Package ‘kst’

February 20, 2015

Version 0.2-1
Date 2014-1-10
Title Knowledge Space Theory
Description Knowledge Space Theory is a set-theoretical framework, which proposes mathematical formalisms to operationalize knowledge structures in a particular domain. The kst-package provides basic functionalities to generate, handle, and manipulate knowledge structures and knowledge spaces.
License GPL (>= 2)
Depends R (>= 2.7.0), proxy, relations (>= 0.4-0), sets (>= 0.3-2)
Suggests Rgraphviz
Author Christina Stahl [aut, cre],
    David Meyer [aut]
Maintainer Christina Stahl <christina.stahl@chello.at>
NeedsCompilation no
Repository CRAN
Date/Publication 2014-01-11 16:02:14

R topics documented:

as.relation .................................................. 2
closure ....................................................... 3
fringe ......................................................... 4
kassess ....................................................... 5
katoms ......................................................... 6
kbase ........................................................ 7
kdomain ...................................................... 8
knotions ..................................................... 9
kstructure .................................................. 10
kstructure_is_wellgraded ................................. 11
ktrace ....................................................... 12
kvalidate .................................................... 13
lpath ......................................................... 15
as.relation

| Description | Computes the surmise relation of knowledge structures. |
| Usage | ```
## S3 method for class 'kstructure'

as.relation(x, ...)
``` |
| Arguments | x | An R object of class `kstructure`. |
| ... | Currently not used. |
| Details | as.relation takes an arbitrary knowledge structure and computes its underlying surmise relation, i.e., the set of item pairs corresponding to the knowledge dependencies. Antisymmetric and transitive surmise relations may then be plotted as a Hasse diagram. |
| Value | An R object of class `relation`. |
| See Also | `kstructure`, `relation`, `plot.relation` |
| Examples | ```r

kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
    set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
as.relation(kst)
``` |
Description
Computes the closure of knowledge structures.

Usage

```r
## S3 method for class 'kstructure'
closure(x, operation=c("union", "intersection"),...)
```

Arguments

- `x`: An R object of class `kstructure`.
- `operation`: The set operation under which the closure is computed.
- `...`: Other arguments to be passed to methods.

Details
The `closure` method for objects of class `kstructure` performs the closure of a knowledge structure by computing the "union", "intersection", "complement", or symmetric difference of any two knowledge states. "union" is also used as a basis for the `kspace` function.

Value
An R object of the same class as `x` where each subset represents one knowledge state of the resulting knowledge structure.

Note
The implementation of union is more efficient than the one in `sets`.

References

See Also
`kstructure, kspace, closure`

Examples

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
                    set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
closure(kst, operation="union")
```
fringe

Fringes of Knowledge States

Description

Computes the inner and outer fringe of knowledge states.

Usage

kfringe(x, state=NULL, operation=c("inner", "outer"))

Arguments

x An R object of class kstructure.
state NULL (computes the fringes for all knowledge states in x) or an R object of class set (computes the fringe of a single knowledge state).
operation Either "inner" or "outer".

Details

Fringes determine the symmetric difference between a given knowledge state and its neighbouring states.

The symmetric difference between a given knowledge state and its predecessor states is referred to as inner fringe. These items can be interpreted as most recently learned and may for example be subject to learner review or assessment.

The symmetric difference between a given knowledge state and its immediate successor states is referred to as outer fringe. These items may be considered those a learner should tackle next and therefore allow for creating personalized learning paths.

Value

A list where each element represents the inner or outer fringe of one knowledge state respectively.

References


See Also

kstructure, set
Examples

```r
kst <- kstructure(set(set(), set("c"), set("a","b"), set("b","c"),
set("c","d"), set("d","e"), set("a","b","c"), set("b","c","d"),
set("c","d","e"), set("a","b","c","d"), set("a","b","d","e"),
set("b","c","d","e"), set("a","b","c","d","e")))

# inner fringe
kfringe(kst, operation="inner")
kfringe(kst, state=set("b", "c", "d", "e"), operation="inner")

# outer fringe
kfringe(kst, operation="outer")
kfringe(kst, state=set("b", "c", "d", "e"), operation="outer")
```

Description

Assigns individuals to their corresponding knowledge states.

Usage

```r
kassess(x, rpatterns=NULL, method="deterministic")
```

Arguments

- `x` An R object of class `kstructure`.
- `rpatterns` A binary data frame or matrix where each row specifies the response pattern of one individual to the set of domain problems in `x`.
- `method` The desired assessment method. Currently only "deterministic" assessment is implemented.

Details

`kassess` assigns individuals to their corresponding knowledge state in a knowledge structure.

Assessing individuals based on a "deterministic" procedure starts by determining a domain problem `a`, which is contained in approximately half of the available knowledge states. If the individual being assessed has successfully solved the respective domain problem `a`, all knowledge states that do not contain domain problem `a` are removed from the set of potential knowledge states of the individual. If, on the other hand, the individual has not solved the respective domain problem `a`, all knowledge states that do contain domain problem `a` are removed from the set of potential knowledge states of the individual. From the remaining knowledge states a domain problem `b`, which again is contained in approximately half of the still available knowledge states, is selected. If the individual has successfully solved the respective domain problem `b`, all knowledge states that do not contain domain problem `b` are removed from the set of potential knowledge states of the individual. If, on the other hand, the individual has solved the respective domain problem `b`, all knowledge states
that do contain domain problem \( b \) are removed from the set of potential knowledge states of the individual. This procedure is repeated until only one knowledge state is left. This is the knowledge state the individual is currently located in.

Value

A list where each element represents the knowledge state of one respondent.

References


See Also

kstructure

Examples

# deterministic assessment
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
                   set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))

rp <- data.frame(a=c(1,1,1,1,1,1,1,0,0,0), b=c(0,1,0,1,0,1,0,1,0,0),
                 c=c(0,0,0,1,1,0,1,0,0,0), d=c(0,0,1,1,1,1,0,0,1,0), e=c(0,0,1,1,1,1,0,0,0,0))
kassess(kst, rpatterns=rp, method="deterministic")

---

**katoms**  
*Atoms of Knowledge Structures*

**Description**

Computes atoms of knowledge structures.

**Usage**

katoms(x, items)

**Arguments**

- \( x \)  
  An \( \mathbb{R} \) object of class kstructure.

- \( \text{items} \)  
  A set of items for which atoms are computed.

**Details**

For any item \( q \) of the knowledge domain \( Q \), an atom at \( q \) is a minimal knowledge state containing \( q \), where minimal refers to the fact that the respective knowledge state is not the union of any other knowledge states.
kbase

Value

A list where each element represents the atom(s) of one item specified in `items`.

References


See Also

kstructure, set

Examples

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
                    set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
katoms(kst, items=set("a","b","c"))
```

---

**kbase**

*Base of a Knowledge Space*

Description

Computes the base of a knowledge space.

Usage

`kbase(x)`

Arguments

- `x`: An R object of class `kspace`.

Details

A base for a knowledge space is a minimal family of knowledge states spanning the knowledge space, i.e., the base includes the minimal states sufficient to reconstruct the full knowledge space. A knowledge structure has a base only if it is a knowledge space.

Value

A set of sets where each subset represents one knowledge state of the base.

References


See Also

kspace, kstructure, set
Examples

```r
kst <- kspace(kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
    set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e"))))
kbase(kst)
```

---

### kdomain

**Domain of Knowledge Structures**

#### Description

Computes the domain of knowledge structures.

#### Usage

```r
kdomain(x)
```

#### Arguments

- `x` An R object of class `kstructure`.

#### Details

A **domain** is a set of questions or items representing a field of knowledge.

#### Value

A **set** of items, each representing one question of the knowledge domain.

#### References


#### See Also

`kstructure`, `set`

#### Examples

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
    set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e"))
kdomain(kst)
```
**knotions**

**Notions of Knowledge Structures**

**Description**
Computes notions of knowledge structures.

**Usage**

```
knotions(x)
```

**Arguments**

- `x` An \( \mathbb{R} \) object of class `kstructure`.

**Details**

A *notion* is a set of items always jointly contained in some knowledge states. Consequently, these items carry the same information and may therefore be considered equivalent. A knowledge structure where each notion contains only one item is considered discriminative.

**Value**

A set of sets, each representing one notion of the knowledge structure.

**References**


**See Also**

`reduction.kstructure`, `kstructure`, `set`

**Examples**

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
    set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
knotions(kst)
```
kstructure

**Knowledge Structure**

**Description**

Creates a knowledge structure from a surmise relation or knowledge states.

**Usage**

```r
kstructure(x)
```

**Arguments**

- `x` Either an `endorelation` representing a surmise relation, or a `set` of sets each representing one knowledge state.

**Details**

The most basic assumption of knowledge space theory is that every knowledge domain can be represented in terms of a set of domain problems $Q$. Moreover, knowledge space theory assumes dependencies between these domain problems in that knowledge of a given domain problem or a subset of problems may be a prerequisite for knowledge of another, more difficult or complex domain problem. These prerequisite relations are realized by *surmise relations*, which create a quasi-order between different domain problems. One advantage of these surmise relations is that they reduce the quantity of all possible solution patterns to a more manageable amount of *knowledge states*. Each of these knowledge states represents the subset of domain problems an individual is capable of solving. The collection of all knowledge states captures the organization of the domain and is referred to as *knowledge structure*.

kstructure takes an endorelation representing a surmise relation or a set of sets each representing one knowledge state (e.g., one clause of a surmise system) and returns the corresponding knowledge structure.

**Value**

An R object of class kstructure.

**Note**

Note that by default the quotes indicate the fact that the items are represented by characters. For displaying purposes, these quotes may be turned off by means of the function `sets_options`.

**References**


**See Also**

`endorelation`, `set`, `sets_options`
### Examples

#### An endorelation representing a surmise relation
```r
kst <- endorelation(graph=set(tuple(1L,1), tuple(2,2), tuple(3,3),
                                 tuple(4,4), tuple(2,1), tuple(3,1), tuple(4,1),
                                 tuple(3,2), tuple(4,2)))
kstructure(kst)
```

#### A set of sets representing knowledge states (e.g., clauses of a surmise system)
```r
kst <- set(set("a"), set("a", "b"), set("a", "c"), set("d", "e"), set("a", "b", "d", "e"),
            set("a", "c", "d", "e"), set("a", "b", "c", "d", "e"))
kstructure(kst)
```

#### Turning off the quotes for displaying purposes
```r
sets_options("quote", FALSE)
kstructure(kst)
```

---

**kstructure_is_wellgraded**  
*Well-Gradedness of Knowledge Structures*

### Description

Tests for the well-gradedness of knowledge structures.

### Usage

```r
kstructure_is_wellgraded(x)
```

### Arguments

- **x**: An R object of class `kstructure`.

### Details

A knowledge structure is considered *well-graded* if any two of its states are connected by a bounded path, i.e., each knowledge state (except the state for the full set of domain problems $\mathcal{Q}$) has at least one immediate successor state that comprises the same domain items plus exactly one and each knowledge state (except the empty set $\emptyset$) has at least one predecessor state that contains the same domain items with the exception of exactly one.

`kstructure_is_wellgraded` takes an arbitrary knowledge structure and tests for its well-gradedness.

### Value

A logical value.

### References

See Also

kstructure

Examples

```r
kst <- kstructure(set(set(), set("a"), set("b"), set("c"), set("a","b"),
                set("b","c"), set("a","b","c")))
kstructure_is_wellgraded(kst)

kst <- kstructure(set(set(), set("a"), set("b"), set("c"), set("a","b"),
                set("a","b","c")))
kstructure_is_wellgraded(kst)
```

**ktrace**

*Trace of Knowledge Structures*

**Description**

Computes the trace of knowledge structures.

**Usage**

`ktrace(x, items)`

**Arguments**

- `x` An \( \mathbb{R} \) object of class `kstructure`.
- `items` A set of items for which the trace is computed.

**Details**

The *trace* of a knowledge structure \( K \) on a set \( A \) is the substructure of the knowledge structure \( K \) on the set \( A \), i.e., the substructure resulting from restricting the knowledge structure \( K \) to the items specified in \( A \).

**Value**

An \( \mathbb{R} \) object of class `kstructure` where each element represents one knowledge state of the knowledge structure on the item specified in `items`.

**References**


**See Also**

kstructure
Examples

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
        set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
ktrace(kst, items=set("c","d","e"))
```

---

**kvalidate**

**Validate Prerequisite Relations or Knowledge Structures**

**Description**

Validates prerequisite relations or knowledge structures.

**Usage**

```r
kvalidate(x, rpatterns=NULL, method=c("gamma","percent","VC","DA"))
```

**Arguments**

- `x`: An R object of class `kstructure`.
- `rpatterns`: A binary data frame or matrix where each row specifies the response pattern of one individual to the set of domain problems in `x`.
- `method`: The desired validation method (see details).

**Details**

`kvalidate` calculates validity coefficients for prerequisite relations and knowledge structures.

The **γ-Index** (method "gamma") validates the prerequisite relation underlying a knowledge structure and assumes that not every response pattern is represented by a prerequisite relation. For this purpose it compares the number of response patterns that are represented by a prerequisite relation (i.e., concordant pairs) with the number of response patterns that are not represented by a prerequisite relation (i.e., discordant pairs). Formally, the γ-Index is defined as

\[
\gamma = \frac{N_c - N_d}{N_c + N_d}
\]

where \( N_c \) is the number of concordant pairs and \( N_d \) the number of discordant pairs. Generally, a positive γ-value supports the validity of prerequisite relations.

The validation method "percent" likewise validates prerequisite relations and assumes that more difficult or complex domain problems are solved less frequently than less difficult or complex domain problems. For this purpose it calculates the relative solution frequency for each of the domain problems in \( Q \).

The **Violational Coefficient** (method "VC") also validates prerequisite relations. For this purpose, the number of violations (i.e., the earlier mentioned discordant pairs) against a prerequisite relation are calculated. Formally, the VC is defined as

\[
VC = \frac{1}{n(|S| - m)} \sum_{x,y} v_{xy}
\]
where \( n \) denotes the number of response vectors, \(|S|\) refers to the number of pairs in the relation, \( m \) denotes the number of items, and \( v_{xy} \) again refers to the number of discordant pairs. Generally, a low VC supports the validity of prerequisite relations.

In contrast to the other three indices, the **Distance Agreement Coefficient** (method "DA") validates the resulting knowledge structure. For this purpose it compares the average symmetric distance between the knowledge structure and response patterns (referred to as \( ddat \)) to the average symmetric distance between the knowledge structure and the power set of response patterns (referred to as \( dpot \)). By calculating the ratio of \( ddat \) and \( dpot \), the DA is determined. Generally, a lower DA-value indicates a better fit between a knowledge structure and a set of response patterns.

**Value**

Depending on the desired assessment method, a numeric value (methods "gamma" and "VC"), a data frame with results for each domain problem (method "percent"), or a list (method "DA") with the following components:

- **ddat** The ddat-value.
- **ddat_dist** The distance table for ddat.
- **dpot** The dpot-value.
- **dpot_dist** The distance table for dpot.
- **DA** The Distance Agreement Coefficient.

**References**


**See Also**

- kstructure

**Examples**

```r
tst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
             set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e"))
rt < data.frame(a=c(1,1,1,1,1,1,1,0,0,0), b=c(0,1,0,1,0,1,0,0,1),
    c=c(0,0,0,1,1,0,1,0,0,0), d=c(0,0,1,1,0,0,0,1), e=c(0,0,1,1,0,0,0,0))

# Gamma Index
kvalidate(tst, rpatterns=rt, method="gamma")

# Percent
```

kvalidate(kst, rpatterns=rp, method="percent")

# Violational Coefficient
kvalidate(kst, rpatterns=rp, method="VC")

# Distance Agreement Coefficient
kvalidate(kst, rpatterns=rp, method="DA")

### lpath

**Learning Paths in a Knowledge Structure**

#### Description
Computes learning paths in a knowledge structure.

#### Usage

```r
lpath(x)
```

#### Arguments

- `x` An R object of class `kstructure`.

#### Details
A learning path in a knowledge structure is a maximal sequence of knowledge states, which allows learners to gradually traverse a knowledge structure from the empty set $\emptyset$ (or any other bottom state) to the full set of domain problems $Q$.

`lpath` takes an arbitrary knowledge structure and computes all possible learning paths in the respective knowledge structure.

#### Value
A list where each element represents one learning path.

#### References

#### See Also

- `kstructure`

#### Examples

```r
kst <- kstructure(set(set()), set("a"), set("b"), set("a","b"),
    set("a","d"), set("b","c"), set("a","b","c"), set("a","b","d"),
    set("b","c","d"), set("a","b","c","d"), set("a","b","c","d","e")))
lpath(kst)
```
lpath_is_gradation  Gradation Property of Learning Paths

Description

Tests for the gradation property of learning paths.

Usage

lpath_is_gradation(x)

Arguments

x  A list of learning paths.

Details

A learning path is considered a gradation if each state in a learning path differs from its predecessor and/or successor state by a single item/notion.

lpath_is_gradation takes an arbitrary list of learning paths and tests for their gradation property.

Value

A list of logical values where each element represents one learning path.

References


See Also

kstructure, lpath

Examples

kst <- kstructure(set(set(), set("c"), set("a","b"), set("b","c"), set("c","d"), set("d","e"), set("a","b","c"), set("b","c","d"), set("c","d","e"), set("a","b","c","d"), set("a","b","d","e"), set("b","c","d","e"), set("a","b","c","d","e")))
lp <- lpath(kst)
lpath_is_gradation(lp)
Description

Plots a Hasse Diagram of knowledge structures.

Usage

```r
## S3 method for class 'kstructure'
plot(x, ...)
```

Arguments

- `x` An R object of class `kstructure`.
- `...` Other arguments to be passed to methods.

Details

`plot` takes an arbitrary knowledge structure and plots a Hasse Diagram of the respective knowledge structure (see README for details).

References


See Also

- `kstructure`

Examples

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
                     set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
if(require("Rgraphviz")) plot(kst)
```
Reduction of Knowledge Structures

Description
Computes the reduction of knowledge structures.

Usage
```r
## S3 method for class 'kstructure'
reduction(x, operation=c("discrimination", "union", "intersection"), ...)
```

Arguments
- `x`: An R object of class `kstructure`.
- `operation`: The set operation under which the reduction is computed.
- `...`: Other arguments to be passed to methods.

Details
`reduction` performs the reduction of a knowledge structure by computing the minimal subset having the same closure as the knowledge structure. Additionally, it allows for computing the discriminative reduction of a knowledge structure. Such a discriminative reduction is a knowledge structure in which each notion contains a single item.

Value
An R object of the same class as `x` where each subset represents one knowledge state of the resulting reduction.

References

See Also
- `kstructure`, `knotions`, `reduction`

Examples
```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
                   set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))
reduction(kst, operation="discrimination")
```
**Description**
Tests for and converts to knowledge space.

**Usage**

```r
kstructure_is_kspace(x)
kspace(x)
```

**Arguments**

- `x` An R object of class `kstructure`.

**Details**
A knowledge structure is considered a knowledge space if it includes one state for the empty set \( \emptyset \), one state for the full set of domain problems \( Q \), and a state for the union of any two knowledge states (i.e., the closure under union).

- `kstructure_is_kspace` takes an arbitrary knowledge structure and tests for its space property.
- `kspace` takes an arbitrary knowledge structure and returns the corresponding knowledge space.

**Value**
For `kstructure_is_kspace` a logical value.
For `kspace` an R object of class `kspace` where each subset represents one knowledge state of the knowledge space.

**References**

**See Also**
`kstructure`, `closure.kstructure`

**Examples**

```r
kst <- kstructure(set(set("a"), set("a","b"), set("a","c"), set("d","e"),
                     set("a","b","d","e"), set("a","c","d","e"), set("a","b","c","d","e")))

# test for knowledge space
kstructure_is_kspace(kst)

# convert to knowledge space
kspace(kst)
```
Index

*Topic math
  as.relation, 2
  closure, 3
  fringe, 4
  kassess, 5
  katoms, 6
  kbase, 7
  kdomain, 8
  knotions, 9
  kstructure, 10
  kstructure_is_wellgraded, 11
  ktrace, 12
  kvalidate, 13
  lpath, 15, 16
  lpath_is_gradation, 16
  plot, 17
  plot.relation, 2
  reduction, 18, 18
  reduction.kstructure, 9
  relation, 2
  set, 4, 6–10
  sets_options, 10
  space, 19

as.relation, 2

closure, 3, 3

closure.kstructure, 19

endorelation, 10

fringe, 4

kassess, 5
katoms, 6
kbase, 7
kdomain, 8
kfringe(fringe), 4
kfringe_inner(fringe), 4
kfringe_outer(fringe), 4
knotions, 9, 18
kspace, 3, 7
kspace(space), 19
kstructure, 2–9, 10, 11–19
kstructure_is_kspace(space), 19
kstructure_is_wellgraded, 11

ktrace, 12
kvalidate, 13

lpath, 15, 16
lpath_is_gradation, 16

plot, 17
plot.relation, 2

reduction, 18, 18
reduction.kstructure, 9
relation, 2

set, 4, 6–10
sets_options, 10
space, 19