N3.js Reasoner: Implementing reasoning in N3.js

Wright, Jesse[1]

[1]Computer Science Department, University of Oxford, UK

Abstract

RDF reasoning typically does not occur in browser-based applications despite being critical to the next generation of Web technologies. The primary constraint is that client-side reasoners do not meet the *performance* requirements of modern applications and the technicalities of using reasoners make them *inaccessible* to front-end developers.

This paper presents our *performant* implementation of a reasoning engine for N3.js that supports Horn Rules. We demonstrate that it is possible to perform reasoning in the browser in a manner that is performant enough for standard use cases to execute without interrupting the user experience.

Keywords

Reasoner, Inference, RDFJS, RDF, JavaScript, TypeScript, Web, Browser, Solid

1. Introduction

RDF reasoning has been integral to the Semantic Web since its inception in 2001 [1]. Most RDF reasoners have been developed in Java or C++ for desktop or server environments [2]. However, server-side reasoning alone is inadequate for a decentralised Semantic Web like Solid [3], where data are distributed across many sources, including local files, Solid Pods, and public knowledge graphs. Client-side reasoners are necessary to handle inferences from multiple sources and to apply inferences to local documents or results from less-intelligent servers.

N3.js [4] is a widely-used JavaScript library in the Semantic Web ecosystem. It has 589 stars on GitHub, 336 downstream packages, including Wikidata [5], and 2605 dependent GitHub repositories. Since N3.js implements the RDF JS model specification [6] and the RDF JS dataset specification [7], our reasoning engine is compatible with RDF JS libraries and applications out of the box.

2. Implementation

We implement Horn Logic [8] reasoning in N3.js by applying the *semi-naive* reasoning algorithm [9] alongside a bespoke technique for indexing rules in-memory to optimise rule evaluation against the 3-layered index of N3.js - which we detail in the remainder of this section.

First, we describe the existing N3.js store index. For each triple in the store, the *subject*, *predicate* and *object* are converted to a canonical string. An internal record maintains a mapping

Posters, Demos, and Industry Tracks at ISWC 2024, November 13–15, 2024, Baltimore, USA

jesse.wright@cs.ox.ac.uk (W. Jesse)

https://www.cs.ox.ac.uk/people/jesse.wright/ (W. Jesse)

0000-0002-5771-988X (W. Jesse)

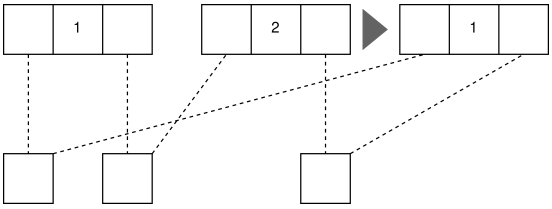
between these canonical strings and a numerical ID. Triples are stored by adding these numerical IDs as keys in nested indexes (dictionaries). The N3.js store uses three, three-layered indexes to store data. The depth of these indexes reflects the fact that a triple has three elements: subject (**s**), predicate (**p**) and object (**o**). The three indexes are ordered as **osp**, **spo** and **pos**.

We assume that most applications have a limited number of rules, i.e. less than 100 as with RDFS and OWL2RL inference. Consequently our reasoner implementation[1] optimises to reduce the time complexity of reasoning with respect to *dataset* size. The first step of reasoning in N3.js is re-writing rules to: (1) convert each non-variable term in the premise and conclusion into the internal ID used by the N3.js store index; (2) convert each variable into a pointer to a shared memory location into which concrete values for the variable will be substituted (c.f. Figure 2a); (3) identify matching patterns between a rule's conclusion and the premise(s) of the same or other rules (c.f. Figure 3a) to establish which rules need to be evaluated next when *new* implicit data is discovered; and (4) precompute which of the 3 N3.js store indexes should be used for looking up triples matching a rule premise. Figure 1b shows how the rule **{?s a ?o . ?o rdfs:subClassOf ?o2} → {?s a ?o2}** (**R1**) is stored in memory with respect to the index mapping in Table 1a. Figure 3a displays the dependencies between **{?o1 rdfs:subClassOf ?o2 . ?o2 rdfs:subClassOf ?o3 } → {?o1 rdfs:subClassOf ?o3}** (**R2**) and **R1**.

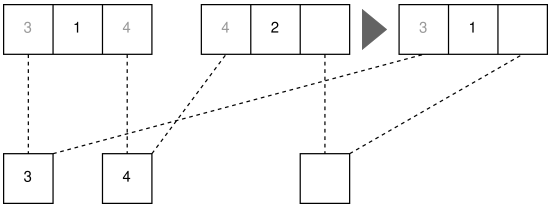
Rules are then evaluated as follows: (1) perform a nested loop join on the premises, the outermost loop iterates through triples matching the first premise - substituting bindings into the variable memory locations (c.f. Figure 2a), subsequent nested loops iterate over the remaining premises and iterate through all triples matching the partially-bound pattern (c.f. Figure 2b); (2) new conclusion triples found added to the three-layered indexes; (3) for each new conclusion, identify matching premises from the same, or other, rules (c.f. Figure 3a); and (4) perform the subsequent reasoning run using rules with these premises pre-filled (c.f. Figure 3b) to avoid re-evaluating the same patterns across reasoning runs.

1	rdf:type
2	rdfs:subClassOf
3	timbl:me
4	ex:computerScientist
5	foaf:Person
6	ex:computerScienceProfessor
7	ex:Armin

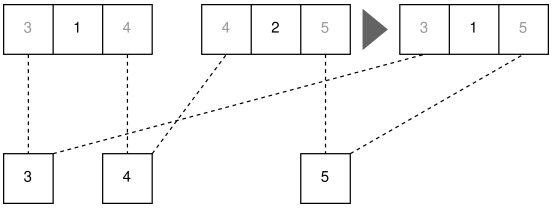
(1a) Mapping between IDs (left) and IRIs (right)



(1b) In-memory representation of **R1**

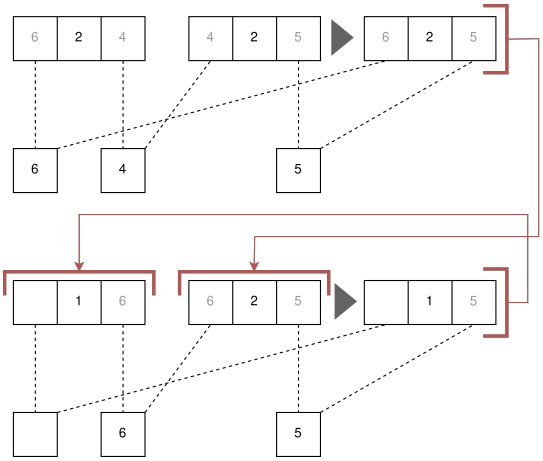


(2a) Matching first premise of **R1** against a triple

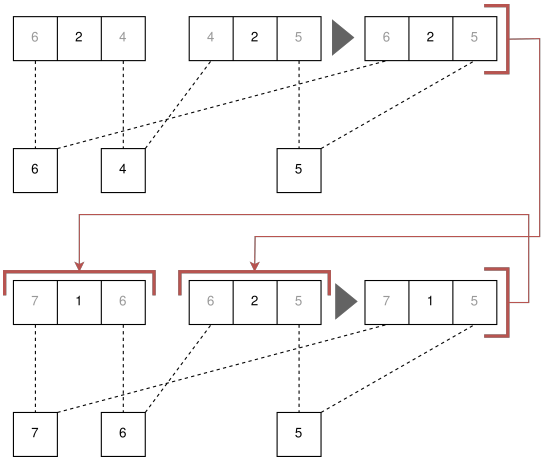


(2b) Matching second premise of **R1** against a triple

[1] https://github.com/rdfjs/N3.js/pull/296



(3a) The dependency between **R2** and **R1** is stored and can be used to pre-fill variables in **R1** after **R2** has been evaluated.



(3b) Matches to the first premise of **R1** can then be iterated through to produce new derivations on subsequent reasoning runs.

3. Performance Results and Conclusion

We evaluate this N3.js reasoner against the only other known browser-available reasoning engines: EYE JS [10], a WebAssembly port of EYE [11], and the HyLAR Reasoner [12]. The results in Table 1 and Table 2 were collected at https://github.com/jeswr/demo-perf-tests/ on a GitHub Actions runner with 2 cores and 7GB of memory, running Ubuntu 22.04. NodeJS results were collected from commit a51bc3e. Chrome and Firefox were run headless using Selenium[2] and the performance results are collected from commit 33cbfb4.

Table 1 presents the time taken to perform RDFS materialisation on Tim Berners-Lee's profile card and the FOAF ontology [13]. The N3.js reasoner outperforms the others, completing the task in under 0.1s, which is within the threshold for an instantaneous user perception [14].

Table 2 presents the time taken to materialise all inferences for the the Data Deep Taxonomy Benchmark (DTB) using the N3.js Reasoner, EYE JS, and HyLAR. This modification of the deep taxonomy benchmark encodes subclasses as facts (**:N0 rdfs:subClassOf :N1**) rather than rules (**?X rdf:type :N0** → **?X rdf:type :N1**), as the N3.js reasoner is optimised to handle a large number of facts. The depth variable indicates the number of nested subclasses in the dataset. The extended (ext) taxonomy benchmark scales the number of instances of the class **:N0** with the depth; consequently the number of implicit facts scales quadratically with depth.

The N3.js Reasoner primarily outperforms HyLAR because (1) HyLAR does not index triples or rules; consequently, HyLAR iterates through all facts when matching each premise taking $O(n)$ time (2) HyLAR does not create internal representations for terms and triples - resulting in (2a) costly string and object comparison operations for rule matching where N3.js is comparing integers and (2b) costly object creation where N3.js is adding integers to an existing index.

These results show that N3.js can efficiently reason over moderately-sized datasets in the browser, proving that we can now perform RDF inference in the client.

[2]https://www.selenium.dev/

Table 1
Time taken to apply RDFS inference to Tim Berners-Lee's profile card and the FOAF ontology [13]. In this experiment, there are 14 rules, 961 facts and 866 derivations.

Depth	N3.js Reason	EYE JS	Hylar
NodeJS	23ms	876ms	126ms
Chrome	28ms	962ms	104ms
Firefox	43ms	1573ms	129ms

Table 2
Result of the Data Deep Taxonomy Benchmark (DTB) in NodeJS, Chrome and Firefox. TIMEOUT occurs after 6 hours for NodeJS and 1 hour for Chrome and Firefox. Times for Firefox are only available to the nearest millisecond.

NodeJS						
Depth	N3.js Reason	EYE JS	Hylar	N3.js Reason (ext)	EYE JS (ext)	Hylar (ext)
10^1	0.185ms	32.2ms	2.91ms	1.30ms	117ms	11.0ms
10^2	1.67ms	484ms	527ms	131ms	31.7s	20.0s
10^3	13.4ms	28.1s	333s	13.4s	TIMEOUT	TIMEOUT
10^4	58.9ms	2540s	TIMEOUT	MemErr	TIMEOUT	TIMEOUT
10^5	618ms	TIMEOUT	TIMEOUT	MemErr	TIMEOUT	TIMEOUT
10^6	MemErr	TIMEOUT	TIMEOUT	MemErr	TIMEOUT	TIMEOUT
Chrome						
Depth	N3.js Reason	EYE JS	Hylar	N3.js Reason (ext)	EYE JS (ext)	Hylar (ext)
10^1	0.110ms	51.5ms	11.6ms	0.575ms	203ms	15.0ms
10^2	0.728ms	594ms	406s	73.1ms	44.2s	17.9s
10^3	8.35ms	34.9s	297s	7.75s	TIMEOUT	TIMEOUT
10^4	55.5ms	TIMEOUT	TIMEOUT	MemErr	TIMEOUT	TIMEOUT
10^5	403ms	TIMEOUT	TIMEOUT	MemErr	TIMEOUT	TIMEOUT
10^6	6.16s	TIMEOUT	TIMEOUT	MemErr	TIMEOUT	TIMEOUT
Firefox						
Depth	N3.js Reason	EYE JS	Hylar	N3.js Reason (ext)	EYE JS (ext)	Hylar (ext)
10^1	0ms	62ms	4ms	0ms	408ms	15ms
10^2	1ms	989ms	670ms	147ms	55.9s	29.1s
10^3	13ms	45.9s	469s	5.17s	TIMEOUT	TIMEOUT
10^4	96ms	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT
10^5	933ms	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT
10^6	12.9s	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT	TIMEOUT

4. Usage

The reasoner is now distributed with N3.js; making it easily accessible to JavaScript developers via NPM. Documentation can be found in the README.

Acknowledgements

Jesse Wright is funded by the Department of Computer Science, University of Oxford.

References

[1] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific american 284 (2001) 34–43.

[2] A. Khamparia, B. Pandey, Comprehensive analysis of semantic web reasoners and tools: a survey, Education and Information Technologies 22 (2017) 3121–3145.

[3] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, Solid protocol, https://solidproject.org/TR/2021/protocol-20211217 (2021).

[4] R. Verborgh, R. Taelman, J. Wright, S. V. Braeckel, L. Rietveld, D. Hurlburt, K. Woudt, T. Bergwinkl, Vincent, T. Tanon, S. ROZE, N. D. Martin, J. Smart, M. Maillard, Martin, M. Fathi, P. Colpaert, P. Heyvaert, Ruben, Shawn, W. Turner, alxflam, elf Pavlik, L. Roy, J. D. Smet, J. Scazzosi, I. Aaronsohn, H. Zuo, G. Middell, rdfjs/n3.js: v1.17.3, 2024. URL: https://doi.org/10.5281/zenodo.10866356. doi:10.5281/zenodo.10866356.

[5] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (2014) 78–85.

[6] T. Bergwinkl, M. Luggen, elf Pavlik, B. Regalia, P. Savastano, R. Verborgh, RDF/JS: Data model specification, W3C Community Group Draft Report, W3C, 2023. http://rdf.js.org/data-model-spec/.

[7] T. Bergwinkl, B. Regalia, V. Felder, R. Taelman, RDF/JS: Dataset specification 1.0, W3C Community Group Final Report, W3C, 2019. https://rdf.js.org/dataset-spec/.

[8] B. N. Grosof, I. Horrocks, R. Volz, S. Decker, Description logic programs: Combining logic programs with description logic, in: Proceedings of the 12th international conference on World Wide Web, 2003, pp. 48–57.

[9] S. Abiteboul, R. Hull, V. Vianu, Foundations of databases, volume 8, Addison-Wesley Reading, 1995, pp. 312–316.

[10] J. Wright, J. D. Roo, I. Smessaert, P. Hochstenbach, W. Slabbinck, eyereasoner/eye-js: v16.22.0, 2024. URL: https://doi.org/10.5281/zenodo.13632329. doi:10.5281/zenodo.13632329.

[11] R. Verborgh, J. De Roo, Drawing conclusions from linked data on the web: The eye reasoner, IEEE Software 32 (2015) 23–27.

[12] M. Terdjimi, L. Médini, M. Mrissa, Hylar: Hybrid location-agnostic reasoning, in: ESWC Developers Workshop 2015, 2015, p. 1.

[13] D. Brickley, L. Miller, Foaf vocabulary specification 0.91, 2007.

[14] F. F.-H. Nah, A study on tolerable waiting time: how long are web users willing to wait?, Behaviour & Information Technology 23 (2004) 153–163.