Verification of Rewriting-based Query Optimizers

Krishnamurthy Balaji[1], Piyush Gupta[1], and Aalok Thakkar[1,2]

1 **Adobe Systems, Noida, India**
kbalaji@adobe.com, piygupta@adobe.com
2 **Chennai Mathematical Institute, Chennai, India**
aalok@cmi.ac.in

Abstract

We report on our ongoing work on automated verification of rewriting-based query optimizers. Rewriting-based query optimizers are a widely adapted in relational database architecture however, designing these rewrite systems remains a challenge. In this paper, we discuss automated termination analysis of optimizers where rewrite-rules are expressed in HoTTSQL. We discuss how it is not sufficient to reason about rule specific (local) properties such as semantic equivalence, and it is necessary to work with set-of-rules specific (global) properties such as termination and loop-freeness to prove correctness of the optimizer. We put forward a way to translate the rules in HoTTSQL to Term Rewriting Systems, opening avenues for the use of termination tools in the analysis of rewriting-based transformations of HoTTSQL database queries.

Keywords and phrases Rewriting, Termination, Verification, Query Optimization

1 Introduction

Relational query languages provide a high-level declarative interface to access data stored in relational databases. Before an execution engine implements a set of physical operators to evaluate the query, heuristics are used to optimize the evaluation with respect to parameters such as execution time, monetary fees in a cloud scenario, and allocated and accessed memory [6], [13].

Since 1990s, a standard technique in query optimization is to use a set of rewrite rules which optimize the objective by transforming the input query to a normal form by repeated applications of the rules until a fixed point is reached [14]. Although this technique has been around for long, there has been only little progress towards proving its correctness. The property of semantics preservation under rewriting has been studied and [3] puts forward a method to verify this property for rules expressed in HoTTSQL, a SQL like query language. While semantics preservation is a necessary property of a query optimizer [12], there are other important properties such as termination or loop-freeness which may be desired or necessary depending on application and implementation.

Modern query optimizers, such as Catalyst for SparkSQL, not only support user-defined types but also allow users to add their own optimization rewrite rules [1], creating a need for guarantees that the mentioned properties are invariant of the new additions. Our paper sheds light on the necessary and desired specifications for such query optimizers, and put forward a way to reason about two such properties – termination and loop-freeness. In Sect. 2 we illustrate the problem with an example and discuss the optimizer properties, and in Sect. 3 we discuss translating the HoTTSQL syntax to a syntax which allows automated termination analysis. In Sect. 4 we discuss a weaker specification than termination and in Sect. 5 we conclude with a discussion of our contribution and its limitations.

Proceedings of the 16th International Workshop on Termination;
Editor: Salvador Lucas

2 Optimizer Properties

Although the execution of an optimizer consists of the application of individual query rewriting rules, not all desired properties of the system can be verified locally (by looking at the rules separately); in many cases, it is necessary to look at rules in context of other rules. One such property is rewriting termination.

▶ **Example 1.** Consider the Selection Push Down rule which moves a selection (filter) directly after the scan of the input table to reduce the amount of data in the execution pipeline as early as possible [4].

```
SELECT * FROM R WHERE a AND b →
SELECT * FROM (SELECT * FROM R WHERE a) WHERE b
```

On other hand, if (`SELECT * FROM R WHERE a`) returns a significant fraction of `R`, the following rule may be added by the user:

```
SELECT * FROM (SELECT * FROM R WHERE a) WHERE b →
SELECT * FROM R WHERE a AND b
```

Though both (Selection Push Down, and the user defined rule) are semantics preserving, having the two rules together introduces a loop in the transition graph, which makes the rewriting process non-terminating.

In order to avoid such conflicts, a user must look into the predefined rules and undertake their tedious analysis in order to check if a given new rule can be added. In this case, one looks at termination. Another property that is often desirable is confluence. In a terminating rewrite system, local confluence is equivalent to confluence, and hence one can check for local confluence to conclude existence and uniqueness of normal forms [2] [11]. The following three properties can be identified as desired specifications for a rewriting-based query optimizer that ensures a unique normal form:

- **Termination:** For all queries Q, any sequence of queries obtained by rewriting starting with Q should terminate.
- **Local Confluence:** For all queries Q, if there are rewrite rules r_1 and r_2 such that $Q \xrightarrow{r_1} Q_1$ and $Q \xrightarrow{r_2} Q_2$, then there exists a query Q' such that $Q_1 \xrightarrow{\hat{\rho}_1} Q'$ and $Q_2 \xrightarrow{\hat{\rho}_2} Q'$ where $\hat{\rho}_1$ and $\hat{\rho}_2$ are sequences of rewrite rules.
- **Semantics Preservation:** If $Q \xrightarrow{r} Q'$ then the result obtained by executing Q on any database D should be the same as the result obtained by executing Q' on D.

3 Reasoning about Termination

HoTTSQL is an SQL-like language that was proposed for checking semantics preservation [3]. The following is a summary of HoTTSQL syntax:

$$
\begin{aligned}
q \in \text{ Query } &::= \mathsf{TABLE} \mid \mathsf{SELECT}\ p\ q \mid \mathsf{FROM}\ q_1, \ldots, q_n \mid q\ \mathsf{WHERE}\ p \\
&\mid q_1\ \mathsf{UNION\ ALL}\ q_2 \mid q_1\ \mathsf{EXCEPT}\ q_2 \mid \mathsf{DISTINCT}\ q \\
b \in \text{ Predicate } &::= e_1 = e_2 \mid \mathsf{NOT}\ b \mid b_1\ \mathsf{AND}\ b_2 \mid b_1\ \mathsf{OR}\ b_2 \mid \mathsf{true} \mid \mathsf{false} \\
e \in \text{ Expression } &::= P2E\ p \mid \mathsf{f}(e_1, \ldots, e_n) \mid \mathsf{agg}(q) \mid \mathsf{CASTEXPR}\ p\ e \\
p \in \text{ Projection } &::= * \mid \mathsf{Left} \mid \mathsf{Right} \mid \mathsf{Empty} \mid p_1.p_2 \mid p_1, p_2 \mid E2P\ e
\end{aligned}
$$

We first translate queries from HoTTSQL to Term Rewriting Systems (TRS) [7] as used in inputs in Termination Problems Data Base (TPDB) so that automated tools for termination analysis such as Tyrolean Termination Tool 2 (TTT2) [8], Automated Program Verification Environment (AProVE) [5], and others. We introduce the following functions:

$$
\begin{aligned}
\text{Query Functions } &: \mathsf{select(p,q)} \mid \mathsf{from1(q)} \mid \mathsf{from2(q1,q2)} \mid \mathsf{where(q,p)} \\
&\mid \mathsf{unionall(q1,q2)} \mid \mathsf{except(q1,q2)} \mid \mathsf{distinct(q)} \\
\text{Predicate Functions } &: \mathsf{eq(e1,e2)} \mid \mathsf{not(b)} \mid \mathsf{and(b1,b2)} \mid \mathsf{or(b1,b2)} \\
\text{Expression Functions } &: \mathsf{p2e(p)} \mid \mathsf{agg(q)} \mid \mathsf{castexpr(p,e)} \\
\text{Projection Functions } &: \mathsf{projdot(p1,p2)} \mid \mathsf{projcom(p1,p2)} \mid \mathsf{e2p(e)}
\end{aligned}
$$

We consider the following as constants:

$$\mathsf{true},\ \mathsf{false},\ \mathsf{star},\ \mathsf{left},\ \mathsf{right},\ \mathsf{empty}$$

Note that we use from1 and from2 to encode FROM of different arities. Higher arities of FROM are encoded using composition of from2. The functions $\mathsf{f}(e_1, \ldots, e_n)$ used to form expressions may be encoded as the functions themselves on a case-to-case basis. Each instance of a TABLE is encoded as a separate variable. The translation from HoTTSQL syntax to the proposed functional syntax is canonical and can be implemented in linear time using a stack.

▶ **Example 2.** We shall encode the Selection Push Down rule from Example 1 in the functional syntax. r, a, and b are variables. Then we have:

select(star,where(from1(r),and(a,b))) →
select(star,where(select(star,where(from1(r),a)),b))

This syntax is designed to be compatible with Tyrolean Termination Tool 2, and so if the rules are once converted from the HoTTSQL syntax to TRS syntax, it can be given as an input to the tool to automatically reason about termination.

4 Weaker Specifications

A major challenge of rewriting systems is that the rewriting process may be of a high computational complexity [10] [15], and we see the same translate to runtime complexity of rewriting based optimizers. In that scenario, the reduction in the runtime for query evaluation may be compensated by the increase in the runtime for query rewriting. A standard ad-hoc method is to introduce a threshold on the number of rewrites [1]. For a rewriting based optimizer that implements this heuristic, termination is not a necessary

requirement[1]. However, the presence of loops can lead to redundant rewritings. We can adapt the results in [9] to check for loop-freeness of by using existing tools for the automatic generation of first-order models.

5 Conclusion

We have put forward a discussion of verification of rewriting-based query optimizers, particularly the need of looking at set-of-rules specific (global) properties, particularly termination and loop-freeness. We have also proposed a translation from HoTTSQL to the Term Rewriting Systems syntax.

A major limitation is the expressiveness of HoTTSQL. A number of rules which are used in practice such as the Predicate Pushdown Rule need pattern matching for their implementation, however, the HoTTSQL syntax does not offer it. We put forward a *wish list* of the types of query-rewrite rules we would like to add to classic term rewriting translation that is proposed in Section 3 in order to encode the rules of popular query optimizers like Catalyst. The following are the three possible extensions:

1. Classic Term Rewriting + Variable Matching: Rules such as predicate pushdown can be encoded by matching bound and free variables according to the schema
2. Classic Term Rewriting + Constrained Rewriting: Rules with integral constraints can be encoded using constrained rewriting
3. Classic Term Rewriting + Pattern Matching: All query-rewrite rules can be encoded by general pattern matching

The future work branches in three directions. Firstly, we would like to work towards developing developer tools which help design rewrite rules keeping the mentioned specifications in perspective. Secondly, we would like to extend the scope of HoTTSQL and our syntax as mentioned in the previous paragraph so we may be able to reason about query optimizers in practice. Finally, we would like to understand other desired and necessary properties of the optimizers, particularly confluence and develop a fuller theory for verified rewriting-based query optimizers.

References

1 Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 1383–1394, New York, NY, USA, 2015. ACM.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

3 Shumo Chu, Konstantin Weitz, Alvin Cheung, and Dan Suciu. Hottsql: Proving query rewrites with univalent sql semantics. *SIGPLAN Not.*, 52(6):510–524, June 2017.

4 Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database systems - the complete book (2. ed.).* Pearson Education, 2009.

[1] Though it may be necessary if one wishes to use it to reason about confluence and existence and uniqueness of normalized form

5 Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with aprove. *Journal of Automated Reasoning*, 58(1):3–31, Jan 2017.

6 Matthias Jarke and Jürgen Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2):111–152, June 1984.

7 J. W. Klop. Handbook of logic in computer science (vol. 2). chapter Term Rewriting Systems, pages 1–116. Oxford University Press, Inc., New York, NY, USA, 1992.

8 Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean termination tool 2. In Ralf Treinen, editor, *Rewriting Techniques and Applications*, pages 295–304, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

9 Salvador Lucas. A semantic approach to the analysis of rewriting-based systems. *CoRR*, abs/1709.05095, 2017.

10 Georg Moser, Andreas Schnabl, and Johannes Waldmann. Complexity Analysis of Term Rewriting Based on Matrix and Context Dependent Interpretations. In Ramesh Hariharan, Madhavan Mukund, and V Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 2 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 304–315, Dagstuhl, Germany, 2008. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

11 M. H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

12 S. T. Shenoy and Z. M. Ozsoyoglu. Design and implementation of a semantic query optimizer. *IEEE Transactions on Knowledge and Data Engineering*, 1(3):344–361, Sep 1989.

13 Immanuel Trummer and Christoph Koch. Approximation schemes for many-objective query optimization. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1299–1310, New York, NY, USA, 2014. ACM.

14 Sieger van Denneheuvel, Karen Kwast, Gerard R. Renardel de Lavalette, and Edith Spaan. Query optimization using rewrite rules. In Ronald V. Book, editor, *Rewriting Techniques and Applications*, pages 252–263, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

15 Chihping Wang and Ming-Syan Chen. On the complexity of distributed query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):650–662, Aug 1996.