Nemo at v0.10: Explainable Web Rule Reasoning for RDF, SPARQL, and More*

Raimund Dachselt[2], Lukas Gerlach[1], Philipp Hanisch[1], Alex Ivliev[1], Markus Krötzsch[1], Maximilian Marx[1], and Julián Méndez[2]

[1] Knowledge-Based Systems Group, TU Dresden, Germany
{lukas.gerlach, philipp.hanisch1, alex.ivliev, markus.kroetzsch, maximilian.marx}@tu-dresden.de
[2] Interactive Media Lab Dresden, TU Dresden, Germany
{raimund.dachselt, julian.mendez2}@tu-dresden.de

Abstract This system demonstration presents the current state of the *Nemo* graph rule engine at version 0.10. The core of Nemo is a robust and versatile inference engine for Datalog rules with datatypes, value invention, negation, and aggregation, based on an RDF-compatible data model. Nemo's new version adds support for reasoning over federated RDF and SPARQL data sources, and for interactive visual explanations of computed results. The system can be accessed as a user-friendly browser application or powerful command-line tool.

1 Introduction

Rule-based computation has many applications related to knowledge graphs and ontologies, including data transformation, (onto)logical reasoning, and query processing. Rule engines with dedicated support for semantic web standards exist, but are either proprietary (RDFox [10], Vadalog [2]) or inactive (VLog [12], Graal [1]). Nemo is an open source alternative [6], whose latest release v0.10 has added optimised SPARQL integration [7] and interactive visual explanations [3].

This system demonstration will showcase Nemo's current capabilities, illustrating various use cases, including *federation* (of distributed RDF sources and SPARQL endpoints), *data integration* (of RDF and other formats), *graph transformation* (e.g., decoding OWL from RDF), and *ontological reasoning* (on large datasets). Our primary demonstration platform will be Nemo Web[3], a public, browser-based rule editor and reasoner. All our tools are free and open source (see https://github.com/knowsys/nemo, https://github.com/knowsys/nemo-web, & https://github.com/imldresden/nev), and come with detailed online documentation (see https://knowsys.github.io/nemo-doc/).

* This system demonstration showcases advancements reported in an ESWC 2026 research track paper [7], at the Transforming Graph Data Workshop 2026 [3], and the Eurographics Conference on Visualization [9].

[3] URL for the latest Nemo Web release: https://tools.iccl.inf.tu-dresden.de/nemo/
URL for submitted version: https://tools.iccl.inf.tu-dresden.de/nemo/eswc-2026/

2 Rules for the Semantic Web

Nemo introduces a variant of Datalog [8] that is widely compatible with RDF, SPARQL, and related standards. This section provides a brief overview of the main language features. Nemo's basic rule syntax adopts familiar conventions from logic programming and RDF, as in the following example program:

```
1 @prefix dblp: <https://dblp.org/rdf/schema#> .
2 @import dblp_spo :- sparql{ endpoint=<https://sparql.dblp.org/sparql> } .
3 %% Find coauthors of Paul Erdős:
4 coAuthor(?author) :-
5    dblp_spo(?pub, dblp:authoredBy, <https://dblp.org/pid/e/PErdos>),
6    dblp_spo(?pub, dblp:authoredBy, ?author) .
```

Line 1 defines a prefix `dblp:` to shorten IRIs, and Line 2 declares the predicate `dblp_spo` to be imported from the DBLP SPARQL service. By default, all RDF triples $\langle s, p, o \rangle$ are virtually imported as facts `dblp_spo(`s`,`p`,`o`)`. The rule in lines 4–6 then finds all authors of publications of Paul Erdős. As in SPARQL, variables are denoted with `?` and constants can be arbitrary RDF terms (IRIs, literals, blank nodes), in all familiar notations. It is also possible to write "abstract" constants, as in `father(alice,bob)`, which Nemo interprets as local IRIs.[4]

The example program does not actually import all DBLP triples but creates optimised SPARQL queries that are evaluated on demand during inferencing, as discussed in the accompanying research paper [7]. The query therefore completes in under one second in a browser (depending on current endpoint latency).

RDF Imports and Exports RDF data in any common syntax can also be imported from other (online or offline) sources, as in the following example:

```
7 @import triples :- rdf{resource=<https://example.org/data.ttl>} .
8 @import quads :- rdf{resource="./mydata.nq"} .
```

SPARQL imports can also fetch results of a specified query, in which case the imported predicate's arity matches the result. Analogous `@export` directives allow exporting ternary predicates to RDF files, where illegal tuples (e.g., with RDF literals as subject) are dropped. Other formats, such as CSV, are also supported.

SPARQL Filter Functions Nemo supports many filter functions known from SPARQL, including functions for strings (`STRLEN`, `CONTAINS`, ...), numbers (`ROUND`, `SQRT`, ...), and RDF term handling (`isIRI`, `DATATYPE`, ...). Common relations and operators, such as `<` or `+`, work as expected.

Aggregates and Negation Nemo supports aggregates, such as `COUNT`, and negation, denoted by `~` and corresponding to SPARQL's `NOT EXISTS`. For example, the following rule counts publications per author, excluding reports:

```
9 pubCount(?author,#count(?pub)) :- authorOf(?author,?pub), ~report(?pub) .
```

[4] Even predicate names are IRIs in this sense, and we could equivalently (but less readably) write the example fact as `<father>(<alice>,<bob>)`.

Both features can only be applied to data that does not recursively depend on the result, a criterion known as *stratification* and checked by Nemo.

Global Parameters Nemo supports program-level parameters that act as placeholders for specific terms. In Line 5 above, e.g., we could use `$authorId` in place of `<https://dblp.org/pid/e/PErdos>`, and add `@parameter $authorId .` Parameter values can be set in the program (using `=`) or be specified at runtime. For the latter, we might call the Nemo command-line client with an additional argument `--param authorId="<https://dblp.org/pid/e/PErdos>"`, where the added quotes protect the value. This can be used for experiments with parametrised Nemo queries or for creating command-line tools that extract web data.

Value Invention (aka Blank Node Creation) Nemo can create fresh blank nodes during inferencing, by using variables with `!` in conclusions. The following rule transforms OWL axioms into an alternative form, using a blank node:

```
10 out(?p,rdfs:subPropertyOf,!invP), out(!invP,owl:inverseOf,?p) :-
11                                  in(?p,rdf:type,owl:SymmetricProperty) .
```

Using multiple conclusions in one rule, like in the example, is convenient for creating highly normalised data structures such as RDF graphs.

3 Nemo Web and the Nemo Explain Visualizer

Nemo Web is a browser-based rule editor, reasoner, and result inspector that includes interactive explanation features. It combines a rich editor (with syntax highlighting, error highlighting, and refactoring capabilities) and a browser-based build of the Nemo reasoner (Web Assembly+TypeScript). Reasoning is performed locally on the browser, without any server, and data is only transferred when remote sources are queried.

Nemo Web is intended as a tool to help experts develop and use Nemo programs. The integrated visual explanation tool *nev* (*Nemo Explain Visualizer*) [9] allows users to inspect the inference chains behind every output as a proof tree (see Fig. 1). Going beyond traditional proof trees, this visualisation also integrates set-based result views (e.g., for aggregates) and support summarised displays of similar proof structures [3].

4 Outlook

Nemo aims to provide a usable platform for research, teaching, and practice, with significant efforts going into usability and robustness in the face of real-world data. While it can already cope with large datasets, even in Nemo Web, further performance improvements and new optimisation methods are already under development, e.g. static filtering [5].

Other future development goals include the integration of more (external) data sources, new expressive language features, incremental reasoning and visualisation, and formal verification [4,11]. Moreover, this demonstration will also be an excellent opportunity to gather wishes from the research community.

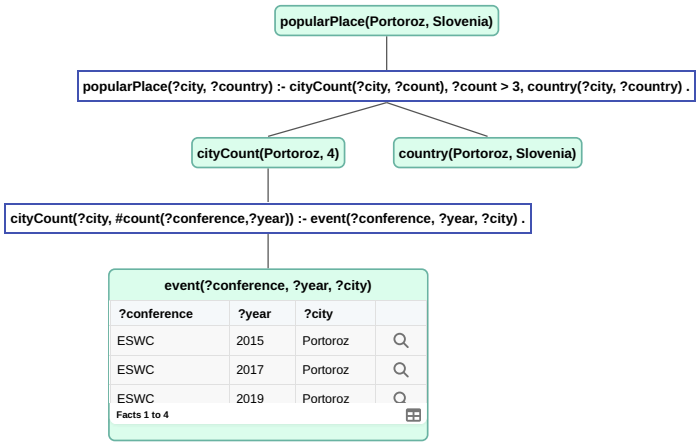

Figure 1. Nev display of a proof of `popularPlace(Portoroz, Slovenia)` via two rule applications that detect frequent conference locations; leaf nodes show input facts

Acknowledgments. This work is funded by Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy: EXC 2050/2, 390696704 – "Centre for Tactile Internet" (CeTI); by DFG grant 389792660 as part of TRR 248 – CPEC; by Bundesministerium für Forschung, Technologie und Raumfahrt (BMFTR, Federal Ministry of Research, Technology and Space) and Saxon State Ministry for Science, Culture and Tourism (SMWK) in Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI, SCADS22B); and by BMFTR and German Academic Exchange Service (DAAD) in project 57616814 (SECAI, School of Embedded and Composite AI).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Baget, J., Leclère, M., Mugnier, M., Rocher, S., Sipieter, C.: Graal: A toolkit for query answering with existential rules. In: Proc. 9th Int. Web Rule Symposium (RuleML'15). LNCS, vol. 9202, pp. 328–344. Springer (2015)
2. Bellomarini, L., Sallinger, E., Gottlob, G.: The Vadalog system: Datalog-based reasoning for knowledge graphs. Proc. VLDB Endowment **11**(9), 975–987 (2018). https://doi.org/10.14778/3213880.3213888
3. Dachselt, R., Gerlach, L., Hanisch, P., Ivliev, A., Krötzsch, M., Marx, M., Méndez, J.: Declarative debugging for Datalog with aggregation. In: Proc. Workshops of the EDBT/ICDT 2026 Joint Conf. CEUR WS Proceedings (2026), https://ceur-ws.org/Vol-4192/TGD-paper4.pdf
4. Gerlach, L.: The chase in Lean - Crafting a formal library for existential rule research. In: Wassermann, R., Mugnier, M.L., Baader, F. (eds.) Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning (2026), to appear, preprint at https://arxiv.org/abs/2604.22531

5. Hanisch, P., Krötzsch, M.: Rule rewriting revisited: A fresh look at static filtering for Datalog and ASP. In: ten Cate, B., Funk, M. (eds.) Proc. 29th Int. Conf. on Database Theory (ICDT'2026). LIPIcs, vol. 365. Dagstuhl Publishing (2026). https://doi.org/10.4230/LIPIcs.ICDT.2026.5
6. Ivliev, A., Gerlach, L., Meusel, S., Steinberg, J., Krötzsch, M.: Nemo: Your friendly and versatile rule reasoning toolkit. In: Proc. 21st Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'24). pp. 743–754. IJCAI Organization (2024). https://doi.org/10.24963/kr.2024/70
7. Ivliev, A., Krötzsch, M., Marx, M.: SPARQLing Datalog for rule-based reasoning over large knowledge graphs. In: Proc. 23rd European Semantic Web Conf. (ESWC'26). LNCS, Springer (2026)
8. Krötzsch, M.: Modern Datalog: Concepts, methods, applications. In: Artale, A., Bienvenu, M., García, Y.I., Murlak, F. (eds.) Joint Proc. of the 20th and 21st Reasoning Web Summer Schools (RW'24 & RW'25). OASIcs, vol. 138. Dagstuhl Publishing (2025). https://doi.org/10.4230/OASIcs.RW.2024/2025.7
9. Méndez, J., Gerlach, L., Wieland, T., Ivliev, A., Krötzsch, M., Dachselt, R.: nev: A visual query tracer and builder for declarative logic programming on Nemo. In: Proceedings EuroVis 2026 - 28th EG Conference on Visualization. EuroVis Posters '26, EG (5 2026)
10. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: RDFox: A highly-scalable RDF store. In: Proc. 14th Int. Semantic Web Conf. (ISWC'15), Part II. LNCS, vol. 9367, pp. 3–20. Springer (2015). https://doi.org/10.1007/978-3-319-25010-6_1
11. Tantow, J., Gerlach, L., Mennicke, S., Krötzsch, M.: Verifying datalog reasoning with Lean. In: Forster, Y., Keller, C. (eds.) Proc. 16th Int. Conf. Interactive Theorem Proving (ITP'25). LIPIcs, vol. 352. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2025)
12. Urbani, J., Jacobs, C., Krötzsch, M.: Column-oriented Datalog materialization for large knowledge graphs. In: Proc. 30th AAAI Conf. on Artificial Intelligence (AAAI'16). pp. 258–264 (2016). https://doi.org/10.1609/aaai.v30i1.9993