A Web Tool for Type Checking and Testing of SPARQL Queries

Jesús M. Almendros-Jiménez(✉) and Antonio Becerra-Terón

Department of Informatics, University of Almería, 04120 Almería, Spain
{jalmen,abecerra}@ual.es

Abstract. In this paper a property-based testing tool for SPARQL is described. The tool randomly generates test cases in the form of instances of an ontology. The tool checks the well typed-ness of the SPARQL query as well as the consistency of the test cases with the ontology axioms. Test cases are after used to execute queries. The output of the queries is tested with a Boolean property which is defined in terms of membership of ontology individuals to classes. The testing tool reports counterexamples when the Boolean property is not satisfied.

1 Introduction

Property-based testing (PBT) is a well-known technique of program testing [9] involving the specification of *properties/assertions* on the output of a program to be tested. Properties on the output describe the required relationships between output data, which should be ensured by *free bug programs*. Normally, *test cases* are generated as input of the program and the properties/assertions are checked on the output of the test cases. When a counterexample is found, that is, an input test case in which the property is not satisfied by the result, the program has a bug. PBT can use either *black-box* techniques (i.e., *randomly generated test cases*) or *white-box* techniques (i.e., *test cases generated from code analysis*) for each kind of program. Among others, PBT has been studied in Java [8], functional languages [5], XQuery [2], model transformation languages [1] and relational databases [4].

In this paper a black-box tool for property-based testing has been designed in which input and output data are modeled in *RDF* and *OWL*, the program is a SPARQL query, and properties have the form of *membership of individuals to classes*. A RDF/OWL ontology to XML Schema *automatic mapping* is carried out by the tool in which classes and properties of the ontology **TBox** (i.e., ontology axioms) are mapped into XML Schema labels and attributes. The XML Schema is used to generate test cases. However, in order to generate useful test cases two additional checks are carried out. Firstly, the tool checks whether the SPARQL query is *well-typed*. A method for *type checking* is used *based on ontology consistency*. Wrongly typed queries *prevent testing*. Secondly, the tool

This work was supported by the EU (FEDER) and the Spanish MINECO Ministry (*Ministerio de Economía y Competitividad*) under grant TIN2013-44742-C4-4-R.

J. Cabot et al. (Eds.): ICWE 2017, LNCS 10360, pp. 535–538, 2017.
DOI: 10.1007/978-3-319-60131-1_38

(1)
```
SELECT ?event ?user2
WHERE {
  ?event sn:created_by ?user1 . ?event  sn:likes ?user2 .
  ?user2 sn:invited_to ?event
  }
```

(2)
```
SELECT ?msg1
WHERE {
  ?msg1 sn:sent_by ?user . ?msg1 sn:replied_by ?msg1
  }
```

(3)
```
SELECT ?user1 ?event ?user2
WHERE {
  ?event sn:added_by ?user1 . ?event sn:date ?date .
  ?user1 sn:friend_of ?user2 . ?user2 sn:age ?age .
  FILTER (?age >= 40 && ?date < '2017-01-01T00:00:00Z'^^xsd:dateTime)
  }
```

Fig. 1. Examples of buggy SPARQL queries

checks the *consistency* of test cases with ontology axioms. Inconsistent test cases lead to *wrong testing results*.

While query testing has been studied in other contexts (e.g., SQL [6]), and programming bugs are well stablished for SQL (e.g., [3]), as far as we know SPARQL testing has not been studied yet. Additionally, type checking is not currently supported by SPARQL implementations. This has as consequence empty/wrong/missing answers for queries.

2 Type Checking and Testing of SPARQL Queries

In order to illustrate the work, an example of ontology defining a *social network* is considered. Testing is a mechanism to detect bugs. In our case, testing is used for the detection of bugs in SPARQL queries. The question now is what does a bug mean in a SPARQL query? Fig. 1 shows some examples of SPARQL queries having a bug. Due to the lack of space, only three cases are shown, but more examples are available at the Web site of the tool (http://minerva.ual.es:8080/SPARQL/). Each case represents a different kind of bug in our approach.

Case (1). In this case the bug is due to *typing*. More concretely it is due to the domain and range of properties. The triple pattern *?event sn:likes ?user2* is wrong, because users like activities (in particular, events), and thus the order on the triple is wrong. From user's point of view, this query returns an empty answer when the bug is present, but some hint could be given. The testing tool uses HermiT ontology reasoner to check types, reporting the following diagnosis:

```
Test cases cannot be generated:
DisjointClasses(#Activity #User)
ClassAssertion(#Activity #event)
ClassAssertion(#User #event)
```

Case (2). In this case, the bug is due to *constraints* on properties (i.e., ontology axioms for properties). The triple pattern *?message1 sn:replied_by ?message1* is wrong, because *replied_by* is an *irreflexive* property: a message cannot be

answered by itself. Thus, again the *answer will be empty*. Consistency of test cases (i.e., agreement with constraints on classes and properties) is checked by the tool through HermiT ontology reasoner:

```
Unable to test the property.
It was not possible to find consistent tests.
```

Case (3). In this case, the query is well-typed and it does not contradict constraints. However, the user intention has to be taken into account. The intended meaning of the query is *"Retrieve events before this year added by users older than 40, and friends of these users"*. Here, the answer could be not empty but the programmer can find *wrong answers*: events added by users younger than 40. Here, there is a *mistake using variables*. The triple pattern *?user2 sn:age ?age* is wrong because *?user1* should be older than 40, instead of *?user2*. Here, a case in which testing of output properties is useful arises. Let us suppose that the testing tool is called with the following property: *Mature(?user1)* where $Mature \equiv age\ some\ integer[>= 40]$. The testing tool reports (using values: tennis for events, 30 and 50 for ages, jesus and antonio for users, and 2016-01-01, 2018-01-01 for dates[1]) the following message:

```
Output Property Falsifiable after 256 tests.
Counterexample:

<rdf:RDF >
  <sn:Event  rdf:about="#tennis">
    <sn:date rdf:datatype="#dateTime">2016-01-01T00:00:00Z</sn:date>
    <sn:added_by rdf:resource="#jesus"/>
  </sn:Event>
  <sn:User  rdf:about="#antonio">
    <sn:age rdf:datatype="#integer">50</sn:age>
    <sn:friend_of rdf:resource="#jesus"/>
  </sn:User>
  <sn:User  rdf:about="#jesus">
    <sn:age rdf:datatype="#integer">30</sn:age>
    <sn:friend_of rdf:resource="#antonio"/>
  </sn:User>
</rdf:RDF>
```

It means that after 256 test cases, the testing tool has found a counterexample for *Mature(?user1)*. The counterexample shows an event *tennis* which has been added by *jesus*, and *antonio* is a friend of *jesus* who is 50 years old, and *jesus* is 30 year old. Thus, *?user1* which is bound to *jesus* is not *Mature*. This counterexample serves as witness of the bug, which can be found in *?user2 sn:age ?age*. Replacing this triple pattern by *?user1 sn:age ?age*, the testing tool answers as follows:

```
Ok: passed 256 tests.
```

3 Web Tool

A *Web tool* available at http://minerva.ual.es:8080/SPARQL/ has been developed enabling the transformation of ontologies into XML Schemas facilitating

[1] Values are added to the XML Schema in our approach.

the customization of XML Schemas, for test case generation, by automatically pruning the XML Schemas. The tool has been implemented under the BaseX XQuery interpreter mainly responsible of test case generation. SPARQL (*Apache Jena ARQ engine*) has been embedded into XQuery thanks to *Java binding* capabilities of BaseX. Also the *ontology reasoner HermiT* [7] has been embedded into XQuery in order to check well typed-ness of SPARQL queries and consistency of test cases.

4 Conclusions and Future Work

A limitation of our approach is that properties can only be specified for individuals with membership to classes, and not for classes and properties. This a limitation of the current implementation but we plan to extend property definition to classes (for instance, disjointness of output classes) and properties (for instance, sub-property relationships). With regard to SPARQL coverage, the testing tool is able to test any SELECT query. ASK, DESCRIBE and CONSTRUCT queries are out of the scope of the testing tool. Finally, our testing tool generates test cases without taking into account the code, and the human tester intervention is required. We plan to extend our work to white box testing which means to automatically generate/prune the XML Schema from the code. The testing tool will become completely automatic without human tester intervention.

References

1. Almendros-Jiménez, J.M., Becerra-Terón, A.: Automatic generation of ecore models for testing ATL transformations. In: Bellatreche, L., Pastor, Ó., Almendros Jiménez, J.M., Aït-Ameur, Y. (eds.) MEDI 2016. LNCS, vol. 9893, pp. 16–30. Springer, Cham (2016). doi:10.1007/978-3-319-45547-1_2
2. Almendros-Jiménez, J.M., Becerra-Terón, A.: Automatic property-based testing and path validation of XQuery programs. Softw. Test. Verif. Reliab. **27**(1–2), 1–29 (2017)
3. Brass, S., Goldberg, C.: Semantic errors in SQL queries: a quite complete list. J. Syst. Softw. **79**(5), 630–644 (2006)
4. Chays, D., Deng, Y., Frankl, P.G., Dan, S., Vokolos, F.I., Weyuker, E.J.: An AGENDA for testing relational database applications. Softw. Test. Verif. Reliab. **14**(1), 17–44 (2004)
5. Claessen, K., Hughes, J.: QuickCheck: a lightweight tool for random testing of Haskell programs. ACM SIGPLAN Not. **46**(4), 53–64 (2011)
6. De La Riva, C., Suárez-Cabal, M.J., Tuya, J.: Constraint-based test database generation for SQL queries. In: Proceedings of the 5th Workshop on Automation of Software Test, pp. 67–74. ACM (2010)
7. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: an OWL 2 reasoner. J. Autom. Reasoning **53**(3), 245–269 (2014)
8. Khurshid, S., Marinov, D.: TestEra: specification-based testing of Java programs using SAT. Autom. Softw. Eng. **11**(4), 403–434 (2004)
9. Utting, M., Pretschner, A., Legeard, B.: A taxonomy of model-based testing approaches. Softw. Test. Verif. Reliab. **22**(5), 297–312 (2012)