CSci 365: Organizations of Programming Languages

Name: Kenneth Jahnke

**Assignment 1: 70 + 10 points (optional)**

Q1. [10] The switch statement in C language has the following format.

**switch**(expression) {

**case** constant-expression:

statement(s);

**break**; /\* optional \*/

/\* you can have any number of case statements \*/

**default**: /\* optional \*/

statements(s);

}

Write an Extended Backus Naur Form (EBNF) description for a **switch** statement, using the following non-terminals < … >.

<switch\_stmt> : specifies the switch statement abstraction

<expr> : specifies an expression,

<literal> : specifies a constant-expression,

<stmt\_list> : specifies a list of statements.

**switch, case, break, default**: keywords in C.

===================BEGIN ANSWER Q1===================

**QUESTION:** The book utilizes `→` for definition. Online resources suggest `::=` for BNF and `=` for EBNF are notations equivalent to `→`. Is this true?

**QUESTION:** When writing in EBNF, can angle brackets `< >` be omitted for non-terminals?

<switch\_stmt> → “switch” “(“ <expr> “)” “{“ {<case\_stmt>} [<default\_stmt>] “}“

<case\_stmt> → “case” <literal> “:” <stmt\_list> [“break” “;”]

<default\_stmt> → “default” “:” <stmt\_list>

Notes:

1. `{ }` surrounds `<case\_stmt>` because any number of cases may be present.

2. `[ ]` surrounds `<default\_stmt>` because this statement is optional.

3. No `”;”` succeeds `<stmt\_list> because each statement must have its own termination (`;`) within itself.

====================END ANSWER Q1===================

Q2. [10] Using the following grammar in BNF, draw a parse tree and write a leftmost derivation for a statement below.

A = A / (B + (C / A))

Grammar in BNF:

<assign>  <id> = <expr>

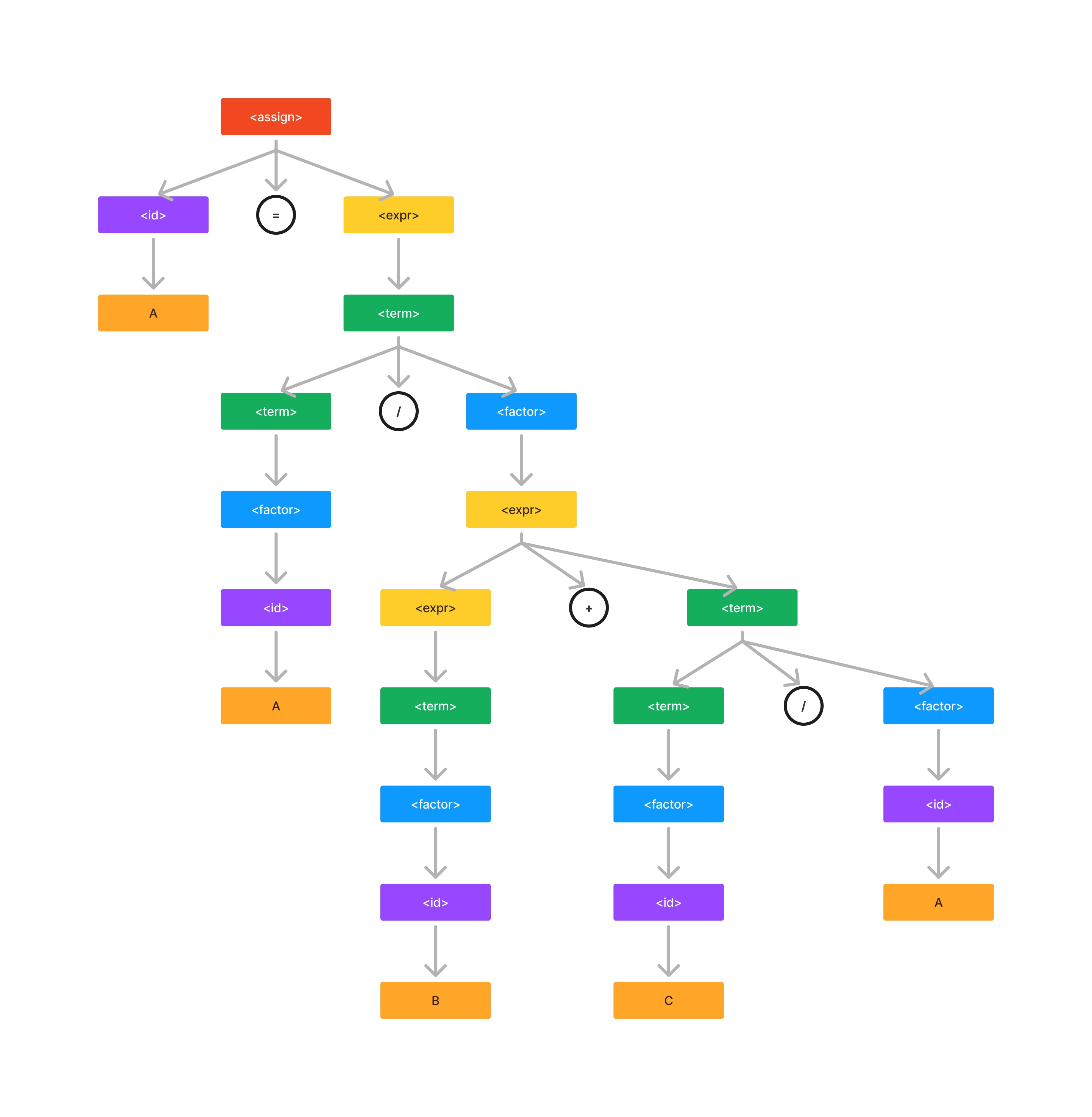
<expr>  <expr> + <term> | <term>

<term>  <term> / <factor> | <factor>

<factor>  (<expr>) | <id>

<id>  A | B | C

===================BEGIN ANSWER Q2===================



====================END ANSWER Q2===================

Q3. [10] Rewrite the BNF of Q2 to give + precedence over / and force + to be right-associative.

===================BEGIN ANSWER Q3===================

<assign> → <id> = <expr>

<expr> → <expr> / <term> | <term>

<term> → <factor> + <term> | <factor>

<factor> → (<expr>) | <id>

<id> → A | B | C

====================END ANSWER Q3===================

Q4. [10] Modify the grammar of Q2 to add a unary minus () operator that has higher precedence than either + or /.

Note: A unary minus operator is, for example,  A in statement B =  A / 2.

Unary operators precede any operator.

===================BEGIN ANSWER Q4===================

<assign>  <id> = <expr>

<expr>  <expr> + <term> | <term>

<term>  <term> / <factor> | <factor>

<factor>   <factor> | (<expr>) | <id>

<id>  A | B | C

====================END ANSWER Q4===================

Q5. [10]

1. [5] Prove that the following grammar is ambiguous.

<S>  <A>

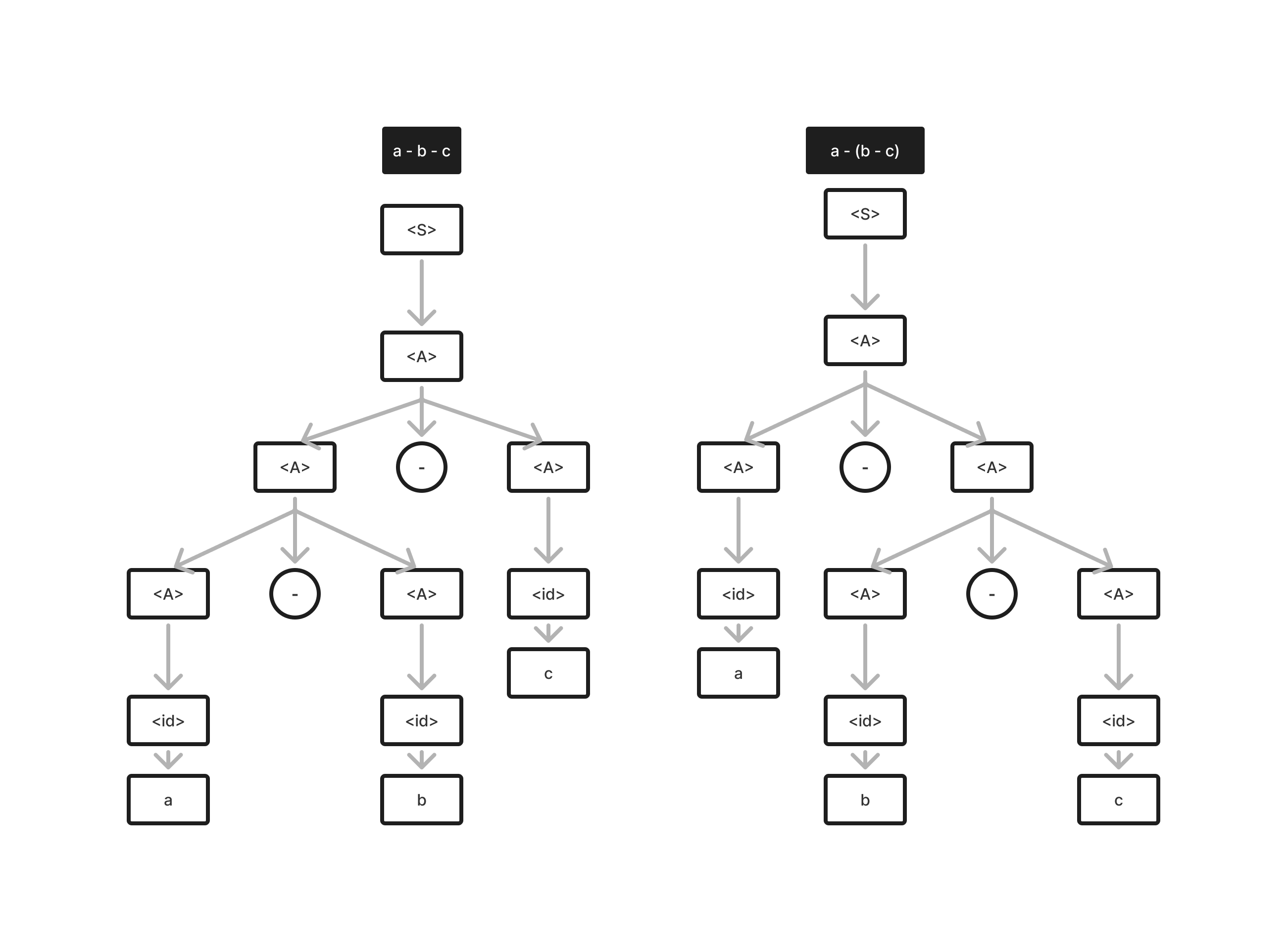
<A>  <A> - <A> | <id>

<id>  a | b | c

1. [5] Modify (1) to be the unambiguous grammar in the simplest way.

===================BEGIN ANSWER Q5===================

1. Consider `a – b – c`. Using the ambiguous grammar, there are two possible parse trees:



2. To make the grammar unambiguous, we must enforce associativity.

Left associativity:

<S>  <A>

<A>  <A> - <id> | <id>

<id>  a | b | c

Right associativity:

<S>  <A>

<A>  <id> - <A> | <id>

<id>  a | b | c

====================END ANSWER Q5===================

Q6. [10] Compute the weakest precondition for the following assignment statements and postconditions:

a = 3 \* (2 \* b - a);

b = 2 \* a – 1;

{b > 5} - postcondition

===================BEGIN ANSWER Q6===================

b > 5 → 2 \* a – 1 > 5 → 2 \* a > 6 → a > 3

a > 3 → 3 \* (2 \* b – a) > 3 → 2 \* b – a > 1

From the text (3.5.3.4): “The inference rule states that to get the sequence precondition, the precondition of the second statement is computed. This new assertion is then used as the postcondition of the first statement, which can then be used to compute the predocondition of the first statement, which is also the precondition of the whole sequence.”

Weakest precondition: {2 \* b – a > 1}

====================END ANSWER Q6===================

Q7. [10] Compute the weakest precondition for the following selection statement and postconditions:

**if** (x < y)

x = x + 1

**else**

x = 2 \* x

{x < 0} - postcondition

===================BEGIN ANSWER Q7===================

postcondition: {x < 0}

x < 0 → x + 1 < 0 → x < -1

x < 0 → 2 \* x < 0 → x < 0 (stronger)

Testing:

x = -0.5

-0.5 < -1 FALSE

-0.5 < 0 TRUE

x = -1.5

-1.5 < -1 TRUE

-1.5 < 0 TRUE

Therefore, x < -1 is stronger

Weakest postcondition: x < -1

====================END ANSWER Q7===================

Q8. [10, optional] Prove the following program is correct by applying Axiomatic Semantics:

{ *n* > 0 } - precondition

count = *n*;

sum = 0;

**while** count  0 **do**

sum = sum + count;

count = count  1;

**end**

{sum = 1 + 2 + …. + *n*} - postcondition