Peter Dillinger

Using ACL2 for Proofs

I will work through an example to demonstrate how to use ACL2 to prove theorems about programs. We will use this implementation of insertion sort:

```
(defun insert (e l)
  (if (endp l)
    (cons e l)
    (if (<= e (car l))
      (cons e l)
      (cons (car l) (insert e (cdr l))))))

(defun isort (l)
  (if (endp l)
    nil
    (insert (car l) (isort (cdr l)))))
```

The function INSERT inserts an element in order into an ordered list:

```
ACL2 >VALUE (insert 2 '(0 1 5 7))
(0 1 2 5 7)
ACL2 >VALUE (insert 2 '())
(2)
ACL2 >VALUE (insert 2 '(3 4))
(2 3 4)
```

And ISORT uses INSERT repeatedly to build a sorted list:

```
ACL2 >VALUE (isort '(9 8 3 7 2))
(2 3 7 8 9)
ACL2 >VALUE (isort '())
NIL
ACL2 >VALUE (isort '(4 5 4 3 3))
(3 3 4 4 5)
```

We can open up ACL2s and create a new Lisp file in Intermediate Mode, add the above definitions, and proceed with attempting the proofs below.

Let us first ask ACL2 to prove that the length of sorting an object is the length of that object:

```
(defthm len--isort
  (equal (len (isort x))
         (len x)))
```

DEFTHM asks ACL2 to attempt to prove a formula and, if successful, remember it for use in future proofs. We give it a name, in this case LEN--ISORT, so that when ACL2 uses it in the future, it can give us the name of what it used.

The proof fails and here is some of the output, with some explanations:

```
<< Starting proof tree logging >>
```

This indicates that ACL2 is sending an outline of the proof attempt to the ACL2s Proof Tree view.

```
Name the formula above *1.
```

This is what ACL2 says when it runs out of things to do without using induction. It will try induction once per proof, so that is what it does next:

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

What are the two functions involed in the conjecture? LEN and ISORT. If you look at the induction schemes suggested by those functions, they are the same.

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

The two subgoals are the base case and the induction step, listed above with an AND.

```
  Subgoal *1/2
  (IMPLIES (NOT (CONSP X))
           (EQUAL (LEN (ISORT X)) (LEN X))).

  But simplification reduces this to T, using the :definitions ISORT
  and LEN and the :executable-counterparts of EQUAL and LEN.
```

ACL2 has completed the entire base case proof in what it considers one step. It tells us it used the definitions of ISORT and LEN and executed the functions EQUAL and LEN to determine this is true.

In fact, (ISORT X) is equal to NIL given the assumption and the definition of ISORT, (LEN NIL) can be executed to get 0, (LEN X) is equal to 0 given the assumption and the definition of LEN, and (EQUAL 0 0) can be executed to get T.

On to the next Subgoal, the inductive step:

```
  Subgoal *1/1
  (IMPLIES (AND (CONSP X)
                (EQUAL (LEN (ISORT (CDR X)))
                       (LEN (CDR X))))
           (EQUAL (LEN (ISORT X)) (LEN X))).

  This simplifies, using the :definitions ISORT and LEN, to

  Subgoal *1/1'
  (IMPLIES (AND (CONSP X)
                (EQUAL (LEN (ISORT (CDR X)))
                       (LEN (CDR X))))
           (EQUAL (LEN (INSERT (CAR X) (ISORT (CDR X))))
                  (+ 1 (LEN (CDR X))))).
```

Those are the two things we would do if we were proving this by hand.

```
  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 *1/1''
  (IMPLIES (AND (CONSP (CONS X1 X2))
                (EQUAL (LEN (ISORT X2)) (LEN X2)))
           (EQUAL (LEN (INSERT X1 (ISORT X2)))
```

```
                (+ 1 (LEN X2))))).
```

  This simplifies, using primitive type reasoning, to

```
  Subgoal *1/1'''
  (IMPLIES (EQUAL (LEN (ISORT X2)) (LEN X2))
           (EQUAL (LEN (INSERT X1 (ISORT X2)))
                  (+ 1 (LEN X2)))).
  ^^^ Checkpoint Subgoal *1/1''' ^^^
```

Now ACL2 used the fact that X is a cons to make the formula a little simpler by using separate variables for what was the CAR of X and the CDR of X.

"Checkpoint" means that it has exhausted its safest proof techniques and might start doing things that be in a wrong direction. In fact, at the beginning of the proof, it said "Checkpoint" because whenever it tries induction, it may not choose the right induction and may go off in a wrong direction.

  We now use the hypothesis by substituting (LEN (ISORT X2)) for (LEN X2) and throwing away the hypothesis. This produces

```
  Subgoal *1/1'4'
  (EQUAL (LEN (INSERT X1 (ISORT X2)))
         (+ 1 (LEN (ISORT X2)))).
  ^^^ Checkpoint Subgoal *1/1'4' ^^^
```

It used one more tool in its arsenal by using the equality assumption and forgetting about it.

  Name the formula above *1.1.

Now it has run out of things to do without using induction...

  No induction schemes were specified for *1.1, and the depth limit for automatic inductions, currently 1, has been reached. Consequently, the proof attempt has failed.

... but it's already done induction once, so it won't again.

```
  Summary
  Form:  ( DEFTHM LEN--ISORT ...)
  Rules: ((:DEFINITION ISORT)
          (:DEFINITION LEN)
          (:ELIM CAR-CDR-ELIM)
          (:EXECUTABLE-COUNTERPART EQUAL)
          (:EXECUTABLE-COUNTERPART LEN)
          (:FAKE-RUNE-FOR-TYPE-SET NIL)
          (:INDUCTION ISORT)
          (:INDUCTION LEN))
  Warnings:  None
  Time:  0.03 seconds (prove: 0.01, print: 0.01, proof tree: 0.01, other: 0.00)
```

Here it tells us the rules it used and how much time it took.

```
  ---
  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 (LEN (ISORT X)) (LEN X))

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

```
  Subgoal *1/1'''
  (IMPLIES (EQUAL (LEN (ISORT X2)) (LEN X2))
           (EQUAL (LEN (INSERT X1 (ISORT X2)))
                  (+ 1 (LEN X2))))

  ACL2 Error in ( DEFTHM LEN--ISORT ...):  See :DOC failure.

  ******** FAILED ********
```

Very often, we do not have to look at all of a failed proof; we can just look at the "key checkpoints". Basically, ACL2 is telling us that it has reduced the original conjecture to the formula

```
  (IMPLIES (EQUAL (LEN (ISORT X2)) (LEN X2))
           (EQUAL (LEN (INSERT X1 (ISORT X2)))
                  (+ 1 (LEN X2))))
```

So if you can help ACL2 prove that, it can prove the original conjecture.

In this case, looking at a formula later in the proof is perhaps even more helpful:

```
  (EQUAL (LEN (INSERT X1 (ISORT X2)))
         (+ 1 (LEN (ISORT X2)))).
```

This is rather simple, and it should be true, right? How about we ask ACL2 to prove this, so that it can try induction on this? It turns out that ACL2 fails to prove this formula exactly. If we look at it carefully, we see that using induction on exactly this formula does not work out well.

What's the technique we've looked at that allows us to "correct" a formula so that induction works out?

Generalization! How can we make the above proposition more general?

Is this a proposition about ISORT? No. In fact, it does not matter that ISORT is being called. We can replace (ISORT X2) with a new variable to make the formula more general:

```
  (EQUAL (LEN (INSERT X1 y))
         (+ 1 (LEN y))).
```

If we put this in a DEFTHM with a name, ACL2 is able to prove this. And with that proven, ACL2 can prove LEN--ISORT, because ACL2 knows it can instantiate y with (ISORT X2).

So after the function definitions, we have

```
  (defthm len--isort--lemma
    (equal (len (insert x y))
           (+ 1 (len y))))

  (defthm len--isort
    (equal (len (isort x))
           (len x)))
```

For appearance, I made everything lowercase and replaced x1 with just x.

We can also ask ACL2 to attempt to prove

```
  (defthm true-listp--isort
    (true-listp (isort x)))
```

It fails, and here's the key checkpoint under induction:

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

  Subgoal *1/2'4'
  (IMPLIES (TRUE-LISTP (ISORT X2))
           (TRUE-LISTP (INSERT X1 (ISORT X2))))
```

And this one calls for the same generalization: the (ISORT X2) could be anything, so we replace that with another variable:

```
  (defthm true-listp--insert
    (implies (true-listp y)
             (true-listp (insert x y))))

  (defthm true-listp--isort
    (true-listp (isort x)))
```