CS313K: Logic, Sets, and Functions

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

Lecture 19 – Chaps 6, 7 (7.1, 7.2, 7.3)

Announcement

The course speeds up.

Elementary Arithmetic Waiver: We'll assume without proof any truth of ACL2 arithmetic. E.g.,

$$(\texttt{natp}(i) \ \wedge \ \texttt{natp}(j) \ \wedge \ \texttt{natp}(k)) \ \rightarrow \ i^{j+k} \ = \ i^j \ \times \ i^k$$

Today we will move on quantifiers ($\forall$ and $\exists$) and then Set Theory.

But before we start on Chapter 7, I want to give an important mini-lecture on the more general treatment of induction.

Induction may be the most important proof technique you ever learn.

The treatment we've see so far is a Very Special Case.

Quiz 19.0 (30 seconds) Press A.

Induction on x to Prove (ϕ x y)

Base:

(endp x) $\longrightarrow$ (ϕ x y).

Induction Step:

(($\neg$(endp x))

 $\wedge$

 (ϕ (rest x) α_1)

 $\wedge$

 (ϕ (rest x) α_2)

 ...)

$\longrightarrow$

 (ϕ x y)

Why is (ϕ '(a b c) 43) true?

B: (endp x) $\longrightarrow$ (ϕ x y)

I: ((¬(endp x)) $\wedge$ (ϕ (rest x) (α x y))) $\longrightarrow$ (ϕ x y)

(ϕ '() (α '(c) (α '(b c) (α '(a b c) 43)))) {by B}

$\longrightarrow$ {by I}

(ϕ '(c) (α '(b c) (α '(a b c) 43)))

$\longrightarrow$ {by I}

(ϕ '(b c) (α '(a b c) 43))

$\longrightarrow$ {by I}

(ϕ '(a b c) 43)

$\square$

Key Idea: A finite chain of I's down to B.

Why is (ϕ '(a b c) 43) true?

B: (endp x) $\longrightarrow$ (ϕ x y)

I: (($\neg$(endp x)) $\wedge$ (ϕ (rest x) (α x y))) $\longrightarrow$ (ϕ x y)

(ϕ '() (α '(c) (α '(b c) (α '(a b c) 43)))) {by B}

$\longrightarrow$ {by I}

(ϕ '(c) (α '(b c) (α '(a b c) 43)))

$\longrightarrow$ {by I}

(ϕ '(b c) (α '(a b c) 43))

$\longrightarrow$ {by I}

(ϕ '(a b c) 43)

$\square$

Key Idea: A finite chain of I's down to B.

Why is (ϕ '(a b c) 43) true?

B: (endp x) $\longrightarrow$ (ϕ x y)

I: ((¬(endp x)) $\wedge$ (ϕ (rest x) (α x y))) $\longrightarrow$ (ϕ x y)

(ϕ '() (α '(c) (α '(b c) (α '(a b c) 43)))) {by B}

$\longrightarrow$ {by I}

(ϕ '(c) (α '(b c) (α '(a b c) 43)))

$\longrightarrow$ {by I}

(ϕ '(b c) (α '(a b c) 43))

$\longrightarrow$ {by I}

(ϕ '(a b c) 43)

$\square$

Key Idea: A finite chain of I's down to B.

Why is (ϕ '(a b c) 43) true?

B: (endp x) $\longrightarrow$ (ϕ x y)

I: ((¬(endp x)) $\wedge$ (ϕ (rest x) (α x y))) $\longrightarrow$ (ϕ x y)

(ϕ '() (α '(c) (α '(b c) (α '(a b c) 43)))) {by B}

$\longrightarrow$ {by I}

(ϕ '(c) (α '(b c) (α '(a b c) 43)))

$\longrightarrow$ {by I}

(ϕ '(b c) (α '(a b c) 43))

$\longrightarrow$ {by I}

(ϕ '(a b c) 43)

$\square$

Key Idea: A finite chain of I's down to B.

Why is (ϕ '(a b c) 43) true?

B: (endp x) $\rightarrow$ (ϕ x y)

I: (($\neg$(endp x)) $\wedge$ (ϕ (rest x) (α x y))) $\rightarrow$ (ϕ x y)

(ϕ '() (α '(c) (α '(b c) (α '(a b c) 43)))) {by B}

$\rightarrow$ {by I}

(ϕ '(c) (α '(b c) (α '(a b c) 43)))

$\rightarrow$ {by I}

(ϕ '(b c) (α '(a b c) 43))

$\rightarrow$ {by I}

(ϕ '(a b c) 43)

$\square$

Key Idea: A finite chain of I's down to B.

Well-Foundedness

A relation, $\prec$, is *well-founded* (on some domain) if there are no infinitely descending chains of objects (in that domain).

That is, this can't go on forever:

$$\ldots \prec x_3 \prec x_2 \prec x_1 \prec x_0$$

This is equivalent to: every non-empty subset of the domain has a minimal element.

Example: Less Than “$<$”

This can't go on forever if all the x_i are natural numbers:

$$\ldots < x_3 < x_2 < x_1 < x_0$$

So $<$ is well-founded on the naturals.

(There are many interesting well-founded relations on things besides natural numbers, but we won't discuss them. We'll always use $<$ on the naturals for $\prec$.)

Measures

We say m is a *measure* if it is a function that returns an element of a well-founded domain.

Example: `len` is a measure. It returns a natural number.

Example: `cons-count` is a measure. It returns a natural number.

Well-Foundedness and Recursion

Suppose $\prec$ is well-founded and you have a measure m for its domain. Suppose you have a recursive definition:

$$
\begin{array}{l}
\texttt{(defun}\ f\ \ (v_1\, v_2\ \ldots\ v_n) \\
\quad \texttt{(if}\ \theta \\
\qquad\quad \texttt{(}\ldots\ \ (f\ \ v_1\, v_2\ \ldots\ v_n)_{/\sigma_1} \\
\qquad\qquad\qquad \ldots \\
\qquad\qquad\qquad (f\ \ v_1\, v_2\ \ldots\ v_n)_{/\sigma_k}\texttt{)} \\
\qquad\quad \text{...no recursive calls...}\texttt{))}
\end{array}
$$

For Example

```
(defun rev1 (x a)
  (if (endp x)
      a
      (rev1 (rest x) (cons (first x) a))))
```

For Example

```
(defun rev1 (x a)
  (if θ
      (rev1 x a)/σ1
      a))
```

where

$\theta = (\neg\ \texttt{(endp x))}$

$\sigma_1 = \{\texttt{x} \leftarrow \texttt{(rest x)}, \texttt{a} \leftarrow \texttt{(cons (first x) a)}\}$

Well-Foundedness and Recursion

```
(defun f (v1 v2 ... vn)
  (if θ
      (... (f v1 v2 ... vn)/σ1
           ...
           (f v1 v2 ... vn)/σk)
      ...no recursive calls...))
```

Suppose it is a theorem that:

$$\theta \rightarrow (m\ v_1\, v_2\ \ldots\ v_n)_{/\sigma_i} \prec (m\ v_1\, v_2\ \ldots\ v_n),$$

then f always terminates.

```
(defun rev1 (x a)
  (if (endp x)
      y
      (rev1 (rest x) (cons (first x) a))))
```

Let `(m x a)` $=$ `(cons-count x)`.

$\sigma_1 = \{$ `x` $\leftarrow$ `(rest x)`, `a` $\leftarrow$ `(cons (first x) a)`$\}$.

Theorem?

$(\neg$ `(endp x)`$)$

$\rightarrow$

`(m x a)`$_{/\sigma_1}$

$<$

`(m x a)`

```
(defun rev1 (x a)
  (if (endp x)
      y
      (rev1 (rest x) (cons (first x) a))))
```

Let (m x a) = (cons-count x).

σ_1 = { x ← (rest x), a ← (cons (first x) a)}.

Theorem?

(¬ (endp x))

→

(m (rest x) (cons (first x) a))

<

(m x a)

```
(defun rev1 (x a)
  (if (endp x)
      y
      (rev1 (rest x) (cons (first x) a))))
```

Let `(m x a)` $=$ `(cons-count x)`.

σ_1 $=$ $\{$ `x` $\leftarrow$ `(rest x)`, `a` $\leftarrow$ `(cons (first x) a)`$\}$.

Theorem:

`(`$\neg$ `(endp x))`

$\longrightarrow$

`(cons-count (rest x))`

$<$

`(cons-count x)`

So `rev1` terminates.

```
(defun treecopy (x)
  (if (consp x)
      (cons (treecopy (first x))
            (treecopy (rest x)))
      x))
```

Let `(m x)` = `(cons-count x)`

Theorems:

`(consp x)` → `(m (first x))` < `(m x)`

`(consp x)` → `(m (rest x))` < `(m x)`

So `treecopy` terminates.

```
(defun up (x a)
  (if (and (natp x) (natp a) (< x a))
      (up (+ 1 x) a)
      a))
```

Let (m x a) $=$ |a $-$ x|

Theorem:

((natp a) $\wedge$ (natp x) $\wedge$ (< x a))

$\rightarrow$

(m (+ 1 x) a)

$<$

(m x a)

So up terminates.

```
(defun qsort (x)  ; Quick Sort
 (if (endp x)
     nil
     (if (endp (rest x))
         x
         (app
          (qsort (all-smaller (first x) (rest x)))
          (cons (first x)
                (qsort (all-others (first x) (rest x))))
```

where `(all-smaller e z)` is the list of all elements of `z` that are smaller than `e` and `(all-others e z)` is the list of all the other elements.

```
(defun qsort (x)  ; Quick Sort
 (if (endp x)
     nil
     (if (endp (rest x))
         x
         (app
          (qsort (all-smaller (first x) (rest x)))
          (cons (first x)
                (qsort (all-others (first x) (rest x))))))))
```

Let (m x) = (len x)

Theorems:

((¬(endp x)) ∧ (¬(endp (rest x))))

→

(m (all-smaller (first x) (rest x))) < (m x)

```
(defun qsort (x)  ; Quick Sort
 (if (endp x)
     nil
     (if (endp (rest x))
         x
         (app
          (qsort (all-smaller (first x) (rest x)))
          (cons (first x)
                (qsort (all-others (first x) (rest x))))))))
```

Let (m x) $=$ (len x)

Theorems:

((¬(endp x)) $\wedge$ (¬(endp (rest x))))

$\rightarrow$

(m (all-smaller (first x) (rest x))) $<$ (m x)

```
(defun qsort (x)  ; Quick Sort
 (if (endp x)
     nil
     (if (endp (rest x))
         x
         (app
          (qsort (all-smaller (first x) (rest x)))
          (cons (first x)
                (qsort (all-others (first x) (rest x))))))))
```

Let (m x) $=$ (len x)

Theorems:

((¬(endp x)) ∧ (¬(endp (rest x))))

$\longrightarrow$

(m (all-others (first x) (rest x))) $<$ (m x)

```
(defun qsort (x)  ; Quick Sort
 (if (endp x)
     nil
     (if (endp (rest x))
         x
         (app
          (qsort (all-smaller (first x) (rest x)))
          (cons (first x)
                (qsort (all-others (first x) (rest x))))))))
```

Let (m x) = (len x)

Theorems:

...

So qsort terminates.

Well-Foundedness and Recursion

```
(defun f (v1 v2 ... vn)
  (if θ
      (... (f v1 v2 ... vn)/σ1
           ...
           (f v1 v2 ... vn)/σk)
      ...no recursive calls...))
```

$$(\texttt{defun}\ f\ \ (v_1\, v_2\ \ldots\ v_n)$$
$$\quad(\texttt{if}\ \theta$$
$$\qquad(\ldots\ \ (f\ \ v_1\, v_2\ \ldots\ v_n)_{/\sigma_1}$$
$$\qquad\qquad\ldots$$
$$\qquad\qquad(f\ \ v_1\, v_2\ \ldots\ v_n)_{/\sigma_k})$$
$$\qquad\text{...no recursive calls...}))$$

Suppose it is a theorem that:

$$\theta \rightarrow (m\ \ v_1\, v_2\ \ldots\ v_n)_{/\sigma_i} \prec (m\ \ v_1\, v_2\ \ldots\ v_n),$$

then f always terminates.

Well-Foundedness and Induction

Suppose we have a set of substitutions σ_i such that

$$\theta \rightarrow (\texttt{m}\ v_1\ v_2\ \ldots\ v_n)_{/\sigma_i} \prec (\texttt{m}\ v_1\ v_2\ \ldots\ v_n)$$

Well-Foundedness and Induction

Suppose we have a set of substitutions σ_i such that

$\theta \rightarrow (\mathtt{m}\ \ v_1\, v_2\ \ \ldots\ \ v_n)_{/\sigma_i} \prec (\mathtt{m}\ \ v_1\, v_2\ \ \ldots\ \ v_n)$

then to prove ϕ prove:

Base:
$\neg\theta \rightarrow \phi$

Induction Step:
$(\theta \wedge \phi_{/\sigma_1} \wedge \ldots \wedge \phi_{/\sigma_k}) \rightarrow \phi$

```
(defun qsort (x)  ; Quick Sort
  (if (endp x)
      nil
      (if (endp (rest x))
          x
          (app
           (qsort (all-smaller (first x) (rest x)))
           (cons (first x)
                 (qsort (all-others (first x) (rest x))))
```

Note that

(a) Given $(m\ \texttt{x}) = \texttt{(len x)}$

(b) $\theta = ((\neg\texttt{(endp x)}) \wedge (\neg\texttt{(endp (rest x))}))$

(c) $\sigma_1 = \{\texttt{x} \leftarrow \texttt{(all-smaller e (rest x))}\}$

(d) $\sigma_2 = \{\texttt{x} \leftarrow \texttt{(all-others e (rest x))}\}$

(e) Theorems: $\theta \rightarrow (m\ \texttt{x})_{/\sigma_i} \prec (m\ \texttt{x})$, $i = 1, 2$

Then a legal induction to prove `(ordp (qsort x))` is:

Base Case:
$(\neg\theta) \rightarrow$ `(ordp (qsort x))`.

Induction Step:

```
(θ
 ∧ (ordp (qsort (all-smaller (first x) (rest x))))
 ∧ (ordp (qsort (all-others (first x) (rest x))))
 )
→
(ordp (qsort x)).
```

Theorem: `(ordp (qsort x))`

If `x` has fewer than 2 things in it, it's obvious.

Otherwise, let e be the `first` element of `x` and let $(a_1\ a_2 \ldots)$ and $(b_1\ b_2 \ldots)$ be the values of the two recursive calls of `qsort`.

By induction, $(a_1\ a_2 \ldots)$ and $(b_1\ b_2 \ldots)$ are ordered. But $a_i < e$ and $e \leq b_i$.

Obviously, $(a_1\ a_2 \ldots\ e\ b_1\ b_2 \ldots)$ is ordered. $\Box$

Key Lemmas

`(qsort x)` returns a list with the same elements in it as `x`.

all elements of `(all-smaller e x)` are smaller than `e`.

all elements of `(all-others e x)` are not smaller than `e`.

if `A` and `B` are ordered and everything in `A` is less than everything in `B`, then `(app A B)` is ordered.

Summary

When proving ϕ by induction you may assume ϕ for arbitrary smaller objects. You get to make up what "smaller" means, but it must be well-founded.

You will see many informal inductive proofs in CS.