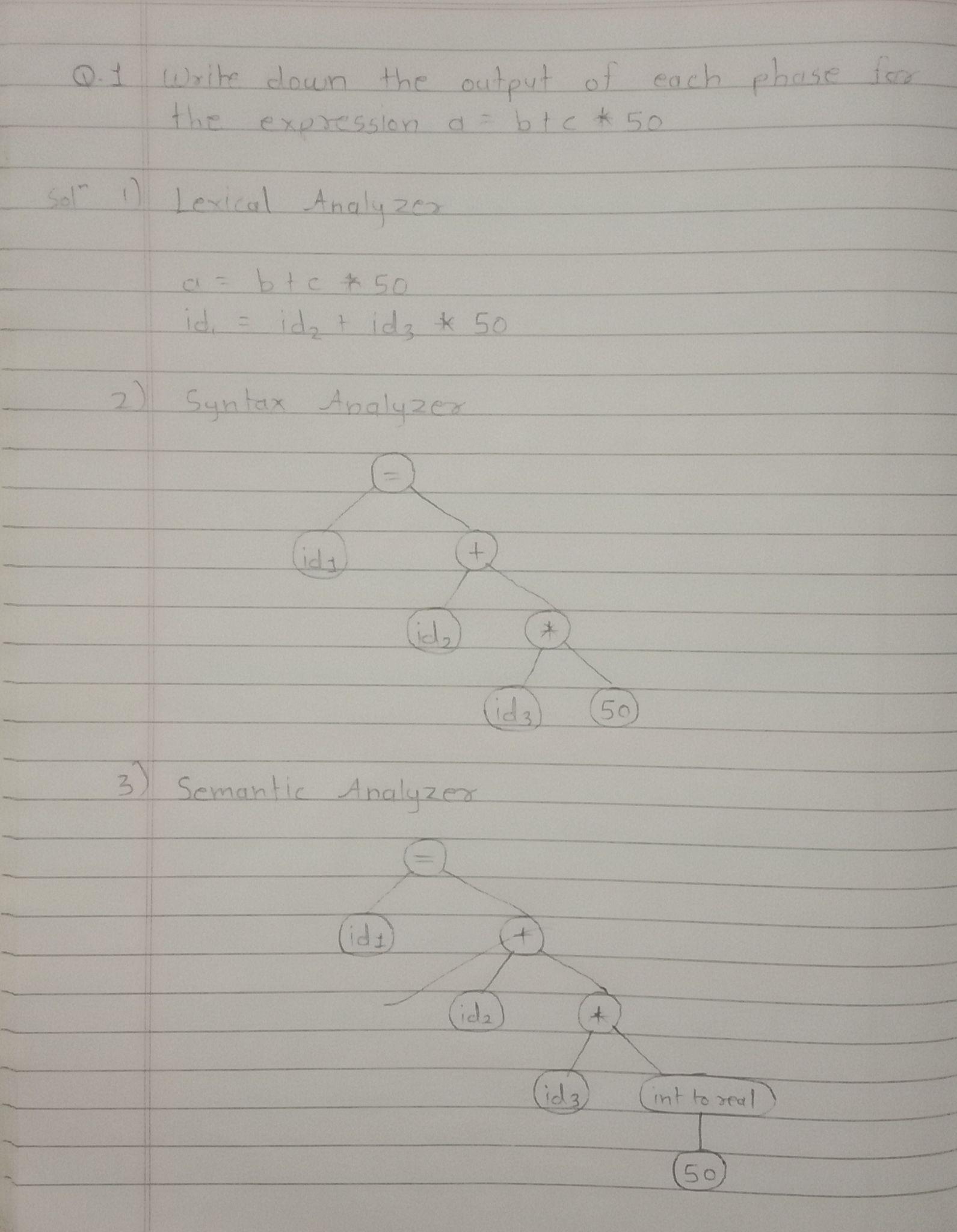
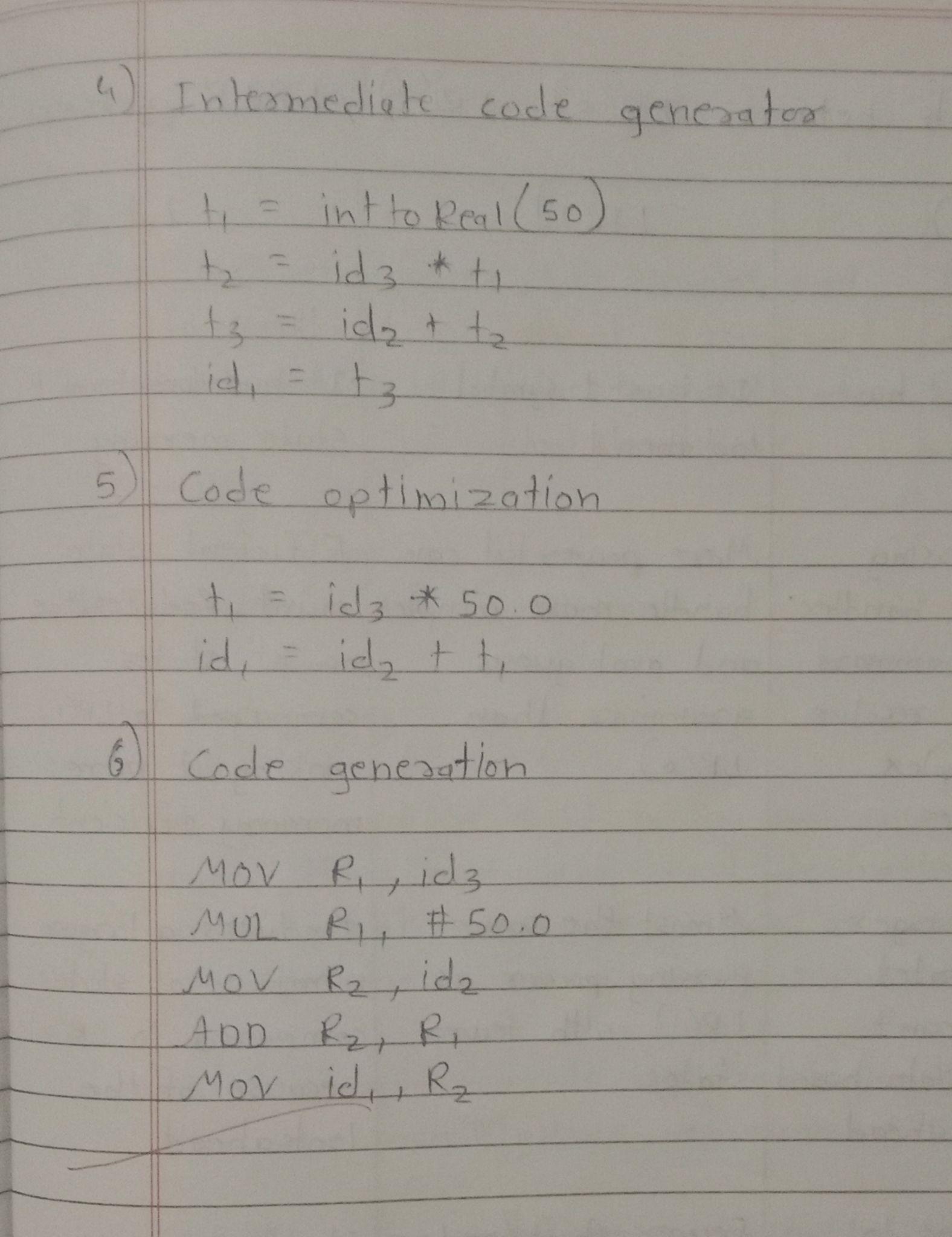
Q.1) Write down the output of each phase for the expression a = b + c \* 50

Ans:



Q.2) Distinguish between LR(0), LR(1), LALR parser.

Ans :

| Sr. No. | LR(0) | LR(1) | LALR |
| --- | --- | --- | --- |
| 1 | It does not have lookahead | It has 1 symbol lookahead | It has lookahead + state merging |
| 2 | Basic parsing power. It handles simple grammars but can’t resolve more complex ambiguities | More powerful can handle more complex and ambiguous grammar than LR(0) | Efficient state count. Reduces the no. of states compared to LR(1) making it more memory efficient |
| 3 | Builds a larger no. of states since it can’t differentiate based on look ahead | Almost the same parsing power as LR(1) with fewer states | Produces a larger number of states compared to LR(0) because of the lookahead |
| 4 | More prone to reduce/shift conflicts due to lack of lookahead | Fewer shift/reduce conflicts because look ahead helps in decision making | Slightly more prone to conflicts compared to LR |
| 5 | Easier and faster to construct compared to LR (1) | More complex and slower to construct due to additional lookahead logic | Preferred in practice due to balance between efficiency and parsing power |

Q.3) Write a short note on:

a) Ambiguity

A grammar is said to be ambiguous if there exists more than one left most derivation or more than one right most derivation or more than one parser tree for a given input string.

If the grammar is not ambiguous then we call it unambiguous grammar

If the grammar has ambiguity then it is good for compiler construction.

Example :

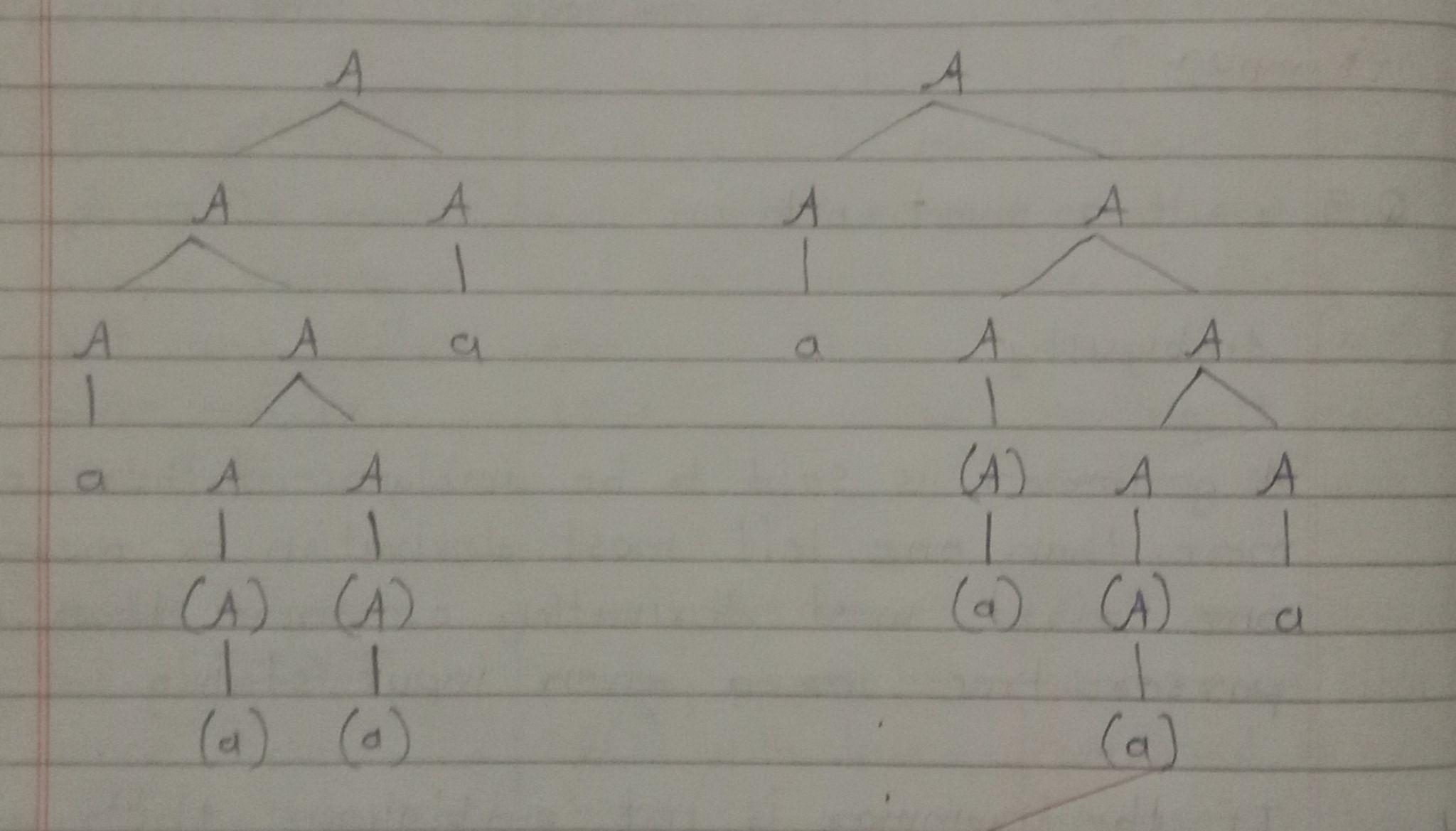
Let us consider a grammar with production rules, as shown below -

A = AA

A = (A)

A = a

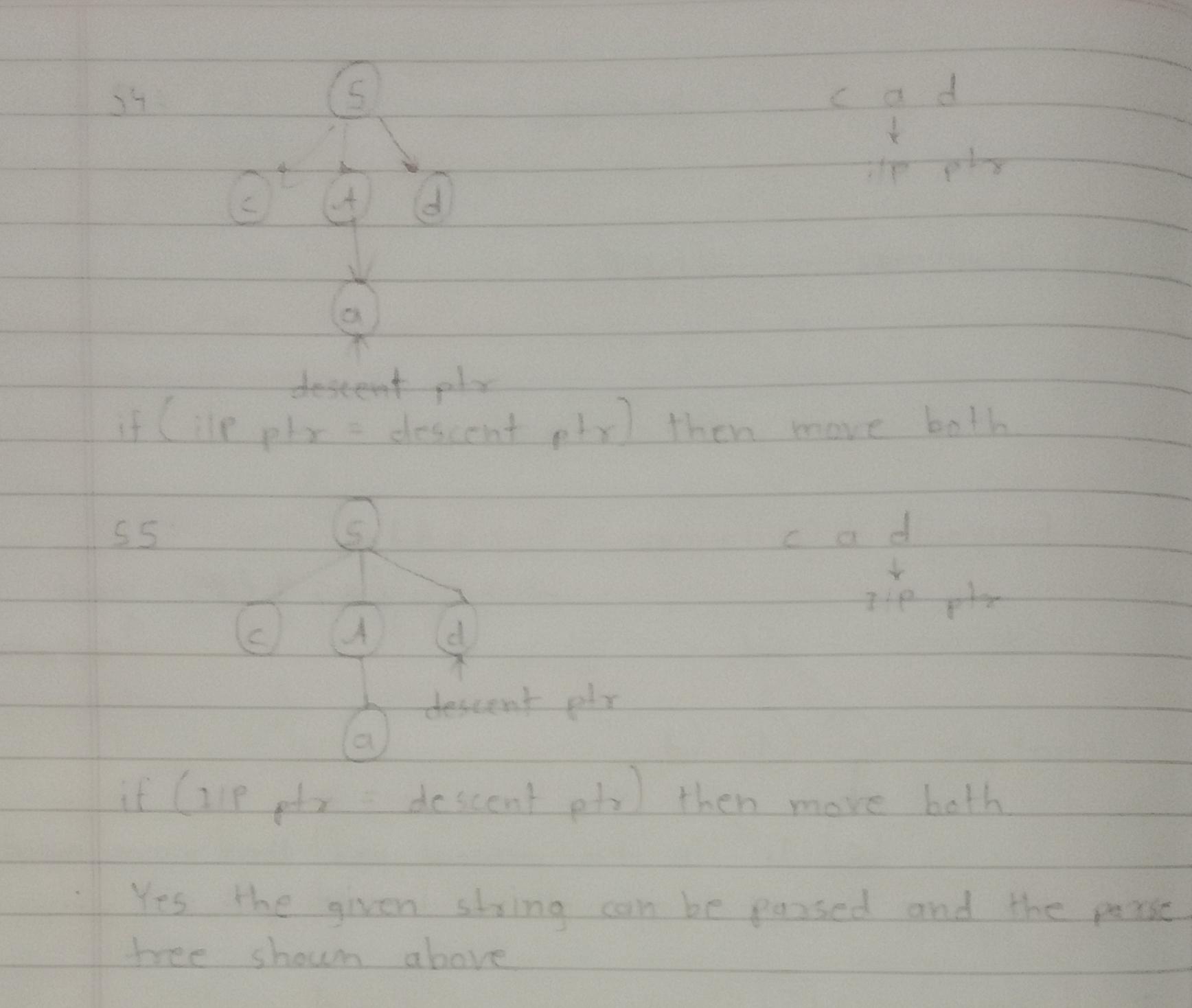
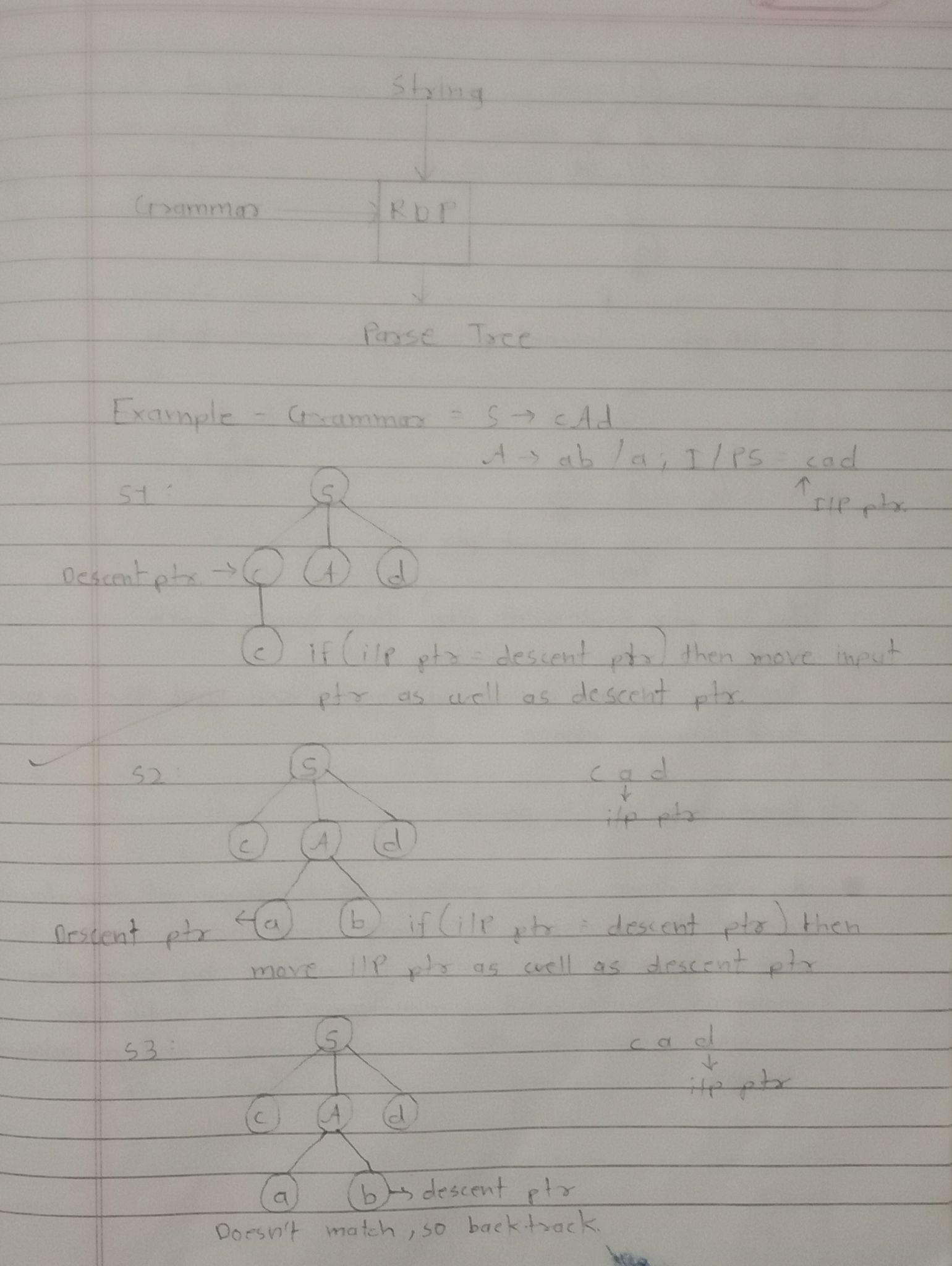
For the string “a(a)(a)a” the above grammar can generate 2 parse trees as given



b) Recursive Descent Parser:

RDP is one of the top down parsing technique that uses a set of recursive procedure to scan its input

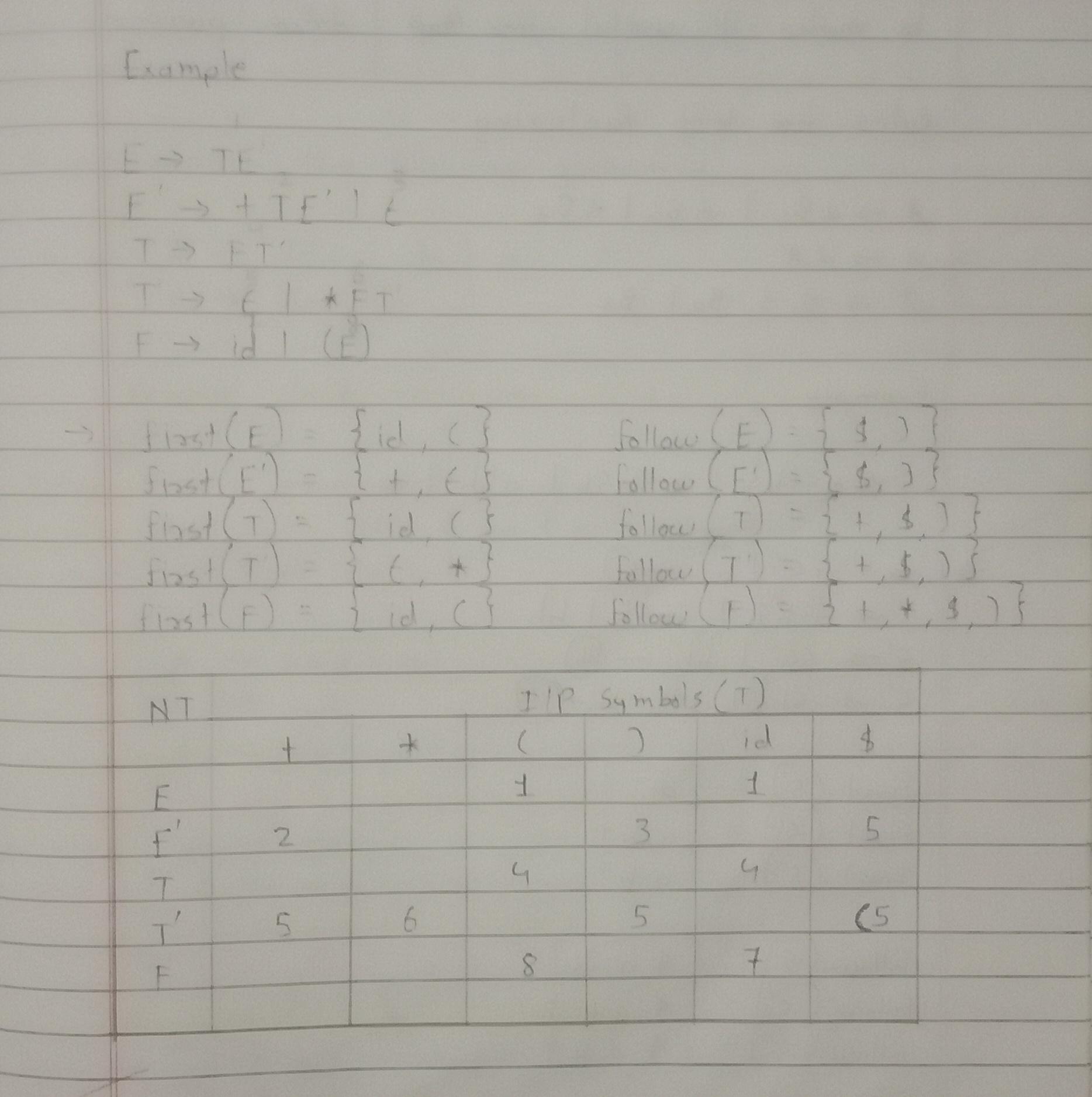
The parsing method may involve backtracking i.e. making repeated scans of its input.



c) Predictive LL(1) parser

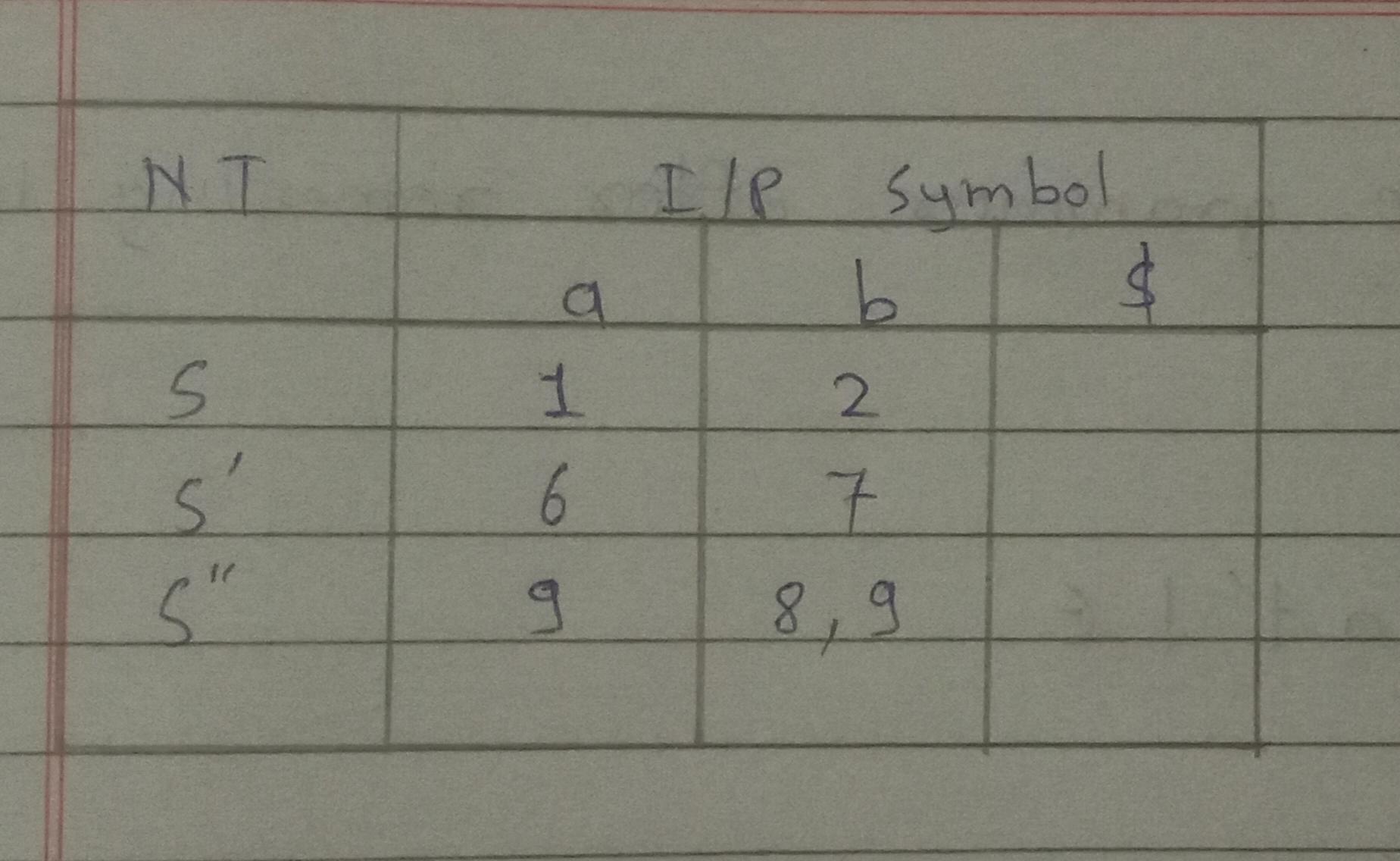
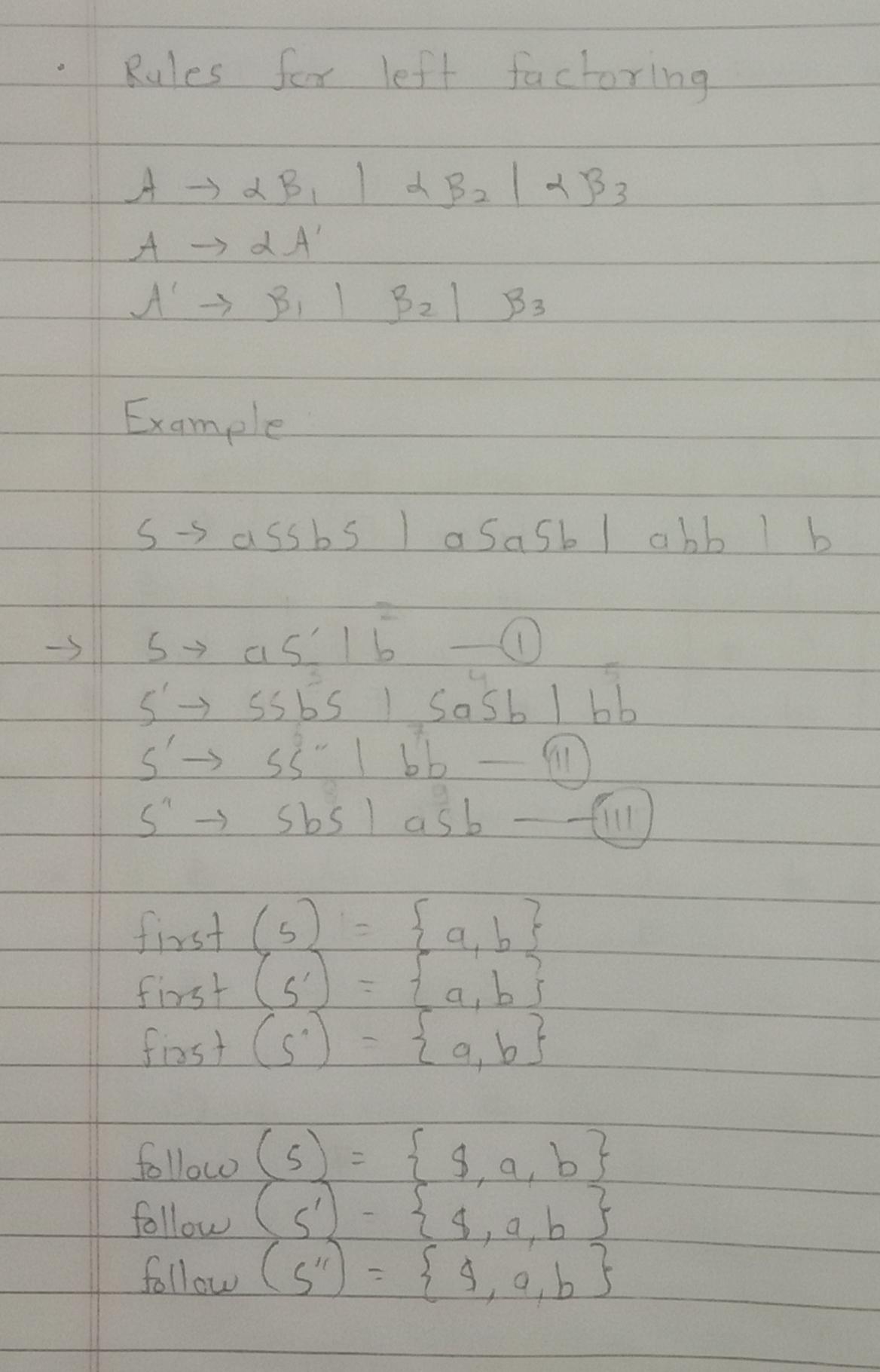
A predictive LL(1) parser is a type of top down parser that uses a single lookahead symbol to make parsing decisions. It works by expanding grammar rules and predicting the production to use based on the current input and a lookahead token.

“LL” stands for scanning the input from left to right and constructing a leftmost derivation. The number “1” indicates the use of one lookahead symbol.



d) Left factoring

Left factoring is a process by which the grammar with common prefixes is transform to make it useful for top-down parser.



Q.4) Consider the following grammar

A -> ABd | Aa | a

B -> Be | b

Remove left recursion

Ans:

Given grammar

A -> ABd | Aa | a

B -> Be | b

A -> ABd | a

A -> aA’

A’ -> BdA’ | ε

A -> Aa | a

A -> aA’

A’ -> aA’ | ε

B -> Be | b

B -> bB’

B’ -> eB’ | ε

Therefore the grammar after removing left recursion is:

A -> aA’

A’ -> BdA’ | aA’ | ε

B -> bB’

B’ -> eB’ | ε