CSL253 - Theory of Computation

Tutorial 7

Team Members

1. Divyansh Dubey - 12340690
2. Naman Sharma - 12341460
3. Lakshay Gupta - 12341300

Question 10

Let $\Sigma = \{0, 1\}$. Show that the problem of determining whether a context-free grammar (CFG) generates some string in $a^n b^n$ is decidable.

Solution

We describe how a Turing Machine (TM) with three tapes can simulate a Pushdown Automaton (PDA) recognizing the language $L = \{a^n b^n \mid n \geq 0\}$. The tapes are organized as follows:

- **Tape 1 (Input Tape):** Contains the input string (a sequence of `a`'s followed by `b`'s).
- **Tape 2 (Control & Transition Function Tape):**
  - **Transition Function Encoding Region:** Holds a complete encoding of the PDA's (or simulated PDA's) transition function. This encoding is a table (or list) of tuples that specify how the machine should react given a combination of the current state, the input symbol from Tape 1, and the top-of-stack symbol from Tape 3.
  - **State Marker Region:** Maintains the current state (for example, initially `q_0`, then updated as the machine processes the input).
- **Tape 3 (Stack Tape):** Simulates the PDA's stack. It is used to store symbols (in this case, essentially the symbol `a` pushed for every input `a`) and later "pop" them when reading `b`'s.

The overall operation now follows these detailed steps:

1. Initialization

- **Tape 1:** The input is written (e.g., `aaa...bbb`) and the head is positioned at the left-most symbol.

a	a	a	a	..	b	b	b	b	..

- **Tape 2:**
  - **Transition Function Encoding Region:** Before the simulation begins, a fixed encoding of the transition function is written onto a designated section of this tape. For example, each entry may be encoded as:
    * `(q_0, a, any)` $\rightarrow$ `(q_0, push a)` meaning "when in state `q_0` and reading an `a` (with any stack symbol), push an `a` onto Tape 3 and remain in state `q_0`."
    * `(q_0, b, a)` $\rightarrow$ `(q_1, pop)` meaning "when switching to a `b` in state `q_0` with an `a` on top of the stack, change to state `q_1` and pop from Tape 3."
    * `(q_1, b, a)` $\rightarrow$ `(q_1, pop)` for continuing the b-phase.
  - **State Marker Region:** Also on Tape 2, a separate area is used to record the current state (initialized to `q_0`).

q_0	q_1		q_0	a	any	q_0	$push$	a	$\#$	q_0	b	a	q_1	pop

- **Tape 3:** The stack is initialized, typically with a bottom-of-stack marker (for example, `$`) or simply empty to indicate that no symbols have yet been pushed.

$	_	_	_

2. Transition Function Lookup

Before each move, the machine will use the current state (stored in the state marker region on Tape 2), the current input symbol on Tape 1, and the top-of-stack symbol on Tape 3 to look up the corresponding transition in the encoded transition function (also on Tape 2). The lookup is performed as follows:

1. **Read the Current Situation:**
   - The current state is read from the state marker region of Tape 2.
   - The input symbol is read from Tape 1.
   - The top symbol of the stack is read from Tape 3.
2. **Scan the Encoded Transitions:**
   - The machine scans the encoding region on Tape 2 to find a tuple matching the triple (current state, input symbol, top-of-stack symbol).

- If a matching tuple is found, it provides the instruction for updating the state, performing a stack operation (push or pop), and moving the input head.

This lookup effectively simulates the finite control's decision-making process.

3. Processing the Input and Simulating the PDA

Phase 1 – Reading `a`'s (Push Operation):

- **For every `a` on Tape 1:**
  - The machine reads the current state (say, `q_0`) and sees that the input symbol is `a`.
  - It consults the transition function on Tape 2. An entry such as `(q_0, a, any)` → `(q_0, push a)` indicates that the machine should push an `a` onto Tape 3.
  - **Actions:**
    - **Tape 3:** Append (write) an `a` at the current blank cell to simulate a push.
    - **Tape 1:** Move the head to the right to read the next symbol.
    - **Tape 2:** Maintain or update the state marker (remains `q_0`) if the transition does not change the state.

Transition – Encountering the First `b`:

- When Tape 1's head first reads a `b`, the machine:
  - Looks up the corresponding transition using the current state (still `q_0`) and the stack's top symbol.
  - Finds a rule such as: `(q_0, b, a)` → `(q_1, pop)` indicating a state change to `q_1` and that a pop operation should be performed.
  - **Actions:**
    - **Tape 2:** Update the state marker from `q_0` to `q_1`.
    - **Tape 3:** Pop the top `a` from the stack (erase it or mark it blank and reposition the head on Tape 3).
    - **Tape 1:** Continue processing the current `b` (or move right, as specified by the transition).

Phase 2 – Reading `b`'s (Pop Operation):

- **For every `b` on Tape 1:**
  - The machine is now in state `q_1`.
  - It reads the current input symbol (a `b`) and checks the top symbol of Tape 3.
  - The encoded transition in Tape 2 (e.g., `(q_1, b, a)` → `(q_1, pop)`) tells the machine to pop an `a`.

- **Actions:**
  - **Tape 3:** Perform the pop operation.
  - **Tape 1:** Move the head right.
  - **Tape 2:** Remain in state `q_1` while continuing to check each symbol.

If at any point no matching transition is found (for instance, when a `b` is read but the top of the stack is not `a`, or if the stack is empty), the machine rejects the input.

4. Final Check and Acceptance

Once Tape 1 reaches the end-of-input (a blank symbol), the machine does a final check:

- **Tape 3 (Stack):**
  - If the stack is empty (or only the bottom marker remains), it means every `a` was matched by a `b`.
  - The machine enters an accepting state.
- Otherwise, the machine rejects the input due to leftover unmatched `a`'s (or an incorrect transition).

By using Tape 2 to encode the transition function, the Turing machine effectively "reads" what action to take at each step. This design mirrors how a real PDA would use its transition rules, ensuring that the Turing machine can correctly simulate the behavior of a PDA for recognizing the language `a^n b^n`.

Question 24

Let $E = \{\langle M\rangle \mid M$ is a DFA that accepts some string with more 1s than 0s$\}$. Show that E is decidable.

Solution

This can be done by using closure properties of CFLs.

Let the language

$$A = \{w \mid w \text{ has more 0s than 1s}\}$$

be the CFL.

We can construct a TM S for the given language in the following way:

We construct B such that $B = A \cap L(M)$ (note that B will be a CFL) We describe a Turing machine that simulates a PDA recognizing

$$B = \{w \in \{0,1\}^* \mid \#1(w) > \#0(w)\}.$$

The simulation uses three tapes, each with a specific role:

- **Tape 1 (Input Tape):** Contains the input string (a sequence of 0's and 1's).
- **Tape 2 (Control & Transition Function Tape):**
  - **Transition Function Encoding Region:** Stores an encoding of the PDA's transition function (a lookup table). For each triple (current state, input symbol, top-of-stack symbol), the table specifies the action to perform.
  - **State Marker Region:** Holds the current state of the PDA. In our simulation the state remains fixed (say, q_0), and the "memory" is maintained on the stack.
- **Tape 3 (Stack Tape):** Simulates the PDA's stack. It is initialized with a special bottom-of-stack marker (denoted by \$). The stack will hold the symbols X and Y as follows.

Stack Operation Rules

The PDA simulation works by processing the input symbol-by-symbol and manipulating the stack on Tape 3 according to these rules:

1. **When reading a** 1**:**
   - **If the top symbol is not** Y**:** Push an X onto the stack.
   - **If the top symbol is** Y**:** perform pop operation and pop Y.
2. **When reading a** 0**:**
   - **If the top symbol is not** X**:** Push a Y onto the stack.
   - **If the top symbol is** X**:** Pop the X (i.e., remove the X from the stack without pushing any symbol).

Overall Operation of the Simulation

1. Initialization:

- **Tape 1:** The input string (for example, `1011100`) is written on Tape 1, and the head is placed at the leftmost symbol.

1	1	0	1	0	1	1	0	1	0

- **Tape 2:**
  - The transition function is encoded in a designated region. Example entries include:
    * $(q_0, 1, \text{not Y}) \rightarrow (q_0, \text{push } X)$
    * $(q_0, 1, Y) \rightarrow (q_0, \text{pop})$
    * $(q_0, 0, X) \rightarrow (q_0, \text{pop } X)$
    * $(q_0, 0, \text{-}) \rightarrow (q_0, \text{push } Y)$
  - A separate region on Tape 2 holds the current state marker, initially set to q_0.

q_0	q_1		q_0	1	*notY*	q_0	*push*	X	#	q_0	1	Y	q_o	*pop*

- **Tape 3:** The stack is initialized with the bottom-of-stack marker, $.

$	_	_	_

2. Transition Function Lookup:

- Before processing each input symbol from Tape 1, the machine reads:
  - The current state from the State Marker Region on Tape 2.
  - The current input symbol from Tape 1.
  - The top symbol from Tape 3 (the simulated stack).
- It then scans the Transition Function Encoding Region on Tape 2 to find the rule corresponding to the triple (current state, input symbol, top symbol).

3. Processing the Input:

1. **Processing a 1:**
   - If the top symbol on Tape 3 is not Y (i.e., it is either X or $), then the rule $(q_0, 1, \text{not } Y) \rightarrow (q_0, \text{push } X)$ is applied. The machine moves to the right blank cell in the stack region and writes an X.
   - If the top symbol is Y, then the rule $(q_0, 1, Y) \rightarrow (q_0, \text{pop})$ applies.
   - The head on Tape 1 moves right to the next symbol.

2. **Processing a 0:**

   - If the top symbol on Tape 3 is X, then the rule $(q_0, 0, X) \to (q_0, \text{pop } X)$ is applied. The machine pops the X (i.e., erases it and moves the stack head left).
   - If the top symbol is not X (i.e., it is Y or \$), then the rule $(q_0, 0, \text{not } X) \to (q_0, \text{push } Y)$ applies. The machine moves to the rightmost blank cell in the stack region and writes a Y.
   - The head on Tape 1 moves right to the next symbol.

4. Final Check and Acceptance:

- After Tape 1 has been completely read (i.e., the head reaches a blank or end-of-input marker), the machine inspects the stack on Tape 3.
- If at least one X remains on the stack above the bottom marker \$, then the input is accepted (indicating that there are more 1's than 0's).
- Otherwise, the input is rejected.

Summary:
The three-tape Turing machine simulation works as follows:

1. **Tape 1** holds the input string.
2. **Tape 2** encodes the PDA's transition function and maintains the current state (q_0).
3. **Tape 3** simulates the stack, starting with a bottom marker \$. When a 1 is read, if the top is not Y, an X is pushed; if the top is Y, pop operation is performed. When a 0 is read, if the top is X, an X is popped; if the top is not X, a Y is pushed.

At the end of the input, the machine accepts if and only if the stack contains at least one X (above the bottom marker), which confirms that the number of 1's exceeds the number of 0's.