`IsMatrix` (in GAP 4.8.7) . . .

. . . is **documented** as a
"list of lists of equal length whose entries lie in a common ring"
–see `lib/arith.gd`.

. . . is **implemented** without the test for equal length
–see `lib/list.gd` and `IsTableListDefault` in `src/lists.c`.

`IsRectangularTable` is a property; however, for plain lists, it behaves like a category, due to kernel support –see `FN_IS_RECT` in `src/gap.h` and `src/plist.c`.

(Some old GAP library/package code calls `IsMatrix` in fact for checking `IsRectangularTable`.)

Decide how to fix the inconsistency.

IsMatrix vs. IsMatrixObj

IsMatrix	IsMatrixObj
nonempty	may have zero rows or columns
dense list of hom. lists	not nec. a list
contains ring elements	not nec. commutative addition of entries
no stored/fixed BaseDomain	stored BaseDomain
(DefaultRing is expensive)	
not nec. rectangular (?)	rectangular
also for Lie matrices	implies IsOrdinaryMatrix

If IsMatrix would imply IsRectangularTable
and **if** the associativity condition for IsMatrixObj would be omitted
and **if** we regard the BaseDomain overhead for IsMatrix as acceptable
then we could define IsMatrix $\Longrightarrow$ IsMatrixObj,
and "migrate IsMatrix to IsMatrixObj".

Otherwise, something like IsMatrixObjOrMatrix would be needed.

Available kinds of matrices

kind	defining filter	BaseD.	file	rows?
plain lists	IsPlistRep	–	list.gd	–
GF(2) matrices	IsGF2MatrixRep	GF(2)	vecmat.gd	+
GF(q) matrices	IsGF2MatrixRep	GF(q)	mat8bit.gd	+
"cmats"	IsCMatRep	GF(q)	pkg/cvec	+
block matrices	IsBlockMatrixRep	–	matblock.gi	–
wrapped lists	IsPlistMatrixRep	yes	matobjplist.gd	+
with memory	IsObjWithMemory	–	memory.gd	
Lie matrix	IsLieObject	–	liefam.gd	
NullMapMatrix	IsNullMapMatrix	–	matrix.gi	–
EmptyMatrix(p)	IsEmpty	–	algmat.gi	–

MatrixObj interface

Defining operations:
`BaseDomain`, `NumberOfRows`, `NumberOfColumns`, ...

Representation preserving constructors:
`NewMatrix`, `(New)ZeroMatrix`, ...

Arithmetics:
addition, multiplication, `Zero`, ...

Access/modification:
`m[i,j]`, `CopySubMatrix`, `PositionNonZero`, ...
(restricted to admissible positions, restricted mutability)

Mathematical operations:
`TraceMat`, `NullspaceMat`, ...
(declare for `IsMatrixObj`, implement generic methods using the low level interface)

In-place conversion

Explicit conversions from Plist representations (or in-place change of the "base domain"):

`ConvertTo(GF2)VectorRep(NC)`, `ConvertTo(GF2)MatrixRep(NC)`, `CONV_GF2VEC`, `CONV_GF2MAT`, `CONV_MAT8BIT`

Silently switching to a Plist representation (due to assignments)

Try to avoid this?

`ImmutableMatrix` is used frequently.
Is this the right solution?

Tasks

- Decide the relation between `IsMatrix` and `IsMatrixObj`,
  define `DefaultRing` for the entries as `BaseDomain` of Plist matrices,
  introduce `ZeroOfBaseDomain`.
- In the (about 240) library methods that **require** `IsMatrix`, adjust the code according to the `IsMatrixObj` interface.
  (Avoid `Zero(m[1][1])`, `m[i][j]`, and working with rows.)
- In the (about 90) library methods that **call** `IsMatrix`, decide if one can use `IsMatrixObj` instead.
- Change `m[i][j]` to `m[i,j]`.
  (Only where really matrices are affected?)
- Check the library methods that **create** matrices:
  What can be done in order to choose a suitable kind of matrix?

Tasks (continued)

- Replace `PositionNot( obj, zero )` by `PositionNonZero( obj )`.
  (And change the default methods for `PositionNonZero`.)
- Replace `EmptyMatrix` and `NullMapMatrix` by `IsMatrixObj` objects.
- Document and implement the interface.
- Provide test code.
- Provide further kinds of vectors/matrices.