# CS 517 Project

June 8, 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 $\varepsilon$ 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: *Assuming that a given a program written in VRA is type correct and is of canonical form, is there exist an equivalent program with less variational points?* 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 canonical form of program represent a class of programs that are semantically equivalent based on equivalency rules of relational algebra and the choices are pushed down into the query as much as possible. A program in the canonical form follows the following order for its operators, if they exist: projection, selection, cross-product; where choices could appear at attribute level, condition level, and query level. [I will provide examples

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

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, $\theta$ is variational conditions, and $q$ is variational queries.

of this for the final report to help the reader understand the concept better.] This problem is similar to the minimization of binary decision diagrams [3] since it determines if a program can be written with less variational points and thus is NP-hard.

# 4 Reduction to SAT

I generate multiple SAT formulas to check: 1) if variations used in the query are valid (reasonable), e.g., the variation in $\texttt{false}\langle q_1, q_2\rangle$ is not reasonable although the query is type correct, elaborated in Section 4.1, 2) if dead alternative branches exist in a choice, e.g., the alternative branch $q_2$ will never be executed in the query $f_1\langle f_1 \vee f_2\langle q_1, q_2\rangle q_3\rangle$, elaborated in Section 4.2, and 3) if a choice has redundancy (this relies on having a structurally equivalence relation for the pure relational queries), e.g., the $q_1$ subquery is redundant in the query $e_1\langle q_1, e_2\langle q_1, q_2\rangle\rangle$, elaborated in Section 4.3. Since the VRA is inductive I generate these SAT problems in a bottom-up approach while keeping and updating an environment of variables introduced for the formulas.

I generate fresh variables as follows as the first step for generating each

SAT formula:

- Attribute names are denoted by variables $a_1, a_2, \cdots$. Note that the empty attribute is associated with `true`.
- Conditions are denoted by variables $\theta_1, \theta_2, \cdots$ except for the variational conditions. Note that it is easy to define semantically equivalence on pure relational conditions, thus, there is no need to break down the conditions except for variational ones. Also, especially since we are not concern with semantical equivalence in this project.
- Relations are denoted by variables $r_1, r_2, \cdots$. And the empty relation $\varepsilon$ is associated with `true`.
- The same feature names are used as their variables in the generated formula.
- The subqueries are denoted by variables $q_1, q_2, \cdots$ and are generated bottom-up.

## 4.1 Generating Validity SAT Formulas

The validity propositional formula for a given query is generated as follows:

- For empty relation $\varepsilon$ generate `true`.
- For each relation $r$ generate the clause $\texttt{true} \wedge r$.
- For each attribute $a$ generate the clause $\texttt{true} \wedge a$.
- For the set of attributes $a_1, \cdots, a_n$ generate the formula $(\texttt{true} \wedge a_1) \wedge \cdots \wedge (\texttt{true} \wedge a_n)$.
- For each condition $\theta$ generate the clause $\texttt{true} \wedge \theta$.
- For a variational attribute $e\langle a_1, a_2\rangle$ generate the clause $(e \wedge (\texttt{true} \wedge a_1)) \vee (\neg e \wedge (\texttt{true} \wedge a_2))$.
- For the query $\pi_A q$ generate the formula $\mathit{valid}\,(A) \wedge \mathit{valid}\,(q)$ where $\mathit{valid}\,(A)$ is the validity formula generated for the set of attributes and $\mathit{valid}\,(q)$ is the validity formula generated for the subquery $q$, both based on the description provided above.

- For the query $\sigma_\theta q$ generate the formula $\mathit{valid}\,(\theta) \wedge \mathit{valid}\,(q)$ where $\mathit{valid}\,(\theta)$ is the validity formula generated for the condition $\theta$ based on the description provided.
- For the query $e\langle q_1, q_2\rangle$ generate the formula $(e \wedge \mathit{valid}\,(q_1)) \vee (\neg e \wedge \mathit{valid}\,(q_2))$.

If the validity formula generated for a query is not satisfiable then the query contains an invalid (unreasonable) variation and can be rewritten to omit such variation.

## 4.2 Generating Dead-Branch SAT Formulas

The formulas to determine if a query has dead branches are generated as follows:

- For every choice $x = e\langle x_1, x_2\rangle$ in a given query where $x$ is a meta-variable that ranges over syntactic categories of conditions, attributes, and queries generate two formulas $\mathit{dead}_l\,(x) = e \to e_l$ and $\mathit{dead}_r\,(x) = \neg e \to e_r$ where $e_l$ is the dimension of the choice within the left branch, i.e., $x_1$ and $e_r$ is the dimension of the choice within the right branch, i.e., $x_2$. And $l \to r$ is implication which can be substituted by $\neg l \vee r$.

Intrinsically, $\mathit{dead}_l\,(.)$ and $\mathit{dead}_r\,(.)$ determine if the dimension of a nested choice is more general that its outside choice which cause some branches to never be executed. Note that $\mathit{dead}_l\,(.)$ and $\mathit{dead}_r\,(.)$ are generated for each syntactic category separately. And they solely focus on the interaction of feature expressions. Thus, if $\mathit{sat}\,(\mathit{dead}_l\,(q))$ or $\mathit{sat}\,(\mathit{dead}_r\,(q))$, then the query $q$ contains some dead alternative branches and can be simplified.

## 4.3 Generating Redundant-Branch SAT Formulas

To determine if a query has redundant branches formulas may be generated for each nested choice of a specific syntactic category, assuming that we have a structurally equivalence relationship ($\equiv$) over pure relational syntactic category of $x$, as follows:

- If the choice has the format $e_1\langle x_1, e_2\langle x_2, x_3\rangle\rangle$ where $x_1$ does not have a top level choice we have:

  - If $x_1 \equiv x_2$ then they both are assigned the same variable name, say $x'$, then generate the formula
$$((e_1 \land x') \lor (\neg e_1 \land e_2 \land x')) \leftrightarrow (e_1 \lor e_2) \land x' \qquad .$$
  - If $x_1 \equiv x_3$ then they both are assigned the same variable name, say $x'$, then generate the formula
$$((e_1 \land x') \lor (\neg e_1 \land \neg e_2 \land x')) \leftrightarrow (e_1 \lor \neg e_2) \land x' \qquad .$$

- If the choice has the format $e_1\langle e_2\langle x_1, x_2\rangle, x_3\rangle$where $x_3$ does not have a top level choice we have:
  - If $x_2 \equiv x_3$ then they both are assigned the same variable name, say $x'$, then generate the formula
$$((\neg e_1 \land x') \lor (e_1 \land \neg e_2 \land x')) \leftrightarrow (e_1 \land e_2) \land x' \qquad .$$
  - If $x_1 \equiv x_3$ then they both are assigned the same variable name, say $x'$, then generate the formula
$$((\neg e_1 \land x') \lor (e_1 \land e_2 \land x')) \leftrightarrow (e_1 \land \neg e_2) \land x' \qquad .$$

- If the choice has the format $e_1\langle e_2\langle x_1, x_2\rangle, e_3\langle x_3, x_4\rangle\rangle$ we have:
  - If $e_2 \equiv e_3$ and $x_2 \equiv x_4$ (thus $x_2$ and $x_4$ are assigned the same variable $x'$) generate the following formula:
$$((e_1 \land \neg e_2 \land x') \lor (\neg e_1 \land \neg e_3 \land x')) \leftrightarrow \neg e_2 \land x'$$
  - If $e_2 \equiv e_3$ and $x_1 \equiv x_3$ (thus $x_1$ and $x_3$ are assigned the same variable $x'$) generate the following formula:
$$((e_1 \land e_2 \land x') \lor (\neg e_1 \land e_3 \land x')) \leftrightarrow e_2 \land x'$$

*rdnt* $(q)$denotes all the formulas generated as described above for a given query $q$. If at least one of these queries is a tautology then we can conclude that there exists a redundant branch in the query.

# 5 Implementation

I am currently implementing the project in Haskell and I use the SBV library for solving the SAT problems. The GitHub repo will be provided with the final submission.

# 6 Discussion

Made the mistake of reading the news and lost track of time! However, the following contains my initial thoughts on the project. I will provide a comprehensive discussion for the final report.

After thinking about this reduction for a week, I do not think that, generally speaking, an AST can be linearized to the extent that it is converted to a single propositional formula to determine variation minimization or even worse structural or semantical equivalence. Thus, after narrowing down the problem definition, I began thinking about different types of minimizations such as redundant branches and dead branches. I also believe I was able to do so because of the nature of choice calculus and that it uses propositional formulas within itself. At the same time, I am skeptical that the reductions I have provided cover all the cases and I am interested to know if there is a way to prove or deny that this reduction can certainly determine if a variational query can be minimized or not. I did not simplified the query language that much and I believe adding operations such as union and intersection would not break down the reduction. However, an important difference of the language that I provided here with the original VRA is that variation appears in the language used in this project in only one manner as opposed to original VRA that provides tagging elements with feature expression for a better usability. Such simplification does not reduce the expressiveness of the language, however, it made it syntactically more consistent which helped with the reduction. Purposeful language design does wonders!

ADD FOR FINAL SUBMISSION: -renaming in lang design!! this would have made abt in stead of ast and we would have to keep track of the bindings.

# 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.