ToolsForHomalg

Special methods and knowledge propagation tools

2022.12-01

6 December 2022

Mohamed Barakat

Sebastian Gutsche

Markus Lange-Hegermann

Mohamed Barakat
Email: mohamed.barakat@uni-siegen.de
Homepage: https://mohamed-barakat.github.io
Address: Walter-Flex-Str. 3
57072 Siegen
Germany

Sebastian Gutsche
Email: gutsche@mathematik.uni-siegen.de
Homepage: https://sebasguts.github.io
Address: Department Mathematik
Universität Siegen
Walter-Flex-Straße 3
57072 Siegen
Germany

Markus Lange-Hegermann
Email: markus.lange.hegermann@rwth-aachen.de
Homepage: http://wwwb.math.rwth-aachen.de/~markus/
Address: Markus Lange-Hegermann
Lehrstuhl B fuer Mathematik, RWTH Aachen
Templergraben 64
52062 Aachen
Germany

Contents

1 Caches **4**
- 1.1 Object constructors . . . 4
- 1.2 Setters, getters . . . 4
- 1.3 Managing functions . . . 5
- 1.4 Install functions . . . 5

2 Lazy arrays **7**
- 2.1 GAP categories . . . 7
- 2.2 Constructors . . . 7

3 Lazy homogeneous lists **8**
- 3.1 GAP categories . . . 8
- 3.2 Constructors . . . 8

4 Lists with attributes **9**
- 4.1 GAP categories . . . 9
- 4.2 Constructors . . . 9

5 ToDo-list **10**
- 5.1 Blueprints . . . 10
- 5.2 ToDo-list entries . . . 12
- 5.3 Category . . . 14
- 5.4 Constructor . . . 14
- 5.5 This is the magic . . . 14
- 5.6 Methods for all objects . . . 15
- 5.7 Proof tracking . . . 15
- 5.8 Maintainance . . . 15

6 Basics **17**
- 6.1 Global variables . . . 17
- 6.2 GAP Categories . . . 17

7 Pointers **19**
- 7.1 Weak pointer objects . . . 19
- 7.2 Pointer objects . . . 20

8 Tools **21**
8.1 Functions 21
8.2 Example functions 28

9 Trees **29**
9.1 Trees 29

Index **31**

Chapter 1

Caches

1.1 Object constructors

Caches are objects which store for a fixed number of keys a value, so they are a map Obj^k -> Obj, while the k is fixed. A cache ususally stores the result in a weak pointer list, which means that if the value which the cache should store is not referenced in the system anymore, it will not be remembered by the cache. However, caches can be set to store the value permanently (crisp), or not to store any new value at all (inaktive). In that case, already stored values are still in the cache and can be accessed once the cache is set active again.

1.1.1 CachingObject

▷ `CachingObject(`*`[k][,] [is_crisp]`*`)` (operation)
▷ `CachingObject(`*`arg`*`)` (operation)
▷ `CachingObject(`*`arg1, arg2`*`)` (operation)

Returns: a cache

If no argument is given, the function returns a weak cache with key length one, if an integer k is given, a weak cache with key length k, and if the bool is_crisp is true, a crisp cache with the corresponding length.

1.1.2 CachingObject (for IsObject, IsObject, IsInt)

▷ `CachingObject(`*`object, cache_name, length[, is_crisp]`*`)` (operation)
▷ `CachingObject(`*`arg1, arg2, arg3, arg4`*`)` (operation)

This methods are not installed, they serve as an interface for InstallMethodWithCacheFromObject.

1.2 Setters, getters

1.2.1 CacheValue (for IsCachingObject, IsObject)

▷ `CacheValue(`*`cache, key`*`)` (operation)

Returns: stored value

If there is a value stored in the cache for key, which can be a single key for caches with key length one or a list of keys depending on the key length of the cache, this method returns a list only contraining the value, otherwise an empty list.

1.2.2 SetCacheValue (for IsCachingObject, IsObject, IsObject)

▷ `SetCacheValue(cache, key, value)` (operation)

Sets the value of key of the cache to value.

1.2.3 IsEqualForCache (for IsObject, IsObject)

▷ `IsEqualForCache(obj1, obj2)` (operation)

Returns: true or false

This function is used to compare objects for the caches. The standard way is IsIdenticalObj, and lists are compared recursive with this function. It is possible and recommended to overload this function as needed.

1.3 Managing functions

1.3.1 SetCachingObjectCrisp

▷ `SetCachingObjectCrisp(cache)` (function)

Returns: nothing

Sets the caching to crisp, weak, or deativates the cache completely.

1.3.2 SetCachingObjectWeak

▷ `SetCachingObjectWeak(arg)` (function)

1.3.3 DeactivateCachingObject

▷ `DeactivateCachingObject(arg)` (function)

1.4 Install functions

1.4.1 InstallMethodWithCache

▷ `InstallMethodWithCache(Like, InstallMethod)` (function)

Installs a method like InstallMethod, but additionally puts a cache layer around it so that the result is cached. It is possible to give the cache as the option Cache, to use the same cache for more than one method or store it somewhere to have access to the cache.

1.4.2 InstallMethodWithCrispCache

▷ `InstallMethodWithCrispCache(`*arg*`)` (function)

Like InstallMethodWithCache, but with a crisp cache.

1.4.3 InstallMethodWithCacheFromObject

▷ `InstallMethodWithCacheFromObject(`*Like, InstallMethod*`)` (function)

This works just like InstallMethodWithCache, but it extracts the cache via the CachingObject method from one of its arguments. The CachingObject must then be implemented for one of the arguments, and the option ArgumentNumber can specify which option to be used. As second argument for CachingObject a string is used, which can identify the cache. Standard is the name of the operation, for which the method is installed, but it can be specified using the CacheName option.

1.4.4 FunctionWithCache

▷ `FunctionWithCache(`*func*`)` (function)

Returns: a function

Creates a cached function out of a given function *func*. If the option Cache is a cache, this cache is used. If the option Cache is the string crisp, a crisp cache is used. All other values for this option lead to a single weak cache.

Chapter 2

Lazy arrays

2.1 GAP categories

2.1.1 IsLazyArray (for IsComponentObjectRep and IsList)

▷ `IsLazyArray(L)` (filter)

Returns: `true` or `false`

The GAP category of lazy arrays.

2.2 Constructors

2.2.1 LazyArrayWithValues

▷ `LazyArrayWithValues(n, func, values, type)` (function)
▷ `LazyArray(n, func)` (function)
▷ `LazyStandardInterval(length)` (function)
▷ `LazyInterval(length, start)` (function)
▷ `LazyConstantArray(n, number)` (function)
▷ `LazyArrayFromList(list)` (function)

Construct a lazy array out of the nonnegative integer *n* and the function *func* defined in the range [0 .. *n*], and possibly empty (sparse) list *values* of a posteriori possibly known values.

Chapter 3

Lazy homogeneous lists

3.1 GAP categories

3.1.1 IsLazyHList (for IsComponentObjectRep and IsList)

▷ `IsLazyHList(L)` (filter)

Returns: `true` or `false`

The GAP category of lazy homogeneous lists.

3.2 Constructors

3.2.1 LazyHList

▷ `LazyHList(L, func)` (function)

Construct a lazy list

Chapter 4

Lists with attributes

These are homogeneous lists which still carry enough information of their context even if they are empty.

4.1 GAP categories

4.1.1 IsListWithAttributes (for IsAttributeStoringRep and IsList)

▷ `IsListWithAttributes(`*L*`)` (filter)

Returns: `true` or `false`

The GAP category of lists with attributes.

4.2 Constructors

4.2.1 TypedListWithAttributes

▷ `TypedListWithAttributes(`*L*`, `*type*`, `*attr1*`, `*value1*`, `*attr2*`, `*value2*`, ...)` (function)

Construct a list with attributes.

4.2.2 ListWithAttributes

▷ `ListWithAttributes(`*L*`, `*attr1*`, `*value1*`, `*attr2*`, `*value2*`, ...)` (function)

Construct a list with attributes of type TheTypeListWithAttributesRep

Chapter 5

ToDo-list

5.1 Blueprints

5.1.1 ToDoListEntryToMaintainEqualAttributes (for IsList, IsList, IsList)

▷ `ToDoListEntryToMaintainEqualAttributes(`*`indicator, objects, attributes`*`)` (operation)

Returns: a todo list entry

The first argument is the *`indicator`*. It is a list of sources like in ToDoListEntry. Each entry *`SP`* has to be a threetuple. First entry of *`SP`* has to an object, for which the second entry of *`SP`*, which has to be the name of an attribute, must become known. Once the attribute is known to the object, it will be compared to the third entry of the list. This can be a value, which is compared directly a function, which is launched and its return value is compared, or a list, consisting of a function and arguments, so the return value of the function with given arguments is compared. If there is no third entry in *`SP`*, it is only looked up if the value is known. Once all entries in *`indicator`* are processed like this, and all returned true in the comparasion, a list of ToDoListEntryForEqualAttributes is installed. They are installed for the two entries of the list *`objects`* which can either be the objects itself or a list containing a function and arguments, which return value is used. For each entry in *`attributes`* such an entry is installed. Such an entry can be the name of an attribute, if both objects in *`objects`* should share the value between attributes with the same name, or a list of two names, if the attributes do not have the same name.

5.1.2 ToDoListEntryToMaintainFollowingAttributes (for IsList, IsList, IsList)

▷ `ToDoListEntryToMaintainFollowingAttributes(`*`indicator, objects, attributes`*`)` (operation)

Returns: a todo list entry

This function creates a ToDoListEntry which can install several ToDoListEntries. The first two arguments, *`indicator`* and *`objects`* except that there will be only ToDoListEntries installed between the two objects in *`objects`*. Each entry in *`attributes`* can either be a string which means that the attribute with the given name will be set from the first to the second object in *`objects`* once it is known. The third argument *`attributes`* is a list of attributes that will be propagated by ToDoListEntries. Each entry *`TP`* can either be a list consisting of a DescriptionOfImplication string and one of the following or just one of the following lists: It can be a string, which means that the Attribute with the given name will be propagated from the first to the second object. It can be a list, consisting of

two entries, where the first entry is a list of sources like in ToDoListEntry and the second might be a function which will be launched once the first part is fulfilled. It can also be a threetuple which will serve as second to fourth argument of ToDoListEntry. Or it can be a string, which will set the attribute named like this of the first object to the one named in the second object

5.1.3 ToDoListEntry (for IsList, IsList)

▷ `ToDoListEntry(`*`source, target_list`*`)` (operation)

Returns: a todo list entry

This function allows to create more than one ToDoListEntry with identical list of sources at one time. First argument is a list of sources like in the other ToDoListEntry functions Second argument is a list of threetuples, which serve as second to fourth argument of ToDoListEntry or a function, which serves as second argument for ToDoListEntry or a tuple with a description string and one of the above.

5.1.4 ToDoList_this_object

▷ `ToDoList_this_object` (global variable)

Represents the objects for which the blueprint is created in the arguments

5.1.5 ToDoListEntryToMaintainEqualAttributesBlueprint (for IsObject, IsList, IsList, IsList)

▷ `ToDoListEntryToMaintainEqualAttributesBlueprint(`*`filter, indicator, objects, attributes`*`)` (operation)

Returns: nothing

This function installs an immediate method which can install ToDoListEntryToMaintainEqualAttributes. First argument must be a filter, and once the filter becomes true the ToDoListEntryToMaintainEqualAttributes is installed with the second to fourth argument as first to third. In those attributes, at any point, the variable ToDoList_this_object can be used. When the entry is installed This will be replaced with the object to which the filters became known, i.e. the one which triggered the immediate method.

5.1.6 ToDoListEntryToMaintainFollowingAttributesBlueprint (for IsObject, IsList, IsList, IsList)

▷ `ToDoListEntryToMaintainFollowingAttributesBlueprint(`*`arg1, arg2, arg3, arg4`*`)` (operation)

Returns: nothing

The same as ToDoListEntryToMaintainEqualAttributesBlueprint for ToDoListEntryToMaintainFollowingAttributes

5.1.7 ToDoListEntryBlueprint (for IsObject, IsList, IsList)

▷ `ToDoListEntryBlueprint(`*`arg1, arg2, arg3`*`)` (operation)

Returns: nothing

The same as ToDoListEntryToMaintainEqualAttributesBlueprint for ToDoListEntry

5.2 ToDo-list entries

5.2.1 AddToToDoList (for IsToDoListEntry)

▷ `AddToToDoList(E)` (operation)

Adds the ToDo-list entry *E* to the ToDo-lists of it's source objects and creates a new one, if this is needed. This function might be called with lists of entries

5.2.2 SourcePart (for IsToDoListEntry)

▷ `SourcePart(entry)` (operation)

Returns: a list

Returns the a list of source parts of the ToDo-list entry *entry*. This is a triple of an object, a name of a filter/attribute, and a value to which the attribute has to be set to activate the entry

5.2.3 TargetPart (for IsToDoListEntry)

▷ `TargetPart(entry)` (operation)

Returns: a list

Returns the target part of the ToDo-list entry *entry*. This is a triple of an object, a name of a filter/attribute, and a value to which the specific filter/attribute should be set. The third entry of the list might also be a function to which return value the attribute is set.

5.2.4 ProcessAToDoListEntry (for IsToDoListEntry)

▷ `ProcessAToDoListEntry(arg)` (operation)

Returns: a boolean

Processes a ToDo-list entry, i.e. sets the information given in TargetPart if the definitions in SourcePart are fulfilled. Returns a function if the entry could be processed, false if not, and fail if SourcePart or TargetPart weren't availible anymore.

5.2.5 ToDoListEntry (for IsList, IsObject, IsString, IsObject)

▷ `ToDoListEntry(arg1, arg2, arg3, arg4)` (operation)

Returns: a ToDoListEntry

The first argument must be a list consisting of two, three or four-tuples where the first entry must be the object to which the attribute given as a string in the second entry must be known to process this entry. The second entry can also be a list of strings, in that case all the attributes given as names must be known. Also, in this case, only two entries in this tuple are allowed. The third part can be a value or a list, consisting of a function followed by arguments which will be computed by the time the attribute given as second entry becomes known to the first entry. If the second part is only a string, and there is a third entry in the tuple the attribute is compared to the third entry. One can set a comparating function as fourth entry, which must take two entries and return false or true. If the value of the attribute matches the (computed) value in the third entry for all members of the list in the first argument the attribute given as third argument, also by name, of the second argument is set to the value of the fourth argument. This can also be a list which has to be computed, or a function, which retun value is used in this case.

5.2.6 ToDoListEntry (for IsList, IsFunction)

▷ `ToDoListEntry(arg1, arg2)` (operation)

Returns: a ToDoListEntry

The first argument is a list of three-tubles like above. Once all preconditions become fulfilled the function given as second argument is launched.

5.2.7 SetTargetValueObject (for IsToDoListEntry, IsObject)

▷ `SetTargetValueObject(entry, value)` (operation)

Returns: nothing

If the given value of the target part is the return value of a function this command sets the target value of the entry to a function. This is done to keep proof tracking availible.

5.2.8 SetTargetObject (for IsToDoListEntry, IsObject)

▷ `SetTargetObject(entry, obj)` (operation)

Returns: nothing

If the target object, i.e. the first entry of the target part, was given as a function, this method can set this entry to the return value computed in ProcessToDoListEntry. This happens atomatically, do not worry about it.

5.2.9 ToDoListEntryWithContraposition (for IsObject, IsString, IsBool, IsObject, IsString, IsBool)

▷ `ToDoListEntryWithContraposition(sobj, source_prop, sval, tobj, target, tval)` (operation)

Returns: a ToDoListEntry

Creates a ToDoListEntry which also installs a contraposition. The arguments `source_prop` and `target` need to be strings which name a property, and `sval` and `tval` need to be boolean values, i.e. true or false.

5.2.10 DescriptionOfImplication (for IsToDoListEntry)

▷ `DescriptionOfImplication(arg)` (attribute)

Returns: a list

Has to be set to a string, which describes the reason for the conclusion. If the ToDo-list entry is displayed, the given string will be displayed with a because before it.

5.2.11 ToDoListEntryForEqualAttributes (for IsObject, IsString, IsObject, IsString)

▷ `ToDoListEntryForEqualAttributes(arg1, arg2, arg3, arg4)` (operation)

Returns: a ToDoListEntry

Creates a ToDoListEntry for two equal attributes, which means that both values of the two attributes will be propagated in both directions.

5.2.12 ToDoListEntryForEquivalentAttributes (for IsObject, IsString, IsObject, IsObject, IsString, IsObject)

▷ `ToDoListEntryForEquivalentAttributes(`*arg1, arg2, arg3, arg4, arg5, arg6*`)` (operation)

Returns: a ToDoListEntry

Creates a ToDoListEntry for two equivalent attributes, which means that both values of the two attributes will be propagated in both directions. Please note that this one does NOT implement contrapositions.

5.3 Category

5.3.1 IsToDoList (for IsObject)

▷ `IsToDoList(`*arg*`)` (filter)

Returns: `true` or `false`

This is the category of ToDo-lists. Every ToDo-list is an object of this category, which basically contains the ToDo-lists.

5.4 Constructor

5.4.1 NewToDoList

▷ `NewToDoList()` (operation)

Returns: nothing

Creates a new empty ToDo-list.

5.5 This is the magic

5.5.1 Process_A_ToDo_List_Entry

▷ `Process_A_ToDo_List_Entry(`*arg*`)` (function)

Returns: a boolean

Gets a ToDo-list entry, which is a pair of a list of strings and a weak pointer object, and processes it. If the action was done, it returns true, if not, it returns false, and it returns fail if the action is not possible anymore due to deleted objects.

5.5.2 ProcessToDoList (for IsObject)

▷ `ProcessToDoList(`*A*`)` (attribute)

Returns: nothing

This is the magic! This attribute is never set. Creating an ToDo-list entry installs an ImmediateMethod for this attribute for the specific category of the object to which ToDo-list is added, and the filter the entry contains. It is then triggert if the filters become applicable, so the ToDo-list is processed

5.6 Methods for all objects

5.6.1 ToDoList (for IsObject)

▷ `ToDoList(arg)` (attribute)

Returns: A ToDo-list

Returns the ToDo-list of an object, or creates a new one.

5.7 Proof tracking

This is a way to track proofs from ToDoLists. Not only for debugging, but also for knowing how things work together.

5.7.1 TraceProof (for IsObject, IsString, IsObject)

▷ `TraceProof(obj, name, val)` (operation)

Returns: a tree

If the object *obj* has the attribute *name*, and its value is *val*, and the knowledge has been obtained trough ToDoList-entries, this method traces the way the property was set, and returns a tree which describes the full way of how the attribute became known.

5.8 Maintainance

5.8.1 ActivateToDoList (for IsObject)

▷ `ActivateToDoList(arg)` (operation)

Returns: nothing

This operation activates ToDoLists for the argument.

5.8.2 ActivateToDoList

▷ `ActivateToDoList()` (operation)

Returns: nothing

This operation activates ToDoLists for all objects.

5.8.3 DeactivateToDoList (for IsObject)

▷ `DeactivateToDoList(arg)` (operation)

Returns: nothing

This operation deactivates ToDoLists for the argument.

5.8.4 DeactivateToDoList

▷ `DeactivateToDoList()` (operation)

Returns: nothing

This operation deactivates ToDoLists for all objects. Note that it is not possible to activate ToDoList for a single object while they are not activated. ToDoListEntries will yet be stored for all

objects that can have ToDoLists. All objects created while ToDoLists are deactivated have by default no ToDoList.

5.8.5 ActivateWhereInfosInEntries

▷ `ActivateWhereInfosInEntries(`*arg*`)` (function)

Returns: nothing

Stores the result of Where(100) in an entry if the entry is triggered. This is not activated by default, since it might slow down the system.

5.8.6 DeactivateWhereInfosInEntries

▷ `DeactivateWhereInfosInEntries(`*arg*`)` (function)

Returns: nothing

Deactives the storage of the result of Where(100) if an entry is triggered. This is the default.

Chapter 6

Basics

6.1 Global variables

6.1.1 HOMALG_TOOLS

▷ HOMALG_TOOLS (global variable)

A central place for configurations.

6.2 GAP Categories

6.2.1 IsStructureObjectOrObjectOrMorphism (for IsAttributeStoringRep)

▷ IsStructureObjectOrObjectOrMorphism(*arg*) (filter)

Returns: true or false

This is the super super GAP-category which will include the GAP-categories IsStructureObjectOrObject and IsHomalgObjectOrMorphism

6.2.2 IsStructureObjectOrObject (for IsStructureObjectOrObjectOrMorphism)

▷ IsStructureObjectOrObject(*arg*) (filter)

Returns: true or false

This is the super GAP-category which will include the GAP-categories IsHomalgRing, IsHomalgModule, IsHomalgRingOrModule and IsHomalgComplex

6.2.3 IsStructureObject (for IsStructureObjectOrObject)

▷ IsStructureObject(*arg*) (filter)

Returns: true or false

This is the super GAP-category which will include the GAP-categories IsHomalgRing we need this GAP-category to define things like Hom(M,R) as easy as Hom(M,N) without distinguishing between structure objects (e.g. rings) and objects (e.g. modules)

6.2.4 IsStructureObjectMorphism (for IsAttributeStoringRep)

▷ `IsStructureObjectMorphism(`*arg*`)` (filter)

Returns: `true` or `false`

This is the super GAP-category which will include the GAP-categories IsHomalgRingMap, etc.

6.2.5 IsHomalgRingOrModule (for IsStructureObjectOrObject)

▷ `IsHomalgRingOrModule(`*arg*`)` (filter)

Returns: `true` or `false`

this is the super GAP-category which will include the GAP-categories IsHomalgRing, IsHomalgModule:

Chapter 7

Pointers

7.1 Weak pointer objects

7.1.1 IsContainerForWeakPointers (for IsComponentObjectRep)

▷ `IsContainerForWeakPointers(`*arg*`)` (filter)

Returns: `true` or `false`

The category for weak pointer objects

7.1.2 ContainerForWeakPointers

▷ `ContainerForWeakPointers(`*arg*`)` (function)

Returns: a list which can store weak pointers

The constructor for lists of weak pointers.

7.1.3 UpdateContainerOfWeakPointers (for IsContainerForWeakPointers)

▷ `UpdateContainerOfWeakPointers(`*arg*`)` (operation)

Updates the weak pointers in a container and deletes the empty ones

7.1.4 _AddElmWPObj_ForHomalg

▷ `_AddElmWPObj_ForHomalg(`*arg*`)` (function)

Adds a weak pointer of an objects to a weak pointer list.

7.1.5 _AddTwoElmWPObj_ForHomalg

▷ `_AddTwoElmWPObj_ForHomalg(`*arg*`)` (function)

Adds a weak pointer which depends on two objects to a list of weak pointers

7.1.6 _ElmWPObj_ForHomalg (for IsContainerForWeakPointers, IsObject, IsObject)

▷ `_ElmWPObj_ForHomalg(`*arg1, arg2, arg3*`)` (operation)

Creates a weak pointer depending on two objects and adds it to the container.

7.2 Pointer objects

7.2.1 IsContainerForPointers (for IsComponentObjectRep)

▷ `IsContainerForPointers(`*arg*`)` (filter)
Returns: `true` or `false`
The category for pointer objects

7.2.2 ContainerForPointers

▷ `ContainerForPointers(`*arg*`)` (function)
Returns: a container for pointers
Creates a container for pointers.

7.2.3 UpdateContainerOfPointers (for IsContainerForPointers)

▷ `UpdateContainerOfPointers(`*arg*`)` (operation)

Updates the container of pointers, removes old.

7.2.4 _AddElmPObj_ForHomalg

▷ `_AddElmPObj_ForHomalg(`*arg*`)` (function)

Adds a pointer to an object to a container for pointers.

7.2.5 _AddTwoElmPObj_ForHomalg

▷ `_AddTwoElmPObj_ForHomalg(`*arg*`)` (function)

Adds a pointer to two objects to a container for pointers

7.2.6 _ElmPObj_ForHomalg (for IsContainerForPointers, IsObject, IsObject)

▷ `_ElmPObj_ForHomalg(`*arg1, arg2, arg3*`)` (operation)
Returns: an object
Returns an object which a pointer refers to.

Chapter 8

Tools

8.1 Functions

8.1.1 homalgTotalRuntimes

▷ `homalgTotalRuntimes(`*arg*`)` (function)

Returns: an integer

A tool to compute the runtime of several methods.

8.1.2 AddLeftRightLogicalImplicationsForHomalg

▷ `AddLeftRightLogicalImplicationsForHomalg(`*arg*`)` (function)

A tool to install equivalence between filters.

8.1.3 LogicalImplicationsForOneHomalgObject

▷ `LogicalImplicationsForOneHomalgObject(`*arg*`)` (function)

Installs a logical implication for one type with all it's contrapositions.

8.1.4 LogicalImplicationsForTwoHomalgBasicObjects

▷ `LogicalImplicationsForTwoHomalgBasicObjects(`*arg*`)` (function)

8.1.5 InstallLogicalImplicationsForHomalgBasicObjects

▷ `InstallLogicalImplicationsForHomalgBasicObjects(`*arg*`)` (function)

8.1.6 LeftRightAttributesForHomalg

▷ `LeftRightAttributesForHomalg(`*arg*`)` (function)

8.1.7 InstallLeftRightAttributesForHomalg

▷ `InstallLeftRightAttributesForHomalg(`*arg*`)` (function)

8.1.8 MatchPropertiesAndAttributes

▷ `MatchPropertiesAndAttributes(`*arg*`)` (function)

A method to match the properties and attributes of two objects.

8.1.9 InstallImmediateMethodToPullPropertyOrAttribute

▷ `InstallImmediateMethodToPullPropertyOrAttribute(`*arg*`)` (function)

Installs methods to pull new known properties and attributes from one object to another

8.1.10 InstallImmediateMethodToConditionallyPullPropertyOrAttribute

▷ `InstallImmediateMethodToConditionallyPullPropertyOrAttribute(`*arg*`)` (function)

Installs methods to pull new known properties and attributes under certain conditions from one object to another.

8.1.11 InstallImmediateMethodToPullPropertyOrAttributeWithDifferentName

▷ `InstallImmediateMethodToPullPropertyOrAttributeWithDifferentName(`*arg*`)` (function)

Installs an immediate method which can pull a property from one object to another with different names.

8.1.12 InstallImmediateMethodToPullPropertiesOrAttributes

▷ `InstallImmediateMethodToPullPropertiesOrAttributes(`*arg*`)` (function)

Installs an immediate method to pull several properties or attributes from one object to another.

8.1.13 InstallImmediateMethodToPullTrueProperty

▷ `InstallImmediateMethodToPullTrueProperty(`*arg*`)` (function)

Installs an immediate method to pull a property if it is true.

8.1.14 InstallImmediateMethodToConditionallyPullTrueProperty

▷ `InstallImmediateMethodToConditionallyPullTrueProperty(`*arg*`)` (function)

Installs an immediate method which conditionally pulls a property if it is true.

8.1.15 InstallImmediateMethodToPullTruePropertyWithDifferentName

▷ `InstallImmediateMethodToPullTruePropertyWithDifferentName(`*arg*`)` (function)

Installs an immediate method which pulls a property with a different name if it is true.

8.1.16 InstallImmediateMethodToPullTrueProperties

▷ `InstallImmediateMethodToPullTrueProperties(`*arg*`)` (function)

Installs an immediate method which pulls several properties if they are true

8.1.17 InstallImmediateMethodToPullFalseProperty

▷ `InstallImmediateMethodToPullFalseProperty(`*arg*`)` (function)

Installs an immediate method to pull a property if it is false.

8.1.18 InstallImmediateMethodToConditionallyPullFalseProperty

▷ `InstallImmediateMethodToConditionallyPullFalseProperty(`*arg*`)` (function)

Installs an immediate method which conditionally pulls a property if it is false.

8.1.19 InstallImmediateMethodToPullFalsePropertyWithDifferentName

▷ `InstallImmediateMethodToPullFalsePropertyWithDifferentName(`*arg*`)` (function)

Installs an immediate method which pulls a property with a different name if it is false.

8.1.20 InstallImmediateMethodToPullFalseProperties

▷ `InstallImmediateMethodToPullFalseProperties(`*arg*`)` (function)

Installs an immediate method which pulls several properties if they are false.

8.1.21 InstallImmediateMethodToPushPropertyOrAttribute

▷ `InstallImmediateMethodToPushPropertyOrAttribute(`*arg*`)` (function)

Installs an immediate method to push a property from one object to another.

8.1.22 InstallImmediateMethodToConditionallyPushPropertyOrAttribute

▷ `InstallImmediateMethodToConditionallyPushPropertyOrAttribute(`*arg*`)` (function)

Installs an immediate method to conditionally push a property from one object to another.

8.1.23 InstallImmediateMethodToPushPropertyOrAttributeWithDifferentName

▷ `InstallImmediateMethodToPushPropertyOrAttributeWithDifferentName(`*arg*`)` (function)

Installs an immediate method which can push a property from one object to another with different names.

8.1.24 InstallImmediateMethodToPushPropertiesOrAttributes

▷ `InstallImmediateMethodToPushPropertiesOrAttributes(`*arg*`)` (function)

Installs an immediate method to push several properties or attributes from one object to another.

8.1.25 InstallImmediateMethodToPushTrueProperty

▷ `InstallImmediateMethodToPushTrueProperty(`*arg*`)` (function)

Installs an immediate method to push a property if it is true.

8.1.26 InstallImmediateMethodToPushTruePropertyWithDifferentName

▷ `InstallImmediateMethodToPushTruePropertyWithDifferentName(`*arg*`)` (function)

Installs an immediate method which pushes a property with a different name if it is true.

8.1.27 InstallImmediateMethodToPushTrueProperties

▷ `InstallImmediateMethodToPushTrueProperties(`*arg*`)` (function)

Installs an immediate method which pushes several properties if they are true

8.1.28 InstallImmediateMethodToPushFalseProperty

▷ `InstallImmediateMethodToPushFalseProperty(`*arg*`)` (function)

Installs an immediate method to push a property if it is false.

8.1.29 InstallImmediateMethodToPushFalsePropertyWithDifferentName

▷ `InstallImmediateMethodToPushFalsePropertyWithDifferentName(`*arg*`)` (function)

Installs an immediate method which pushes a property with a different name if it is false.

8.1.30 InstallImmediateMethodToPushFalseProperties

▷ `InstallImmediateMethodToPushFalseProperties(`*arg*`)` (function)

Installs an immediate method which push several properties if they are false.

8.1.31 DeclareAttributeWithCustomGetter

▷ `DeclareAttributeWithCustomGetter(arg)` (function)

Installs an attribute with a coustom getter function.

8.1.32 AppendToAhomalgTable

▷ `AppendToAhomalgTable(arg)` (function)

Appends an entry to a homalg table.

8.1.33 homalgNamesOfComponentsToIntLists

▷ `homalgNamesOfComponentsToIntLists(arg)` (function)
Returns: a list of integers
Creates a list of integers out of the names of components.

8.1.34 IncreaseExistingCounterInObject

▷ `IncreaseExistingCounterInObject(arg)` (function)

Increases an existing counter in an object.

8.1.35 IncreaseExistingCounterInObjectWithTiming

▷ `IncreaseExistingCounterInObjectWithTiming(arg)` (function)

Increases an existiing counter on an object with timing.

8.1.36 IncreaseCounterInObject

▷ `IncreaseCounterInObject(arg)` (function)

Increases a counter in an object and creates one if it not exists

8.1.37 MemoryToString

▷ `MemoryToString(arg)` (function)

Converts the current memory state to a string

8.1.38 PrimePowerExponent

▷ `PrimePowerExponent(n, p)` (function)
Returns: A nonnegative integer
Returns the p-exponent of the integer n, where p is a rational prime.

8.1.39 ViewList (for IsList)

▷ `ViewList(L)` (operation)

Returns: nothing

Apply ViewObj to the list `L`.

8.1.40 homalgLaTeX (for IsObject)

▷ `homalgLaTeX(arg)` (operation)

8.1.41 IdenticalPosition (for IsList, IsObject)

▷ `IdenticalPosition(L, o)` (operation)

Returns: a positive integer or fail

Return the position of the object identical to `o` in the list `L`

8.1.42 PositionsOfMaximalObjects (for IsList, IsFunction)

▷ `PositionsOfMaximalObjects(L, f)` (operation)

Returns: a list

Return the list of positions of maximal objects in `L` w.r.t. the partial order defined by the binary function `f`.

8.1.43 MaximalObjects (for IsList, IsFunction)

▷ `MaximalObjects(L, f)` (operation)

Returns: a list

Return the sublist of maximal objects in `L` w.r.t. the partial order defined by the binary function `f`.

8.1.44 CollectEntries

▷ `CollectEntries(list)` (function)

Returns: a list

returns a new list that contains for each element *elm* of the list *list* a list of length two, the first element of this is *elm* itself and the second element is the number of times *elm* appears in list until the next different element. The default comparing function is `\=`, which can be changed by passing an optional value to *ComparingFunction*.

8.1.45 MakeShowable

▷ `MakeShowable(mime_types, filter)` (function)

Installs a method for `IsShowable` such that `IsShowable( mime_type, object )` returns `true` for any `mime_type` in the list *mime_types* and `object` in the filter *filter*.

8.1.46 MakeShowableWithLaTeX

▷ `MakeShowableWithLaTeX(`*`filter`*`)` (function)

Installs a method for `IsShowable` such that `IsShowable( "text/latex", object )` and `IsShowable( "application/x-latex", object )` return `true` for an `object` in the filter *`filter`*.

8.1.47 ReplacedStringViaRecord

▷ `ReplacedStringViaRecord(`*`string, record`*`)` (function)

Searches for the keys of *`record`* in *`string`* and replaces them by their values. The values can be strings or lists of strings. In the second case, the search term must be followed by `...` and the replacement string is formed by joining the entries of the list with the separator `", "`.

8.1.48 StartTimer

▷ `StartTimer(`*`name`*`)` (function)

(Re-)Starts a timer with the given name.

8.1.49 StopTimer

▷ `StopTimer(`*`name`*`)` (function)

Stops a timer with the given name.

8.1.50 DisplayTimer

▷ `DisplayTimer(`*`name`*`)` (function)

Displays the current value of the timer with the given name.

8.1.51 ListImpliedFilters

▷ `ListImpliedFilters(`*`filt`*`)` (function)

List the all filters implied by filter *`filt`*.

8.1.52 Breakpoint

▷ `Breakpoint(`*`name[, break_at[, break_function]]`*`)` (function)

If only a string *`name`* is given, displays an incrementing number every time a breakpoint with this name is visited. If additionally an integer *`break_at`* is given, enters a break-loop if the breakpoint has been visited the specified number of times. If a function *`break_function`* is given, it is executed before entering the break-loop.

8.1.53 ReadPackageOnce

▷ `ReadPackageOnce(`*name*`)` (function)

Like `ReadPackage` but reads the file only once in the runnig GAP session.

8.2 Example functions

8.2.1 ExamplesForHomalg

▷ `ExamplesForHomalg()` (operation)

Returns: `true` or `false`

Runs the examples for homalg if the package is loadable.

8.2.2 ExamplesForHomalg (for IsInt)

▷ `ExamplesForHomalg(`*arg*`)` (operation)

Returns: `true` or `false`

Runs the named example for homalg

Chapter 9

Trees

The trees are used in ToDoLists. They are a technical feature, and fairly general, so they also can be used somewhere else.

9.1 Trees

9.1.1 IsTree (for IsObject)

▷ `IsTree(`*arg*`)` (filter)

Returns: `true` or `false`

The category of trees. A tree may have a content, a list of successors, a predecessor and it knows if it is a leave of a tree or not.

9.1.2 Content (for IsTree)

▷ `Content(`*arg*`)` (attribute)

Returns: object

The content of the tree. May be any object.

9.1.3 ListOfSuccessors (for IsTree)

▷ `ListOfSuccessors(`*arg*`)` (operation)

Returns: a list of trees

Returns the list of successors of a tree.

9.1.4 Predecessor (for IsTree)

▷ `Predecessor(`*arg*`)` (operation)

Returns: a tree or fail

Returns the predecessor of a tree, or fail if there is none.

9.1.5 ListOfSentinels (for IsTree)

▷ `ListOfSentinels(`*arg*`)` (operation)

Returns: a list

Returns a list of leaves of the tree.

9.1.6 RemoveHead (for IsTree)

▷ `RemoveHead(`*arg*`)` (operation)

Returns: a tree

Returns the first successor of the tree, and adds all other successors of the tree to the tree that is returned. If the tree is a leave, it returns an empty tree. If the tree is empty, it returns the tree itself.

9.1.7 Tree

▷ `Tree()` (operation)

Returns: a tree

Returns an empty tree.

9.1.8 Tree (for IsObject)

▷ `Tree(`*obj*`)` (operation)

Returns: a tree

Returns a tree with argument *obj*.

9.1.9 Add (for IsTree, IsTree)

▷ `Add(`*tree, new_tree*`)` (operation)

Returns: nothing

Adds the [list of] tree[s] *new_tree* as successor to the tree *tree*.

9.1.10 ContentListFromSentinelToHead (for IsTree)

▷ `ContentListFromSentinelToHead(`*sent*`)` (operation)

Returns: a list

Returns a list of the contents of the trees from the leave *sent* up to the content of the head of the tree.

9.1.11 PostOrder (for IsTree)

▷ `PostOrder(`*arg*`)` (operation)

Returns: a list

Returns the contents of the nodes of the tree in post-order.

Index

_AddElmPObj_ForHomalg, 20
_AddElmWPObj_ForHomalg, 19
_AddTwoElmPObj_ForHomalg, 20
_AddTwoElmWPObj_ForHomalg, 19
_ElmPObj_ForHomalg
    for IsContainerForPointers, IsObject, IsObject, 20
_ElmWPObj_ForHomalg
    for IsContainerForWeakPointers, IsObject, IsObject, 20

ActivateToDoList, 15
    for IsObject, 15
ActivateWhereInfosInEntries, 16
Add
    for IsTree, IsTree, 30
AddLeftRightLogicalImplicationsForHomalg, 21
AddToToDoList
    for IsToDoListEntry, 12
AppendToAhomalgTable, 25

Breakpoint, 27

CacheValue
    for IsCachingObject, IsObject, 4
CachingObject, 4
    for IsObject, 4
    for IsObject, IsObject, 4
    for IsObject, IsObject, IsInt, 4
    for IsObject, IsObject, IsInt, IsBool, 4
CollectEntries, 26
ContainerForPointers, 20
ContainerForWeakPointers, 19
Content
    for IsTree, 29
ContentListFromSentinelToHead
    for IsTree, 30

DeactivateCachingObject, 5
DeactivateToDoList, 15
    for IsObject, 15
DeactivateWhereInfosInEntries, 16
DeclareAttributeWithCustomGetter, 25
DescriptionOfImplication
    for IsToDoListEntry, 13
DisplayTimer, 27

ExamplesForHomalg, 28
    for IsInt, 28

FunctionWithCache, 6

homalgLaTeX
    for IsObject, 26
homalgNamesOfComponentsToIntLists, 25
homalgTotalRuntimes, 21
HOMALG_TOOLS, 17

IdenticalPosition
    for IsList, IsObject, 26
IncreaseCounterInObject, 25
IncreaseExistingCounterInObject, 25
IncreaseExistingCounterInObjectWithTiming, 25
InstallImmediateMethodToConditionallyPullFalseProperty, 23
InstallImmediateMethodToConditionallyPullPropertyOrAttribute, 22
InstallImmediateMethodToConditionallyPullTrueProperty, 22
InstallImmediateMethodToConditionallyPushPropertyOrAttribute, 23
InstallImmediateMethodToPullFalseProperties, 23
InstallImmediateMethodToPullFalseProperty, 23
InstallImmediateMethodToPullFalsePropertyWithDifferentName, 23

InstallImmediateMethodToPull-
PropertiesOrAttributes, 22
InstallImmediateMethodToPullProperty-
OrAttribute, 22
InstallImmediateMethodToPullProperty-
OrAttributeWithDifferentName,
22
InstallImmediateMethodToPullTrue-
Properties, 23
InstallImmediateMethodToPullTrue-
Property, 22
InstallImmediateMethodToPullTrue-
PropertyWithDifferentName, 23
InstallImmediateMethodToPushFalse-
Properties, 24
InstallImmediateMethodToPushFalse-
Property, 24
InstallImmediateMethodToPushFalse-
PropertyWithDifferentName, 24
InstallImmediateMethodToPush-
PropertiesOrAttributes, 24
InstallImmediateMethodToPushProperty-
OrAttribute, 23
InstallImmediateMethodToPushProperty-
OrAttributeWithDifferentName,
24
InstallImmediateMethodToPushTrue-
Properties, 24
InstallImmediateMethodToPushTrue-
Property, 24
InstallImmediateMethodToPushTrue-
PropertyWithDifferentName, 24
InstallLeftRightAttributesForHomalg, 22
InstallLogicalImplicationsForHomalg-
BasicObjects, 21
InstallMethodWithCache, 5
InstallMethodWithCacheFromObject, 6
InstallMethodWithCrispCache, 6
IsContainerForPointers
for IsComponentObjectRep, 20
IsContainerForWeakPointers
for IsComponentObjectRep, 19
IsEqualForCache
for IsObject, IsObject, 5
IsHomalgRingOrModule
for IsStructureObjectOrObject, 18
IsLazyArray
for IsComponentObjectRep and IsList, 7
IsLazyHList
for IsComponentObjectRep and IsList, 8
IsListWithAttributes
for IsAttributeStoringRep and IsList, 9
IsStructureObject
for IsStructureObjectOrObject, 17
IsStructureObjectMorphism
for IsAttributeStoringRep, 18
IsStructureObjectOrObject
for IsStructureObjectOrObjectOrMorphism,
17
IsStructureObjectOrObjectOrMorphism
for IsAttributeStoringRep, 17
IsToDoList
for IsObject, 14
IsTree
for IsObject, 29

LazyArray, 7
LazyArrayFromList, 7
LazyArrayWithValues, 7
LazyConstantArray, 7
LazyHList, 8
LazyInterval, 7
LazyStandardInterval, 7
LeftRightAttributesForHomalg, 21
ListImpliedFilters, 27
ListOfSentinels
for IsTree, 29
ListOfSuccessors
for IsTree, 29
ListWithAttributes, 9
LogicalImplicationsForOneHomalgObject,
21
LogicalImplicationsForTwoHomalgBasic-
Objects, 21

MakeShowable, 26
MakeShowableWithLaTeX, 27
MatchPropertiesAndAttributes, 22
MaximalObjects
for IsList, IsFunction, 26
MemoryToString, 25

NewToDoList, 14

PositionsOfMaximalObjects
for IsList, IsFunction, 26
PostOrder
for IsTree, 30
Predecessor
for IsTree, 29
PrimePowerExponent, 25
ProcessAToDoListEntry
for IsToDoListEntry, 12
ProcessToDoList
for IsObject, 14
Process_A_ToDo_List_Entry, 14

ReadPackageOnce, 28
RemoveHead
for IsTree, 30
ReplacedStringViaRecord, 27

SetCacheValue
for IsCachingObject, IsObject, IsObject, 5
SetCachingObjectCrisp, 5
SetCachingObjectWeak, 5
SetTargetObject
for IsToDoListEntry, IsObject, 13
SetTargetValueObject
for IsToDoListEntry, IsObject, 13
SourcePart
for IsToDoListEntry, 12
StartTimer, 27
StopTimer, 27

TargetPart
for IsToDoListEntry, 12
ToDoList
for IsObject, 15
ToDoListEntry
for IsList, IsFunction, 13
for IsList, IsList, 11
for IsList, IsObject, IsString, IsObject, 12
ToDoListEntryBlueprint
for IsObject, IsList, IsList, 11
ToDoListEntryForEqualAttributes
for IsObject, IsString, IsObject, IsString, 13
ToDoListEntryForEquivalentAttributes
for IsObject, IsString, IsObject, IsObject, IsString, IsObject, 14
ToDoListEntryToMaintainEqualAttributes
for IsList, IsList, IsList, 10
ToDoListEntryToMaintainEqual-
AttributesBlueprint
for IsObject, IsList, IsList, IsList, 11
ToDoListEntryToMaintainFollowing-
Attributes
for IsList, IsList, IsList, 10
ToDoListEntryToMaintainFollowing-
AttributesBlueprint
for IsObject, IsList, IsList, IsList, 11
ToDoListEntryWithContraposition
for IsObject, IsString, IsBool, IsObject, IsString, IsBool, 13
ToDoList_this_object, 11
TraceProof
for IsObject, IsString, IsObject, 15
Tree, 30
for IsObject, 30
TypedListWithAttributes, 9

UpdateContainerOfPointers
for IsContainerForPointers, 20
UpdateContainerOfWeakPointers
for IsContainerForWeakPointers, 19

ViewList
for IsList, 26