

Formatting SPARQL 1.1 via Parsing Expression Grammar

Hirokazu Chiba

Database Center for Life Science, Joint Support-Center for Data Science Research, Research Organization of Information and Systems, 178-4-4 Wakashiba, Kashiwa, Chiba, 277-0871, Japan

Abstract

The core programming language of the Semantic Web is SPARQL. To increase the productivity of Semantic Web programming, reusability of the SPARQL queries is key. Here, we aim to implement a formatter that fully supports SPARQL 1.1 to enhance the reusability of SPARQL. First, SPARQL 1.1 grammar defined in EBNF was transformed into Parsing Expression Grammar (PEG) to generate a SPARQL 1.1 parser. The parser accepts a valid SPARQL query and constructs the abstract syntax tree (AST) of the input SPARQL query. Next, a formatter was implemented to output a SPARQL query from this AST. The SPARQL formatter is available on the command line and also available as a JavaScript library for developing websites. SPARQL ASTs are represented internally in JSON, and can be reused for purposes other than formatting. Furthermore, the PEG expression developed here can be readily modified for similar subgraph matching problems on knowledge graphs. The source code and a website of the SPARQL formatter are available at <https://github.com/sparqling/sparql-formatter>.

Keywords

SPARQL, parser, formatter, PEG, grammar

1. Introduction

The core of Semantic Web programming is SPARQL [1]. To increase the productivity of Semantic Web programming, enhancing the reusability of SPARQL queries is key. Especially, the readability of code is essential for reusability. However, reformatting the SPARQL query is not a trivial issue. We need to parse the input SPARQL and output it appropriately to reformat it. Parsing a SPARQL query is a complex task because the SPARQL specification has a complex grammar, which is comprised of 173 rules in the format of EBNF. Thus, it is difficult to implement a parser in a conventional sequential programming language.

Several programming languages including JavaScript support Parsing Expression Grammar (PEG) [2] for describing definitions of a grammar in a declarative manner to generate a parser. Here, we aim to implement a formatter that fully supports SPARQL 1.1 by using PEG for command-line use and website development.


IJCKG 2023 Poster and Demo track, December 8–9, 2023, Tokyo, Japan

✉ chiba@dbcls.rois.ac.jp (H. Chiba)

🆔 0000-0003-4062-8903 (H. Chiba)



© 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)

```

// [2] Query ::= Prologue ( SelectQuery | ConstructQuery | DescribeQuery | AskQuery ) ValuesClause
Query = p:Prologue WS* q:( SelectQuery / ConstructQuery / DescribeQuery / AskQuery ) v:ValuesClause
{
  let ast = { type: 'Query' };
  if (p.length) {
    ast.prologue = p;
  }
  ast.queryBody = q;
  if (v) {
    ast.values = v;
  }

  return ast;
}

```

Figure 1: An excerpt from the EBNF of SPARQL and the corresponding PEG rules

2. Implementation

We used PEG.js (<https://pegjs.org/>) to generate a SPARQL 1.1 parser. 173 rules expressed in EBNF were extracted from SPARQL 1.1 Query Language specification [1] and translated into PEG rules for constructing abstract syntax trees (ASTs). An excerpt from the EBNF rules of SPARQL and the corresponding PEG rules for constructing the AST is shown in Figure 1. Notably, white spaces and comments in SPARQL are not included in EBNF rules. As comments are often inserted in SPARQL queries in practice, comments besides white spaces are also expressed in our PEG expressions. Comments are kept separate from the AST. The position from the beginning of the SPARQL input was obtained and included in the returned values.

The formatter was implemented in JavaScript, which accepts an AST of SPARQL and outputs each element of the SPARQL uniformly. The position of each element is compared with the positions of comments, so that the comments are added at the appropriate positions.

91 test queries were extracted from the SPARQL 1.1 Query Language [1] specifications, and 16 from SPARQL 1.1 Update [3]. All those queries were formatted and then manually curated for use as test cases.

3. Use Cases

The SPARQL formatter is published as an npm library, so that it is easy to install and use in a Node.js environment as the `sparql-formatter` command. The Docker version is also available and published on Docker Hub.

The JavaScript code of the parser and formatter has been bundled and made available on Content Delivery Network (CDN), so that users can incorporate the SPARQL formatter into their websites. Figure 2 shows a website using the SPARQL formatter. All the queries extracted from the SPARQL 1.1 specifications are available as example queries on the website.

The AST of a SPARQL query is represented internally in JSON and can be reused for purposes

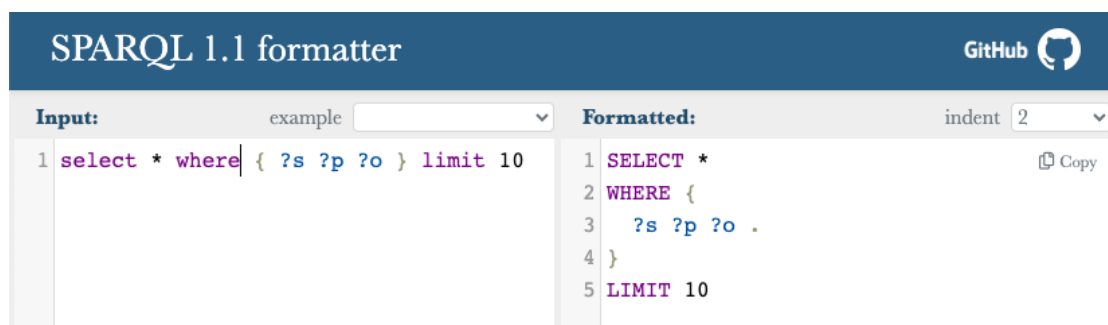


Figure 2: A website of the SPARQL 1.1 formatter

other than formatting. Figure 3 shows the AST obtained for an example SPARQL query, “**SELECT** * **WHERE** { ?s ?p ?o } **LIMIT** 10”.

4. Discussion

By generating a SPARQL parser via PEG.js, the SPARQL formatter was implemented. This work contributes to the increased reusability of SPARQL queries. The queries can be formatted in a uniform way, thus the readability will be increased. While other efforts to implement SPARQL parsers have been made in projects such as RSFStore-js [4] and SPARQL.js (<https://github.com/RubenVerborgh/SPARQL.js>), they could not format queries as intended. Notably, by adding a JSON-LD context to the obtained AST for a SPARQL, the query can be viewed as RDF, thus the SPARQL queries themselves can be treated as RDF resources. It may contribute to the FAIRification [5] of the Semantic Web queries. The PEG.js code developed for SPARQL 1.1 can be modified for extended specifications of SPARQL [6] or other subgraph matching problems on knowledge graphs [7].

References

- [1] SPARQL 1.1 Query Language, 2013. <https://www.w3.org/TR/sparql11-query/>.
- [2] B. Ford, Parsing Expression Grammars: a recognition-based syntactic foundation, in: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, 2004, pp. 111–122.
- [3] SPARQL 1.1 Update, 2013. <https://www.w3.org/TR/sparql11-update/>.
- [4] A. G. Hernández, M. García, A JavaScript RDF store and application library for linked data client applications, in: Devtracks of the WWW2012 conference. Lyon, France, 2012.
- [5] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., The FAIR Guiding Principles for scientific data management and stewardship, *Scientific Data* 3 (2016) 1–9.

```

{
  "type": "Query",
  "queryBody": {
    "select": [
      "*"
    ],
    "where": [
      {
        "type": "TriplesBlock",
        "triplePattern": [
          {
            "subject": {
              "variable": "s"
            },
            "properties": [
              {
                "predicate": {
                  "variable": "p"
                },
                "objects": [
                  {
                    "variable": "o"
                  }
                ]
              }
            ]
          }
        ]
      }
    ],
    "limitOffset": [
      {
        "limit": 10
      }
    ]
  }
}

```

Figure 3: AST obtained for an example SPARQL query

- [6] H. Chiba, I. Uchiyama, SPANG: a SPARQL client supporting generation and reuse of queries for distributed RDF databases, *BMC Bioinformatics* 18 (2017) 1–6.
- [7] H. Chiba, R. Yamanaka, S. Matsumoto, G2GML: Graph to graph mapping language for bridging RDF and property graphs, in: *International Semantic Web Conference*, Springer, 2020, pp. 160–175.