Simulating classical query rewriting algorithms with SLD-resolution

Christophe Rey, Elias Tahhan-Bittar, Jerzy Tomasik

Simulating classical query rewriting algorithms with SLD-resolution

Christophe Rey
Université Clermont Auvergne
LIMOS, IUT d'Allier
Clermont-Ferrand, France
christophe.rey@uca.fr

Elias Tahhan-Bittar
Universidad Simon Bolivar
Caracas, Venezuela
etahhan@usb.ve

Jerzy Tomasik
Université Clermont Auvergne
LIMOS, CNRS
Clermont-Ferrand, France
jerzy.tomasik@udamail.fr

ABSTRACT

We present experimental results that indicate that SLD-resolution could be considered as a unifying framework for the studying of query rewriting algorithms. Indeed, adding constraints to the control of SLD-resolution makes it simulate some of the classical query rewriting algorithms used in mediation systems. We propose 4 such constraints and link SLD-resolution to 3 classical algorithms: the bucket, the inverse-rules and the MINICON algorithms.

CCS CONCEPTS

• **Information systems → Relational database model**; **Mediators and data integration**; • **Theory of computation → Constraint and logic programming**;

KEYWORDS

SLD-resolution ; maximally contained rewritings ; certain answers ; definite program ; conjunctive query; local-as-view ; mediation ; data integration ; rewriting algorithm ; open world assumption ; algorithmic framework

ACM Reference Format:
Christophe Rey, Elias Tahhan-Bittar, and Jerzy Tomasik. 2018. Simulating classical query rewriting algorithms with SLD-resolution . In *Proceedings of Unpublished.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In data integration, more specifically in the setting of mediation, the problem of answering queries using views has been studied for about 25 years [2, 3, 9, 18]. The basic issue, in the relational database setting, aims at querying many heterogeneous data sources from a single user query expressed on a virtual schema $\mathcal{G}$ and using only a set $\mathcal{S}$ of materialized views defined on the sources schemas. A well-studied approach is the query rewriting one in which the user query is rewritten as possibly many queries, each of them being expressed exclusively using available views. Generally, the obtained rewritings are the maximally contained rewritings under the certain answers semantics [1]. Many alorithmical approaches have been proposed [6–8, 11–13, 15–17]. Among these, [8] shows that the classical SLD-resolution procedure, which is at the heart of logic programming and deductive databases, enables to compute such rewritings. This idea is also further evoked in [11]. SLD-resolution [14] is based on the resolution inference rule which rewrites a definite goal into another one using a definite clause, by unifying the head of this definite clause to an atom of the initial definite goal. The choice of both the definite clause among a definite program (i.e. a finite set of definite clauses), and of the atom among all the atoms of the definite goal, is called the computation rule $\Re$ of SLD-resolution.

The on-going work presented in this paper is about how we can envision different SLD-resolution strategies (e.g. by choosing a specific computation rule $\Re$) to make it simulates classical rewriting algorithms, namely the bucket [12], the inverse-rules [5] and the MINICON algorithms [15]. This work is mainly based on an experimental study and still needs future formal proofs to be acertained.

In section 2, we recall some notions about mediation and SLD-resolution and how the latter can be used to compute query rewritings. We sum up this by an elegant property concerning the antichains of resolution trees. In section 3, we present our experimental results based on an implementation of SLD-resolution with varying strategies. These results are essentially a set of 4 constraints that can make SLD-resolution simulate classical query rewriting algorithms. We then conclude.

2 SLD-RESOLUTION AND MEDIATION

2.1 Recalls about mediation

In order to link views of $\mathcal{S}$ to relations of $\mathcal{G}$, mediation systems are given a set $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ of mappings. There are three kinds of mappings: local-as-view (LAV), global-as-view (GAV) and local-and-global-as-view (GLAV). LAV mappings express views as queries using relations of $\mathcal{G}$ and thus need query rewriting algorithm to process a user query. GAV mappings express relations of $\mathcal{G}$ as queries using views as relations and thus allow query processing by query unfolding. GLAV mappings are containment relationships between couples of queries made of one query using views as relations and one query using relations of $\mathcal{G}$. As LAV ones, they imply query rewriting. In this work, we focus on LAV mappings since classical algorithms are designed to handle them.

Example 2.1. This example is about scientific papers, denoted by numbers, and their topics and references. We have:

- $\mathcal{G} = \{cites^{(2)}, sameTopic^{(2)}\}$, which means there are 2 relations in $\mathcal{G}$ each of which have arity 2.
- $\mathcal{S} = \{v4^{(1)}, v5^{(2)}, v6^{(2)}\}$
- $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ contains the following three mappings:

- $v4(X) \leftarrow cites(X, Y), cites(Y, X)$
- $v5(X, Y) \leftarrow sameTopic(X, Y)$
- $v6(X, Y) \leftarrow cites(X, Z), cites(Z, Y), sameTopic(X, Z)$

These mappings are conjunctive queries. For example, the first mapping says that for each X which is a singleton data of $v4$, it must be the first part of a tuple $< X, Y >$ of the relation $cites$ and the second part of a tuple (not necessarily another one) $< Y, X >$ of $cites$. This, of course, if $\mathcal{G}$ were materialized.

- The query q is the following:
  $q(A_0) \leftarrow cites(A_0, B_0), cites(B_0, A_0), sameTopic(A_0, B_0)$
  which means the user is looking for all paper number A_0 that cites a paper B_0 having the same topic as and which cites A_0.

The maximally contained rewriting is the query q':
$q'(A_0) \leftarrow v6(A_0, A_0)$.
which means that only view $v6$ may contains certain answers to the user query.

2.2 Recalls about SLD-resolution

SLD-resolution is a procedure which is based on the exhaustive application of the resolution inference rule to a definite goal [14]. In the previous example, the conjunctive query q is such a definite goal. When we trace all possible applications of the inference rule to the definite goal, we obtain the resolution tree. If one branch of this tree ends with a contradiction, then it means that a solution of the goal has been found.

Used to compute query rewritings, SLD-resolution must get a definite program that is equivalent to $\mathcal{M}_{\mathcal{G},\mathcal{S}}$. [5, 8, 11] show that this definite program is obtained by inversing mappings into the so-called inverse-rules.

Example 2.2.
The inverse-rules of $\mathcal{M}_{\mathcal{G},\mathcal{S}}$ from the previous example are:
$cites(X, f1(X)) \leftarrow v4(X)$.
$cites(f1(X), X) \leftarrow v4(X)$.
$sameTopic(X, Y) \leftarrow v5(X, Y)$.
$cites(X, f3(X, Y)) \leftarrow v6(X, Y)$.
$cites(f3(X, Y), Y) \leftarrow v6(X, Y)$.
$sameTopic(X, f3(X, Y)) \leftarrow v6(X, Y)$.
We remark the appearance of functional terms which stand for existential variables.
Then, the resolution inference rule can for example be applied on q as follows:

- the $cites(A_0, B_0)$ atom from q can be unified with the head atom $cites(X_1, f1(X_1))$ of the first inverse-rule (in order to avoid variable capture, variables of the inverse-rules are always refreshed before each resolution rule application).
- the unifier is the following substitution: $\{X_1/A_0, B_0/f1(A_0)\}$
- the new goal defining q is now
  $q(A_0) \leftarrow v4(A_0), cites(f1(A_0), A_0), sameTopic(A_0, f1(A_0))$

The resolution tree is given in figure 1.

In the previous example, no branch of the resolution tree ends with a contradiction. This is explained by the fact that the input definite program (i.e. the inverse-rules) does not contain any fact. Besides, each node of the resolution tree gives a rewriting of the initial query. By the soundness and completeness of SLD-resolution [4], and if we view the resolution tree as the representation of a partial ordered set (the set of nodes), it is easy to see that each maximal antichain (wrt inclusion) allows to compute all possible answers to our query. In other words, given any maximal antichain of the resolution tree, each answer of the query is mandatorily obtained from one element of this antichain, provided some substitution is applied to it. Up to our knowledge, this way of formulating soundness and completeness of SLD-resolution is new.

Example 2.3. In the previous example, the set containing the leaves of the 5 branches os the resolution tree is an antichain, and it is of course maximal. So, assuming that the views contain data, we know that each certain answer to the initial user query can be obtained by finding a substitution and applying it to one of the five nodes of the antichain. Since there is only one node without functional term, then the maximal contained rewriting is this single node (certain answers cannot contain functional terms), namely: $v6(A_0, A_0), v6(A_0, A_0), v6(A_0, A_0)$ which clearly defines q' as $q'(A_0) \leftarrow v6(A_0, A_0)$.

What is really interesting in the notion of maximal antichain of the resolution tree is that it makes clear the fact that the strategy with which SLD-resolution is executed will have an impact on the way the maximally contained rewriting will be found: it is clear that once an antichain of the resolution tree will be found which elements use only views from $\mathcal{S}$, and not relations from $\mathcal{G}$ anymore, then the maximally contained rewriting is this antichain. So the computation rule $\Re$ may be defined according to this. For example, in[11], the so-called "unit clauses" are used so that contradictions are mandatorily found as leaves of all branches. Now we know that in such an algorithm, $\Re$ must be defined so that these unit clauses are examined after all other clauses, otherwise there may be no maximal antichain of the resolution tree in which will appear the maximally contained rewriting. This is because some atom may have been resolved by the unit clauses.

Coming from the maximal antichain point of view, the idea of SLD-resolution strategies is further developed in next section where we present different SLD-resolution stratégies, experimentally concluding that SLD-resolution can act as some classical query rewriting algorithms provided that it is lead by the right strategy.

3 SLD-RESOLUTION STRATEGIES AND QUERY REWRITING ALGORITHMS

To go further in testing different SLD-resolutions, we have studied three classical query rewriting algorithms, namely the bucket [12], the inverse-rules [5] and the MINICON algorithms [15]. We have identified 4 constraints that we have used as SLD-resolution parameters. These 4 constraints have a same objective which is to avoid generating parts of the rewriting containing functional terms during SLD-resolution. We recall that certain answers cannot contain functional terms, since data in views do not contain functional terms. These 4 constraints are the following:

- C1 During unification, it is not allowed to map distinguished variables (those in the head of the query) to functional terms.
- C2 During unification, it is not allowed to map variables from the clauses body to functional terms.

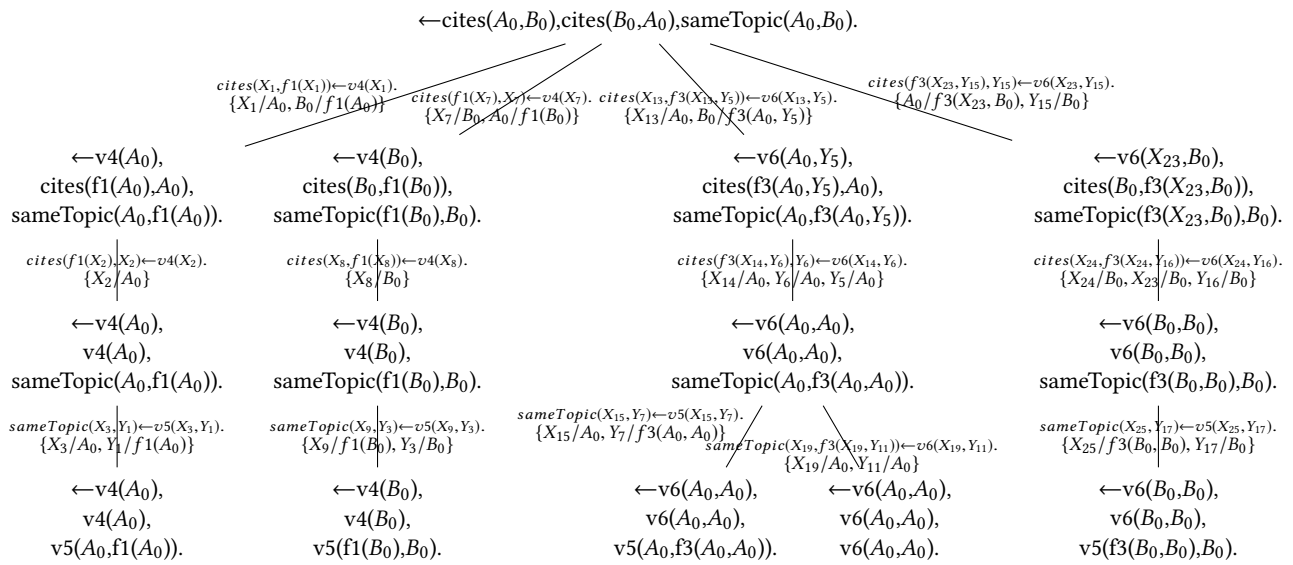

Figure 1: Resolution tree of example 2.2

.

C3 For all existential variable v in the query body, query atoms that contain v must be unified with heads of clauses that have the same body.

C4 All query atom must be unified with only one clause head.

Although we do not have formal proofs yet, after executing SLD-resolution with any subset of these set of 4 constraints, it seems that we have the following results (summed up in table 2):

- SLD-resolution-based query rewriting algorithms [8, 11] follow constraints C3 and C4.
- Once the inverse-rules have been obtained, the SLD-resolution seems to simulate the inverse-rules algorithm [5], provided constraints C2, C3 and C4 are applied.
- Once the inverse-rules have been obtained, SLD-resolution seems to simulate the MINICON algorithm [15], provided constraints C1, C3 and C4 are applied.
- The bucket algorithm [12] follows only constraint C4.

Let's make a few remarks about these results. First, the exploration of the SLD-resolution tree can be executed in a depth-first or in a breadth-first manner. When we say it seems to simulate either the inverse-rules or the MINICON algorithm, then it implies that SLD-resolution is executed in a breadth-first manner. Constraint C1 is the one that corresponds to the creation of MINICON descriptions in the MINICON algorithm. Constraint C2 is implicitely implied during step 2 in the inverse-rules algorithm when it removes functional terms by creating new predicates. C3 is de facto ensured by skolemization during the generation of inverse-rules. C4 is de facto ensured by SLD-resolution.

Example 3.1. Going on with the previous example, figure 3 shows the impact of constraint C1 on the resolution tree exploration.

The case of the bucket algorithm is a bit different from the others. Indeed, the bucket algorithm seems to be a sort of degenerated

Constr.	SLD-resolution [8, 11]	Inverse-rules [5]	Minicon [15]	Bucket [12]
C1			X	
C2		X		
C3	X	X	X	
C4	X	X	X	X

Figure 2: Constraints and corresponding classical query rewriting algorithms.

form of SLD-resolution which would not be applied to the standard inverse-rules, but to naively inverse-rules where no functional term would be introduced. Therefore, the execution of SLD-resolution to such a program would result in wrong parts in the obtained rewriting since unification is far less constrained. This is what explains the final step of the bucket algorithm where the containment of each part of the maximally contained rewriting candidate must be tested.

Example 3.2. In the running example, the degenerated inverse-rules used by the SLD-resolution to simulate the bucket algorithm would be the following ones:
$cites(X, Y) \leftarrow v4(X).$
$cites(Y, X) \leftarrow v4(X).$
$sameTopic(X, Y) \leftarrow v5(X, Y).$
$cites(X, Z) \leftarrow v6(X, Y).$
$cites(Z, Y) \leftarrow v6(X, Y).$
$sameTopic(X, Z) \leftarrow v6(X, Y).$

Our prototype.
We have implemented a parameterized SLD-resolution procedure to be able to test mainly constraints C1 and C2. It has developed

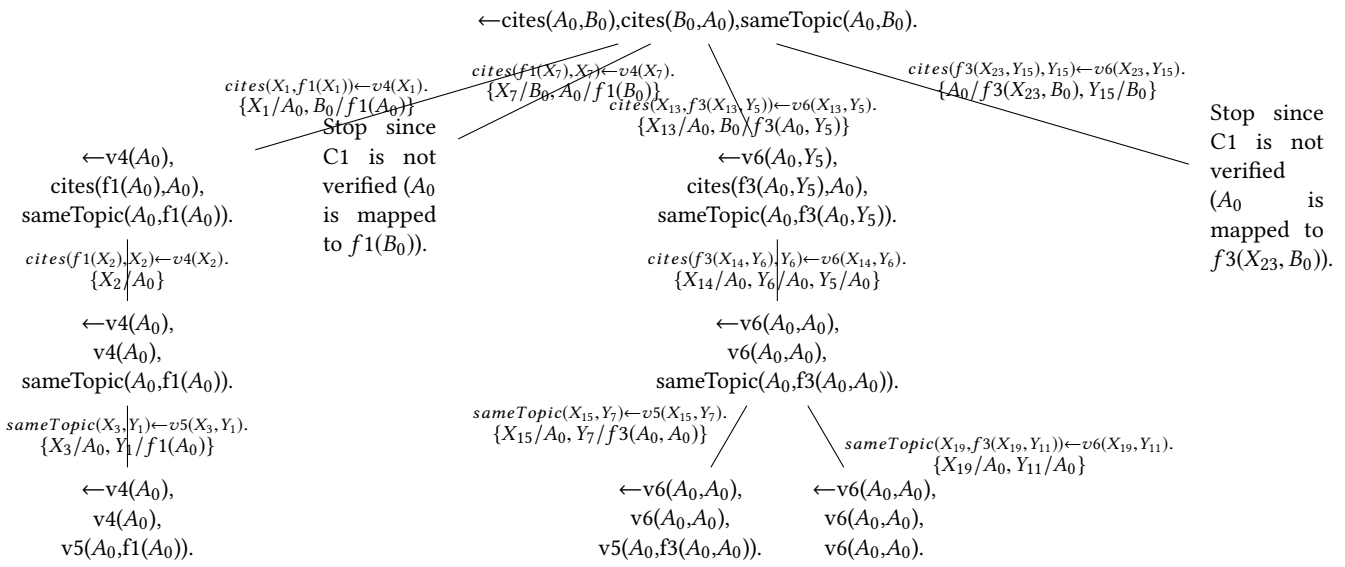

Figure 3: Resolution tree showing the impact of constraint C1.

.

as a meta interpreter with the Goedel programming language [10]. Goedel is a programming language which offers meta programming facilities. It is not maintained anymore since about 20 years. It has been chosen since we would like our meta interpreter to follow a ground approach, in which the first order semantics is clear and ensured. Up to our knowledge, no other existing or past programming language is completely oriented toward such a meta programming paradigm. Using Goedel libraries, our prototype is 8000 lines of code long (5400 for the SLD-resolution and 3600 for tests and examples).

4 CONCLUSION

In this work, we present experimental results that indicate that SLD-resolution could be considered as a unifying framework for the studying of query rewriting algorithms. However, formal proofs still needs to be established. The interest of having such a framework would be to have a way to fairly compare query rewriting algorithms. Of course, classical query rewriting algorithms need not to be studied anymore. But it may be useful for actual query rewriting algorithms, such as the ones in the field of ontological query answering for example.

REFERENCES

[1] S. Abiteboul and O. Duschka. 1998. Complexity of answering queries using materialized views. *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'98)* (1998), 254–265.

[2] Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. 2012. *Web Data Management.* Cambridge University Press. 432 pages. http://hal.inria.fr/hal-00847933 Open access of the full text on the Web.

[3] Foto Afrati and Rada Chirkova. [n. d.]. *Answering Queries Using Views.*

[4] K. Clark. [n. d.]. Predicate logic as a computational formalism.

[5] O.M. Duschka. 1997. Query Optimization Using Local Completeness. In *Proceedings of the Fourteenth AAAI National Conference on Artificial Intelligence, AAAI-97.*

[6] Oliver M. Duschka and Michael R. Genesereth. 1997. Answering Recursive Queries Using Views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 12-14, 1997, Tucson, Arizona, USA.* 109–116. https://doi.org/10.1145/263661.263674

[7] Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. 2000. Recursive Query Plans for Data Integration. *J. Log. Program.* 43, 1 (2000), 49–73. https://doi.org/10.1016/S0743-1066(99)00025-4

[8] John Grant and Jack Minker. 2002. A Logic-based Approach to Data Integration. *Theory Pract. Log. Program.* 2, 3 (May 2002), 323–368. https://doi.org/10.1017/S1471068401001375

[9] Alon Y. Halevy. 2001. Answering queries using views: A survey. *The VLDB Journal* 10, 4 (01 Dec 2001), 270–294. https://doi.org/10.1007/s007780100054

[10] Patricia Hill and John Lloyd. [n. d.]. *The Goedel Programming Language.*

[11] Christoph Koch. 2004. Query rewriting with symmetric constraints. *AI Commun.* 17, 2 (2004), 41–55.

[12] A. Levy, A. Rajaraman, and J. Ordille. 1996. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22nd VLDB Conference, Bombay, India,* T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda (Eds.). Morgan Kaufmann, 251–262.

[13] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. 1995. Answering Queries Using Views. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA.* 95–104. https://doi.org/10.1145/212433.220198

[14] Ulf Nilsson and Jan Maluszynski. 1995. *Logic, Programming, and PROLOG* (2nd ed.). John Wiley & Sons, Inc., New York, NY, USA.

[15] R. Pottinger and A. Halevy. 2001. MiniCon: A scalable algorithm for answering queries using views. *The VLDB Journal* 10, 2-3 (2001), 182–198.

[16] Xiaolei Qian. 1996. Query Folding. In *Proceedings of the Twelfth International Conference on Data Engineering (ICDE '96).* IEEE Computer Society, Washington, DC, USA, 48–55. http://dl.acm.org/citation.cfm?id=645481.655581

[17] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. 1995. Answering Queries Using Templates with Binding Patterns. In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 22-25, 1995, San Jose, California, USA.* 105–112. https://doi.org/10.1145/212433.220199

[18] J.D. Ullman. 2000. Information Integration Using Logical Views. *Theoretical Computer Science* 239, 2 (2000), 189–210.