15-150 Assigment 8

Jonathan Li
jlli
Section S
June 12, 2016

Task 4.1

Theorem 1:

(i.) $\mathcal{R}$(`LQ.emp`, `LLQ.emp`)

(ii.) $\forall$ `x : int`, `l : int list`, `f : int list`, `b : int list`:
If $\mathcal{R}$(`l`, (`f`, `b`)), then $\mathcal{R}$(`LQ.ins(x,l)`, `LLQ.ins(x,(f,b))`)

(iii.) $\forall$ `l : int list`, `f : int list`, `b : int list`,
if $\mathcal{R}$(`l`, (`f`, `b`)), then one of the following is true:

(a) `LQ.rem l` $\cong$ `NONE` and `LLQ.rem (f,b)` $\cong$ `NONE`

(b) $\exists$ `x : int`, `y : int`, `l' : int list`, `f' : int list`, `b' : int list` such that:

i. `LQ.rem l` $\cong$ `SOME(x, l')`

ii. `LLQ.rem (f, b)` $\cong$ `SOME(y, (f',b'))`

iii. `x` $\cong$ `y`

iv. $\mathcal{R}$(`l'`, (`f'`, `b'`))

Here,

$$\mathcal{R}(\texttt{l : int list}, (\texttt{f : int list}, \texttt{b : int list})) \iff \texttt{l} \cong \texttt{f@(LLQ.rev b)}$$

Using this definition for $\mathcal{R}$, I will write this proof sequentially, starting with Theorem 1 (i.).

Theorem 1 (i.): $\mathcal{R}$(`LQ.emp`, `LLQ.emp`)
Upon inspection of the code for the `LLQ` structure, we find this line:

```
val emp = ([], [])
```

By Referential Transparency, what we really want to show for this section of the proof, then, is $\mathcal{R}$(`LQ.emp`, `([], [])`). By our definition of $\mathcal{R}$:
To Show: `LQ.emp` $\cong$ `[] @ (LLQ.rev [])`
Let's begin with the right hand side of this equivalence, and work from there.
Proof:

`[] @ (LLQ.rev [])`	$\cong$	`[] @ []`	[step]
	$\cong$	`[]`	[Lemma 2/3]
	$\cong$	`LQ.emp`	[defn of LQ.emp]

Thus, by extensional equivalence, `[] @ (LLQ.rev [])` $\cong$ `LQ.emp`. Considering that `LLQ.emp` $\cong$ `([], [])` and our definition of $\mathcal{R}$, we have shown:

$\mathcal{R}$(`LQ.emp`, `LLQ.emp`)

Thus, Theorem 1 (i.) holds.

Task 4.1 (cont.)

Theorem 1 (ii.):
$\forall$ `x : int`, `l : int list`, `f : int list`, `b: inst list`:
If $\mathcal{R}$(`l`, (`f`, `b`)), then $\mathcal{R}$(`LQ.ins(x, l)`, `LLQ.ins(x, (f, b))`.

Assumption (ii.): $\mathcal{R}$(`l`, (`f`, `b`)) $\implies$ `l` $\cong$ `f@(LLQ.rev b)`

Inspecting the code for `LLQ.ins` in the `LLQ` structure, we find that:

`LLQ.ins(x,(f,b))`	$\cong$	`(f, x::b)`	[step]

This step is valid if we assume that `LLQ.ins` is total, and `x : int` is a value. Now, considering our definition for $\mathcal{R}$, by Referential Transparency:
To Show: `LQ.ins(x,l)` $\cong$ `f@(LLQ.rev (x::b))`
Proof:

`LQ.ins(x,l)`	$\cong$	`l@[x]`	[step]

Similarly to above, this step is valid if we consider that `LQ.ins` is total, and that `x` is a value.

`LQ.ins(x,l)`	$\cong$	`(f@(LLQ.rev b))@[x]`	[Assumption (ii.), Referential Transparency]
	$\cong$	`f@((LLQ.rev b)@[x])`	[Lemma 1]

Now, let us consider the right hand side of the congruence in our 'To Show' statement:

`f@(LLQ.rev (x::b))`	$\cong$	`f@((LLQ.rev b)@[x])`	[step]

By extensional equivalence, we have shown that `LQ.ins(x,l)` $\cong$ `f@(LLQ.rev (x::b))`. Considering that `LLQ.ins(x,(f,b))` $\cong$ `(f, x::b)` and our definition for $\mathcal{R}$, we have shown that:

$\mathcal{R}$(`LQ.ins(x,l)`, `LLQ.ins(x,(f,b))`)

Thus, Theorem 1 (ii.) holds.

Task 4.1 (cont.)

Theorem 1 (iii.): $\forall$ `l : int list`, `f : int list`, `b : int list`,
if $\mathcal{R}$(`l`, (`f`, `b`)), then one of the following is true:

(a) `LQ.rem l` $\cong$ `NONE` and `LLQ.rem (f,b)` $\cong$ `NONE`

(b) $\exists$ `x : int`, `y : int`, `l' : int list`, `f' : int list`, `b' : int list` such that:

  i. `LQ.rem l` $\cong$ `SOME(x, l')`

  ii. `LLQ.rem (f, b)` $\cong$ `SOME(y, (f',b'))`

  iii. `x` $\cong$ `y`

  iv. $\mathcal{R}$(`l'`, (`f'`, `b'`))

Assumption (iii.): For some `l : int list`, `f : int list`, `b : int list`, $\mathcal{R}$(`l`, (`f`, `b`))
Upon inspection of the code for `LQ.rem` in the `LQ` structure, we find that when `LQ.rem` is applied to some `l : int list`, there can only be two cases:

`LQ.rem(l)`	$\cong$	`NONE`
	OR	
`LQ.rem(l)`	$\cong$	`SOME(x, l')`

...where `x` is some value of type `int`, and `l'` is some value of type `int list`. Let us approach this section of the proof be separating out these cases.

Case 1: `LQ.rem(l)` $\cong$ `NONE`
This case only results when `l` $\cong$ `[]`. We can say then that:

`LQ.rem(l)`	$\cong$	`NONE`	[Case Assumption]
`l`	$\cong$	`[]`	[stepping through `LQ.rem`, symmetry]
`l`	$\cong$	`LQ.emp`	[defn of `LQ.emp`]

Now, if we consider Assumption (iii.) ($\mathcal{R}$(`l`, (`f`, `b`))) together with
Theorem 1 (i.) ($\mathcal{R}$(`LQ.emp`, `LLQ.emp`)), we can see that in this case,

`(f,b)`	$\cong$	`LLQ.emp`	[Assumption (iii.), Theorem 1 (i.)]
	$\cong$	`([],[])`	[defn of `LLQ.emp`]

Now, if we apply `LLQ.rem` to `(f,b)`,

`LLQ.rem(f,b)`	$\cong$	`LLQ.rem([],[])`	[Referential Transparency]
	$\cong$	`case ([],[]) of`...	[step]
	$\cong$	`NONE`	[step]

Thus, we have shown that assuming $\mathcal{R}$(`l`, (`f`, `b`)) for some values `l : int list`, `f : int list`, `b : int list`, `LQ.rem(l)` $\cong$ `NONE` necessitates that `LLQ.rem(f,b)` $\cong$ `NONE` as well.
Theorem 1 (iii.) (a) holds.

Task 4.1 (this is getting long...)

<u>Case 2:</u> `LQ.rem(l)` $\cong$ `SOME(x, l')`, where `x : int` and `l' : int list` are values.

For this section of the proof, let's examine the code for `LQ.rem` in the `LQ` structure:

```
fun rem [] = NONE
  |rem (y::ys) = SOME(y, ys)
```

From the above, we can reason that:

`LQ.rem(l)`	$\cong$	`SOME(x, l')`	[Case Assumption]
`l`	$\cong$	`x::l'`	[stepping through `LQ.rem`, symmetry]

We can now restate Assumption (iii.) with this congruence. Now, $\mathcal{R}$(`l`,(`f`, `b`)) tells us:

`l`	$\cong$	`f@(LLQ.rev b)`	[defn of $\mathcal{R}$]
`x::l'`	$\cong$	`f@(LLQ.rev b)`	[Referential Transparency]

From this congruence, we can reason that the `int list f@(LLQ.rev b)` contains at least one element at the head, `x`. Since `f : int list` and, since `b : int list`, according to the implementation `(LLQ. rev b) : int list`, we again have two situations that we can case on:

`f`	$\cong$	`[]`
	OR	
`f`	$\cong$	`y::ys`

...where `y` is some value of type `int` and `ys` is some value of type `int list`.

<u>Case 2.1:</u> `f` $\cong$ `[]`
In this case, let us consider `LLQ.rem(f,b)`:

`LLQ.rem(f,b)`	$\cong$	`LLQ.rem([],b)`	[Case Assumption]
	$\cong$	`case ([],b) of ([],[]) => ...`	
		`|(y::ys,_) => ...`	
		`|([],_) => LLQ.rem(LLQ.rev b, [])`	[step]
	$\cong$	`LLQ.rev(LLQ.rev b, [])`	[step]
	$\cong$	`case (LLQ.rev b, []) of ...`	[step]

Here, remember our restated Assumption (iii.):

`f@(LLQ.rev b)`	$\cong$	`x::l'`	[Assumption (iii.)]
`[]@(LLQ.rev b)`	$\cong$	`x::l'`	[Case Assumption, Referential Transparency]
`LLQ.rev b`	$\cong$	`x::l'`	[Lemma 2]

Essentially, we learn that `(LLQ.rev b)` is not the empty list. Let us now consider variables `y : int` and `ys : int list`, such that `y` $\cong$ `x` and `ys` $\cong$ `l'`, and that:

`LLQ.rev b`	$\cong$	`y::ys`	[Referential Transparency]

Task 4.1 (Almost there!)

Case 2.1 (cont.)
Returning to our analysis of `LLQ.rem(f,b)`,

`LLQ.rem(f,b)`	$\cong$	`case (LLQ.rev b, []) of ...`	[stepping through `LLQ.rem`]
	$\cong$	`case (y::ys, []) of ...`	[Referential Transparency]
	$\cong$	`SOME(y,(ys,[]))`	[step]

Let the variables in the problem statement `f' : int list`, `b' : int list` be defined

`f'` $\cong$ `ys` $\cong$ `l'`
`b'` $\cong$ `[]`

Then, from the above analysis, we have that `LLQ.rem(f,b)` $\cong$ `SOME(y, (f',b'))`, fulfilling part (ii.) of Theoem (iii.) (b). Part (i.) is fulfilled by our Case Assumption (`LQ.rem(l)` $\cong$ `SOME(x,l')`), and by construction we fulfill part (iii.) (`x` $\cong$ `y`). All that is left is part (iv.) of this case in the theorem. If we consider `f'@(LLQ.rev b')`,

`f'@(LLQ.rev b')`	$\cong$	`l'@(LLQ.rev [])`	[Referential Transparency]
	$\cong$	`l'@[]`	[stepping through `LLQ.rev`]
	$\cong$	`l'`	[Lemma 3]

Thus, by extensional equivalence, `f'@(LLQ.rev b')` $\cong$ `l'`, or $\mathcal{R}$(`l'`,(`f'`, `b'`)), fulfilling part (iv.). Thus, in this case, with all four parts fulfilled, Theorem 1 (iii.) (b) holds.

Case 2.2: `f` $\cong$ `y::ys` for some values `y : int`, `ys : int list`
Now is a good time to look at our restated Assumption (iii.) from page 4:

`x::l'`	$\cong$	`f@(LLQ.rev b)`	[Case 2 restated Assumption (iii.)]
	$\cong$	`(y::ys)@(LLQ.rev b)`	[Case Assumption, Referential Transparency]

We know the following from stepping through the definition of `@`:

`(x::xs)@l`	$\cong$	`x::(xs@l)!`	[step through `@`]

Using this congruence, we can move forward.

`x::l'`	$\cong$	`y::(ys@(LLQ.rev b))`	[step through `@`]
`x`	$\cong$	`y`	[Lemma 4]
`l'`	$\cong$	`ys@(LLQ.rev b)`	[Lemma 4]

Task 4.1 (Last page!)

Case 2.2 (cont.)
With this information, let us again consider `LLQ.rem(f,b)`:

`LLQ.rem(f,b)`	$\cong$	`LLQ.rem((y::ys),b)`	[Case Assumption]
	$\cong$	`case (y::ys,b) of ...`	[step]
	$\cong$	`SOME(y,(ys,b))`	[step]

As before, let the variables in our problem statement `f'` and `b'` be defined:

`f'` $\cong$ `ys`
`b'` $\cong$ `b`

Then, from our earlier congruences:

`l'`	$\cong$	`ys@(LLQ.rev b)`	[Lemma 4]
`l'`	$\cong$	`f'@(LLQ.rev b')`	[Referential Transparency]

Which proves part (iv.) of this case in the theorem: $\mathcal{R}$(`l'`,(`f'`, `b'`).
...
Well, we're done! For this case at least, part (i.) is the larger case assumption in Case 2. If we look at how we just defined `f'` and `b'`, we can see that:

`LLQ.rem(f,b)`	$\cong$	`SOME(y,(ys,b))`	[stepping through `LLQ.rem`]
	$\cong$	`SOME(y,(f',b'))`	[Referential Transparency]

...which is exactly what is asked of us in part (ii.). Early, we also showed that, by Lemma 4, `x` $\cong$ `y`, covering part (iii.), so we have shown all four parts of this case to be true. Thus, Theorem 1 (iii.) (b) holds.

We have exhaustively (lol I'm exhausted) shown that parts (i), (ii), and (iii) hold $\forall$ `l : int list`, `f : int list`, `b : int list`.

$\therefore$ Theorem 1, as a whole, must be true.

This was actually kind of fun. Don't tell anyone I said that though. Pretty unfortunate if I missed something, but whatever, I'll learn.