N3X: Notation3 with SPARQL Expressions

Matthias Farnbauer-Schmidt[1][0000−0001−6858−0257], Victor Charpenay[1][0000−0002−9210−1583], and Andreas Harth[1][0000−0002−0702−510X]

Friedrich-Alexander Universität Erlangen-Nürnberg,
Chair of Technical Information Systems
matthias.farnbauer-schmidt@fau.de
victor.charpenay@fau.de
andreas.harth@fau.de

Abstract. Writing calculations with many intermediate steps in Notation3 (N3) rules is complex and verbose. This issue is addressed by extending N3 with SPARQL expressions. In this paper, we introduce and evaluate the syntax of this approach, called N3X. In our examples N3X reduces the number of triples in calculation heavy N3 rules by 30 %, the number of triples with a *math:**-predicate by 65 % and the number of non-blank characters by 24.4 % on average.

Keywords: Notation3 · N3X · SPARQL expressions.

1 Introduction

Notation3 (N3) is a logical framework for the Semantic Web [2]. Originally designed as a more readable syntax for humans (compared to RDF/XML). It also includes N3 Logic, an extension to RDF by universally quantified variables and quoted graphs. The latter, called formulae, allow to express statements about graphs. Existentially quantified variables are already included in RDF as blank nodes. The combination of quoted graphs, universally quantified variables and the predicate *log:implies* enables users to express first-order logic in N3. In addition, predicates for logical relationships and for retrieving information from the Web are given.

The serialization format Turtle, derived from N3, is nowadays widely used to present RDF in an human-readable way. However, the adoption of N3 Logic lags behind [1].

The adoption of the Semantic Web in the Internet of Things causes an increase of numeric data in RDF graphs [5]. However, writing arithmetic calculations in N3 is cumbersome. Besides *log:implies*, N3 provides further built-in predicates to perform calculations during rule evaluation. The use of predicates dictates that for each calculation step one statement must be provided. For complex expressions, this becomes increasingly error-prone and verbose.

In this paper we tackle this issue by extending the definition of N3 terms to SPARQL expressions [4]. This extension, called Notation3 Expressions (N3X), allows one to nest expressions and write less triples overall. Furthermore, this supports N3's objective of human-readability.

Listing 1. Distance between two 2D points as N3 rule.

```
1 { ?p1 :x ?x1; :y ?y1. ?p2 :x ?x2; :y ?y2.
2   ?p1 log:notEqualTo ?p2.
3   (?x1 ?x2) math:difference ?dx.
4   (?y1 ?y2) math:difference ?dy.
5   (?dx 2) math:exponentiation ?dx2.
6   (?dy 2) math:exponentiation ?dy2.
7   (?dx2 ?dy2) math:sum ?sum.
8   ?sum math:sqrt ?sqrt. }
9 => { :result :value ?sqrt. }.
```

To illustrate the syntactic discrepancies between N3 and N3X we use the example presented in Listing 1. It is a rule to calculate the distance between two points in a 2D Cartesian coordinate system. The prefixes of *cwm*, a N3-engine, built-in predicates[1] *math:* and *log:* are used.

The semantics of of N3X requires little change compared to N3. In fact, every N3X document can be translated back into N3. Formal semantics are out of the scope of this paper.

In the next section we present alternative approaches to simplify expressions in N3. Then, in section 3 we present the syntax of N3X. An evaluation of how N3X affects the length of rules is given in section 4. Finally, we give a conclusion and an outlook for N3X's future in section 5.

2 Comparable Approaches

In fact, N3's path syntax `!` can be used to nest expressions in a similar fashion to postfix notation. For example, line 5 of Listing 1 could be written as:

```
((?x1 ?x2)!math:difference 2) math:exponentiation ?dx2.
```

However, this contradicts the objective of human-readability, especially for deeper nested expressions.

Another approach is implemented in the N3-engine EYE which is based on Prolog [6]. It provides a built-in predicate *e:calculate* that takes an arithmetic expression provided as string, substitutes given variables and passes this to the underlying Prolog instance for evaluation. EYE's test suite makes extensive use of *e:calculate* in computation heavy tests. In fact, this shows that there is a need for a more simple way to write complex expressions in N3.

3 Syntax of N3X

The syntax of SPARQL expressions is taken almost as-is in N3X, up to two exceptions: SPARQL's comparators `=` and `<=` are also defined in N3 as predicate

[1] https://www.w3.org/2000/10/swap/doc/CwmBuiltins

Listing 2. Distance between two 2D points as N3X rule.

```
{ ?p1 :x ?x1; :y ?y1. ?p2 :x ?x2; :y ?y2.
  ?p1 log:notEqualTo ?p2.
  ?x1 - ?x2 = ?dx. ?y1 - ?y2 = ?dy. }
=> { :result :value math:sqrt(?dx*?dx + ?dy*?dy). }.
```

shorthands, the comparators are replaced by `==` and `=<` respectively to avoid ambiguity between expressions and other terms.

In fact, SPARQL's comparison and N3's built-in comparators overlap in their semantics but are not the same. In SPARQL the meaning of a comparator is defined by the values compared [4], whereas the function of N3's comparators only consider the lexical values of literals regardless of datatypes[1]. The alignment of both is a matter for future improvements.

The translation of the example in Listing 1 to N3X is shown in Listing 2. The number of triples is reduced from 12 to 8 (-33 %) and the number of non-blank characters is reduced from 242 to 126 (-48 %).

In Listing 2 the triples using predicates from the *math:* namespace are rewritten to N3X expressions. Instead, of using a *math:**-predicate to create a new variable binding N3's shorthand `=` for *owl:sameAs* is used. In this context, N3 creates a binding to `?dx` (or `?dy`) with the evaluation's result of the left-hand side expression.

4 Evaluation

We used EYE's test suite[2] to evaluate N3X's gain in conciseness compared to N3. There are 13 tests in the suite that include nested N3 calculations. To name a few, these range from calculating Pi over calculating the date of Easter to calculating the distance between GPS coordinates up to accounting. In addition, there are 6 tests making extensive use of *e:calculate* that we could not translate due to use of built-in Prolog predicates. Furthermore, we added the example of Listing 1 and 2, a rule to iteratively calculate square root and one for Fibonacci numbers. The results of rewriting those 16 examples are shown in Figure 1. On average we reduced the number of triples by 30 %, the number of triples with *math:**-predicates by 65.2 % and the number of non-blank characters by 24.4 %. In general, *math*-comparators can not be removed by N3X as they are used to filter solutions rather than calculating new values.

N3X can only remove those triples with a functional predicate but never adds one. Accordingly, N3X is never longer than N3 and the more functional predicates are included, the more can be reduced. In some cases it was even possible to remove all *math:**-predicates (see Figure 1 cases 6, 8 and 11).

The full grammar of N3X and the evaluation's documentation can be found at http://github.com/MattesWhite/n3x.

[2] https://github.com/josd/eye/tree/master/reasoning/

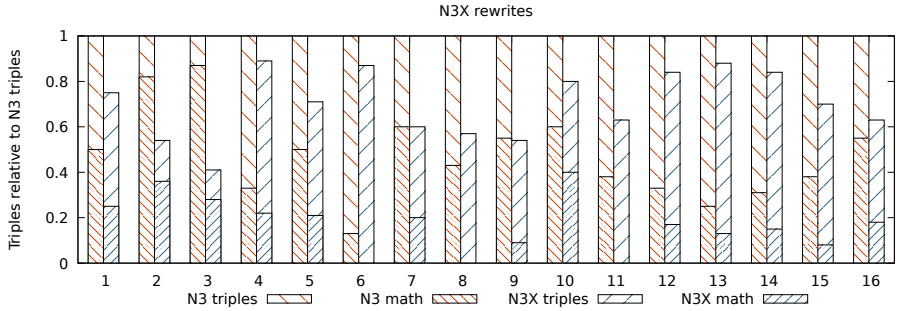

Fig. 1. Results of evaluation. Comparison of triples in rules. Included are the number of triples with a *math:**-predicate.

5 Conclusion and Future Work

N3X introduces SPARQL expressions to N3. Compared to N3, rules become shorter, depending on the number of functional statements included. The basics of the syntax for this extension was presented.

In the future, we will provide formal semantics for N3X based on the *core logic* presented in [1]. With syntax and semantics defined, we will implement a prototype N3X-engine to compare it with existing N3-engines and Prolog implementations.

N3 includes the built-in predicate *log:semantics* which allows engines to fetch and parse documents from the Web to extend their knowledge base. N3X introduces explicit function calls. We plan to leverage this as a hook to retrieve functions from the Web, e.g. in the form of Web Assembly (WASM) modules [3].

References

1. Arndt, D., Schrijvers, T., De Roo, J., Verborgh, R.: Implicit quantification made explicit: How to interpret blank nodes and universal variables in Notation3 Logic. Journal of Web Semantics **58**, 100501 (Oct 2019). https://doi.org/10.1016/j.websem.2019.04.001
2. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C team submission, W3C (Mar 2011), https://www.w3.org/TeamSubmission/2011/SUBM-n3-20110328/
3. Rossberg, A.: Webassembly core specification. W3C recommendation, W3C (Dec 2019), https://www.w3.org/TR/2019/REC-wasm-core-1-20191205/
4. Seaborne, A., Harris, S.: SPARQL 1.1 query language. W3C recommendation, W3C (Mar 2013), http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
5. Szilagyi, I., Wira, P.: Ontologies and Semantic Web for the Internet of Things - a survey. In: IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society. pp. 6949–6954 (Oct 2016). https://doi.org/10.1109/IECON.2016.7793744

6. Verborgh, R., De Roo, J.: Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. IEEE Software **32**(3), 23–27 (May 2015). https://doi.org/10.1109/MS.2015.63, conference Name: IEEE Software