SCSK

VDMTools

VDM++ 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++ Sorting Algorithms 2.0
— Revised for VDMTools v9.0.2

Contents

1 Introduction

This document contains a sorting example. The class diagram can be seen in Figure 1. The structure of the example is known as the *strategy* pattern. This pattern defines a family of algorithms, encapsulates each one and make them interchangeable. The *strategy* pattern lets the algorithm vary independently from clients that use it. The `SortMachine` class is the client that uses the different sorting algorithms. The `Sorter` class is an abstract class that defines a common interface to all supported algorithms.

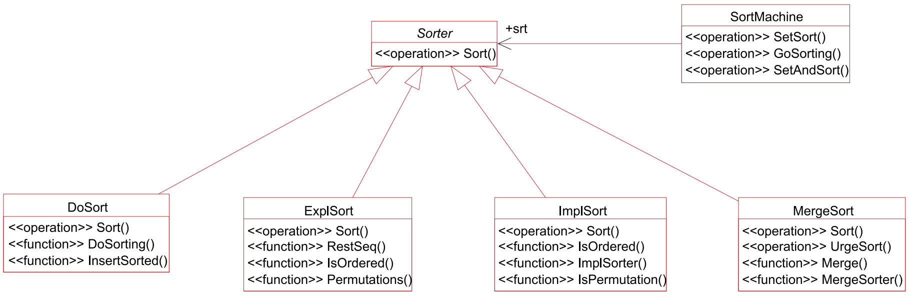

Figure 1: Class diagram for the sort example

80028888attern ”Strategy”. This is the class that uses the 0ifferent sorting algorithms.

2 Sort Machine

```
class SortMachine

instance variables
  srt: Sorter := new MergeSort();
```

The instance variable “srt” is an object reference to the sorting algorithm currently in use. The initial sorting algorithm is MergeSort.

Setting/changing which sorting algorithm to use.

```
operations

  public SetSort: Sorter ==> ()
  SetSort(s) ==
    srt := s;
```

Sorting with the sorting algorithm currently in use.

```
  public GoSorting: seq of int ==> seq of int
  GoSorting(arr) ==
    return srt.Sort(arr);
```

Set/change first the sorting algorithm and sort afterwards.

```
  public SetAndSort: Sorter * seq of int ==> seq of int
  SetAndSort(s, arr) ==
  ( srt := s;
    return srt.Sort(arr)
  )

end SortMachine
```

3 Sorter class

```
class Sorter

operations

  public
  Sort: seq of int ==> seq of int
  Sort(arg) ==
    is subclass responsibility

end Sorter
```

4 MergeSort

```
class MergeSort is subclass of Sorter

operations

  public Sort: seq of int ==> seq of int
  Sort(l) ==
    return MergeSorter(l)

functions

  MergeSorter: seq of real -> seq of real
  MergeSorter(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 = MergeSorter(l1),
                       l_r = MergeSorter(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



end MergeSort
```

5 DoSort

```
class DoSort is subclass of Sorter

operations

  public Sort: seq of int ==> seq of int
  Sort(l) ==
    return DoSorting(l)

functions

  DoSorting: seq of int -> seq of int
  DoSorting(l) ==
    if l = [] then
      []
    else
      let sorted = DoSorting (tl l) in
        InsertSorted (hd l, sorted);

  InsertSorted: int * seq of int -> seq of int
  InsertSorted(i,l) ==
    cases true :
      (l = [])    -> [i],
      (i <= hd l) -> [i] ^ l,
      others      -> [hd l] ^ InsertSorted(i,tl l)
    end



end DoSort
```

An overview of the test coverage information for the *DoSort* class is listed in the table below. The test coverage information is generated using the argument file *sort.arg*.

6 ImplSort

The class *ImplSort* is an example of an sorting algorithm defined by implicit functions.

```
class ImplSort is subclass of Sorter

operations

  public Sort: seq of int ==> seq of int
  Sort(l) ==
    return ImplSorter(l);

functions

  public ImplSorter(l: seq of int) r: seq of int
  post IsPermutation(r,l) and IsOrdered(r);

  IsPermutation: seq of int * seq of int -> 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 int -> bool
  IsOrdered(l) ==
    forall i,j in set inds l & i > j => l(i) >= l(j)

end ImplSort
```

7 ExplSort

The class *ExplSort* is a refinement of the algorithm described in *ImplSort*.

```
class ExplSort is subclass of Sorter

operations

  public Sort: seq of int ==> seq of int
  Sort(l) ==
    let r in set Permutations(l) be st IsOrdered(r) in
    return r

functions

  Permutations: seq of int -> set of seq of int
  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 int * nat -> seq of int
  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;

  IsOrdered: seq of int -> bool
  IsOrdered(l) ==
    forall i,j in set inds l & i > j => l(i) >= l(j)

end ExplSort
```