Question 1: Derivations and Parse Trees

Leftmost and rightmost derivation of the sentence "case a in 0: b=1 1: b=5 end"

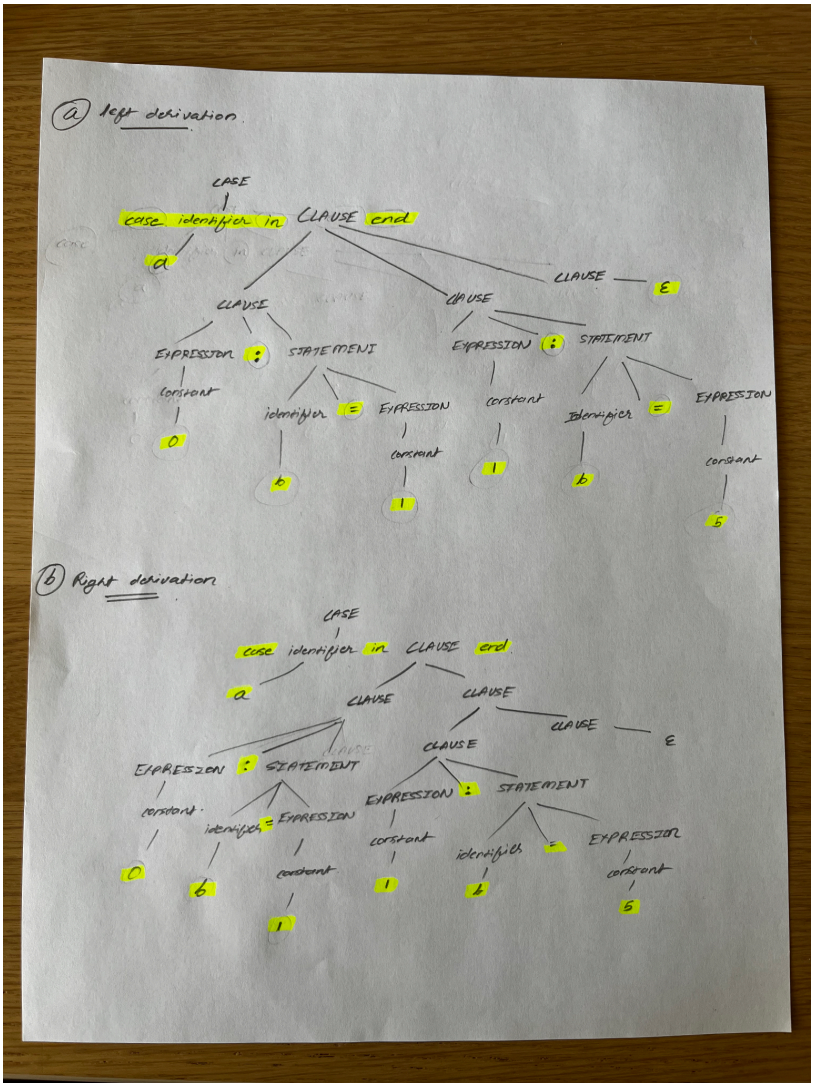
Leftmost derivation

```
CASE
case identifier in CLAUSES end
case a in CLAUSES end
case a in CLAUSE CLAUSES end
case a in EXPRESSION : STATEMENT CLAUSES end
case a in constant : STATEMENT CLAUSES end
case a in 0 : STATEMENT CLAUSES end
case a in 0 : identifier = EXPRESSION CLAUSES end
case a in 0 : b = EXPRESSION CLAUSES end
case a in 0 : b = constant CLAUSES end
case a in 0 : b = 1 CLAUSES end
case a in 0 : b = 1 CLAUSE CLAUSES end
case a in 0 : b = 1 EXPRESSION : STATEMENT CLAUSES end
case a in 0 : b = 1 constant : STATEMENT CLAUSES end
case a in 0 : b = 1 1 : STATEMENT CLAUSES end
case a in 0 : b = 1 1 : identifier = EXPRESSION CLAUSES end
case a in 0 : b = 1 1 : b = EXPRESSION CLAUSES end
case a in 0 : b = 1 1 : b = constant CLAUSES end
case a in 0 : b = 1 1 : b = 5 CLAUSES end
case a in 0 : b = 1 1 : b = 5 ε end
```

Rightmost derivation

```
CASE
case identifier in CLAUSES end
case identifier in CLAUSE CLAUSES end
case identifier in CLAUSE CLAUSE CLAUSES end
case identifier in CLAUSE CLAUSE ε end
case identifier in CLAUSE EXPRESSION : STATEMENT end
case identifier in CLAUSE EXPRESSION : identifier = EXPRESSION end
case identifier in CLAUSE EXPRESSION : identifier = constant end
case identifier in CLAUSE EXPRESSION : identifier = 5 end
case identifier in CLAUSE EXPRESSION : b = 5 end
case identifier in CLAUSE constant : b = 5 end
case identifier in CLAUSE 1 : b = 5 end
case identifier in EXPRESSION : STATEMENT 1 : b = 5 end
case identifier in EXPRESSION : identifier = EXPRESSION 1 : b = 5 end
```

```
case identifier in EXPRESSION : identifier = constant 1 : b = 5 end
case identifier in EXPRESSION : identifier = 1 1 : b = 5 end
case identifier in EXPRESSION : b = 1 1 : b = 5 end
case identifier in constant : b = 1 1 : b = 5 end
case identifier in 0 : b = 1 1 : b = 5 end
case a in 0 : b = 1 1 : b = 5 end
```



Question 2 : Ambiguity

The ambiguity arises from the production rule for

Grammer: `CLAUSES -> CLAUSE CLAUSES | ε`

While this specific rule form is technically unambiguous (it always creates a right-branching tree, as seen in Q1), the concept of a list of clauses without separators is a source of structural ambiguity. A sequence like C1 C2 (where C is a CLAUSE) could be interpreted in two ways, with two different associations:

```
left associative: (C1) C2 - The clauses are grouped to the left.
right associative: C1 (C2) - The clauses are grouped to the right.
```

Let's take an example where the grammar allowed both interpretations. For the string "C1 C2", we could have two different parse trees:

```
Tree 1 (Right-Associative): CLAUSES -> C1 (CLAUSES -> C2 (CLAUSES -> ε))
Tree 2 (Left-Associative): CLAUSES -> (CLAUSES -> (CLAUSES -> ε) C1) C2
```

The grammar is ambiguous since a single string can produce two different parse trees.

Question 3 : Resolving Ambiguity (using Terminal)

To disambiguate the grammar, a semicolon terminal (;) can be added to explicitly separate clauses. This forces a single, unambiguous parse.

```
case a in 0: b=1; 1: b=5 end
```

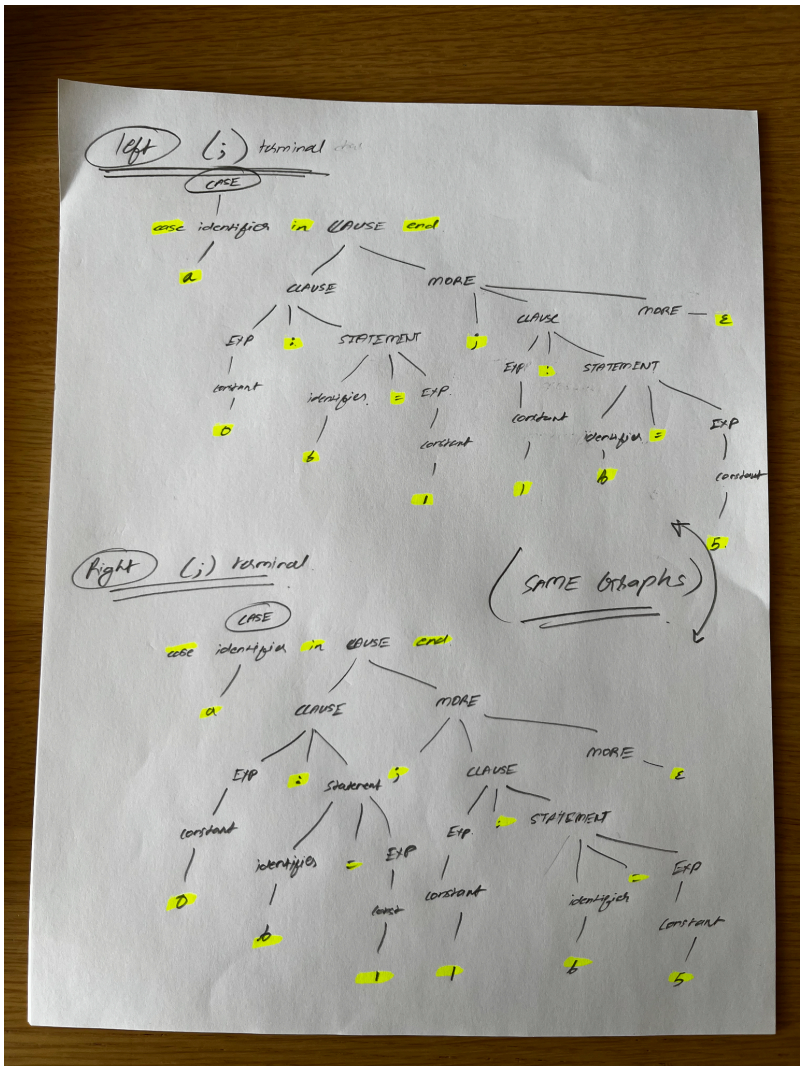
New Grammer:

```
CLAUSES -> CLAUSE MORE | ε
MORE -> ; CLAUSE MORE | ε
```

Old Grammer:

```
CLAUSES -> CLAUSE CLAUSES | ε
```

This resolved the ambiguity by adding a terminal `;`. Both the left and right derivative graphs now look the same with the new implementation.



Question 4 : Resolving Ambiguity (using Non Terminal)

To disambiguate without new terminals, we enforce a left-associative structure, which is more intuitive for programmers as it implies left-to-right sequential evaluation.

New Grammar:

```
CLAUSES -> CLAUSE_LIST | ε
CLAUSE_LIST -> CLAUSE_LIST CLAUSE | CLAUSE
```

The reworked grammar is left-recursive, which causes infinite loops in recursive descent parsers and requires more complex parsing techniques or grammar transformations to handle.