Non-deterministic Turing Machine Case Study Term 3 AY 2022 - 2023

Ralph Gabriel Bautista
ralph_gabriel_bautista@dlsu.edu.ph

Allen Andrei Gutierrez
allen_gutierrez@dlsu.edu.ph

Abstract

This paper is about the study of our chosen model which is the Nondeterministic Turing Machine and its capabilities compared to the other models discussed in class. A Nondeterministic Turing Machine (NTM) consists of a tape divided into cells containing symbols from a finite alphabet, a read-/write head moving left or right along the tape, a finite set of states, a transition function determining possible state changes based on the current state and tape symbol, acceptance criteria defining when the machine halts and accepts input, and a computational model exploring multiple possible paths concurrently to recognize recursively enumerable languages. Its operations include reading and potentially writing symbols on an infinite tape, moving the read/write head left or right, transitioning between states based on current symbols and the machine's state, branching at non-deterministic points, and halting in an accepting or rejecting state, or continuing indefinitely. According to the Chomsky Hierarchy the NTM belongs to the highest category which is Type 0 as it is a variant of the Turing machine. Lastly, the machines are used in algorithm design, where they can explore multiple potential solutions simultaneously to optimize complex computational problems, and in cryptography, where they are used in cryptographic protocols such as RSA encryption, relying on the presumed difficulty of solving certain non deterministic problems for security.

Machine Definition

A Non-deterministic Turing Machine (NDTM) is a theoretical model of computation that extends the capabilities of a deterministic Turing Machine (DTM) by allowing multiple possible transitions from a given state on a given input symbol. Consequently, the transitions are not deterministic in this case. A NDTM's computation is a tree of configurations achieved from the initial configuration. Input is accepted only if there exists at least one node of the tree, which is an accept state. The NDTM is called a Decider if all branches of the computational tree terminate on all inputs. If all branches are rejected for some input, the input itself is rejected as well.

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Formal Defintion:

The non-deterministic Turing machine is a 6 tuple machine: (**Q, Σ,** q_0**, Γ, δ, F**)

Wherein…

Q - set of states

Σ - set of input alphabets

q_0 - initial state

Γ - set of tape alphabets

δ - transition function that is $Q \ * \ \Gamma^i \rightarrow P(Q \ * \Gamma^i \ * \ (Left, \ Right, \ Stationary)^i)$

F - set of final states

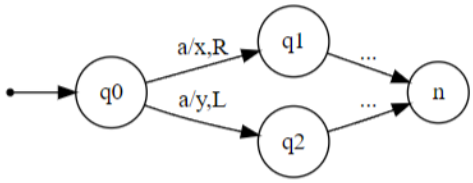


Figure 1. Formal Definition

Furthermore, the contents of the tape as well as the present state make up the complete state of an NDTM. The total state includes all possible combinations of states the machine can be in at any given moment in its calculation because the NDTM can be in numerous states at once. The configuration of the tape, including the symbols printed on it and the tape head's location, is also included in the entire state. Expanding on the description provided earlier, the input to a NDTM is typically a finite sequence of symbols from an input alphabet. Initially, this information is fed into the input tape of the machine, with the tape head positioned at the far left symbol. As it receives symbols from the input tape and moves between states in accordance with its transition function, the NDTM's computation progresses.

Regarding output, NDTM can result in various outcomes. The output provides information about the decision or result obtained from processing the input string, reflecting the machine's computational behavior and the specific problem it aims to solve. Such possible outputs include:

Acceptance: The input string is accepted when the machine stops in an accepting state, meaning it meets the requirements (e.g., reaching an accepting state, satisfying a given condition).

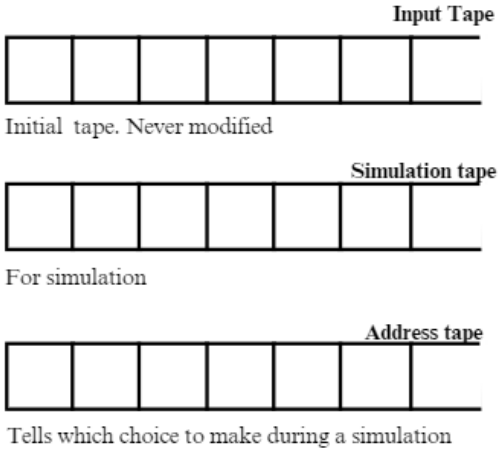
Rejection: When an input string does not meet the predetermined requirements, the machine stops and does not accept the input string.

Final Configuration: When there are no more configuration changes possible, the machine stops. The contents of the tape at the conclusion of the computation, which could represent a conclusion, a choice, or other pertinent data, could be the output in this scenario.

Infinite Loop: In some cases, the NDTM may occasionally go into an endless loop without stopping. If there isn't a legitimate transition for the input symbol combination and current state, this might happen in certain situations, the computation is deemed failed and the outcome is undefined. Such situations entail that NDTM is most likely the wrong type of machine to use for the problem.

Given such functionality, the operation of a NDTM is characterized by its ability to explore multiple computation paths simultaneously. An NDTM, in contrast to deterministic machines, is capable of switching between several states from a single state and input symbol. Its non-deterministic feature enables it to investigate multiple options concurrently, which makes it effective in solving some kinds of issues. An NDTM's functions include reading symbols from the input tape, modifying the tape's contents, and switching between states in accordance with its transition function. In order to showcase the abilities of this type of machine a sample machine is listed below:

Key terminologies to note for the sample problem:

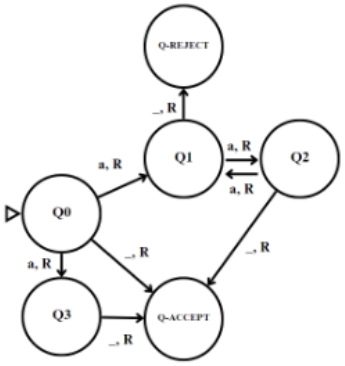


Algorithm to follow:

Initially: Tape 1 is the input tape Tape 2 is the simulation tape which initially is empty Tape 3 is the address tape

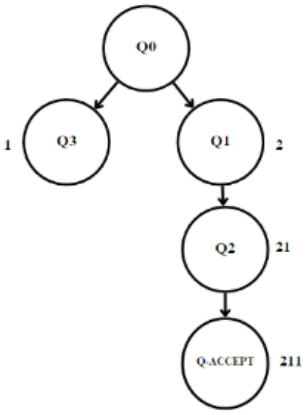
1. Copy tape 1 to tape 2.
2. Run the simulation.
3. Use tape 2 as the main tape.
4. When choices occur, consult tape 3.
5. Tape 3 contains a path. Each number tells which choice to make.
6. Run the simulation all the way down the branch as far as the address path/path goes.
7. Try the next branch.
8. Increment the address on tape 3.
9. Repeat.

Sample machine:



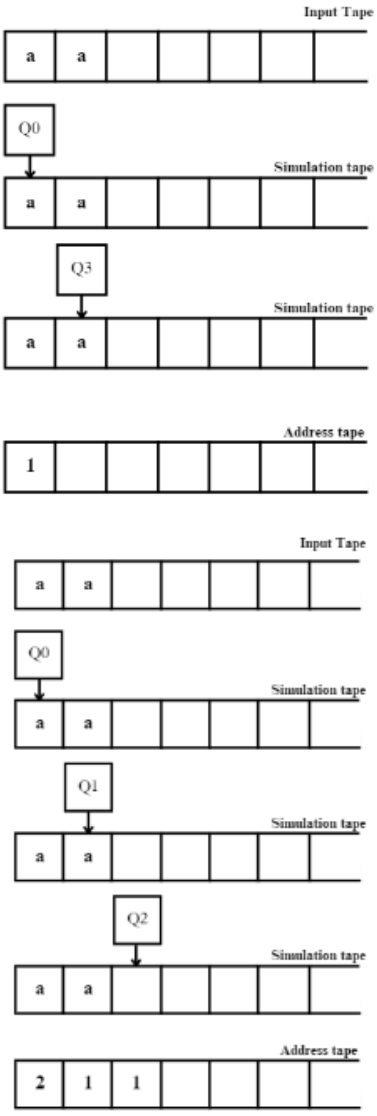
Given this non deterministic Turing machine solve for the outcomes of the input aa; note that the _ stands for blank. . .

Computation tree for input aa:



Case 1: In this case you reset the loop as there are no more symbols on the address tape and neither a reject or accept state is reached

Case 2: In this case you will reach a state in which the input is being accepted, hence, the input aa is accepted. This case highlights the multiple paths functionality of NDTM. It is important to note that based on the computation tree the paths 21, 111, 112, 121, and 122.



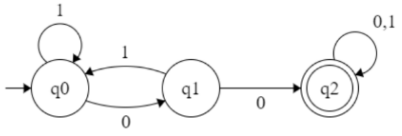
Formal Language and Computational Power

The Chomsky Hierarchy categorizes grammar that is accepted by different machines into four types, Type 0, Type 1, Type 2, Type 3; Type 0 being the highest and Type 4 being the lowest. Type 3 is known as Regular Grammar, it is the simplest and most restricted out of the four types. It is accepted by finite-state machines and characterized by their regularity and simplicity. Type 2 is known as Context-free grammar, it is more expressive compared to Regular Languages but less powerful compared to unrestricted grammar. It is recognized by Pushdown Automata and is characterized by their context-free nature. Type 1 is known as context-sensitive grammar, and it is more expressive than context-free languages but less expressive than unrestricted grammars. It is recognized by the Linear Bound Automata and is characterized by their context-sensitive nature. Type 0 is known as unrestricted language and is the most powerful of the four as it encapsulates all formal languages it is also known as the Recursively Enumerable languages. It is recognized by Turing machines and is characterized by their ability to describe highly complex languages without any restrictions. The computational power of the NDTM with respect to the hierarchy is classified as Type 0, as it is a variant form of the Turing Machine. This can be proven by the following below that the NDTM can simulate the other formal models computation discussed in class such as FSM, PDA and TM. Note that since the group is already discussing a Turing machine and mainly DTM were discussed the group opted to solve a DTM using a NDTM, even though the process is the same it shows how NDTM is in the same type (Type 0) as DTM.

Finite-State Machine: Given a DFA, construct an TM

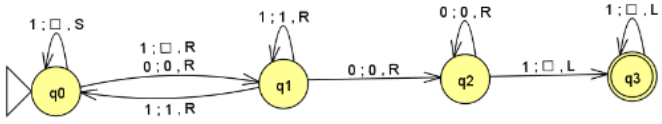
- The TM has the same input alphabet and states as DFA.
- For each transition of the DFA from state A to state B on a given input symbol, the TM has a corresponding transition from state A to state B on that input.
- If there is no transition defined in the DFA for an input symbol in a certain state, the TM can have transitions to multiple states, effectively simulating non-determinism.
- The TM accepts if any of its computations reach an accepting state.

Sample problem: Given this DFA that recognizes strings over the alphabet 0, 1and accepts strings that end with '1'. Construct a NDTM and verify if string ending with '1' is accepted by both machines



Solution for DFA: For each transition in the DFA, create corresponding transitions in the NDTM. Since NDTM is non-deterministic, there can be multiple transitions from a state given the same input.

Convert to TM:



TM transitions:
From state q0:

- Read 0: Move right, transition to q1
- Read 1: Move right, stay in q0 or transition to q1

From state q1:

- Read 0: Move right, transition to q2

- Read 1: Move right, stay in q0 or transition to q1

From state q2:

- Read 0: Move right, stay in q2

- Read 1: Move left, transition to q3

The acceptance logic ensures that the NDTM halts and accepts the input string when it reaches an accepting state (q3) and positions the tape head on the leftmost '1' symbol.

Conclusion: Since the NDTM can simulate the behavior of a Deterministic Finite Automata, it recognizes languages by Finite-State Machines.

Pushdown Automata: Given a PDA, construct an TM

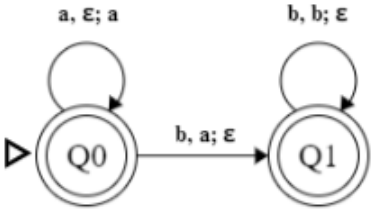
- The TM has the same input alphabet, stack alphabet, states, and transitions as the PDA.

- For each transition of the PDA, the NDTM has a corresponding transition.

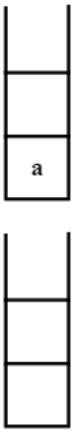
- The TM can simulate non-deterministic choices by branching into multiple computations whenever there are multiple possible transitions from a state with the same input symbol and stack symbol.

- The TM accepts if any of its computations reach an accepting state with an empty stack.

Sample problem: Given this PDA that accepts $a^n b^n$ construct a NDTM. Verify by checking if the input string "ab" is accepted by both machines.

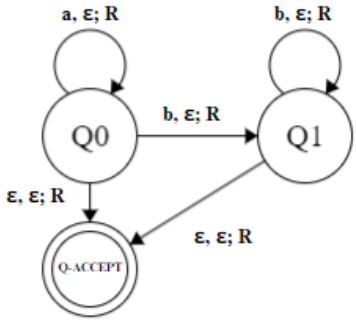


Solution for PDA:

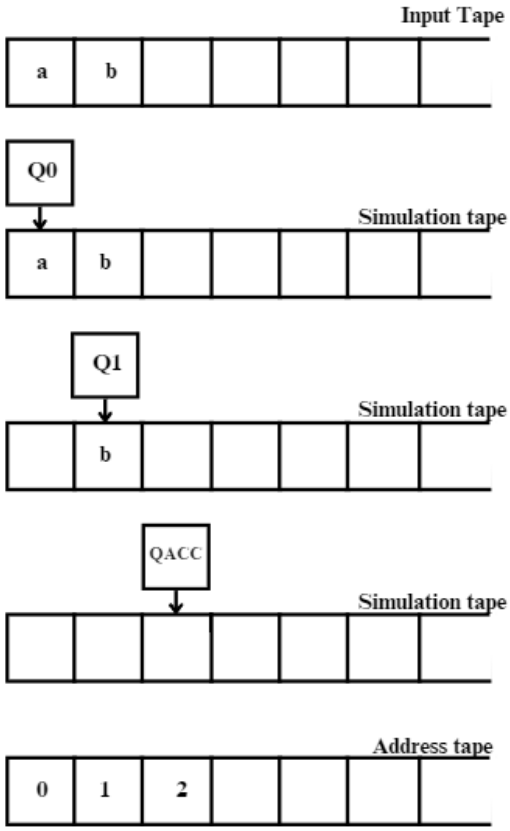


Logic: Once the machine reads a it pops nothing and pushes a, once the a's are finished and it reads b for the first time it will pop a and push nothing, lastly in order to eliminate the remaining b's it will pop b once it reads b and pushes nothing

Converted to TM:



Solution for TM:



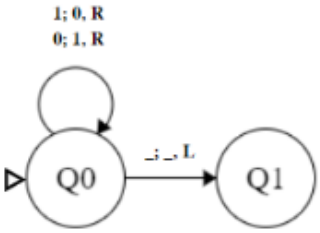
Logic: Use intuition. Think of the pop/stack system of the PDA as a tape and follow the behaviour of the machine. In this simple problem the group only added one more state in order to accept the string input, however, a lot of the original PDA machine state was retained except for the labels as it was changed to work with tapes

Conclusion: Since the NDTM can simulate the behavior of a Pushdown Automata, it can recognize languages recognized by Pushdown Automata.

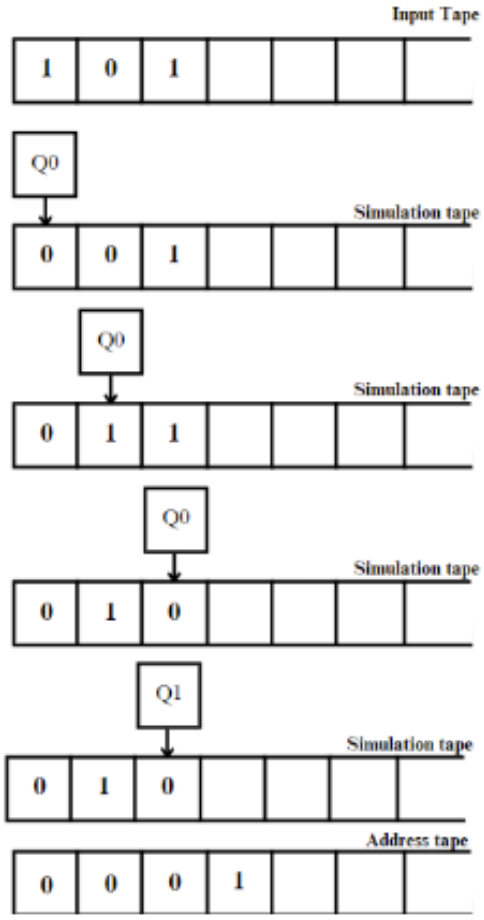
Turing Machine: Given a DTM, construct an NDTM

- The NDTM has the same input alphabet, tape alphabet, states, and transitions as the DTM.
- Unlike DTM, the NDTM can simulate non-deterministic choices by branching into multiple computations whenever there are multiple possible transitions from a state with the same input symbol and tape symbol.
- The NDTM accepts if any of its computations reach an accepting state.

Sample Problem: Given this deterministic turing machine to compute the 1's complement of a number (example taken from one of the slides in class) construct a non-deterministic turing machine



Solution: Given that the only difference between deterministic and non-deterministic machines is the ability of the latter to have two or more transitions the machine and how it works essentially stays the same as show below:



Conclusion: Since the NDTM can simulate the behavior of a DTM, they are the same type. It is important to note that every NDTM has an equivalent DTM, hence, the solution shown above.

Given these proofs we can say that the NDTM is classified as Type 0 in the Chomsky Hierarchy as it has all the specifications of a Type 0 which recognizes all formal language.

Applications

It is important to note that many sources state that NDTM's indeed have theoretical significance in computer science, particularly in understanding the theoretical limits of computation and complexity theory. However prior to choosing the topic the group didn't expect that the machines practical coding applications are little to none based on several reasons:

Theoretical Construct: Nondeterministic Turing machines are primarily a theoretical construct used in theoretical computer science to analyze and understand the computational complexity of problems. They are not directly translatable into practical coding applications in the same way deterministic Turing machines are.

Non-Physical Implementation: While deterministic Turing machines have direct analogs in real-world computing devices (such as conventional computers), nondeterministic Turing machines are more abstract and do not have direct physical implementations. This makes it challenging to directly apply them in practical coding scenarios.

Complexity and Efficiency: NDTMs can explore multiple computation paths simultaneously, which makes them potentially more powerful than deterministic Turing machines for certain tasks. However, this also introduces complexity and inefficiency in practical implementations. Coding for NDTMs would require handling non-determinism, which can significantly complicate algorithms and make them harder to reason about and debug.

Despite this knowledge, the group opted to utilize a TM applied in the field of cryptography. More specifically text encryption by utilizing the concept of a Caesar Cipher. The Caesar Cipher works by shifting the letters in the plaintext message by a certain number of positions, known as the "shift" or "key". The code for this application is shown below:

Code:

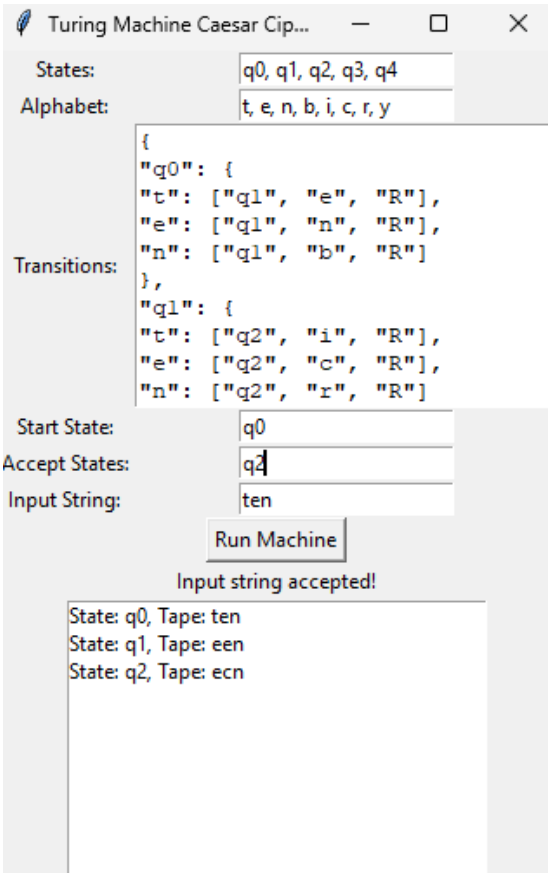
```
import tkinter as tk

class TuringMachine:
    def __init__(self, states, alphabet, transitions, start_state, accept_states):
        self.states = states
        self.alphabet = alphabet
        self.transitions = transitions
        self.start_state = start_state
        self.accept_states = accept_states
        self.current_state = start_state
        self.current_position = 0
        self.tape = []

    def initialize_tape(self, input_string):
        self.tape = list(input_string)

    def step(self):
        if self.current_state in self.transitions and self.tape[self.current_position] in self.transitions[self.current_state]:
            transition = self.transitions[self.current_state][self.tape[self.current_position]]
            self.tape[self.current_position] = transition[1]
            if transition[2] == 'R':
                self.current_position += 1
            elif transition[2] == 'L':
                self.current_position -= 1
            self.current_state = transition[0]

    def run(self):
        steps = []
        while self.current_state not in self.accept_states:
            steps.append((self.current_state, ''.join(self.tape)))
            self.step()
            if self.current_position < 0 or self.current_position >= len(self.tape):
                return steps, False
        steps.append((self.current_state, ''.join(self.tape)))
        return steps, self.current_state in self.accept_states

class TuringMachineGUI:
    def __init__(self, master):
        self.master = master
        self.master.title("Turing Machine Caesar Cipher")
        self.machine = None
        self.input_string = ""

        self.states_label = tk.Label(master, text="States:")
        self.states_label.grid(row=0, column=0)
        self.states_entry = tk.Entry(master)
        self.states_entry.grid(row=0, column=1)

        self.alphabet_label = tk.Label(master, text="Alphabet:")
        self.alphabet_label.grid(row=1, column=0)
        self.alphabet_entry = tk.Entry(master)
        self.alphabet_entry.grid(row=1, column=1)

        self.transitions_label = tk.Label(master, text="Transitions:")
        self.transitions_label.grid(row=2, column=0)
        self.transitions_entry = tk.Text(master, height=10, width=30)
        self.transitions_entry.grid(row=2, column=1)

        self.start_state_label = tk.Label(master, text="Start State:")
        self.start_state_label.grid(row=3, column=0)
        self.start_state_entry = tk.Entry(master)
        self.start_state_entry.grid(row=3, column=1)

        self.accept_states_label = tk.Label(master, text="Accept States:")
        self.accept_states_label.grid(row=4, column=0)
        self.accept_states_entry = tk.Entry(master)
        self.accept_states_entry.grid(row=4, column=1)

        self.input_label = tk.Label(master, text="Input String:")
        self.input_label.grid(row=5, column=0)
        self.input_entry = tk.Entry(master)
        self.input_entry.grid(row=5, column=1)

        self.run_button = tk.Button(master, text="Run Machine", command=self.run_machine)
        self.run_button.grid(row=6, columnspan=2)

        self.result_label = tk.Label(master, text="")
        self.result_label.grid(row=7, columnspan=2)

        self.state_listbox = tk.Listbox(master, height=10, width=40)
```

```
        self.state_listbox.grid(row=8, column=0, columnspan
            =2)

    def get_machine_definition(self):
        states = self.states_entry.get().split(",")
        alphabet = list(self.alphabet_entry.get())
        transitions = eval(self.transitions_entry.get
            ("1.0", tk.END))
        start_state = self.start_state_entry.get()
        accept_states = self.accept_states_entry.get().
            split(",")
        return states, alphabet, transitions, start_state,
            accept_states

    def run_machine(self):
        self.state_listbox.delete(0, tk.END)
        states, alphabet, transitions, start_state,
            accept_states = self.get_machine_definition()
        self.machine = TuringMachine(states, alphabet,
            transitions, start_state, accept_states)
        self.input_string = self.input_entry.get()
        self.machine.initialize_tape(self.input_string)

        steps, accepted = self.machine.run()
        for step in steps:
            self.state_listbox.insert(tk.END, f"State: {
                step[0]}, Tape: {step[1]}")

        if accepted:
            self.result_label.config(text="Input string
                accepted!")
        else:
            self.result_label.config(text="Input string not
                 accepted since the final state is not an
                accepted state.")

if __name__ == "__main__":
    root = tk.Tk()
    app = TuringMachineGUI(root)
    root.mainloop()
```

Table 1. Transition Table for Turing Machine

State	Symbol Read	Next State	Symbol Write	Move
q0	t	q1	e	R
q0	e	q1	n	R
q0	n	q1	b	R
q1	t	q2	i	R
q1	e	q2	c	R
q1	n	q2	r	R
q2	t	q3	y	R
q2	e	q3	t	R
q2	n	q3	e	R
q3	b	q4	i	R
q3	i	q4	n	R
q3	c	q4	t	R
q3	r	q4	e	R
q3	y	q4	n	R
q3	t	q4	b	R
q3	e	q4	i	R
q3	n	q4	c	R

Sample for the input "ten":



References

Dhakad, A. (2024, May 27). Non-Deterministic Turing Machine. Code 360 by coding ninjas. https://www.naukri.com/code360/library/non-deterministic-turing-machine

Diligent, J. (2020, December 4). How to simulate non-deterministic Turing machines on a deterministic Turing machine. YouTube. https://www.youtube.com/watch?v=u0t7zJi8Fno

Easy Theory. (2021, November 10). Deterministic Finite Automaton to Turing Machine Conversion (DFA to TM). https://www.youtube.com/watch?v=df4TK-eDAw4

GeeksforGeeks. (2023, October 31). Chomsky Hierarchy in Theory of Computation. https://www.geeksforgeeks.org/chomsky-hierarchy-in-theory-of-computation/

GeeksforGeeks. (2023, May 11). Caesar cipher in cryptography. https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

Ilyas, Z. (n.d.). Non-deterministic Turing machine (NTM). Educative. https://www.educative.io/answers/non-deterministic-turing-machine-ntm

Neso Academy. (2018, January 23). Nondeterministic Turing Machine (part 2). YouTube.

https://www.youtube.com/watch?v=9Bk11XgiC1E

Appendix

Group Members:

- P1: Ralph Gabriel Bautista
- P2: Allen Andrei Gutierrez

Activity	**P1**	**P2**
Topic Formulation	50.0	50.0
Machine Definition	75.0	25.0
Formal Language and Computational Power	40.0	60.0
Applications	50.0	50.0
Raw Total	215.0	185.0
TOTAL	**58.2**	**41.8**

Allen Andrei D. Gutierrez April 8, 2024

Ralph Gabriel R. Bautista April. 8, 2024