CISC 603 Theory of Computation

HW2: Modeling Computation

Rishabh Agarwal

Describe in detail about all the machine models we have discussed, including FA, DFA, NFA, PDA, DPDA, NPDA, TM, DTM, NTM, and UTM. What are they? What are they capable of doing in terms of computation and language /recognition/acceptance? What are their limitations of doing these?

Notes:

- Please *use your own words* to describe your own understanding of computation models as it is not about memorizing but absorbing.
- Preparing such notes is a valuable learning experience and one that allows you to internalize the material at a new level. Take pride in your work.

FA (Finite Automata):

Finite Automata is a simplest form of computer which can perform basic operation. For example, it can take an input and result whether the input is accepted or rejected. It doesn't have its own memory.

DFA (Deterministic Finite Automata):

Deterministic Finite Automate is a Finite Automata follow deterministic constraint.

- No Contradiction
- No Omissions

A DFA can be represented by a 5-tuple $(Q, \sum, \delta, q_0, F)$ where −

- **Q** is a finite set of states.
- $\sum$ is a finite set of symbols called the alphabet.
- **δ** is the transition function where $\delta: Q \times \sum \rightarrow Q$
- $\mathbf{q_0}$ is the initial state from where any input is processed ($q_0 \in Q$).

- **F** is a set of final state/states of Q ($F \subseteq Q$).

NDFA (Non-Deterministic Finite Automata):

Deterministic Finite Automate is a Finite Automata with relief deterministic constraints.
NDFA is more capable than DFA. It allows machine to have contradicting rule to allow more execution path. For example, Node A will go to Node B when rule is 1 and Node A will recursively call Node A when rule is 0.

Pushdown Automata (PDA):

PDA are FA with internal memory. The memory is in a form of stack and can always refer to the top element in the stack. PDA are more powerful than DFA and NDFA because it can store more than just the current state of the machine. This allow it to perform more complex operation.
A PDA can be formally described as a 7-tuple ($Q, \sum, S, \delta, q_0, I, F$) –

- **Q** is the finite number of states
- $\sum$ is input alphabet
- **S** is stack symbols
- **δ** is the transition function: $Q \times (\sum \cup \{\varepsilon\}) \times S \times Q \times S^*$
- $\mathbf{q_0}$ is the initial state ($q_0 \in Q$)
- **I** is the initial stack top symbol ($I \in S$)
- **F** is a set of accepting states ($F \in Q$)

Deterministic PDA (DPDA):

DPDA are the PDA which will respect deterministic constraint. These are the most used PDA. Most of the modern-day programing language use deterministic PDA. DPDA can have single move

Non-Deterministic PDA (NPDA):

Unlike DPDA only NPDA define all context free languages. NPDA are PDA with relieved deterministic constraints. Unlike other Non-determinism doesn't bring any more capabilities to the existing PDA.

Turing Machine (TM) or Deterministic Turning Machine (DTM):

A finite state machine with access to an infinitely long tape is called a Turing Machine (TM). A Turing Machine (TM) is a mathematical model which consists of an infinite length tape divided into cells on which input is given. It consists of a head which reads the input tape. A state register stores the state of the Turing machine. After reading an input symbol, it is replaced with another symbol, its internal state is changed, and it moves from one cell to the right or left. If the TM reaches the final state, the input string is accepted, otherwise rejected.

A TM can be formally described as a 7-tuple (Q, X, ∑, δ, q_0, B, F) where −

- **Q** is a finite set of states
- **X** is the tape alphabet
- ∑ is the input alphabet

- **δ** is a transition function; $\delta : Q \times X \rightarrow Q \times X \times \{Left\_shift, Right\_shift\}$.
- $\mathbf{q_0}$ is the initial state
- **B** is the blank symbol
- **F** is the set of final states

Non-Deterministic Turning Machine (NDTM):

Non-determinism makes no difference and bring no extra capability to Deterministic Turning Machine.

Universal Turning Machine (UTM):

A UTM is a machine that can simulate any other DTM by reading its rules, accept states, and initial configuration from the tape and stepping through its execution, essentially acting as a Turing machine rulebook interpreter.