ICS 3103: AUTOMATA THEORY - ASSIGNMENT 2

BICS 3C

DONE BY:

SINGH SEHMI - 146254 - ICS3C

VASANI AMAN - 144914 - ICS3C

Contents

1 Short Notes on Turning Machines

Definition Turing Machines (TMs) are fundamental theoretical models crucial in automata theory. Developed by Alan Turing in the 1930s, TMs are abstract machines that simulate computation, providing a theoretical framework for understanding the limits of computability and the foundations of algorithmic processes.

Components TMs contain three main components such as a tape divided into cells capable of storing symbols, a head that can read and write symbols on the tape one cell at a time, and a state control unit that governs the machine's actions based on the current state and the symbol under the head.

Uses and Implementation TMs have undergone extensions and variants to explore different facets of computation. For example, Non-deterministic TMs permit multiple potential transitions from a given state and symbol combination, expanding the range of possible computations. Multi-Tape Turing Machines employ multiple tapes, facilitating more efficient algorithms for certain problems by allowing simultaneous operations

Conclusion Turing Machines often form the foundation of Automata Theory, providing a powerful and flexible model for understanding computation and its limits. By studying these machines, researchers can delve into areas such as the nature of algorithmic processes, computability, and the complexity of problems.

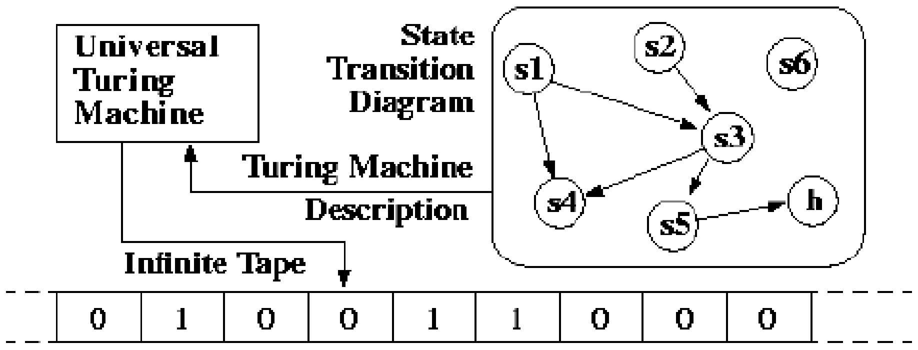


Figure 1: Typical Turing Machine

2 Illustrate the working of the Turing Machines

2.1 Multi-tape vs Single tape Turing Machines

2.1.1 Multi-Tape Turing Machines

Definition A multi-tape Turing Machine is a continuation of the original single-tape Turing Machine, featuring multiple tapes that operate in parallel. Each tape has its read/write head and is initially blank, except for the first tape, which holds the input. The tapes can interact with each other through the machine's transition rules.

How it Works The working process involves each head reading the symbol under it, followed by the state control unit analyzing the current state and symbols read by each head. This analysis determines the subsequent state and the symbols that must be written on the tapes. The heads then proceed to write the corresponding symbols on their respective tapes. Additionally, each head can move independently, either to the left or right, or to remain in place. This flexibility in movement allows the heads to traverse the tapes and interact with different symbols as required.

Example Consider a multi-tape Turing Machine designed to recognize the language L = $a^n b^n c^n \mid n \geqslant 0$. Suppose we have the input string "*abcbcbc*"

Initially, the first tape contains the input, while the other tapes are empty. The Turing Machine starts by reading an "a" from Tape 1 and writes an "X" on Tape 2. It then reads a "b" from Tape 1 and writes a "Y" on Tape 3. Moving forward, it encounters a "c" on Tape 1 and writes a "Z" on Tape 4. Afterward, the Turing Machine moves back to the leftmost positions on all tapes. It checks if the symbols on Tape 2, Tape 3, and Tape 4 match the pattern "X", "Y", and "Z", respectively. If they match, the Turing Machine erases the symbols and moves right on all tapes. Steps 3 to 7 are repeated until all symbols on Tape 1 are erased. Finally, if all tapes are blank, the input string "*abcbcbc*" is recognized to be in the language L; otherwise, it is not.

Complexity The multi-tape Turing Machine can be more efficient than a single-tape Turing Machine for certain problems due to its ability to perform simultaneous operations. It can reduce the time complexity for certain computations, making it more powerful in handling complex tasks. Hence, for the above example, the complexity would be O(m) + O(m) + O(m) = O(m).

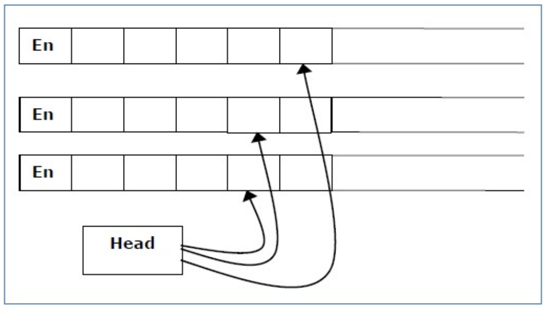


Figure 2: Multi-tape Turing Machine

2.1.2 Single-Tape Turing Machines

Definition A single-tape Turing Machine is a classic form of a Turing Machine, which serves as the fundamental model for computation. It consists of a single tape divided into discrete cells, where each cell can hold a symbol from a finite set. The tape extends infinitely in both directions, allowing for an unbounded amount of storage

How it Works A single-tape Turing Machine operates through four essential steps. First, the head of the machine reads the symbol in the cell beneath it on the tape. Then, the state control unit of the machine examines the current state and the symbol read, determining the following state and the symbol to be written on the tape. The head then writes the determined symbol onto the tape, overwriting the original symbol. Finally, the head can move one cell to the left or right or remain in its current position.

Example Consider the language L = $a^n b^n c^n \mid n \geq 0$. Using the initial input string ”aaabbbccc”:
The Turing Machine starts by reading the first symbol ”a” from the tape and replaces it with ”X”. It then moves right and encounters another ”a”, which is also replaced with ”X”. This process continues until all ”a”s in the input are replaced with ”X”. The Turing Machine then moves right and checks if the next symbol is ”b”. If it is indeed ”b”, the Turing Machine writes ”Y” in place of the ”b” and moves right. These steps are repeated until all ”b”s in the input are replaced with ”Y”. Similarly, the Turing Machine proceeds to replace all ”c”s in the input with ”Z” while moving right. Finally, if the tape is empty at the end of the process, it indicates that the input string ”aaabbbccc” is in the language L; otherwise, it does not belong to the language L.

Complexity The single-tape Turing Machine is less efficient for certain computations than a multi-tape Turing Machine. It may require more time and steps to perform complex operations, resulting in higher complexity. However, despite potentially increased computational overhead, the single-tape Turing Machine can still simulate any algorithmic process.

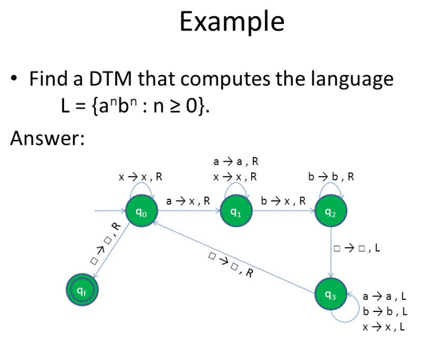
2.2 Deterministic vs Non-deterministic Turing Machines

2.2.1 Deterministic Turing Machines

Definition A deterministic Turing Machine (DTM) is a classical model that operates on a single tape, much like the single-tape Turing Machine. The DTM, however, differs in its transition rules, which are uniquely defined for each combination of the current state and symbol read. This means that given a particular state and symbol, there is only one specific next state and symbol to write. The DTM follows a precise and deterministic path through its computation, leaving no room for ambiguity.

How it Works The working mechanism begins with the head of the machine reading the symbol present on the tape under it. The state control unit of the machine then analyzes the current state and the symbol read to determine the next state and the symbol to write on the tape. The head proceeds to write this symbol on the tape. Additionally, the head can move either left or right or stay in place per the predefined transition rules of the machine.

Example In the below example (Figure 3), the DTM showcases the methodology above, with the start state q0 and end state qf.


Example

- Find a DTM that computes the language
  $L = \{a^n b^n : n \geq 0\}$.

Answer:

Figure 3: DTM Example

Complexity The Turing Machine's deterministic nature ensures that it follows a fixed computation path for a given input, thus resulting in a complexity proportional to the number of states traversed. While DTMs are powerful and capable of solving a wide range of problems, they may need help with complex tasks that require multiple branching possibilities, leading to a high time complexity.

2.2.2 Non-Deterministic Turing Machines

Definition A non-deterministic Turing Machine (NDTM) extends the concept of the DTM by allowing multiple possible transitions from a given state and symbol combination. In contrast to the DTM, an NDTM can explore several computation paths simultaneously, operating with a set of possible next states for each state-symbol pair. This non-deterministic behavior gives the NDTM greater flexibility and potentially more efficient problem-solving abilities for certain tasks.

How it Works The mechanism starts at the head of the machine, reading the symbol present on the tape under it. Instead of having a unique transition rule, the machine's state control unit can non-deterministically choose one of the possible next states and the symbol to write based on the current state and the symbol read. The head then proceeds to write this symbol on the tape. Furthermore, the head can move left or right or stay in place following the chosen transition. Hence allowing multiple computation paths to be explored simultaneously.

Example In the below example (Figure 4), the NDTM showcases the methodology above, where multiple routes are possible after a certain state. Hence, they can be multiple routes that end at the final state or that stay at the start state for the full cycle.

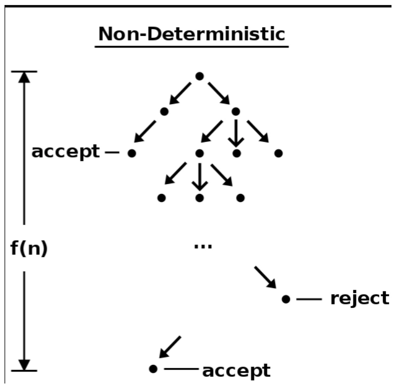


Figure 4: NDTM Tree Diagram

Complexity The non-deterministic behavior of the Turing Machine allows for parallel exploration of multiple computation paths, potentially reducing the time complexity for certain problems. However, it is important to note that NDTMs sometimes solve problems faster than DTMs. The time complexity of an NDTM is usually equal to or greater than the corresponding DTM. Nevertheless, NDTMs are useful in theoretical studies and can provide insights into the possibilities of computation.

2.3 Universal Touring Machine

Definition The Universal Turing Machine (UTM) is a theoretical construct representing a Turing Machine capable of simulating any other. It serves as a general-purpose computing device and plays a fundamental role in the theory of computation.

How it Works The UTM begins with reading, followed by processing, writing, and moving steps. First, the UTM's head reads the symbol under it from the input tape and the current state of the simulated Turing Machine encoded on the tape. Next, the UTM's state control unit interprets the current state and symbol read, consulting a transition function or table encompassing all possible transitions of all Turing Machines. Thus, the UTM determines the next state and the symbol to write on the input tape. The UTM then proceeds to write the symbol, representing the updated state of the simulated Turing Machine. Finally, the main head can move left or right on the input tape or remain in place, following the directions provided by the transition table.

Example Below is a classic example of a UTM, which in this case, possesses 3 different tapes(Encoded description of Mx, Internal state of Mx, and Tape contents of Mx).

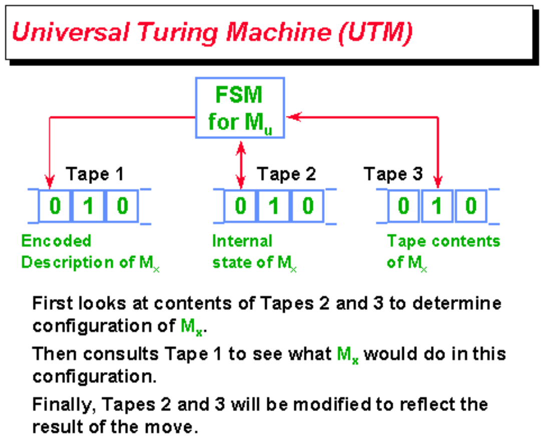

Universal Turing Machine (UTM)
FSM for M_u
Tape 1
Tape 2
Tape 3
0 1 0
0 1 0
0 1 0
Encoded Description of M_x
Internal state of M_x
Tape contents of M_x
First looks at contents of Tapes 2 and 3 to determine configuration of M_x.
Then consults Tape 1 to see what M_x would do in this configuration.
Finally, Tapes 2 and 3 will be modified to reflect the result of the move.

Figure 5: Universal Turing Machine

3 Importance of the Church-Turing Thesis

Definition The Church-Turing Thesis is a hypothesis suggesting that a Turing Machine can compute any effectively computable function. The term "effectively computable" refers to a function for which an algorithm exists that can compute its values within a finite number of steps. The thesis proposes that this class of effectively computable functions is equivalent to the class of functions that a Turing Machine can compute.

Theory The Church-Turing Thesis contains four main principles: Read, Process, Write and Move. Beginning with Read, the Church-Turing Thesis assumes that any input can be represented as a sequence of symbols on a tape, just like a Turing Machine's input tape. Afterward, any effective computation can be broken down into simple steps. In Processing, these steps are analogous to the transitions between states in a Turing Machine. The process involves performing basic operations on the input symbols, one at a time, following a set of well-defined rules. In Write, this corresponds to the computation's changing state, updating the input symbols' values per the algorithm's instructions. Finally, Move shows the movement of the head in a Turing Machine corresponds to the sequence of steps taken in an algorithm during the computation. It represents the computation flow from one state to another based on the rules and input values.

Example Below is an illustration of how a typical Chruch-Turing Machine would operate, containing components such as the tape mover, which would perform the four principles mentioned in the Theory.

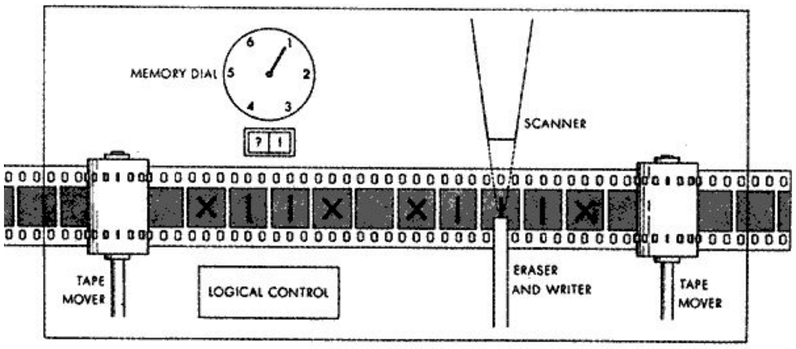


Figure 6: Church - Turing Machine

4 Explanation of Theoretical Concepts

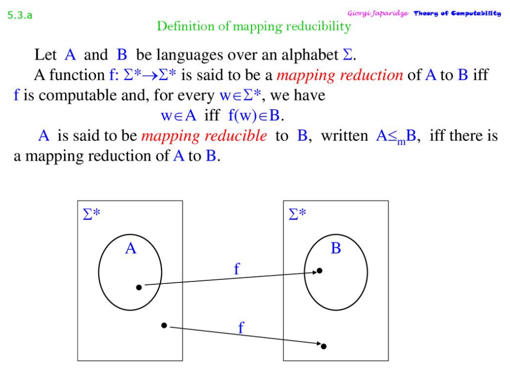
4.1 Reducibility

Definiton Reducibility is a fundamental theoretical, computational concept with significant implications for modern computation. It refers to the ability to transform one problem into another so that solving the second problem helps solve the first. In computational complexity theory, reducibility is crucial in classifying problems based on their difficulty.

Importance The importance of reducibility lies in its role in establishing relationships between computational problems, enabling the comparison of their complexity and determination of relative difficulty. For example, by reducing a given problem to a similar problem and recognizing the difficulty of that problem, it follows that the initial problem must be difficult. Hence, researchers can identify challenging problems and construct a hierarchy of computational complexity classes, including P, NP, and NP-hard.

It also applies to the modern era, where reducibility is a valuable tool for comprehending the tractability of diverse tasks and designing efficient algorithms. Reducing problems to well-studied ones offers a shortcut to solving new problems using existing solutions. An example of using this theory is in creating Artificial intelligence, which is extremely optimized today.

Illustration The below illustration showcases a Venn Diagram that explains what reducibility is alphabetically. The Venn Diagram showcases 2 sets, A and B. The arrow f shows that there is a linkage both inside the sets as well as the empty set outside.



5.3.a Giorgi Japaridze Theory of Computability

Definition of mapping reducibility

Let A and B be languages over an alphabet Σ.

A function $f: \Sigma^* \rightarrow \Sigma^*$ is said to be a *mapping reduction* of A to B iff f is computable and, for every $w \in \Sigma^*$, we have

$$w \in A \text{ iff } f(w) \in B.$$

A is said to be *mapping reducible* to B, written $A \leq_m B$, iff there is a mapping reduction of A to B.

Figure 7: Venn Diagram and Explanation for Reducibility

4.2 P = NP

Definiton P and NP are denoted as complexity classes in Automata Theory. The variable P represents the class of problems that can be solved in polynomial time. In contrast, NP represents the class of problems for which a solution can be verified in polynomial time.

Importance If P = NP, every problem for which a solution can be verified in polynomial time would be solved in a given time. This would imply that many computationally exhausting real-world problems would have efficient solutions.

Example The most common example is the Traveling Salesman Problem (TSP). It searches for the shortest possible route to visit a given city exactly once while returning to the original starting city.

Solving a TSP optimally for large problem instances is computationally infeasible with current algorithms. If P = NP, an efficient algorithm could be implemented to solve the

TSP and similar problems. This could impact several computing fields, such as network planning and route optimization.

Illustration The below illustration typically shows the format that P = NP problems would showcase in terms of their complexity.

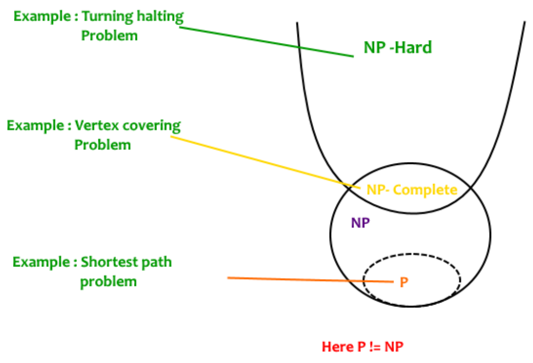

Example : Turning halting Problem
NP -Hard
Example : Vertex covering Problem
NP- Complete
NP
Example : Shortest path problem
P
Here P != NP

Figure 8: Venn Diagram for P=NP

4.3 Decidability

Definition Decidability is a fundamental concept in theoretical computer science related to determining whether an algorithm can solve or recognize a given problem or language. It has significant significance in modern computation, as seen below.

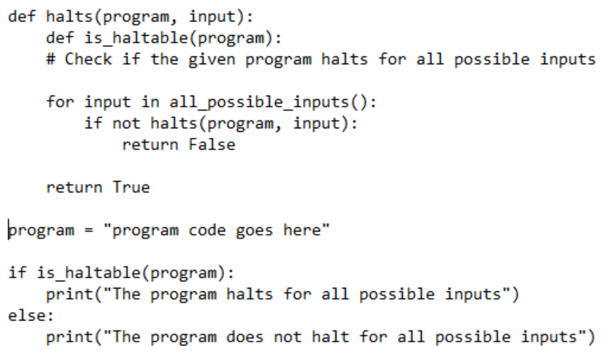
Importance Decidability plays a vital role in modern computation as it helps characterize the solvability of problems, determine the limits of computation, and classify computational complexity classes. It allows us to identify problems that can be efficiently solved and recognize those that are beyond the reach of algorithms. This knowledge guides the design of algorithms and aids in problem-solving strategies. Also, it contributes to advancements in various areas, such as formal languages and theorem proving.

Example The first example showcases how decidability is implemented using code snippets. The following code is done in Python, which is meant to prove whether a given program halts or runs forever for all given inputs.

Illustration The following Venn Diagram showcases the relationship between decidability, semidecidability and undecidability.

4.4 Dynamic Programming

Definition Dynamic Programming is a fundamental computational concept that has revolutionized algorithm design and optimization. It is a useful technique to efficiently solve complex problems by breaking them down by subdividing them, overlapping subproblems,



```
def halts(program, input):
    def is_haltable(program):
    # Check if the given program halts for all possible inputs

    for input in all_possible_inputs():
        if not halts(program, input):
            return False

    return True

program = "program code goes here"

if is_haltable(program):
    print("The program halts for all possible inputs")
else:
    print("The program does not halt for all possible inputs")
```

Figure 9: Code Snippet for Halting Problem

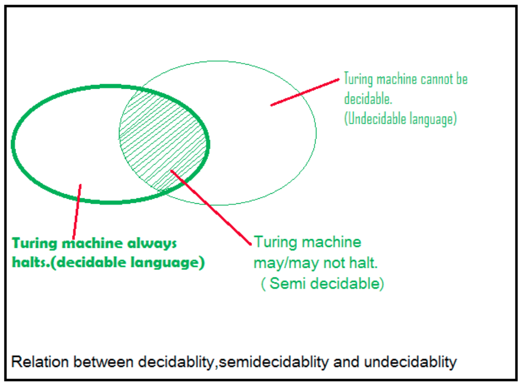
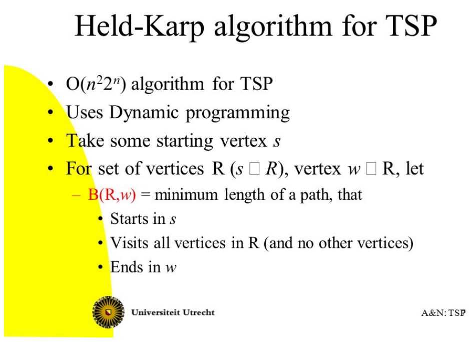

Turing machine cannot be decidable. (Undecidable language)
Turing machine always halts.(decidable language)
Turing machine may/may not halt. (Semi decidable)
Relation between decidablity,semidecidablity and undecidablity

Figure 10: Venn Diagram for Decidability

and reusing computed results. It provides a systematic and efficient approach to problem-solving, enabling significant improvements in time and space complexity.

Importance Dynamic Programming is a key concept in modern computation, enabling efficient solutions to complex problems by breaking them down into smaller, more manageable subproblems. It eliminates redundant computations by storing and reusing intermediate results, saving time and space. It is able to utilize applications in various domains, including optimization and algorithms.

Example Below shows an example of Dynamic Programming named as the Held-Karp algorithm. This is explained in the following diagram.



Held-Karp algorithm for TSP

- O($n^2 2^n$) algorithm for TSP
- Uses Dynamic programming
- Take some starting vertex s
- For set of vertices R (s □ R), vertex w □ R, let
  - B(R,w) = minimum length of a path, that
    - Starts in s
    - Visits all vertices in R (and no other vertices)
    - Ends in w

Universiteit Utrecht A&N: TSP

Figure 11: Explanation of Dynamic Programming

5 References

1. **Held-Karp Algorithim** https://medium.com/@data-overload/unveiling-the-held-karp/

2. **Reducability** https://www.geeksforgeeks.org/undecidability-and-reducibility-in-toc/

3. **Decidability** https://www.geeksforgeeks.org/undecidability-and-reducibility-in-toc/

4. **Multi-Tape** https://www.geeksforgeeks.org/variation-of-turing-machine/

5. **Single Tape** https://www.sciencedirect.com/science/article/pii/S001999588480042X/

6. **Accepted States** https://www.tutorialspoint.com/automata$_t heory$/