Aufgabe 4

Tim Neutze
5578777
Lorenzo Tecchia
5581906
2023.06.22

Contents

Chapter 1

Task 3

Q: Set of states Σ: Input alphabet ($\Sigma = \{0, 1, \#\}$)
Γ: Stack alphabet ($\Gamma = \{0, 1, Z_0\}$)
δ: Transition function
q_0: Start state
Z_0: Initial stack symbol
F: Set of accepting states
$Q = q_0, q_1, q_2, q_3, q_4, q_5$
$\delta = \{0, 1, \#\}$
$\Gamma = \{0, 1, \#, Z_0\}$
q_0 = Start state Z_0 = Initial stack symbol
$F = q_5$
The transition function δ is defined as follows:
$\delta(q_0, \epsilon, Z_0) = (q_1, Z_0)$ - Move to q1 and push Z0 onto the stack.
$\delta(q_1, 0, Z_0) = (q_1, 0Z_0)$ - Push 0 onto the stack.
$\delta(q_1, 1, Z_0) = (q_1, 1Z_0)$ - Push 1 onto the stack.
$\delta(q_1, \#, Z_0) = (q_2, Z_0)$ - Move to q2 and pop Z0 from the stack.
$\delta(q_2, \epsilon, Z_0) = (q_3, Z_0)$ - Move to q3 and push Z0 onto the stack.
$\delta(q_3, 0, 0) = (q_3, 0)$ - Match 0 on the stack.
$\delta(q_3, 1, 1) = (q_3, 1)$ - Match 1 on the stack.
$\delta(q_3, \epsilon, Z_0) = (q_4, Z_0)$ - Move to q4 and pop Z0 from the stack.
$\delta(q_4, \epsilon, Z_0) = (q_5, Z_0)$ - Move to q5 and pop Z0 from the stack.

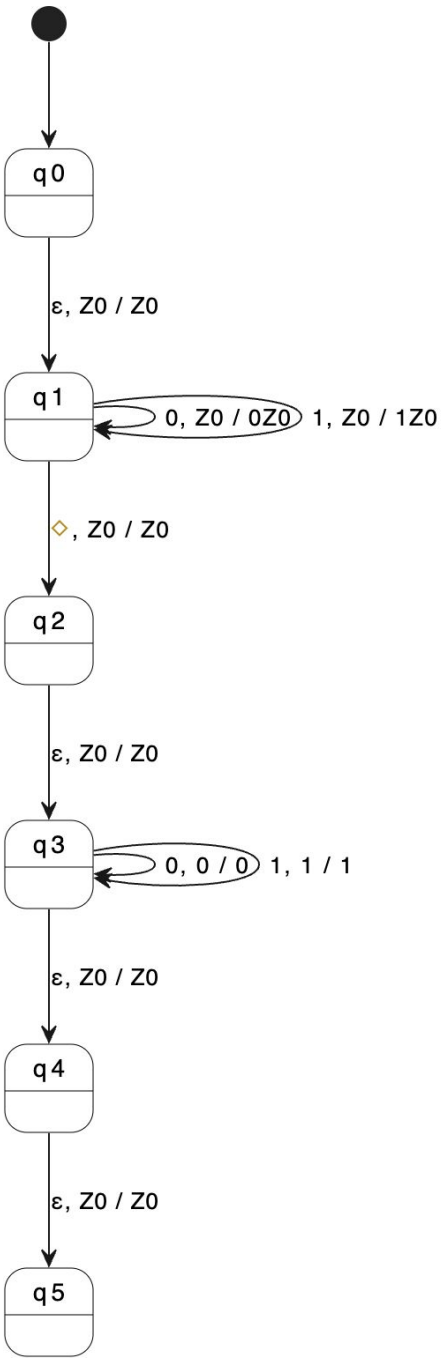

Figure 1.1: PDA

Chapter 2

Task 4

Chapter 3

Task 6

3.1 6.a

To show that if L is a decidable language, then L* (the Kleene star of L) is also a decidable language, we need to demonstrate that there exists a Turing machine that can decide L*.

Let's assume that L is a decidable language, which means there exists a Turing machine M that can decide L. We will construct a new Turing machine N that can decide L*.

Turing machine N works as follows on input w:

1. N first splits w into all possible partitions of substrings: w = $w_1 w_2 \dots w_k$, where each wi is a substring of w.

2. For each partition, N simulates M on each substring wi.
   - If M accepts every substring w_i, N proceeds to the next partition.
   - If M rejects any substring w_i, N rejects the partition and moves on to the next one.

3. If N has gone through all possible partitions and M has accepted every substring in each partition, N accepts w. Otherwise, N rejects w.

By construction, N will accept w if and only if every substring of w is accepted by M. This means N can decide L*.

Therefore, if L is a decidable language, then L* is also a decidable language.

3.2 6.b

We can prove this by showing that if a language L is semi-decidable, there exists a Turing machine that can semi-decide L*, i.e., accept all strings in L* and either reject or loop indefinitely on strings not in L*.

Assume L is a semi-decidable language, which means there exists a Turing machine M that can enumerate the strings in L. We will construct a new Turing machine N that can semi-decide L*.

Turing machine N works as follows:

1. N receives an input string w.
2. N generates all possible partitions of substrings of w: w = $w_1 w_2 \dots w_k$, where each w_i is a substring of w.
3. For each partition, N simulates M on each substring w_i.
   - If M accepts any substring wi, N proceeds to the next partition.
   - If M rejects or loops indefinitely on any substring wi, N rejects the partition and moves on to the next one.
4. If N has gone through all possible partitions and M has accepted at least one substring in each partition, N accepts w. Otherwise, N rejects w.

By construction, N will accept w if and only if there exists a partition of w such that M accepts at least one substring in each partition. This implies that all strings in L* will be accepted by N.

However, it is important to note that N may loop indefinitely on some inputs not in L*. This is because if M loops indefinitely on any substring w_i, N will also loop indefinitely on the corresponding partition. Thus, N is only semi-decidable on L*.

Therefore, if L is a semi-decidable language, then L* is also semi-decidable.

Chapter 4

Task 7

4.1 7.a

Encode each symbol from the input alphabet of the original Turing machine into a sequence of symbols from the {0, 1, _} tape alphabet. This encoding should be designed in such a way that the original symbols can be uniquely decoded later.

Simulation of States and Transitions: Define a mapping between the states of the original Turing machine and states of the {0, 1, _} Turing machine.
For each transition in the original Turing machine, define a corresponding set of transitions in the {0, 1, _} Turing machine that mimic the behavior of the original transition. This includes reading and writing symbols on the tape, moving the tape head, and transitioning to the next state.
The encoding of symbols ensures that the {0, 1, _} Turing machine can correctly interpret the transitions according to the behavior of the original Turing machine.

Simulation of Tape:
As the {0, 1, _} Turing machine executes the transitions, it reads and writes the encoded symbols on its tape. Whenever an encoded symbol is read, it is decoded back to the original symbol from the arbitrary tape alphabet. Similarly, whenever a symbol needs to be written on the tape, it is encoded before being written.

By encoding and decoding the symbols appropriately and simulating the behavior of the original Turing machine using the {0, 1,_ } Turing machine, we can effectively simulate a Turing machine with an arbitrary tape alphabet using a Turing machine with the tape alphabet {0, 1, }.

4.2 7.c

Simulating a Deterministic Finite Automaton (DFA) through a Turing machine involves creating a Turing machine that mimics the behavior of the DFA. The DFA can either accept or reject an input string based on its current state and the transition function, while a Turing machine has multiple acceptance modes, including accept by final state, accept by halting, or accept by empty tape. Let's discuss the steps involved in simulating a DFA using a Turing machine and how the different acceptance modes can be implemented:

1.

2. Input Encoding:
   - The Turing machine needs to encode the input in a format that can be processed by its tape alphabet. Common encoding methods include using symbols to represent individual characters of the input alphabet or encoding the input as a string of symbols separated by special markers.
   - The encoding should allow the Turing machine to read the input one symbol at a time, just like the DFA.

3. Tape Configuration: - The DFA's input string is represented on the Turing machine's tape.
   - The Turing machine's tape initially contains the encoded input string, along with additional symbols to indicate the current position and any necessary markers for partitioning the input.

4. Transition Function Simulation:
   - The Turing machine needs to simulate the DFA's transition function.
   - Based on the current state and the symbol read from the tape, the Turing machine moves to the appropriate state, updates the tape head's position, and writes the corresponding symbol on the tape.

5. Acceptance Modes:
   - Accept by Final State: The Turing machine can be designed to transition to an accepting state when it reaches the end of the input string, indicating acceptance by final state.
   - Accept by Halting: The Turing machine can halt when it reaches the end of the input string without transitioning to an accepting state. This indicates acceptance by halting.
   - Accept by Empty Tape: After processing the input string, the Turing machine can continue scanning the tape and reject if it encounters any non-blank symbols. If the tape becomes empty, it indicates acceptance by empty tape.

6. Rejection:
   - If, during the simulation, the Turing machine encounters an undefined transition for the current state and symbol combination, it can transition to a designated rejecting state or reject by halting.

By designing the Turing machine to simulate the behavior of the DFA's transition function, tape configuration, and acceptance modes, we can effectively simulate a DFA using a Turing machine. The specific implementation details will depend on the particular DFA and the desired acceptance mode(s) of the Turing machine.

Chapter 5

Task 8

To design a Turing machine that increments a binary number represented by the input tape and halts in different states depending on whether the input is empty or not, we can follow these steps:

1. Set up the Turing machine:
   - Define the states: q_start, q_check, $q_increment$, q_hold, q_no.
   - Define the tape alphabet: {0, 1, #}, where # represents a blank symbol.
   - Set the initial state to q_start.

2. Read and validate the input:
   - The Turing machine starts in state q_start and scans the input tape from left to right.
   - If the input tape is empty (contains only # symbols), transition to the q_no state and halt.

3. Move to the rightmost symbol:
   - While scanning the tape, move to the rightmost non-blank symbol (the least significant bit) of the input.

4. Check the rightmost symbol:
   - In state q_check, check the value of the rightmost symbol.
   - If it is 0, transition to $q_increment$.
   - If it is 1, move left to the next bit.

5. Increment the binary number:
   - In state $q_increment$, change the rightmost 0 to 1.
   - Move the tape head back to the rightmost position.

6. Carry propagation:
   - Move left until a 0 is encountered.
   - Change that 0 to 1.
   - Move the tape head back to the rightmost position.

7. Repeat steps 4-6 until no more carry propagations are required:
   - Continue moving left, checking and incrementing bits until no more carry propagations are needed (encountering a 0).

8. Halt and output the result:
   - In state q_hold, halt the Turing machine.
   - The tape will now contain the binary representation of (w + 1), where w is the original binary number provided as input.

The Turing machine described above will halt in different states depending on whether the input is empty or not. If the input is empty, it will halt in state q_no. If the input is a non-empty binary number, it will halt in state q_hold after incrementing the binary number by 1.