The Query Language TQL
Demo Presentation

Giovanni Conforti, Giorgio Ghelli
Antonio Albano, Dario Colazzo, Paolo Manghi, and Carlo Sartiani

Dipartimento di Informatica
Università di Pisa, Pisa, Italy

Abstract. This work presents the query language TQL, a query language for semistructured data, that can be used to query XML files. TQL substitutes the standard path-based pattern-matching mechanism with a logic-based mechanism, where the programmer specifies the properties of the pieces of data she is trying to extract. This feature makes some queries easier to express, and should allow the adoption of better optimization techniques. Through a set of examples, which will be presented at the demo, we show that the range of queries that can be declaratively expressed in TQL is quite wide.

1 Introduction

The language TQL [1] is a query language for semi-structured data. The language is based on the set comprehension (match-filter-construct) paradigm, in the tradition of SQL, StruQL, Lorel, Quilt, XQuery (among many others). However, the match-filter operation is expressed in TQL using a variant of the ambient logic [2], a logic defined to describe process structure and behaviour. TQL adopts a subset of that logic to express the binding (match-filter) part of a query. The same logic can be exploited to describe those properties of the data that are usually expressed through types and constraints. The promise of combining the expression of types, constraints, and queries in just one language, and to use this synergy for optimization and error-checking purposes, is the kernel of the TQL project. But the language is also worth studying for its ability to express complex queries by declaring the properties of what one is looking for, instead of describing a path to arrive there.

In this paper we show, through a set of examples which will be presented at the SEBD demo, that the range of queries that can be declaratively expressed in TQL is quite wide. We implictly report about the current status of the implementation by writing and testing all queries using the version of TQL that has been implemented, and that can be freely downloaded from `http://tql.di.unipi.it/tql`.

The contact author is Giovanni Conforti `confor@di.unipi.it`

2 The Simplest Queries

2.1 The Input Data

We begin with some standard queries, borrowed from the W3C XMP Use Case [3]. These queries operate over the document `http://tql.di.unipi.it/bib.xml`, which we assume to be bound to the variable `$Bib` in the global environment (the TQL system allows any document on the web to be bound to a variable). In this paper we present the XML file using its compact TQL-syntax representation, which looks as follows:

```
bib[
  book[ year[1992] | title[FoundationsDatabases]
    | author[ first[Serge]   | last[Abiteboul] ]
    | author[ first[Richard] | last[Hull] ]
    | author[ first[Victor]  | last[Vianu] ]
    | publisher[Addison] | price[60]
  ]
  | book[year[1990]  | title[SistemiOperativi]
    | author[ first[Piero] | last[Maestrini] ]
    | publisher[McGrawHill] | price[38]
  ]
...
```

In this format, `bib[C]` stands for an element tagged `bib` whose content is `C`, while `C1 | C2` is the concatenation of two elements, or, more generally, of two sets of elements. We use this non-XML notation because TQL is born as a language to query semistructured data in general, i.e. unordered trees with labelled edges, and not just XML. XML is just one way to construct such trees, using tagged elements (and attributes) to build labelled edges.

2.2 Matching and Binding

The basic TQL query is `from Q |= A select Q'`, where `Q` is the *subject* (or *data source*) to be matched against the formula `A`, and `Q'` is the result expression. The matching of `Q` and `A` returns a set of bindings for the variables that are free in `A`. `Q'` is evaluated once for each of these bindings, and the concatenation of the results of all these evaluations is the query result.

For example, consider the following TQL query, that returns the titles of all books written in 1991, and is evaluated in an environment where `$Bib` is bound as specified above.

```
from $Bib |= .bib[ .book[ .year[1991] And .title[$t] ] ]
select title[$t]
```

The formula `.bib[ .book[ .year[1991] And .title[$t] ] ]` is an ambient logic formula, which should be read as: "there is a path `.bib[ .book[ ]]` that

reaches a place that matches `.year[1991] And .title[$t]`, i.e. a place where you find both a path `.year[]` leading to `1991` and a path `.title[ ]` leading to something, that you will call `$t`".

The formula `.tag[A]`, read "there exists an element `tag` whose content satisfies `A`", is the most useful operator, but is actually defined in terms of three more basic operators, truth `T`, vertical splitting `A' | A''`, and element matching `tag[A]`.

The element formula `tag[A]` only matches a one-element document: while `.t[A]` matches both trees `t[D]` and `t[D] | t2[D2] | ...` (provided that `A` matches `D`), the formula `t[A]` only matches the first one. The truth formula `T` matches every tree. Finally, the formula A_1 | A_2 matches `D` iff `D` is equal, modulo reordering, to D_1 | D_2, with A_i matching D_i. For example, the following pairs match, provided that `$a` is bound to `Date`:

```
title[IDB] | author[Date] | year[94]   author[$a] | title[IDB] | year[94]
title[IDB] | year[94]                 T
title[IDB] | author[Date] | year[94]   author[$a] | T
author[Date]                          author[$a] | T
```

The third formula can be read as: there is an `author $a` and something else, hence it is equivalent to `.author[$a]`; the fourth pair matches as well, since the empty tree matches `T`. Hence, `m[A] | T` is equivalent to `.m[A]`; this is actually the official definition of the semantics of `.m[A]`.

While in this example we matched `$t` with a leaf, a TQL variable can be matched against any tree, or against a tag.

For example, the following query returns any tag inside a `book` whose content is `Serge`; `.a.b[A]` abbreviates `.a[.b[A]]`.

```
from $Bib |= .bib.book.$tag.first[Serge]
select SergeTag[$tag]
```

Hereafter, as a convention, we use lowercase initials for variables that are bound to tags and uppercase initials for variables that are bound to trees.

Finally, the following query matches the formula `year[1992] | $EveryThingElse` against any book, hence it returns, for any book whose year is 1992, everything but the year:

```
from $Bib |= .bib.book[year[1992] | $EveryThingElse]
select BookOf1992[$EveryThingElse]
```

Since we have two books of 1992, there are two possible bindings for `$EveryThingElse`, each corresponding to the whole content af a 1992 book without its `year` subtree; hence the result is:

```
BookOf1992[
    title[FoundationsDatabases]
    | author[ first[Serge] | last[Abiteboul] ]
    ...
]
```

```
| BookOf1992[
    | title[Interpreters]
    | author[ first[Vincent] | last[Aho] ]
    ...
]
```

2.3 Matching and Logic

TQL logic allows the programmer to combine matching and logical operators. For example, the condition in the following query combines the request for the existence of a `title` field, of a `$x` field containing `Springer`, and of either an `author.last` or an `editor.last` path leading to `Buneman`.

```
from $Bib |= .bib.book[
              .title[$t] And Exists $x. .$x[Springer]
              And (.author.last[Buneman] Or .editor.last[Buneman])
             ]
select title[$t]
```

Another interesting feature of the language is its simple compositional semantics [1]. Blending compositional properties with structural/logical formulas gives rise to several algebraic equivalences over queries, which can then be used to identify implementable algebraic optimisations. Conjunction, disjunction, and universal quantification are operators that can be found in many match-based languages. TQL, however, has the full power of first-order logic, hence we can express universal quantification and negation of arbitrary formulas. This is exemplified in [5].

3 Other Demo Examples

The demo goes on with a set of example queries borrowed by [5]. For reason of space we report here a candidate query only for each interesting feature of TQL presented in [5]:

- **Restructuring data source by nesting:** build a bibliography grouped by author (nesting of queries)

```
from   $Bib |= .bib.book.author[$A]
select author[authorname[$A]
              | from $Bib |= .bib.book[author[$A]
                                       | $OtherFields
                                      ]
                select book[$OtherFields]
              ]
```

- **Querying in absence of schema:** search all tags whose content satisfies `.first[Serge]`

```
from $Bib |= .bib.book.$tag.first[Serge]
select SergeTag[$tag]
```

– **Checking a given propery:** is title a mandatory tag?

```
from $Bib |= .bib!book.title[T]
select titleIsMandatory
```

– **Extract pieces of data that satisfy a property:** extract all mandatory tags inside a book

```
from $Bib |= .bib!book.$x[T]
select mandatoryTag[$x]
```

These examples sketch an idea of the TQL expressive power, showing its ability to express properties of semistructured data and to extract all pieces of data that satisfy a given property.

4 Conclusions

All queries presented in this paper and in [5] are executable in the prototype version of the TQL evaluator, and can be found in the file `demo.tql` in the standard distribution. The current version of the prototype still works by loading all data in main memory, but is already based on a translation into an intermediate TQL Algebra [4], with logical optimizations carried on both at the source and at the algebraic level. The intermediate algebra works on infinite tables of trees, represented in a finite way, and supports such operations as complement, to deal with negation, co-projection, to deal with universal quantification, several kinds of iterators, to implement the | operator, and a recursion operator.

TQL is currently based on a unordered nested multi-sets data model. The extension of TQL's data model with ordering is an important open issue.

References

1. L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *Proc. of European Symposium on Programming (ESOP), Genova, Italy*, 2001. Available from http://www.di.unipi.it/~ghelli/papers.html.
2. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. of Principles of Programming Languages (POPL)*. ACM Press, January 2000.
3. Don Chamberlin, Peter Fankhauser, Massimo Marchiori, and Jonathan Robie. XML Query Use Cases. Technical report, World Wide Web Consortium, April 2002. W3C Working Draft.
4. G. Conforti, O. Ferrara, and G. Ghelli. TQL Algebra and its Implementation. In *Proc. of IFIP International Conference on Theoretical Computer Science (IFIP TCS)*, Montreal, Canada, 2002. To appear.
5. G. Conforti, G. Ghelli, A. Albano, D. Colazzo, P. Manghi, and C. Sartiani. The Query Language TQL. In *Proc. of Workshop on the Web and Data Bases (WebDB)*, Madison, Wisconsin, 2002. To appear.