



VDMTools

VDM-SL Sorting Algorithms ver.1.0



How to contact:

http://fmvdm.org/ http://fmvdm.org/tools/vdmtools inq@fmvdm.org VDM information web site(in Japanese) VDMTools web site(in Japanese) Mail

VDM-SL Sorting Algorithms 1.0

— Revised for VDMTools v9.0.6

© COPYRIGHT 2016 by Kyushu University

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement.

This document is subject to change without notice



Contents

1	Introduction	1
2	Specifications	1



1 Introduction

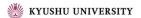
This document is part of the examples released with the *VDM-SL Toolbox* and it is located in the vdmhome/examples directory. The document illustrates a number of specifications of sorting algorithms and it is used in the *Getting Started* section in the *User Manual* to introduce the basic functionality of the Toolbox.

2 Specifications

The first example shows the standard merge sort algorithm well known from text books.

functions

```
MergeSort: seq of real -> seq of real
MergeSort(1) ==
  cases 1:
    -> 1,
    [e]
             -> 1,
    others
             \rightarrow let 11^12 in set {1} be st abs (len 11 - len 12) < 2
                   let 1_1 = MergeSort(11),
                        l_r = MergeSort(12) in
                    Merge(l_l, l_r)
  end;
Merge: seq of int * seq of int -> seq of int
Merge(11,12) ==
  cases mk_{-}(11,12):
    mk_{-}([],1),mk_{-}(1,[]) \rightarrow 1,
                           \rightarrow if hd l1 <= hd l2 then
    others
                                 [hd l1] ^ Merge(tl l1, l2)
                                 [hd 12] ^ Merge(11, tl 12)
  end
```



```
pre forall i in set inds l1 & l1(i) >= 0 and forall i in set inds l2 & l2(i) >= 0
```

The next example shows an implicit specification of a sorting algorithm. The ImplSort function cannot be interpreted as it is described here, but the other VDM-SL tools like the latex generator, the type checker and the syntax checker can process the full VDM-SL language and therefore also this specification.

```
PosReal = real
inv r == r >= 0

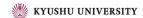
functions

ImplSort(1: seq of PosReal) r: seq of PosReal
post IsPermutation(r,1) and IsOrdered(r);

IsPermutation: seq of real * seq of real -> bool
IsPermutation(11,12) ==
  forall e in set (elems 11 union elems 12) &
     card {i | i in set inds 11 & 11(i) = e} =
     card {i | i in set inds 12 & 12(i) = e};

IsOrdered: seq of real -> bool
IsOrdered(1) ==
  forall i,j in set inds 1 & i > j => 1(i) >= 1(j);
```

In the following example we have changed the implicit function ImplSort to an explicit version ExplSort. This is done by changing the IsPermutation test to a generator function.



```
ExplSort : seq of PosReal -> seq of PosReal
ExplSort (1) ==
  let r in set Permutations(1) be st IsOrdered(r) in r;
Permutations: seq of real -> set of seq of real
Permutations(1) ==
  cases 1:
    [],[-] -> {1},
    others \rightarrow dunion \{\{[l(i)]^j \mid
                         j in set Permutations(RestSeq(1,i))} |
                         i in set inds 1}
  end;
RestSeq: seq of real * nat -> seq of real
RestSeq(1,i) ==
  [l(j) \mid j \text{ in set (inds } l \setminus \{i\})]
pre i in set inds l
post elems RESULT subset elems 1 and
     len RESULT = len l - 1;
```

The last example is also a standard algorithm based on the principle of sorting by insertion.

```
DoSort: seq of real -> seq of real
DoSort(1) ==
   if 1 = [] then
      []
   else
      let sorted = DoSort (tl 1) in
        InsertSorted (hd 1, sorted);

InsertSorted: PosReal * seq of PosReal -> seq of PosReal
InsertSorted(i,1) ==
   cases true :
```

```
(1 = [])    -> [i],
  (i <= hd 1) -> [i] ^ 1,
  others    -> [hd 1] ^ InsertSorted(i,tl 1)
end
```