40006 Reasoning about Programs

May 2021

1. a i) $[5, 3, 1, 2, 4]$

ii) Induction over xs : `[a]`

- **Base Case**:

To show: `revTR ([] ++ ys) zs = revTR ys ((rev []) ++ zs)`

Take ys, zs : `[a]` arbitrary,

(1) `revTR ([] ++ ys) zs`

(2) `= revTR ys zs` from (D)

(3) `= revTR ys ([] ++ zs)` from (D)

(4) `= revTR ys ((rev []) ++ zs)` from definition of `rev`

- **Inductive Step**:

Take x: `a`, xs: `[a]` arbitrary,

IH: $\forall ys$: `[a]`.$\forall zs$: `[a]`.

[`revTR (xs++ys) zs = revTR ys ((rev xs) ++ zs)`]

To show: $\forall ys'$: `[a]`.$\forall zs'$: `[a]`.

`revTR ((x : xs) ++ ys') zs' = revTR ys' ((rev(x : xs)) ++ zs')`

Take ys', zs' : `[a]` arbitrary,

(1) `revTR ((x : xs) ++ ys') zs'`

(2) `= revTR (([x] ++ xs) ++ ys') zs'` from (B)

(3) `= revTR ([x] ++ (xs ++ ys')) zs'` from (A)

(4) `= revTR (x : (xs ++ ys')) zs'` from (B)

(5) `= revTR (xs ++ ys') (x : zs')` from definition of `revTR`

(6) `= revTR ys' ((rev xs) ++ (x : zs'))` from IH

(7) `= revTR ys' ((rev xs) ++ [x] ++ zs')` from (B)

(8) `= revTR ys' ((rev (x : xs)) ++ zs')` from definition of `rev` ■

iii) Prove $\forall xs$: `[a]`.[`revTR xs [] = rev xs`]

(1) `revTR xs []`

(2) `= revTR (xs ++ []) []` from (C)

(3) `= revTR [] ((rev xs) ++ [])` from $a.i$

(4) `= rev xs ++ []` from definition of `revTR`

(5) `= rev xs` from C ■

b i. `v` $\triangleq$ `IntV 4`

ii. `e'` $\triangleq$ `Cond (IntE 1) (IntE 2) (IntE 3)`

iii. $\forall i : \texttt{Int}.[\exists \text{v}.[EVal(\texttt{IntE i}, \texttt{v}) \land VType(\texttt{v}, \texttt{IntT})]]$

$\land \forall b : \texttt{Bool}.[\exists \text{v}.[EVal(\texttt{Bool b}, \texttt{v}) \land VType(\texttt{v}, \texttt{BoolT})]]$

$\land \forall e_1, e_2, e_3 : \texttt{Exp}.\forall t' : \texttt{TypeT}.$

$[EType(e_1, \texttt{Bool}) \land \exists v.[EVal(e_1, v) \land VType(v, \texttt{BoolT})]$

$\land EType(e_2, t') \land \exists v_1[EVal(e_2, v_1) \land VType(v_1, t')]$

$\land EType(e_3, t') \land \exists v_2[Eval(e_3, v_2) \land VType(v_2, t')]$

$\to \exists v_3[EVal(\texttt{Cond e1 e2 e3}) \land VType(v_3, t')]]$

$\to \forall e : \texttt{Exp}.\forall t' : \texttt{TypeT}.\ [EType(e, t) \to \exists v.[EVal(e, v) \land VType(v, t)]]$

PART 2

a.

i. The result is 2.

ii.[['w'],[],['+','?'], null, null, null]

b.

i. $M_1 \triangleq \mathtt{in}[\ldots) \approx \mathtt{in}_{pre}[\ldots) \land \exists k : N.[Occurs(\mathtt{in}[\,..), c) = k \land \mathtt{in}[\cdots) \approx Flatten(\mathtt{out}[\cdots), c, k) : \mathtt{in}[\mathtt{start}\cdots)]$

$M_2 \triangleq \mathtt{in}[\ldots) \approx \mathtt{in}_{pre}[\ldots) \land \exists k : N.[Occurs(\mathtt{in}[\,..), c) = k \land \mathtt{in}[\cdots) \approx Flatten(\mathtt{out}[\cdots), c, k) : \mathtt{out}[k]]$

ii. $I \triangleq 0 \le \mathtt{pos} \le \mathtt{in.length} \land 0 \le \mathtt{start} \le \mathtt{in.length} \land \mathtt{start} - \mathtt{pos} \le 1 \land \mathtt{in}[\cdots) \approx \mathtt{in}_{pre}[\cdots) \land \mathtt{found} = Occurs(\mathtt{in}[\cdots\mathtt{pos}), c) \land \mathtt{in}[\cdots\mathtt{start}) \approx Flatten(\mathtt{out}[\cdots), \mathtt{c}, \mathtt{found})$

iii. $V \triangleq \mathtt{in.length} - \mathtt{pos}$.

c. To show:

$I[\mathtt{pos} \to \mathtt{pos}_{old}, \mathtt{start} \to \mathtt{start}_{old}, \mathtt{found} \to \mathtt{found}_{old}, \mathtt{out} \to \mathtt{out}_{old}] \land COND[\mathtt{pos} \to \mathtt{pos}_{old}] \land CODE \to INV.$

Note that in is not changed.

Assume:

1)	$\mathtt{in}[\ldots) \approx \mathtt{in}_{pre}[\ldots)$	from INV
2)	$0 \le \mathtt{pos}_{old} \le \mathtt{in.length}$	from INV
3)	$0 \le \mathtt{start}_{old} \le \mathtt{in.length}$	from INV
4)	$\mathtt{start}_{old} - \mathtt{pos}_{old} \le 1$	from INV
5)	$\mathtt{found}_{old} = Occurs(\mathtt{in}[\cdots\mathtt{pos}_{old}), c)$	from INV
6)	$\mathtt{in}[\cdots\mathtt{start}_{old}) \approx Flatten(\mathtt{out}_{old}[\cdots), c_{old}, \mathtt{found}_{old})$	from INV
7)	$\mathtt{out}[\mathtt{found}_{old}] \approx \mathtt{out}_{old}[\mathtt{found}_{old}]$	implicit
8)	$\mathtt{pos}_{old} < \mathtt{in.length}$	from COND
9)	$\mathtt{out}[\mathtt{found}_{old}] \approx \mathtt{in}[\mathtt{start}_{old}\cdots\mathtt{pos}_{old})$	from COND &

POST of slice

10)	$\mathtt{found} = \mathtt{found}_{old} + 1$	from CODE
11)	$\mathtt{start} = \mathtt{pos}_{old} + 1$	from CODE
12)	$\mathtt{pos} = \mathtt{pos}_{old} + 1$	from CODE

13) $\texttt{int}[\texttt{pos}_{old}] = \texttt{c}$ from 2c

Proof:

14) $\texttt{in}[\ldots) \approx \texttt{in}_{pre}[\ldots)$ 1)

15) $0 \leq \texttt{pos} \leq \texttt{in.length}$ 2), 8), 12)

16) $\texttt{pos} = \texttt{start}$ 11), 12)

17) $0 \leq \texttt{start} \leq \texttt{in.length}$ 15), 16)

18) $\texttt{start} - \texttt{pos} \leq 1$ 16)

19) $Occurs(\texttt{in}[\cdots\texttt{pos}), \texttt{c}) = Occurs(\texttt{in}[\cdots\texttt{pos}_{old}), \texttt{c}) + 1$ 12), 13),

def of Occurs

20) $Occurs(\texttt{in}[\cdots\texttt{pos}), \texttt{c}) = \texttt{found}_{old} + 1 = \texttt{found}$ 5), 10),

13), 19)

21) $Flatten(\texttt{out}[\cdots), \texttt{c}, \texttt{found})$

$\approx$ $Flatten(\texttt{out}[\cdots), \texttt{c}, \texttt{found}_{old} + 1)$ 10)

$\approx$ $Flatten(\texttt{out}[\cdots), \texttt{c}, \texttt{found}_{old}) : \texttt{out}[\texttt{found})_{old}] : \texttt{c}$ def of

Flatten

$\approx$ $\texttt{in}[\cdots\texttt{start}_{old}) : \texttt{in}[\texttt{start}_{old}\cdots\texttt{pos}_{old}) : \texttt{c}$ 6), 9)

$\approx$ $\texttt{in}[\cdots\texttt{pos}_{old}) : \texttt{int}[\texttt{pos}_{old}]$ 13)

$\approx$ $\texttt{in}[\cdots\texttt{start})$ 11)

d. No. Consider a string full of c, the char we want to split at. For example, let c = 'c' and the string is ['c', …, 'c'] of length n. Then out should be [[], …, []] with n + 1 []s. Thus the effective length is in.length + 1 in the worst case.