2. **Grammars: Synthetic Examples**.

(a) Consider the following grammar:

A ::= *A* & *A* | *V*

V ::= a | b

Recall that a grammar defines inductively a set of syntactic objects (i.e., a language). We can also use judgments to define a language.

For this exercise, rewrite this grammar using the following two judgment forms:

A ∈ **AObjects** meaning Syntactic object *A* is in the set **AObjects**.

V ∈ **VObjects** meaning Syntactic object *V* is in the set **VObjects**.

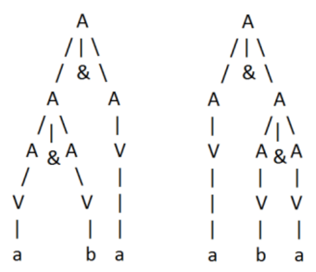
(A_1 & A_2) ∈ AObjects OR V ∈ AObjects

A ∈ VObjects OR b ∈ VObjects

(b) Show that the grammar in the previous part is ambiguous.

The statement is ambiguous because we can interpret multiple parse trees for the same situations. For example:

Parse Trees for: a & b & a



(c) Describe the language defined by the following grammar:

S ::= *A* | *B* | *C*

A ::= a *A* | a

B ::= b *B* | *ε*

C ::= c *C* | c

Syntactic objects A, B, OR C are elements in the set of SObjects.

A ∈ SObjects OR C ∈ SObjects OR B ∈ SObjects

Any series of a, greater than or equal to 1, is within the set AObjects

(a A) ∈ AObjects OR a ∈ AObjects

Any series of b, greater than or equal to 0, is within the set BObjects, *ε* is an empty character

(b B) ∈ BObjects OR *ε* ∈ AObjects

Any series of c, greater than or equal to 1, is within the set CObjects

(c C) ∈ CObjects OR c ∈ CObjects

(d) Consider the following grammar:

S ::= *A* a *B* b

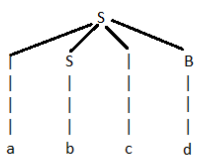
A ::= *A* b | b
B ::= a *B* | a
Which of the following sentences are in the language generated by this grammar? For the sentences that are described by this grammar, demonstrate that they are by giving **derivations**.
1. baab

```
--------------    --------------    --------------    --------------
b ϵ Aobject       a ϵ Sobject       a ϵ Bobject       b ϵ Sobject
--------------    --------------    --------------    --------------
A ϵ Sobject                         B ϵ Sobject
-------------------------------------------------------------------------------
b                 a                 a                 b ϵ Sobject     =>     baab ϵ Sobject
```

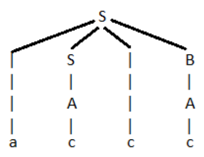
2. bbbab Impossible in this grammar
3. bbaaaaa Impossible in this grammar
4. bbaab

```
--------------    --------------    --------------    --------------
b b ϵ Aobject     a ϵ Sobject       a ϵ Bobject       b ϵ Sobject
--------------    --------------    --------------    --------------
A ϵ Sobject                         B ϵ Sobject
-------------------------------------------------------------------------------
b b               a                 a                 b ϵ Sobject     =>     bbaab ϵ Sobject
```

(e) Consider the following grammar:
S ::= a *S* c *B* | *A* | b
A ::= c *A* | c
B ::= d | *A*
Which of the following sentences are in the language generated by this grammar? For the sentences that are described by this grammar, demonstrate that they are by giving **parse trees**.
1. abcd



2. acccbd Impossible in this grammar
3. acccbcc Impossible in this grammar
4. acd Impossible in this grammar
5. accc



3. **Grammars: Understanding a Language**.
(a) Consider the following two grammars for expressions *e*. In both grammars, *operator* and *operand* are the same; you do not need to know their productions for this question.

e ::= *operand* | *e operator operand*
e ::= *operand esuffix*
esuffix ::= *operator operand esuffix* | ε
i. Intuitively describe the expressions generated by the two grammars.
ii. Do these grammars generate the same or different expressions? Explain.

i. Both of the e grammars describe different possibilities:
The first e means the set e contains operand OR the set e contains the statement e operator operand
The second e means the set e contains the statement operand esuffix, where esuffix is the statements operator operand esuffix, a recursive statement, OR empty character, ε, the base case.

ii. Though they differ a bit, the results of both of these grammars are exactly the same, if operand and operator are the same. We consider ε an empty/null character, which terminates the grammar without a contribution to the result. With this in mind, both will recursively call operand operator operand… with operand being the smallest case output.

(b) Write a Scala expression to determine if - has higher precedence than << or vice versa. Make sure that you are checking for precedence in your expression and not for left or right associativity. Use parentheses to indicate the possible abstract syntax trees, and then show the evaluation of the possible expressions. Finally, explain how you arrived at the relative precedence of - and << based on the output that you saw in the Scala interpreter.

expressions	*e* ::= *x* | *n* | *b* | *str* | **undefined** | *uope*1 | *e*1 *bop e*2 | *e*1 ? *e*2 : *e*3 | **const** *x* = *e*1; *e*2 | **console**.**log**(*e*1)
values	*v* ::= *n* | *b* | **undefined** | *str*
unary operators	*uop* ::= - | !
binary operators	*bop* ::= , | + | - | * | / | === | !== | < | <= | > | >= | && | ||
variables	*x*
numbers (doubles)	*n*
booleans	*b* ::= **true** | **false**
strings	*str*

Figure 1: Abstract Syntax of JAVASCRIPTY

statements	*s* ::= **const** *x* = *e* | *e* | { *s*1 } | ; | *s*1 *s*2
expressions	*e* ::= … | ~~**const** *x* = *e*1; *e*2~~ | (*e*1)

Figure 2: Concrete Syntax of JAVASCRIPTY

I’ve determined that – operator has higher precedence over << using this expression:

println(69 - 34 << 8 - 1) = 4480 ➔ println(35 << 7) = 4480

If << had precedence, then it would have processed 34 << 8 first, which would be:

println(69 - 34 << 8 - 1) = -8636 ➔ println(69 - 8704 – 1) = -8636

(c) Give a BNF grammar for floating point numbers that are made up of a fraction (e.g., 5.6 or 3.123 or -2.5) followed by an optional exponent (e.g., E10 or E-10). The exponent, if it exists, is the letter ‘E’ followed by an integer. For example, the following are floating point numbers: 3.5E3, 3.123E30, -2.5E2, -2.5E-2, and 3.5. The following are not examples of floating point numbers: 3.E3, E3, and 3.0E4.5. More precisely, our floating point numbers must have a decimal point, do not have leading zeros, can have any number of trailing zeros, non-zero exponents (if it exists), must have non-zero fraction to have an exponent, and cannot have a ‘-’ in front of a zero number. The exponent cannot have leading zeros. For this exercise, let us assume that the tokens are characters in the following alphabet ∑:

∑ def= {0,1, 2, 3, 4, 5, 6, 7, 8, 9,E, -, .}

Your grammar should be completely defined (i.e., it should not count on a non-terminal that it does not itself define).

A ::= -XS.N E NZ | -XS.N E -NZ | XS.N E NZ | XS.N E -NZ | XS.SZ | -XS.SZ | YS.XZ | -YS.XZ
// Example for First: (-2.5E2) Second: (-0.5E-2)
// Example for Third: (3.123E30) Fourth: (0.123E-30)
// Example for Fifth: (0.0, 0.1) Sixth: (0.0, -0.1)
// Example for Seventh: (3.5) Eighth: (-3.5)
// E is NOT an Object, it is the exponent

Z ::= 0 | ZZ | ε
// Possible Z cases: (0, 00, 000, …)
// Set of all chains of zeros, of any size.

X ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
// Possible X cases: (0, 1, 2, …, 9)
// Set of ONLY 0 to 9, for single digit numbers

Y ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
// Possible Y cases: (1, 2, 3, …, 9)
// Similar set to X, used for negative numbers that don't allow for a leading zero

S ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | SS | ε
// Set of all positive integers, can have a leading zero

N ::= 1Z | 2Z | 3Z | 4Z | 5Z | 6Z | 7Z | 8Z | 9Z | NN | ε
// Set of all positive integers, does not allow for a leading zeros.