FusionRings

Fusion rings as GAP objects

0.1.0

2026-02-01

Contents

Chapter 1

Introduction

1.1 Overview

FusionRings provides GAP objects representing fusion rings with label-based APIs, multiple internal representations, and validation utilities.

1.2 Installation and Loading

If the package is installed in a GAP package directory, load it via `LoadPackage("FusionRings")`.

For direct loading without GAP's package mechanism, use the helper loader in `read_direct.g`:

Gap
```
Read("/Users/cesargalindo/Documents/FusionRings/read_direct.g");
```

1.3 Quick Start

Gap
```
G := Group((1,2,3));;
F := PointedFusionRing(G);;
MultiplyBasis(F, (1,2,3), (1,2,3));
FusionRings_TestAllStrict();
Display(F);
```

Example output:

plain
```
FusionRing
  rep: rule
  rank: 3
  one: ()
  labels: [ (), (1,2,3), (1,3,2) ]
```

Chapter 2

API

2.1 Constructors

2.1.1 FusionRing

▷ `FusionRing(I, one, dual, multOrData, opts)` (function)
▷ `FusionRingByRule(I, one, dual, mult, opts)` (function)
▷ `FusionRingBySparseConstants(I, one, dual, prodTable, opts)` (function)
▷ `FusionRingByFusionMatrices(labels, one, dual, fusionMatrices, opts)` (function)
▷ `PointedFusionRing(G)` (function)
▷ `CyclicPointedFusionRing(n)` (function)
▷ `FibonacciFusionRing()` (function)
▷ `IsingFusionRing()` (function)
▷ `TambaraYamagamiFusionRing(A)` (function)
▷ `NearGroupFusionRing(G, k)` (function)
▷ `TYMLabel(F)` (attribute)
▷ `NGRhoLabel(F)` (attribute)

Constructors for fusion rings and the pointed fusion ring of a group. See also `PointedFusionRing`.

2.2 Core Operations

2.2.1 MultiplyBasis

▷ `MultiplyBasis(F, i, j)` (operation)
▷ `FusionCoefficient(F, i, j, k)` (operation)
▷ `DualLabel(F, i)` (operation)
▷ `FusionMatrix(F, i)` (operation)
▷ `CheckFusionRingAxioms(F, level)` (operation)
▷ `CheckFusionRingAxiomsSample(F, level, samples)` (function)

Basic algebraic operations and verification. See `MultiplyBasis` and `CheckFusionRingAxioms`.

2.3 Attributes and Indexing

2.3.1 BasisLabels

▷ `BasisLabels(F)` (attribute)
▷ `OneLabel(F)` (attribute)
▷ `DualData(F)` (attribute)
▷ `RepresentationType(F)` (attribute)
▷ `LabelsList(F)` (attribute)
▷ `DualTable(F)` (attribute)
▷ `FusionMatrices(F)` (attribute)
▷ `PositionOfLabel(F, i)` (operation)
▷ `LabelOfPosition(F, p)` (operation)
▷ `NormalizeProductList(list[, F])` (function)

Accessors and helpers for label sets, indexing, dual tables, and fusion matrices.

2.4 Testing Helpers

2.4.1 FusionRings_TestAll

▷ `FusionRings_TestAll()` (function)
▷ `FusionRings_TestAllStrict()` (function)
▷ `FusionRings_RewriteTests()` (function)

Convenience wrappers for running and normalizing the package tests.

2.5 Export / Import

2.5.1 FusionRingRecord

▷ `FusionRingRecord(F)` (function)
▷ `FusionRingFromRecord(rec)` (function)
▷ `SaveFusionRing(filename, F)` (function)
▷ `LoadFusionRing(filename)` (function)

Serialize and restore fusion rings in a GAP record format. Rule-based representations are not serializable.

2.6 Options

Constructors accept an optional record `opts` with fields:

- `storeRepresentation`: "rule", "sparse", or "matrices".
- `check`: 0, 1, or 2 (verification level).
- `inferDual`: true/false to infer the dual when possible.

- `buildIndex`: true/false to build a labels index.
- `makeImmutable`: true/false to finalize as immutable.

2.7 ModularData (Quick API)

2.7.1 ModularData

▷ `ModularData(rec, |, S, T[, labels])` (function)
▷ `ModularDataFromNsdRecord(rec)` (function)
▷ `ModularDataFromST(S, T[, labels])` (function)
▷ `ValidateModularData(md[, level])` (function)
▷ `LoadNsdGOL(rank)` (function)
▷ `GetModularData(rank, iGO, iMD)` (function)
▷ `FusionRingFromModularData(md)` (function)
▷ `SMatrix(md)` (attribute)
▷ `TMatrix(md)` (attribute)
▷ `MDLabels(md)` (attribute)
▷ `MDSpins(md)` (attribute)
▷ `MDTwists(md)` (attribute)
▷ `MDQuantumDimensions(md)` (attribute)
▷ `MDGlobalDimensionSquared(md)` (attribute)
▷ `MDFusionCoefficients(md)` (attribute)
▷ `MDOrderT(md)` (attribute)

Modular data support (rank ≤ 12 database, reconstruction via balancing equation, and validation levels 1–7). See the full technical guide in `doc/modular_data.md`.

Chapter 3

Data Formats

3.1 Product Lists

The product `i*j` is represented as a list of pairs `[[k1,n1],[k2,n2],...]` with integer coefficients `ni >= 0`. The helper `NormalizeProductList` combines repeated terms and removes zeros.

3.2 Sparse Table

The sparse table can be given as a list of triples `[i, j, productList]`, or as a record keyed by pairs. See examples in the tests.

Chapter 4

Representations and Performance

4.1 Representations

- `rule`: multiplication by function; minimal storage.
- `sparse`: explicit sparse products per pair.
- `matrices`: fusion matrices `N_i` stored explicitly.

4.2 Performance Tips

- For large objects, prefer `sparse` or `matrices`.
- Disable heavy checks with `check := 0` and use on-demand validation.
- Keep `buildIndex := true` for faster label lookup.
- Use `FusionMatrix` only when needed to avoid large allocations.

4.3 Printing

Fusion rings have custom `ViewObj` and `PrintObj` output to show representation, rank, and labels.

Chapter 5

Verification Levels

5.1 CheckFusionRingAxioms

Levels:

- `0`: internal consistency only (fast).
- `1`: unit, involution, Frobenius reciprocity, and basic dual checks.
- `2`: full associativity and Frobenius checks (expensive).

For large rings, run level 0 or 1 during construction and reserve level 2 for small examples or dedicated validation runs.

5.2 Sampled Checks

For large rings you may use `CheckFusionRingAxiomsSample` to test randomly selected triples of labels.

Gap
```
CheckFusionRingAxiomsSample(F, 2, 100);
```

Chapter 6

Design Decisions

6.1 Labels and Duality

- All public APIs use labels from `I`, not integer indices.
- Duality is stored as part of the object and is always available.

6.2 Immutability and Caches

- Objects are immutable after construction.
- Derived data (dual table, matrices, labels list) are cached via attributes.

6.3 Multiple Representations

- Rule-based, sparse, and matrices representations share one public API.
- Internal indexing is optional and used for performance.

Chapter 7

Limitations and Roadmap

7.1 Current Limitations

- No modular data (S/T matrices) in this version.
- Minimal pretty-printing; focus is on core algebra.
- Associativity checks at level 2 can be expensive for large rings.

7.2 Planned Extensions

- Modular data and categorical invariants.
- Serialization/import/export tools.
- Enhanced docs and examples.

Chapter 8

Testing

8.1 Running Tests

After loading via `read_direct.g`, use:

```
FusionRings_TestAll();
```

This runs all `.tst` files in the package test directory.
For strict checking and normalized outputs:

```
FusionRings_RewriteTests();    # normalize outputs once per GAP install
FusionRings_TestAllStrict();   # strict (ignores newline-only diffs)
```

8.2 Test Suite Contents

- `tst/test_pointed.tst`: pointed fusion ring basics.
- `tst/test_sparse.tst`: sparse representation (Z/2 example).
- `tst/test_matrices.tst`: matrices representation (Z/2 example).
- `tst/test_axioms2.tst`: level-2 axioms on small cases.
- `tst/test_families.tst`: Fibonacci/Ising/TY/cyclic constructors.
- `tst/test_neargroup.tst`: near-group (G+k) rules.
- `tst/test_export.tst`: save/load roundtrip.
- `tst/test_export_matrices.tst`: save/load roundtrip (matrices).

Chapter 9

Building Documentation

9.1 Manual Build

If GAP is available, build the manual by reading:

```
Gap
Read("/Users/cesargalindo/Documents/FusionRings/pkg/FusionRings/doc/build_manual.g");
```

This generates `manual.six`, `chap0.html`, and `manual.pdf` (PDF requires a TeX installation).

Chapter 10

Examples

10.1 Z/2Z Pointed Ring

```
labels := [ "1", "x" ];;
prodTable := [
  [ "1", "1", [ [ "1", 1 ] ] ],
  [ "1", "x", [ [ "x", 1 ] ] ],
  [ "x", "1", [ [ "x", 1 ] ] ],
  [ "x", "x", [ [ "1", 1 ] ] ]
];;
F := FusionRingBySparseConstants(labels, "1", [ "1", "x" ], prodTable, rec(check := 1));;
MultiplyBasis(F, "x", "x");
```

10.2 Fibonacci and Ising

```
F := FibonacciFusionRing();;
MultiplyBasis(F, "x", "x");
F := IsingFusionRing();;
MultiplyBasis(F, "sigma", "sigma");
```

10.3 Pointed Cyclic

```
F := CyclicPointedFusionRing(4);;
CheckFusionRingAxioms(F, 1);
```

10.4 Near-group (G + k)

```
G := CyclicGroup(3);;
F := NearGroupFusionRing(G, 1);;
rho := NGRhoLabel(F);;
```

```
MultiplyBasis(F, rho, rho);
```

10.5 Tambara–Yamagami

```gap
A := CyclicGroup(3);;
F := TambaraYamagamiFusionRing(A);;
m := TYMLabel(F);;
m <> fail;
MultiplyBasis(F, m, m);
CheckFusionRingAxioms(F, 1);
```