CS2800 Homework 8 Extra Problems

Definitions

Below are definitions that will be used in the problem statements. `fact`, `app`, etc. are omitted here since they are given in the homework.

```
;; choose : nat x nat -> nat
;; Binomial choose function
(defun choose (n k)
  (/ (fact n) (* (fact k) (fact (- n k)))))

;; posp : any -> Boolean
;; T iff given a positive integer
(defun posp (n)
  (not (zp n)))
```

Problems

Prove the following theorems. If you use induction, clearly indicate what functions were used to generate the induction schemes.

1.
```
(implies (and (posp n)
              (posp k)
              (< k n))
         (equal (+ (choose (- n 1) (- k 1)) (choose (- n 1) k))
                (choose n k)))
```

   The above is not a theorem without the hypotheses. Why not?

2.
```
(implies (and (natp n)
              (natp k)
              (<= k n))
         (natp (choose n k)))
```

3.
```
(booleanp (in a X))
```

4.
```
(implies (and (not (= a b))
              (in a (rem-el b X)))
         (in a X))
```

5.
```
(implies (and (not (= a b))
              (in a X))
         (in a (rem-el b X)))
```

6.
```
(implies (not (= a b))
         (= (in a (rem-el b X))
            (in a X)))
```

7.
```
(implies (in a X)
         (not (in a (diff Y X))))
```

8.
```
(implies (consp X)
         (not (=< X (diff Y X))))
```

Solutions

1. We solve this with equational reasoning.

```
Context:
--------------
(1) (posp n)
(2) (posp k)
(3) (< k n)
--------------
  (+ (choose (- n 1) (- k 1)) (choose (- n 1) k))
= {def. choose}
  (+ (/ (fact (- n 1))
        (* (fact (- k 1)) (fact (- (- n 1) (- k 1)))))
     (/ (fact (- n 1))
        (* (fact k) (fact (- (- n 1) k)))))
= {arithmetic. a/b = (a*c)/(b*c), c!=0, (2), (3)}
  (+ (/ (* k (fact (- n 1)))
        (* k (fact (- k 1)) (fact (- n k))))
     (/ (* (- n k) (fact (- n 1)))
        (* (- n k) (fact k) (fact (- (- n 1) k)))))
= {def. fact, (2) commutativity of *, (a-1)-b=(a-b)-1}
  (+ (/ (* k (fact (- n 1)))
        (* (fact k) (fact (- n k))))
     (/ (* (- n k) (fact (- n 1)))
        (* (fact k) (- n k) (fact (- (- n k) 1)))))
= {def. fact, (3)}
  (+ (/ (* k (fact (- n 1)))
        (* (fact k) (fact (- n k))))
     (/ (* (- n k) (fact (- n 1)))
        (* (fact k) (fact (- n k)))))
= {arithmetic. a/c + b/c = (a+b)/c}
  (/ (+ (* k (fact (- n 1)))
        (* (- n k) (fact (- n 1))))
     (* (fact k) (fact (- n k))))
= {arithmetic. distributivity, canceling}
  (/ (* n (fact (- n 1)))
     (* (fact k) (fact (- n k))))
= {def. fact, (1)}
  (/ (fact n)
     (* (fact k) (fact (- n k))))
= {def. choose}
  (choose n k)
Q.E.D.
```

2. We prove this with induction on n, given the result of 3. in the lab questions (call this fact-nat lemma). In this case, we have the induction hypothesis stated twice, giving two substitutions for the free variable k.

```
Induction scheme:
(and (implies (and (natp k)
                   (zp n)
                   (<= k n))
              (natp (choose n k)))
     (implies (and (not (zp n))
                   (natp k)
                   (<= k n)
                   (implies (and (natp (- n 1))
                                 (natp k)
                                 (<= k (- n 1)))
                            (natp (choose (- n 1) k)))
                   (implies (and (natp (- n 1))
                                 (natp (- k 1))
                                 (<= (- k 1) (- n 1)))
                            (natp (choose (- n 1) (- k 1)))))
              (natp (choose n k))))
Base Case:
Context:
---------------
(1) (natp k)
(2) (zp n)
(3) (<= k n)
(4) (zp k)  {(1) (2) (3)}
---------------
= {def. choose, def. fact, (2), (4)}
  (natp 1)
= {1 is a natural}
  T
```

```
Induction Step:
Context:
---------------
(1) (not (zp n))
(2) (natp k)
(3) (<= k n)
(4) (implies (and (natp (- n 1))
                  (natp k)
                  (<= k (- n 1)))
             (natp (choose (- n 1) k)))
(5) (implies (and (natp (- n 1))
                  (natp (- k 1))
                  (<= (- k 1) (- n 1)))
             (natp (choose (- n 1) (- k 1))))
(6) (natp n) {1}
(7) (< 0 n)  {1}
(8) (natp (- n 1)) {6, 7}
(9) (<= (- k 1) (- n 1)) {3, arithmetic}
(A) (implies (<= k (- n 1))
             (natp (choose (- n 1) k))) {2, 4, 8}
(B) (implies (natp (- k 1))
             (natp (choose (- n 1) (- k 1)))) {5, 8, 9}
(C) (= (natp (- k 1)) (posp k))  {2}
(D) (= (<= k (- n 1)) (< k n)) {2}
---------------

[Case Split]
  C1. (= k n)
  (natp (choose n k))
= {def. choose, (C1)}
  (natp (/ (fact n) (* (fact n) (fact (- n n)))))
= {def. fact, fact-nat lemma}
  (natp 1)
= {1 is a natural}
  T

  C2. (not (= k n))
[Case Split]
  C3. (= k 0)
  (natp (choose n k))
= {def. choose, (C3)}
  (natp (/ (fact n) (* (fact 0) (fact (- n 0)))))
= {def. fact, fact-nat lemma}
  (natp 1)
= {1 is a natural}
  T
```

```
  C4. (not (= k 0))
  (natp (choose n k))
= {Theorem 1, (C4), def. posp, (1), (2), (3)}
  (natp (+ (choose (- n 1) (- k 1)) (choose (- n 1) k)))
= {Arithmetic}
  (and (natp (choose (- n 1) (- k 1)))
       (natp (choose (- n 1) k)))
= {(C4), (2), (B)}
  (and T
       (natp (choose (- n 1) k)))
= {(3), (C2), (A)}
  (and T
       T)
= {Propositional Logic}
  T
Q.E.D.
```

3.

```
Induct on (true-listp X) with scheme
(and (implies (endp X)
              (booleanp (in a X)))
     (implies (and (consp X)
                   (booleanp (in a (cdr X))))
              (booleanp (in a X))))
Base case:
Context:
-------------------
(1) (endp X)
-------------------
  (booleanp (in a X))
= {def. in, (1)}
  (booleanp nil)
= {def. booleanp}
  T

Induction Step:
Context:
------------------
(1) (consp X)
(2) (booleanp (in a (cdr X)))
------------------
  (booleanp (in a X))
= {def. in, (1)}
  (booleanp (or (= a (car X))
                (in a (cdr X))))
= {= is boolean, (2), or of booleans is boolean}
  T
Q.E.D.
```

```
We induct on (true-listp X) with scheme
(and (implies (and (endp X)
                   (not (= a b))
                   (in a (rem-el b X)))
              (in a X))
     (implies (and (consp X)
                   (not (= a b))
                   (in a (rem-el b X))
                   (implies (and (not (= a b))
                                 (in a (rem-el b (cdr X))))
                            (in a (cdr X))))
             (in a X)))
```

4.

```
Base Case:
Context:
--------------------
(1) (endp X)
(2) (not (= a b))
(3) (in a (rem-el b X))
(4) (in a nil) {1, 3, def. rem-el}
(5) nil {4, def. in}
-------------------
  (in a X)
= {Propositional Logic}
  T
```

```
Induction Step:
Context:
-------------------
(1) (consp X)
(2) (in a (rem-el b X))
(3) (not (= a b))
(4) (implies (and (not (= a b))
                  (in a (rem-el b (cdr X))))
             (in a (cdr X)))
(5) (implies (in a (rem-el b (cdr X)))
             (in a (cdr X)))  {3, 4}
--------------------
  (in a X)
= {def. in, (1)}
  (or (= a (car X))
      (in a (cdr X)))
[Case Split]
Context:
---------------------
(C1) (= b (car X))
(6) (in a (rem-el (cdr X)) {1, 2, C1, def. rem-el}
(7) (in a (cdr X)) {5, 6}
--------------------
  (or (= a (car X))
      (in a (cdr X)))
= {(3), (7), Propositional Logic}
  T
Context:
-------------------
(C2) (not (= b (car X)))
(6) (in a (cons (car X) (rem-el b (cdr X)))) {1, 2, C2, def. rem-el}
(7) (or (= a (car X))
        (in a (rem-el b (cdr X)))) {6, def. in}
-------------------
```

```
[Case Split]
Context:
-------------------
(C3) (= a (car X))
-------------------
  (or (= a (car X))
      (in a (cdr X)))
= {(C3), Propositional Logic}
  T
Context:
------------------
(C4) (not (= a (car X)))
(8) (in a (rem-el b (cdr X))) {C4, 7}
(9) (in a (cdr X)) {5, 8}
------------------
  (or (= a (car X))
      (in a (cdr X)))
= {(9), Propositional Logic}
  T
Q.E.D.
```

```
We induct on (true-listp X) with scheme
(and (implies (and (endp X)
                   (not (= a b))
                   (in a X))
              (in a (rem-el b X)))
     (implies (and (consp X)
                   (not (= a b))
                   (in a X)
                   (implies (and (not (= a b))
                                 (in a (cdr X)))
                            (in a (rem-el b (cdr X)))))
5.            (in a (rem-el b X))))
Base Case:
Context:
------------------
(1) (endp X)
(2) (not (= a b))
(3) (in a X)
(4) nil {def. in, 1}
-----------------
  (in a (rem-el b X))
= {Propositional Logic}
  T
```

```
Induction step:
Context:
-----------------------
(1) (consp X)
(2) (not (= a b))
(3) (in a X)
(4) (implies (and (not (= a b))
                  (in a (cdr X)))
             (in a (rem-el b (cdr X))))
(5) (implies (in a (cdr X))
             (in a (rem-el b (cdr X)))) {2}
----------------------
[Case Split]
Context:
----------------------
(C1) (= a (car X))
----------------------
  (in a (rem-el b X))
= {def. rem-el, (1), (2)}
  (in a (cons (car X) (rem-el b (cdr X))))
= {def. in, car/cons axiom, (C1)}
  (or (= a a)
      (in a (rem-el b (cdr X))))
= {reflexivity axiom}
  T
Context:
---------------------
(C2) (not (= a (car X))
(6) (in a (cdr X)) {C2, 3}
(7) (in a (rem-el b (cdr X))) {5, 6}
---------------------
  (in a (rem-el b X))
= {def. rem-el, (1)}
  (in a (if (= b (car X))
            (rem-el b (cdr X))
          (cons (car X) (rem-el b (cdr X)))))
= {if lifting}
  (if (= b (car X))
      (in a (rem-el b (cdr X)))
    (in a (cons (car X) (rem-el b (cdr X)))))
= {def. in, car/cons axiom, (2)}
  (if (= b (car X))
      (in a (rem-el b (cdr X)))
    (or nil (in a (rem-el b (cdr X)))))
= {propositional logic}
  (in a (rem-el b (cdr X)))
= {(7)}
  T
Q.E.D.
```

```
(implies (not (= a b))
         (= (in a (rem-el b X))
            (in a X)))
Context:
------------------
(1) (not (= a b))
------------------
```

6.

```
  (= (in a (rem-el b X))
     (in a X))
= {in is boolean, produces two subgoals:}
  SG1: (implies (in a (rem-el b X))
                (in a X))
  SG2: (implies (in a X)
                (in a (rem-el b X)))
Each are discharged by the previously proved theorems with (1)
Q.E.D.
```

```
(implies (in a X)
         (not (in a (diff Y X))))

Induct on (true-listp X) with scheme
(and (implies (and (endp X)
                   (in a X))
              (not (in a (diff Y X))))
     (implies (and (consp X)
                   (in a X)
                   (not (in a (diff Y (cdr X)))))
              (not (in a (diff Y X)))))
Base Case:
Context:
--------------------
(1) (endp X)
(2) (in a X)
(3) nil {1, 2, def. in}
--------------------
  (not (in a (diff Y X)))
= {Propositional Logic}
  T
```

7.

```
Induction Step
Context:
--------------------
(1) (consp X)
(2) (in a X)
(3) (not (in a (diff Y (cdr X))))
--------------------
  (not (in a (diff Y X)))
= {def. diff, (1)}
  (not (in a
           (rem-el (car X) (diff Y (cdr X)))))
[Case Split]
  C1. (= a (car X))
= {(C1), Q5}
  T

  C2. (not (= a (car X)))
= {Above theorem, C2}
  (not (in a (diff Y (cdr X))))
= {(3)}
  T
Q.E.D.
```

```
   Context:
   ---------------
   (1) (consp X)
   ---------------
     (not (=< X (diff Y X)))
   = {def. =<, (1)}
     (not (and (in (car X) (diff Y X))
               (=< (cdr X) (diff Y X))))
8. = (Propositional logic}
     (or (not (in (car X) (diff Y X)))
         (not (=< (cdr X) (diff Y X))))
   = {Above theorem, (in (car X) X) = t}
     (or t
         (not (=< (cdr X) (diff Y X))))
   = {Propositional logic}
     T
   Q.E.D.
```