a) i) [7, 5, 4]

Node (Node (Leaf 7) (Leaf 5)) (Leaf 4)

[4, 5, 7]

ii) ∀x:a.[P(Leaf x)] ∧ ∀t1,t2:Tree a.[P(t1) ∧ P(t2) → P(Node t1 t2)] → ∀t:Tree a.P(t)

iii) Base: ∀x:a.[rev (fringe (Leaf x)) = fringe (mirror (Leaf x))]

IS: ∀t1,t2:Tree a.[(rev (fringe t1) = fringe (mirror t1)) ∧ (rev (fringe t2) = fringe (mirror t2)) →

→ rev (fringe (Node t1 t2)) = fringe (mirror (Node t1 t2))]

b)

Base: ∀x:a.∀ts:[Tree a].∀xs.[a].[frngTR((Leaf x) : ts) xs = frngTR ts (xs++(fringe (Leaf x)))]

1. Take arb. x:a, ts:[Tree a], xs:[a]
2. frngTR((Leaf x) : ts) xs = frngTR ts (xs ++ [x] ) by def. frngTR
3. fringe (Leaf x) = [x] by def. fringe
4. frngTR((Leaf x) : ts) xs = frngTR ts (xs ++ (fringe (Leaf x))) by sub.(2,3)
5. ∀x:a.∀ts:[Tree a].∀xs.[a].[frngTR((Leaf x) : ts) xs = frngTR ts (xs++(fringe (Leaf x)))] by ∀I(1,4)

Inductive step:

1. Take arbitrary t1,t2:Tree a
2. Hypo: ∀ts1:[Tree a].∀xs.[a].[frngTR(t1 : ts1) xs = frngTR ts1 (xs++(fringe t1))] ∧

∀ts:[Tree a].∀xs2.[a].[frngTR(t2 : ts) xs2 = frngTR ts (xs2++(fringe t2))]

1. Take arbitrary ts, instantiate ts1 as (t2 : ts), take arbitrary xs, instantiate xs2 as xs ++ (fringe t1)
2. fringe (Node t1 t2) = (fringe t1) ++ (fringe t2) by def. fringe
3. frngTR ((Node t1 t2):ts) xs = frngTR (t1:t2:ts) xs by def. frngTR
4. = frngTR (t1:ts1) xs by sub.(3,5)
5. = frngTR ts1 (xs++(fringe t1))] by Hypo
6. = frngTR (t2:ts) xs2 by sub.(3,7)
7. = frngTR ts (xs2++(fringe t2)) by Hypo
8. = frngTR ts (xs++(fringe t1)++(fringe t2)) by sub.(3,9)
9. = frngTR ts (xs++fringe(Node t1 t2) ) by sub.(4,10)
10. ∀t1,t2:Tree a.∀ts:[Tree a].∀xs.[a].[frngTR((Node t1 t2) : ts) xs = frngTR ts (xs++(fringe (Node t1 t2)))]

by ∀I(1,11)

^missing a few finishing touches, but I cba

b)

(ii)

Take arbitrary t : Tree a.

frngTR [t] []

= frngTR (t : []) [] (by def. of (:) operator)

= frngTR [] ([] ++ (fringe t)) (by result from part i)

= frngTR [] (fringe t) (by lemma: [] ++ xs = xs)

= fringe t (by def. of frngTR)

Therefore ∀t : Tree a (fringe t = frngTR [t] [])

2

a)

Swap(a, b, i, j) <-> a.length = b.length

^ i, j ∈ [0..a.length)

^ b[i] = a[j]

^ b[j] = a[i]

^ ∀k∈[0..a.length)\{i,j}.b[k] = a[k]

b) a != null ^ i ∈ [0..a.length] ^ Perm(a, a0)

been initialised yet and therefore it will not hold before the loop)

(You cannot talk about variables that are declared in the loop, such as r, in the Invariant as they have not

I’m guessing it’s this: <https://piazza.com/class/if3o7nvl3hq4dm>? cid=98

no, thats what i get as well. I thought of adding the swap post condition, but that wouldn’t make much sense as after two loops, a0[i] = a[j] ^ a[i] = a0[j] would be wrong, right?

c)

Given:

1. 0 <= i <= a.length INV
2. *Perm*(a, a0) INV
3. a != null INV
4. i > 0 cond
5. ∀a,b,i,j Swap(a, b, i j) => Perm(a, b) given lemma
6. r’ = rand(i - 1) code
7. swap(a’, a, i-1, r’) code
8. i’ = i - 1 code
9. a’.length = a.length implicit from code

To show:

(**α)** a’ != null/

(**β**) 0 <= i’ <= a’.length

(**γ**) *Perm*(a’, a0)

Proof:

(**α)** follows from (**3**)

(**10**) 0 < i <= a.length from (**1**), (**4**)

(**11**) 0 <= i - 1 <= a.length from (**10**), arithmetic

(**12**) 0 <= i’ <= a.length from (**8**), (**11**)

(**β**) follows from (**8**), (**11**)

(**13**) i >= 0 from (**4**)

We have shown PRE of rand holds (**13**), so now we can use POST of rand

(**14**) r’ ∈ [0 .. i - 1) from POST of rand and (**6**)

(**15**) i-1, r’ ∈ [0 .. a.length) from (**1**) and (**14**)

So now we can use the POST from swap as we have shown it’s PRE holds

(**16**) Swap(a’, a0, i -1, r’) from (**15**) and (**7**)

(**17**) *Perm*(a’, a0) from (**16**) and (**5**)

(**γ**) follows from (**15**) and (**7**)

d) [b,c,d,e,a]

e) INV2: ∀k∈[ i .. a.length) (a[k] = a0[ (k + 1) % a.length]) (I1) ^

0 <= i <= a.length. (I2) ^

a[0] = a0[i % a.length] = a0[i] (I3) ^

∀j∈[1 .. i) a[j] = a0[j] (I4)

f)

Given:

1. I1 INV
2. I2 INV
3. I3 INV
4. I4 INV
5. i <= 0 !cond

To show:

(**α)** ∀k∈[ 0 .. a.length). a[k] = a0[ (k + 1) % a.length]

Proof:

(**6**) i = 0 from (**2**), (**5**)

(**α**) follows from (**6**), (**1**)

# Question 3

## Part a

∨

→

q

∧

⊥

p

¬

p

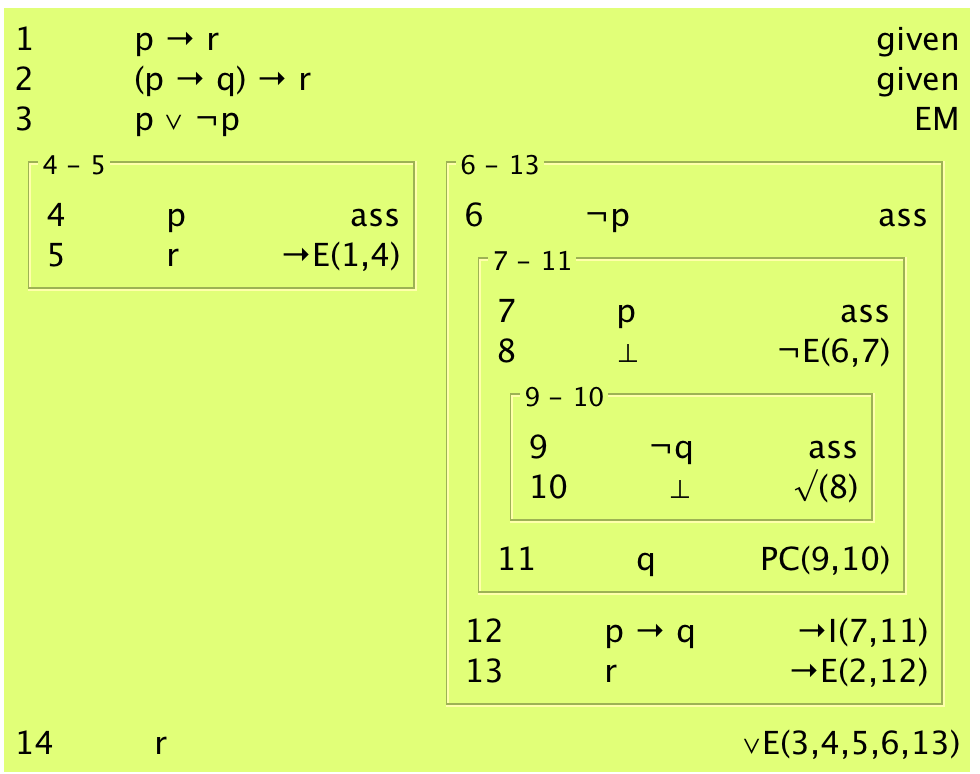
## i)

### ii)

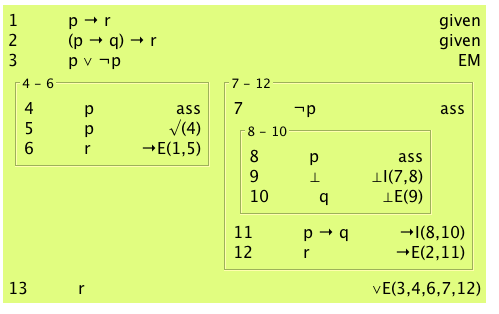
|  |  |  |
| --- | --- | --- |
|  | (p -> q) ^ (-q -> p) |  |
| = | (-p | q) ^ (-q -> p) | a -> b == -a | b |
| = | (-p | q) ^ (--q | p) | a -> b == -a | b |
| = | (-p | q) ^ (q | p) | --a == a |
| = | (q | -p) ^ (q | p) | a | b == b | a |
| = | q | (-p ^ p) | (a | b) ^ (a | c) == a | (b ^ c) |
| = | q | FALSITY | -a ^ a == FALSITY`` |
| = | q | a | FALSITY == a |

I’ve gone a bit further than required, but the answer is still correct and it is in DNF

### iii)



A slightly shorter proof…



A even shorter proof without lemma…

1 p -> r given

2 (p -> q) -> r given

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| 3 ~r ass |

| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

| | 4 p ass | |

| | 5 r -> E (1, 4) | |

| | 6 ⊥ ⊥ I (3, 5) | |

| | 7 q ⊥ E (6) | | |

| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

| 8 p -> q -> I (4, 7) |

| 9 r -> E (2, 8) |

| 10 ⊥ ⊥ I (3, 9) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

11 r PC (3, 10)

NB: Wow such box, ~~F is bottom~~

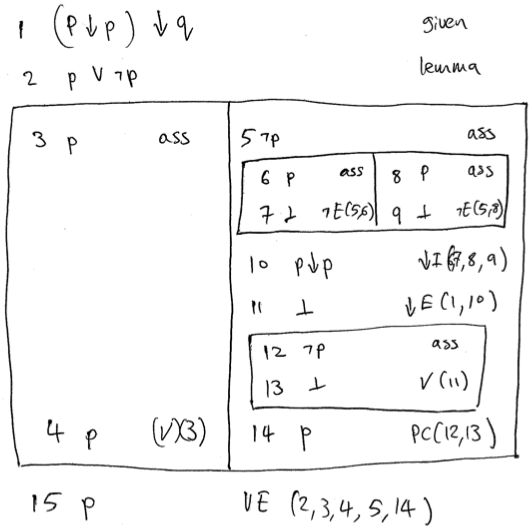
## Part b

i) - p == p ↓ p

ii) p | q == (p ↓ q) ↓ (p ↓ q)

## Part cIMG_20160420_144700.jpg

N.b. first line should say given



4a)13148419_871054476338629_1534770423_o.jpg

4b)

i) ∃n:Nat( in(n, xs) ∧ in(n, ys) )

ii) in(n, xs) ∧ ∀x:Nat( in(x, xs) -> n ≤ x )

iii) ∃zs:[Nat]( merge(t ￫ m = 1 + count(n, zs)) ∧ ￢∃zs:[Nat](merge([n], zs, xs)) ￫ m = 0

∃zs,ys:[Nat]( merge(ys, zs, xs) ∧ ∀y(in(y, ys) ￫ y = n) ∧ ¬in(n, zs) ∧ m = #(ys) )

iv) ∃zs : Nat . ∃ys : Nat .

merge(ys, zs, xs) ^ ∀m : Nat ( in(m, zs) -> in(m, ys) ) ^

∀m1 : Nat . ∀m2 : Nat ( 0 < m1 ^ m1 < m2 ^ m2 < #(ys) -> ys !! m1 != ys !! m2) ^

n = #(ys)

4c)

i) ys = [1,2,3] .

xs = [1,2,3,4,5]

n = 6

ii) The postcondition allows us to remove anything as long as there is no n left in the resulting list

iii) ∃zs(merge(zs, ys, xs) ∧ ￢in(n, ys) ∧ ∀z(in(z, zs) -> z = n))

Or: ∃zs(merge(zs, ys, xs) ∧ ￢in(n, ys) ∧ ∀z(in(z, xs) ∧ z ≠ n -> in(z, ys)))