Internal Note #185
Computational Logic, Inc.
April 10, 1990

An Instructive Example for Beginning Users of the Boyer-Moore Theorem Prover

Matt Kaufmann[1]

Computational Logic, Inc.
1717 W. Sixth Street, Suite 290
Austin, TX 78703

kaufmann@cli.com

In this note we present a solution to a little exercise[2]: Use the Boyer-Moore theorem prover to prove that if one rotates the elements of a list by **n**, where **n** is the length of the given list, then the result is equal to the given list. These notes are rather sketchy in places; hence their use is recommended primarily in conjunction with a talk, or for those who have a little familiarity with the Boyer-Moore theorem prover. In order to illustrate the use of the Boyer-Moore theorem prover (and, to a lesser extent, of its interactive enhancement PC-NQTHM[3]), we'll work through this entire example, giving lots of comments. A summary of that proof effort, i.e. the final input file, is in the first appendix. An alternate solution is presented in the second appendix.

---

[1]This work was supported in part by ONR Contract N00014-88-C-0454. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Computational Logic, Inc., the Office of Naval Research or the U.S. Government.

[2]first brought to my attention by Jeff Cook

[3]''PC'' for ''proof-checker'', ''NQTHM'' for a name used sometimes for the Boyer-Moore prover

For a complete description of the logic of Boyer and Moore and for more information on how to use their theorem prover, see their book [1]. For more information about PC-NQTHM see [2] and [3].

Boyer and Moore's prover nqthm can currently be obtained through anonymous ftp from cli.com. Ftp to cli.com, login as anonymous, any password, cd to /pub/nqthm, get the file README, and follow the directions. PC-NQTHM may be similarly obtained from the directory /pub/proof-checker/, also under ~ftp.

Notational conventions. We use the following notational convention from Lisp: comments begin with semicolons (;) and extend to the end of the line. The symbol '>' on the left margin in displays is a Lisp prompt. The DEFN and PROVE-LEMMA forms that follow such prompts and are displayed in **`UPPER CASE TYPEWRITER FONT`** form a list of *events* that can be run successfully through the Boyer-Moore theorem prover.

Let's begin.[4] Here is a recursive definition of a function that rotates a list. Notice that this definition uses some built-in functions: **`ZEROP`** is true of 0 and of all objects that are not natural numbers, **`CAR`** returns the first member of a list, **`CDR`** returns all but the first member of a list, and **`APPEND`** concatenates two lists into a single list.

```
(DEFN ROTATE (N LST)
  (IF (ZEROP N)
      LST
    (ROTATE (SUB1 N)
            (APPEND (CDR LST) (LIST (CAR LST))))))
```

The theorem prover's output is as follows. (We often omit its output henceforth.)

```
     Linear arithmetic, the lemma COUNT-NUMBERP, and the definition of ZEROP
establish that the measure (COUNT N) decreases according to the well-founded
relation LESSP in each recursive call.  Hence, ROTATE is accepted under the
principle of definition.  From the definition we can conclude that:
      (OR (LISTP (ROTATE N LST))
          (EQUAL (ROTATE N LST) LST))
is a theorem.




[ 0.1 0.1 0.0 ]
ROTATE

>
```

Here is an example, using the read-eval-print loop for the logic, of how **`ROTATE`** functions.

[4] Normally one would start with the initialization command **`(BOOT-STRAP NQTHM)`** -- this is actually not necessary if one loads a saved core image.

```
>(r-loop)

Abbreviated Output Mode:  On
Type ? for help.
*(rotate 2 '(a b c d e f))
 '(C D E F A B)
*
```

Next we need the definition of the length of a list.

```
(DEFN LENGTH (X)
  (IF (LISTP X)
      (ADD1 (LENGTH (CDR X)))
    0))
```

Let's try to prove some version of our main theorem right away. Maybe we'll be lucky.

```
>(prove-lemma rotate-length ()
  (implies (listp lst)
           (equal (rotate (length lst) lst)
                  lst)))


     Name the conjecture *1.


     We will appeal to induction.  There is only one plausible induction.  We
will induct according to the following scheme:
      (AND (IMPLIES (AND (LISTP LST) (p (CDR LST)))
                    (p LST))
           (IMPLIES (NOT (LISTP LST)) (p LST))).
Linear arithmetic and the lemma CDR-LESSP inform us that the measure
(COUNT LST) decreases according to the well-founded relation LESSP in each
induction step of the scheme.  The above induction scheme produces the
following two new formulas:

Case 2. (IMPLIES (AND (NOT (LISTP (CDR LST)))
                      (LISTP LST))
                 (EQUAL (ROTATE (LENGTH LST) LST)
                        LST)).

  This simplifies, applying SUB1-ADD1, and expanding LENGTH, APPEND, and
  ROTATE, to:

        (IMPLIES (AND (NOT (LISTP (CDR LST)))
                      (LISTP LST))
                 (EQUAL (ROTATE (LENGTH (CDR LST))
                                (LIST (CAR LST)))
                        LST)).

  This further simplifies, appealing to the lemmas CAR-CONS and CDR-CONS, and
  opening up LENGTH, EQUAL, and ROTATE, to the goal:
```

```
      (IMPLIES (AND (NOT (LISTP (CDR LST)))
                    (LISTP LST))
               (EQUAL (LIST (CAR LST)) LST)).

  Appealing to the lemma CAR-CDR-ELIM, we now replace LST by (CONS Z X) to
  eliminate (CDR LST) and (CAR LST).  We must thus prove:

      (IMPLIES (NOT (LISTP X))
               (EQUAL (LIST Z) (CONS Z X))).

  However this further simplifies, rewriting with CAR-CONS, CDR-CONS, and
  CONS-EQUAL, to:

      (IMPLIES (NOT (LISTP X))
               (EQUAL NIL X)),

  which has two irrelevant terms in it.  By eliminating these terms we get the
  new formula:

    F.

Why say more?

************** F  A  I  L  E  D **************


[ 0.0 0.6 0.3 ]
NIL

>
```

Did the proof attempt fail because this is not a theorem, or is it because the Boyer-Moore theorem prover's heuristics aren't sufficient in this case? A clue is provided by the final goal printed out before the proof failed: **(IMPLIES (NOT (LISTP X)) (EQUAL NIL X))**. This says that every object that is not a non-empty list is in fact the object **nil**. But that's clearly false; consider e.g. the number 3. We can enter the read-eval-print loop for the logic to see that the theorem really does fail. In the following example, **'(a b c . d)** is a standard Lisp abbreviation for **(cons 'a (cons 'b (cons 'c 'd)))**, while **'(a b c)** is a standard Lisp abbreviation for **(cons 'a (cons 'b (cons 'c nil)))**.

```
>(r-loop)

Abbreviated Output Mode:  On
Type ? for help.
*(rotate 3 '(a b c . d))
 '(A B C)
*
```

The thing to notice above is that the innermost (last) **cons** from **'(a b c . d)**, i.e. **(cons c d)**, has a second argument that is not **nil**. So, perhaps we need some notion of ''a list terminating in **NIL**.''

```
(DEFN PROPERP (X)
  (IF (LISTP X)
      (PROPERP (CDR X))
    (EQUAL X NIL)))
```

One can see how this function works by tracing it.[5] Here are a couple of such traces, which we present without comment on the chance that they may be illuminating.

```
*(properp '(a b c . d))
  1> (<<PROPERP>> '((A B C . D)))
    2> (<<PROPERP>> '((B C . D)))
      3> (<<PROPERP>> '((C . D)))
        4> (<<PROPERP>> '(D))
        <4 (<<PROPERP>> F)
      <3 (<<PROPERP>> F)
    <2 (<<PROPERP>> F)
  <1 (<<PROPERP>> F)
 F
*(properp '(a b c))
  1> (<<PROPERP>> '((A B C)))
    2> (<<PROPERP>> '((B C)))
      3> (<<PROPERP>> '((C)))
        4> (<<PROPERP>> '(NIL))
        <4 (<<PROPERP>> T)
      <3 (<<PROPERP>> T)
    <2 (<<PROPERP>> T)
  <1 (<<PROPERP>> T)
 T
*
```

Now we might try the proof again. Notice that **`(rotate 0 nil)`** = **`nil`**, so we don't need a hypothesis that the list is non-empty. It's usually good to state theorems without extraneous hypotheses, so as not to confuse the prover (or the reader of its output) with irrelevant information. So, we replace the unnecessary hypothesis **`(listp lst)`** with the necessary one **`(properp lst)`**.

```
(prove-lemma rotate-length ()
  (implies (properp lst)
           (equal (rotate (length lst) lst)
                  lst)))
```

Unfortunately, the proof still fails in this case. The output includes the following goals and description.

[5] The patch that allows such tracing is available upon request from Matt Kaufmann for use with akcl implementations of the Boyer-Moore prover.

```
        (IMPLIES (PROPERP X)
                 (EQUAL (ROTATE (LENGTH X)
                                (APPEND X (LIST Z)))
                        (CONS Z (ROTATE (LENGTH X) X)))),
```

which we generalize by replacing (LENGTH X) by Y. We restrict the new variable by recalling the type restriction lemma noted when LENGTH was introduced. This produces the new formula:

```
        (IMPLIES (AND (NUMBERP Y) (PROPERP X))
                 (EQUAL (ROTATE Y (APPEND X (LIST Z)))
                        (CONS Z (ROTATE Y X)))),
```

which we will name *1.1.

We may as well abort the proof attempt at that point, since this goal is clearly not a theorem; when **Y** = 0 and **X** is a **PROPERP**, it says that **(APPEND X (LIST Z))** = **(CONS Z X)**, which is clearly false in general. In fact generalization tends to be the least reliable heuristic in the Boyer-Moore theorem prover. But perhaps the first goal displayed above suggests (this is debatable) that we might try proving something more general, by ‘‘decoupling’’ the term **(LENGTH X)** from the term **X**. That is, let’s consider how to rewrite **(ROTATE N X)** where **N** is any number not exceeding the length of **X**. If you think about how to prove the main theorem on paper, you may well be led to such a generalization, since it has the feel of something that might be proved by induction on **N**. Here is an informal statement of the theorem, with an informal proof by induction on **n**.

Theorem: if **n** $\leq$ **k**, *then*

$\mathtt{rotate(n,\ <}x_1 \ldots x_k\mathtt{>)\ =}$
$\mathtt{<}x_{n+1} \ldots x_k\ x_1 \ldots x_n\mathtt{>}$

Proof sketch: by induction on **n**. Base step (**n** = 0): trivial.

Inductive step: if $0 < n \leq k$, then

$\mathtt{rotate(n,\ <}x_1 \ldots x_k\mathtt{>)}$
= *{by definition of rotate}*
$\mathtt{rotate(n-1,\ <}x_2 \ldots x_k\ x_1\mathtt{)}$
= *{by the inductive hypothesis}*
$\mathtt{<}x_{n+1} \ldots x_k\ x_1\quad x_2 \ldots x_n\mathtt{>}$

and we’re done!

Now we formalize the operations that take the first **n** or last **n** elements of a list. **(FIRSTN N LST)** returns the first **n** elements of lst (assuming that there are at least **n** elements). Similarly, **(NTHCDR N LST)** returns the last **n** elements of lst (assuming that there are at least **n** elements).

```
(DEFN FIRSTN (N LST)
  (IF (ZEROP N)
      NIL
    (CONS (CAR LST)
          (FIRSTN (SUB1 N) (CDR LST)))))

(DEFN NTHCDR (N LST)
  (IF (ZEROP N)
      LST
    (NTHCDR (SUB1 N) (CDR LST))))
```

The big subgoal, then, is as follows.

```
(prove-lemma rotate-append ()
  (implies (and (properp lst)
                (not (lessp (length lst) n)))
           (equal (rotate n lst)
                  (append (nthcdr n lst)
                          (firstn n lst)))))
```

Having formulated this nice lemma, we are sorely tempted to hand it off to the theorem prover and hope that it is proved automatically. An alternate approach suggests thinking about the hand proof a little more and suggesting an appropriate induction scheme to the theorem prover, and we take that approach in the appendix below. But for now, let us continue in the standard manner, i.e. let us succumb to the temptation to hand off the lemma ROTATE-APPEND (that we have just stated above) to the theorem prover.

Unfortunately the proof of this lemma fails. The prover generalizes the following subgoal to one that's not a theorem.

```
(IMPLIES (AND (NUMBERP X)
              (EQUAL (ROTATE X Z)
                     (APPEND (NTHCDR X Z) (FIRSTN X Z)))
              (PROPERP Z)
              (NOT (LESSP (LENGTH Z) X)))
         (EQUAL (ROTATE X (APPEND Z (LIST V)))
                (APPEND (NTHCDR X Z)
                        (CONS V (FIRSTN X Z)))))
```

It appears that the inductive hypothesis doesn't match up very well with the induction conclusion, since the conclusion mentions the notion of **ROTATE**-ing an **APPEND** term, **(ROTATE X (APPEND Z (LIST V)))**. So let us ''generalize'' the theorem we are trying to prove by considering what happens when we append extra stuff to the end of the list before rotating it.

```
(prove-lemma rotate-append ()
  (implies (and (properp lst)
                (not (lessp (length lst) n)))
           (equal (rotate n (append lst extra))
                  (append (nthcdr n lst)
                          (append extra (firstn n lst))))))
```

This lemma's proof was not successful -- it generated the unprovable subgoal **(EQUAL EXTRA (APPEND EXTRA NIL))**. Inspection suggests that we need **(PROPERP EXTRA)** in order for the **N** = 0 case to be true, which makes sense since then **(EQUAL EXTRA (APPEND EXTRA NIL))** is true. But with that added hypothesis it isn't too hard to see that we no longer need to assume **(PROPERP LST)**.

Thus our next try is as follows.

```
(prove-lemma rotate-append ()
  (implies (and (properp extra)
                (not (lessp (length lst) n)))
           (equal (rotate n (append lst extra))
                  (append (nthcdr n lst)
                          (append extra (firstn n lst))))))
```

However, the proof still fails. Inspection of the output reveals the term **(APPEND (APPEND Z EXTRA) (LIST V))**, which suggests (at least to an experienced user) that we should prove the associativity of **APPEND**. Some day it will be the case that Boyer-Moore theorem prover users always load libraries with such basic facts built in. Anyhow, let's prove that lemma now before looking any further at the failed proof of ROTATE-APPEND.

```
(PROVE-LEMMA ASSOCIATIVITY-OF-APPEND (REWRITE)
  (EQUAL (APPEND (APPEND X Y) Z)
         (APPEND X (APPEND Y Z))))
```

We may as well also prove the aforementioned fact about **APPEND**ing to **NIL**.

```
(PROVE-LEMMA APPEND-NIL (REWRITE)
  (IMPLIES (PROPERP X)
           (EQUAL (APPEND X NIL)
                  X)))
```

Now we try again to prove ROTATE-APPEND. The proof still fails. Again the theorem prover chooses to generalize. The goal printed out just before generalization is hard to understand (in my opinion, anyhow). So, I'll enter the ''proof-checker'' enhancement of the Boyer-Moore prover, PC-NQTHM. The VERIFY form below causes us to enter PC-NQTHM with the indicated goal, which is the statement of the theorem ROTATE-APPEND.

```
>(verify (implies (and (properp extra)
                       (not (lessp (length lst) n)))
                  (equal (rotate n (append lst extra))
                         (append (nthcdr n lst)
                                 (append extra (firstn n lst))))))
```

;; All input below follows the PC-NQTHM prompt, **'->: '** *; the rest (except for italicized comments) is*
;; output from the system.
->: p *;; Print the ''current term''.*

```
(IMPLIES (AND (PROPERP EXTRA)
              (NOT (LESSP (LENGTH LST) N)))
         (EQUAL (ROTATE N (APPEND LST EXTRA))
                (APPEND (NTHCDR N LST)
                        (APPEND EXTRA (FIRSTN N LST)))))
```

->: induct *;; Divide into subgoals using a heuristically-chosen induction scheme.*

```
Inducting according to the scheme:

      (AND (IMPLIES (OR (EQUAL N 0) (NOT (NUMBERP N)))
                    (p EXTRA LST N))
           (IMPLIES (AND (NOT (OR (EQUAL N 0) (NOT (NUMBERP N))))
                         (OR (EQUAL (LENGTH LST) 0)
                             (NOT (NUMBERP (LENGTH LST)))))
                    (p EXTRA LST N))
           (IMPLIES (AND (NOT (OR (EQUAL N 0) (NOT (NUMBERP N))))
                         (NOT (OR (EQUAL (LENGTH LST) 0)
                                  (NOT (NUMBERP (LENGTH LST)))))
                         (p EXTRA (CDR LST) (SUB1 N)))
                    (p EXTRA LST N)))

Creating 3 new subgoals, (MAIN . 1), (MAIN . 2), and (MAIN . 3).

The proof of the current goal, MAIN, has been completed.  However, the
following subgoals of MAIN remain to be proved: (MAIN . 1), (MAIN . 2), and
(MAIN . 3).
Now proving (MAIN . 1).
->: p

(IMPLIES (OR (EQUAL N 0) (NOT (NUMBERP N)))
         (IMPLIES (AND (PROPERP EXTRA)
                       (NOT (LESSP (LENGTH LST) N)))
                  (EQUAL (ROTATE N (APPEND LST EXTRA))
                         (APPEND (NTHCDR N LST)
                                 (APPEND EXTRA (FIRSTN N LST))))))
```

```
->: prove   ;; We call the Boyer-Moore theorem prover, since this looks like an easy base case.


***** Now entering the theorem prover *****:

This simplifies, applying APPEND-NIL, and opening up NOT, OR, EQUAL, LESSP,
ROTATE, NTHCDR, and FIRSTN, to:

      T.

Q.E.D.


The current goal, (MAIN . 1), has been proved, and has no dependents.
Now proving (MAIN . 2).
->: p

(IMPLIES (AND (NOT (EQUAL N 0))
              (NUMBERP N)
              (OR (EQUAL (LENGTH LST) 0)
                  (NOT (NUMBERP (LENGTH LST)))))
         (IMPLIES (AND (PROPERP EXTRA)
                       (NOT (LESSP (LENGTH LST) N)))
                  (EQUAL (ROTATE N (APPEND LST EXTRA))
                         (APPEND (NTHCDR N LST)
                                 (APPEND EXTRA (FIRSTN N LST))))))
->: prove   ;; We call the prover again, since this also looks like an easy base case.


***** Now entering the theorem prover *****:

This formula simplifies, opening up NOT, OR, EQUAL, and LESSP, to:

      T.

Q.E.D.


The current goal, (MAIN . 2), has been proved, and has no dependents.
Now proving (MAIN . 3).
```

```
->: p

(IMPLIES (AND (NOT (EQUAL N 0))
              (NUMBERP N)
              (NOT (EQUAL (LENGTH LST) 0))
              (NUMBERP (LENGTH LST))
              (IMPLIES (AND (PROPERP EXTRA)
                            (NOT (LESSP (LENGTH (CDR LST)) (SUB1 N))))
                       (EQUAL (ROTATE (SUB1 N)
                                      (APPEND (CDR LST) EXTRA))
                              (APPEND (NTHCDR (SUB1 N) (CDR LST))
                                      (APPEND EXTRA
                                              (FIRSTN (SUB1 N) (CDR LST)))))))
         (IMPLIES (AND (PROPERP EXTRA)
                       (NOT (LESSP (LENGTH LST) N)))
                  (EQUAL (ROTATE N (APPEND LST EXTRA))
                         (APPEND (NTHCDR N LST)
                                 (APPEND EXTRA (FIRSTN N LST))))))
->: split   ;; Split into subgoals based on the propositional structure of the goal.


Creating 2 new subgoals, ((MAIN . 3) . 1) and ((MAIN . 3) . 2).

The proof of the current goal, (MAIN . 3), has been completed.  However, the
following subgoals of (MAIN . 3) remain to be proved: ((MAIN . 3) . 1) and
((MAIN . 3) . 2).
Now proving ((MAIN . 3) . 1).
->: th   ;; Mnemonic for ''theorem'' -- display the current goal.

*** Active top-level hypotheses:
H1.  (NOT (EQUAL N 0))
H2.  (NUMBERP N)
H3.  (NOT (EQUAL (LENGTH LST) 0))
H4.  (NUMBERP (LENGTH LST))
H5.  (EQUAL (ROTATE (SUB1 N)
                    (APPEND (CDR LST) EXTRA))
            (APPEND (NTHCDR (SUB1 N) (CDR LST))
                    (APPEND EXTRA
                            (FIRSTN (SUB1 N) (CDR LST)))))
H6.  (PROPERP EXTRA)
H7.  (NOT (LESSP (LENGTH LST) N))

*** Active governors:
There are no governors to display.

The current subterm is:
(EQUAL (ROTATE N (APPEND LST EXTRA))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
->: goals   ;; List the names of the unproved goals, top one first.

((MAIN . 3) . 1)
((MAIN . 3) . 2)
->: change-goal   ;; Change to the other goal; maybe it's easier!

Now proving ((MAIN . 3) . 2).
```

->: goals *;; Notice that the goals have indeed been switched.*

```
((MAIN . 3) . 2)
((MAIN . 3) . 1)
```
->: th *;; Mnemonic for ‘‘theorem’’ -- display the current goal.*

```
*** Active top-level hypotheses:
H1.  (NOT (EQUAL N 0))
H2.  (NUMBERP N)
H3.  (NOT (EQUAL (LENGTH LST) 0))
H4.  (NUMBERP (LENGTH LST))
H5.  (LESSP (LENGTH (CDR LST)) (SUB1 N))
H6.  (PROPERP EXTRA)
H7.  (NOT (LESSP (LENGTH LST) N))

*** Active governors:
There are no governors to display.

The current subterm is:
(EQUAL (ROTATE N (APPEND LST EXTRA))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
```
->: prove *;; Hypotheses 5 and 6 are contradictory.*

```
***** Now entering the theorem prover *****:

This simplifies, applying the lemma SUB1-ADD1, and unfolding LENGTH and LESSP,
to:

      T.

Q.E.D.


The current goal, ((MAIN . 3) . 2), has been proved, and has no dependents.
Now proving ((MAIN . 3) . 1).
```
->: goals *;; Only one goal remains, but it’s the hard one.*

```
((MAIN . 3) . 1)
```

```
->: th

*** Active top-level hypotheses:
H1.  (NOT (EQUAL N 0))
H2.  (NUMBERP N)
H3.  (NOT (EQUAL (LENGTH LST) 0))
H4.  (NUMBERP (LENGTH LST))
H5.  (EQUAL (ROTATE (SUB1 N)
                    (APPEND (CDR LST) EXTRA))
            (APPEND (NTHCDR (SUB1 N) (CDR LST))
                    (APPEND EXTRA
                            (FIRSTN (SUB1 N) (CDR LST)))))
H6.  (PROPERP EXTRA)
H7.  (NOT (LESSP (LENGTH LST) N))

*** Active governors:
There are no governors to display.

The current subterm is:
(EQUAL (ROTATE N (APPEND LST EXTRA))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
->: (dive 1 2)   ;; Focus on the subterm obtained by diving to the first argument of
                 ;; the EQUAL term and then to the second argument of the ROTATE term.

->: p   ;; Notice what the current subterm is now.

(APPEND LST EXTRA)
->: x   ;; Expand this call of APPEND and simplify.

->: p

(IF (LISTP LST)
    (CONS (CAR LST)
          (APPEND (CDR LST) EXTRA))
    EXTRA)
->: comm   ;; Since the term (LISTP LST) didn't simplify to T (true), we might want to
           ;; back up and claim it to hold.  But first we might wish to recall where we are.

The top-level commands thus far (in reverse order, i.e. last one first) have
been:
1. X
2. (DIVE 1 2)
3. PROVE
4. CHANGE-GOAL
5. SPLIT
6. PROVE
7. PROVE
8. INDUCT
9. START
->: (undo 2)   ;; Pop off the two top proof states.

Last instruction undone: (DIVE 1 2).
```

```
->: th

*** Active top-level hypotheses:
H1.  (NOT (EQUAL N 0))
H2.  (NUMBERP N)
H3.  (NOT (EQUAL (LENGTH LST) 0))
H4.  (NUMBERP (LENGTH LST))
H5.  (EQUAL (ROTATE (SUB1 N)
                    (APPEND (CDR LST) EXTRA))
            (APPEND (NTHCDR (SUB1 N) (CDR LST))
                    (APPEND EXTRA
                            (FIRSTN (SUB1 N) (CDR LST)))))
H6.  (PROPERP EXTRA)
H7.  (NOT (LESSP (LENGTH LST) N))

*** Active governors:
There are no governors to display.

The current subterm is:
(EQUAL (ROTATE N (APPEND LST EXTRA))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
->: (claim (listp lst))   ;; The theorem prover should be able to prove this, given hypothesis H3.


***** Now entering the theorem prover *****:

This formula simplifies, unfolding LENGTH and EQUAL, to:

      T.

Q.E.D.

->: (dive 1 2)   ;; same explanation as before

->: p

(APPEND LST EXTRA)
->: x   ;; same explanation as before

->: p   ;; Notice that this time the ‘‘test’’ (LISTP LST) of the IF term simplified to T.

(CONS (CAR LST)
      (APPEND (CDR LST) EXTRA))
->: pp-top   ;; Pretty-print from the top of the conclusion, highlighting the current term.

(EQUAL (ROTATE N
               (*** (CONS (CAR LST)
                          (APPEND (CDR LST) EXTRA))
                    ***))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
->: up   ;; Move up one level, so that we can expand the call of ROTATE.
```

```
->: pp-top

(EQUAL (*** (ROTATE N
                    (CONS (CAR LST)
                          (APPEND (CDR LST) EXTRA)))
            ***)
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
->: x

->: p

(ROTATE (SUB1 N)
        (APPEND (APPEND (CDR LST) EXTRA)
                (LIST (CAR LST))))
```

->: top *;; Move to the top of the conclusion, so that the current term is the entire conclusion.*

```
->: th

*** Active top-level hypotheses:
H1.  (NOT (EQUAL N 0))
H2.  (NUMBERP N)
H3.  (NOT (EQUAL (LENGTH LST) 0))
H4.  (NUMBERP (LENGTH LST))
H5.  (EQUAL (ROTATE (SUB1 N)
                    (APPEND (CDR LST) EXTRA))
            (APPEND (NTHCDR (SUB1 N) (CDR LST))
                    (APPEND EXTRA
                            (FIRSTN (SUB1 N) (CDR LST)))))
H6.  (PROPERP EXTRA)
H7.  (NOT (LESSP (LENGTH LST) N))
H8.  (LISTP LST)

*** Active governors:
There are no governors to display.

The current subterm is:
(EQUAL (ROTATE (SUB1 N)
               (APPEND (APPEND (CDR LST) EXTRA)
                       (LIST (CAR LST))))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
```

->: (s lemmas) *;; Simplify using rewrite rules, which including the associativity of* **APPEND**.

```
->: th

*** Active top-level hypotheses:
H1.  (NOT (EQUAL N 0))
H2.  (NUMBERP N)
H3.  (NOT (EQUAL (LENGTH LST) 0))
H4.  (NUMBERP (LENGTH LST))
H5.  (EQUAL (ROTATE (SUB1 N)
                    (APPEND (CDR LST) EXTRA))
            (APPEND (NTHCDR (SUB1 N) (CDR LST))
                    (APPEND EXTRA
                            (FIRSTN (SUB1 N) (CDR LST)))))
H6.  (PROPERP EXTRA)
H7.  (NOT (LESSP (LENGTH LST) N))
H8.  (LISTP LST)

*** Active governors:
There are no governors to display.

The current subterm is:
(EQUAL (ROTATE (SUB1 N)
               (APPEND (CDR LST)
                       (APPEND EXTRA (LIST (CAR LST)))))
       (APPEND (NTHCDR N LST)
               (APPEND EXTRA (FIRSTN N LST))))
->: exit   ;; Let's quit, now that we see that the inductive hypothesis (H5) isn't quite what we want.


Quitting the interactive proof checker.  Submit (VERIFY) to get back in at
this state.  **NOTE** -- No event has been stored.
NIL

>
```

Recall the induction scheme printed at the outset of the proof above:

```
(AND (IMPLIES (OR (EQUAL N 0) (NOT (NUMBERP N)))
              (p EXTRA LST N))
     (IMPLIES (AND (NOT (OR (EQUAL N 0) (NOT (NUMBERP N))))
                   (OR (EQUAL (LENGTH LST) 0)
                       (NOT (NUMBERP (LENGTH LST)))))
              (p EXTRA LST N))
     (IMPLIES (AND (NOT (OR (EQUAL N 0) (NOT (NUMBERP N))))
                   (NOT (OR (EQUAL (LENGTH LST) 0)
                            (NOT (NUMBERP (LENGTH LST)))))
                   (p EXTRA (CDR LST) (SUB1 N)))
              (p EXTRA LST N)))
```

The interactive proof attempt shows that we should perhaps modify this scheme slightly. We indicate the changes in lower case below.

```
(AND (IMPLIES (OR (EQUAL N 0) (NOT (NUMBERP N)))
              (p EXTRA LST N))
     (IMPLIES (AND (NOT (OR (EQUAL N 0) (NOT (NUMBERP N))))
                   (OR (EQUAL (LENGTH LST) 0)
                       (NOT (NUMBERP (LENGTH LST)))))
              (p EXTRA LST N))
     (IMPLIES (AND (NOT (OR (EQUAL N 0) (NOT (NUMBERP N))))
                   (NOT (OR (EQUAL (LENGTH LST) 0)
                            (NOT (NUMBERP (LENGTH LST)))))
                   (p (append EXTRA (list (car lst)))
                      (CDR LST) (SUB1 N)))
              (p EXTRA LST N)))
```

How do we instruct the theorem prover to induct in the manner suggested above? We define a function whose recursion is ‘‘analogous’’ to this scheme, as the following example will illustrate. Later we’ll give a hint to the theorem prover to use the induction scheme suggested by this definition.

```
(DEFN ROTATE-APPEND-INDUCTION (N LST EXTRA)
  (IF (OR (EQUAL N 0) (NOT (NUMBERP N)))
      T
    (IF (NLISTP LST)
        T
      (ROTATE-APPEND-INDUCTION
       (SUB1 N)
       (CDR LST)
       (APPEND EXTRA (LIST (CAR LST)))))))
```

Another attempt at proving ROTATE-APPEND now leads to a term of the form

```
(IMPLIES (AND (NUMBERP V)
              (NOT (PROPERP (APPEND EXTRA (LIST X))))
              (PROPERP EXTRA)
              (NOT (LESSP (LENGTH Z) V)))
         .....)
```

which is absurd since the second hypothesis must fail. So we ask the theorem prover to prove the following rewrite rule in order to help it with the proof of ROTATE-APPEND.

```
(PROVE-LEMMA PROPERP-APPEND (REWRITE)
  (EQUAL (PROPERP (APPEND X Y))
         (PROPERP Y)))
```

Now the proof of ROTATE-APPEND (with an appropriate hint to induct as in the definition of **ROTATE-APPEND-INDUCTION**) succeeds. That’s because the problematic term **(NOT (PROPERP (APPEND EXTRA (LIST X))))**, discussed above, is rewritten to **(NOT (PROPERP (LIST X)))** (using the rule PROPERP-APPEND that is displayed immediately above), and the theorem prover’s simplifier can simplify this to **F** (*false*). Notice that there is no point in making ROTATE-APPEND a rewrite rule, since we’ll just instantiate it in order to get the the main theorem ROTATE-LENGTH proved. (The **APPEND** term in **(ROTATE N (APPEND LST EXTRA))** prevents it from being used automatically in the proof of ROTATE-LENGTH.)

```
(PROVE-LEMMA ROTATE-APPEND ()
  (IMPLIES (AND (PROPERP EXTRA)
                (NOT (LESSP (LENGTH LST) N)))
           (EQUAL (ROTATE N (APPEND LST EXTRA))
                  (APPEND (NTHCDR N LST)
                          (APPEND EXTRA (FIRSTN N LST)))))
  ((INDUCT (ROTATE-APPEND-INDUCTION N LST EXTRA))))
```

Now we can try to get the theorem prover to prove the main theorem, by giving it an appropriate hint. Unfortunately, there seems to be some problem, as we now see.

```
>(prove-lemma rotate-length ()
  (implies (properp lst)
           (equal (rotate (length lst) lst)
                  lst))
  ((use (rotate-append (extra nil) (n (length lst))))))

This conjecture simplifies, applying the lemma APPEND-NIL, and expanding the
definitions of PROPERP, NOT, AND, LISTP, APPEND, and IMPLIES, to the following
two new formulas:

Case 2. (IMPLIES (AND (LESSP (LENGTH LST) (LENGTH LST))
                      (PROPERP LST))
                 (EQUAL (ROTATE (LENGTH LST) LST)
                        LST)).

  However this again simplifies, using linear arithmetic, to:

        T.

Case 1. (IMPLIES (AND (EQUAL (ROTATE (LENGTH LST) LST)
                             (APPEND (NTHCDR (LENGTH LST) LST)
                                     (FIRSTN (LENGTH LST) LST)))
                      (PROPERP LST))
                 (EQUAL (ROTATE (LENGTH LST) LST)
                        LST)).

  We use the above equality hypothesis by substituting:
        (APPEND (NTHCDR (LENGTH LST) LST)
                (FIRSTN (LENGTH LST) LST))
  for (ROTATE (LENGTH LST) LST) and keeping the equality hypothesis.  The
  result is:

        (IMPLIES (AND (EQUAL (ROTATE (LENGTH LST) LST)
                             (APPEND (NTHCDR (LENGTH LST) LST)
                                     (FIRSTN (LENGTH LST) LST)))
                      (PROPERP LST))
                 (EQUAL (APPEND (NTHCDR (LENGTH LST) LST)
                                (FIRSTN (LENGTH LST) LST))
                        LST)).
```

```
  This further simplifies, clearly, to:

        (IMPLIES (AND (EQUAL (ROTATE (LENGTH LST) LST)
                             (APPEND (NTHCDR (LENGTH LST) LST)
                                     (FIRSTN (LENGTH LST) LST)))
                      (PROPERP LST))
                 (EQUAL (ROTATE (LENGTH LST) LST)
                        LST)),

  which we would normally push and work on later by induction.  But if we must
  use induction to prove the input conjecture, we prefer to induct on the
  original formulation of the problem.  Thus we will disregard all that we
  have previously done, give the name *1 to the original input, and work on
  it.

[.... proof aborted by me ....]
```

The goal shown immediately above would be sufficient if only the theorem prover knew the following two obvious facts, which we'll ask it to prove and to store as rewrite rules. These can be proved in less than a second each on a Sun3/60.

```
(PROVE-LEMMA NTHCDR-LENGTH (REWRITE)
  (IMPLIES (PROPERP LST)
           (EQUAL (NTHCDR (LENGTH LST) LST)
                  NIL)))

(PROVE-LEMMA FIRSTN-LENGTH (REWRITE)
  (IMPLIES (PROPERP LST)
           (EQUAL (FIRSTN (LENGTH LST) LST)
                  LST)))
```

Finally the theorem prover is able to prove our main theorem.

```
(PROVE-LEMMA ROTATE-LENGTH ()
  (IMPLIES (PROPERP LST)
           (EQUAL (ROTATE (LENGTH LST) LST)
                  LST))
  ((USE (ROTATE-APPEND (EXTRA NIL) (N (LENGTH LST))))))
```

Appendix A
List of events

Here is the sequence of events presented above.

```
(DEFN ROTATE (N LST)
  (IF (ZEROP N)
      LST
    (ROTATE (SUB1 N)
            (APPEND (CDR LST) (LIST (CAR LST))))))

(DEFN LENGTH (X)
  (IF (LISTP X)
      (ADD1 (LENGTH (CDR X)))
    0))

(DEFN PROPERP (X)
  (IF (LISTP X)
      (PROPERP (CDR X))
    (EQUAL X NIL)))

(DEFN FIRSTN (N LST)
  (IF (ZEROP N)
      NIL
    (CONS (CAR LST)
          (FIRSTN (SUB1 N) (CDR LST)))))

(DEFN NTHCDR (N LST)
  (IF (ZEROP N)
      LST
    (NTHCDR (SUB1 N) (CDR LST))))

(PROVE-LEMMA ASSOCIATIVITY-OF-APPEND (REWRITE)
  (EQUAL (APPEND (APPEND X Y) Z)
         (APPEND X (APPEND Y Z))))

(PROVE-LEMMA APPEND-NIL (REWRITE)
  (IMPLIES (PROPERP X)
           (EQUAL (APPEND X NIL)
                  X)))

(DEFN ROTATE-APPEND-INDUCTION (N LST EXTRA)
  (IF (OR (EQUAL N 0) (NOT (NUMBERP N)))
      T
    (IF (NLISTP LST)
        T
      (ROTATE-APPEND-INDUCTION
       (SUB1 N)
       (CDR LST)
       (APPEND EXTRA (LIST (CAR LST)))))))
```

```
(PROVE-LEMMA PROPERP-APPEND (REWRITE)
  (EQUAL (PROPERP (APPEND X Y))
         (PROPERP Y)))

(PROVE-LEMMA ROTATE-APPEND ()
  (IMPLIES (AND (PROPERP EXTRA)
                (NOT (LESSP (LENGTH LST) N)))
           (EQUAL (ROTATE N (APPEND LST EXTRA))
                  (APPEND (NTHCDR N LST)
                          (APPEND EXTRA (FIRSTN N LST)))))
  ((INDUCT (ROTATE-APPEND-INDUCTION N LST EXTRA))))

(PROVE-LEMMA NTHCDR-LENGTH (REWRITE)
  (IMPLIES (PROPERP LST)
           (EQUAL (NTHCDR (LENGTH LST) LST)
                  NIL)))

(PROVE-LEMMA FIRSTN-LENGTH (REWRITE)
  (IMPLIES (PROPERP LST)
           (EQUAL (FIRSTN (LENGTH LST) LST)
                  LST)))

(PROVE-LEMMA ROTATE-LENGTH ()
  (IMPLIES (PROPERP LST)
           (EQUAL (ROTATE (LENGTH LST) LST)
                  LST))
  ((USE (ROTATE-APPEND (EXTRA NIL) (N (LENGTH LST))))))
```

Appendix B
An alternate approach

Here is a list of events that suggests an alternate approach to the proof, in which we do not consider what happens when one rotates the **APPEND** of a list with some ''extra'' list. The definitions of **ROTATE**, **LENGTH**, **PROPERP**, **FIRSTN**, and **NTHCDR** are the same as in the proof described above, as are the statements of the lemmas PROPERP-APPEND, APPEND-NIL, ASSOCIATIVITY-OF-APPEND, NTHCDR-LENGTH, and FIRSTN-LENGTH. However, we include those too, for completeness. Note: a replay of these 16 events on a Sun3/60 with 20 megabytes main memory took just over a half minute (real time), while a replay of the original 13 events described in the text above took about 20 seconds real time.

```
(DEFN ROTATE (N LST)   ;; same as before
  (IF (ZEROP N)
      LST
      (ROTATE (SUB1 N)
              (APPEND (CDR LST) (LIST (CAR LST))))))

(DEFN LENGTH (X)   ;; same as before
  (IF (LISTP X)
      (ADD1 (LENGTH (CDR X)))
      0))

(DEFN PROPERP (X)   ;; same as before
  (IF (LISTP X)
      (PROPERP (CDR X))
      (EQUAL X NIL)))

(DEFN FIRSTN (N LST)   ;; same as before
  (IF (ZEROP N)
      NIL
      (CONS (CAR LST)
            (FIRSTN (SUB1 N) (CDR LST)))))

(DEFN NTHCDR (N LST)   ;; same as before
  (IF (ZEROP N)
      LST
      (NTHCDR (SUB1 N) (CDR LST))))

(DEFN ROTATE-INDUCTION (N LST)
  ;; This function suggests the induction carried out in the informal proof above.
  (IF (ZEROP N)
      T
      (ROTATE-INDUCTION (SUB1 N)
                        (APPEND (CDR LST) (LIST (CAR LST))))))

(PROVE-LEMMA PROPERP-APPEND (REWRITE)   ;; same as before
  (EQUAL (PROPERP (APPEND X Y))
         (PROPERP Y)))
```

```
(PROVE-LEMMA APPEND-NIL (REWRITE)   ;; same as before
  (IMPLIES (PROPERP X)
           (EQUAL (APPEND X NIL) X)))

(PROVE-LEMMA ASSOCIATIVITY-OF-APPEND (REWRITE)   ;; same as before
  (EQUAL (APPEND (APPEND X Y) Z)
         (APPEND X (APPEND Y Z))))
```

;; The need for the following 3 lemmas was suggested by examining failed proofs of ROTATE-LENGTH-LEMMA.

```
(PROVE-LEMMA LENGTH-APPEND (REWRITE)
  (EQUAL (LENGTH (APPEND X Y))
         (PLUS (LENGTH X) (LENGTH Y))))

(PROVE-LEMMA LEMMA-1 (REWRITE)
  (IMPLIES (NOT (LESSP (LENGTH Z) N))
           (EQUAL (NTHCDR N (APPEND Z Y))
                  (APPEND (NTHCDR N Z) Y))))

(PROVE-LEMMA LEMMA-2 (REWRITE)
  (IMPLIES (NOT (LESSP (LENGTH Z) X))
           (EQUAL (FIRSTN X (APPEND Z Y))
                  (FIRSTN X Z))))

(PROVE-LEMMA ROTATE-LENGTH-LEMMA (REWRITE)
  (IMPLIES (AND (PROPERP LST)
                (NOT (LESSP (LENGTH LST) N)))
           (EQUAL (ROTATE N LST)
                  (APPEND (NTHCDR N LST)
                          (FIRSTN N LST))))
  ((INDUCT (ROTATE-INDUCTION N LST))))

(PROVE-LEMMA NTHCDR-LENGTH (REWRITE)   ;; same as before
  (IMPLIES (PROPERP LST)
           (EQUAL (NTHCDR (LENGTH LST) LST)
                  NIL)))

(PROVE-LEMMA FIRSTN-LENGTH (REWRITE)   ;; same as before
  (IMPLIES (PROPERP LST)
           (EQUAL (FIRSTN (LENGTH LST) LST)
                  LST)))

(PROVE-LEMMA ROTATE-LENGTH NIL   ;; same as before
  (IMPLIES (PROPERP LST)
           (EQUAL (ROTATE (LENGTH LST) LST)
                  LST)))
```

References

1. R. S. Boyer and J S. Moore, *A Computational Logic Handbook,* Academic Press, Boston, 1988.

2. Matt Kaufmann, ‘‘A User’s Manual for an Interactive Enhancement to the Boyer-Moore Theorem Prover’’, Technical Report 19, Computational Logic, Inc., May 1988.

3. M. Kaufmann, ‘‘Addition of Free Variables to an Interactive Enchancement of the Boyer-Moore Theorem Prover’’, Tech. report 42, Computational Logic, Inc., 1717 West Sixth Street, Suite 290 Austin, TX 78703, 1989.

An Instructive Example
for Beginning Users of
the Boyer-Moore Theorem Prover

Table of Contents