**Homework 3 Questions**

# **Problem 1**: (20 points) Use CKY Algorithm to decide if is in if the grammar for G is:

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

## Completed CKY Triangular Table:

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

## Steps using CKY Algorithm:

### Row 1

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

### Row 2

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

### Row 3

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

### Row 4

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

### Row 5

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

## Conclusion:

Using the CKY algorithm and triangular table, I was able to prove that the string is in . After successfully populating the table, I can verify . Since the start symbol,, then .

# **Problem 2**: (20 points) In each section below, give the sequence of configurations that Turing Machine enters when started on the indicated input string.

Diagram

Description automatically generated

|  |  |
| --- | --- |
| 7-Tuple () | |
|  | Set of States |
|  | Input Alphabet |
|  | Tape Alphabet |
|  | Start State |
|  | Set of Accept States |
|  | Set of Reject States |

## - Rejected

### (starting state)

A picture containing table

Description automatically generated

### 

A picture containing table

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

A picture containing table

Description automatically generated

### 

Table

Description automatically generated

### 

Table

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### Halted, invalid input for , Enters

## - Accepted

### (starting state)

A picture containing diagram

Description automatically generated

### 

A picture containing diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Table

Description automatically generated with medium confidence

### 

Table

Description automatically generated

### 

Table

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Table

Description automatically generated with medium confidence

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Diagram

Description automatically generated

### 

Table

Description automatically generated

# **Problem 3**: (10 points) What is Turing recognizable? What is Turing decidable? What is the difference between them?

Language is considered Turing Recognizable if there exists a Turing Machine that determines if a given input string is a member of the language or not by accepting and halting, rejecting and halting, or looping. Of these three possible outcomes, if , then halts in and if , either halts in or enters a loop and does not halt. The language recognized by the Turing Machine is denoted as .

The terms Recursive Enumerability and Turing recognizability are often used interchangeably due to the Turing Machine adaptation, the enumerator. An enumerator is a Turing Machine with an attached printer where it produces, or enumerates, strings to an output tape. Language is considered Turing Recognizable “if and only if some enumerator enumerates it.” In other words, is Turing Recognizable if and only if there exists a Turing Machine that accepts and halts on all strings in and loops or rejects all strings not in .

However, this concept has its limitations. In some cases, it can be difficult to determine if a machine is looping or is still computing. This problem arises because there is no way to predict how long it will take for a machine to solve a problem. For example, if an enumerator’s output tape does not contain string , there is no possible way to distinguish if the machine is still processing or if is not part of the language recognized by the machine.

Turing Decidability arises precisely to address this challenge. Language is considered Turing Decidable if there exists a Turing Machine that determines if a given input string is a member of the language or not, halting in a state of accept or reject. Of these two possible outcomes, if , then halts in and if , halts in .

The main difference between Turing Decidable and Turing Recognizable is that Turing Decidable machines will always halt and ‘decide’ a definitive answer on all inputs, rather than continuing to loop or process indefinitely. As a result, the concept of Turing Decidability can be seen as a stronger condition than Turing Recognizability as the former can determine whether an input belongs to a language or not, whereas the latter may not necessarily provide a definite answer for all cases.

It is important to note that the concept of Turing Decidability is a subset of Turing Recognizability, which means that every Turing decidable problem is also Turing recognizable. However, it is essential to note that not all problems are Turing decidable. Turing Recognizability is a broader concept that encompasses all problems that can be recognized by a Turing Machine, regardless of whether they can be decided by such machines or not. As such, it is crucial to understand the subtle differences between these two concepts and their implications in the field of computation theory.

# **Problem 4**: (10 points) Answer each part for the following Language

|  |
| --- |
|  |

## Is a regular language?

By applying the pumping lemma for regular languages, I successfully proved that is not a regular language. I began the proof under the assumption that was a regular language and then identified a contradiction in the necessary conditions. As this assumption led to a contradiction, does not satisfy the conditions required for a language to be considered regular.

### Proof

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:

|  |  |
| --- | --- |
| (1) |  |
| (2) |  |
| (3) |  |

We can test these conditions by doing the following:

* + Assume is regular
  + Letbe the pumping length
  + Choose a string in language to test:
    - As we assume is regular, is a member of
    - Because , and , then
    - Since both statements above are true, the Pumping Lemma definition tells us that can be split into three pieces
  + For conditions (2) and (3) to be met, piece must contain only ’s
    - For example, consider

|  |
| --- |
|  |
|  |

* + - To split , with *,* may only contain ’s

|  |
| --- |
|  |
|  |

* According to condition (1)
  + - assume
    - Visual Representation

|  |
| --- |
|  |
|  |

* + - * However, there is no possible division of that will result in the required format. This is evident in the visual representation above as it has the format and . As increases, this also remains true since additional ’s are placed at the beginning of the string, creating a further imbalance between and (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 is regular is contradicted and proves that is not regular.

Q.E.D

## Is a context free language?

By applying the pumping lemma for context-free languages, I successfully proved that is not a context free language. I began the proof under the assumption that was a context free language and then identified a contradiction in the necessary conditions. As this assumption led to a contradiction, does not satisfy the conditions required for a language to be considered context free.

### Proof

The Context- Free Pumping Lemma definition tells us that any string in can be ‘pumped’ at least a ‘pumping length’ of , then divided into five pieces . For it to be regular, it must satisfy the following conditions:

|  |  |
| --- | --- |
| (1) |  |
| (2) |  |
| (3) |  |

We can test these conditions by doing the following:

* + Assume is a context-free language (CFL)
  + Letbe the pumping length
  + Choose a string in language to test:
    - As we assume is CFL, is a member of
    - Because , and , then
    - Since both statements above are true, the Pumping Lemma definition tells us that can be split into five pieces

### Case 1: and contain only one type of symbol

* + - Testing conditions (2) and (3) under Case 1
      * assume

|  |
| --- |
|  |
|  |

* Split , with

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

* + - Testing condition (1) under Case 1
      * assume
* Visual Representation

|  |
| --- |
|  |
|  |

* + There is no possible division of that will result in the required format. This is evident in the visual representation above which contains the format and . As increases, this remains true since additional ’s are placed in the middle of the string, creating a further imbalance between the and (which should be equivalent ’s). Therefore, languagedoes not meet condition (1) for Pumping Lemma when split in such a way that and contain only one type of symbol.

### Case 2: Either or contain more than one type of symbol

* + - Testing conditions (2) and (3) under Case 2
      * assume

|  |
| --- |
|  |
|  |

* Split , with

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

* + - Testing condition (1) under Case 2
      * assume
* Visual Representation

|  |
| --- |
|  |
|  |

* + There is no possible division of that will result in the required format. This is evident in the visual representation above which contains the format as . As increases, this will remain true since contains more than one type of symbol; as is pumped, the string places ’s in the incorrect spot, disrupting the language format entirely. Therefore, languagedoes not meet condition (1) for Pumping Lemma when split in such a way that either or contain more than one type of symbol.

Since not all pumping lemma conditions are met in either Case 1 or Case 2, the assumption that is a CFL is contradicted and proves that is not context-free.

Q.E.D

## Is Turing recognizable? Justify your answer.

Yes, the language is Turing recognizable as there exists a Turing machine that accepts and halts on all input strings . An example of this Turing machine is below.

Diagram

Description automatically generated

## Is Turing decidable? Justify your answer.

Yes, the language is Turing decidable as there exists a Turing machine that accepts and halts on all input strings and either loops or rejects all input strings . As a reminder, Turing Decidability is a subset of Turing Recognizability. This implies that since machine halts and produces a definitive answer for all inputs, it meets the conditions for Turing Decidability, and subsequently, Turing Recognizability. A copy of this Turing machine is below.

Diagram

Description automatically generated

# **Problem 5**: (20 points) Let and . Answer each part and give a reason for each negative answer for the functions and

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

## Is one-to-one?

* + No, function is not one-to-one because multiple elements in maps to the same element in . For a function to be one-to-one, it must map each unique input from (denoted as and ) to its own unique output ( and). In other words, when . However, function violates this property because unique inputs , , and produce the same output value , and unique inputs and , produce the same output value . Thus, is not one-to-one because it maps multiple inputs to the same output value.

## Is onto?

* + No, function is not onto because each element in is not mapped to by . For a function to be onto, there must be at least one for every such that . Thus, function can not be onto as there are no input values which map to, , or .

## Is a correspondence?

* + No, function is not a correspondence. A correspondence is a function that is both one-to-one and onto; since does not meet both of these criteria, then it cannot be a correspondence.

## Is one-to-one?

* + Yes, function is one-to-one because every element in maps to a unique element in . Therfore, it satisfies the property that when .

## Is onto?

* + Yes, function is onto because every element in is mapped to by a unique element in and there are no elements in that are left unmapped.

## Is a correspondence?

* + Yes, functionis a correspondence. A correspondence is a function that is both one-to-one and onto; since meets both of these criteria, it is a correspondence.

# **Problem 6**: (20 points) Let . Show that is countable.

For a set to be countable, it must either be finite or it have the same size as. , the set of all ordered triples where , , and are natural numbers, is clearly infinite as it contains every possible combination of , , and . However, it is not immediately clear whether is countable or uncountable. To determine this, we will use Cantor's method of size comparison, which is a useful tool for determining the relative sizes of infinite sets as it allows us to compare the sizes of sets that are too large to be counted directly. Cantor's method compares the cardinalities of two sets by establishing a one-to-one (injective) correspondence between their elements; if such correspondence exists between and the set of natural numbers , then they have the same size.

1. Create a Matrix of
   * + To analyze the size of the set , we need to consider all possible ordered triples where . To approach this methodically, we create a matrix of to list all these possible ordered triples in a table format, where each row represents a specific value of the first index . For example, the first row represents all ordered triples that start with 1, the second row represents all ordered triples that start with 2, and so on. By continuing the table in this manner, we ensure that every possible combination of of , , and will be included and appear in a unique cell. Therefore, we can use the table below to analyze the size of the set.

Table

Description automatically generated

1. Establish a correspondence between and the set of natural numbers
   * + Using the matrix above, we can create a correspondence table to begin mapping each element of T to a natural number N. As shown in the figure below, highlighted in green, our method will begin at the top left corner {1,1,1} and traverse the matrix in a zig-zag pattern, assigning natural numbers to the elements of T as we progress. The arrows demonstrate the specific route, which includes cells {1,1,2}, {2,1,1}, {3,1,1}, {2,1,2}, and so on." This approach ensures that every element of T will eventually be assigned to a natural number. If we were to start with just the first row of the matrix, we would never assign natural numbers to the elements in the subsequent rows, as there are infinitely many elements in the first row alone. The resulting correspondence table is also shown below.

A picture containing text

Description automatically generated

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

From the correspondence table above, it is clear that T is an injective function as every element in N maps to a unique element in T, and a surjective function as each element of T is assigned a unique natural number. Therefore, T is a bijection between N and T, and thus countable according to Cantor's method of size comparison. In conclusion, the zig-zag matrix T provides a bijective correspondence between T and the set of natural numbers, demonstrating that the set T is countable.