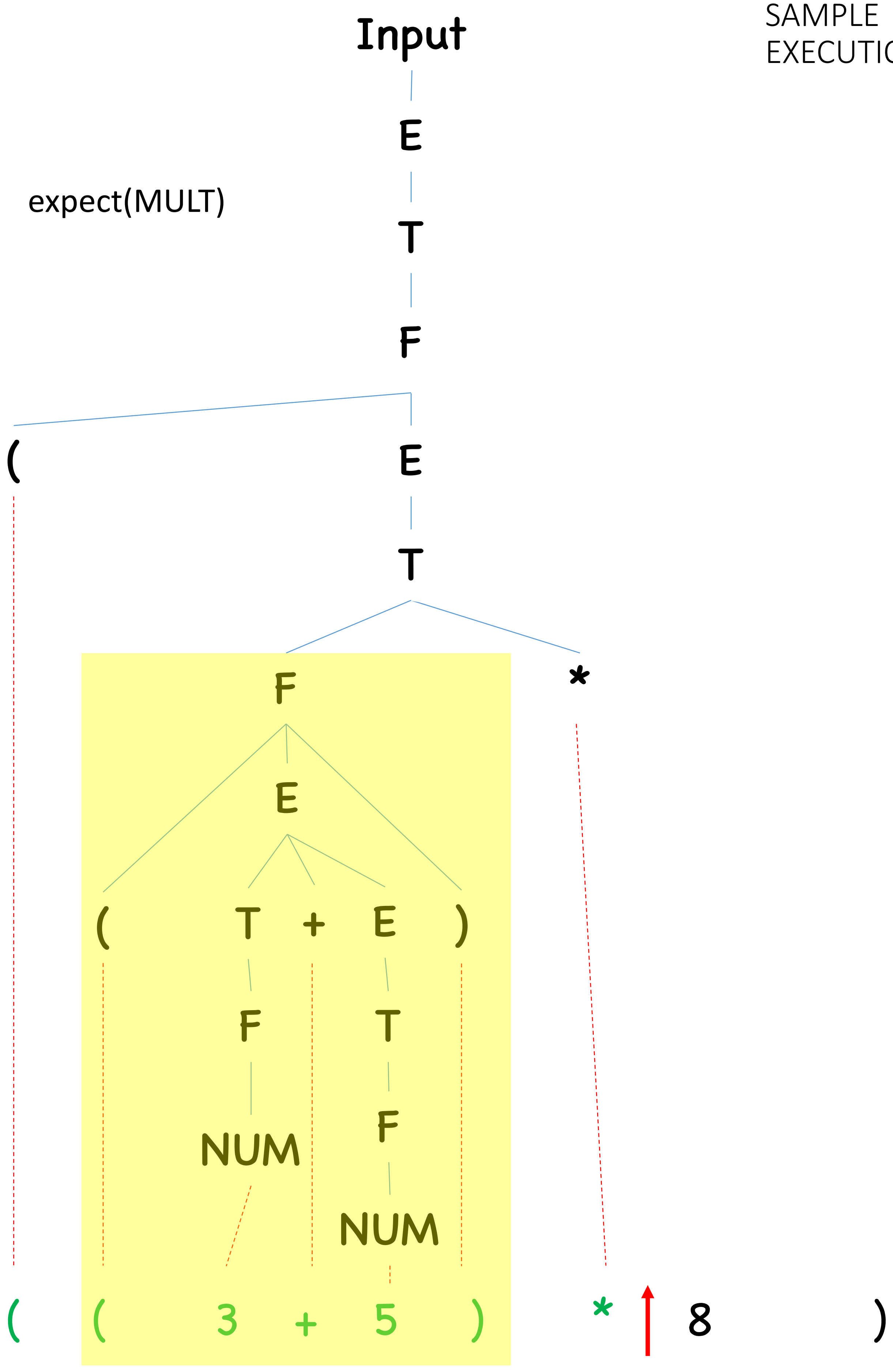
SAMPLE EXECUTION

peek(1)

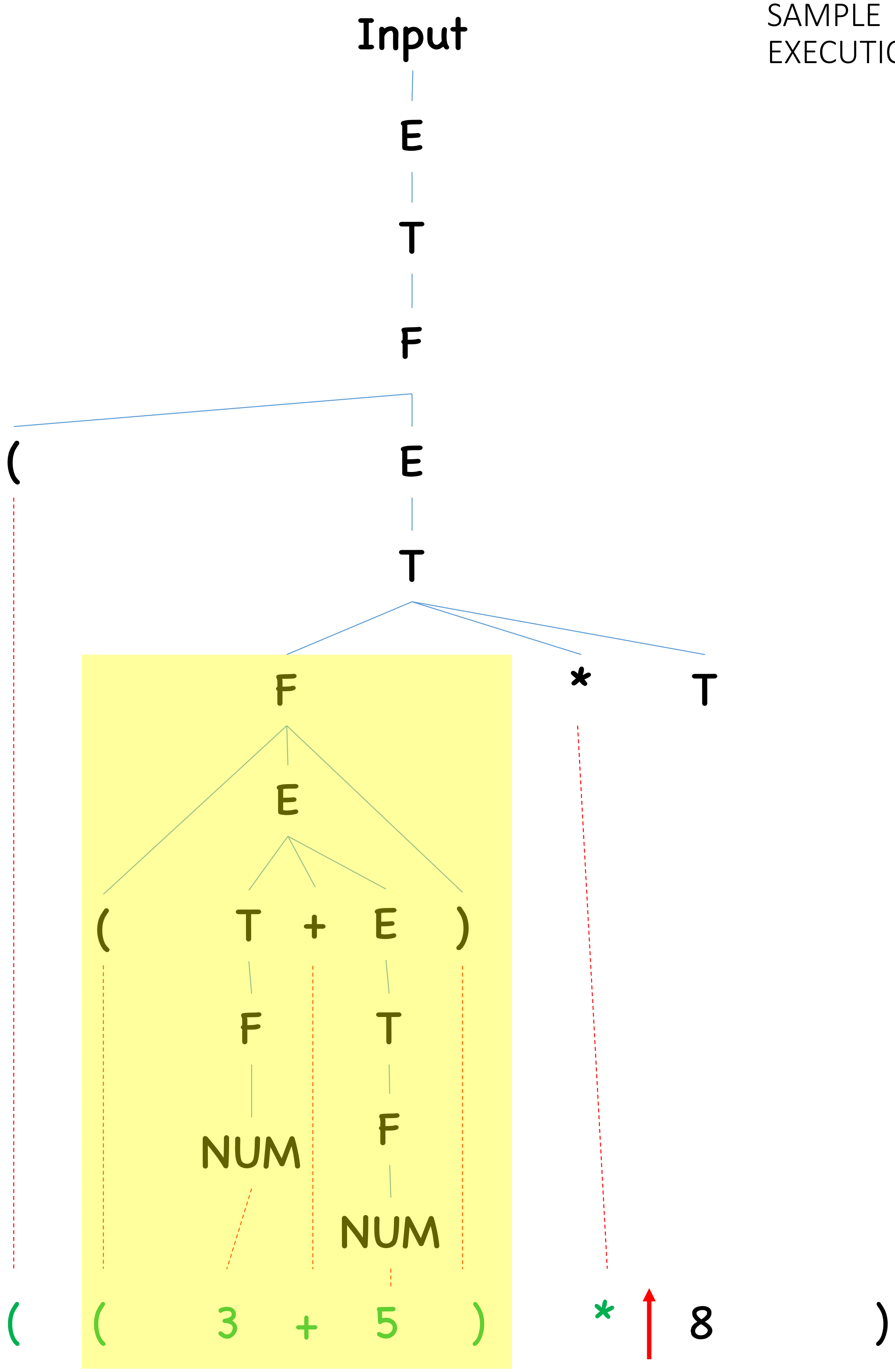


SAMPLE EXECUTION

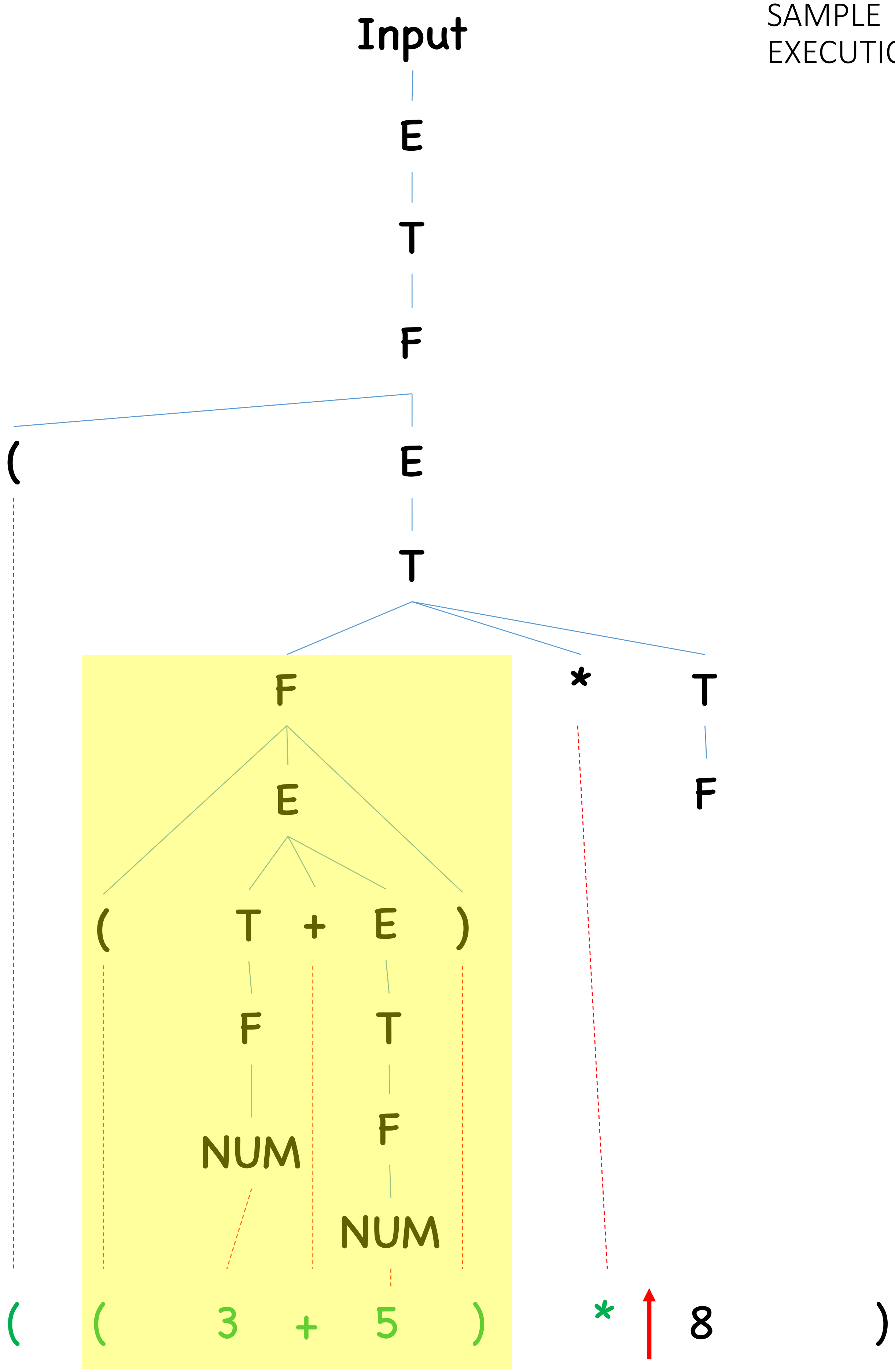
expect(MULT)



SAMPLE
EXECUTION

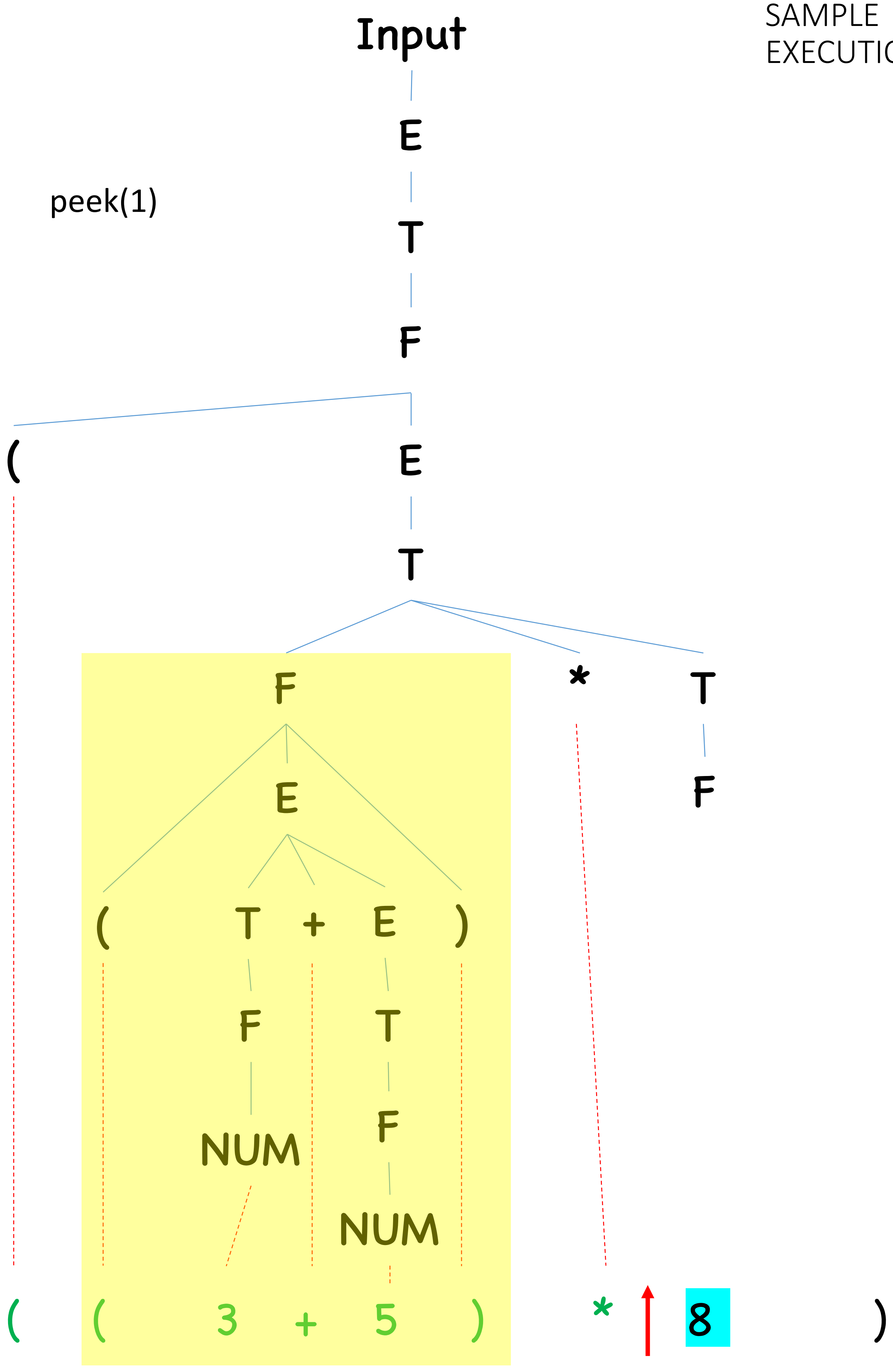


SAMPLE
EXECUTION

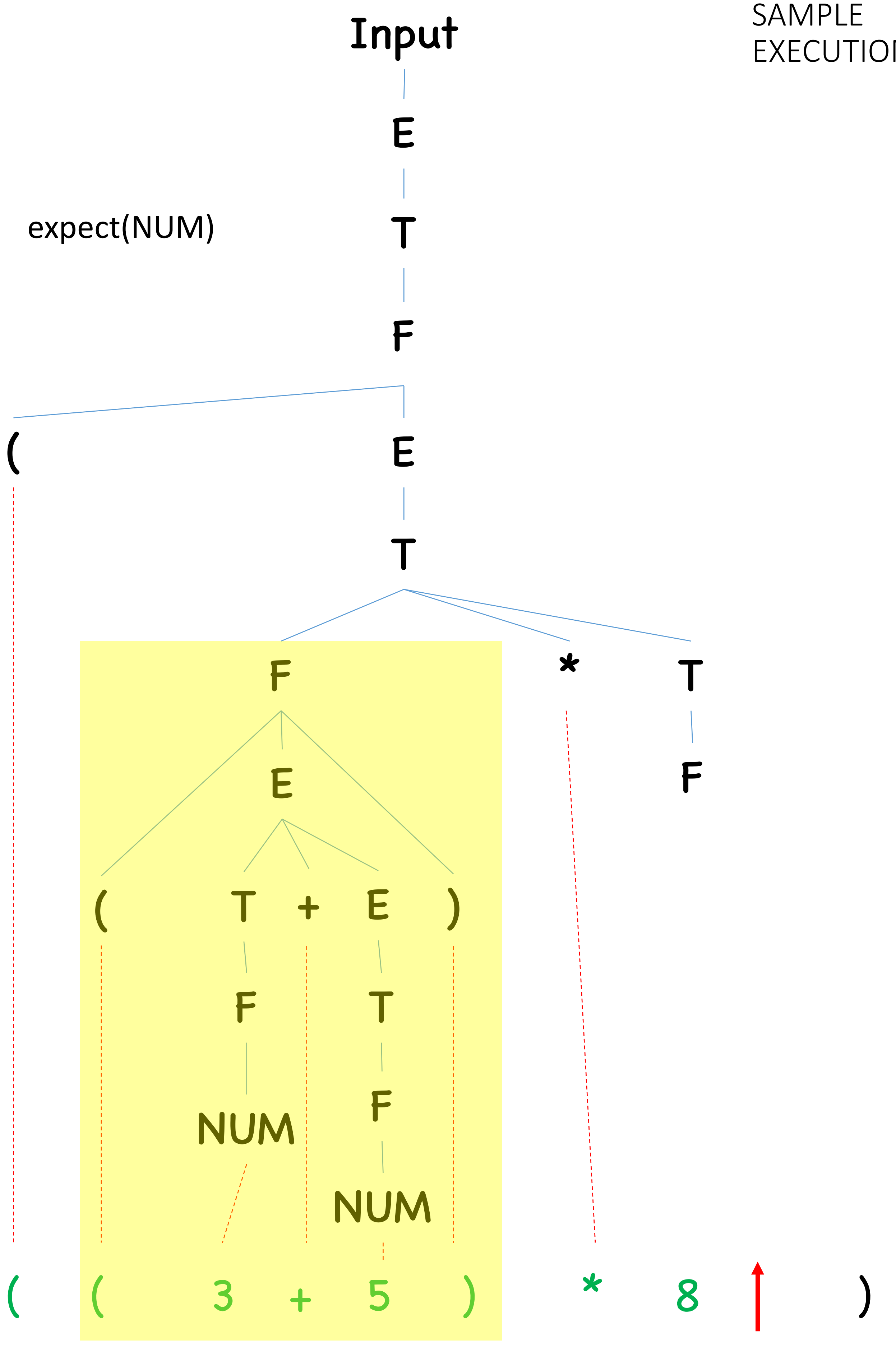


SAMPLE EXECUTION

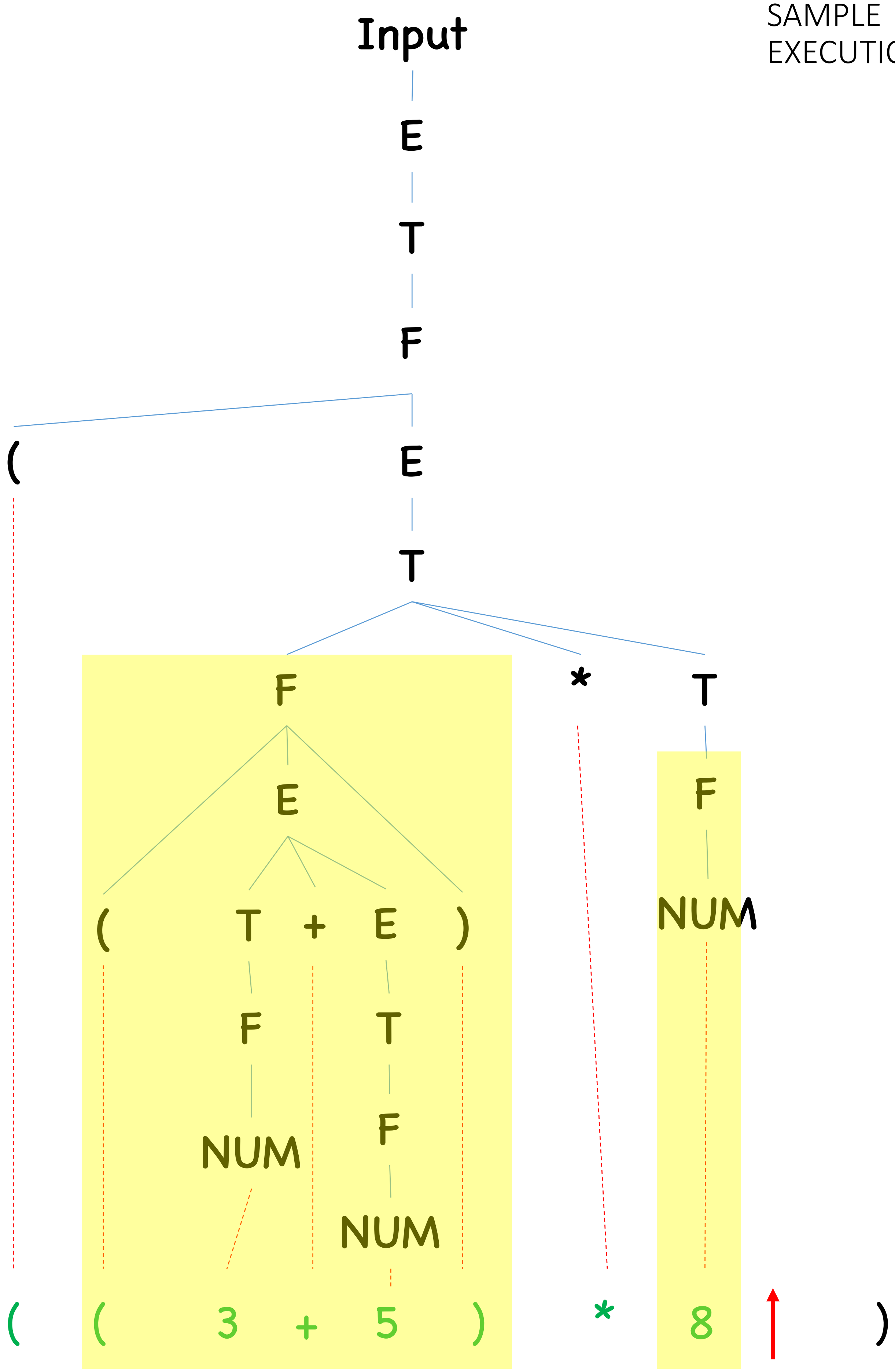
peek(1)



SAMPLE EXECUTION

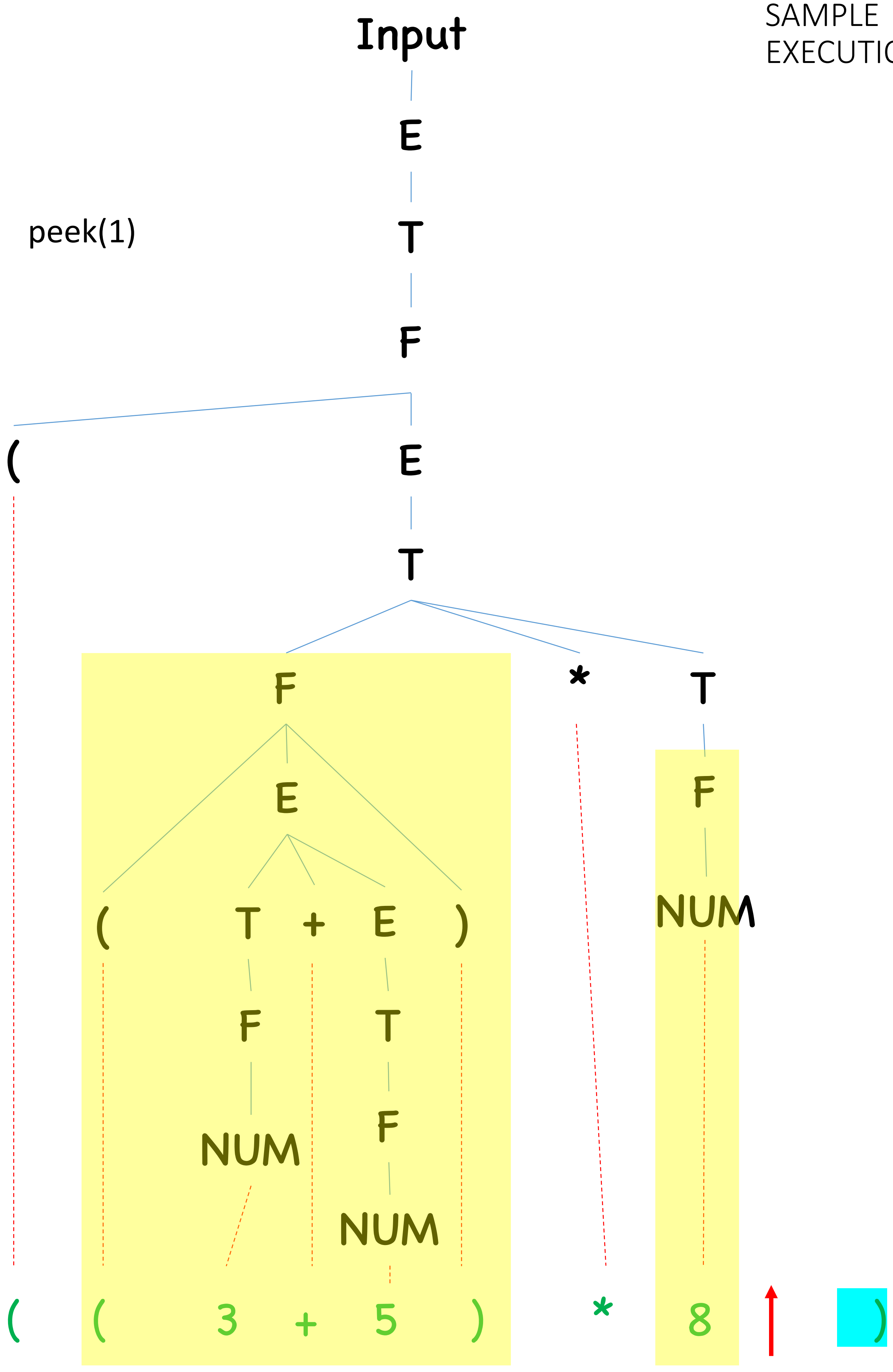


SAMPLE EXECUTION

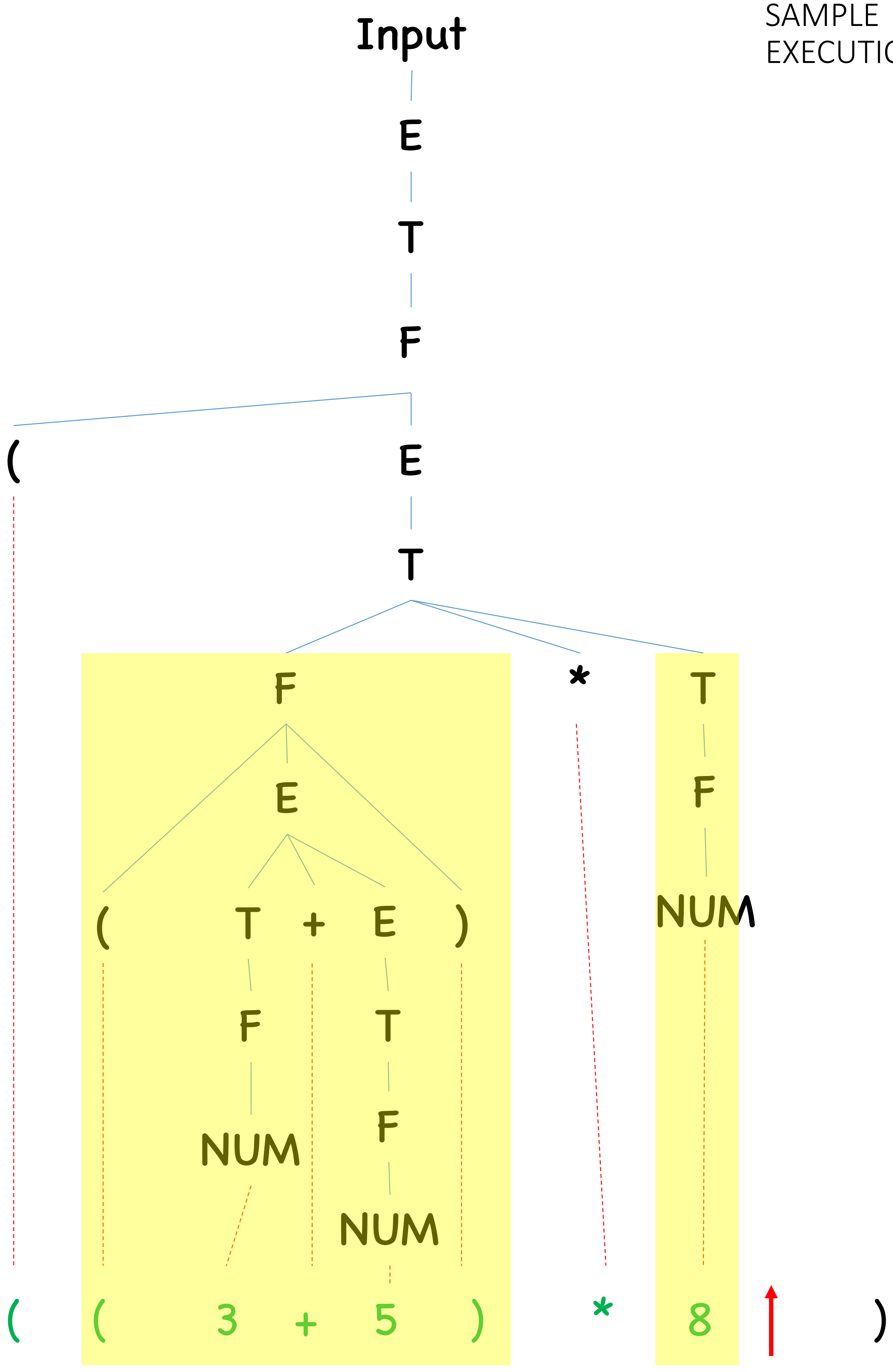


SAMPLE EXECUTION

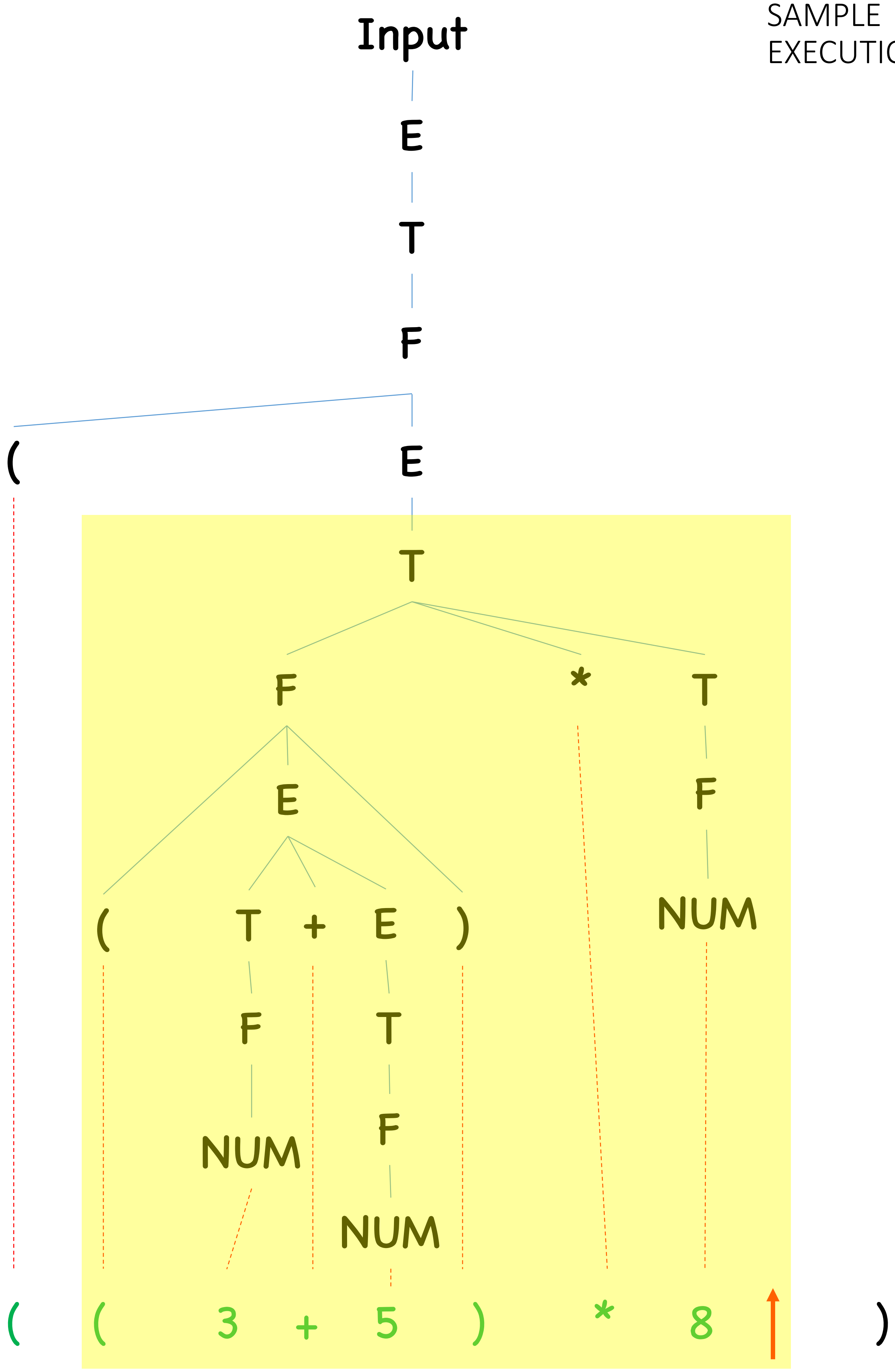
peek(1)



SAMPLE EXECUTION

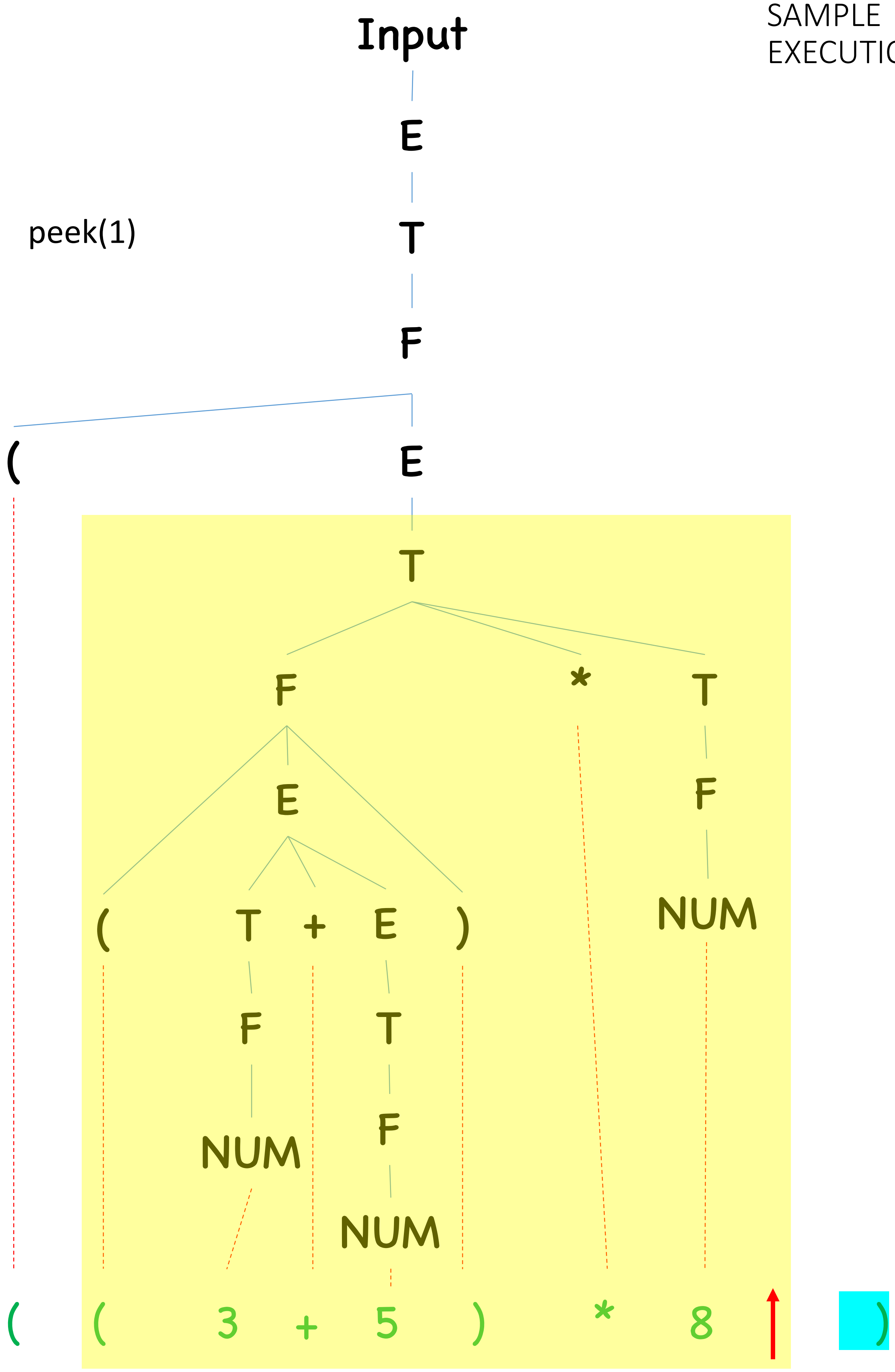


SAMPLE EXECUTION

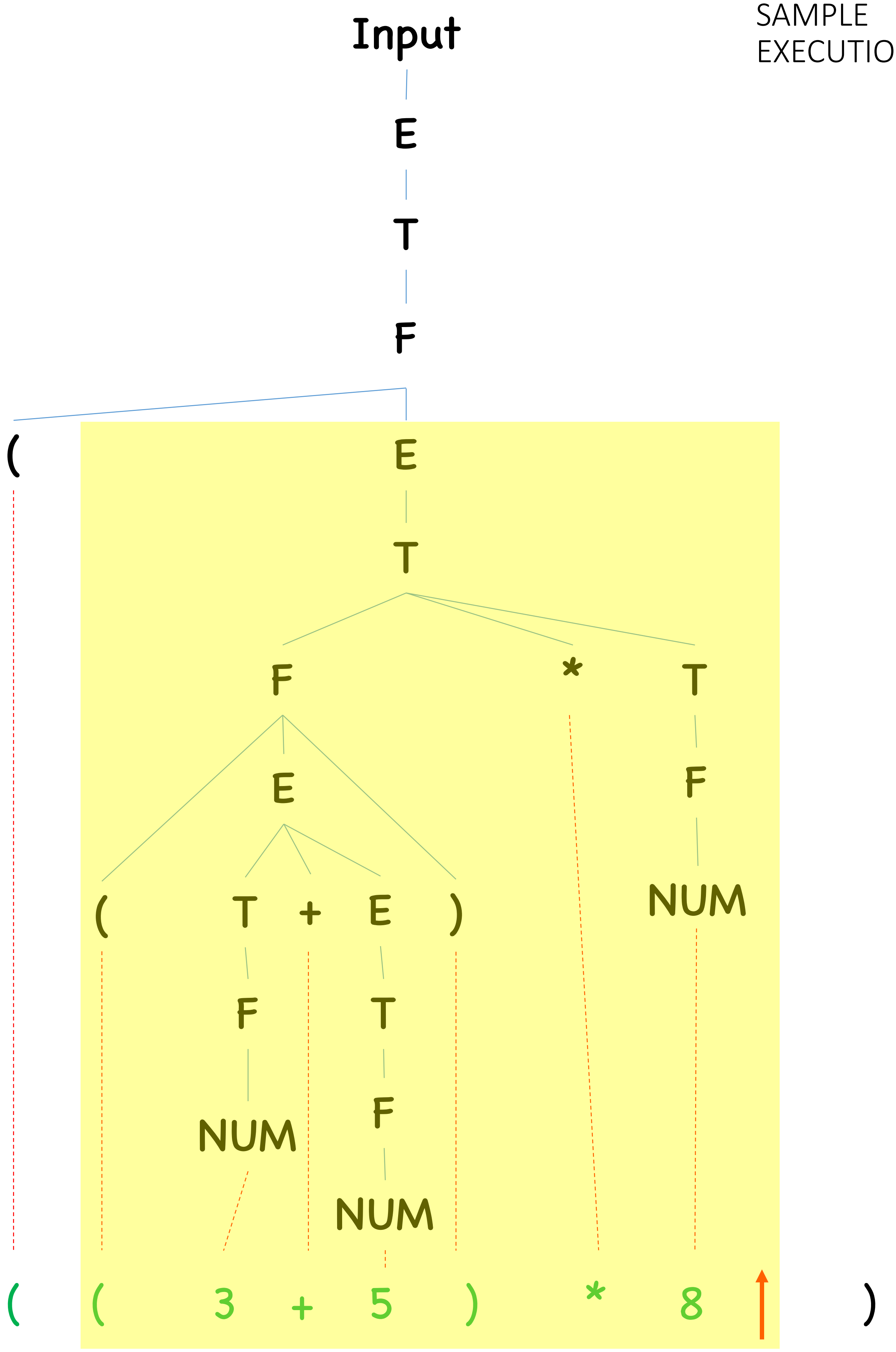


SAMPLE EXECUTION

peek(1)

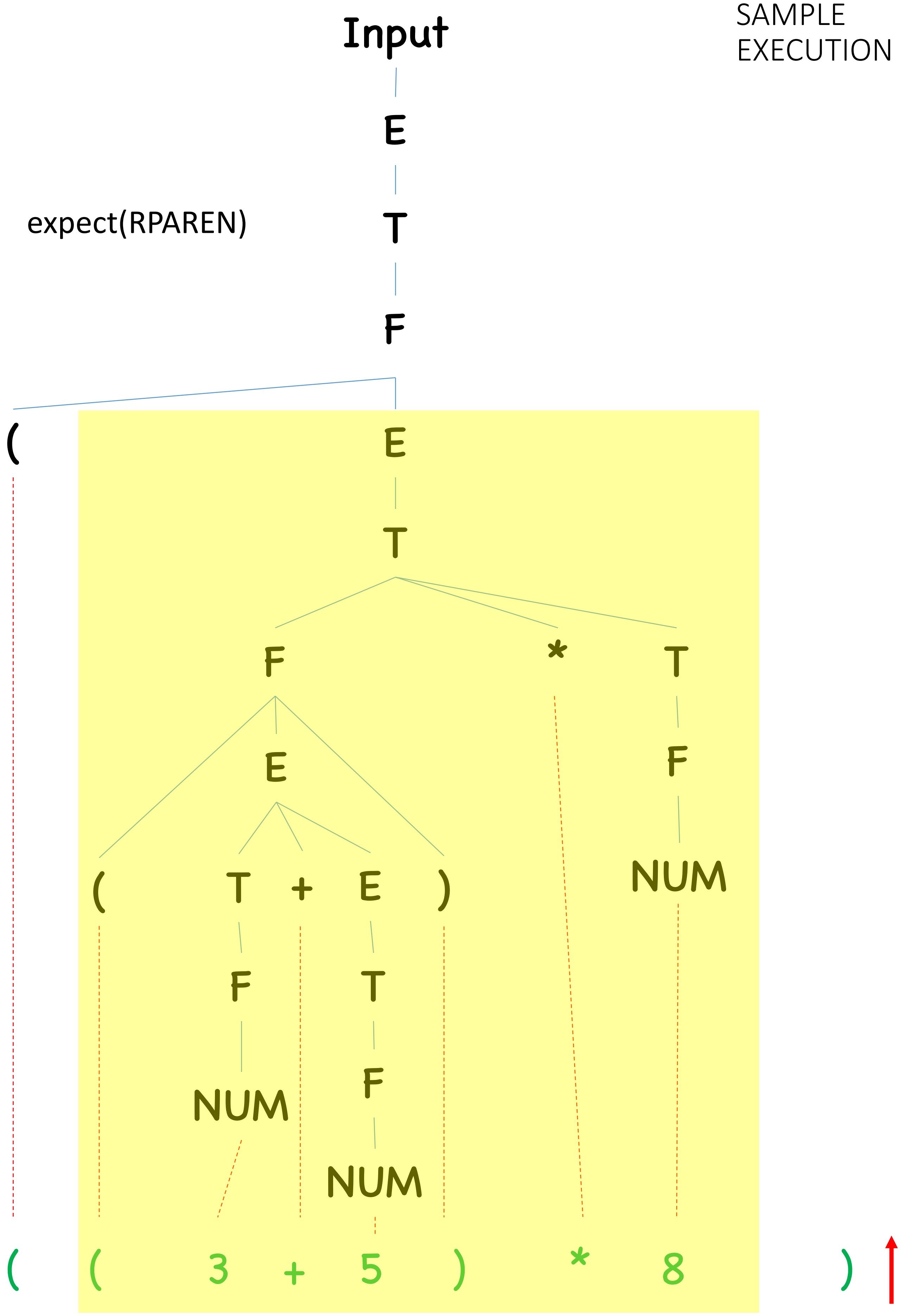


SAMPLE
EXECUTION

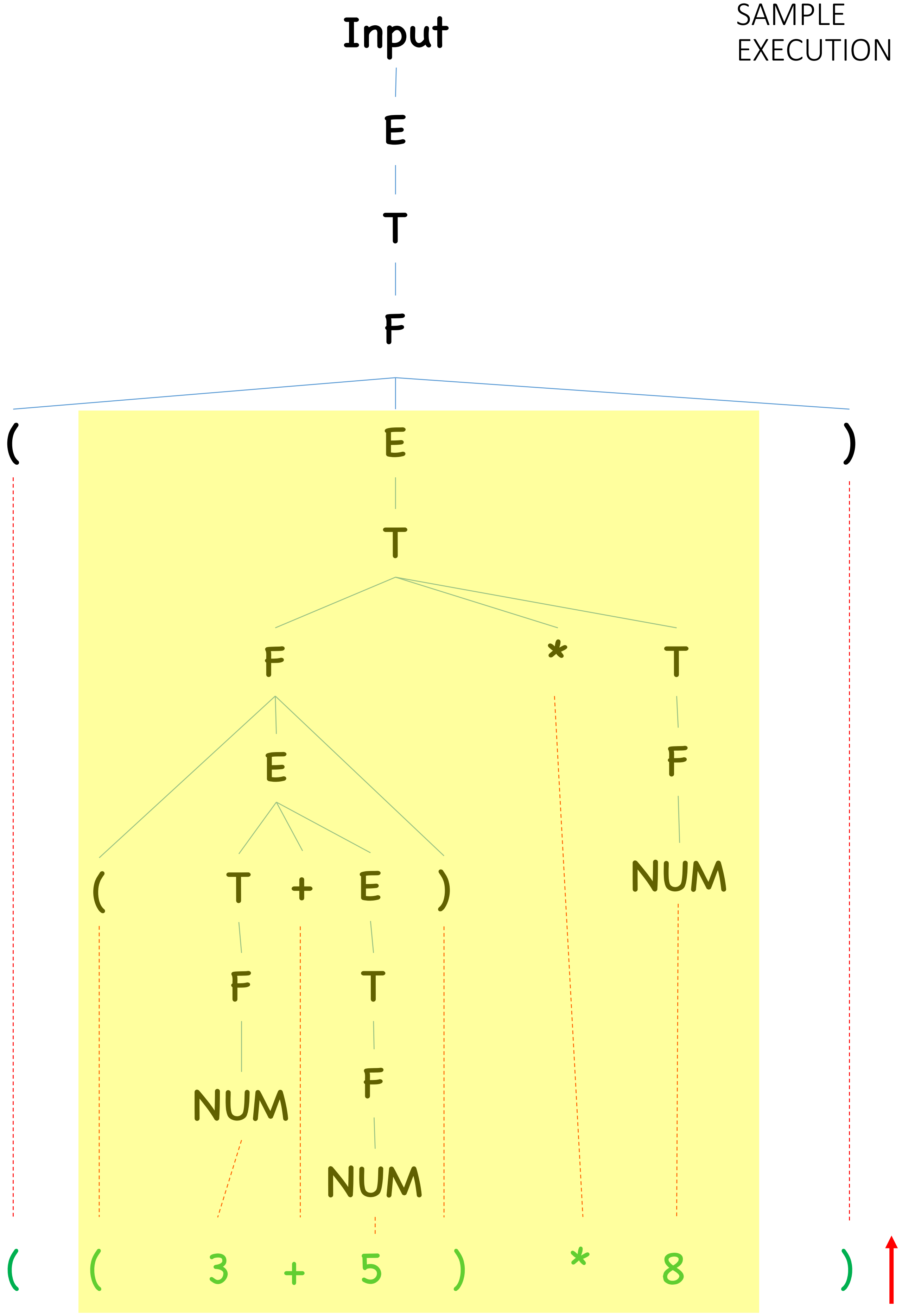


SAMPLE
EXECUTION

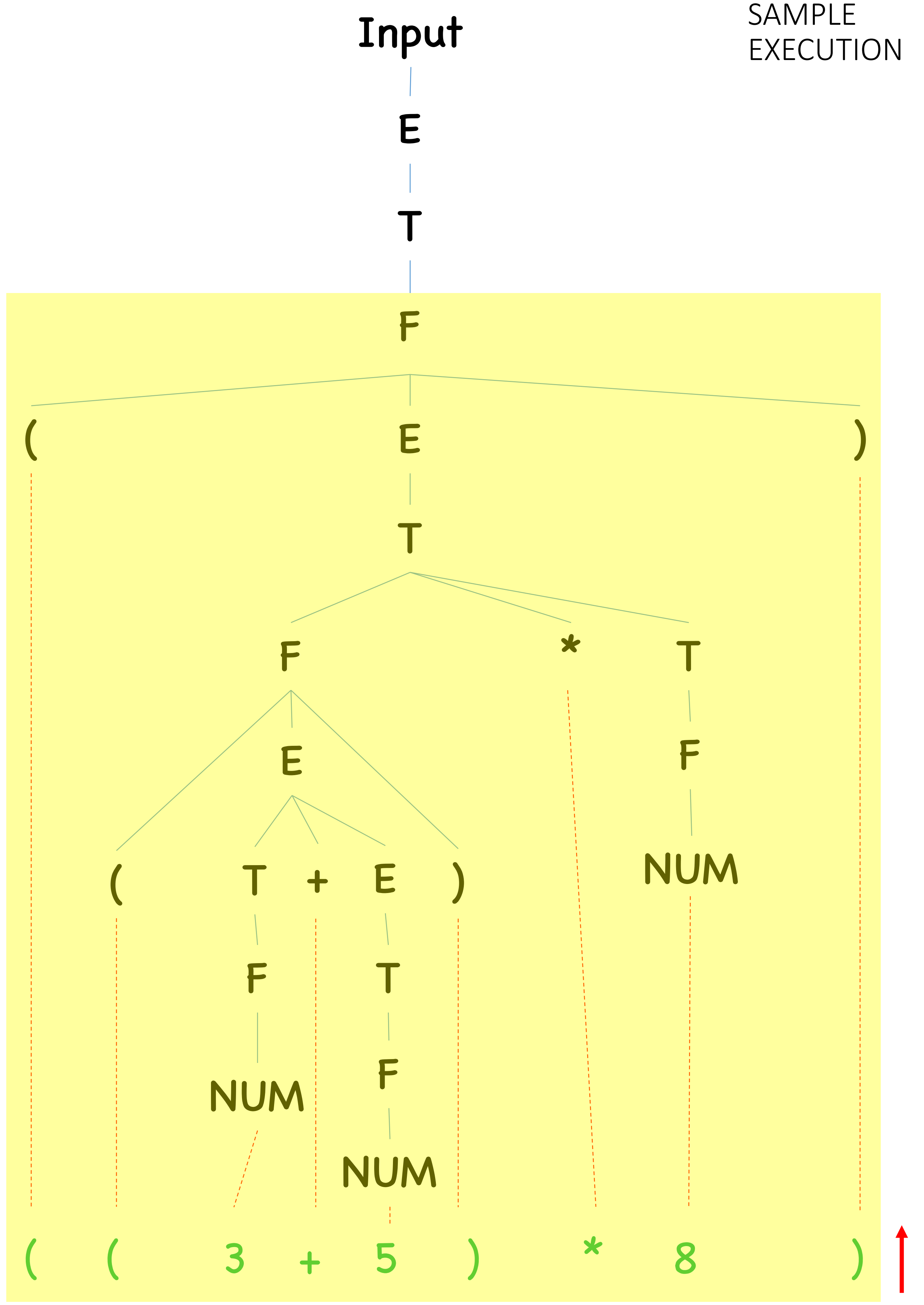
expect(RPAREN)



SAMPLE EXECUTION

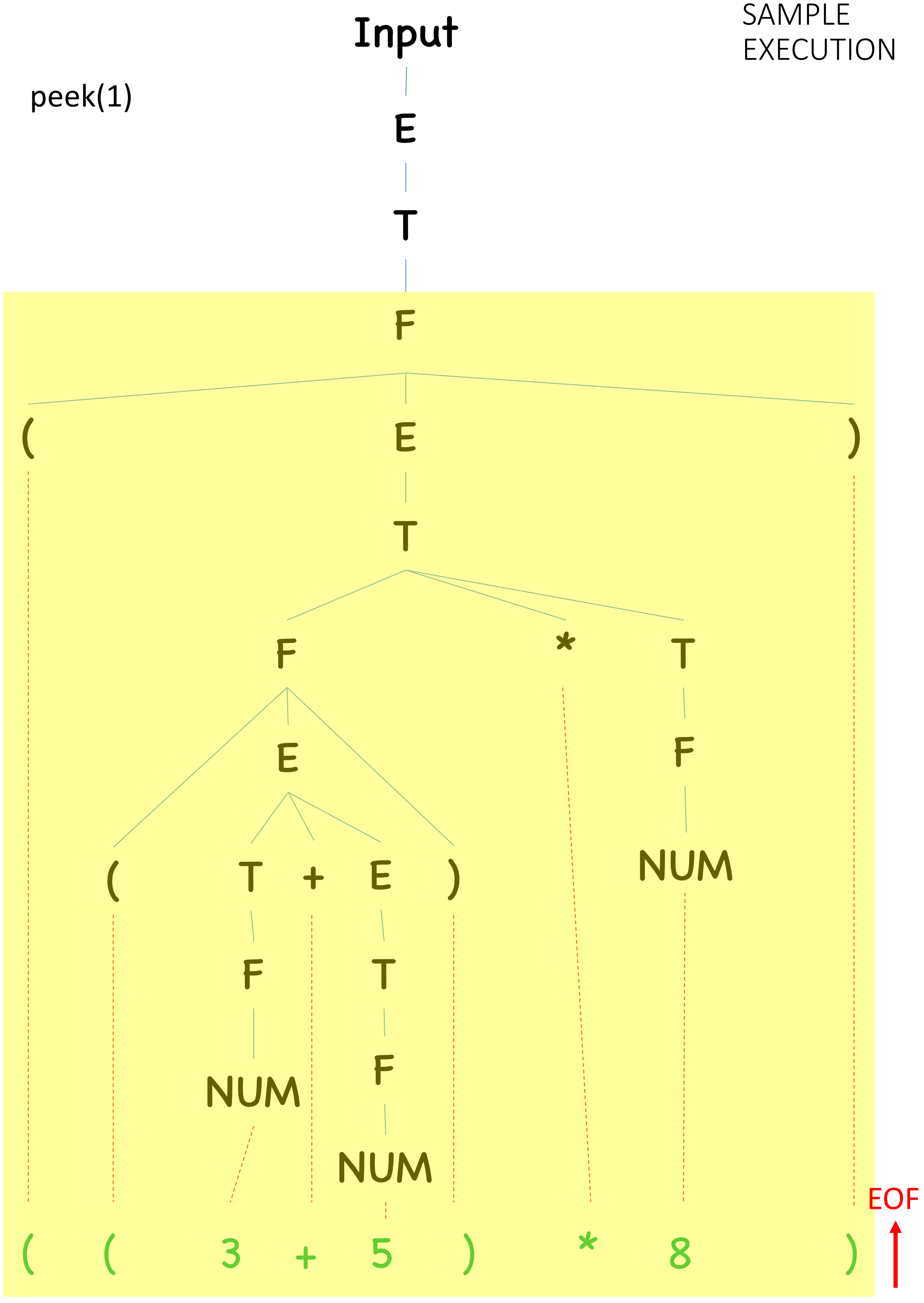


SAMPLE EXECUTION

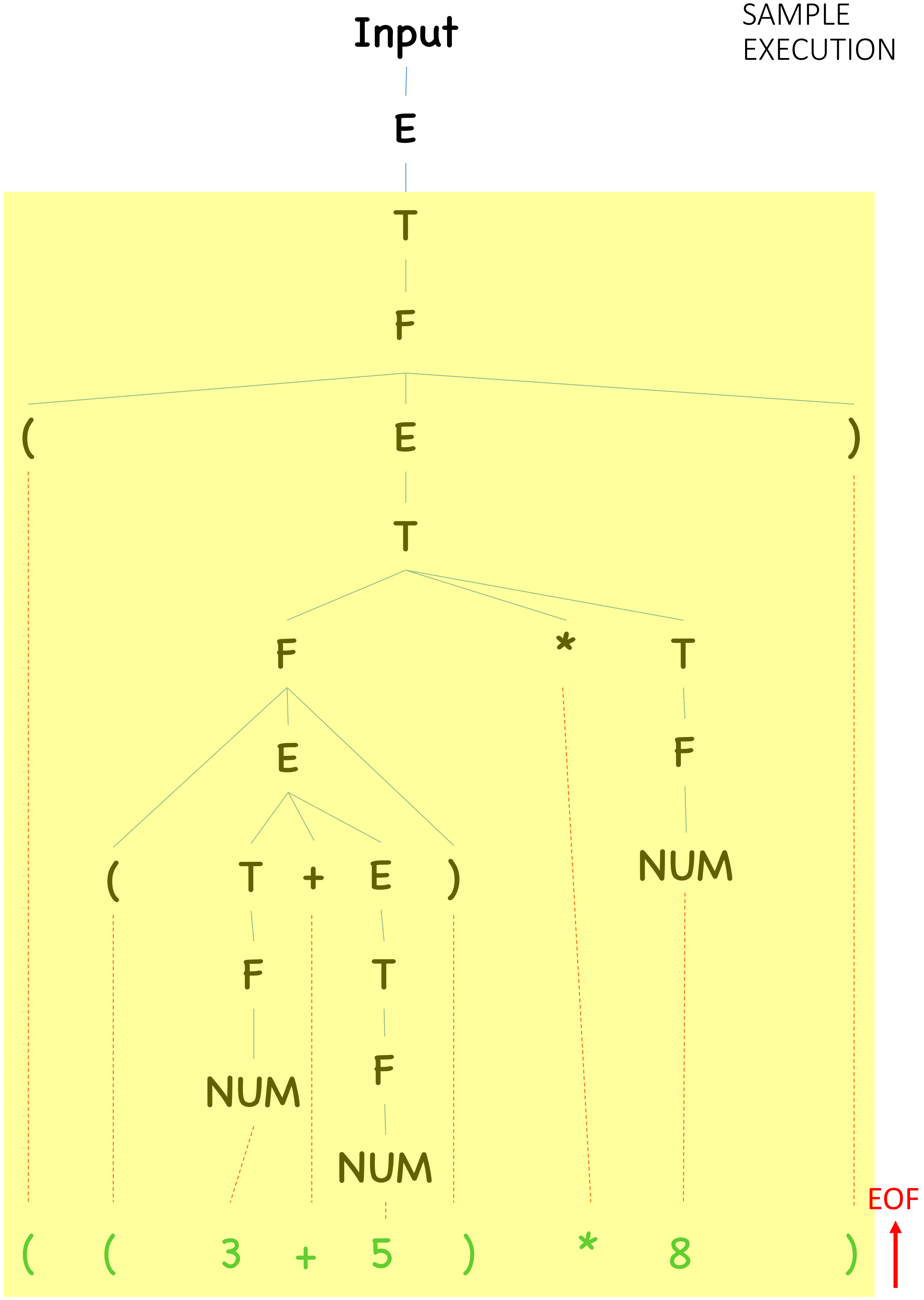


SAMPLE
EXECUTION

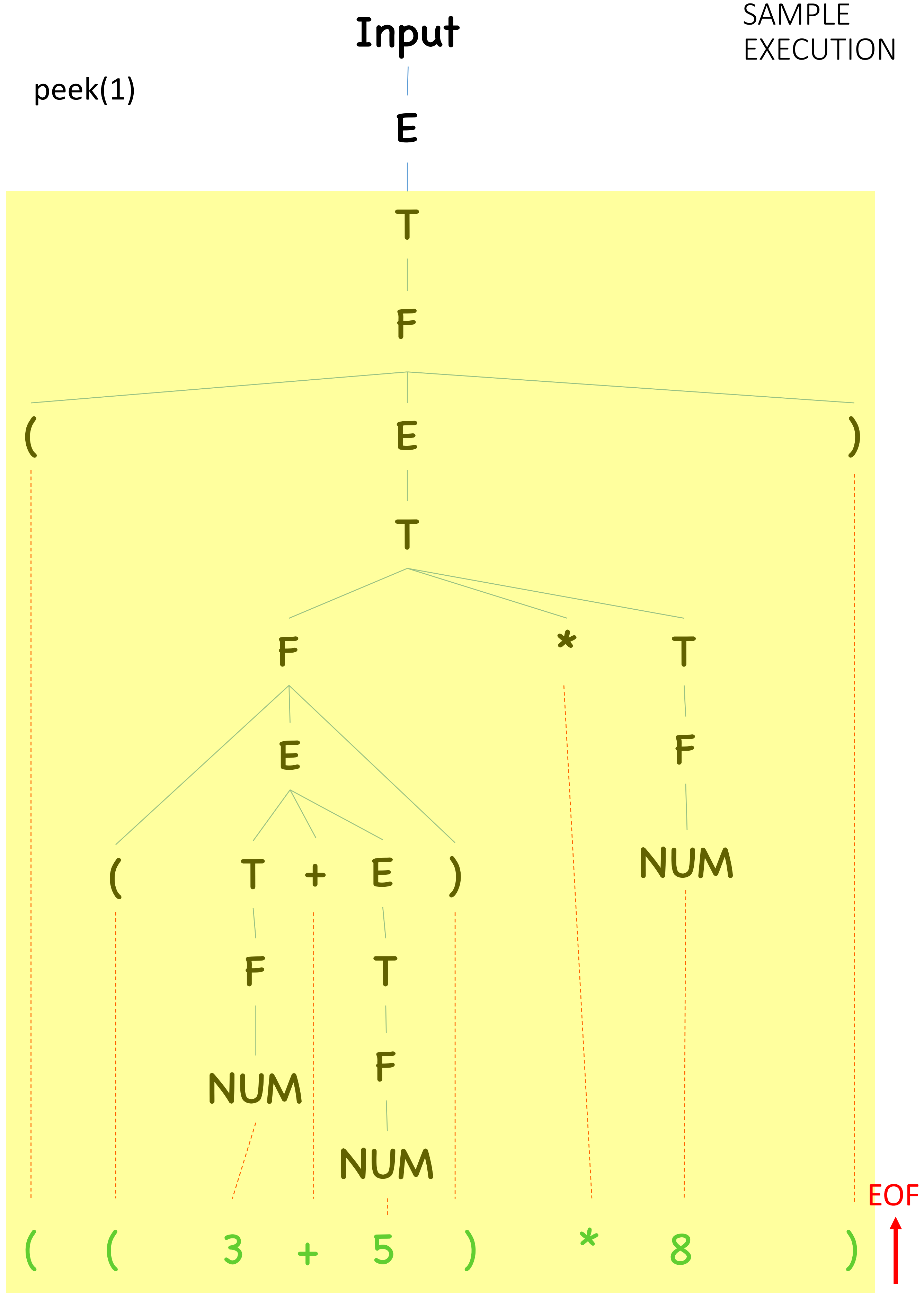
peek(1)



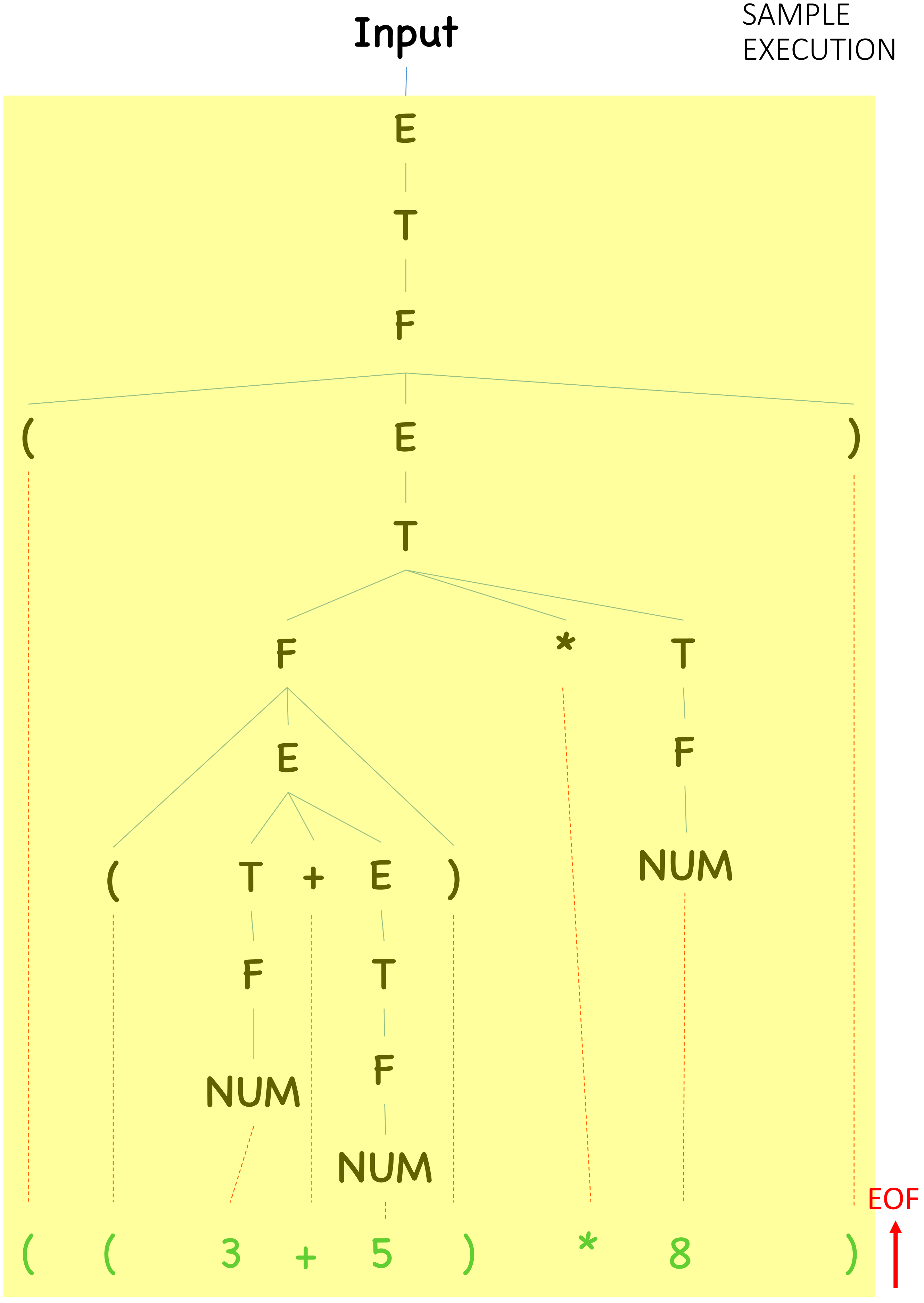
SAMPLE
EXECUTION



SAMPLE
EXECUTION



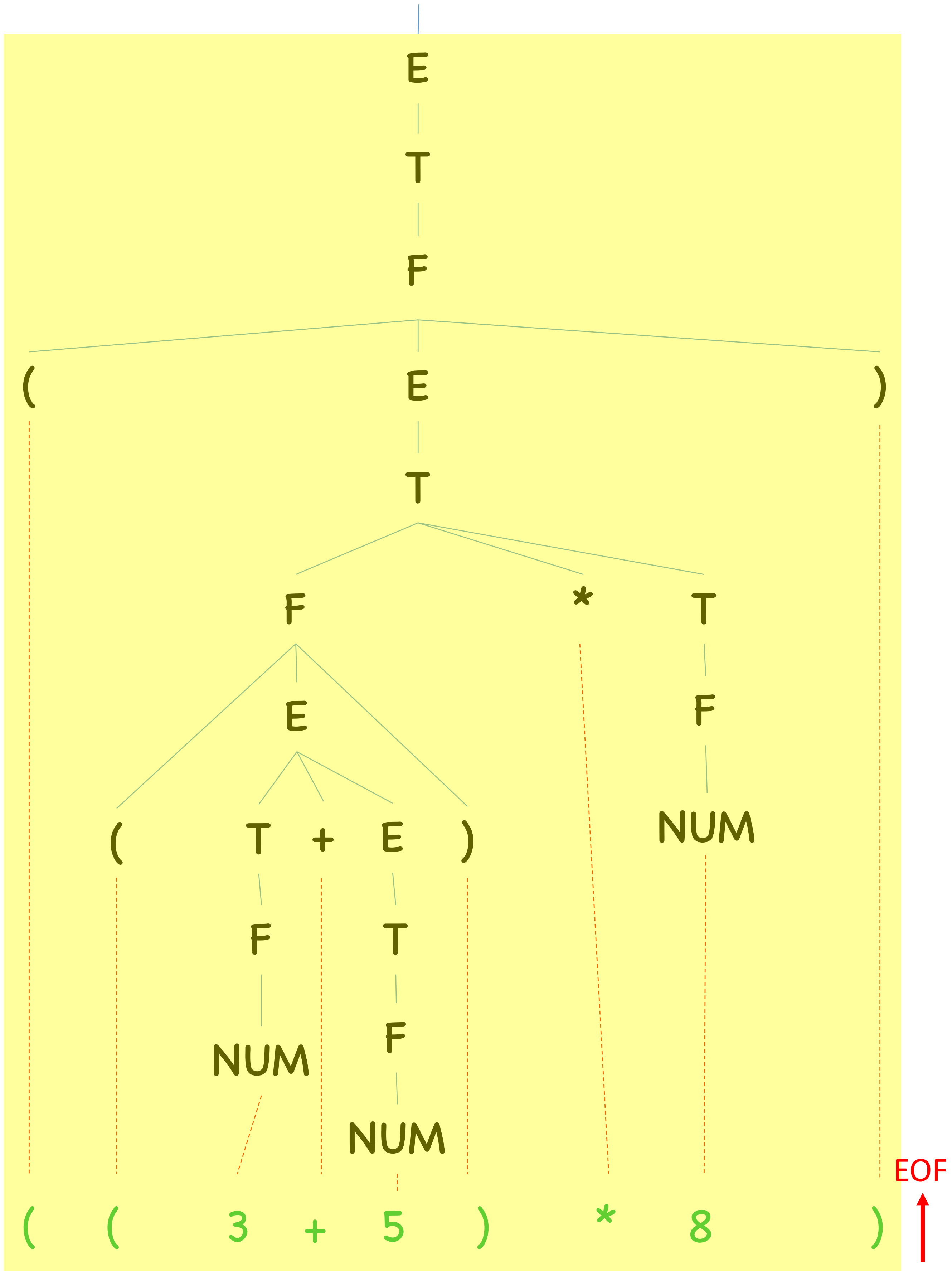
SAMPLE
EXECUTION



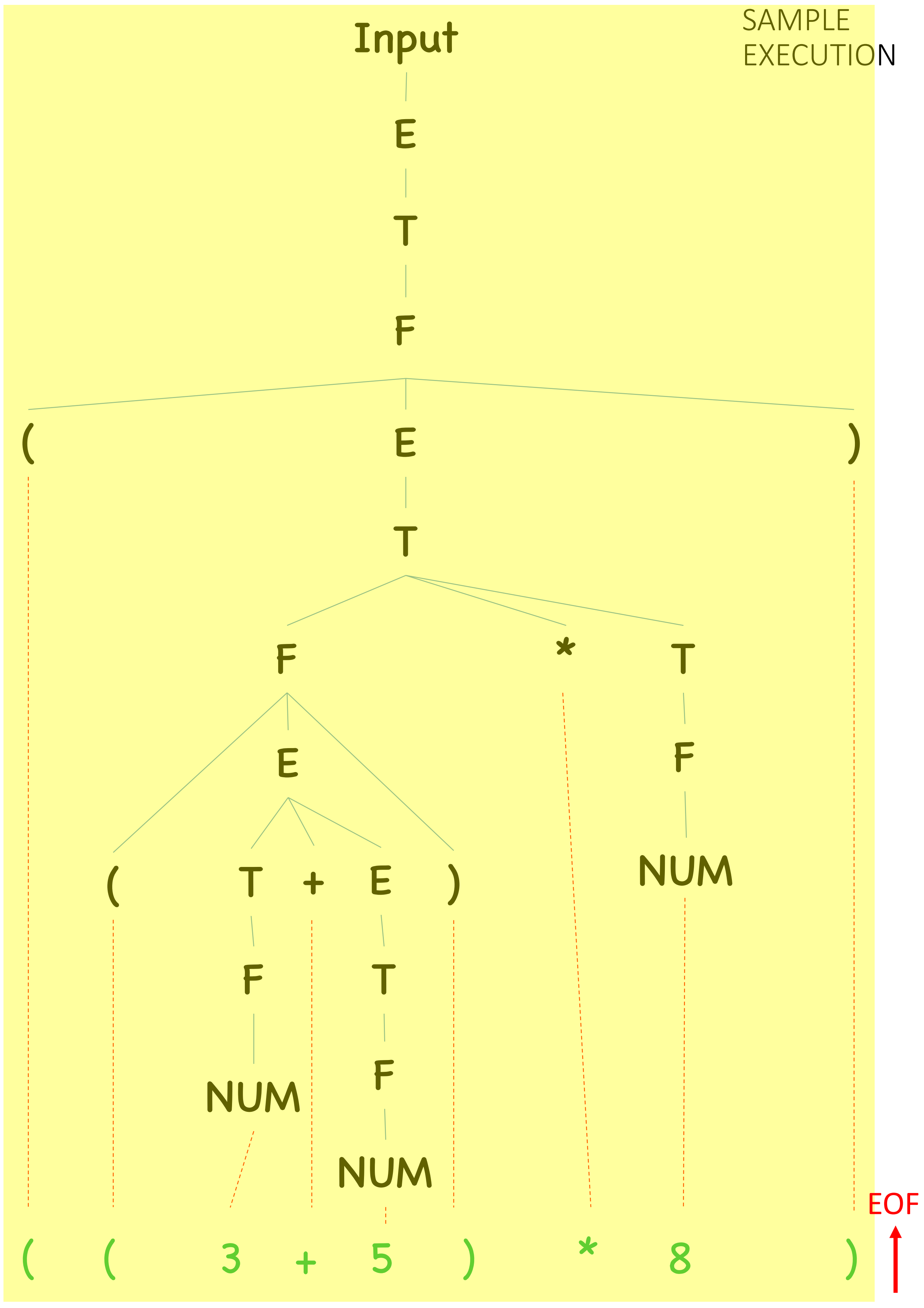
expect(EOF)

Input

SAMPLE EXECUTION



SAMPLE EXECUTION



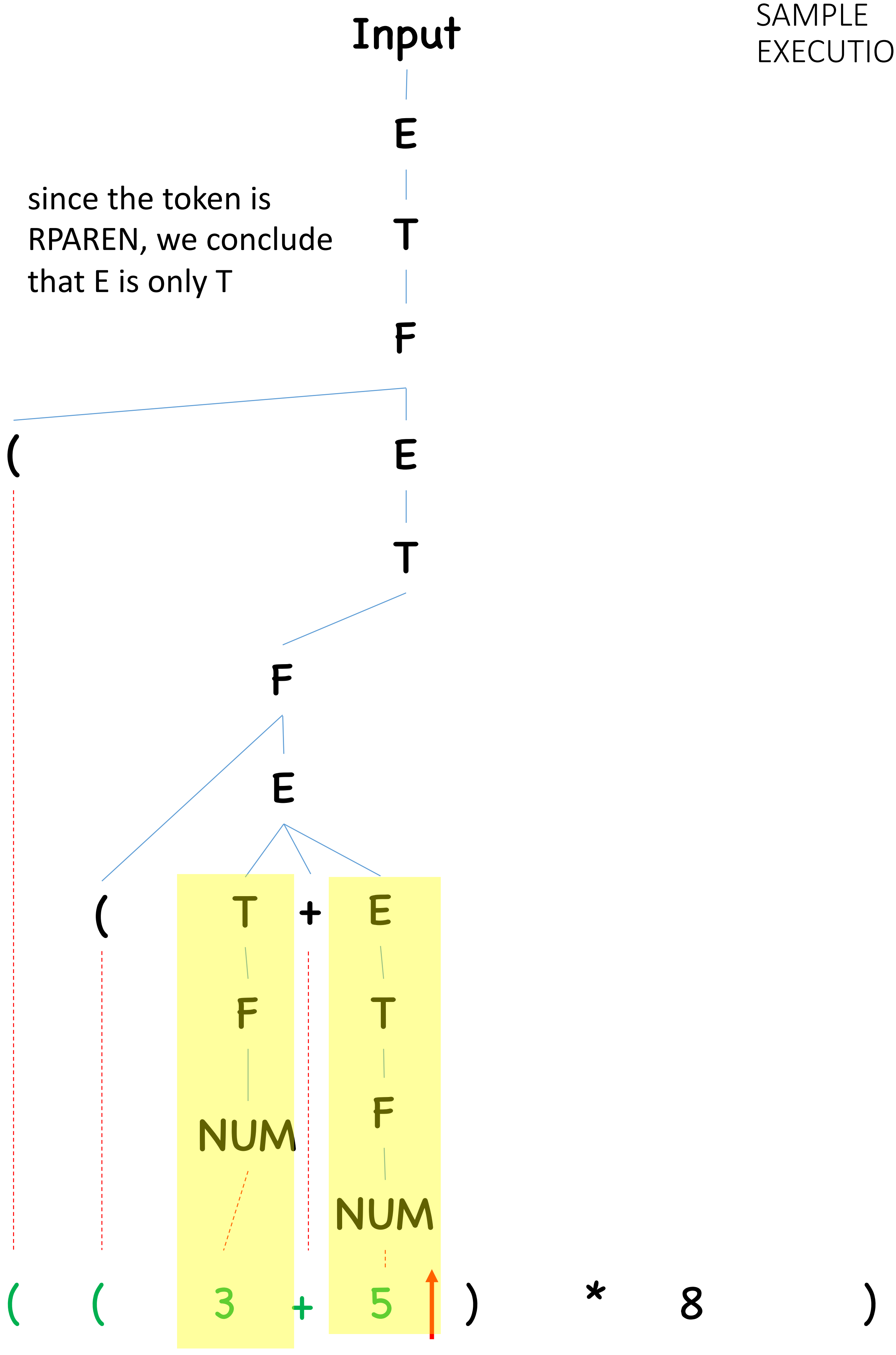
SAMPLE EXECUTION

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

```
parse_input()
```

SAMPLE EXECUTION

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



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()
                                                   T       F  parse_factor()
                                                                     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()
                                                            T       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
```

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()
                                     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()                )
```

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()
                                                   T       F  parse_factor()
                                                                     peek(1);expect(NUM); 3
                                                              peek()                  +
                                                   +   peek(1);expect(PLUS);          +
                                                       parse_expr()
                                            E                 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()
                                                   T       F  parse_factor()
                                                                     peek(1);expect(NUM); 3
                                                              peek()                  +
                                                   +   peek(1);expect(PLUS);          +
                                            E          parse_expr()
                                                              parse_term()
                                                                     parse_factor()
                                                   E       T F         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()
                                                    T        F    parse_factor()
                                                                         peek(1);expect(NUM); 3
                                                                  peek()               +
                                                    +     peek(1);expect(PLUS);        +
                                                          parse_expr()
                                        F     E                   parse_term()
                                                                         parse_factor()
                                                    E       T      F       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()
                                                 T       F   parse_factor()
                                                                    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()
                                     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()                             *
                          *   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()
                                                   T       F  parse_factor()
                                                                     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()
                                            T   F     parse_factor()
                                                            peek(1);expect(NUM); 3
                                                      peek()                      +

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

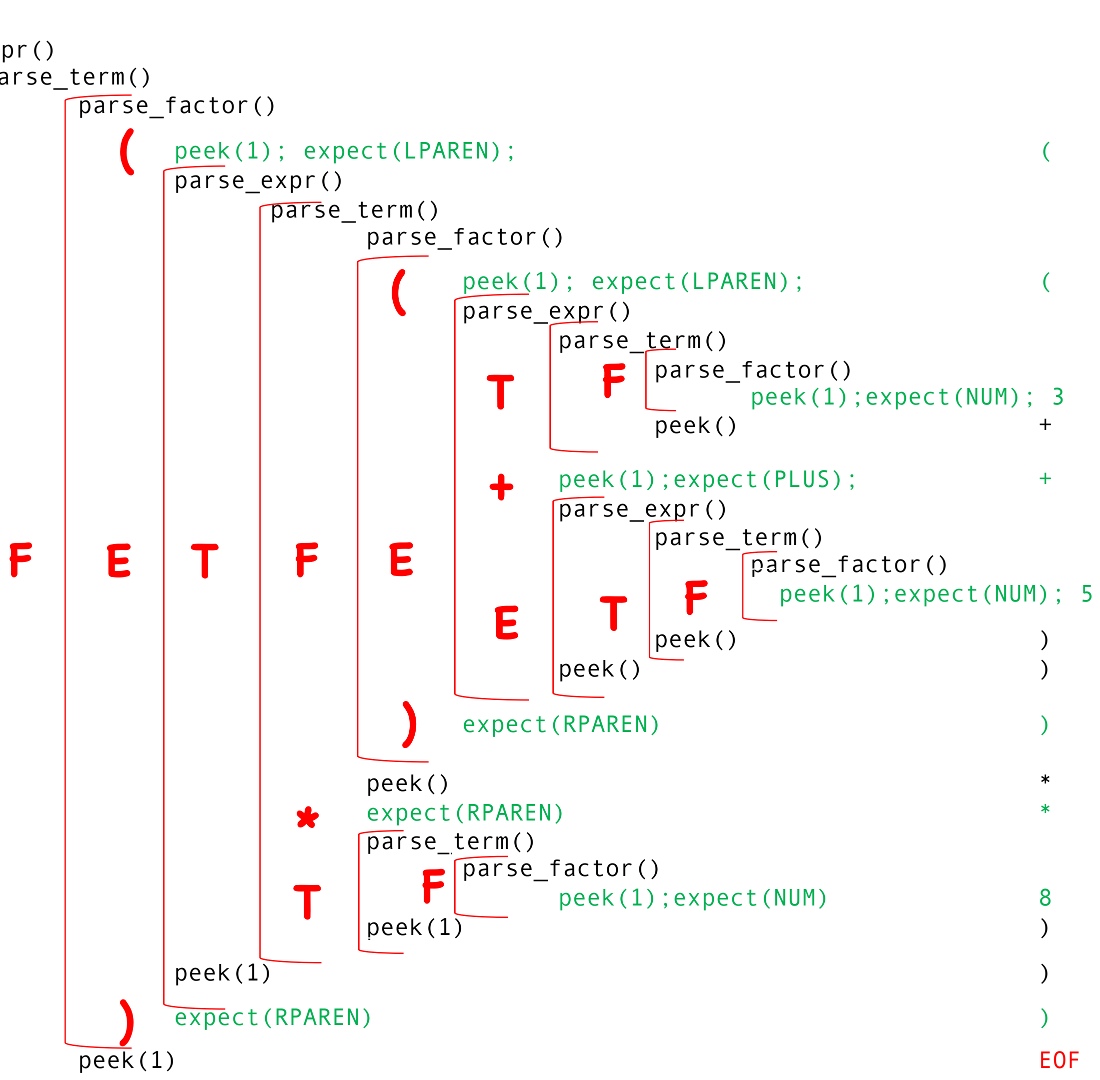
SAMPLE EXECUTION

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

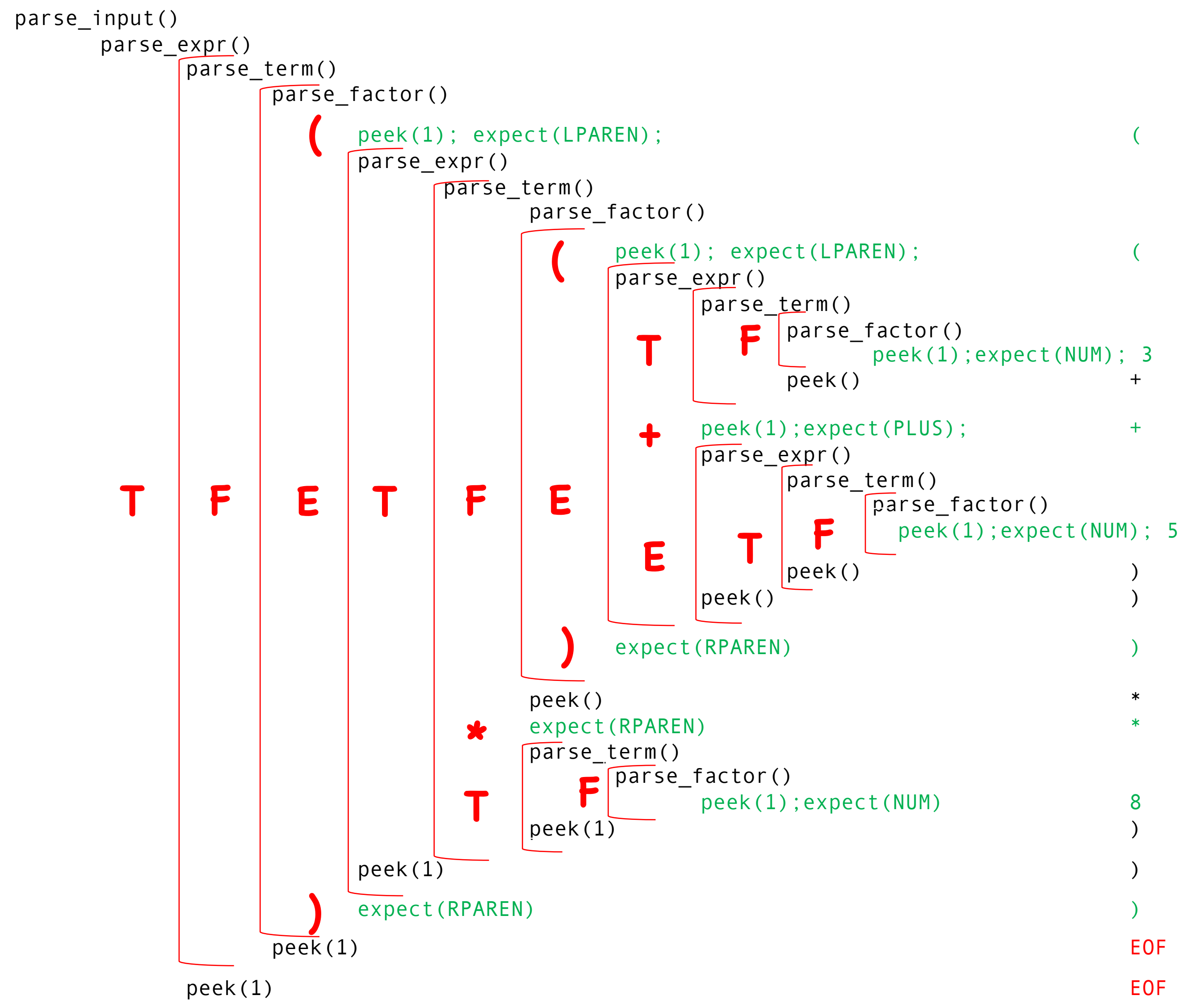
SAMPLE EXECUTION

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

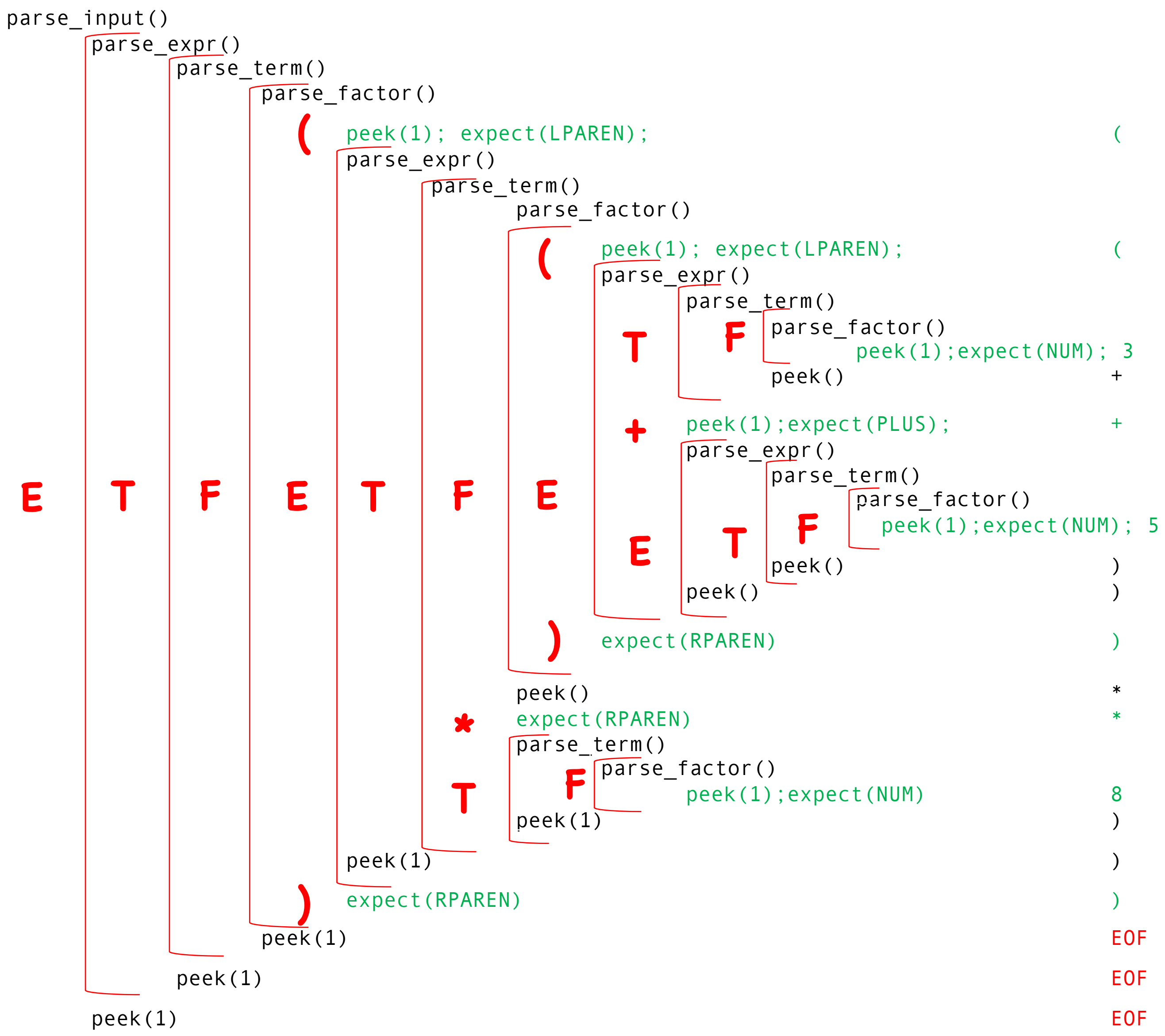
SAMPLE EXECUTION

((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)

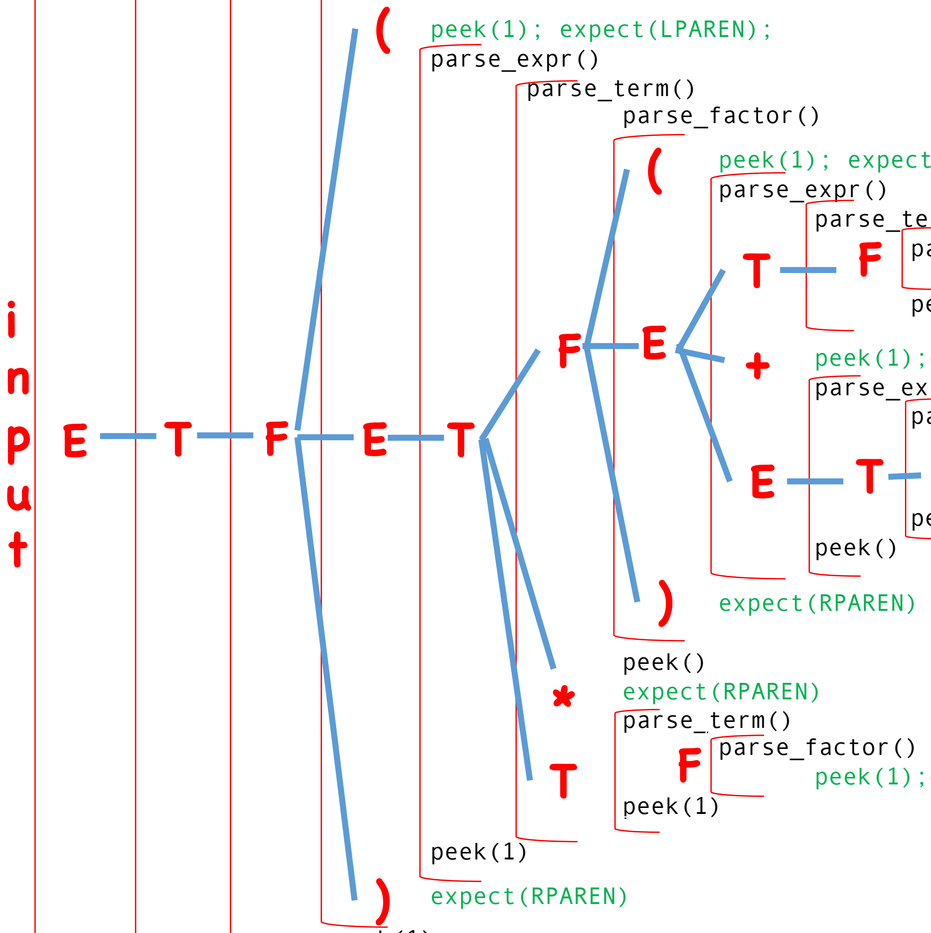


```
parse_input()
       parse_expr()
              parse_term()
                     parse_factor()
                            peek(1); expect(LPAREN);                              (
                            parse_expr()
                                   parse_term()
                          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

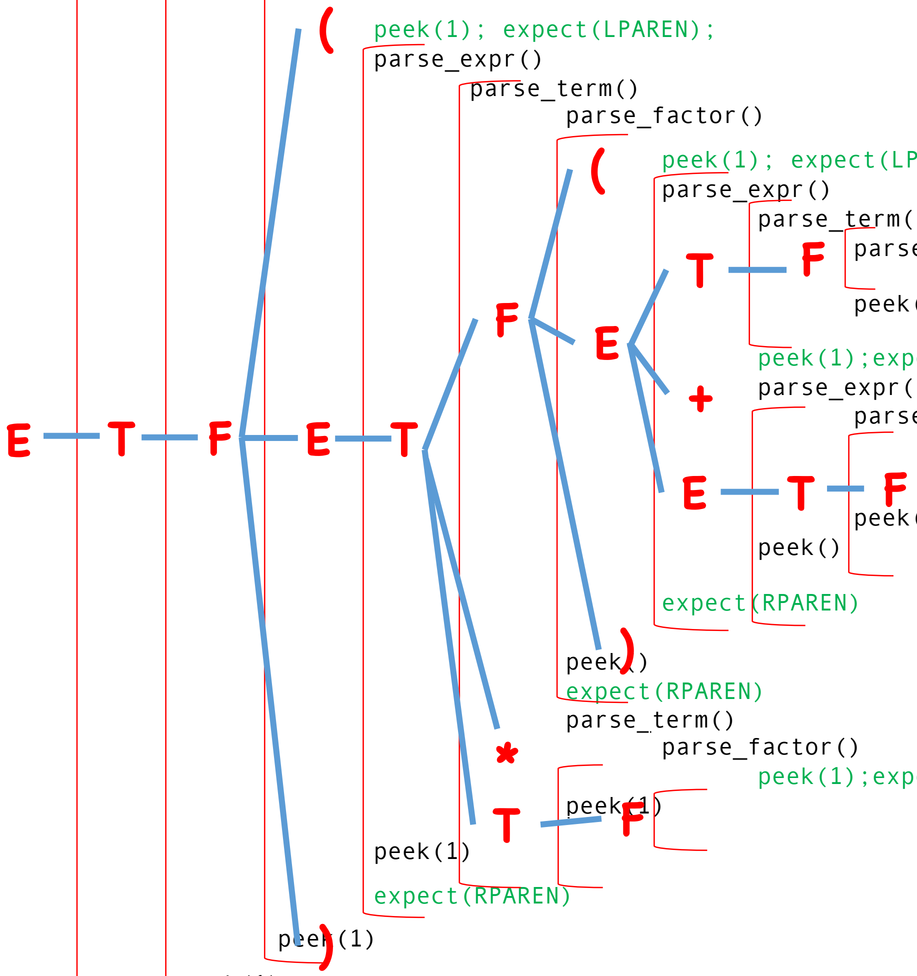
```
parse_input()
          parse_expr()
                    parse_term()
                              parse_factor()
                                        peek(1); expect(LPAREN);                         (
                                        parse_expr()
                                     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


E
T
F
(
E
T
F
(
E
T
F
+
E
T
F
)
*
T
F
)