Logic and Computation – CS 2800
Fall 2019

Lecture 29

Induction continued

Stavros Tripakis

Northeastern University
Khoury College of
Computer Sciences

A quick lesson in political science

Political systems in a nutshell

- Anarchy: rule of none / gangs
  - Cannot protect human rights
- Monarchy: rule of one
  - Cannot protect human rights
- Oligarchy: rule of a few
  - Cannot protect human rights
- Democracy: rule of the majority
  - Can it protect human rights?

The exam saga

- Dec 2 is not possible: Registrar said no
- Registrar proposes Friday December 13th , 1-3pm in Shillman Hall 335

Outline

- Induction continued
- More examples, sorting a list
- Induction like a pro

Insertion sort functions

```
(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)))))

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

Quiz

```
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)))
```

- Do we need induction to prove the above lemmas?
    A. Both lemmas require induction
    B. Lemma3 requires induction but Lemma2 does not
    C. Lemma2 requires induction but Lemma3 does not
    D. None of them requires induction

Quiz

- A function `mysort` which satisfies the claim below is a correct sorting function: YES/NO

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

More proofs by induction

Claim 1

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

- A couple of lectures ago we started proving this:

```
(tlp a) & (tlp b) & (tlp c) =>
(aapp (aapp a b) c) = (aapp a (aapp b c))
```

- Let's complete the proof, doing PO3:

```
PO1: (not (tlp a)) => Phi
PO2: (tlp a) & (endp a) => Phi
PO3: (tlp a) & (not (endp a)) & Phi|((a (cdr a))) => Phi
```

Claim 2

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

- What if we tried to prove this?

```
(aapp (aapp x x) x) = (aapp x (aapp x x))
```

- First we must do contract completion:

```
                  (tlp x) =>
(aapp (aapp x x) x) = (aapp x (aapp x x))
```

- Can we prove it using just equational reasoning?
- No: length of list `x` is not bounded.
- Can we prove it using induction?
- Let's try induction on true list `x`.

Claim 2

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

- So how can we prove this?

```
(tlp x) =>
(aapp (aapp x x) x) = (aapp x (aapp x x))
```

- Simply by instantiating the previous theorem:

```
(tlp a) & (tlp b) & (tlp c) =>
(aapp (aapp a b) c) = (aapp a (aapp b c))
```

- **Generalization!**

Claim 3

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))

Claim: (tlp x) => (rrev (rrev x)) = x
```

- Which induction scheme to use?
- True lists!

```
PO1: (not (tlp x)) => Phi
PO2: (tlp x) & (endp x) => Phi
PO3: (tlp x) & (not (endp x)) & Phi|((x (cdr x))) => Phi
```

Claim 3

```
(definec rrev (x :tl) :tl
  (if (endp x)
    nil
    (aapp (rrev (rest x)) (list (first x)))))

Claim: (tlp x) => (rrev (rrev x)) = x
```

- Proof obligations:

```
PO1: (not (tlp x)) => ((tlp x) => (rrev (rrev x)) = x)
PO2: (tlp x) & (endp x) => ((tlp x) => (rrev (rrev x))=x)
PO3: (tlp x) & (not (endp x)) &
     ((tlp (cdr x)) => (rrev (rrev (cdr x))) = (cdr x))
                   => ((tlp x) => (rrev (rrev x)) = x)
```

Claim 3

- For proof obligation 3, some lemmas may be useful:

Prove these lemmas at home!

```
Lemma 1: (tlp a) & (tlp b)
      => (rrev (aapp a b)) = (aapp (rrev b) (rrev a))
Lemma 2: (rrev (list x)) = (list x)
Lemma 3: (tlp x) & (not (endp x))
      => (aapp (list (car x)) (cdr x)) = x
```

- Strategy for lemmas:
  1. Come up with the lemma during the proof
  2. Use the lemma to complete the proof: this ensures that the lemma is actually useful (i.e., allows us to complete the proof)
  3. Prove the lemma.

Claim 4

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))

Claim: (tlp x) => (subset x x)
```

- How to prove this?
- Induction? On what?
- Let's try true lists.

Do this at home!

Claim 4

Generalization!

But which one?

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

```
C1. (tlp x)
C2. (not (endp x))
C3. (tlp (cdr x)) => (subset (cdr x) (cdr x))
D1. (tlp (cdr x)) { C1, C2 }
D2. (subset (cdr x) (cdr x)) { C3, C1, MP }
Proof:
(subset x x) = {def subset, C2 }
(in (car x) x) & (subset (cdr x) x) = { L1 }
(subset (cdr x) x) = ???
```

```
Lemma L1:
(tlp x) & (not (endp x)) => (in (car x) x)
```

Claim 4

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

- We could try these:

They won't work!

```
Generalized claim attempt 1:
(tlp x) & (not (endp x)) => (subset (cdr x) x)

Generalized claim attempt 2:
(tlp x) & (subset x x) => (subset x (cons a x))

...
```

Claim 4

We need to distinguish the two arguments of subset!

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

- E.g., for the first claim, we will end up trying to prove:

```
Generalized claim attempt 1:
(tlp x) & (not (endp x)) => (subset (cdr x) x)

...
IH: (subset (cdr (cdr x)) (cdr x))
...
Proof:
... (subset (cdr (cdr x)) x) ...   ???
```

Claim 4

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

- The art of generalization:

```
Generalized claim attempt 3 (the right one):

(tlp x) & (tlp y) & (subset x y)
             => (subset x (cons a y))
```

Try these three claims at home in ACL2s, see what you get.

Claim 4

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

- Let’s try to prove this claim:

```
Generalized claim attempt 3 (the right one):

(tlp x) & (tlp y) & (subset x y)
           => (subset x (cons a y))
```

- Induction: which induction scheme?
- True list x

Claim 4: proof

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

```
C1. (tlp x)   C2. (tlp y)   C3. (subset x y)
C4. (not (endp x)) ...
Dnnn. (subset (cdr x) (cons a y)) { ... }
Dmmm. (in (car x) y) { ... }

Proof:
(subset x (cons a y)) = { def subset, C4 }
(in (car x) (cons a y)) & (subset (cdr x) (cons a y))
= { Dnnn }
(in (car x) (cons a y)) = { def in, cons axioms }
(car x) = a   or   (in (car x) y)
= { Dmmm, PL }
t
```

QED!

Claim 5:

```
(definec in (a :all X :tl) :bool
  (cond ((endp x) nil)
        ((equal a (car X)) t)
        (t (in a (cdr X)))))

(definec subset (x :tl y :tl) :bool
  (cond ((endp x) t)
        ((in (car x) y) (subset (cdr x) y))
        (t nil)))
```

- Subset is transitive:

```
(defthm transitive
  (implies (and (tlp x) (tlp y) (tlp z)
                (subset x y) (subset y z))
           (subset x z)))
```

Do this at home!

Next time

- Induction continued