Lecture 4

Pete Manolios
Northeastern

Equality

- Equality (equal, or =) is an *equivalence relation*
  - Reflexivity: $x = x$
  - Symmetry of Equality: $x = y \Rightarrow y = x$
  - Transitivity of Equality: $x = y \wedge y = z \Rightarrow x = z$
- Equality Axiom Schema for Functions: For every function symbol *f* of arity *n* we have the axiom
- $x_1 = y_1 \wedge \ldots \wedge x_n = y_n \Rightarrow (f\ x_1 \ldots x_n) = (f\ y_1 \ldots y_n)$
- In ACL2, we would write `(len (cons x z)) = (len (cons y z))` as

```
(equal (len (cons x z))
       (len (cons y z)))
```

- = and ≠ bind more tightly than any of the propositional operators

Built-in Functions

- Axioms for built-in functions, such as `cons`, `car`, and `cdr`
- Axioms are theorems we get for “free” characterizing `cons`, `car`, `cdr`, `consp`, `if`, `equal`, etc.
  - `(car (cons x y)) = x`
  - `(cdr (cons x y)) = y`
  - `(consp (cons x y)) = t`
  - `x = nil ⇒  (if x y z) = z`
  - `x ≠ nil ⇒ (if x y z) = y`
- Reason about constant expressions using evaluation
  - `t ≠ nil, (cons 1 ()) = (list 1), 3/9 = 1/3, ( ) = ’nil, …`
- Note: from the the semantics of the built-in functions

Built-in Functions

- Propositional Logic
  - (not p) = (if p nil t)
  - (implies p q) = (if p (if q t nil) t)
  - (iff p q) = (if p (if q t nil) (if q nil t))
- By embedding propositional calculus and = in term language, terms (τ) can be interpreted as formulas ($\tau \neq$ nil)
  - e.g., x as a formula is x $\neq$ nil
  - (foo x y z) as a formula is (foo x y z)$\neq$ nil
- Similarly, we add axioms for numbers, strings, etc.
- This is all in GZ, the "ground-zero theory"

Instantiation

- A substitution σ is a list of the form ((var_1 $term_1$) … (var_n $term_n$))
  - the vars are the "targets" (no repetitions) and the terms are their "images"
  - by f|σ we mean, substitute every free occurrence of a target by its image
  - `(cons x (let ((y z)) y))|((x a) (y b) (z c) (w d)) =`
    `(cons a (let ((y c)) y))`
- Instantiation: If f is a *theorem*, so is f|σ
  - `(len (list x)) = 1` is theorem, so is `(len (list (list x y))) = 1`
- `Are the following substitutions correct?`
- `(cons 'a b)|((a (cons a (list c))) (b (cons c nil)))`
  - `(cons 'a (cons c nil))`
- `(cons x (f x y f))|((x (cons a b)) (f x) (y (app y x)))`
  - `(cons (cons a b) (f (cons a b) (app y x) x))`

Inference Rules

- Evaluation
- Propositional calculus validities
  - Includes exportation, Modus Ponens, Proof by contradiction, …
- Equality axioms
  - equality is an equivalence relation, equality schema for functions
- Instantiation
  - Start with built-in axioms
  - New axioms are added via definitional principle
  - Also defaxiom, defchoose, encapsulation, etc can add axioms

How to Prove Theorems

- Once you are done with contract checking, completion & generalization
- Extract the context by rewriting the conjecture into the form:
  $[C1 \wedge C2 \wedge \ldots \wedge Cn] \Rightarrow$ RHS where there are as many hyps as possible
- Derived context. What obvious things follow? Common patterns:
  - `(endp x), (true-listp x): x=nil`
  - `(true-listp x), (consp x): (true-listp (rest x))`
  - $\phi_1 \wedge \ldots \wedge \phi_n \Rightarrow \psi$: Derive $\phi_1,\ldots,\phi_n$ and use MP to ψ
- Proof. Use the proof format from RAP.
  - For equality, start with LHS/RHS and end with RHS/LHS or start w/ LHS & reduce, then start w/ RHS & reduce to the same thing
  - For transitive relation ($\Rightarrow$, $<$, $\leq$, …) same proof format works
  - For anything else reduce to t

Equational Reasoning

```
(implies (and (true-listp x)
              (true-listp y))
         (implies (and (consp x)
                       (not (equal a (first x)))
                       (implies (true-listp (rest x))
                                (implies (in a (rest x))
                                         (in a (app (rest x) y)))))
                  (implies (in a x)
                           (in a (app x y)))))
```

Contract completion adds hypotheses.

ER Example

```
(implies (and (true-listp x)
              (true-listp y))
         (implies (and (consp x)
                       (not (equal a (first x)))
                       (implies (true-listp (rest x))
                                (implies (in a (rest x))
                                         (in a (app (rest x) y)))))
                  (implies (in a x)
                           (in a (app x y)))))
```

Next: Prepare context

ER Example

```
(implies (and (true-listp x)                              A
              (true-listp y))
         (implies (and (consp x)
                       (not (equal a (first x)))
                       (implies (true-listp (rest x))     B
                                (implies (in a (rest x))
                                         (in a (app (rest x) y)))))
                  (implies (in a x)
                           (in a (app x y)))))            C
```

Exportation: $A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$

ER Example

```
(implies (and (true-listp x)              A
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (true-listp (rest x))        B
                       (implies (in a (rest x))
                                (in a (app (rest x) y)))))
         (implies (in a x)
                  (in a (app x y)))))     C
```

Exportation: $A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (true-listp (rest x))
                       (implies (in a (rest x))
                                (in a (app (rest x) y)))))
         (implies (in a x)
                  (in a (app x y)))))
```

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (true-listp (rest x))
                       (implies (in a (rest x))
                                (in a (app (rest x) y)))))
         (implies (in a x)
                  (in a (app x y)))))
```

A

B

C

Exportation again: $A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (true-listp (rest x))
                       (implies (in a (rest x))
                                (in a (app (rest x) y)))))
              (in a x))
         (in a (app x y)))))
```

A

B

C

Exportation again: $A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (true-listp (rest x))
                       (implies (in a (rest x))
                                (in a (app (rest x) y))))
              (in a x))
         (in a (app x y))))
```

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (true-listp (rest x))  A
                       (implies (in a (rest x))  B
                                (in a (app (rest x) y)))))  C
              (in a x))
         (in a (app x y)))))
```

Exportation again: $A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (and (true-listp (rest x))   A
                            (in a (rest x)))        B
                       (in a (app (rest x) y)))     C
              (in a x))
         (in a (app x y)))))
```

Exportation again: $A \Rightarrow (B \Rightarrow C) \equiv (A \wedge B) \Rightarrow C$

ER Example

```
(implies (and (true-listp x)
              (true-listp y)
              (consp x)
              (not (equal a (first x)))
              (implies (and (true-listp (rest x))
                            (in a (rest x)))
                       (in a (app (rest x) y)))
              (in a x))
         (in a (app x y))))
```

Notice that we cannot use exportation in the 5th hypothesis

ER Example

```
(defunc in (a X)
  :input-contract (true-listp x)
  :output-contract (booleanp (in a X))
  (if (endp x)
      nil
    (or (equal a (first X))
        (in a (rest X)))))
```

```
(implies
  (and (true-listp x)
       (true-listp y)
       (consp x)
       (not (equal a (first x)))
       (implies (and (true-listp (rest x))
                     (in a (rest x)))
                (in a (app (rest x) y)))
       (in a x))
  (in a (app x y))))
```

```
C1. (true-listp x)
C2. (true-listp y)
C3. (consp x)
C4. a ≠ (first x)
C5. (true-listp (rest x)) ∧ (in a (rest x))
      ⇒ (in a (app (rest x) y))
C6. (in a x)
-----------------------------------------------
C7. (true-listp (rest x)) { C1, Def true-listp, C3 }
C8. (in a (rest x))        { C6, Def in, C3, C4, PL }
C9. (in a (app (rest x) y)) { C5, C7, C8, MP }
```

```
(defunc true-listp (l)
  :ic t
  :oc (booleanp (true-listp l))
  (if (consp l)
      (true-listp (rest l))
    (equal l () )))
```

ER Example

```
C1. (true-listp x)
C2. (true-listp y)
C3. (consp x)
C4. a ≠ (first x)
C5. (true-listp (rest x)) ∧ (in a (rest x))
      ⇒ (in a (app (rest x) y))
C6. (in a x)
-----------------------------------------------
C7. (true-listp (rest x)) { C1, Def true-listp, C3 }
C8. (in a (rest x))  { C6, Def in, C3, C4, PL }
C9. (in a (app (rest x) y)) { C5, C7, C8, MP }

  (in a (app x y))
=   { Def app }
  (in a (cons (first x) (app (rest x) y)))
=   { Def in, first-rest-cons axioms }
  (or (equal a (first x)) (in a (app (rest x) y)))
=   { C9, PL }
  t
```

```
(defunc in (a X)
  :input-contract (true-listp x)
  :output-contract (booleanp (in a X))
  (if (endp x)
      nil
    (or (equal a (first X))
        (in a (rest X)))))
```

```
(defunc true-listp (l)
  :ic t
  :oc (booleanp (true-listp l))
  (if (consp l)
      (true-listp (rest l))
     (equal l () )))
```

Induction Schemes

```
(defunc f (x1 . . . xn)
  :input-contract ic
  :output-contract oc
  (cond (t1 c1)
        (t2 c2)
        ...
        (tm cm)
        (t cm+1)))
```

- Given a function definition of the form:
- If c_i contains a call to f, we say it is a recursive case
  - else it is a base case
- Let t_{m+1} be t.
- Let $Case_i$ be $t_i \wedge \neg t_j$ for all $j < i$
- The function f gives rise to the following induction scheme:
- To prove ϕ, you can instead prove
  - 1. $\neg ic \Rightarrow \phi$
  - 2. For all c_i that are base cases: $[ic \wedge Case_i] \Rightarrow \phi$
  - 3. For all c_i that are recursive cases: $[ic \wedge Case_i \wedge_{1 \le j \le R_i} \phi|_{\sigma_{ij}}] \Rightarrow \phi$
- If c_i is a recursive case, then it includes at least one call to f.
- Say there are R_i calls to f and they are `(f x1 . . . xn)`$|_{\sigma_{ij}}$, for $1 \le j \le R_i$

Slides by Pete Manolios for CS4820

Induction Schemes

```
(defunc nind (x)
  :input-contract (natp x)
  :output-contract t
  (cond ((= x 0) x)
        (t (nind (1- x)))))
```

Induction on natural numbers

```
(defunc tree-ind (x)
  :input-contract t
  :output-contract t
  (cond ((atom x) x)
        (t (list (tree-ind (car x))
                 (tree-ind (cdr x))))))
```

Induction on trees

```
(defunc true-listp (l)
  :ic t
  :oc (booleanp (true-listp l))
  (if (consp l)
      (true-listp (rest l))
     (equal l () )))
```

Induction on true lists

Can turn if into cond

Common themes:

Induction on data definitions

Induction on functions in conjectures

Custom inductions

Can direct ACL2s to use specific induction scheme

Professional Method

```
(defunc app (a b)
  :input-contract (and (tlp a) (tlp b))
  :output-contract (tlp (app a b))
  (if (endp a)
      b
    (cons (car a)
          (app (cdr a) b))))
```

```
(defunc rev (x)
  :input-contract (tlp x)
  :output-contract (tlp (rev x))
  (if (endp x)
      nil
    (app (rev (cdr x))
         (list (car x)))))
```

Prove: (rev (rev x)) = x No quite right, why?

Prove: (tlp x) ⇒ (rev (rev x)) = x Contract completion!

Professional Method: use abbreviations, discover induction scheme

We'll induct on (... x). Base case is trivial, so go to induction step

```
          (R (R x))
= {Def R} (R (A (R (cdr x)) (L (car x))))
= {L1}    (A (R (L (car x))) (R (R (cdr x))))
= {IH}    (A (R (L (car x))) (cdr x))
= {Def R} (A (L (car x)) (cdr x))
= {Def A} x
```

Hm, to use IH, need lemma

Now I can use IH

Just equational reasoning

L1.(R (A x y)) = (A (R y) (R x))

What Induction scheme?

(tlp x) or (rev x): minor differences

Professional Method

```
(defunc app (a b)
  :input-contract (and (tlp a) (tlp b))
  :output-contract (tlp (app a b))
  (if (endp a)
      b
    (cons (car a)
          (app (cdr a) b))))
```

```
(defunc rev (x)
  :input-contract (tlp x)
  :output-contract (tlp (rev x))
  (if (endp x)
      nil
    (app (rev (cdr x))
         (list (car x)))))
```

Prove: (tlp x) ∧ (tlp y) ⇒ (R (A x y)) = (A (R y) (R x))

Professional Method: induct on?   x controls both LHS, RHS, so probably x

Start with induction step

```
          (R (A x y))
= {Def A} (R (cons (car x) (A (cdr x) y)))
= {Def R} (A (R (A (cdr x) y)) (L (car x)))
= {IH}    (A (A (R y) (R (cdr x))) (L (car x)))
= {Ass A} (A (R y) (A (R (cdr x)) (L (car x))))
= {Def R} (A (R y) (R x))
```

Ass A: (A (A x y) z) = (A x (A y z))
What Induction scheme?
(tlp x) or (rev x): minor differences

Base case?

```
          (R (A x y))
= {Def A} (R y)

          (A (R y) (R x))
= {Def R} (A (R y) nil)
= {L2!}   (R y)
```

L2: (A x nil) = x
Needs proof by induction!

Defun vs Defunc

- Defunc is defined in terms of defun
- Defun doesn't have contracts and you get a *total* function
- Defun has guards, which are similar to input contracts
  - but they do not have a logical meaning
- For example the defun version is non-terminating

```
(defunc !(x)
  :input-contract (natp x)
  :output-contract (posp (! x))
  (if (= x 0)
      1
    (* x (! (1- x)))))
```

```
(defun !(x)
  (declare (xargs :guard (natp x)))
  (if (= x 0)
      1
    (* x (! (1- x)))))
```

- As per CAR, use the appropriate idiom, eg:

```
(defun !(x)
  (declare (xargs :guard (natp x)))
  (if (zp x)
      1
    (* x (! (1- x)))))
```