**Midterm Questions**

## **Question 1 (15 points):** Complete the NFA (adding transition functions) based on the following requirements.

|  |  |  |
| --- | --- | --- |
|  |  | *States* |
|  |  | *Alphabet* |
|  |  | *Transition Function* |
|  |  | *Start State* |
|  |  | *Set of Accept States* |

Diagram

Description automatically generated

## **Question 2 (15 points)**: Use regular language pumping lemma to prove that the language is not regular.

|  |
| --- |
|  |

If we assume that is regular, the Pumping Lemma definition tells us that any string in can be ‘pumped’ at least a ‘pumping length’ of , then divided into three pieces . For it to be regular, it must satisfy the following conditions:

We can test these conditions by doing the following:

* Assume A is regular
* Letbe the pumping length
* Choose a stringto test
  + As we assume A is regular, is a member of
  + Because , and , then and therefore can be split into three pieces
* For conditions (2) and (3) to be met, piece must contain only ’s
  + For example, consider
  + For *,* can only contain ’s

|  |
| --- |
|  |
|  |

* According to condition (1) , for any instance ,
  + assume
  + Visual Representation

|  |
| --- |
|  |
|  |

* + - However, there is no possible division of this string that will result in the required format. This also remains true as increases as additional zeros are placed at the beginning of the string, creating a further imbalance between the beginning and end of the string (which should be identical ’s). Therefore, languagedoes not meet condition (1) for Pumping Lemma.

Since not all pumping lemma conditions are met, the assumption that A is regular is contradicted and proves that A is not regular.

## **Question 3 (20 points):** Categorize the following languages (note: provide answers only, no need to proof):

|  |  |  |  |
| --- | --- | --- | --- |
| Regular languages: | a | | |
|  |  | | |
| Context-free languages but non-regular languages: | | | c, d |
|  | | |  |
| Non-context-free languages: | | b, e | |

## **Question 4 (20 points):** Answer each part for the following context-free grammar G.

|  |
| --- |
|  |
|  |
|  |
|  |

### What are the variables of G?

### What are the terminals of G?

### Which is the start variable of G?

### Give three strings in L(G).

#### 

|  |  |
| --- | --- |
|  | starting variable |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  | starting variable |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
|  | starting variable |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Give three strings not in L(G).

### True or False:

* False, can only derive , or in a single step

### True or False: .

#### True, as the definition for states the following:

|  |
| --- |
| derives , if or if:  a sequence exists for , and |
| *(Sipser, pg 103)* |

* The sequence that satisfies this condition is listed below

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### True or False: .

* False, can only derive , or in a single step

### True or False: .

#### True, as the definition for states the following:

|  |
| --- |
| derives , if or if:  a sequence exists for , and |
| *(Sipser, pg 103)* |

* Therefore, if , the zero-step derivation meets this condition.

### True or False:

* False, there are no series of steps where can derive

## **Question 5 (15 points):** Give context-free grammars that generate the languages

### Define

|  |  |  |
| --- | --- | --- |
| CFG 4-tuple | | |
|  | Variables *finite set* |  |
|  | Terminals *finite set, disjoint from* |  |
|  | Rules *finite set, variable and string of variables/terminals* |  |
|  | Start Variable |  |

### Construct First Grammar for

### Construct Second Grammar for

### Merge rules to get the CFG

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

## **Question 6 (15 points)**: Show that is ambiguous. Let <STMT> be the following grammar.

|  |  |  |
| --- | --- | --- |
| CFG 4-tuple | | |
|  | Variables | { <STMT>, <IF-THEN>, <IF-THEN-ELSE>, <ASSIGN> } |
|  | Terminals | {if, condition, then, else, a:=1} |
|  | Rules | *listed below* |
|  | Start Variable | <STMT> |

|  |  |  |
| --- | --- | --- |
| <STMT> |  | <ASSIGN> | <IF-THEN> | <IF-THEN-ELSE> |
| <IF-THEN> |  | if condition then <STMT> |
| <IF-THEN-ELSE> |  | if condition then <STMT> else <STMT> |
| <ASSIGN> |  | a:=1 |

if condition then if condition then if condition then a:=1 else a:=1

The string above shows that is ambiguous, as it can be derived multiple ways. Each instance of ambiguity, including their derivation and parsing tree, is shown below.

### First Instance

#### Derivation

|  |  |
| --- | --- |
| <STMT> | starting variable |
| <IF-THEN> | <STMT> <IF-THEN> |
| if condition then <STMT> | <IF-THEN>if condition then <STMT> |
| if condition then <IF-THEN> | <STMT> <IF-THEN> |
| if condition then if condition then <STMT> | <IF-THEN>if condition then <STMT> |
| if condition then if condition then <IF-THEN-ELSE> | <STMT> <IF-THEN-ELSE> |
| if condition then if condition then if condition then <STMT> else <STMT> | <IF-THEN-ELSE> if condition then <STMT> else <STMT> |
| if condition then if condition then if condition then <ASSIGN> else <STMT> | <STMT> <ASSIGN> |
| if condition then if condition then if condition then a:=1 else <STMT> | <ASSIGN> a:=1 |
| if condition then if condition then if condition then a:=1 else <ASSIGN> | <STMT> <ASSIGN> |
| if condition then if condition then if condition then a:=1 else a:=1 | <ASSIGN> a:=1 |

#### Parsing Tree

Diagram

Description automatically generated

### Second Instance

#### Derivation

|  |  |
| --- | --- |
| <STMT> | starting variable |
| <IF-THEN-ELSE> | <STMT> <IF-THEN-ELSE> |
| if condition then <STMT> else <STMT> | <IF-THEN-ELSE> if condition then <STMT> else <STMT> |
| if condition then <IF-THEN> else <STMT> | <STMT> <IF-THEN> |
| if condition then if condition then <STMT> else <STMT> | <IF-THEN>if condition then <STMT> |
| if condition then if condition then <IF-THEN> else <STMT> | <STMT> <IF-THEN> |
| if condition then if condition then if condition then <STMT> else <STMT> | <IF-THEN>if condition then <STMT> |
| if condition then if condition then if condition then <ASSIGN> else <STMT> | <STMT> <ASSIGN> |
| if condition then if condition then if condition then a:=1 else <STMT> | <ASSIGN> a:=1 |
| if condition then if condition then if condition then a:=1 else <ASSIGN> | <STMT> <ASSIGN> |
| if condition then if condition then if condition then a:=1 else a:=1 | <ASSIGN> a:=1 |

#### Parsing Tree

Diagram

Description automatically generated