Uebungsblatt 6 Sorting

The goal of this exercise is to specify and verify the *insertion sort algorithm* which intuitively works as follows.

To insert an element into an already sorted list you just go along the list until you find the position where the new element belongs.

To sort a list you start with the empty list (which is trivially sorted) and then successively insert the elements of your input list.

If you have trouble understanding this think of a pile of exam papers you want to sort lexicographically. You put them to your left and to your right you create a sorted pile by moving the exam papers over one at a time and inserting them in the right position.

Your task is to prove all the theorems in the following PVS theory. You can use whatever commands you wish. You do not need to do the proofs in order; you do not need to do all of them though this will incur a reduction in marks.

```
insert : THEORY
  BEGIN

    t : TYPE

    IMPORTING list_adt[t]

    leq : [t,t->boolean]

    leq_trans : AXIOM
        FORALL(x,y,z:t):leq(x,y) AND leq(y,z) IMPLIES leq(x,z)

    leq_refl : AXIOM
        FORALL(x:t): leq(x,x)

    leq_total : AXIOM
        FORALL(x,y:t):leq(x,y) OR leq(y,x)

    length: [list[t] -> nat] =
       reduce_nat(0,lambda(x:t,m:nat):m+1)

    insert(x:t, l:list[t]) : RECURSIVE list[t] =
      IF null?(l)
        THEN cons(x,null)
      ELSIF leq(x,car(l))
        THEN cons(x,l)
        ELSE cons(car(l),insert(x,cdr(l)))
      ENDIF
    MEASURE length(l)

    sort(l:list[t]) : RECURSIVE list[t] =
      IF null?(l)
         THEN null
```

```
      ELSE insert(car(l),sort(cdr(l)))
    ENDIF
  MEASURE length(l)

  occ(x:t,l:list[t]) : RECURSIVE nat =
    CASES l
      OF null : 0 ,
      cons(y,l) : IF x=y THEN occ(x,l)+1 ELSE occ(x,l) ENDIF
    ENDCASES
  MEASURE length(l)

  perm(l1,l2 : list[t]) : bool = FORALL(x:t):occ(x,l1) = occ(x,l2)

  sorted(l:list[t]) : RECURSIVE boolean =
    IF null?(l)
       THEN TRUE
       ELSE sorted(cdr(l)) AND (null?(cdr(l)) OR leq(car(l),car(cdr(l))))
    ENDIF
  MEASURE length(l)

  sort1 : LEMMA % 5/100 marks
    FORALL (x,y:t , l:list[t]):
      sorted(cons(y,l)) AND leq(x,y) IMPLIES
          sorted(cons(x,cons(y,l)))

  insert_lemma : THEOREM % 5/100 marks
      FORALL(x,y:t, l:list[t]):
           (NOT null?(l)) AND  leq(x,y) AND leq(x,car(l)) AND sorted(l) IMPLIES
                  leq(x,car(insert(y,l)))

  insert1 : THEOREM % 40/100 marks
       FORALL(x:t, l:list[t]):
          sorted(l) IMPLIES sorted(insert(x,l))

  insert2 : THEOREM % 30/100 marks
       FORALL(x:t, l:list[t]):
          perm(cons(x,l) , insert(x,l))

  sort_correct : THEOREM % 20/100 marks
       FORALL(l:list[t]):
          sorted(sort(l)) AND perm(l,sort(l))
END insert
```

If that wasn't enough you can give the merge sort algorithm a try, this time with an irreflexive, antisymmetric relation. Notice that the definition of mergesort raises a nontrivial typechecking condition which you should also prove. First, do `M-x tcp` to attempt automatic proofs of all TCCs. Then display the TCCs with `M-x show-tccs` and prove the one marked "unfinished". Here's the merge sort theory. You don't need to do all the theorems:

```
mergesort : THEORY
  BEGIN
```

```
t: TYPE
lt:[t,t->bool]
lt_ax : AXIOM
  FORALL (x,y:t):NOT (lt(x,y) AND lt(y,x))

split(l:list[t],flag:boolean) : RECURSIVE [list[t],list[t]] =
   IF null?(l)
     THEN (null,null)
     ELSE LET (u,v) = split(cdr(l),NOT flag) IN
       IF flag THEN (cons(car(l),u),v)
         ELSE (u,cons(car(l),v))
       ENDIF
  ENDIF
MEASURE length(l)

split1 : LEMMA
  FORALL(l:list[t], flag:boolean):
    LET (u,v) = split(l,flag) IN
      length(u) <= length(l) AND
      length(v) <= length(l) AND
      (NOT null?(l) AND NOT null?(cdr(l)) IMPLIES
        length(u) < length(l) AND
        length(v) < length(l))

merge(u:list[t], v:list[t]) : RECURSIVE list[t] =
  IF null?(u) THEN v
  ELSIF null?(v) THEN u
  ELSIF lt(car(u),car(v))
    THEN cons(car(u),merge(cdr(u),v))
    ELSE cons(car(v),merge(u,cdr(v)))
  ENDIF
MEASURE length(u)+length(v)

sort(l:list[t]) : RECURSIVE list[t] =
  IF null?(l)
    THEN null
  ELSIF null?(cdr(l))
    THEN l
    ELSE LET (u,v) = split(l,TRUE) IN
      merge(sort(u),sort(v))
  ENDIF
MEASURE length(l)

occ(x:t,l:list[t]) : RECURSIVE nat =
  CASES l
    OF null : 0 ,
    cons(y,l) : IF x=y THEN occ(x,l)+1 ELSE occ(x,l) ENDIF
  ENDCASES
MEASURE length(l)

sorted(l:list[t]) : RECURSIVE boolean =
```

```
    IF null?(l)
       THEN TRUE
       ELSE sorted(cdr(l)) AND (null?(cdr(l)) OR NOT lt(car(cdr(l)),car(l)))
    ENDIF
  MEASURE length(l)

  sort1 : LEMMA
    FORALL (x,y:t , l:list[t]):
      sorted(cons(y,l)) AND lt(x,y) IMPLIES
          sorted(cons(x,cons(y,l)))


  perm(l1,l2 : list[t]) : bool = FORALL(x:t):occ(x,l1) = occ(x,l2)

  split2 : LEMMA
     FORALL(l:list[t],flag:boolean):
       LET (u,v) = split(l,flag) IN
         FORALL(x:t):occ(x,l) = occ(x,u) + occ(x,v)

  list_double_induction : LEMMA
     FORALL(p:[list[t],list[t]->boolean]):
          p(null,null) AND
          (FORALL(x:t,u:list[t]):p(u,null) IMPLIES p(cons(x,u),null)) AND
          (FORALL(y:t,v:list[t]):p(null,v) IMPLIES p(null,cons(y,v))) AND
          (FORALL(x,y:t,u,v:list[t]):p(u,v) AND p(cons(x,u),v)
                     AND p(u,cons(y,v)) IMPLIES p(cons(x,u),cons(y,v)))
               IMPLIES
     FORALL(u,v:list[t]):p(u,v)



 test : LEMMA
    FORALL(a,a1,b:boolean):(a1 IMPLIES b) AND a IMPLIES b
  merge1 : LEMMA
     FORALL(u,v:list[t]):
         FORALL(x:t):occ(x,merge(u,v)) = occ(x,u)+occ(x,v)

  merge2 : LEMMA
     FORALL(u,v:list[t]):
         sorted(u) AND sorted(v) IMPLIES sorted(merge(u,v))

  sort_correct : THEOREM
       FORALL(l:list[t]):
          sorted(sort(l)) AND perm(l,sort(l))
END mergesort
```