Technische Universität Dresden

Alex Ivliev Markus Krötzsch Maximilian Marx
Knowledge-Based Systems Group · TU Dresden

SPARQLing Datalog for Rule-Based Reasoning over Large Knowledge Graphs

ESWC 2026 · Research Track

14 May 2026

Combining SPARQL and Datalog

Datalog Engines such as **Nemo** support SPARQL imports

```
labelProp(rdfs:label) .                                    % initial fact

@import subPropOf :- sparql {
  endpoint = <https://query.wikidata.org/sparql>,
  query = "SELECT ?x ?y WHERE { ?x rdfs:subPropertyOf ?y }"
} .

labelProp(?x) :- subPropOf(?x, ?y), labelProp(?y) .        % recursive rule
```

Combining SPARQL and Datalog

Datalog Engines such as **Nemo** support SPARQL imports

```
labelProp(rdfs:label) .                                    % initial fact

@import subPropOf :- sparql {
  endpoint = <https://query.wikidata.org/sparql>,
  query = "SELECT ?x ?y WHERE { ?x rdfs:subPropertyOf ?y }"
} .

labelProp(?x) :- subPropOf(?x, ?y), labelProp(?y) .        % recursive rule
```

Problems:

- Its expensive to pre-fetch all the data
- Its difficult to handle *recursive dependencies* between imports and rules

Combining SPARQL and Datalog

Datalog Engines such as **Nemo** support SPARQL imports

```
labelProp(rdfs:label) .                                    % initial fact

@import triple :- sparql {
  endpoint = <https://query.wikidata.org/sparql>,
  query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }"
} .

labelProp(?x) :- triple(?x, rdfs:subPropertyOf, ?y), labelProp(?y) .
```

Goals:

- Method to dynamically fetch only the data that is needed
- Optimizations that make that feasable in practice

Incremental SPARQL Imports

```
@import triple :- sparql { query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }" } .

result(?s, ?o) :- triple(?s, ?p, ?o), a(?s, ?z), b(?z, ?o) .
```

import atom

binding atoms

Incremental SPARQL Imports

```
@import triple :- sparql { query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }" } .

result(?s, ?o) :- triple(?s, ?p, ?o) , a(?s, ?z), b(?z, ?o) .
```

import atom

binding atoms

Bindings $\left(\pi_{?s,?o}(a \bowtie b)\right)$:

?s	?o
alice	london
bob	london
bob	berlin

Incremental SPARQL Imports

```
@import triple :- sparql { query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }" } .

result(?s, ?o) :- triple(?s, ?p, ?o) , a(?s, ?z), b(?z, ?o) .
```

import atom

binding atoms

Bindings $\left(\pi_{\texttt{?s},\texttt{?o}}(\texttt{a} \bowtie \texttt{b})\right)$:

?s	?o
alice	london
bob	london
bob	berlin

SPARQL query:

```
SELECT ?s ?p ?o WHERE {
  ?s ?p ?o .
  VALUES (?s ?o) {
    (alice london)
    (bob london)
    (bob berlin)
  }
}
```

Incremental SPARQL Imports

```
@import triple :- sparql { query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }" } .

result(?s, ?o) :- triple(?s, ?p, ?o), a(?s, ?z), b(?z, ?o) .
```

import atom — binding atoms

Bindings $\left(\pi_{?s,?o}(a \bowtie b)\right)$:

?s	?o
alice	london
bob	london
bob	berlin
carol	paris

SPARQL query:

```
SELECT ?s ?p ?o WHERE {
  ?s ?p ?o .
  VALUES (?s ?o) {
    (carol paris)
  }
}
```

Merging Multiple Import Atoms

```
@import ep1 :- sparql { endpoint = <wikidata>, query = "..." } .
@import ep2 :- sparql { endpoint = <dblp>, query = "..." } .
ep1(?a, <p1>, ?b), ep1(?b, <p2>, ?c), ep2(?a, <p3>, ?d), ~ep2(?d, <p4>, ?e), local(?a)
```

endpoint 1 | endpoint 2 | binding

1. Group import atoms by endpoint and compute binding tables
2. For each endpoint, issue a SPARQL query
3. Join query results and evaluate additional functions

Merging Multiple Import Atoms

```
@import ep1 :- sparql { endpoint = <wikidata>, query = "..." } .
@import ep2 :- sparql { endpoint = <dblp>, query = "..." } .
```

$$\underbrace{\texttt{ep1(?a, <p1>, ?b), ep1(?b, <p2>, ?c)}}_{\text{endpoint 1}},\ \underbrace{\texttt{ep2(?a, <p3>, ?d), } \sim\texttt{ep2(?d, <p4>, ?e)}}_{\text{endpoint 2}},\ \underbrace{\texttt{local(?a)}}_{\text{binding}}$$

```
SELECT ?a ?b ?c WHERE {
  ?a <p1> ?b . ?b <p2> ?c .
  VALUES (?a) {
    ...
  }
}
```

```
SELECT ?a ?d WHERE {
  ?a <p3> ?d .
  FILTER NOT EXISTS {
    ?d <p4> ?e
  }
  VALUES (?a) { ... }
}
```

Cartesian Products

```
@import triple :- sparql { query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }" } .

result(?s, ?o) :- triple(?s, ?p, ?o), a(?s, ?x), b(?x, ?y), c(?o, ?z) .
```

import atom | group 1 | group 2

Bindings 1

?s
alice
bob

Bindings 2

?o
london
berlin

SPARQL query:

```
SELECT ?s ?p ?o WHERE {
  ?s ?p ?o .
  VALUES (?s) { (alice) (bob) }
  VALUES (?o) { (london) (berlin) }
}
```

Cartesian Products

```
@import triple :- sparql { query = "SELECT ?s ?p ?o WHERE { ?s ?p ?o }" } .

  result(?s, ?o) :- triple(?s, ?p, ?o), a(?s, ?x), b(?x, ?y), c(?o, ?z) .
```

import atom | group 1 | group 2

Bindings 1

?s
alice
bob
carol

Bindings 2

?o
london
berlin
paris

Variant 1:

```
SELECT ?s ?p ?o WHERE {
  ?s ?p ?o .
  VALUES (?s) { (carol) }
  VALUES (?o) {
    (london)
    (berlin)
    (paris)
}}
```

Variant 2:

```
SELECT ?s ?p ?o WHERE {
  ?s ?p ?o .
  VALUES (?s) {
    (alice) (bob)
  } VALUES (?o) {
    (paris)
  }
}
```

Example Applications

Handcrafted examples combining Datalog and SPARQL imports

Common Ancestors

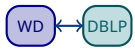
Find the most recent common ancestors of two people on Wikidata

Winning Streaks

Compute the longest winning streaks in a series or league of sports events

Constraint Violations

Detect disjoint-class, inverse-property, and none-of violations in Wikidata



Open Access

Find Oxford employees who published in open-access journals

Evaluation

System Comparison

Runtime on all benchmarks

Nemo v0.10.0: new implementation

Nemo v0.8.0: no optimizations

VLog: incremental imports

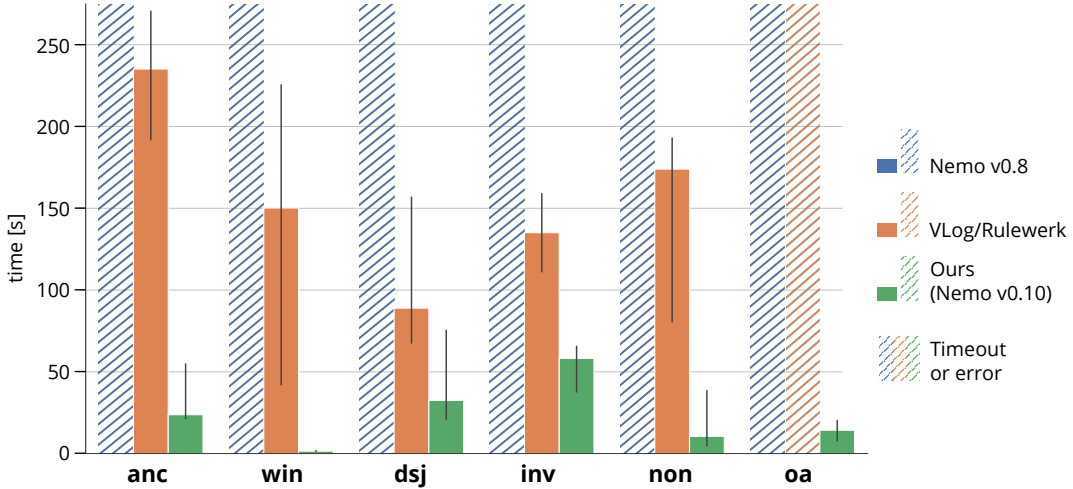
Ablation Study

Disable individual optimizations

- Merging import atoms
- Avoiding Cartesian products

Timeout: 10 min • 16 GiB memory limit • Median of 10 runs

Evaluation: System Comparison



Evaluation: Ablation Study

Test impact of different optimizations

Benchmark	Full	No Merging	No Cart. Prod.
Common Ancestors	24 s	29 s	32 s
Winning Streaks	1 s	42 s	1 s
Disjoint Classes	32 s	timeout	25 s
Inverse Constraint	48 s	timeout	44 s
Non-of Constraint	10 s	timeout	9 s
Open Access	14 s	67 s	timeout

Every optimization contributes to performance gains

Summary

- New method for on-demand SPARQL imports in rule languages
- Optimizations that improve performance significantly in practice
- Try Nemo at: `tools.iccl.inf.tu-dresden.de/nemo`