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, **??** 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 first order rewritable queries, we are comparing the efficiency of the generate-and-test method against 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.

Inconsistency is a common problem with databases as they grow larger. The primary is not respected anymore and a query cannot get a consistent answer. Certainty(q) is a problem that decide if returns true if the query q is true for every repair of the database.

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.

$$
\begin{aligned}
q_1(z) &:= \exists x, y, v, w(R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, v, w)) \\
q_2(z, w) &:= \exists x, y, v(R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, v, w)) \\
q_3(z) &:= \exists x, y, v, u, d(R_1(\underline{x}, y, z) \wedge R_3(\underline{y}, v) \wedge R_2(\underline{v}, u, d)) \\
q_4(z, d) &:= \exists x, y, v, u, d(R_1(\underline{x}, y, z) \wedge R_3(\underline{y}, v) \wedge R_2(\underline{v}, u, d)) \\
q_5(z) &:= \exists x, y, v, w(R_1(\underline{x}, y, z) \wedge R_4(\underline{y, v}, w)) \\
q_6(z) &:= \exists x, y, x', w, d(R_1(\underline{x}, y, z) \wedge R_2(\underline{x'}, y, w) \wedge R_5(\underline{x}, y, d)) \\
q_7(z) &:= \exists x, y, w, d(R_1(\underline{x}, y, z) \wedge R_2(\underline{y}, x, w) \wedge R_5(\underline{x}, y, d))
\end{aligned}
$$

3 QUERIES IMPLEMENTATION IN ASP

3.1 First order rewriting

Example of a first order rewriting on the fifth query:

$$
\begin{aligned}
q_5(z) := (R_1(\underline{x}, y, z) \wedge R_4(\underline{y, v}, w)) \\
R_1^+ = x, z \\
R_4^+ = y, v \\
AttackGraph : R_1 \ > R_4 \\
WetakeR_1 = RandR_4 = S
\end{aligned}
$$

$$\begin{aligned}
&\exists x \left[\exists y R(\underline{x}, y, z) \wedge \forall y \forall z' \left[R(\underline{x}, y, z') \implies \left[z' = z \wedge \exists v \exists w S(\underline{y}, v, w) \right] \right] \right] \\
\iff\ &\exists x \left[\exists y R(\underline{x}, y, z) \wedge \neg \left[\exists y \exists z' R(\underline{x}, y, z') \wedge \left[z' \neq z \vee \exists v \exists w S(\underline{y}, v, w) \right] \right] \right] \\
\iff\ &\exists x \left[\exists y R(\underline{x}, y, z) \wedge \neg \left(\exists y \exists z' R(\underline{x}, y, z') \wedge z' \neq z \right) \right. \\
&\left. \wedge \neg \left(\exists y \exists z' R(x, y, z') \wedge \neg \left(\exists v \exists w S(\underline{y}, v, w) \right) \right) \right]
\end{aligned}$$

$$\begin{aligned}
p(x, z) &\longleftarrow r(x, y, z'), z' \neq z, r(x, y'z, z) \\
t(x) &\longleftarrow r(x, y, z'), \neg q(y) \\
q(y) &\longleftarrow s(y, v, w) \\
andswer(z) &\longleftarrow r(x, y, z), \neg p(x, z), \neg t(x)
\end{aligned}$$

3.2 First query

FO Rewriting:

```
certainty(Z):-r1(X,Y,Z),not p0(X,Z),not p1(x).
p0(X,Z):-r1(X,Y,Z1),r1(X,_,Z),not Z=Z1.
p1(X):-r1(X,Y,Z1),not p2(Y).
p2(Y):-r2(Y,V,W).

certainty(Z) :- not certainty(Z), r1(_,_,Z).

#show certainty/1.
```

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).
```

3.3 Second query

FO Rewriting:

```
certainty(W,Z):-r1(X,Y,Z),not p0(Z,X),not q0(W,X), r2(P,Q,W).
p0(Z,X):-r1(X,Y,Z1),not Z1=Z,r1(X,_,Z).
q0(W,X):-r1(X,Y,Z1),not q1(W,Y), r2(P,Q,W).
q1(W,Y):-r2(Y,V,W),not q2(W,Y).
q2(W,Y):-r2(Y,V,W1),not W1=W,r2(Y,_,W).

#show certainty/2.
```

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).
```

3.4 Third query

FO Rewriting:

```
%%% The following rule is optional.
cqa(Z) :- not cqa(Z), r1(_,_,Z).
%%%
cqa(Z) :- r1(X,_,Z), not p_0(X,Z), not p_1(X).
p_0(X,Z) :- r1(X,_,W), W != Z, r1(_,_,Z).
p_1(X) :- r1(X,Y,_), not q_3(Y).
q_3(Y) :- r3(Y,_), not q_2(Y).
q_2(Y) :- r3(Y,V), not q_1(V).
q_1(V) :- r2(V,_,_).

#show cqa/1.
```

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).
```

3.5 Fourth query

FO Rewriting:

```
%%% The following rule is optional.
cqa(Z,D) :- not cqa(Z,D), r1(_,_,Z), r2(_,_,D).
%%%
cqa(Z,D) :- r1(X,_,Z), not p_0(X,Z), not p_1(X), r2(V,_,D).
p_0(X,Z) :- r1(X,_,W), W != Z, r1(_,_,Z).
p_1(X) :- r1(X,Y,_), not q_3(Y).
q_3(Y) :- r3(Y,_), not q_2(Y).
q_2(Y) :- r3(Y,V), not q_1(V).
q_1(V) :- r2(V,_,_), not q_0(V).
q_0(V) :- r2(V,_,W), W != D, r2(_,_,D).

#show cqa/2.
```

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), rr2(V,U,D), rr3(Y,V).

#show certainty/2.
```

3.6 Fifth query

FO Rewriting:

```
%q5(z): r1(X,Y,Z), r4(Y,V,W)

p(X,Z) :- r1(X,Y,Z2), Z2!=Z, r1(X,Y2,Z).
t(X) :- r1(X,Y,Z), not q(Y).
q(Y) :- r4(Y,V,W).
answer(Z) :- r1(X,Y,Z), not p(X,Z), not t(X).

#show answer/1.
```

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,Y,_).

:- rr1(X,Y,Z), rr4(Y,V,W).
```

3.7 Sixth query

FO Rewriting:

```
certainty(Z):-not certainty(Z),r1(_,_,Z).
certainty(Z):-r1(_,_,Z),q1(X1,Z).

q1(X1,Z):-r2(X1,Y,W),not q2(X1,Z,X,D),r5(X,Y,D),r1(X,Y,Z).

q2(X1,Z,X,D):-r2(X1,Y,W),q4(X,Y,Z,D),r5(X,Y,D),r1(X,Y,Z).
q2(X1,Z,X,D):-r2(X1,Y,W),not q3(Y,Z,W),r5(X,Y,D),r1(X,Y,Z).

q3(Y,Z,W):-r5(X,Y,D),r1(X,Y,Z),r2(_,Y,W).

q4(X,Y,Z,D):-r5(X,Y1,D1),r1(X,Y2,Z1),not Z=Z1,r1(X,Y,Z),r5(X,Y,D).
q4(X,Y,Z,D):-r5(X,Y1,D1),r1(X,Y2,Z1),not Y=Y1,r1(X,Y,Z),r5(X,Y,D).
q4(X,Y,Z,D):-r5(X,Y1,D1),r1(X,Y2,Z1),not Y=Y2,r1(X,Y,Z),r5(X,Y,D).
q4(X,Y,Z,D):-r5(X,Y1,D1),r1(X,Y2,Z1),not D=D1,r1(X,Y,Z),r5(X,Y,D).

#show certainty/1.
```

Generate and Test:

3.8 Seventh query

FO Rewriting:

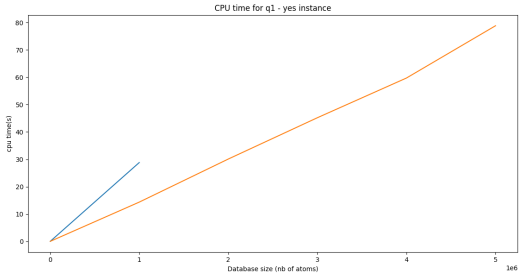
```
certainty(Z) :- not d1(Z,Y), r2(Y,X,W), r1(X,Y,Z).
d1(Z,Y) :- not d2(Z,Y,X,W), r2(Y,X,W), r1(X,Y,Z).
d2(Z,Y,X,W) :- not d3(Z,Y,X,W), r2(Y,X,W), r1(X,Y,Z).
d3(Z,Y,X,W) :- r2(Y,X,W), not d4(Z,Y,X,W,P,Q), r1(X,P,Q), r1(X,Y,Z).
d4(Z,Y,X,W,P,Q) :- r1(X,P,Q), P=Y, r2(Y,X,W), Q=Z, d5(Z,Y,X,W).
d5(Z,Y,X,W) :- r5(X,Y,D), not d6(Z,Y,X,W), r2(Y,X,W), r1(X,Y,Z).
d6(Z,Y,X,W) :- not d7(Z,Y,X,W,P,D), r2(Y,X,W), r5(X,P,D), r1(X,Y,Z).
d7(Z,Y,X,W,P,D) :- r2(Y,X,W), r5(X,Z_5_0,D), r1(X,Y,Z), P=Y.

#show certainty/1.
```

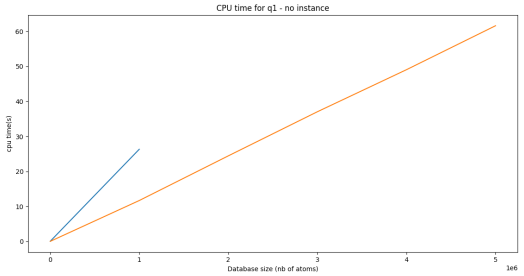
Generate and Test:

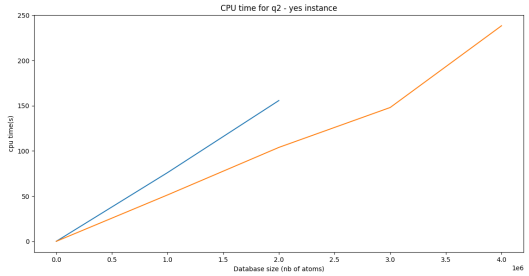
4 RESULTS

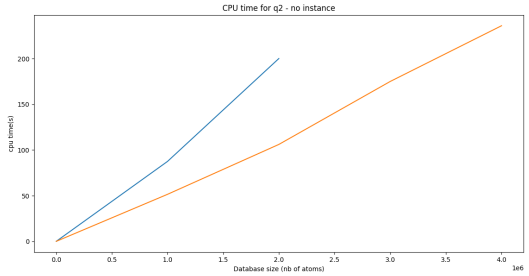
For each graph, the blue line corresponds to the time taken by the generate-and-test method and the orange line to the time taken by the FO method. When a result is not in the graph, that means that the execution of the program was interrupted for insufficient memory. For example, the times for the generate-and-test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

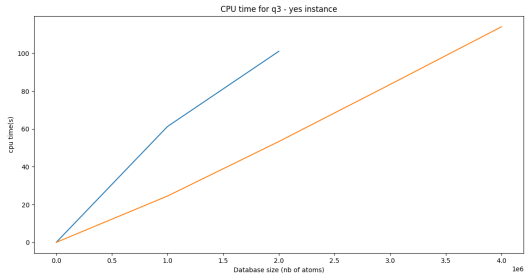
The databases used here were generated specially for the tests through a python script, and have 20 % of inconsistency. The inconsistency is calculated by dividing the number of relations without any primary key violation by the number of relations with at least one primary key violation.

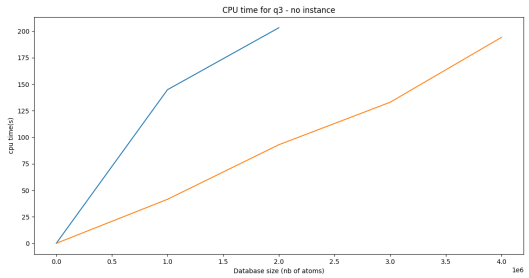












We see that the fo rewriting leads to better results, in terms of cpu time, that the generate-and-test method.

5 CONCLUSION

We rewrited 7 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. Our experiments show that the first order rewriting is always faster than the generate in test method wheter the inconsistency is low or high and whether the database is small or large. We would advice to rewrite any of your queries in first order if they are rewritable in first order.