**Theoretical Computer Science – CTS**  
 **Notes  
Winter Semester 2024   
Aqeel, Muazzam Bin**

Definition 2.11 – Countable and uncountable sets  
A set M is called countably infinite, if it has the same cardinality as the set N of natural numbers, i.e. if there is a bijective function (1-to-1-mapping) f:M -> N where every element m ∈ M of the set is assigned to an enumeration position f(m) and for every position there is a value m ∈ M too

A set M is called uncountable, if it is infinite and not countable, i.e. there is no 1-to-1 mapping to the natural numbers

We can basically map the numbers from the set to natural numbers are called countable numbers

A set is uncountable if no such bijection exists  
Show that the set of all real numbers between 0 and 1 is countable

Communication should be more highlhted, maybe there should

**Countable Set Example:**  
Set of Natural Numbers between 0 to 1 – Because there are specific amount of entries that can be considered  
  
**Uncountable Set Example:**  
Set of Real Numbers between 0 to 1 --- Because we can always construct a new value  
  
Due to the Cantor’s diagonal argument, which shows that you can always construct a **new** real number that differs from each number in the list by changing the nth digit in the nth place (like creating a new number based on the diagonal)

Number

Computing the degree of a vertex from adjacency matrix representation:

**Chapter - Graphs**

**Undirected Graph**Count ones in the column (or row) of the vertex, count diagonal elements twice (loops count twice!)  
Count diagonal element twice, does this mean the same thing like that vertex having a link to themselves will be considered with a value two

A screenshot of a graph

Description automatically generated

**Directed graph**number of ones in row of vertex + number of ones in line of vertex.

How can you compute the degree of a vertex from the adjacency lists representation?  
 **Undirected Graph**length of adjacency list  
add 1, if the vertex is contained in its adjacency (for loop)

A white and red text with black text

Description automatically generated with medium confidence

**Directed Graph**Length of the adjacency list of the vertex (outdegree) + number of occurrences of the vertex in all adjacency lists (indegree)

Let G = (V, E) be a directed graph, where V = {1, 2, 3, 4} E = { (1,2), (1,4), (2,2), (2,3), (2,4), (3,1), (4,3) }   
  
a) Draw G as a diagram.

First question: any advice of drawing the diagram to follow the planer concepts, like can i connect with a curve.

b) Determine the indegree and the outdegree of vertex 2.

c) Is G planar?

This means that the graph can be drawn in the plane without crossing edges 🡪 plane What does plane mean, does it mean that the arrows need to be straight or can be connected with curve-based arrows, similar question in part a

d) Is G strongly connected?

This means that each vertex can be reached from each other vertex. All vertices should be able reach each other

e) Is G acyclic?

So, if a cycle exists, this case as there exist cycles such as: (1,2,3,1) or (1,2,4,1), so because of the cycles this will be called not acyclic.

Exercise 1.4

**Steps to Solve the "Knows" Problem (Exercise 1.4):**

1. **List the People**:
   * Write down the names or use abbreviations (Anna = A, Bruno = B, Cindy = C, Dave = D, Emma = E).
   * This helps keep the problem organized and makes it easier to visualize relationships.
2. **Identify the Person with the Most Connections**:
   * In the problem, Dave knows the most people (four). Start with him since this will lay a strong foundation for the graph.
   * **Explanation**: Starting with the person who has the most connections reduces the number of possible placements for other connections.
3. **Add Connections for Dave**:
   * Draw Dave (D) and connect him with the other four people. This forms the basic structure of the graph.
   * **Explanation**: Since Dave knows everyone, this step gives you the maximum number of connections, reducing uncertainty for others.
4. **Move to the Next Person with Fewer Connections**:
   * After Dave, move to Cindy (C), who knows three people.
   * Based on Dave's existing connections, add Cindy's connections to the graph.
   * **Explanation**: Handling the second most-connected person next helps refine the graph further and provides more structure to guide the remaining steps.
5. **Continue with Bruno**:
   * Bruno (B) knows two people. Add his connections, making sure that the relationships are symmetric (if Bruno knows Cindy, Cindy must know Bruno).
   * **Explanation**: Since fewer people are left to connect, this step should start becoming easier. You can only place Bruno's connections where they haven't been connected yet.
6. **Handle Anna's Connection**:
   * Anna (A) knows just one person. Based on the existing structure, determine Anna's connection, ensuring the "knows" relationship is symmetric.
   * **Explanation**: By this stage, most of the graph is already formed, and Anna's connection should be the final piece to add without ambiguity.
7. **Determine Emma’s Connections**:
   * Since everyone else’s connections are already set, the graph will reveal that Emma (E) knows two people. Ensure this fits within the remaining available connections.
   * **Explanation**: After placing all the other people, Emma's connections should naturally follow as the remaining unconnected relationships.
8. **Double-Check Symmetry**:
   * Ensure that all connections are symmetric, meaning if one person knows another, the reverse relationship is also true.
   * **Explanation**: Symmetry is a key rule in this problem, so verifying this ensures your solution is accurate.

**Key Explanation of the Strategy:**

* **Start with the most complex person**: Dave, having the most connections, simplifies the graph as most relationships become fixed early on.
* **Work in decreasing order of connections**: Handling people with fewer connections later reduces the complexity of choices.
* **Iterative correction**: If something seems wrong, adjusting along the way helps prevent errors from propagating throughout the graph.
* **Symmetry ensures accuracy**: The relation "knows" is symmetric, so checking for symmetry throughout avoids mistakes.

Exercise 1.5 – Obligatory

**A Eulerian Cycle (also known as Eulerian Tour):**  
**A Eulerian Cycle is a path in a graph that**

1. Starts and ends at the same vertex
2. Traverses every edge exactly once

**For a graph to have a Eulerian cycle, the following conditions must be met:**

1. All vertices must have an even degree (i.e., an even number of edges connected to each vertex)
   1. Edges are the connections, basically the lines, whereas the vertices are the nodes, like the points themselves
2. The graph must be connected (there must be a path between any two vertices).

If a graph satisfies these conditions, it is said to have a Eulerian Cycle

**A Eulerian Path is a path in a graph that:**

1. Traverses every edge exactly once, but does not necessarily start and end the same vertex

For a graph to have a Eulerian Path, the following conditions must be met

1. Exactly two vertices must have an odd degree
2. All other vertices must have an even degree
3. The graph must be connected

The path will start at one of the vertices with an odd degree and end at the other

Important points

* Yes, vertices can be visited multiple times as long as you use different edges each time.
* For an Eulerian Path, you must connect the two odd-degree vertices, and all edges must be traversed exactly once.

**Bipartite graph**

**S**

**Chapter - DFA / E-NFA**

**Computing all Equivalent states of the DFA (Algorithm 7.23)**

**Prerequisite:**Equivalent means:  
A state is considered equivalent to another state if, for all input strings, both states transition lead to the same type of state (accept or non-accept) with respect to a same input string for both state which defines equivalence behaviour.

**Steps**

1. Create a table
2. Write all the states from the graph in ascending order, vertically and horizontally
3. Cross out with alphabet “X” for the accepts states
4. Perform Passes (**Can be multiple**)
   1. **1st Pass**
      1. Look at the intersecting states (i.e., the states represented by the row and column of the empty box).
      2. Write down the states under your pass that have an empty box in their corresponding table entry. Example (b,c), (c,e)
      3. Now for each fair check if they are equivalent
         1. If they are not equivalent to each other, mark that empty box with a X1
   2. **2nd Pass**
      1. Only check those entries that had no new entries, in order words equivalent

**Minimizing DFA**

**Steps**

1. Check if there are any non reachable states **(Remove them)**
2. Check if there are any equivalent states **(Remove them)**

**Conversion e-NFA to DFA – Improved Subset Construction (Algorithm 8.19)**

**Steps**

1. **Create the Customized Table**
   1. List all states from the given e-NFA, even if a state appears multiple times in different sections for each input strings.
   2. For each string section
      1. For each state, write down the states that is reachable with this string
      2. **Special Case: Handling Empty Strings in the Transition Loop**
         1. If there is an epsilon (𝜀) transition (empty string) between two states and the same string (e.g., 𝑎) forms a part of the loop between these states:
         2. Consider the initial state as reachable through 𝑎 as well and write that in the final answer
      3. **Special Case: Handling Empty Strings**
         1. If a state transitions to another state via an input string (e.g. 𝑎) and the next state has an 𝜀 - transition to another state, include this additional state in **the final answer. 𝑎 𝜀**
         2. If a state transit firstly to a state through an empty string and then that state transit to a state where the input string (e.g. 𝑎) is there, then put only the state that produces the a string not the initial state that is connected with the current state you are considering to process, **the final answer 𝜀 𝑎 = a – Ignore the state that has 𝜀**  
            Example is in Problem 5)ii) TCTS\_Exam\_SS18
2. **Graph**
   1. **Initial State**
      1. From the initial state, identify and mark all states that are reachable through epsilon (𝜀) transitions (empty string transitions).
   2. **Using the Customized Table**
      1. In the initial state for all the states check for a specific string lets say a
         1. Using the result from the customized table, combine and make an answer by accumulating all the reachable states for the states identified from the initial state
      2. Repeat this process for all input strings (e.g., 𝑎 , 𝑏, etc.) to accumulate reachable states for each input.
      3. For the new states do the same step unless no new state is being created
      4. **Handling the Empty State**
         1. Create an empty state to represent cases where no states are reachable for a specific input string.
         2. If, after accumulating all reachable states for a given string, the result is empty, connect the current state to this empty state for that specific input string.

**Conversion Regular Expression – e-NFA (Procedure 9.1)**

**General Rules**

A diagram of a machine

Description automatically generated  
A diagram of a diagram

Description automatically generated

**Example**  
If it was just 1 not the star stuff then we would not have an empty string as there is no transition – Check before the first state small arrow no empty

A diagram of a diagram

Description automatically generated

**Conversion of e-NFA into Regular Expressions**

Steps

1. Add a new start state z and a new accept state z’
2. Connect with an e-transition from z to the older start state
3. Connect with an e-transition from z to the older accept states, so that z’ becomes the only single accept state
4. Apply elimination until you get one edge as z node to z’ node where, on the edge there is one regular expression itself
   1. **Lesson 16**  
      A diagram of a diagram

      Description automatically generated with medium confidence

**Chapter - PDA**

**Conversion from Context-Free Grammer to PDA**

PDA (Definition 10.13)

* **∑ = {a, b, c}** 
  + Includes all the Terminal Symbols
* **Γ = {a, b, c, A, B, k0}** 
  + k0 is the stacks start symbol that brings S in our case
* **Z = {z0, z1, ze}** 
  + zo is to process the start symbol,
  + z1 is used to consume all the terminal symbols
  + z1 is then also used to expand all the nonterminal symbol’s production, step by step
  + ze is used at the end to transit from z1 to ze to end the state if stack is empty

**Sequence of Steps that shows that the PDA accepts the string**

**Rules:**

1. **Use a Right-Hand Derivation Tree:**
   * It is recommended to create a right-hand derivation tree so that you do not struggle with the production steps.
   * Use right-side derivation for producing the string, as the stack processes data with the topmost element first.
2. **Add Comments:**
   * When expanding a nonterminal symbol, add a comment "expand".
   * When processing a terminal symbol, add a comment "consume". In the next step, remove that terminal symbol from the string after the comma.
3. **Steps:**
   * Push **k0**
   * Push **S**
   * Expand **S**
   * Now depedning on the first termial symbol on the most left side, consmue it
   * Repeat until you have consumed all the terminal symbols
   * Lastly push the empty string on the stack
   * Consume the empty string “**𝜀”** and at the same time change the state to **ze**

**Chapter - Nullable, First, Follow, Parsing Table  
  
Context Free Grammer to nullable, first and follow**A group of black text

Description automatically generated

**Step 1: (Nullable)**

1. Check for nonterminal symbols which can produce empty string
   1. Nonterminal symbols that led to another nonterminal symbol that produces an empty string is also considered to be nullable, e.g. like S (but if there is even one terminal symbol like this, (S 🡪 AAa) Its not then considered to be fully nullable

**Step 2: (First)**

1. **Step 2.1**   
   We need to through every production of every nonterminal symbol  
   *cBAc  
   AA  
   acS  
   e  
   Sb  
   Ab*
2. **Step 2.2**  
   For each production we need to just check the first nonterminal symbol that gets produced and put in the intermediate table
   1. If there is production where there is one possibility as empty or do we check from the table of column nullable
      1. Example S🡪 AA 🡪 empty  
         This creates and empty result
   2. If there is production where the first nonterminal symbol can be nullable but the second is a terminal symbol
      1. Example B🡪 Sb 🡪 AA 🡪 not empty 🡪 b  
         then the answer is b, because there is a terminal symbol
3. **Step 2.3**
   1. Repeat Step 2.2 at least once, and stop until you get the same data in the previous table result

**Step 3: (Follow)**

1. **Step1 (Inisde Positions)**
   1. We conisder all the productions, so in each production, where the non-terminal symbols have other terminal/nonterminal symbols (From Left to Right) then create a case for that nonterminal symbol  
        
      We dont check those **Non-Terminal** that are being processed as **follow**  
      1. **Scenario 1 – Here we dont check if a Non-Terminal Symbol is Nullable:**  
         Example S 🡪 **A**C 🡪 Follow (A) 🡪 add First (C)   
         Now Check From the latest table of first in the row of nonterminal symbol C
      2. **Scenario 2 – Where we Check if a Non-Terminal Symbol is Nullable:**Example
         1. S 🡪**A**CB  
            Example S 🡪 **A**CB 🡪 Follow (A) 🡪 add First (CB) 🡪 .....
            1. If C is nullable check B, if it has content in the first row of nonterminal symbol, add in follow of A table
         2. S 🡪A**C**B  
            Example S 🡪 A**C**B 🡪 Follow (C) 🡪 add First (B)
            1. Now Check From the latest table of first in the row of nonterminal symbol B
2. **Step2 (End Positions)**We conisder all the cases of each production but this time we write the nonterminal symbols that are in the LHS of the Production
   1. **1st Pass  
      Considering Every Production (Start From Exterme Right to Left):  
        
       Dont even process this**
      1. **If in the end position** (We have a terminal symbol) **Then** we dont conisder that Production  
         Example S 🡪AC**a**
      2. **If in end positon** (We have a Non-TerminalSymbol) **Then** we conisder that production  
         Example: S 🡪BA**C**
         1. If this letter is nullable, we consider the pervious Non-Terminalletter, If A is also nullable we conisder B  
            S 🡪BAC
         2. If its not nullable we stop
   2. **2nd Pass**
      1. Just repeat pass 1 to observe if there are any changes

**Creating a Prasing Table**