Lecture 5

Pete Manolios
Northeastern

Questions?

- Piazza
  - Update you email
  - Check regularly
  - Hwk, announcements
- HWK 2 went up yesterday
  - Due in a week (9/23)
  - Get partners now!
  - Update ACL2s

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 ACL2s, we would write `(len (cons x z)) = (len (cons y z))` as

```
(equal/==/= (len (cons x z))   ; equal & == are equal
            (len (cons y z))) ; ='s contract requires numbers
```

- = 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 (τ ≠ `nil`)
  - e.g., x as a formula is `x ≠ nil`
  - `(foo x y z)` as a formula is `(foo x y z)≠ nil`
- Similarly, we add axioms for numbers, strings, etc.
- This is all in GZ, the "ground-zero theory"

Built-in Functions

- Similarly, we add axioms for numbers, strings, etc.
- This is all in GZ, the "ground-zero theory"
- Inference rules include
  - propositional calculus
  - equality
  - instantiation
- Well-foundedness of ϵ_0
- GZ also is inductively complete: for every ϕ, GZ contains the first order induction axioms
  - $\langle \forall y{<}\epsilon_0 :: \langle \forall x{<}y :: \phi(x)\rangle \rightarrow \phi(y)\rangle \rightarrow \langle \forall y{<}\epsilon_0 :: \phi(y)\rangle$
- When GZ is extended (definitions), the resulting theory is the inductive completion of the extension
- Extension principles: defchoose, encapsulation, defaxiom

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? (Review RAP)
  - `(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), (tlp x): x=nil`
  - `(tlp x), (consp x): (tlp (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

```
(=> (and (tlp x)
         (tlp y))
    (=> (and (consp x)
             (not (equal a (first x)))
             (=> (tlp (rest x))
                 (=> (in a (rest x))
                     (in a (app (rest x) y)))))
        (=> (in a x)
            (in a (app x y)))))
```

- First step: Exportation, PL simplification
- The goals are
  - have as many hypotheses as possible
  - flatten & simplify the propositional structure of the conjecture

ER Example

```
(=> (and (tlp x)                                     A
         (tlp y))
    (=> (and (consp x)
             (not (equal a (first x)))
             (=> (tlp (rest x))                      B
                 (=> (in a (rest x))
                     (in a (app (rest x) y)))))
        (=> (in a x)                                 C
            (in a (app x y)))))
```

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

ER Example

```
(=> (and (tlp x)                                  A
         (tlp y)
         (consp x)
         (not (equal a (first x)))                    B
         (=> (tlp (rest x))
             (=> (in a (rest x))
                 (in a (app (rest x) y)))))
    (=> (in a x)                                  C
        (in a (app x y)))))
```

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

ER Example

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (tlp (rest x))
             (=> (in a (rest x))
                 (in a (app (rest x) y)))))
    (=> (in a x)
        (in a (app x y)))))
```

ER Example

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (tlp (rest x))
             (=> (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

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (tlp (rest x))
             (=> (in a (rest x))
                 (in a (app (rest x) y))))
         (in a x))
    (in a (app x y)))))
```

ER Example

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (tlp (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

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (and (tlp (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

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (and (tlp (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

Equational Reasoning

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (and (tlp (rest x))
                  (in a (rest x)))
             (in a (app (rest x) y)))
         (in a x))
    (in a (app x y)))))
```

- Second Step: contract completion
  - do we need any hypotheses?
- You can do this first, but it is easier to check after Exportation

Equational Reasoning

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (and (tlp (rest x))
                  (in a (rest x)))
             (in a (app (rest x) y)))
         (in a x))
    (in a (app x y)))))
```

- Third Step: Generate context
  - List all hypotheses, derived context
  - Can then focus on remaining goal

ER Example

```
(=> (and (tlp x)
         (tlp y)
         (consp x)
         (not (equal a (first x)))
         (=> (and (tlp (rest x))
                  (in a (rest x)))
             (in a (app (rest x) y)))
         (in a x))
    (in a (app x y)))))
```

```
C1. (tlp x)
C2. (tlp y)
C3. (consp x)
C4. a ≠ (first x)
C5. (tlp (rest x)) ∧ (in a (rest x))
      ⇒ (in a (app (rest x) y))
C6. (in a x)
----------------------------------------------
D1. (tlp (rest x))  { C1, Def tlp, C3 }
D2. (in a (rest x)) { C6, Def in, C3, C4, PL }
D3. (in a (app (rest x) y)) { C5, MP, D1, D2 }

Goal: (in a (app x y))
```

```
(definec tlp (l :all) :bool
  (if (consp l)
      (tlp (rest l))
    (equal l () )))
```

```
(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (== a (first X))
           (in a (rest X)))))
```

Equational Reasoning

```
C1. (tlp x)
C2. (tlp y)
C3. (consp x)
C4. a ≠ (first x)
C5. (tlp (rest x)) ∧ (in a (rest x))
     ⇒ (in a (app (rest x) y))
C6. (in a x)
----------------------------------------------
D1. (tlp (rest x))  { C1, Def tlp, C3 }
D2. (in a (rest x)) { C6, Def in, C3, C4, PL }
D3. (in a (app (rest x) y)) { C5, MP, D1, D2 }

Goal: (in a (app x y))
```

- Fourth Step: Prove the goal
  - Term manipulation is now limited to the goal!

ER Example

```
C1. (tlp x)
C2. (tlp y)
C3. (consp x)
C4. a ≠ (first x)
C5. (tlp (rest x)) ∧ (in a (rest x))
      ⇒ (in a (app (rest x) y))
C6. (in a x)
----------------------------------------------
D1. (tlp (rest x))  { C1, Def tlp, C3 }
D2. (in a (rest x)) { C6, Def in, C3, C4, PL }
D3. (in a (app (rest x) y)) { C5, MP, D1, D2 }

Goal: (in a (app x y))

  (in a (app x y))
=   { Def app, C3 }
  (in a (cons (first x) (app (rest x) y)))
=   { Def in, car-cdr-cons axioms }
  (or (equal a (first x)) (in a (app (rest x) y)))
=   { D3, PL }
  t
```

```
(definec app (x :tl y :tl) :tl
  (if (endp x)
      y
    (cons (first x)
          (app (rest x) y))))

(definec tlp (l :all) :bool
  (if (consp l)
      (tlp (rest l))
    (equal l () )))

(definec in (a :all X :tl) :bool
  (and (consp X)
       (or (== a (first X))
           (in a (rest X)))))
```

Equational Reasoning is Easy Peasy Lemon Squeezy

Fermat's last theorem:

For all positive integers x, y, z and n, where $n > 2$, $x^n + y^z \neq z^n$

I have a truly marvelous proof of this proposition which this margin is too narrow to contain.

Fermat, 1637

It took 357 years for a correct proof to be found (by Andrew Wiles in 1995).

Fermat's Last Theorem

For all positive integers x, y, z and n, where $n > 2$, $x^n + y^z \neq z^n$

We can use Fermat's last theorem to construct a conjecture that is hard to prove.

```
(definec fermat (x :pos y :pos z :pos n :pos) :bool
  :ic (> n 2)
  (!= (+ (expt x n) (expt y n)) (expt z n)))

(property (x :pos y :pos z :pos n :pos)
  (=> (> n 2)
      (fermat x y z n)))
```

OR we can define a function that is hard to admit:

```
(defdata true t)
(definec fermat (x :pos y :pos z :pos n :pos) :true
  :ic (> n 2)
  (!= (+ (expt x n) (expt y n)) (expt z n)))
```

We can play this trick with any conjecture.

Even restricted to integers, =, +, *, the validity problem is undecidable, so equational reasoning can be hard.

Questions?