CS 517 Project

June 11, 2020

Parisa S. Ataei

1 Introduction

Program equivalence is generally an undecidable problem, however, arguably, it is easier to solve compared to program correctness although the two have lots of similarities that one can exploit to solve one using the other. While for program correctness one needs to explicitly define the formal semantics of a language, equivalency rules can be defined for a language to search the space for a given program to find its equivalence programs. However, for such an expensive search to be conclusive, the set of reduction rules must be terminating and confluent.

I designed the *Variational Relational Algebra (VRA)* for my research which combines Formula Choice Calculus [2] and relational algebra. In this project, I reduce a program written in VRA to a propositional formula and run a SAT solver on it to see if there exists an equivalence program w.r.t. the variational point. The simplified syntax of VRA is given in Section 2, the problem formalization is given in Section 3, and the reduction of program equivalence to a propositional formula is provided in Section 4. Finally, Section 5 elaborates on the implementation and Section 6 discusses the assumptions made to allow a reasonable reduction of AST to SAT formulas and how different features to the language break the reduction. Additionally, I discuss why I only consider the structural equivalence of programs w.r.t. the variational point instead of a more general equivalency.

2 Background

Figure 1 defines the syntax of the simplified VRA used in this project. Feature expressions are propositional formulas of the set of features defined for a database. Assume we have a database with features f_1 and f_2 and two tables $r_1(a_1, a_2)$ and $r(a_1, a_3)$. For simplicity, I assume the database in use is a traditional database and not a variational database. r_1 refers to the relation r_1 in the database and returns the table as stored in the database. Projection $\pi_A q$ projects a set of attributes A from the subquery q. For example, the query $\pi_{a_1} r_1$ projects the attribute a_1 from the relation r_1 stored in the database. As an example of variational query $q_1 = \pi_{f_1\langle a_1, a_2\rangle} r_1$ projects the attribute a_1 when f_1 evaluates to `true` and attribute a_2 otherwise. Selection filters tuples returned from a subquery based on a condition where the condition can also be variational. Cross product just provides the cross product of the two given tables. A choice of queries allows one to write queries variationally on the top level. For example, one can write q_1 as $f_1\langle \pi_{a_1} r_1, \pi_{a_2} r_1\rangle$. Note that a choice cannot necessarily be pushed in or out. For example, the query $f_1\langle r_1, r_2\rangle$ cannot be simplified. Finally, the empty relation ε is introduced as a convenient for users to indicate that an alternative of a choice does not do anything while the other alternative is only valid under the condition provided by the dimension of the choice (in the choice $e\langle q_1, q_2\rangle$ the feature expression e is called the dimension of the choice). Throughout this report a *variational point* refers to where a choice has been used in query, this could be at the query level, the attribute level, or the condition level.

3 Problem Definition

The problem I will be attacking is: *Given a type-correct program written in VRA is there exist a smaller equivalent program?* A program is type correct if it passes the type system of VRA [1], i.e., the query conforms to the underlying schema of the database. The size of a program is the depth of its abstract syntax tree. I only consider syntactical equivalence and not semantical. For example, assume we have the relation $r(a_1, a_2)$. The query r and $\pi_{a_1,a_2} r$ are semantically equivalent since both are projecting the same set of attributes from the given relation, however, the former does this implicitly. This problem (and possible solutions to it) could be use to address a harder problem: *finding the global minimal program written in VRA*, which is similar

$$
\begin{array}{rcll}
e \in \mathbf{E} & ::= & \texttt{true} \mid \texttt{false} \mid f \mid \neg f \mid e \wedge e \mid e \vee e & \\
\theta \in \mathbf{\Theta} & ::= & \texttt{true} \mid \texttt{false} \mid a \bullet k \mid a \bullet a \mid \neg\theta \mid \theta \vee \theta & \\
 & \mid & \theta \wedge \theta \mid e\langle\theta, \theta\rangle & \\
A \in \mathbf{A} & ::= & a \mid e\langle a, a\rangle \mid a, A \mid \varepsilon \mid e\langle a, a\rangle, A & \\
q \in \mathbf{Q} & ::= & r & \textit{Relation reference} \\
 & \mid & \pi_A q & \textit{Projection} \\
 & \mid & \sigma_\theta q & \textit{Selection} \\
 & \mid & q \times q & \textit{Cross product} \\
 & \mid & e\langle q, q\rangle & \textit{Choice} \\
 & \mid & \varepsilon & \textit{Empty relation}
\end{array}
$$

Figure 1: Syntax of variational relational algebra, where $\bullet$ ranges over comparison operators ($<, \leq, =, \neq, >, \geq$), k over constant values, a over attribute names, and A over lists of variational attributes. The syntactic category e represents feature expressions, θ is variational conditions, and q is variational queries.

to the minimization of binary decision diagrams [3].

4 Reduction to SAT

The overall algorithm works as follows:

on input q (assuming q is type-correct):
generate the corresponding SAT formulas for q: Π_q, Σ_q, Ψ_q
for every subquery q_i of q
    restructure q' to generate a new type-correct subquery q_i' and
        replace q_i' with q_i in q resulting in the new query q'
    generate the corresponding SAT formulas for q': $\Pi_{q'}, \Sigma_{q'}, \Psi_{q'}$
    check if $|q'| < |q|$ and $taut\,(\Pi_q \leftrightarrow \Pi_{q'})$ and $taut\,(\Sigma_q \leftrightarrow \Sigma_{q'})$ and $taut\,(\Psi_q \leftrightarrow \Psi_{q'})$
        then say yes
    otherwise choose another subquery q_j of q and do the above
if for all possible restructuring of subqueries of q no smaller equivalent program
    could be found say no

A subquery (subprogram) is a subtree of the AST of the given query (program). Thus, restructuring a subquery is practically restructuring a program. However, not every restructuring is acceptable since it may generate type-ill queries. Thus, I use a set of rules to ensure that the new query is not ill-typed [1]

Three SAT formula is generated for a given query q:

- The formula Π_q takes the attributes of a query that uses the projection operator and assigns a boolean variable to each attribute with the same name as the attribute (this is under the assumption that attribute names are unique). It encodes a choice of attributes $e\langle a_1, a_2\rangle$ to $(e \wedge a_1) \vee (\neg e \wedge a_2)$ and encodes a list of attributes a_1, a_2 to $a_1 a_2$. For example, for the query $q_1 = \pi_{a_1, e\langle a_2, a_3\rangle} r$ we have the formula: $\Pi = a_1 \wedge ((e \wedge a_2) \vee (\neg e \wedge a_3))$.

- The formula Σ_q takes the conditions of a query that uses the selection operator and assigns a variable name to a new atomic condition and keeps these assignments in an environment. An atomic condition has the form $a \bullet k$ or $a \bullet a$. Choices of conditions are encoded similar to choices of attributes and the conjunction and disjunction between conditions are kept as is. For example, for the query $q_2 = \sigma_{e\langle a_1 > 100, a_2 = a_3 \vee a_4 < 100\rangle} r$ we have the formula: $\Sigma = (e \wedge c_1) \vee (\neg e \wedge (c_2 \vee c_3))$, where c_1 associates to condition $a_1 > 100$ and c_2 and c_3 associate to conditions $a_2 = a_3$ and $a_4 < 100$, respectively.

- The formula Ψ_q encodes the relation of subqueries within a query by assigning a variable to relations with the same name as the relation and generating new variables for each subquery. For example, for q_1 we have $\Psi_{q_1} = r$. For the query $q_3 = r_1 \times r_2$ we have $\Psi_{q_3} = r_1 \wedge r_2$. For a choice of relations $q_4 = e\langle r_1, r_2\rangle$ we have $\Psi_{q_4} = (e \wedge r_1) \vee (\neg e \wedge r_2)$. For the query ε we have $\Psi = \texttt{true}$. For the query $\pi_{a_1}(\sigma_{a_2 = a_3} r)$, we have: $\Psi = r \wedge p_1$ where p_1 associates to the subquery $\sigma_{a_2 = a_3} r$. *Concern:* assigning a variable to a new subquery and saving it in an environment is problematic since when it comes to searching the environment to see if a subquery is already assigned a variable we need to have a notion of equivalent for queries which is partly the point of constructing these

[1] I'm not sure if doing so violates the guessing part since it is following a set of rules. However, I couldn't think of a completely random approach of generating new queries.

formulas. At the moment, such notion of equivalency is only based on the structure of a query. This was not a problem for the other two generated formulas (Π and Σ) because we have a notion of semantic equivalence for attributes and conditions (and their environment can be searched correctly) and we are generating their corresponding formulas to help check the equivalency of the queries and not themselves (attributes and conditions). I appreciate any suggestion you may have for this problem.

Finally, for checking the equivalency of two programs q_1 and q_2 we can check [2]: $taut\,(\Pi_q \leftrightarrow \Pi_{q'})$ and $taut\,(\Sigma_q \leftrightarrow \Sigma_{q'})$ and $taut\,(\Psi_q \leftrightarrow \Psi_{q'})$. The tautology can be checked by negating the unsatisfiability of a formula.

5 Implementation

The project is implemented in Haskell and uses the SBV library for solving the SAT problems. The project can be find in this GitHub repo. This project turned out to be more of an exploratory project and the implementation does not completely address the problem until the research questions mentioned in Section 4 are addressed.

6 Future Work

Disclaimer [3]*:* After your feedback, it took me a day (and some discussion with a colleague) to realize that I should change my entire way of thinking and move away from trying to find equivalence rules (how I am trained to think about these kinds of problems as a PL researcher) and converting them to SAT problems. Your comment of thinking about it in terms of guessing and checking really helped, however, I was really stressed out that I have to change my entire project based on your feedback and I realized guessing an equivalent program and checking it with SAT solver might not have been as straight forward as I thought. I was planning on doing some experiments for the discussion part of the project on Wednesday and Thursday but unfortunately I got food poisoning and could only manage to write this report. Still,

[2] Idea of checking equivalency like this is taken from here.

[3] I hate that I am writing this disclaimer but it is Thursday and I just want to give you a sense of my stress this week

this is a research project that I am interested in (also my research group found this problem rather interesting and beneficial) and I'm planning on continuing to work on it when I find the time, thus, instead of discussion I provide how I want to proceed this problem in future. I hope you consider the current and my personal situation while reviewing and grading the project and thanks for allowing us to define our own projects (although I'm not happy with how I'm turning in this project it's a great side project for future):

- Improve the subtree restructuring of the guessed equivalent program based on some rules.
- Address the concern mentioned in Section 4.
- Generate random type-correct queries (programs) written in VRA for a given database schema. This would also be a side project that would help my own research!
- Analyze the approach in terms of the size of the input program, how long it takes to find a possible smaller equivalent program, instances that it fails, etc. Explore if this is a reasonable and performant approach to address finding the global minimum of a program written in VRA.

References

[1] Parisa Ataei, Qiaoran Li, Eric Walkingshaw, and Arash Termehchy. Managing variability in relational databases by vdbms, 2019.

[2] Spencer Hubbard and Eric Walkingshaw. Formula Choice Calculus. In *Int. Work. on Feature-Oriented Software Development (FOSD)*, pages 49–57. ACM, 2016.

[3] A. L. Oliveira, L. P. Carloni, T. Villa, and A. L. Sangiovanni-Vincentelli. Exact minimization of binary decision diagrams using implicit techniques. *IEEE Transactions on Computers*, 47(11):1282–1296, 1998.