Mapping SPARQL Query to temporal logic query based on NμSMV Model Checker to Query Semantic Graphs

Mahdi Gueffaz, Sylvain Rampacek, Christophe Nicolle

Mapping SPARQL Query to temporal logic query based on NµSMV Model Checker to Query Semantic Graphs

Mahdi Gueffaz[1], Sylvain Rampacek[1], Christophe Nicolle[1],
1 LE2I, UMR CNRS 5158
University of Bourgogne,
21000 Dijon, France
{Mahdi.Gueffaz, Sylvain.Rampacek, Christophe.Nicolle}@u-bourgogne.fr

ABSTRACT

The RDF (W3C standard for meta-modeling) language is the most frequently used to represent the semantic graphs. This paper presents a new research combining different fields that are: the semantic web and the model checking. We developed a tool, RDF2NµSMV, which converts RDF graphs into NµSMV language. This conversion aims checking the semantic graphs that have numerous errors of interpretation with the model checker NµSMV in order to verify the consistency of the data. The SPARQL query language is the standard for querying the semantic graph but have a lot of limitations. To this purpose, we define a translation from the SPARQL query language into the temporal logic query language. This language is a graph manipulation language implemented in our toolbox. This translation makes it possible to extend the expressive power of SPARQL naturally by adding temporal logic formulas characterizing sequences, trees, or general sub-graphs of the RDF graph. Our approach exhibits a performance comparable to dedicated SPARQL query evaluation engines, as illustrated by experiments on large RDF graphs. We developed the STL Resolver tool to resolve the temporal logic query. This tool is based on the model checker NµSMV algorithms.

KEYWORDS

Semantic graph, RDF, Model Checking, Temporal logic, NµSMV, Query checking, SPARQL, temporal logic query.

1 INTRODUCTION

The increasing development of networks and especially the internet has greatly developed the heterogeneous gap between information systems. In glancing over the studies about interoperability of disparate information systems, we discover that all works tend to the resolution of semantic heterogeneity problems. The W3C[1] suggests norms to represent the semantic by ontology. Ontology is becoming an inescapable support for information system's interoperability and particularly in the Semantic. Literature now generally agrees on the Gruber's terms to define an ontology: explicit specification of a shared conceptualization of a domain [1]. The physical structure of ontology is a combination of concepts, properties and relationships. This combination is also called a semantic graph.

Several languages have been developed in the context of Semantic Web, and most of these languages use

[1] World Wide Web Consortium

XML2 as syntax [2]. The OWL3 [3] and RDF4 [4] are the most important languages of the semantic web, they are based on XML. OWL allows representing the ontology, and it offers large-capacity machines performing web content. RDF enhances the ease of automatic processing of Web resources. The RDF (Resource Description Framework) is the first W3C standard for enriching resources on the web with detailed descriptions. The descriptions may be characteristics of resources, such as author or content of a website. These descriptions are metadata. Enriching the Web with metadata allows the development of so-called Semantic Web [5]. The RDF is also used to represent a semantic graph corresponding to a specific knowledge modeling. In this paper, we propose a new way using formal verification, which consists in the transformation of semantic graphs into a model and verifying them with a Model Checker [6].

We developed two tools. The first one called "RDF2NµSMV" that transforms semantic graphs into a model represented in NµSMV [7] language. After this transformation, NµSMV verifies the correctness of the model written in NµSMV language with temporal logic in order to verify the consistency of the data described in the model of the huge semantic graphs. The second tool, called "STL RESOLVOR", aims resolving the queries destined to the model of the semantic graph. This query was introduced the first time by William Chan in his innovative work [8]. These requests are not used to verify the model representing the RDF graph, but rather to recognize it.

[2] eXtensible Markup Language
[3] Web Ontology Language
[4] Resource Description Framework

Our primary goal in this paper is to define a powerful and expressive query language for semantic graphs and to align with SPARQL [9], in order to improve the interoperability of applications on the Semantic Web. The other rather competing goal is to keep the query language simple enough that it can be easily built. To satisfy these requirements, we define a new query language that uses the operators of the temporal logic.

The rest of this paper is organized as follows. In Section 2 we present an overview of the semantic graphs, especially the structure of the RDF graphs and the model checking. Then, in section 3, we describe the SPARQL query. Section 4 presents, the temporal logic and the query checking. Section 5 refers to the mapping of the semantic graphs into models, section 6 to the transformation of SPARQL query to temporal logic query. Section 7 defines the functionalities of the STL Resolver tool, and we present some benchmarks in section 9. Finally, we end with a conclusion.

2 AN OVERVIEW OF SEMANTIC GRAPHS AND MODEL CHECKING

Semantic graphs - The RDF is also used to represent semantic graphs corresponding to a specific knowledge modeling. It is a language developed by the W3C to bring a semantic layer to the Web [10]. It allows the connection of the Web resources using directed labeled edges. The structure of the RDF documents is a complex directed labeled graph. An RDF document is a set of triples <subject, predicate, object> as shown in the Figure1. In addition, the predicate (also called property) connects the subject (resource) to the object

(value). Thus, the subject and the object are nodes of the graph connected by an edge directed from the subject towards the object. The nodes and the edges belong to the "resource" types. A resource is identified by an URI5 [11, 12].

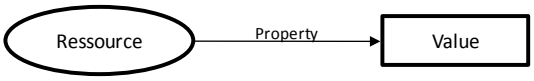

Figure 1. RDF triplet.

The declarations can also be represented as a graph, the nodes as resources and values, and the arcs as properties. The resources are represented in the graph by circles; the properties are represented by directed arcs and the values by a box (a rectangle). Values can be resources if they are described by additional properties. For example, when a value is a resource in another triplet, the value is represented by a circle.

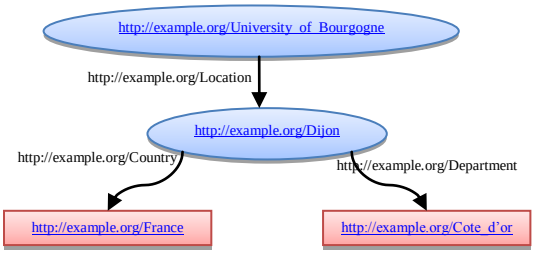

Figure 2. Example of partial RDF graph.

The RDF graph in the Figure 2 defines a node "University of Bourgogne" located at "Dijon", having as country "France" and as a department named "Cote d'Or". RDF documents can be written in various syntaxes, e.g., N3 [13], N-Triple [14], and RDF/XML. Below, we present the RDF\XML document corresponding to Figure 2.

```
<rdf:Description
rdf:about="http://example.org/univer
sity of Bourgogne">
  <ex:Location>
      <rdf:Description
rdf:about="http://example.org/Dijon"
>
      <ex:Country>
France</ex:Country>
      <ex:Department>Cote
d'or</ex:Department>
      </rdf:Description>
   </ex:Location>
</rdf:Description>
```

Model checking - The model checking [15] described in Figure 3 is a verification technique that explores all possible system states in a brute-force manner. Similar to a computer chess program that checks all possible moves, a model checker, the software tool that performs the model checking, examines all possible system scenarios in an organized manner. In this way, it can be shown that a given system model truly satisfies a certain property. Even the subtle errors that remain undiscovered using emulation, testing and simulation can potentially be revealed using model checking.

To make a rigorous verification possible, properties should be described in a precise unambiguous way. It is the temporal logic that is used in order to express these properties. The temporal logic is a form of modal logic that is appropriate to specify relevant properties of the systems. It is basically an extension of traditional propositional logic with operators that refer to the behavior of systems over time.

[5] Uniform Resource Identifier

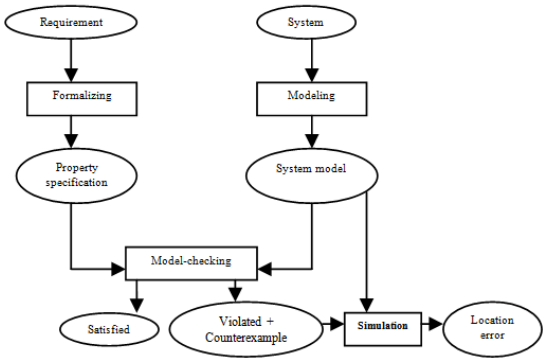

Figure 3. Model Checking approach**.**

The following algorithm explains the way that the model checking works. First, we put in the stack all the properties expressed in the temporal logic. All of them are verified one by one in the model and if a property does not satisfy the model, it is whether the model or the property that we must refine. In case of a memory overflow, the model must be reduced. Whereas formal verification techniques such as simulation and model checking are based on model description from which all possible system states can be generated, the test, that is a type of verification technique, is unvarying applicable in cases where it is hard or even impossible to obtain a system model.

3 THE SPARQL QUERY

SPARQL [9] is a query language for querying metadata and extraction data from an RDF graph or, more precisely a query language for RDF triples.

In SPARQL, different query forms are available:

- **Select**: return the value of variables, which may be bound by a matching query pattern.
- **Ask**: return true if a given query match and false if not.
- **Construct**: return an RDF graph by substituting the values in given templates.
- **Describe**: return an RDF graph which defines the matching resource.

The Select form is the most used. In this article, we showed only the SPARQL query with the select form. A basic SPARQL query has the following form:

Select ?variable1, ?variable2,…
Where {pattern1.pattern2. …}

Where each pattern consists of subject, predicate, object, and each of these is either a variable or a literal. The query model is query-by-example style: the query specifies the known literals and leaves the unknowns as variables. Variables can occur in multiple patterns and thus imply joins. The query processor needs to find all possible variable bindings that satisfy the given patterns and return the bindings from the projection clause to the application. Note that not all variables are necessarily bound (e.g., if a variable only occurs in the projection and not in a pattern), which results in NULL values.

Relational algebra [16] is introduced to facilitate the mapping of SPARQL query to the applications in temporal logic. We define the operators in RDF relations.

2.1 Selection

Selection (σ), sometimes also called restriction, is an unary operator that selects only those tuples of a relation for which a propositional formula holds.

The propositions are assumed to have the expressivity of SPARQL Filter expressions.

2.1 Projection

The projection operator (π) restricts a relation to a subset of its attributes.

2.1 Inner Join and Left Outer Join

The inner join ($\bowtie$) joins two relations on their shared attributes. A$\bowtie$B contains all combinations of a tuple from A and a tuple from B, minus those where the shared attributes are not equal.

The left outer join ($⟕$) additionally contains all those tuples from the first relation that have no matching tuple in the second.

2.1 Union

The union ($\cup$) of two relations A and B is the set of union of the tuples of A and B. unlike in regular relational algebra, the headings of A and B do not need to be identical.

4 TEMPORAL LOGIC AND THE QUERY CHECKING

The concepts of temporal logic used for the first time by Pnueli [17] in the specification of formal properties are fairly easy to use. The operators are very close in terms of natural language. The formalization in temporal logic is simple enough although this apparent simplicity therefore requires significant expertise. Temporal logic allows representing and reasoning about certain properties of the system, so it is well-suited for the systems verification. There are two main temporal logics that are linear time and branching time. In linear time temporal logic, each execution of the system is independently analyzed. In this case, a system satisfies a formula f, if f holds along every execution. The branching time combines all possible executions of the system into a single tree. Each path in the tree is a possible representation of the system execution [18].

- **Linear Temporal Logic** or LTL allow representing the behavior of reactive systems using properties that describe the system in which time proceeds linearly. Clearly, we specify the expected behavior of a system, by specifying the only possible future as a sequence of actions that follow, LTL uses for that temporal operators: X (Next), F (Finally), G (Always), U (Until).
- **Computation Tree Logic** or CTL suggests several possible futures from a system state rather than having a linear view of the system considered. The operators of CTL are obtained by adding A (for any execution) or E (there is an execution) before the operators of linear temporal logic that are: AX φ (all successor states immediately satisfy φ), EX φ (there is an execution whose next state satisfies φ), AF φ (for any execution, there is a state where φ is true), EF φ (there is an execution, leading to a true state φ), AG φ (for any execution, φ is always true), EGϕ (there is an execution, where φ is always true), AφUψ (for any execution φ is true until ψ is true), EφUψ (there is an execution in which φ is true until ψ is true).

The Model-Checking was proposed as a verification technique, it is valuable for understanding the model: The user

formulates a hypothesis of the system behavior, expressed as a formula in temporal logic, and tries to use the Model Checker to validate this hypothesis. This use of the model checking has not been sufficiently emphasized in the literature. So, in order to help the user to understand the system behavior, Chan [8] introduced the queries in temporal logic and used a technique similar to the Model-Checking to determine the temporal properties in contrast to simply verifying them.

The query checking is an extension of the Model checking who, instead of asking "does the system satisfy a temporal logic formula φ", allows us to ask "for what value of X does the system satisfy $\varphi(X)$?" Here, X is not a system parameter, but a property setting, that we seek to satisfy. These queries do not allow the verification of a specific property of the model, but they allow the examination of the model by questioning it. The technique of query-checking can also be used to provide more information to the user in the Model Checking.

The query checking allows the writing of temporal logic formulas easily and can therefore verify any properties on both the data contained in the graph and the structure of the data.

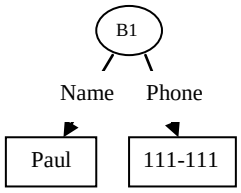

Figure 4. The query execution time.

In the graph of the Figure 4, we can see that there are two RDF triples (B1, name, Paul) and (B1, Phone, 111-111). The following SPARQL query:

```
SELECT ?x
WHERE {
        ?x Name
"Paul"}
```

whose representation in relational algebra is:

$$\pi_{?x} \rightarrow \sigma_{\substack{?predicate=Name \\ ?object=Paul}} \rightarrow Triples$$

looks for a subject ?x which has the predicate "Name" and an object "Paul" (?x, name, Paul). The equivalent of the previous query in query using the temporal logic operators is:

Finally (?x → Next "Paul") (1)

This temporal logic query looks for the same subject ?x, as defined in the SPARQL query above.

5 THE RDF GRAPH TRANSFORMATION

This section speaks about our approach, which consists in the transformation of semantic graphs into a model in order to verify them with the model-checker. For this, we developed "RDF2NμSMV" tool that transforms a semantic graph into NμSMV [7] language for the Model-checker NμSMV.

NμSMV is the amelioration of SMV model checker; it works on the sample principles as SMV. NμSMV verifies the properties in both linear time logic and computation tree logic.

The RDF graphs considered here are represented as XML verbose files, in which the information is not stored hierarchically (so-called graph point of view). On the one hand, these RDF graphs are not necessarily connected, meaning they may have no root vertex from which all the other vertices are reachable. On the other hand, the NμSMV language manipulated by the

verification tools of NµSMV always has a root vertex, which corresponds to the initial state of the system whose behavior is represented by the NµSMV language. The RDF graph transformation into NµSMV language is articulated in three steps: exploring the RDF graph, determining a root vertex and final step, generating the Model of the RDF graph. This final step is divided into three sub-steps. The first and the second steps consist in generating two tables (triples table and correspondence table). Firstly, the table of all triples is built by exploring the entire graph. The graph traversal algorithms go through the RDF graph and create a table consisting of resources, properties and values. In the source RDF graph, the resource is a vertex. The property represents the edge, and the value is the successor vertex corresponding to the edge of the vertex. The table of triples of RDF graph is useful for the next sub-step.

Secondly, a correspondence table is generated. To build the table of correspondence, the algorithm associates an identifier for each resource, property and value.

The last step consists in producing from these tables the model writing in NµSMV language for the Model checker NµSMV [19]. This transformation is very useful to qualify RDF graph that their construction is a source of numerous errors of interpretation [20].

Example of Transformation. Consider the graph RDF of the figure below that represent a. From this graph, we generate a table of triplets. This table is composed of RDF triples i.e. "*resource - property - value*".

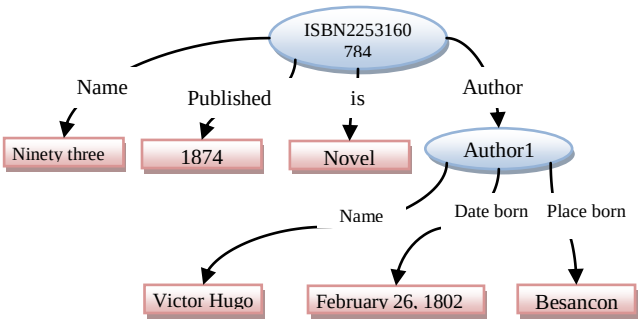

Figure 5. RDF graph

Table 1

Resource	Property	Value
ISBN2253160784	Name	Ninety three
ISBN2253160784	Published	1874
ISBN2253160784	Is	Novel
ISBN2253160784	Author	Author1
Author1	Name	Victor Hugo
Author1	Date born	February 26, 1802
Author1	Place born	Besançon

The next step is to generate a table of association. This table is represented by two tables. The table 2 contains all the state of the RDF graph and its messages. The table 3 gives for each state and message the next state. In our example, the tables are represented in the tables below. We notice that the names of the states are changed comparing with the RDF graph. All the black space is replaced by the symbol "_".

Table 2

State	Message
ISBN2253160784	Name, Published, Is, Author
Author1	Name, Date_born, Place_born
Ninty_three	Nil
1874	Nil
Novel	Nil
Victor Hugo	Nil
February_26_1802	Nil
Besançon	Nil

Table 3

State	Message	Next state
ISBN2253160784	Name	Ninty_three
ISBN2253160784	Published	1874
ISBN2253160784	Is	Novel
ISBN2253160784	Author	Author1
Author1	Name	Victor_Hugo
Author1	Date_born	February_26_ 1802
Author1	Place_born	Besançon

The last step of processing is the generation of the NµSMV language that uses the previous tables. You can see below the representation of the RDF graph in figure 5 written in NµSMV language.

```
/* Declaration of all states and
messages that represent the node
and property of the RDF graph
*/

MODULE main

VAR

state : { ISBN2253160784,
Ninety_three, 1874, Novel,
Author1, Victor_Hugo,
February_26_1802, Besançon};
msg : {Name, published, is,
author, Date_born, Place_born};

/*  Initialisation  of  the
variable state by the root node
of  the  RDF  graph  and  the
variable message by nop
*/

INIT
state = ISBN2253160784;
INIT
msg = nop;

ASSIGN

/* For each state we give its
messages
*/

next(msg):=
case
state= ISBN2253160784 : {Name,
published, is, Author};
state= Ninety_three: nop;
state= 1874 : nop;
state= Novel : nop;
state= Author1: {Name,
Date_born, Place_born};
state= Victor_Hugo: nop;
state= February_26_1802: nop;
state= Besançon : nop;
TRUE : msg;
esac;

/* For each state and message we
give the next state
*/

next(state) :=
case
state= ISBN2253160784 & msg=Name
: Ninety_three;
state= ISBN2253160784 &
msg=published : 1874;
state= ISBN2253160784 & msg=is :
Novel;
state= ISBN2253160784 &
msg=Author : Author1;
state= Author1 & msg=Name :
Victor_Hugo;
state= Author1 & msg=Date_born :
February_26_1802;
state= Author1 & msg=Place_born
: Besançon;
TRUE : state;
esac;
```

We can use this graph represented in NµSMV language to verify the data consistency by using the temporal logic formula.

Example of verification. We use also the same RDF graph of the figure 5. The NµSMV model checker use both linear temporal logic and computation tree logic (see section 4). In this example we use only the computation tree logic. The temporal logic formulas are summarized in the table below.

Table 4.

Temporal logic	Temporal logic NµSMV
Always (ISBN2253160784 → next Ninety-three and ISBN2253160784 → next Novel)	**AF** (state = ISBN2253160784 → E**X** state= Ninety-three **&** state = ISBN2253160784 → E**X** state= Novel)
Always (ISBN2253160784 → next next Victor Hugo)	**AF** (state= ISBN2253160784 → **EX** E**X** state= Victor_Hugo)

The table 4 is divided into two columns. The first one described the temporal logic formula, but the second one described the formula with the operators of the NµSMV model checker. The first formula checks if the Ninety-three is a Novel with the ISBN[6] "ISBN2253160784" and the second checks if the author of the ISBN "ISBN2253160784" is "Victor Hugo". We put all this formula in the bottom of the NµSMV file. The result of this formula is presented below.

```
C:\Program
Files\NuSMV\2.5.2\bin>NuSMV.exe
journal.smv
*** This is NuSMV 2.5.2
(compiled on Fri Oct 29 11:33:56
UTC 2010)
*** Enabled addons are: compass
*** For more information on
NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-
users@list.fbk.eu>.
*** Please report bugs to
<nusmv-users@fbk.eu>

*** Copyright (c) 2010,
Fondazione Bruno Kessler

*** This version of NuSMV is
linked to the CUDD library
version 2.4.1
*** Copyright (c) 1995-2004,
Regents of the University of
Colorado

*** This version of NuSMV is
linked to the MiniSat SAT
solver.
*** See
http://www.cs.chalmers.se/Cs/Res
earch/FormalMethods/MiniSat
*** Copyright (c) 2003-2005,
Niklas Een, Niklas Sorensson

-- specification AF (((state =
ISBN2253160784 & EX (EX state =
Ninety_three)) &
state = ISBN2253160784) & EX (EX
state = Novel))  is true
-- specification AF (state =
ISBN2253160784 & EX (EX (EX (EX
state = Victor_Hugo
))))  is true
```

6 SPARQL QUERY TO TEMPORAL LOGIC QUERY

This section gives an overview of SPARQL query transformation into Temporal Logic query. We focus on the SELECT form. To illustrate this section, we use the RDF example shown in Figure 6.

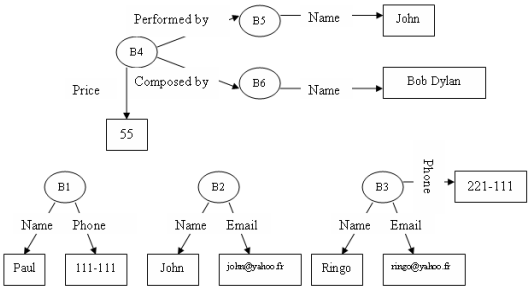

Figure 6. RDF graph for SPARQL query.

From the graph in Figure 6, we construct SPARQL queries and their query equivalent in temporal logic. The SPARQL query below selects the subject with the variable ?x which has Paul as object.

[6] International Standard Book Number

SPARQL	SELECT ?x WHERE { ?x nom "Paul"}
LT query	Finally (?x → Next "Paul")
Relational algebra	$\pi_{?x} \rightarrow \sigma_{?predicate=Name \wedge ?object=Paul} \rightarrow Triples$

The SPARQL query below selects the subject ?x which has the variable ?y as object, who, at its turn, has "Bob Dylan" as object.

SPARQL	SELECT ?x WHERE { ?x composed_by ?y. ?y name "Bob Dylan"}
LT query	Finally (?x → Next Next "Bob Dylan")

The purpose of an optional pattern is to supplement the solution with additional information. If the pattern within an OPTIONAL clause matches, the variables defined within that pattern are bound to one or to many solutions. If the pattern does not match, the solution remains unchanged. The SPARQL query represented below selects the subject ?x that has "Paul" and/or "paul@yahoo.fr" as object.

SPARQL	SELECT ?x WHERE { ?x name "Paul" OPTIONAL {?x email paul@yahoo.com"}}
LT query	Finally (?x → Next "Paul" ∧ Finally ?x → "paul@yahoo.com")

A SPARQL **FILTER** function can be added to a basic graph pattern in order to restrict the result according to Boolean conditions. The SPARQL query below selects the subject ?x which has a word that contains at least the letter P as object.

SPARQL	SELECT ?x WHERE { ?x name ?y FILTER regex (?y, "P") }
LT query	Finally (?x → Next *P*)

Relational algebra	$\pi_{?x} \rightarrow \sigma_{?predicate=Name \wedge ?object=?y \wedge regex(?y,P)} \rightarrow Triples$

The SPARQL query below selects the subjects ?x and ?y that have "John" and respectively "Paul" as objects.

SPARQL	SELECT ?x ?y WHERE { {?x name "John"} UNION { ?y name "Paul"}}
LT query	Finally (?x → Next "John" ∧ ?y → Next "Paul")

The SPARQL query below selects the objects ?x where "name" is the predicate.

SPARQL	SELECT ?x WHERE { ?y name ?x }
LT query	Finally (?y → Next ?x) where predicate=name/ ?x
Relational algebra	$\pi_{?x} \rightarrow \sigma_{?predicate=Name \wedge ?subject=?y} \rightarrow Triples$

The SPARQL query below selects the object ?y. This query represents a SPARQL's join.

SPARQL	SELECT ?y WHERE { ?x name ?y. ?z performed_by ?x. ?z composed_by ?p. ?p name "Bob Dylan"}
LT query	Finally (?z → Next Next ?y ∧ ?z → Next Next "Bob Dylan") where predicate=nom/ ?y

The transformation of SPARQL queries into queries using operators of temporal logic was based on the representation of SPARQL queries in the relational algebra seen above.

The advantage of temporal logic queries is their simplicity to write. That means that the temporal logic is closer to the natural language and in addition, one

of the great advantages of the temporal logic queries is that they are more expressive than the SPARQL queries using the temporal logic operators. [21]

We developed a tool called "SPARQL2LTQ" which aims to transform SPARQL queries into queries using operators of the temporal logic.

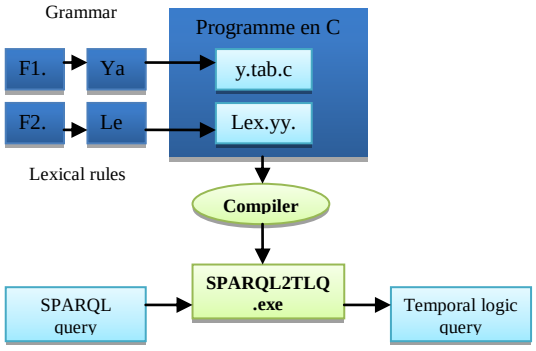

Figure 7. The architecture of the transformation tool "SPARQL2RLT".

For the development of this tool, we use LEX & YACC to decompose the SPARQL query in order to facilitate the processing. LEX is used to recognize the lexical entities and replace them with keywords that will be recognized in the grammar of the language defined in YACC; then YACC will recognize and respect the expressions and will verify if they belong to the grammar. LEX & YACC are two very powerful tools, facilitating the lexical and respectively the syntactic analysis, which represents two stages of compilation difficult to program.

In order to demonstrate the usefulness of temporal logic queries, we will illustrate an example. Here are two tables in a relational database, table 1 and table 2.

Table 1

Person	Name
id1	Alice
id3	Christophe
id2	Bob

Table 2

Name	Age
Alice	33
Bob	42
Christophe	15

```
SELECT ?x
WHERE {
id1 ex :Name ?z
?z ex :HasAge ?x
}
```

The SPARQL query above seeks the age of the person identified by id1, corresponding to "Alice" in our case.

To answer this SPARQL query we must first make a join between the two previous relational database tables. A join is used for joining two multi sets with a constraint. In our example, the constraint is the name, see table 3 below.

Table 3

Person	Name	Age
1	Alice	33
2	Bob	42
3	Christophe	15

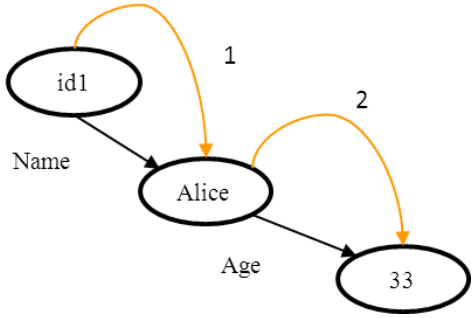

Figure 8. The RDF graph model showing the movement to be done to achieve results.

The equivalent of the previous SPARQL query in a query using operators of the temporal logic is as follows:

Finally (id → **next next** ?x)

In this query, in order to retrieve the age of the person identified by id1, one just moves two states (two "next" operators in our query represented by X) in the semantic graph model to find the result, as shown in Figure 8.

For results with temporal logic query one just moves around the graph with the operators of temporal logic. The new interrogation technique will allow us to avoid scanning the graph several times as with SPARQL queries.

7 THE STL RESOLVER

The ScaleSem toolbox contains a tool used to resolve queries in temporal logic that we have previously seen. This tool is called STL RESOLVOR, it takes as input the temporal logic query and the NµSMV graph representing the semantic graph.

The query checking is an extension of the Model checking. A temporal logic query is a formula with a missing propositional formula, designated by a placeholder ("?"). A solution to a temporal logic query is the set of all propositional formulas that satisfy the query, in our case the formulas are the states represented in the NµSMV graph.

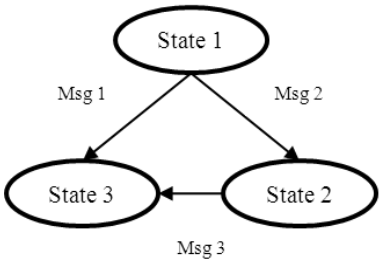

Figure 9. An example of NµSMV graph.

TABLE 4

	Query	Result
1	Finally (?x → X State 3)	?x ={State 1, State 2}
2	Finally (?x → X X state 3)	?x={State 1}

The table above gives some example of temporal logic queries and their results.

8 BENCHMARCK

We tested several RDF graphs on our tool "RDF2NµSMV", using a machine that runs on a processor with a capacity of 2.4 GHz and 4 GB of RAM, calculating the time of conversion as shown in Figure 10. Note that the RDF2NµSMV tool is faster in converting semantic graphs. We have almost 22 seconds for a graph of 53 MB size. The transformation tool follows a polynomial curve. In Figure 11, we see the size of the converted semantic graphs from RDF to NµSMV language.

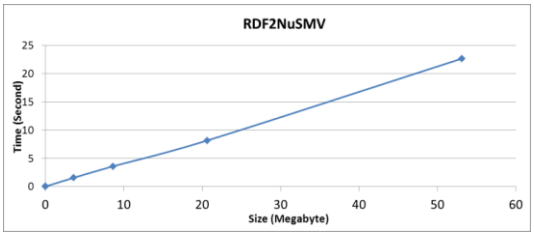

Figure 10. Time conversion of Semantic graphs.

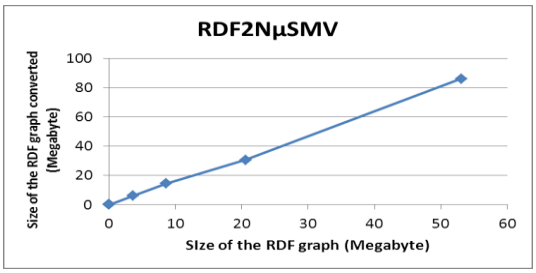

Figure 11. Size of the Models.

We calculate the time of transformation of the SPRAQL query into a query using the operators of the temporal logic with the SPARQL2TLQ tool. The graph of the Figure 12 shows that for 50 000 queries, we have over than 2 minutes and for 100 000 queries, we have 17 minutes. This transformation follows a polynomial curve. In Figure 13, we notice that the size of the queries in temporal logic is smaller than the size of the equivalent SPARQL queries.

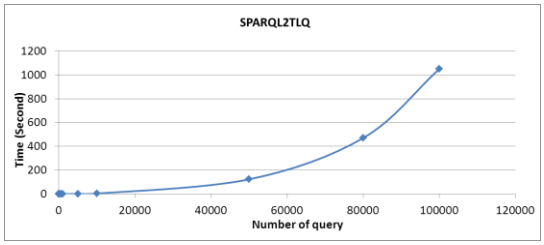

Figure 12. Time conversion of SPARQL query.

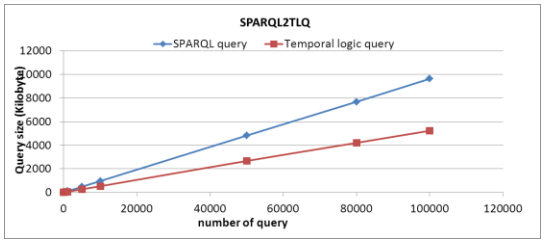

Figure 13. Comparison of size in both SPARQL query and temporal logic query.

9 CONCLUSION

This paper presents how to transform a semantic graph into a model for verification by using a powerful formal method, that is the "model checking." Knowing that the model checker does not understand the semantic graphs, we developed a tool called "RDF2NµSMV" to convert them into NµSMV graph in order to be verified with the temporal logics. This transformation is made for the purpose of classifying large semantic graphs in order to verify the consistency of the data from a different ontology. We notice the advantage of NµSMV, whose verification can be made with both linear time logic and computation tree logic formulas.

We also introduce a new tool called "STL RESOLVOR" that is used to find the solution of temporal logic queries to know better the model of the RDF graph used by the model checker NµSMV. These temporal logic queries are obtained from SPARQL query by using "SPARQL2RLT" tool. This paper described also this transformation.

We continue our research, understanding the SPARQL queries and trying to convert them into queries using the operators of the temporal logic. The goal of this transformation is to study a modern way of expressing a new possibility to explore the semantic graphs.

6 REFERENCES

1. Gruber, T. R.: Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology, later published in International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, November 1995, pp. 907-928. March 1993.
2. Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., Cowan, J.: Extensible Markup Language (XML) 1.1 (second edition) W3C recommendation, http://www.w3.org/TR/2006/REC-xml11-20060816/. (2006)
3. Bechhofer, S., van Harmelen, F., Hendler J., Horrocks, I., McGuinness, D., Patel-Schneijder, P., Andrea Stein, L., OWL Web Ontology Language Reference, World Wide Web Consortium (W3C), http://www.w3.org/TR/owl-ref/, (2004).
4. Becket, D., McBride, B.: RDF/ XML Syntax Specification (Revised). W3C recommendation. http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/. (2004)
5. Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. Scientific American. pp. 34–43. 2001.
6. Clarke, E.M. The birth of Model checking. 25 Years of Model Checking Lecture Notes in Computer Science, Volume 5000, pp. 1-26, 2008.
7. Cimatti, A., Clarke, E.M, Giunchiglia, F., Roveri, M. NuSMV: a new symbolic model checker, pp 410-425. 2000.
8. Chan, W. Temporal-logic queries. In Proceedings of Computer Aided Verification, LNCS 1855, pp 450-463. 2000.
9. Chebotko, C., Lu, S., Fotouhi, F. Semantics preserving SPARQL-to-SQL translation. Data & Knowledge Engineering, Elsevier. 2009.

10. Klyne, J. J. C. G.: Resource Description Framework (rdf): Concepts and abstract syntax. Tech. rep., W3C. (2004)
11. Bönström, V., Hinze, A., Schweppe, H.: Storing RDF as a graph. Latin American WWW conference, Santiago, Chile. (2003)
12. Berners-Lee, T. W3C recommendation. http://www.w3.org/DesignIssues/ HTTP-URI. 2007.
13. Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C recommendation, http://www.w3.org/TeamSubmission/n3/. (2008)
14. Becket, D., McBride, B.: RDF test cases. W3C Working draft. http://www.w3.org/TR/rdf-testcases/ (2004)
15. Katoen, J. P., 2002. The principal of Model Checking. University of Twente.
16. Cyganiak, R. A relational algebra for SPARQL. Digital Media Systems Laboratory, HP Laboratories Bristol. September 2005.
17. Pnueli, A. The temporal logic of programs. In proc. 18th IEEE Symp. Foundations of Computer Science (FOCS'77), Providence, RI, USA. Pages 46-57. 1977.
18. Mukund, M. Model Checking, Automated Verification of Computational Systems, pp. 667-681. 2009.
19. Gueffaz, M., Rampacek, S., Nicolle, C. ScaleSem: Evaluation of semantic graph based on Model Checking", Webist 2011- The 7th International Conference on Web Information Systems and Technologies, Noordwijkerhout, Hollande, May 2011.
20. Gueffaz, M., Rampacek, S., Nicolle, C. Qualifying Semantic Graphs Using Model Checking, 7th International Conference on Innovations in Information Technology (Innovations'11), Abu Dhabi, United Arab Emirates, April 2011.
21. Mateescu, R., Meriot, S., Rampacek, S. Extending SPARQL with Temporal logic. Technical report. 2009.