$SPAD/src/algebra sets.spad

Michael Monagan, Richard Jenks

May 1991

Abstract

Contents

1 domain SET Set

⟨*domain SET Set*⟩≡

```
)abbrev domain SET Set
++ Author: Michael Monagan; revised by Richard Jenks
++ Date Created: August 87 through August 88
++ Date Last Updated: May 1991
++ Basic Operations:
++ Related Constructors:
++ Also See:
++ AMS Classifications:
++ Keywords:
++ References:
++ Description:
++ A set over a domain D models the usual mathematical notion of a finite set
++ of elements from D.
++ Sets are unordered collections of distinct elements
++ (that is, order and duplication does not matter).
++ The notation \spad{set [a,b,c]} can be used to create
++ a set and the usual operations such as union and intersection are available
++ to form new sets.
++ In our implementation, \Language{} maintains the entries in
++ sorted order.  Specifically, the parts function returns the entries
++ as a list in ascending order and
++ the extract operation returns the maximum entry.
++ Given two sets s and t where \spad{#s = m} and \spad{#t = n},
++ the complexity of
++   \spad{s = t} is \spad{O(min(n,m))}
++   \spad{s < t} is \spad{O(max(n,m))}
++   \spad{union(s,t)}, \spad{intersect(s,t)}, \spad{minus(s,t)}, \spad{symmetricDifferen
++   \spad{member(x,t)} is \spad{O(n log n)}
++   \spad{insert(x,t)} and \spad{remove(x,t)} is \spad{O(n)}
Set(S:SetCategory): FiniteSetAggregate S == add
   Rep := FlexibleArray(S)
   # s       == _#$Rep s
   brace()   == empty()
   set()     == empty()
   empty()   == empty()$Rep
   copy s    == copy(s)$Rep
   parts s   == parts(s)$Rep
   inspect s == (empty? s => error "Empty set"; s(maxIndex s))

   extract_! s ==
     x := inspect s
     delete_!(s, maxIndex s)
     x
```

```
find(f, s) == find(f, s)$Rep

map(f, s) == map_!(f,copy s)

reduce(f, s) == reduce(f, s)$Rep

reduce(f, s, x) == reduce(f, s, x)$Rep

reduce(f, s, x, y) == reduce(f, s, x, y)$Rep

if S has ConvertibleTo InputForm then
  convert(x:%):InputForm ==
     convert [convert("set"::Symbol)@InputForm,
                       convert(parts x)@InputForm]

if S has OrderedSet then
  s = t == s =$Rep t
  max s == inspect s
  min s == (empty? s => error "Empty set"; s(minIndex s))

  map_!(f,s) ==
    map_!(f,s)$Rep
    sort_!(s)$Rep
    removeDuplicates_! s

  construct l ==
    zero?(n := #l) => empty()
    a := new(n, first l)
    for i in minIndex(a).. for x in l repeat a.i := x
    removeDuplicates_! sort_! a

  insert_!(x, s) ==
    n := inc maxIndex s
    k := minIndex s
    while k < n and x > s.k repeat k := inc k
    k < n and s.k = x => s
    insert_!(x, s, k)

  member?(x, s) == -- binary search
    empty? s => false
    t := maxIndex s
    b := minIndex s
    while b < t repeat
      m := (b+t) quo 2
      if x > s.m then b := m+1 else t := m
```

```
  x = s.t

remove_!(x:S, s:%) ==
  n := inc maxIndex s
  k := minIndex s
  while k < n and x > s.k repeat k := inc k
  k < n and x = s.k => delete_!(s, k)
  s

-- the set operations are implemented as variations of merging
intersect(s, t) ==
  m := maxIndex s
  n := maxIndex t
  i := minIndex s
  j := minIndex t
  r := empty()
  while i <= m and j <= n repeat
    s.i = t.j => (concat_!(r, s.i); i := i+1; j := j+1)
    if s.i < t.j then i := i+1 else j := j+1
  r

difference(s:%, t:%) ==
  m := maxIndex s
  n := maxIndex t
  i := minIndex s
  j := minIndex t
  r := empty()
  while i <= m and j <= n repeat
    s.i = t.j => (i := i+1; j := j+1)
    s.i < t.j => (concat_!(r, s.i); i := i+1)
    j := j+1
  while i <= m repeat (concat_!(r, s.i); i := i+1)
  r

symmetricDifference(s, t) ==
  m := maxIndex s
  n := maxIndex t
  i := minIndex s
  j := minIndex t
  r := empty()
  while i <= m and j <= n repeat
    s.i < t.j => (concat_!(r, s.i); i := i+1)
    s.i > t.j => (concat_!(r, t.j); j := j+1)
    i := i+1; j := j+1
  while i <= m repeat (concat_!(r, s.i); i := i+1)
  while j <= n repeat (concat_!(r, t.j); j := j+1)
```

```
    r

  subset?(s, t) ==
    m := maxIndex s
    n := maxIndex t
    m > n => false
    i := minIndex s
    j := minIndex t
    while i <= m and j <= n repeat
      s.i = t.j => (i := i+1; j := j+1)
      s.i > t.j => j := j+1
      return false
    i > m

  union(s:%, t:%) ==
    m := maxIndex s
    n := maxIndex t
    i := minIndex s
    j := minIndex t
    r := empty()
    while i <= m and j <= n repeat
      s.i = t.j => (concat_!(r, s.i); i := i+1; j := j+1)
      s.i < t.j => (concat_!(r, s.i); i := i+1)
      (concat_!(r, t.j); j := j+1)
    while i <= m repeat (concat_!(r, s.i); i := i+1)
    while j <= n repeat (concat_!(r, t.j); j := j+1)
    r

else
  map_!(f,s) ==
    map_!(f,s)$Rep
    removeDuplicates_! s

  insert_!(x, s) ==
    for k in minIndex s .. maxIndex s repeat
      s.k = x => return s
    insert_!(x, s, inc maxIndex s)

  remove_!(x:S, s:%) ==
    n := inc maxIndex s
    k := minIndex s
    while k < n repeat
      x = s.k => return delete_!(s, k)
      k := inc k
    s
```

2 License

⟨*license*⟩≡

```
--Copyright (c) 1991-2002, The Numerical ALgorithms Group Ltd.
--All rights reserved.
--
--Redistribution and use in source and binary forms, with or without
--modification, are permitted provided that the following conditions are
--met:
--
--    - Redistributions of source code must retain the above copyright
--      notice, this list of conditions and the following disclaimer.
--
--    - Redistributions in binary form must reproduce the above copyright
--      notice, this list of conditions and the following disclaimer in
--      the documentation and/or other materials provided with the
--      distribution.
--
--    - Neither the name of The Numerical ALgorithms Group Ltd. nor the
--      names of its contributors may be used to endorse or promote products
--      derived from this software without specific prior written permission.
--
--THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
--IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
--TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
--PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
--OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
--EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
--PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
--PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
--LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
--NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
--SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

⟨***⟩≡

⟨*license*⟩

⟨*domain SET Set*⟩

References

[1] nothing