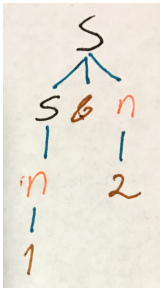
1. **25 points : Understanding Grammars** :: Each of the sub-questions of question 1 is dependent on the following grammar, Grammar 1 :

   Grammar 1 :

   S ::= n | S & n | n * S

   n is meta-variable for a number

   a. **7 points :** The sentence "1 & 2" exist in the language defined by Grammar 1. Use the space below to draw a parse tree to prove that the sentence exists in the language.

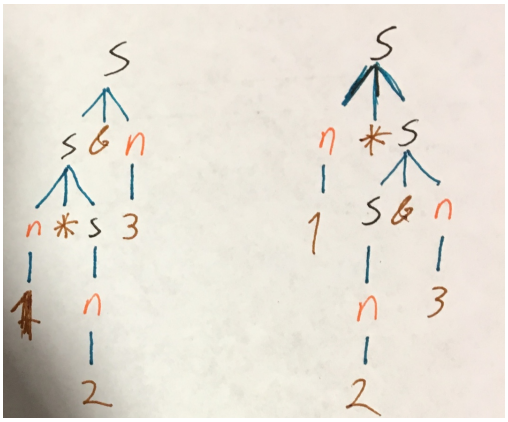
   

   b. **5 points :** Grammar 1 is ambiguous. Identify a sentence is ambiguously defined by the grammar.

   (For your reference: Since the second and third production combined make the grammar ambiguous. I likely need to use both productions. Paying close attention the * will need to come first to create ambiguity)

   1 * 2 & 3

   c. **12 points :** Demonstrate the ambiguity of Grammar 1 by writing two parse trees for the sentence identified in question 1.b.

2. **20 points : Writing inference rules::** Write a set of inference rules that operates on Grammar 2 (a language over linked lists) using the operational semantic `(S).append(n) => ***S'`.** Here '().append()=>' is the judgment form that we could view as ' (Input1).append(Input2) => Output'. The output ***S'*** should represent the result of appending a n to the input list S. Below, we have provided a few judgments that your inference rules should be able to derive.

Grammar 2 :

S ::= n -> S | End

n is a meta-variable for numbers

Example judgments : (you do NOT need to derive these)

i. (1->End).append(2) => 1->2->End
ii. (3->5->End).append(1) => 3->5->1->End

(For your reference: there potentially many ways to write this.)

(one possible solution. Note that in R2 I do not need to subscript my S, I just choose to do so. The output of the premise in R2 should be labels with S<stuff> as it is here with the <suff> $_2$'…. This is because the operational semantic specifies that the output of (S).append(n) => should exist in `S`)

R1----------

(End).append(n) => n -> End

(S_2).append(n) => S_2'

R2----------

(n_1 -> S_2).append(n) => n_1 -> S_2'

(another solution on the next page)

(another possible solution… this is possible because I am maintaining my sets.)

$$\text{R1}\ \frac{S' = n \rightarrow \text{End}}{(\text{End}).\text{append}(n) \Rightarrow S'}$$

$$\text{R2}\ \frac{(S_2).\text{append}(n) \Rightarrow S_2' \qquad S' = n_1 \rightarrow S_2'}{(n_1 \rightarrow S_2).\text{append}(n) \Rightarrow S'}$$

3. **20 points: Performing Derivations::** Consider the information provided bellow, use it to complete question 3.a. and 3.b.

Grammar 3 (binary tree):

S ::= X | (S) <- n -> (S)

Operational semantics: **Σ** S = n
Rules of Inference:

Sum_X

$$\frac{}{\Sigma\ X = 0}$$

Sum_SnS

$$\frac{\Sigma\ S_1 = n_1 \qquad \Sigma\ S_2 = n_2 \qquad n_{sum} = n_1 + n_2 + n}{\Sigma\ (S_1) \text{ <- } n \text{ -> } (S_2) = n_{sum}}$$

a. **10 point:** Using the inference rules provided, derive that
` **Σ** ((X)<- 5 -> (X))<- 10 -> (X) = 15`

(NOTE: you do not have to label the horizontal lines… I just like to do it)

$$\dfrac{\dfrac{\dfrac{}{\Sigma\ X = 0} \quad \dfrac{}{\Sigma\ X = 0}}{\Sigma\ (X) \text{<- } 5 \text{ -> } (X) = 5} \quad \dfrac{}{\Sigma\ X = 0}}{\Sigma\ ((X) \text{<- } 5 \text{ -> } (X)) \text{<- } 10 \text{ -> } (X) = 15} \quad 5 = 0 + 0 + 5 \quad 15 = 5 + 0 + 10$$

a. **10 points:** Complete the following subparts using the information provided at the top of in question 3:
   i. **2 points:** Identify a sentence representing a binary tree with 3 data points that exists in the language defined by Grammar 3

   ((X)<-1->(X))<-2->((X)<-3->(X))

   ii. **3 points:** If I apply the operational semantic '**Σ** S = n' to the sentence identified in question 3.b.i. what the value of n would be?

   6

   iii. **5 points**: Prove your solution to question 3.b.ii.

To save space I've created some extra variables. I suppose this is okay on the exam… But I would avoid it

(NOTE: you do not have to label the horizontal lines… I just like to do it … Here I didn't do it… above I did))

Let SubDeriv0 =

---------- ----------
Σ X = 0 **Σ** X = 0 1 = 0 + 0 + 1

Σ (X)<-1->(X) = 1

Let SubDeriv1 =

---------- ----------
Σ X = 0 **Σ** X = 0 3 = 0 + 0 + 3

Σ (X)<-3->(X) = 3

SubDeriv0 SubDeriv1 6 = 1 + 3 + 2

Σ ((X)<-1->(X))<-2->((X)<-3->(X)) = 6

4. **35 points: Coding::** All subparts of question 4 use the information provided on the last sheet of the exam. Feel free to tear off that sheet if you have not already. You can staple it back to your exam upon submission.

    a. **10 points :** In Scala, implement the logic of the provided inference rules over the operational sematic `S.max() = n` by writing a function named **'max`** that takes as input something of type **'S'** and returns something of type **'Int`**.

```
( first step, I like to declare the function )
def max(s:S):Int


(then I like to define the function)
(NOTE:: you do not have to provide the comments as I did)
def max(s:S):Int = s match {
        case Nil => ???                 // max_e
        case Node(_, n, Nil) => n       // max_Sne
        case Node(_, _, s2) => max(s2)  // max_SnS
}
```

    b. **10 points :** In Scala, implement the logic of the provided inference rules over the operational sematic `S.min() = n` by writing a function named **'min`** that takes as input something of type **'S'** and returns something of type **'Int`**.

```
def min(s:S):Int = s match {
        case Nil => ???                 // min_e
        case Node(Nil, n, _) => n       // min_enS
        case Node(s1, _, _) => min(s1)  // min_SnS
}
```

c. **15 points :** In Scala, implement the logic of the provided inference rules over the operational sematic `S.ordered() = b` by writing a function named **'ordered`** that takes as input something of type **'S'** and returns something of type **'Boolean`.** You may call the functions defined in 4.a. and 4.b.

(technically there was a typo in the provided inference rules… let me know if you figure out what that was)

```
def ordered(s:S):Boolean =  s match {
        case Nil => true                        // ordered_e
        case Node(s1, n ,s2) => {               // ordered_SnS
                val n1 = min(s1)
                val n2 = max(s2)
                if (n1 < n && n >= n2) {
                        val b1 = ordered(s1)
                        val b2 = ordered(s2)
                        val b = b1 && b2
                        b
                } else { false }
        }
}
```

Grammar 4 :

S ::= e | (S) <- n -> (S)

Meta-variables : n denotes the language of computer Integers. b denotes Booleans.

Operational Semantics : S.ordered() = b S.max() = n S.min() = n

Inference rules :

ordered_e $\dfrac{}{\text{e.ordered()} = \text{true}}$

ordered_SnS $\dfrac{S_1.\text{min}() < n >= S_2.\text{max}() \quad S_1.\text{ordered}() = b_1 \quad S_2.\text{ordered}() = b_2 \quad b = b_1 \,\&\&\, b_2}{S_1 \text{ n } S_2 \text{ .ordered()} = b}$

min_e $\dfrac{}{\text{e.min()} = ???}$ min_enS $\dfrac{}{\text{e n S .min()} = n}$ min_SnS $\dfrac{S_1.\text{min}() = n_1}{S_1 \text{ n } S_2 \text{ .min()} = n_1}$

max_e $\dfrac{}{\text{e.max ()} = ???}$ max_Sne $\dfrac{}{\text{S n e .max ()} = n}$ max _SnS $\dfrac{S_2.\text{max}() = n_2}{S_1 \text{ n } S_2 \text{ .max()} = n_2}$

AST definition:

```
/* S */
sealed abstract class S
/* e */
case object Nil extends S
/*  (Sl) <- n -> (Sr) */
case class Node(l:S, d:Int, r:S) extends S
```

A few examples of parsing: The sentence ((e)<-1->(e))<-2->((e)<-3->(e)) is exists in the language defined by our grammar and can be represented as an AST of type S as
Node(Node(Nil,1, Nil),2,Node(Nil,3, Nil))