I4-D17

Testing of the Language Implementations

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Munich/I4-D17/D/PU/a1
Responsible editors:	Tim Furche
Reviewers:	Liviu Badea and Tim Geisler
Contributing participants:	Munich
Contributing workpackages:	I4
Contractual date of deliverable:	29 February 2008
Actual submission date:	10 March 2008

Abstract

This deliverable is devoted to the testing and improvement of the language prototypes developed by I4. In the first half of the deliverable testing and improvement of single-rule programs will be the focus. In the second half of the deliverable, we report on testing and implementation of the RDFLog prototype.

Though feature-by-feature testing has been performed for both prototypes, we report here mainly performance results. Testing and improvement of the language prototypes continues after REWERSE and is reported on at `http://amachos.com/Comparison/` and `http://rdflog.com/Comparison/`.

Keyword List

Xcerpt, prototype, RDFLog, implementation, testing, benchmarks

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Testing of the Language Implementations

François Bry[1], **Tim Furche**[1], **Benedikt Linse**[1]

[1] Institute for Informatics, University of Munich, Germany
http://pms.ifi.lmu.de/

10 March 2008

Abstract
This deliverable is devoted to the testing and improvement of the language prototypes developed by I4. In the first half of the deliverable testing and improvement of single-rule programs will be the focus. In the second half of the deliverable, we report on testing and implementation of the RDFLog prototype.

Though feature-by-feature testing has been performed for both prototypes, we report here mainly performance results. Testing and improvement of the language prototypes continues after REWERSE and is reported on at http://amachos.com/Comparison/ and http://rdflog.com/Comparison/.

Keyword List
Xcerpt, prototype, RDFLog, implementation, testing, benchmarks

Contents

Overview of this Deliverable

For the rule-based, I4 query and reasoning language Xcerpt, we are currently developing a novel implementation based on the principles outlined in deliverables I4-D15a and I4-D15b. This prototype advances the state-of-the-art considerably over existing Xcerpt prototypes as well as implementations of other Web query languages:

- It is based on a novel algebra, called CIQCAG, that is expressive enough to provide evaluation not only for Xcerpt but also for XPath, XQuery, SPARQL, and many other Web query languages. Translators for (large fragments of all) four named languages are specified.
- Though capable of evaluating arbitrary graph shaped Xcerpt or XQuery queries, the CIQCAG algebra still provides linear time and space evaluation of tree queries even on many non-tree graphs (viz. so-called continuous-image graphs). No previous approach scales as well as the CIQCAG algebra with respect to query and data shape.

We are currently in the process of finalizing and testing that prototype. In this deliverable, we discuss some preliminary test results. For more up-to-date and extensive tests, see `http://amachos.com/Comparison` where we continue to provide updates on the prototype and its implementation.

The rest of this deliverable is divided in two sections. The first section details test results on a preliminary version of the above prototype published in [2]. The second section illustrates part of the reduction from SPARQL, its rule-based extensions, and similar RDF query languages to CIQCAG. It uses the RDFLog framework discussed in previous deliverables (I4-D14) and shows that even a very naive implementation of RDFLog can compete with existing SPARQL processors and that the reduction of blank nodes to function symbols is efficient and feasible.

1 Experimental Evaluation of Memoization in the Xcerpt Prototype

The experimental evaluation is based on both synthetic and on real data. The set of structural relations is extended by the additional relations ATTRIBUTE and VALUE in order to support attribute queries. The tests have been executed on an AMD Athlon 2400XP machine with 1GB main memory. The algorithms are implemented in Java executed on JVM version 1.5. All tests show the processing time without data parsing. Each measurement is averaged over 500 runs.

Synthetic data is used to confirm the complexity of the presented algorithms. The real data scenarios stem from the University of Washington XMLData repository[1], and demonstrate the competitiveness of the algorithms.

The first experiment confirms on synthetic data how essential the memoization of intermediary results is, not only for the complexity but also for the experimental query evaluation time. The $\mathsf{Match}_{\downarrow}$ algorithm without memoization of variable domains (i.e., the helper structure ρ) exhibits an exponential growth of time consumption in the size of the query (cf. Fig. 1), because several common sub-matrices are built repeatedly. In contrast, Fig. 2 depicts the effect of increasing arity in a worst-case scenario, where the query is unrestrictive, i.e., a binding for one answer variable is related to all bindings of another one.

Fig. 3 shows a *comparison between the two approaches* for matrix population discussed in this article. A path query consisting of four variables and CHILD* (descendant) relations only, but without label restrictions, is used. This query exhibits worst case complexity for the top-down algorithm $\mathsf{Match}_{\downarrow}$, as the match context is never restricted by a previous context. As expected, the plot shows a quadratic runtime growth in the data size for the top-down algorithm and the bottom-up algorithm with CHILD* index. Without this index, the bottom-up approach exhibits a cubic runtime.

At least the top-down algorithm performs quite well even in its basic form discussed here in real query scenarios. Fig. 4 shows how the runtime of the top-down algorithm scales with the data size for path, tree and graph shaped queries. These queries are executed over the MONDIAL[2] database of geographical information. The plot shows additionally that already for path queries the bottom-up algorithm exhibits polynomial runtime; the naive bottom-up approach has an average runtime that is very close to its worst-case. On the other hand, the $\mathsf{Match}_{\downarrow}$ exhibits a linear runtime in all queries, even in the graph query experiment.

The final test on increasing fragments of a large XML document, the Nasa dataset from the above mentioned repository, shows that the runtime of $\mathsf{Match}_{\downarrow}$ scales nicely with the data size and is very competitive even in the basic form implemented for this experiment.

[1] `http://www.cs.washington.edu/research/xmldatasets/`
[2] `http://www.dbis.informatik.uni-goettingen.de/Mondial/`

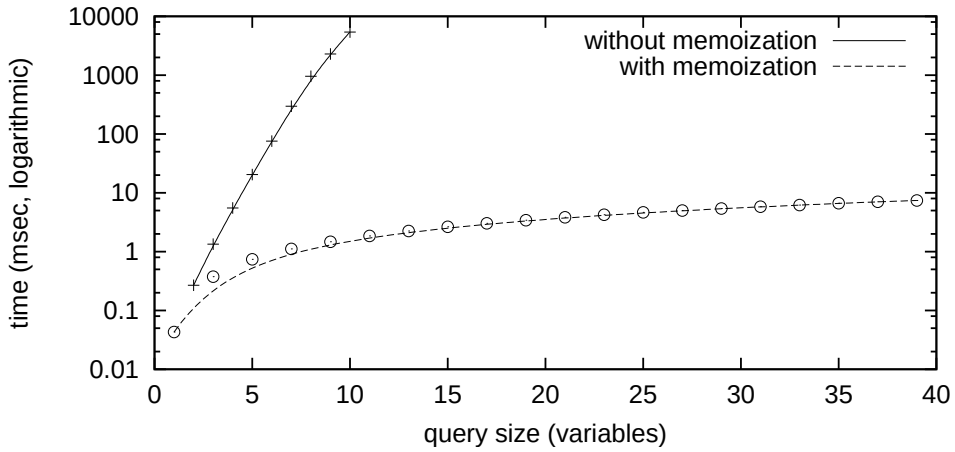

Figure 1: Effect of Memoization over Query Size (data synthetic, uniform, deeply nested; bindings for query variables overlap considerably)

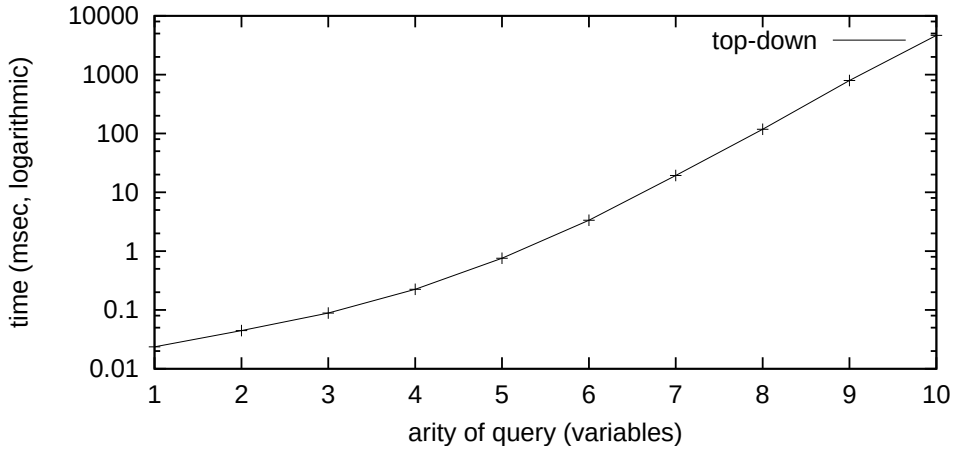

Figure 2: Worst-Case Effect of Query Arity (data and query as before)

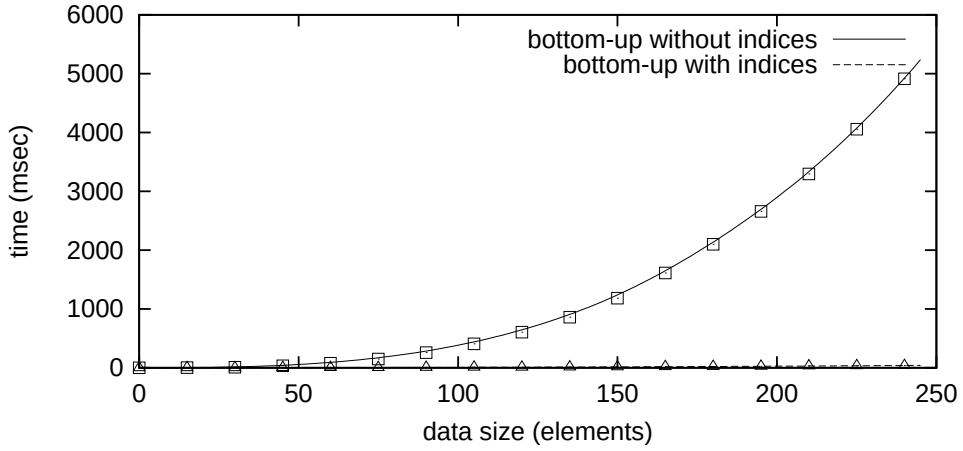

Figure 3: Comparison Top-Down and Bottom-Up (data synthetic, size increased by adding in depth; query small, containing many descendants)

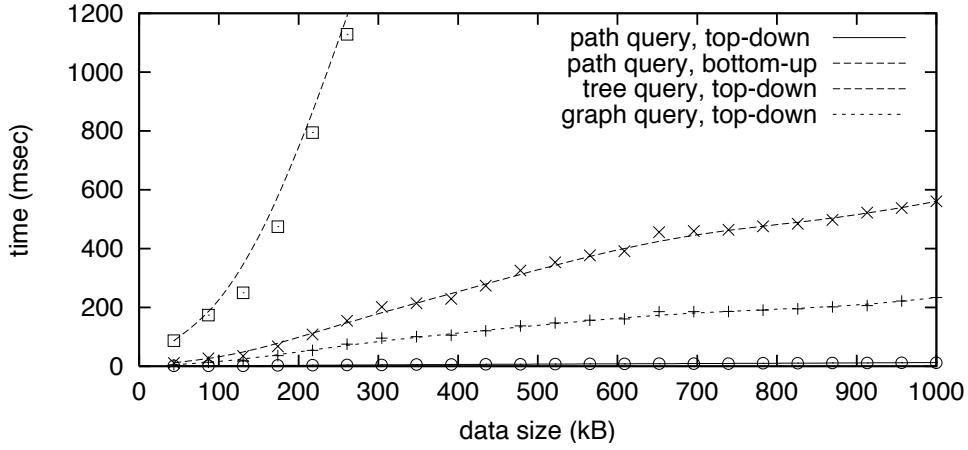

Figure 4: Query Classes over Real-life Data (Mondial; queries simple, but with arity > 1)

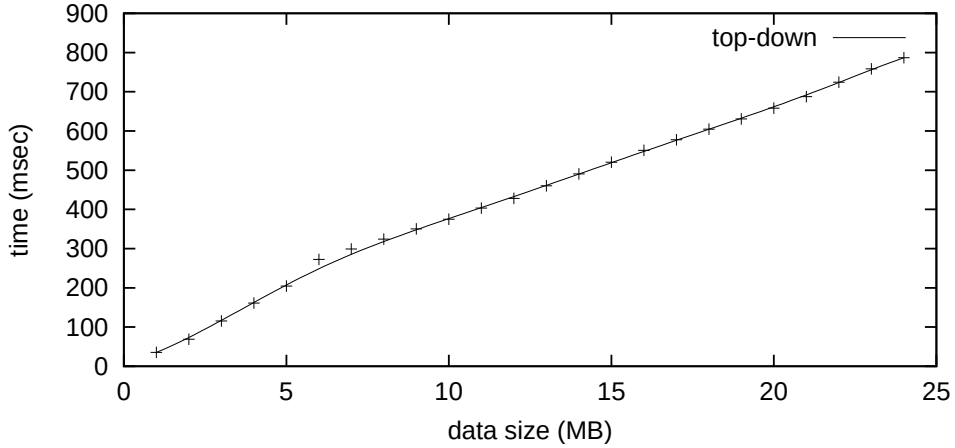

Figure 5: Top-Down over Large, Real-life Nasa Data (binary tree query)

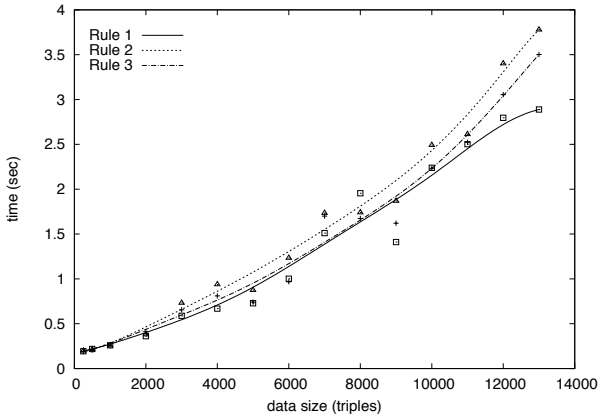

Figure 6: Performance of RDFLog on rules 1, 2 and 3

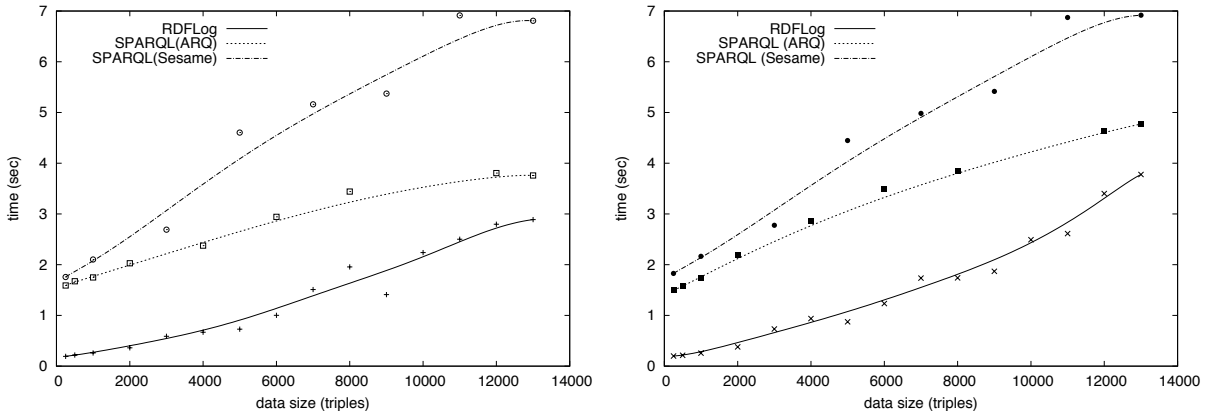

Figure 7: Performance comparison of RDFLog, ARQ and Sesame on rule 1 (left hand side) and on rule 2 (right hand side)

2 Experimental Evaluation of the RDFLog Prototype

In this section we experimentally confirm the central theoretical results of [1] (viz. that Skolemisation and un-Skolemisation is in LOGSPACE) and validate the claim that the resulting implementation of RDFLog yields a competitive performance by a comparison with two other in-memory processors for RDF queries, the ARQ (Version 2.1) SPARQL processor of Jena and the SPARQL engine provided by the Sesame RDF Framework. For Sesame, we choose the main-memory store ("by far the fastest type of repository that can be used" according to the Sesame Manual, `http://www.openrdf.org/doc/sesame2/users/`). With this store, Sesame becomes a main-memory, ad-hoc query engine just like RDFLog and ARQ. We have implemented RDFLog using a combination of Perl pre- and post-filters for Skolemisation and un-Skolemisation of RDFLog programs and XSB Prolog to evaluate the Skolemised programs. The experiments have been carried out on a Intel Pentium Mobile Dual-Core with 1.86 GHz, 1024 KB cache and 2 GB memory.

In the experiments we query an RDF dump of Wikipedia meta-data and compare three different kinds of rules:

- A pure datalog rule of the form $\forall\ \mathsf{X}\ \forall\ \mathsf{Y}\ \mathsf{a}(\mathsf{X},\mathsf{Y}) \rightarrow \mathsf{b}(\mathsf{X},\mathsf{Y})$ which does not construct any blank nodes and is also expressible in SPARQL.
- An RDFLog rule with the existential quantifier within the scope of universal quantifiers of the form $\forall\ \mathsf{X}\ \forall\ \mathsf{Y}\ \exists\ \mathsf{Z}\ \mathsf{a}(\mathsf{X},\mathsf{Y}) \rightarrow \mathsf{b}(\mathsf{X},\mathsf{Z})$. This rule is also expressible in the SPARQL query language.
- An RDFLog rule with the existential quantifier outside of the scope of universal quantifiers of the form $\exists\ \mathsf{Z}\ \forall\ \mathsf{X}\ \forall\ \mathsf{Y}\ \mathsf{a}(\mathsf{X},\mathsf{Y}) \rightarrow \mathsf{b}(\mathsf{X},\mathsf{Z})$. This rule is *not* expressible in SPARQL.

The experiments have been performed against RDF graphs consisting in between 250 and 13000 triples. For each setting, the results are averaged over 10–50 runs.

Figure 6 shows the performance of RDFLog for each of the three rule classes discussed above. Note, that the increasing expressiveness of the blank node construction (from rule type 1 to rule type 2) does not have any visible effect on the performance. The kind of blank node construction not supported by SPARQL (rule 3) even performs slightly better than rule 2. This may be attributed to the lower amount of blank nodes generated as the existential quantifier is outside of the scope of all universal quantifiers.

Figure 7 compares the performance of RDFLog with that of ARQ and Sesame for rule 1 (left hand side) and rule 2 (right hand side). Despite its light-weight, ad-hoc implementation, RDFLog outperforms ARQ and Sesame in this setting. The figures show moreover that also for ARQ and Sesame, blank node construction does not bear any additional computational effort.

Acknowledgements.

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

References

[1] F. Bry, T. Furche, C. Ley, and B. Linse. Rdflog: Filling in the blanks in rdf querying. Technical Report PMS-FB-2008-01, University of Munich, 2007.

[2] F. Bry, T. Furche, B. Linse, and A. Schroeder. Efficient Evaluation of n-ary Conjunctive Queries over Trees and Graphs. In *Proc. ACM Int'l. Workshop on Web Information and Data Management (WIDM)*. ACM Press, 2006.