QUESTION 1 [12 marks] (Natural Deduction)

The following questions ask for proofs using natural deduction. Present your proofs in the Fitch style as used in lectures. You may use the introduction and elimination rules given in Appendix 1, as well as the following derived rule:

$$\text{(contradiction)}\ \frac{p \wedge \neg p}{q}$$

Number each line and include justifications for each step in your proofs.

(a) Give a natural deduction proof of $\dfrac{(p \wedge q) \Rightarrow r}{\neg(p \wedge (q \wedge \neg r))}$

QUESTION 1(a) **[4 marks]**

(b) Give a natural deduction proof of $\dfrac{c \Rightarrow (a \vee b)}{(c \wedge \neg a) \Rightarrow b}$

QUESTION 1(b) **[4 marks]**

(c) Give a natural deduction proof of $\dfrac{P \wedge \exists x.Q(x)}{\exists x.(P \wedge Q(x))}$ where x does not appear free in P.

Take care with parentheses and variable names.

QUESTION 1(c) **[4 marks]**

QUESTION 2 [12 marks] (Structural Induction)

In all proofs indicate the justification (eg, the line of a definition used) for each step.

(a) Given the following definitions of `(++)` and `count`:

```
[]     ++ ys  = ys                -- (A1)
(x:xs) ++ ys  = x : (xs ++ ys)    -- (A2)

count []      = 0                 -- (C1)
count (x:xs)  = 1 + count xs      -- (C2)
```

Prove the following

```
count(xs ++ ys) = (count xs) + (count ys)
```

(i) State and prove the base case goal

QUESTION 2(a)(i) **[1 mark]**

(ii) State the inductive hypothesis

QUESTION 2(a)(ii) **[1 mark]**

(iii) State and prove the step case goal:

QUESTION 2(a)(iii) **[3 marks]**

(b) Binary trees can be represented in Haskell with the following algebraic data type:

```
data Tree a = Nul | Node a (Tree a) (Tree a)
```

We can flatten the tree to a list using a pre-order or in-order traversal, given by functions `pre` and `in` below:

```
pre  Nul             = []                              -- (PR1)
pre (Node a t1 t2)   = (a : (pre t1)) ++ (pre t2)      -- (PR2)

in  Nul              = []                              -- (IN1)
in (Node a t1 t2)    = (in t1) ++ (a : (in t2))        -- (IN2)
```

Prove the following property:

```
count(pre t) = count(in t)
```

You may use the result you proved in Part (a) of this question:

```
count(xs ++ ys) = (count xs) + (count ys)          -- (LEMMA1)
```

(i) State and prove the base case goal.

QUESTION 2(b)(i) **[1 mark]**

(ii) State the inductive hypotheses.

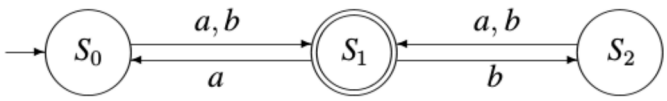
QUESTION 2(b)(ii) **[1 mark]**

(iii) State and prove the step case goal.

QUESTION 2(b)(iii) **[5 marks]**

QUESTION 5 [12 marks] (Finite State Automata)

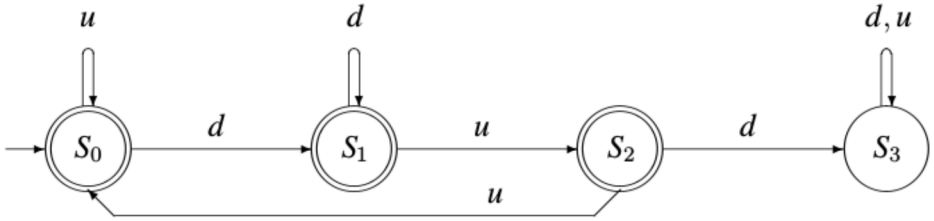
(a) Consider the deterministic finite state automaton A:



Define a new deterministic finite state automaton that accepts the same language as A but is ***minimal***.

QUESTION 5(a) **[2 marks]**

(b) Consider the deterministic finite state automaton B:



Give a fully rigorous proof that any string of the form $\alpha\, d\, u\, d\, \beta$, where α, β are arbitrary strings in $\{d,\ u\}^*$, is ***not*** accepted by B.

You may use without proof the ***lemma***

$$N^*(S_3,\ \beta)\ =\ S_3 \ \text{ for all } \ \beta \in \{d,\ u\}^*$$

as long as you are explicit about when you are using it.

QUESTION 1 [12 marks]

(Structural Induction)

In all proofs indicate the justification (eg, the line of a definition used) for each step.

(a) The function `revA` provides an efficient way to reverse a list: `revA xs ys` reverses the list `xs` and appends the result to (the front of) the list `ys`.

If we wish to modify members of these lists with `map f` and also to apply `revA` to the two lists, show that these operations can be done in either order. That is, prove

```
map f (revA xs ys) = revA (map f xs) (map f ys)
```

The definitions of `map` and `revA`:

```
map f []     = []                                  -- M1
map f (x:xs) = f x : map f xs                      -- M2

revA [] ys     = ys                                -- RA1
revA (x:xs) ys = revA xs (x : ys)                  -- RA2
```

(i) State and prove the base case goal

QUESTION 1(a)(i) **[2 marks]**

In the remaining parts of your solution, indicate the use of $\forall$ quantifiers in the property to be proved and in the inductive hypotheses, and indicate how these are instantiated in proving the step case goal.

(ii) State the inductive hypothesis

QUESTION 1(a)(ii) **[1 mark]**

(iii) State and prove the step case goal:

QUESTION 1(a)(iii) **[4 marks]**

(b) Binary trees can be represented in Haskell with the following algebraic data type:

```
data Tree a = Nul | Node a (Tree a) (Tree a)
```

We can get the sum of entries in a tree by the function `sumT`

```
sumT :: Tree Int -> Int
sumT  Nul            = 0                          -- (ST1)
sumT (Node n t1 t2) = n + sumT t1 + sumT t2       -- (ST2)
```

We can reverse a tree (get its mirror image) by the function `revT`

```
revT :: Tree a -> Tree a
revT  Nul            = Nul                              -- (RT1)
revT (Node n t1 t2) = Node n (revT t2) (revT t1)        -- (RT2)
```

This question is about proving, by structural induction on the structure of the tree argument, that for all trees `t :: Tree Int`,

```
sumT (revT t) = sumT t
```

(i) State and prove the base case goal.

QUESTION 1(b)(i) **[1 mark]**

(ii) State the inductive hypotheses.

QUESTION 1(b)(ii) **[1 mark]**

(iii) State and prove the step case goal.

QUESTION 1(b)(iii) **[3 marks]**

QUESTION 5 [13 marks]

(Finite State Automata)

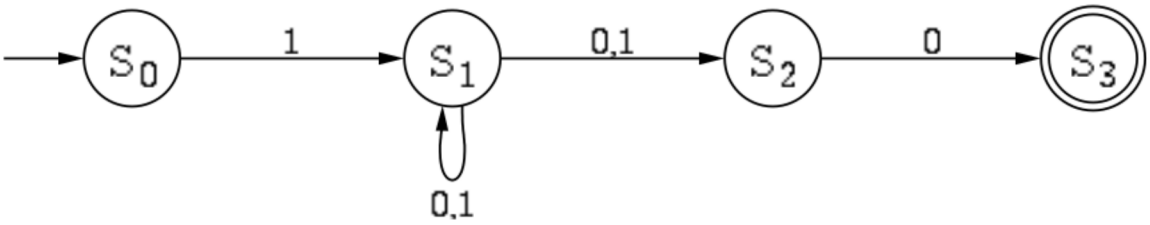
(a) Design a finite state automaton to accept bit strings which both start and finish with 1. That is, your FSA should accept strings $\alpha \in \{0,1\}^*$ where the first digit and the last digit are both 1. (Hint: note that the single digit string 1 is included in this description, and should be accepted by your automaton).

QUESTION 5(a) **[2 marks]**

(b) Give a regular expression for this language.

QUESTION 5(b) **[1 mark]**

(c) The following FSA accepts a language L consisting of certain bit strings, ie, $L \subseteq \{0,1\}^*$.



Describe (in English) the language L which is accepted by the automaton.

QUESTION 5(c) **[2 marks]**

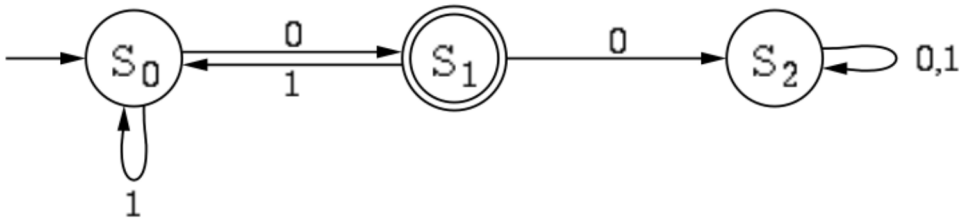
(d) Give a regular grammar (which may be right-linear *or* left-linear) which generates the language L of part (c).

QUESTION 5(d) **[2 marks]**

(e) Design a *deterministic* FSA which recognises exactly the same language L of part (c).

QUESTION 5(e) **[3 marks]**

(f) Consider the following DFA



Show that the automaton does not accept strings ending in 00, that is, for any $\alpha \in \Sigma^*$, $N^*(S_0, \alpha 00)$ is not a final state.

QUESTION 5(f) **[3 marks]**

QUESTION 6 [11 marks]

(Natural Deduction)

Some of the following questions ask for proofs using natural deduction. Present your proofs in the Fitch style as used in lectures. Use only the introduction and elimination rules given in Appendix 1. Number each line and include justifications for each step in your proofs.

(a) Use truth tables to determine whether the following two propositions are (always) equivalent, or not; if not, give an example of values of p, q, r for which they are not equivalent.

$$p \vee (q \rightarrow r) \qquad (p \vee q) \rightarrow (p \vee r)$$

QUESTION 6(a) **[2 marks]**

p	q	r	$q \rightarrow r$	$p \vee (q \rightarrow r)$	$p \vee q$	$p \vee r$	$(p \vee q) \rightarrow (p \vee r)$
T	T	T					
T	T	F					
T	F	T					
T	F	F					
F	T	T					
F	T	F					
F	F	T					
F	F	F					

(b) Give a natural deduction proof of $\dfrac{(p \rightarrow r) \wedge (q \rightarrow r)}{(p \vee q) \rightarrow r}$

QUESTION 6(b) **[3 marks]**

(c) Give a natural deduction proof of $\dfrac{\neg (p \wedge \neg q)}{(p \rightarrow q)}$

QUESTION 6(c) **[3 marks]**

(d) Give a natural deduction proof of $\dfrac{\exists x.\, Q \to P(x)}{Q \to (\exists x.\, P(x))}$ (x does not appear free in Q).

Take care with parentheses and variable names.

QUESTION 6(d) **[3 marks]**

QUESTION 1 [12 marks] (Natural Deduction)

The following questions ask for proofs using natural deduction. Present your proofs in the Fitch style as used in lectures. Use only the introduction and elimination rules given in Appendix 1. Number each line and include justifications for each step in your proofs.

(a) Give a natural deduction proof of $\dfrac{p \to p \wedge q}{p \vee q \to q}$, that is, $\dfrac{p \to (p \wedge q)}{(p \vee q) \to q}$.

QUESTION 1(a) **[4 marks]**

(b) What follows is a proof of $(p \vee q) \rightarrow (\neg p \rightarrow q)$ with the justifications, and the layout showing assumptions, missing.

$p \vee q$

$\neg p$

p

$\neg q$

$p \wedge \neg p$

q

q

q

q

$\neg p \rightarrow q$

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

Reproduce the proof below, adding justifications which show which rules are used on each statement. Use the Fitch proof style which shows the scope of each assumption. (Hint: one of the lines is simply a repeat of a previous line).

QUESTION 1(b) **[4 marks]**

(c) Give a natural deduction proof of $\dfrac{(\forall x.\, P(x)) \vee Q}{\forall x.\, P(x) \vee Q}$ that is $\dfrac{(\forall x.\, P(x)) \vee Q}{\forall x.\, (P(x) \vee Q)}$

Take care with parentheses and variable names.

QUESTION 1(c) **[4 marks]**

QUESTION 2 [13 marks] (Structural Induction)

(a) Give an inductive proof of the fact that mapping a function over a list, and then filtering the result, can be achieved by filtering the list first, and then mapping. That is:

```
filter p (map f xs)  =  map f (filter (p.f) xs)
```

The definitions of `map`, `filter` and compose `(.)` are:

```
map f []     = []                                                  -- M1
map f (x:xs) = f x : map f xs                                      -- M2

(p . f) x    =  p (f x)                                            -- C

filter p []     = []                                               -- F1
filter p (x:xs) = if p x then x : filter p xs else filter p xs     -- F2
```

(i) State and prove the base case goal

QUESTION 2(a)(i) **[2 marks]**

(ii) State the inductive hypothesis

QUESTION 2(a)(ii) **[1 mark]**

(iii) State and prove the step case goal: where the goal is of the form $P(x : xs)$ (based on inductive hypothesis $P(xs)$), you need deal only with the case where `p (f x)` is true; you may omit the case where `p (f x)` is false.

QUESTION 2(a)(iii) **[4 marks]**

(b) Binary trees can be represented in Haskell with the following algebraic data type:

```
data Tree a = Nul | Node a (Tree a) (Tree a)
```

We can turn a tree into a list containing the same entries with the tail recursive function `flat`, where `flat t acc` returns the result of flattening the tree `t`, appended to the front of the list `acc`. Thus, for example,

`flat (Node 5 (Node 3 Nul Nul) (Node 6 Nul Nul)) [1,2]` = `[3,5,6,1,2]`

```
flat :: Tree a -> [a] -> [a]
flat Nul acc = acc                                  -- (F1)
flat (Node a t1 t2) acc =
    flat t1 (a : flat t2 acc)                       -- (F2)
```

We can get the sum of entries in a list by the function `sumL`

```
sumL :: [Int] -> Int
sumL []      = 0                                    -- (S1)
sumL (x:xs)  = x + sumL xs                          -- (S2)
```

We can get the sum of entries in a tree by the function `sumT`

```
sumT :: Tree Int -> Int
sumT  Nul           = 0                             -- (T1)
sumT (Node n t1 t2) = n + sumT t1 + sumT t2         -- (T2)
```

This question is about proving, by structural induction on the structure of the tree argument, that for all `t` and `acc`,

```
sumL (flat t acc) = sumT t + sumL acc
```

(i) State and prove the base case goal.

QUESTION 2(b)(i)	**[1 mark]**

In the remaining parts of your solution, indicate the use of $\forall$ quantifiers in the property to be proved and in the inductive hypotheses, and indicate how these are instantiated in proving the step case goal.

(ii) State the inductive hypotheses.

QUESTION 2(b)(ii) **[1 mark]**

(iii) State and prove the step case goal.

QUESTION 2(b)(iii) **[4 marks]**

QUESTION 6 [12 marks] (Finite State Automata)

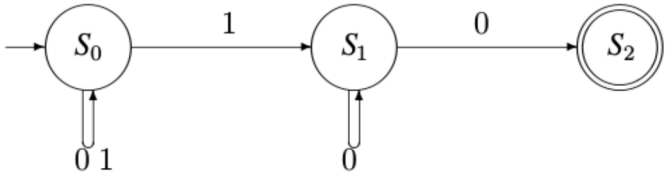
(a) Design a finite state automaton to accept bit strings which start and finish with the same binary digit. That is, your FSA should accept strings $\alpha \in \{0,1\}^*$ where the first digit equals the last digit. (Hint: note that the single digit strings 0 and 1 are included in this description, and should be accepted by your automaton).

QUESTION 6(a) **[3 marks]**

(b) Give a regular expression for this language.

QUESTION 6(b) **[2 marks]**

(c) The following FSA accepts a language L consisting of certain bit strings, ie, $L \subseteq \{0,1\}^*$.



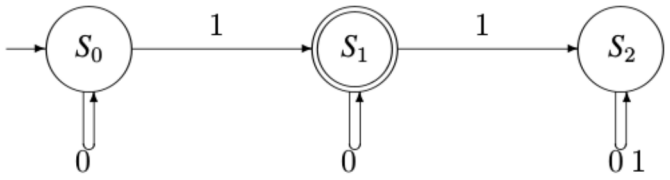
Describe the language L which is accepted by the automaton.

QUESTION 6(c) **[2 marks]**

(d) Design a *deterministic* FSA which recognises exactly the same language L.

QUESTION 6(d) **[2 marks]**

(e) Consider the following DFA



Prove that the automaton does not accept strings ending in 11;
that is, prove that for any $\alpha \in \Sigma^*$, $N^*(S_0, \alpha 11)$ is not a final state.

QUESTION 6(e) **[3 marks]**