Orik: Explaining the Universe One Negation at a Time

Kathryn Dahlgren
kmdahlgr@ucsc.edu

Peter Alvaro
palvaro@ucsc.edu

Last Updated : October 23, 2017

Abstract

Distributed application correctness depends upon events happening and not happening during an execution. Many techniques exist for describing why events occur, but comparatively few methods attempt to justify why events DO NOT occur. This work explores the latter arena by harnessing established data provenance and query rewriting concepts and strategies to present a novel approach to describing the provenance of non-events within the context of a declaratively defined modeling environment. Lineage-Driven Fault Injection (LDFI), a framework for characterizing, challenging, and explaining the correctness of distributed protocols, serves as a platform for analyzing the practical utility of the technique.

1 Introduction

//intro

1.1 Background

//

1.1.1 Provenance Rewrites

//

2 Methods

The program rewriting approach utilized in Orik to describe the provenance of negated subgoals relies upon the direct application of DeMorgan's Law upon Datalog rules indirectly modeled as Boolean Formulas in Disjunctive Normal Form (DNF). The crux of the approach is mapping the DNF formula resulting from the negation and subsequent simplification to a set of corresponding Datalog rules capturing the complete space of possible explanations for why data do not appear in particular relations over the course of a program execution. Practicality necessitates the incorporation of additional logic to reduce the space of explanations into a manageable set useful within the context of the particular target Datalog program.

At a high level, limiting the theoretically infinite range of explanations for negative derived facts depends upon the integration of value constraints upon the attributes of negated subgoals informed from previously derived program evaluation results. Because applying DeMorgan's law upon an arbitrary formula may translate into the negation of additional intensional databases (IDBs), the process repeats across a series of successive rewriting and program evaluation cycles until every rule in the final version of the rewritten program contains only positive IDB subgoals. End state convergence is guaranteed by the finite nature of practical and safe Datalog programs. The rewriting method, applied in conjunction with the existing provenance rewriting method core to the original LDFI approach [1], results in a final Datalog program promising evaluation results possessing all information necessary to describe the provenance of all positive and negative subgoals constituting in the original program.

Text subsequent to a detailed description of the DeMorgan's rewriting approach illustrates the ideas within the context of two examples. The Message Omission example depicts a relatively simple Datalog program and the corresponding version after the application of the DeMorgan's rewrite process. The Simplog Delivery Protocol represents a more involved example illustrating the process of handling recursively defined IDB rules.

2.1 Algorithm

The Negative Rewrite algorithm exerts a series of transformations upon an arbitrary input Datalog program such that the final output Datalog program includes an additional set of rules capturing the complete set of explanations for why particular data do not appear within specific relations in the context of program evaluations. Furthermore, the space of explanations captured by the additional rules is bounded by relevance to the context of the original Datalog program by the introduction of additional extensional databases (EDBs) defined using the grounded results of previous evaluations (see Section 2.2). Algorithm 1 details the steps in pseudocode.

Performing a DeMorgan's Rewrite upon some input Datalog program follows an iterative succession of program evaluation and program rewrite phases until no rules predicate on negated IDBs. After obtaining the evaluation results of the input program, the algorithm examines every rule for instances of negated IDB subgoals. Upon finding such a subgoal n, the algorithm collects all rules defining n and transforms the rules into a corresponding Boolean formula in DNF after ensuring the goal attributes across rule definitions conform to a uniform schema. The transformation associates every subgoal, unique across the subset of program rules, as a unique literal in the formula. The list of subgoals per rule correspond to a single clause of conjuncted literals in the Boolean formula. The final formula manifests from the disjunction of all conjuncted clauses derived from individual n rules.

The enforcement of a uniform schema allows the presence of semantically identical literals in the final Boolean formula. Such a circumstance affords potential formula simplification opportunities. The optimization is accordingly far more appealing than the alternative because not enforcing a uniform goal attribute schema requires regarding all subgoals across the subset of n IDB rules per negated subgoal N as unique literals. Such a perspective consequently provides no additional formula simplification opportunities.

All rules containing only positive subgoals are copied directly into the output program. Additionally, all EDB facts declared in the input program are copied directly into the output program.

Intuitively, the series of rewrites triggers the generation of a series of versions of the original Datalog program such that each new version increases the granularity of explanations for data not appearing in particular relations. The final version allows the articulation of explanations purely in terms of EDB facts. Accordingly, rendering the presence of negative EDBs (absence of positive EDB facts) cited in explanations in the original program will cause absent data to appear in targeted relations negated as subgoals in particular rules.

Algorithm 1 NegativeRewrite

Input: a Datalog program P ;
Output: a rewritten Datalog program P' ;

```
1:  Let E store results of evaluating P.
2:  Let P' be an empty Datalog program.
3:  for each rule r in P do
4:      if r contains no negated IDB subgoal n then
5:          Add r directly to P'.
6:      else
7:          Rewrite all n rules to expose a uniform set of goal variables, denoted U.
8:          Regard the set of positive n rules as a single Boolean Formula in DNF, denoted F.
9:          Let not_F be the Boolean Formula obtained by simplifying the negation of F into DNF.
10:         for each clause in not_F do
11:             Add a positive not_n(U) rule to program P'.
12:         end for
13:         for each new not_n rule do
14:             Add an EDB subgoal dom_not_n defining the domain for each attribute in the goal attribute list U.
15:         end for
16:         Substitute the negated n in r with not_n to create the new rule r'.
```

Add r' to P'.
for each rule q in P **do**
Replace any instance of the negated subgoal n in q with not_n to create the new rule q'.
Add q' to P'.
end for
end if
end for
Add all EDB facts from P to P'.
if the rules P' contain negated IDBs **then**
Call $DeMorgansRewrite(P')$.
else
return P'
end if

2.2 Defining Domain Subgoals

The derivation of a Negative Rewrite for a subgoal n negated in some rule r results in a set of additional rules encoding the logical complement n. However, relation complements inspire interesting philosophical questions about the definition of the universe under consideration. Under a very liberal definition, the complement of a relation is as infinite as the space of strings representable in a computer. Such a universe is obviously intractable. A more practical definition would characterize relation complements relative to the active domain of the database under consideration. Unfortunately, such an approach is also intractable when considering problems of non-trivial scale .

Accordingly, rendering the Negative Rewrites approach tractable requires a more intelligent means of imposing of bounds upon relation complements. Orik utilizes a technique for constructing minimal boundaries for $not\_$ rules using evaluation results from the calling and positive rules associated with a particular negative subgoal. //.

2.3 Example 1 : Message Omission

//

2.4 Example 2 : Simplog Delivery Protocol

//

3 Application

//intro

3.1 Lineage-Driven Fault Injection

//

3.2 Dedalus

//

4 Evaluation

//intro

4.1 Complexity Analysis

//program rewrite complexity
//provenance graph complexity
//boolean formula complexity
//time complexity

4.2 Runtime Analysis

//comparison with negative provenance support in Molly.

5 Related Work

//

6 Future Work

//

7 Conclusion

//

References

[1] ALVARO, P., ROSEN, J., AND HELLERSTEIN, J. M. Lineage-driven fault injection. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2015), SIGMOD '15, ACM, pp. 331–346.