CS2800 Exam 3 Review Solutions

Below are some extra problems in the form of the last homework.

1. Substitution

a. `(subst 'a a (foo b (rev (app y x))))`
`(subst 'a (* x y) (foo 'b (rev (app (list y) (cdr x)))))`
Solution: `<(a (* x y)) (b 'b) (y (list y)) (x (cdr x))>`

b. `(+ x y (+ x y) (* z w))`
`(+ (/ w z) x (+ (/ w z) x) (* (- (sum-n (1- z)) (expt 2 w)) w))`
Solution: `<(x (/ w z)) (y x) (z (- (sum-n (1- z)) (expt 2 w))) (w w)>` *or*
`<(x (/ w z)) (y x) (z (- (sum-n (1- z)) (expt 2 w)))>` *(since w fixed).*

2. App nil identity

Below is a proof of the following

```
(and (implies (endp x)
              (= (listfix x) (app x nil)))
     (implies (and (consp x)
                   (= (listfix (cdr x)) (app (cdr x) nil)))
              (= (listfix x) (app x nil))))
\End{code}
By propositional logic we split this into two subgoals
\begin{code}
Subgoal 1:
Context:
------------
(1) (endp x)
------------
  (listfix x)
= {Def. listfix, (1)}
  nil
= {Def. app, (1)}
  (app x nil)
Thus the equality is T

Subgoal 2:
Context:
------------
(1) (consp x)
(2) (= (listfix (cdr x)) (app (cdr x) nil))
------------
  (listfix x)
= {def. listfix, (1)
  (cons (car x) (listfix (cdr x)))
= {(2)}
  (cons (car x) (app (cdr x) nil))
= {Def. app, (1)}
  (app x nil)
Thus the equality is T
Q.E.D.
```

3. App rev rev is rev app

Assuming the following are theorems,

```
(implies (and (true-listp x)
              (true-listp y)
              (true-listp z))
         (= (app (app x y) z) (app x (app y z))))

; Sorry these weren't given, these are needed
(implies (true-listp y)
         (= (listfix y) y))

(= (listfix x) (app x nil))
```

a prove of the following is below.

```
(and (implies (and (endp x)
                   (true-listp y)) ; Sorry this wasn't there, this is needed
              (= (rev (app x y)) (app (rev y) (rev x))))
     (implies (and (consp x)
                   (true-listp x)
                   (true-listp y)
                   (= (rev (app (cdr x)) y) (app (rev y) (rev (cdr x)))))
              (= (rev (app x y)) (app (rev y) (rev x)))))
```

By propositional logic we split this into two subgoals

```
Subgoal 1:
Context:
------------
(1) (endp x)
(2) (true-listp y)
------------
  (rev (app x y))
= {Def. app, (1)}
  (rev y)
= {App nil theorem, TL listfix theorem, (2)}
  (rev (app y nil))
= {Def. rev, (1)}
  (rev (app y (rev x)))
Thus the equality is T

Subgoal 2:
Context:
------------
(1) (consp x)
(2) (true-listp x)
(3) (true-listp y)
(4) (= (rev (app (cdr x) y)) (app (rev y) (rev (cdr x))))
------------
  (rev (app x y))
= {Def. app, (1)}
  (rev (cons (car x) (app (cdr x) y)))
= {Def. rev, cons is not an atom}
  (app (rev (app (cdr x) y)) (list (car x)))
= {(4)}
  (app (app (rev y) (rev (cdr x))) (list (car x)))
= {Given theorem, (2), (3), (list d) is a TL}
  (app (rev y) (app (rev (cdr x)) (list (car x))))
= {Def. rev, (1)}
  (app (rev y) (rev x))
Thus the equality is T
Q.E.D.
```

using the definitions

```
(defun true-listp (x)
  (if (endp x)
    (= x nil)
    (true-listp (cdr x))))

(defun rev (x)
  (if (endp x)
    nil
    (app (rev (cdr x)) (list (car x)))))
```

4. Arithmetic identity

(Without nfix, the conjecture below is false. Try x = -10) A proof of the following is below:

```
(and (implies (zp x)
              (= (sum-n x) (/ (* (nfix x) (+ (nfix x) 1)) 2)))
     (implies (and (not (zp x))
                   (= (sum-n (- x 1)) (/ (* (- (nfix x) 1) (nfix x)) 2)))
              (= (sum-n x) (/ (* (nfix x) (+ (nfix x) 1)) 2))))
```

By propositional logic we split this into two subgoals

```
Subgoal 1:
Context:
------------
(1) (zp x)
------------
  (sum-n x)
= {Def. sum-n, (1)}
  0
= {Arithmetic}
  (/ 0 2)
= {Arithmetic}
  (/ (* 0 1) 2)
= {Arithmetic}
  (/ (* 0 (+ 0 1)) 2)
= {Def. nfix, (1)}
  (/ (* x (+ x 1)) 2)
Thus the equality is T

Subgoal 2:
Context:
------------
(1) (not (zp x))
(2) (= (sum-n (- (nfix x) 1)) (/ (* (- (nfix x) 1) (nfix x)) 2))
------------
  (sum-n x)
= {Def. sum-n, (1)}
  (+ x (sum-n (- x 1)))
= {(2)}
  (+ x (/ (* (- (nfix x) 1) (nfix x)) 2))
= {Def. nfix, (1)}
  (+ x (/ (* (- x 1) x) 2))
= {Arithmetic}
  (/ (+ (* 2 x) (* (- x 1) x)) 2)
= {Arithmetic}
  (/ (+ (* x x) x) 2)
= {Arithmetic}
  (/ (* x (+ x 1)) 2)
= {Def. nfix, (1)}
  (/ (* (nfix x) (+ (nfix x) 1)) 2)
Thus the equality is T
Q.E.D.
```

using the definition

```
(defun sum-n (n)
  (if (zp n)
    0
    (+ n (sum-n (- n 1)))))
```