HCGene package: Reference Manual

(v. 1.0 - October 2007)

Giorgio Valentini
D.S.I.
Dipartimento di Scienze dell' Informazione
Università degli Studi di Milano
e-mail : valentini@dsi.unimi.it

Index

`hcgene-package`	*HCGene: a software tool to support the hierarchical classification of genes*

Description

The R package *HCGene* (*H*ierarchical *C*lassification of *Gene*s), built on top of Bioconductor libraries, implements methods to process and analyze the directed acyclic graphs of the *Gene Ontology (GO)* and the trees of the *FunCat* taxonomy of genes. *HCGene* has been designed to support the hierarchical classification of genes, but in principle it can also be used for other prediction or exploratory tasks that take into account the hierarchical structure of gene ontologies. *HCGene* allows the extraction of subgraphs and subtrees related to specific biological problems, the labelling of genes and gene products with multiple and hierarchical functional classes, and the association of different types of bio-molecular data to genes for learning to predict their functions.

Details

Package:	hcgene
Type:	Package
Version:	1.0
Date:	2007-10-15
License:	GNU/GPL

Assigning functions to unannotated gene products using large-scale bio-molecular data is a key issue in functional genomics and bioinformatics.

Most computational methods to predict the functions of genes have been based on a "flat" multiclass setting of the classification problem, without taking into account the hierarchical structure of gene classes.

Nevertheless, the structure of the *Gene Ontology (GO)* and *FunCat* taxonomies, requires a new setting of the classical multi-class classification problem. Indeed these ontologies assign multiple classes to gene/gene products, with annotations available at different degree of resolution and reliability, and with hierarchical relationships between the classes. More precisely the *GO* provides a taxonomy structured as a Directed Acyclic Graph (DAG), while *FunCat* organizes functional categories of genes according to a tree.

In this context the problem of the gene function prediction must be restated in a multi-label, hierarchical and partial-path setting of the classical statistical and machine learning multi-class classification problem.

HCGene implements methods to process complex taxonomies structured as graphs (e.g. DAG or trees) with thousands of nodes and edges, in order to extract subgraphs related to the specific biological process under investigation, or to properly associates multiple functional classes to gene and gene products according to the hierarchical structure of the FunCat trees or GO DAGs, or to associate gene products to multiple data types (e.g. gene

expression data, phylogenetic or protein interaction data) used to infer the function of unknown genes.

Schematically the main functionalities of the software library can be summarized as follows:

Graph processing: building of hierarchical structures covering the main ontologies (trees and graphs); methods to analyze the structure and the relationships between functional classes (e.g. distribution of node/classes with respect to their depth, in and out-degree, cardinality of classes or labels, distribution of leaves at different levels); methods to extract biologically meaningful structures from GO DAGs and FunCat trees.

Multi-label generation: extraction of the most specific annotations and building of the full annotation of genes exploiting the transitive relationships between classes; building of the multi-labeling for each gene using compact representations; mapping functions to associate gene names or identifiers (e.g. ORF ID or EntrezGene IDs) to functional classes

Data building: methods to associate gene names to different types of data; methods to select positive and negative examples for each class according to different strategies; methods to build data relative to specific functional classes.

Moreover functions to graphically show the results of statistical analyses, as well as to draw subgraphs of GO and FunCat ontologies are provided .

The software library is built on top of the *Bioconductor* packages **graph**, **GO**, **GOstats**, and **Rgraphviz** to provide functionalities related to the *GO* ontology, while for the *FunCat* taxonomies the software has been built from scratch using the hierarchical schemes and functional annotations obtained from the MIPS website (`http://mips.gsf.de`).

HCGene has been designed to support the supervised hierarchical classification of genes, but it can also be used to support the development of methods that incorporate a priori biological knowledge into analysis, by exploiting the *HCGene* functions for the Funcat and GO graph processing, and for the association of genes to the functional classes.

Author(s)

Giorgio Valentini <valentini@dsi.unimi.it>.

References

Barutcuoglu, Z., Schapire, R.E. and Troyanskaya, O.G., Hierarchical multi-label prediction of gene function, *Bioinformatics*, 22(7), 830-836, 2006.

Dopazo, J., Functional Interpretation of Microarray Experiments, *OMICS*, 10(3), 2006.

Harris, M.A. and others, The Gene Ontology (GO) database and informatics resource, *Nucleic Acid Res.*, 32, D258–D261, 2004.

Lewis, D.P., Jebara, T. and Noble, W.S., Support vector machine learning from heterogeneous data: an empirical analysis using protein sequence and structure, *Bioinformatics*, 22(22), 2753-2760, 2006.

Pavlidis, P., Weston, J., Cai, J. and Noble, W.S., Learning gene functional classification from multiple data, *J. Comput. Biol.*, 9, 401-411, 2002.

Ruepp, A and others, The FunCat, a functional annotation scheme for systematic classification of proteins from whole genomes, *Nucleic Acid Res.*, 32, 5539-5545, 2004.

See Also

graph, *GO*, *GOstats*, *Rgraphviz*

`AnalyzeFuncatDepth` *Functions to analyze the depth of FunCat nodes*

Description

Functions to analyze the depth of FunCat nodes and the distribution of the leaves in FunCat trees

Usage

```
Get.depth.Funcat.labels(nodes)
Get.leaf.distribution.by.depth.Funcat.labels(g)
```

Arguments

`nodes`	a list(vector) of FunCat nodes
`g`	a FunCat graph

Details

`Get.depth.Funcat.labels` is a function that provides the depths of FunCat classes. `Get.leaf.distribution.by.depth.Funcat.labels` computes the distribution of the leaves per depth of a FunCat tree. It returns how many leaves are present for each level (depth) of the tree. Note that the level of the root nodes is set to 1.

Value

`Get.depth.Funcat.labels` returns a vector with the depth of Funcat classes supplied as arguments.

`Get.leaf.distribution.by.depth.Funcat.labels` returns a list with 4 elements:

`num.levels`	the number of levels (depth) of the FunCat tree. Note that the root is considered of depth
`num.leaves.per.level`	a vector with the number of leaves for each depth
`num.no.leaves.per.level`	a vector with the number of nodes that are not leaves for each depth
`ratio.leaves.per.level`	the ratio of the leaves for each depth

See Also

`GetFuncatDepth`

Examples

```
# doing universal graph of Funcat classes
g <- Do.universal.tree.Funcat();
# Getting leaf distribution of the universal FunCat tree
Get.leaf.distribution.by.depth.Funcat.labels(g);
# Getting the depths of all FunCat classes
Get.depth.Funcat.labels(nodes(g))
```

ATDESCRIPTION *Description of A. thaliana AGI identifiers*

Description

This is an R environment (hash table) mapping AGI identifiers to their corresponding protein description.

Usage

```
data(ATDESCRIPTION)
```

Format

An environment with 29993 entries

Details

The environment provides protein descrition of the model higher plant Arabidopsis thaliana. AGI identifiers are Keys and the corresponding protein description are Values. Values are character vectors.

Source

Data have been downloaded from Arabidopsis Information Resource (TAIR):
http://www.arabidopsis.org

Examples

```
data(ATDESCRIPTION);
str(ATDESCRIPTION);
get("at1g01100",ATDESCRIPTION);
```

ATFUNCAT *AGI identifiers to Functional Classification (FunCat) mapping*

Description

This environment provides a mapping between A.thaliana genes and FunCat classes.

Usage

```
data(ATFUNCAT)
```

Format

An environment with 26639 entries

Details

The environment ATFUNCAT maps AGI IDs of the A.thaliana to the Funcat ID of the classes (http://mips.gsf.de/projects/funcat). Each entry associates the AGI ID to the the list of the Funcat ID. More precisely each Value is a list of lists. Each element of the outer list correspond to an association to a Funcat class. The elements of the inner list are : FuncatID the store the identifier of the Funcat class and Evidence that stores the Funcat numeric identifier of the evidence of the annotation. No evidence code is available at the moment (all the Evidence fields are empty). The annotation is conformed to the funcat-2.1 scheme.

Source

Data have been downloaded from the MIPS Arabidopsis thaliana Database:
ftp://ftpmips.gsf.de/plants/cress/funcatdump_v220306

Examples

```
data(ATFUNCAT)
get("at1g01100",ATFUNCAT);
```

ATGO *AGI identifiers to Gene Ontology (GO) mapping*

Description

This environment provides a mapping between A.thaliana genes and GO terms.

Usage

```
data(ATGO)
```

Format

An environment (hash table) with 28234 entries

Details

The environment provides GO annotations for the model plant Arabidopsis thaliana. AGI identifiers are Keys and GO terms Values. More precisely each Value is a list of lists. Each element of the outer list represent a mapping to a GO term. The elements of the inner list are:

GOID : GO Identifier

Evidence : Evidence code

Ontology : "BP", "MF" or "CC"

Source

Data have been downloaded from Arabidopsis Information Resource (TAIR):
http://www.arabidopsis.org.
The actual file used for the annotations has been downloaded from:
ftp://ftp.arabidopsis.org/home/tair/Ontologies/Gene_Ontology/ATH_GO_GOSLIM.20070929.txt

Examples

```
data(ATGO)
get("at1g01100",ATGO);
```

Build.class.labels.from.selected.classes

Extraction of a subset of classes from a given functional table

---

Description

It extracts from a given table of genes–>classes (functional table) a subtable with the given subset of classes. It may be used with both GO and Funcat ontologies If some of the selected classes are not present in the Table the execution is aborted with an error message.

Usage

```
Build.class.labels.from.selected.classes(Table.classes, classes)
```

Arguments

`Table.classes`	a data frame with rows corresponding to genes and columns to classes. If a gene belongs to the GO/Funcat class the corresponding variable is set to 1 otherwise to 0
`classes`	list or character vector of the selected Funcat/GO classes

Value

a data frame extracted from Table.classes according to the selected classes

See Also

`Build.GO.class.labels`, `Build.Funcat.Table.labels`

Examples

```
# building of a data frame with FunCat classes with more than 100 examples for the yeast
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Yeast.Funcat.general <- Get.Funcat.all.classes(Yeast.Funcat.specific);
Yeast.Funcat.Table <- Build.Funcat.Table.labels(Yeast.Funcat.general);
nodes <- Select.functional.classes.by.cardinality(Yeast.Funcat.Table, 100);
Yeast.Funcat.Table.100 <- Build.class.labels.from.selected.classes(Yeast.Funcat.Table, nodes);
```

`Build.Funcat.graph.from.selected.nodes`

Construction of a subgraph of a given FunCat graph

Description

It builds a subgraph of a given FunCat graph from a list of Funcat classes (nodes).

Usage

```
Build.Funcat.graph.from.selected.nodes(g, nodes, transitive.closure = TRUE)
```

Arguments

`g`	a FunCat graph
`nodes`	list or character vector of the selected nodes
`transitive.closure`	if TRUE (default) the nodes used for building the graph are obtained by transitive closure of nodes

Details

The subgraph can be obtained from a list of selected nodes or by transitive closure from a list of selected nodes If some of the selected nodes are not present in the graph the execution is aborted with an error message. Note that we always obtain trees, because all FunCat graphs are trees.

Value

a subgraph of class graphNEL extracted from the graph g

See Also

Funcat.transitive.closure, Do.universal.tree.Funcat

Examples

```
# building a subtree with yeast specific nodes
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
t <- Do.universal.tree.Funcat();
nodes <- Get.classes(Yeast.Funcat.specific);
st <- Build.Funcat.graph.from.selected.nodes(t, nodes, transitive.closure = TRUE)
```

Build.Funcat.Table.labels

Construction of a data frame that maps genes to multiple FunCat functional classes

---

Description

It builds a boolean data.frame with genes (rows) and class codes (columns/variables). If gene i belongs to class j the data.frame d[i,j] = 1, otherwise it is set to 0.

Usage

```
Build.Funcat.Table.labels(NamesToFuncat, store = FALSE,
                          file = "Table.gene.Funcat.classes.rda")
```

Arguments

NamesToFuncat	named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding Funcat classes ID.
store	boolean: if TRUE the result is stored in a binary file
file	name of the file where the data.frame is stored (if store == TRUE)

Value

a data.frame with columns corresponding to Funcat classes and rows to a given gene. If a gene belongs to the Funcat class the corresponding column is set to 1 otherwise to 0.

See Also

Get.Funcat.specific.classes, Get.Funcat.all.classes

Examples

```
# building of a data frame with multiple FunCat functional labels for the yeast
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Yeast.Funcat.general <- Get.yeast.Funcat.all.classes(Yeast.Funcat.specific);
Yeast.Funcat.Table <- Build.Funcat.Table.labels(Yeast.Funcat.general);
```

Build.GO.class.labels

Construction of a data frame that maps genes to multiple GO classes

---

Description

It builds from the list of gene->GOID a boolean data frame with genes (rows) and class codes (columns). If gene i belongs to class j the data frame d[i,j] = 1, otherwise it is set to 0.

Usage

```
Build.GO.class.labels(NamesToGO, store = FALSE, file = "Table.gene.GO.classes.rda")
```

Arguments

NamesToGO	named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding GO classes ID
store	if TRUE the result is stored in a binary file
file	name of the file where the data frame is stored (if store == TRUE)

Value

a data frame (functional table) with variables corresponding to GO classes and rows to a given gene. If a gene belongs to the GO class the corresponding variable is set to 1 otherwise to 0.

See Also

Get.GO.specific.classes, Get.GO.all.classes

Examples

```
# building of a data frame with multiple GO functional labels for the yeast
Yeast.specific <- Get.yeast.GO.specific.classes();
Yeast.general <- Get.yeast.GO.all.classes(Yeast.specific);
Yeast.general.classes <- Build.class.labels(Yeast.general);
```

Build.GO.graph.from.selected.nodes

Construction of a subgraph of a given GO graph

Description

It builds a subgraph of a given GO graph form a list of GO classes (nodes)

Usage

```
Build.GO.graph.from.selected.nodes(g, nodes, transitive.closure = TRUE,
                                   ontology="BP")
```

Arguments

g	a GO graph
nodes	list or character vector of the selected nodes
transitive.closure	if TRUE (default) the nodes used for building the graph are obtained by transitive closure of nodes
ontology	a character vector denoting the ontology : BP (default), MF, CC

Details

The subgraph can be obtained from a list of selected nodes or by transitive closure from a list of selected nodes If some of the selected nodes are not present in the graph the execution is aborted with an error message.

Value

a subgraph extracted from the graph g

See Also

GO.transitive.closure, Build.universal.graph.ontology.down

Examples

```
# building a subgraph with yeast specific nodes
g <- Build.universal.graph.ontology.down();
Yeast.specific <- Get.yeast.GO.specific.classes();
selected.classes <- Get.classes(Yeast.specific);
sg <- Build.GO.graph.from.selected.nodes(g, selected.classes,
                                         transitive.closure = TRUE);
```

Build.list.homology.data

Construction of the list of lists of homology data for different species

---

Description

Functions to build the list of lists of homology data for different species.

Usage

```
Build.list.arabidopsis.homology.data(list.HGID = NULL)
Build.list.human.homology.data(list.EntrezGeneID = NULL)
Build.list.yeast.homology.data(list.ORFID = NULL)
Build.list.mouse.homology.data(list.EntrezGeneID = NULL)
```

Arguments

`list.HGID` a vector of HomoloGeneID for the Arabidopsis species. If NULL (default) the list of HomoloGeneID is obtained from the the vector athhomology-HGID of the package athhomology

`list.EntrezGeneID`

`list.ORFID`

a vector of ORF ID for the yeast species. If NULL (default) the list of ORFID is obtained from the the environment YEASTGO of the package YEAST

Details

Each element of the list corresponds to a specific gene; each gene has a list of homology data (one for each species and type of homology). Data are obtained from Bioconductor packages **athhomology**, **hsahomology**, **scehomology**, **mmuhomology** version 1.16.0.

Value

a list of lists of homology data: for each gene ID a list of homology data is given. The element of the inner list are:

homoORG	organism for which there is a homologous gene. This will be an abbreviation of the first letter of the genus and the first two letters of the species.
homoType	type of similarity measurement. This can be either B, indicating a reciprocal best best match between three or more organisms, b, indicating a reciprocal best match between two organisms, or c, indicating a curated homology relationship between two organisms.
homoPS	percentage of identical base pair alignments between the homologous genes
homoHGID	the internal HomoloGeneId.
homoURL	url to the source for a curated homology

Examples

```
l.human <- Build.list.human.homology.data();
l.Ath <- Build.list.arabidopsis.homology.data();
```

Build.universal.graph.ontology

Construction of the graph of an overall GO ontology

Description

It builds the DAG of an given GO ontology; both the BP (Biological Process), the MF (Molecular Function) and CC (Cellular Component) ontologies may be represented.

Usage

```
Build.universal.graph.ontology.down(ontology = "BP")
Build.universal.graph.ontology.up(ontology = "BP")
```

Arguments

ontology	it may be one of the following character vectors: BP (default), MF, CC

Details

The direction of the edges are from parent to child (from the less to the more specific GO term) for the function `Build.universal.graph.ontology.down`.
The function `Build.universal.graph.ontology.up` builds a DAG with the edges from children to parents.

Value

a graph of class graphNEL of the corresponding ontology

Examples

```
# bulding of the Cellular component GO ontology
g.CC <- Build.universal.graph.ontology.down("CC");
```

`ClassNames` *Getting names or information about GO and FunCat classes*

Description

These are a set of functions to obtain information or class names from the FunCat taxonomy and Gene Ontology.

Usage

```
Get.Term.Definition(goid)
Get.classes(NamesToGO)
Get.all.Funcat.classes()
Get.all.GO.classes(ontology = "BP")
```

Arguments

`goid`	GO ID of a GO class (character)
`NamesToGO`	named list of vectors. Each element of the list corresponds to a named yeast gene and the elements of the vector are the corresponding GO/FunCat classes ID.
`ontology`	a char vector denoting the ontology: BP (default), MF, CC

Details

`Get.Term.Definition` returns the GO term and definition for a given GO ID. The function `Get.classes` obtains the GO or Funcat classes from the named list of genes. More precisely It performs of a union of all the classes associated to the named list of genes. `Get.all.Funcat.classes` returns the FunCat ID of all the FunCat classes. The function `Get.all.GO.classes` gets all the GO classes from a given ontology

Value

`Get.Term.Definition` returns the GO term and definition for a given GO ID. `Get.classes` returns a vector of GOID or FunCat ID. `Get.all.Funcat.classes` returns a character vector of all the Funcat classes. `Get.all.GO.classes` returns a character vector of the GO classes.

Examples

```
Get.Term.Definition("GO:0008150");
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Get.classes(Yeast.Funcat.specific);
Get.all.Funcat.classes();
Get.all.GO.classes(ontology = "MF");
```

Code.decode.classes *Coding GO/FunCat classes to integers*

Description

Function that codes GO/FunCat classes to integers It provides a vector of class ID: the index of the vector is the corresponding integer code of the class. The function return also an hash table to obtain the integer code of the class from the GO/FunCat ID. This function is primarily intended for internal usage. It is used by `Build.Funcat.Table.labels` and `Build.GO.class.labels`.

Usage

```
Code.decode.classes(NamesToClasses)
```

Arguments

`NamesToClasses`
: named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding GO/FunCat classes ID.

Value

A list with two elements:

`Code.to.ClassID`
: A vector of GOID. The indices of the vector correspond to the integer codes of the classes. It maps Class integer codes –> GOID

`ClassID.to.Code`
: an environment (hash table) that associates ClassIDs to integer codes of the classes. It maps GO/FunCat ID –> Class integer codes

See Also

`Build.Funcat.Table.labels`, `Build.GO.class.labels`

Examples

Code.homology.org *Coding species names to integer*

Description

It finds all the homologous species from the list of lists of homology data and code the organism names to integers. The list of lists of homology data may be obtained from e.g. `Build.list.human.homology.data` or other similar functions for the species A. thaliana, yeast and mouse. It returns an evironment that associates the species name with an integer

Usage

```
Code.homology.org(l)
```

Arguments

`l`	a list of lists of homology data

Value

An environment that associates a three character identifier of a species with an integer

See Also

`Build.list.human.homology.data`, `Build.list.mouse.homology.data`

Examples

```
l<-Build.list.arabidopsis.homology.data();
sp2code <- Code.homology.org(l);
```

Compute.depth.statistics.Funcat.labels *Statistics about the distribution of the depth of FunCat classes*

Description

Function that provides statistics about the distribution of the depth of FunCat classes. It computes the average median, standard deviation and main quantiles of the depths of FunCat nodes.

Usage

```
Compute.depth.statistics.Funcat.labels(nodes)
```

Arguments

nodes	a list(vector) of FunCat nodes

Value

A list with four elements:

avg	mean value of the depth of FunCat classes
median	median value of the depth of FunCat classes
sd	standard deviation of the depth of FunCat classes
quant	0%, 25%, 50%, 75%, 100% quantile of the the depth of FunCat classes

See Also

Compute.depth.statistics.GO.graph

Examples

```
# Computing depth statistics of the FunCat nodes related to S. cerevisiae
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Yeast.Funcat.general <- Get.Funcat.all.classes(Yeast.Funcat.specific);
classes<-Get.classes(Yeast.Funcat.general);
Compute.depth.statistics.Funcat.labels(classes);
```

Compute.depth.statistics.GO.graph

Statistics about the "depth" of the nodes in a given GO graph

---

Description

Function that provides statistics about the "depth" of the nodes in a given GO graph. It can be used also with Funcat trees (represented as graphs) to provide basic statistics about the depth of the tree. The distances (shortest paths) from the root of the DAG are computed for all nodes and basic statistic are returned. This function can be used also with graphs relative to Funcat hierarchical trees. It computes the average median, standard deviation and main quantiles of the distances from the root.

Usage

```
Compute.depth.statistics.GO.graph(g, ontology = "BP")
```

Arguments

g	a graph of a GO/FunCat ontology: it must be rooted to the root note of the ontology
ontology	BP (default), MF, CC, 00 (FunCat)

Value

A list with four elements:

avg	mean value of the distances from the root
median	median value of the distances from the root
sd	standard deviation of the distances from the root
quant	0%, 25%, 50%, 75%, 100% quantile of the distances from the root

See Also

Compute.depth.statistics.Funcat.labels

Examples

```
# Computing depth statistics of the GO terms related to "universal" BP GO graph
gBP.universal.ontology <- Build.universal.graph.ontology.down();
Compute.depth.statistics.GO.graph(gBP.universal.ontology);
```

Compute.distances.from.root

Computing all the distances from the root of a given ontology

---

Description

Function to compute all the distances from the root of a given ontology The Dijkstra algorithm is used to compute the shortest paths (distances) from the root of the ontology to all the other nodes of the graph. It may be used also to compute the path length from the root of the FunCat ontology to all the other nodes of the tree.

Usage

```
Compute.distances.from.root(g, ontology = "BP")
```

Arguments

g	a graph of a GO or Funcat ontology: it must be rooted to the root note of the ontology
ontology	BP (default), MF, CC, 00 (Funcat)

Value

a named vector with the distance from the root of the GO DAG or FunCat tree

See Also

Compute.depth.statistics.GO.graph

Examples

```
# Computing the distances from the root of the GO terms related to "universal" BP GO graph
gBP.universal.ontology <- Build.universal.graph.ontology.down();
d <- Compute.distances.from.root(gBP.universal.ontology, ontology="BP");
```

Compute.statistics.degree

Statistics about the distribution of the in and out degree of GO and FunCat nodes

---

Description

The functions provide basic statistics about the distribution of the in and out degree of GO nodes and the out degree of FunCat nodes.

Usage

```
Compute.statistics.Funcat.degree(g)
Compute.statistics.GO.degree(g)
```

Arguments

g	a GO or FunCat graph

Details

The function `Compute.statistics.Funcat.degree` provides statistics about the out degree of FunCat nodes: the average, median, standard deviation and main quantiles of the nodes degrees are computed, as well as the degree of each individaul node. The function `Compute.statistics.GO.degree` is similar, but provides both the in and out degree statistics for GO nodes (GO is a DAG indeed).

Value

For `Compute.statistics.Funcat.degree`, a list with two elements:

`outDegree`	a list of 4 elements: avg : mean value of the out degree per Funcat class; median : median value of the out degree per Funcat class sd : standard deviation of the out degree per Funcat class quant : 0%, 25%, 50%, 75%, 100% quantile of the out degree per Funcat class
`degree`	a list with the out degree for each Funcat class
`outDegree`	a list of 4 elements: avg : mean value of the out degree per GO class; median : median value of the out degree per GO class sd : standard deviation of the out degree per GO class quant : 0%, 25%, 50%, 75%, 100% quantile of the out degree per GO class

inDegree	a list of 4 elements: avg : mean value of the in degree per GO class; median : median value of the in degree per GO class sd : standard deviation of the in degree per GO class quant : 0%, 25%, 50%, 75%, 100% quantile of the in degree per GO class
degree	a list with the in and out degree for each GO class

Examples

```
# Computing the distances from the root of the GO terms related to "universal" CC GO graph
gBP.universal.ontology <- Build.universal.graph.ontology.down("CC");
l <- Compute.statistics.GO.degree(gBP.universal.ontology);
```

Compute.statistics.functional.classes

Statistics about the distribution of the functional classes

Description

Function that provides statistics about the distribution of GO or FunCat functional classes. It may be used with both GO and Funcat ontologies. It computes the average, median, standard deviation and main quantiles of the cardinality of the functional gene classes.

Usage

```
Compute.statistics.functional.classes(Table.gene.class)
```

Arguments

Table.gene.class	a data frame with variables corresponding to functional classes and rows to a given gene. If a gene belongs to the functional class the corresponding variable is set to 1 otherwise to 0

Value

A list with four elements:

avg	mean value of the number of genes per functional class
median	median value of the numberof genes per functional class
sd	standard deviation of the number of genes per functional class
quant	0%, 25%, 50%, 75%, 100% quantile of the number of genes per functional class

Examples

```
# computing statistics about yeast FunCat classes
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Yeast.Funcat.general <- Get.Funcat.all.classes(Yeast.Funcat.specific);
Yeast.Funcat.Table <- Build.Funcat.Table.labels(Yeast.Funcat.general);
# the first variable refers to the dummy "00" class
Compute.statistics.functional.classes(Yeast.Funcat.Table[,2:ncol(Yeast.Funcat.Table)]);
```

Compute.statistics.gene.labels

Statistics about the distribution of the gene labels

Description

Function that provides statistics about the distribution of the gene labels. It may be used with both GO and Funcat ontologies. It provides the mean, median, standard deviation and main quantiles of the number of labels for each gene in GO or FunCat ontologies.

Usage

```
Compute.statistics.gene.labels(NamesToGO)
```

Arguments

NamesToGO	named list of vectors. Each element of the list corresponds to a yeast gene and the elements of the vector are the corresponding GO/Funcat classes ID.

Value

A list with four elements:

avg	mean value of the number of labels per gene
median	median value of the number of labels per gene
sd	standard deviation of the number of labels per gene
quant	0%, 25%, 50%, 75%, 100% quantile of the number of labels per gene

Examples

```
# computing statistics about FunCat labels per gene in the yeast
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Yeast.Funcat.general <- Get.Funcat.all.classes(Yeast.Funcat.specific);
Compute.statistics.gene.labels(Yeast.Funcat.general);
```

Count.examples.per.class *Counting the number of examples for each class*

Description

Function to compute the number of examples per class. It may be used with both GO and Funcat ontologies

Usage

```
Count.examples.per.class(Table.gene.class)
```

Arguments

Table.gene.class	a data frame with variables corresponding to GO/Funcat classes and rows to a given gene. If a gene belongs to the GO/Funcat class the corresponding variable is set to 1 otherwise to 0

Value

A list with two elements:

counts	a vector with the number of examples for each class
counts.freq	a table with the frequencies of the classes with respect to the number of gene/examples

Examples

```
# counting the number of examples for each class in yeast (FunCat)
data(YEASTFUNCAT);
Yeast.Funcat.specific <- Get.Funcat.specific.classes(YEASTFUNCAT);
Yeast.Funcat.general <- Get.Funcat.all.classes(Yeast.Funcat.specific);
Yeast.Funcat.Table <- Build.Funcat.Table.labels(Yeast.Funcat.general);
l <- Count.examples.per.class(Yeast.Funcat.Table);
```

Do.data.homology *Function to build homology (phylogenetic) data.*

Description

These functions build homology data matrices for different species. The homology data can be in binary format (1 homology), (0 no homology) or real values between 0 and 1 corresponding to the percentage of identical base pair alignments between the homologous genes. Data matrices are built using lists obtained by `Build.list.arabidopsis.homology.data`, `Build.list.mouse.homology.data`, `Build.list.human.homology.data`, `Build.list.yeast.homology.data`.

Usage

```
Do.binary.data.homology(l)
Do.float.data.homology(l)
```

Arguments

`l`	a list of lists of homology data. Note that this must be a named list having name of the genes as names

Details

Each element of the list represent a list of four elements of the same types as described in `Build.list.homology.data`. The function `Do.binary.data.homology` returns binary homology data: each column of the matrix corresponds to a given genome and it is set to 1 if there is a homology, 0 otherwise. The function `Do.float.data.homology` returns real valued homology data: each column of the matrix corresponds to a given genome and it is set with a value from 0 to 1 depending on the percentage of identical base pair alignments between the homologous genes.

Value

a matrix : rows are genes and columns correspond to homologies with other species. Each column corresponds to a species: a 1 entry corresponds to a homology, 0 otherwise (`Do.binary.data.homology`), or values are between 0 and 1 (`Do.float.data.homology`)

See Also

`Build.list.arabidopsis.homology.data`, `Build.list.mouse.homology.data`, `Build.list.human.homology.data`, `Build.list.yeast.homology.data`

Examples

```
# Building the matrix m of binary homology data for the yeast
l<-Build.list.yeast.homology.data();
m <- Do.binary.data.homology(l);
```

`Do.universal.tree.Funcat` *Generation of the universal tree of the FunCat taxonomy*

Description

Functions to build the universal tree of the FunCat ontology.
The function `Do.universal.tree.Funcat` builds a tree with edges from the root to the leaves (from top to bottom). The function `Do.universal.tree.Funcat.up` builds a tree with edges from the children to the parent (from bottom to up).

Usage

```
Do.universal.tree.Funcat()
Do.universal.tree.Funcat.up()
```

Details

The "universal" tree of the FunCat ontology (http://mips.gsf.de/projects/funcat) is built and represented as a graph of class graphNEL (see package **graph**). It collects all the available FunCat classes according to the FunCat scheme 2.1. Actually 1363 nodes are available.

Value

the tree of the Funcat classes stored as a graph of class graphNEL.

Note

A dummy root node with Funcat ID 00 is added in order to obtain a single tree from the FunCat forest. In the scheme 2.1 the node 40.02 is missed (but its child 40.02.03 is present instead), hence to avoid inconsistencies we added the node 40.02.

Examples

```
# Building the universal tree of FunCat
t <- Do.universal.tree.Funcat();
t
```

Extract.class	*Function to extract the examples belonging to a given GO/Funcat class*

Description

Function to extract the examples belonging to a given GO/Funcat class. It may be used with both GO and Funcat ontologies

Usage

```
Extract.class(Table.gene.class, class)
```

Arguments

Table.gene.class	a data frame with variables corresponding to GO/Funcat classes and rows to a given gene. If a gene belongs to the GO/Funcat class the corresponding variable is set to 1 otherwise to 0
class	a GO/Funcat class. It may be the integer code of the class or a string of the corresponding GO/FunCat ID

Value

a data frame: Each row corresponds to a gene belonging to class. The variables are: gene.names : strings with the name of the gene indices : the integer index of the gene

See Also

Get.matrix.data.without.ancestors, Get.matrix.data.from.parent.only

Examples

Funcat.navigation	*Navigation in FunCat trees: getting children, parent and ancestors*

Description

Functions to navigate inside FunCat trees and to obtain children, parent and ancestors of a given node. Also functions giving the depth of each node and to know if a node is a leaf or an internal node are given. Two types of functions to navigate the trees: the first set is based on a given FunCat tree, the second set exploits the name of the FunCat nodes to obtain children, parent and ancestors.

Usage

```
Funcat.child(g, node)
Funcat.parent(gup, node)
Funcat.ancestors(gup, node)
GetFuncatParent(funcatID)
GetFuncatAncestors(funcatID)
GetFuncatDepth(funcatID)
Is.leaf(x, g)
```

Arguments

g	graph of the Funcat tree. It must have edges from parent to children
gup	graph of the Funcat tree. It must have edges from children to parent
node	vector of the nodes (FuncatID)
funcatID	Funcat ID
x	Funcat ID

Details

The function `Funcat.child` provides the children of a given node. The functions `Funcat.parent` and `GetFuncatParent` provide the parent of a given node. The functions `Funcat.ancestors` and `GetFuncatAncestors` provide the ancestors of a given node. `GetFuncatDepth` provides the depth of a given node and `Is.leaf` reveals if a given node is a leaf. The main difference between the navigation functions are related the structure of a given tree and on the name of the FunCat class. In the first case results depends on the structure of the tree given as argument to the function. In the second case results depends only on the overall "universal" FunCat tree. The second type of functions are faster.

Value

A vector of the children (`Funcat.child`), parent (`Funcat.parent`, `GetFuncatParent`), ancestors (`Funcat.ancestors`,`GetFuncatAncestors`) of funcatID. The function `Is.leaf` returns TRUE if x is a leaf, otherwise FALSE. The function `GetFuncatDepth` returns the depth of the node.

Examples

```
t <- Do.universal.tree.Funcat();
t.up <- Do.universal.tree.Funcat.up();
Funcat.child(t, "01");
Funcat.ancestors(t.up,"01.03.01.01");
GetFuncatAncestors("01.03.01.01");
Funcat.parent(t.up, "01.03.01.01");
Is.leaf("01", t);
GetFuncatDepth("01.03.01.01");
```

`Funcat.transitive.closure` *Function to compute the transitive closure of a given list of FunCat classes*

Description

This function deduces all the FunCat classes of a given set of classes, using the transitive property of the relationships between FunCat classes. It obtains all the ancestors of a given list of FunCat classes. This fuction can by used with lapply to obtain all the Funcat classes from a list of named vectors of specific Funcat classes

Usage

```
Funcat.transitive.closure(x)
```

Arguments

x	a vector of Funcat classes

Value

A vector of Funcat classes obtained by transitive closure from the x Funcat classes

See Also

`Get.Funcat.all.classes`, `Build.Funcat.graph.from.selected.nodes`

Examples

```
Funcat.transitive.closure("12.01.01");
Funcat.transitive.closure(c("12.01.01","42.16"));
# Obtaining the list of all the ancestors of all the classes
# with some annotated genes in the yeast
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
l <- lapply(Yeast.Funcat.specific, Funcat.transitive.closure);
```

`Get.all.common.genes` *Getting genes common to different data sets*

Description

Function that returns the names of genes common to different data sets and a given ontology It may be used with both GO and Funcat ontologies and with any species. An arbitrary number of data sets in matrix format may be passed as arguments (but at least 1 is needed)

Usage

```
Get.all.common.genes(Table.gene.class, data1, ...)
```

Arguments

Table.gene.class	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0
data1	the first data set (mandatory)
...	other possible data sets

Value

a vector of common gene names among the different data sets

See Also

Get.matrix.data.for.classid, Get.matrix.data.without.ancestors, Get.matrix.data.from.parent.only

Examples

```
# Selection of yeast genes with TAS annotation common to the gene expression
# data spYCCES  and to the phylogenetic data obtained from the package scehomology
Yeast.specific.TAS <- Get.yeast.GO.specific.classes(evidence="TAS");
Yeast.general.TAS <- Get.yeast.GO.all.classes(Yeast.specific.TAS);
Yeast.GO.Table.TAS <- Build.GO.class.labels(Yeast.general.TAS);
library(yeastCC);
data.expression <- exprs(spYCCES);
l<-Build.list.yeast.homology.data();
data.phylo<-Do.binary.data.homology(l);
common.genes <- Get.all.common.genes(Yeast.GO.Table.TAS, data.expression, data.phylo);
```

Get.Funcat.all.classes

Function that provides all the FunCat classes for a list of genes

---

Description

Function that provides all the FunCat classes for a list of genes. The classes are obtained by transitive closure.

Usage

```
Get.Funcat.all.classes(NamesToFuncat)
Get.yeast.Funcat.all.classes(NamesToFuncat)
```

Arguments

`NamesToFuncat`	named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding Funcat classes ID.

Details

For each gene and its associated more specific FunCat class annotation, the corresponding ancestor FunCat classes are obtained. More precisely a list whose elements are vectors of FunCat ID associated to each gene is returned. The argument of the function is a list of named vectors, as returned by `Get.Funcat.specific.classes`. From the list of the most specific annotated genes, all the FunCat annotations, by transitive closure, are returned. Note that a dummy "00" class (corresponding to the overall root node) is added to all the annotated genes: in this way we obtain a unique tree form the FunCat forest. `Get.yeast.Funcat.all.classes` is equal to `Get.Funcat.all.classes` and it is maintained only for back-compatibility. Its use is deprecated.

Value

A list : each element of the list corresponds to a named vector (the name corresponds to the name of a gene) and the elements of the vector are the corresponding Funcat classes ID obtained by transitive closure from `NamesToFuncat`.

See Also

`Funcat.transitive.closure`, `Get.Funcat.specific.classes`

Examples

```
# Obtaining all the available FunCat annotations for the genes of A. thaliana
data(ATFUNCAT);
AT.specific <- Get.Funcat.specific.classes (ATFUNCAT, evidence="");
AT.general <- Get.Funcat.all.classes(AT.specific);
```

`Get.Funcat.specific.classes`

Function to get all the most specific FunCat classes for each gene of a given species

---

Description

Function to get all the most specific FunCat classes for each gene of a given species, for a given evidence. It obtains, from the environment mapping gene IDs of a given species to the most specific FunCat annotation, a corresponding list of named vectors. Each elemnent of the list corresponds to a named gene, and the vector to the FunCat IDs of the most specific annotated classes.

Usage

```
Get.Funcat.specific.classes(GeneID2Funcat, evidence = "")
Get.yeast.Funcat.specific.classes(evidence = "")
```

Arguments

GeneID2Funcat	an environment mapping gene IDs to Funcat classes
evidence	evidence code. It may be a vector with 1 or more Funcat evidence code entries entries. if an empty string is provided,all the evidence codes are accepted (default)

Details

The semantic of `Get.yeast.Funcat.specific.classes` is the same of `Get.Funcat.specific.classes`, but restricted to species yeast. For this reason no environment need to be supplied to the function `Get.yeast.Funcat.specific.classes`

Value

A named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding Funcat classes ID

See Also

`Get.Funcat.all.classes`

Examples

```
# Obtaining the most specific FunCat annotations for the genes of A. thaliana
data(ATFUNCAT);
Ath.specific <- Get.Funcat.specific.classes (ATFUNCAT, evidence="");
```

Get.GO.all.classes *Function that provides all the GO classes for a list of genes*

Description

Function that provides all the FunCat classes for a list of genes. The classes are obtained by transitive closure.

Usage

```
Get.GO.all.classes(NamesToGO, ontology="BP")
Get.yeast.GO.all.classes(NamesToGO, ontology="BP")
```

Arguments

NamesToGO	named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding GO classes ID.
ontology	a character vector denoting the ontology : BP (default), MF, CC

Details

For each gene and its associated more specific GO class annotation, the corresponding ancestor GO classes are obtained. More precisely a list whose elements are vectors of GO ID associated to each gene is returned. The argument of the function is a list of named vectors, as returned by `Get.GO.specific.classes`. From the list of the most specific annotated genes, all the GO annotations, by transitive closure, are returned. The function `Get.yeast.GO.all.classes` is equal to `Get.GO.all.classes`, and it is maintained only for back-compatibility. In any case its use is deprecated.

Value

A list : each element of the list corresponds to a named gene and the elements of the vector are the corresponding GO classes ID obtained by transitive closure from `NamesToGO`.

See Also

`GO.transitive.closure`, `Get.GO.specific.classes`

Examples

```
# Obtaining all the available GO annotations for the Cellular
# Component ontology for the genes of Mus musculus
data(MOUSEGO)
Mouse.specific <- Get.GO.specific.classes (ontology="CC", gene2GO=MOUSEGO, evidence="");
Mouse.general <- Get.GO.all.classes(Mouse.specific,ontology="CC");
```

Get.GO.specific.classes

Function to get all the most specific GO classes for each gene

---

Description

This function gets all the most specific classes for each gene, for a given species, GO ontology and evidence. It obtains, from the environment mapping gene IDs of a given species to the most specific GO annotation, a corresponding list of named vectors. Each elemnent of the list corresponds to a named gene, and the vector to the GO IDs of the most specific annotated classes.

Usage

```
Get.GO.specific.classes(ontology = "BP", gene2GO = YEASTGO, evidence = "")
Get.yeast.GO.specific.classes(ontology = "BP", evidence = "")
```

Arguments

`ontology`	the ontology to be selected: it needs to be: "BP" (def.), "MF" or "CC"
`gene2GO`	an environment that maps genes for a given species to GO classes.
`evidence`	evidence code. It may be a vector with 1 or more of the following entries:
	IMP: inferred from mutant phenotype
	IGI: inferred from genetic interaction
	IPI: inferred from physical interaction
	ISS: inferred from sequence similarity
	IDA: inferred from direct assay
	IEP: inferred from expression pattern
	IEA: inferred from electronic annotation
	TAS: traceable author statement
	NAS: non-traceable author statement
	ND: no biological data available
	IC: inferred by curator
	"": no entry: all the evidence codes are accepted (default)

Details

The function `Get.yeast.GO.specific.classes` it is equivalent to `Get.GO.specific.classes`, but restricted to the yeast.

Value

It returns a named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding GO classes ID.

See Also

`Get.GO.all.classes`

Examples

```
# Obtaining the most specific GO annotations for the Cellular
# Component ontology for the genes of Mus musculus
data(MOUSEGO);
Mouse.specific <- Get.GO.specific.classes (ontology="CC", gene2GO=MOUSEGO, evidence="");
```

`Get.matrix.data.for.classid` *Data matrix construction for a binary classification problem: negatives as not positive*

Description

This function builds a data set for a specific class within the FunCat taxonomy or GO. The positive classes are assigned according the specified GO/Funcat class, while the negative examples are all the others not labelled as positive.

Usage

```
Get.matrix.data.for.classid(Table.gene.class, classid, data.matrix,
      ontology = "Funcat", ratio.negative = 0, common.genes = NULL,
    seed = 1, include.unknown = FALSE)
```

Arguments

`Table.gene.class`	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0.
`classid`	GO/FunCat ID of the the class to be extracted from Table.gene.class
`data.matrix`	matrix of the data from which genes of the positive and negative classes are extracted. Rows correspond to genes and columns to the features associated to each gene
`ontology`	a character vector corresponding to the selected ontology: Funcat (default), BP, MF, CC
`ratio.negative`	the proportion of negative examples w.r.t the positives. If 0 (def.) all the available negative examples are used
`common.genes`	character vector: if it set to NULL (default) only the genes common to the given ontology and data set are considered; otherwise only the genes of the common.genes are considered
`seed`	seed for the random generator
`include.unknown`	boolean. If FALSE (def) classes labeled as unknown are not included as possible negative examples

Details

For a gene and its associate data x with multilabel $y \in 0, 1^N$ where N is the number of nodes classes, we have that the positive examples for node i are

$$D(i)^+ = \{x | y_i = 1\}$$

while the negatives are

$$D(i)^- = \{x | y_i = 0\}$$

. The data set given as input must be in matrix format, with genes on the rows and variables associated to genes as columns; the rows of the data matrix needs to be named with the gene names (e.g. ORF ID for yeast or Entrez Gene IDs for human or mouse). Possible data could be, e.g., gene expression or phylogenetic data It may be used with both GO and Funcat ontologies and with any species. The positive classes are assigned according the specified GO/Funcat class, while the negative examples are all the others not labelled as positive. It is possible to select a random subset of the negatives of a specified size. The data are associated to the selected genes and are returned together with the labels of the classes. The function considers only gene names that are common to the ontologies and the available data. N.B. : gene names of the data matrix need to be of the same type of the gene names stored in the rows of `Table.gene.class`. For instance they must be both systematic names of the genes for the yeast or Locus Link identifiers for human gene names. The argument `common.genes` is a character vector with the names of genes common to multiple data sets. If it set to NULL (default) only the genes common to the given ontology and data set are considered; otherwise only the genes of the common.genes are considered: this may be useful only if you want to use multiple data sets and the function `Get.all.common.genes` has been previously called to select the genes common to multiple data sets.

Value

A list with four elements:

`X`	matrix of the data values (rows are genes, columns variables)
`labels`	a factor with the labels of the classes: 1 for classid (positives), 2 for negatives (not belonging to classid).
`n.pos`	number of positive examples
`n.neg`	number of negaive examples

See Also

`Get.matrix.data.from.parent.only`, `Get.matrix.data.without.ancestors`

Examples

```
# building a data set of gene expression data for the yeast
# (considering only TAS annotations) for the class "GO:0000902",
# with a number of negative examples equal to the number of positive ones
Yeast.specific.TAS <- Get.yeast.GO.specific.classes(evidence="TAS");
Yeast.general.TAS <- Get.yeast.GO.all.classes(Yeast.specific.TAS);
Yeast.GO.Table.TAS <- Build.GO.class.labels(Yeast.general.TAS);
library(yeastCC);
data.expression <- exprs(spYCCES);
data.expression.specific.class <- Get.matrix.data.for.classid(Yeast.GO.Table.TAS,
      classid="GO:0000902", data.expression, ontology = "BP", ratio.negative = 1);
# setting a number of negative examples equal to the three times the number of positive ones
data.expression.specific.class.3 <- Get.matrix.data.for.classid(Yeast.GO.Table.TAS,
       classid="GO:0000902", data.expression, ontology = "BP", ratio.negative = 3);
```

`Get.matrix.data.for.two.selected.classes`
Data matrix construction for a binary classification problem: negatives as belonging to a specific class

Description

This function builds a data set for a specific class within the FunCat taxonomy or GO. The positive classes are assigned according the specified GO/Funcat class, while the negative examples belongs to another specific GO/Funcat class.

Usage

```
Get.matrix.data.for.two.selected.classes(Table.gene.class, classid1,
                                         classid2, data.matrix)
```

Arguments

`Table.gene.class`	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0.
`classid1`	GO/FunCat ID of the the positive class to be extracted from Table.gene.class
`classid2`	GO/FunCat ID of the the negative class to be extracted from Table.gene.class
`data.matrix`	matrix of the data from which genes of the positive and negative classes are extracted. Rows correspond to genes and columns to the features associated to each gene

Details

This function returns a dataset for dichotomic classification of GO/Funcat classes of genes where the two classes are two distinct GO/Funcat classes. It may be used with both GO and Funcat ontologies and with any species. N.B. : gene names of the data (rownames) need to be of the same type of the gene names stored in the Table.gene.class. For instance they must be both systematic names of the genes for the yeast or Locus Link identifiers for human gene names.

Value

A list with four elements:

`X`	data frame of the data values (rows are genes, columns variables)
`labels`	a factor with the labels of the classes: 1 for `classid1`, 2 for `classid2`
`n.pos`	number of positive examples (`classid1`)
`n.neg`	number of negaive examples (`classid2`)

See Also

Get.matrix.data.for.classid

Examples

```
# building a data set of gene expression data for the yeast
# (considering only TAS annotations) for the classes "GO:0000902" and "GO:0048015",
Yeast.specific.TAS <- Get.yeast.GO.specific.classes(evidence="TAS");
Yeast.general.TAS <- Get.yeast.GO.all.classes(Yeast.specific.TAS);
Yeast.GO.Table.TAS <- Build.GO.class.labels(Yeast.general.TAS);
library(yeastCC);
data.expression <- exprs(spYCCES);
data.expression.spec.classes <- Get.matrix.data.for.two.selected.classes
 (Yeast.GO.Table.TAS, classid1="GO:0000902", classid2="GO:0048015", data.expression);
```

Get.matrix.data.from.parent.only

Data matrix construction for a binary classification problem: negatives from parents only

---

Description

This function builds a data set for a specific class within the FunCat taxonomy or GO. The positive classes are assigned according the specified GO/Funcat class. A negative example for a node is any example whose multilabel does not include that node and includes at least one of its parents.

Usage

```
Get.matrix.data.from.parent.only(Table.gene.class, classid, data.matrix,
  ontology = "Funcat", ratio.negative = 0, common.genes = NULL, seed = 1)
```

Arguments

Table.gene.class	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0.
classid	GO/funcat ID of the the class to be extracted from Table.gene.class
data.matrix	matrix of the data from which genes of the positive and negative classes are extracted. Rows correspond to genes and columns to the features associated to each gene
ontology	a character vector corresponding to the selected ontology: Funcat (default), BP, MF, CC
ratio.negative	the proportion of negative examples w.r.t the positives. If 0 (def.) all the available negative examples are used

`common.genes`	character vector: if it set to NULL (default) only the genes common to the given ontology and data set are considered; otherwise only the genes of the common.genes are considered
`seed`	seed for the random generator

Details

For a gene and its associate data x with multilabel $y \in 0, 1^N$ where N is the number of nodes classes, with $PAR(i)$ parent nodes of i, we have that the data set $D(i)$ associate to node will be:

$$D(i) = \{x | y_{PAR(i)} = 1\}$$

. The positive examples are

$$D(i)^+ = \{x | y_{PAR(i)} = 1, y_i = 1\}$$

while the negatives are

$$D(i)^- = \{x | y_{PAR(i)} = 1, y_i = 0\}$$

The data set given as input must be in matrix format, with genes on the rows and variables associated to genes as columns; the rows of the data matrix needs to be named with the gene names (e.g. ORF ID for yeast or Entrez Gene IDs for human or mouse). Possible data could be, e.g., gene expression or phylogenetic data. It may be used with both GO and Funcat ontologies and with any species. It is possible to select a random subset of the negatives of a specified size. The data are associated to the selected genes and are returned together with the labels of the classes: 1 for positive examples and 2 for negatives. The function considers only gene names that are common to the ontologies and the available data. N.B. : gene names of the data matrix need to be of the same type of the gene names stored in the `Table.gene.class`. For instance they must be both systematic names of the genes for the yeast or Locus Link identifiers for human gene names.

Value

A list with four elements:

`X`	matrix of the data values (rows are genes, columns variables)
`labels`	a factor with the labels of the classes: 1 for classid (positives), 2 for negatives (not belonging to classid).
`n.pos`	number of positive examples
`n.neg`	number of negaive examples

See Also

`Get.matrix.data.for.classid`, `Get.matrix.data.without.ancestors`

Examples

```
# building a data set of gene expression data for the yeast  for the class "GO:0000902",
# with a number of negative examples equal to the number of positive ones
Yeast.specific <- Get.yeast.GO.specific.classes(evidence="");
Yeast.general <- Get.yeast.GO.all.classes(Yeast.specific);
Yeast.GO.Table <- Build.GO.class.labels(Yeast.general);
library(yeastCC);
data.expression <- exprs(spYCCES);
data.expression.specific.class <-Get.matrix.data.from.parent.only(Yeast.GO.Table,
     classid="GO:0000902", data.expression, ontology = "BP", ratio.negative = 1);
```

Get.matrix.data.without.ancestors

Data matrix construction for a binary classification problem: negatives not from ancestors

---

Description

This function builds a data set for a specific class within the FunCat taxonomy or GO. The positive classes are assigned according the specified GO/Funcat class. A negative example for a node is any example whose multilabel does not include that node and any of its ancestors.

Usage

```
Get.matrix.data.without.ancestors(Table.gene.class, classid, data.matrix,
            ontology = "Funcat", ratio.negative = 0, common.genes = NULL,
seed = 1, include.unknown = FALSE)
```

Arguments

`Table.gene.class`	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0.
`classid`	GO/FunCat ID of the the class to be extracted from Table.gene.class
`data.matrix`	matrix of the data from which genes of the positive and negative classes are extracted. Rows correspond to genes and columns to the features associated to each gene
`ontology`	a character vector corresponding to the selected ontology: Funcat (default), BP, MF, CC
`ratio.negative`	the proportion of negative examples w.r.t the positives. If 0 (def.) all the available negative examples are used

`common.genes` character vector: if it set to NULL (default) only the genes common to the given ontology and data set are considered; otherwise only the genes of the common.genes are considered

`seed` seed for the random generator

`include.unknown` boolean. If FALSE (def) classes labeled as unknown are not included as possible negative examples

Details

For a gene and its associate data x with multilabel $y \in 0, 1^N$ where N is the number of nodes classes, with $PAR(i)$ parent nodes of i, and $ANC(i)$ the ancestors of i, we have that the data set $D(i)^+$ of the positive examples is:

$$D(i)^+ = \{x | y_i = 1\}$$

while the negatives are

$$D(1)^- = \{x | y_{ANC(i)} = 0, y_i = 0\}$$

The data set given as input must be in matrix format, with genes on the rows and variables associated to genes as columns; the rows of the data matrix needs to be named with the gene names (e.g. ORF ID for yeast or Entrez Gene IDs for human or mouse). Possible data could be, e.g., gene expression or phylogenetic data. This function can be used with both GO and Funcat ontologies and with any species. It is possible to select a random subset of the negatives of a specified size. Note the by default examples labeled as unknown (that is genes of classes 98 and 99 with FunCat, and genes with only 1 label with GO ontologies) are not included as possible negative examples, but it is possible to include them as negatives by setting to TRUE the boolean argument `include.unknown`. The data are associated to the selected genes and are returned together with the labels of the classes: 1 for positive examples and 2 for negatives. The function considers only gene names that are common to the ontologies and the available data. N.B. : gene names of the data matrix need to be of the same type of the gene names stored in the `Table.gene.class`. For instance they must be both systematic names of the genes for the yeast or Locus Link identifiers for human gene names.

Value

A list with four elements:

`X` matrix of the data values (rows are genes, columns variables)

`labels` a factor with the labels of the classes: 1 for classid (positives), 2 for negatives (not belonging to classid).

`n.pos` number of positive examples

`n.neg` number of negaive examples

See Also

`Get.matrix.data.from.parent.only`, `Get.matrix.data.for.classid`

Examples

```
# building a data set of gene expression data for the yeast  for the class "GO:0000902",
Yeast.specific <- Get.yeast.GO.specific.classes(evidence="");
Yeast.general <- Get.yeast.GO.all.classes(Yeast.specific);
Yeast.GO.Table <- Build.GO.class.labels(Yeast.general);
library(yeastCC);
data.expression <- exprs(spYCCES);
data.expression.specific.class2 <-Get.matrix.data.without.ancestors(Yeast.GO.Table,
                          classid="GO:0000902", data.expression, ontology = "BP");
```

Get.matrix.positive.data.for.classid

Data matrix construction for a binary classification problem: positives only

---

Description

This function builds a data set for a specific class within the FunCat taxonomy or GO using positive examples only, that is only examples beloning to the selected functional class. No negative examples are added.

Usage

```
Get.matrix.positive.data.for.classid(Table.gene.class, classid, data.matrix)
```

Arguments

`Table.gene.class`	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0.
`classid`	GO/FunCat ID of the the class to be extracted from Table.gene.class
`data.matrix`	matrix of the data from which genes of selected class are extracted. Rows correspond to genes and columns to the features associated to each gene

Value

a matrix of the data values (rows are genes, columns variables)

See Also

`Get.matrix.data.for.classid`, `Get.matrix.data.from.parent.only`, `Get.matrix.data.without.ancestors`

Examples

```
# building a data set of gene expression data for the yeast
# (considering only TAS annotations) for the class "GO:0000902"
Yeast.specific.TAS <- Get.yeast.GO.specific.classes(evidence="TAS");
Yeast.general.TAS <- Get.yeast.GO.all.classes(Yeast.specific.TAS);
Yeast.GO.Table.TAS <- Build.GO.class.labels(Yeast.general.TAS);
library(yeastCC);
data.expression <- exprs(spYCCES);
data.expression.specific.class <- Get.matrix.positive.data.for.classid(Yeast.GO.Table.TAS,
                                                    classid="GO:0000902", data.expression);
```

GO.transitive.closure

Function to compute the transitive closure of a given list of GO classes

---

Description

This function deduces all the GO classes of a given set of classes, using the transitive property of the relationships between GO classes. It obtains all the ancestors of a given list of GO classes, using the environments GOXXANCESTOR of the GO package. This fuction should by used with lapply to obtain all the GO classes from a list of named vectors of specific GO classes

Usage

```
GO.transitive.closure(x, ontology="BP")
```

Arguments

x	a vector of GO classes
ontology	a character vector denoting the ontology : BP (default), MF, CC

Value

A vector of GO classes obtained by transitive closure of the x GO classes

Note

If in the argument x are supplied non existing GO IDs, then the GOID of the root of the selected ontology is returned. This could be useful if you supply new GO IDs that are no registered in the current version of the GO library.

See Also

Build.GO.graph.from.selected.nodes, Get.GO.all.classes

Examples

```
GO.transitive.closure("GO:0000723")
GO.transitive.closure(c("GO:0000723","GO:0005977","GO:0030437"));
```

Graphics.cardinality.labels

Graphic functions to represent the distribution of the cardinality of gene labels

---

Description

These functions plot the distribution of the cardinality of the labels associated to each gene. It may be used with both GO and Funcat ontologies. The functions plot the empirical cumulative distribution (`Do.ecdf.cardinality.labels`), and the histograms of the cardinality of the labels associated to each gene (`Do.table.hist.cardinality.labels`).

Usage

```
Do.table.hist.cardinality.labels(NamesToGO, title = "")
Do.ecdf.cardinality.labels(NamesToGO, title = "")
```

Arguments

`NamesToGO`	named list of vectors. Each element of the list corresponds to a named gene and the elements of the vector are the corresponding functional classes ID.
`title`	character vector for the title of the ecdf

Value

summary table of the frequencies of the cardinality of the labels per gene is returned by the function `Do.table.hist.cardinality.labels`

Examples

```
# Plot of the histogram and ecdf of the cardinality of gene labels in yeast
Yeast.specific <- Get.yeast.GO.specific.classes();
Do.table.hist.cardinality.labels (Yeast.specific,
title="Distribution of the numbers of labels per gene (more specific classes only) in S.cerevisiae");
x11();Do.ecdf.cardinality.labels(Yeast.specific);
```

HUMANGO	*Human Entrez Gene ID identifiers to Gene Ontology (GO) mapping*

Description

This environment provides a mapping between human genes and GO terms.

Usage

```
data(HUMANGO)
```

Format

An environment (hash table) with 17281 entries

Details

The environment provides GO annotations for human genes. EntrezGene identifiers are Keys and GO terms Values. More precisely each Value is a list of lists. Each element of the outer list represent a mapping to a GO term. The elements of the inner list are:

GOID : GO Identifier

Evidence : Evidence code

Ontology : "BP", "MF" or "CC"

Source

The environment is built using the Bioconductor environments `humanLLMappingsLL2GO` and `GOENTREZID2GO`

Examples

```
data(HUMANGO);
get("23",HUMANGO);
```

Map.FID2Term *Environments mapping Funcat class ID to the corresponding Funcat functional term and viceversa*

Description

The environments realize a mapping between Funcat class ID and the description of the corresponding Funcat functional terms

Usage

```
data(FID2TERM)
data(TERM2FID)
```

Format

Environments (hash table) mapping Funcat ID (character vector) to their description (character vector) and viceverca

Note

A dummy root node with Funcat ID 00 is added. The node 40.02.03 (child of 40.02) is present, while its parent 40.02 not; to avoid inconsistencies a new entry 40.02 has been added to the funcat-2.1_scheme

Source

These environments are based on scheme 2.1 of FunCat (http://mips.gsf.de/projects/funcat).

Examples

```
data(FID2TERM);
date(TERM2FID);
```

Map.ORF *Yeast ORF mappings*

Description

Environments mapping yeast ORFs to RefSeq and to HomoloGene IDs and viceversa.

Usage

```
data(ORF2HGID)
data(HGID2ORF)
data(ORF2RefSeq)
data(RefSeq2ORF)
```

Format

`ORF2HGID` is an environment (hash table) with 4739 entries `HGID2ORF` is an environment (hash table) with 4739 entries `ORF2RefSeq` is an environment (hash table) with 5877 entries `RefSeq2ORF` is an environment (hash table) with 5873 entries

Details

`ORF2HGID` and `HGID2ORF` realize a mapping between yeast ORF ID and Homologene ID and viceversa. `ORF2RefSeq` and `RefSeq2ORF` realize a mapping between yeast ORF ID to Refseq ID and viceversa.

Source

Data for `ORF2HGID` and `HGID2ORF` have been obtained from the *ensembl* web site (`http://www.ensembl.org`) and from the Bioconductor package **scehomology**. Data for `ORF2RefSeq` and `RefSeq2ORF` have been obtained from the *ensembl* web site (`http://www.ensembl.org`).

Examples

```
data(ORF2HGID);
get("YPR104C",ORF2HGID);
data(RefSeq2ORF);
get("NP_878148", RefSeq2ORF);
```

MOUSEFUNCAT *Mouse Mfun MIPS ID identifiers to FunCat mapping*

Description

This environment provides a mapping between *Mus musculus* genes and FunCat classes

Usage

```
data(MOUSEFUNCAT)
```

Format

An environment (hash table) with 18850 entries

Details

The environment provides FunCat annotations for mouse genes (http://mips.gsf.de/projects/funcat). Mfun identifiers are Keys and GO FunCat Values. More precisely each Value is a list of lists. Each element of the outer list represent a mapping to a GO term. The elements of the inner list are:

FuncatID : FunCat Identifier

Evidence : Evidence code: either "automatic" or "manually reviewed"

Mgi : Mgi identifier

Gene.Name : Name of the gene

Prot.Descr : Description of the gene product

Source

It is conformed to the Funcat funcat-2.1 scheme with the MfungdAnnotation.xml annotation file available from MIPS: ftp://ftpmips.gsf.de/MfunGD/MfungdAnnotation.xml.gz.

Examples

```
data(MOUSEFUNCAT)
mget(c("mcx001628","mcx001636"), MOUSEFUNCAT)
```

`MOUSEGO` *Mouse Entrez Gene ID identifiers to Gene Ontology (GO) mapping*

Description

This environment provides a mapping between Mus musculus genes and GO terms.

Usage

```
data(MOUSEGO)
```

Format

An environment (hash table) with 19354 entries

Details

The environment provides GO annotations for mouse genes. EntrezGene identifiers are Keys and GO terms Values. More precisely each Value is a list of lists. Each element of the outer list represent a mapping to a GO term. The elements of the inner list are:

GOID :GO Identifier

Evidence : Evidence code

Ontology : "BP", "MF" or "CC"

Source

The environment is built using the Bioconductor environments `mouseLLMappingsLL2GO` and `GOENTREZID2GO`.

Examples

```
data(MOUSEGO)
get("53970",MOUSEGO);
```

Plot.distribution.gene.per.class

Plot of the distribution of the number of examples per class

Description

The function plots the number of classes with more than a given number of examples. It may be used with both GO and Funcat ontologies.

Usage

```
Plot.distribution.gene.per.class(counts, card = 5:6000)
```

Arguments

`counts`	vector with the number of examples for each class
`card`	vector with the considered number of examples per class

Details

Abscissa represents the number of examples; the ordinate the number of classes with equal or more examples than the corresponding abscissa.

Value

A graph representing the number of classes with equal or more examples than a given value represented in abscissa. Abscissa is represented in logarithmic scale.

See Also

Plot.histogram.gene.per.class

Examples

```
Yeast.specific <- Get.yeast.GO.specific.classes();
Yeast.specific.classes <- Build.GO.class.labels(Yeast.specific);
l <- Count.examples.per.class(Yeast.specific.classes);
Plot.distribution.gene.per.class(l[[1]], card=1:5813)
```

Plot.hist.Funcat.depth.labels

Histogram of the depth of FunCat classes

Description

Graphic function to plot the histograms of the depth of FunCat clasees

Usage

```
Plot.hist.Funcat.depth.labels(nodes)
```

Arguments

nodes	a list(vector) of FunCat nodes

See Also

Plot.histogram.gene.per.class

Examples

Plot.histogram.gene.per.class

Plot of the histograms of the number of genes per class

Description

Function to plot the histogram of the number of genes per functional class. It may be used with both GO and Funcat ontologies.

Usage

```
Plot.histogram.gene.per.class(Table.gene.class, title = "")
```

Arguments

`Table.gene.class`	a data frame with columns corresponding to Funcat or GO classes and rows to a given gene. If a gene belongs to the Funcat/GO class the corresponding column is set to 1 otherwise to 0.
`title`	title of the plot

Value

The histogram of the number of genes per functional class is plot.

See Also

`Plot.distribution.gene.per.class`

Examples

```
# plot of the histogram of the number of genes per class in the yeast,
# considering only the most specific annotations
Yeast.specific <- Get.yeast.GO.specific.classes();
Yeast.specific.classes <- Build.GO.class.labels(Yeast.specific);
Plot.histogram.gene.per.class(Yeast.specific.classes);
```

`Plot.ontology.graph` *Plotting graphs of the GO and FunCat*

Description

Functions to plot in compact way complex graphs of the GO or FunCat

Usage

```
Plot.ontology.graph(g, node.label = "x", fcolor = "black")
Pretty.plot.graph(g, fontsize = 12, fillcolor = "lightgreen", height = 0.6,
                         width = 0.9, color = "black", fontcolor = "black")
```

Arguments

g	a GO or FunCat graph
node.label	a label plotted for each node
fcolor	color of the node label
fontsize	size of the fonts
fillcolor	color to fill the nodes
height	height of the nodes
width	width of the nodes
color	color of the edges
fontcolor	color of the font

Details

The functions use the plotting graph facilities provided by the package **Rgraphviz**. `Plot.ontology.graph` plot in compact way complex graphs. `Pretty.plot.graph` to "pretty" plot not too complex graphs.

Examples

```
# Plotting Yeast FunCat classes with more than 100 examples and with TAS evidence:
Yeast.specific.TAS <- Get.yeast.GO.specific.classes(evidence="TAS");
Yeast.general.TAS <- Get.yeast.GO.all.classes(Yeast.specific.TAS);
Yeast.general.classes.TAS <- Build.GO.class.labels(Yeast.general.TAS);
GO.classes.TAS.100 <-
   Select.functional.classes.by.cardinality(Yeast.general.classes.TAS, 100);
gBP.universal.ontology <- Build.universal.graph.ontology.down();
gYeast.TAS.card.100 <- subGraph(GO.classes.TAS.100, gBP.universal.ontology);
Pretty.plot.graph(gYeast.TAS.card.100,fontsize=12,fillcolor="lightgreen",
                  height=0.9,width=1.2,color="black", fontcolor="black");
```

`Select.Funcat.classes.by.depth`

Selection of nodes at a given depth in the FunCat trees

Description

It selects a set of nodes at a given distance from the root of the Funcat tree In particular it selects all the nodes at a distance equal or less or equal than a given depth.

Usage

```
Select.Funcat.classes.by.depth(g, distance = 1, only.equal = FALSE)
```

Arguments

g	a graph of the FunCat tree
distance	distance from the root: all the nodes at a distance equal or less or equal (default) than distance are chosen
only.equal	if TRUE only the nodes at a given distance are selected, otherwise all the nodes at a distance equal or less (default).

Value

a vector of the Funcat ID of the selected nodes

See Also

Select.functional.classes.by.cardinality

Examples

```
# selection at the nodes up to 2 depth in the universal FunCat tree:
gUniversalFuncat <- Do.universal.tree.Funcat();
nodes<-Select.Funcat.classes.by.depth(gUniversalFuncat,2);
```

Select.functional.classes.by.cardinality

Selection of functional classes on the basis of their cardinality

---

Description

The function selects functional classes (GO or FunCat classes) on the basis of their cardinality. By this function we can select only the classes with a number of positive examples larger than a given quantity.

Usage

```
Select.functional.classes.by.cardinality(Table.gene.class, min.cardinality = 20)
```

Arguments

Table.gene.class	a data frame with variables corresponding to GO/Funcat classes and rows to a given gene. If a gene belongs to the GO/Funcat class the corresponding variable is set to 1 otherwise to 0
min.cardinality	a GO/Funcat class is selected if its cardinality is equal or larger than min.cardinality

Value

a character vector with the GO/FunCat ID of the selected classes

See Also

Select.GO.classes.by.distance, Select.Funcat.classes.by.depth

Examples

```
Yeast.Funcat.specific <- Get.yeast.Funcat.specific.classes();
Yeast.Funcat.general <- Get.yeast.Funcat.all.classes(Yeast.Funcat.specific);
Yeast.Funcat.Table <- Build.Funcat.Table.labels(Yeast.Funcat.general);
# Selection of yeast classes with more than 50 genes
nodes <- Select.functional.classes.by.cardinality(Yeast.Funcat.Table, 50);
```

Select.GO.classes.by.distance

Selection of nodes at a given distance from the root of a GO ontology

---

Description

It selects a set of GO nodes at a given distance from the root of a given ontology In particular it selects all the nodes at a distance less or equal than the argument distance. The distance is computed according to the shortest path from the root: all the nodes at a distance less or equal than a given distance are selected.

Usage

```
Select.GO.classes.by.distance(g, distance = 1, ontology = "BP")
```

Arguments

g	a GO graph
distance	distance (shortest path) from the root: all the nodes at a distance less or equal than distance are chosen
ontology	BP (default), MF, CC

Value

a vector of the GOID of the selected nodes

See Also

Select.Funcat.classes.by.depth, Select.functional.classes.by.cardinality

Examples

```
# Selection of the nodes at distance equal or less than 3
# in the GO BP ontology of the yeast.
gBP.universal.ontology2 <- Build.universal.graph.ontology.down();
Yeast.specific <- Get.yeast.GO.specific.classes();
Yeast.general <- Get.yeast.GO.all.classes(Yeast.specific);
BP.Yeast.classes <- Get.classes(Yeast.general);
gYeast.BP.2 <- subGraph(BP.Yeast.classes, gBP.universal.ontology2);
nodes<-Select.GO.classes.by.distance(gYeast.BP.2,3);
```

Select.GO.rooted.classes

Selection of nodes rooted on a set of given nodes in a given GO graph

---

Description

It selects from a GO graph nodes rooted in a set of given nodes From a given GO graph a subset of nodes rooted on a set of given nodes is selected. Only nodes that belong to the graph are selected: nodes rooted on the given nodes in th ontology but that do not belong to the given Go graph are discarded.

Usage

```
Select.GO.rooted.classes(g, root.nodes, ontology="BP")
```

Arguments

g	a GO graph
root.nodes	character vector of the root nodes
ontology	BP (default), MF, CC

Value

a vector of the GOID of the selected nodes

See Also

Select.GO.classes.by.distance, Select.functional.classes.by.cardinality, Subtree.nodes

Examples

```
# Selection of the nodes rooted in the node GO:0006605 (protein targeting)
g <- Build.universal.graph.ontology.down();
Select.GO.rooted.classes(g, "GO:0006605", ontology="BP")
```

`Select.ontology` *Selection of functional classes associated to genes*

Description

These functions select GO or FunCat classes associated to gene names according to a given ontology and evidence

Usage

```
Select.ontology(x, ontology = "BP")
Select.ontology.evidence(x, ontology = "BP", evidence = "")
Select.Funcat.evidence(x, evidence = "")
```

Arguments

`x`	a generic entry of an environment representing an assay or platform specific annotation
`ontology`	the GO ontology to be selected: it needs to be: "BP" (def.), "MF" or "CC"
`evidence`	evidence code. For FunCat (`Select.Funcat.evidence`) it can be a vector with one ore more FunCat numeric evidence codes. For GO it may be a vector with 1 or more of the following entries: IMP: inferred from mutant phenotype IGI: inferred from genetic interaction IPI: inferred from physical interaction ISS: inferred from sequence similarity IDA: inferred from direct assay IEP: inferred from expression pattern IEA: inferred from electronic annotation TAS: traceable author statement NAS: non-traceable author statement ND: no biological data available IC: inferred by curator "": no entry: all the evidence codes are accepted (default)

Details

`Select.ontology` selects a GO class, while `Select.ontology.evidence` selects genes with respect to an ontology and evidence code of the annotations. It can be used with eapply, to extract from the environment that associates a gene/gene product to its gene ontology classes (e.g. YEASTGO) the entries that match a specific ontology and evidence. `Select.Funcat.evidence` select a FunCat class with respect to the evidence code of the

annotation. It can be used with eapply, to extract from the environment that associates a gene/gene product to its FunCat classes (e.g. YEASTFUNCAT) the entries that match a specific ontology and evidence.

Value

`Select.ontology` returns the GOID of the GO class if it matches the ontology, otherwise the string "NOCLASS". `Select.ontology.evidence` returns a vector with the GO ID of the ontology classes associated to the entry x, for the selected ontology and evidence codes. Duplicated classes are removed. `Select.Funcat.evidence` returns a vector with the Funcat ID of the classes associated to the entry x, for the selected evidence codes. Duplicated classes are removed.

See Also

`Get.all.GO.classes`, `Get.GO.specific.classes`

Examples

```
# Selection of genes annotated with TAS evidence in the MF GO ontology in A. thaliana;
 data(ATGO);
 A.th.names.to.GO <- eapply(ATGO, Select.ontology.evidence,
                            ontology="MF", evidence="TAS");
```

Subtree.nodes *Selection of subtree nodes in FunCat trees*

Description

It returns the nodes of the subtree(s) associated to given node(s) in FunCat trees.

Usage

```
Subtree.nodes(g, nodes)
```

Arguments

g	a graph representing the tree
nodes	a set of nodes from which the subtrees are obtained

Value

a vector of the nodes belonging to the subtree

See Also

`Select.Funcat.classes.by.depth`, `Select.functional.classes.by.cardinality`

Examples

```
# Selection of the nodes of the FunCat subtree rooted in the node "01" (Metabolism)
gUniversalFuncat <- Do.universal.tree.Funcat();
s01.nodes <- Subtree.nodes(gUniversalFuncat, "01");
```

Write.gene.classes.associations

Writing to a file gene - functional class relationships

Description

Function to write to a file the associations gene –> ontology classes. It may be used with both GO and Funcat ontologies.

Usage

```
Write.gene.classes.associations(NamesToGO, filename)
```

Arguments

NamesToGO	named list of vectors. Each element of the list corresponds to a named yeast gene and the elements of the vector are the corresponding functional classes classes ID.
filename	name of the file where the gene–>GO/Funcat associations are written.

Value

a file where the gene - GO/Funcat associations are written. Each row represents a gene (entrez gene ID) and a list of associated GO classes (GO/Funcat ID), separated by blanks.

See Also

Get.GO.specific.classes, Get.Funcat.specific.classes, Get.GO.all.classes, Get.Funcat.all.classes

Examples

YEASTFUNCAT *Yeast ORF identifiers to FunCat mapping*

Description

This environment provides a mapping between *S. cerevisiae* genes and FunCat classes

Usage

```
data(YEASTFUNCAT)
```

Format

An environment (hash table) with 6167 entries

Details

Each entry associates the ORF ID to the the list of the Funcat ID and the associated evidence code:

FuncatID : FunCat Identifier

Evidence : FunCat evidence code

Source

It is conformed to the Funcat funcat-2.1 scheme with the funcat-2.1_data_20070316, available from: ftp://ftpmips.gsf.de/yeast/catalogues/funcat/funcat-2.1_data_20070316

Examples

```
data(YEASTFUNCAT)
get("YGR036C",YEASTFUNCAT)
```