CS 2800: Homework 7 SOlutions

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 $\widehat{(}$f (f (g (h (f z)) x)))
Sol: Rule 5

(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?
Sol: Rule 3. (*1pt*) Because ACL2 tries rules in reverse chronological order, it always tries Rule 5 before Rule 3(*1pt*), and you can plainly see that Rule 5 will match all formulas matched by Rule 3(*1pt*). So Rule 3 will never get a chance to fire.

(c)[7pts] (*Rule 6 is 2 pts, all rest are 1pt*) Show all steps in rewriting the following to its final form:

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

= Rule 5 σ=((x a) (y (h b)))

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

= Rule 1 σ=((x c))

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

= Rule 5 σ=((x c) (y c))

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

= Rule 6 σ=((x a) (y (h c)))

```
(f (f a))
```

= Rule 1 σ=((x a))

```
(g a a)
```

= Rule 5 σ=((x a) (y a))

```
(h a)
```

(c)[8pts] (*Each step 1pt*) Show all steps in rewriting the following(rewrite until no more rules apply):

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

= Rule 6 σ: ((x a) (y b))

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

= Rule 1 σ: ((x a))

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

= Rule 5 σ: ((x a) (y a))

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

= Rule 5 σ: ((x a) (y b))

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

= Rule 4 σ: ((z a))

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

= Rule 6 σ: ((x a) (y a))

```
(f (f a))
```

= Rule 1 σ: ((x a))

```
(g a a))
```

= Rule 5 σ: ((x a) (y a))

```
(h a))
```

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)))

Solution:
(defthm app-in-lemma-conditional
  (implies (in x B)
           (in x (app A B))))
(defthm app-in-lemma-better
  (equal (in x (app A B))
         (or (in x A)
             (in x B))))
```

(*Grading: Full pts if they got any of the above. If they just copied original checkpoint, 2 pts. If they replaced (list x1) with B, but didnt generalize (rev X2) to A, then 4 pts. If the lemma is plainly false, as in if one can think of a counterexample right away, then 0 pts.*)

(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))))
Solution:
(defthm sol1-app-len-list-lemma
 (equal (len (app A (list x)))
        (+ 1 (len A))))
(defthm sol2-app-len-lemma
 (equal (len (app A B)
        (+ (len A) (len B))))
```

(*Grading: Full pts if they got one of the above(in latter, order of A,B does not matter) If they just copied, 1 pt. If they got rid of the equality hypothesis by using it in the conclusion then +3. If in lhs, they have still have (rev X2), then take off just 1 pt. If the lemma is plainly false, as in if one can think of a counterexample right away, then 0 pts.*)

(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. [2pt]`(compress (list 1 2 2 1 1 0))` Sol: (list 1 2 1 0)
2. [2pt]`(compress nil)` Sol: nil
3. [2pt]`(compress (list 4 5 4 5))` Sol: (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. [8 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.

```
Sol1:
(defthm consp-compress-lemma1
  (equal (consp (compress s))
         (consp s)))
Sol2:
(defthm consp-compress-lemma2
  (implies (consp s)
           (consp (compress s))))
```

(*Any of above gets 5 points. If they are close, 3 pts. If they just copy, 1pt. If the lemma they use, is plainly false, then 0pts. *) Explanation(*3pts*): These are true, since a cons has atleast one element, and the definition of compress says that, compressing a list with one element, returns the very

same list. So the only way I can get an empty list by compressing is if I start with an empty list(by def of compress)

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. As you expect, the solution is different, for the Sol1 and Sol2, simply because both lemmas give rise to different rewrite rules. Sol1: consp-compress-lemma1 gives a unconditional rewrite rule, and when given a choice, one should choose such lemmas, they are better than the conditional ones.

This rewrite rule matches `(CONSP (COMPRESS S4))` (*6 pts*) This subexpression is rewritten to `(CONSP S4)`. (*6 pts *)

This simplifies the conclusion to `(not (consp s4))`, and since `(consp s4)` is already there in the hypotheses, this leads to a propositionally simpler formula. I wont go into details, but if you are curious, you can ask me in the review session.

Sol2: consp-compress-lemma2 is a conditional rewrite rule, to use it, ACL2 needs to relieve/satisfy all conditions.

This rewrite rule matches `(CONSP (COMPRESS S4))` (*6 pts*) This subexpression is rewritten to `T`, since the condition under which we can apply the rule `(consp S4)` can be relieved, since it is in the context. (*6 pts *) This simplifies the conclusion to `nil`, which means, that ACL2 knows the assumptions it has are contradictory and has extra information to proceed with this new knowledge.

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.

```
Sol1(*4pts*):
(defthm consp-compress-first-elem-eq
  (implies (consp S)
           (equal (car (compress s))
                  (car s))))

Explanation(*2pts*): This is true by an argument following the definition of compress. H
argument: Case 1(endp (cdr s)) , S has one element, since compress returns S, the car is
obviously same. Case 2 ((car s) = (cadr s)), compress ignores the first element, and
recurs till it finds a second element which is different from the first, in which case
Case 3((car s) != (cadr s))  argument works where, the first element of s and (compress
are same.
```

The following term in the hypothesis matches: `(CAR (COMPRESS S4))`. (*2pts*)
It gets rewritten to `(CAR S4)`. (*2pts*)

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 $\widehat{\text{(f y (f (h z) x))}}$ in the expression above. Which is the first rule that will match and be applied? Sol: rule 4(*3pts*)

(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? Sol: Rule 3.(*1pt*) Because Rule 4 lhs matches all terms (and more) by Rule 3 lhs (*1pt*) and Rule 4 comes first in reverse chronological order(*1pt*).

(c) [each step 2 pts] What is the final result of applying all applicable rewritings to the expression? Show the sequence of rewrite steps that led to your answer.

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

=Rule 4

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

=Rule 2

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

=Rule 4

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

=Rule 4

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

(d) [15pts extra credit] Now rewrite this:

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

Long. Basically, This is what will be applied: Rules 1,2,4,4,2,4,4

(e) [15pts extra credit]One more rewriting exercise:

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

Long. Rules 4,2,4,4,1,1,1,4,4,4

Problem 4(20pts)

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)))
```

Sol: TRUE (*2pts*) The instantiation is obvious, use the following substitution:
σ: ((x x) (y x)) (*2 pts*)

2. ϕ(Original):
```
(implies (consp x)
         (consp (app x x)))
```

ψ(Generalization):
```
(implies (or (consp x)
             (consp y))
         (consp (app x y)))
```

Sol: TRUE (*2pts*) The instantiation is obvious, use the following substitution:
σ: ((x x) (y x)), but also use Prop tautology (A A = A)(*2pts*)

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)))
```

Sol: TRUE (*DONE IN LECTURE NOTES *)

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)))
Sol: FALSE (*DONE IN LECTURE NOTES *)
```

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)))
```

   Sol: FALSE (*2 pts*) Clearly one cant in a small number of obvious steps derive ϕ from ψ. (*2pts*)

6. ϕ(Original):

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

   ψ(Generalization):

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

   Sol: TRUE (*2pts*) Use the following substitution to instantiate ψ: σ:((x x) (acc 1)) (*1pt*) The hyps of ψ are easily relieved, since (natp 1) is true by evaluation, and also (* A 1) = A is simple arithmetic.(*1pt*)

7. ϕ(Original):

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

   ψ(Generalization):

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

   Sol: TRUE(*2pts*) Use the following substitution to instantiate ψ: σ:((x x) (y (cons y nil)))(*1pt*) The hyps of ψ are easily relieved, since (true-listp (cons y nil)) is true by opening the definition of true-listp.(*1pt*)