I217: Functional Programming

12. Program Verification – Lists

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Lists
- Associativity of _@_
- Correctness of a Tail Recursive Reverse

Lists

Lists can be specified in CafeOBJ as follows:

```
mod! LIST1 (E :: TRIV) {
  [List]
  op nil : -> List {constr}
  op _|_ : Elt.E List -> List {constr} .
  …
}
```

Terms nil, e1 | nil, e1 | e2 | nil, e1 | e2 | e3 | nil, e1 | e2 | e3 | e4 | nil denote lists that consist of 0, 1, 2, 3, 4 elements, respectively, if e1, e2, e3, e4 are terms of Elt.E.

Lists

For every sort *S*, the following operator and equations are prepared in the built-in module EQL that is imported by BOOL:

```
op _=_ : S S -> Bool {comm} .
eq (X = X) = true .
eq (true = false) = false .
```

where X is a variable of *S*.

We declare the following equations in LIST1 for List:

```
eq (nil = E | L1) = false .
eq (E | L1 = E2 | L2) = (E = E2) and (L1 = L2) .
```

where E,E2 are variables of Elt.E and L1,L2 are those of List.

Let L3 be a variable of List in the rest of the slides as well.

Lists

Concatenation of lists is defined as follows:

```
op _@_ : List List -> List .
eq nil @ L2 = L2 .                      -- (@1)
eq (E | L1) @ L2 = E | (L1 @ L2) .      -- (@2)
```

```
(e1 | e2 | nil) @ (e3 | e4 | nil)
→ e1 | ((e2 | nil) @ (e3 | e4 | nil))   by (@2)
→ e1 | e2 | (nil @ (e3 | e4 | nil))     by (@2)
→ e1 | e2 | e3 | e4 | nil               by (@1)
```


Lists

Reverse of lists is defined as follows:

```
op rev1 : List -> List .
eq rev1(nil) = nil .                          -- (r1-1)
eq rev1(E | L1) = rev1(L1) @ (E | nil) .      -- (r1-2)
```

```
rev1(e1 | e2 | e3 | nil)
→ rev1(e2 | e3 | nil) @ (e1 | nil)                          by (r1-2)
→ (rev1(e3 | nil) @ (e2 | nil)) @ (e1 | nil)                by (r1-2)
→ ((rev1(nil) @ (e3 | nil)) @ (e2 | nil)) @ (e1 | nil)      by (r1-2)
→ ((nil @ (e3 | nil)) @ (e2 | nil)) @ (e1 | nil)            by (r1-1
→ ((e3 | nil) @ (e2 | nil)) @ (e1 | nil)                    by (@2)
→ (e3 | (nil @ (e2 | nil))) @ (e1 | nil)                    by (@2)
→ (e3 | e2 | nil) @ (e1 | nil)                              by (@1)
→ e3 | ((e2 | nil) @ (e1 | nil))                            by (@2)
→ e3 | e2 | (nil @ (e1 | nil))                              by (@2)
→ e3 | e2 | e1 | nil                                        by (@1)
```

Lists

A tail recursive reverse of lists is defined as follows:

```
op rev2 : List -> List .
op sr2 : List List -> List .
eq rev2(L1) = sr2(L1,nil) .                 -- (r2)
eq sr2(nil,L2) = L2 .                       -- (sr2-1)
eq sr2(E | L1,L2) = sr2(L1,E | L2) .        -- (sr2-2)
```

rev2(e1 | e2 | e3 | nil)
→ sr2(e1 | e2 | e3 | nil, nil) by (r2)
→ sr2(e2 | e3 | nil, e1 | nil) by (sr2-2)
→ sr2(e3 | nil, e2 | e1 | nil) by (sr2-2)
→ sr2(nil, e3 | e2 | e1 | nil) by (sr2-2)
→ e3 | e2 | e1 | nil by (sr2-1)

Lists

Let l(L) and r(L) be terms of a same sort in which a variable L of List occurs. Then, the following (A) and (B) are equivalent:

(A) (∀L:List) l(L) = r(L)

(B) I. l(nil) = r(nil)

II. If l(l) = r(l), then l(e | l) = r(e | l), where e is a fresh constant of Elt.E and l is a fresh constant of List.

It suffices to prove (B) so as to prove (A). This is called proof by *structural induction* on List L. I is called the *base case*, II is called the *induction case*, and l(l) = r(l) is called the *induction hypothesis*.

Associativity of _@_

(L1 @ L2) @ L3 equals L1 @ (L2 @ L3) for lists L1,L2,L3. This is what we will prove by structural induction on L1.

Theorem 1 [Associativity of _@_ (assoc@)]
(L1 @ L2) @ L3 = L1 @ (L2 @ L3)

Proof of Theorem 1 By structural induction on L1.

Let e be a fresh constant of Elt.E, l1,l2,l3 be fresh constants of List.

I. Base case
What to show is (nil @ l2) @ l3 = nil @ (l2 @ l3).

(nil @ l2) @ l3
→ l2 @ l3 by (@1)

nil @ (l2 @ l3)
→ l2 @ l3 by (@1)

Associativity of _@_

II. Induction case
What to show is ((e | l1) @ l2) @ l3 = (e | l1) @ (l2 @ l3)
assuming the induction hypothesis
(l1 @ L2) @ L3 = l1 @ (L2 @ L3) -- (IH)

((e | l1) @ l2) @ l3
→ (e | (l1 @ l2)) @ l3 by (@2)
→ e | ((l1 @ l2) @ l3) by (@2)
→ e | (l1 @ (l2 @ l3)) by (IH)

(e | l1) @ (l2 @ l3)
→ e | (l1 @ (l2 @ l3)) by (@2)

End of Proof of Theorem 1

Associativity of _@_

Theorem 1 [Associativity of _@_ (assoc@)]
(L1 @ L2) @ L3 = L1 @ (L2 @ L3)

Proof of Theorem 1 By structural induction on L1.

I. Base case

```
open LIST1 .
  -- fresh constants
  ops l2 l3 : -> List .
  -- check
  red (nil @ l2) @ l3 = nil @ (l2 @ l3) .
close
```


Associativity of _@_

II. Induction case

```
open LIST1 .
  -- fresh constants
  ops l1 l2 l3 : -> List .
  op e : -> Elt.E .
  -- induction hypothesis
  eq (l1 @ L2) @ L3 = l1 @ (L2 @ L3) .
  -- check
  red ((e | l1) @ l2) @ l3 = (e | l1) @ (l2 @ l3) .
close
```

End of Proof of Theorem 1

Correctness of a Tail Recursive Reverse

Theorem 2 [Correctness of a tail recursive reverse (ctrr)]
rev1(L1) = rev2(L1)

Proof of Theorem 2 By structural induction on L1.

Let e be a fresh constant of Elt.E, l1 be a fresh constant of List.

I. Base case

What to show is rev1(nil) = rev2(nil).

rev1(nil)
→ nil by (r1-1)

rev2(nil)
→ sr2(nil,nil) by (r2)
→ nil by (sr2-1)

Correctness of a Tail Recursive Reverse

II. Induction case
What to show is rev1(e | l1) = rev2(e | l1)
assuming the induction hypothesis
rev1(l1) = rev2(l1) -- (IH)

rev1(e | l1)
→ rev1(l1) @ (e | nil) by (r1-2)
→ rev2(l1) @ (e | nil) by (IH)
→ sr2(l1,nil) @ (e | nil) by (r2)

rev2(e | l1)
→ sr2(e | l1,nil) by (r2)
→ sr2(l1,e | nil) by (r2-2)

Correctness of a Tail Recursive Reverse

Both sr2(l1,nil) @ (e | nil) and sr2(l1,e | nil) cannot be rewritten any more, and then we need a lemma. One possible candidate is as follows:

sr2(L1,E | nil) = sr2(L1,nil) @ (E | nil)

However, this seems too specific. Therefore, we make it more generic:

sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2) -- (p-sr2)

<u>sr2(l1,e | nil)</u>
→ sr2(l1,nil) @ (e | nil) by (p-sr2)

<u>End of Proof of Theorem 2</u>

Correctness of a Tail Recursive Reverse

<u>Lemma 1</u> [A property of sr2 (p-sr2)]
sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2)

<u>Proof of Lemma 1</u> By structural induction on L1.

Let e,e2 be fresh constants of Elt.E, l1,l2 be fresh constants of List.

I. Base case

What to show is sr2(nil,e2 | l2) = sr2(nil,nil) @ (e2 | l2).

<u>sr2(nil,e2 | l2)</u>
→ e2 | l2 by (sr2-1)

<u>sr2(nil,nil)</u> @ (e2 | l2)
→ <u>nil @ (e2 | l2)</u> by (sr2-1)
→ e2 | l2 by (@1)

Correctness of a Tail Recursive Reverse

II. Induction case
What to show is sr2(e | l1,e2 | l2) = sr2(e | l1,nil) @ (e2 | l2)
assuming the induction hypothesis
sr2(l1,E2 | L2) = sr2(l1,nil) @ (E2 | L2) -- (IH)

sr2(e | l1,e2 | l2)
→ sr2(l1,e | e2 | l2) by (sr2-2)
→ sr2(l1,nil) @ (e | e2 | l2) by (IH)

Correctness of a Tail Recursive Reverse

sr2(e | l1,nil) @ (e2 | l2)
→ sr2(l1,e | nil) @ (e2 | l2) by (sr2-2)
→ (sr2(l1,nil) @ (e | nil)) @ (e2 | l2) by (IH)
→ sr2(l1,nil) @ ((e | nil) @ (e2 | l2)) by (assoc@)
→ sr2(l1,nil) @ (e | (nil @ (e2 | l2)) by (@2)
→ sr2(l1,nil) @ (e | e2 | l2) by (@1)

End of Proof of Lemma 1

Correctness of a Tail Recursive Reverse

Lemma 1 [A property of sr2 (p-sr2)]
sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2)

Proof of Lemma 1 By structural induction on L1.

I. Base case

```
open LIST2 .
  -- fresh constants
  op l2 : -> List .
  op e2 : -> Elt.E .
  -- check
  red sr2(nil,e2 | l2) = sr2(nil,nil) @ (e2 | l2) .
close
```


Correctness of a Tail Recursive Reverse

II. Induction case

```
open LIST2 .
  -- fresh constants
  ops l1 l2 : -> List .
  ops e e2 : -> Elt.E .
  -- induction hypothesis
  eq sr2(l1,E2 | L2) = sr2(l1,nil) @ (E2 | L2) .
  -- check
  red sr2(e | l1,e2 | l2) = sr2(e | l1,nil) @ (e2 | l2) .
close
```

End of Proof of Lemma 1

Correctness of a Tail Recursive Reverse

Theorem 2 [Correctness of a tail recursive reverse (ctrr)]

rev1(L1) = rev2(L1)

Proof of Theorem 2 By structural induction on L1.

I. Base case

```
open LIST2 .
  -- check
  red rev1(nil) = rev2(nil) .
close
```


Correctness of a Tail Recursive Reverse

II. Induction case

```
open LIST2 .
  -- fresh constants
  op l1 : -> List .
  op e : -> Elt.E .
  -- induction hypothesis
  eq rev1(l1) = rev2(l1) .
  -- lemmas
  eq sr2(L1,E2 | L2) = sr2(L1,nil) @ (E2 | L2) .
  -- check
  red rev1(e | l1) = rev2(e | l1) .
close
```

End of Proof of Theorem 2

Exercises

1. Write the specifications and proof scores used in the slides and feed them to the CafeOBJ systems. Write all proofs used on the slides by hand as well.

2. Write manual proofs verifying that rev1(rev1((L)) equals L for all lists L, and write proof scores formally verifying that rev1(rev1((L)) equals L for all lists L.

3. Conjecture many things on lists that seem to be true and prove them.