Formal approaches to the semantics of databases and database languages can have immediate and practical consequences in extending database integration technologies to include a vastly greater range of data sources and data structures. We consider three broad areas --- collection types, schema transformation, and partial information --- that are central to obtaining interoperability of heterogeneous data sources. In each of these areas we have developed working prototypes that have been put to practical use. This proposal describes work on collection types and schema transformation, and outlines a plan for the development of both principles and implementation of practical languages and tools that will extend database integration technology well beyond its current confines to cope with legacy systems, structured files, data-intensive applications, and other non-standard data sources.
Final Report on ARO Contract DAAH 04-95-1-0169

"Extending Database Integration Technology"
P. Buneman, S. Davidson and V. Tannen

The contract has partially supported the principal investigators and several PhD students.

**Schema integration and transformation** The need to transform data between heterogeneous databases arises from a number of critical tasks in data management. These problems are further complicated by schema evolution in the underlying databases, and by the presence of non-standard database constraints.

Davidson and Kosky describe a declarative language, WOL, for specifying such transformations, and an implementation, Morphase, based on this language. WOL is designed to allow transformations between the complex data structures which arise in object-oriented databases, as well as complex relational databases, and to allow for reasoning about the interactions between database transformations and constraints [21].

Kosky, Davidson and Buneman [1] discuss database transformations arising in many different settings including database integration, evolution of database systems, and implementing user views and data-entry tools. They also consider the problem of insuring the correctness of database transformations. In particular, we demonstrate that the usefulness of correctness conditions such as information preservation are hindered by the interactions of transformations and database constraints, and the limited expressive power of established database constraint languages.

**Semantics of collection types** Relying on previous work [3, 2] with R. Subrahmanyan and S. Naqvi (Bellcore) Buneman and Tannen have identified primitives based on instances of structural recursion on collections. Category theory served us to understand the central role played by a particular instance: monad primitives. Together with L. Wong, we were able to propose and exploit a partial foundation to programming with collections in query languages [9, 4]. Buneman, Libkin, Suciu, Tannen and Wong continued the study of the use of collection comprehensions in database programming languages. The syntax of comprehensions is very close to the syntax of a number of practical database query languages and is, they believe, a better starting point than first-order logic for the development of database languages [9, 8].

In collaboration with the Penn bioinformatics group this has in turn led to a system for information integration, Kleisli, that was specialized to molecular biology data sources, with significant practical impact [11, 5]. Davidson, Hara, and Popa have further extended the query system Kleisli to provide an interface to the Shore object-oriented database system [10].

While collection restructuring (e.g. the nested relational algebra) was nicely explained by the framework in [9, 4] aggregate operations on collections, collection constructors, and conversions between different kinds of collections were not. The monoid comprehension calculus of Fegaras and Maier provided such an approach. Together with K. Lellahi of University of Paris 13, Tannen was able to propose a more general approach, based on monad algebras and on a new robust notion of "enrichment" for monads [12]. Using this foundation, Tannen has designed the
core of our second-generation information integration system, K2, currently developed in our
Penn Center for Bioinformatics.

Focusing on another collection type, Libkin, Machlin and Wong have developed an array query
language and optimization techniques [13].

**Semi-structured data** A new kind of data model has recently emerged in which the database
is not constrained by a conventional schema. Systems like ACeDB, which has become very
popular with biologists, and the recent Tsimmis proposal for data integration organize data in
tree-like structures whose components can be used equally well to represent sets and tuples.
Such structures allow great flexibility in data representation.

Buneman, Davidson, Fernandez, Hillebrand and Suciu [7, 6, 15] propose a simple language
UnQL for querying data organized as a rooted, edge-labeled graph. In this model, relational
data may be represented as fixed-depth trees, and on such trees UnQL is equivalent to the
relational algebra. The novelty of UnQL consists in its programming constructs for arbitrarily
deep data and for cyclic structures. While strictly more powerful than query languages with path
expressions like XSQL, UnQL can still be efficiently evaluated. We describe new optimization
techniques for the deep or “vertical” dimension of UnQL queries. Furthermore, they show
that known optimization techniques for operators on flat relations apply to the “horizontal”
dimension of UnQL.

Fernandez, Popa and Suciu [14] have proposed a method of storing and querying semi-structured
data, using storage schemas, which are closely related to recently introduced graph schemas. A
storage schema splits the graph’s edges into several relations, some of which may have labels of
known types (such as strings or integers) while others may be still dynamically typed. They show
that all positive queries in UnQL, a query language for semistructured data, can be translated
into conjunctive queries against the relations in the storage schema. This result may be surpising,
because UnQL is a powerful language, featuring regular path expressions, restructuring
queries, joins, and unions.

**Path constraints** This class of constraints has been proposed for semistructured data to gener-
alize integrity constraints that are found in traditional database management systems. Implication
problems have been investigated by Buneman, Fan and Weinstein [16]. They characterized
a schema in M in terms of a type constraint and an equality constraint, and investigate the
interaction between these constraints and word constraints. They show that in the presence of
equality and type constraint, the implication and finite implication problems for word constraints
are also decidable, by giving a small model argument.

Looking at differences between semi-structured and structured data, one is tempted to think
that adding structure simplifies reasoning about path constraints. Surprisingly, this is not the
case. In the same paper it is shown that there is a fragment of the previously considered language
whose associated implication and finite implication problems are decidable in PTIME, but are
undecidable in the presence of type constraint.

**Descriptive complexity and parallel query compilation** Suciu and Tannen have proposed
a new framework for parallel processing of collections. Its theoretical justification is a charac-
terization (over ordered models) of the complexity class NC in terms of a divide-and-conquer
form of recursion on finite sets [19, 17]. In order to support the efficient parallel compilation
of expressive query languages, they have defined and implemented a high-level language called CoPa for parallel processing of nested sets, bags, and sequences (a generalization of arrays and lists), featuring a powerful form of parallelizable recursion. CoPa has a formal declarative definition of parallel complexity as part of its operational specification and it was used to prove that the compilation process (architecture-independent in its majority) preserves the asymptotic complexity of the code [18, 20]. This implementation has allowed them to conduct speedup and scaleup experiments on a LogP simulator for the cost of data communication, control communication, and local computations involved in the parallel implementation of query languages for object-oriented or object-relational databases [20].

References


