CS2800 Lecture Notes   Lecture 18   27 Oct 2010

Given definitional axioms:

```
(equal (tlp x)
       (if (endp x)
            (equal x nil)
           (tlp (cdr x))))

(equal (add-to-end e l)
       (app l (cons e nil)))

(equal (integer-listp x)
       (if (endp x)
            (equal x nil)
           (and (integerp (car x))
                (integer-listp (cdr x)))))
```

Let us first prove

```
 (tlp (add-to-end e l))
```

using the lemma **tlp-app**

```
  (implies (tlp y)
           (tlp (app x y)))
```

This isn't an equality proof, so we need to make a series of reductions:

```
  (tlp (add-to-end e l))
<= { Def. add-to-end }
   (tlp (app l (cons e nil)))
<= { Instantiation: Lemma tlp-app }
   (tlp (cons e nil))
<= { Def. tlp, IF-false axiom, endp def., consp-cons axiom }
   (tlp (cdr (cons e nil)))
<= { cdr-cons axiom }
   (tlp nil)
= { Evaluation }
   t
```

Basically, using a propositional deduction on `(tlp (cons e nil))` and an instantiation of the lemma,

```
  (implies (tlp (cons e nil))
           (tlp (app l (cons e nil))))
```

allowed me to conclude `(tlp (app l (cons e nil)))`. In fact, the particular propositional deduction is Modus Ponens.

What if I said to prove `(consp 42)` using the lemma `(integerp nil)`?
Believe it or not, I can construct such a proof:

```
  (consp 42)
<= { Propositional deduction }
  nil
<= { Evaluation }
  (integerp nil)
<= { Lemma }
  t
```

What is interesting is that the assumption that `(integerp nil)` is a lemma stipulates that it is a theorem. In this case, the lemma given was not a theorem; in fact, it always evaluates to `nil`! This allowed us--through legal deductions on flawed assumptions--to conclude nil is a theorem. Once we have concluded nil, or "false", we can use a propositional deduction to conclude anything. (false -> p is a tautology.)

Is this really a proof? Yes it is, but not with regard to any theory ACL2 will allow. If an extension of ACL2's base theory is able to prove `(integerp nil)` then that extension is **unsound**, because it allows us to conclude two contradictary propositions: `(not (integerp nil))` and `(integerp nil)`.

ACL2 does not allow unsound extensions of its theory, so this will not happen if the Lemmas you are given are actually theorems in a proper extension of ACL2's base theory.

(It's not important that you completely understand soundness yet.)

===

Prove

```
 (implies (and (integer-listp l)
               (integerp e))
          (integer-listp (add-to-end e l)))
```

using the lemma **integer-listp-app**:

```
 (implies (and (integer-listp x)
               (integer-listp y))
          (integer-listp (app x y)))
```

Once again, we are not proving an equality, but this time we have assumptions we can use, we will put it in the context box:

```
Context:        Assumption1:(integerp-listp l)
                Assumption2: (integerp e)
```

```
(integer-listp (add-to-end e l))
<= {  Defn add-to-end  }
  (integer-listp (app l (cons e nil)))
<= {  Lemma Integer-listp-app }
  (and (integer-listp l)
       (integer-listp (cons e nil)))
<= {  Prop. deduction, assumption 1  }
  (integer-listp (cons e nil))
<= {  Defn integer-listp, IF axiom, not endp cons }
  (and (integerp (car (cons e nil)))
       (integer-listp (cdr (cons e nil))))
<= {  car-cons and cdr-cons  }
  (and (integerp e)
       (integer-listp nil))
<= { Prop. deduction with Assumption 2  }
  (integer-listp nil)
<= {  Evaluation  }
  t
```

A proof displaying Case-analysis:

Given the function definitions:

```
(defun integer-listp (x)
  (if (endp x)
    (equal x nil)
    (and (integerp (car x))
         (integer-listp (cdr x)))))

(defun keep-ints (l)
  (if (endp l)
    nil
    (if (integerp (car l))
      (cons (car l)
            (keep-ints (cdr l)))
      (keep-ints (cdr l)))))
```

Let us prove some cases of

```
 (integer-listp (keep-ints x))
```

Let us start with

```
  (implies (endp x)
           (integer-listp (keep-ints x)))
```

Context
A1: `(endp x)`

```
  (integer-listp (keep-ints x))
<= { Def keep-ints, A1, if-true axiom }
  (integer-listp nil)
<= { Evaluation }
  t
```

===

Let us next prove

```
 (implies (and (consp x)
               (integer-listp (keep-ints (cdr x))))
          (integer-listp (keep-ints x)))
```

Context
Assumption1: `(consp x)`
Assumption2: `(integer-listp (keep-ints (cdr x)))`

```
  (integer-listp (keep-ints x))
<= { Def keep-ints, assumption 1, if-false axiom}
  (integer-listp (if (integerp (car x))
                     (cons (car x)
                           (keep-ints (cdr x)))
                     (keep-ints (cdr x))))
```

Now there does not seem to be much we can do here, because (car x) could be an integer or it could be something else. How do we prove it holds in both cases?

One of our rules of inference can help us out here. A kind of propositional deduction is **case analysis**, which comes from the tautology

```
  ((p -> q) /\ (~p -> q)) -> q
```

That says that if q is true when we assume p and also true when we assume ~p, then q is always true. That means we can reduce proving any proposition q to proving p -> q and ~p -> q for any formula p. This is a case of propositional deduction.

For the problem we are working on, let us first assume `(integerp (car x))` and prove the conjecture, and then we will assume `(not (integerp (car x)))` and prove the conjecture. That will complete the proof of the original conjecture. This splits my proof into two cases(subgoals) since there is a conjunction in `((p -> q) /\ (~p -> q))!`

Case1: (integerp (car x)) is true, so we update our context box for only this sub-proof:

Context(Case1)
Assumption1: `(consp x)`
Assumption2: `(integer-listp (keep-ints (cdr x)))`
CaseAssumption1: `(integerp (car x))`

```
 (integer-listp (if (integerp (car x))
                    (cons (car x)
                          (keep-ints (cdr x)))
                    (keep-ints (cdr x))))
<= { CaseAssumption1, if-true }
  (integer-listp (cons (car x)
                       (keep-ints (cdr x))))
<= { Def integer-listp, consp-cons, endp def, if-false }
  (and (integerp (car (cons (car x)
                            (keep-ints (cdr x)))))
       (integer-listp (cdr (cons (car x)
                                 (keep-ints (cdr x))))))
<= { car-cons, cdr-cons axioms}
  (and (integerp (car x))
       (integer-listp (keep-ints (cdr x))))
<= { CaseAssumption1, Assumption2, propositional deduction }
  t
```

That completes proof of first case(subgoal).

Case 2: (not (integerp (car x)))

```
Context(Case2)
Assumption1: (consp x)
Assumption2: (integer-listp (keep-ints (cdr x)))
CaseAssumption2: (not (integerp (car x)))
```

```
  (integer-listp (if (integerp (car x))
                     (cons (car x)
                           (keep-ints (cdr x)))
                     (keep-ints (cdr x))))
<= { CaseAssumption2, if-false }
  (integer-listp (keep-ints (cdr x)))
<= { Assumption2 }
  t
```

That completes proof of second case(subgoal).
Having proven both cases, the proof is complete.