Package: hset (via r-universe)

August 30, 2024

Type Package

Title Sets of Numbers Implemented with Hash Tables

Version 0.1.1

Author Giacomo Ceoldo [aut, cre, cph]

Maintainer Giacomo Ceoldo <giacomo.ceoldo@usi.ch>

Description Implementation of S4 class of sets and multisets of numbers. The implementation is based on the hash table from the package 'hash'. Quick operations are allowed when the set is a dynamic object. The implementation is discussed in detail in Ceoldo and Wit (2023) <arXiv:2304.09809>.

Imports hash, methods

Encoding UTF-8

License MIT + file LICENSE

NeedsCompilation no

Repository CRAN

Date/Publication 2023-04-28 22:20:18 UTC

Contents

hset 2
in 4
operations 5
subset 6

Index **8**

hset hset *data structure for the R language*

Description

Functions to construct and access objects of class "hset", that implements sets and multisets.

Usage

```
hset(members = NULL, multiplicities = NULL, generalized = FALSE)

is.hset(x)
as.hset(x)

is.generalized(hset)
as.generalized(hset, suppress.warning = FALSE)
as.not.generalized(hset, suppress.warning = FALSE)

clone.of.hset(current.hset, generalized = NA_integer_)
refer.to.hset(current.hset, generalized = NA_integer_)

members(hset)
multiplicities(hset)
size.support(hset)
cardinality(hset)
```

Arguments

members	collection of elements of the set.
multiplicities	collection of multiplicities of the elements in members, argument used only when generalized is TRUE.
generalized	logical value indicating whether the returned object is a set, or a multiset.
x	object to be coerced or tested.
hset	object of class "hset".
suppress.warning	avoid warning when hset is already generalized, or is already not generalized.
current.hset	object of class "hset" to be copied, or referred to

Details

hset is the constructor of a set or a multiset, implemented using the hash table from package "hash".

For multisets, the arguments members and multiplicities have to be compatible. If the latter is NULL all members have multiplicity 1 by default, otherwise the two arguments must have the same length. If multiplicities is not NULL, it must be a vector of class "numeric" with the same length

of members. The valid classes for the argument members are "NULL", "hset", or "vector", in the third case, members can be of sub-class "atomic", or of sub-class "list" containing objects of the classes indicated so far. The recursive definition of "list" objects allows the definition of elements of a set or a multiset that are themselves sets.

The function is.hset is used to check whether the object x is of class "hset", whereas as.hset is used to coerce x to an "hset" object.

The function is.generalized is used to check whether an object of class "hset" is a set or a multiset, as.generalized and as.not.generalized convert a set to a multiset and viceversa.

The functions clone.of.hset and refer.to.hset are used to copy an object of class "hset", and refer to it, respectively.

The functions members and multiplicities return a vector of elements, with their corresponding multiplicities, respectively. The functions size.support and cardinality return the number of elements and the sum of the multiplicities, respectively.

Value

Functions hset, as.hset, as.generalized, as.not.generalized, clone.of.hset, and refer.to.hset return an object of class "hset"; is.hset and is.generalized return a one dimensional logical value; members and multiplicities return a vector of class "character"; size.support and cardinality return a number.

Examples

```
## create an empty set and an empty multiset
hset() # equivalent to hset(NULL), as.hset(list()), or hset(list())
hset(generalized = TRUE)

## create set {1,3,4} and multiset {1[2], 3[1], 4[.5]}
hset(c(1,3,4)) # equivalent to hset(c(1,1,3,4)) or hset(c(1,4,3))
hset(c(1,3,4), c(2,1,.5)) # equivalent to hset(c(1,1,3,4), c(1,1,1,.5))

## recursive definition of sets
hset(hset()) # equivalent to hset(list(list()))
hset(list(1, list(1,list()) )) # {{{},1},1}

## check and coerce hset objects
is.hset(hset())
as.hset(list())
# note that hset(hset()) and as.hset(hset()) are not equivalent

## canonical map from sets to multisets, and vice versa
hs <- hset(); is.generalized(hs)
as.generalized(hs); is.generalized(hs)
as.not.generalized(hs); is.generalized(hs)
# note reference semantic of hs

## value and reference semantics
hs <- hset(c(1,3,4))
hsc <- clone.of.hset(hs)
hsc <- union(hsc, hset(c(4,5)))
```

```
hsc; hs # value semantic
hsr <- refer.to.hset(hs) # equivalent to: hsr <- hs, or hsr = hs
hsc <- union(hsc, hset(c(4,5)))
hsr; hs # they refer to the same updated object in memory

## information extraction about hset objects
hm <- hset(c(1,4,3), c(2,.5,1), generalized = TRUE)
members(hm)
multiplicities(hm)
size.support(hm)
cardinality(hm)
```

`in` *Parametrized inclusion predicate between a member and an* `"hset"` *object*

Description

`TRUE` is returned when the argument `member` is included in the set (or multiset) `hset`, otherwise `FALSE` is returned.

Usage

```
inclusion.member(member, hset, multiplicity = 1L, type.relation = `<=`)
member %in% hset # default for multiplicity and type.relation
```

Arguments

`member`	vector of length 1 that, when converted to a `"character"`, labels the element that can be included in the set.
`hset`	object of class `"hset"` that could contain `member`.
`multiplicity`	how many times `member` is inclued in `hset`.
`type.relation`	one of the six binary relational operator, e.g., `<`.

Details

Arguments `multiplicity` and `type.relation` used only when argument `hset` is generalized.

Value

`TRUE` is returned if `member` and `hset` are in relation, otherwise `FALSE` is returned.

Examples

```
inclusion.member(2, hset(c(2,3)))
2 %in% hset(c(2,3))
inclusion.member(1, hset(c(2,3))) # 1 %in% hset(c(2,3))

inclusion.member(2, hset(c(2,3), generalized = TRUE))
inclusion.member(2, hset(c(2,3), c(.5, 1))) # default multiplicity is 1
inclusion.member(2, hset(c(2,3), c(.5, 1)), .5)
inclusion.member(2, hset(c(2,3), c(.5, 1)), .5, `<`) # strict inclusion
```

`operations` *Set and multiset operations*

Description

Operations between "hset" objects.

Usage

```
intersection(hset1, ..., semantic = "refer")
hset1 %&%  hset2 # refer semantic, operator equivalent to %&&%  , %and%
hset1 %&~% hset2 # value semantic, operator equivalent to %&&~% , %and~%

union(hset1, ..., semantic = "refer")
hset1 %|%  hset2 # refer semantic, operator equivalent to %||%  , %or%
hset1 %|~% hset2 # value semantic, operator equivalent to %||~% , %or~%

difference(hset1, ..., semantic = "refer")
hset1 %-%  hset2 # refer semantic, operator equivalent to %!imp%
hset1 %-~% hset2 # value semantic, operator equivalent to %!imp~%

symmdiff(hset1, ..., semantic = "refer")
hset1 %--%  hset2 # refer semantic, operator equivalent to %xor%
hset1 %--~% hset2 # value semantic, operator equivalent to %xor~%

setsum(hset1, ..., semantic = "refer")
hset1 %+%  hset2 # refer semantic, operator equivalent to %sum%
hset1 %+~% hset2 # value semantic, operator equivalent to %sum~%
```

Arguments

hset1	first operand, which is an object of class "hset".
hset2	second operand, which is an object of class "hset", used with infix operators.
...	other operands, which are all objects of class "hset".
semantic	either "refer" or "value".

Details

If ... is empty, hset1 is returned.

The returned object is a multiset if at least one operand is as such, otherwise the returned object is a set.

If reference semantic is used, the returned value and the first operand point to the same object in memory, as the first operand has been modified in place to produce the returned value. So after an operation with reference semantic, the original operand can not be accessed directly.

Value

Object of class "hset" that is the result of the operation.

Examples

```
## operations between sets
X1 <- hset(c(2,3,4)); X2 <- hset(c(2,3,5))
intersection(X1, X2, semantic = "value") # X1 %&~% X2
union(X1, X2, semantic = "value") # X1 %|~% X2
difference(X1, X2, semantic = "value") # X1 %-~% X2
symmdiff(X1, X2, semantic = "value") # X1 %--~% X2
setsum(X1, X2, semantic = "value") # X1 %+~% X2

## semantic of operations
X1 # same as before, as value semantic is used so far
union(X1, X2) # union with reference semantic
X1 # result of previous computation

## operations between multisets
Y1 <- hset(c(2,3,4), c(2,1,.5)); Y2 <- hset(c(2,3,5), c(1,1,.5))
intersection(Y1, Y2, semantic = "value") # Y1 %&~% Y2
union(Y1, Y2, semantic = "value") # Y1 %|~% Y2
difference(Y1, Y2, semantic = "value") # Y1 %-~% Y2
symmdiff(Y1, Y2, semantic = "value") # Y1 %--~% Y2
setsum(Y1, Y2, semantic = "value") # Y1 %+~% Y2

## mixed operation
Y3 <- setsum(Y1, X2, semantic = "value")
Y3
```

---

subset *Binary subset and equality relations*

Description

Parametrized subset and equality relations between two "hset" objects.

Usage

```
hset1.included.to.hset2(hset1, hset2, strictly = FALSE, exactly = FALSE)
hset1 <= hset2 # strictly = FALSE, exactly = FALSE
hset1 >= hset2
hset1 <  hset2 # strictly = TRUE,  exactly = FALSE
hset1 >  hset2
hset1 %=<=% hset2 # strictly = FALSE, exactly = TRUE
hset1 %=>=% hset2
hset1 %=<%  hset2 # strictly = TRUE,  exactly = TRUE
hset1 %=>%  hset2

hsets.are.equal(hset1, hset2)
hset1 == hset2
hset1 != hset2
```

Arguments

`hset1`, `hset2`	objects of class "`hset`".
`strictly`, `exactly`	logical flags indicating which type of inclusion relation is evaluated.

Details

Argument `exactly` not used if the first two arguments are not generalized.

Value

`TRUE` is returned if `hset1` and `hset2` are in the indicated relation, otherwise `FALSE` is returned.

Examples

```
# subset and equality relation between sets
X1 <- hset(c(2,3)); X2 <- hset(c(2,3))
hset1.included.to.hset2(X1, X2) # X1 <= X2
hset1.included.to.hset2(X1, X2, strictly = TRUE) # X1 < X2
hsets.are.equal(X1, X2)
X1 == X2; X1 != X2

# subset relations between multisets
Y1 <- hset(c(2,3), c(1,2))
Y2 <- hset(c(2,3,4), c(1,2,1))
Y3 <- hset(c(2,3,4), c(2,2,1))

hset1.included.to.hset2(Y1, Y2)
Y1 <= Y2; Y1 >= Y2; Y1 != Y2; Y1 < Y2
hset1.included.to.hset2(Y1, Y2, exactly = TRUE) # Y1 %=<=% Y2
Y1 %=<=% Y3; Y1 %=<% Y2; Y1 %=<% Y3
```

Index

!= *(subset)*, 6
<, *4*
< *(subset)*, 6
<= *(subset)*, 6
== *(subset)*, 6
> *(subset)*, 6
>= *(subset)*, 6
%!imp% *(operations)*, 5
%!imp~% *(operations)*, 5
%+% *(operations)*, 5
%+~% *(operations)*, 5
%−% *(operations)*, 5
%−~% *(operations)*, 5
%-% *(operations)*, 5
%-~% *(operations)*, 5
%=<=% *(subset)*, 6
%=<% *(subset)*, 6
%=>=% *(subset)*, 6
%=>% *(subset)*, 6
%&% *(operations)*, 5
%&&% *(operations)*, 5
%&&~% *(operations)*, 5
%&~% *(operations)*, 5
%and% *(operations)*, 5
%and~% *(operations)*, 5
%in% *(in)*, 4
%or% *(operations)*, 5
%or~% *(operations)*, 5
%sum% *(operations)*, 5
%sum~% *(operations)*, 5
%xor% *(operations)*, 5
%xor~% *(operations)*, 5

as.generalized *(hset)*, 2
as.hset *(hset)*, 2
as.not.generalized *(hset)*, 2

cardinality *(hset)*, 2
clone.of.hset *(hset)*, 2

difference *(operations)*, 5

hset, 2
hset1.included.to.hset2 *(subset)*, 6
hsets.are.equal *(subset)*, 6

in, 4
inclusion.member *(in)*, 4
intersection *(operations)*, 5
is.generalized *(hset)*, 2
is.hset *(hset)*, 2

members *(hset)*, 2
multiplicities *(hset)*, 2

operations, 5

refer.to.hset *(hset)*, 2

setsum *(operations)*, 5
size.support *(hset)*, 2
subset, 6
symmdiff *(operations)*, 5

union *(operations)*, 5