Ax — predefined axioms

Table of contents

1 Axioms

In MuPAD an algebraic structure may be represented by a *domain*. Parameterized domains may be defined by *domain constructors*. Many domain constructors are defined in the library package `Dom`.

Domains which have an similar mathematical structure may be members of a *category*. A category adds a level of abstraction because it postulates conditions which must hold for a domain in order to become a valid member of the category. Operations may be defined for all members of a category based on the assumptions and basic operations of that category, as long as they make no assumptions about the representation of the elements of the domain that belong to the category. Categories may also depend on parameters and are created by *category constructors*. The category constructors of the MuPAD library are contained in the library package `Cat`.

Attributes of domains and categories are defined in terms of so-called *axioms*. Axioms state properties of domains or categories. They may also depend on parameters and are defined by *axiom constructors*. The axiom constructors of the MuPAD library are contained in the package `Ax`, which is described in this document.

Please note that most axioms of the domains and categories defined in the MuPAD library are not stated explicitly. Only axioms which are not implied by the definition of a category are stated explicitly. The category of groups for example has no axiom stating that the multiplication is invertible because that is implied by the definition of a group. Most axioms defined in this package are of technical (i.e. algorithmic nature).

The definition of new axiom constructors is described in detail in the paper "Axioms, Categories and Domains" [1].

Changes since Version 1.4

The definition and implementation of axiom constructors has been changed considerably:

- A new syntax has been introduced for axiom constructors.
- Constructor parameters and local variables are now bound lexically instead of being substituted into the methods.
- The former special name `this` has been renamed to `dom`.

This changes are described in detail in the paper "Axioms, Categories and Domains" [1].

Changes since Version 1.2.2

Since MuPAD version 1.2.2 the following changes where made:

Most notably, all the constructors have been inserted into three library domains, in order to avoid global names and naming conflicts:

- All domain constructors and domains have been inserted into the new library domain `Dom`.
- The category constructors and categories have been inserted into the library domain `Cat`.
- The axioms have been inserted into the library domain `Ax`.

Thus the domain constructor for matrices now is called `Dom::Matrix` instead of simply `Matrix`, and the category of rings is called `Cat::Ring` instead of `Ring`.

The former global names may be exported from these library domains. With `export(Dom)` one gets all the former domain constructor and domain names for example.

The method names of the category `Cat::FactorialDomain` (formerly `FactorialDomain`) have been changed slightly, which involves the sub-categories and domains of this category.

References

[1] K. Drescher. Axioms, Categories and Domains. *Automath Technical Report* No. 1, Univ. GH Paderborn 1995.

`Ax::canonicalOrder` – the axiom of canonically ordered sets

`Ax::canonicalOrder` states that domain elements are canonically ordered.

Generating the axiom:

- `Ax::canonicalOrder`

Details:

- The axiom `Ax::canonicalOrder` is used to state that a domain has an order < (`_less`) which is defined by the canonical order of the MuPAD-expressions.
- This implies that the order of two elements is defined by the system function `_less`.

Changes:

- No changes.

`Ax::canonicalRep` – the axiom of canonically representation

`Ax::canonicalRep` states that domain elements are canonically represented.

Generating the axiom:

- `Ax::canonicalRep`

Details:

- The axiom `Ax::canonicalRep` is used to state that the elements of a domain are represented canonically, i.e. that each element of the domain has only one unique expression which represents it.
- This axiom implies that for an abelian monoid the axiom `Ax::normalRep` also holds. This is not enforced by the category but must be stated by the implementor of a domain.

Changes:

- No changes.

`Ax::canonicalUnitNormal` – the axiom of canonically unit normals

`Ax::canonicalUnitNormal` states that the method `"unitNormal"` of an integral domain (category `Cat::IntegralDomain`) returns a unique unit normal.

Generating the axiom:

- `Ax::canonicalUnitNormal`

Details:

- The axiom `Ax::canonicalUnitNormal` is used to state that the unit normals of an integral domain (category `Cat::IntegralDomain`) returned by the method `"unitNormal"` are unique.
- This means that for each non-zero element `x` of the integral domain there exists an unique associate among the associate class of `x`, i.e. for any `x` and `y` of a domain `dom` of category `Cat::IntegralDomain` where `dom::associates(x, y)` returns `TRUE` the equation `dom::equal(dom::unitNormal(x), dom::unitNormal(y)) = TRUE` must hold.
- Note that this axiom does not imply that the unit normals are canonically represented. The unit normals of `x` and `y` must be mathematically equal in the sense of the method `"equal"`, they need not be structurally equal as MuPAD objects.

Changes:

- No changes.

`Ax::closedUnitNormals` – the axiom of closed unit normals

`Ax::closedUnitNormals` states that the unit normals of an integral domain are closed under multiplication.

Generating the axiom:

- `Ax::closedUnitNormals`

Details:

- The axiom `Ax::closedUnitNormals` is used to state that the unit normals of an integral domain are closed under multiplication, i.e., that `dom::equal(x, dom::unitNormal(a) * dom::unitNormal(b)) = TRUE` implies `dom::equal(x, dom::unitNormal(x)) = TRUE` for all elements `x`, `a` and `b` of the domain `dom`.
- This axiom may be used only in conjunction with the axiom `Ax::canonicalUnitNormal`. If an integral domain has no unique unit normals, this axiom may not be stated.

Changes:

- No changes.

`Ax::efficientOperation` – the axiom of efficient operations

`Ax::efficientOperation((oper))` states that operation `oper` can be performed efficiently.

Generating the axiom:

- `Ax::efficientOperation(oper)`

Parameters:

`oper` — A string which defines the efficient operation.

Details:

- The axiom `Ax::efficientOperation(oper)` is used to state that the operation `oper` can be performed efficiently.
- The string `oper` must be the name of the operations slot in the domain stating the axiom. Examples are `"_mult"`, `"_invert"` or `"_divide"`.

Changes:

- No changes.

`Ax::normalRep` – the axiom of normal representation

`Ax::normalRep` states that an abelian monoid has a canonically representation of its zero element.

Generating the axiom:

- `Ax::normalRep`

Details:

- The axiom `Ax::normalRep` is used to state that an abelian monoid has a canonically representation of its zero element, i.e., that there is only one unique expression to represent zero.
- If the axiom `Ax::normalRep` holds for a domain `dom`, one may test for zero by comparing an element with `dom::zero` using the system function `_equal`.

Changes:

- No changes.

`Ax::noZeroDivisors` – the axiom of rng's with no zero divisor

`Ax::noZeroDivisors` states that a ring without a unit has no zero divisors.

Generating the axiom:

- `Ax::noZeroDivisors`

Details:

- The axiom `Ax::noZeroDivisors` is used to state that a ring without a unit has no zero divisors, i.e. that the product of two non-zero elements is never zero.
- Note that an integral domain implizitly has no zero divisors.

Changes:

- No changes.

`Ax::systemRep` – the axiom of facade domains

`Ax::systemRep` states that domain elements are represented by elements of built-in domains.

Generating the axiom:

- `Ax::systemRep`

Details:

- There are principally two ways to represent the elements of a domain: On the one hand the elements may be created explicitly by the system function `new`, on the other hand one may use the built-in (or basic) domains of MuPAD (like `DOM_INT`) to represent the elements.
- Domains which don't create elements of their own but use elements of basic domains instead are called *facade domains*.
- The usage of basic domains for the representation has the advantage that system functions may be used directly as methods of the domain without the overhead caused by overloading and procedure calls. But it has some severe limitations, see the domain `Dom::Expression` for details.
- The axiom `Ax::systemRep` is used to state that the elements of a domain are represented by basic domains and are not created by `new`.

Changes:

- No changes.