# Querying and Managing Navigational Databases

Research proposal

Filip Murlak

March 31, 2010

The diversity of data that needs to be handled by applications and humans has inspired a large variety of formal data models, different from the most popular relational model. Two models attracting a lot of attention in recent years are the hierarchical model and the network model.

The investigations into the *hierarchical model* are geared by the interest in the XML technology, which involves documents that have natural hierarchical structure. At mathematical level, those documents are modelled as trees storing data in their nodes. When querying such documents (databases), one makes use of the child and descendant relations between the nodes. More complex queries also refer to sibling order and data equality and inequality.

The *network model* is rapidly gaining importance owing to the proliferation of the RDF/S technology, social networks and biological networks. In this model data is organized into a graph. Pieces of data are stored in nodes, while labelled edges express various relations between them. One typically queries those graphs for tuples of nodes connected with paths whose sequence of labels satisfy some regular conditions.

The common feature of those models is that accessing the data involves navigation in some structure (a tree or a graph), rather that querying a collection of tuples. This requires a completely new approach, with tools based on automata theory and modal logic rather than finite model theory, which is the main theoretical framework for relational databases. The aim of the present project is to develop such tools. The focus is on two particular challenges: effective evaluation of expressive queries and handling heterogeneous data.

## 1 Query languages

Already early query languages for graph-structured data were refering to paths specified with regular expressions over the edge-labelling alphabet [6, 11]. Shortly thereafter, conjunctive regular path queries (CRPQs) became a paradigm for graph query languages [5, 9]

Emerging applications in social, criminal and biological networks, as well as the need to manage RDF data (see [4] for references) motivate search of more expressive, yet not too costly query languages for graph-structured data. A systematic study of such languages was recently commenced [4], resulting in a powerful extension of CRPQs. The extended queries not only express automata-based conditions on paths between the returned nodes, but are also able to return paths. This requires dealing with the fact that such sets of paths can be infinite even with finite graphs. This problem is typically resolved by returning a finite automaton representing the set of paths.

Nevertheless, many essential properties required by social and biological networks applications are still not expressible. Examples include one path being a subword, a suffix or a subsequence of another path. These properties on their own can be easily added to the language, but they become notoriously difficult when combined with regular conditions on paths. We would like to see how far one can extend CRPQs while preserving decidability, and examine the cost of adding various features to the query language. One example of mathematical problems that lay at the core of the above task is the following:

- Given a regular relation $R$, are there words $u, v$ such that $R(u, v)$ and $u$ is a subsequence of $v$?

## 2 Heterogeneous data

One of the main challenges of modern data management is dealing with heterogeneous data. Two typical scenarios involve data exchange and data integration.

In the *data exchange* scenario [8], one needs to restructure data stored in a source database under a target database schema, following a specification. The specification is given by a so-called *schema mapping*, a collection of logical formulae describing dependencies between the source schema and the target schema. The produced instance of data is called a *solution* for the source data with respect to the schema mapping.

In XML context (see [3]) the target schema, DTD or XML Schema, imposes complex conditions on the structure of the solution. One of the consequences is that in practise schema mappings are often *over-specified* such that for some source instances there is no solution satisfying the specification and conforming to the target schema. One needs to be able to verify that a mapping admits a solution for every source instance.

Slightly paradoxically, the mappings are often hugely *under-specified*, giving rise to many possible ways of populating the target database. The goal is to pick the best solution. The criterion for good solutions is to facilitate the so-called *certain-answers semantics* of queries. Ultimately, one wants to answer queries to the target database in a way independent from the decisions made when constructing the solution. More precisely, since there are many possible solutions, one can get different answers to the same query, depending on the target instance chosen. In the certain answers semantics this is remedied by

declaring true only those queries, that are true in every possible solution - regardless of the choices made. Thus, a good solution is one that satisfies the same properties that are satisfied by all solutions. Intuitively, good solutions, usually called *universal*, are those that do not invent facts. The existence of universal solutions depends on the monotonicity of the query language, and the degree to which the mapping is specified.

Nowadays, schema mapping design is usually assisted by specialized software. The ability to issue warnings that a schema is underspecified or overspecified would be a most welcome feature.

When a mapping is already there, one has to materialize the appropriate solution for the given source data. This requires checking if a universal solution exists, and constructing it. If there is no universal solution, one might need to resort to an arbitrary one.

One of the aims of the project is to analyze the existence of solutions, and universal solutions for different mapping languages. Examples of questions we are interested in are the following:

- Is there a (universal) solution for every possible source instance?
- Is there a (universal) solution for a given source instance?

Up to now, existence of solutions for arbitrary source instances was only investigated for very simple mappings, speaking only of child and descendant relation in the XML document [2]. More practical mapping languages involve sibling order and data comparisons. For such languages [2] considers only the problem of existence of solution for some source data, originally introduced in [3].

Materializing solutions was only considered for idealized mappings for which a natural universal solution always exists [7]. Again, this is usually not the case, and non-universal solutions need to be considered.

In contrast to the data exchange setting, *data integration* scenario [10] does not require materializing the target data, as it is assumed that the source data are constantly available. Instead, the queries to the target (global) database are rewritten and evaluated against the source (local) databases [1]. Again, the desired semantics is that of certain answers. Ideally, one would like to obtain a rewrite that returns exactly certain answers to the original query. Exact rewrites rarely exist and one usually relies on maximal consistent rewrites, i.e., rewrites returning only certain answers to the original query, and not properly contained in any other rewrite returning only certain answers.

Classically, certain answer is defined as the intersection of possible answers. This definition however only makes sense for queries returning tuples of values, and thus has very limited applicability to practical query languages returning structured documents rather then sets of tuples.

Recently, we have introduced a novel model-theoretic framework that generalizes the certain answers semantics to queries returning arbitrarily complex structures [7]. With the extended semantics at hand one should be able to

revisit the data integration tasks in XML context and see if query rewriting techniques can be developed for real-life languages like XQuery.

# References


[1] S. Abiteboul, O. Duschka. Complexity of answering queries using materialized views. Proc. PODS 1998, pp. 254–263.

[2] S. Amano, L. Libkin, F. Murlak. XML Schema Mappings. Proc. PODS 2009, pp. 33–42.

[3] M. Arenas, L. Libkin. XML data exchange: consistency and query answering. *J. ACM* 55(2): (2008).

[4] P. Barceló, C. Hurtado, L. Libkin, P. Wood. Expressive Languages for Path Queries over Graph-Structured Data. Proc. PODS 2010 (to appear).

[5] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. Proc. KR 2000, pp. 176–185.

[6] M. P. Consens, A. O. Mendelzon. GraphLog: a visual formalism for real life recursion. Proc. PODS 1990, pp. 404–416.

[7] C. David, L. Libkin, F. Murlak. Ceratin Answers for XML Queries. Proc. PODS 2010 (to appear).

[8] R. Fagin, Ph. Kolaitis, R. Miller, L. Popa. Data exchange: semantics and query answering. *TCS* 336 (2005), 89–124.

[9] D. Florescu, A. Levy, D. Suciu. Query containment for conjunctive queries with regular expressions. In PODS 1998, pp. 139–148.

[10] M. Lenzerini. Data integration: a theoretical perspective. Proc. PODS 2002, pp. 233–246.

[11] A. O. Mendelzon, P. T. Wood. Finding regular simple paths in graph databases. SIAM J. Comput., 24(6):1235–1258, 1995.