VDMTools

VDM-SL Sorting Algorithms

SCSK CORPORATION

How to contact SCSK:

http://www.vdmtools.jp/	VDM information web site(in Japanese)
http://www.vdmtools.jp/en/	VDM information web site(in English)
http://www.scsk.jp/	SCSK Corporation web site(in Japanese)
http://www.scsk.jp/index_en.html	SCSK Corporation web site(in English)
vdm.sp@scsk.jp	Mail

VDM-SL Sorting Algorithms 2.0
— Revised for VDMTools v9.0.2

Contents

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(l) ==
    cases l:
      []       -> l,
      [e]      -> l,
      others   -> let l1^l2 in set {l} be st abs (len l1 - len l2) < 2
                  in
                    let l_l = MergeSort(l1),
                        l_r = MergeSort(l2) in
                     Merge(l_l, l_r)
    end;


  Merge: seq of int * seq of int -> seq of int
  Merge(l1,l2) ==
    cases mk_(l1,l2):
      mk_([],l),mk_(l,[]) -> l,
      others              -> if hd l1 <= hd l2 then
                                [hd l1] ^ Merge(tl l1, l2)
                             else
                                [hd l2] ^ Merge(l1, tl l2)
    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.

```
types

  PosReal = real
  inv r == r >= 0


functions

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

  IsPermutation: seq of real * seq of real -> bool
  IsPermutation(l1,l2) ==
    forall e in set (elems l1 union elems l2) &
      card {i | i in set inds l1 & l1(i) = e} =
      card {i | i in set inds l2 & l2(i) = e};

  IsOrdered: seq of real -> bool
  IsOrdered(l) ==
    forall i,j in set inds l & i > j => l(i) >= l(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 (l) ==
  let r in set Permutations(l) be st IsOrdered(r) in r;

Permutations: seq of real -> set of seq of real
Permutations(l) ==
  cases l:
    [],[-] -> {l},
    others -> dunion {{[l(i)]^j |
                        j in set Permutations(RestSeq(l,i))} |
                        i in set inds l}
  end;

RestSeq: seq of real * nat -> seq of real
RestSeq(l,i) ==
  [l(j) | j in set (inds l \ {i})]
pre i in set inds l
post elems RESULT subset elems l 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(l) ==
  if l = [] then
    []
  else
    let sorted = DoSort (tl l) in
      InsertSorted (hd l, sorted);

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

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