ToC Prep\Some Questons.md

Questions and Answers

1. Define null production

Ans: A null production is a production rule in a context-free grammar that doesn't produce any new output in terms of the symbols in the language being parsed. It is represented as epsilon (ε).

2. Define Recursively Enumerable Language

Ans: A recursively enumerable language (RE language) is a language where a "computable" algorithm can list all the strings in the language. Think of it like a language we can "read" through.

3. State Church Turing Thesis

Ans: The Church-Turing Thesis is a fundamental assertion in computer science. It states that any problem that can be solved by a Turing Machine (TM) can be solved by a program written in a Turing Machine's language. It essentially equates "computation" with any problem.

4. Define instantaneous description of TM. Give Example

Ans: Think of a TM as a 'black box' machine: It has a few key components. It has:

States: Each state is a different configuration of the TM.

Tape: The tape is the area the TM uses to store its information.

Head: The head is the pointer that scans the tape. Instantaneous: It can't remember the past, and the current state of the TM determines how the TM processes a string.

Example

```
Imagine a TM for checking if a string is a palindrome:
Initial state: The TM starts in a specific state.
Read the string: The head moves across the tape, one character at a time.
Check for matching: The TM looks at the character it is currently reading, and compares it to the character on the other side of the tape.
Move to next state: Depending on the match, the TM will move to the next state.
```

```
Repeat steps 2-4: This process continues until the end of the string
is reached.
```

5. Define left recursion

Ans: Left recursion is a type of grammar that's often used to define language patterns. The recursive rules can start at the beginning of the expression, and this is a common problem in compilers.

6. Design a CFG for the language L = {a^3 b^n c^n | n >= 0} and convert it into CNF

1. CFG Design

We can build this CFG in a recursive fashion:

```
S -> a^3
```

Explanation:

- `S` is our start symbol for the production.
- `a^3` means we need to produce a sequence of three a's.

2. CFG for L

The grammar for the language L can be described as follows:

- **Start symbol:** S
- **Non-terminal symbol:** a, b, c
- **Production rules:**
    - S -> a^3
    - B -> b^n where n >= 0
    - C -> c^n where n >= 0

3. Conversion to CNF

- **CNF (Chomsky Normal Form)** represents grammars in a simpler and more concise form, making it easier to analyze.

Step 1: Build a chart:

- The chart will represent all possible non-terminals, rules, and how they relate to each other.

Step 2: Apply a CFG reduction:

- Reduce each non-terminal to its simpler form.

7. Construct PDA for the Language L = {a^n b^3n | n>=1} acceptance by empty stack

Understanding the Language

The language L = {a^n b^(3n) | n >= 1} has a specific structure:

- Each string in the language consists of alternating ‘a’s and ’b’s in the specified pattern.

Constructing the PDA

The following PDA will accept the strings in L = {a^n b^(3n) | n >= 1}, where strings must be in a specific order:

States:

- **S:** Initial state
- **A:** State for accepting a string in the language.
- **E:** State for the final string in the language.

Input Symbols:

- **a:** The symbol used to denote an ‘a’ in the string.
- **b:** The symbol used to denote a ‘b’ in the string.

Transitions:

- **S -> a** : Move to the next state with a string that starts with an ‘a’
- **S -> b** : Move to the next state with a string that starts with a ‘b’
- **A -> S** : Move to the state S

Accept State:

- **A** is the acceptance state. The string will be accepted if it ends in an ‘A’.

Working of the PDA:

1. **Initialization:** The PDA starts in state S, looking at the first letter of the input string.
2. **Processing:** The PDA reads characters from the input string and performs actions based on the current state.
3. **Determining Acceptance:** After processing the entire input string, the PDA will be either in the accepting state or rejecting state. If it’s in the accepting state, the string is accepted; otherwise, it’s rejected.

8. Design a TM which recognize the language L = {a^r cd b^r | r > 0}

Understanding the Language

The language L comprises strings that consist of alternating repetitions of ‘a’ and ‘b’ based on the number ‘r’.

Building the TM

To design a TM for L, we’ll focus on:

1. **States:** We need a state to handle both ‘a’ and ‘b’ and transitions to handle their processing.
2. **Tape:** A fixed length tape that holds the input.
3. **Head:** A pointer that scans the tape.
4. **Accepting:** The TM will stop when the pattern is complete.

Steps

1. Defining States

- **S:** The start state.
- **A:** State representing “a”
- **B:** State representing “b”
- **E:** The accepting state.

2. Transition Table (T)

Input	Head	S	A	B	E
‘a’	‘a’	S	A		
‘b’	‘b’	S	A	B	
‘c’	‘c’		A	B	E

3. State Changes:

- The TM moves its head to the next symbol in the tape according to its input.

4. Acceptance: The TM is said to accept the input if the head reaches the accepting state.

Explanation:

- **Initialization:** The TM starts at the initial state ‘S’ and scans the first symbol of the input.
- **Processing:** The TM processes the input symbol and transitions to a new state.
- **Final State:** When it reaches the accepting state, the TM accepts the string.

Key Concepts:

- **TM:** This TM is designed to handle strings with alternating ‘a’, ‘b’, and ‘c’ symbols.
- **States:** States represent different parts of the TM’s internal representation.
- **Transitions:** Transitions represent how the TM moves from one state to another based on input and current state.

Design a TM to compute subtraction of two numbers

The Problem: We aim to design a TM that takes two numbers as input (represented by binary strings) and outputs the difference between them. The TM should also ensure that the input is valid.

Challenges:

- **Limited Memory:** TMs generally have a finite number of states.
- **Integer Representation:** How do we represent and store the numbers for subtraction?
- **Subtraction Logic:** We need a way to implement the standard subtraction logic.

TM Design

- **States:**
  - **S0:** Initial state
  - **S1:** State to handle the first number
  - **S2:** State to handle the second number
  - **S3:** State for the result.
- **Input Symbols:**
  - **0 and 1:** To represent digits of the numbers.

Steps:

1. **Initialization:** The TM begins in state S0 with the input numbers as strings.
2. **Processing:**
   - The TM reads the input symbols one by one from the tape.
   - Based on the current input and the states, the TM will move to a new state.
   - It then processes the current input symbol.
   - The TM compares the current numbers.
3. **Subtraction Logic:**
   - The TM uses a series of steps and transitions to find the difference between the numbers,
   - It will have rules for handling both positive and negative values.

Explanation of the TM * The TM will handle a binary representation of a number. * The TM will check to ensure that both inputs are valid.

Key Considerations:

- **Efficiency:** How to minimize the number of transitions required to compute the difference between two numbers.

10. Write a short note on Chomsky Hierarchy. Differentiate between Recognizer and Decider. Discuss TM as enumerator

The Chomsky Hierarchy

The Chomsky Hierarchy is a classification of languages based on their computational power and how easily we can determine if a language belongs to it.

- **The Hierarchy:** This hierarchy arranges languages according to their complexity, with the "strongest" languages at the bottom, and the weakest at the top.
- **Key Levels:**
    - **L(1):** Languages that can be recognized by a Turing Machine.
    - **L(2):** Languages that can be recognized by a Deterministic Finite Automaton (DFA).
    - **L(3):** Languages that can be recognized by a Context-Free Grammar (CFG)
    - **L(4):** Languages that can be recognized by a Context-Sensitive Grammar.
    - ...and so on.

Recognizing vs. Deciding

- **Recognizable Language (recognizer):** A language can be recognized if we can determine, in a deterministic way, if a string belongs to that language (without any ambiguity).
- **Decidability Language (decipher):** A language is decidable if we can determine whether a string belongs to that language, but we don't need to determine if it's valid.

Turing Machines as Enumerators

Turing Machines (TMs) can be considered enumerators. They can enumerate the strings within a language.

- **The Concept:** A TM, by default, accepts the string by scanning it. Turing machines are able to do this in a very systematic way.
- **Strengths:** They are capable of recognizing strings within a language in a systematic way, and they don't need to determine if the string is valid.

Key Differences

- **Recognition vs. Decipherability:** TMs can recognize (classify) languages, but not necessarily decipher them in a more strict sense.
- **Enumerability vs. Decidability:** TMs are enumerators by definition, but not necessarily decidable.