Sequences, Datalog and Transducers*

Giansalvatore Mecca†

Dipartimento di Informatica e Sistemistica
Universitá di Roma "La Sapienza" — Italy
mecca@infokit.dis.uniroma1.it

Anthony J. Bonner‡

Department of Computer Science
University of Toronto — Canada
bonner@db.toronto.edu

Abstract

This paper develops a query language for sequence databases, such as genome databases and text databases. The language, called *Sequence Datalog*, extends classical Datalog with interpreted function symbols for manipulating sequences. It has both a clear operational and declarative semantics, based on a new notion called the *extended active domain* of a database. The extended domain contains all the sequences in the database and all their *subsequences*. This idea leads to a clear distinction between *safe* and *unsafe* recursion over sequences: safe recursion stays inside the extended active domain, while unsafe recursion does not. By carefully limiting the amount of unsafe recursion, the paper develops a safe and expressive subset of Sequence Datalog. As part of the development, a new type of transducer is introduced, called a *generalized sequence transducer*. Unsafe recursion is allowed only within these generalized transducers. Generalized transducers extend ordinary transducers by allowing them to invoke other transducers as "subroutines." Generalized transducers can be implemented in Sequence Datalog in a straightforward way. Moreover, their introduction into the language leads to simple conditions that guarantee safety and finiteness. This paper develops two such conditions. The first condition expresses *exactly* the class of PTIME sequence functions; and the second expresses *exactly* the class of *elementary* sequence functions.

*Appears in *Proceedings of the Fourteenth ACM Symposium on Principles of Database Systems (PODS)*, pages 23–35. Symposium held May 22–25 1995, San Jose, Ca. Invited to a special issue of *Journal of Computer and System Sciences (JCSS)*.

†Research partially supported by MURST and Consiglio Nazionale delle Ricerche (CNR).

‡Research partially supported by an operating grant from the Natural Sciences and Engineering Research Council of Canada (NSERC).

1 Introduction

Sequences represent an important feature of *Next Generation Database Systems* [3, 27]. In recent years, new applications have arisen in which the storage and manipulation of sequences of unbounded length is a crucial feature. A prominent example is *genome databases*, in which long sequences representing genetic information are stored, and sophisticated pattern matching and restructuring facilities are needed [11]. These new applications have led to the introduction of sequence types in recent data models and query languages (e.g. [2, 4, 5, 28]). In many cases, however, queries over sequences are described only by means of a set of predefined, *ad hoc* operators, and are not investigated in a theoretical framework. In other cases, (e.g. [16, 25]) query languages concentrate on *pattern extraction* capabilities and do not consider *sequence restructurings*. Although pattern recognition is a fundamental feature of any language for querying sequences, sequence restructurings are equally important. For example, in genome databases, one needs to compute the reverse of a sequence, concatenate sequences together, splice out selected subsequences, etc.

Sequence data presents interesting challenges in the development of query languages. For instance, the query language should be *expressive* both in terms of pattern matching and sequence restructurings. At the same time, it should have a natural syntax and a clear semantics. Finally, it should be *safe*. Safety and finiteness of computations is a major concern when dealing with sequences, since by growing in length, sequences can easily become infinite, even when the underlying alphabet—or domain—is finite. This means that, unlike traditional query languages, sequence queries can end up in nonterminating computations. To achieve expressiveness, database researchers have developed sequence query languages based on abstract machines, such as *transducers* (finite automata that can generate strings as well as read them). Unfortunately, to achieve safety, they have had to impose restrictions that severely limit expressiveness (*e.g.*, [12, 30, 14]).