CS 2800: Homework 7

Due Date: 6pm Tuesday Nov 30 2010

Problem 1(20 pts)

:

Suppose you just completed a session with ACL2s where you proved theorems leading to the following rewrite rules.

1. `(f (f x)) = (g x x)`
2. `(f (g (g x y) z)) = x`
3. `(g x y) = (h y)`
4. `(f (h z)) = z`
5. `(g x y) = (h x)`
6. `(g (h x) y) = (f x)`
7. `(f (g x y)) = (g x y)`

Suppose further that these rewrite rules were admitted in the order given above (that is, 1 was admitted first, then 2, then 3, then 4, then 5, then 6).

(a) [2pts] Which rule is applied first when rewriting the expression (f (f (g (h (f z)) x)))

(b) [3pts] One of the rewrite rules above can *never* be applied to *any* expression. Which rule is that? Why can it never be applied?

(c)[7pts] Show all steps in rewriting the following to its final form:

```
(f (g (g a (h b)) (f (f c))))
```

(c)[8pts] Show all steps in rewriting the following(rewrite until no more rules apply):

```
(f (g (f (g (h a) b)) (f (g a b))))
```

Problem2 (40 pts)

Consider the following functions:

```
;; app : tlp x tlp -> tlp
;; Append two lists
(defun app (x y)
  (if (endp x)
    y
    (cons (car x) (app (cdr x) y))))

;; rev : tlp -> tlp
;; Reverse a list
(defun rev (x)
  (if (endp x)
    nil
    (app (rev (cdr x)) (list (car x)))))

;; returns true if X contains a, nil otherwise
(defun in (a X)
 (cond ((endp X) nil)
       ((equal a (car X)) t)
       (t (in a (cdr X)))))

(defun len (l)
 (if (endp l)
     0
   (+ 1 (len (cdr l)))))
```

(a) (5pts) Lets try to prove the following theorem in ACL2:

```
(defthm in-rev
  (equal (in e (rev x))
         (in e x)))
```

ACL2 got stuck at this checkpoint, what lemma can you give it to help it prove the above theorem.

```
Subgoal *1/2'4'

(IN X1 (APP (REV X2) (LIST X1)))
```

(b) (5 pts) Lets try to prove the following in ACL2:

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

ACL2 got stuck at this checkpoint, what lemma can you give it to help it prove the above theorem.

```
Subgoal *1/1'''

(IMPLIES (EQUAL (LEN (REV X2)) (LEN X2))
         (EQUAL (LEN (APP (REV X2) (LIST X1)))
                (+ 1 (LEN X2))))
```

(c) Consider the following definition for compressing a list of elements.

```
(defun compress (s)
  (cond ((endp s) s)
        ((endp (cdr s)) s)
        ((equal (first s) (second s))
         (compress (rest s)))
        (t (cons (first s)
                 (compress (rest s))))))
```

Evaluate the following.

1. [1pt]`(compress (list 1 2 2 1 1 0))`
2. [1pt]`(compress nil)`
3. [1pt]`(compress (list 4 5 4 5))`

You are trying to prove the following theorem with ACL2s

```
(defthm compress-compress
  (equal (compress (compress s))
         (compress s)))
```

But ACL2s fails. The relevant part of what ACL2s reports is:

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

Goal
(EQUAL (COMPRESS (COMPRESS S))
       (COMPRESS S))

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

Subgoal *1/4.3.4'
(IMPLIES (AND (CONSP S4)
              (NOT (EQUAL (CAR (COMPRESS X4)) (CAR X4)))
              (EQUAL (COMPRESS (COMPRESS S4))
                     (COMPRESS S4)))
         (NOT (CONSP (COMPRESS S4))))

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

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

4. [5 pts] What lemma would you prove so that ACL2s can make progress with Subgoal *1/4.3.4'? You don't have to prove anything, but informally explain why you think your conjecture is true.

5. [12 pts] Show how ACL2s will use the theorem you identified above to go further than it did previously. All you need to demonstrate is that your theorem, when used as a rewrite rule by ACL2s, enables ACL2s to simplify Subgoal *1/4.3.4'. Identify the subexpression that your theorem matches and show what it gets rewritten to.

6. [10 pts] You made some progress (I hope). Congratulations! You try to prove `compress-compress` again. Here is what you see.

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

Goal
(EQUAL (COMPRESS (COMPRESS S))
       (COMPRESS S))

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

Subgoal *1/4.3'
(IMPLIES (AND (CONSP S4)
              (NOT (EQUAL (CAR (COMPRESS S4)) (CAR S4))))
         (NOT (EQUAL (COMPRESS (COMPRESS S4))
                     (COMPRESS S4))))

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

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

Same as before. What theorem would you prove so that ACL2s can make progress with Subgoal *1/4.3'? Informally explain why you think your conjecture is true and show how ACL2s will use the theorem you identified above as a rewrite rule to simplify Subgoal *1/4.3'. Identify the subexpression that your theorem matches and show what it gets rewritten to.

Problem 3 (30 points)

Suppose you just completed a session with ACL2 where you proved theorems leading to the following rewrite rules.

1. `(g (h z)) = (g z)`
2. `(g (f x y)) = (f x (f y x))`
3. `(f y (f (h z) x)) = (h z)`
4. `(f x (f y z)) = (f (f x y) z)`

Assume the rewrite rules were admitted in the order given above (that is, 1 was admitted first, then 2, then 3, then 4). Answer all questions based on what ACL2s will do.

Consider the following expression to be rewritten:

```
(g (f y (f (h z) x)))
```

(a) [3pts] Consider subexpression (f y (f (h z) x)) in the expression above. Which is the first rule that will match and be applied?

(b) [3pts] One of the rewrite rules above can *never* be applied to *any* expression. Which rule is that? Why can it never be applied?

(c) [8pts] What is the final result of applying all applicable rewritings to the expression? Show the sequence of rewrite steps that led to your answer.

(d) [8pts] Now rewrite this:

```
(g (f (g (f (h y) (g (h x)))) (f z (h x))))
```

(e) [8pts]One more rewriting exercise:

```
(g (h (f (g (f x (f (h y) y))) (g (h (h z))))))
```

Problem 4(10pts)

Note:This problem may appear similar to the problem you have seen earlier, but in the reverse. But generalization is a stronger concept, it is not as restricted as simple substitution. Intuitively ψ is a generalization of ϕ, if its easy to prove ϕ from ψ, but not vice-versa. Lets make this definition a little more precise:

ψ_σ
$\Rightarrow$ {`Instantiation, plus reasons that relieve hyps`}
ϕ

In the following problems, write True, if ψ is a generalization of ϕ, otherwise write False. If true, give the instantiation, that is give the substitution σ such that $\psi|_\sigma \rightarrow \phi$. And also mention how the assumptions(hypotheses) of ψ are getting relieved by ϕ.

1. ϕ(Original):
```
(implies (consp x)
         (consp (app x x)))
```
   ψ(Generalization):
```
(implies (and (consp x)
              (consp y))
         (consp (app x y)))
```

2. ϕ(Original):
```
(implies (consp x)
         (consp (app x x)))
```
   ψ(Generalization):
```
(implies (or (consp x)
              (consp y))
         (consp (app x y)))
```

3. ϕ(Original):
```
(implies (and (integer-listp x)
              (integer-listp y))
         (true-listp (app y x)))
```
   ψ(Generalization):
```
(implies (and (true-listp x)
              (true-listp y))
         (true-listp (app x y)))
```

4. ϕ(Original):
```
(implies (and (integer-listp x)
              (integer-listp y))
         (integer-listp (app x y)))
```
   ψ(Generalization):
```
(implies (and (integer-listp x)
              (integer-listp y))
         (true-listp (app x y)))
```

5. ϕ(Original):
```
(implies (and (true-listp x)
              (integer-listp y))
```

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

ψ(Generalization):

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

6. ϕ(Original):

```
(= (fact*-acc x 1)
   (fact x))
```

ψ(Generalization):

```
(implies (natp acc)
         (= (fact*-acc x acc)
            (* (fact x) acc)))
```

7. ϕ(Original):

```
(true-listp (app x (cons y nil)))
```

ψ(Generalization):

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