Decidability and undecidability of equivalence for linear Datalog, with applications to normal-form optimizations

Tomás Feder\textsuperscript{1} and Yatin Saraiya\textsuperscript{2}

\textsuperscript{1} IBM Almaden Research Laboratory
\textsuperscript{2} Bellcore

\textbf{Abstract.} A variety of optimizations in logic databases \cite{20} involve the transformation of safe, function-free (Datalog) logic programs to simpler, more efficiently evaluable programs. The essence of these transformations is the detection of program equivalence. Equivalence has been shown to be undecidable in \cite{1, 17, 18, 19}, but those results are unsatisfactory in that they involve highly nonlinear rules, or an unbounded number of linear recursive rules, and hence yield no insight into the behaviour of small, linear recursive programs. In this paper, we consider programs that consist of one nonrecursive linear rule, one linear recursive rule and one initialization rule. We provide a tight characterization of the decidability of program equivalence, proving it undecidable for such programs, and proving it decidable for a subclass that includes binary chain programs and programs with no repeated predicates in the rule body. We then apply these results to various normal-form optimizations \cite{12, 18}, including the detection of serializability, commutativity, basis-linearizability and rule redundancy.

1 Introduction

A variety of optimizations in logic databases \cite{20} involve the transformation of safe, function-free (Datalog) logic programs to simpler, more efficiently evaluable programs. Many such transformations involve the detection of normal forms for the derivations of the program \cite{12, 18}.

\textbf{Example 1.} Consider the following Datalog program $\mathcal{P}$.

$r_1 : p(X, Y) :- p(X, u^1), p(u^1, Y)$.
$r_2 : p(X, Y) :- p(Y, X)$.
$b_1 : p(X, Y) :- e(X, Y)$.

This program computes, for $p$, the symmetric, transitive closure of the relation for the binary predicate $e$.

It is easy to see that the symmetric, transitive closure can also be computed in a bottom-up fashion by first initializing $p$ using the basis rule $b_1$; then computing the symmetric closure by closing under the recursive rule $r_2$; and then taking the transitive closure of the result using rule $r_1$. In this case, we say that $r_2$ is \emph{serializable under} $r_1$. Now, we may use the fact that any proof-tree consisting only of
applications of the rule \( r_4 \) may be transformed into a *right-linear* proof tree (i.e., a proof-tree in which only the rightmost recursive subgoal is expanded using the rule \( r_4 \)) to rewrite \( \mathcal{P} \) as the following program. The correctness of the translation is proved in [18]. In this new program, the new intensional predicate \( q \) represents the symmetric closure of \( e \).

\[
\begin{align*}
\mathcal{P}_1 &: p(X,Y) \leftarrow q(X,u^1), p(u^1, Y). \\
\mathcal{P}_2 &: p(X,Y) \leftarrow q(X,Y). \\
\mathcal{P}_3 &: q(X,Y) \leftarrow q(Y,X). \\
\mathcal{P}_4 &: q(X,Y) \leftarrow e(X,Y). \\
\end{align*}
\]

The key to the equivalence is the property that for any instantiation of \( e \), the facts obtained by \( \mathcal{P} \) are exactly those facts obtained by those proof-trees of \( \mathcal{P} \) that have the form shown in Figure 1(b), where (every instantiation of) the tree \( T \) consists only of applications of the rules \( r_2 \) and \( b_1 \) as shown in part (a) of the figure. The form of these proof-trees is said to be a *normal form* for \( \mathcal{P} \). The detection of such normal forms has the effect of reducing redundant derivations in the computation of the program. The transformed program may be further transformed into a linear recursive program (see [18]), for which special-purpose query evaluators become available. For a description of other benefits of normal-form recognition, see [18].

![Diagram](image)

**Fig. 1.** Normal form for the program \( \mathcal{P} \) of Example 1.

The essence of these transformations is the detection of program equivalence, which has been proved undecidable in restricted cases ([1, 17, 18, 19]). These proofs are unsatisfactory in that they involve highly nonlinear rules, or an unbounded number