Peter Dillinger

Induction Involving Multiple Variables

So far we've seen induction involving just one variable, because the schemes have been based on functions that take just one parameter, like TRUE-LISTP. Let's now consider proofs that involve induction with multiple variables. Consider the function

```
(defun weave (x y)
  (if (endp x)
    y
    (if (endp y)
      x
      (cons (car x)
            (cons (car y)
                  (weave (cdr x) (cdr y)))))))
```

It will "stitch" or "weave" together two lists like so:

```
(weave '(1 2 3) '(4 5 6)) = '(1 4 2 5 3 6)
(weave '(1) '(2 3 4))     = '(1 2 3 4)
(weave '(1 2 3) '(4))     = '(1 4 2 3)
```

And recall the definition of len:

```
(defun len (x)
  (if (endp x)
    0
    (+ 1 (len (cdr x)))))
```

Suppose we want to prove

```
(equal (len (weave x y))
       (+ (len x) (len y)))
```

If we try to prove it without induction, we don't make any progress. We could also try to prove it by induction using the scheme we're most familiar with, that based on TRUE-LISTP or LEN (same scheme, base test (endp x) and replace x with (cdr x)). This would be the inductive step:

```
(implies (and (not (endp x))
              (equal (len (weave (cdr x) y))
                     (+ (len (cdr x)) (len y))))
         (equal (len (weave x y))
                (+ (len x) (len y))))
```

We would start with

```
(len (weave x y))
```

and expand the definition of weave, but since we only know (not (endp x)), we get

```
  (len (if (endp y)
           x
           (cons (car x)
                 (cons (car y)
                       (weave (cdr x) (cdr y))))))
```

If we did case analysis for (endp y) and (not (endp y)), we would be able to prove the (endp y) case, but for (not (endp y)), we need to know something about (weave (cdr x) (cdr y)). However, we only know about (weave (cdr x) y), according to the induction hypothesis given by the induction scheme we used. This suggests we used the wrong induction scheme.

We actually need to induct based on the scheme given by (weave x y). Usually the correct induction scheme will come from the most important function in your conjecture, the one that the conjecture seems to be about. This induction scheme will be rather different because (1) the function takes two parameters and (2) the function has two base cases. When a function has multiple IF branches, we can combine all those into one base case, in which the base test covers all cases in which we do not make a recursive call:

```
  (implies (or (endp x)                  Combined Base Case
               (endp y))
           PHI)
```

where PHI is the formula we're trying to prove. We would need to prove that by case analysis on (endp x) and (not (endp x)). Equivalently, we can just construct two base cases, each with hypotheses of what needs to be true to reach that base case in the function:

```
  (implies (endp x)                      Base Case 1
           PHI)
  (implies (and (not (endp x))           Base Case 2
                (endp y))
           PHI)
```

For the PHI we are interested in,

```
  (equal (len (weave x y))
         (+ (len x) (len y)))
```

both of these are easy to prove using the definitions of WEAVE and LEN. (Try them!)

The inductive step is also interesting, because we have to replace two variables according to the recursive call in WEAVE:

```
  (implies (and (not (endp x))            Inductive Step
                (not (endp y))
                (let ((x (cdr x))
                      (y (cdr y)))
                  PHI))
           PHI)
```

So we would have assumptions

```
    1. (not (endp x))
    2. (not (endp y))
    3. (equal (len (weave (cdr x) (cdr y)))
              (+ (len (cdr x)) (len (cdr y))))
```

and start with (len (weave x y)) and show that is equal to (+ (len x) (len y)).

```
 (len (weave x y))
= { Def weave, assumptions }
 (len (cons (car x)
            (cons (car y)
                  (weave (cdr x) (cdr y)))))
= { Def len, not endp cons, cdr-cons }
 (+ 1 (len (cons (car y)
                 (weave (cdr x) (cdr y)))))
= { Def len, not endp cons, cdr-cons }
 (+ 1 (+ 1 (len (weave (cdr x) (cdr y)))))
= { Assumption 3 }
 (+ 1 (+ 1 (+ (len (cdr x)) (len (cdr y)))))
= { Arith (comm. and assoc. of addition) }     <- At this point I'm carefully
 (+ (+ 1 (len (cdr x))) (+ 1 (len (cdr y))))    "working backwards".  You might
= { Def len * 2, assumptions 1 & 2 }            prefer to restart from the goal
 (+ (len x) (len y))                          <- here.
```

Another example

Let's look at an accumulator version of LEN, called LEN-AC:

```
  (defun len-ac (l n)
    (if (endp l)
      (+ 0 n)                      ; make sure return value is a number
      (len-ac (cdr l) (+ 1 n))))
```

We want to relate LEN-AC to LEN, by proving

```
  (equal (len-ac l n)
         (+ (len l) n))
```

If we tried proving this without induction or by induction based on (len l), we fail. The most important function in this conjecture is LEN-AC, so we are likely to have success using induction based on (len-ac l n):

```
  (implies (endp l)                                  Base Case
           (equal (len-ac l n)
                  (+ (len l) n)))

  (implies (and (not (endp l))                       Inductive Step
                (equal (len-ac (cdr l) (+ 1 n))
                       (+ (len (cdr l)) (+ 1 n))))
           (equal (len-ac l n)
                  (+ (len l) n)))
```

Notice we had to replace both l and n in the induction hypothesis. The proof of each obligation is reasonably simple. (Try them!)