

# SiN3: Scalable Inferencing with SPARQL CONSTRUCT Queries

Dörthe Arndt<sup>1,\*</sup>, William Van Woensel<sup>2,†</sup> and Dominik Tomaszuk<sup>3,†</sup>

<sup>1</sup>Computational Logic group, Technische Universität Dresden, Dresden, Germany

<sup>2</sup>Telfer School of Management, University of Ottawa, Ottawa, ON, Canada

<sup>3</sup>University of Białystok, Białystok, Poland

## Abstract

SPARQL is a widely used and well-established standard in the Semantic Web to efficiently query explicitly stated RDF data. However, the authors (and others before us) suggest that this ignores the full potential of the language: when used recursively, CONSTRUCT queries can serve as fully-fledged rules allowing for reasoning over the data. For this purpose, we present SiN3, a tool that translates SPARQL queries to Notation3 (N3) rules; an N3-compliant reasoner (such as *eye*) can then recursively generate inferences from RDF data. We provide a demo that uses CONSTRUCT queries to infer the need for Zika screening, based on the Turtle HL7 FHIR notation. We show that a reasoning-based approach, compared to standard SPARQL, makes such use cases more efficient, as rule-based reasoning is optimized for recursively generating inferences; and easier to implement, as only one execution run is necessary to apply interdependent queries, and supporting queries can be used to modularize the code. We compare the performance of SiN3 with SpinRDF, which supports reasoning over the SPIN serialization of SPARQL but is an order of magnitude slower on our test data. SiN3 thus enables efficient reasoning using SPARQL CONSTRUCT queries, opening the door for scalable Semantic Web reasoning using a well-known semantic formalism.

## Keywords

SPARQL, Semantic reasoning, SiN3, SPIN, Notation3,

## 1. Introduction

The query language SPARQL [1] is a well-established standard in the Semantic Web: it provides an easy-to-learn syntax—especially for people used to databases—together with a lot of useful features like filter functions, aggregation, and property paths to extract information from RDF graphs. This information is then typically provided to the user in table form (SELECT) or in an RDF graph (CONSTRUCT). However, even in the latter case, CONSTRUCT queries are not applied recursively, that is, queries do not act on each others' results. We aim to change this situation by providing SiN3 (**Sparql in N3**), a tool that translates SPARQL queries to Notation3 (N3) [2], a rule language for the Semantic Web. Subsequently, reasoners like *eye* [3] or *jen3*<sup>1</sup>, which are

---


ISWC'23: International Semantic Web Conference, November 06–10, 2023, Athens, Greece


\*Corresponding author.

†These authors contributed equally.

✉ doerthe.arndt@tu-dresden.de (D. Arndt); wvanwoensel@uottawa.ca (W. V. Woensel); d.tomaszuk@uwb.edu.pl (D. Tomaszuk)

ORCID 0000-0002-7401-8487 (D. Arndt); 0000-0002-7049-8735 (W. V. Woensel); 0000-0003-1806-067X (D. Tomaszuk)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

<sup>1</sup><https://github.com/william-vw/jen3>

designed for scalable reasoning, can infer new data based on CONSTRUCT queries. Using SiN3, complex use cases—such as our Zika use case, which involves many logical steps—become easier and more efficient to implement: (i) *modularity*: each step can be encapsulated in an individual query, and its results can be re-used by other queries. This is in contrast to a single monolithic query with likely code duplication, or ad-hoc application code executing multiple queries and manually tying the results together (e.g., with a query being executed for all results of another query in a for-loop); (ii) *expressivity*: instead of using “vanilla” N3, features provided by SPARQL (e.g., property paths, aggregation), which have proven to be very useful, are directly available in SiN3; (iii) *performance*: rule-based reasoning is optimized for recursively deriving new information, with the *eye* N3 reasoner [3] being state-of-the-art and having been used in many projects since 2006; (iv) *Web-compatibility*: SiN3 acts on and produces RDF graphs.

The idea of using SPARQL recursively is not new and has been investigated before. Polleres [4] translated the—back then—upcoming standard SPARQL 1.0 into Datalog to better understand the complexity of the language; a (blank node-free) recursive extension of SPARQL is discussed. Similarly, Kostylev et al. [5] investigated how different restrictions (e.g., lack of blank nodes) of recursive CONSTRUCT queries relate to existing logical frameworks; they proposed a recursive extension of SPARQL resembling that of SQL. Atzori et al. [6] and Reutter et al. [7] take a more practical approach by adding to SPARQL (respectively) a recursion function, which can be called within a SPARQL query to execute a given query string; and a recursion operator, which calculates the fixed point-based recursion results of given queries. The aim of our work differs from the approaches above in the sense that we want to offer a practical tool, enabling the user who is more familiar with SPARQL to make use of N3 logic, its features (e.g., recursive reasoning, range of built-ins) and N3 rules written by third parties. As a secondary goal, we wanted to show that the built-in functions of the logic [8] are strong enough to cover SPARQL.

Section 2 provides a use case that illustrates the usefulness of our work. The example is available in our demonstrator<sup>2</sup> and git repo [9]. Section 3 provides an in-depth discussion of the implementation. Finally, in Section 4, we present the conclusion, summarize key findings, and discuss future research directions.

## 2. Use Case: Zika Screening using HL7 FHIR

As an example, we consider a case in which we query patient data in an electronic health record (EHR). The data is represented in HL7 FHIR<sup>3</sup>, an interoperability standard for EHR that offers an RDF representation (Turtle syntax) and allows covering detailed information about—among other things—health conditions, test results, and personal data. In Listing 1 we display an extract of such data<sup>4</sup>: patient :p0 has condition :po\_pregn which is a pregnancy (terminology code 77386006). We observe that the information about the patient and condition is structured in a rather complex way, as it relies on many nested blank nodes, and the relation between concepts is expressed by the fact that the id value of the patient (line 1) is the same as the condition’s subject’s reference value (line 3). This pattern of expressing relationships is

---

<sup>2</sup><https://n3-editor.herokuapp.com/n3/spin3/s/CT3InugS>

<sup>3</sup><http://build.fhir.org/rdf>

<sup>4</sup>Namespaces are omitted for brevity.

---

```
1 :p0 a fh:Patient ; fh:id [ fh:v "Patient/p0" ] .
2 :p0_pregn a fh:Condition ;
3   fh:subject [ fh:reference [ fh:v "Patient/p0" ] ] ;
4   fh:code [
5     fh:coding ( [
6       fh:system [ fh:v 'http://snomed.org/sct' ] ;
7       fh:code [ fh:v 77386006 ] # (SNOMED) pregnancy
8     ] ) ] .
```

---

Listing 1: Extract of an EHR in HL7 FHIR RDF. Patient0 (:p0) is pregnant (condition :p0\_pregn). Patient and condition are related through the id ("Patient/p0").

---

```
1 CONSTRUCT {
2   ?p ut:has ?r
3 } WHERE {
4   ?p fh:id/fh:v ?id .
5   ?r fh:subject/fh:reference/fh:v ?id .}
```

---

Listing 2: CONSTRUCT query for conditions or observations of patients.

typical for HL7 FHIR, but similar constructs can also be found in many other RDF vocabularies, such as Bio2RDF [10].

We implemented the CDC testing guidance for Zika<sup>5</sup> using SPARQL CONSTRUCT queries. Our queries, based on the EHR, determine whether a patient should be tested for Zika. The recommendation relies on different factors such as pregnancy, relevant symptoms, recent travel to Zika areas, or sexual contacts with residents or travelers to Zika areas. To illustrate the *modularity* offered by SiN3, we encapsulate the aforementioned relationship pattern (i.e., shared id and reference values) in an individual CONSTRUCT query, so other queries can more easily refer to a patient’s observations and conditions (Listing 2). As a result, other queries can simply refer to the triple pattern `?p ut:has ?r` instead of having to repeat the query’s WHERE clause, as would be the case in standard SPARQL. Secondly, being a resident of, or having traveled to, a Zika area (a list of 92 countries) should be checked multiple times, i.e., for the patient as well as their recent sexual partners. We similarly encapsulate the checks for residence and travel status of any individual within two separate CONSTRUCT queries, so they do not have to be repeated each time. We refer to our git repository for the full code for the Zika use case [9].

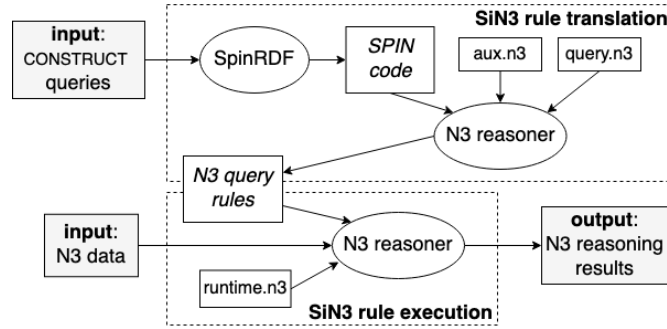
### 3. SPARQL-in-N3 Reasoning Implementation

To implement a recursive version of SPARQL CONSTRUCT queries, we translate SPARQL queries to N3 [2], due to (a) the variety of built-ins that N3 provides [8], e.g., supporting scoped negation-as-failure to implement aggregation; (b) support for the production of new blank nodes, which is crucial for many cases (such as our Zika use case), and (c) tooling support for the language.

Figure 1 illustrates the overall workflow. In **SiN3 rule translation**, the input *CONSTRUCT queries* are first translated into *SPIN code* [11], an RDF representation of SPARQL, using *Spin-RDF* [12]. Next, we translate this *SPIN code* into *N3 query rules*, i.e., N3 rules that implement

---

<sup>5</sup><https://www.cdc.gov/zika/hc-providers/testing-guidance.html>



**Figure 1:** Overall workflow: input/outputs are solid rectangles, tools are ellipses; italics indicate intermediate results. Shaded rectangles indicate external inputs/outputs into or out of SiN3.

the CONSTRUCT queries; this is done by executing the *aux.n3* and *query.n3* code on the SPIN code using an *N3 reasoner*. We point out that this step is only needed when first creating or modifying the CONSTRUCT queries, or when using SiN3 interactively (such as our demonstrator). In **SiN3 rule execution**, the *N3 query rules* are executed on the input *N3 data* using an *N3 reasoner*, together with *runtime.n3* (needed for property paths), which yields *N3 reasoning results*.

Note that, for the *input CONSTRUCT queries* file, queries currently should be separated by a dot and newline character. Further, we allow marking queries as filters: by adding the keyword `# @filter` right before a query, only the results of the particular query will appear in the output; all other construct queries are still recursively applied.

In order to get a first indication of the performance of SiN3, we compared our reasoning times with those of SpinRDF [12], which also offers a fixpoint-style algorithm to generate inferences from SPIN code. We ran SiN3 and SpinRDF on the Zika use case presented in the previous section. We modified one CONSTRUCT query introducing new blank nodes as this is not supported by SpinRDF. On a Macbook Pro with Apple M1 Pro, 32Gb RAM and 1Tb SSD (OS Ventura 13.4.1), averaging over 10 runs, generating *SPIN code* took avg. 420ms; for SiN3, generating *N3 query rules* took avg. 93ms, and executing *N3 query rules* took avg. 81ms; for SpinRDF, reasoning over the *SPIN code* took avg. 645ms. We thus found that SiN3, leveraging the state-of-the-art *eye* reasoner [3], was an order of magnitude faster than SpinRDF, while providing the same results. This result has to be interpreted with care: SPIN is a complex language – it does not only cover SPARQL but also a modeling vocabulary [13] – and the SpinRDF reasoner was likely not optimized for use cases as presented here. Thus, a preliminary conclusion is that, for the recursive application of CONSTRUCT queries as presented here, SiN3 is a better choice.

## 4. Conclusion

In this demo paper, we have presented SiN3, a tool that unlocks the full potential of SPARQL by enabling recursive application of CONSTRUCT queries using Notation3 rules. While SPARQL is widely used for querying RDF data on the Semantic Web, its potential as a language for complex reasoning has often been overlooked. By translating SPARQL queries into Notation3 rules, SiN3 allows scalable N3-compliant reasoners, such as EYE, to generate inferences recursively from RDF data. We demonstrated the effectiveness of SiN3 through a demo that utilized CONSTRUCT queries to infer the need for Zika screening based on the HL7 FHIR notation.

Future work includes extending the feature support of our tool (currently, solution modifiers and federated querying are not covered), more extensive testing, and an extension towards supporting other languages like for example, SWRL (e.g., via SWRL2SPIN [14]). We also plan to investigate whether SiN3 could be used to embed SPARQL expressions in N3 [15]. We see this as a first step to finally establish rule-based inferencing as a first-class citizen in the Semantic Web.

## Acknowledgments

Dörthe Arndt was supported by funding from BMBF within projects KIMEDS (grant no. GW0552B) and M/EDGE (grant no. 16ME0529).

## References

- [1] S. Harris, A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013. <https://www.w3.org/TR/2013/REC-sparql11-query-20130321/>.
- [2] W. Van Woensel, D. Arndt, P.-A. Champin, D. Tomaszuk, G. Kellogg, Notation3 Language, W3C Community Group Report, W3C, 2023. <https://w3c.github.io/N3/reports/20230703/>.
- [3] R. Verborgh, J. De Roo, Drawing conclusions from linked data on the web: The eye reasoner, *IEEE Software* 32 (2015) 23–27.
- [4] A. Polleres, From SPARQL to rules (and back), in: Proceedings of the 16th international conference on World Wide Web, 2007, pp. 787–796.
- [5] E. Kostylev, J. Reutter, M. Ugarte, Construct queries in SPARQL, in: 18th International Conference on Database Theory (ICDT 2015), 2015.
- [6] M. Atzori, Computing recursive SPARQL queries, in: 2014 IEEE International Conference on Semantic Computing, IEEE, 2014, pp. 258–259.
- [7] J. Reutter, A. Soto, D. Vrgoč, Recursion in SPARQL, *Semantic Web* 12 (2021) 711–740.
- [8] W. Van Woensel, P. Hochstenbach, Notation3 Builtin Functions, W3C Community Group Report, W3C, 2023. <https://w3c.github.io/N3/reports/20230703/builtins.html>.
- [9] D. Arndt, SiN3: Git repository, 2023. <https://github.com/doerthe/SPARQL-to-N3>.
- [10] F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, J. Morissette, Bio2rdf: towards a mashup to build bioinformatics knowledge systems, *Journal of biomedical informatics* 41 (2008) 706–716.
- [11] H. Knublauch, SPIN - SPARQL Syntax, W3C Member Submission, W3C, 2011. <http://www.w3.org/Submission/2011/SUBM-spin-sparql-20110222/>.
- [12] M. J. Andy Seaborne, Spinrdf, 2019. <https://github.com/spinrdf/spinrdf>.
- [13] H. Knublauch, SPIN - Modeling Vocabulary, W3C Member Submission, W3C, 2011. <http://www.w3.org/Submission/2011/SUBM-spin-modeling-20110222/>.
- [14] N. Bassiliades, SWRL2SPIN: Converting swrl to spin, in: RuleML+ RR (Supplement), 2018.
- [15] M. Farnbauer-Schmidt, V. Charpenay, A. Harth, N3x: Notation3 with SPARQL expressions, in: European Semantic Web Conference, Springer, 2020, pp. 79–83.