- 2

017/18 Past Paper Ql

a)

generalise the behavior of

dup2 to remove the hard-cooled ( [] passed to it by dap

Auxiliary Lemma:

(\*\*) Yxs, y5 : [a*]*. du*p2* xs ys = ys ++xs Proof of (\*\*)

let P(xs) be Vys: [a], dup 2 xs ys = ystxs

we prove Vxs: [a]. P(xs) by structural induction over list

Base Case :

To show: *Vy*s: [«]*da*p2 [] ys = ys ++ []

Take ys : [a] arbitrary.

dup2 [] ys = ys def dupa

= ys++ [] from (B)

Inductive Step:

Take arbitrary xia, xs: [a] IH: Vys: [á]. dap2 xs *ys - ys* ++ xs

using ys to disambiguate between To show : Vys: [a] dup 2 (x:xs) ys = ys'++ (x:xs)

Its and what we are preving here Take ys' arbitrary

dup 2 (x:xs) ys' = dup2 xs (ys'++ [x]) def dup2

= (ys' +[x*])* ++ x*s*

by IH (take ys to be ys'++ [x]) YS' H (x: xs)

by A

now we wse (\*\*) to prove (\*)

. a) continued...

Proof of (\*)

Substitute ys by [] in (\*\*) to obtain Vys: []. dep 2 xs [] = [] ++ xs

-> Vxri [a]. dap 2 xs [] = xs - Exs: [c]. dup xs xs

by (B) by def

dus

Vi: Int. Vc: Char. P(C1 i c)

P(c2) lî Vis : Lowve]. We, : 77. [P(6.) → P(C3 85 *4,)])*

*I* → VEL: 71. P*(+1)*

note be careful to guantify correctly

;;) / Vt, :II. Q((4 €) don't need to break into Tl deletype here as this is induction are T2

In Wta, ti, te : 12. [Q[ta)Q(tn)Q(€.) → Q (cs ta ty te)*] )* → V*42*:1*2. Q (+2*)

-

717) note at Bal - b= Bool

note the ca and b are included in the subtype here. *Iv*bbe! Bel. RacOb b

^ Hta, th : (13 Beel Boi). [R(ta) R(tv) = R(C7 tx +b)] ) 14 Vbs: [Beel]. Vez (13 Bell Bel). [R(+3) >R (c8 bs Ez)]

+ V63(T3 Boel Beel). R(43)

*s) ;*)

using the provided auxiliary predi*cate* we get (P(LJ) ( 1 Ves: 117. [All () PCN*A e.))) → V*e: T. Ple)

(without wing All (ts) you'll need to create your own way

of covering P(E) for each t in the list ts. I

Core

need to

add

Q (e,t) to LHS of

inclective

steps.

in*dach*

hive

I

steps

c) )

Hej, k', {": T. [R(4,4') ^ Q (4,x') a RC4",{*")* « Q (4,") → Q (4,{")] A Vt, t':T, VES: [1]. [R(4,6') Q(4*,6') –* Q(N*il (4:4*8), Nd (e":65))] Ves: [1].[Q *(Nel (@VA* ) : to), Nd es)] Yes, Ex": [1]. [Q*(Nl (Nd* (49:45)): 40, *Nd (val t):4*8:LG! Es)]

base cases

→ *V6*, 6': T. (R (, 6') *= Q(*6,+)7

All RHS (or baie cases) just need to establis? Q(6,6)

2017/18 Past Paper Q2

almost no-one remembered to get xx a) The definition of Sorted is:

both bounds correct! Sorted ( a[x..y)) — Vije [0.length). [xsisjky >> a[i] = a[j*]]*

note: this makes a pairwise comparison of ☺

elements in a other possibilities:

Sorted (a[x.y)) Étke [o.au length-1). [xskky-1 > a[k] < a[kul]]

I must be careful net, note: compares just neighbours and relies ". I to go out of bounds

on transitivity of < Sorted (a[x..y)) # the [.. a.length). [xsk<y> a[k] <a[k..y)]

note: makes nice we of our array slice notation

ed

b) 7) The invariant I, is:

I Å a(...) ~ a[..) oren (done Sorted (a...)))

A note: & here is too strong! xx

Sorted (a [..)) could be true at the start of . the method, but done will always be set to false.)

ii) The proof corresponds to proving the obligation

. In-(! done) →Q

Giveni

. (1) a[..) ~ a[..) pre

(2) done Sorted (a[

(3) (+done) To show:

@) a[..) ~0 [..)pre (3) Sorted (a(..))

from from from

I, I rcond

note: no code is executed here, so

there is no need for any-old subu.

mm

Prof: (a) follows immediately from (1)

(4) done

e from (3) (1) follows from (2) and (4)

Von

1.

c) The proof corresponds to proving the obligation:

Iz done to donedd, is idd] ^ ?ld < a length-la done (donedd a a[idd] <a [idd +1] a t= iddal

ī old subs -

L cond Iz

with old subs

Given

from Iz from I from Iz

this should incleed be < a length (there was a type in the paper) but it doesn't really effect the proef here

1

from cond

aal

(1) a[..) ~ a [..) ore (2) Os Toldi Galength t (3) donedd Sorted (a[..&dd +()) (4) iuld < a.length - (5) done (donedd a afield] sa [hold +1]) (6) i = Add to

- you could do this is To show:

with = too. u (d) al.) ~ a [..) pre (B) osis a length (8) done → Sorte*d (*al . Thl))

from cale line 16 from calle line 17

m2

W

note: some people chose to

care split here on the value of donedd

11

.. (a) follows imme*diatel*y from (1)

(7) Osidd <a, lengthal from (2) and (4) (8) osłoid +1 <a.length from (7) and writh. (a) os iddtl < a length from (8) relaxing < to s (8) fellows from (9) and (6) (10) done → (Sorted (alv. idd +1)) ^ alield] = a[i*dd +1]*) (11) done → Sorted (al. Teid +2) (12) done → Sort*ed (* aliu*ld +*+1)). (8 follows from (12) and (6)

from (5) and (3) from (10) and def. Sorted from (11) and arith.

Variant

*d) ;*) The variant Ve is:

note: a common alternative was V2 = a.lengthai

- a length-loi which was also fine

(though the -I is not strictly necessary) ii) To prove that the loop terminates, we have to show that the variant V2 is

bounded below and decreases on every loop iteration. 1. The proof corresponds to proving the then obligations : V2>1

Uz [intold, done to doneld] > V2 Given: same as (c) To show: on a length -:>,

(B) a length-ield> a.length-i

Proof: (7

cm

(7) E*el*d < ailength - 1 (8) 0 < ailength - - *Told* (9) 0 < ailength - *( iodd +1)* (ia) I ailength *-(evid +1)* (2) follows from (10) and (6) by (1*1) Teld = -l* (2) | > c (3) alen A - + > A. less+- (14) a length-(i-1) > a lengthai (13) follows from (14) and (11)

from (4) from (7) [-teld from each side]

from (8) by arith.

from (9) shifting <tee for integer values flipping the two sides

from (6) known fact of mathematics from (12) [+ V2 to each side 7 from (13) by arith,

e) The shuffle method is not guaranteed to actually change the array

(the original array itself would be a valid permutation of the input) so the overall shuffle sort merlod might never terminate

cany similar argument is fine ☺ keywords I was locking for lack of progress infinite loop

permutations don't have to change

etc...

veni

f) :) The property is that "different values of the integer input must lead to

different permutations of the input array." Then, the shufflesort me that would be guaranteed to make progress through all the possible permutations of the array, and thus eventually find the sented one.

a course, this may still take a very long time for large inputs!

were

some

ii) We were looking for some thing like:

Ya,b,ce int[], Ek, kze {.. Carlength)!}

kitkę n Shaff(a.), b[..), k.) 1 Shuft(a[.), C[..), kz) = b[..) \* ¢[...]

this wa little loose as it won't perfectly handle

repeated elements nearly badean done = false;

but we were just locking The code should be updated. int cut = 0;

for this kind of shape. iii) as such

while (! done) {

- shuffle Calent);

(or similar)

anttt

i