GALACTIC: towards a generic and scalable platform for complex and heterogeneous data using Formal Concept Analysis

Christophe Demko[1,*], Salah Eddine Boukhetta[1], Jérémy Richard[1], Guillaume Savarit[1], Karell Bertet[1], Cyril Faucher[1] and Damien Mondou[1]

[1]*Laboratory L3i, La Rochelle University, La Rochelle, France*

Abstract

In a recent paper, we presented a new pattern discovery algorithm, NextPriorityConcept, in order to take into account complex and heterogeneous data using Formal Concept Analysis. We implemented this algorithm and developed a `python 3` library whose acronym `GALACTIC` means **Ga**lois **La**ttices, **C**oncept **T**heory, **I**mplicational systems and **C**losures. It is opened to the community using a BSD-3 license and its architecture allows the writing of plugins to take into account new datatypes. In this article we will present the architecture of our software solution, we will explain how to add new plugins to the core of our system by giving the UML diagram of each kind of plugins and we will give some examples of plugins developed within our team.

Keywords

Formal Concept Analysis, Pattern Structures, Complex Data, Heterogeneous data

1. Introduction

`GALACTIC` is a new platform for computing patterns from heterogeneous and complex data that extend the approach of pattern structures [1] and logical concept analysis [2]. It's a development platform for a generic implementation of the NextPriorityConcept[3] algorithm allowing easy integration of new plugins for characteristics, descriptions, strategies and meta-strategies. The `GALACTIC` eco-system is organized with:

- A **core** which implements the NextPriorityConcept algorithm and a lot of tools for visualizing lattices and reduced contexts in python notebooks;
- A set of **characteristic plugins** defining new types of data;
- A set of **description plugins** defining new types of descriptions and their associated predicates;
- A set of **strategy plugins** defining new types of strategies for a given characteristic;

Published in Pablo Cordero, Ondrej Kridlo (Eds.): The 16th International Conference on Concept Lattices and Their Applications, CLA 2022, Tallinn, Estonia, June 20–22, 2022, Proceedings, pp. 191–198.

*Corresponding author.

christophe.demko@univ-lr.fr (C. Demko); salah.boukhetta@univ-lr.fr (S. Boukhetta); jeremy.richard2@univ-lr.fr (J. Richard); guillaume.savarit1@univ-lr.fr (G. Savarit); karell.bertet@univ-lr.fr (K. Bertet); cyril.faucher@univ-lr.fr (C. Faucher); damien.mondou@univ-lr.fr (D. Mondou)

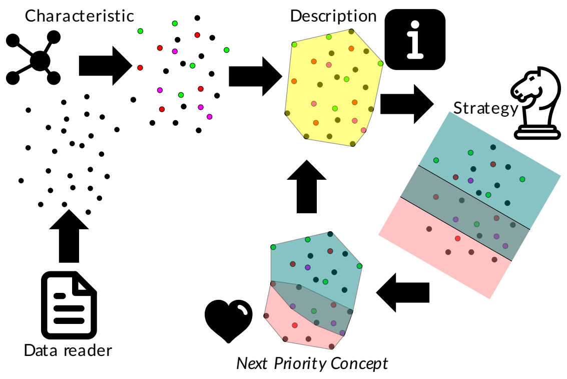

Figure 1: the NextPriorityConcept algorithm process

- A set of **measure plugins** useful for the filter meta-strategies;
- A set of **data reader plugins** allowing `GALACTIC` to read any type of data file;
- A set of **applications** using the core library and the different plugins;
- A set of **localization plugins** for translating the different applications.

Each plugin must register with the core library by declaring an entry point in the configuration file of the `setuptools`[1] (`setup.py`) named `py_galactic_extension`. The declared function informs the library that a new extension is available.

The construction of the lattice is carried out as shown in Fig. 1: the data are read from a file; characteristics are extracted; a description is produced for each concept; strategies generate selectors for the exploration of potential new concepts and the NextPriorityConcept algorithm selects the predecessors and maintains the lattice structure.

2. `GALACTIC` platform

2.1. Data reader plugins

A data reader plugin is responsible for reading a file according to its extension and producing the list of individuals of the population. It must implement two class methods:

- `read(cls, data_file: TextIO): Iterable[Any]` that reads a file and produces an iterable of objects.
- `extensions(cls): Iterator[str]` that produces the list of file extensions supported by this plugin.

[1]https://pypi.org/project/setuptools/

2.2. Characteristic plugins

Each concept is composed of a subset of objects together with a set of predicates describing them, each predicate being specific to one type of characteristics. Such generic use of predicates makes it possible to consider heterogeneous data as input, i.e. numeric, discrete or more complex data.

A characteristic plugin (cf Fig. 3 in appendix) is responsible of extracting a value from a python object. It must implement the `__call__` magic method which will be applied to each individual of the population. This method should return the characteristic of the individual. It is also reasonable to implement the other magic methods `__eq__`, `__hash__`, `__str__`.

2.3. Description plugins

The algorithm introduces the notion of *description* δ as an application to provide predicates describing a set of objects A according to their characteristics, that corresponds to a concept $(A, \delta(A))$. At each iteration, predicates describing the objects A of the current concept are computed "on the fly" by a specific treatment for each type of characteristics, depending on whether it is numeric, discrete or more complex, and the final description δ is the union of these predicates. In order to obtain a lattice, the description must verify $\delta(A) \sqsubseteq \delta(A')$ for $A' \subseteq A$. NextPriorityConcept computes the concept lattice of the context composed of individuals in rows and predicates issued from the description in columns[3].

A description plugin (cf Fig. 4 in appendix) is used to describe a collection of individuals using a set of predicates. It usually defines a new predicate class and a new description class. The description class is responsible of calculating a set of descriptors representing the convex hull of a collection of individuals. These descriptors must be predicates that describe half-spaces on the set of individuals. The convex hull is represented by the intersection of the set of descriptors.

2.4. Strategy plugins

The algorithm also introduces the notion of *strategy* σ to provide selectors generating the predecessors of a concept $(A, \delta(A))$. The selectors propose a way to refine or cut the description $\delta(A)$. The purpose of a strategy plugin is then to produce a set of selectors restricting the set of individuals in order to obtain new potential concepts containing fewer individuals. The NextPriorityConcept algorithm selects among the set of selectors produced by the strategies that generates new effective concepts (i.e. with a reduction of individuals). The lattice property is maintained using propagation of cross and residuals constraints and a level by level generation of concepts using a priority queue[3].

2.4.1. Basic strategies

A basic strategy plugin proposes selectors to the NextPriorityConcept algorithm. These selectors act as cut into the set of individuals. A basic strategy must be initialized with a description and must implement the `selectors` method (cf Fig. 5 in appendix).

2.4.2. Meta strategies and measure plugins

The core of the **GALACTIC** library defines two meta-strategies that act as filters for other strategies:

- `LimitFilter` which selects predecessors whose measure is above/below a threshold;
- `SelectionFilter` which selects the best/worst predecessors relatively to a measure.

They are initialized with a set of strategies and with a measure. A measure is a class for measuring a predicate on a concept. It must implement the `__call__` magic method with a concept and a predicate as parameters (cf FIG. 6 in appendix).

3. Some particular plugins

We implemented some basic description and strategies plugins to manage boolean, categorical, numerical characteristic, but also more complex characteristics such as temporal sequences[4].

3.1. Logical description and strategy plugin

When characteristics are of the same type, NEXTPRIORITYCONCEPT offers the possibility to process them separately or together. An immediate way to process with several characteristics together would be to merge the predicates obtained in individual cases, both for the descriptions and for the strategies. But it is possible to obtain more relevant predicates by using a specific process.

For boolean characteristics for example, the classical FCA approach describes a set of objects A thanks to the set of attributes $B = \{x \;:\; a \in A \text{ and } a \text{ possesses } x\}$ and the strategy would consider as selectors the set of all the attributes that are not in B. These two sets described by predicates of the form "possesses attribute x" would respectively corresponds to $\delta(A)$ and $\sigma(A)$.

It is possible to consider other descriptions of A by the introduction of negative attributes, for example, the disjunction of clauses or its minimalization.

$$\bigvee_{a\in A} \bigwedge \begin{cases} x & \text{if } a \text{ possesses } x \\ \bar{x} & \text{otherwise} \end{cases} \tag{1}$$

We defined plugins allowing to manage several boolean variables and their negations in the same description using the well-known Quine-McCluskey's dual algorithm (or the method of prime implicants) that computes the minimization of a set of disjunctive clauses with a time complexity in $O(3^n \log n)$[5] where n is the number of attributes. For example, using the context defined in FIG 2, the concept containing the even numbers is defined by $\{\bar{o}, e, c + \bar{s}, c + p\}$ and the concept containing the odd numbers is defined by $\{\bar{e}, o, \bar{c} + s, p + s\}$.

The basic strategy associated to that description consists in simply adding a boolean variable or its negation to a concept to generate its potential predecessors.

#	**c**omposite	**e**ven	**o**dd	**p**rime	**s**quare
0	⊠	⊠			⊠
1			⊠		⊠
2		⊠		⊠	
3			⊠	⊠	
4	⊠	⊠			⊠
5			⊠	⊠	
6	⊠	⊠			
7			⊠	⊠	
8	⊠	⊠			
9	⊠		⊠		⊠

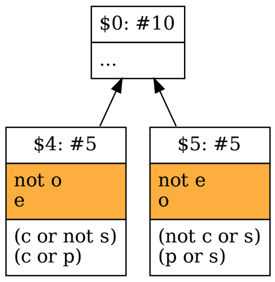

Figure 2: Digits example context and a fragment of the concept lattice with the even number and odd number concepts

3.2. Numerical description and strategy plugin

As already proposed in pattern structure theory[6], it is possible to describe a set of points A of $\mathbb{R}^n$ using the usual convex hull represented by a polytope, the description $\delta(A)$ is then composed of predicates describing the borders of the convex hull. For points in two and three dimensions, output-sensitive algorithms are known to compute the convex hull in time $O(n \log n)$, where n is the number of points.

We defined a description calculating the convex hull, and two strategies for cutting the n-dimensional space by projecting the data on the main axis of inertia:

- we can define two predicates, $x \geq m - \alpha\sigma$ and $x \leq m + \alpha\sigma$ where α is a parameter of this strategy, m is the mean and σ is the standard deviation;
- we can define a set of similar predicates by using the quantiles of the numerical data. The number of quantiles is a parameter of the strategy.

3.3. Entropy measure plugin

The measure plugins are mainly used in the `LimitFilter` and `SelectionFilter` meta-strategies. They must give a numerical value evaluating a predicate in relation to a concept. In the core, we developed the confidence, support and cardinality measures and we defined a plugin for evaluating the entropy relatively to a categorical characteristic, i.e. a class attribute.

For a given concept (A, P), a predicate p that select only some individuals from A and a categorical characteristic Ω, the entropy $H_\theta(A, p, \Omega)$ is defined by:

$$\begin{aligned}
H_\theta(A, p, \Omega) &= \theta H(\{a \in A | p(a)\}, \Omega) + (1 - \theta) H(\{a \in A | \neg p(a)\}, \Omega) \\
H(A', \Omega) &= \sum_{\omega \in D_\Omega} p_\omega(A', \Omega) log(p_\omega(A', \Omega)) \\
p_\omega(A', \Omega) &= \frac{|\{a \in A' | \Omega(a) = \omega\}|}{|A'|}
\end{aligned}$$

where θ is a $[0,1]$ parameter and $p_\omega(A', \Omega)$ is the ratio of individual in A' that have their characteristic Ω equal to ω.

4. Conclusion

In this article, we described the software architecture of our **GALACTIC** platform that implements the NextPriorityConcept algorithm to generate formal concepts for heterogeneous and complex data.

We explained the different types of plugins (reader, characteristic, description, strategy, measure) and we specified the methods to implement new ones. Then we presented some interesting plugins that we had developed, that can be used either to mine data or to develop new plugins. We are currenly working for plugins on time series.

Inspired from pattern structures, the generated lattices are often smaller with more relevant concepts thanks to the notion of strategy, and a user driven approach to discover patterns is possible, in an interactive way for instance. The **GALACTIC** engine is data agnostic since it only handles predicates generated by strategies and descriptions for each type of characteristics.

We are currently working on adding a graphical interface and the possibility of changing strategies on demand. This will ease the use of the **GALACTIC** platform for non-IT users. Future work will also be devoted to the extraction of minimal generators and logical rules from the concepts.

References

[1] B. Ganter, S. Kuznetsov, Pattern structures and their projections, in: LNCS of International Conference on Conceptual Structures (ICCS'01), 2001, pp. 129–142.

[2] S. Ferré, O. Ridoux, A logical generalization of formal concept analysis, LNCS 1867 (2000) 371–384.

[3] Ch. Demko, K. Bertet, C. Faucher, J.-F. Viaud, S. O. Kuznetsov, NextPriorityConcept: A new and generic algorithm computing concepts from complex and heterogeneous data, Theoretical Computer Science 845 (2020) 1–20. doi:10.1016/j.tcs.2020.08.026.

[4] S. E. Boukhetta, C. Demko, K. Bertet, J. Richard, C. Cayèré, Temporal sequence mining using fca and galactic, in: International Conference on Conceptual Structures, Springer, 2021, pp. 185–199.

[5] W. V. O. Quine, The problem of simplifying truth functions, The American Mathematical 59 (1952) 521–531.

[6] M. Kaytoue, V. Codocedo, A. Buzmakov, J. Baixeries, S. Kuznetsov, A. Napoli, Pattern structures and concept lattices for data mining and knowledge processing, in: In Proceedings of ECML-PKDDl, 2015, pp. 227–231.

Appendix

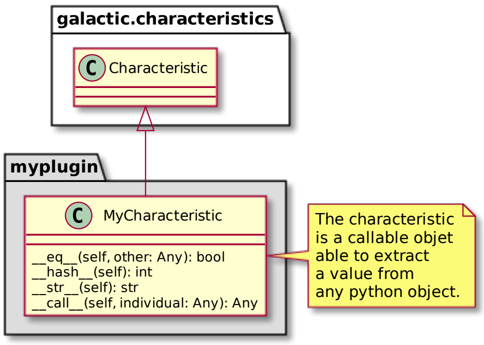

Figure 3: characteristic plugin

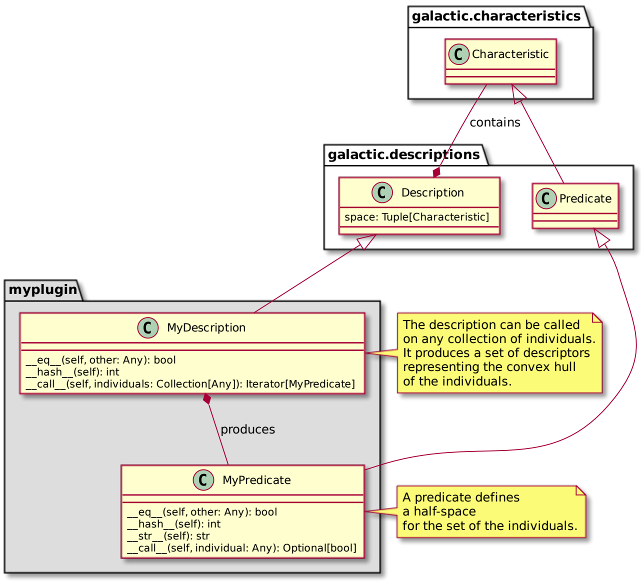
galactic.characteristics
Characteristic
contains
galactic.descriptions
Description
space: Tuple[Characteristic]
Predicate
myplugin
MyDescription
__eq__(self, other: Any): bool
__hash__(self): int
__call__(self, individuals: Collection[Any]): Iterator[MyPredicate]
The description can be called
on any collection of individuals.
It produces a set of descriptors
representing the convex hull
of the individuals.
produces
MyPredicate
__eq__(self, other: Any): bool
__hash__(self): int
__str__(self): str
__call__(self, individual: Any): Optional[bool]
A predicate defines
a half-space
for the set of the individuals.

Figure 4: description plugin

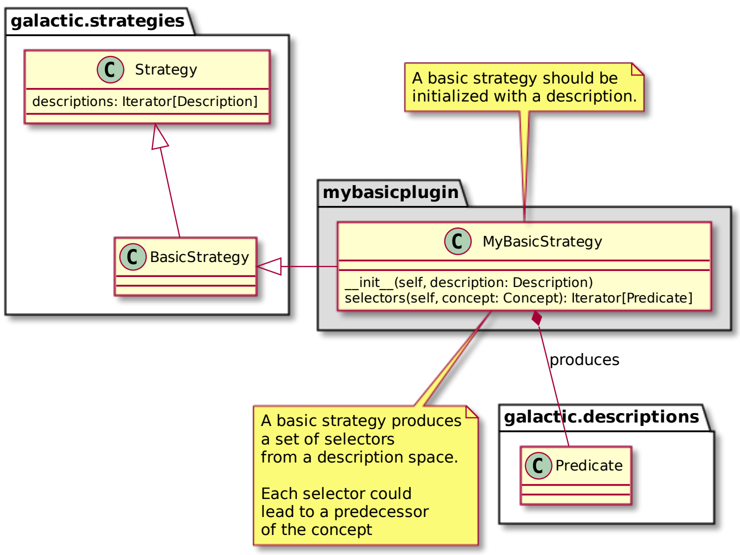
galactic.strategies
Strategy
descriptions: Iterator[Description]
BasicStrategy
A basic strategy should be initialized with a description.
mybasicplugin
MyBasicStrategy
__init__(self, description: Description)
selectors(self, concept: Concept): Iterator[Predicate]
produces
A basic strategy produces a set of selectors from a description space.
Each selector could lead to a predecessor of the concept
galactic.descriptions
Predicate

Figure 5: basic strategy plugin

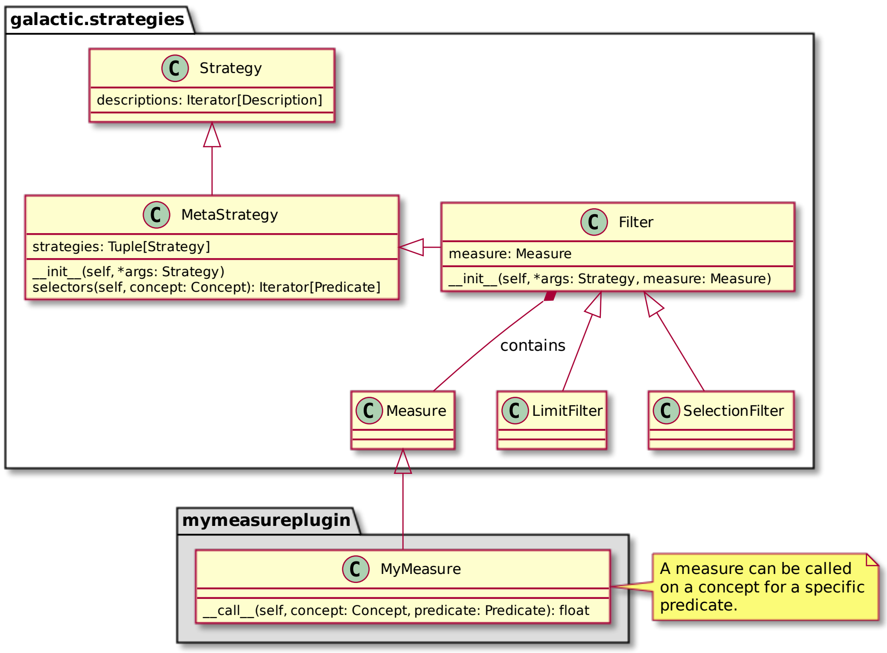
galactic.strategies
Strategy
descriptions: Iterator[Description]
MetaStrategy
strategies: Tuple[Strategy]
__init__(self, *args: Strategy)
selectors(self, concept: Concept): Iterator[Predicate]
Filter
measure: Measure
__init__(self, *args: Strategy, measure: Measure)
contains
Measure
LimitFilter
SelectionFilter
mymeasureplugin
MyMeasure
__call__(self, concept: Concept, predicate: Predicate): float
A measure can be called on a concept for a specific predicate.

Figure 6: measure plugin