Theorem prover ACL2

–handout–

Some primitive (built-in) functions

`(cons` x y`)`	constructs the ordered pair $\langle x, y\rangle$
`(car` x`)`	left component of x, if x is a pair; `nil` otherwise
`(cdr` x`)`	right component of x, if x is a pair; `nil` otherwise
`(consp` x`)`	`t` if x is a pair; `nil` otherwise
`(if` x y z`)`	z if x is `nil`; y otherwise
`(equal` x y`)`	`t` if x is y; `nil` otherwise

Some primitive (built-in) axioms

1. `t` $\neq$ `nil`
2. `x` $\neq$ `nil` $\rightarrow$ `(if x y z)` $=$ `y`
3. `x` $=$ `nil` $\rightarrow$ `(if x y z)` $=$ `z`
4. `(equal x y)` $=$ `nil` $\vee$ `(equal x y)` $=$ `t`
5. `x` $=$ `y` $\leftrightarrow$ `(equal x y)` $=$ `t`
6. `(consp x)` $=$ `nil` $\vee$ `(consp x)` $=$ `t`
7. `(consp (cons x y))` $=$ `t`
8. `(consp nil)` $=$ `(consp t)` $=$ `(consp 'ok)` $=$ `(consp 0)` $=$ `(consp 1)` $= \ldots =$ `nil`
9. `(car (cons x y))` $=$ `x`
10. `(cdr (cons x y))` $=$ `y`
11. `(consp x)` $=$ `t` $\rightarrow$ `(cons (car x) (cdr x))` $=$ `x`

Function definition

`(defun` f `(`a_1 a_2 $\ldots$ a_n`)` β`)` creates the function f with arguments $a_1, a_2, \ldots, a_n$ and body β

(Built-in) Lisp definitions of standard logic connectives

```
(defun not (p) (if p nil t))
(defun and (p q) (if p q nil))
(defun or (p q) (if p p q))
(defun implies (p q) (if p (if q t nil) t))
(defun iff (p q) (and (implies p q) (implies qp)))
```

Examples of recursive function definitions

`dup` - duplicates each element in a list

```
(defun dup (x)
  (if (consp x)
      (cons (car x)
            (cons (car x)
                  (dup (cdr x))))
      nil))
```

`app` - concatenates two lists

```
(defun app (x y)
  (if (consp x)
      (cons (car x) (app (cdr x) y))
      y))
```

A simple proof

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

Theorem: (equal (treecopy x) x)).

Proof: Name the formula above *1.

Perhaps we can prove *1 by induction. One induction scheme is suggested by this conjecture.

We will induct according to a scheme suggested by (treecopy x). This suggestion was produced using the induction rule treecopy. If we let (φ x) denote *1 above then the induction scheme we'll use is

```
(and (implies (not (consp x)) (φ x))
     (implies (and (consp x)
                   (φ (car x))
                   (φ (cdr x)))
              (φ x))).
```

This induction is justified by the same argument used to admit treecopy. When applied to the goal at hand the above induction scheme produces two nontautological subgoals.

Subgoal *1/2

```
(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)).
```

But simplification reduces this to t, using the definition treecopy, primitive type reasoning and the rewrite rule cons-car-cdr.

That completes the proof of *1.

Q.E.D. □

Simplification of Subgoal *1/1

```
(treecopy x) = (if (consp x)                   ; treecopy definition
                   (cons (treecopy (car x))
                         (treecopy (cdr x)))
                   x)
             = (if t                           ; hypothesis 1
                   (cons (treecopy (car x))
                         (treecopy (cdr x)))
                   x)
             = (cons (treecopy (car x))        ; axioms 1 and 2
                     (treecopy (cdr x)))
             = (cons (car x)                   ; hypothesis 2
                     (treecopy (cdr x)))
             = (cons (car x)                   ; hypothesis 3
                     (cdr x))
             = x                               ; axiom 11 and hypothesis 1
```

For more information visit http://www.cs.utexas.edu/users/moore/acl2/