CS 181 HW 2

Exercise 4.5 - Half and full adders.

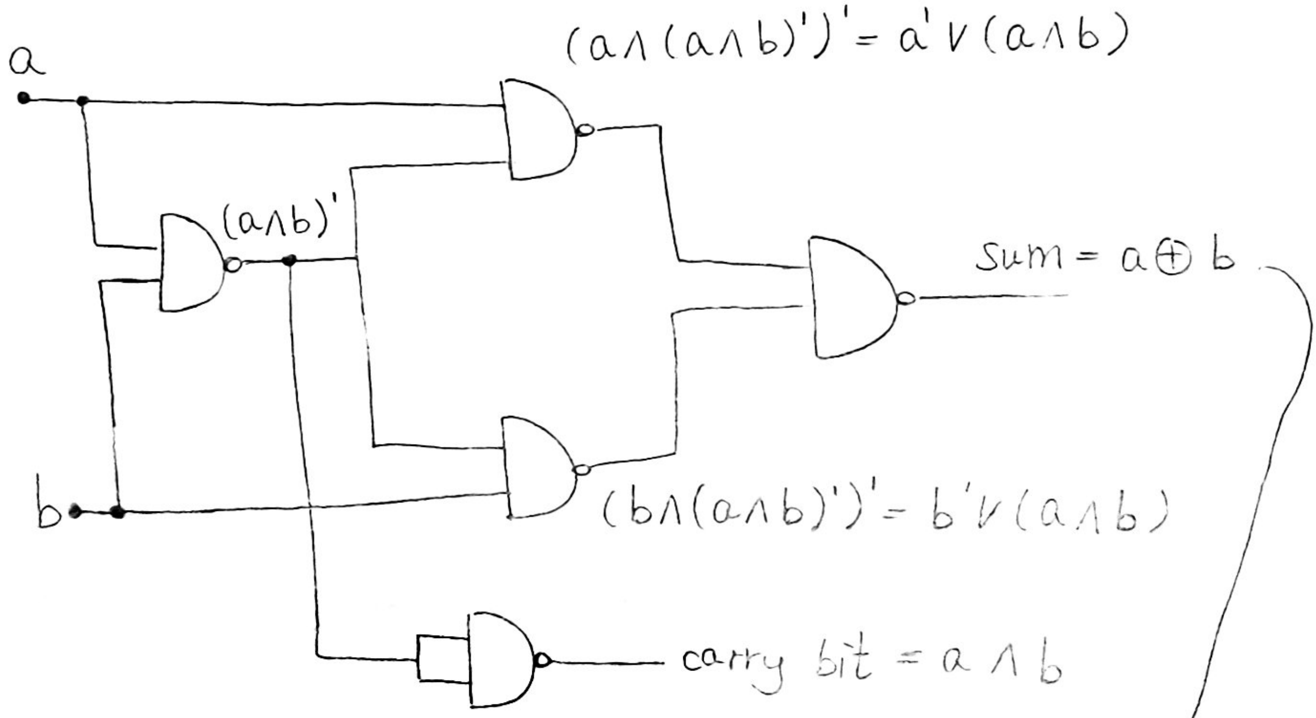
① $HA : \{0,1\}^2 \to \{0,1\}^2$ st. for every $a, b \in \{0,1\}$,

$HA(a,b) = (e,f)$ where $2e+f = a+b$

Proof Half Adder using NAND Gates

$Sum = a \text{ XOR } b = a \oplus b = (a \vee \bar{b}) \wedge (\bar{a} \vee b)$

$\text{Carry Bit} = a \text{ and } b = a \wedge b$



$$sum = [(a' \vee (a \wedge b)) \wedge (b' \vee (a \wedge b))]'$$

$$= ((a' \wedge b') \vee (a \wedge b))'$$

XNOR

$$= (a \odot b)' = a \oplus b$$ XOR

Exercise 4.5

② Full Adder, $FA : \{0,1\}^3 \to \{0,1\}^2$ st. for every $a, b, c \in \{0,1\}$, $FA(a,b,c) = (e,f)$ st. $2e+f = a+b+c$

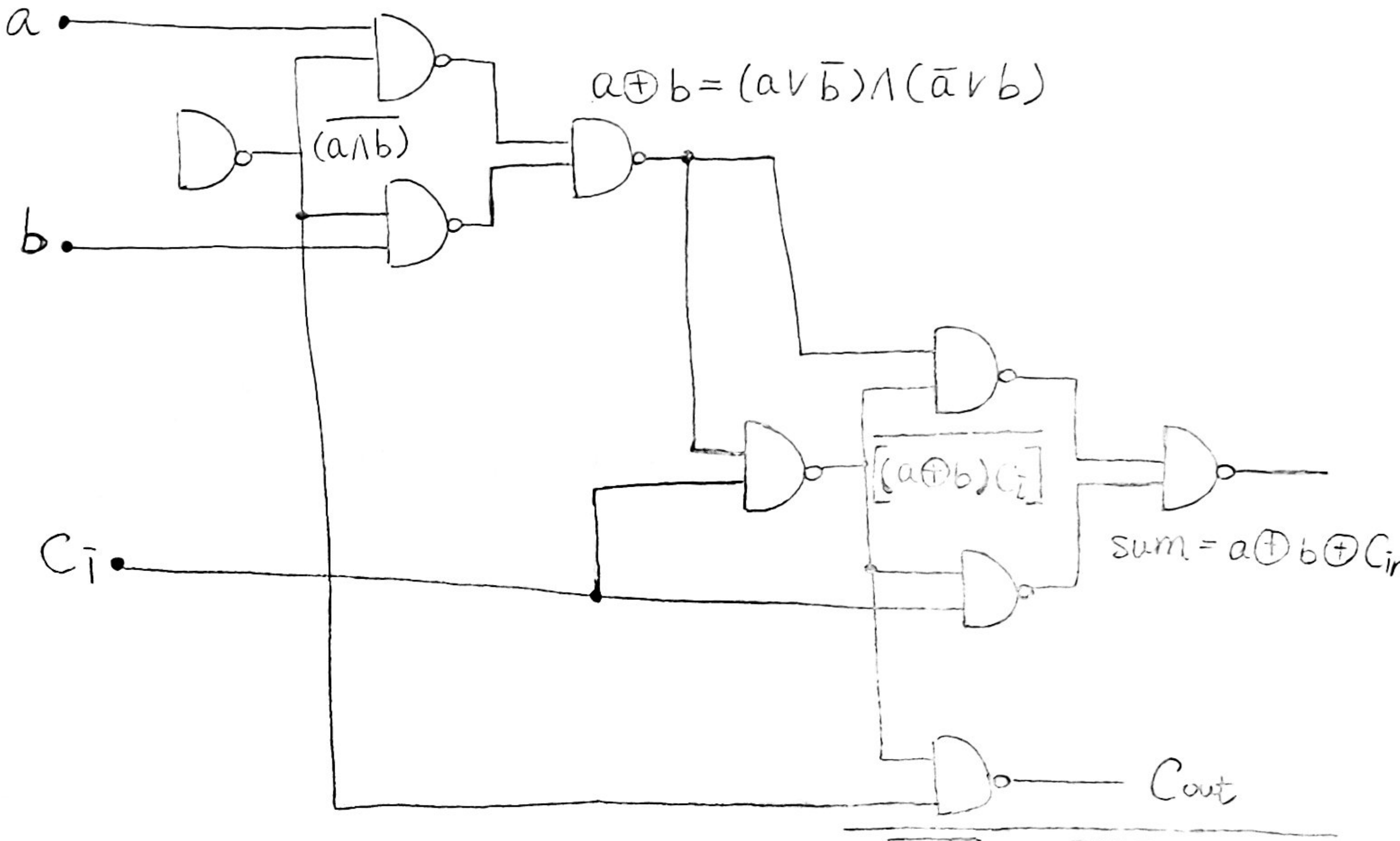
<u>Proof</u>: Full Adder using NAND Gates

$$Sum \doteq (a \oplus b) \oplus C_{in} = [(a \vee \bar{b}) \wedge (\bar{a} \vee b)] \oplus C_{in}$$

$$C_{out} = (a \wedge b) \vee (c \wedge (a \oplus b))$$

$$\boxed{C_{out} = (a \wedge b) \vee (c \wedge [(a \vee \bar{b}) \wedge (\bar{a} \vee b)])}$$

$$Sum = (\overline{[(a \vee \bar{b}) \wedge (\bar{a} \vee b)]} \vee C_{in}) \wedge (\overline{C_{in}} \vee [(a \vee \bar{b}) \wedge (\bar{a} \vee b)])$$



$$\overline{[\overline{(a \wedge b)} \wedge ((a \oplus b) \wedge C_i)]}$$

$$C_{out} = (a \wedge b) \vee [C_i \wedge (a \oplus b)]$$

③ Proof by induction to show there is a circuit of cn gates that compute $ADD_n = \{0,1\}^{2n} \to \{0,1\}^{n+1}$

Exercise 4.5 part 3

① When $n = 1$

$$ADD_1 : \{0,1\}^{2(1)} \to \{0,1\}^{1+1}$$

$$ADD_1 : \{0,1\}^{2} \to \{0,1\}^{2}$$

This resemble the HA function which we already show in part 1 that there is a NAND circuit of at most 5 NAND gates that compute HA

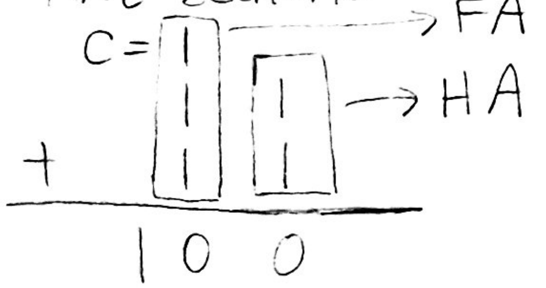
② When $n = 2$

$$ADD_2 : \{0,1\}^{2(2)} \to \{0,1\}^{2+1}$$

$$ADD_2 : \{0,1\}^{4} \to \{0,1\}^{3}$$

This is referring to ADD_2 outputs the addition of two input 2-bit numbers. For example, let (a b) represent the first 2-bit input & let (c d) represent the second 2-bit input. For the leftmost column, I use a HA to add two 1 bit number to get a 1 bit output & 1 carry bit. For the rightmost column, I use a FA to add two 1 bit number & a carry bit to get 2 bit output. So the final output is a 3 bit output.

First scenario



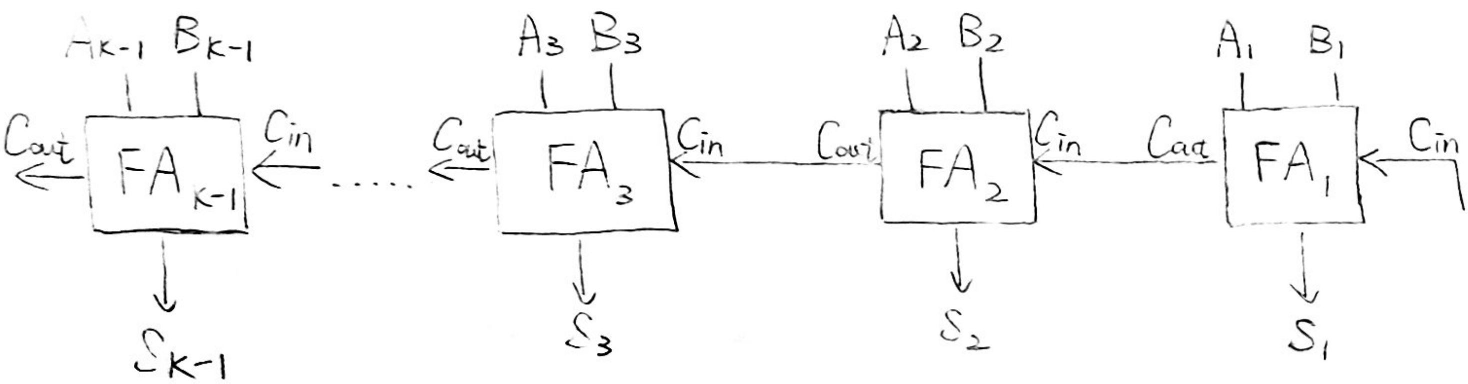
The number of needed NAND gates is $(1 \times 5) + (1 \times 9)$ $= 14$ NAND gates

(3) Assume when $n=k-1$ is true,

$ADD_n : \{0,1\}^{2(k-1)} \rightarrow \{0,1\}^k$

This is refers that ADD_{k-1} output the addition of two input (k-1)bit numbers. We can use the idea of ripple carry adder circuits. Multiple full adder circuits can be cascaded in parallel to add an (k-1)bit number. For an (k-1) bit parallel adder, there must be (k-1) number of full adder circuits. The ripple carry adder is a logic circuit in which the carry out of each full adder is the carry in of the succeeding next most significant full adder. It is because each carry bit gets ripped into the next stage.

For example, the first input $(A_1, A_2, \ldots, A_{K-1})$ & the second input $(B_1, B_2, \ldots, B_{K-1})$ $\}$ input to ADD_{K-1}



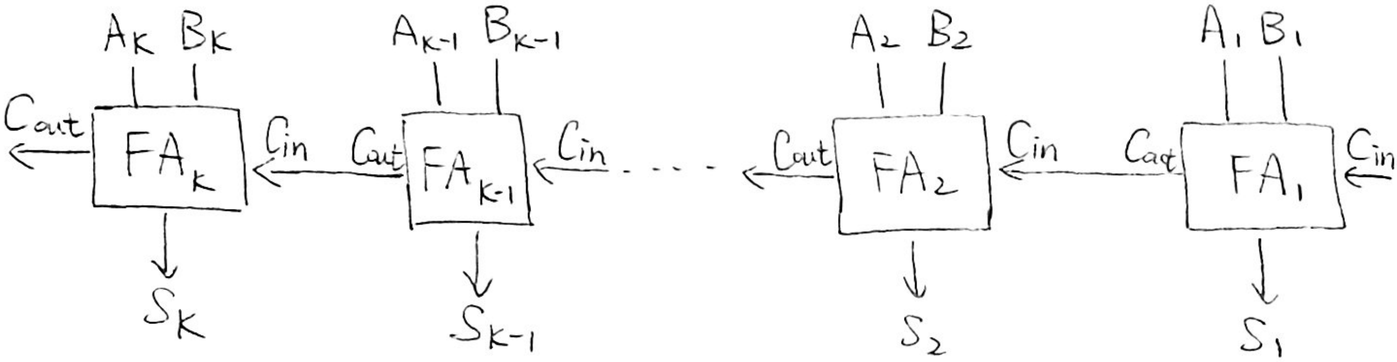
Assume that $n=k-1$ is true and there is a NAND circuit of c gates that compute each FA, then we need a total of $c(k-1)$ gates to compute ADD_n based on assumption and (k-1) number of used FA.

④ Prove when $n = k$,

$ADD_k : \{0,1\}^{2k} \rightarrow \{0,1\}^{k+1}$

This is refers that ADD_k output the addition of two input k bit numbers. According to the idea of ripple carry adder circuit, having an extra column of addition of 2 bits plus a carry bit resemble adding an extra full adder at the end. This means the input of FA_k are A_k, B_k, C_{out} (from FA_{k-1}).

For example, inputs to ADD_k: first input $= (A_1, A_2, \dots, A_k)$ &
second input $= (B_1, B_2, \dots, B_k)$



Since we know multiple full adder circuit can be cascade in parallel to add two k bit number together, therefore just need to add one more FA to the $n = k-1$ case. As a result, the needed number of gates to compute ADD_k is $c(k-1) + c = ck - c + c$
$= ck$

It is because each FA requires a NAND circuit of c gates & $n = k$, so there is a circuit of cn gates that computes ADD_n ▪

Exercise 5.4 - Counting lower bound for multibit function.
Prove that there exist a number $\delta > 0$ st. for every n, m there exists a function $f: \{0,1\}^n \to \{0,1\}^m$ that requires at least $\delta m \cdot \frac{2^n}{n}$ NAND Gates to compute.

Proof: let define two classes of functions

① $Size_n(s)' = \{f: \{0,1\}^n \to \{0,1\}^m$ that have a NAND Circuit Program of size $\leq S\}$
This is the set of all functions that can be computed by circuit size $\leq S$.

② $ALL_n' = \{f: \{0,1\}^n \to \{0,1\}^m\}$
This is the set of all functions on n bits.

③ The statement we want to prove can be rephrase as :
$ALL_n' \not\subset Size_n'(\delta m \cdot \frac{2^n}{n})$. That is, $\forall n$, $\exists$ a function $f: \{0,1\}^n \to \{0,1\}^m$ that does not have a NAND circuit program of size $< \delta m \frac{2^n}{n}$ for some $\delta > 0$.

④ Next, we need to find the number of funtions from $\{0,1\}^n$ to $\{0,1\}^m$, it is $2^{m(2^n)}$. It is because there are 2^n inputs & each input has 2^m options.
Therefore, the number of functions in $ALL_n' = 2^{m(2^n)}$

(5) Then, we need to find the number of functions in $Size_n' = \{f : \{0,1\}^n \to \{0,1\}^m$ that can be computed by NAND Circuit program of size $\leq s\}$.

$|Size_n'| \leq 2^{O(s \cdot \log s)}$ because according to theoreum 5.1 & 5.2, every circuit of size $= s$ can be represented as a binary string of length $\leq O(s \cdot \log s)$ and for every $s \in \mathbb{N}$, there are at most $2^{O(s \cdot \log s)}$ functions computed by NAND-Circ program of at most s lines.

(6) After that, $\exists$ a constant B such that

$$(\text{the number circuits of size} \leq s) \leq \sum_{\ell=1}^{s} 2^{B\ell(\log \ell)} \leq 2^{2B(s \cdot \log s)}$$

(7) Proof by contradiction

If $|ALL_n'| < |Size_n'(s)|$

$$\Rightarrow \quad 2^{m(2^n)} < 2^{2B(s \cdot \log s)}$$

Aside: δm is some constant > 0

$$\Rightarrow \quad m(2^n) < 2B(s \cdot \log s)$$

$$\Rightarrow \quad \frac{(2^n)m}{2B} < s \cdot \log s$$

(8) Plug $s = \delta m \frac{2^n}{n}$ into $s \cdot \log s$ to show contradiction

$$s \cdot \log s = \delta m \left(\frac{2^n}{n}\right) \log\left(\delta m \cdot \frac{2^n}{n}\right)$$

$$\log\left(\frac{2^n}{n}\right) \leq n$$

plug into

(9) $s \cdot \log s = \delta m \left(\frac{2^n}{\cancel{n}}\right)(\cancel{n}) = \delta m (2^n)$

$$\frac{2^n(\cancel{m})}{2B} < \delta \cancel{m} (2^n)$$

$$\frac{2^n}{2B} < \delta (2^n)$$

$$\frac{2^n}{2} < B\delta(2^n)$$

That is if $ALL_n' \subset Size_n'\left(\delta \cdot \frac{2^n}{n}\right)$

then $\delta > \frac{1}{2B} \iff \frac{1}{2B} \leq \delta$

To get a contradiction, we can plug in $\delta = \frac{1}{2B}$ in the statement. Since contradiction, so there exists a function $f: \{0,1\}^n \to \{0,1\}^m$ that requires at least

$\delta m \cdot \frac{2^n}{n}$ NAND gates to compute ∎

Exercise 3

(a) what is the start state?

The start state is the circle with number zero inside.

$\rightarrow$ 0 ; S_0

(b) What is the set of accept state.

state 3 (circle with 3 inside), 3 ; S_3

(c) what sequence of states does the machine goes through on input 010111

start state = S_0

input 0 = S_1

1 = S_3

0 = S_2

1 = S_2

1 = S_2

1 = S_2

$$S_0 \xrightarrow{0} S_1 \xrightarrow{1} S_3 \xrightarrow{0} S_2 \xrightarrow{1} S_2 \xrightarrow{1} S_2 \xrightarrow{1} S_2$$

(d) Does the machine accept the string 010111?

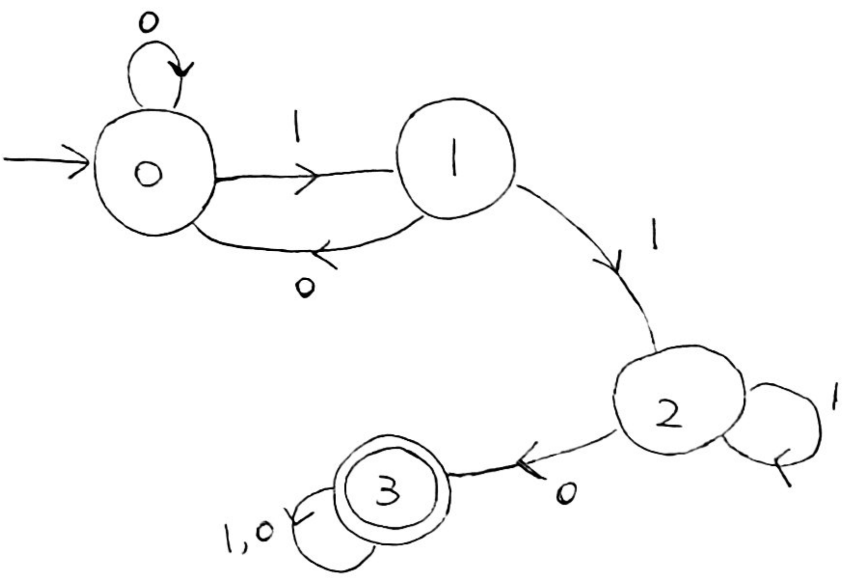
No, it does not accept the string 010111 because final state is S_2 which is not an accepting state. The accepting state is only S_3.

(e) Write down an accepting input for the machine

the string $\boxed{11}$ or $\boxed{01}$

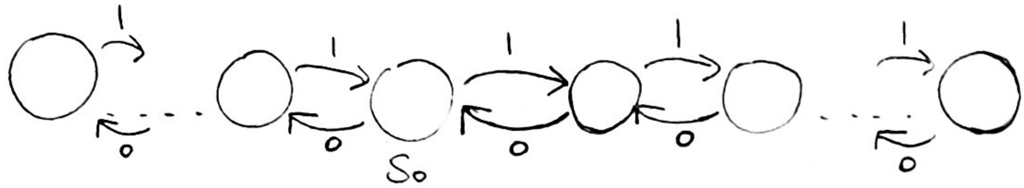
Problem 4

DFA that accepts strings which contain 110 as substring



Problem 5

Design an DFA that accepts strings that have more 1's than 0's.



We cannot design a DFA that accept string that have more 1's than 0's because the input string could be infinite in a sense that it does not use constant memory algorithm to transition to a new state. If an algorithm does not use c bits of memory, then the contents of its memory cannot be represented as a string of length c. Therefore, it requires more than 2^c states at any point in the execution.

The difficulty are I have to keep track of the number of visited 1's & 0's infinitely if the input string is really long, but the working memory is constant.

It is impossible to keep track of infinite many such states.