Interactive Proof Tools
Assignment 4

Hans de Nivelle, Patrick Maier

INFORMATIK

http://www.mpi-sb.mpg.de/~nivelle/teaching/intprooftools2003/main.html

Exercise 4.1 Informally, a multiset is a set that can count how often an element occurs in it. Formally, a *multiset* (over some type `T`) corresponds to a function[1] from `T` to `nat`, and we say that x is an element of a multiset S iff $S(x) > 0$. Below, you find a PVS theory `multiset_defs` which defines multisets and some operations on them analogous to the prelude theory `sets`. Go through the prelude theory `sets_lemmas` and generalize as many of the lemmata as possible to multisets.

```
multiset_defs[T: TYPE]: THEORY
BEGIN
  multiset: TYPE = [T -> nat]

  x, y, z: VAR T
  s, t: VAR multiset

  count(x,s): nat = s(x)
  member(x,s): bool = s(x) > 0

  submultiset?(s,t): bool = FORALL x: s(x) <= t(x)

  empty: multiset = LAMBDA x: 0
  union(s,t): multiset = LAMBDA x: s(x) + t(x)
  intersection(s,t): multiset = LAMBDA x: min(s(x), t(x))
  difference(s,t): multiset = LAMBDA x: max(s(x) - t(x), 0)

  singleton(x): multiset =
    LAMBDA y: IF x = y THEN 1 ELSE 0 ENDIF
  add(x,s): multiset =
    LAMBDA y: IF x = y THEN s(y)+1 ELSE s(y) ENDIF
  remove(x,s): multiset =
    LAMBDA y: IF x = y THEN max(s(y)-1, 0) ELSE s(y) ENDIF
END multiset_defs
```

Hints:

- If a lemma generalizes to multisets then its proof is easy.
- Prove (and use) the following version of extensionality:
  `(FORALL x: count(x,s) = count(x,t)) IMPLIES s = t`

[1] Recall that a subset of some set X corresponds to its *characteristic function* from X to $\{0, 1\}$.

Exercise 4.2 Sorting lists.

1. Prove that the function `mergesort` below is well-defined, i. e., prove the TCCs from its recursive definition. See exercises 1.3 and 3.3 for definitions of the functions `split` and `merge`.

   ```
   mergesort(xs:list[t]): RECURSIVE list[t] =
     LET n = length(xs), (ys,zs) = split(xs, floor(n/2)) IN
       IF n <= 1
         THEN xs
         ELSE merge(mergesort(ys),mergesort(zs))
       ENDIF
   MEASURE length(xs)
   ```

   Hint: Instead of working with the recursive definition of `split` directly, you should probably derive its specification in terms of `length` first, i. e., prove the lemmata

   ```
   split_length1: LEMMA
     n <= length(xs) IMPLIES length(split(xs,n)`1) = n

   split_length2: LEMMA
     n <= length(xs) IMPLIES length(split(xs,n)`2) = length(xs) - n
   ```

2. Prove that `mergesort` applied to any list yields a sorted list, i. e., prove the theorem `mergesort_sorted` below. See exercise 3.3 for a definition of the predicate `sorted?`.

   ```
   mergesort_sorted: THEOREM sorted?(mergesort(xs))
   ```

   Hints:

   - Use `measure-induct`.
   - It might be useful to derive the specification of `split` in terms of `append`, i. e., prove the lemma

     ```
     split_append: LEMMA n <= length(xs) IMPLIES append(split(xs,n)) = xs
     ```