# EECS3311 – Software Design

## Visitor Pattern

## A CFG for Arithmetic Expressions

An example grammar that generates strings representing arithmetic expressions with the four operators +, -, *, /, and numbers as operands is:

1. <expression> --> number
2. <expression> --> ( <expression> )
3. <expression> --> <expression> + <expression>
4. <expression> --> <expression> - <expression>
5. <expression> --> <expression> * <expression>
6. <expression> --> <expression> / <expression>

The only nonterminal symbol in this grammar is <expression>, which is also the start symbol. The terminal symbols are {+,-,*,/, (,),number}. (We will interpret "number" to represent any valid number.)

The first rule (or production) states that an <expression> can be rewritten as (or replaced by) a number. In other words, a number is a valid expression.

The second rule says that an <expression> enclosed in parentheses is also an <expression>. Note that this rule defines an expression in terms of expressions, an example of the use of recursion in the definition of context-free grammars.

The remaining rules say that the sum, difference, product, or division of two <expression>s is also an expression.

# A simple grammar

$$v ::= v\delta \mid \delta$$
$$\delta ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

$$\mathbf{V}: \mathbf{Num} \rightarrow \mathbf{Int}$$
$$\mathbf{V}[\![v\delta]\!] = 10 \times \mathbf{V}[\![v]\!] + \mathbf{V}[\![\delta]\!]$$
$$\mathbf{V}[\![0]\!] = 0 \quad \mathbf{V}[\![1]\!] = 1$$
$$\mathbf{V}[\![2]\!] = 2 \quad \mathbf{V}[\![3]\!] = 3$$
$$\mathbf{V}[\![4]\!] = 4 \quad \mathbf{V}[\![5]\!] = 5$$
$$\mathbf{V}[\![6]\!] = 6 \quad \mathbf{V}[\![7]\!] = 7$$
$$\mathbf{V}[\![8]\!] = 8 \quad \mathbf{V}[\![9]\!] = 9$$

$$\begin{aligned}\mathbf{V}[\![123]\!] &= 10 \times \mathbf{V}[\![12]\!] + 3 \\ &= 10 \times (10 \times \mathbf{V}[\![1]\!] + 2) + 3 \\ &= 123\end{aligned}$$

# A BNF grammar is a way to express a Context Free Grammar

```
expression = term | expression, "+" , term;
term       = factor | term, "*" , factor;
factor     = constant | variable | "(" , expression , ")";
variable   = "x" | "y" | "z";
constant   = digit , {digit};
digit      = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9";
```

- Each rule is called a production
- Green is non-terminal
- In quotes are terminals

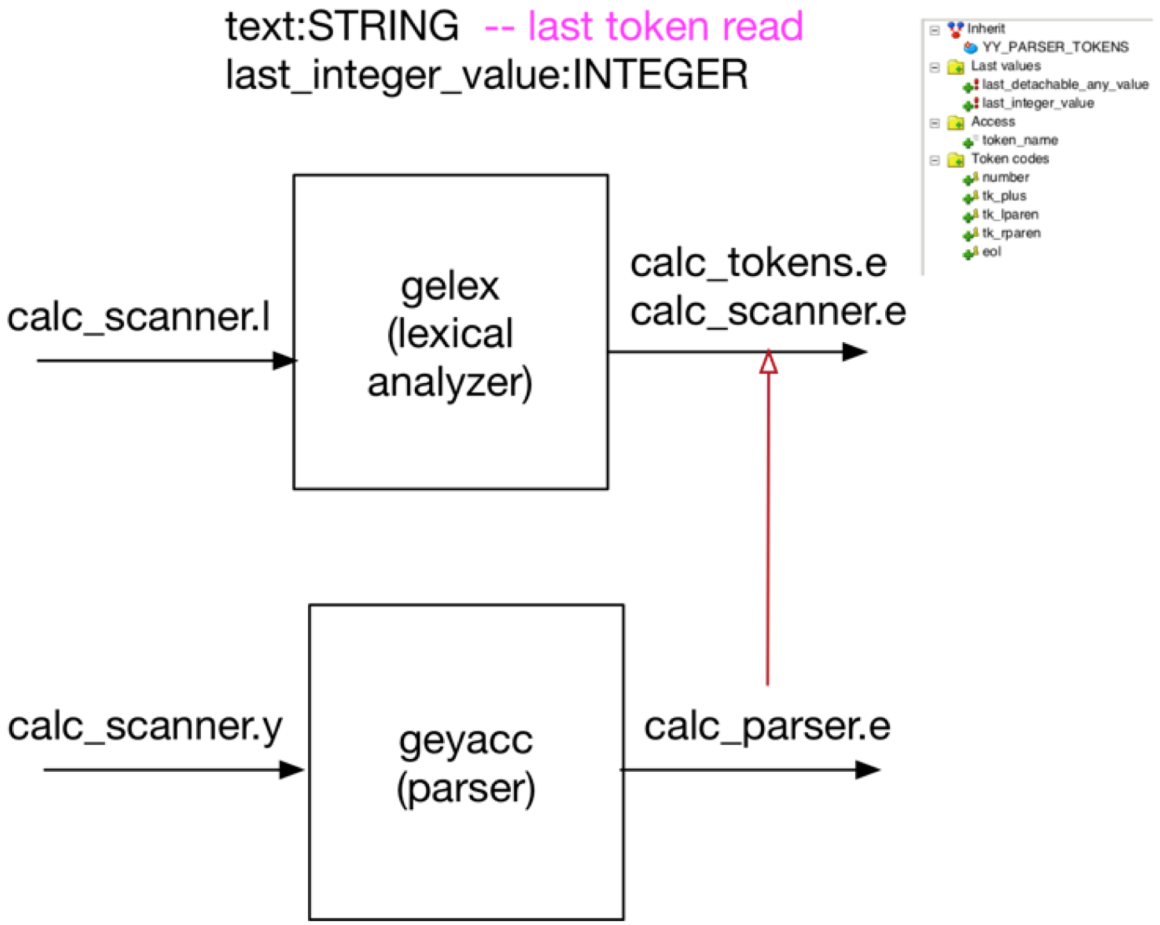
(Wikipedia)

https://svn.eecs.yorku.ca/repos/sel-open/misc/tutorial/yacc-tutorial/docs/index.html
FYI: not needed for the course
Theory of Computation

## Lexer and Parser

Tutorial Documentation (login with `anonymous` and empty password).

The basic idea is as follows:



# Syntax and Semantics

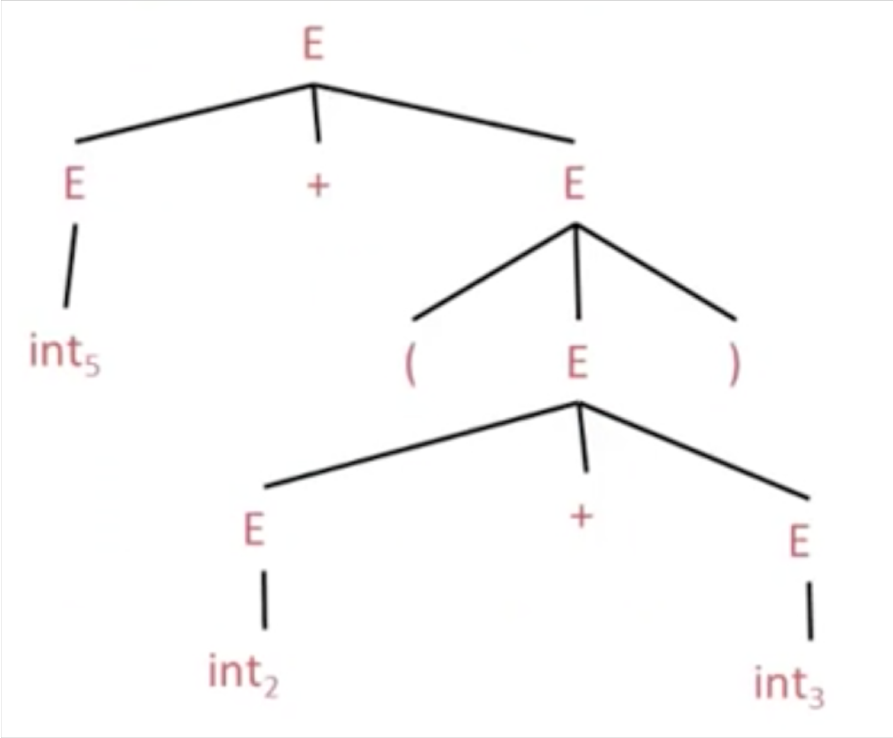
- Syntax is the **grammar**. It describes the way to construct a correct sentence.
  - For example, *this water is triangular* is syntactically correct (<subject><predicate>)
  - A program that is syntactically correct will compile and run.
- Semantics relates to the **meaning**.
  - *this water is triangular* does not mean anything, though the grammar is ok.
  - A program that is semantically correct will actually do what you as the programmer intended it to do (it will also not crash or have bugs/errors)

# Grammar (for expression E)

- E→ Int | (E) | E + E

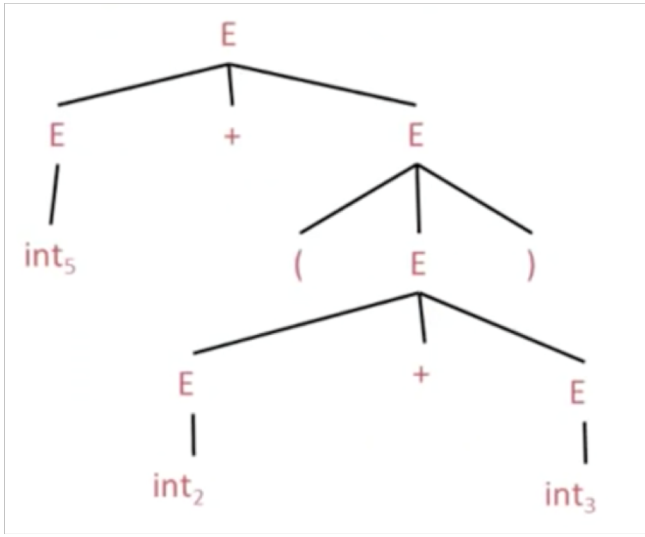
# Parse Tree

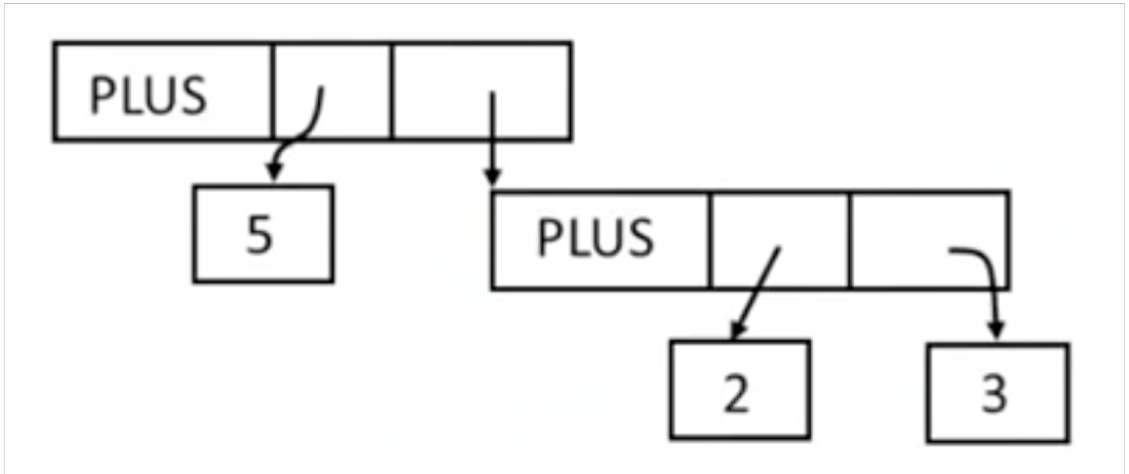
- E → Int | (E) | E + E
- 5 + (2 + 3)



# AST (Abstract Syntax Tree)

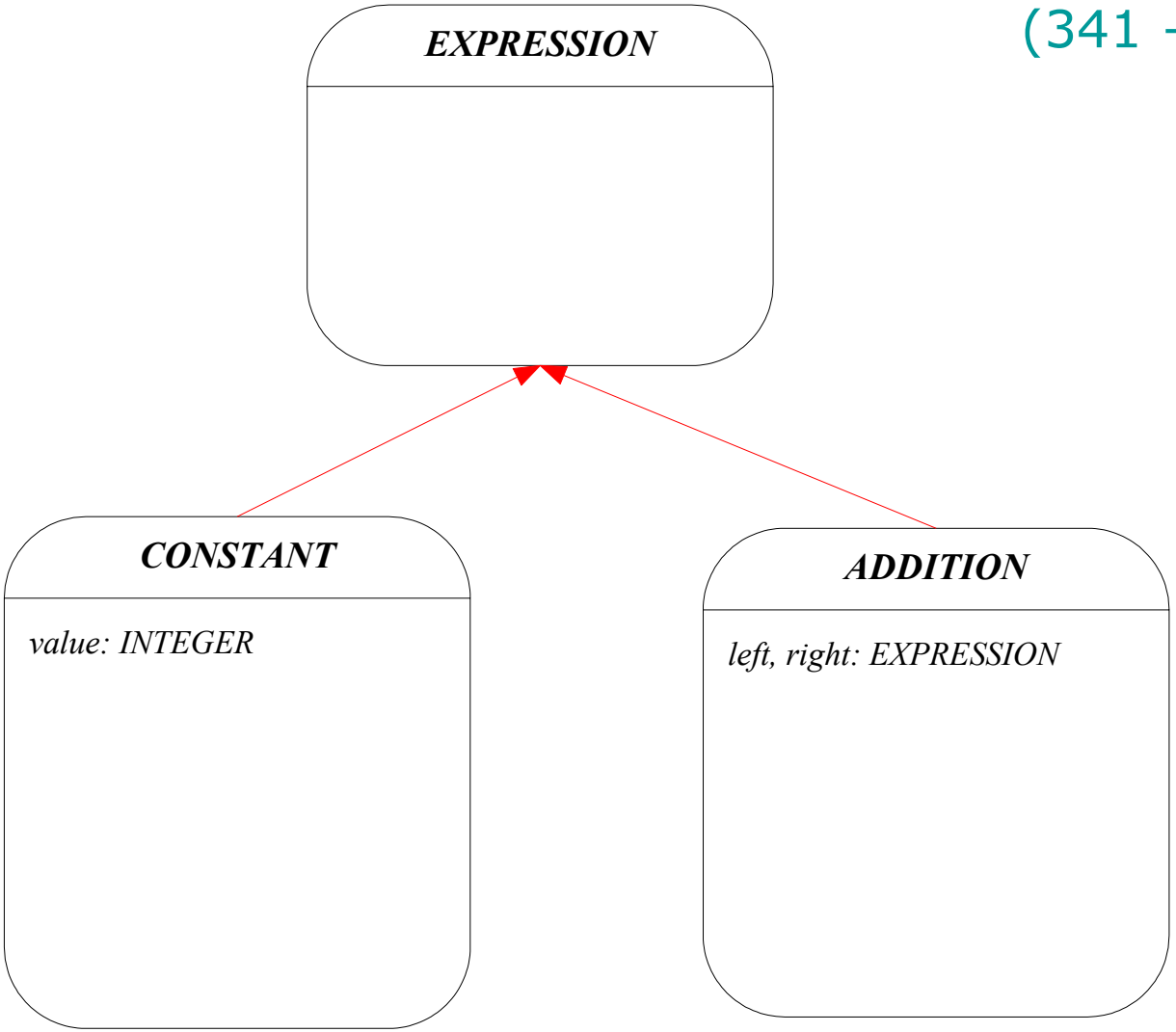
- E → Int | (E) | E + E
- 5 + (2 + 3)



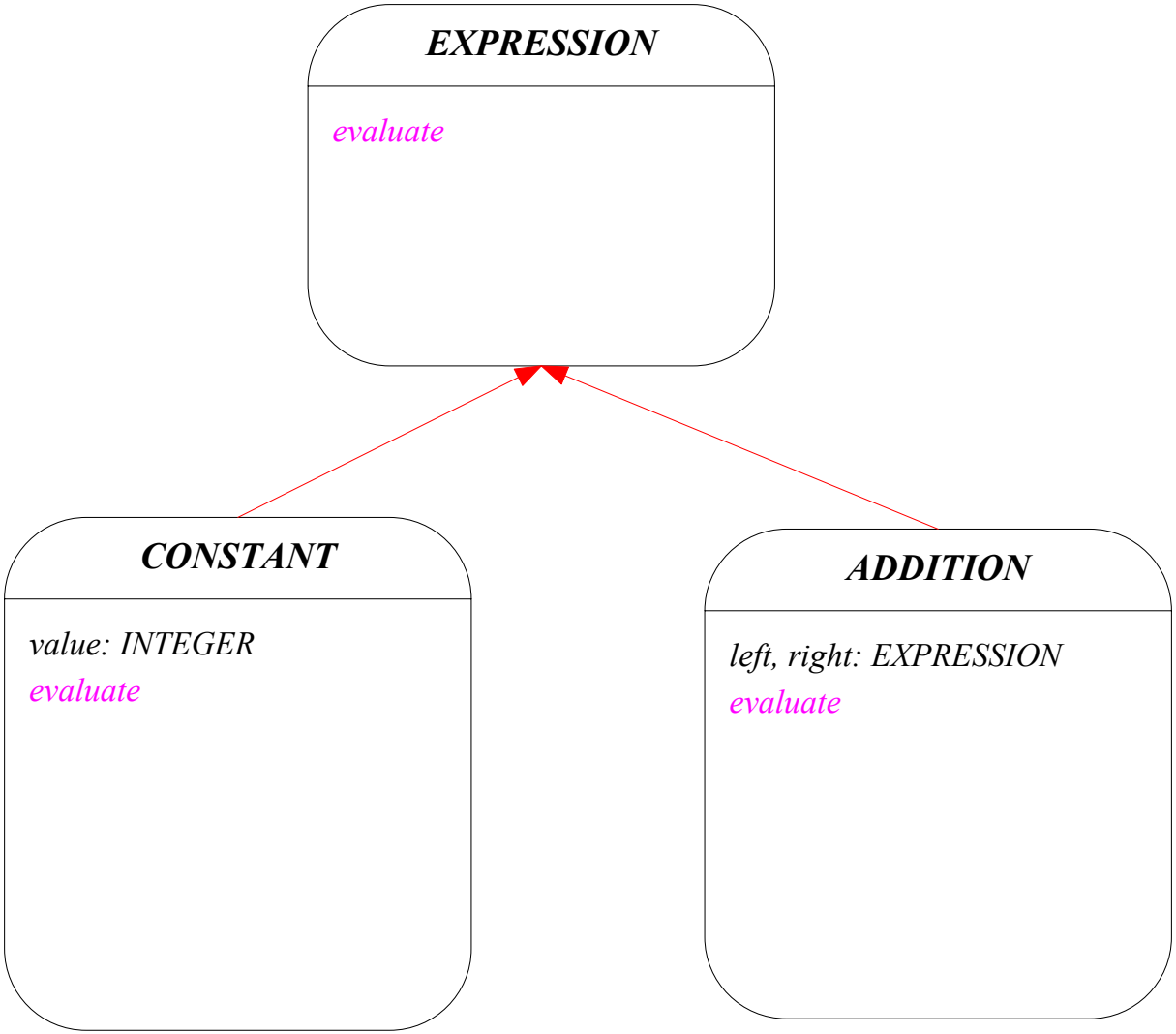



# A parsing problem – Integer expressions

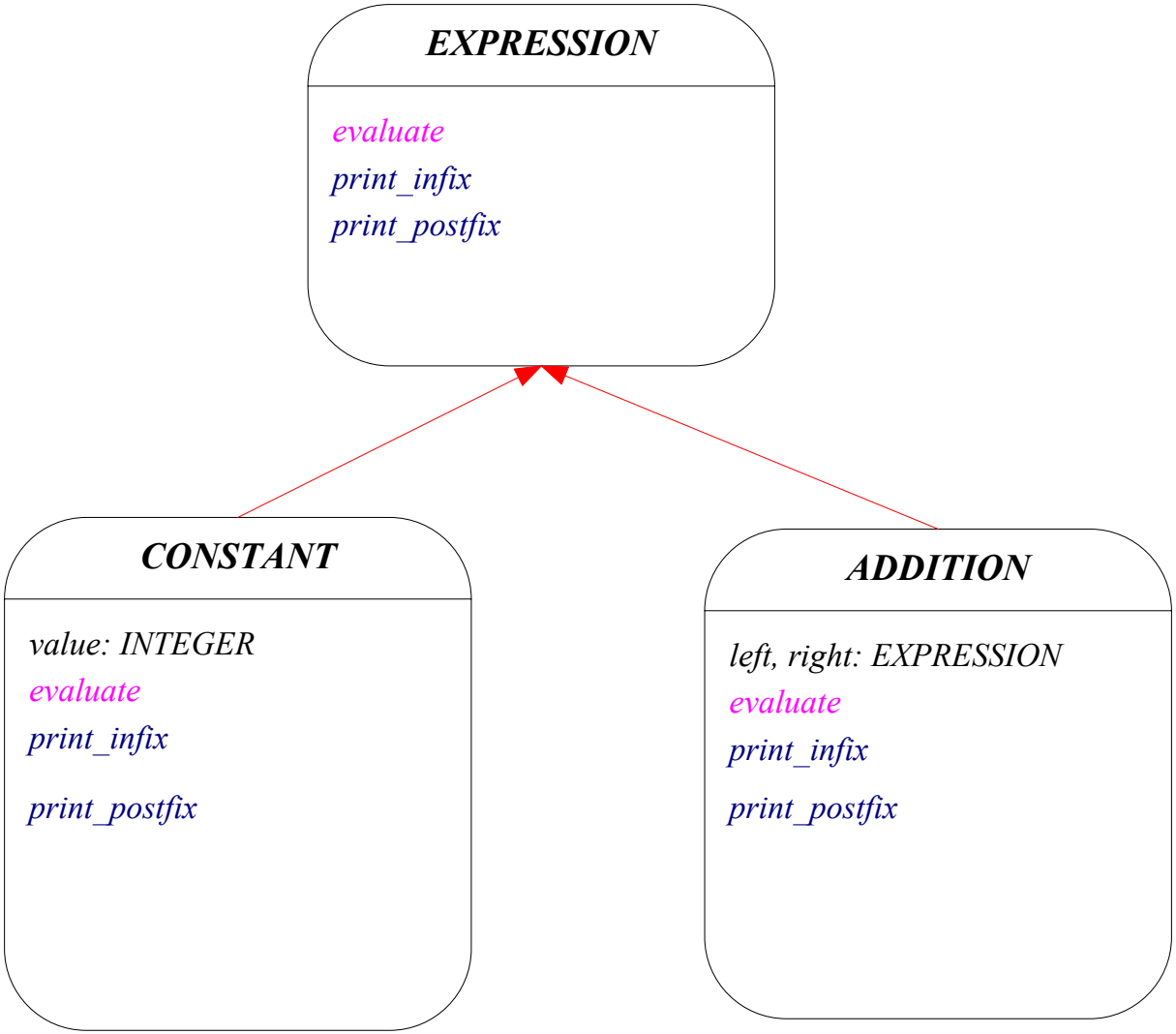
(341 + 2)



# Evaluate – (341 + 2) = 343



# More functions?



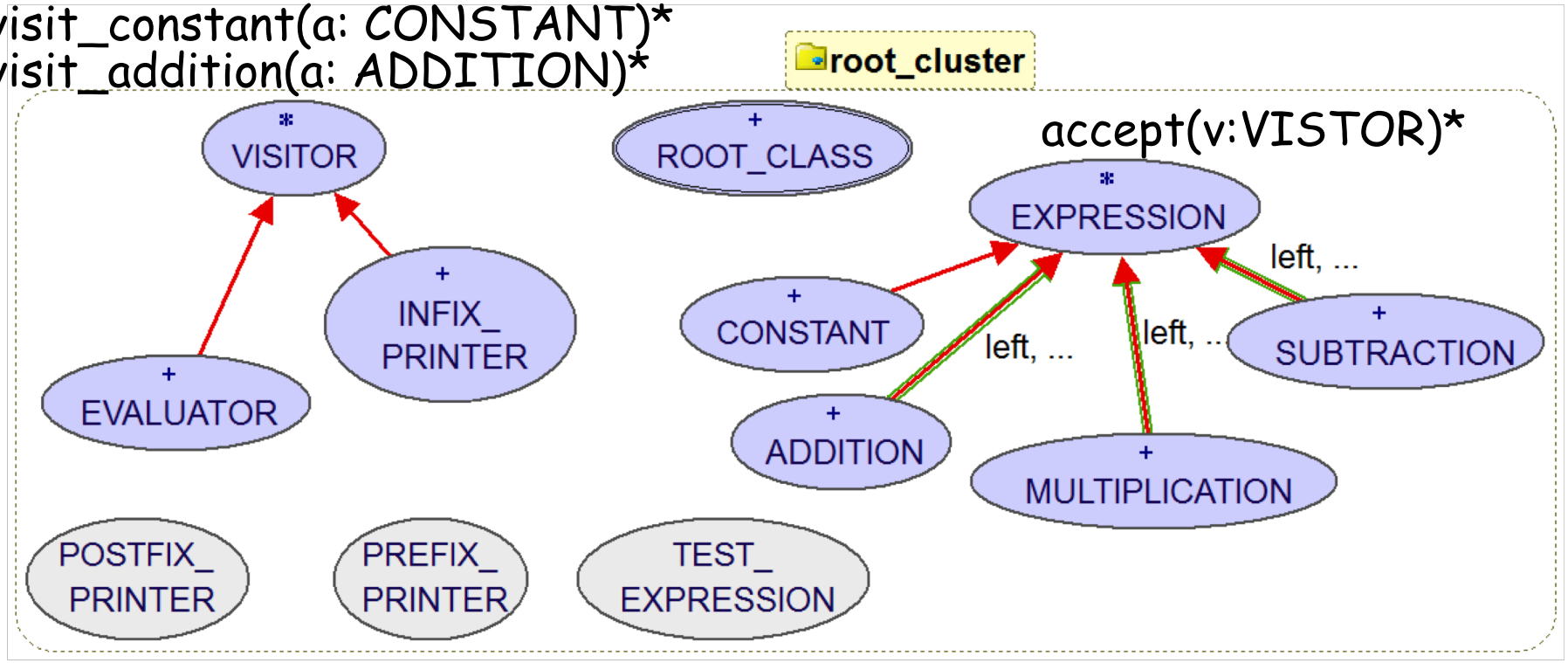
# Problems

- Distributing the various operations for varieties of semantic analysis across the nodes of the abstract syntax tree
  - leads to a system that is hard to understand
  - hard to maintain
  - hard to change
- It is confusing to have evaluation code mixed with pretty printing or type checking code.

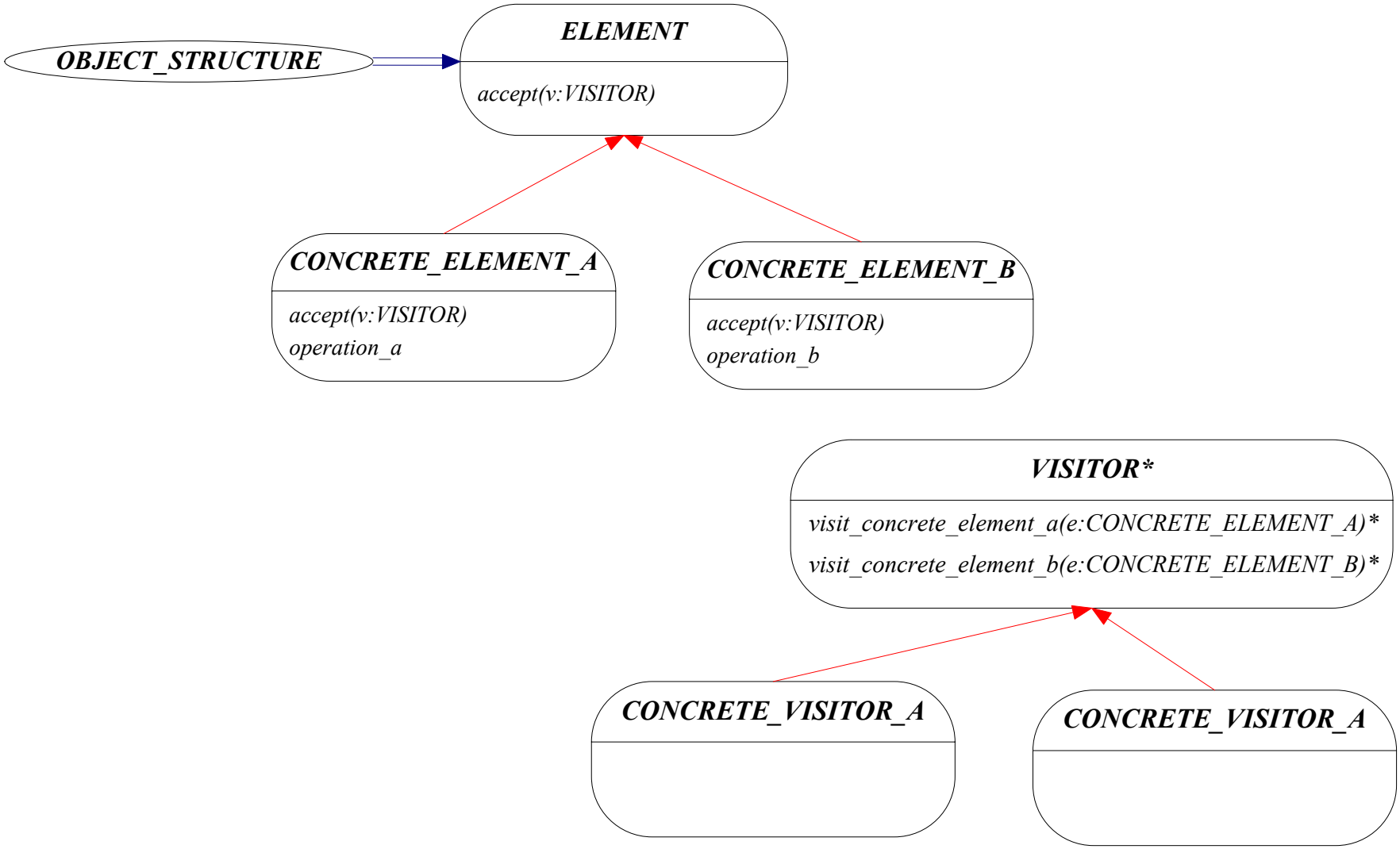
# Visitor Pattern

- An object structure contains many classes (expressions) with the need for unrelated operations (e.g. *evaluate*, *pretty print*, etc) to be performed on the classes.
- We want to avoid "polluting" the classes with these various unrelated operations.
- Define a general interface (accept) to apply various computations to a set of classes (e.g. expressions) without modifying them each time.
- "Call back" type of a mechanism

visit_constant(a: CONSTANT)*
visit_addition(a: ADDITION)*



# Static Diagram

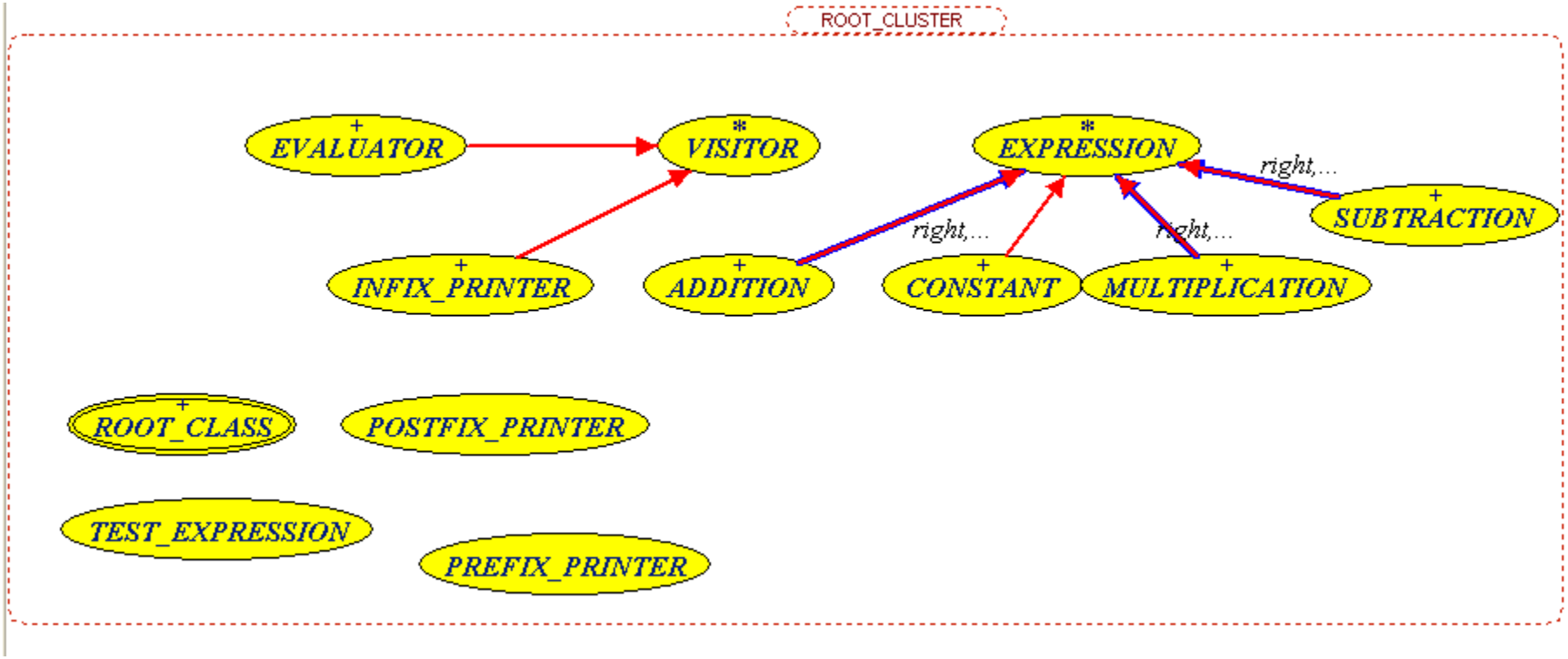


# Key to visitor pattern

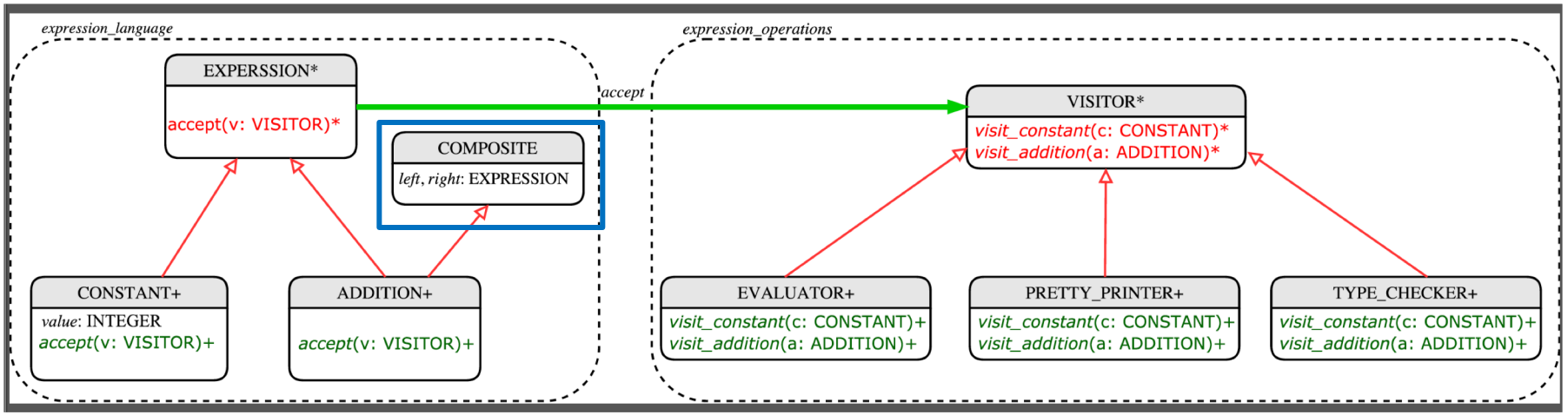
- Call back mode (double dispatch)
- The operation that gets executed depends on both the type of VISITOR and the type of ELEMENT it visits.

# Parsing expression

- See visitor-code.zip (in SVN)
- Sequence diagram on next few slides/with debugger



# Top Level View
# Draw.io Template



# Debugger/UML Sequence Diagram

- Follow in debugger using an ESpec unit test
- During usage, a Visitor visits each element in the structure via recursive descent
  - Top Down
  - Left to Right
- See Prof. Wang's Video https://www.youtube.com/watch?v=z0bjLzs_2-s
- https://www.eecs.yorku.ca/~jackie/teaching/lectures/index.html#EECS3311_F17
  - For code

# Test: (341 +2) = 343

```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
        Result := evaluator.value = 343
    end
```

# Test: (341 +2) = 343

```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
        Result := evaluator.value = 343
    end
```

# Test: (341 +2) = 343

```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
        Result := evaluator.value = 343
    end
```

Objects

| Name | Value | Type | Address |
|---|---|---|---|
| Current object | <0x102C27FD0> | ROOT_CLASS | 0x102C27FD0 |
| Locals | | | |
| constant2 | <0x102C27FF8> | CONSTANT | 0x102C27FF8 |
| value | 2 | INTEGER_32 | |
| Once routi… | | | |
| constant341 | <0x102C28000> | CONSTANT | 0x102C28000 |
| value | 341 | INTEGER_32 | |
| Once routi… | | | |
| evaluator | <0x102C27FE8> | EVALUATOR | 0x102C27FE8 |
| value | 0 | INTEGER_32 | |
| Once routi… | | | |
| plus | <0x102C27FF0> | ADDITION | 0x102C27FF0 |
| left | <0x102C28000> | CONSTANT | 0x102C28000 |
| value | 341 | INTEGER_32 | |
| Once ro… | | | |
| right | <0x102C27FF8> | CONSTANT | 0x102C27FF8 |
| value | 2 | INTEGER_32 | |
| Once ro… | | | |
| Once routi… | | | |
| Result | | | |
| Result | False | BOOLEAN | |

# Test: (341 +2) = 343

```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
        .value = 343
```

Flat view of feature `accept' of class ADDITION

```
accept (visitor: VISITOR)
        -- Traverse with `visitor'.
    do
        visitor.visit_addition (Current)
    end
```

Feature | Class

Objects

| Name | Value | Type | Address |
|---|---|---|---|
| Current object | <0x102C27FF0> | ADDITION | 0x102C27FF0 |
| left | <0x102C28000> | CONSTANT | 0x102C28000 |
| value | 341 | INTEGER_32 | |
| Once routi... | | | |
| right | <0x102C27FF8> | CONSTANT | 0x102C27FF8 |
| value | 2 | INTEGER_32 | |
| Once routi... | | | |
| Once routines | | | |
| Arguments | | | |
| visitor | <0x102C27FE8> | EVALUATOR | 0x102C27FE8 |
| value | 0 | INTEGER_32 | |
| Once routi... | | | |

# Test: (341 +2) = 343

```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
```

= 343

Flat view of feature `visit_addition' of class EVALUATOR

```
visit_addition (a: ADDITION)
        -- Process an addition expression.
    local
        left_evaluator, right_evaluator: EVALUATOR
    do
        create left_evaluator
        create right_evaluator
        a.left.accept (left_evaluator)
        a.right.accept (right_evaluator)
        value := left_evaluator.value + right_evaluator.value
    end
```

Feature | Class

Objects {EVALUATOR}

| Name | Value | Type | Address |
|---|---|---|---|
| Current object | <0x102C27FE8> | EVALUATOR | 0x102C27FE8 |
| value | 0 | INTEGER_32 | |
| Once routines | | | |
| Arguments | | | |
| a | <0x102C27FF0> | ADDITION | 0x102C27FF0 |
| left | <0x102C28000> | CONSTANT | 0x102C28000 |
| value | 341 | INTEGER_32 | |
| Once ro... | | | |
| right | <0x102C27FF8> | CONSTANT | 0x102C27FF8 |
| value | 2 | INTEGER_32 | |

# Test: (341 +2) = 343

```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
```

Flat view of feature `visit_addition' of class EVALUATOR

```
visit_addition (a: ADDITION)
        -- Process an addition expression.
    local
        left_evaluator, right_evaluator: EVALUATOR
    do
        create left_evaluator
        create right_evaluator
        a.left.accept (left_evaluator)
        a.right.accept (right_evaluator)
        value := left_evaluator.value + right_evaluator.value
    end
```

Feature | Class

Objects {EVALUATOR} . visit

| Name | Value | Type | Address |
|---|---|---|---|
| Current object | <0x102C27FE8> | EVALUATOR | 0x102C27FE8 |
| value | 343 | INTEGER_32 | |
| Once routines | | | |
| Arguments | | | |
| a | <0x102C27FF0> | ADDITION | 0x102C27FF0 |
| left | <0x102C28000> | CONSTANT | 0x102C28000 |
| value | 341 | INTEGER_32 | |
| Once ro... | | | |
| right | <0x102C27FF8> | CONSTANT | 0x102C27FF8 |
| value | 2 | INTEGER_32 | |
| Once ro... | | | |
| Once routi... | | | |
| Locals | | | |
| left_evaluator | <0x102C28030> | EVALUATOR | 0x102C28030 |
| value | 341 | INTEGER_32 | |
| Once routi... | | | |
| right_evaluator | <0x102C28028> | EVALUATOR | 0x102C28028 |
| value | 2 | INTEGER_32 | |
| Once routi... | | | |

# Test: (341 +2) = 343

Flat view of feature `visit_addition' of class EVALUATOR

```
visit_addition (a: ADDITION)
        -- Process an addition expression.
    local
        left_evaluator, right_evaluator: EVALUATOR
    do
        create left_evaluator
        create right_evaluator
        a.left.accept (left_evaluator)
        a.right.accept (right_evaluator)
        value := left_evaluator.value + right_evaluator.value
    end
```

Feature | Class

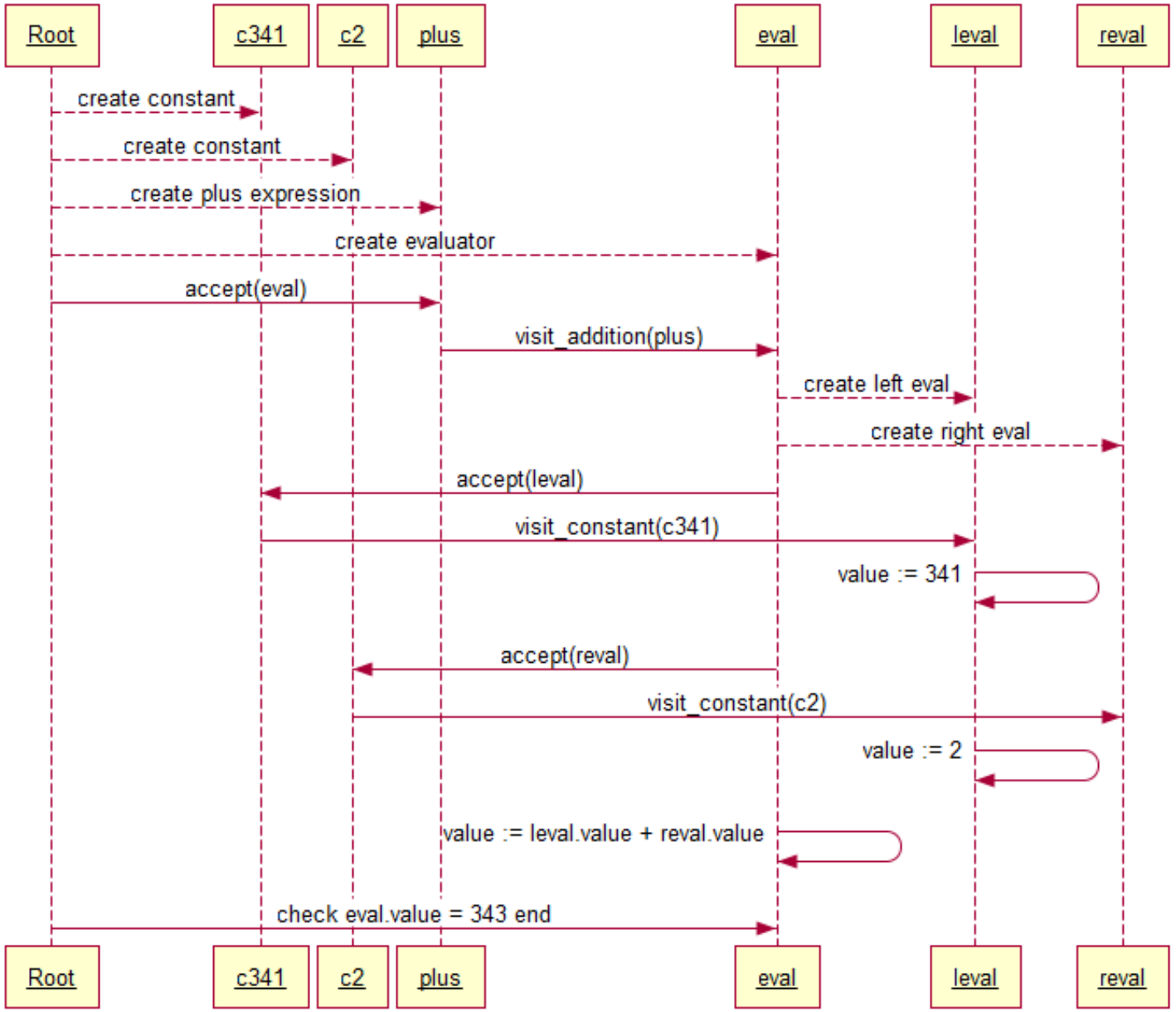
Objects {EVALUATOR} . visi

| Name | Value | Type | Address |
|---|---|---|---|
| Current object | <0x102C27FE8> | EVALUATOR | 0x102C27FE8 |
| value | 343 | INTEGER_32 | |
| Once routines | | | |
| Arguments | | | |
| a | <0x102C27FF0> | ADDITION | 0x102C27FF0 |
| left | <0x102C28000> | CONSTANT | 0x102C28000 |
| value | 341 | INTEGER_32 | |
| Once ro… | | | |
| right | <0x102C27FF8> | CONSTANT | 0x102C27FF8 |
| value | 2 | INTEGER_32 | |
| Once ro… | | | |
| Once routi… | | | |
| Locals | | | |
| left_evaluator | <0x102C28030> | EVALUATOR | 0x102C28030 |
| value | 341 | INTEGER_32 | |
| Once routi… | | | |
| right_evaluator | <0x102C28028> | EVALUATOR | 0x102C28028 |
| value | 2 | INTEGER_32 | |
| Once routi… | | | |

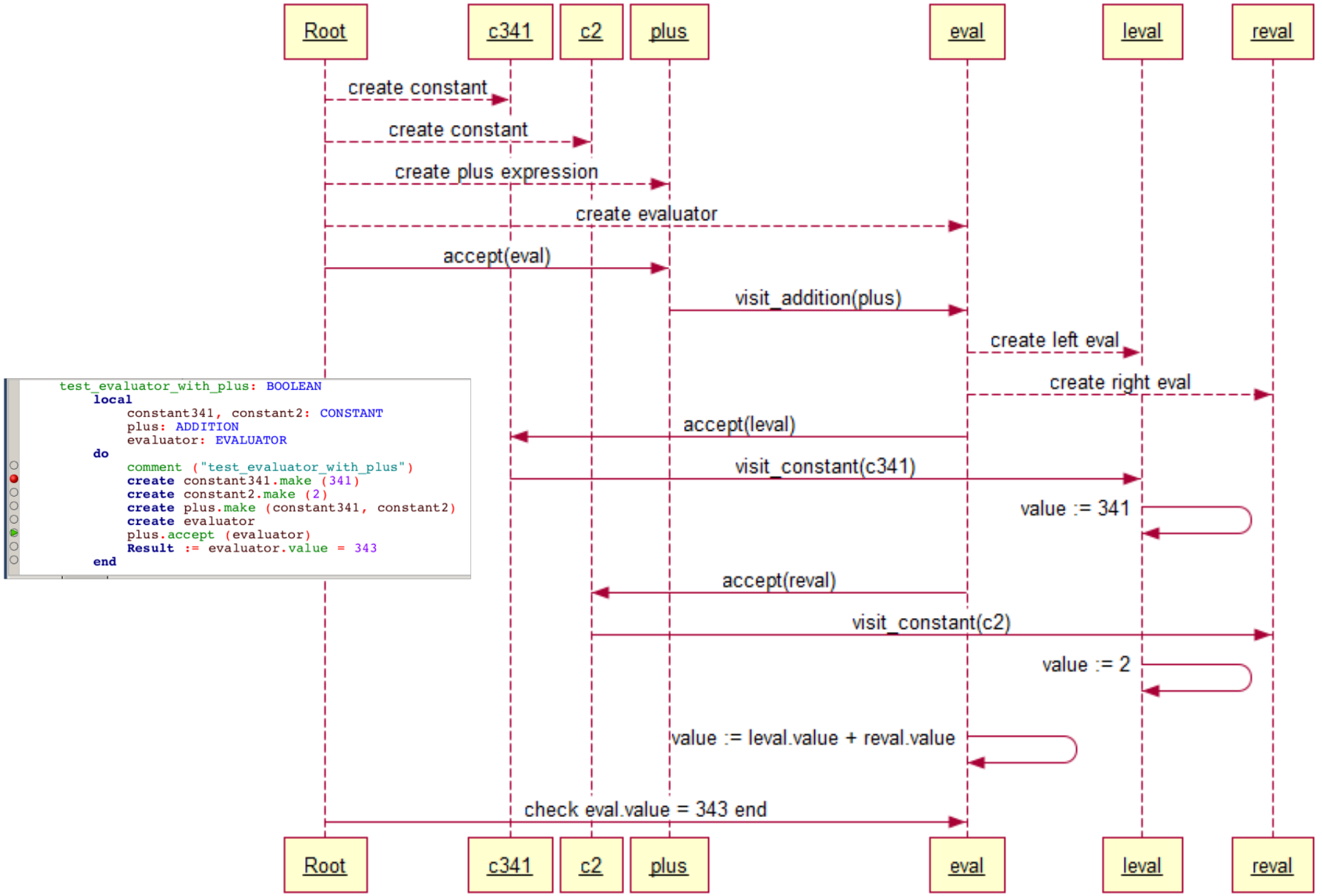
Flat view of feature `test_evaluator_with_plus' of class

```
test_evaluator_with_plu
    local
        constant341, c
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
        Result := evaluator.value = 343
    end
```

# Visitor Sequence Diagram: 341 + 2 = 343



# Visitor Sequence Diagram: 341 + 2 = 343




```
test_evaluator_with_plus: BOOLEAN
    local
        constant341, constant2: CONSTANT
        plus: ADDITION
        evaluator: EVALUATOR
    do
        comment ("test_evaluator_with_plus")
        create constant341.make (341)
        create constant2.make (2)
        create plus.make (constant341, constant2)
        create evaluator
        plus.accept (evaluator)
        Result := evaluator.value = 343
    end
```

# Problems with visitor pattern

- Adding a new CONCRETE_ELEMENT_C class is hard
  - Every concrete visitor must be provided with a *visit_concrete_element_c* operation
- Depends on what extent ELEMENT structure will change.
  - Use visitor when the algorithms applied to the object structure is likely to change
  - Do not use when the classes of elements are likely to change.

# Visitors Exercise

- Consider a grammar for expressions like
  (1.6*2 = 3.2) /\ (2.8*4.6-9.7 <= 3.1)
- Arithmetic expressions that may be embedded in Boolean expressions
- Assume that the input syntax is correct, but now you must type check before evaluation. For example, (2>3) + (1<3) may be syntactically correct according to a grammar, but is not type correct. So the type is either Arithmetic, Boolean or Unknown.
- Design two visitors, one to type check, and one to evaluate. The precondition for evaluation is that it should be type correct (either Arithmetic or Boolean).