S.E. Theory: SOFTENG211 Assignment #4

Nisarag Bhatt

Problem 1

Question:

Design an algorithm that given an input string from the alphabet consisting of symbols from PROP, logical connectives $\vee$, $\wedge$, $\rightarrow$, $\neg$ and the left parenthesis (, and the right parenthesis), decides if the string is a formula. Your algorithm should consist of several lines of instructions written in English.

Answer:

First we define our alphabet to be $\Sigma = \{p_1, ..., p_n, \wedge, \vee, \neg, \rightarrow,), (\}$

Then let s be a string over this alphabet.

We also define these formulas below as ***proper***, where $p, q \in PROP$

- $\neg(p)$
- $(p \vee q)$
- $(p \wedge q)$
- $(p \rightarrow q)$

The below algorithm will find is s is a formula.

Note that in line 9, our string s is reduced.

Also since we are replacing our proper formula ψ with a proposition γ we know that s' is formula if and only if s is a formula.

Algorithm 1 Deciding if a string is a formula

```
 1: procedure isFormula($s$)
 2:     if $s \in PROP$ then
 3:         return $True$
 4:     else
 5:         $\psi \leftarrow$ The first occurrence of a proper formula in $s$.
 6:         if such a $\psi$ does not exist then
 7:             return $False$
 8:         else
 9:             $s' \leftarrow s$ but replace $\psi$ with a new proposition $\gamma$.
10:             return isFormula($s'$)
11:         end if
12:     end if
13: end procedure
```

Problem 2

Question:

Provide a linear time algorithm that given a formula in DNF decides if the formula is satisfiable. If you do not know linear time algorithm (that is, did not take SOFTENG 250 class, then provide an algorithm that checks satisfiability of DNF without going through of all truth assignments of the formula).

Answer:

Before we begin our algorithm we note that our algorithm takes in a string f which is in the form

$$f = a_1 \vee a_2 \vee ... \vee a_n$$

for some $n > 0$.

Where $b_i \in PROP$ and b_i is in the form $(b_1 \wedge b_2 \wedge ... \wedge b_m)$ for some $m > 0$.

Algorithm 2 Checking if a formula in DNF is satisfiable

```
1: procedure DNFSatisfiable($f$)    ▷ Size: $f = n$, Note that $f$ is a string containing $n$ sub formulas
2:     for $a_1$ to $a_n$ subformulas in $f$ do    ▷ $\mathcal{O}(n)$
3:         if $a_i$ does not contain a pair of complementary pair of literals $(p \wedge \neg p)$ where $p \in b_i$ then
4:             return $True$    ▷ If one subformula is true then the whole formula is satisfiable
5:         end if
6:     end for
7:
8:     return $False$    ▷ If all subformulas have a complementary pair of literals
9: end procedure
```

Note that this algorithm is linear because at most you have to check n subformulas hence the algorithm runs in $\mathcal{O}(n)$ time.

You only return false when all formulas $(a_1, ..., a_n)$ evaluate are false.

Problem 3

Question:

Consider the following formula:

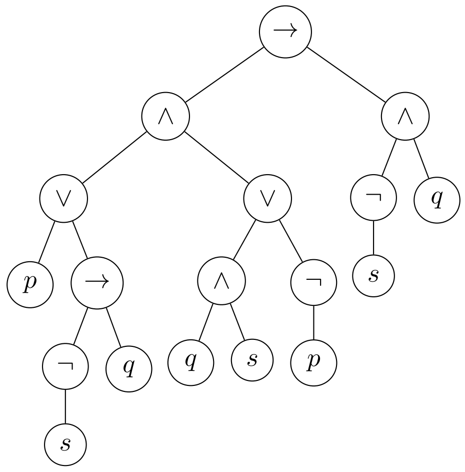
$$(((p \vee (\neg s \rightarrow q)) \wedge ((q \wedge s) \vee \neg p)) \rightarrow (\neg s \wedge q))$$

Do the following:

(a) Write down the labeled tree representation of the formula.
(b) Write down the truth table for the formula.
(c) Find a DNF formula equivalent to it.

Answer:

(a)



(b) First we shall simplify the formula: $(((p \vee (\neg s \rightarrow q)) \wedge ((q \wedge s) \vee \neg p)) \rightarrow (\neg s \wedge q))$

Note that: $(p \vee (\neg s \rightarrow q)) = (p \vee s \vee q)$

Hence

$$\begin{aligned} f &= (((p \vee (\neg s \rightarrow q)) \wedge ((q \wedge s) \vee \neg p)) \rightarrow (\neg s \wedge q)) \\ &= (((p \vee s \vee q) \wedge ((q \wedge s) \vee \neg p)) \rightarrow (\neg s \wedge q)) \\ &= (((s \wedge q) \vee (s \wedge \neg p) \vee (\neg p \wedge q)) \rightarrow (\neg s \wedge q)) \\ &= \neg((s \wedge q) \vee (s \wedge \neg p) \vee (\neg p \wedge q)) \vee (\neg s \wedge q) \\ &= ((\neg s \wedge p) \vee (\neg s \wedge \neg q) \vee (p \wedge \neg q) \vee (\neg s \wedge q)) \\ &= \neg s \vee (p \wedge \neg q) \end{aligned}$$

Problem 3 continued on next page. . .

p	q	s	$\neg s \vee (p \wedge \neg q)$
T	T	T	F
T	T	F	T
T	F	T	T
T	F	F	T
F	T	T	F
F	T	F	T
F	F	T	F
F	F	F	T

(c) Reading from the truth table, we get that the disjunctive normal form of the formula is:

$$DNF = ((p \wedge q \wedge \neg s) \vee (p \wedge \neg q \wedge s) \vee (p \wedge \neg q \wedge \neg s) \vee (\neg p \wedge q \wedge \neg s) \vee (\neg p \wedge \neg q \wedge \neg s))$$

Problem 4

Question:

Write down all languages over two letter alphabet $\{a, b\}$ recognised by two state DFA.

Answer:

I have expressed the languages in regular expressions since it is much easier to write.

All answers assume $w \in \{a, b\}^*$ and note that $\lambda =$ the empty string

The reasoning behind how there are 26 languages is as follows:

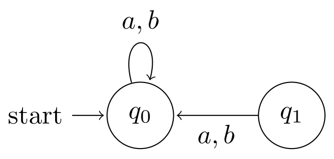
We start with a two state DFA which contains two states $Q = \{q_0, q_1\}$. This DFA is over the binary alphabet $\Sigma = \{a, b\}$.

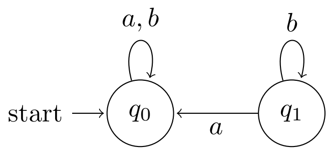
Note that the transition function is as follows :

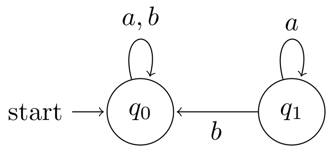
$$\delta : Q \times \Sigma \to Q$$
$$\begin{aligned} \delta(q_0, a) &= \ q \in Q \\ \delta(q_0, b) &= \ q \in Q \\ \delta(q_1, a) &= \ q \in Q \\ \delta(q_1, b) &= \ q \in Q \end{aligned}$$

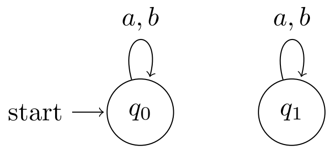
Since there are 2 options for $q \in Q$ and there are 4 places to place these options there are $2^4 = 16$ ways to arrange q_1, q_0 between the 4 transitions. This means there are 16 possible transition functions and hence 16 DFA's possible for a 2 state DFA over a binary alphabet.

However, these specific transitions are disconnected:









These 4 transition functions were function in which the initial state could not reach the second state, so effectively there was only one possible state so it they are trivial to analyse:

Problem 4 continued on next page. . .

Hence there are only 12 transitions to consider. Now, since there are two states : q_0, q_1 there are $2^2 = 4$ ways to pick which ones are accepting and which are non accepting. Namely these are:

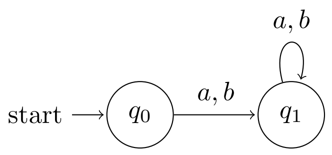
Case	q_0	q_1
1	Accepting	Accepting
2	Accepting	Not Accepting
3	Not Accepting	Accepting
4	Not Accepting	Not Accepting

For each possible case there are 16 different possible transitions. Which means for each case there are 16 languages however one may observe that:

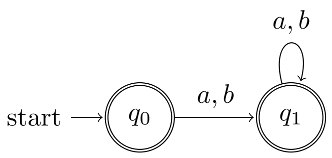
- For case 1, both q_0 and q_1 are accepting which means there is only one possible language since it doesn't matter what transitions exist in the transition function because any of them will give the same language. Hence we can reduce the number of languages for this case from $16 \to 1$.
- For case 2, we need to analyse 12 transition functions. Hence the number of languages for this case is 12.
- For case 3, we need to analyse 12 transition functions. Hence the number of languages for this case is 12.
- For case 4, both q_0 and q_1 are not accepting which means there is effectively no language for this DFA so any transition function will result in the empty set. Hence we can reduce the number of languages for this case from $16 \to 1$.

Therefore the total number of languages must be $12 + 12 + 1 + 1 = 26$.

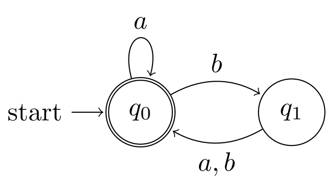
Below, I have shown all the DFA's corresponding to the 26 possible languages and expressed the languages in a regular expression form.



Language is $L_0 = \phi = \{\}$ (Doesn't matter what our transition is since there is no accepting state)

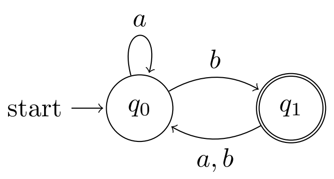


Language is $L_1 = \{w|w \text{ is in the form } (a+b)^{\star}\}$ (Any transitions resulting in this regular expression)

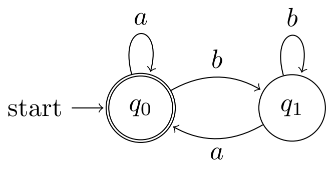


Language is $L_2 = \{w|w \text{ is in the form } (a + b(a+b))^{\star}\}$

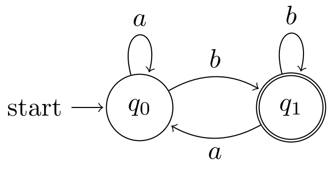
Problem 4 continued on next page...



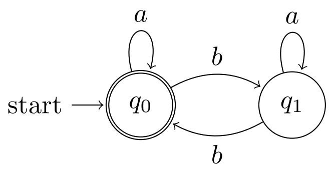
Language is $L_3 = \{w|w \text{ is in the form } (a + b(a + b))^\star b\}$



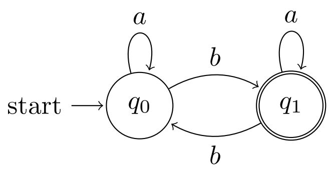
Language is $L_4 = \{w|w \text{ is in the form } (a + bb^\star a)^\star\}$



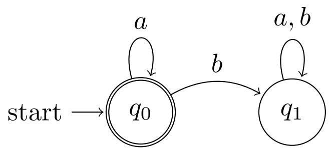
Language is $L_5 = \{w|w \text{ is in the form } (a + bb^\star a)^\star bb^\star\}$



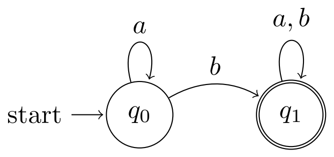
Language is $L_6 = \{w|w \text{ is in the form } (a + ba^\star b)^\star\}$



Language is $L_7 = \{w|w \text{ is in the form } (a + ba^\star b)^\star ba^\star\}$

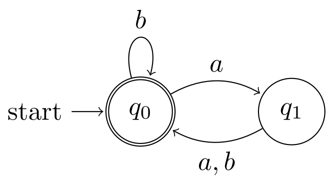


Language is $L_8 = \{w|w \text{ is in the form } a^\star\}$

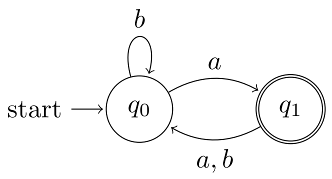


Language is $L_9 = \{w|w \text{ is in the form } a^\star b(a + b)^\star\}$

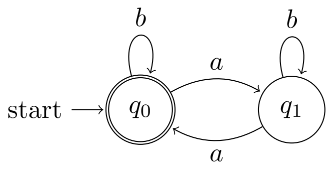
Problem 4 continued on next page. . .



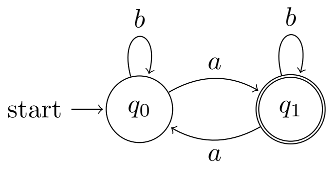
Language is $L_{10} = \{w | w$ is in the form $(b + a(a + b))^\star\}$



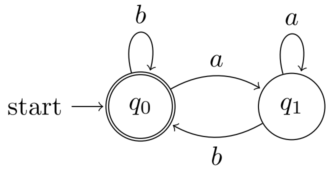
Language is $L_{11} = \{w | w$ is in the form $(b + a(a + b))^\star a\}$



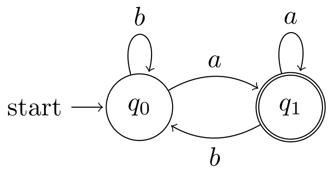
Language is $L_{12} = \{w | w$ is in the form $(b + ab^\star a)^\star\}$



Language is $L_{13} = \{w | w$ is in the form $(b + ab^\star a)^\star ab^\star\}$

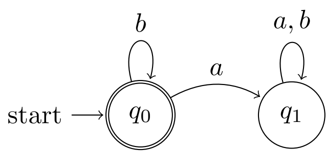


Language is $L_{14} = \{w | w$ is in the form $(b + aa^\star b)^\star\}$

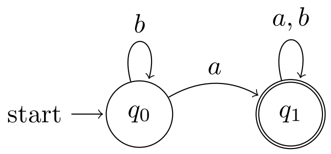


Language is $L_{15} = \{w | w$ is in the form $(b + aa^\star b)^\star aa^\star\}$

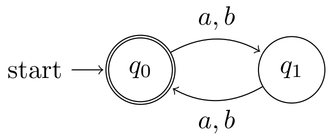
Problem 4 continued on next page. . .



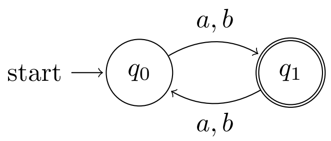
Language is $L_{16} = \{w|w \text{ is in the form } b^{\star}\}$



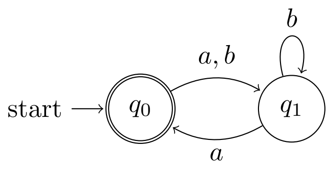
Language is $L_{17} = \{w|w \text{ is in the form } b^{\star}a(b+a)^{\star}\}$



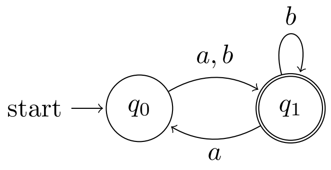
Language is $L_{18} = \{w|w \text{ is in the form } ((b+a)(b+a))^{\star}\}$



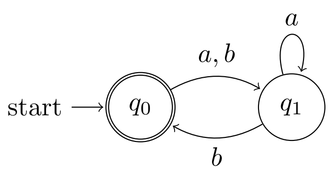
Language is $L_{19} = \{w|w \text{ is in the form } ((b+a)(b+a))^{\star}(b+a)\}$



Language is $L_{20} = \{w|w \text{ is in the form } ((a+b)b^{\star}a)^{\star}\}$

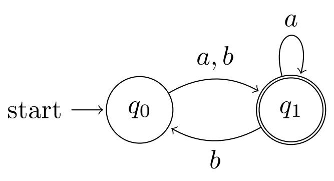


Language is $L_{21} = \{w|w \text{ is in the form } ((a+b)b^{\star}a)^{\star}(a+b)b^{\star}(b+\lambda)\}$

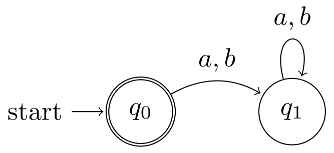


Language is $L_{22} = \{w|w \text{ is in the form } ((a+b)a^{\star}b)^{\star}\}$

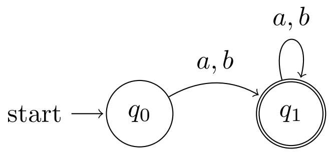
Problem 4 continued on next page. . .



Language is $L_{23} = \{w|w \text{ is in the form } ((a+b)a^\star b)^\star(a+b)a^\star(a+\lambda)\}$



Language is $L_{24} = \{w|w \text{ is in the form } \lambda\}$



Language is $L_{25} = \{w|w \text{ is in the form } (a+b)(b+a)^\star(b+a+\lambda)\}$

Problem 5

Question:

Let L be a language over the alphabet $\{a, b, c\}$ recognised by NFA. Consider the language L' such that every string $w' \in L'$ is obtained from a string $w \in L$ by replacing all occurrences of c in w by a. Prove that L' is NFA recognizable.

Answer:

Proof. Consider a NFA: $\mathcal{M}$ which recognizes L. To complete this proof we must prove that there exists an NFA: $\mathcal{M}'$ in which $\mathcal{M}'$ recognizes L'.

First, we construct $\mathcal{M}'$ by taking $\mathcal{M}$ and considering all the transitions of $\mathcal{M}$. For the transitions that are labelled c, replace c with a. By doing this we produce a language $L(\mathcal{M}')$.

To complete this proof, it suffices to show that $L(\mathcal{M}') = L'$

Proof. Prove that: $L' \subseteq L(\mathcal{M}')$

Consider a string $w' \in L'$. By the definition of how L' was constructed there must exist a $w \in L$ in which w' is constructed by replacing all occurrences of c in w by a.

From this we must know that there exists an accepting *run* in $\mathcal{M}$ for the string w. Since we constructed $\mathcal{M}$ by replacing all transitions that are labelled c with a, this means the accepting run is preserved since anytime $\mathcal{M}'$ processes a c, $\mathcal{M}'$ will behave the same way as $\mathcal{M}$. (I.e. there is an accepting run of $\mathcal{M}'$ on w')

This must mean that $w' \in L(\mathcal{M}')$ and because w' was picked arbitrarily, this argument works for any string ($w' \in L'$). This means that $L' \subseteq L(\mathcal{M}')$. □

Proof. Prove that: $L(\mathcal{M}') \subseteq L'$

We once again consider a string $w' \in L(\mathcal{M}')$. Consider an accepting run for w' by $\mathcal{M}'$, call this run ζ.

We will now take w', we then construct a new string w by taking w' and replacing an occurrence of a with c when ζ goes through a transition which would have been labelled by c in $\mathcal{M}$.

Now $\mathcal{M}$ will have an accepting run for w and because of this $w \in L$. Since we can make w' again by taking w and replacing all occurrences of c with a.

This must mean that $w' \in L'$. Hence, $L(\mathcal{M}') \subseteq L'$.

□

Since $L' \subseteq L(\mathcal{M}')$ and $L(\mathcal{M}') \subseteq L'$, it must mean that $L(\mathcal{M}') = L'$. This means that L' is NFA recognizable since it is a language recognized by the NFA $\mathcal{M}'$.

□

Problem 6

Question:

Over the alphabet $\{a, b\}$, consider the language $L = \{w|$ the string w starts with an $a\}$. Prove that no DFA with two states recognises L.

Answer:

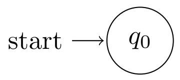
We shall prove this by contradiction:

Proof. Assume that there does exists a two state DFA that recognises $L = \{w|$ the string w starts with an $a\}$ over the alphabet $\Sigma = \{a, b\}$.

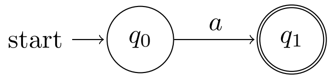
We first note that a two state $S = (q_0, q_1)$ DFA over a binary alphabet must contain 4 transitions because $S \times \Sigma = \{(q_0, a), (q_0, b), (q_1, a), (q_1, b)\}$

Now we begin by construction

We start of with one state:

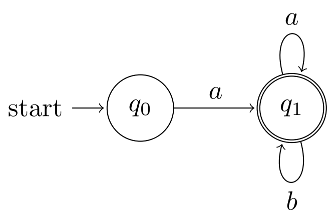


If a string w begins with an a then when q_0 receives an a from a string it *must* go into an accepting state (q_1) so we have a transition $(q_0, a) = q_1$:



When in the accepting state (q_1), if it receives either a or b it has to stay *accepted* because you can't go back to a not accepting state (q_0) since w already started with a so it must be accepting.

So we must have two more transitions $(q_1, a) = q_1$ and $(q_1, b) = q_1$.



We have now used 3 of our transitions: since we used $(q_0, a) = q_1, (q_1, a) = q_1, (q_1, b) = q_1$. The remaining transition is (q_0, b).

The only states the transition (q_0, b) can go to is either $(q_0, b) = q_0$ or $(q_0, b) = q_1$

If $(q_0, b) = q_0$ then that means if a string that contains a b stays in the same state (q_0). However this would mean a string such as "ba" is accepted because when you begin the DFA you start at q_0 then you process a b : $(q_0, b) = q_0$, after you process an a : $(q_0, a) = q_1$ you end up in q_1 which means the string "ba" is accepted. However this string does *not* start with a so we can't have the transition: $(q_0, b) = q_0$.

If $(q_0, b) = q_1$ then that means a string such as "b" is accepted however this can't be the case because only strings that begin with a are accepted.

Since we can't have either $(q_0, b) = q_0$ and $(q_0, b) = q_1$ we must require atleast another state for (q_0, b) to transition to. Therefore contradiction. Hence, we cannot construct a DFA with two states that recognises L. □

Problem 6 continued on next page. . .

Problem 7

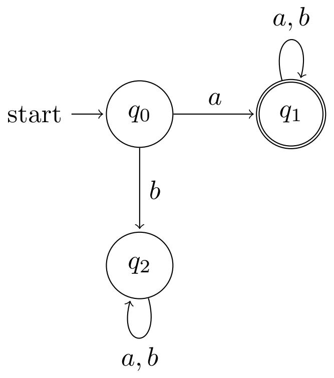
Question:

Over the alphabet $\{a, b\}$, consider the languages $L_1 = \{w|\text{the string } w \text{ starts with an } a\}$ and $L_2 = \{w|\text{the number of symbols } b \text{ in } w \text{ is 1 modulo 3}\}$. Do the following:

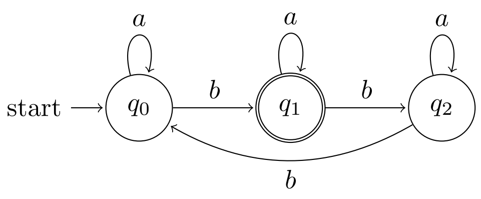
- Draw diagrams for DFA recognising both L_1 and L_2.
- Draw a diagram for DFA recognising $L_1 \cap L_2$.

Answer:

$$L_1 = \{w|\text{the string } w \text{ starts with an } a\}$$

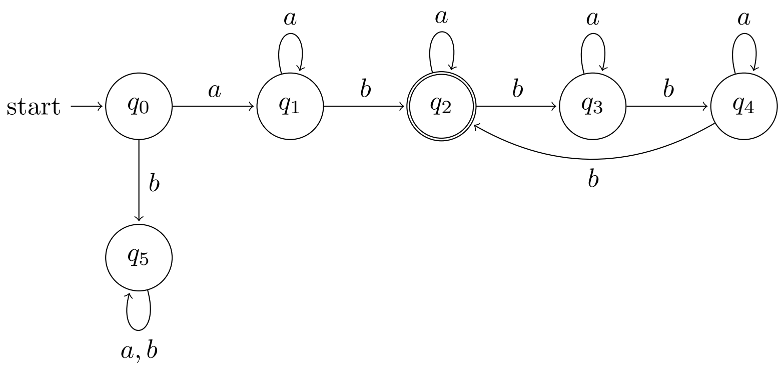


$$L_2 = \{w \mid \text{the number of symbols } b \text{ in } w \text{ is 1 modulo 3}\}$$



- Draw a diagram for DFA recognising $L_1 \cap L_2$.

$$DFA(L_1 \cap L_2)$$


start
q_0
q_1
q_2
q_3
q_4
q_5
a
a
a
a
a
b
b
b
b
b
a, b