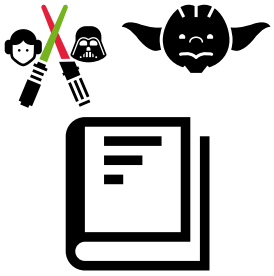
`galactic` user guide

The Galactic Organization <contact@thegalactic.org>



0.5.0.dev Fri Mar 3 10:40:53 UTC 2023

Contents

List of Figures

1 Introduction

The `py-galactic-apps` project includes several applications using the `py-galactic-core` library .

- `galactic-laser-bin` and `galactic-laser-ui` are able to draw the Hasse diagram of a concept lattice using a data file and an optional explorer file.

[1] © 2018-2023 the Galactic Organization. This document is licensed under CC-by-nc-nd (https://creativecommons.org/licenses/by-nc-nd/4.0/deed.en)

- `galactic-rules-bin` and `galactic-rules-ui` are able to produce basis of rules of a concept lattice using a data file and an explorer file.
- `galactic-lattice-cardinalities` is able to compute the number of concepts, level by level

2 Applications

2.1 The `galactic-laser-bin` tool

`galactic-laser-bin` is the console application able to draw the Hasse diagram of a concept lattice.

You can check the available options using:

```
$ galactic-laser-bin -h
```

or

```
$ galactic-laser-bin --help
```

```
$ galactic-laser-bin -h
usage: galactic-laser-bin [-h] [--explorer EXPLORER] [--mode
↪  {compact,full}]
                          [--join-irreducible-color JOIN-IRREDUCIBLE-COLOR]
                          [--meet-irreducible-color MEET-IRREDUCIBLE-COLOR]
                          [--log]
                          DATA [OUTPUT]

Create Hasse diagram image file from a data file.

positional arguments:
  DATA                  data file using format: .yaml, .yml, .toml, .txt,
                        .slf, .json, .ini, .dat, .csv, .cxt
  OUTPUT                Hasse diagram image file in .pdf, .svg, .png,
↪  .jpg,
                        .jpeg or .dot format

optional arguments:
  -h, --help            show this help message and exit
  --explorer EXPLORER   explorer in .yaml format
  --mode {compact,full}
                        display individuals in compact or full form
  --join-irreducible-color JOIN-IRREDUCIBLE-COLOR
                        graphviz color for join irreducible concept (see
                        http://graphviz.org/doc/info/colors.html) or use
                        '#colorcode' from '#000000' 'to '#FFFFFF'
  --meet-irreducible-color MEET-IRREDUCIBLE-COLOR
                        graphviz color for meet irreducible concept (see
```

```
                        http://graphviz.org/doc/info/colors.html) or use
                        '#colorcode' from '#000000' 'to '#FFFFFF'
  --log                 log operations

Do. Or do not. There is no try. - Yoda
```

You need to indicate a path for the image that will be generated and its extension.

Example:

```
$ galactic-laser-bin \
    --join-irreducible-color '#123456' \
    /data/animals.yaml \
    /data/animals.svg
```

2.2 The `galactic-laser-ui` tool

To launch the application, type in your terminal

```
$ galactic-laser-ui
```

The application window will open. Then you have to click on the data button.

Figure 1: galactic-laser data window

If nothing happens, check your installation of the *galactic* eco-system: Installation guide.

This action will open a dialog box, then you will have to look for a supported data file.

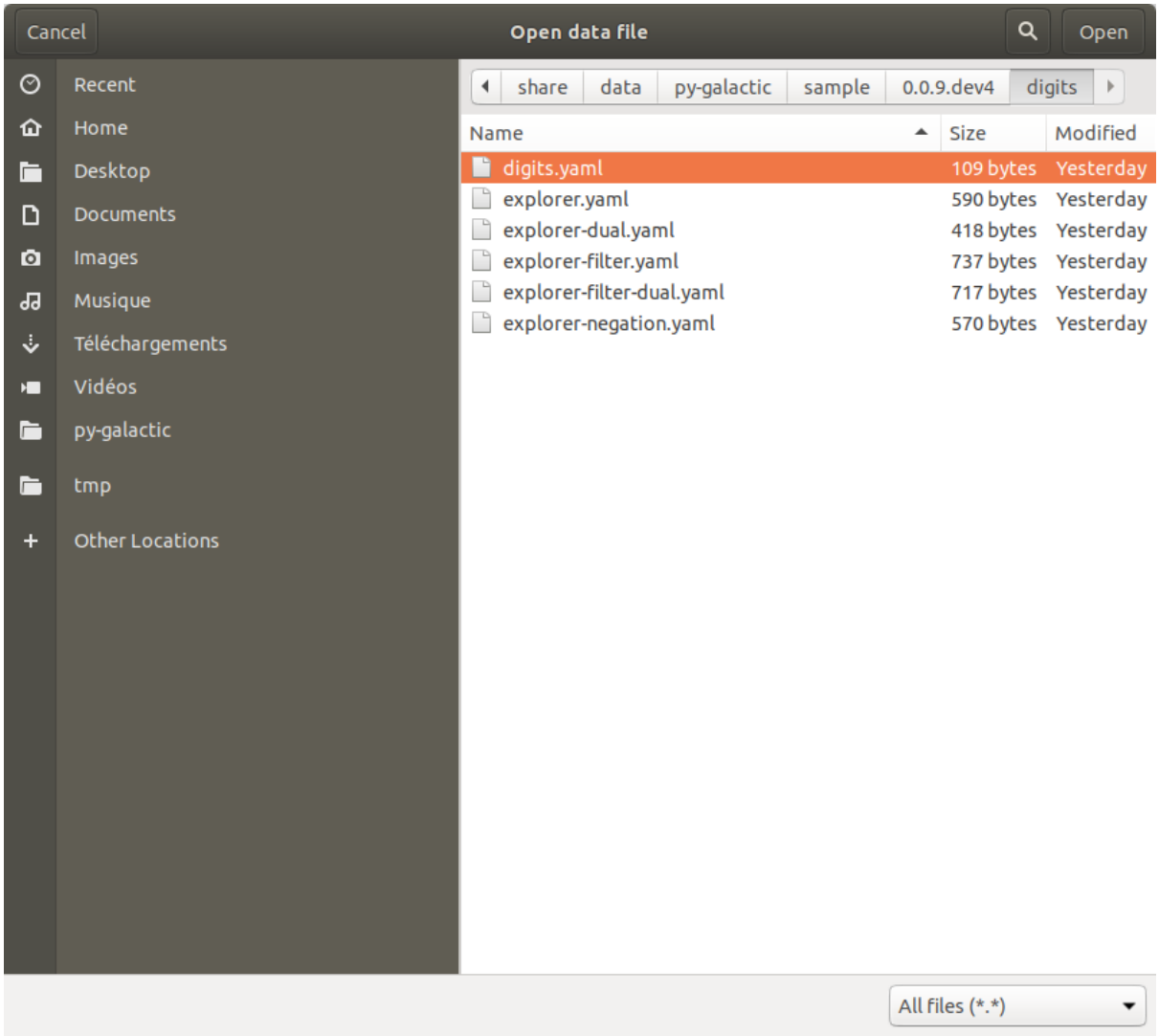


Figure 2: galactic-laser data dialog

Once the data file is chosen, you have several possible interactions.

It is possible to add an explorer file or the application will try to use a default behaviour.

You can change:

- the colors of the nodes for the current concept, the join-irreducible and the meet-irreducible;
- the node mode to the compact or complete nodes;
- the trace mode
- the draw mode.

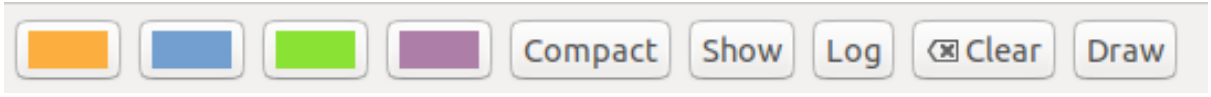


Figure 3: galactic-laser option chooser

To create the graph, just click on the “Start” button.

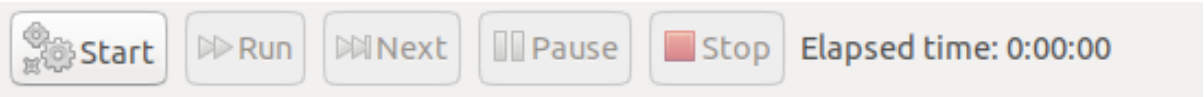


Figure 4: galactic-laser action buttons

Then you have two choices:

- click on the “Next” button if you want to process step by step;
- click on the “Run” button with the possibility to pause or stop.

If the drawing is just paused, there are possibilities to either continue or switch to the step-by-step mode.

If there is a press on the “Stop” button the drawing will be stopped and you will have to start again from the beginning.

3 Data readers

Data readers are used by the different applications to read contexts. They are organized in plugins, so the *py-galactic-core* engine can be easily extended. For now on, here is the list of readers we have developed:

- *YAML* data reader
- *JSON* data reader
- *CSV* data reader
- *TOML* data reader
- *INI* data reader
- *TEXT* data reader
- *FIMI* data reader
- *Burmeister* data reader
- *SLF* data reader

Data reader	Boolean characteristics	Heterogenous characteristics	Complex characteristics
YAML	yes	yes	yes
JSON	yes	yes	yes
CSV	yes	yes	no
TOML	yes	yes	no
INI	yes	yes	no
FIMI	yes	no	no
Text	yes	no	no
Burmeister	yes	no	no

Data reader	Boolean characteristics	Heterogenous characteristics	Complex characteristics
SLF	yes	no	no

3.1 *YAML* data reader

py-galactic-reader-yaml is a YAML data reader plugin for *py-galactic*.

It uses the `yaml.load` function of the PyYAML library.

The file extension is `.yaml` or `.yml`. The individuals are represented either by:

- a list: individuals are named by an integer starting from 0;
- a dictionary: individuals are named by their keys.

For example:

```
0: [c, e, s]
1: [o, s]
2: [e, p]
3: [o, p]
4: [c, e, s]
5: [o, p]
6: [c, e]
7: [o, p]
8: [c, e]
9: [c, o, s]
```

3.2 *JSON* data reader

py-galactic-reader-json is a JSON data reader plugin for *py-galactic*.

The file extension is `.json`. The individuals are either represented by a list or by a dictionary. For example:

```
{
  "#1": {
    "name": "Galois",
    "firstname": "Évariste"
  },
  "#2": {
    "name": "Wille",
    "firstname": "Rudolf"
  }
}
```

This reader uses the `json.load` function of the python core library.

3.3 *CSV* data reader

py-galactic-reader-csv is a CSV data reader plugin for *py-galactic*.

The file extension is `.csv`. The first line of each column contains the names of the characteristics. Each new line contains a new individual with its values. For example, an extraction of Fisher's IRIS data:

```
"sepal length","sepal width","petal length","petal width","class"
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolor
6.4,3.2,4.5,1.5,Iris-versicolor
6.9,3.1,4.9,1.5,Iris-versicolor
5.5,2.3,4.0,1.3,Iris-versicolor
6.5,2.8,4.6,1.5,Iris-versicolor
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
7.1,3.0,5.9,2.1,Iris-virginica
6.3,2.9,5.6,1.8,Iris-virginica
6.5,3.0,5.8,2.2,Iris-virginica
```

This reader uses the `csv.DictReader` class of the python core library.

3.4 *TOML* data reader

py-galactic-reader-toml is a TOML data reader plugin for *py-galactic*.

The file extension is `.toml`. The individuals are represented in named sections by key = "value" pairs. For example:

```toml
[individual1]
    name = "Galois"
    firstname = "Évariste"
[individual2]
    name = "Wille"
    firstname = "Rudolf"
```

This reader uses the `toml.load` function of the toml library.

3.5 *INI* data reader

py-galactic-reader-ini is an INI data reader plugin for *py-galactic*.

The file extension is `.ini`. The individuals are represented in named sections by key = "value" pairs. For example:

```
[#1]
name = "Galois"
firstname = "Évariste"
[#2]
name = "Wille"
firstname = "Rudolf"
```

This reader uses the `configparser.ConfigParser` class of the python core library.

3.6 *TEXT* data reader

py-galactic-reader-text is a *TEXT* data reader plugin for *py-galactic*.

The file extension is `.txt`. There is a first section describing the observations and the attributes using the keywords `Observations` and `Attributes`, then each individual is described by a space separated list of attributes. For example:

```
Observations: 1 2 3 4
Attributes: a b c d e
1: a c
2: a b
3: b d e
4: c e
```

3.7 *FIMI* data reader

py-galactic-reader-fimi is a data reader plugin for *py-galactic*.

It uses the FIMI format.

The file extension is `.dat` and the file format consists in describing each individual in one line using a space separated list of attribute names. For example:

```
c e s
o s
e p
o p
c e s
o p
c e
o p
c e
c o s
```

3.8 *Burmeister* data reader

py-galactic-reader-burmeister is a data reader plugin for *py-galactic*.

The *Burmeister* context format uses the extension `.cxt`.

Each file is structured as follows:

- the first line consists of a single B;
- the second line is an empty line (its content is ignored);
- the third and fourth lines contain the object and attribute count;
- the fifth line is an empty line (its content is ignored);
- after that, all objects and all attributes are listed, each on a line
- finally, the context is given as a combination of `.` and X, each row on its own line.

```
B

10
5

0
1
2
3
4
5
6
7
8
9
c
e
o
p
s
XX..X
..X.X
.X.X.
..XX.
XX..X
..XX.
XX...
..XX.
XX...
X.X.X
```

3.9 *SLF* data reader

py-galactic-reader-slf is a data reader plugin for *py-galactic*.

The file extension is `.slf`. The internal format is composed of 4 sections:

- the first section starts with a header containing the keyword `Lattice` surrounded by square brackets. It is followed by two lines containing the number of objects and the number of attributes;
- the second section starts with the keyword `Objects` surrounded by square brackets. It is followed by lines containing the names of the objects;
- the third section starts with the keyword `Attributes` surrounded by square brackets. It is followed by lines containing the names of the attributes;
- the fourth section starts with the keyword `Relation` surrounded by square brackets. It is followed by lines containing the binary relation expressed as a matrix composed of 0 and 1.

For example:

```
[Lattice]
10
5
[Objects]
0
1
2
3
4
5
6
7
8
9
[Attributes]
c
e
o
p
s
[Relation]
1 1 0 0 1
0 0 1 0 1
0 1 0 1 0
0 0 1 1 0
1 1 0 0 1
0 0 1 1 0
1 1 0 0 0
0 0 1 1 0
1 1 0 0 0
1 0 1 0 1
```

4 Explorers

Explorer files describe how a data file will be processed using a set of strategies. It's a file using the `yaml` format.

```
# ...
- !strategy.categorized.basic.Category
  # parameters of the strategy
- !strategy.numerical.basic.Normal
  # parameters of the strategy
# ...
```

Each tagged element is described using a dictionary containing:

- `arguments`: a list of arguments;
- `params`: a list of named parameters.

If the parameters of the element are only a list of arguments or a list of named parameters, you can use directly that list without the use of the ("`arguments`", "`params`") notation.

The components supported by the *py-galactic-core* engine comes either from the core library or from plugins.

4.1 Characteristics

4.1.1 *core*

The *core* library exposes 4 characteristic classes known to the console utilities by the following names:

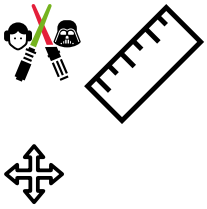
- `characteristic.core.Key` is used to describe an characteristic for a python object using array-like access;
- `characteristic.core.Property` is used to describe a characteristic for a python object using property-like access;
- `characteristic.core.Item` is used to describe a characteristic for a python object using array access;
- `characteristic.core.Slice` is used to describe a characteristic for a python object using slice access;

In the following table, the prefix `characteristic.core.` must be added to the names.

Name	Keyword parameters
Key	• name: to specify the characteristic name • `characteristic`: to specify an internal characteristic
Property	• name: to specify the characteristic name • `characteristic`: to specify an internal characteristic

Name	Keyword parameters
`Item`	• `nth`: to specify the characteristic number • `characteristic`: to specify an internal characteristic
`Slice`	• `start`: to specify the slice start • end: to specify the slice end • `characteristic`: to specify an internal characteristic

4.1.2 *logical* characteristic plugin



py-galactic-characteristic-logical is a plugin package for *py-galactic*.

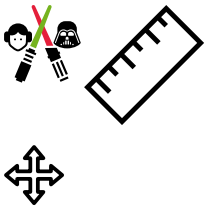
It exposes 1 characteristic classes known to the console utilities by the following name:

- `characteristic.logical.Bool`: to transform a characteristic to a boolean value;

In the following table, the prefix `characteristic.logical.` must be added to the names.

Name	Keyword Parameters
`Bool`	`characteristic`: a characteristic to transform

4.1.3 *categorized* characteristic plugin



py-galactic-characteristic-categorized is a plugin package for *py-galactic*.

It exposes 1 characteristic class known to the console utilities by the following name:

- `characteristic.categorized.Category`: to specify a simple category characteristic;

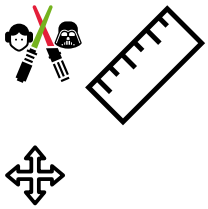
In the following table, the prefix `characteristic.categorized.` must be added to the names.

Name	Keyword parameters
`Category`	• `domain`: to specify the domain admitted by this category • `characteristic`: a characteristic containing the category • `label`: a label • `explanation`: an explanation using markdown formatting

For example, to use the category characteristic stored in a key characteristic named `class` and to compare it with the admitted domain `"Iris-setosa"`, `"Iris-versicolor"`, `"Iris-virginica"`, the following notation must be used in the *yaml* explorer file:

```
!characteristic.categorized.Category
domain: ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
characteristic: !characteristic.core.Key
  name: class
```

4.1.4 *numerical* characteristic plugin



py-galactic-characteristic-numerical is a plugin package for *py-galactic*.

It exposes 3 characteristic classes known to the console utilities by the following names:

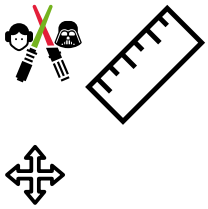
- `characteristic.numerical.Number`: to convert a characteristic to a number;
- `characteristic.numerical.Linear`: to apply a linear transformation to a number;
- `characteristic.numerical.Sum`: to compute a sum over numerical characteristics.

In the following table, the prefix `characteristic.numerical.` must be added to the names.

Name	Keyword parameters
`Number`	* `characteristic`: to specify the characteristic to convert
`Linear`	* `characteristic`: to specify the characteristic to linearize * `coefficient`: the coefficient of linearization

Name	Parameters
Sum	* components: to specify the characteristics to sum

4.1.5 *string* plugin



py-galactic-characteristic-string is a plugin package for *py-galactic*.

It exposes 1 characteristic class known to the console utilities by the following name:

`characteristic.string.String` used to hold string values.

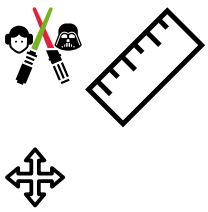
In the following table, the prefix `characteristic.string.` must be added to the names.

Name	Keyword Arguments
String	* characteristic: to specify the characteristic that hold the string * alphabet: to specify the alphabet of this characteristic

For example, to use the `String` characteristic stored in a key characteristic named "`string`" and `alphabet` "`abcdef`", the following notation must be used in the *yaml* explorer file:

```
!characteristic.string.String
characteristic: !characteristic.core.Key
  name: string
alphabet: "abcdef"
```

4.1.6 *chain* plugin



py-galactic-characteristic-chain is a plugin package for *py-galactic*.

It exposes 1 characteristic class known to the console utilities by the following name:

`characteristic.chain.Chain` used to hold chain values.

In the following table, the prefix `characteristic.chain.` must be added to the names.

Name	Keyword Arguments
`Chain`	* `characteristic`: to specify the characteristic that hold the chain * `alphabet`: to specify the alphabet of this characteristic

For example, to use the `Chain` characteristic stored in a key characteristic named "`chain`" and alphabet `{"A","B","C"}`, the following notation must be used in the *yaml* explorer file:

```
!characteristic.chain.Chain
characteristic: !characteristic.core.Key
  name: chain
alphabet: ["A","B","C"]
```

4.1.7 *sequence* plugin

py-galactic-characteristic-sequence is a plugin package for *py-galactic*.

It exposes 1 characteristic class known to the console utilities by the following name:

`characteristic.sequence.Sequence` used to hold sequence values.

In the following table, the prefix `characteristic.sequence.` must be added to the names.

Name	Keyword Arguments
`Sequence`	* `characteristic`: to specify the characteristic that hold the sequence * `alphabet`: to specify the alphabet of this characteristic

For example, to use the sequence predicate stored in a key characteristic named "`sequence`" and alphabet `{"A","B","C"}`, the following notation must be used in the *yaml* explorer file:

```
!characteristic.sequence.Sequence
characteristic: !characteristic.core.Key
  name: "sequence"
alphabet: ["A","B","C"]
```

4.2 Descriptions and predicates

4.2.1 *core*

4.2.1.1 Predicates The *core* library exposes 2 predicate classes known to the console utilities by the following names:

- `predicate.core.Member` is used to describe a predicate for a python object using membership-like access.
- `predicate.core.Equal` is used to describe a predicate for a python object using equality.

In the following table, the prefix `predicate.core.` must be added to the names.

Name	Keyword parameters
`Member`	• `name`: to specify the member name • `characteristic`: to specify an internal characteristic
`Equal`	• `name`: to specify the characteristic name • `data`: to specify the value to compare

For example:

```
!predicate.core.Member
  params:
    name: c
```

describes a membership-like characteristic named `c`.

For example:

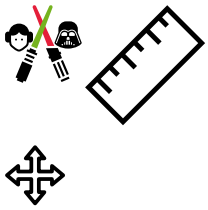
```
!predicate.core.Member
  characteristic:
    !characteristic.core.key
      name: v
  params:
    name: c
```

describe a membership-like characteristic named `c` accessible from the key-like parameter `v`. Thus, for the `yaml` data file:

```
1:
  v: [a,c]
  w: 2.3
2:
  v: [a,b]
  w: 1.2
3:
  v: [b,c]
  w: -5.6
```

The values of this characteristic for the individuals `1`, `2` and `3` will be `True`, `False` and `True`.

4.2.2 Classical *logical* description plugin



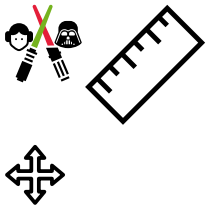
py-galactic-description-logical-classic is a plugin package for *py-galactic*.

4.2.2.1 Descriptions

This plugin defines 1 description spaces for boolean characteristics:

- `description.logical.classic.Boolean`: the classical one defined on one boolean predicate producing either the predicate itself or none predicate.

4.2.3 *logical* clause description plugin



py-galactic-description-logical-clause is a plugin package for *py-galactic*.

4.2.3.1 Predicates

It exposes 6 predicate classes known to the console utilities by the following names:

- `predicate.logical.clause.Not`: to get the *negation* of another boolean predicate;
- `predicate.logical.clause.And`: to get a logical *and* between other boolean predicates;
- `predicate.logical.clause.Or`: to get a logical *or* between other boolean predicates;
- `predicate.logical.clause.Equivalence`: to get a logical *equivalence* between other boolean predicates;
- `predicate.logical.clause.Nor`: to get a logical *nor* between other boolean predicates;
- `predicate.logical.clause.Nand`: to get a logical *nand* between ther boolean predicates.

In the following table, the prefix `predicate.logical.clause.` must be added to the names.

Name	Arguments
`Not`	a predicate to negate
`And`	a list of boolean predicates

Name	Arguments
Or	a list of boolean predicates
Equivalence	a list of boolean predicates
Nor	a list of boolean predicates
Nand	a list of boolean predicates

For example, to use the *negation* of a boolean predicate stored in a key membership characteristic named "prime", the following notation must be used in the *yaml* explorer file:

```
!predicate.logical.clause.Not
characteristic: !predicate.core.Member
  name: prime
```

For example, to use the logical *and* of two boolean predicates stored in key membership characteristic named "prime" and "odd", the following notation must be used in the *yaml* explorer file:

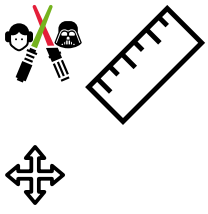
```
!predicate.logical.clause.And
- !predicate.core.Member
  name: "prime"
- !predicate.core.Member
  name: "odd"
```

4.2.3.2 Descriptions

This plugin defines 1 description space for boolean characteristics:

- `description.logical.clause.Logical`: the *logical* clause description defined on multiple boolean predicates producing clauses using predicates and their negations.

4.2.4 *categorized* description plugin



py-galactic-description-categorized-subset is a plugin package for *py-galactic*.

4.2.4.1 Predicates

It exposes 1 predicate classes known to the console utilities by the following name:

- `predicate.categorized.subset.Category`: to specify a simple category predicate;

In the following table, the prefix `predicate.categorized.subset.` must be added to the names.

Name	Keyword parameters
Category	* characteristic: to specify the characteristic containing the category * values: to specify the values admitted by this category * strict: to specify the strict property

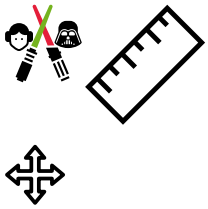
For example, to use the category predicate stored in a key characteristic named "class" and to compare it with the admitted values "Iris-setosa", "Iris-versicolor", the following notation must be used in the *yaml* explorer file:

```
!predicate.categorized.subset.Category
characteristic: !characteristic.categorized.Category
  characteristic: !characteristic.core.Key
    name: "class"
  domain: ["Iris-setosa", "Iris-versicolor", "Iris-virginica"]
  unknown: unspecified
values: ["Iris-setosa", "Iris-versicolor"]
```

4.2.4.2 Descriptions

This plugin exposes one description `description.categorized.subset.Category`.

4.2.5 *numerical* hull description plugin



py-galactic-description-numerical-hull is a plugin package for *py-galactic*.

4.2.5.1 Predicates

It exposes 1 predicate classes known to the console utilities by the following names:

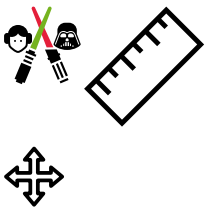
- `predicate.numerical.hull.Halfspace`: to test if a characteristic is in a half-space;

In the following table, the prefix `predicate.numerical.hull.` must be added to the names.

Name	Keyword parameters
HalfSpace	* characteristic: to specify the characteristic to test * threshold: to specify the threshold * strict: to specify the strict property * lower: to specify the lower property

4.2.5.2 Descriptions This plugin defines 1 description space `description.numerical.hull.Numerical` for numerical characteristics producing the equations of the convex hull defined by the individuals.

4.2.6 *string* match description plugin



py-galactic-description-string-match is a plugin package for *py-galactic*.

4.2.6.1 Predicates It exposes 2 predicate classes known to the console utilities by the following names:

- `predicate.string.SimpleMatch`: to specify a simple string predicate;
- `predicate.string.PrefixMatch`: to specify a string predicate with match to the prefix.

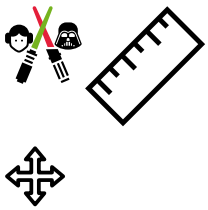
In the following table, the prefix `predicate.string.` must be added to the names.

Name	Keyword parameters
SimpleMatch	* characteristic: to specify the characteristic that hold the string * string: to specify the string of this characteristic
PrefixMatch	* characteristic: to specify the characteristic that hold the string * prefix: to specify the prefix of this characteristic

4.2.6.2 Descriptions This plugin defines 3 description space for string characteristics producing the regular expressions defined by the individuals.

- `description.string.SimpleMatch`
- `description.string.CompleteMatch`
- `description.string.PrefixMatch`

4.2.7 *string* distance description plugin



py-galactic-description-string-distance is a plugin package for *py-galactic*.

4.2.7.1 Predicates It exposes 1 predicate classes known to the console utilities by the following name:

- `predicate.string.DistanceMatch`: to specify a string predicate with match cardinality and distances.

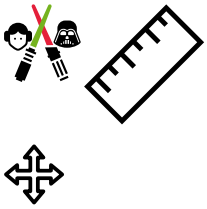
In the following table, the prefix `predicate.string.` must be added to the names.

Name	Keyword parameters
`DistanceMatch`	* `characteristic`: to specify the characteristic that hold the string * `string`: to specify the string of this characteristic * `cardinality_min` to specify the minimal number of match with this string * `cardinality_max` to specify the maximal number of match with this string * `distances` contains a set of minimal and maximal distances between the elements of the string

4.2.7.2 Descriptions This plugin defines 1 description space for string characteristics producing the regular expressions defined by the individuals.

- `description.string.DistanceMatch`

4.2.8 *chain* description plugin



py-galactic-description-chain-match is a plugin package for *py-galactic*.

4.2.8.1 Predicates It exposes 3 predicate classes known to the console utilities by the following names:

- `predicate.chain.SimpleMatch`: to specify a simple chain predicate;
- `predicate.chain.PrefixMatch`: to specify a chain predicate with defined by the prefix of the chain.
- `predicate.chain.AffixMatch`: to specify a chain predicate with defined by the affix of the chain. the affix is a subchain of the chain that appears in the first or in the end, or in the middle, the user has to specify the orientation and the position of start.

In the following table, the prefix `predicate.chain.` must be added to the names.

Name	Keyword parameters
`SimpleMatch`	* `characteristic`: to specify the characteristic that hold the chain * `chain`: to specify the chain of this characteristic
`PrefixMatch`	* `characteristic`: to specify the characteristic that hold the chain * `prefix`: to specify the prefix of the characteristic
`AffixMatch`	* `characteristic`: to specify the characteristic that hold the chain * `affix`: to specify the affix of the characteristic * `start`: to specify the start position of affix in the chain * `orientation`: to specify the orientation of the search, 1 to specify from the beginning, and -1 to specify from the end.

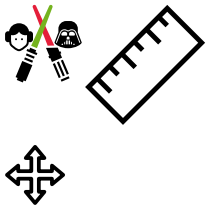
For example, to use the `Chain` characteristic stored in a key characteristic named "`chain`" and to match it with the chain "`chaintest`",the following notation must be used in the *yaml* explorer file:

```
!predicate.chain.SimpleMatch
characteristic: !characteristic.core.Key
  name: "chain"
chain: "chaintest"
```

4.2.8.2 Descriptions This plugin defines 3 description space for chain characteristics producing the common sub-chain, the common prefix, of the common affix defined by a collection of individuals.

- `description.chain.SimpleMatch`
- `description.chain.PrefixMatch`
- `description.chain.AffixMatch`

4.2.9 *sequence* match description plugin



py-galactic-description-sequence-match is a plugin package for *py-galactic*.

4.2.9.1 Predicates It exposes two predicate classes known to the console utilities by the following names:

- `predicate.sequence.SimpleMatch`: to specify a simple sequence predicate;

In the following table, the prefix `predicate.sequence.` must be added to the names.

Name	Keyword parameters
`SequenceMatch`	* `characteristic`: to specify the characteristic that hold the sequence * `sequence`: to specify the sequence of this characteristic

For example, to use the sequence predicate stored in a key characteristic named `"sequence"` and to match it with the sequence `"sequencetest"`, the following notation must be used in the *yaml* explorer file:

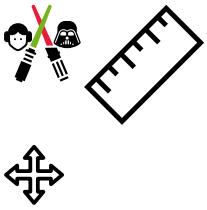
```
!predicate.sequence.SequenceMatch
params:
  characteristic: !characteristic.core.Key
    params:
      name: "sequence"
  sequence: "sequencetest"
```

4.2.9.2 Descriptions

This plugin defines one description spaces for sequence characteristics:

- `description.sequence.Simple`

4.2.10 *sequence* distance description plugin



py-galactic-description-sequence-distance is a plugin package for *py-galactic*.

4.2.10.1 Predicates

It exposes one predicate classes known to the console utilities by the following names:

- `predicate.sequence.DistanceMatch`: to specify a distance sequence predicate.

In the following table, the prefix `predicate.sequence.` must be added to the names.

Name	Keyword parameters
`DistanceMatch`	* `characteristic`: to specify the characteristic that hold the sequence * `sequence`: to specify the sequence of this characteristic * `distances` contains a set of minimal and maximal distances between the elements of the string

For example, to use the sequence predicate stored in a key characteristic named `"sequence"` and to match it with the sequence `"sequencetest"`, the following notation must be used in the *yaml* explorer file:

```
!predicate.sequence.SequenceMatch
params:
  characteristic: !characteristic.core.Key
    params:
      name: "sequence"
  sequence: "sequencetest"
```

4.2.10.2 Descriptions

This plugin defines one description spaces for sequence characteristics:

- `description.sequence.Distance`

The extensions/description-triadic-classic/description.rst file is empty.

4.3 Strategies

Strategies are used by the *galactic* core engine to produce new boolean attributes based on the individuals to partition. These attributes are in fact predicates defined on a description space.

4.3.1 *core* strategies

There are 2 strategies defined in the *core* library:

- `strategy.core.LimitFilter` which limits the successors to those whose measure does not exceed a limit;
- `strategy.core.SelectionFilter` which select the best or the worst successors.
- `strategy.core.Absurd` which select no successors.

They are a kind of filter strategy. They have to use a measure.

In the following tables, the prefix `strategy.core.` must be added to the names.

Name	Arguments
`LimitFilter`	a list of strategies
`SelectionFilter`	a list of strategies
`Absurd`	a list of descriptions

Name	Keyword parameter
`LimitFilter`	• `measure`: the measure used to select the strategies • `limit`: the limit not to be exceeded • `strict`: is the limit strict? • `lower`: is the limit a lower limit or an upper limit?
`SelectionFilter`	• `measure`: the measure used to select the strategies • `keep`: the number of best strategies (1 allows to keep only the very best ones, 2 allows to keep only the best and those whose measure is the second value, ...) • `maximize`: keep the best strategies or the worst • `decrease`: force the measure to increase (or decrease) • `strict`: decrease or increase strictly • `rate`: rate applied to the predecessor measure to force the decreasing (or increasing)

For example, to apply the "`SelectionFilter`" over the quantile numerical strategy for a numerical characteristic named "`sepal width`", the following notation must be used in the *yaml* explorer file:

```
!strategy.core.SelectionFilter
arguments:
  - !strategy.numerical.quantile.Quantile
    arguments:
      - !characteristic.numerical.Number
        characteristic: !characteristic.core.Key
          name: "sepal width"
    params:
      quantile: null
      lower: true
      count: null
params:
  measure: !measure.entropy.Entropy
    characteristic: !characteristic.core.Key
      name: "class"
  maximize: false
```

The extensions/strategy-logical-classic-basic/description.rst file is empty.

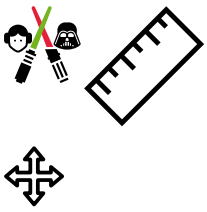
The extensions/strategy-logical-clause-basic/description.rst file is empty.

The extensions/strategy-categorized-subset-basic/description.rst file is empty.

The extensions/strategy-numerical-hull-basic/description.rst file is empty.

The extensions/strategy-numerical-hull-quantile/description.rst file is empty.

4.3.2 *string match basic* strategies



py-galactic-strategy-string--match-basic is a plugin package for *py-galactic*.

It exposes three strategies classes known to the console utilities by the following name:

- `strategy.string.basic.SimpleMatch`: to specify a simple string strategy.
- `strategy.string.basic.PrefixMatch`: to specify a string strategy using the common prefix.
- `strategy.string.basic.CompleteMatch`: to specify a string strategy using maximal subsequences.

In the following tables, the prefix `strategy.string.basic.` must be added to the names.

Name	Arguments
SimpleMatch	* characteristic: to specify the characteristic that hold the sequence. * length: to specify the length of the subsequence match * window: to specify a limit window to search for subsequences * gaps: to specify a limit gaps between elements of the subsequences
CompleteMatch	* characteristic: to specify the characteristic that hold the string.
PrefixMatch	* characteristic: to specify the characteristic that hold the string.

For example, to use the distance string strategy for a modal parameter named "class", and for length of match 3, and limit of 10, the following notation must be used in the *yaml* explorer file:

```
!strategy.string.basic.SimpleMatch
arguments:
  - !characteristic.core.Key
      name: "class"
params:
  length: 3
  window: 10
```

Note

This strategy can currently deal with only 1 modal characteristic.

4.3.3 *string distance basic* strategies

py-galactic-strategy-string-distance-basic is a plugin package for *py-galactic*.

It exposes three strategies classes known to the console utilities by the following name:

- strategy.string.distance.basic.DistanceMatch: to specify a string strategy that uses cardinality and distances between elements of the match.

In the following tables, the prefix strategy.string.distance.basic. must be added to the names.

Name	Arguments
DistanceMatch	* characteristic: to specify the characteristic that hold the string

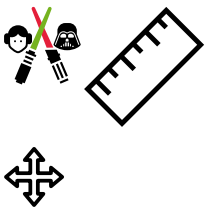
For example, to use the distance string strategy for a modal parameter named "class", and for length of match 3, and limit of 10, the following notation must be used in the *yaml* explorer file:

```
!strategy.string.distance.basic.DistanceMatch
arguments:
  - !characteristic.core.Key
      name: "class"
params:
  length: 3
  limit: 10
```

Note

This strategy can currently deal with only 1 modal characteristic.

4.3.4 *chain match basic* strategies



py-galactic-strategy-chain-match-basic is a plugin package for *py-galactic*.

It exposes three strategies classes known to the console utilities by the following name:

- strategy.chain.basic.SimpleChainMatch: to specify a simple chain strategy.
- strategy.chain.basic.PrefixChainMatch: to specify a chain strategy using common prefix.

In the following tables, the prefix strategy.chain.basic. must be added to the names.

Name	Arguments
SimpleMatch	* characteristic: to specify the characteristic that hold the chain * length: to specify the length of the match.
PrefixMatch	* characteristic: to specify the characteristic that hold the chain

For example, to use the distance chain strategy for a modal parameter named "class", and for length of match 3, and limit of 10, the following notation must be used in the *yaml* explorer file:

```
!strategy.chain.basic.SimpleMatch
arguments:
  - !characteristic.core.Key
      name: "class"
params:
  length: 3
```

Note

This strategy can currently deal with only 1 modal characteristic.

4.3.5 *sequence match basic* strategy

py-galactic-strategy-sequence-match-basic is a plugin package for *py-galactic*.

It exposes a strategy classe known to the console utilities by the following name:

- `strategy.sequence.basic.SimpleMatch`: to specify a simple sequence strategy that use the subsequence of the given length, else of length=2.
- `strategy.sequence.basic.PrefixMatch`: to specify a prefix sequence strategy that use the common prefix.

In the following tables, the prefix `strategy.sequence.basic.` must be added to the names.

Name	Arguments
`SimpleMatch`	* `characteristic`: to specify the characteristic that hold the sequence. * `length`: to specify the length of the subsequence match
`PrefixMatch`	* `characteristic`: to specify the characteristic that hold the sequence.

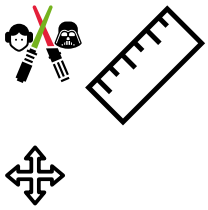
For example, to use the basic sequence strategy for a modal parameter named "`sequence`" and a length=4 of subsequence match, the following notation must be used in the *yaml* explorer file:

```
- !strategy.sequence.basic.SimpleMatch
  arguments:
    - !characteristic.core.Key
        name: "sequence"
  params:
    length: 4
```

Note

This strategy can currently deal with only 1 modal characteristic.

4.3.6 *sequence distance basic* strategy



py-galactic-strategy-sequence-distance-basic is a plugin package for *py-galactic*.

It exposes a strategy classe known to the console utilities by the following name:

- `strategy.sequence.basic.DistanceMatch`: to specify a simple sequence strategy that use the subsequence of the given length, else of length=2.

In the following tables, the prefix `strategy.sequence.basic.` must be added to the names.

Name	Arguments
`DistanceMatch`	* `characteristic`: to specify the characteristic that hold the sequence. * `length`: to specify the length of the subsequence match

For example, to use the basic sequence strategy for a modal parameter named "`sequence`" and a length=4 of subsequence match, the following notation must be used in the *yaml* explorer file:

```
- !strategy.sequence.basic.DistanceMatch
  arguments:
    - !characteristic.core.Key
          name: "sequence"
  params:
    length: 4
```

Note

This strategy can currently deal with only 1 modal characteristic.

The extensions/strategy-triadic-classic-basic/description.rst file is empty.

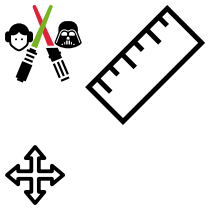
4.4 Measures

4.4.1 *core* measures

There are 3 measures in the *core* library:

- `measure.core.Cardinality` which computes the cardinality of the successor;
- `measure.core.PredecessorCardinality` which computes the cardinality of the predecessor;
- `measure.core.Confidence` which computes the cardinality ratio between the successor and the predecessor.

4.4.2 *entropy* measure



py-galactic-measure-entropy is a plugin package for *py-galactic*.

It exposes 1 measure class known to the console utilities by the following name:

- `measure.entropy.Entropy`: to compute the entropy of a successor.

In the following table, the prefix `measure.entropy.` must be added to the names.

Name	Keyword parameters
`Entropy`	* `characteristic`: to specify the characteristic containing the category * `theta`: the float coefficient applied to individuals kept in the successor

For example, to use the entropy measure of a key characteristic named "`class`" for a characteristic named "sepal width" using the quantile strategy, the following notation must be used in the *yaml* explorer file:

```
!strategy.core.SelectionFilter
arguments:
  - !strategy.numerical.quantile.Quantile
    arguments:
      - !characteristic.numerical.Number
        characteristic: !characteristic.core.Key
          name: "sepal width"
    params:
      quantile: null
      lower: true
      count: null
params:
  measure: !measure.entropy.Entropy
    characteristic: !characteristic.core.Key
      name: "class"
```

```
  best: false
```

List of characteristics

List of predicates

List of descriptions

List of strategies

List of measures