CS 2800: Homework 5

Due Date: 11pm Monday Nov 1st 2010

Given are the following axioms:

Consp-nil (Axiom)

```
(equal (consp nil)
       nil)
```

Consp-cons (Axiom)

```
(equal (consp (cons a b))
       t)
```

Car-cons (Axiom)

```
(equal (car (cons a b))
       a)
```

Cdr-cons (Axiom)

```
(equal (cdr (cons a b))
       b)
```

If-true (Axiom)

```
(implies x
         (equal (if x y z)
                y))
```

If-false (Axiom)

```
(implies (not x)
         (equal (if x y z)
                z))
```

Def. axiom true-listp (Definitional Axiom)

```
(equal (true-listp x)
       (if (endp x)
         (equal x nil)
         (true-listp (cdr x))))
```

Def. axiom app (Definitional Axiom)

```
(equal (app x y)
       (if (endp x)
           y
         (cons (car x) (app (cdr x) y))))
```

Def. axiom rev (Definitional Axiom)

```
(equal (rev x)
       (if (endp x)
           nil
         (app (rev (cdr x))
              (cons (car x) nil))))
```

Def. axiom len (Definitional Axiom)

```
(equal (len x)
       (if (endp x)
           0
         (+ 1 (len (cdr x)))))
```

Theorem len-app (Given in present theory)

```
(equal (len (app x y))
       (+ (len x) (len y)))
```

Practice Question: Len is invariant under rev

Try to do it for 10 minutes without looking and then look at the solution. Rest of the questions try to follow the same proof format. I will intersperse the proof with comments in italic(thats not part of the solution). Remember successfully proving a formula, means that the formula is a theorem, a true statement, valid, etc albeit under the given logic(ACL2) and present theory(Axioms and previously proven/given theorems).

Prove the following:

```
(and (implies (endp x)
              (equal (len (rev x))
                     (len x)))
     (implies (and (consp x)
                   (equal (len (rev (cdr x)))
                          (len (cdr x))))
              (equal (len (rev x))
                     (len x))))
```

*The most important thing in a proof is to *write* down what exactly are you trying to prove. Initially the formula is called a conjecture, after the proof is done, its called a theorem or lemma.*
Solution:
Now we need to prove `(and F1 F2)`*, to do this all we need to do is independently prove* `F1` *and* `F2`*. Why? Simple,* `(and F1 F2)` *is valid if and only if both* `F1` *and* `F2` *are valid.* We split the conjunction(and) and prove the two subgoals(subformulas) and work at them independently: **Subgoal 1**:
To Prove:

```
(implies (endp x)
         (equal (len (rev x))
                (len x)))
```

Remember that to prove an implication, all you need to do is prove the conclusion assuming the antecedent, i.e. everything in the hypotheses(first argument of `implies`*) goes into the context. Any Proof should always start with the formula you want to prove, that starts the ball rolling:*

```
    (equal (len (rev x))
           (len x))
⇐ {Def. rev}
    (equal (len (if (endp x)
                    nil
                  (app (rev (cdr x))
                       (cons (car x) nil))))
           (len x))
⇐ {Context 1, if-true axiom}
    (equal (len nil)
           (len x))
⇐ {Evaluate[a]  (len nil), Def. len}
    (equal 0
           (if (endp x)
               0
             (+ 1 (len (cdr x)))))
⇐ {Context 1, if-true axiom}
    (equal 0
           0)
⇐ {Equality (built into ACL2 Logic)}
    t
```

Context

1. `(endp x)`

[a] Remember you can evaluate any expression that has no free variables, we will use this implicit rule of inference often and name it **Evaluation**

Note that if true $\rightarrow \phi$ is valid, then P is valid and hence a theorem. i.e. if you deduce ϕ from true then ϕ is a theorem(valid) and the sequence of formulas in the deduction is what is called a ***Proof****. Basically we are working backwards – starting with the original conjecture, step by step we reduce(transform) the conjecture to simpler and simpler formulas until we arrive at true. As I explained in class, $\Leftarrow$ indicates that we conclude the top formula from the bottom one by some axioms and rules of inference.*

Note: In this proof we didnt use the alternate(concise) proof format of reducing the LHS and RHS to a common formula and then using the transitivity of equality to justify the proof

Subgoal 2:

```
(implies (and (consp x)
              (= (len (rev (cdr x)))
                 (len (cdr x))))
         (= (len (rev x))
            (len x)))
```

Like before, we assume the parts of the conjunction(and) in the antecedent are true and prove the conclusion, so my context now has 2 formulas, this time we will use the LHS = RHS proof format we saw in class

```
  (len (rev x))
= { Def. rev, Context 1,if-false axiom }
  (len (app (rev (cdr x))
               (cons (car x) nil)))
= { len-app theorem[a] }
  (+ (len (rev (cdr x)))
        (len (cons (car x) nil)))
= { Context 2[b] }
  (+ (len (cdr x))
        (len (cons (car x) nil)))
= { Def. len , consp-cons, if-false }
 (+ (len (cdr x))
    (+ 1 (len (cdr (cons (car x) nil)))))
= { cdr-cons axiom}
  (+ (len (cdr x))
     (+ 1 (len nil)))
= { Evaluation }
  (+ (len (cdr x))
     (+ 1 0))
= { arithmetic axioms[c]}
  (+ 1 (len (cdr x)))
= { Def. len[d], Context 1, if-false }
  (len x)
```

Context

```
1. (consp x)

2. (= (len (rev (cdr x)))
      (len (cdr x)))
```

[a] Use Instantiation
[b] Use Equals for Equals
[c] Assume common arithmetic knowledge
[d] Equality is symmetric

Note: I didnt go into as much detail as in the previous proof, you can combine multiple steps into one, but dont combine too many. Reason each step, no need to mention the Rules of Inferences(although I do mention them in the footnotes), just mention the axioms/theorems/assumptions(in Context) used. You should understand how a theorem prover works, but dont become the machine, you will get bored.

Both of the subgoals are true(we just proved them), therefore the original conjecture is valid, i.e. it is a theorem, this completes the final proof. Woohoo! □

When we later talk about induction you will realise that the 2 subgoals we proved imply that `(len (rev x)) = (len x)` for all x, basically the first subgoal was the "Base Case" and the second subgoal was the "Induction Step", but lets not jump ahead of us. So for now we have one more theorem in our present theory and you will have to use it in one or more of the questions that follow.

Theorem len-rev-same (Given/Proved in present theory)

```
(equal (len (rev x))
       (len x))
```

Note: In general, whatever you have proved upto a certain point, you can use all those theorems(Instatiation, or Equals for Equals) in your current proof. So for example in Question N, you can use all the theorems proved in Questions 1 to N-1.

Question A

[Grading 25pts, Points distributed in the proof format(reasons), and the context]
Prove in the present theory the following conjecture:

```
(equal (len (rev (app (rev x)
                      (rev y))))
       (len (app (rev y)
                 (rev x))))
```

Question B

[Grading 25pts, Points distributed in the proof format(reasons), and the context]
Prove in the present theory the following conjecture:

```
(equal (len (app (rev x)
                 (app y (rev z))))
       (+ (len (app x z))
          (len (rev y))))
```

Question Z (wont be graded) For practice

- Prove

  ```
  (implies (and (endp x)
                (true-listp y))
           (true-listp (app x y)))
  ```

- Prove:

  ```
  (implies (and (consp x)
                (true-listp (app (cdr x) y)))
           (true-listp (app x y)))
  ```

  Now for the rest of the assignment, you may assume without proof the following theorem/lemma[1]:
  true-listp-app

[1] From now on its part of your present theory

```
(implies (true-listp y)
         (true-listp (app x y)))
```

- Prove:

```
(implies (endp x)
         (true-listp (rev x)))
```

- Prove:

```
(implies (consp x)
         (true-listp (rev x)))
```

- Use the previous two results to prove[2]:
  true-listp-rev

```
(true-listp (rev x))
```

Question C

[Grading 25pts(5 each), one question's solution will be given in class or lab]
Given the following definitions:

```
;; remove-all: All tlp -> tlp
(defun remove-all (a X)
 ``removes all occurrences of a from X''
  (cond ((endp X) nil)
        ((equal a (car X)) (remove-all a (cdr X)))
        (t (cons (car X) (remove-all a (cdr X))))))

;; contained : any x tlp -> Boolean
(defun in (a Y)
  ``Returns t iff a occurs in Y''
  (if (endp Y)
      nil
    (or (equal a (car Y))
        (in a (cdr Y)))))
```

[2] Case analysis

Describe what is wrong with each of the following proof steps(which are missing reasons), if anything is wrong. Be concise. If its a allowed proof step, specify the reason, if not, then concisely say whats wrong.

1.
```
(remove-all 5 X)
=
(cond ((endp X) nil)
      ((equal 5 (car X)) (remove-all a (cdr X)))
      (t (cons (car X) (remove-all a (cdr X)))))
```
   What, if anything, is wrong?

2.
```
(endp (remove-all 5 (list 5 5 5)))
⇐
t
```
   What, if anything, is wrong?

3.
```
(and (endp X)
     (not (in a X)))
⇐
(not (in a X))
```
   What, if anything, is wrong?

4.
```
(not (in a X))
⇐
(endp X)
```
   What, if anything, is wrong?

5.
```
(endp (remove-all a X))
⇐
(in a X)
```
   What, if anything, is wrong?

6.
```
(in a (remove-all a (cdr X)))
⇐
(in a (remove-all a X))
```
   What, if anything, is wrong?

Part D

[Grading 25 pts similar to A and B] Given this definition and the arithmetic axioms [3]:

```
(defun scale (A x)
 `` Adds every element of A by x''
  (if (endp A)
    nil
    (cons (+ (car A) x)
          (scale (cdr A) x))))
```

For example a common arithmetic law is "+" is associative: **Assoc-+** (applies for any x, y, and z even non-numbers in ACL2)

```
(equal (+ x (+ y z))
       (+ (+ x y) z))
```

Prove this conjecture:

```
(and (implies (endp A)
              (equal (scale (scale A x) y)
                     (scale A (+ x y))))
     (implies (and (consp A)
                   (equal (scale (scale (cdr A) x) y)
                          (scale (cdr A) (+ x y))))
              (equal (scale (scale A x) y)
                     (scale A (+ x y)))))
```

Part Y (will not be graded) Practice only

Given the following definitions:

```
;;in : any x tlp -> Boolean
(defun in (a Y)
  ``Returns t iff a occurs in Y''
```

[3] Assume common arithmetic laws you know from high-school

```
  (if (endp Y)
      nil
    (or (equal a (car Y))
        (in a (cdr Y)))))

; subset : tlp x tlp -> Boolean
; Checks if every element of x is contained in y.
(defun subset (X Y)
  (if (endp X)
      t
    (and (in (car X) Y)
         (subset (cdr X) Y))))
```

Prove the following conjectures:

1. ```
   (and (implies (endp X)
                 (subset X X))
        (implies (and (consp X)
                      (subset (cdr X) (cdr X)))
                 (subset X X)))
   ```

   Hint: To prove the second subgoal, you need to prove another formula(such formulas which need to proved in order to prove our original conjecture are called ***lemmas****. For now its okay to specify the lemma and assume it true, but then for your original conjecture to be a theorem, you need to relieve this assumption, by actually proving it, often the proof of lemmas takes more time than the original proof itself.*

2. ```
   (and (implies (endp X)
                 (booleanp (in a X)))
        (implies (and (consp X)
                      (booleanp (in a (cdr X))))
                 (booleanp (in a X))))
   ```