`galactic` experiment guide

The Galactic Organization <contact@thegalactic.org>

0.3.0

Contents

1 Introduction

This document is produced under the CC-by-nc-nd licence [1]

This experiment guide is a collection of all the jupyter notebooks present in the data projects.

All lines

```
print("test")
```

are python input.

And all lines

```
test
```

are python output.

By default the following colors are used for drawing concept lattices:

- for generators;
- for pseudo-generators;
- for prototypes.

2 Sample data

2.1 `Iris` data set

2.1.1 `Iris` data set, limiting the cardinality of concepts

The lattice construction from the `Iris` data set could lead to very big lattice (several million concepts).

We can use the `Population.from_file` function to load a population in memory and the `Explorer.from_file` function to load a set of strategies described in a `yaml` file.

We can construct a concept lattice from a population and a list of strategies using the `ConceptLattice` class.

The Hasse diagram of a lattice can be visualized using the `HasseDiagram` class, the reduced context can be displayed using the `BinaryTable` class and the summary table can be displayed using the `ConceptTable` class.

```
from galactic.population import Population
from galactic.concepts import ConceptLattice, ConceptRenderer,
↪  ConceptTable
from galactic.algebras.poset import HasseDiagram
from galactic.algebras.relational import BinaryTable
from galactic.strategies import Explorer
from project_data import share_path
```

```python
import sys
import os
data_path = os.path.join(
    share_path,
    "sample",
    "data",
    "iris",
    "iris.csv"
)
```

```python
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
population
```

```
<galactic.population.Population at 0x7f49a8b9d200>
```

```python
len(population)
```

```
150
```

2.1.1.1 Limiting the cardinality of concepts to 110

```python
explorer_path = os.path.join(
    share_path,
    "sample",
    "data",
    "iris",
    "explorer-110.yaml"
)
```

```python
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```


```
characteristics:
- &id001 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "sepal length"
- &id002 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "sepal width"
- &id003 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "petal length"
- &id004 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "petal width"
- &id005 !characteristic.categorized.Category
  characteristic: !characteristic.core.Key
```

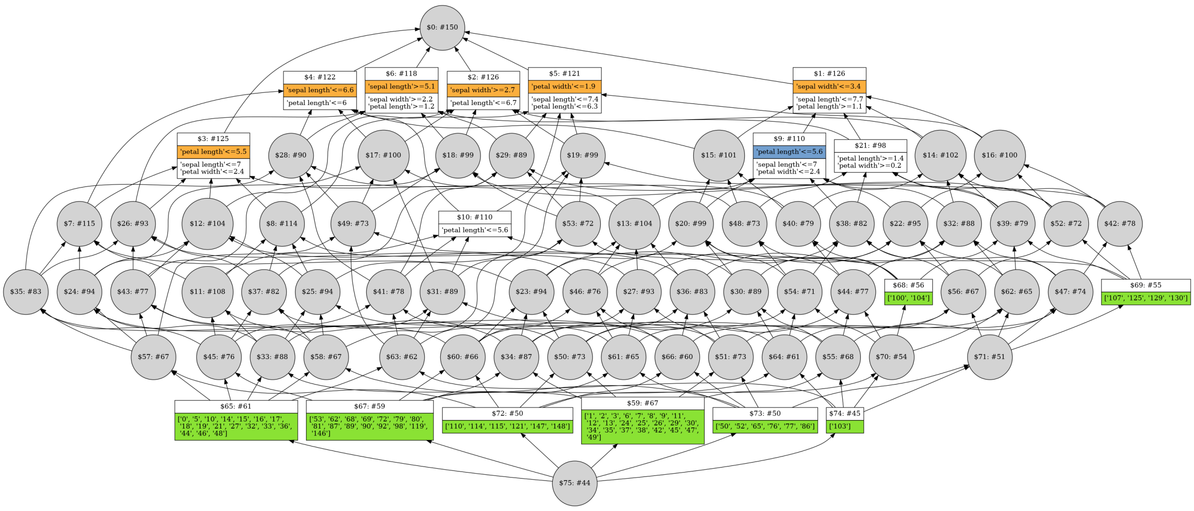
```
    name: "class"

descriptions:
- !description.numerical.hull.Numerical
  - *id001
- !description.numerical.hull.Numerical
  - *id002
- !description.numerical.hull.Numerical
  - *id003
- !description.numerical.hull.Numerical
  - *id004
- !description.categorized.subset.Category
  - *id005

strategies:
- !strategy.core.LimitFilter
  arguments:
  - !strategy.numerical.hull.basic.Normal
    arguments:
      - *id001
  - !strategy.numerical.hull.basic.Normal
    arguments:
      - *id002
  - !strategy.numerical.hull.basic.Normal
    arguments:
      - *id003
  - !strategy.numerical.hull.basic.Normal
    arguments:
      - *id004
  params:
    measure: !measure.core.Cardinality
    limit: 110
```

```python
explorer
```

```
<galactic.strategies.Explorer at 0x7f49dbf43b80>
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(
        show_predicates=True,
        compact=False
    )
)
```



```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6
['1', '2', '3', '6', '7', '8', …]		✓	✓	✓	✓	✓	✓
['0', '5', '10', '14', '15', '16', …]	✓	✓		✓	✓	✓	
['20', '23', '28', '31', '39', '51', …]	✓	✓	✓		✓	✓	✓
['20', '23', '28', '31', '39', '51', …]	✓	✓	✓	✓	✓		✓

['20', '23', '28', '31', '39', '50', ...]	✓		✓	✓	✓	✓	✓
['20', '23', '28', '31', '39', '51', ...]	✓	✓	✓	✓		✓	✓
['20', '23', '28', '31', '39', '50', ...]	✓		✓	✓		✓	
['20', '23', '28', '31', '39', '51', ...]	✓	✓	✓	✓			

```python
ConceptTable(lattice)
```

Concept	Individuals	Predicates
0		'sepal length'>=4.3'sepal length'<=7.9'sepal width'>=2'sepal width'<=4.4'petal length'>=1'petal length'<=6.9'petal width'>=0.1'petal width'<=2.5class <= {'Iris-virginica', 'Iris-versicolor', 'Iris-setosa'}
1		'sepal length'<=7.7'sepal width'<=3.4'petal length'>=1.1

2		'sepal width'>=2.7'petal length'<=6.7
3		'sepal length'<=7'petal length'<=5.5'petal width'<=2.4
4		'sepal length'<=6.6'petal length'<=6
5		'sepal length'<=7.4'petal length'<=6.3'petal width'<=1.9
6		'sepal length'>=5.1'sepal width'>=2.2'petal length'>=1.2
9		'sepal length'<=7'petal length'<=5.6'petal width'<=2.4
10		'petal length'<=5.6
18	['109', '117', '131']	
21	118	'petal length'>=1.4'petal width'>=0.2
33	['4', '22', '40', '43']	
34	['41', '57', '60', '93', '106']	
39	['102', '105', '120', '122', '124', '135', ...]	

52	108
59	['1', '2', '3', '6', '7', '8', ...]
60	113
62	140
64	134
65	['0', '5', '10', '14', '15', '16', ...]
66	['112', '139', '141', '145']
67	['53', '62', '68', '69', '72', '79', ...]
68	['100', '104']
69	['107', '125', '129', '130']
70	['128', '132', '136']
72	['110', '114', '115', '121', '147', '148']
73	['50', '52', '65', '76', '77', '86']
74	103
75	['20', '23', '28', '31', '39', '51', ...]

2.1.1.2 Limiting the cardinality of concepts to 100 and mixing categorized characteristic and numerical characteristic

```python
explorer_path = os.path.join(
    share_path,
    "sample",
    "data",
    "iris",
    "explorer-class.yaml"
)
```

```python
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```


```
characteristics:
- &id001 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "sepal length"
- &id002 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "sepal width"
- &id003 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "petal length"
- &id004 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "petal width"
- &id005 !characteristic.categorized.Category
  characteristic: !characteristic.core.Key
    name: "class"
  domain:
  - Iris-setosa
  - Iris-versicolor
  - Iris-virginica

descriptions:
- !description.numerical.hull.Numerical
  - *id003
- !description.numerical.hull.Numerical
  - *id004
- !description.categorized.subset.Category
  - *id005

strategies:
- !strategy.core.LimitFilter
  arguments:
```

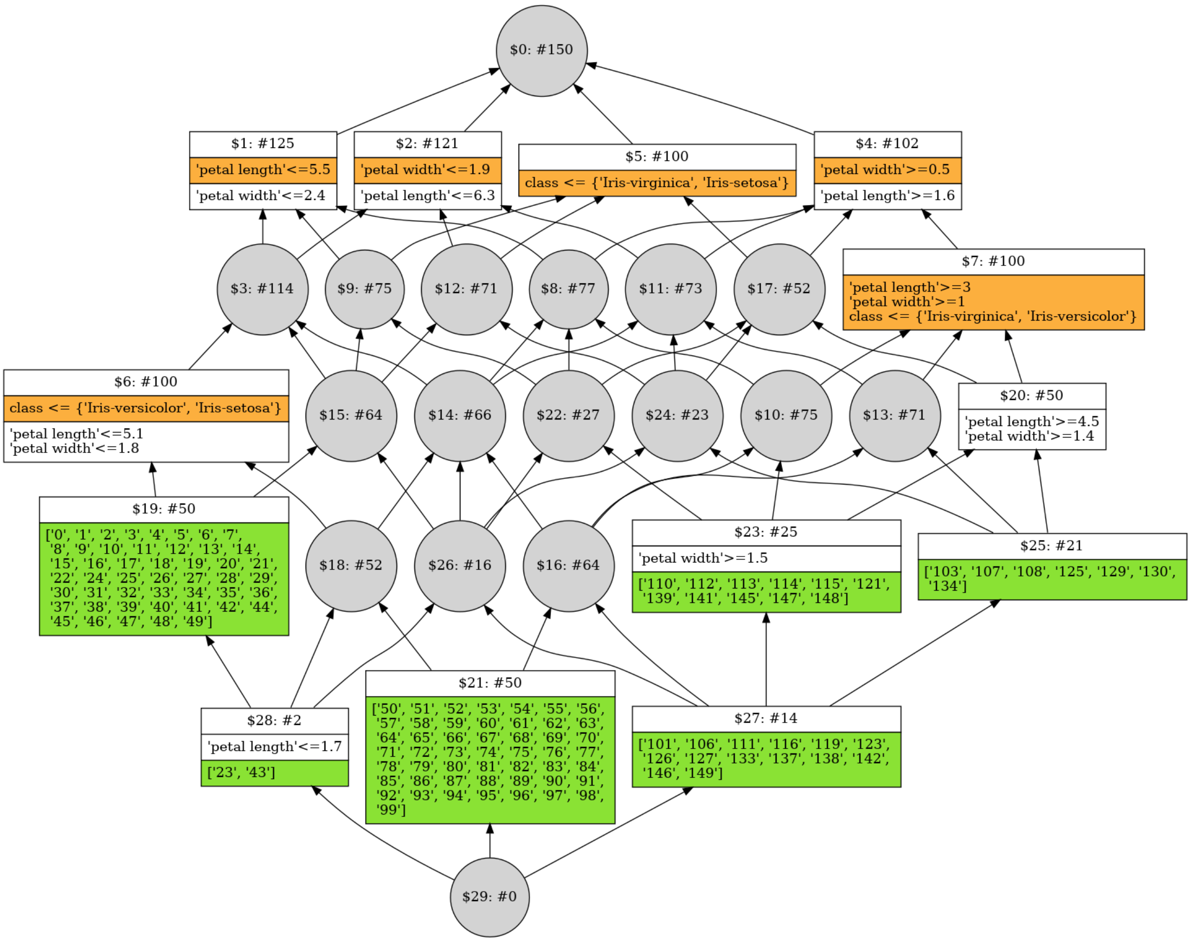
```
  - !strategy.numerical.hull.basic.Normal
    arguments:
    - *id003
    params:
      coefficient: 1
  - !strategy.numerical.hull.basic.Normal
    arguments:
    - *id004
    params:
      coefficient: 1
  params:
    measure: !measure.core.Cardinality
    limit: 100
- !strategy.categorized.subset.basic.Category
  - *id005
```


```python
explorer
```

```
<galactic.strategies.Explorer at 0x7f49a8bc12c0>
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(
        show_predicates=True,
        compact=False
    )
)
```




```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5
['50', '51', '52', '53', '54', '55', …]	✓	✓	✓		✓	✓
['101', '106', '111', '116', '119', '123', …]	✓	✓	✓	✓	✓	
['23', '43']	✓	✓	✓	✓		✓
['0', '1', '2', '3', '4', '5', …]		✓	✓	✓		✓

['101', '106', '110', '111', '112', '113', ...]	✓	✓		✓	✓
['101', '103', '106', '107', '108', '111', ...]	✓		✓	✓	✓

```python
ConceptTable(lattice,concept_width=5,individual_width=20,predicate_width=40)
```

Concept	Individuals	Predicates
0		'petal length'>=1'petal length'<=6.9'petal width'>=0.1'petal width'<=2.5
1		'petal length'<=5.5'petal width'<=2.4
2		'petal length'<=6.3'petal width'<=1.9
4		'petal length'>=1.6'petal width'>=0.5
5		class <= {'Iris-virginica', 'Iris-setosa'}
6		'petal length'<=5.1'petal width'<=1.8class <= {'Iris-versicolor', 'Iris-setosa'}
7		'petal length'>=3'petal width'>=1class <= {'Iris-virginica', 'Iris-versicolor'}
19	['0', '1', '2', '3', '4', '5', ...]	'petal length'<=1.9'petal width'<=0.6class <= {'Iris-setosa'}
20	['100', '102', '104', '105', '109', '117', ...]	'petal length'>=4.5'petal width'>=1.4class <= {'Iris-virginica'}
21	['50', '51', '52', '53', '54', '55', ...]	class <= {'Iris-versicolor'}

23	['110', '112', '113', '114', '115', '121', ...]	'petal width'>=1.5
25	['103', '107', '108', '125', '129', '130', ...]	
27	['101', '106', '111', '116', '119', '123', ...]	
28	['23', '43']	'petal length'<=1.7
29		'petal length'>=nan'petal length'<=nan'petal width'>=nan'petal width'<=nanclass <= {}

2.1.1.3 Using a categorized characteristic and the entropy measure

```python
explorer_path = os.path.join(
    share_path,
    "sample",
    "data",
    "iris",
    "explorer-entropy.yaml"
)
```

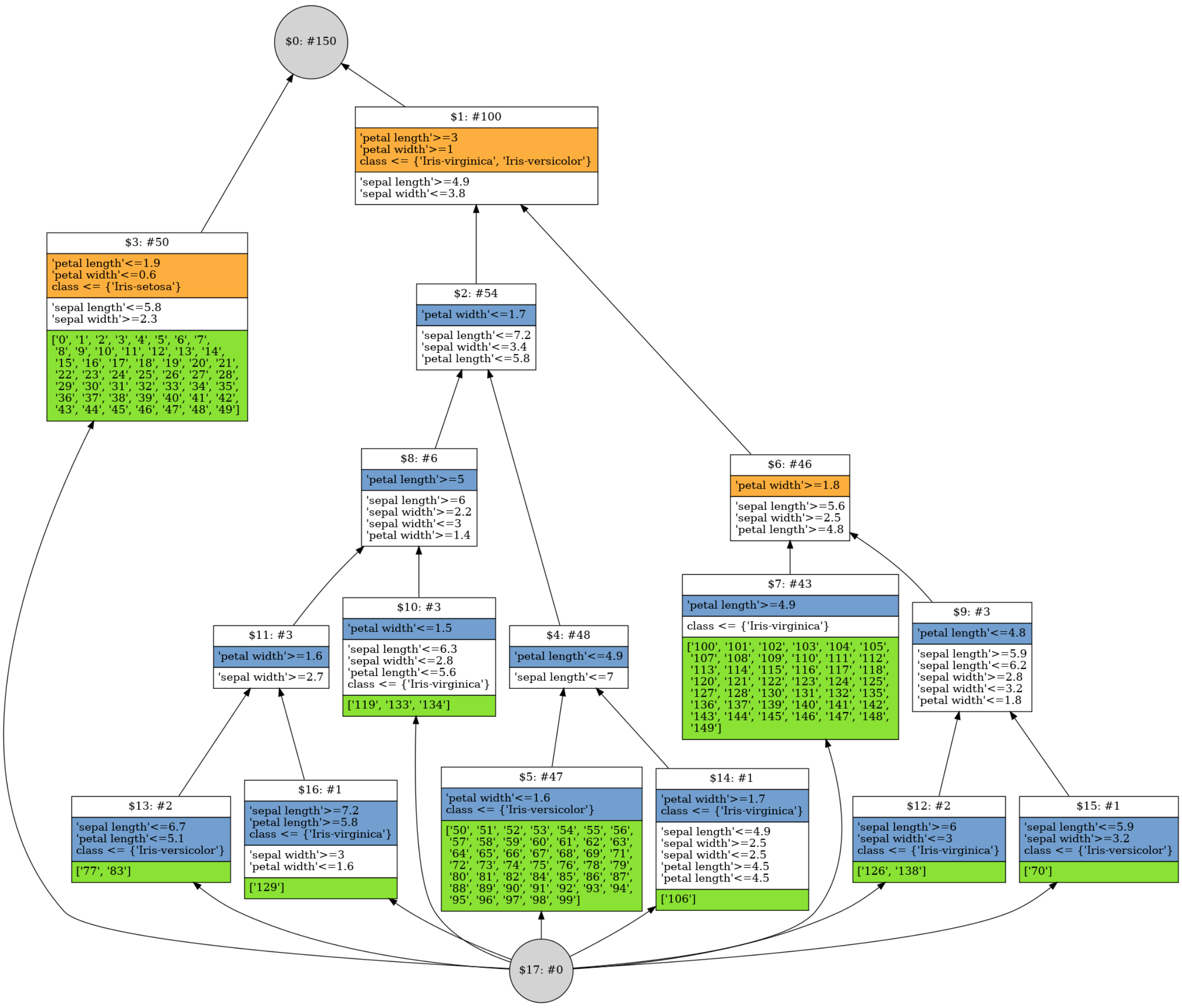
```python
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```

```yaml
characteristics:
- &id001 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "sepal length"
- &id002 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "sepal width"
- &id003 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "petal length"
- &id004 !characteristic.numerical.Number
  characteristic: !characteristic.core.Key
    name: "petal width"
- &id005 !characteristic.categorized.Category
  characteristic: !characteristic.core.Key
    name: "class"

descriptions:
- !description.numerical.hull.Numerical
  - *id001
- !description.numerical.hull.Numerical
  - *id002
- !description.numerical.hull.Numerical
  - *id003
- !description.numerical.hull.Numerical
  - *id004
- !description.categorized.subset.Category
  - *id005

strategies:
- !strategy.core.SelectionFilter
  arguments:
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id001
    params:
      lower: true
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id001
    params:
      lower: false
```

```
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id002
    params:
      lower: true
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id002
    params:
      lower: false
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id003
    params:
      lower: true
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id003
    params:
      lower: false
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id004
    params:
      lower: true
      count: null
  - !strategy.numerical.hull.quantile.Quantile
    arguments:
    - *id004
    params:
      lower: false
      count: null
  params:
    measure: !measure.entropy.Entropy
      category: *id005
    maximize: false
    strict: true
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(
        show_predicates=True,
        compact=False
    )
)
```



3 Sequence experiments

3.1 String data

3.1.1 Chemical Formula example

We can use the Population.from_file function to load a population in memory and the Explorer.from_file function to load a set of strategies described in a yaml file.

We can construct a concept lattice from a population and a list of strategies using the Lattice class.

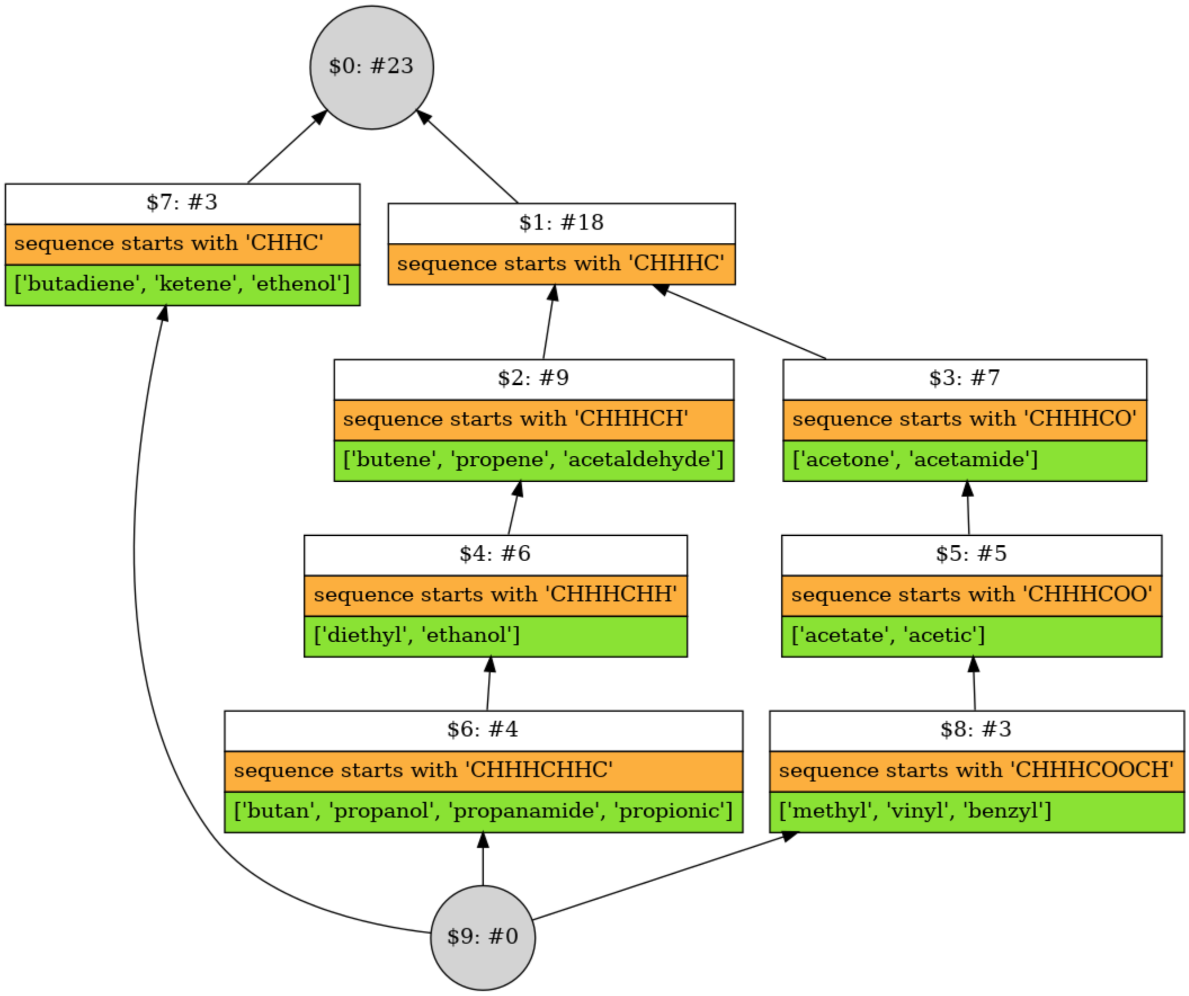
The Hasse diagram of a lattice can be visualized using the HasseDiagram class, the reduced context can be displayed using the ReducedContext class and the summary table can be displayed using the Table class.

```python
from galactic.population import Population
from galactic.strategies import Explorer
from galactic.concepts import (
    ConceptLattice,
    ConceptRenderer,
    ConceptTable
)
from galactic.algebras.poset import HasseDiagram
from galactic_strategy_string_match_basic import (
    CompleteMatchStrategy,
    PrefixMatchStrategy
)
from galactic_description_string_match import (
    PrefixDescription
)
from galactic.strategies import LimitFilter
from galactic.characteristics import Key
from galactic.strategies import Cardinality
from galactic.algebras.relational import BinaryTable
from galactic_characteristic_string import String
from project_data import share_path
import sys
import os

data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "examples",
    "string-example-chemicalformula-23.yaml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
len(population)
```

```
23
```

```python
characteristics = [
    String(characteristic=Key(name="sequence"))
]
descriptions = [
    PrefixDescription(characteristics[0])
]
strategies = [
    LimitFilter(
        PrefixMatchStrategy(characteristics[0]),
        measure=Cardinality(),
        limit=3
    )
]
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```




```
display(*lattice.domain)
```

Predicates:

- `sequence starts with 'CHH'`

Individuals:

- butadiene
- ketene
- ethenol
- formaldehyde
- ethylene
- propyne
- butene
- …

Predicates:

- `sequence starts with 'CHHHC'`

Individuals:

- propyne
- butene
- propene
- acetaldehyde
- butan
- propanol
- propanamide
- …

Predicates:

- `sequence starts with 'CHHHCH'`

Individuals:

- butene
- propene
- acetaldehyde
- butan
- propanol
- propanamide
- propionic
- …

Predicates:

- `sequence starts with 'CHHHCO'`

Individuals:

- acetone
- methyl
- acetamide
- acetate
- vinyl
- benzyl
- acetic

Predicates:

- `sequence starts with 'CHHHCHH'`

Individuals:

- butan
- propanol
- propanamide

- propionic
- diethyl
- ethanol

Predicates:

- `sequence starts with 'CHHHCOO'`

Individuals:

- methyl
- acetate
- vinyl
- benzyl
- acetic

Predicates:

- `sequence starts with 'CHHHCHHC'`

Individuals:

- butan
- propanol
- propanamide
- propionic

Predicates:

- `sequence starts with 'CHHC'`

Individuals:

- butadiene
- ketene
- ethenol

Predicates:

- `sequence starts with 'CHHHCOOCH'`

Individuals:

- methyl
- vinyl
- benzyl

Predicates:

- `sequence starts with {}`

Individuals:

3.1.2 Verbs example

```python
data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "examples",
    "string-example-conjugation-13.yaml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
len(population)
```

```
13
```

```python
explorer_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "explorers",
    "string",
    "complete-match-basic-verbes.yaml"
)
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```

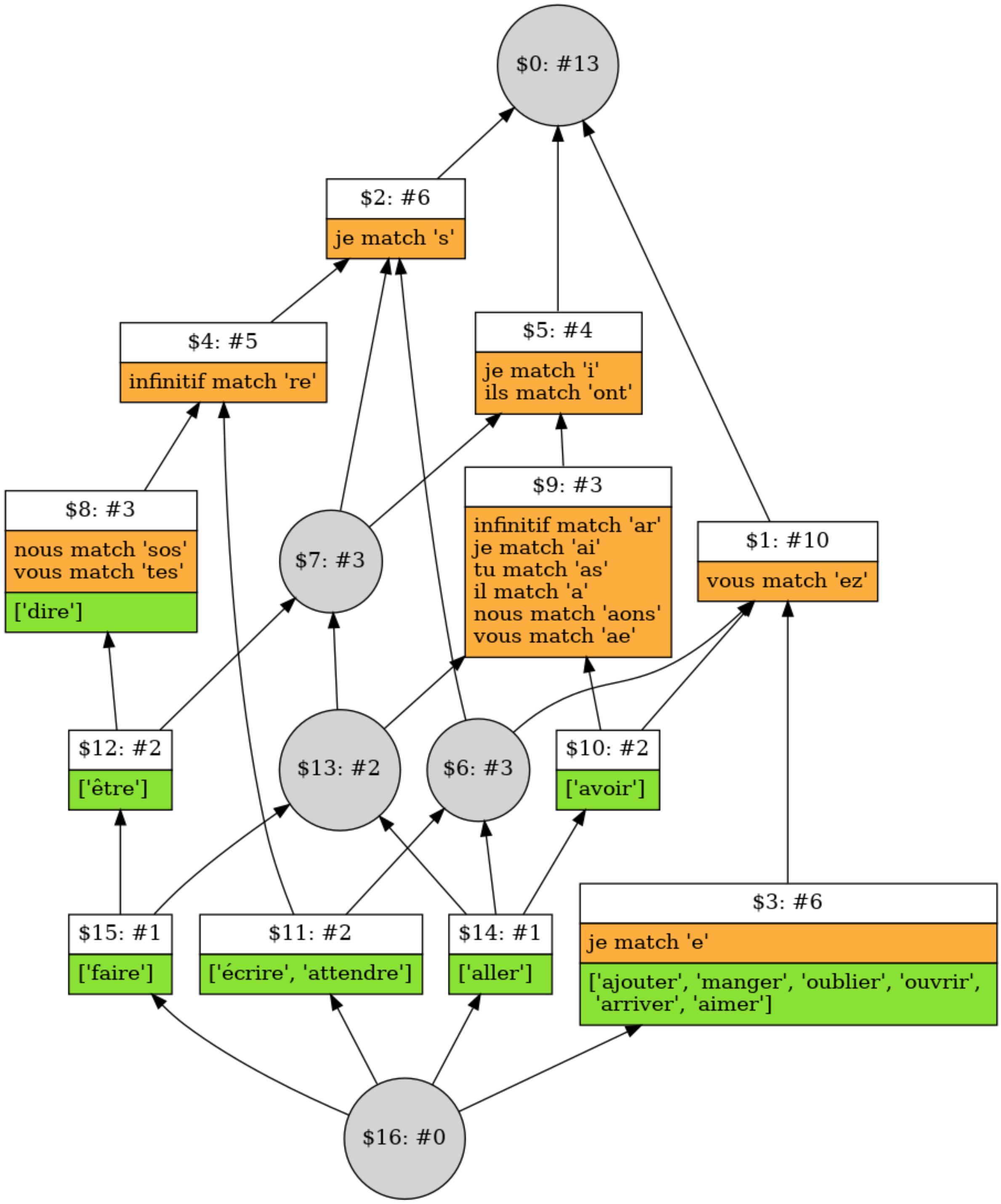

```
characteristics:
- &id001 !characteristic.string.String
  characteristic: !characteristic.core.Key
    name: "infinitif"
- &id002 !characteristic.string.String
  characteristic: !characteristic.core.Key
    name: "je"
- &id003 !characteristic.string.String
  characteristic: !characteristic.core.Key
    name: "tu"
- &id004 !characteristic.string.String
  characteristic: !characteristic.core.Key
    name: "il"
- &id005 !characteristic.string.String
  characteristic: !characteristic.core.Key
    name: "nous"
- &id006 !characteristic.string.String
  characteristic: !characteristic.core.Key
    name: "vous"
- &id007 !characteristic.string.String
```

```
  characteristic: !characteristic.core.Key
    name: "ils"

descriptions:
- !description.string.Complete
  arguments:
  - *id001
- !description.string.Complete
  arguments:
  - *id002
- !description.string.Complete
  arguments:
  - *id003
- !description.string.Complete
  arguments:
  - *id004
- !description.string.Complete
  arguments:
  - *id005
- !description.string.Complete
  arguments:
  - *id006
- !description.string.Complete
  arguments:
  - *id007

strategies:
- !strategy.string.match.basic.Complete
  arguments:
  - *id001
- !strategy.string.match.basic.Complete
  arguments:
  - *id002
- !strategy.string.match.basic.Complete
  arguments:
  - *id003
- !strategy.string.match.basic.Complete
  arguments:
  - *id004
- !strategy.string.match.basic.Complete
  arguments:
  - *id005
- !strategy.string.match.basic.Complete
```

```
  arguments:
  - *id006
- !strategy.string.match.basic.Complete
  arguments:
  - *id007
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```




```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6
[‘ajouter’, ‘manger’, ‘oublier’, ‘ouvrir’, ‘arriver’, ‘aimer’]		✓					✓
[‘écrire’, ‘attendre’]	✓	✓				✓	
aller	✓	✓	✓	✓			
faire	✓		✓	✓	✓	✓	
[‘faire’, ‘être’]	✓		✓		✓	✓	
[‘avoir’, ‘aller’]		✓	✓	✓			
[‘faire’, ‘être’, ‘dire’]	✓				✓	✓	

```
ConceptTable(lattice)
```

Concept	Individuals	Predicates
0		infinitif match 'r'je match {}tu match 's'nous match 'os'vous match 'e'ils match 'nt'
1		nous match 'ons'vous match 'ez'
2		infinitif match 'e'je match 's'

3	['ajouter', 'manger', 'oublier', 'ouvrir', 'arriver', 'aimer']	je match 'e'tu match 'es'il match 'e'ils match 'ent'
4		infinitif match 're'
5		je match 'i'il match {}ils match 'ont'
7		je match 'is'
8	dire	il match 't'nous match 'sos'vous match 'tes'
9		infinitif match 'ar'je match 'ai'tu match 'as'il match 'a'nous match 'aons'vous match 'ae'
10	avoir	vous match 'aez'
11	['écrire', 'attendre']	ils match 'ent'
12	être	
13		infinitif match 'ae'je match 'ais'
14	aller	infinitif match 'aller'je match 'vais'tu match 'vas'il match 'va'nous match 'allons'vous match 'allez'ils match 'vont'
15	faire	infinitif match 'aire'tu match 'ais'il match 'ait'nous match 'aisons'vous match 'aites'

3.2 Chain data

3.2.1 Wine-City data set

The Wine-City dataset is issued from the museum data "La cité du vin" in Bordeaux, France (https://www.laciteduvin.com/en), gathered from the visits on a period of one year (May 2016 to May 2017). The museum is a large "open-space", where visitors are free to explore the museum the way they want, without predetermined path. When they arrive at the museum, they receive a small personal device to detect whenever a visitor is close to an animation spot called a *module*. The museum contains 20 modules, so we can say that extracted sequences for this data is quite short. By extracting the sequences of activation of each module, we end up with a precise enough idea of what the visit looked like for each visitor of the museum.

The lattice construction from the `Wine-City` data set could lead to very big lattice (several hundred thousand concepts) and takes a lot of time to precess. Here we select random portion from the dataset to test our strategies, also we may use the meta strategies to limite the generation of concepts according to the support or confedence.

We can use the `Population.from_file` function to load a population in memory and the `Explorer.from_file` function to load a set of strategies described in a `yaml` file.

We can construct a concept lattice from a population and a list of strategies using the `Lattice` class.

The Hasse diagram of a lattice can be visualized using the `HasseDiagram` class, the reduced context can be displayed using the `ReducedContext` class and the summary table can be displayed using the `Table` class.

```python
from galactic.population import Population
from galactic.strategies import Explorer
from galactic.concepts import (
    ConceptRenderer,
    ConceptLattice,
    ConceptTable
)
from galactic.algebras.poset import HasseDiagram
from galactic.algebras.relational import BinaryTable
from project_data import share_path
```

```python
import sys
import os
data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "examples",
    "chain-example-wine-city-20.yaml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
population
len(population)
```

```
10
```

3.2.1.1 Simple Match Strategy

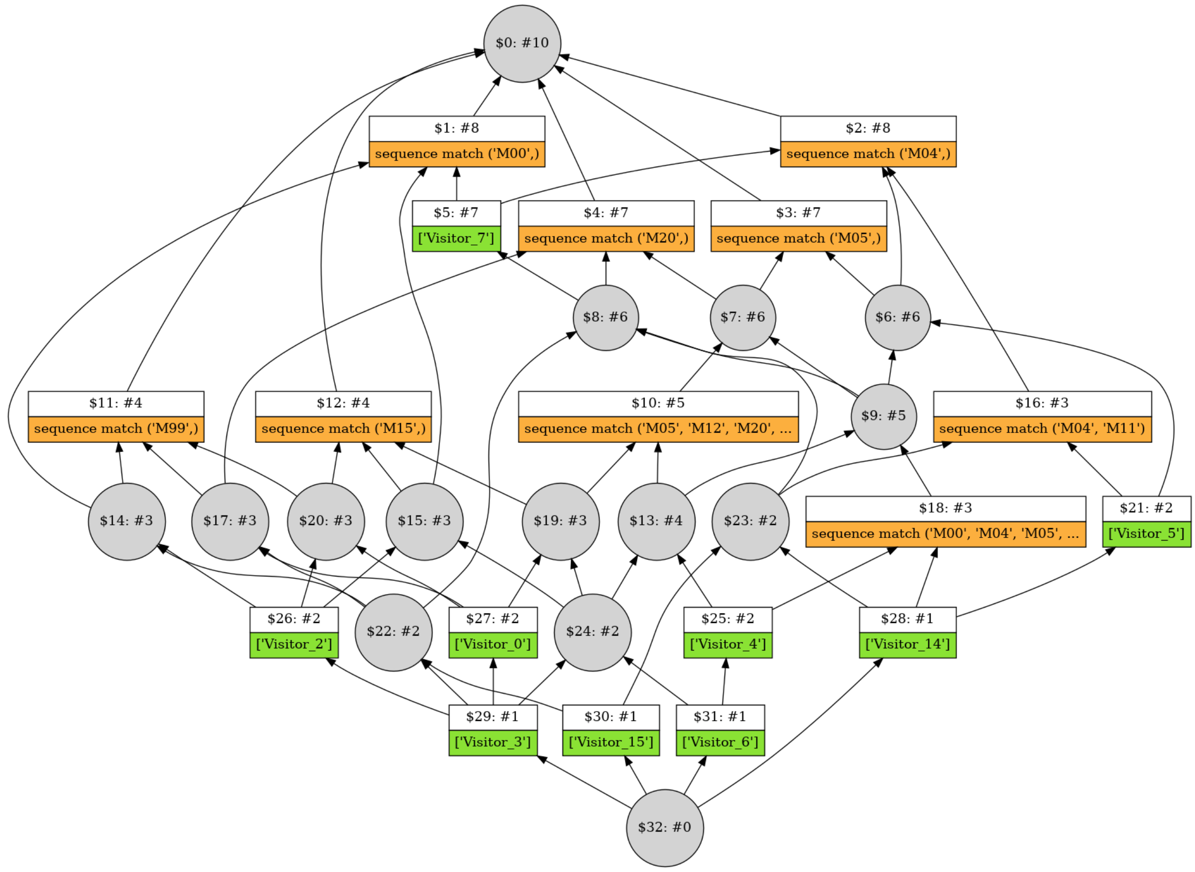
```python
explorer_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "explorers",
    "chain",
    "simple-match-basic.yaml"
)
```

```python
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```


```yaml
characteristics:
- &id001 !characteristic.chain.Chain
  characteristic: !characteristic.core.Key
    name: "sequence"
descriptions:
- !description.chain.Simple
  - *id001
strategies:
- !strategy.chain.match.basic.Simple
  arguments:
  - *id001
  params:
    length: 2
```

```python
from galactic_strategy_chain_match_basic import (
    CompleteMatchStrategy,
    PrefixMatchStrategy
)
from galactic_description_chain_match import (
    CompleteDescription
)
from galactic.strategies import LimitFilter
from galactic_characteristic_chain import Chain
from galactic.characteristics import Key
from galactic.strategies import Cardinality

characteristics = [
    Chain(characteristic=Key(name="sequence"))
]
descriptions = [
    CompleteDescription(characteristics[0])
]
strategies = [
    LimitFilter(
        CompleteMatchStrategy(characteristics[0]),
        measure=Cardinality(),
        limit=2
    )
]

lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```




```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6	@7	@8	@9
['Visitor_4', 'Visitor_6']		✓	✓	✓	✓			✓		✓
Visitor_3	✓	✓	✓	✓	✓	✓	✓			✓
Visitor_14		✓	✓	✓	✓			✓	✓	
Visitor_15	✓		✓	✓	✓				✓	
['Visitor_5', 'Visitor_14']		✓		✓					✓	

['Visitor_0', 'Visitor_3']	✓	✓			✓	✓	✓			✓
['Visitor_2', 'Visitor_3']	✓		✓			✓				
['Visitor_3', 'Visitor_6']		✓	✓	✓	✓	✓	✓			✓
['Visitor_1', 'Visitor_3', 'Visitor_4', 'Visitor_6', 'Visitor_7', 'Visitor_14', …]			✓	✓						

```
ConceptTable(lattice)
```

Concept	Individuals	Predicates
0		sequence match {}
1		sequence match ('M00',)
2		sequence match ('M04',)
3		sequence match ('M05',)
4		sequence match ('M20',)
5	Visitor_7	sequence match ('M00', 'M04')

6		sequence match ('M04', 'M05')
7		sequence match ('M05', 'M12', 'M20')
8		sequence match ('M00', 'M04', 'M20')
9		sequence match ('M00', 'M04', 'M05', 'M12', 'M20')
10		sequence match ('M05', 'M12', 'M20', 'M23')
11		sequence match ('M99',)
12		sequence match ('M15',)
13	Visitor_1	sequence match ('M00', 'M04', 'M05', 'M12', 'M20', 'M23')
14		sequence match ('M00', 'M99')
15		sequence match ('M00', 'M15')
16		sequence match ('M04', 'M11')
17		sequence match ('M20', 'M99')
18		sequence match ('M00', 'M04', 'M05', 'M12', 'M20', 'M22')
19		sequence match ('M05', 'M07', 'M09', 'M12', 'M15', 'M20', 'M23')

20		sequence match ('M15', 'M99')
21	Visitor_5	sequence match ('M04', 'M05', 'M11')
22		sequence match ('M00', 'M03', 'M04', 'M17', 'M20', 'M99')
23		sequence match ('M00', 'M04', 'M10', 'M11', 'M20')
24		sequence match ('M00', 'M04', 'M05', 'M07', 'M09', 'M12', 'M15', 'M17', 'M19', 'M20', 'M23')
25	Visitor_4	sequence match ('M00', 'M04', 'M05', 'M09', 'M12', 'M20', 'M22', 'M23')
26	Visitor_2	sequence match ('M00', 'M15', 'M99')
27	Visitor_0	sequence match ('M05', 'M07', 'M09', 'M12', 'M15', 'M20', 'M23', 'M99')
28	Visitor_14	sequence match ('M00', 'M04', 'M05', 'M06', 'M10', 'M11', 'M12', 'M19', 'M20', 'M22')
29	Visitor_3	sequence match ('M00', 'M03', 'M04', 'M05', 'M07', 'M08', 'M09', 'M12', 'M15', 'M17', 'M19', 'M20', 'M23', 'M99')

30	Visitor_15	sequence match ('M00', 'M03', 'M04', 'M10', 'M11', 'M17', 'M20', 'M99')
31	Visitor_6	sequence match ('M00', 'M04', 'M05', 'M06', 'M07', 'M09', 'M12', 'M14', 'M15', 'M17', 'M19', 'M20', 'M22', 'M23')
32		sequence match {}

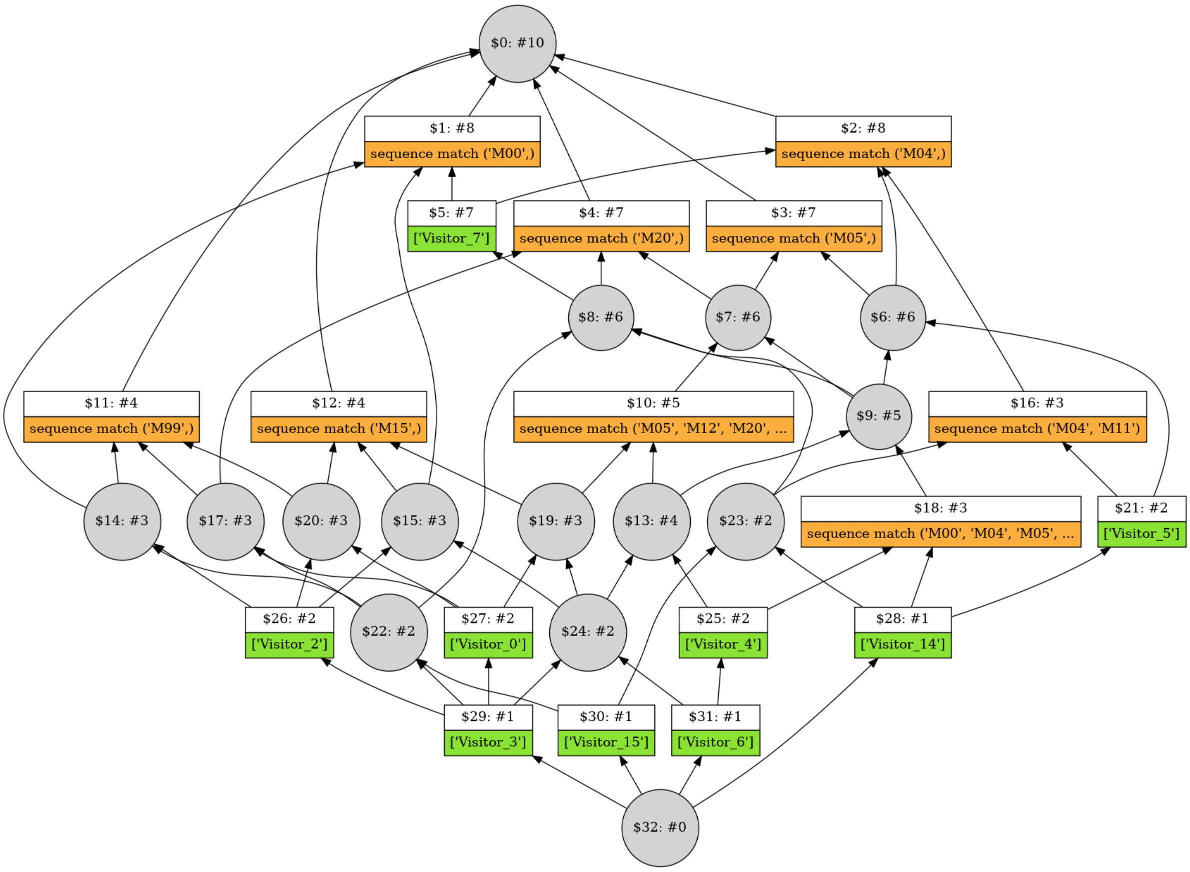
3.2.1.2 Complete Match Strategy

```python
explorer_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "explorers",
    "chain",
    "complete-match-basic.yaml"
)
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```

```yaml
characteristics:
- &id001 !characteristic.chain.Chain
  characteristic: !characteristic.core.Key
    name: "sequence"
descriptions:
- !description.chain.Complete
  - *id001
strategies:
- !strategy.chain.match.basic.Complete
  - *id001
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```




```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6	@7	@8	@9
[‘Visitor_4’, ‘Visitor_6’]		✓	✓	✓	✓			✓		✓
Visitor_3	✓	✓	✓	✓	✓	✓	✓			✓
Visitor_14		✓	✓	✓	✓			✓	✓	
Visitor_15	✓		✓	✓	✓				✓	
[‘Visitor_5’, ‘Visitor_14’]		✓		✓					✓	
[‘Visitor_0’, ‘Visitor_3’]	✓	✓			✓	✓	✓			✓
[‘Visitor_2’, ‘Visitor_3’]	✓		✓			✓				
[‘Visitor_3’, ‘Visitor_6’]		✓	✓	✓	✓	✓	✓			✓
[‘Visitor_1’, ‘Visitor_3’, ‘Visitor_4’, ‘Visitor_6’, ‘Visitor_7’, ‘Visitor_14’, …]			✓	✓						

```python
ConceptTable(
    lattice,
    concept_width=5,
    individual_width=10,
    predicate_width=40
)
```

Concept	Individuals	Predicates
0		sequence match {}
1		sequence match ('M00',)
2		sequence match ('M04',)
3		sequence match ('M05',)
4		sequence match ('M20',)
5	Visitor_7	sequence match ('M00', 'M04')
6		sequence match ('M04', 'M05')
7		sequence match ('M05', 'M12', 'M20')
8		sequence match ('M00', 'M04', 'M20')
9		sequence match ('M00', 'M04', 'M05', 'M12', 'M20')
10		sequence match ('M05', 'M12', 'M20', 'M23')
11		sequence match ('M99',)
12		sequence match ('M15',)
13	Visitor_1	sequence match ('M00', 'M04', 'M05', 'M12', 'M20', 'M23')
14		sequence match ('M00', 'M99')
15		sequence match ('M00', 'M15')
16		sequence match ('M04', 'M11')
17		sequence match ('M20', 'M99')
18		sequence match ('M00', 'M04', 'M05', 'M12', 'M20', 'M22')

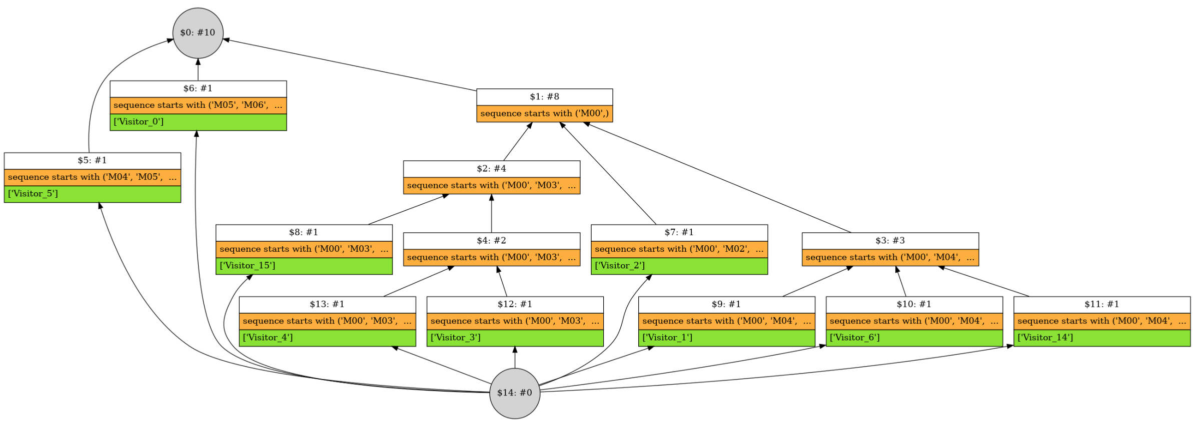
19		sequence match ('M05', 'M07', 'M09', 'M12', 'M15', 'M20', 'M23')
20		sequence match ('M15', 'M99')
21	Visitor_5	sequence match ('M04', 'M05', 'M11')
22		sequence match ('M00', 'M03', 'M04', 'M17', 'M20', 'M99')
23		sequence match ('M00', 'M04', 'M10', 'M11', 'M20')
24		sequence match ('M00', 'M04', 'M05', 'M07', 'M09', 'M12', 'M15', 'M17', 'M19', 'M20', 'M23')
25	Visitor_4	sequence match ('M00', 'M04', 'M05', 'M09', 'M12', 'M20', 'M22', 'M23')
26	Visitor_2	sequence match ('M00', 'M15', 'M99')
27	Visitor_0	sequence match ('M05', 'M07', 'M09', 'M12', 'M15', 'M20', 'M23', 'M99')
28	Visitor_14	sequence match ('M00', 'M04', 'M05', 'M06', 'M10', 'M11', 'M12', 'M19', 'M20', 'M22')
29	Visitor_3	sequence match ('M00', 'M03', 'M04', 'M05', 'M07', 'M08', 'M09', 'M12', 'M15', 'M17', 'M19', 'M20', 'M23', 'M99')
30	Visitor_15	sequence match ('M00', 'M03', 'M04', 'M10', 'M11', 'M17', 'M20', 'M99')
31	Visitor_6	sequence match ('M00', 'M04', 'M05', 'M06', 'M07', 'M09', 'M12', 'M14', 'M15', 'M17', 'M19', 'M20', 'M22', 'M23')
32		sequence match {}

3.2.1.3 Prefix Match Strategy

```python
explorer_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "explorers",
    "chain",
    "prefix-match-basic.yaml"
)
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```


```yaml
characteristics:
- &id001 !characteristic.chain.Chain
  characteristic: !characteristic.core.Key
    name: "sequence"
descriptions:
- !description.chain.Prefix
  - *id001
strategies:
- !strategy.chain.match.basic.Prefix
  arguments:
  - *id001
```


```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```



```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	@11	@12
Visitor_0			✓										
Visitor_5	✓												
Visitor_2		✓								✓			
Visitor_15		✓							✓				✓
Visitor_14		✓						✓				✓	
Visitor_6		✓					✓					✓	
Visitor_1		✓				✓						✓	
Visitor_4		✓			✓						✓		✓
Visitor_3		✓		✓							✓		✓

```python
ConceptTable(
    lattice,
    concept_width=5,
    individual_width=10,
    predicate_width=40
)
```

Concept	Individuals	Predicates
0		`sequence starts with {}`
1		`sequence starts with ('M00',)`
2	`Visitor_7`	`sequence starts with ('M00', 'M03', 'M04')`
3		`sequence starts with ('M00', 'M04', 'M05', 'M06')`
4		`sequence starts with ('M00', 'M03', 'M04', 'M05')`
5	`Visitor_5`	`sequence starts with ('M04', 'M05', 'M11')`

6	Visitor_0	sequence starts with ('M05', 'M06', 'M07', 'M09', 'M12', 'M14', 'M15', 'M20', 'M23', 'M99')
7	Visitor_2	sequence starts with ('M00', 'M02', 'M15', 'M99')
8	Visitor_15	sequence starts with ('M00', 'M03', 'M04', 'M10', 'M11', 'M17', 'M20', 'M99')
9	Visitor_1	sequence starts with ('M00', 'M04', 'M05', 'M06', 'M12', 'M14', 'M20', 'M23')
10	Visitor_6	sequence starts with ('M00', 'M04', 'M05', 'M06', 'M07', 'M09', 'M12', 'M14', 'M15', 'M17', 'M19', 'M20', 'M22', 'M23')
11	Visitor_14	sequence starts with ('M00', 'M04', 'M05', 'M06', 'M10', 'M11', 'M12', 'M19', 'M20', 'M22')
12	Visitor_3	sequence starts with ('M00', 'M03', 'M04', 'M05', 'M07', 'M08', 'M09', 'M12', 'M15', 'M17', 'M19', 'M20', 'M23', 'M99')
13	Visitor_4	sequence starts with ('M00', 'M03', 'M04', 'M05', 'M09', 'M10', 'M12', 'M20', 'M22', 'M23')
14		sequence starts with {}

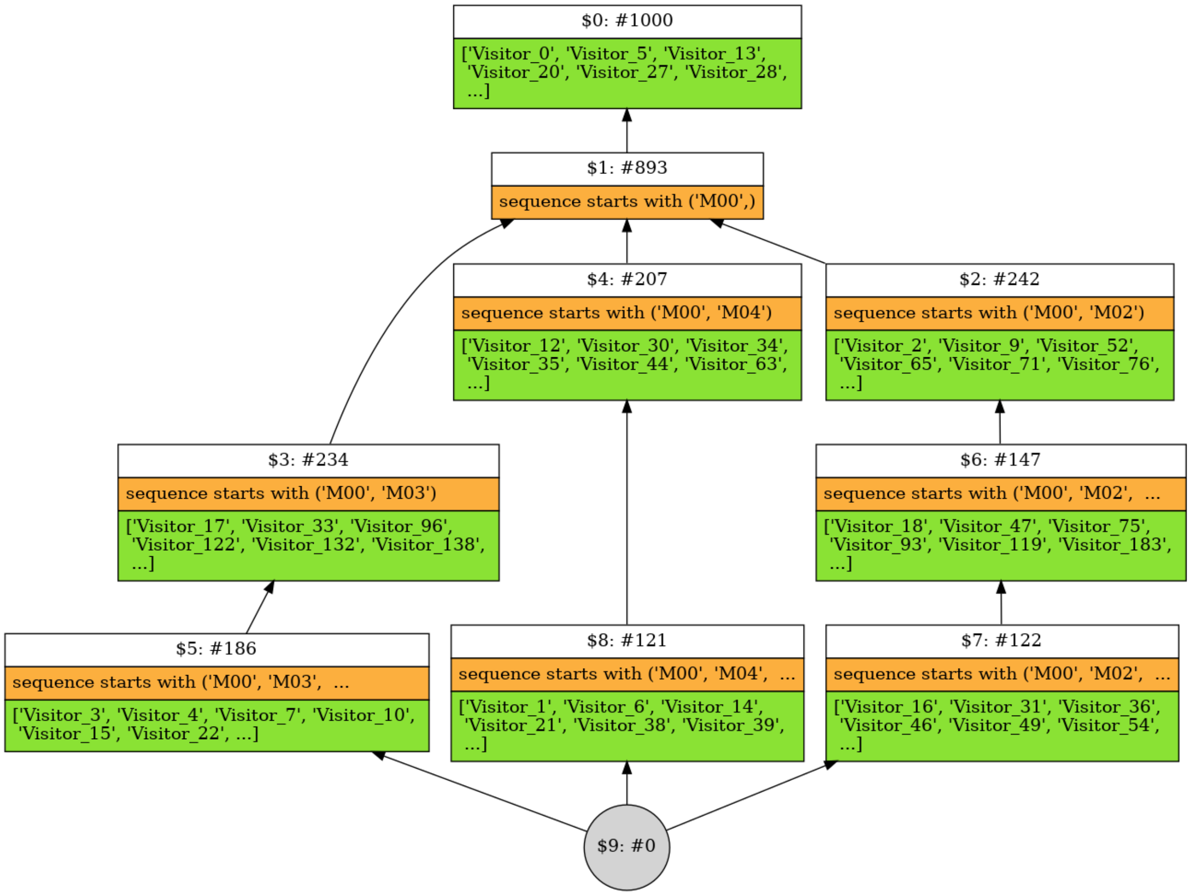
3.2.1.4 Wine-City with 1000 trajectories

```python
import sys
import os
data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "Wine-City",
    "chain",
    "chain-wine-city-1000.yml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
population
len(population)
```

```
1000
```

```python
from galactic_description_chain_match import (
    PrefixDescription
)

characteristics = [
    Chain(characteristic=Key(name="sequence"))
]
descriptions = [
    PrefixDescription(characteristics[0])
]
strategies = [
    LimitFilter(
        PrefixMatchStrategy(characteristics[0]),
        measure=Cardinality(),
        limit=100
    )
]
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies,
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```



3.3 Sequence data

3.3.1 Daily-Actions data set

The Daily-Actions dataset is a small database of sequences that represents daily actions of 25 individuals of L3i laboratory[1]., where a daily action can be $[Wakeup, Breakfast, Work, Dinner..]$.

We can use the Population.from_file function to load a population in memory and the Explorer.from_file function to load a set of strategies described in a yaml file.

We can construct a concept lattice from a population and a list of strategies using the Lattice class.

The Hasse diagram of a lattice can be visualized using the HasseDiagram class, the reduced context can be displayed using the ReducedContext class and the summary table can be displayed using the Table class.

1 : https://l3i.univ-larochelle.fr/

```python
from galactic.population import Population
from galactic.strategies import Explorer
from galactic.concepts import (
    ConceptLattice,
    ConceptRenderer,
    ConceptTable
)
from galactic.algebras.poset import HasseDiagram
from galactic.algebras.relational import BinaryTable
from project_data import share_path
```

```python
import sys
import os

data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "Daily-Actions",
    "sequence-daily-actions-25.yaml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
population
len(population)
```

```
25
```

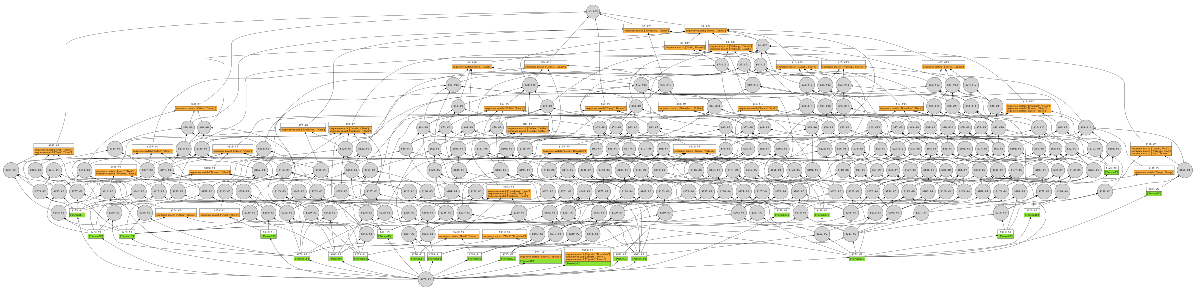
3.3.2 Simple Match Basic

```python
explorer_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "explorers",
    "sequence",
    "simple-match-basic.yaml"
)
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```

```
characteristics:
- &id001 !characteristic.sequence.Sequence
  characteristic: !characteristic.core.Key
    name: "sequence"
descriptions:
- !description.sequence.Simple
  - *id001
strategies:
- !strategy.sequence.match.basic.Simple
  - *id001
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```

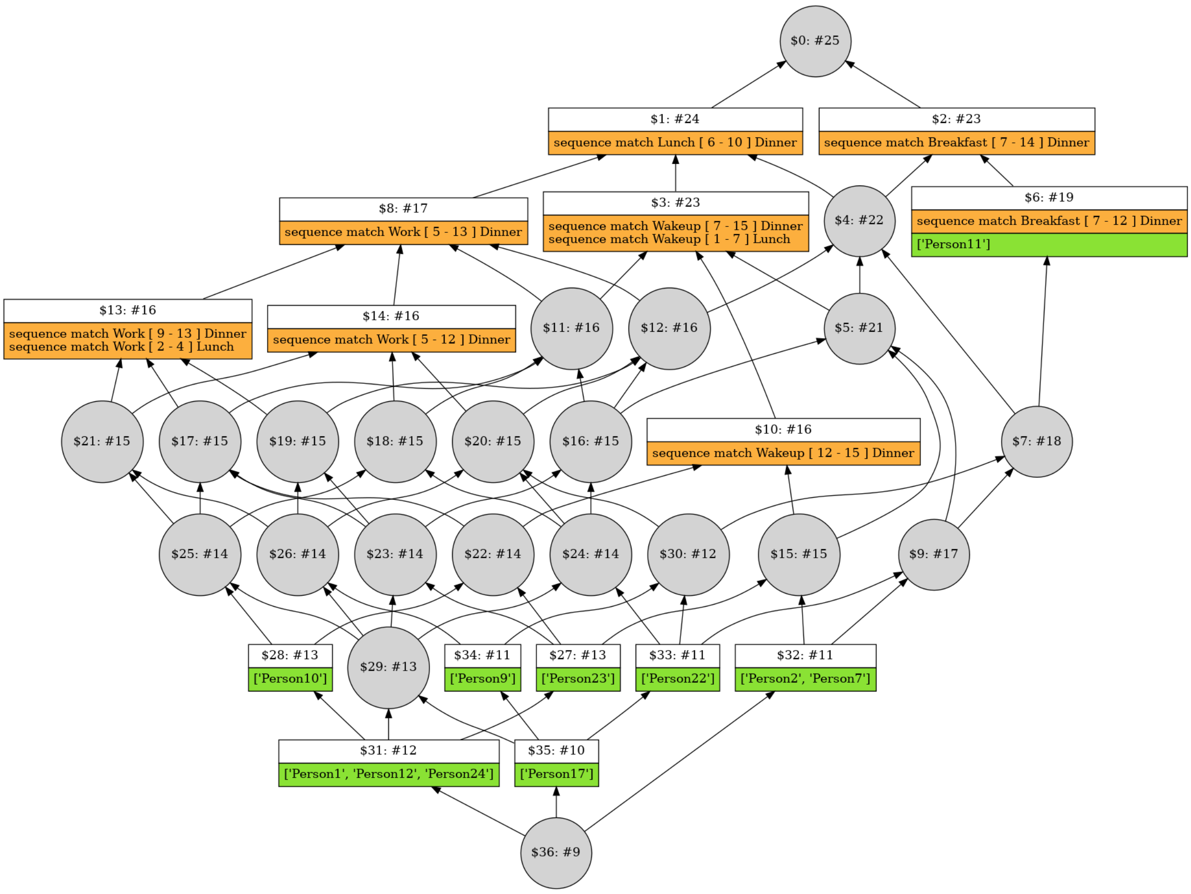


3.3.3 Distance Match Basic

```python
from galactic_strategy_sequence_distance_basic import
↪   DistanceMatchStrategy
from galactic_description_sequence_distance import DistanceDescription
from galactic.strategies import LimitFilter
from galactic.strategies import Cardinality
from galactic_characteristic_sequence import Sequence
from galactic.characteristics import Key

characteristics = [
    Sequence(characteristic=Key(name="sequence"))
]
descriptions = [
    DistanceDescription(characteristics[0])
]
strategies = [
    LimitFilter(
```

```
        DistanceMatchStrategy(characteristics[0]),
        measure= Cardinality(),
        limit=15
    )
]
```

```
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies,
)
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(width=80)
)
```




```
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6	@7	@8	@9
[‘Person1’, ‘Person3’, ‘Person5’, ‘Person6’, ‘Person8’, ‘Person10’, …]	✓				✓	✓		✓		✓
[‘Person1’, ‘Person3’, ‘Person5’, ‘Person6’, ‘Person8’, ‘Person9’, …]	✓	✓			✓	✓		✓		
[‘Person1’, ‘Person3’, ‘Person5’, ‘Person6’, ‘Person8’, ‘Person12’, …]	✓	✓		✓	✓		✓	✓		✓

[‘Person3’, ‘Person5’, ‘Person6’, ‘Person8’, ‘Person14’, ‘Person16’, …]	✓	✓	✓			✓		✓	✓	✓
[‘Person2’, ‘Person3’, ‘Person5’, ‘Person6’, ‘Person7’, ‘Person8’, …]	✓	✓					✓		✓	✓
[‘Person1’, ‘Person3’, ‘Person5’, ‘Person6’, ‘Person8’, ‘Person12’, …]	✓	✓				✓		✓		✓

['Person3', 'Person5', 'Person6', 'Person8', 'Person9', 'Person14', …]	✓	✓	✓			✓		✓	✓	
['Person2', 'Person3', 'Person4', 'Person5', 'Person6', 'Person7', …]		✓							✓	
['Person1', 'Person3', 'Person5', 'Person6', 'Person8', 'Person10', …]	✓			✓	✓		✓	✓		✓

['Person1', 'Person3', 'Person5', 'Person6', 'Person8', 'Person12', ...]	✓	✓	✓	✓	✓

```
ConceptTable(lattice)
```

Concept	Individuals	Predicates
1		sequence match Lunch [6 - 10] Dinner
2		sequence match Breakfast [7 - 14] Dinner
3	Person15	sequence match Wakeup [7 - 15] Dinnersequence match Wakeup [1 - 7] Lunch
4		sequence match Breakfast [8 - 14] Dinnersequence match Breakfast [2 - 5] Lunch
5		sequence match Wakeup [1 - 3] Breakfastsequence match Wakeup [9 - 15] Dinnersequence match Wakeup [3 - 7] Lunch

6	Person11	sequence match Breakfast [7 - 12] Dinner
7		sequence match Breakfast [8 - 12] Dinnersequence match Lunch [6 - 8] Dinner
8		sequence match Work [5 - 13] Dinner
9	['Person4', 'Person13', 'Person20', 'Person25']	sequence match Wakeup [9 - 13] Dinner
10		sequence match Wakeup [12 - 15] Dinnersequence match Wakeup [4 - 7] Lunch
11		sequence match Wakeup [11 - 15] Dinnersequence match Wakeup [1 - 6] Worksequence match Wakeup [4 - 7] Lunch
12		sequence match Breakfast [10 - 14] Dinner
13		sequence match Work [9 - 13] Dinnersequence match Work [2 - 4] Lunch

14		sequence match Work [5 - 12] Dinnersequence match Lunch [6 - 9] Dinner
15		sequence match Breakfast [10 - 14] Dinner
17		sequence match Wakeup [1 - 3] Work
18		sequence match Wakeup [11 - 14] Dinner
20		sequence match Breakfast [10 - 13] Dinner
21		sequence match Work [9 - 12] Dinner
22		sequence match Work [10 - 13] Dinnersequence match Work [3 - 4] Lunch
27	Person23	sequence match Wakeup [5 - 7] Lunch
28	Person10	sequence match Work [10 - 12] Dinnersequence match Wakeup [12 - 14] Dinner
30		sequence match Breakfast [10 - 12] Dinner
31	['Person1', 'Person12', 'Person24']	

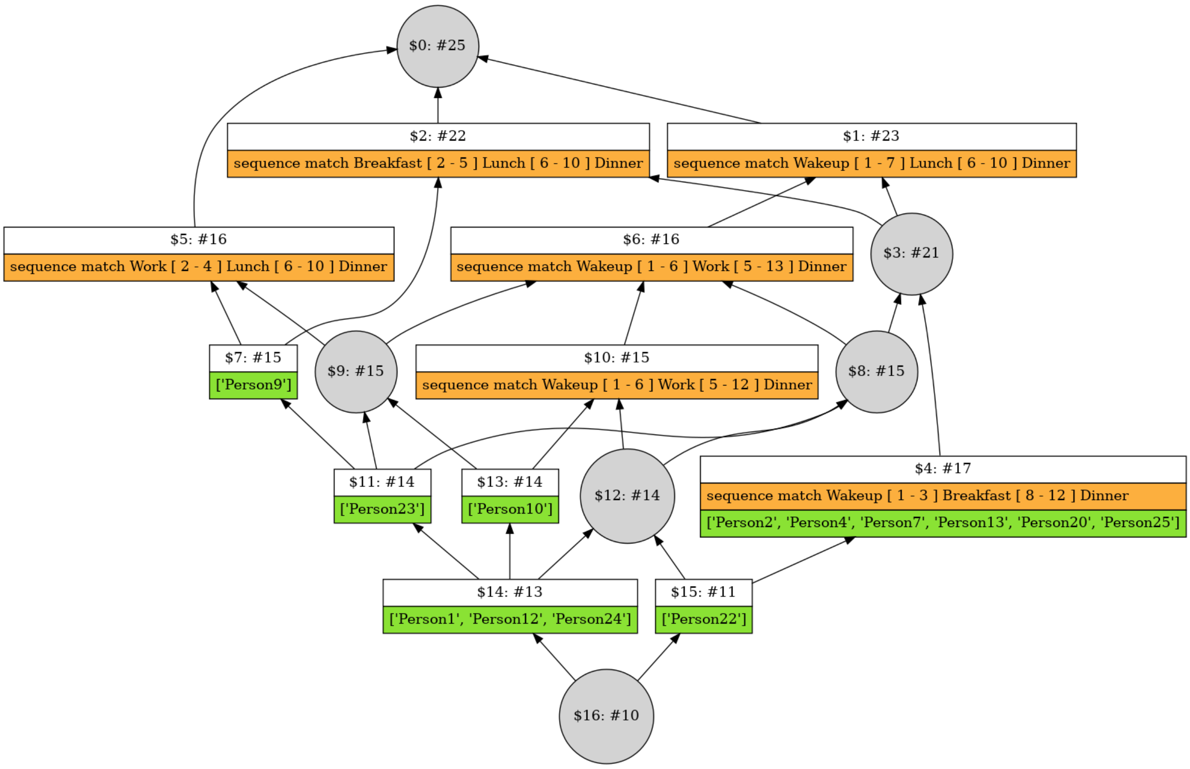
32	['Person2', 'Person7']	sequence match Breakfast [10 - 12] Dinnersequence match Wakeup [12 - 13] Dinner
33	Person22	sequence match Wakeup [11 - 13] Dinner
34	Person9	
35	Person17	
36	['Person3', 'Person5', 'Person6', 'Person8', 'Person14', 'Person16', ...]	

3.3.4 Distance: Length = 3

```python
characteristics = [
    Sequence(characteristic=Key(name="sequence"))
]
descriptions = [
    DistanceDescription(
        characteristics[0],
        length=3
    )
]
strategies = [
    LimitFilter(
        DistanceMatchStrategy(
            characteristics[0],
            length=3
        ),
        measure=Cardinality(),
        limit=15
    )
]
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies,
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(width=80)
)
```




```python
ConceptTable(lattice)
```

Concept	Individuals	Predicates
0	Person11	
1	Person15	sequence match Wakeup [1 - 7] Lunch [6 - 10] Dinner
2		sequence match Breakfast [2 - 5] Lunch [6 - 10] Dinner

3		sequence match Wakeup [1 - 3] Breakfast [8 - 14] Dinnersequence match Wakeup [1 - 3] Breakfast [2 - 5] Lunchsequence match Wakeup [3 - 7] Lunch [6 - 10] Dinner
4	['Person2', 'Person4', 'Person7', 'Person13', 'Person20', 'Person25']	sequence match Breakfast [2 - 5] Lunch [6 - 8] Dinnersequence match Wakeup [1 - 3] Breakfast [8 - 12] Dinnersequence match Wakeup [3 - 7] Lunch [6 - 8] Dinner
5		sequence match Work [2 - 4] Lunch [6 - 10] Dinner
6		sequence match Wakeup [1 - 6] Work [5 - 13] Dinnersequence match Wakeup [4 - 7] Lunch [6 - 10] Dinner
7	Person9	
8		sequence match Wakeup [1 - 3] Breakfast [10 - 14] Dinner

9		sequence match Wakeup [1 - 3] Work [9 - 13] Dinnersequence match Wakeup [1 - 3] Work [2 - 4] Lunch
10		sequence match Wakeup [1 - 6] Work [5 - 12] Dinnersequence match Wakeup [4 - 7] Lunch [6 - 9] Dinner
11	Person23	
12		sequence match Breakfast [2 - 5] Lunch [6 - 9] Dinnersequence match Wakeup [1 - 3] Breakfast [10 - 13] Dinner
13	Person10	sequence match Work [2 - 4] Lunch [6 - 9] Dinnersequence match Wakeup [1 - 3] Work [9 - 12] Dinner
14	['Person1', 'Person12', 'Person24']	
15	Person22	sequence match Wakeup [1 - 3] Breakfast [10 - 12] Dinnersequence match Wakeup [4 - 7] Lunch [6 - 8] Dinner

16	['Person3', 'Person5', 'Person6', 'Person8', 'Person14', 'Person16', ...]	sequence match Work [2 - 4] Lunch [6 - 8] Dinner

3.4 Interval data

3.4.1 Wine-City data set

The Wine-City dataset is issued from the museum data "La cité du vin" in Bordeaux, France (https://www.laciteduvin.com/en), gathered from the visits on a period of one year (May 2016 to May 2017). The museum is a large "open-space", where visitors are free to explore the museum the way they want, without predetermined path. When they arrive at the museum, they receive a small personal device to detect whenever a visitor is close to an animation spot called a *module*. The museum contains 20 modules, so we can say that extracted sequences for this data is quite short. By extracting the sequences of activation of each module, we end up with a precise enough idea of what the visit looked like for each visitor of the museum.

The lattice construction from the `Wine-City` data set could lead to very big lattice (several hundred thousand concepts) and takes a lot of time to precess. Here we select random portion from the dataset to test our strategies, also we may use the meta strategies to limite the generation of concepts according to the support or confedence.

We can use the `Population.from_file` function to load a population in memory and the `Explorer.from_file` function to load a set of strategies described in a `yaml` file.

We can construct a concept lattice from a population and a list of strategies using the `Lattice` class.

The Hasse diagram of a lattice can be visualized using the `HasseDiagram` class, the reduced context can be displayed using the `ReducedContext` class and the summary table can be displayed using the `Table` class.

```python
from galactic.population import Population
from galactic.strategies import Explorer
from galactic.concepts import (
    ConceptLattice,
    ConceptRenderer,
    ConceptTable
)
from galactic.algebras.poset import HasseDiagram
from galactic.algebras.relational import BinaryTable
from project_data import share_path
```

```python
import sys
import os

data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "Wine-City",
    "interval",
    "interval-wine-city-20.yml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
population
len(population)
```

```
20
```

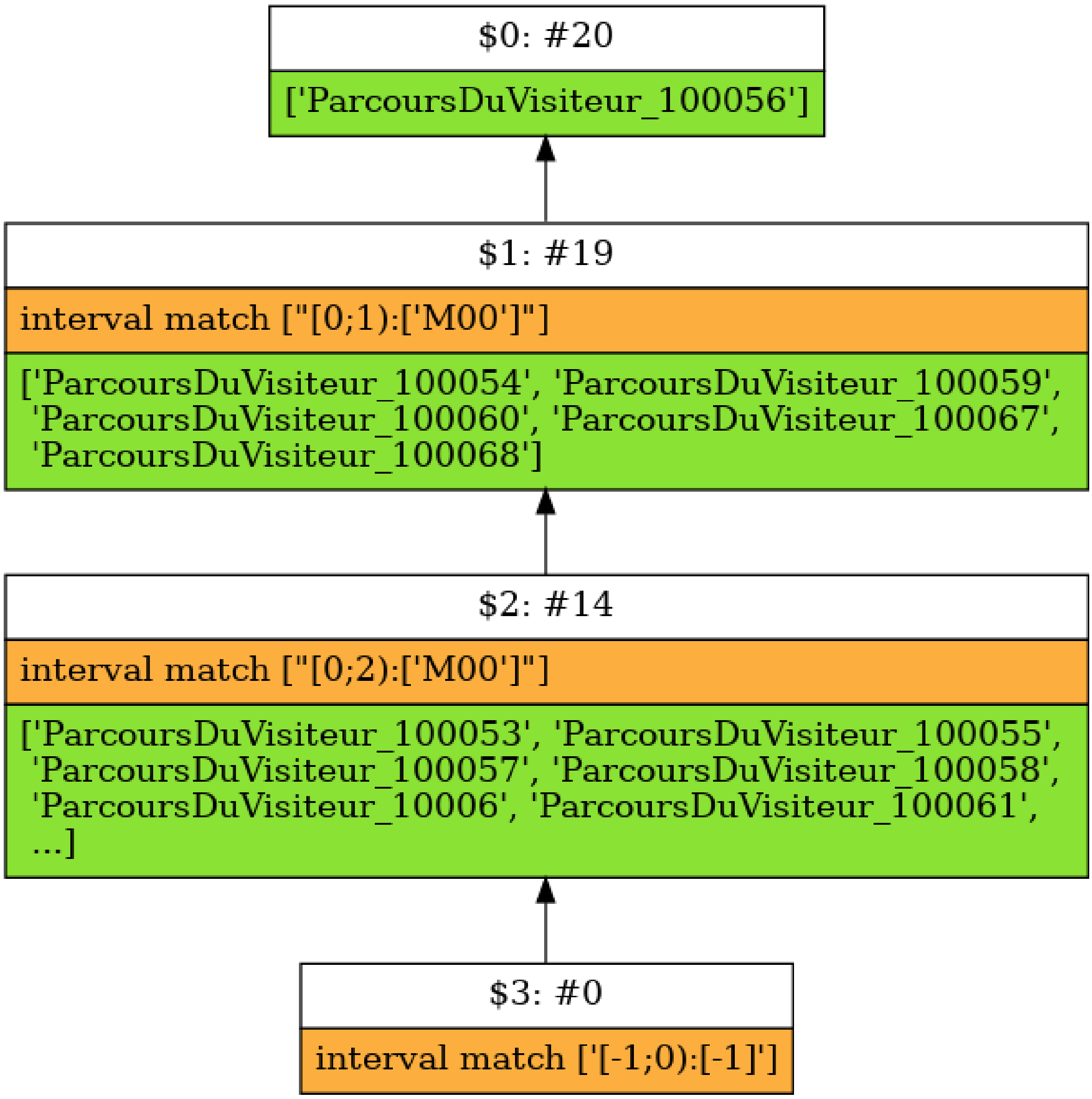
3.4.1.1 Simple Match Strategy

```python
explorer_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "explorers",
    "interval-sequence",
    "atf-match-basic.yaml"
)
with open(explorer_path, "r") as explorer_file:
    print(explorer_file.read())
    explorer_file.seek(0)
    explorer = Explorer.from_file(explorer_file)
```


```yaml
characteristics:
- &id001 !characteristic.interval.Interval
  characteristic: !characteristic.core.Key
    name: "interval"
descriptions:
- !description.interval.MaximalCommonInterval
  - *id001
strategies:
- !strategy.interval.intersection.basic.AlphabetTimeFrame
  arguments:
  - *id001
```

```python
lattice = ConceptLattice.create(
    population=population,
    descriptions=explorer.descriptions,
    strategies=explorer.strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```




```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2
['ParcoursDuVisiteur_100053', 'ParcoursDuVisiteur_100055', 'ParcoursDuVisiteur_100057', 'ParcoursDuVisiteur_100058', 'ParcoursDuVisiteur_10006', 'ParcoursDuVisiteur_100061', ...]	✓		✓
['ParcoursDuVisiteur_100053', 'ParcoursDuVisiteur_100054', 'ParcoursDuVisiteur_100055', 'ParcoursDuVisiteur_100057', 'ParcoursDuVisiteur_100058', 'ParcoursDuVisiteur_100059', ...]	✓		
['ParcoursDuVisiteur_100053', 'ParcoursDuVisiteur_100054', 'ParcoursDuVisiteur_100055', 'ParcoursDuVisiteur_100056', 'ParcoursDuVisiteur_100057', 'ParcoursDuVisiteur_100058', ...]			

```
ConceptTable(lattice)
```

Concept	Individuals	Predicates
0	ParcoursDuVisiteur_100056	interval match ['[-1;0):[-1]']
1	['ParcoursDuVisiteur_100054', 'ParcoursDuVisiteur_100059', 'ParcoursDuVisiteur_100060', 'ParcoursDuVisiteur_100067', 'ParcoursDuVisiteur_100068']	
2	['ParcoursDuVisiteur_100053', 'ParcoursDuVisiteur_100055', 'ParcoursDuVisiteur_100057', 'ParcoursDuVisiteur_100058', 'ParcoursDuVisiteur_10006', 'ParcoursDuVisiteur_100061', ...]	

3.4.2 GeoLuciole data set

Is issued from classical GPS trajectories of people's deplacements in the city of La Rochelle in France. By matching the GPS coordinates to districts of the city, raw data are transformed into semantic sequences. The data have been collected by a specific application named GeoLuciole that we have developed for the DA3T1. project. The data contains only 15 trajectories with an average size of sequences equals to 2 2.

1 : System for the Analysis of Numerical Traces for the development of Tourist Territories (Dispositif d'Analyse des Traces numériques pour la valorisation des Territoires Touristiques)

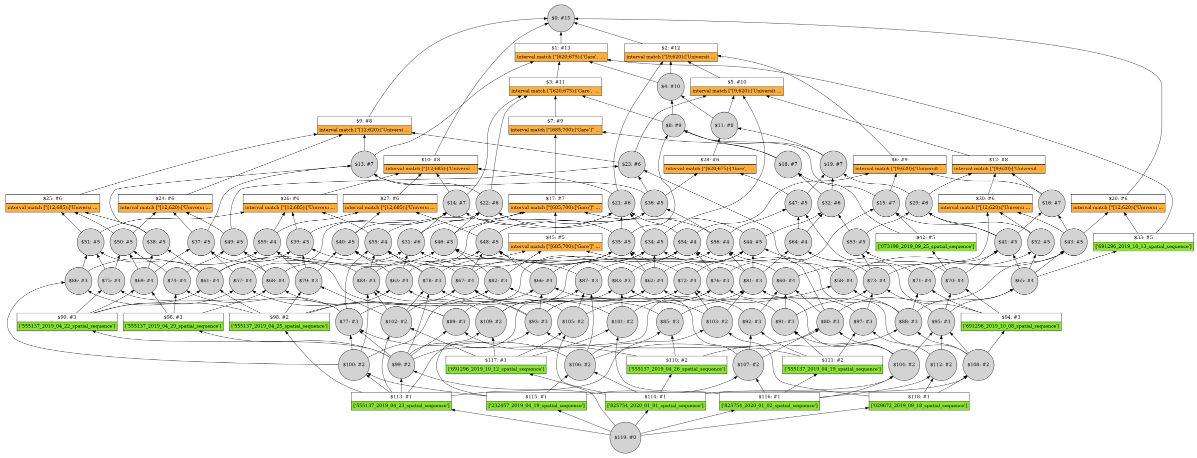
2 : It was planned to collect more data during the holidays on Mars and April, but unfortunately, this was impossible due to the world pandemic Covid-19

```python
data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "GeoLuciole",
    "interval-geoluciole-15.yml"
)
with open(data_path, "r") as data_file:
    population = Population.from_file(data_file)
population
len(population)
```

```
15
```

```python
from galactic_strategy_interval_sequence_basic import (
    AugmentedMinimumCardinalityStrategy,
    AlphabetTimeFrameStrategy,
    BoundsTimeFrameStrategy,
    SimpleTimeFrameStrategy,
    WindowAffixTimeFrameStrategy
)
from galactic_description_interval_sequence import (
    SharedIntervalDescription
)
from galactic.strategies import LimitFilter
from galactic_characteristic_interval import Interval
from galactic.characteristics import Key
from galactic.strategies import Cardinality

characteristics = [
    Interval(characteristic=Key(name="interval"))
]
descriptions = [
    SharedIntervalDescription(characteristics[0])
]
strategies = [
    LimitFilter(
        BoundsTimeFrameStrategy(characteristics[0]),
        measure=Cardinality(),
        limit=5
    )
]
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer()
)
```



```
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10	@11	@12	@13	@14	@15	@16	@17	@18	@19	@20	@21	@22
555137_2019_04_23_spatial_sequence		✓	✓	✓	✓			✓	✓			✓	✓	✓	✓	✓	✓			✓		✓	✓
825754_2020_01_01_spatial_sequence	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
232457_2019_04_19_spatial_sequence	✓	✓	✓		✓		✓				✓		✓	✓	✓	✓			✓		✓	✓	
825754_2020_01_02_spatial_sequence	✓		✓	✓	✓			✓	✓	✓	✓					✓	✓	✓	✓	✓	✓	✓	✓
691296_2019_10_12_spatial_sequence			✓	✓	✓										✓	✓		✓			✓		✓
029672_2019_09_18_spatial_sequence	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓					✓	✓	✓	✓	✓	✓	✓
['555137_2019_…tial_sequence', '825754_2020_…tial_sequence']			✓	✓	✓			✓	✓	✓	✓					✓	✓		✓	✓		✓	✓
['029672_2019_…tial_sequence', '691296_2019_…tial_sequence', '825754_2020_…tial_sequence']			✓	✓						✓		✓						✓	✓	✓	✓	✓	✓
['029672_2019_…tial_sequence', '825754_2020_…tial_sequence', '825754_2020_…tial_sequence']	✓		✓	✓				✓		✓	✓						✓	✓	✓	✓	✓	✓	✓
['555137_2019_…tial_sequence', '825754_2020_…tial_sequence']	✓	✓	✓	✓		✓	✓			✓	✓	✓	✓					✓	✓		✓	✓	✓

['555137_2019_…tial_sequence', '555137_2019_…tial_sequence', '825754_2020_…tial_sequence']		✓		✓	✓								✓	✓	✓		✓			✓			✓
['555137_2019_…tial_sequence', '555137_2019_…tial_sequence', '825754_2020_…tial_sequence']		✓	✓	✓								✓	✓	✓								✓	
['555137_2019_…tial_sequence', '555137_2019_…tial_sequence']		✓		✓	✓				✓					✓	✓	✓	✓			✓			✓
['555137_2019_…tial_sequence', '555137_2019_…tial_sequence', '825754_2020_…tial_sequence']		✓	✓	✓								✓	✓									✓	✓
['029672_2019_…tial_sequence', '555137_2019_…tial_sequence', '825754_2020_…tial_sequence']	✓	✓	✓	✓		✓	✓			✓	✓	✓						✓	✓		✓	✓	✓
['029672_2019_…tial_sequence', '073198_2019_…tial_sequence', '691296_2019_…tial_sequence', '825754_2020_…tial_sequence', '825754_2020_…tial_sequence']			✓	✓														✓		✓	✓		✓
['029672_2019_…tial_sequence', '555137_2019_…tial_sequence', '825754_2020_…tial_sequence', '825754_2020_…tial_sequence']	✓		✓	✓						✓	✓							✓	✓		✓	✓	✓
['029672_2019_…tial_sequence', '555137_2019_…tial_sequence', '691296_2019_…tial_sequence', '825754_2020_…tial_sequence', '825754_2020_…tial_sequence']	✓			✓																			

```python
ConceptTable(
    lattice,
    concept_width=5,
    individual_width=20,
    predicate_width=40
)
```

Concept	Individuals	Predicates
0		interval match ["[9;620):['Universite']", "[620;675):['Gare', 'Saint-Nicolas', 'Universite']", "[675;685):['Notre-Dame_Arsenal', 'Universite']", "[685;700):['Gare', 'Notre-Dame_Arsenal', 'Universite']", "[700;736):['Notre-Dame_Arsenal', 'Universite']", "[736;868):['Notre-Dame_Arsenal', 'Port_de_plaisance', 'Universite']", "[868;934):['Le_gabut', 'Notre-Dame_Arsenal', 'Port_de_plaisance', 'Saint-Nicolas', 'Universite']", "[934;953):['Notre-Dame_Arsenal',

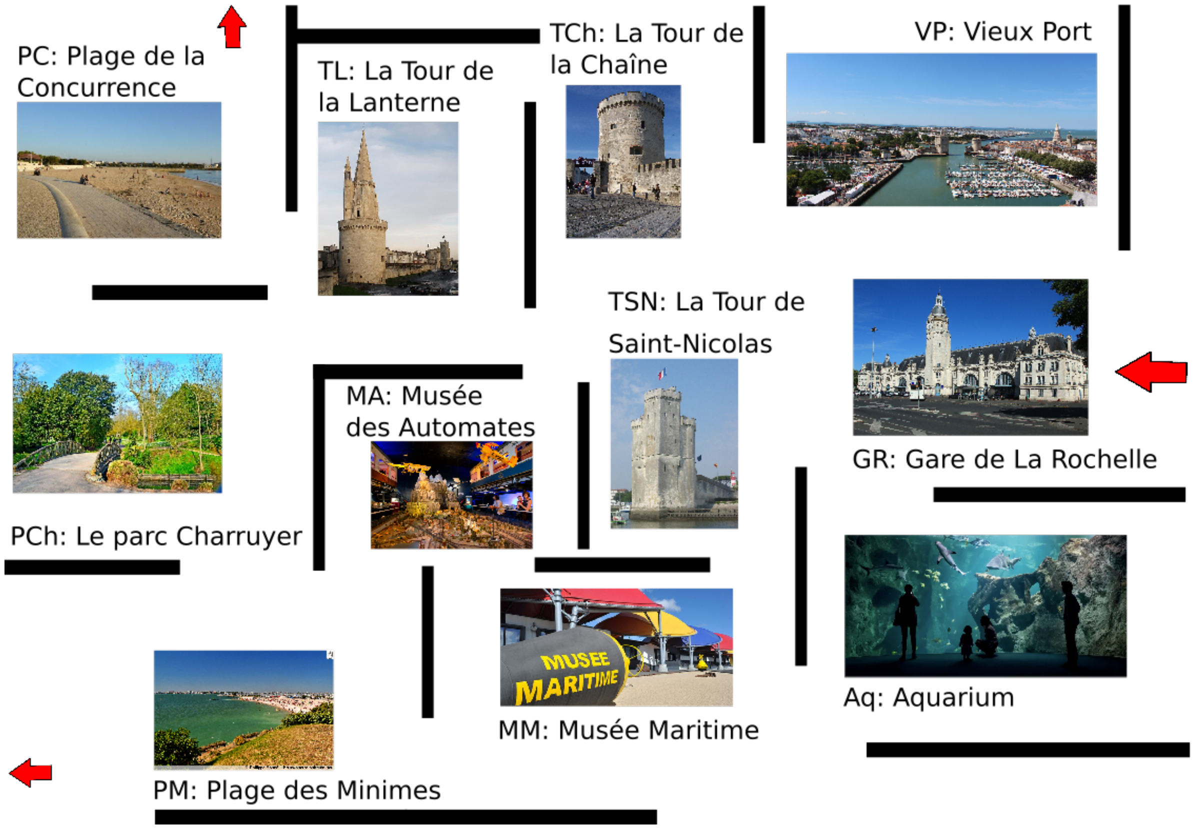
16	558026_2019_07_11_spatial_sequence
33	691296_2019_10_13_spatial_sequence
42	073198_2019_09_25_spatial_sequence
90	555137_2019_04_22_spatial_sequence
94	691296_2019_10_08_spatial_sequence
96	555137_2019_04_29_spatial_sequence
98	555137_2019_04_25_spatial_sequence
110	555137_2019_04_26_spatial_sequence
111	555137_2019_04_19_spatial_sequence
113	555137_2019_04_23_spatial_sequence
114	825754_2020_01_01_spatial_sequence
115	232457_2019_04_19_spatial_sequence
116	825754_2020_01_02_spatial_sequence
117	691296_2019_10_12_spatial_sequence
118	029672_2019_09_18_spatial_sequence

3.5 Science Party data

3.5.1 Fête de Science

3.5.1.1 Exemple de Labyrinthe

```python
from project_data import share_path
from IPython.display import Image
import sys
import os
data_path = os.path.join(
    share_path,
    "sequence",
    "data",
    "images",
    "labyrinthe-f.png"
)

Image(filename=data_path)
```

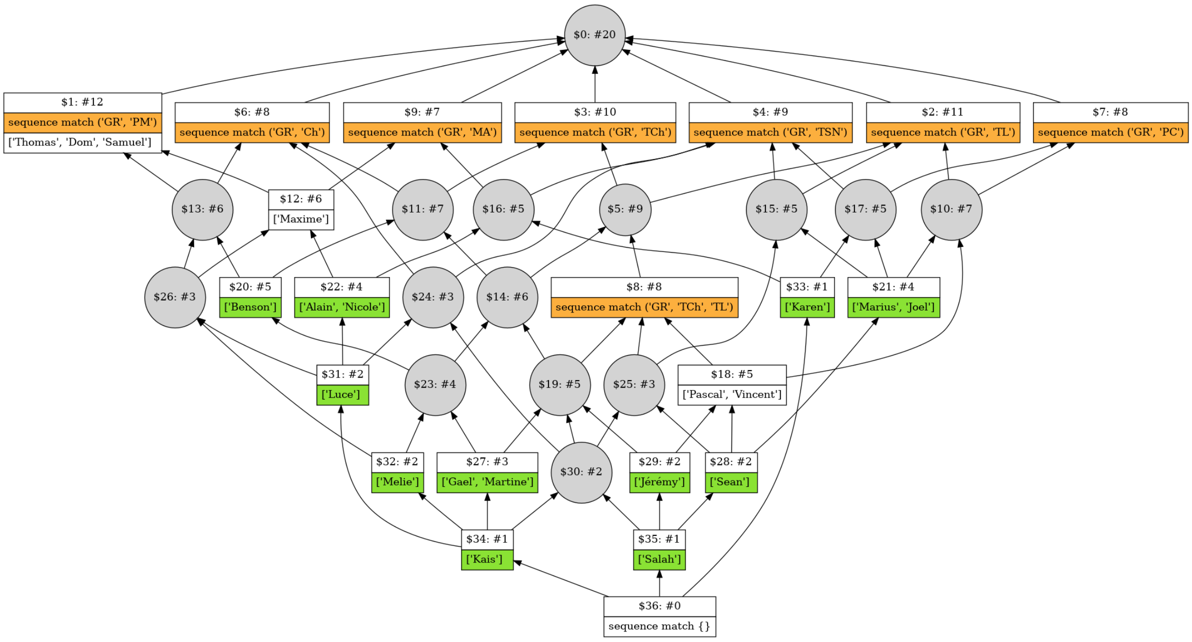



```python
from galactic.population import Population
from galactic.concepts import (
    ConceptLattice,
    ConceptTable
)
from galactic.concepts import (
    ConceptRenderer,
    ConceptTable
)
from galactic.algebras.poset import HasseDiagram
from galactic_strategy_chain_match_basic import (
    CompleteMatchStrategy,
    PrefixMatchStrategy
)
from galactic_description_chain_match import (
    CompleteDescription,
    PrefixDescription
)
from galactic.strategies import LimitFilter
from galactic_characteristic_chain import Chain
from galactic.characteristics import Key
from galactic.strategies import Cardinality
from galactic.algebras.relational import BinaryTable
```

```python
# Alphabet: "GR", "TSN", "TCh", "TL", "Ch", "PC", "VP", "MA", "PM", "MM",
↪  "Aq"
data = {
    "Salah" : {"sequence": ["GR", "TSN", "TCh", "TL", "Ch", "PC"]},
    "Jérémy" : {"sequence": ["GR" , "VP", "TCh","TL","Ch","PC"]},
    "Thomas" : {"sequence": ["GR", "Aq", "MM", "PM"]},
    "Dom" : {"sequence": ["GR", "Aq", "MM", "PM"]},
    "Kais" : {"sequence": ["GR", "TCh", "TL", "TSN", "MM", "MA", "Ch",
    ↪  "TL", "TSN", "MM", "PM"]},
    "Marius" : {"sequence": ["GR", "TSN", "TL", "PC"]},
    "Joel" : {"sequence": ["GR", "TSN", "TL", "PC"]},
    "Alain" : {"sequence": ["GR", "TSN", "MM", "MA", "PM"]},
    "Sean" : {"sequence": ["GR", "TSN", "TCh", "TL", "PC"]},
    "Samuel" : {"sequence": ["GR", "Aq", "MM", "PM"]},
    "Benson" : {"sequence": ["GR", "TCh", "MM", "Ch", "PM"]},
    "Gael" : {"sequence": ["GR", "VP", "TCh", "TL", "Ch", "PM"]},
    "Martine" : {"sequence": ["GR", "VP", "TCh", "TL", "Ch", "PM"]},
    "Melie" : {"sequence": ["GR", "Aq", "TL", "TCh", "MA", "Ch", "PM"]},
    "Luce" : {"sequence": ["GR", "TSN", "MM", "Aq", "MA", "Ch", "PM"]},
    "Pascal" : {"sequence": ["GR", "VP", "TCh", "TL", "PC"]},
    "Vincent" : {"sequence": ["GR", "VP", "TCh", "TL", "PC"]},
    "Nicole" : {"sequence": ["GR", "Aq", "TSN", "MM", "MA","PM"]},
    "Karen" : {"sequence": ["GR", "TSN", "MA", "PC"]},
    "Maxime" : {"sequence": ["GR", "MM", "MA", "PM"]},
}
population = Population(data)
```

```python
characteristics = [
    Chain(characteristic=Key(name="sequence"))
]
descriptions = [
    CompleteDescription(characteristics[0])
]
strategies = [
    CompleteMatchStrategy(characteristics[0])
]
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies
)
```

```python
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(
        width=80,
        show_individuals=True,
        show_predicates=True)
)
```




```python
BinaryTable(
    lattice.reduced_context,
    domain_renderer=ConceptRenderer(join_irreducible=True),
    co_domain_renderer=ConceptRenderer(meet_irreducible=True)
)
```

	@0	@1	@2	@3	@4	@5	@6	@7	@8
Karen	✓		✓				✓		
Kais	✓	✓	✓	✓	✓	✓		✓	✓
Salah		✓	✓	✓		✓	✓		✓
['Salah', 'Sean']			✓	✓		✓	✓		✓
['Salah', 'Jérémy']		✓		✓		✓	✓		✓
['Kais', 'Gael', 'Martine']		✓		✓	✓	✓			✓
['Kais', 'Luce']	✓	✓	✓		✓				
['Kais', 'Melie']	✓	✓		✓	✓	✓		✓	

['Kais', 'Alain', 'Luce', 'Nicole']	✓		✓		✓		
['Salah', 'Marius', 'Joel', 'Sean']			✓	✓			✓
['Kais', 'Gael', 'Martine', 'Melie']		✓		✓	✓	✓	
['Kais', 'Benson', 'Gael', 'Martine', 'Melie']		✓			✓	✓	

```python
ConceptTable(
    lattice,
    concept_width=5,
    individual_width=10,
    predicate_width=40
)
```

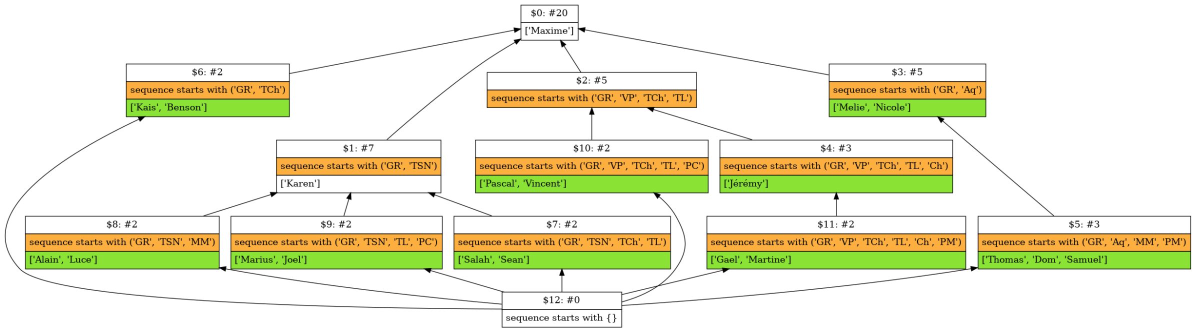
Concept	Individuals	Predicates
0		`sequence match ('GR',)`
1	`['Thomas', 'Dom', 'Samuel']`	`sequence match ('GR', 'PM')`
2		`sequence match ('GR', 'TL')`
3		`sequence match ('GR', 'TCh')`
4		`sequence match ('GR', 'TSN')`
6		`sequence match ('GR', 'Ch')`

7		sequence match ('GR', 'PC')
8		sequence match ('GR', 'TCh', 'TL')
9		sequence match ('GR', 'MA')
10		sequence match ('GR', 'TL', 'PC')
11		sequence match ('GR', 'TCh', 'Ch')
12	Maxime	sequence match ('GR', 'MA', 'PM')
13		sequence match ('GR', 'Ch', 'PM')
14		sequence match ('GR', 'TL', 'Ch')
15		sequence match ('GR', 'TSN', 'TL')
16		sequence match ('GR', 'TSN', 'MA')
17		sequence match ('GR', 'TSN', 'PC')
18	['Pascal', 'Vincent']	sequence match ('GR', 'TCh', 'TL', 'PC')
19		sequence match ('GR', 'TCh', 'TL', 'Ch')
20	Benson	sequence match ('GR', 'TCh', 'Ch', 'PM')
21	['Marius', 'Joel']	sequence match ('GR', 'TSN', 'TL', 'PC')
22	['Alain', 'Nicole']	sequence match ('GR', 'TSN', 'MM', 'MA', 'PM')
23		sequence match ('GR', 'TL', 'Ch', 'PM')
24		sequence match ('GR', 'TSN', 'Ch')
26		sequence match ('GR', 'MA', 'Ch', 'PM')
27	['Gael', 'Martine']	sequence match ('GR', 'TCh', 'TL', 'Ch', 'PM')
28	Sean	sequence match ('GR', 'TSN', 'TCh', 'TL', 'PC')
29	Jérémy	sequence match ('GR', 'TCh', 'TL', 'Ch', 'PC')

31	Luce	sequence match ('GR', 'TSN', 'MM', 'MA', 'Ch', 'PM')
32	Melie	sequence match ('GR', 'TL', 'MA', 'Ch', 'PM')sequence match ('GR', 'TCh', 'MA', 'Ch', 'PM')
33	Karen	sequence match ('GR', 'TSN', 'MA', 'PC')
34	Kais	sequence match ('GR', 'TCh', 'TL', 'TSN', 'MM', 'MA', 'Ch', 'TL', 'TSN', 'MM', 'PM')
35	Salah	sequence match ('GR', 'TSN', 'TCh', 'TL', 'Ch', 'PC')
36		sequence match {}

3.5.2 Avec limitation de cardinalité

```python
characteristics = [
    Chain(characteristic=Key(name="sequence"))
]
descriptions = [
    PrefixDescription(characteristics[0])
]
strategies = [
    LimitFilter(
        PrefixMatchStrategy(characteristics[0]),
        measure=Cardinality(),
        limit=2
    )
]
lattice = ConceptLattice.create(
    population=population,
    descriptions=descriptions,
    strategies=strategies
)
HasseDiagram(
    lattice,
    domain_renderer=ConceptRenderer(
        width=80,
        show_individuals=True,
        show_predicates=True
    )
)
```


$0: #20
['Maxime']
$6: #2
sequence starts with ('GR', 'TCh')
['Kais', 'Benson']
$2: #5
sequence starts with ('GR', 'VP', 'TCh', 'TL')
$3: #5
sequence starts with ('GR', 'Aq')
['Melie', 'Nicole']
$1: #7
sequence starts with ('GR', 'TSN')
['Karen']
$10: #2
sequence starts with ('GR', 'VP', 'TCh', 'TL', 'PC')
['Pascal', 'Vincent']
$4: #3
sequence starts with ('GR', 'VP', 'TCh', 'TL', 'Ch')
['Jérémy']
$8: #2
sequence starts with ('GR', 'TSN', 'MM')
['Alain', 'Luce']
$9: #2
sequence starts with ('GR', 'TSN', 'TL', 'PC')
['Marius', 'Joel']
$7: #2
sequence starts with ('GR', 'TSN', 'TCh', 'TL')
['Salah', 'Sean']
$11: #2
sequence starts with ('GR', 'VP', 'TCh', 'TL', 'Ch', 'PM')
['Gael', 'Martine']
$5: #3
sequence starts with ('GR', 'Aq', 'MM', 'PM')
['Thomas', 'Dom', 'Samuel']
$12: #0
sequence starts with {}