Assignment Three

Abanob Tawfik
z5075490

April 2019

1 Problem 1

The skip command is an $\mathcal{L}$ program that has the effect of "do nothing". That is, skip; P has the same behaviour as P;skip and the same behaviour as P.

(a) Define skip using the default $\mathcal{L}$ commands – that is, write an $\mathcal{L}$ program for skip. (3 marks)

Solution:
A very simple and short program to achieve this could be assigning a variable to itself, in turn having no effect on the current state of the program.

```
x := x;
```

by setting the variable to itself when the following program is executed we have not modified our state and this has no effect.

(b) Based on your definition and the rules of denotational semantics discussed in lectures, determine the semantic object $[\![\text{skip}]\!]$. (4 marks)

Solution:
The semantic object $[\![\text{skip}]\!]$ also defined as $[\![\text{x} := \text{x}]\!]$ can be defined by the following:
$[\![\text{skip}]\!] = \{(\eta,\eta')$ if and only if $\eta' = \eta[\text{x} \mapsto [\![x]\!]^{\eta}\}]$.
The binary predicate for our semantic object $[\![\text{skip}]\!]$ will always be true in any state as this can simply be seen as a loop to the current state since we are not modifying our state.

(c) Suppose skip was a default command in L. Propose a suitable rule for Hoare Logic that handles skip. (3 marks)

Solution:
Since our program skip has the effect of doing nothing, we can define our program skip having the same pre and post condition in the following:
Suppose our program is in the state ψ before execution; then our skip program can be given the Hoare Logic rule

$$\{\psi\} \text{ skip } \{\psi\}.$$

2 Problem 2

Recall the diagonally-moving robot example from the lectures: from position (x, y) the robot can move to
any of: (x + 1, y + 1), (x + 1, y - 1), (x - 1, y + 1), (x - 1, y - 1).

(a) Write a program in $\mathcal{L}^+$ that, on termination, will confirm that a location (m, n) is reachable by the robot starting at (0, 0). That is, the program Reach should be such that the Hoare triple

$$\{(x = 0) \wedge (y = 0)\}\ \text{REACH}\ \{(x = m) \wedge (y = n)\}$$

is valid if and only if (m, n) is reachable from (0, 0). Note: your program does not have to terminate. (14 marks)

Solution:
For this solution to save space we will use labels for each transition in the following:
NE = (x := x + 1; y := y + 1;)
SE = (x := x + 1; y := y - 1;)
NW = (x := x - 1; y := y + 1;)
SW = (x := x - 1; y := y = 1;)
guard = ¬((x = m) ∧ (y = n))

```
x := 0;
y := 0;
// program for reach starts here!!!
// if this program were to be written in ℒ then we would have a loop
// which exits when x = m and y = n, and on each iteration we would check that
// our transition for x and y gets us closer to the goal, however since ℒ+
// is non-deterministic we can make our program choose between all transitions for us
// below is a while b do .. od loop in ℒ+
(guard;NE + SE + NW + SW)*;¬guard;
```

The above program was written purely in $\mathcal{L}^+$, however it can also be written using a while loop to make it more clear what is going on. we will not be using this program below for the Hoare logic derivation, we will be using the program above which was written purely in $\mathcal{L}^+$.

```
x := 0;
y := 0;
// program for reach starts here!!!
while guard do
// non deterministically choose our next transition
    (NE + SE + NW + SW);
od;
```

A quick explanation of the program:
Since the program is written in $\mathcal{L}^+$, we can utilise non-determinism to choose the transition that optimally reaches the goal state. We create a loop which has the guard x is not equal to m and y is not equal to n, and within the loop we will constantly perform transitions non-deterministically. IF the program terminates (successful exit of while loop) then we have the statement ¬guard, which ends up being ¬¬((x = m) ∧ (y = n)). By using double negation

elimination, we end up with x = m ∧ y = n if and only if the program terminates. The first program was written purely in $\mathcal{L}^+$, however the second program was written using the while rule from $\mathcal{L}$ and the remaining written in $\mathcal{L}^+$. This was done because in the second program it is clearer to see how the transitions occur within the loop where it may not be as obvious in the first program. Note that also termination will only occur when m + n is even. Due to the nature of movement, we begin on tile (0, 0), and 0 + 0 is even. Movement in transition x + 1; y + 1, we get a total change of 2 which is also even. Movement in transition x - 1; y - 1, we get a total change of -2 which is also even. Movement in transition x + 1; y - 1, we get a total change of 0 which is also even. Movement in transition x - 1; y + 1, we get a total change of 0 which is also even. Since our initial state (0, 0) has x + y = 0, and all transitions have an even net change {0, 2, -2} this means we can only move to states where x + y is even.

(b) Prove that your program is correct (i.e. show the validity of the above Hoare triple). Annotating your code with appropriate assertions, as long as the proof is recoverable, is sufficient. (6 marks)

Solution:
To perform this proof we will first acquire our Hoare logic derivation, and use this to annotate the programs. For this solution conditions have been compacted in the following:
{invariant} = {(x + y) mod 2 = 0}
NE = (x := x + 1; y := y + 1;)
SE = (x := x + 1; y := y - 1;)
NW = (x := x - 1; y := y + 1;)
SW = (x := x - 1; y := y = 1;)
guard = ¬((x = m) ∧ (y = n))

1. {0 = 0} x := 0 {x = 0} (assignment inference rule)
2. {0 = 0} y := 0 {y = 0} (assignment inference rule)
3. {0 = 0 ∧ 0 = 0} x := 0; y := 0 {x = 0 ∧ y = 0} (assignment inference rule)
4. {invariant} x := x + 1; {¬((x + y) mod 2 = 0)} (proof obligation)
5. {¬((x + y) mod 2 = 0)} y := y + 1; {invariant} (proof obligation)
6. {invariant} x := x + 1; y := y + 1; {invariant} (sequence inference rule: 4, 5)
7. {invariant} NE {invariant} (substitution)
8. {invariant} x := x + 1; {¬((x + y) mod 2 = 0)} (proof obligation)
9. {¬((x + y) mod 2 = 0)} y := y - 1; {invariant} (proof obligation)
10. {invariant} x := x + 1; y := y - 1; {invariant} (sequence inference rule: 8, 9)
11. {invariant} SE {invariant} (substitution)
12. {invariant} x := x - 1; {¬((x + y) mod 2 = 0)} (proof obligation)
13. {¬((x + y) mod 2 = 0)} y := y + 1; {invariant} (proof obligation)
14. {invariant} x := x - 1; y := y + 1; {invariant} (sequence inference rule: 12, 13)
15. {invariant} NW {invariant} (substitution)
16. {invariant} x := x - 1; {¬((x + y) mod 2 = 0)} (proof obligation)
17. {¬((x + y) mod 2 = 0)} y := y - 1; {invariant} (proof obligation)
18. {invariant} x := x - 1; y := y - 1; {invariant} (sequence inference rule: 16, 17)

19. {invariant} SW {invariant} (substitution)
20. {invariant} NE + SE {invariant} (choice inference rule: 7, 11)
21. {invariant} NE + SE + NW{invariant} (choice inference rule: 20, 15)
22. {invariant} NE + SE + NW + SW{invariant} (choice inference rule: 21, 19)
23. {invariant $\wedge$ guard} (NE + SE + NW + SW)* {invariant $\wedge$ guard} (Loop inference rule: 22)
24. {invariant $\wedge$ guard} LOOP {invariant $\wedge \neg$guard} (Loop sequence in $\mathcal{L}^+$: 23)
25. {x = 0 $\wedge$ y = 0} REACH {invariant $\wedge \neg$ guard} (seq)
26. (invariant $\wedge \neg$guard)$\rightarrow$(x = m $\wedge$ y = n) (proof obligation)
27. {x = 0 $\wedge$ y = 0} REACH {x = m $\wedge$ y = n} (consequence inference rule: 25, 26)

Proof obligations used in the Hoare logic derivation: In the Hoare logic we assume our invariant holds before the loop begins, since x + y = 0 + 0 = 0 which is even. 0 $\mod 2 = 0$. Note our substitutions are the following:
{invariant} = {(x + y) $\mod 2 = 0$}
NE = (x := x + 1; y := y + 1;)
SE = (x := x + 1; y := y - 1;)
NW = (x := x - 1; y := y + 1;)
SW = (x := x - 1; y := y = 1;)
guard = $\neg$((x = m) $\wedge$ (y = n))

Proof for Line 4 and line 5

Line 4 - {invariant} x := x + 1; {$\neg$((x + y) $\mod 2 = 0$)}
And
Line 5 - {$\neg$((x + y) $\mod 2 = 0$)} y := y + 1; {invariant}
An informal way for the proof obligation:
On line 4 before execution we have x + y is even, then when we add 1 to x, so x + y is no longer even. This is because adding 1 to an even number will make it odd. after we add 1 to y on line 5, we get x + y is even again because when we add 1 to an odd number we get an even number. More formally this can be shown as:

Line 4 - {invariant} x := x + 1; {$\neg$((x + y) $\mod 2 = 0$)}
initially we have our invariant (x + y) $\mod 2 = 0$. Then we add 1 to x, so if we add 1 on both sides we get (x + y) $\mod 2 = 1$ (note x = x + 1 after x := x + 1 execution). Since (x + y) $\mod 2 = 1$, this also implies $\neg$((x + y) $\mod 2 = 0$).

Line 5 - {$\neg$((x + y) $\mod 2 = 0$)} y := y + 1; {invariant}
initially we have $\neg$((x + y) $\mod 2 = 0$) which is equivalent to (x + y) $\mod 2 = 1$, since any number mod 2 can only be 0 or 1. After execution of y := y + 1, if we add 1 to both sides we get (x + y) $\mod 2 = 0$ (note y = y + 1 after y := y + 1 execution) which is also our invariant.

The proof method above is used for the following proof obligations:

- Line 8 and Line 9
- Line 12 and Line 13
- Line 17 and Line 18

Since each transition follows the same manner, where the total distance changed is either $\pm$ 2 or 0 the change can be seen in the following for all transitions:

- (x + 1, y + 1) NE change = 1 + 1 = 2 (even)
- (x + 1, y - 1) SE change = 1 - 1 = 0 (even)
- (x - 1, y + 1) NW change = 1 - 1 = 0 (even)
- (x - 1, y - 1) SW change = -1 - 1 = -2 (even)

Also note the definition of an even is (even number) $\mod 2 = 0$.

proof for line 26

Line 26 - (invariant $\wedge$ $\neg$guard)$\rightarrow$(x = m $\wedge$ y = n)
using our substitution we can get:
$(((x + y) \mod 2 = 0) \wedge \neg(\neg(x = m \wedge y = n))) \rightarrow (x = m \wedge y = n)$.
We can use and elimination to simplify down to:
$(\neg(\neg(x = m \wedge y = n)) \rightarrow (x = m \wedge y = n)$
This is simply just a proof of double negation elimination which is done by natural deduction.

1. $\neg\neg\psi$	
2. $\neg\psi$	
3. $\perp$	$\perp$ **Intro:** 1, 2
4. ψ	IP: 1-3

As shown by natural deduction $\neg\neg(x = m \wedge y = n) \rightarrow (x = m \wedge y = n)$. (note our loop will never terminate if our invariant doesn't hold (if m + n is not even it will never terminate).

By using our Hoare logic deduction for our program in $\mathcal{L}^+$ we can annotate our program in the following. (please note our program will not terminate if m + n is not even as it will never be able to transition to a tile on those coordinates, we can add a if statement if need be to assure termination is always possible).
The proof will be performed in the next page so it is not split between pages.

ALSO NOTE i will be expanding out my substitutions NE, SE, NW, SW for this proof, and the following will be substituted for the proof:
guard = $\neg((x = m) \wedge (y = n))$
invariant = $(x + y) \mod 2 = 0$.

```
                                    //{0 = 0}
x := 0;                             //{x = 0}
y := 0;                             //{y = 0}
// program for reach starts here!!!
                                    //{x = 0 ∧ y = 0}
(guard;
                                    //{invariant}
(x := x + 1;                        //{¬((x + y) mod 2 = 0)}
y := y + 1);                        //{invariant}
+                                   //{invariant}
(x := x + 1;                        //{¬((x + y) mod 2 = 0)}
y := y - 1);                        //{invariant}
+                                   //{invariant}
(x := x - 1;                        //{¬((x + y) mod 2 = 0)}
y := y + 1);                        //{invariant}
+                                   //{invariant}
(x := x - 1;                        //{¬((x + y) mod 2 = 0)}
y := y - 1);)*;                     //{invariant}
¬guard;                             //{x = m ∧ y = n}
```

Using our substitutions NE, SE, NW, SW we can simply further to get the exact same proof.

```
                                    //{0 = 0}
x := 0;                             //{x = 0}
y := 0;                             //{y = 0}
// program for reach starts here!!!
                                    //{x = 0 ∧ y = 0}
(guard;
                                    //{invariant}
NE                                  //{invariant}
+
                                    //{invariant}
SE                                  //{invariant}
+
                                    //{invariant}
NW                                  //{invariant}
+
                                    //{invariant}
SW);)*;                             //{invariant}
¬guard;                             //{x = m ∧ y = n}
```

3 Problem 3

Suppose you have ten coins arranged in a line. A move consists of taking any three adjacent coins and turning them over (changing heads to tails and vice-versa). For example, if the coins were arranged as:

HHTTHTHTHT

one move could be to flip the second, third and fourth coins to get the arrangement:

HTHHHTHTHT

(a) Model this situation as a transition system, carefully defining your states and the transition relation. (4 marks)

Solution:
To model this problem, we can use a Boolean form representation of the coins, true = heads, false = tails. We have 10 Boolean values and we take the cross product with the integers 1-10 for position, so each element of our set is a Boolean for head/tails and a integer for position 1-10.
For our transition system, our set of states S is given by the following:
S = {a, b, c, d, e, f, g, h, i, j} where a, b, c, d, e, f, g $\in \mathbb{B} \times [1, 10]$ for example the initial state above could be the following:
{(true,1), (true,2), (false,3), (false,4), (true,5), (false,6), (true,7), (false,8), (true,9), (false,10)}
Our transition relation $\rightarrow$ can be defined as taking three adjacent elements of our set and flipping the Boolean expression using the ! Boolean operator (!true = false and !false = true).
$\rightarrow$ is given by

- (a = (!bool), 1)) $\wedge$ (b = (!bool, 2)) $\wedge$ (c = (!bool), 3))
- (b = (!bool), 2)) $\wedge$ (c = (!bool, 3)) $\wedge$ (d = (!bool), 4))
- (c = (!bool), 3)) $\wedge$ (d = (!bool, 4)) $\wedge$ (e = (!bool), 5))
- (d = (!bool), 4)) $\wedge$ (e = (!bool, 5)) $\wedge$ (f = (!bool), 6))
- (e = (!bool), 5)) $\wedge$ (f = (!bool, 6)) $\wedge$ (g = (!bool), 7))
- (f = (!bool), 6)) $\wedge$ (g = (!bool, 7)) $\wedge$ (h = (!bool), 8))
- (g = (!bool), 7)) $\wedge$ (h = (!bool, 8)) $\wedge$ (i = (!bool), 9))
- (h = (!bool), 8)) $\wedge$ (i = (!bool, 9)) $\wedge$ (j = (!bool), 10))

(b) By considering the coins in positions 1,2,4,5,7,8, and 10 (i.e. positions not divisible by 3) find a preserved invariant of this system. (4 marks)

Solution:
A preserved invariant of this system would be that the coins in position 1, 2, 4, 5, 7, 8, 10 (positions not divisible by 3) preserve the parity for head and tails at their initial state, regardless of state change. For example if position 1, 2, 4, 5, 7, 8 and 10 initially has an even number of heads and an odd number of tails, then regardless of state change this will always remain true. Also note that heads and tails cannot have the same parity for these positions since there are 7 positions. A formal way to put this is:

Let parity_of_heads be the variable that has the parity of the number of heads at the initial state in position 1, 2, 4, 5, 7, 8, 10.
Let parity_of_tails be the variable that has the parity of the number of tails at the initial state in position 1, 2, 4, 5, 7, 8, 10.
let ψ_1 be the proposition that parity_of_heads remains the same on each transition.
let ψ_2 be the proposition that parity_of_tails remains the same on each transition.
Our invariant is $\psi = \psi_1 \wedge \psi_2$.
This is because on each move, we will always modify 2 of the coins in the position 1, 2, 4, 5, 7, 8 and 10. This can be seen in each transition in part (a). Since we always modify 2 of the coins we can check each combination of HH, TT, HT/TH to verify that this property will always remain true.

If we have HH, and we swap to TT, the parity will stay the same as a subtraction of 2 for T and addition of 2 to H will have no effect on parity because $2 \mod 2 = 0$.

If we have TT, and we swap to HH, the parity will stay the same as a subtration of 2 will have no effect on parity, similair to the reason above.

If we have HT/TH, if we swap to TH/HT this will also have no effect on the parity as our net change will be 0, we are simply swapping Heads to tails and tails to head so our number of heads and tails in those position stay the same and our parity remains the same.

The reason we only look at 2 coins, is that on each transition possible, we modify exactly 2 coins from the possible positions 1, 2, 4, 5, 7, 8, 10. This can be seen directly from (a).
Therefore our invariant is true, and was proven using proof by cases.

(c) Show that the arrangement $TTTTTTTTTT$ is not reachable from the arrangement HHHHHHHHHH. (2 marks)

Solution:
The arrangement $TTTTTTTTTT$ is not reachable from HHHHHHHHHH. This is because in our initial arrangement HHHHHHHHHH, The number of heads in position 1, 2, 4, 5, 7, 8, 10 is 7. $7 \mod 2 = 1$ so the parity of our heads is odd. The number of tails in those positions however is 0 which is even. Due to our invariant ψ which states that the parity of heads and parity of tails in those positions remain the same on each move, we can never reach the state $TTTTTTTTTT$, because in that state, the number of tails in those position is 7 which is odd, however our number of tails in those position MUST BE EVEN due to our invariant ψ.

4 Problem 4

Let $\Sigma = \{0,1\}^3$, so each element of Σ is a triple of symbols that are either 0 or 1.
Let $f, s, t : \Sigma^* \to \{0,1\}^*$ be the functions that take a word from Σ^* and return the word of $\{0,1\}^*$ that is defined by considering only the symbols in the first, second or third (respectively) component.
So if w = $\binom{0}{0}\binom{1}{1}\binom{1}{0}\binom{0}{1}\binom{1}{0}$, then $f(w) = 0110$, $s(w) = 0101$ and $t(w) = 0010$. Finally let
bin : $\{0,1\}^* \to \mathbb{N}$ be the function that treats a word of $\{0,1\}^*$ as the binary representation of a non-negative integer, with the last symbol being the least-significant.
So $bin(110) = bin(00110) = 6$ and $bin(\lambda) = 0$.
Design a DFA that acceptes the following language:

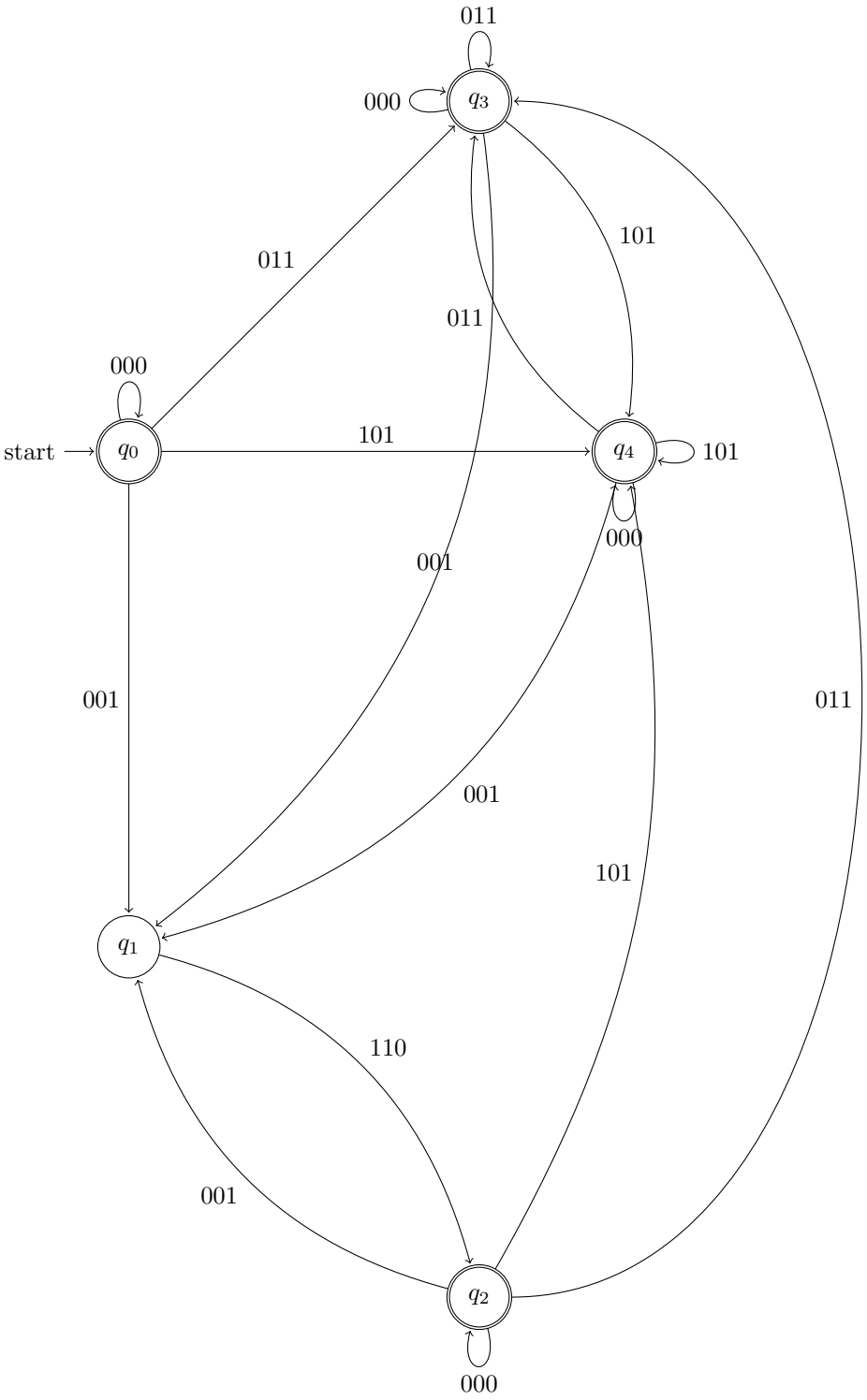
$$\{\mathrm{w} \in \Sigma^* : bin(t(\mathrm{w})) = bin(f(\mathrm{w})) + bin(s(\mathrm{w}))\}.$$

(10 marks)

Solution:
The way we will represent the input will be through a three digit integer (e.g.) 010 represents $\binom{0}{1}\binom{1}{0}\binom{0}{1}$.
PLEASE NOTE I COULD NOT FIT ALL POSSIBLE STATES SO ANY INPUT WHICH DOES NOT FOLLOW A TRANSITION WILL BE ASSUMED TO BE TAKEN TO A ERROR STATE (a non-final state) LOOPING TO ITSELF ON WHATEVER INPUT IT GETS.
The DFA is on the next page.
Below this will be a quick explanation of the DFA.
When we begin processing input we can have 3 possible transitions. We can move to a final state from 101 since 1 + 0 = 1 with no carry over. We can move to final state from 011 since 0 + 1 = 1 with no carry over. And we can move to a state which checks if the carry over was from a previous result. Since input is taken left to right and binary is read from right to left, we have to process the carry over first. If we have 001 (indicating 0 + 0 = 1) we will check if our next input has 110 (1 + 1 = 0 carry 1). From all these final states we can then add links to each other and loop 000 on all final states. We can also add loops to states that have 101 and 011 since they will always be true and have no carry over. Some example runthroughs with simple accepted and non-accepted words.

1. accepted word = $\binom{0}{0}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{0}{1}$.
   from our DFA we begin at q0 and transition to q1 from 001, next we transition to q2 from 110. we stay at q2 from self loop from 000, and transition to final state q4 from 101. Since we finish at a final state this sequence of input is accepted and to further prove it we can show:
   $bin(t(\mathrm{w})) = 1001 = 9$, $bin(f(\mathrm{w})) = 0101 = 5$ and $bin(s(\mathrm{w})) = 0100 = 4$.
   as can be seen $bin(t(\mathrm{w})) = bin(f(\mathrm{w})) + bin(s(\mathrm{w}))$ since 9 = 5 + 4.

2. rejected word = $\binom{0}{0}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{1}{1}$.
   from our DFA we begin at q0 and transition to q1 from 001, next we transition to our error state where all inputs loop to the same state which is not a final state. Since we finish at a non-final state this input is rejected. To further prove this is:
   $bin(t(\mathrm{w})) = 1101 = 13$, $bin(f(\mathrm{w})) = 0101 = 5$ and $bin(s(\mathrm{w})) = 0100 = 4$. as can be seen this is not true and rejected since 13 != 5 + 4 and our property does not hold.


011
000
q_3
011
101
011
000
start
q_0
101
q_4
101
000
001
001
011
001
101
q_1
110
001
q_2
000

5 Problem 5

Let Ł $\subseteq \Sigma^*$ be a language over Σ. Recall the definition of $\equiv_L \subseteq \Sigma^* \times \Sigma^*$:

$$w \equiv_L v \text{ iff } \forall z \in \Sigma^* : wz \in Ł \Leftrightarrow vz \in Ł$$

(a) Show that $\equiv_L$ is an equivalence relation. (6 marks)

Solution:
To prove that $\equiv_L$ is an equivalence relation we need to show that:

1. $\equiv_L$ is symmetric
   To show $\equiv_L$ is symmetric, we have to show:
   a $\equiv_L$ b $\iff$ b $\equiv_L$ a, where a, b $\in \Sigma^*$.
   This is true from the definition of $\equiv_L$ since it is a bi-directional implication,
   $\forall z \in \Sigma^* : az \in Ł \Leftrightarrow bz \in Ł$, $\equiv_L$ is symmetric.
2. $\equiv_L$ is reflexive
   To show $\equiv_L$ is reflexive, we have to show:
   a $\equiv_L$ a where a $\in \Sigma^*$.
   since $\forall z \in \Sigma^* : az \in L \Leftrightarrow az \in L$, as a = a, then $\equiv_L$ is also reflexive.
3. $\equiv_l$ is transitive
   To show $\equiv_L$ is transitive, we have to show:
   ((a $\equiv_L$ b) $\wedge$ (b $\equiv_L$ c)) $\rightarrow$ (a $\equiv_L$ c) where a, b, c $\in \Sigma^*$.
   $\forall z \in \Sigma^* : az \in L \Leftrightarrow bz \in L$. Since $bz \in L$ then $cz \in L(b \equiv_L c)$. This means that:
   $\forall z \in \Sigma^* : az \in L \Leftrightarrow bz \in L \Leftrightarrow cz \in L$. Essentially what we are saying is that if az is in the language, then bz is in the language, this in turn means cz is in the language due to (a $\equiv_L$ b) and (b $\equiv_L$ c). This means that a $\equiv_L$ c, thus $\equiv_L$ is transitive.

Since $\equiv_L$ is symmetric, reflexive and transitive, then $\equiv_L$ is an equivalence relation.

(b) if Ł is regular, show that for any word w, [w] (the equivalence class of w under $\equiv_L$) is also regular. (4 marks)

Solution:
To show that for any word w, [w] is regular under $\equiv_L$, we need to show that the equivalence class of [w] is finite. This will then allow us to use the Myhill-Nerode theorem to prove [w] is regular. Given that L is regular there are a finite amount of [w] from Myhill-Nerode theorem. Since the equivalence class [w] can be seen as a subset of our language Ł which is finite (since it is regular); then [w] must also be finite since it is a subset of Ł. Since [w] is finite then [w] is also regular.