Extending SPARQL with Temporal Logic

Radu Mateescu, Sébastien Meriot, Sylvain Rampacek

Extending SPARQL with Temporal Logic

Radu Mateescu[1], Sébastien Meriot[1], and Sylvain Rampacek[2]

[1] Inria Grenoble – Rhône-Alpes / Vasy
Inovallée, 655, av. de l'Europe, Montbonnot, F-38334 Saint Ismier, France
{Radu.Mateescu,Sebastien.Meriot}@inria.fr
[2] Laboratoire Electronique Informatique et Image (Le2i)
Faculté des Sciences Mirande, Université de Bourgogne, F-21078 Dijon, France
Sylvain.Rampacek@u-bourgogne.fr

Abstract. The data integration and sharing activities carried on in the framework of the Semantic Web lead to large knowledge bases that must be queried, analyzed, and exploited efficiently. Many of the knowledge representation languages of the Semantic Web, starting with Rdf, are based on directed, labeled graphs, which can be also manipulated using graph algorithms and tools coming from other domains. In this paper, we propose an analysis approach of Rdf graphs by reusing the verification technology developed for concurrent systems. To this purpose, we define a translation from the Sparql query language into Xtl, a general-purpose graph manipulation language implemented in the Cadp verification toolbox for asynchronous concurrent systems. This translation makes it possible to extend the expressive power of Sparql naturally by adding Xtl temporal logic formulas characterizing sequences, trees, or general subgraphs of the Rdf graph. Our approach exhibits a performance comparable with that of dedicated Sparql query evaluation engines, as illustrated by experiments on large Rdf graphs.

1 Introduction

During the last decade, the research and development activities performed within the framework of the Semantic Web [5] led to large and complex knowledge bases represented using the languages standardized by the World Wide Web (W3C) consortium, such as Owl [11] and Rdf [20]. As the quantity of information contained in structured ontologies grows, the need for expressive languages and efficient tools performing consistency checking, querying, and exploration becomes more stringent. Several technologies dedicated to these aspects are now well-established, taking their roots in description logics (reasoners for Owl ontologies, such as FaCT++ [31]), relational databases (query languages for Rdf documents, such as Sparql [29]), or graph manipulation (navigation languages for Xml [30] documents, such as XPath [3]). The recent research efforts are oriented towards the integration of different semantic features into the same language, as illustrated by the XQuery [6] language for querying Xml documents, which combines the path description constructs of XPath and the relational database operators of Sql [19].

The knowledge bases built using the Semantic Web languages share a common underlying semantic model, namely the directed, labeled graphs provided by their representation as RDF documents. This allows the manipulation of knowledge bases by means of graph-based algorithms and tools available from other domains, such as the formal verification of concurrent systems. In this paper, we propose an extension of the SPARQL language with temporal logic properties enabling to characterize sequences, trees, and/or cycles in RDF graphs. We achieve this extension by reusing the technology available in the CADP [15] verification toolbox for asynchronous, concurrent systems. First, we give a scheme for encoding RDF graphs into the compact BCG format used by CADP for representing the state spaces of concurrent programs. Then, we devise a translation of SPARQL queries into the XTL [27] language used by CADP for exploring graphs encoded in the BCG format. Although originally defined for the model checking of temporal logic operators extended with data, XTL is a general-purpose functional language able to describe various kinds of graph explorations, and proves to be useful also for encoding relational algebraic operators. Finally, we propose to extend the SPARQL syntax with two new clauses enabling to evaluate temporal logic properties on nodes of the RDF graph.

To assess the usefulness of our approach, we carried out several query evaluation experiments on RDF graphs of increasing size, by using both the BCG and XTL-based approach and a specialized SPARQL evaluation engine. The results obtained show that our approach yields comparable performance in memory and time w.r.t. the specialized one, and therefore can provide a useful alternative for querying and analyzing RDF graphs.

Related Work. Searching the information contained in graphs is a long-standing problem in computer science. Since the early works on graph querying that led to the languages G [9] and CECIL [23], regular expressions and extensions thereof appeared as a handful means for characterizing and searching sequences in directed graphs. Regular expressions have also been successfully used in the database field as an extension of the traditional query mechanisms, for instance in the GRAPHLOG [8] language for searching databases containing graphs or in the LOREL [1] language for querying object-oriented databases containing semi-structured data.

In the field of Semantic Web, several approaches were proposed for querying graphs, most of them having led to languages standardized by the W3C, such as XPATH, XQUERY, and SPARQL. More specifically related to RDF, various extensions of SPARQL were proposed in order to increase its expressiveness. SPARQLER [21] and PSPARQL [2] are extensions of SPARQL with regular expressions involving data, which allow the characterization of paths of arbitrary length in addition to the relational query mechanisms based on join operations. Other extensions of SPARQL with XPATH operators, extended regular expressions (similar to those of the `egrep` UNIX utility), and graph patterns, were incorporated into the ARQ[3] evaluation engine or in additional libraries, such as GLEEN[12].

[3] `http://jena.sourceforge.net/ARQ`

The applications of verification in this setting so far were mainly focused on checking properties of the World Wide Web seen as a graph [10] and on the usage of model checking as evaluation engine for XPath queries [14]. Our approach relies upon the Bcg environment for compact graph representation equipped with Xtl, a general language able to describe arbitrary fixed point computations on graphs and to extract and manipulate the data values contained in transition labels. From this point of view, Xtl offers graph manipulation primitives of the same nature as those available in XQuery for searching Xml documents, and allows to freely integrate relational and graph-based operations.

Paper Outline. Section 2 gives a quick overview of Rdf, Sparql, and the verification technology provided by the Cadp toolbox. Section 3 describes the encodings of Rdf graphs as Ltss in the Bcg format accepted by Cadp, the translation of Sparql queries to Xtl, and their extension with temporal logic properties. Section 4 shows experimental results comparing the performance of our approach and of dedicated Sparql evaluation engines on large Rdf graphs. Finally, Section 5 summarizes the results and gives directions for future work.

2 Background

RDF and SPARQL. Rdf (*Resource Description Framework*) [20] is a knowledge representation language dedicated to the annotation of resources within the framework of the Semantic Web. An Rdf document can be represented as a set of triples $\langle subject, predicate, object \rangle$, where the predicate (denoted by an Iri, an *Internationalized Resource Identifier*) expresses the relationship between a subject (denoted by an Iri or a blank node) and an object (denoted by an Iri, a blank node or a literal). Another equivalent representation is as a directed labeled graph in which nodes are labeled by subjects or objects, edges are labeled by predicates, and each edge connecting two nodes corresponds to a triple relating the subject label of the source node to the object label of the target node via the predicate label of the edge. The World Wide Web Consortium (W3C) defined two standardized syntaxes for Rdf graphs: the N3 [4] notation and the Xml [30] serialization.

To search and extract information from Rdf documents, the W3C defined the Sparql [29] query language, which takes its roots in the Sql [19] language used for querying relational databases.

CADP and XTL. Cadp[4] (*Construction and Analysis of Distributed Processes*) [15] is a state-of-the-art verification toolbox for asynchronous concurrent systems. It has been applied over the years for validating over 110 industrial case-studies[5] and for developing over 40 derived research tools[6]. The toolbox accepts as input several description languages with process algebraic flavour (such as

[4] `http://www.inrialpes.fr/vasy/cadp`
[5] See the online catalog at `http://www.inrialpes.fr/vasy/cadp/case-studies`
[6] See the online catalog at `http://www.inrialpes.fr/vasy/cadp/software`

LOTOS [18], FSP [25], and CHP [26]) and also lower-level formalisms such as the EXP [24] language for networks of communicating automata. All these languages are compiled into labeled transition systems (LTSs), which are state/transition graphs representing the behaviour of concurrent systems.

CADP provides several representations for LTSs, among which the BCG (*Binary Coded Graphs*) compact file format equipped with specific compression algorithms and with a set of tools and libraries implementing various features (reading/writing, relabeling, conversion to other formats, visualization, etc.). XTL (*eXecutable Temporal Language*) [27] is a functional language and tool for the description and evaluation of temporal logic properties involving data values on BCG graphs. The language enables to handle sets of states and transitions, to extract the data values contained in transition labels, and to describe the fixed point computations underlying temporal logic operators by means of forward or backward traversals of the graph. These features make BCG and XTL particularly suitable as targets for translating RDF graphs and SPARQL queries, respectively.

3 Translating SPARQL to XTL

We begin this section by showing how RDF graphs can be encoded as LTSs in the BCG format, then we present the translation from SPARQL to XTL, and finally we propose to extend SPARQL queries with new clauses allowing to check temporal logic properties on the nodes of RDF graphs.

3.1 Encoding RDF graphs in BCG

The directed, labeled graphs representing RDF documents associate information both to nodes (subjects and objects) and to transition labels (predicates). The LTSs representing the state spaces of concurrent programs attach information only to transition labels (events or actions). Therefore, in order to convert an RDF graph into an LTS, all the information attached to the nodes must be moved into the transition labels. A natural way of achieving this is by converting every edge $s \xrightarrow{p} o$ of the RDF graph into an edge of the form $s \xrightarrow{s\ p\ o} o$ in the LTS. This is the classical translation from state-based models (Kripke structures) to action-based models (LTSs) defined, e.g., in [7]. It has the advantage of being succinct, the resulting LTS having exactly as many states and transitions as nodes and edges in the RDF graph.

In the BCG format, states are encoded as natural numbers and transition labels have the form $G\ v_1 ... v_n$, where G is the name of a gate (communication channel) and $v_1, ..., v_n$ are data values exchanged during communication. To represent in the BCG format an LTS corresponding to an RDF graph, the character strings denoting subjects and objects are given unique indexes encoding states, and the transition labels are of the form "$A\ s\ p\ o$", where A is a fictitious gate name adopted by convention and s, p, o are the character strings of the corresponding subject, predicate, and object. Using this encoding scheme,

RDF graphs represented in the N3 format can be straightforwardly converted into LTSs represented in the BCG format.

3.2 Encoding SPARQL operators in XTL

XTL implements various operations for manipulating sets of states, labels, and edges of the LTS, including the enumeration of all elements in a set. It is also equipped with a preprocessor enabling to define parameterized macros and to group them into reusable libraries. The principle we adopt for encoding SPARQL queries in XTL is to enumerate and display all the solutions of a relational algebraic term. This is illustrated in Figure 1 for the BGP (*Basic Graph Pattern*) operator of SPARQL, which matches a subset of the triples contained in an RDF graph by extracting their fields in data variables and/or matching them against data values. The BGP is translated into an XTL iteration macro BGP taking four parameters: *pattern* contains the three fields of the BGP (constants or variable declarations); *property* denotes a boolean expression that may use the data variables extracted by the BGP; and b_1, b_2 represent the bodies of the iteration, i.e., the XTL printing statements that must be executed for each triple matching the BGP and when no triple matches the BGP, respectively.

```
macro BGP (pattern, property, b1, b2) =
  if for E:edge
       apply or
       from false
       to if E -> [ A pattern where property ] then
            let a:action = b1 in true end_let
          else
            false
          end_if
     end_for
  then
     nop
  else
     b2
  end_if
end_macro
```

Fig. 1. XTL iteration macro enumerating all triples matching a BGP

The XTL "for" expression enumerates all edges in the graph and accumulates in an internal boolean variable (initialized to false and updated at each iteration) the fact that a matching edge has been encountered or not. The label of the current edge E, which has the form "$A\ s\ p\ o$", is then matched against the *pattern* and the boolean *property* by means of the "-> [...]" label matching

operator of XTL. If the matching succeeds, then the b_1 statement (of type action) is executed. The static semantics of XTL ensures that all data variables declared in the *pattern* are visible inside the expression contained in the "if" branch, and therefore they can be used within b_1. If no successful match was encountered upon the end of the iteration (i.e., the value of the internal variable returned by the "for" expression is false), then the b_2 statement is executed.

The translation of a SPARQL query is compositional: each relational operator has a specific XTL iteration macro, and a query containing nested relational operators yields an appropriate nesting of the corresponding XTL iteration macros. Table 1 shows a simplified version of the translation rules for the main relational operators of SPARQL, by considering that all operands are BGPs.

Table 1. (Simplified) translation rules of SPARQL operators into XTL

Op.	SPARQL	XTL
BGP	$\{?x\ ?y\ ?z\}$	BGP ($?x$:string $?y$:string $?z$:string, true, b_1, b_2)
	$\{c\ ?y\ ?z\}$	BGP ($!c$ $?y$:string $?z$:string, true, b_1, b_2)
FILTER	$\{?x\ ?y\ ?z.$ FILTER $f(?x,?y,?z)\}$	BGP ($?x$:string $?y$:string $?z$:string, $f(x,y,z)$, b_1, b_2)
JOIN	$\{?x\ ?y\ ?z.$ $!z\ ?t\ ?u\}$	BGP ($?x$:string $?y$:string $?z$:string, true, BGP ($!z$ $?t$:string $?u$:string, true, b_1, nop), nop)
LJOIN	$\{?x\ ?y\ ?z.$ OPTIONAL$\{?z\ ?t\ ?u\}\}$	BGP ($?x$:string $?y$:string $?z$:string, true, BGP ($!z$ $?t$:string $?u$:string, true, b_1, b_2), nop)
UNION	$\{\{?x\ ?y\ ?z\}$ UNION $\{?t\ ?u\ ?v\}\}$	BGP ($?x$:string $?y$:string $?z$:string, true, b_1, b_2) fby BGP ($?t$:string $?u$:string $?v$:string, true, b_1, b_2)

The two examples of BGPs show how variable declarations and constants are translated into XTL label patterns using the "?" and the "!" notations, respectively. Here we consider that all fields are of string type, but XTL can recognize also label fields of other types (booleans, integers, reals, etc.). When the BGP is equipped with a FILTER clause, the boolean expression $f(?x,?y,?z)$ is passed as the *property* parameter to the XTL iteration macro. The JOIN operator yields a nested iteration, the invocation of the BGP iteration macro for the second triple being passed as body b_1 to the invocation of the BGP macro for the first iteration. The field z_1 on which the join takes place, which was extracted by the first BGP macro, is used as a constant value in the second BGP macro. The LJOIN (left join) operator is translated similarly, except that the body b_2 is not set to the empty action nop, but consists in printing empty slots instead of the fields of the second triple. The UNION operator is translated as the sequential composition of the two BGP iteration macros.

Finally, the iteration bodies b_1, b_2 are derived from the SELECT projection operator present at the top of each SPARQL query, which indicates the list of

variables to be printed as the result of the query. This is illustrated below by the translation of an LJOIN query in XTL:

SPARQL	SELECT $?x$ $?t$ WHERE { $?x$ $?y$ $?z$. OPTIONAL { $?z$ $?t$ $?u$ } }
XTL	BGP ($?x$:string $?y$:string $?z$:string, true, BGP (!z $?t$:string $?u$:string, true, print (x, t), print (x, " ")), nop)

The body b_1 of the second BGP iteration macro prints the values of variables x and t assigned at each iteration, whereas the body b_2 prints the value of x and an empty slot corresponding to the situation when no RDF triple matches the BGP specified by the OPTIONAL clause.

Optimizations. The XTL iteration macro shown in Figure 1 can be optimized whenever the subject or the object field in *pattern* has a fixed value (either a constant, or a variable extracted by an enclosing iteration macro). In this case, the iteration is no more performed on all edges E, but only on those edges going out of the subject node or coming to the object node. XTL provides specialized operators for performing these iterations efficiently (originally devised for the evaluation of modal logic operators), by translating them in terms of the iteration operators provided by the BCG libraries.

3.3 Adding temporal logic properties

Once a SPARQL query is translated into an XTL expression, it is very easy to extend it with structural properties on the RDF graph, expressed, e.g., as temporal logic formulas. The CADP toolbox provides XTL libraries [27] encoding the operators (extended with data) of several branching-time logics, such as HML [17], ACTL [28], and fragments of the modal μ-calculus [22].

The basic idea is to constrain further the SPARQL query by associating temporal properties to the subjects and objects occurring in BGPs in a way similar to the FILTER clause. To this purpose, we define the additional clauses S-PROP and O-PROP, which associate temporal properties to subjects and objects, respectively. A subject or object constrained by the S-PROP or O-PROP clause can be part of the query solution iff its corresponding node satisfies the temporal logic property evaluated on the RDF graph. The syntactic extension of the BGPs of SPARQL with the S-PROP and O-PROP clauses and its translation in XTL are shown below.

SPARQL	{ $?s$ $?p$ $?o$. S-PROP P . O-PROP Q }
XTL	BGP ($?s$:string $?p$:string $?o$:string, (source (E) among P) and (target (E) among Q), b_1, b_2)

The S-PROP P clause amounts to check that the subject node (i.e., the source node of the current triple E) belongs to the set of nodes produced by evaluating the temporal formula P. The O-PROP Q clause has a symmetric interpretation in terms of the target node. P and Q are XTL expressions of type stateset (i.e.,

they denote sets of nodes in the RDF graph) that are syntactically copied in the *property* parameter of the XTL iteration macro.

The data handling features of XTL enable to go beyond classical temporal logics, for instance by allowing the formula P to use the data values extracted by the BGP and stored in the s, p, o variables. For example, the extended BGP below expresses the existence of a cycle of unknown length going from the object of a triple back to the subject of the same triple:

$$\{\ ?s\ ?p\ ?o\ .\ \mathsf{O\text{-}PROP\ EF\_A}\ (\mathsf{true},\ \mathsf{Dia}\ (\mathsf{EVAL\_A}\ (A\ \_\ \_\ !s),\ \mathsf{true}))\ \}$$

The XTL formula in the O-PROP clause combines the potentiality operator EF_α of ACTL, which states the existence of a sequence of 0 or more transitions labeled by actions satisfying α (here α is true, meaning that we impose no constraint on transition labels) and the HML diamond modality $\langle A\ \_\ \_\ !s\rangle\mathsf{true}$, which states the existence of a transition going out from the current state (i.e., the last state of the sequence matched by the EF operator) and labeled by a triple with object s (the two other fields are ignored, being matched by the wildcards '_'). Each temporal or modal operator is implemented by an XTL function (such as EF_A and Dia) returning the set of graph nodes satisfying it. The $\mathsf{EVAL\_A}(a)$ macro operator stands for $\{\ L$:label where L -> [a] $\}$, the set of transition labels satisfying a pattern a.

The inevitability operator AF_α of ACTL allows the specification of branching-time properties characterizing subtrees of the RDF graph. For example, the extended BGP below matches all triples for which all sequences going out of their subject node lead, after a finite number of steps, to an object v:

$$\{\ ?s\ ?p\ ?o\ .\ \mathsf{S\text{-}PROP\ AF\_A}\ (\mathsf{true},\ \mathsf{Dia}\ (\mathsf{EVAL\_A}\ (A\ \_\ \_\ !v),\ \mathsf{true}))\ \}$$

Other kinds of temporal properties can be specified by using various operators within the S-PROP and O-PROP clauses. For instance, the fixed point operators of modal μ-calculus can be used to encode sequences matching regular expressions or complex cycles, which are expressible within the μ-calculus fragments of alternation 1 and 2, respectively [13].

4 Performance Measures

We carried out several experiments in order to study the behaviour of our BCG and XTL-based query evaluation approach w.r.t. dedicated SPARQL evaluation engines. As candidate for comparison, we chose the ARQ engine used by the JENA Semantic Web framework for JAVA developed by HP Labs[7]. To obtain RDF graphs of increasing size, we used the LUBM benchmark[8] [16], which enables to generate randomly new ontologies in OWL starting from a basic ontology representing a university. Given a number N of universities, LUBM generates N

[7] http://jena.sourceforge.net
[8] http://swat.cse.lehigh.edu/projects/lubm

Owl files containing random variations of the basic ontology. These N files were merged and converted using the Cwm tool[9] into a single Rdf file in the N3 format, which was subsequently encoded in the Bcg format. Table 2 gives the numbers of nodes and edges of the Rdf graphs obtained for increasing values of N and compares the sizes of the corresponding files. The Bcg format is almost two times more compact than the N3 format.

Table 2. Sizes of the Rdf graphs used in query evaluation experiments

N	Graph size		N3 size	Bcg size	Ratio
	nodes	edges	(MB)	(MB)	(%)
1	26 476	102 741	6.59	3.81	57.8
3	84 720	346 813	22.27	12.72	57.1
9	278 456	1 159 196	74.50	42.39	56.8
27	881 828	3 687 700	238.61	136.72	57.2

On each Rdf graph, we checked several Sparql queries involving various algebraic operators, some of them being based on the example queries contained in the Lubm benchmark. All experiments were performed on a 2.2 GHz Cpu, 1 GByte machine running Linux. The results are shown in Figure 2. The first three pictures (a), (b), (c) compare the times of evaluating requests made of Bgps, JOIN, and UNION operators by using Arq on the N3 files and Xtl on the Bcg files. For large files, we observe that Arq becomes increasingly faster than Xtl. However, the quasi-totality of the evaluation time using Xtl is taken by the initial phase of loading the necessary information from the Bcg file into memory, illustrated in picture (d) by evaluating an empty request. By comparing picture (d) with pictures (a), (b), and (c), we see that the time taken by Xtl for the pure evaluation of queries is comparable with the time taken by Arq.

We can therefore take advantage of the fact that we can group several queries into the same Xtl program, which enables to perform the loading a single time. The effect of grouping is illustrated by picture (e), which compares the time of evaluating all Xtl queries (grouped into a single program) and the sum of the times taken by Arq to evaluate each individual query. We notice a decrease of the difference in time w.r.t. Arq when the number of queries in the group increases, the two tools becoming equally efficient when the Xtl file contains about 10 requests.

Finally, Picture (f) shows the memory consumption, which is almost independent from the type of query. Xtl is less memory-consuming than Arq an all examples, the two tools exhibiting a quasi-linear variation of the memory with the size of the Rdf files.

[9] `http://www.w3.org/2000/10/swap/doc/cwm`

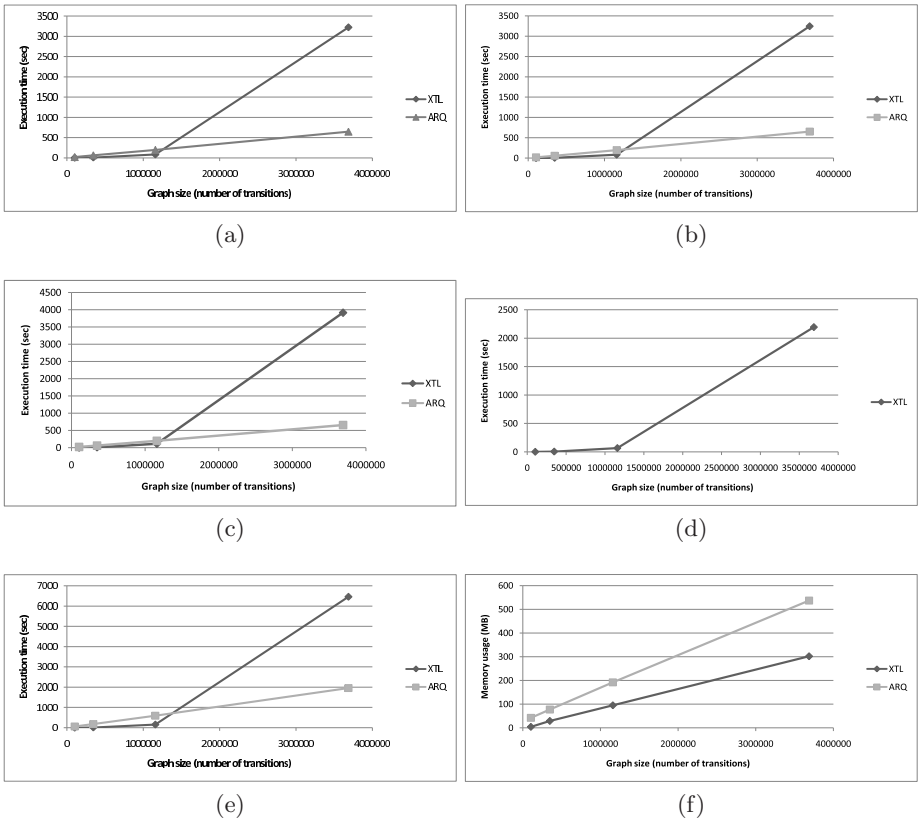

Fig. 2. Performance measures using ARQ on N3 files and XTL on BCG files

5 Conclusion and Future Work

We proposed an approach for evaluating queries on large RDF graphs by reusing the verification technology available from the domain of concurrent systems. Our approach involves two aspects: an encoding of RDF graphs as state spaces in the BCG format used by the CADP verification toolbox, and a translation of SPARQL queries in the XTL language dedicated to the exploration of BCG graphs. This provides a natural way of extending SPARQL with the possibility of evaluating temporal logic properties characterizing sequences, subtrees, and/or cycles in the RDF graph, and more generally with the ability to perform arbitrary graph explorations described in XTL.

The experiments we carried out on medium and large RDF graphs have shown that the performances obtained by using BCG and XTL are comparable with those of the ARQ evaluation engine dedicated to SPARQL. The increased expressiveness brought by the temporal logic operators available in the XTL libraries of CADP goes beyond existing work, which mainly focused on extending SPARQL with the detection of sequences matching regular expressions.

Our work can be continued on several directions. First, the translation to XTL can be extended in order to cover the whole SPARQL language, and not only the basic relational algebraic operators we shown in Section 3.2. Second, the translation should be further automated and experimented on other examples of domain-specific RDF graphs. Finally, it would be interesting to study the classes of temporal logic properties that are most useful for analyzing RDF graphs.

Acknowledgements

We are grateful to Jérôme Euzenat (INRIA / EXMO project-team) for useful discussions and for providing numerous sources of RDF graphs.

References

1. Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.
2. Faisal Alkhateeb, Jean-François Baget, and Jérôme Euzenat. Extending sparql with regular expression patterns (for querying rdf). *Web Semantics*, 7(2):57–73, 2009.
3. Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon. Xml path language (xpath) 2.0. W3c recommendation, World Wide Web Consortium, 2007.
4. T. Berners-Lee. A readable language for data on the web. W3c recommendation, World Wide Web Consortium, 2006.
5. Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American Magazine*, 2001.
6. Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. Xquery 1.0: An xml query language. W3c recommendation, World Wide Web Consortium, 2007.

7. Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 2000.
8. Mariano P. Consens and Alberto O. Mendelzon. Graphlog: A visual formalism for real life recursion. In *Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems PODS'90 (Nashville, Tennessee, USA)*, pages 404–416. ACM Computer Society Press, 1990.
9. Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. *ACM SIGMOD Record*, 16(3):323–330, 1987.
10. Luca de Alfaro. Model checking the world wide web. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Proceedings of the 13th International Conference on Computer Aided Verification CAV'01 (Paris, France)*, number 2102, pages 337–349, July 2001.
11. Mike Dean and Guus Schreiber. Owl web ontology language reference. W3c recommendation, World Wide Web Consortium, 2004.
12. Landon T. Detwiler, Dan Suciu, and James F. Brinkley. Regular paths in sparql: Querying the nci thesaurus. In *Proceedings of the Annual Symposium of the American Medical Informatics Association AMIA'08*, 2008.
13. E. Allen Emerson and C-L. Lei. Efficient model checking in fragments of the propositional mu-calculus. In *Proceedings of the 1st International Symposium on Logic in Computer Science LICS'86*, pages 267–278, 1986.
14. Xiang Fu, Tevfik Bultan, and Jianwen Su. Model checking xml manipulating software. In George S. Avrunin and Gregg Rothermel, editors, *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis ISSTA'2004 (Boston, Massachusetts, USA)*, pages 252–262. ACM Computer Society Press, July 2004.
15. Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. Cadp 2006: A toolbox for the construction and analysis of distributed processes. In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification CAV'2007 (Berlin, Germany)*, volume 4590, pages 158–163, July 2007.
16. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics*, 3(2–3):158–182, 2005.
17. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32:137–161, 1985.
18. ISO/IEC. Lotos — a formal description technique based on the temporal ordering of observational behaviour. International Standard 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1989.
19. ISO/IEC. Information technology – database language – sql. International Standard 9075, International Organization for Standardization, 2008.
20. G. Klyne and J. J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. W3c recommendation, World Wide Web Consortium, 2004.
21. Krys J. Kochut and Maciej Janik. Sparqler: Extended sparql for semantic association discovery. In *The Semantic Web: Research and Applications*, 2007.
22. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
23. Leon J. Osterweil Kurt M. Olender. Cecil: A sequencing constraint language for automatic static analysis generation. *IEEE Transactions of Software Engineering*, 16(3):268–280, 1990.

24. Frédéric Lang. Exp.open 2.0: A flexible tool integrating partial order, compositional, and on-the-fly verification methods. In Jaco van de Pol, Judi Romijn, and Graeme Smith, editors, *Proceedings of the 5th International Conference on Integrated Formal Methods IFM'2005 (Eindhoven, The Netherlands)*, volume 3771, pages 70–88, November 2005. Full version available as INRIA Research Report RR-5673.
25. Jeff Magee and Jeff Kramer. *Concurrency: State Models and Java Programs*. Wiley, 2006.
26. Alain J. Martin. Compiling communicating processes into delay-insensitive VLSI circuits. *Distributed Computing*, 1(4):226–234, 1986.
27. Radu Mateescu and Hubert Garavel. Xtl: A meta-language and tool for temporal logic model-checking. In Tiziana Margaria, editor, *Proceedings of the International Workshop on Software Tools for Technology Transfer STTT'98 (Aalborg, Denmark)*, pages 33–42. BRICS, 1998.
28. R. De Nicola and F. W. Vaandrager. *Action versus State Based Logics for Transition Systems*.
29. E. Prud'hommeaux and A. Seaborne. Sparql query language for rdf. W3c recommendation, World Wide Web Consortium, 2008.
30. C.M. Sperberg-McQueen E. Maler T. Bray, J. Paoli and F. Yergeau. Extensible markup language (xml) 1.0 (fifth edition). W3c recommendation, World Wide Web Consortium, 2008.
31. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the International Joint Conference on Automated Reasoning IJCAR'06 (Seattle, WA, USA)*, volume 4130, pages 292–297, August 2006.