More Proof Examples in the ACL2 Theorem Prover

Last time, we started to look at how the ACL2 theorem prover can be used to prove the kind of theorems that we have been proving, and yielding the same kind of proofs that we have been getting.

We tested it on simple examples, and for those, all went well.

Today we look at more interesting examples, again proving theorems we have proved in class, and compare the ACL2 proofs from those that we did by hand.

Let's first define `rev`.

```
ACL2 > (defun rev (x)
         (if (endp x)
           NIL
           (app (rev (cdr x)) (list (car x)))))

The admission of REV is trivial, using the following CCMs:
(REV CCG (ACL2-COUNT X)).    We observe that the type of REV is described
by the theorem (OR (CONSP (REV X)) (EQUAL (REV X) NIL)).  We used primitive
type reasoning and the :type-prescription rule APP.

Summary
Form:  ( DEFUN REV ...)
Rules: NIL
Warnings:  None
Time:  0.00 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other:
   0.00)
 REV
```

Let's warm ourselves up, and prove that `(rev x)` is always a true list. That should be a simple proof by induction, and it is.

```
ACL2 > (defthm true-listp-rev
         (true-listp (rev x)))

^^^ Checkpoint Goal ^^^
```

```
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 (REV X).  This suggestion
was produced using the :induction rule REV.  If we let (:P X) denote
*1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (TRUE-LISTP (REV (CDR X))))
         (TRUE-LISTP (REV X))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/2'
(IMPLIES (AND (CONSP X)
              (TRUE-LISTP (REV (CDR X))))
         (TRUE-LISTP (REV X))).

But simplification reduces this to T, using the :definition REV, primitive
type reasoning and the :rewrite rule TRUE-LISTP-APP.

Subgoal *1/1
(IMPLIES (ENDP X) (TRUE-LISTP (REV X))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (NOT (CONSP X))
         (TRUE-LISTP (REV X))).

But simplification reduces this to T, using the :definition REV and
```

```
the :executable-counterpart of TRUE-LISTP.

That completes the proof of *1.

Q.E.D.

The storage of TRUE-LISTP-REV depends upon the :type-prescription rule
TRUE-LISTP.

Summary
Form:  ( DEFTHM TRUE-LISTP-REV ...)
Rules: ((:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:EXECUTABLE-COUNTERPART TRUE-LISTP)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION REV)
        (:REWRITE TRUE-LISTP-APP)
        (:TYPE-PRESCRIPTION TRUE-LISTP))
Warnings:  None
Time:  0.01 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other:
   0.00)
 TRUE-LISTP-REV
```

A completely straightforward proof by induction. Make sure you understand the above.

Now, let's try to prove something that is *not* provable. Remember trying to reverse a list twice: that only gives you back the original list if that original list was in fact a true list. So trying to prove `(= (rev (rev x)) x)` should fail.

```
ACL2 > (thm (= (rev (rev x)) x))

By the simple :definition = we reduce the conjecture to

Goal'
(EQUAL (REV (REV X)) X).
^^^ Checkpoint Goal' ^^^

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 (REV X).  This suggestion
was produced using the :induction rule REV.  If we let (:P X) denote
*1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (REV (CDR X))) (CDR X)))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/2'
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (REV (CDR X))) (CDR X)))
         (EQUAL (REV (REV X)) X)).

This simplifies, using the :definition REV, to

Subgoal *1/2''
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (REV (CDR X))) (CDR X)))
         (EQUAL (REV (APP (REV (CDR X)) (LIST (CAR X))))
                X)).

The destructor terms (CAR X) and (CDR X) can be eliminated by using
CAR-CDR-ELIM to replace X by (CONS X1 X2), (CAR X) by X1 and (CDR X)
by X2.  This produces the following goal.

Subgoal *1/2'''
(IMPLIES (AND (CONSP (CONS X1 X2))
              (EQUAL (REV (REV X2)) X2))
         (EQUAL (REV (APP (REV X2) (LIST X1)))
                (CONS X1 X2))).

This simplifies, using primitive type reasoning, to
```

```
Subgoal *1/2'4'
(IMPLIES (EQUAL (REV (REV X2)) X2)
         (EQUAL (REV (APP (REV X2) (LIST X1)))
                (CONS X1 X2))).
^^^ Checkpoint Subgoal *1/2'4' ^^^

We now use the hypothesis by cross-fertilizing (REV (REV X2)) for X2
and throwing away the hypothesis.  This produces

Subgoal *1/2'5'
(EQUAL (REV (APP (REV X2) (LIST X1)))
       (CONS X1 (REV (REV X2)))).
^^^ Checkpoint Subgoal *1/2'5' ^^^

We generalize this conjecture, replacing (REV X2) by RV.  This produces

Subgoal *1/2'6'
(EQUAL (REV (APP RV (LIST X1)))
       (CONS X1 (REV RV))).
^^^ Checkpoint Subgoal *1/2'6' ^^^

Name the formula above *1.1.

Subgoal *1/1
(IMPLIES (ENDP X)
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (NOT (CONSP X))
         (EQUAL (REV (REV X)) X)).

This simplifies, using the :definition REV, the :executable-counterpart
of REV and primitive type reasoning, to

Subgoal *1/1''
(IMPLIES (NOT (CONSP X)) (NOT X)).
^^^ Checkpoint Subgoal *1/1'' ^^^

Name the formula above *1.2.
```

```
No induction schemes are suggested by *1.2.  Consequently, the proof
attempt has failed.

Summary
Form:  ( THM ...)
Rules: ((:DEFINITION =)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:ELIM CAR-CDR-ELIM)
        (:EXECUTABLE-COUNTERPART REV)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION REV))
Warnings:  None
Time:  0.03 seconds (prove: 0.01, print: 0.01, proof tree: 0.01, other:
   0.00)

---
The key checkpoint goals, below, may help you to debug this failure.
See :DOC failure and see :DOC set-checkpoint-summary-limit.
---

*** Key checkpoint at the top level: ***

Goal'
(EQUAL (REV (REV X)) X)

*** Key checkpoints under a top-level induction: ***

Subgoal *1/2'4'
(IMPLIES (EQUAL (REV (REV X2)) X2)
         (EQUAL (REV (APP (REV X2) (LIST X1)))
                (CONS X1 X2)))

Subgoal *1/1''
(IMPLIES (NOT (CONSP X)) (NOT X))

******** FAILED ********  See :DOC failure  ******** FAILED ********
```

When a proof fail, it could be for two reasons: either the prover was not clever or lucky enough to find the right proof, or you asked it to prove something that was not valid. This is

what is happening here. In fact, if you look at what it does trying to prove subgoal `*1/1`, that is, the base case of the induction, it gets stucks trying to prove ¬`(consp x)` ⟹ `(not x)`, and for good reason: that's not a valid formula (try `x`=5). This suggests the fix, in fact. The only way in which a ¬`(consp x)` can make `(not x)` true is if `x` is `NIL`, that is, if `x` is a true list.

Let's correct the theorem and prove it.

```
ACL2 > (thm (implies (true-listp x)
                     (= (rev (rev x)) x)))

By the simple :definition = we reduce the conjecture to

Goal'
(IMPLIES (TRUE-LISTP X)
         (EQUAL (REV (REV X)) X)).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.

Perhaps we can prove *1 by induction.  Two induction schemes are suggested
by this conjecture.  These merge into one derived induction scheme.

We will induct according to a scheme suggested by (REV X).  This suggestion
was produced using the :induction rules REV and TRUE-LISTP.  If we
let (:P X) denote *1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
three nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/3
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/3'
```

```
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

This simplifies, using the :definitions REV and TRUE-LISTP, to

Subgoal *1/3''
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP (CDR X)))
         (EQUAL (REV (APP (REV (CDR X)) (LIST (CAR X))))
                X)).

The destructor terms (CAR X) and (CDR X) can be eliminated by using
CAR-CDR-ELIM to replace X by (CONS X1 X2), (CAR X) by X1 and (CDR X)
by X2.  This produces the following goal.

Subgoal *1/3'''
(IMPLIES (AND (CONSP (CONS X1 X2))
              (EQUAL (REV (REV X2)) X2)
              (TRUE-LISTP X2))
         (EQUAL (REV (APP (REV X2) (LIST X1)))
                (CONS X1 X2))).

This simplifies, using primitive type reasoning, to

Subgoal *1/3'4'
(IMPLIES (AND (EQUAL (REV (REV X2)) X2)
              (TRUE-LISTP X2))
         (EQUAL (REV (APP (REV X2) (LIST X1)))
                (CONS X1 X2))).
^^^ Checkpoint Subgoal *1/3'4' ^^^

We now use the first hypothesis by cross-fertilizing (REV (REV X2))
for X2 and throwing away the hypothesis.  This produces

Subgoal *1/3'5'
(IMPLIES (TRUE-LISTP X2)
         (EQUAL (REV (APP (REV X2) (LIST X1)))
                (CONS X1 (REV (REV X2))))).
^^^ Checkpoint Subgoal *1/3'5' ^^^
```

```
We generalize this conjecture, replacing (REV X2) by RV.  This produces

Subgoal *1/3'6'
(IMPLIES (TRUE-LISTP X2)
         (EQUAL (REV (APP RV (LIST X1)))
                (CONS X1 (REV RV)))).
^^^ Checkpoint Subgoal *1/3'6' ^^^

We suspect that the term (TRUE-LISTP X2) is irrelevant to the truth
of this conjecture and throw it out.  We will thus try to prove

Subgoal *1/3'7'
(EQUAL (REV (APP RV (LIST X1)))
       (CONS X1 (REV RV))).
^^^ Checkpoint Subgoal *1/3'7' ^^^

Name the formula above *1.1.

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (NOT (TRUE-LISTP (CDR X)))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But we reduce the conjecture to T, by primitive type reasoning.

Subgoal *1/1
(IMPLIES (AND (ENDP X) (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (AND (NOT (CONSP X)) (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But simplification reduces this to T, using the :definition TRUE-LISTP,
the :executable-counterparts of CONSP, EQUAL and REV and primitive
type reasoning.

So we now return to *1.1, which is
```

```
(EQUAL (REV (APP RV (LIST X1)))
       (CONS X1 (REV RV))).

Perhaps we can prove *1.1 by induction.  Two induction schemes are
suggested by this conjecture.  Subsumption reduces that number to one.

We will induct according to a scheme suggested by (REV RV).  This suggestion
was produced using the :induction rules APP and REV.  If we let (:P RV X1)
denote *1.1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP RV)) (:P (CDR RV) X1))
              (:P RV X1))
     (IMPLIES (ENDP RV) (:P RV X1))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1.1 ^^^

Subgoal *1.1/2
(IMPLIES (AND (NOT (ENDP RV))
              (EQUAL (REV (APP (CDR RV) (LIST X1)))
                     (CONS X1 (REV (CDR RV)))))
         (EQUAL (REV (APP RV (LIST X1)))
                (CONS X1 (REV RV)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1.1/2'
(IMPLIES (AND (CONSP RV)
              (EQUAL (REV (APP (CDR RV) (LIST X1)))
                     (CONS X1 (REV (CDR RV)))))
         (EQUAL (REV (APP RV (LIST X1)))
                (CONS X1 (REV RV)))).

But simplification reduces this to T, using the :definitions APP and
REV, primitive type reasoning and the :rewrite rules CAR-CONS and CDR-
CONS.

Subgoal *1.1/1
(IMPLIES (ENDP RV)
         (EQUAL (REV (APP RV (LIST X1)))
                (CONS X1 (REV RV)))).
```

```
By the simple :definition ENDP we reduce the conjecture to

Subgoal *1.1/1'
(IMPLIES (NOT (CONSP RV))
         (EQUAL (REV (APP RV (LIST X1)))
                (CONS X1 (REV RV)))).

But simplification reduces this to T, using the :definitions APP and
REV, the :executable-counterparts of CONSP and REV, primitive type
reasoning and the :rewrite rules CAR-CONS and CDR-CONS.

That completes the proofs of *1.1 and *1.

Q.E.D.

Summary
Form:  ( THM ...)
Rules: ((:DEFINITION =)
        (:DEFINITION APP)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:DEFINITION TRUE-LISTP)
        (:ELIM CAR-CDR-ELIM)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART EQUAL)
        (:EXECUTABLE-COUNTERPART REV)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION APP)
        (:INDUCTION REV)
        (:INDUCTION TRUE-LISTP)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS))
Warnings:  None
Time:  0.03 seconds (prove: 0.01, print: 0.01, proof tree: 0.01, other:
   0.00)

Proof succeeded.
```

So the proof succeeds. But not quite for the right reasons. In particular, if you look carefully, there are two inductions done in the proof, and one generalization at checkpoint `*1/3'5'`.

That means that ACL2 was "guessing" something to make it unstuck, and here it happened to be lucky, but the result formula:

```
(EQUAL (REV (APP RV (LIST X1)))
       (CONS X1 (REV RV))).
```

proved too hard to prove there on the spot, so the prover stored it away as a goal to come back to later, calling it `*1.1`, and returning to it later and proving it by induction. Before looking at this carefully, let's first figure out why there are *three* proof obligations for the main induction, which is the standard induction scheme.

Here are the standard proof obligations we get:

P1. `(endp x)` $\wedge$ `(true-listp x)` $\Longrightarrow$ `(= (rev (rev x)) x)`

P2. $\neg$`(endp x)`
$\wedge$ (`(true-listp (cdr x))` $\Longrightarrow$ `(= (rev (rev (cdr x))) (cdr x))`)
$\wedge$ `(true-listp x)`
$\Longrightarrow$ `(= (rev (rev x)) x)`

That's what we've been doing.[1] ACL2 pushes this one step further. Recall from propositional reasoning that $\varphi \Longrightarrow \psi$ is equivalent to $\neg\varphi \vee \psi$. So let's modify P2 accordingly:

$\neg$`(endp x)`
$\wedge$ ($\neg$`(true-listp (cdr x))` $\vee$ `(= (rev (rev (cdr x))) (cdr x))`)
$\wedge$ `(true-listp x)`
$\Longrightarrow$ `(= (rev (rev x)) x)`

Now, we can distribute conjunction over negation, and simplify. Our new version of P2 has the form $A \wedge (B \vee C) \wedge D \Longrightarrow E$, which is equivalent to $((A \wedge B \wedge D) \vee (A \wedge C \wedge D)) \Longrightarrow E$. But this is itself equivalent to $(A \wedge B \wedge D \Longrightarrow E) \wedge (A \wedge C \wedge D \Longrightarrow E)$. So we can split P2 into the two proof obligations:

P2a. $\neg$`(endp x)`
$\wedge$ $\neg$`(true-listp (cdr x))`
$\wedge$ `(true-listp x)`
$\Longrightarrow$ `(= (rev (rev x)) x)`

P2b. $\neg$`(endp x)`
$\wedge$ `(= (rev (rev (cdr x))) (cdr x))`
$\wedge$ `(true-listp x)`
$\Longrightarrow$ `(= (rev (rev x)) x)`

[1] Note that I messed up this part in lecture—I completely forgot to maintain the context.

And ACL2 tries to prove the three proof obligations P1, P2a, and P2b.

When trying to prove P2b, however, it gets stuck, and tries to generalize. Now, that generalization happens to be a lucky generalization, and the proof goes through eventually, but it requires a sub-induction—ACL2 reaches a point where it needs to prove a formula which it calls `*1.1`, saves it on the stack to take care of the other proof obligations, and eventually returns to it, and proves it by induction.

One thing we will try to do when doing our proofs is avoid having ACL2 generalize. That's because generalization is fraught with uncertainty—generalization is hard, and usually getting a good generalization depends on an understanding of the formula that we are trying to prove. And second, when ACL2 tries to generalize, that's usually an indication that there is a re-usable lemma somewhere that we have forgotten to identify. So we will make sure that all of our ACL2 proofs do not use generalization, and moreover that all of our ACL2 inductive proofs never use more than a single induction. This will force us to identify all the lemmas that are needed to push the proof through. Note that this is exactly the technique we advocated for paper proofs—identify lemmas when needed, and prove them.

So what is the lemma that ACL2 wants to prove in the above proof of the double-reverse formula? This is what ACL2 gets stuck on:

```
(= (rev (app x (list a)))
   (cons a (rev x)))
```

Looks reasonable enough, and moreover, looks like something which might be useful other places. So let's prove it, making sure that we use `defthm` so that we can use the result later in the proof of the double-reverse formula.

```
ACL2 > (defthm rev-app-cons
         (= (rev app x (list a))
            (cons a (rev x))))

By the simple :definition = we reduce the conjecture to

Goal'
(EQUAL (REV (APP X (LIST A)))
       (CONS A (REV X))).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.

Perhaps we can prove *1 by induction.  Two induction schemes are suggested
by this conjecture.  Subsumption reduces that number to one.

We will induct according to a scheme suggested by (REV X).  This suggestion
```

```
was produced using the :induction rules APP and REV.  If we let (:P A X)
denote *1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P A (CDR X)))
              (:P A X))
     (IMPLIES (ENDP X) (:P A X))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (APP (CDR X) (LIST A)))
                     (CONS A (REV (CDR X)))))
         (EQUAL (REV (APP X (LIST A)))
                (CONS A (REV X)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/2'
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (APP (CDR X) (LIST A)))
                     (CONS A (REV (CDR X)))))
         (EQUAL (REV (APP X (LIST A)))
                (CONS A (REV X)))).

But simplification reduces this to T, using the :definitions APP and
REV, primitive type reasoning and the :rewrite rules CAR-CONS and CDR-
CONS.

Subgoal *1/1
(IMPLIES (ENDP X)
         (EQUAL (REV (APP X (LIST A)))
                (CONS A (REV X)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (NOT (CONSP X))
         (EQUAL (REV (APP X (LIST A)))
                (CONS A (REV X)))).
```

```
But simplification reduces this to T, using the :definitions APP and
REV, the :executable-counterparts of CONSP and REV, primitive type
reasoning and the :rewrite rules CAR-CONS and CDR-CONS.

That completes the proof of *1.

Q.E.D.

Summary
Form:  ( DEFTHM REV-APP-CONS ...)
Rules: ((:DEFINITION =)
        (:DEFINITION APP)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART REV)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION APP)
        (:INDUCTION REV)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS))
Warnings:  None
Time:  0.01 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other:
   0.00)
 REV-APP-CONS
```

Happily, the proof goes through completely cleanly, in a single induction over `x`.

Now, let's prove the double-reverse formula again, see if ACL2 can use our lemma.

```
ACL2 > (defthm rev-rev
         (implies (true-listp x)
                  (= (rev (rev x)) x)))

By the simple :definition = we reduce the conjecture to

Goal'
(IMPLIES (TRUE-LISTP X)
         (EQUAL (REV (REV X)) X)).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.
```

```
Perhaps we can prove *1 by induction.  Two induction schemes are suggested
by this conjecture.  These merge into one derived induction scheme.

We will induct according to a scheme suggested by (REV X).  This suggestion
was produced using the :induction rules REV and TRUE-LISTP.  If we
let (:P X) denote *1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
three nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/3
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/3'
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But simplification reduces this to T, using the :definitions REV and
TRUE-LISTP, primitive type reasoning and the :rewrite rules CONS-CAR-
CDR and REV-APP-CONS.

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (NOT (TRUE-LISTP (CDR X)))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But we reduce the conjecture to T, by primitive type reasoning.

Subgoal *1/1
```

```
(IMPLIES (AND (ENDP X) (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (AND (NOT (CONSP X)) (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But simplification reduces this to T, using the :definition TRUE-LISTP,
the :executable-counterparts of CONSP, EQUAL and REV and primitive
type reasoning.

That completes the proof of *1.

Q.E.D.

Summary
Form:  ( DEFTHM REV-REV ...)
Rules: ((:DEFINITION =)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:DEFINITION TRUE-LISTP)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART EQUAL)
        (:EXECUTABLE-COUNTERPART REV)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION REV)
        (:INDUCTION TRUE-LISTP)
        (:REWRITE CONS-CAR-CDR)
        (:REWRITE REV-APP-CONS))
Warnings:  None
Time:  0.01 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other:
   0.00)
 REV-REV
```

Voilà. In the proof of the `*1/3` obligation of the induction, we see that the prover used `rev-app-cons`. And it did not need generalization, or to use a nested induction. The whole proof goes through in a clean single induction.

Now, how did we prove this theorem in class? If you look back on your notes, you'll see that

we did end up invoking a lemma, but it was not quite the one above. Rather, it was the slightly more general lemma that said that reversing an append of two lists was the same as appending the reverses of the lists. So let's prove that, and see if we can't use *that* lemma to prove the double-reverse formula. First, undo the last two proofs we have done, and get the prover back to a state before we asked it to prove `rev-app-cons` and `rev-rev`.

```
ACL2 > (defthm rev-app
         (= (rev (app x y)) (app (rev y) (rev x))))

By the simple :definition = we reduce the conjecture to

Goal'
(EQUAL (REV (APP X Y))
       (APP (REV Y) (REV X))).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.

Perhaps we can prove *1 by induction.  Three induction schemes are
suggested by this conjecture.  Subsumption reduces that number to two.
However, one of these is flawed and so we are left with one viable
candidate.

We will induct according to a scheme suggested by (APP X Y).  This
suggestion was produced using the :induction rules APP and REV.  If
we let (:P X Y) denote *1 above then the induction scheme we'll use
is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X) Y))
              (:P X Y))
     (IMPLIES (ENDP X) (:P X Y))).
This induction is justified by the same argument used to admit APP.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (APP (CDR X) Y))
                     (APP (REV Y) (REV (CDR X)))))
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).
```

```
By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/2'
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (APP (CDR X) Y))
                     (APP (REV Y) (REV (CDR X)))))
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

This simplifies, using the :definitions APP and REV, primitive type
reasoning and the :rewrite rules CAR-CONS and CDR-CONS, to

Subgoal *1/2''
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (APP (CDR X) Y))
                     (APP (REV Y) (REV (CDR X)))))
         (EQUAL (APP (REV (APP (CDR X) Y))
                     (LIST (CAR X)))
                (APP (REV Y)
                     (APP (REV (CDR X)) (LIST (CAR X)))))).

The destructor terms (CAR X) and (CDR X) can be eliminated by using
CAR-CDR-ELIM to replace X by (CONS X1 X2), (CAR X) by X1 and (CDR X)
by X2.  This produces the following goal.

Subgoal *1/2'''
(IMPLIES (AND (CONSP (CONS X1 X2))
              (EQUAL (REV (APP X2 Y))
                     (APP (REV Y) (REV X2))))
         (EQUAL (APP (REV (APP X2 Y)) (LIST X1))
                (APP (REV Y)
                     (APP (REV X2) (LIST X1))))).

This simplifies, using primitive type reasoning, to

Subgoal *1/2'4'
(IMPLIES (EQUAL (REV (APP X2 Y))
                (APP (REV Y) (REV X2)))
         (EQUAL (APP (REV (APP X2 Y)) (LIST X1))
                (APP (REV Y)
                     (APP (REV X2) (LIST X1))))).
^^^ Checkpoint Subgoal *1/2'4' ^^^
```

```
We now use the hypothesis by substituting (APP (REV Y) (REV X2)) for
(REV (APP X2 Y)) and throwing away the hypothesis.  This produces

Subgoal *1/2'5'
(EQUAL (APP (APP (REV Y) (REV X2)) (LIST X1))
       (APP (REV Y) (APP (REV X2) (LIST X1)))).

But we reduce the conjecture to T, by the simple :rewrite rule APP-
ASSOC.

Subgoal *1/1
(IMPLIES (ENDP X)
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (NOT (CONSP X))
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

This simplifies, using the :definitions APP and REV, to

Subgoal *1/1''
(IMPLIES (NOT (CONSP X))
         (EQUAL (REV Y) (APP (REV Y) NIL))).
^^^ Checkpoint Subgoal *1/1'' ^^^

We generalize this conjecture, replacing (REV Y) by RV.  This produces

Subgoal *1/1'''
(IMPLIES (NOT (CONSP X))
         (EQUAL RV (APP RV NIL))).
^^^ Checkpoint Subgoal *1/1''' ^^^

We suspect that the term (NOT (CONSP X)) is irrelevant to the truth
of this conjecture and throw it out.  We will thus try to prove

Subgoal *1/1'4'
(EQUAL RV (APP RV NIL)).
```

```
^^^ Checkpoint Subgoal *1/1'4' ^^^

Name the formula above *1.1.

Perhaps we can prove *1.1 by induction.  One induction scheme is suggested
by this conjecture.

We will induct according to a scheme suggested by (APP RV 'NIL).  This
suggestion was produced using the :induction rule APP.  If we let (:P RV)
denote *1.1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP RV)) (:P (CDR RV)))
              (:P RV))
     (IMPLIES (ENDP RV) (:P RV))).
This induction is justified by the same argument used to admit APP.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1.1 ^^^

Subgoal *1.1/2
(IMPLIES (AND (NOT (ENDP RV))
              (EQUAL (CDR RV) (APP (CDR RV) NIL)))
         (EQUAL RV (APP RV NIL))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1.1/2'
(IMPLIES (AND (CONSP RV)
              (EQUAL (CDR RV) (APP (CDR RV) NIL)))
         (EQUAL RV (APP RV NIL))).

But simplification reduces this to T, using the :definition APP, primitive
type reasoning and the :rewrite rule CONS-CAR-CDR.

Subgoal *1.1/1
(IMPLIES (ENDP RV)
         (EQUAL RV (APP RV NIL))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1.1/1'
(IMPLIES (NOT (CONSP RV))
         (EQUAL RV (APP RV NIL))).
```

```
This simplifies, using the :definition APP and primitive type reasoning,
to

Subgoal *1.1/1''
(IMPLIES (NOT (CONSP RV)) (NOT RV)).
^^^ Checkpoint Subgoal *1.1/1'' ^^^

Name the formula above *1.1.1.

No induction schemes are suggested by *1.1.1.  Consequently, the proof
attempt has failed.

Summary
Form:  ( DEFTHM REV-APP ...)
Rules: ((:DEFINITION =)
        (:DEFINITION APP)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:ELIM CAR-CDR-ELIM)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION APP)
        (:INDUCTION REV)
        (:REWRITE APP-ASSOC)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS)
        (:REWRITE CONS-CAR-CDR))
Warnings:  None
Time:  0.03 seconds (prove: 0.01, print: 0.01, proof tree: 0.01, other:
   0.00)

---
The key checkpoint goals, below, may help you to debug this failure.
See :DOC failure and see :DOC set-checkpoint-summary-limit.
---

*** Key checkpoint at the top level: ***

Goal'
(EQUAL (REV (APP X Y))
       (APP (REV Y) (REV X)))
```

```
*** Key checkpoint under a top-level induction: ***

Subgoal *1/1''
(IMPLIES (NOT (CONSP X))
         (EQUAL (REV Y) (APP (REV Y) NIL)))

******** FAILED ********  See :DOC failure  ******** FAILED ********
```

Oops. What happened? Let's look at the transcript, the first place where ACL2 goes astray and tries to be creative. Look at the checkpoints in order. The prover tries a proof by induction on `x`. Reasonable. You note then that when trying to prove subgoal `*1/1`, it tries to prove

```
(= (rev x) (app (rev x) NIL))
```

While this is valid, unfortunately, ACL2 does know it—we never proved that. It tries to prove, but doesn't get very far. In fact, the first thing it tries to do is generalize, to get

```
(= y (app y NIL))
```

which is *not valid* (try `y=5` and see what happens). So by generalizing ACL2 is trying to prove something false. In other words, wrong guess.

Let's help ACL2 along and prove a result that pushes the proof through. The generalization that ACL2 tried is not valid, but it is if we know that `y` is a true list, and indeed, in the case that actually interest us, `(rev x)` is in fact a true list. So let's prove:

```
ACL2 > (defthm true-listp-rev-app
         (implies (true-listp x)
           (= (app x NIL) x)))

By the simple :definition = we reduce the conjecture to

Goal'
(IMPLIES (TRUE-LISTP X)
         (EQUAL (APP X NIL) X)).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.

Perhaps we can prove *1 by induction.  Two induction schemes are suggested
by this conjecture.  These merge into one derived induction scheme.

We will induct according to a scheme suggested by (APP X 'NIL).  This
```

```
suggestion was produced using the :induction rules APP and TRUE-LISTP.
If we let (:P X) denote *1 above then the induction scheme we'll use
is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
This induction is justified by the same argument used to admit APP.
When applied to the goal at hand the above induction scheme produces
three nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/3
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (APP (CDR X) NIL) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (APP X NIL) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/3'
(IMPLIES (AND (CONSP X)
              (EQUAL (APP (CDR X) NIL) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (APP X NIL) X)).

But simplification reduces this to T, using the :definitions APP and
TRUE-LISTP, primitive type reasoning and the :rewrite rule CONS-CAR-
CDR.

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (NOT (TRUE-LISTP (CDR X)))
              (TRUE-LISTP X))
         (EQUAL (APP X NIL) X)).

But we reduce the conjecture to T, by primitive type reasoning.

Subgoal *1/1
(IMPLIES (AND (ENDP X) (TRUE-LISTP X))
         (EQUAL (APP X NIL) X)).

By the simple :definition ENDP we reduce the conjecture to
```

```
Subgoal *1/1'
(IMPLIES (AND (NOT (CONSP X)) (TRUE-LISTP X))
         (EQUAL (APP X NIL) X)).

But simplification reduces this to T, using the :definition TRUE-LISTP,
the :executable-counterparts of APP, CONSP and EQUAL and primitive
type reasoning.

That completes the proof of *1.

Q.E.D.

Summary
Form:  ( DEFTHM TRUE-LISTP-REV-APP ...)
Rules: ((:DEFINITION =)
        (:DEFINITION APP)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION TRUE-LISTP)
        (:EXECUTABLE-COUNTERPART APP)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART EQUAL)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION APP)
        (:INDUCTION TRUE-LISTP)
        (:REWRITE CONS-CAR-CDR))
Warnings:  None
Time:  0.01 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other:
   0.00)
 TRUE-LISTP-REV-APP
```

The proof goes through cleanly, in a single induction on `x`. Let's see if this was enough to prove the theorem we wanted:

```
ACL2 > (defthm rev-app
         (= (rev (app x y)) (app (rev y) (rev x))))

By the simple :definition = we reduce the conjecture to

Goal'
(EQUAL (REV (APP X Y))
```

```
       (APP (REV Y) (REV X))).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.

Perhaps we can prove *1 by induction.  Three induction schemes are
suggested by this conjecture.  Subsumption reduces that number to two.
However, one of these is flawed and so we are left with one viable
candidate.

We will induct according to a scheme suggested by (APP X Y).  This
suggestion was produced using the :induction rules APP and REV.  If
we let (:P X Y) denote *1 above then the induction scheme we'll use
is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X) Y))
              (:P X Y))
     (IMPLIES (ENDP X) (:P X Y))).
This induction is justified by the same argument used to admit APP.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (APP (CDR X) Y))
                     (APP (REV Y) (REV (CDR X)))))
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/2'
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (APP (CDR X) Y))
                     (APP (REV Y) (REV (CDR X)))))
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

This simplifies, using the :definitions APP and REV, primitive type
reasoning and the :rewrite rules CAR-CONS and CDR-CONS, to

Subgoal *1/2''
```

```
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (APP (CDR X) Y))
                     (APP (REV Y) (REV (CDR X)))))
         (EQUAL (APP (REV (APP (CDR X) Y))
                     (LIST (CAR X)))
                (APP (REV Y)
                     (APP (REV (CDR X)) (LIST (CAR X))))))).
```

The destructor terms (CAR X) and (CDR X) can be eliminated by using CAR-CDR-ELIM to replace X by (CONS X1 X2), (CAR X) by X1 and (CDR X) by X2. This produces the following goal.

```
Subgoal *1/2'''
(IMPLIES (AND (CONSP (CONS X1 X2))
              (EQUAL (REV (APP X2 Y))
                     (APP (REV Y) (REV X2))))
         (EQUAL (APP (REV (APP X2 Y)) (LIST X1))
                (APP (REV Y)
                     (APP (REV X2) (LIST X1))))).
```

This simplifies, using primitive type reasoning, to

```
Subgoal *1/2'4'
(IMPLIES (EQUAL (REV (APP X2 Y))
                (APP (REV Y) (REV X2)))
         (EQUAL (APP (REV (APP X2 Y)) (LIST X1))
                (APP (REV Y)
                     (APP (REV X2) (LIST X1))))).
^^^ Checkpoint Subgoal *1/2'4' ^^^
```

We now use the hypothesis by substituting (APP (REV Y) (REV X2)) for (REV (APP X2 Y)) and throwing away the hypothesis. This produces

```
Subgoal *1/2'5'
(EQUAL (APP (APP (REV Y) (REV X2)) (LIST X1))
       (APP (REV Y) (APP (REV X2) (LIST X1)))).
```

But we reduce the conjecture to T, by the simple :rewrite rule APP-ASSOC.

```
Subgoal *1/1
(IMPLIES (ENDP X)
```

```
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (NOT (CONSP X))
         (EQUAL (REV (APP X Y))
                (APP (REV Y) (REV X)))).

But simplification reduces this to T, using the :definitions APP and
REV, primitive type reasoning and the :rewrite rules TRUE-LISTP-REV
and TRUE-LISTP-REV-APP.

That completes the proof of *1.

Q.E.D.

Summary
Form:  ( DEFTHM REV-APP ...)
Rules: ((:DEFINITION =)
        (:DEFINITION APP)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:ELIM CAR-CDR-ELIM)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION APP)
        (:INDUCTION REV)
        (:REWRITE APP-ASSOC)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS)
        (:REWRITE TRUE-LISTP-REV)
        (:REWRITE TRUE-LISTP-REV-APP))
Warnings:  None
Time:  0.02 seconds (prove: 0.01, print: 0.00, proof tree: 0.00, other:
   0.00)
 REV-APP
```

Beautiful—a clean proof with a single induction on `x`. Now, equipped with this lemma, we should be able to prove the double-reverse formula again:

```
ACL2 > (defthm rev-rev
         (implies (true-listp x)
                  (= (rev (rev x)) x)))

By the simple :definition = we reduce the conjecture to

Goal'
(IMPLIES (TRUE-LISTP X)
         (EQUAL (REV (REV X)) X)).
^^^ Checkpoint Goal' ^^^

Name the formula above *1.

Perhaps we can prove *1 by induction.  Two induction schemes are suggested
by this conjecture.  These merge into one derived induction scheme.

We will induct according to a scheme suggested by (REV X).  This suggestion
was produced using the :induction rules REV and TRUE-LISTP.  If we
let (:P X) denote *1 above then the induction scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).
This induction is justified by the same argument used to admit REV.
When applied to the goal at hand the above induction scheme produces
three nontautological subgoals.
^^^ Checkpoint *1 ^^^

Subgoal *1/3
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/3'
(IMPLIES (AND (CONSP X)
              (EQUAL (REV (REV (CDR X))) (CDR X))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But simplification reduces this to T, using the :definitions APP, REV
```

```
and TRUE-LISTP, the :executable-counterparts of CONSP and REV, primitive
type reasoning and the :rewrite rules CAR-CONS, CDR-CONS, CONS-CAR-
CDR and REV-APP.

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (NOT (TRUE-LISTP (CDR X)))
              (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But we reduce the conjecture to T, by primitive type reasoning.

Subgoal *1/1
(IMPLIES (AND (ENDP X) (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (AND (NOT (CONSP X)) (TRUE-LISTP X))
         (EQUAL (REV (REV X)) X)).

But simplification reduces this to T, using the :definition TRUE-LISTP,
the :executable-counterparts of CONSP, EQUAL and REV and primitive
type reasoning.

That completes the proof of *1.

Q.E.D.

Summary
Form:  ( DEFTHM REV-REV ...)
Rules: ((:DEFINITION =)
        (:DEFINITION APP)
        (:DEFINITION ENDP)
        (:DEFINITION NOT)
        (:DEFINITION REV)
        (:DEFINITION TRUE-LISTP)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART EQUAL)
        (:EXECUTABLE-COUNTERPART REV)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
```

```
        (:INDUCTION REV)
        (:INDUCTION TRUE-LISTP)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS)
        (:REWRITE CONS-CAR-CDR)
        (:REWRITE REV-APP))
Warnings:  None
Time:  0.01 seconds (prove: 0.00, print: 0.00, proof tree: 0.00, other:
   0.00)
 REV-REV
```

And indeed, in the proof of the `*1/3` obligation of the induction, we see that the prover used `rev-app`. And it did not need generalization, or to use a nested induction. The whole proof goes through in a clean single induction, once again.