Q1: Translation Grammar (S) for AST Generation

This document provides a translation grammar that generates an Abstract Syntax Tree (AST) for an expression grammar
with operations such as addition, subtraction, multiplication, and division. The grammar uses semantic actions to
construct the AST as the expression is parsed.

Expression Grammar

The following grammar defines how expressions are structured. We assume the grammar contains the non-terminals
E (expressions), T (terms), and F (factors), and the terminals +, -, *, /, (,), and id (identifiers).

1. E → E + T
2. E → E - T
3. E → T
4. T → T * F
5. T → T / F
6. T → F
7. F → (E)
8. F → id

Semantic Actions for AST Construction

The semantic actions build the AST during parsing. For each operation (like +, -, *, /), the semantic action creates a
node with the operation as the label and the subexpressions as children. For terminal symbols like id, a leaf node
is created.

S-attributed Grammar

In this grammar, the AST is built using synthesized attributes that are passed up the parse tree from child nodes to

parent nodes.

Productions and Semantic Actions:

1. E → E + T
   Action: E.node = make_node('+', E1.node, T.node)

2. E → E - T
   Action: E.node = make_node('-', E1.node, T.node)

3. E → T
   Action: E.node = T.node

4. T → T * F
   Action: T.node = make_node('*', T1.node, F.node)

5. T → T / F
   Action: T.node = make_node('/', T1.node, F.node)

6. T → F
   Action: T.node = F.node

7. F → (E)
   Action: F.node = E.node

8. F → id
   Action: F.node = make_leaf('id')

In this translation scheme, the `node` attribute is synthesized and carries the AST node that is created for each
production. The nodes are passed up from the leaves (identifiers and factors) to the root (complete expressions).
This approach guarantees that the AST accurately represents the structure of the parsed expression.