

# CS313K: Logic, Sets, and Functions

J Strother Moore  
Department of Computer Sciences  
University of Texas at Austin

(Lecture 17)

# Announcements

Some TAs and Tutors have adjusted their office hours; consult the web page.

Today I'm going to wrap up induction and move to quantifiers.

# About Induction

You've seen Structural Induction:  $(\phi \ x)$  is a theorem if BC and IS are theorems:

$$\text{BC: } \neg(\text{consp } x) \rightarrow (\phi \ x)$$

$$\text{IS: } (\text{consp } x) \wedge (\phi \ (\text{cdr } x)) \rightarrow (\phi \ x)$$

# About Induction

You've seen Structural Induction:  $(\phi \ x \ a)$  is a theorem if BC and IS are theorems:

BC:  $\neg(\text{consp } x) \rightarrow (\phi \ x \ a)$

IS:  $(\text{consp } x)$   
 $\wedge (\phi \ (\text{cdr } (\text{cdr } x)) \ a_1)$   
 $\wedge (\phi \ (\text{car } x) \ a_2)$   
 $\rightarrow$   
 $(\phi \ x \ a)$

```
(defun len (x)
  (if(consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if(consp x)
      (len1 (cdr x) (+ 1 a))
      a))
```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

```

(defun len (x)
  (if(consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if(consp x)
      (len1 (cdr x) (+ 1 a))
      a))

```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

Proof: Induct on  $x$ .

BC:  $\neg(\text{consp } x) \rightarrow a + (\text{len } x) = (\text{len1 } x \ a)$

```

(defun len (x)
  (if (consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if (consp x)
      (len1 (cdr x) (+ 1 a))
      a))

```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

Proof: Induct on  $x$ .

BC:  $\neg(\text{consp } x) \rightarrow a + 0 = (\text{len1 } x \ a)$

```

(defun len (x)
  (if (consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if (consp x)
      (len1 (cdr x) (+ 1 a))
      a))

```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

Proof: Induct on  $x$ .

BC:  $\neg(\text{consp } x) \rightarrow a + 0 = a$

```

(defun len (x)
  (if (consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if (consp x)
      (len1 (cdr x) (+ 1 a))
      a))

```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

Proof: Induct on  $x$ .

IS:  $(\text{consp } x)$

$\wedge$

$(+ 1 \ a) + (\text{len } (\text{cdr } x)) = (\text{len1 } (\text{cdr } x) (+ 1 \ a))$

$\rightarrow$

$a + (\text{len } x) = (\text{len1 } x \ a)$ .

(defun len (x)	(defun len1 (x a)
(if(consp x)	(if(consp x)
(+ 1 (len (cdr x))))	(len1 (cdr x)(+ 1 a)
0))	a))

Theorem:  $a+(\text{len } x) = (\text{len1 } x \text{ } a)$ .

Proof: Induct on  $x$ .

IS: (consp  $x$ )

$\wedge$

$(+ 1 a)+(\text{len } (\text{cdr } x))=(\text{len1 } (\text{cdr } x)(+ 1 a))$

$\rightarrow$

$a+(+ 1 (\text{len } (\text{cdr } x)))=(\text{len1 } x \text{ } a)$ .

(defun len (x)	(defun len1 (x a)
(if(consp x)	(if(consp x)
(+ 1 (len (cdr x)))	(len1 (cdr x) (+ 1 a)
0))	a))

Theorem:  $a + (\text{len } x) = (\text{len1 } x \text{ } a)$ .

Proof: Induct on  $x$ .

IS: (consp  $x$ )

$\wedge$

$(+ 1 a) + (\text{len } (\text{cdr } x)) = (\text{len1 } (\text{cdr } x) (+ 1 a))$

$\rightarrow$

$a + (+ 1 (\text{len } (\text{cdr } x))) = (\text{len1 } (\text{cdr } x) (+ 1 a))$

```

(defun len (x)
  (if (consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if (consp x)
      (len1 (cdr x) (+ 1 a))
      a))

```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

Proof: Induct on  $x$ .

IS:  $(\text{consp } x)$

$\wedge$

$(+ 1 \ a) + (\text{len } (\text{cdr } x)) = (\text{len1 } (\text{cdr } x) (+ 1 \ a))$

$\rightarrow$

$a + (+ 1 \ (\text{len } (\text{cdr } x))) = (\text{len1 } (\text{cdr } x) (+ 1 \ a))$

```

(defun len (x)
  (if (consp x)
      (+ 1 (len (cdr x)))
      0))

(defun len1 (x a)
  (if (consp x)
      (len1 (cdr x) (+ 1 a))
      a))

```

Theorem:  $a + (\text{len } x) = (\text{len1 } x \ a)$ .

Proof: Induct on  $x$ .

IS:  $(\text{consp } x)$

$\wedge$

$(+ \ 1 \ a) + (\text{len } (\text{cdr } x)) = (\text{len1 } (\text{cdr } x) (+ \ 1 \ a))$

$\rightarrow$

$a + (+ \ 1 \ (\text{len } (\text{cdr } x))) = (+ \ 1 \ a) + (\text{len } (\text{cdr } x))$

# About Induction

Structural Induction is just the tip of the iceberg.

To prove  $\phi$  by induction on  $x$  you can set up an arbitrary case split (“BC” and “IS”) as long as they are mutually exclusive and exhaustive. In the IS you can assume  $\phi$  for *anything* that is smaller.

# About Induction

$$\text{BC: } (z_p \ n) \rightarrow (\psi \ n)$$

$$\text{IS: } \neg(z_p \ n) \wedge (\psi \ n-1) \rightarrow (\psi \ n)$$

# About Induction

$$\text{BC1: } (z_p \ n) \rightarrow (\psi \ n)$$

$$\text{BC2: } n=1 \rightarrow (\psi \ n)$$

$$\text{IS: } \neg(z_p \ n) \wedge n \neq 1 \wedge (\psi \ n-2) \rightarrow (\psi \ n)$$

# About Induction

BC1:  $(zp\ n) \rightarrow (\psi\ n)$

BC2:  $n=1 \rightarrow (\psi\ n)$

IS:  $\neg(zp\ n) \wedge n \neq 1 \wedge (\psi\ n-2) \rightarrow (\psi\ n)$

```
(defun evenp (n)
  (if (zp n)
      t
      (if (equal n 1)
          nil
          (evenp (- n 2))))))
```

# About Induction

```
(defun evenp (n)
  (if (zp n) t
      (if (equal n 1) nil
          (evenp (- n 2))))))
```

Theorem:

$$(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$$

# About Induction

```
(defun evenp (n)
  (if (zp n) t
      (if (equal n 1) nil
          (evenp (- n 2))))))
```

Theorem:

$$(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$$

BC1:  $(\text{evenp } 0) \wedge (\text{evenp } j) \rightarrow (\text{evenp } 0+j)$

# About Induction

```
(defun evenp (n)
  (if (zp n) t
      (if (equal n 1) nil
          (evenp (- n 2))))))
```

Theorem:

$$(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$$

BC2:  $(\text{evenp } 1) \wedge (\text{evenp } j) \rightarrow (\text{evenp } 1+j)$

# About Induction

```
(defun evenp (n)
  (if (zp n) t
      (if (equal n 1) nil
          (evenp (- n 2))))))
```

Theorem:

$$(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$$

IS:  $\neg(\text{zp } i) \wedge i \neq 1$   
 $\wedge [(\text{evenp } i-2) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i-2+j)]$   
 $\rightarrow$   
 $(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$

# About Induction

```
(defun evenp (n)
  (if (zp n) t
      (if (equal n 1) nil
          (evenp (- n 2))))))
```

Theorem:

$$(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$$

IS:  $\neg(\text{zp } i) \wedge i \neq 1$   
 $\wedge [(\text{evenp } i-2) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i-2+j)]$   
 $\rightarrow$   
 $(\text{evenp } i-2) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$

# About Induction

```
(defun evenp (n)
  (if (zp n) t
      (if (equal n 1) nil
          (evenp (- n 2))))))
```

Theorem:

$$(\text{evenp } i) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i+j)$$

IS:  $\neg(\text{zp } i) \wedge i \neq 1$

$$\wedge [(\text{evenp } i-2) \wedge (\text{evenp } j) \rightarrow (\text{evenp } i-2+j)]$$

$\rightarrow$

$$(\text{evenp } i-2) \wedge (\text{evenp } j) \rightarrow (\text{evenp } (i+j)-2)$$

# About Induction

```
(defun qsort (x)
  (cond ((endp x) nil)
        ((endp (cdr x)) (list (car x)))
        (t (app (qsort (all (cdr x) '< (car x)))
                 (cons (car x)
                       (qsort (all (cdr x) '>= (car x))))))
```

# About Induction

```
(defun qsort (x)
  (cond ((endp x) nil)
        ((endp (cdr x)) (list (car x)))
        (t (app (qsort (all (cdr x) '< (car x)))
                 (cons (car x)
                       (qsort (all (cdr x) '>= (car x))))))
```

Theorem: (ordp (qsort x))

IS:  $\neg(\text{endp } x) \wedge \neg(\text{endp } (\text{cdr } x))$

$\wedge (\text{ordp } (\text{qsort } (\text{all } (\text{cdr } x) '< (\text{car } x))))$

$\wedge (\text{ordp } (\text{qsort } (\text{all } (\text{cdr } x) '>= (\text{car } x))))$

→

(ordp (qsort x))

# About Induction

Induction is probably the most often-used (and seldom mentioned) proof technique computer science. We use it constantly (and often implicitly) to reason about iterative and recursive methods.

Learn how to use it.

## Correction (top of page 109 of notes):

- define

```
(defun f (lst  $x_1$  ...  $x_n$ )  
  (if (endp lst)  
      t  
      (or (let (( $v$  (car lst)))  $\phi$ )  
          (f (cdr lst)  $x_1$  ...  $x_n$ ))))
```

Then  $(\forall v : \phi)$  is equivalent to  $(f \mathcal{D})$ .

## Correction (top of page 109 of notes):

- define

```
(defun f (lst  $x_1 \dots x_n$ )
  (if (endp lst)
      t
      (or (let (( $v$  (car lst)))  $\phi$ )
          (f (cdr lst)  $x_1 \dots x_n$ ))))
```

Then  $(\forall v : \phi)$  is equivalent to  $(f \mathcal{D})$ .

## Correction (top of page 109 of notes):

- define

```
(defun f (lst  $x_1 \dots x_n$ )  
  (if (endp lst)  
      nil  
      (or (let (( $v$  (car lst)))  $\phi$ )  
          (f (cdr lst)  $x_1 \dots x_n$ ))))
```

Then  $(\exists v : \phi)$  is equivalent to  $(f \mathcal{D})$ .