Extending the SPARQL Algebra for the optimization of Property Paths

Louis Jachiet, Pierre Genevès, Nabil Layaïda and Nils Gesbert

ENS, Inria, Université Grenoble Alpes, LIG

Introduction

SPARQL is the W3C standard language for querying RDF graphs. Since its 1.1 version, SPARQL allows queries with Property Paths. They correspond to a form of regular path queries over graphs that raises new challenges for SPARQL evaluators and RDF stores.

Related Work

The evaluation of the conjunctive fragment of SPARQL is a well-studied subject. Property Paths can be recursive and the optimization of recursive queries remains a challenge both in the relational and in the semantic web world [**?**].

Our proposal

We present μ-algebra, a variation of the SPARQL Algebra that allows for the optimization of SPARQL especially with Property Paths. Similarly to the SPARQL Algebra, SPARQL 1.1 queries can be translated into our algebra but the obtained μ-algebra terms can be rewritten into multiple equivalent terms. Each of these terms is a possible execution plan of the initial SPARQL query.

Our μ-algebra algebra

Our proposed algebra is:

- derived from the SPARQL algebra
- equipped with fixpoints
- equipped with a type analysis
- allowing powerful rewriting rules
- compilable into efficient code
- compilable into distributed code

Algebra

$\varphi ::=$	formula
$\varphi_1 \cup \varphi_2$	union
$\varphi_1 \backslash\backslash \varphi_2$	normal minus
$\varphi_1 - \varphi_2$	set minus
$\varphi_1 \setminus \varphi_2$	strict minus
$\varphi_1 \ltimes \varphi_2$	left-join
$\varphi_1 \bowtie \varphi_2$	join
$\rho_a^b(\varphi)$	exchange column (or rename)
$\pi_a(\varphi)$	column dropping (projection)
$\beta_a^b(\varphi)$	column multiplying
$\theta(\varphi, f : \mathcal{C} \to \mathcal{D})$	UDF map
$\Theta(\varphi, f : \mathcal{C} \to \mathcal{D})$	UDF reduce
$\sigma_f(\varphi)$	row filtering
$\mu(X = \varphi)$	fixpoint
let $(X = \varphi)$ in ψ	let-binder
X	variable
$\emptyset$	no mapping
$\lvert c_1 \to v_1, \ldots, c_n \to v_n \rvert$	constant

Rewritings

Nam quis odio enim, in molestie libero. Vivamus cursus mi at nulla elementum sollicitudin. Nam quis odio enim, in molestie libero. Vivamus cursus mi at nulla elementum sollicitudin. Nam quis odio enim, in molestie libero. Vivamus cursus mi at nulla elementum sollicitudin. Nam quis odio enim, in molestie libero. Vivamus cursus mi at nulla elementum sollicitudin.

Fixpoints rewritings

We implemented a prototype based on this algebra equipped

Treatments	Response 1	Response 2
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Table 1:Table caption

Experiments

Nunc tempus venenatis facilisis. Curabitur suscipit consequat eros non porttitor. Sed a massa dolor, id ornare enim:

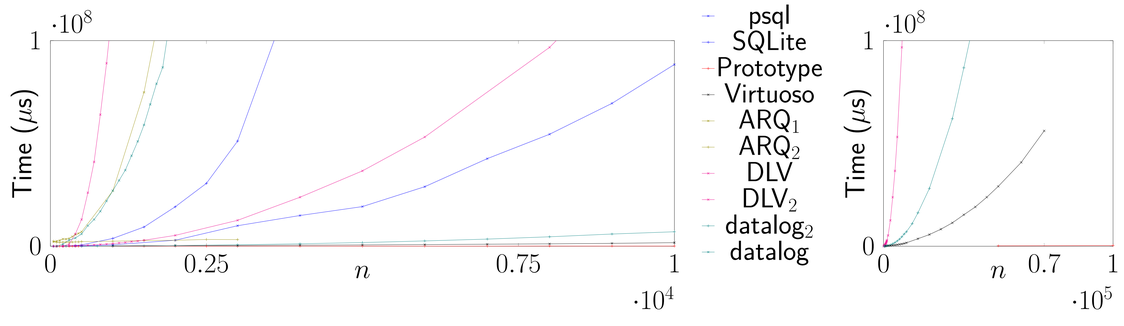

Figure 1:Evaluation time for the query $\{?x\ knows*\ ?y\ .?x\ named\ name\_42.\}$ on a graph of size n

Conclusion

Our method generates terms corresponding to execution plans that are not considered by existing methods and we experimentally demonstrate the advantages of our approach.
Our prototype that translates, optimizes and evaluates SPARQL queries using our algebra shows on experiments that our method outperforms other existing methods on recursive queries.

Future work

Our algebra can be seen as a step toward the ambitious goal of unifying various traits of the relational algebra with traits of NoSQL languages in a common framework (syntax, semantics, typing, rewriting schemes). As a perspective for further work, we plan to investigate how our approach can be improved along several directions: finer-grained cardinality estimation, distributed implementations for evaluating terms of our algebra, and extensions for the compilation of query languages with other data models.

Contact Information

- Web: http://tyrex.inria.fr
- Email: research@jachiet.com

References