Logic and Computation – CS 2800 Fall 2019

Lecture 28

Induction like a pro

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

Quiz

- The function below does not generate a valid induction scheme: YES/NO

```
(definec f (x :nat y :nat) :bool
  (and (> x 10) (< y 42)))
```

Quiz

- The functions below generate the same induction scheme: YES/NO

```
(definec f1 (x :tl y :tl) :tl
  (if (endp x)
      (if (endp y)
          nil
          (cons (car y) (f1 x (cdr y))))
      (cons (car x) (f1 (cdr x) y))))
```

```
(definec f2 (x :tl y :tl) :bool
  (cond ((and (endp x) (endp y)) t)
        ((endp x) (f2 x (cdr y)))
        ((endp y) (f2 (cdr x) y))
        (t (f2 (cdr x) (cdr y)))))
```

Quiz

- The functions below generate the same induction scheme: YES/NO

```
(definec f1 (x :tl y :tl) :tl
  (if (endp x)
      (if (endp y)
          nil
          (cons (car y) (f1 x (cdr y))))
      (cons (car x) (f1 (cdr x) y))))
```

```
(definec f3 (x :tl y :tl) :tl
  (cond ((and (endp x) (endp y)) nil)
        ((endp x) (f3 x (cdr y)))
        ((endp y) (f3 (cdr x) y))
        (t (cons (list x) (f3 (cdr x) y)))))
```

Outline

- Induction continued
- Induction like a pro
- Sorting a list

The “professional” method

- Start doing the equational proof
- Discover cases as you go along
  - Do book-keeping to make sure you cover all cases
- Discover and use induction hypotheses as you go along
  - Do book-keeping to make sure you remember that these are hypotheses
- Discover and use lemmas as you go along
  - Do book-keeping to make sure you remember all the lemmas that you have to prove later

Sorting a list

Insertion sort

```
(defdata lor (listof rational))


(definec insert (e :rational L :lor) :lor
  (cond ((endp L) (list e))
        ((<= e (car L)) (cons e L))
        (t (cons (car L) (insert e (cdr L))))))

(definec isort (L :lor) :lor
  (if (endp L)
      L
    (insert (car L) (isort (cdr L)))))
```

Sorted (“ordered”) lists

```
(defdata lor (listof rational))


(definec orderedp (L :lor) :bool
  (or (endp (cdr L))
      (and (<= (car L) (second L))
           (orderedp (cdr L)))))
```

Sorted ("ordered") lists

```
(defdata lor (listof rational))


(definec orderedp (L :lor) :bool
  (or (endp (cdr L))
      (and (<= (car L) (second L))
           (orderedp (cdr L)))))
```

- Claim: insertion sort produces an ordered list:

```
(defthm isort-ordered
  (implies (lorp L)
           (orderedp (isort L))))
```

- What induction scheme to use?

Induction on a list of rationals

```
(defdata lor (listof rational))

(definec lorind (L :lor) :bool
  (if (endp L) t
    (lorind (cdr L))))

(defthm isort-ordered
  (implies (lorp L)  (orderedp (isort L))))
```

- Proof obligations:

This is the same induction scheme as for true lists!

```
PO1: (not (lorp L)) => isort-ordered
PO2: (lorp L) & (endp L) => isort-ordered
PO3: (lorp L) & (not (endp L)) &
      isort-ordered|((L (cdr L))
                    => isort-ordered
```

Proof obligation 3 suggests a lemma

```
(defdata lor (listof rational))

(defthm isort-ordered
  (implies (lorp L)  (orderedp (isort L))))

PO3: (lorp L) & (not (endp L)) &
     isort-ordered|((L (cdr L))
                    => isort-ordered
```

- Proof of PO3:

```
C1. (lorp L)        C2. (not (endp L))
C3. (lorp (cdr L)) => (orderedp (isort (cdr L)))
D1. (lorp (cdr L)) { C1, C2 }
D2. (orderedp (isort (cdr L))) { C3, D1, MP }
Proof: (orderedp (isort L)) = { def isort, C2 }
(orderedp (insert (car L) (isort (cdr L)))) = ???
```

Proof obligation 3 suggests a lemma

```
(defdata lor (listof rational))
(defthm isort-ordered
  (implies (lorp L)  (orderedp (isort L))))

Lemma1: (lorp L) & (orderedp L) & (rationalp e)
      => (orderedp (insert e L))
```

- Proof of PO3:

```
C1. (lorp L)       C2. (not (endp L))
C3. (lorp (cdr L)) => (orderedp (isort (cdr L)))
D1. (lorp (cdr L)) { C1, C2 }
D2. (orderedp (isort (cdr L))) { C3, D1, MP }
D3. (rationalp (car L)) { C1, C2 }
D4. (orderedp (insert (car L) (isort (cdr L))))
      { D1, D2, D3, Lemma1, MP }
Proof: (orderedp (isort L)) = { def isort, C2 }
(orderedp (insert (car L) (isort (cdr L)))) = { D4 } t
```

Proving Lemma1

```
(defdata lor (listof rational))

Lemma1: (lorp L) & (orderedp L) & (rationalp e)
      => (orderedp (insert e L))
```

- Now we know that Lemma1 is helpful to complete the original proof.
- But we still need to prove the lemma!
- Which induction scheme to use for Lemma1?
- We can first try the scheme for true lists.
- Trying out to prove it we will realize that we need to consider the cases `(<= e (car L))` or not.
- The induction scheme for `insert` gives us those cases automatically!

Proving Lemma1 using the induction scheme of `insert`

```
(defdata lor (listof rational))
(definec insert (e :rational L :lor) :lor
  (cond ((endp L) (list e))
        ((<= e (car L)) (cons e L))
        (t (cons (car L) (insert e (cdr L))))))

Lemma1: (lorp L) & (orderedp L) & (rationalp e)
     => (orderedp (insert e L))
```

- Proof obligations:

```
PO1: (not (and (rationalp e) (lorp L))) => Lemma
PO2: (rationalp e) & (lorp L) & (endp L) => Lemma
PO3: (rationalp e) & (lorp L) & (not (endp L))
      & (<= e (car L)) => Lemma1
PO4: (rationalp e) & (lorp L) & (not (endp L))
      & (> e (car L)) & Lemma1|((L (cdr L)))
           => Lemma1
```

During the proof of Lemma1 we discover that we need new lemmas!

```
Lemma2: (lorp L) & (rationalp e)
            => (not (endp (insert e L)))

Lemma3: (lorp L) & (rationalp e) & (orderedp L)
        & (not (endp L)) & (e > (car L))
            =>
        (car L) < (car (insert e (cdr L)))
```

- Finish all these proofs at home!
  - All the lemmas, as well as the original claim
- Make sure all your proofs are formal (with context, derived context, etc)

Next time

- Induction continued