ASP for Consistent Query Answering

YACINE SAHLI, University of Mons, belgium
JOACHIM SNEESSENS, University of Mons, belgium
MAXIME DANIELS, University of Mons, belgium

Fig. 1

Consistent query answering for inconsistent databases is a running problem. We realized an ASP implementation of the consistent query answering problem and ran some experiments comparing a generate and test method against a first-order rewriting.

CCS Concepts: • **Information systems → Database design and models**; **Database query processing**.

Additional Key Words and Phrases: Answer Set Programming, Consistent Query Answering

ACM Reference Format:
Yacine Sahli, Joachim Sneessens, and Maxime Daniels. 2020. ASP for Consistent Query Answering. In *Galway '20: ACM International Conference on Information and Knowledge Management, October 19–23, 2020, Galway, Ireland.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Galway '20, October 19–23, 2020, Galway, Ireland

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The aim of this article is to present a fair comparison between two methods for solving the problem of CERTAINTY(q). Considering an inconsistent database, a repair is a maximal set of tuples from this database that respects his constraints. The CERTAINTY(q) problem consists in answering the question of knowing if it exists a repair that falsifies the query. Depending on the query, the CERTAINTY(q) problem can have a first order complexity. For the queries that are in first order, we want to compare the efficiency of the generate-and-test method and of the first order rewriting method.

The comparison is realised here on few queries with ASP. For each query, we have measured the execution times of the two methods on databases of different sizes, while distinguishing the yes-instances (databases for which the CERTAINTY(q) problem is true) and the no-instances.

2 CHOSEN QUERIES

To make a one to one comparison with the results found by Akhil A. Dixit and Phokion G. Kolaitis in their "A SAT-Based System for Consistent Query Answering", we decided to reuse the same FO-rewritable queries they used to prove that the KW-fo rewriting can be more efficient by using ASP instead of SQL.

For a more simple implementation, we remove the free variables of the queries. At the end, here are the queries used for the tests we performed.

$$q_1 := \exists x, y, z, v, w(R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, v, w))$$
$$q_2 := \exists x, y, z, v, u, p(R_1(\underline{x}, y, z) \wedge R_3(\underline{y}, v) \wedge R_2(\underline{v}, u, p))$$
$$q_3 := \exists x, y, z, v, u, (R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, v, d))$$

Notice that d is a constant in q_3.

3 QUERIES IMPLEMENTATION IN ASP

3.1 First query

FO Rewriting:

```
q1:-r1(X,Y,Z),not p1(X).
p1(X):-r1(X,Y,Z),not p2(Y).
p2(Y):-r2(Y,V,W).

certainty:-q1.
certainty:-not certainty.

#show certainty/0.
```

Generate and Test:

```
1{rr1(X,Y,Z):r1(X,Y,Z)}1:-r1(X,_,_).
1{rr2(X,Y,Z):r2(X,Y,Z)}1:-r2(X,_,_).
:-rr1(X,Y,Z),rr2(Y,V,W).
```

3.2 Second query

FO Rewriting:

```
q1 :- r1(X,_,_), not p1(X).
p1(X) :- r1(X,Y,_), not q3(Y).
q3(Y) :- r3(Y,_), not q2(Y).
q2(Y) :- r3(Y,V), not q1(V).
q1(V) :- r2(V,_,_).

certainty:-q1.
certainty:-not certainty.

#show certainty/0.
```

Generate and Test:

```
1{rr1(X,Y,Z):r1(X,Y,Z)}1:-r1(X,_,_).
1{rr2(X,Y,Z):r2(X,Y,Z)}1:-r2(X,_,_).
1{rr3(X,Y):r3(X,Y)}1:-r3(X,_).
:-rr1(X,Y,Z),rr3(Y,V),rr2(V,U,D).
```

3.3 Third query

FO Rewriting:

```
q1:-r1(X,Y,Z),not p1(X).
p1(X):-r1(X,Y,Z),not p2(Y).
p2(Y):-r4(Y,V,W),W==w.

certainty:-q1.
certainty:-not certainty.

#show certainty/0.
```

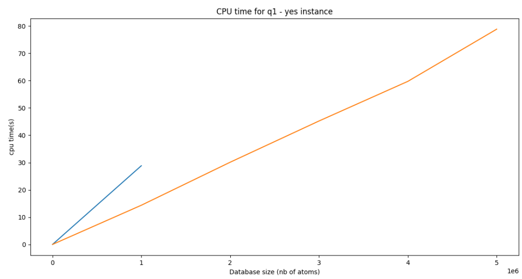
Generate and Test:

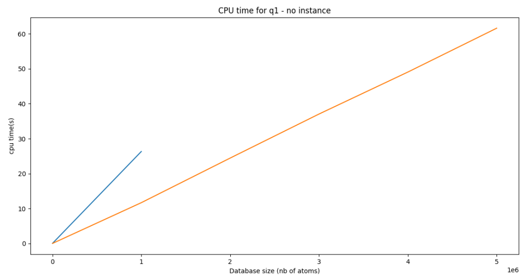
```
1{rr1(X,Y,Z):r1(X,Y,Z)}1:-r1(X,_,_).
1{rr4(X,Y,Z):r4(X,Y,Z)}1:-r4(X,_,_).
:-rr1(X,Y,Z),rr4(Y,V,w).
```

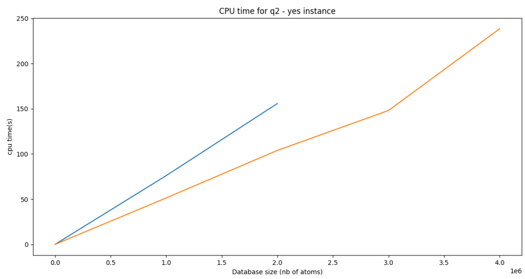
4 RESULTS

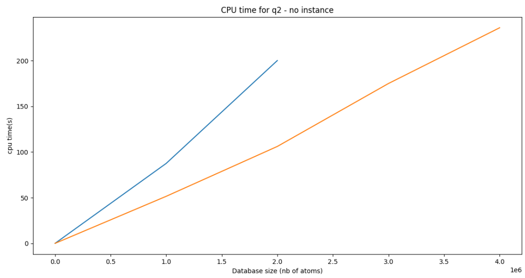
For each graph, the blue line corresponds to the time taken by the generate-and-test method and the orange line to the time taken by the FO method. When a result is not in the graph, that means that the execution of the program was interrupted for insufficient memory. For example, the times for the generate-and-test for the yes-instance for $q1$ for the databases size greater than 1 million are absent.

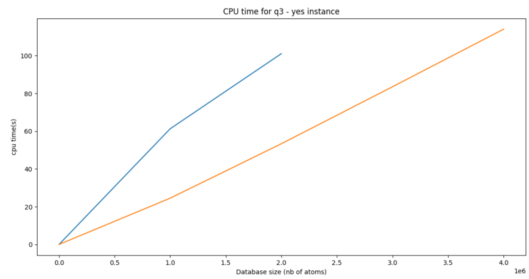
The databases used here were generated specially for the tests through a python script, and have 20 % of inconsistency.

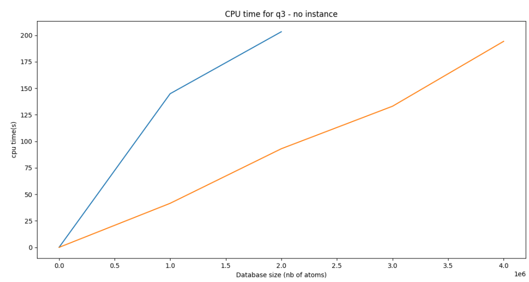

CPU time for q1 - yes instance
cpu time(s)
Database size (nb of atoms)
1e6


CPU time for q1 - no instance
cpu time(s)
Database size (nb of atoms)
1e6


CPU time for q2 - yes instance
cpu time(s)
Database size (nb of atoms)
1e6


CPU time for q2 - no instance
cpu time(s)
Database size (nb of atoms)
1e6





We see that the fo rewriting leads to better results, in terms of cpu time, that the generate-and-test method.

5 CONCLUSION

We rewrited 3 first-order rewritable queries in ASP and showed that the fo rewriting are more efficient than a generate and test method. The first order rewriting consumes a lot less memory than the generate and test which cannot run on 8GB of RAM on a database containing 3 millions or more entries.