The NextPriorityConcept Algorithm

A generic algorithm computing concepts from heterogeneous and complex data

The Galactic Organization <contact@thegalactic.org>

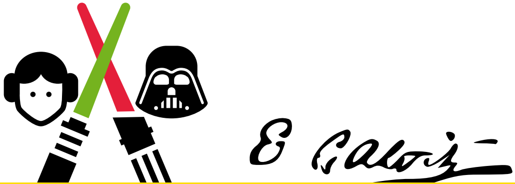
2018-2022 [1]

Motivations

- data scientist driven pattern mining;

GALACTIC stands for

GAlois
LAttices,
Concept
Theory,
Implicational systems and
Closures.

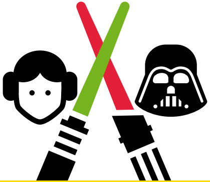


Motivations

- data scientist driven pattern mining;
- consideration of heterogeneous and complex data;

GALACTIC stands for

GAlois
LAttices,
Concept
Theory,
Implicational systems and
Closures.

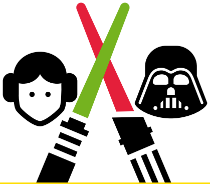


Motivations

- data scientist driven pattern mining;
- consideration of heterogeneous and complex data;
- generation of implication rules;

GALACTIC stands for

GAlois
LAttices,
Concept
Theory,
Implicational systems and
Closures.

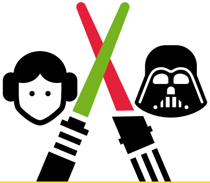


Motivations

- data scientist driven pattern mining;
- consideration of heterogeneous and complex data;
- generation of implication rules;
- extracted information size adapted to the goals.

GALACTIC stands for

GAlois
LAttices,
Concept
Theory,
Implicational systems and
Closures.



Founding ideas

Founding ideas

- inspired by the Bordat algorithm;

Founding ideas

Founding ideas

- inspired by the Bordat algorithm;
- use of a priority queue;

Founding ideas

Founding ideas

- inspired by the Bordat algorithm;
- use of a priority queue;
- use of first order monadic predicates;

Founding ideas

Founding ideas

- inspired by the Bordat algorithm;
- use of a priority queue;
- use of first order monadic predicates;
- constraint propagation mecanism.

Founding ideas

Founding ideas

- inspired by the Bordat algorithm;
- use of a priority queue;
- use of first order monadic predicates;
- constraint propagation mecanism.

Founding ideas

Founding ideas

- inspired by the Bordat algorithm;
- use of a priority queue;
- use of first order monadic predicates;
- constraint propagation mecanism.

arXiv

https://arxiv.org/abs/1912.11038

Theoretical Computer Sciences

https://doi.org/10.1016/j.tcs.2020.08.026

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A : b \in M \setminus B \right\}$$

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A : b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta)\rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A \; : b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A', B') ∈ LP do
            Add (A', B') to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A : b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
    end
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A \;:\; b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta)\rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A', B') ∈ LP do
            Add (A', B') to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
    end
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A : b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
        A′ ← β(b) ∩ A ;
    end
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A \; : \; b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
        A′ ← β(b) ∩ A ;
    end
end
```

La Rochelle Université

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A \; : b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
        A′ ← β(b) ∩ A ;
        if A′ maximal in L then Add A′ to L;
    end
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A : b \in M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
        A′ ← β(b) ∩ A ;
        if A′ maximal in L then Add A′ to L;
    end
end
```

Bordat algorithm as basis

A dual version of Bordat theorem

There is a bijection between the immediate predecessors of a concept (A, B) and the inclusion maximal subsets of the family:

$$\left\{ \beta(b) \cap A \ : b \in \ M \setminus B \right\}$$

Concepts($\langle G, M, (\alpha, \beta) \rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q
        end
    end
end
```

Immediate-Predecessors((A, B))

```
begin
    L ← ∅ ;
    forall b ∈ M \ B do
        A′ ← β(b) ∩ A ;
        if A′ maximal in L then Add A′ to L;
    end
    LP ← ∅ ;
    forall A′ ∈ L do
        Add (A′, α(A′)) to LP
    end
    return LP
end
```

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D))

begin
  $L \leftarrow \emptyset$;
  forall $b \in M \setminus B$ **do**
    $A' \leftarrow \beta(b) \cap A$;
    if A' *maximal in* L **then** Add A' to L;
  end
  $LP \leftarrow \emptyset$;
  forall $A' \in L$ **do**
    Add $(A', \alpha(A'))$ to LP;
  end
  return LP
end

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

begin
  $L \leftarrow \emptyset$;
  forall $b \in$ $M \setminus B$ **do**
    $A' \leftarrow \beta(b) \cap A$;
    if A' *maximal in* L **then** Add A' to L;
  end
  $LP \leftarrow \emptyset$;
  forall $A' \in L$ **do**
    Add $(A', \alpha(A'))$ to LP;
  end
  return LP
end

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

begin
  $L \leftarrow \emptyset$;
  forall $b \in$ $\sigma(A)$ **do**
    $A' \leftarrow \beta(b) \cap A$;
    if A' *maximal in* $L \wedge A' \subset A$ **then** Add A' to L;
  end
  $LP \leftarrow \emptyset$;
  forall $A' \in L$ **do**
    Add $(A', \alpha(A'))$ to LP;
  end
  return LP
end

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

begin
  $L \leftarrow \emptyset$;
  forall $b \in$ $\sigma(A)$ **do**
    $A' \leftarrow \beta(b) \cap A$;
    if A' *maximal in* $L \wedge A' \subset A$ **then** Add A' to L;
  end
  $LP \leftarrow \emptyset$;
  forall $A' \in L$ **do**
    Add $(A', \alpha(A'))$ to LP;
  end
  return LP
end

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

```
begin
    L ← ∅ ;
    forall b ∈ σ(A) do
        A′ ← β(b) ∩ A ;
        if A′ maximal in L ∧ A′ ⊂ A then Add A′ to L;
    end
    LP ← ∅ ;
    forall A′ ∈ L do
        Add (A′, α(A′)) to LP;
    end
    return LP
end
```

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Constraints

Constraints are needed to ensure that meet are correctly computed.

Constraints associate attributes $\mathcal{C}[A]$ to each subset $A \subseteq G$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

begin
- $L \leftarrow \emptyset$;
- **forall** $b \in$ $\sigma(A)$ $\cup$ $\mathcal{C}[A]$ **do**
  - $A' \leftarrow \beta(b) \cap A$;
  - **if** A' *maximal in* $L \wedge A' \subset A$ **then** Add A' to L;
- **end**
- $LP \leftarrow \emptyset$;
- **forall** $A' \in L$ **do**
  - Add $(A', \alpha(A'))$ to LP;
- **end**
- **return** LP

end

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Constraints

Constraints are needed to ensure that meet are correctly computed.

Constraints associate attributes $\mathcal{C}[A]$ to each subset $A \subseteq G$.

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

begin
- $L \leftarrow \emptyset$;
- **forall** $b \in$ $\sigma(A)$ $\cup$ $\mathcal{C}[A]$ **do**
  - $A' \leftarrow \beta(b) \cap A$;
  - **if** A' *maximal in* L $\wedge A' \subset A$ **then** Add A' to L;
- **end**
- $LP \leftarrow \emptyset$;
- **forall** $A' \in L$ **do**
  - Add $(A', \alpha(A'))$ to LP;
  - Compute the cross and residual constraints $\mathcal{C}[A']$
- **end**
- **return** LP

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Constraints

Constraints are needed to ensure that meet are correctly computed.

Constraints associate attributes $\mathcal{C}[A]$ to each subset $A \subseteq G$.

La Rochelle Université

Selection of attributes: a strategy σ

Definition

Instead of all the possibles attributes in $M \setminus B$, we only consider some attributes, given by a strategy. A strategy σ is an application from 2^G to 2^M which associates a subset of selected attributes $\sigma(A) \subseteq M$ to every $A \subseteq G$.

Immediate-Predecessors((A, D) $,\sigma$)

begin
- $L \leftarrow \emptyset$;
- **forall** $b \in \sigma(A) \cup \mathcal{C}[A]$ **do**
  - $A' \leftarrow \beta(b) \cap A$;
  - **if** A' *maximal in* $L \wedge A' \subset A$ **then** Add A' to L;
- **end**
- $LP \leftarrow \emptyset$;
- **forall** $A' \in L$ **do**
  - Add $(A', \alpha(A'))$ to LP;
  - Compute the cross and residual constraints $\mathcal{C}[A']$
- **end**
- **return** LP

Selected attributes P

The set of selected attributes is denoted P. We denote (A, D) a concept of $\langle G, P, I_P \rangle$.

Constraints

Constraints are needed to ensure that meet are correctly computed.

Constraints associate attributes $\mathcal{C}[A]$ to each subset $A \subseteq G$.

Selection of concepts: a priority queue

Concepts($\langle G, M, (\alpha, \beta)\rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q;
        end
    end
end
```

Selection of concepts: a priority queue

Concepts($\langle G, M, (\alpha, \beta)\rangle$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, B) ← Q.pop();
        produce (A, B);
        LP ← Immediate-Predecessors((A, B));
        forall (A′, B′) ∈ LP do
            Add (A′, B′) to Q;
        end
    end
end
```

Strategy

The strategy σ is given as input of the main algorithm.

Selection of concepts: a priority queue

Concepts$(\langle G, M, (\alpha, \beta)\rangle, \sigma)$

begin
  top $\leftarrow (G, \alpha(G))$;
  Add top to a queue Q;
  while Q **not empty do**
    $(A, D) \leftarrow Q$.pop();
    produce (A, D);
    $LP \leftarrow$ Immediate-Predecessors$((A, D), \sigma)$;
    forall $(A', D') \in LP$ **do**
      Add (A', D') to Q;
    end
  end
end

Strategy

The strategy σ is given as input of the main algorithm.

Selection of concepts: a priority queue

Concepts($\langle G, M, (\alpha, \beta)\rangle$ $,\sigma$)

```
begin
    top ← (G, α(G));
    Add top to a queue Q;
    while Q not empty do
        (A, D) ← Q.pop();
        produce (A, D);
        LP ← Immediate-Predecessors((A, D) ,σ);
        forall (A′, D′) ∈ LP do
            Add (A′, D′) to Q;
        end
    end
end
```

Strategy

The strategy σ is given as input of the main algorithm.

The priority queue Q

We use a priority queue according to the support of concepts to ensure that concepts are generated level by level, i.e. each concept is generated before its predecessors.

Selection of concepts: a priority queue

Concepts($\langle G, M, (\alpha, \beta)\rangle$ $,\sigma$)

```
begin
    top ← (G, α(G));
    Add ( |G| , top) to a priority queue Q;
    while Q not empty do
        (A, D ) ← Q.pop();
        produce (A, D );
        LP ← Immediate-Predecessors((A, D ) ,σ );
        forall (A′, D′ ) ∈ LP do
            Add ( |A′| ,(A′, D′ )) to Q;
        end
    end
end
```

Strategy

The strategy σ is given as input of the main algorithm.

The priority queue Q

We use a priority queue according to the support of concepts to ensure that concepts are generated level by level, i.e. each concept is generated before its predecessors.

Selection of concepts: a priority queue

Concepts($\langle G, M, (\alpha, \beta)\rangle$ $,\sigma$)

begin
  top $\leftarrow (G, \alpha(G))$;
  Add ($|G|$, top) to a priority queue Q;
  while Q **not empty do**
    $(A,$ D $) \leftarrow Q.\text{pop}()$;
    produce $(A,$ D $)$;
    $LP \leftarrow$ Immediate-Predecessors($(A,$ D $)$ $,\sigma$);
    forall $(A',$ D' $) \in LP$ **do**
      Add ($|A'|$, $(A',$ D' $)$) to Q;
    end
  end
end

Strategy

The strategy σ is given as input of the main algorithm.

The priority queue Q

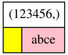
We use a priority queue according to the support of concepts to ensure that concepts are generated level by level, i.e. each concept is generated before its predecessors.

Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \text{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept



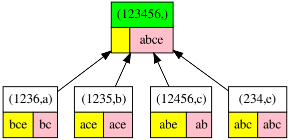
La Rochelle Université

Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept



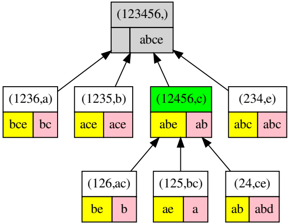
La Rochelle Université

Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

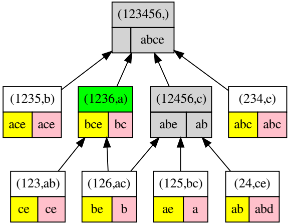


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

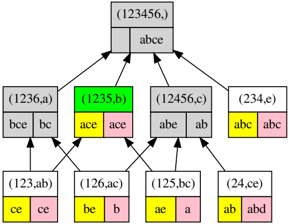


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \text{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

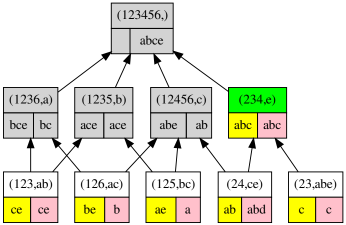


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

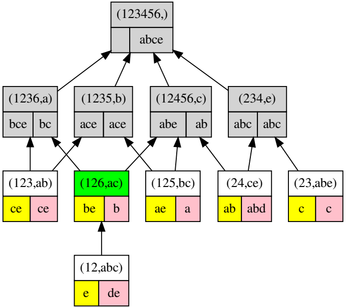


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

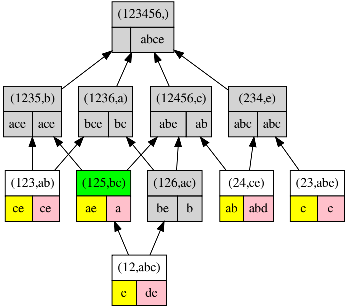


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

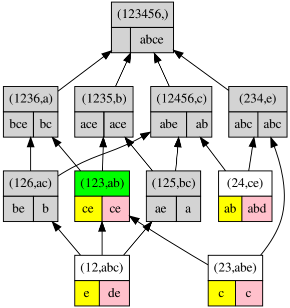


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

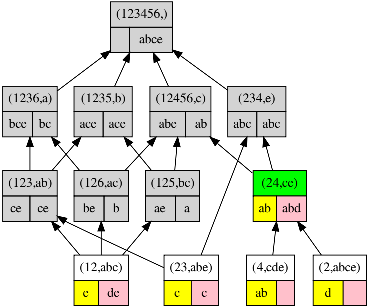


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept



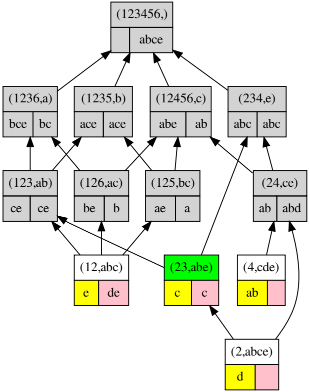
La Rochelle Université

Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

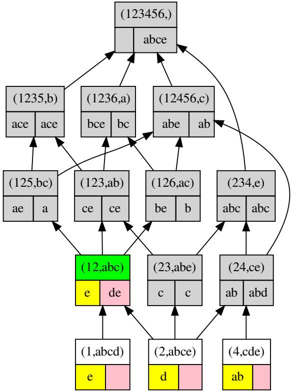


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept



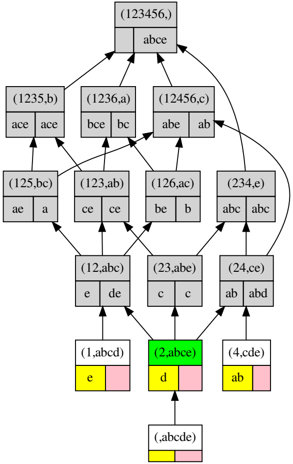
La Rochelle Université

Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

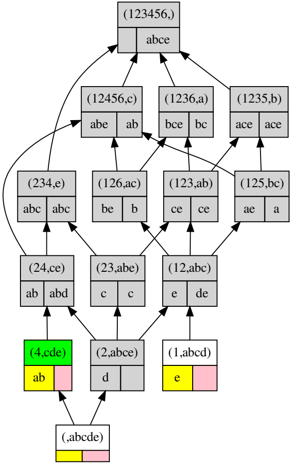


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \text{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

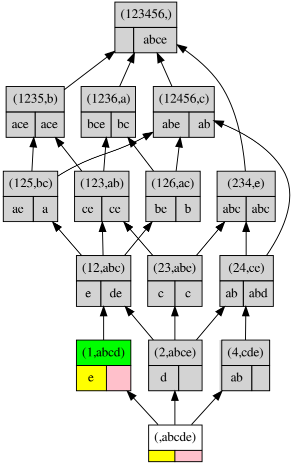


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \text{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

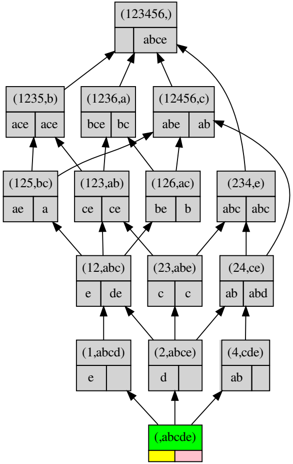


Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \text{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept



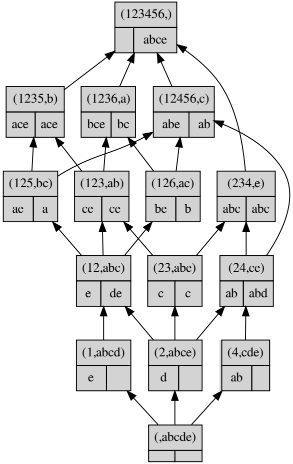
La Rochelle Université

Example

Sample data

(α, β)	a	b	c	d	e
1	✓	✓	✓	✓	
2	✓	✓	✓		✓
3	✓	✓			✓
4			✓	✓	✓
5		✓	✓		
6	✓		✓		

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept

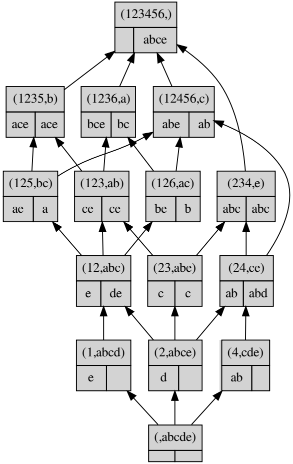


Example

Sample data

(α_P, β_P)	a	b	c	d|abc	d|ce	e
1	✓	✓	✓	✓		
2	✓	✓	✓			✓
3	✓	✓				✓
4			✓		✓	✓
5		✓	✓			
6	✓		✓			

- $\sigma(A) = \{b \in M | \mathrm{Conf}(\alpha(A) + b) \geq 0.5\}$
- constraints
- current concept



NextPriorityConcept: the main theorem

Theorem (Demko et al. 2020)

This NextPriorityConcept algorithm computes the concept lattice of $(G, P, (\alpha_P, \beta_P))$
Where:

- P is the set of selected attributes
- (α_P, β_P) is the associated Galois connection

Heterogeneous data as input

Concepts($\langle G, S, (S^i, \sigma^i, \delta^i)\rangle$)

```
begin
    top ← (G, δ(G));
    Add (|G|, top) to a priority queue Q;
    while Q not empty do
        (A, D) ← Q.pop();
        produce (A, D);
        LP ← Immediate-Predecessors((A, D), σ, δ);
        forall (A', D') ∈ LP do
            Add (|A'|, (A', D')) to Q;
        end
    end
end
```

Heterogeneous data as input

Concepts($\langle G, S, (S^i, \sigma^i, \delta^i)\rangle$)

begin
  top $\leftarrow (G, \delta(G))$;
  Add $(|G|, \text{top})$ to a priority queue Q;
  while Q **not empty do**
    $(A, D) \leftarrow Q.\text{pop}()$;
    produce (A, D);
    $LP \leftarrow$ Immediate-Predecessors$((A, D), \sigma, \delta)$;
    forall $(A', D') \in LP$ **do**
      Add $(|A'|, (A', D'))$ to Q;
    end
  end
end

Groups of charateristics

Charateristics are given by a **family** (S^i) where each S^i contains charateristics of the same domain.

Heterogeneous data as input

Concepts($\langle G, S, (S^i, \sigma^i, \delta^i)\rangle$)

begin
  top ← $(G, \delta(G))$;
  Add $(|G|, \text{top})$ to a priority queue Q;
  while Q **not empty do**
    $(A, D) \leftarrow Q.\text{pop}()$;
    produce (A, D);
    $LP \leftarrow$ Immediate-Predecessors$((A, D), \sigma, \delta)$;
    forall $(A', D') \in LP$ **do**
      Add $(|A'|, (A', D'))$ to Q;
    end
  end
end

Groups of charateristics

Charateristics are given by a **family** (S^i) where each S^i contains charateristics of the same domain.

Descriptions and predicates

Each group of charateristics S^i is provided with a **description** δ^i of objects by predicates.
A description is an application which associates a subset of predicates $\delta(A)$ describing a subset of objects $A \subseteq G$.

Heterogeneous data as input

Concepts($\langle G, S, (S^i, \sigma^i, \delta^i)\rangle$)

begin
- top $\leftarrow (G, \delta(G))$;
- Add $(|G|, \text{top})$ to a priority queue Q;
- **while** Q **not empty do**
  - $(A, D) \leftarrow Q$.pop();
  - **produce** (A, D);
  - $LP \leftarrow$ Immediate-Predecessors($(A, D), \sigma, \delta$);
  - **forall** $(A', D') \in LP$ **do**
    - Add $(|A'|, (A', D'))$ to Q;
  - **end**
- **end**

end

Groups of charateristics

Charateristics are given by a **family** (S^i) where each S^i contains charateristics of the same domain.

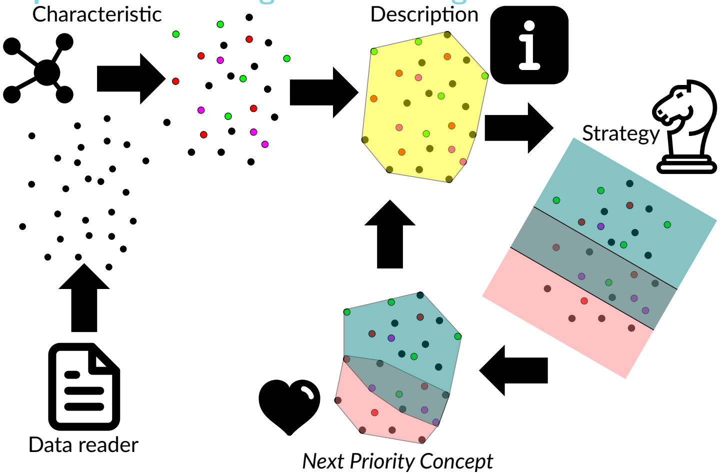
Descriptions and predicates

Each group of charateristics S^i is provided with a **description** δ^i of objects by predicates.
A description is an application which associates a subset of predicates $\delta(A)$ describing a subset of objects $A \subseteq G$.

Strategies

Each group of charateristics S^i is provided with a **strategy** σ^i which defines a set of predicates from which the predecessors are generated.

Descriptions and strategies for heterogenous data



Descriptions and strategies for heterogenous data

Description

The description $\delta(A)$ is composed of predicates describing the borders of the **generalized** convex hull of A

Descriptions and strategies for heterogenous data

Description

The description $\delta(A)$ is composed of predicates describing the borders of the **generalized** convex hull of A

Strategy

The strategy $\sigma(A)$ is composed of predicates describing a "cut" of the **generalized** convex hull of A from which the predecessors are generated.

The NextPriorityConcept algorithm

Remark

Our algorithm is a **pattern discovery** approach where each $(S^i, \sigma^i, \delta^i)$ corresponds to a pattern structure:

- the description δ^i corresponds to the patterns given by predicates
  $=>$ **heterogeneous data are possible**
- the strategy σ^i allows a predecessor generation "on the fly" for each subsets A of objects
  $=>$ **discovered patterns are more suited to the data**

La Rochelle Université

NextPriorityConcept

Theorem (Demko et al. 2020)

If each description δ^i verifies $\delta^i(A) \sqsubseteq \delta^i(A')$ for $A' \subseteq A$ then:
The NextPriorityConcept algorithm computes the concept lattice of $(G, P, (\alpha_P, \beta_P))$ where P is the set of predicates issued from the descriptions.