CSU290 Lecture Notes Lecture 24 13 Nov 2008

Peter Dillinger

A Complicated Induction Proof

We have seen a function that returns the maximum element of a list.
Let’s consider an accumulator version of such a function:

```
  (defun maxl-ac (x m)
    (if (endp x)
      m
      (maxl-ac (cdr x) (max (car x) m))))
```

where

```
  (defun max (x y)
    (if (>= x y)
      x
      y))
```

MAXL-AC takes a list and the maximum so far and returns the maximum among that so far and all the elements of the given list. Here are some examples:

```
  (maxl-ac '(1 2 3) 1)  = 3
  (maxl-ac '(1 2 3) 4)  = 4
  (maxl-ac '(4 -3 5) 0) = 5
```

We want to prove a kind of type theorem about MAXL-AC: if we give MAXL-AC a list of natural numbers for x and a natural number for m, it returns a natural number. We have a function that recognizes natural numbers, NATP, but we need a function that recognizes lists of natural numbers:

```
  (defun nat-listp (x)
    (if (endp x)
      (equal x nil)
      (and (natp (car x))
           (nat-listp (cdr x)))))
```

We want to prove

```
  (implies (and (nat-listp x)
                (natp m))
           (natp (maxl-ac x m)))
```

If we try to prove this without induction, or induction based on (nat-listp x), we don’t make a lot of progress. We need induction based on (maxl-ac x m), which involves two variables. Also, the conjecture we are trying to prove is an implication, so we should use our method of proving implications by induction (see lecture 22):

```
  B = (endp x)
  H = (and (nat-listp x)  ; H1
           (natp m))      ; H2
  H'= (and (nat-listp (cdr x))      ; H'1
           (natp (max (car x) m)))  ; H'2
  C = (natp (maxl-ac x m))
  C'= (natp (maxl-ac (cdr x) (max (car x) m)))
```

Notice that because H and H’ are the AND of two formulas, we name the two parts H1 and H2, and H’1 and H’2.

First we need to prove the Base Case, which is

```
  (implies (and B H)
           C)
```

Thus, we have assumptions: B, H1, H2

```
  (natp (maxl-ac x m))
<= { Def maxl-ac, B }
  (natp m)
<= { H2 }
  t
```

Next we need to do Induction Hypothesis Chaining, which is

```
  (implies (and (not B)
                H)
           H')
```

Assumptions: ~B, H1, H2

We need to prove H', which is the AND of H'1 and H'2. By propositional deduction, we can prove H' by proving H'1 and H'2.

First H'1: (This requires some cleverness.)

```
  (nat-listp (cdr x))
<= { Prop. ded. }
  (and (natp (car x))
       (nat-listp (cdr x)))
<= { Def nat-listp, ~B }
  (nat-listp x)
<= { H1 }
  t
```

That could be a little confusing because we used the definition of nat-listp in reverse. Another way we could prove this is start with something we know, H1, and prove that that implies what we are trying to prove (note that the arrows are pointing the other direction):

```
   { Assumption H1 }
  (nat-listp x)
=> { Def nat-listp, ~B }
  (and (natp (car x))
       (nat-listp (cdr x)))
=> { Prop. ded. }
  (nat-listp (cdr x))
```

Now let's prove H'2:

```
  (natp (max (car x) m))
<= { Def max }
  (natp (if (>= (car x) m)
          (car x)
          m))
```

Case analysis. Case 1: (not (>= (car x) m))

```
<= { (not (>= (car x) m)) }
  (natp m)
<= { H2 }
  t
```

```
Case 2:  (>= (car x) m)

<= { (>= (car x) m) }
  (natp (car x))
<= { Prop. ded. }
  (and (natp (car x))
       (nat-listp (cdr x)))
<= { Def. nat-listp, ˜B }
  (nat-listp x)
<= { H1 }
  t
```

That was another case of needing to be clever in using the definition "backwards." That completes H’2 and, thus, the Induction Hypothesis Chaining.

Now we need to prove the Indution Step, which is

```
  (implies (and (not B)
                H
                H’
                C’)
           C)
```

Assumptions: ˜B, H1, H2, H’1, H’2, C’

```
  (natp (maxl-ac x m))
<= { Def. maxl-ac, ˜B }
  (natp (maxl-ac (cdr x) (max (car x) m)))
<= { C’ }
  t
```

Boy, that part was easy. Having complete the base case, the lengthy induction hypothesis chaining, and the short induction step, the proof is complete.