ICS 2018 Problem Sheet #11 Solutions

Tianyao Chen

December 6, 2018

1 Problem 11.1

1.
The FSM $(\Sigma, S, s_0, \delta, F)$ with $\Sigma = \{a, b\}, S = \{S0, S1, S2\}$, $s_0 = S0$, $F = \{S2\}$, and

$$\delta = \{((S0, a), S1), ((S0, b), S2),\\ ((S1, a), S1), ((S1, b), S1),\\ ((S2, a), S0), ((S2, b), S2)\}$$

2.

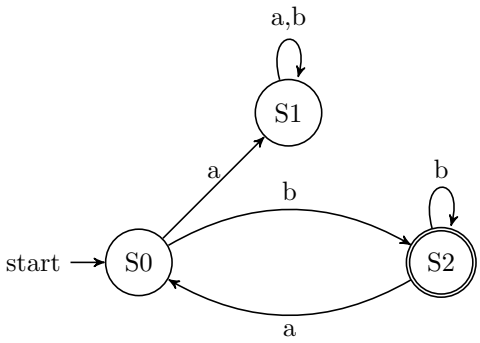

Figure 1: The FSM of Problem 11.1

3.

```haskell
data State = S0 | S1 | S2

accepts :: State -> String -> Bool
accepts S0 ('a':xs) = accepts S1 xs
accepts S0 ('b':xs) = accepts S2 xs
accepts S1 ('a':xs) = accepts S1 xs
accepts S1 ('b':xs) = accepts S1 xs
accepts S2 ('a':xs) = accepts S0 xs
accepts S2 ('b':xs) = accepts S2 xs
accepts S2 [] = True
accepts _ _ = False

decide :: String -> Bool
decide = accepts S0
```

4.
The language L can be generated by the (right) regular grammar $G_r = (N, \Sigma, P_r, S)$ with

- $N = \{S, T\}$
- $\Sigma = \{a, b\}$

- start symbol S
- $P_r = \{S \mapsto bT, T \mapsto aS, T \mapsto bT, T \mapsto \epsilon\}$

2 Problem 11.2

a)

The Turing Machine here works like this: first, in state S0, the head hits the $ sign and changes to state S1; second, in state S1, the head moves to the right until the sign $ is hit; third, in state S2, the head moves to the left while changing 1s to 0s until a 0 is hit and changed to 1; finally, the state S3 is the accepting state and the Turing Machine halts.

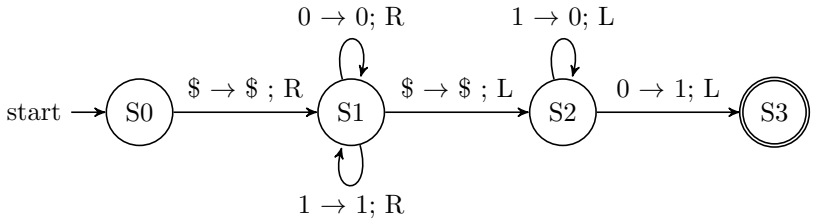

Figure 2: The Turing Machine of Problem 11.2 a

```haskell
--This is the Increment Turing Machine for problem 11.2a
-- we can test this program , for example, by calling run "£0101£", and we get "£0110£."
import Prelude hiding (head)

data State = S0 | S1 | S2 | S3 deriving (Show)
data Tape = Tape String Int deriving (Show)

head :: Tape -> Char -> Bool
head (Tape xs i) c = xs !! i == c

content :: Tape -> String
content (Tape xs _) = xs

left :: Tape -> Tape
left (Tape xs i)
    | i == 0 = Tape ("0" ++ xs) 0
    | otherwise = Tape xs (i - 1)

right :: Tape -> Tape
right (Tape xs i)
    | i + 1 == length xs = Tape (xs ++ "0") (i + 1)
    | otherwise = Tape xs (i + 1)

write :: Tape -> Char -> Tape
write (Tape xs i) c = Tape (take i xs ++ [c] ++ drop (i + 1) xs) i

delta :: State -> Tape -> Tape
delta S0 tape
    | head tape '$' = delta S1 $ right $ write tape '$'
delta S1 tape
    | head tape '0' = delta S1 $ right $ write tape '0'
    | head tape '1' = delta S1 $ right $ write tape '1'
    | head tape '$' = delta S2 $ left $ write tape '$'
delta S2 tape
    | head tape '0' = delta S3 $ left $ write tape '1'
```

```
    | head tape '1' = delta S2 $ left $ write tape '0'
delta S3 tape = tape

run :: String -> String
run xs = content (delta S0 (Tape xs 0))
```

b)

The Turing Machine here works like this: first, in state S0, the head the head hits the $ sign and changes to state S1; second, in state S1, the head moves to the right while inverting the bit string until the sigh $ is hit; third, in state S2, the head moves to the left while changing 1s to 0s until a 0 is hit and changed to 1; fourth, in state S3, the head keeps moving to the left until the sign $ is hit; fifth, in state S4, the head, again, moves to the right while inverting the bit string until the sign $ is hit; finally, the state S5 is the accepting state and the Turing Machine halts.

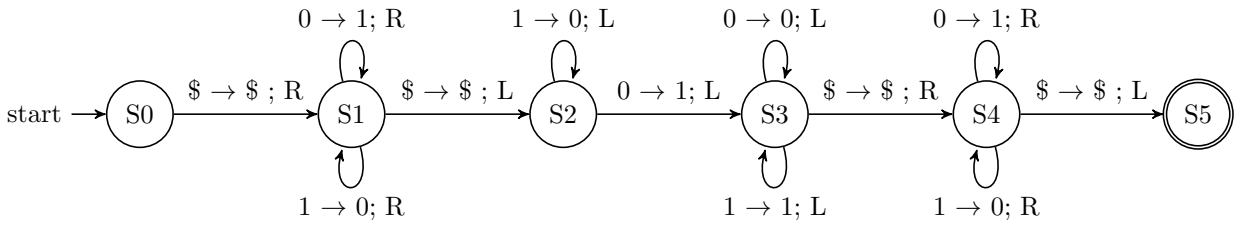

Figure 3: The Turing Machine of Problem 11.2 b

```
--This is the Decrement Turing Machine for problem 11.2b
-- we can test this program , for example, by calling run "£0111£", and we get "£0110£."
import Prelude hiding (head)

data State = S0 | S1 | S2 | S3 | S4 | S5 deriving (Show)
data Tape = Tape String Int deriving (Show)

head :: Tape -> Char -> Bool
head (Tape xs i) c = xs !! i == c

content :: Tape -> String
content (Tape xs _) = xs

left :: Tape -> Tape
left (Tape xs i)
    | i == 0 = Tape ("0" ++ xs) 0
    | otherwise = Tape xs (i - 1)

right :: Tape -> Tape
right (Tape xs i)
    | i + 1 == length xs = Tape (xs ++ "0") (i + 1)
    | otherwise = Tape xs (i + 1)

write :: Tape -> Char -> Tape
write (Tape xs i) c = Tape (take i xs ++ [c] ++ drop (i + 1) xs) i

delta :: State -> Tape -> Tape
delta S0 tape
    | head tape '$' = delta S1 $ right $ write tape '$'
delta S1 tape
    | head tape '0' = delta S1 $ right $ write tape '1'
    | head tape '1' = delta S1 $ right $ write tape '0'
    | head tape '$' = delta S2 $ left $ write tape '$'
```

```
delta S2 tape
    | head tape '0' = delta S3 $ left $ write tape '1'
    | head tape '1' = delta S2 $ left $ write tape '0'
delta S3 tape
    | head tape '0' = delta S3 $ left $ write tape '0'
    | head tape '1' = delta S3 $ left $ write tape '1'
    | head tape '$' = delta S4 $ right $ write tape '$'
delta S4 tape
    | head tape '0' = delta S4 $ right $ write tape '1'
    | head tape '1' = delta S4 $ right $ write tape '0'
    | head tape '$' = delta S5 $ left $ write tape '$'
delta S5 tape = tape


run :: String -> String
run xs = content (delta S0 (Tape xs 0))
```

c)

The Turing Machine here works like this: first, it goes through the decrement Turing Machine steps and gets to state S5; second, it goes through the increment Turing Machine steps and gets to state S7; third, the head keeps moving to the left until the second $ sign is hit and the Turing Machine changes to state S8; fourth, the head goes through the first bit string from the right to left to check whether it's value is 0: if all the digits are 0, the head hits the first $ sign and halts in state S10; if the head hits a digit 1, the Turing Machine changes to state 9 and the head keeps moving to the left until and first $ sigh is hit and gets to state S1 again.

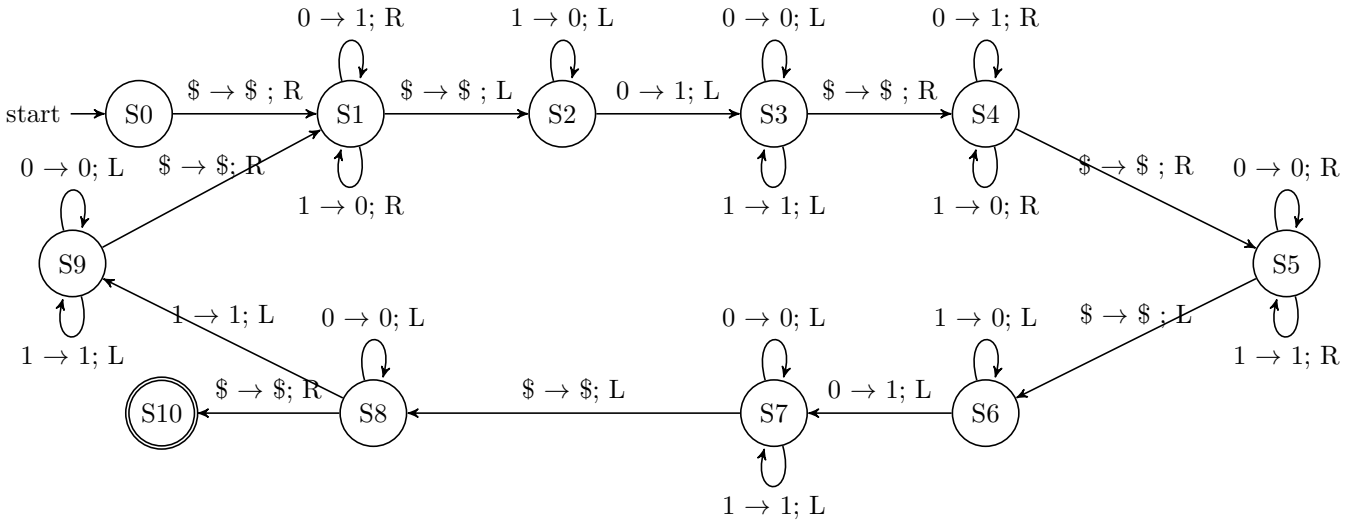

Figure 4: The Turing Machine of Problem 11.2 c

```
--This is the Binary Adder Turing Machine for problem 11.2c
-- we can test this program , for example, by calling run "£0100£0010£", and we get "£0000£0110£."
import Prelude hiding (head)

data State = S0 | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 deriving (Show)
data Tape = Tape String Int deriving (Show)

head :: Tape -> Char -> Bool
head (Tape xs i) c = xs !! i == c

content :: Tape -> String
content (Tape xs _) = xs
```

```haskell

left :: Tape -> Tape
left (Tape xs i)
    | i == 0 = Tape ("0" ++ xs) 0
    | otherwise = Tape xs (i - 1)

right :: Tape -> Tape
right (Tape xs i)
    | i + 1 == length xs = Tape (xs ++ "0") (i + 1)
    | otherwise = Tape xs (i + 1)

write :: Tape -> Char -> Tape
write (Tape xs i) c = Tape (take i xs ++ [c] ++ drop (i + 1) xs) i

delta :: State -> Tape -> Tape
delta S0 tape
    | head tape '$' = delta S1 $ right $ write tape '$'
delta S1 tape
    | head tape '0' = delta S1 $ right $ write tape '1'
    | head tape '1' = delta S1 $ right $ write tape '0'
    | head tape '$' = delta S2 $ left $ write tape '$'
delta S2 tape
    | head tape '0' = delta S3 $ left $ write tape '1'
    | head tape '1' = delta S2 $ left $ write tape '0'
delta S3 tape
    | head tape '0' = delta S3 $ left $ write tape '0'
    | head tape '1' = delta S3 $ left $ write tape '1'
    | head tape '$' = delta S4 $ right $ write tape '$'
delta S4 tape
    | head tape '0' = delta S4 $ right $ write tape '1'
    | head tape '1' = delta S4 $ right $ write tape '0'
    | head tape '$' = delta S5 $ right $ write tape '$'
delta S5 tape
    | head tape '0' = delta S5 $ right $ write tape '0'
    | head tape '1' = delta S5 $ right $ write tape '1'
    | head tape '$' = delta S6 $ left $ write tape '$'
delta S6 tape
    | head tape '0' = delta S7 $ left $ write tape '1'
    | head tape '1' = delta S6 $ left $ write tape '0'
delta S7 tape
    | head tape '0' = delta S7 $ left $ write tape '0'
    | head tape '1' = delta S7 $ left $ write tape '1'
    | head tape '$' = delta S8 $ left $ write tape '$'
delta S8 tape
    | head tape '0' = delta S8 $ left $ write tape '0'
    | head tape '1' = delta S9 $ left $ write tape '1'
    | head tape '$' = delta S10 $ right $ write tape '$'
delta S9 tape
    | head tape '0' = delta S9 $ left $ write tape '0'
    | head tape '1' = delta S9 $ left $ write tape '1'
    | head tape '$' = delta S1 $ right $ write tape '$'
delta S10 tape = tape

run :: String -> String
run xs = content (delta S0 (Tape xs 0))
```