ACL2 Axioms

```
(not (equal t nil))                                                    ;; 1
(implies (equal (not x) nil) (equal (if x y z) y))                     ;; 2
(implies (equal x nil) (equal (if x y z) z))                           ;; 3
(iff (equal x y) (or nil (equal (equal x y) t)))                       ;; 4
(iff (= x y) (equal (equal x y) t))                                    ;; 5
(iff (consp x) (or nil (equal (consp x) t)))                           ;; 6
(equal (consp (cons x y)) t)                                           ;; 7
(equal (consp nil) nil)                                                ;; 8
(equal (car (cons x y)) x)                                             ;; 9
(equal (cdr (cons x y)) y)                                             ;; 10
(implies (equal (consp x) t) (equal (cons (car x) (cdr x)) x))         ;; 11
```

ACL2 Objects

ACL2 permits only objects constructed from

1. rationals
2. complex rationals
3. legal ACL2 characters
4. simple strings of these characters
5. symbols constructed from such strings and interned in the ACL2 packages
6. cons trees of such objects.

ACL2 Proofs - Induction

Induction proofs are structural on lists and binary trees.

Hence `(car x)` and `(cdr x)` are smaller than `x`

when `(consp x)` is `T` is needed to support the induction.

ACL2 Proofs - Induction

Then to prove `(func x y)` is to prove:

base case: `(implies (not (consp x)) (func x y))`
induction: `(implies (and (consp x)`
`(func (car x) alpha)` → hypothesis 1
`(func (cdr x) beta))` → hypothesis 2
`(func x y))` → conclusion

where alpha and beta are specific to the problem.

ACL2 Proofs - Induction, Example

```
(defun lstnats (n)
  (if (zp n)
      nil
      (cons n (lstnats (1- n)))))

(defun addlst (lst)
  (if (not (consp lst))
      0
      (+ (car lst) (addlst (cdr lst)))))

(thm
  (let ((x (lstnats n)))
       (and
           (implies (not (consp x)) (equal x nil))
           (implies (and (consp x)
                         (equal (addlst (cdr lst))
                                (* 1/2 (car lst) (1- (car lst)))))
                    (equal (addlst lst)
                           (* 1/2 (car lst) (+ 1 (car lst)))))))))
```

ACL2 Proofs - Induction, Example

```
(defun sum (n)
    (if (zp n)
        0
        (+ n (sum (1- n)))))

(thm
    (or (and (zp n) (equal (sum 0) 0))          → base case
        (equal (sum n) (* 1/2 n (+ n 1))))      → induction

(thm
    (equal (sum n) (* 1/2 n (+ n 1))))          ⇐ doesn't work

    (implies (and (zp n) (acl2-numberp n))
        (equal 0 (+ (* 1/2 n) (* 1/2 n n))))    ⇐ fails here
```

ACL2 Proofs - Induction, Example

```
(defun treecopy (x)
    (if (consp x)
        (cons (treecopy (car x)) (treecopy (cdr x)))
        x

(thm
    (equal (treecopy x) x)



;; (IMPLIES (NOT (CONSP X))
;;          (EQUAL (TREECOPY X) X)).
;;
;; But simplification reduces this to T, using the :definition TREECOPY
;; and primitive type reasoning.
;;
;; Subgoal *1/1
;; (IMPLIES (AND (CONSP X)
;;               (EQUAL (TREECOPY (CAR X)) (CAR X))
;;               (EQUAL (TREECOPY (CDR X)) (CDR X)))
;;          (EQUAL (TREECOPY X) X)).
```

ACL2 Proofs - Induction, Example

```
(treecopy x) =

    (if (consp x) (cons (treecopy (car x)) (treecopy (cdr x))) x)

    (if t (cons (treecopy (car x)) (treecopy (cdr x))) x)          Axiom 6

    (cons (treecopy (car x)) (treecopy (cdr x)))                   Axioms 1,2

    (cons (car x) (treecopy (cdr x)))                              Hypothesis

    (cons (car x) (cdr x))                                         Hypothesis

    x                                                              Axiom 11
```

ACL2 Proofs - Strong Theorem

A strong theorem is less general than a weak theorem.

example: the fundamental theorem of algebra says that every non-constant complex polynomial has at least one root. But it is true that every non-constant complex polynomial has as many roots as the degree of the polynomial if the roots are counted according to multiplicity. The second not only says that roots exist, it actually tells how many there are.

ACL2 Proofs - Strong Theorem

```
(defun memb (e l)
    (if (not (consp l))
        nil
        (if (equal e (car l))
            't
            (memp e (cdr l)))))

(thm (equal (memp e (app a b)) (or (memp e a) (memp e b))))
```

Does not work - too general - too weak. Consider this stronger theorem:

```
(thm (equal (memp e (app a a)) (memp e a)))
```

ACL2 Proofs - Rewriting

Rewriting is a simplification by

reduction to some preferred form using some rules
rules are derived from definitions, axioms, and proved theorems
about a dozen kinds of rules - here we speak of
no rule, rewrite rule, linear arithmetic rule

Rewrite example:

`(implies (and hyp1 hyp2 hyp3...)  (equal a b))`
will replace occurrence of `b` with `a`
provided `hyp1..` rewrite to `T`.

ACL2 Proofs - Rewriting

But arithmetic is hard to figure out - these do not work

```
(thm
    (implies
        (and (< 0 (len w)) (<= (len q) (len w)))
        (<= (/ (len q) (len w)) 1)))
```

ACL2 Proofs - Rewriting

```
(defun P (e w) (/ (len e) (len w)))

(defun set-union (e w)
    (if (endp e)
        w
        (if (endp w)
            e
            (if (member (car e) w)
                (set-union (cdr e) w)
                (cons (car e) (set-union (cdr e) w))))))

(defun set-intersect (e w)
    (if (or (endp e) (endp w))
        nil
        (if (member (car e) w)
            (cons (car e) (set-intersect (cdr e) w))
            (set-intersect (cdr e) w))))
```

ACL2 Proofs - Rewriting

```
(thm (<= 0 (P a w)))
```
⇐ works

```
(thm
    (implies
        (and (< 0 (len w)) (<= (len a) (len w))) (<= (P a w) 1)))
```
⇐ does not work

```
(thm
    (implies
        (and (subsetp a w) (subsetp b w))
            (equal (P (set-union a b) w)
            (- (+ (P a w) (P b w)) (P (set-intersect a b) w)))))
```
⇐ but this does work!

```
(thm
    (implies
        (and (subsetp a w) (subsetp c w))
        (equal (* (P (set-intersect a c) (set-intersect c w))
                  (P c w))
               (P (set-intersect a c) w))))
```
⇐ yet this does not!

ACL2 Proofs - Arithmetic

Use canned packages for help!!

```
(include-book "arithmetic/top-with-meta" :dir :system)
```

ACL2 maintains a graph of terms involved in current conjecture and relates the terms to inequalities.

For example, may be used to derive `(<= 0 (+ a (* b b)))` from `(<= 0 a)` and `(<= 0 (* b b))`

```
(thm (<= 0 (* b b))) ;; fails
(thm (implies (integerp b) (<= 0 (* b b)))) ;; succeeds
```