



Human-centric algebraic machine learning

AML-DL Debugger - Initial

Deliverable D3.10

WP3 - AML Description Language



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952091.

Consortium



ALGEBRAIC AI



Chantalimaud
Foundation



German
Research Center
for Artificial
Intelligence



TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN



Project

Title: ALMA: Human Centric Algebraic Machine Learning
 Acronym: ALMA
 Coordinator: Proyectos y Sistemas de Mantenimiento SL (eProsima)
 Grant agreement: 952091
 Call: H2020-EIC-FETPROACT-2019
 Program: Horizon 2020
 Start: 01 September 2020
 Duration: 48 months
 Website: <https://alma-ai.eu>
 E-mail: alma@eprosima.com
 Consortium: **Proyectos y Sistemas de Mantenimiento SL (eProsima) (EPROS)**, Spain
Champalimaud Foundation (CF), Portugal
ALGEBRAIC AI (ALG), Spain
German Research Center for Artificial Intelligence (DFKI), Germany
Inria (Inria), France
TU Kaiserslautern (TUK), Germany
FIWARE Foundation e.V. (FF), Germany
Technical Research Centre of Finland (VTT), Finland
Carlos III University (UC3M), Spain

Deliverable

Number: **D3.10**
 Title: **AML-DL Debugger - Initial**
 Month: 36
 Work Package: WP3 - AML Description Language
 Work Package leader: Algebraic AI (ALG)
 Deliverable leader: Algebraic AI (ALG)
 Deliverable type: Report (R)
 Dissemination level: Public (PU)
 Date of submission: 2023-09-29
 Version: v1.0
 Status: Finished

Version history

Version	Date	Responsible	Author/Reviewer	Comments
v0.1	2023-06	ALG	Nabil Abderrahaman-Elena	Draft
v0.2	2023-06	ALG	Fernando Martin-Maroto	Review
v0.3	2023-07	ALG	Nabil Abderrahaman-Elena	Review
v0.4	2023-07	ALG	Fernando Martin-Maroto	Final review
0.3	2023-07	ALG	Nabil Abderrahaman-Elena	Corrections

Executive summary

Algebraic Machine Learning (AML) is a novel machine learning technique based on algebraic representations of data. Unlike statistical learning, AML algorithms are robust regarding the statistical properties of the data and are parameter-free. AML systems can retain what they learn, including sociocultural ethical practices for human–artificial intelligence systems. The aim of the EU-funded ALMA project is to study the mathematical properties of AML and to leverage them to develop a new generation of interactive human-centric machine learning systems. These systems are intended to minimize bias in order to facilitate trust while better enabling distributed collaborative learning.

The AML-DL Debugger is a software tool to visually explore semantic embeddings built using AML-DL. Given a semantic embedding expressed using the AML Description Language, it can display its constants, positive and negative duples, as well as determine its consistency. The AML-DL Debugger has been built in a modular way, so it is simple to extend with new functionality.

This deliverable contains the AMLDL Consistency Checker as well as examples of consistent and inconsistent embeddings.

Contents

Executive Summary	3
Contents	4
Acronyms	5
1 AML-DL Debugger	6
1.1 Components of the AML-DL Debugger (AML-DL Debugger)	6
2 Demonstrator	6
2.1 Instructions	6
2.2 Examples	6
3 Future work	7
3.1 Current state	7
3.2 Potential future enhancements	7
4 AML-DL Debugger walkthrough	7
4.1 Constants definition	8
4.2 Relations declaration	8
4.3 Consistency	11
References	14

Acronyms

AI	Artificial Intelligence.
AML	Algebraic Machine Learning.
AML-DL	AML Description Language.
AML-DL Debugger	AML-DL Debugger.
ML	Machine Learning.
TUI	Text User Interface.

1 AML-DL Debugger

AML[1] is a new Artificial Intelligence (AI) paradigm that combines user-defined symbols with self-generated symbols. This permits AML to learn from the data and adapt to the world as neural networks do, combined with the power for explainability of Symbolic AI. AML is a purely symbolic approach and neither uses neurons nor is a neuro-symbolic method. AML does not use parameters and it does not rely on fitting, regression, backtracking, constraint satisfiability, logical rules, production rules, or error minimization.

The user should describe the Machine Learning (ML) problem at hand using a set of constants, that represent concepts, and a set of algebraic order equations and inequations, in the form of positive and negative duples, that represent relationships between subsets of concepts. These equations are referred to as the *embedding*, and the solutions of the embedding as the *model*. If the set of equations has at least one solution the embedding is said to be consistent. Inconsistent embeddings have no solutions, i.e. they have no representation as semilattice models. Further detail about semantic embeddings is provided in *Deliverable D3.6 - AML-DL Interpreter - Revised* and [2].

The AML-DL Debugger is a software tool to visualize semantic embeddings and some of their properties. Currently the AML-DL Debugger can

- organize and display the embedding constants,
- display positive and negative duples using their given name instead of their indices,
- study and explore embedding consistency,
- be extended to implement new modules.

These properties allow for a quick feedback loop when developing or modifying semantic embedding using AML Description Language (AML-DL).

1.1 Components of the AML-DL Debugger

The AML-DL Debugger combines a user interface and a module loader:

- A Text User Interface (TUI) that displays several options in text view, so it can easily be used in remote machines or from a docker container.
- A module loader that allows for new tools to be easily added.

2 Demonstrator

This deliverable contains the AML-DL Debugger. Two consistent and two inconsistent semantic embeddings are also provided as examples.

2.1 Instructions

Instructions for installation and usage are included in the `README.md` file together with the program.

2.2 Examples

Two consistent embeddings are provided for illustrative purposes. The same two embedding are also provided with an additional AML-DL code line that turns them inconsistent.

The two examples included correspond to the algebraic semantic embedding of the n-queens completion problem, as described in [1], and the embedding of a cyclic group [2].

3 Future work

3.1 Current state

The current AML-DL Debugger is fully functional and in use. It can help debugging hard and complex semantic embeddings such as the ones provided as examples.

3.2 Potential future enhancements

Currently, the TUI and the module loader are intertwined to some extent. Further enhancements are planned to decouple them to simplify and automate the process of adding new tools. Additionally, more user experience (UX) enhancements are planned, with special consideration for newcomers to this technology. New functionalities for debugging of semantic embeddings could also be developed and integrated into this tool.

4 AML-DL Debugger walkthrough

In [3] the semantic embedding for the n-queens completion problem is described. Here we follow the third variant and use it to illustrate the development of a semantic embedding in AML-DL using the AML-DL Debugger.

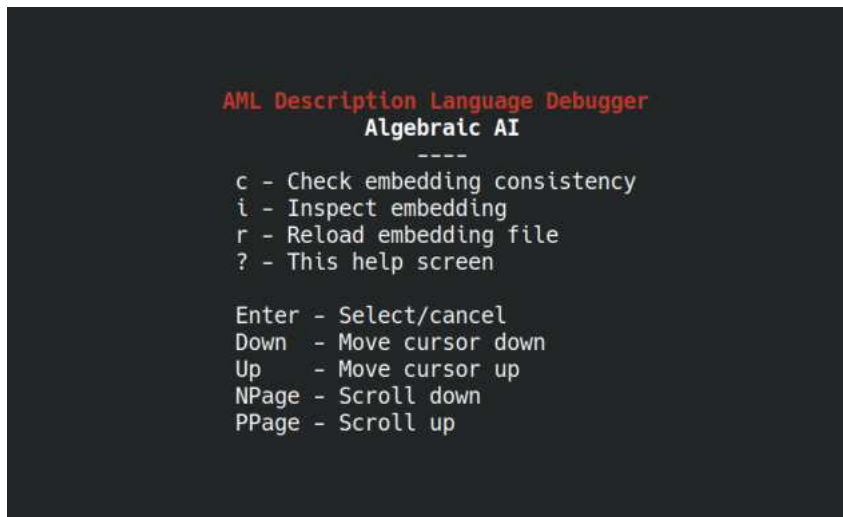
Several embeddings are provided together with the AML-DL Debugger. To load one of them and start the application we execute the following command:

```
1 python amlDebugger.py <embedding file> [additional parameters]
```

For instance, for a 4x4 board we would use the command:

```
1 python amlDebugger.py embeddings/queensEmbedding.py 4
```

Figure 1 shows the main menu of the application with the instructions.



```
AML Description Language Debugger
Algebraic AI
----
c - Check embedding consistency
i - Inspect embedding
r - Reload embedding file
? - This help screen

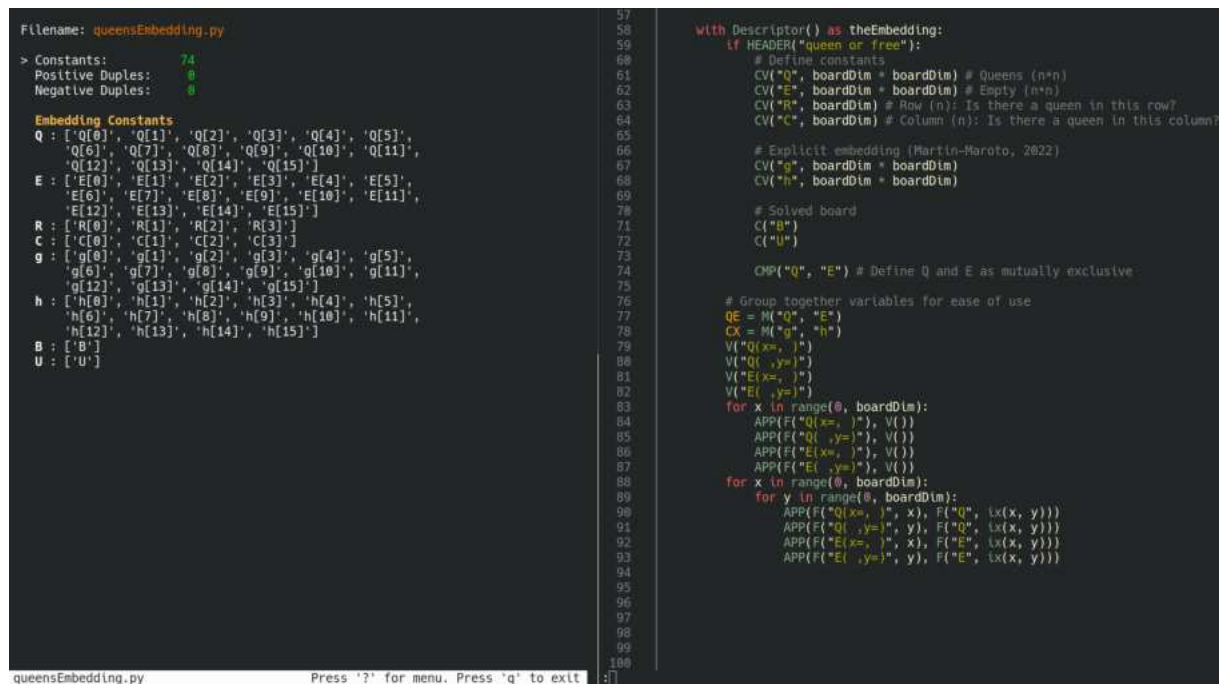
Enter - Select/cancel
Down  - Move cursor down
Up    - Move cursor up
NPage - Scroll down
PPage - Scroll up
```

Figure 1: AML-DL Debugger main menu.

4.1 Constants definition

To write a semantic embedding, we first need to define the constants that describe the problem at hand. These constants encapsulate concepts, such as “queen at cell 3,2” or “there is at least one queen in row 5”, these concepts can later be combined to form larger, more complex ideas, or terms.

In the particular case of the third embedding proposed for the n-queens problem, we define constants such as having a queen at each cell, having empty cells, or having a solved board. Additionally, AML-DL allows us to create intermediate variables that group together constants. Figure 2 displays the first step of definition of constant as well as the corresponding visualisation in the AML-DL Debugger. Notice that it is possible to create multiple constants using a single AML-DL operator.



The figure shows a screenshot of the AML-DL Debugger interface. The left pane displays the file `queensEmbedding.py` with the following content:

```

> Constants: 74
Positive Dupes: 0
Negative Dupes: 0

Embedding Constants
Q : ['Q[0]', 'Q[1]', 'Q[2]', 'Q[3]', 'Q[4]', 'Q[5]',
     'Q[6]', 'Q[7]', 'Q[8]', 'Q[9]', 'Q[10]', 'Q[11]',
     'Q[12]', 'Q[13]', 'Q[14]', 'Q[15]']
E : ['E[0]', 'E[1]', 'E[2]', 'E[3]', 'E[4]', 'E[5]',
     'E[6]', 'E[7]', 'E[8]', 'E[9]', 'E[10]', 'E[11]',
     'E[12]', 'E[13]', 'E[14]', 'E[15]']
R : ['R[0]', 'R[1]', 'R[2]', 'R[3]']
C : ['C[0]', 'C[1]', 'C[2]', 'C[3]']
g : ['g[0]', 'g[1]', 'g[2]', 'g[3]', 'g[4]', 'g[5]',
     'g[6]', 'g[7]', 'g[8]', 'g[9]', 'g[10]', 'g[11]',
     'g[12]', 'g[13]', 'g[14]', 'g[15]']
h : ['h[0]', 'h[1]', 'h[2]', 'h[3]', 'h[4]', 'h[5]',
     'h[6]', 'h[7]', 'h[8]', 'h[9]', 'h[10]', 'h[11]',
     'h[12]', 'h[13]', 'h[14]', 'h[15]']
B : ['B']
U : ['U']

```

The right pane shows the AML-DL code defining the constants:

```

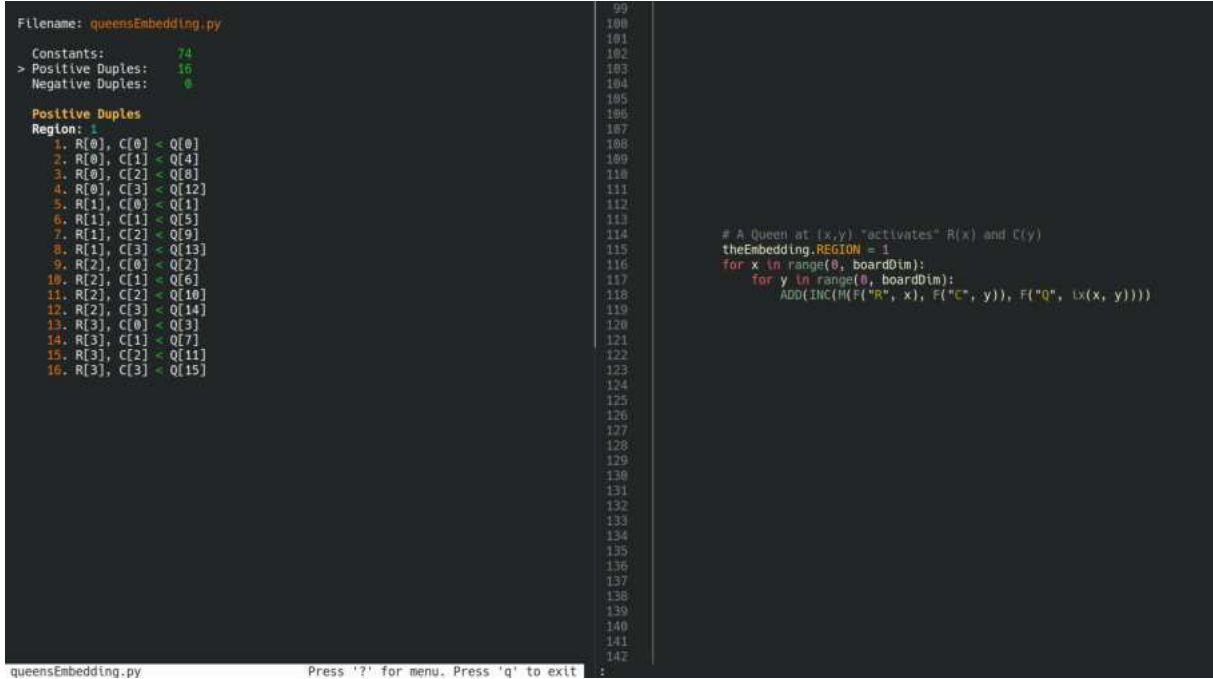
57
58
59
60 with Descriptor() as theEmbedding:
61     if HEADER("queen or free"):
62         # Define constants
63         CV("Q", boardDim * boardDim) # Queens (n*n)
64         CV("E", boardDim * boardDim) # Empty (n*n)
65         CV("R", boardDim) # Row (n): Is there a queen in this row?
66         CV("C", boardDim) # Column (n): Is there a queen in this column?
67
68         # Explicit embedding (Martin-Maroto, 2022)
69         CV("g", boardDim * boardDim)
70         CV("h", boardDim * boardDim)
71
72         # Solved board
73         C("B")
74         C("U")
75
76         CMP("Q", "E") # Define Q and E as mutually exclusive
77
78         # Group together variables for ease of use
79         QE = M("Q", "E")
80         CX = M("g", "h")
81         V("Q(x=, )")
82         V("Q(, y=)")
83         V("E(x=, )")
84         V("E(, y=)")
85
86         for x in range(0, boardDim):
87             APP(F("Q(x=, )", x), V())
88             APP(F("Q(, y=)", y), V())
89             APP(F("E(x=, )", x), V())
90             APP(F("E(, y=)", y), V())
91
92         for x in range(0, boardDim):
93             for y in range(0, boardDim):
94                 APP(F("Q(x=, )", x), F("Q", lx(x, y)))
95                 APP(F("Q(, y=)", y), F("Q", lx(x, y)))
96                 APP(F("E(x=, )", x), F("E", lx(x, y)))
97                 APP(F("E(, y=)", y), F("E", lx(x, y)))
98
99
100

```

Figure 2: Right: Definition of constants in the semantic embedding of the n-queens completion problem using AML-DL. Left: Visualisation of the constants in the embedding using the interpreter module.

4.2 Relations declaration

Once the embedding constants have been defined, it is possible to use them to build terms. The positive or negative relationship between terms is expressed using relations. In figure 3 we add a set of positive relations. The *ADD* command adds a relation to the final embedding. The commands *INC* and *EXC* create a positive and a negative relation, respectively. The *M* command merges constants, giving as a result a term. And the *F* command performs a lookup in the set of constants, and in particular, finds the *i*-th element in a vector.



```

Filename: queensEmbedding.py

Constants: 74
> Positive Duples: 16
Negative Duples: 0

Positive Duples
Region: 1
1. R[0], C[0] < Q[0]
2. R[0], C[1] < Q[4]
3. R[0], C[2] < Q[8]
4. R[0], C[3] < Q[12]
5. R[1], C[0] < Q[1]
6. R[1], C[1] < Q[5]
7. R[1], C[2] < Q[9]
8. R[1], C[3] < Q[13]
9. R[2], C[0] < Q[2]
10. R[2], C[1] < Q[6]
11. R[2], C[2] < Q[10]
12. R[2], C[3] < Q[14]
13. R[3], C[0] < Q[3]
14. R[3], C[1] < Q[7]
15. R[3], C[2] < Q[11]
16. R[3], C[3] < Q[15]

# A Queen at (x,y) "activates" R(x) and C(y)
theEmbedding.REGION = 1
for x in range(0, boardDim):
    for y in range(0, boardDim):
        ADD(INC(M(F("R", x), F("C", y)), F("Q", Lx(x, y))))

```

Figure 3: Right: Declaration of a first set of positive relations in the semantic embedding of the n-queens completion problem using AML-DL. Left: Visualisation of the corresponding positive duples that are created in the embedding using the interpreter module.

Figure 4 illustrates how we can slowly add more relations, and see the corresponding changes that the interpreter module displays in the AML-DL Debugger. The amount of relations increase, and they get organised accordingly to their type and region.

```

Filename: queensEmbedding.py
Constants: 74
Positive Duples: 16
> Negative Duples: 8

Negative Duples
Region: 1
1. R[0] = Q[1], Q[2], Q[3], Q[5], Q[6], Q[7], Q[9], Q[10],
   Q[11], Q[13], Q[14], Q[15], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]
2. R[1] = Q[0], Q[2], Q[3], Q[4], Q[6], Q[7], Q[8], Q[10],
   Q[11], Q[12], Q[14], Q[15], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]
3. R[2] = Q[0], Q[1], Q[3], Q[4], Q[5], Q[7], Q[8], Q[9],
   Q[11], Q[12], Q[13], Q[15], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]
4. R[3] = Q[0], Q[1], Q[2], Q[4], Q[5], Q[6], Q[8], Q[9],
   Q[10], Q[12], Q[13], Q[14], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]

Region: 2
1. C[0] = Q[4], Q[5], Q[6], Q[7], Q[8], Q[9], Q[10], Q[11],
   Q[12], Q[13], Q[14], Q[15], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]
2. C[1] = Q[0], Q[1], Q[2], Q[3], Q[8], Q[9], Q[10], Q[11],
   Q[12], Q[13], Q[14], Q[15], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]
3. C[2] = Q[0], Q[1], Q[2], Q[3], Q[4], Q[5], Q[6], Q[7],
   Q[12], Q[13], Q[14], Q[15], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]
4. C[3] = Q[0], Q[1], Q[2], Q[3], Q[4], Q[5], Q[6], Q[7],
   Q[8], Q[9], Q[10], Q[11], E[0], E[1], E[2], E[3],
   E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
   E[12], E[13], E[14], E[15]

# A Queen at (x,y) "activates" R(x) and C(y)
theEmbedding.REGION = 1
for x in range(0, boardDim):
    for y in range(0, boardDim):
        ADD(INC(M(F("R", x), F("C", y)), F("Q", lx(x, y))))

theEmbedding.REGION = 1
# R(x) can only be produced by queens in the row x
ADD(EXC(T("R", _l), M(R("Q(x=", _l), T("Q(x=", _l), "E"))))

theEmbedding.REGION = 2
# C(y) can only be produced by queens in the column y
ADD(EXC(T("C", _l), M(R("Q(,y=", _l), T("Q(,y=", _l), "E"))))

```

Figure 4: Right: Declaration of two new sets of negative relations in the semantic embedding of the n-queens completion problem using AML-DL. These relations use the T operator, so the produce several duples. Left: Visualisation of the corresponding negative duples that are created in the embedding using the interpreter module.

As shown in figure 5, it is also possible to add fixed values to an embedding. Here we indicate that the solution has a queen at a certain position using a positive relation.

```

Filename: queensEmbedding.py
Constants: 112
> Positive Duples: 377
Negative Duples: 35

34. E[15], E[11], E[3] < U, Q[7]
35. E[16], E[4], E[20], E[8] < Q[12], U
36. E[13], E[9], E[21] < Q[17], U
37. E[18], E[14] < Q[22], U
38. E[15], E[7], E[11] < Q[3], U
39. E[16], E[4], E[12], E[20] < Q[8], U
40. E[17], E[9], E[21] < Q[13], U
41. E[14], E[22] < Q[18], U
42. E[19] < U, Q[23]
43. E[16], E[12], E[20], E[8] < Q[4], U
44. E[13], E[17], E[21] < Q[9], U
45. E[18], E[22] < Q[14], U
46. E[23] < Q[19], U

Region: 9
1. Q[12] < B

Region: 10
1. U < B
2. B < Q[0], Q[1], Q[2], Q[3], Q[4], Q[5], Q[6], Q[7],
   Q[8], Q[9], Q[10], Q[11], Q[12], Q[13], Q[14],
   Q[15], Q[16], Q[17], Q[18], Q[19], Q[20], Q[21],
   Q[22], Q[23], Q[24], E[0], E[1], E[2], E[3], E[4],
   E[5], E[6], E[7], E[8], E[9], E[10], E[11], E[12],
   E[13], E[14], E[15], E[16], E[17], E[18], E[19],
   E[20], E[21], E[22], E[23], E[24], U

Region: 12
1. R[0], R[1], R[2], R[3], R[4] < B
2. C[0], C[1], C[2], C[3], C[4] < B

Region: 13
1. Q[0], g[0] < g[0], B
2. Q[1], g[1] < g[1], B
3. Q[2], g[2] < g[2], B
4. Q[3], g[3] < B, g[3]

# preset some queens
theEmbedding.REGION = 9
if boardDim == 5:
    ADD(INC(F("Q", Lx(2, 2)), "B"))

```

Figure 5: Right: Declaration of a positive relations in the semantic embedding of the n-queens completion problem using AML-DL. This relation fixes the position of a Queen, and serves as an initial condition for the problem. Left: Visualisation of the corresponding positive duples that are created in the embedding using the interpreter module.

4.3 Consistency

The consistency checker module allows us to verify that the embedding is consistent from a mathematical point of view. Figure 6 indicates that the embedding that we have built contains at least one inconsistency. It points to the regions that most likely caused the issue. Then it is possible to analyse our embedding using a text editor and modify it until we fix the issue. During this process, it is possible to refresh the analysis dynamically on the AML-DL Debugger, without the need to exit and relaunch it.

```

Filename: queensEmbedding.py
Inconsistent embedding
Inconsistency in region: 1
R[0] < Q[1], Q[2], Q[3], Q[5], Q[6], Q[7], Q[9], Q[10],
Q[11], Q[13], Q[14], Q[15], E[0], E[1], E[2], E[3],
E[4], E[5], E[6], E[7], E[8], E[9], E[10], E[11],
E[12], E[13], E[14], E[15]

Potential conflict with region: 1
R[1], C[0] < Q[1]
R[1], C[1] < Q[5]
R[1], C[2] < Q[9]
R[1], C[3] < Q[13]
R[2], C[0] < Q[2]
R[3], C[0] < Q[3]
R[0], C[1] < Q[4]
Potential conflict with region: 14
Q[4] < E[4]

# A Queen at (x,y) "activates" R(x) and C(y)
theEmbedding.REGION = 1
for x in range(0, boardDim):
    for y in range(0, boardDim):
        ADD(INC(M(F("R", x), F("C", y)), F("Q", lx(x, y))))

theEmbedding.REGION = 1
# R(x) can only be produced by queens in the row x
ADD(EXC(T("R", _l), M(R(*Q(x=, _l), T(*Q(x=, _l), "E"))))

theEmbedding.REGION = 14
ADD(INC(F("Q", 4), F("E", 4)))

```

Figure 6: Left: Result of the analysis of the consistency of the semantic embedding using the consistency checker module. Right: Regions involved in the inconsistency as pointed out by the analysis.

When the semantic embedding is eventually fixed, the consistency checker displays the message shown in figure 7.

```

Filename: queensEmbedding.py
Consistent embedding
We have not been able to detect any inconsistency.
Logical sentences in the embedding provided
seem to not contradict each other.

# A Queen at (x,y) "activates" R(x) and C(y)
theEmbedding.REGION = 1
for x in range(0, boardDim):
    for y in range(0, boardDim):
        ADD(INC(M(F("R", x), F("C", y)), F("Q", lx(x, y))))

theEmbedding.REGION = 1
# R(x) can only be produced by queens in the row x
ADD(EXC(T("R", _l), M(R(*Q(x=, _l), T(*Q(x=, _l), "E"))))

# theEmbedding.REGION = 14
# ADD(INC(F("Q", 4), F("E", 4)))

```

Figure 7: Left: Result of the analysis of the consistency of the semantic embedding using the consistency checker module. Right: Regions that were initially involved in the inconsistency.



References

- [1] Fernando Martin-Maroto and Gonzalo G. de Polavieja. “Algebraic Machine Learning”. In: *arXiv:1803.05252* (2018).
- [2] Fernando Martin-Maroto and Gonzalo G. de Polavieja. “Semantic Embeddings in Semilattices”. In: *arXiv:2205.12618* (2022).
- [3] Fernando Martin-Maroto and Gonzalo G. de Polavieja. “Finite Atomized Semilattices”. In: *arXiv:2102.08050* (2021).