System Testing and Validation (STV), Potsdam, 2006, pp. 45-52.

# Testing XPath Queries using Model Checking

Claudio de la Riva, Javier Tuya, José García-Fanjul
Computer Science Department, University of Oviedo
Phone +34 98 518 26 64, Fax +34 98 518 21 56
[claudio|tuya|jgfanjul]@uniovi.es

Abstract:
XML's rapid adoption as the data representation standard in web based systems is increasing the interest in applying XML query languages (as XPath) to access XML repositories. This technology entails new challenges related to testing, mainly derived from the hierarchical data representation in XML documents and the expressiveness of the query language. In this paper, we present a technique for the automatic generation of test cases for XPath expressions using the SPIN model checker. Both the XML schema and XPath query are previously modeled in the SPIN language and the test cases are obtained from the counterexamples that it generates.

## 1 Introduction

During last years, web based software systems have rapidly evolved from simple isolated HTML pages to Web applications interacting with others to implement complex business processes and web services. For example, the investment in web services of 1,1 billions of dollars in 2003 will be multiplied by ten in 2008, according to a recent study of IDC [7]. One of the factors contributing to this increment is the development of technologies that allow the information exchange between Web applications. XML [10] is playing an important role as the standard language for data representation in Internet. There is also a growing interest in the use of XML to manage information repositories (*native XML data stores*) [9].
However, the complex structure of XML documents and the diversity of emergent XML technologies for data access (XPath, XQuery, XSLT, etc.) make the verification and validation processes costly and difficult. Other problems are similar to the ones found in testing database applications: the design of the initial load, the existence of unknown information and the lack of specific adequacy criteria. Although there is recent research addressing some of these problems [8] [2], the works in testing XML data access are scarce [3].
In this paper, we present the initial efforts for testing XPath queries over XML documents using the model checker SPIN [6]. First, the XML Schema is coded as a finite transition system, and then we show how to check some features of the XPath query

over the transition system using the model checker. The obtained counterexamples are used to generate test cases for the XPath query.
The rest of this paper is organized as follows. Section 2 provides basic definitions of XML technologies and the notations that will be used in the paper. The modeling procedure from the XML Schema to SPIN is given in Section 3. Section 4 describes the generation of test cases for the XPath queries. Finally, Section 5 presents conclusions and future work.

## 2 Background

This section provides a brief overview of the XML technologies used in the rest of the paper (XML, XML Schema and XPath). We describe, also, the example that will be used to illustrate the methods and techniques of the paper.

### 2.1 XML and XML Schema

*Extensible Markup Language* (XML) is a markup hierarchical language used to describe semi-structured data in a way that is platform and language independent. XML documents are structured using tags, where the data is represented between `<tag>` and `</tag>`. Fig 1 shows an XML document example containing data for professors and subjects they teach, and the equivalent hierarchical representation structured in levels.

```
<Professors>
   <Prof id="1">
       <Sub> A1 </Sub>
       <Sub> A2 </Sub>
   </Prof>
   <Prof id="2">
       <Sub> A2 </Sub>
       <Sub> A3 </Sub>
   </Sub>
</Professors>
```

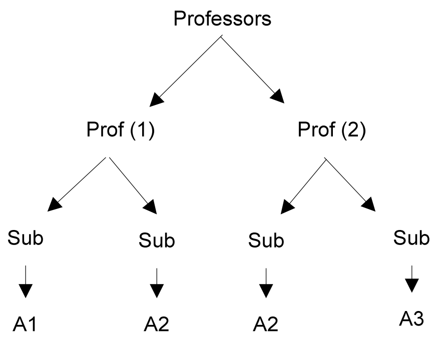

Fig 1: An XML document and the corresponding hierarchical representation

XML documents can be constructed by grammar rules. The grammars can be defined in either *Document Type Definitions* (DTD) or more recent *XML Schemas* [12]. Usually, the standard XML Schema provides more capabilities for data type definitions. Fig 2 shows a schema for the document in Fig 1.

```
<element name="Professors">
  <complexType>
    <sequence>
      <element name="Prof" minOccurs="1" maxOccurs="10" >
        <complexType>
          <sequence>
            <element name="Sub" type="string" maxOccurs="5"/>
          </sequence>
          <attribute name="id" type="int" use="required"/>
        </complexType>
      </element>
  </complexType>
  </element>
```
Fig 2: An XML Schema example

## 2.2 XPath

*XML Path Language* (*XPath*) [11] is an expression-based language used to navigate and select elements (nodes) in an XML document. In this work, we use a subset of XPath similar to the one described in [4]. It consists of the following operations: the child axis (/), the descendant axis (//), self-reference (.), parent-reference (..), node name and predicates ([]). An XPath expression comprises *location paths* and constant values. A location path can be absolute or relative. The absolute location path starts with / or //. The relative location path consists of a list of *steps* connected with / or //. The step can be a self-reference, parent-reference or more complex expressions formed by node names and predicates.
An XPath expression is evaluated over an XML document and the output is a set of nodes from that document. For example, the expression `/Professors/Prof[@id="2"]/Sub` over the XML document in Fig 1 returns the subjects taught by the professor `2`, that is, the elements A2 and `A3`.

## 3 Modeling XML Schemas in SPIN

SPIN [6] is a model checker based in a explicit enumeration of the state space, mainly used for the formal verification and simulation of software systems. The system is coded in Promela (SPIN's input language) and the properties to verify are specified either as linear temporal logic (LTL) formulas or as asserts. A system specification in Promela consists of a type declaration, a global variables declaration and a set of concurrent processes, which describe the software system behaviour.
Given an XML Schema specification, the corresponding type declaration in Promela can be represented as follows [4]. Basic types (`int`, `bool`, etc.) are mapped to basic types in Promela. The string type is mapped to an enumerated type (`mtype`). Complex types are declared as record definitions (`typedef`). When an element has multiple occurrences, for example the element `Prof` in Fig 2, it is defined as an array with its max occurrence as the array size. In addition, we use a variable to record the

actual number of occurrences. Fig 3 shows the representation of the XML Schema in Fig 2 in Promela code.

```
typedef Subject{
      mtype value;
};
typedef Professor{
      int id;
      Subject Sub[5];
      int n_sub;
};
typedef Professors{
      Professor Prof[10];
      int n_prof;
};
```

Fig 3: Promela specification of the XML Schema in Fig 2.

SPIN processes must be specified by finite state transition systems. In our approach, we use a variation of the XML document model given in [5]. We represent an XML Schema instance (XML document) by means of a set of nodes, which define the XML document elements, and a set of edges, which specify the navigation between the nodes. This transition system is translated to a process where each XML document element represents a state. The child, parent and sibling relations between the nodes in the XML document are modelled as transitions between the states. In Promela, each level of the XML document can be specified as `if` sentences, and the transitions as choices using the unconditional jump (`goto`). Fig. 4 shows a summarized Promela code for the `Prof` level in the XML Schema in Fig 2.

```
mtype dir = {Down, Up, Right, Left, Stop} /* Axis */
Professors d; /* XML document */
int iP,iS;    /* Professors and Subjects indexes */
[...]
init { /* Process */

  [...]   /* Initial Load of XML document (Fig 1) */

  RootLevel: [...]
  ProfLevel: if
             ::(d.Prof[iP].n_sub >0) ->
                     dir=Down ; goto SubLevel;
             :: dir=Up ; goto RootLevel;
             ::(iP<9) ->
                     dir=Right; iP++; goto ProfLevel;
             ::(iP>0) ->
                     dir=Left; iP--; goto ProfLevel;
             :: dir=Stop ; goto End;
            fi ;
  SubLevel: [...]
  End: skip;
}
```

Fig 4: Promela code for the professor level of the XML document in Fig 1.

The `init` process starts with the definition of the data in the XML document (initialization of the data structures in Fig. 3). The first choice (`dir=Down`) implements the navigation to a node at the subject level. The second choice represents the transition to the parent node (`dir=Up`). The right (`dir=Right`) and left (`dir=Left`) navigation to the sibling nodes at the professor level are represented in the third and fourth choices, respectively. The last choice (`dir=Stop`) represents the final state of the transition system.

## 4 Test Case Generation

Traditionally [1], the testing approach using model checking techniques is based in the specification of test criteria as formulas which express the "negation" of the requirements. Thus, the model checker is instructed to search a counterexample showing that the test criteria is satisfied, that is, the negation of the test criteria is inconsistent in the system.

Our approach to test XPath queries follows a similar way, where the test criteria consists of a set of test case specifications derived from the original XPath expression. For example, suppose the XPath query `/Professors/Prof[@id=2]/Sub`, which returns the list of the subjects of professor `2` in the XML document in Fig 1. A possible test case specification could be *"there are no professors with id = 2*". This test case specification will be coded as the LTL formula `[]`$\vee_{iP}$`(professor.prof[iP]==2)`, that represents "*there are always professors with id = 2*".

Nevertheless, the simple XML document representation described in Section 3 and the previous LTL formula codification do not guarantee the generation of a counterexample that can be used to generate valid test cases. Indeed, if we consider that there is an initial load in the XML document (Fig 1) and we verify the previous LTL property, SPIN does not return a counterexample because there is a node at the professor level with `id=2`. A similar situation occurs when the XML document is empty. In this case, SPIN returns a counterexample, but it is not relevant as a test case because it does not contain data.

These problems arise since the state space (XML document) is not modified during the model checker exploration. In other words, our transition system representation does not include any strategy to change the XML document in order to obtain a counterexample that violates the LTL formula.

To avoid the above restriction, we define a set of transitions associated with insert (`I`), modify (`M`) and delete (`D`) operations, which are triggered to represent modifications in the XML document. The insert transition creates new nodes with random values, the modify transition changes the node with a random value and the delete transition marks the node with a special value (`Null`). Fig 5 shows the effect of these operations in a node of an XML document. Given an initial load of the XML document, the execution of these transitions during the state space exploration will produce changes (or new nodes) in the structure of the initial document. In SPIN, according to the transition system defined in Section 3, these transitions are represented as new choices associated to each level of the XML document.

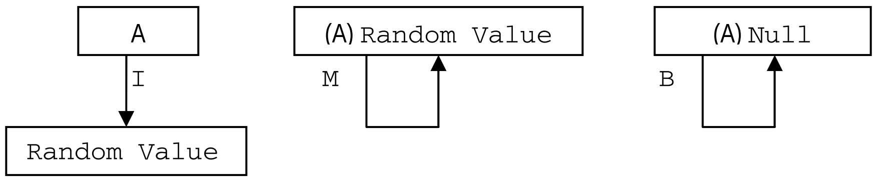

Fig 5: Corresponding actions to insert (`I`), modify (`M`) and delete (`D`) transitions

A test case specification (`prop`) will be coded as the LTL formula (`[](exp` • `[]!prop))`, where `exp` is a expression that represents the execution of the previously defined transitions.
For example, the test case specification *"there are no professors with id = 2*" related to the XPath query `/Professors/Prof[@id=2]/Sub` will be specified as the LTL formula:

```
[]( (act==I|act==M|act==B)
      • [] ∨iP (professor.prof[iP]==2) )
```

Fig 6 shows graphically the generated counterexample for this formula using the XML document in Fig. 1 as the initial load.

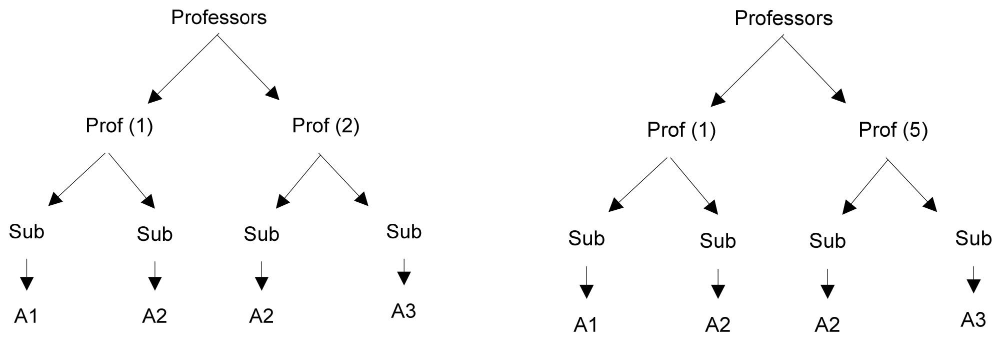

Fig 6: Initial load (left) and test case (right) for the XPath query `/Professors/Prof[@id=2]/Sub`

The counterexample is created according to the next navigation sequence. The model checker starts the state space exploration at the root level (`Professors` node). It moves to the first node at the professor level (`Prof(1)`) by means of the execution of the transition corresponding to the child relation (`dir=Down`). Then, the `Prof(2)` node is visited executing the right sibling transition (`dir=Right`). At this point, a modify transition (`act=M`) is triggered, which changes the node value to `5`. In the next step, the state space exploration ends (`dir=Stop`) and the original LTL property is false. The final configuration of the data structures that comprise the XML document (final state of the obtained counterexample) is, precisely, the test case.

## 5 Conclusions and Future Work

In this paper, we present the initial efforts for testing XPath queries over XML documents based on the use of model checkers. Our approach generates automatically test cases as XML documents obtained from model checker counterexamples by means of specific transitions inserted in the state transition system that represents the XML document instances. Although, more experimentation over XML repositories, XML structures and XPath queries is needed, the initial results show the feasibility of this approach. One of the drawbacks is the use of the SPIN model checker as front-end due to the explicit enumeration of the state space. Thus, the codification of both the XML document and the XPath query using symbolic techniques could be an interesting line for future work. The definition of adequacy criteria for testing, for example using coverage analysis, and the specification of testing heuristics are other research lines.

## Acknowledgement

This work was funded by the Ministry of Education and Science (SPAIN), National Plan I+D+i, under the projects IN2TEST (TIN2004-06689-C03-02) and REPRIS (TIN2005-24792-E).

## References

1. Ammann, P.E., Black, P.E., Majurski, W.: Using Model Checking to Generate Tests from Specifications. Proceedings of the 2nd IEEE International Conference on Formal Engineering Methods, pp. 46-54, (1998)
2. Chays, D., Deng, Y., Frankl, P.G., Dan, S., Vokolos, F., Weyuker E.J.: An AGENDA for Testing Relational Database Applications. Software Testing, Verification and Reliability, 14(1):17-44, (2004)
3. de la Riva, C., Tuya, J.: A Survey on Testing XML-Based Applications. Proceedings of the International Conference on WWW/Internet, volume II, pp.349-352 (2005)
4. Fu, X., Bultan, T., Su, J.: Model Checking XML Manipulating Software. Proceedings of the International Symposium of Software Testing and Analysis, SIGSOFT Software Engineering Notes, 29(4):252-262, (2004).
5. Hartel, P.H.: A Trace Semantics for Positive Core XPath. Proceedings of Temporal Representation and Reasoning 2005, pp. 103-112, (2005).
6. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley, Boston, Massachusetts (2003)

7. Leavitt, N.: Are Web Services Finally Ready to Deliver? IEEE Computer, 37(11):14-18, (2004)
8. Suárez, M.J, Tuya, J.: Using a SQL Coverage Measurement for Testing Database Applications. Proceedings of the ACM SIGSOFT FSE, pp. 253-262 (2004)
9. Vakali, A., Catania, B., Magdalena, A: XML Data Stores: Emerging Practices. IEEE Internet Computing, 9(2):62-69.(2005)
10. World Wide Web Consortium: Extensible Markup Language (XML) http://www.w3.org/XML (accessed December 2005)
11. World Wide Web Consortium: XML Path Language (XPath). http://www.w3.org/TR/xpath (accessed December 2005)
12. World Wide Web Consortium: XML Schema. http://www.w3.org/XML/Schema (accessed December 2005)