The Complexity of Reusing and Modifying Rulebases*

Anthony J. Bonner

University of Toronto
Department of Computer Science
Toronto, Ontario M5S 1A4
Canada

bonner@db.toronto.edu

Abstract

This paper develops a method for reusing and modifying deductive databases. Such methods are needed when new rulebased applications differ only slightly from existing ones or when an application is to be incrementally updated. In order to facilitate reuse, we extend deductive databases by the concept of *predicate substitution*. In this way, during query evaluation, not only variables, but also predicates can be substituted. This paper continues our earlier work on predicate substitution in two directions: (i) We extend the concept to a wider class of rulebase modifications, and (ii) we establish tight bounds on the data complexity of Datalog augmented with substitution, showing it to be EXPTIME-complete. Predicate substitution thus increases the power of Datalog to express database queries. The paper presents a proof theory and model theory for the language, including a fixpoint semantics.

1 Introduction

This paper develops a method for reusing and modifying deductive databases. The need for such techniques occurs when new rulebased applications differ only slightly from existing ones or when an application is to be incrementally updated. Such techniques are particularly important when reprogramming is expensive or unreliable. Unfortunately, these issues have not yet received significant attention in the deductive database literature. This paper addresses attempts to address them, focussing on a particular class of modifications that we call *predicate substitution*. The idea is to reuse and modify a set of rules by substituting one predicate symbol for another. Augmented with this capability, deductive databases can better exploit the polymorphism that exists in many database operations.

Transitive closure is a simple example of such an operation. It is polymorphic in that every binary relation has a transitive closure. Nevertheless, in many deductive database systems, one cannot write a rulebase that computes the transitive closure of an arbitrary binary relation. Instead, one can only write rules that compute the transitive closure of a particular relation. The following rules, for example, compute the transitive closure of the relation R:

$$\begin{array}{l} TR(x,y) \leftarrow R(x,y). \\ TR(x,z) \leftarrow R(x,y),\, TR(y,z). \end{array} \tag{1}$$

To compute the transitive closure of another relation, say S, one must write a new set of rules, defining a new predicate TS in terms of S. In this way, a user is forced to rewrite rulebases if he wants to apply them to different sets of data.

Predicate substitution carries out this rewriting automatically. For instance, to compute the transitive closure of S, the user specifies that S be substituted for R in the two rules above. Two new rules, equivalent to the following, are then generated:

$$\begin{array}{l} TS(x,y) \leftarrow S(x,y). \\ TS(x,z) \leftarrow S(x,y),\, TS(y,z). \end{array} \tag{2}$$

If necessary, new predicates such as TS are also generated. In this way, predicate substitution allows a user to specify new rulebases by reusing and modifying old ones.

We believe that predicate substitution is a principle that can be applied to logic-based languages generally in order to facilitate reuse and modification. In this paper, however, we focus on Horn logics and on Datalog in particular. That is, we develop a logic programming language that combines logical inference with predicate substitution. The rules in this logic, called *rules*

*This paper appears in the *Proceedings of the ACM Symposium on the Principles of Database Systems (PODS)*, pages 316–330. Conference held June 2–4 1992 at San Diego, CA. Invited to a special issue of *Journal of Computer and System Sciences (JCSS)*.