Innovating Sales and Planning of Complex Industrial Products
Exploiting Artificial Intelligence

Deliverable D3.6
Inference Engine

Deliverable type:	Software
Deliverable reference number:	ITEA 20054 | D3.6
Related Work Package:	WP 3
Due date:	2024-08-21
Actual submission date:	2024-08-30
Responsible organisation:	TUD
Editor:	Stefan Ellmauthaler
Dissemination level:	Public
Revision:	Final | Version 1.0

Abstract:	We present “Nemo”, an in-memory Datalog reasoner, which allows for an easy modelling of expert knowledge. The framework offers a web-interface, language server extensions, and multiple access points for design and usage.

Keywords:	Datalog, Existential Rules, Nemo language, Rules

Table_head	Name 1 (partner)	Name 2 (partner)	Approval date (1 / 2)
Approval at WP level	DAKIK	Konecranes	15.08.2024
Veto Review by All Partners			30.08.2024

Editor
Stefan Ellmauthaler (TUD)

Contributors
Alex Ivliev (TUD)
Stefan Ellmauthaler (TUD)

Executive Summary

The design, development, and testing of the Inference Engine is governed in this task. It processes data and knowledge from various sources in the Knowledge Base and from other static and dynamic input sources to infer new knowledge via rules. This becomes fruitful in the product configuration tasks, where consistent and correct answers to customer inquiries are generated.
We have designed a novel inference engine architecture, based on cutting edge technologies. This led to the introduction of the Nemo language, a Datalog dialect with a vast set of features. These range from native understanding of datatypes, mixing of datatypes in predicates, and various mathematical operations to multiple modern input formats, explanation and tracing, and a fast online web-interface, where the whole process runs directly in the browser without any data exchange with other servers.
In the following we will introduce the new language, give a stronger spotlight on the various features, and show how the engine can be used in the LLE use-case.

Table of Content

Figures

1 Introduction

InnoSale aims to optimize the sales process of specialized and therefore highly customizable products. As their complexity does not allow for a complete representation within a catalog, customer requests usually arrive as unstructured, free-form text that often omits crucial details. Such cases require substantial back-office support, relying on the expertise of sales engineers to infer missing information and to spot inconsistencies. Towards automating this task, deductive rule-based systems offer clear advantages. By encoding expert knowledge as simple if-then statements, it is possible to validate and to complete customer requests in an understandable and explainable manner. Systems using this approach are easy to maintain and can scale effectively as rules provide a clear, logical structure that is easy to understand, even for people without programming experience.

Existing solutions in this space are either limited in scope [1], do not scale well in terms of performance [2], or are closed-sourced commercial projects [3], [4]. We therefore develop Nemo, an open-source rule-based reasoning toolkit [5], [6]. Its rule-language is based on the recursive query language Datalog, extended with various features to accommodate the use cases of InnoSale. In addition, Nemo provides a convenient web interface and supports the Language Server Protocol enabling efficient rule editing within compatible editors. Furthermore, Nemo can explain why certain facts were inferred, which improves the trustworthiness of the system. Experiments show that it outperforms existing systems on common benchmarks.

Within the InnoSale architecture, Nemo comprises the Inference Engine and takes as input structured data produced by the Semantic Search (T3.2) and Customer Segmentation (T3.4) components. Inferences are made using expert rules developed as part of the Knowledge Base (T3.3) based on concepts from the Knowledge Model (T3.1). The resulting product configuration serves as input for the Optimal Pricing component (T3.5) and is displayed within the User Dialog Component (T4.1). The web version of the tool and LSP support for the rule language are part of the Knowledge Acquisition Component (T4.2). Note, this task was focused on rule-based reasoning, therefore this deliverable will solely report on Nemo and its integrations to other reasoning systems and components of InnoSale.

This deliverable gives a high-level overview of the Nemo system, including its usage, installation process and features of the Nemo rule language.

2 Installation and Usage

Nemo consists of several applications and program libraries. This section contains installation instructions and describes the usage of the respective component. The source-code for the Nemo project is publicly available on GitHub: https://github.com/knowsys/nemo.

2.1 Command-Line Application

The standard way to use Nemo is as a command-line application. Nemo is written in Rust and can therefore be easily compiled from source. We start by downloading the project:

```
git clone https://github.com/knowsys/nemo
```

The application can then be built using Rust's package manager cargo:

```
cd nemo
cargo build -r
```

Depending on the current version of Nemo, it might be necessary to switch to the nightly toolchain, which can be done with the following commands:

```
rustup toolchain install nightly
rustup default nightly
```

In addition, we prebuild binaries are provided for all common operating systems, including Linux, Windows and MacOS. These can be accessed and downloaded directly from the GitHub release page: https://github.com/knowsys/nemo/releases.

After installation, Nemo can be run via the following command:

```
./nmo <RULES>
```

Executing the above command computes all inferences implied by the provided rule file, which may reference additional input data via import directives, and saves the result according to the export statements (see section **Error! Reference source not found.**).

Further information can be found using the command-line option help:

```
Usage: nmo [OPTIONS] <RULES>...

Arguments:
  <RULES>...
          One or more rule program files

Options:
  -e, --export <EXPORT_SETTING>
          Override export directives in the program

  -D, --export-dir <EXPORT_DIRECTORY>
          Base directory for exporting files

  -o, --overwrite-results
          Replace any existing files during export

  -g, --gzip
          Use gzip to compress exports by default

  -I, --import-dir <IMPORT_DIRECTORY>
          Base directory for importing files (default is working directory)

      --trace <FACTS_TO_BE_TRACED>
          Facts for which a derivation trace should be computed
```

Figure 1: Excerpt of Nemo's command line options

2.2 Web Application

Nemo can be used without any installation as a web application that can be accessed here: https://tools.iccl.inf.tu-dresden.de/nemo.

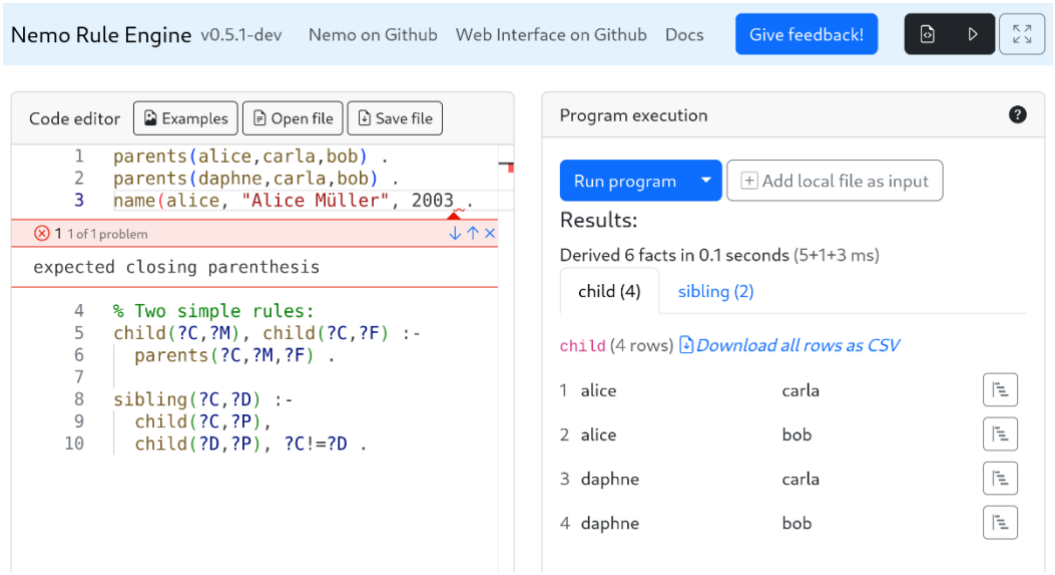

Figure 2: Web Interface of Nemo

Figure 2 shows the web interface of Nemo. The panel on the left contains a text-editor for editing rule files that is capable of basic syntax highlighting as well as displaying error messages. The displayed rules derive the sibling relation by using ancestry information that are given as facts. Clicking on the "Run program" button on the right panel evaluates the rules. Additional data can be included via "Add local file as input". Note that all computation is performed locally on the user's machine, so none of the local input files are uploaded to a server. However, it is also possible to reference remote data, which is downloaded upon execution. After inferencing is finished, the resulting facts are displayed as tables to the user and can be saved in CSV format. By clicking the button next to each fact, the tracing panel is brought up, which displays the origin of the fact as a tree.

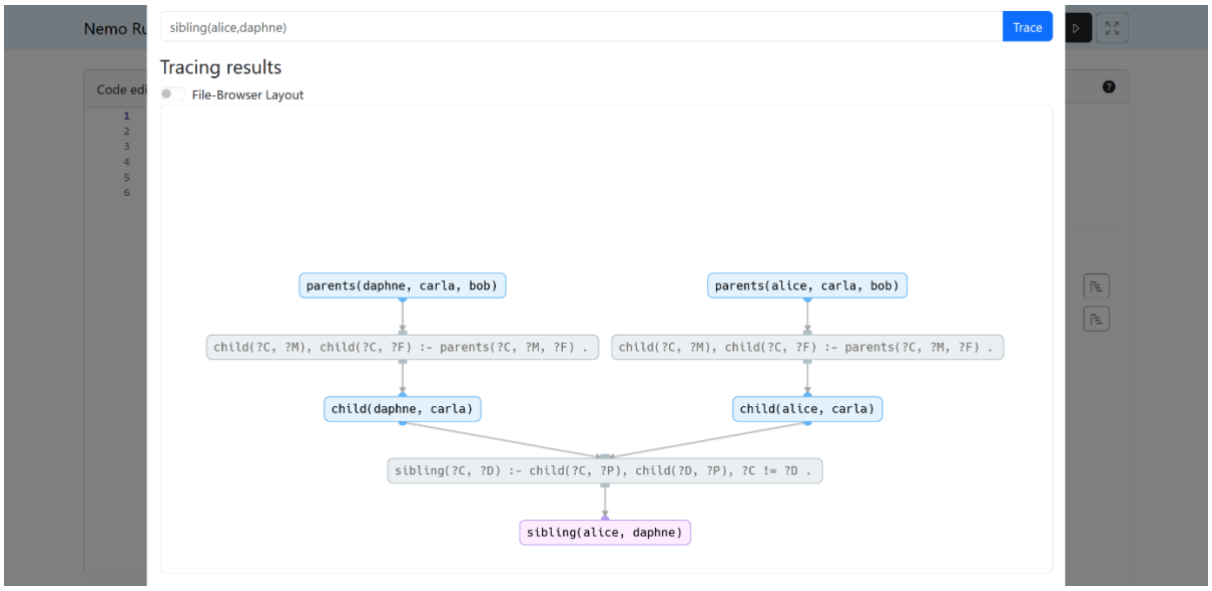

Figure 3: Tracing panel in the Nemo Web App

Figure 3 shows an example output for tracing the fact `sibling(alice, daphne)` given the above rules. This display format allows us to easily understand why the result appears in the output. In this case, we can see that Alice and Daphne are both children of Carla meaning that they are siblings.

2.3 Developer Interfaces

Nemo also provides interfaces aimed at developers who want to include the reasoner as part of another application. Currently, Nemo offers APIs for Rust, Python, and Web Assembly, which is used to build the web application. In general, they allow developers to

- Import and export data in formats supported by Nemo
- Parse and manipulate rules
- Compute the inferences for a given rule set

At the time of writing, these interfaces are not completed and are therefore subject to change. For up-to-date information, refer to the documentation.

3 Nemo Language Features

The rule language of Nemo is based on Datalog and extended with many features of modern query and rule languages. This section gives a brief introduction to all relevant features using rules from the Light-Lifting domain as examples. A more extensive and up-to-date documentation can be found here: http://knowsys.github.io/nemo-doc.

A simple program, i.e. a collection of rules, is shown below:

```
is_a("KBK_light_crane") .
is_a("KBK_light_crane", "crane") .
max_load("KBK_light_crane", 3200) .

max_load(?Crane, ?Weight)
    :- is_a(?Crane, ?Category), max_load(?Category, ?Weight) .
is_a(?A, ?C) :- is_a(?A, ?B), is_a(?B, ?C) .
```

In the program above, we see part of a concept hierarchy describing the “subclass of” and “instance of” relationship between different types of cranes. Such hierarchies form the basis of many ontologies. Furthermore, the program includes the maximum load capacity for a specific type of crane. Rules are read from right to left, where the right side encodes its condition while the left side contains its conclusion. Variables are denoted with a question mark. Intuitively, the first rule states that if a category of cranes has a given maximum load capacity, then every crane belonging to that category must also have the same load capacity. The second rule demonstrates the recursive computation of the `is_a` relationship. The above ruleset therefore derives that KBK Aluline is a crane with a maximum load capacity of 3200 kg.

3.1 Import and Export

Additional input data may be provided through import statements as shown below:

```
@import data :- csv { resource="https://remote.com/data.cvs.gz" } .
```

The currently supported formats are CSV (using any kind of delimiter) and RDF, including the triple formats NTriples, Turtle, RDF/XML and quad formats TRiG and NQuads. Data can be loaded from disk or be downloaded from an online URL. All data formats are also available for export. Additional parameters may be provided and be optionally compressed with GZip.

3.2 Data types and Built-in functions

Nemo supports many different data types, including integer, single and double precision float, string, language-tagged string, and Boolean as well as related functions known from programming and query languages. This latter includes

- Function on numbers, such as arithmetic operations, trigonometric functions, square root, rounding
- Function on strings, such as capitalize, substring, finding, regular expressions
- Function on Booleans: and, or, exclusive or, not
- Functions for checking the type of a value

A complete and up-to-date list can be found in the documentation.

3.3 Aggregates

An important feature of Nemo's rule language are aggregates. We support the most common aggregates: maximum, minimum, and sum, which work on numbers, as well as count. Recursive computation of aggregates is not allowed. However, aggregates may be used in combination with built-in functions. The following example determines the maximum load capacity of a crane by computing the minimum load capacity of each of its parts:

```
max_load(?Crane, #min(?Load))
    :- max_load(?Part, ?Load), part_of(?Crane, ?Part) .
```

3.4 Negation

Nemo allows for negation using the following syntax:

```
ok("no error") :- ~error(?Error) .
```

As with aggregation, negation in Nemo is stratified meaning there cannot be a cyclic dependency of rules using negation.

3.5 Existential Rules

Existential rules are an important formalism in the field of Knowledge Representation and Reasoning used to represent integrity constraints or to capture implicit knowledge in ontologies [7]. Nemo supports existential rules by implementing the restricted chase [8]. Existential variables, which may only occur in the rule's conclusion, are represented with an exclamation mark. Intuitively, the rule in the following example asserts that a cross-travel-limit switch must be included in the final offer given that the user requested a crane with a wireless control pendant and an electric cross travel motor.

```
offer("cross_travel_limit_switch", !parameter) :-
      request("control_pendant", wireless),
      request("cross_travel_motion", electric) .

error("missing value") :- offer(?Parameter, ?Value), isNull(?Value) .
```

The restricted chase derives a special place holder value, called nulls, when there is no entity that satisfies the rule's conclusion. Hence, we can check if a value is missing by using the isNull function.

4 Conclusion

Rule based inference is essential in the automatic refinement of customer request for complex products. This deliverable presented Nemo, a reasoning engine that uses the rule-based language Datalog to complete missing information with additional background knowledge from the various information sources provided by the overarching InnoSale framework. It offers multiple support tools for knowledge engineers and sales experts to model the needed expertise. This is achieved by an easily accessible web-editor and reasoner, as well as LSP integration for currently available editors and IDEs. It has a wide compatibility to other software systems with multiple API access points and is therefore an easy to utilize and integrate piece of artificial intelligence reasoning. Additionally, Nemo offers multiple use-case specific additions to the language to support needed operations when dealing with real world applications. Finally, it proves to be scalable to the needed tasks due to modern implementation and reasoning methods. It has been shown to be competitive to and often outperforming against already existing approaches and tools.

5 Abbreviations

API	Application Programming Interfaces
CSV	Comma Separated Values
IDE	Integrated Development Environment
KBK	“Kranbaukasten” – a modular system to design cranes
kg	Kilogram
LSP	Language Server Protocol
RDF	Resource Description Framework
XML	Extensible Markup Language

6 References

[1] D. Carral, I. Dragoste, L. González, C. Jacobs, M. Krötzsch, and J. Urbani, "VLog: A Rule Engine for Knowledge Graphs," in *The Semantic Web – ISWC 2019*, C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, I. Cruz, A. Hogan, J. Song, M. Lefrançois, and F. Gandon, Eds., Cham: Springer International Publishing, 2019, pp. 19–35. doi: 10.1007/978-3-030-30796-7_2.

[2] K. Angele, J. Angele, U. Şimşek, and D. Fensel, "RUBEN: A Rule Engine Benchmarking Framework".

[3] Y. Nenov, R. Piro, B. Motik, I. Horrocks, Z. Wu, and J. Banerjee, "RDFox: A Highly-Scalable RDF Store," in *The Semantic Web - ISWC 2015*, vol. 9367, M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, Eds., in Lecture Notes in Computer Science, vol. 9367. , Cham: Springer International Publishing, 2015, pp. 3–20. doi: 10.1007/978-3-319-25010-6_1.

[4] L. Bellomarini, E. Sallinger, and G. Gottlob, "The Vadalog system: datalog-based reasoning for knowledge graphs," *Proc. VLDB Endow.*, vol. 11, no. 9, pp. 975–987, May 2018, doi: 10.14778/3213880.3213888.

[5] A. Ivliev *et al.*, "Nemo: First Glimpse of a New Rule Engine," in *Proceedings 39th International Conference on Logic Programming (ICLP 2023)*, E. Pontelli, S. Costantini, C. Dodaro, S. Gaggl, R. Calegari, A. D. Garcez, F. Fabiano, A. Mileo, A. Russo, and F. Toni, Eds., in EPTCS, vol. 385. Sep. 2023, pp. 333–335. doi: 10.4204/EPTCS.385.35.

[6] A. Ivliev, L. Gerlach, S. Meusel, J. Steinberg, and M. Krötzsch, "Nemo: Your Friendly and Versatile Rule Reasoning Toolkit," in *Proc21st*,

[7] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat, "On rules with existential variables: Walking the decidability line," *Artificial Intelligence*, vol. 175, no. 9, pp. 1620–1654, Jun. 2011, doi: 10.1016/j.artint.2011.03.002.

[8] A. Deutsch, A. Nash, and J. Remmel, "The chase revisited," in *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Vancouver Canada: ACM, Jun. 2008, pp. 149–158. doi: 10.1145/1376916.1376938.