# VeriFLog: A Constraint Logic Programming approach to Verification of Website Content

Jorge Coelho[1] and Mário Florido[2]

[1] Instituto Superior de Engenharia do Porto & LIACC
[2] University of Porto, DCC-FC & LIACC
Rua do Campo Alegre, 823, 4150-180 Porto, Portugal
Tel. +351 2260778830, Fax. +351 226003654
{jcoelho,amf}@ncc.up.pt

**Abstract.** Web site semantic content verification can be a tedious and error prone task. In this paper we propose a framework for syntactic validation and semantic verification based on the logic programming language XCentric. The high declarative model of this language based on a new unification algorithm along with an interface to semistructured data provides an elegant framework for semantic error detection. The result is an easy to follow model to improve website quality and management.

## 1 Introduction

Managing the semantic content of a website is not easy and it is even harder when it is built by many different persons. Consider for example a website of a university. There, different persons such as, teachers and administrative staff have access to some part of the website, for example their homepage. The website manager or the university administration may impose some constraints in the website construction, for example, every person must have his/her curriculum in his/her homepage or every teacher must have information about his/her teaching activities. Verifying these constraints may be a difficult task when we have a high number of pages to look up. Another question arises from usefulness of the information available in the website: can it be used to infer more information? For example, in our research laboratory, technical reports and other publications are added to a central web page. It is desirable to infer some statistical data about scientific production activities. Both of these tasks can be accomplished by a logic-based language with an interface to semistructured data and an inference engine to impose semantic constraints and infer new data.

In this work we propose a framework for web site verification based on a constraint-logic programming language named XCentric [3]. This language achieves a high level of expressivity through a new unification model based on flexible arity function symbols and sequence variables. It provides a very pleasant way to query data in html/xml documents and to write verification rules. The framework works by translating documents to terms (an internal representation for documents), and then optionally applying syntactic validation, verifying semantics and inferring new data. All the different modules are implemented in

XCentric which is built on top of SWI-Prolog [11], thus the programmer can use all the potential of SWI-Prolog in addition to XCentric.

As motivation, consider the following simple example: suppose we want to verify if the teachers of our university have in their XML-based home pages a correct email address. We have an XML file generated from our database with their data and can use the following code:

```
check(N,URL):-
    http2pro(URL,T),
    T =*= hpage(_,email(E),_),
    xml2pro('dbase.xml',DB),
    DB =*= db(_,record(name(N),_,email(E),_),_).
```

This program, starts by using *http2pro* to retrieve from the web address in *URL* the XML code, translating it to an internal notation and storing it in variable *T*. Then the non-standard unification operator $=*=$ is used to find the teacher's email and store it in variable *E*. Note that the anonymous variable '_' allows the programmer to ignore parts of the document (this is not possible with standard unification). Using *xml2pro* the database file is translated to the internal representation and stored in variable *DB*. Then the record of the teacher named *N* is searched and the program succeeds only if both the email found in the web page and the one found in the database are the same. We can easily generalize the program to accept a list of teachers, verify the entire website and output error messages.

In the rest of the paper we assume that the reader is familiar with logic programming ([10]) and XML ([13]).

We start in section 2 by presenting terms with flexible arity symbols and sequence variables. Then, in section 3 we present our framework along with some examples. In section 4 we propose XCentric as the intermediate language for another language based in XML itself. Then, in section 5 we present related work and in section 6 we conclude.

## 2 Terms with Flexible Arity Symbols and Sequence Variables

Here we present briefly the concept behind XCentric along the lines presented in [4].

### 2.1 Constraint Logic Programming

Constraint Logic Programming (CLP) [7] is the programming paradigm used in a class of languages based on rule-based constraint programming. Each different language is obtained by specifying the domain of discourse and the functions and relations on the particular domain. This framework extends the logic programming framework because it extends the Herbrand universe, the notion of *unification* and the notion of equation, accordingly to the new computational domains. A complete description of the major trends of the fundamental concepts about CLP can be found in [7].

### 2.2 XCentric

XCentric extends Prolog with terms with flexible arity symbols and sequence variables. We now describe the syntax of XCentric programs and their intuitive semantics.

In XCentric we extend the domain of discourse of Prolog (trees over uninterpreted functors) with finite sequences of trees.

**Definition 21** *A* sequence $\tilde{t}$*, is defined as follows:*

- $\varepsilon$ *is the empty sequence.*
- $t_1, \tilde{t}$ *is a sequence if* $t_1$ *is a term and* $\tilde{t}$ *is a sequence*

**Example 21** *Given the terms* $f(a)$*,* $b$ *and* $X$*, then* $\tilde{t} = f(a), b, X$ *is a sequence.*

Equality is the only relation between trees. Equality between trees is defined in the standard way: two trees are equal if and only if their root functor are the same and their corresponding subtrees, if any, are equal.

We now proceed with the syntactic formalization of XCentric, by extending the standard notion of Prolog term with flexible arity function symbols and sequence variables.

We consider an alphabet consisting of the following sets: the set of standard variables, the set of sequence variables (variables are denoted by upper case letters), the set of constants (denoted by lower case letters), the set of fixed arity function symbols and the set of flexible arity function symbols.

**Definition 22** *The set of terms over the previous alphabet is the smallest set that satisfies the following conditions:*

1. *Constants, standard variables and sequence variables are terms.*
2. *If* $f$ *is a flexible arity function symbol and* $t_1, \ldots, t_n$ $(n \geq 0)$ *are terms, then* $f(t_1, \ldots, t_n)$ *is a term.*
3. *If* $f$ *is a fixed arity function symbol with arity* $n$*,* $n \geq 0$ *and* $t_1, \ldots, t_n$ *are terms such that for all* $1 \leq i \leq n$*,* $t_i$ *does not contain sequence variables as subterms, then* $f(t_1, \ldots, t_n)$ *is a term.*

Terms of the form $f(t_1, \ldots, t_n)$ where $f$ is a function symbol and $t_1, \ldots, t_n$ are terms are called *compound terms.*

**Definition 23** *If* $t_1$ *and* $t_2$ *are terms then* $t_1 = t_2$ *(standard Prolog unification) and* $t_1 = * = t_2$ *(unification of terms with flexible arity symbols) are constraints.*

A constraint $t_1 = * = t_2$ or $t_1 = t_2$ is solvable if and only if there is an assignment of sequences or ground terms, respectively, to variables therein such that the constraint evaluates to *true*, i.e. such that after that assignment the terms become equal.

**Remark 21** *In what follows, to avoid further formality, we shall assume that the domain of interpretation of variables is predetermined by the context where they occur. Variables occurring in a constraint of the form $t_1 = * = t_2$ are interpreted in the domain of sequences of trees, otherwise they are standard Prolog variables. In XCentric programs, therefore, each predicate symbol, functor and variable is used in a consistent way with respect to its domain of interpretation.*

XCentric programs have a syntax similar to Prolog extended with the new constraint $= * =$. The operational model of XCentric is the same of Prolog.

### 2.3 Constraint Solving

Constraints of the form $t_1 = * = t_2$ are solved by a non-standard unification that calculates the corresponding minimal complete set of unifiers. Details about the implementation of this non-standard unification can be found in [3]. As motivation we present some examples of unification:

**Example 22** *Given the terms $f(X, b, Y)$ and $f(a, b, b, b)$ where $X$ and $Y$ are sequence variables, $f(X, b, Y) = * = f(a, b, b, b)$ gives three results:*

1. *$X = a$ and $Y = b, b$*
2. *$X = a, b$ and $Y = b$*
3. *$X = a, b, b$ and $Y = \varepsilon$*

**Example 23** *Given the terms $f(b, X)$ and $f(Y, d)$ where $X$ and $Y$ are sequence variables, $f(b, X) = * = f(Y, d)$ gives two possible solutions:*

1. *$X = d$ and $Y = b$*
2. *$X = N, d$ and $Y = b, N$ where $N$ is a new sequence variable.*

Note that this non-standard unification is conservative with respect to standard unification: in the last example the first solution corresponds to the use of standard unification. Soundness and completeness of this non-standard unification were proved in [8] and [4].

### 2.4 XML Processing

In XCentric an XML document is translated to a term with flexible arity function symbol. This term has a main functor (the root tag) and zero or more arguments. Although our actual implementation translates attributes to a list of pairs, since attributes do not play a relevant role in this work we will omit them in the examples, for the sake of simplicity. Consider the simple XML file presented bellow:

```
<addressbook>
        <record>
                <name>John</name>
                <address>New York</address>
```

```
                <email>john.ny@mailserver.com</email>
        </record>
...
</addressbook>
```

The equivalent term is:

```
addressbook(record(name('John'),address('New York'),
                   email('john.ny@mailserver.com')),...)
```

**Example 24** *Suppose that the term Doc is the XCentric representation of the document "addressbook.xml". If we want to gather the names of the people living in New York we can simply solve the following constraint:*

$$Doc = * = addressbook(\_, record(name(N), address('New\ York'), \_), \_).$$

*All the solutions can then be found by backtracking.*

Note that '_' is an unnamed sequence variable which unifies with any sequence. Further details and examples can be found in [3, 4].

## 3 Website verification framework

The main idea of this work is to provide an interface to semistructured data, syntactic validation and use XCentric as the rule language for semantic verification. By semantic verification we mean verifying if the content of a website is correct with relation to a given criteria. For example, we have a web page with all the staff of the department grouped by category (Senior researcher, Phd researcher, Researcher and Assistant researcher), one verification we can do, is searching for people catalogued as one of the above mentioned categories but that doesn't belong to that category.

With our framework the programmer can also use the high declarative model of XCentric to infer new knowledge from web pages. For example, if every researcher of our department has its own publications in his/her homepage we can easily retrieve this data and get new statistical data from it, for example, how many publications in journals and international conferences by year.

### 3.1 Examples

Due to space limitations, we only present some simple examples of how this framework can be used. We encourage the reader to test the full versions of these examples and many others available at:

`http://www.ncc.up.pt/~jcoelho/veriflog/examples.html`.

There are four ways to retrieve HTML/XML data: directly from the web using *http2pro(URL, Term)*; directly from the web with validation using *http2pro(URL, DTDFile, Term)*, where the file in *URL* is validated against the DTD given in *DTDFile*; from an xml file using *xml2pro(XMLFile, Term)* and from an xml file with validation using *xml2pro(XMLFile,DTDFile, Term)*.

**Example 31** *In our department website we have an HTML page with a list of technical reports indexed by year. A similar page can be found in* `http://www.ncc.up.pt/~jcoelho/veriflog/examples/techreports.html`. *Suppose that we want to know if some publications are out of place with respect to the year of publication. We can simply do:*

```
verify(URL):-
     http2pro(URL,TR),
     TR =*= html(_,body(_,h4(Y1),Seq,h4(Y2),_),_),
     atom_number(Y1,N1),atom_number(Y2,N2),
     N2 is N1 - 1,write('Year: '),write(Y1),nl,
     process(Seq,Y1).

process(Seq,Y):-
      Seq =*= ul(_,li(C),_),
      not(substring(Y,C)),write(C).
```

We start by using http2pro/2 to translate the web page given in variable URL into its internal representation, then we get the sequences of elements (variable Seq), between two adjacent years (variables Y1 and Y2). Note that the way we use variables (unifying with zero or more elements) is not the standard way in Prolog but very useful for XML queries:

```
      TR =*= html(_,body(_,h4(Y1),Seq,h4(Y2),_),_),
```

We then check if some string (variable C) describing a technical report does not include the year that indexes that report (variables '_' are used to ignore sequences of elements):

```
      Seq =*= ul(_,li(C),_),
      not(substring(Y,C)),
```

Running this program over the previously mentioned web page will present the following output:

```
Year: 2005
Year: 2004
Sandra Alves and Mario Florido. Type Inference for Programming
Languages: A Constraint Logic Programming Approach, Technical
Report DCC -  FC & LIACC, Universidade do Porto
```

Meaning that for the year 2004 it was found one publication that doesn't have a reference to the year it was published or the year is different from 2004.

**Example 32** *Given a teacher homepage, for example this one:*

```
<teacher>
   <name>Mario</name>
   <phone>+351 123456789</phone>
   <email>amf@ncc.up.pt</email>
   <courses>
     <name>Compilers</name>
   </courses>
</teacher>
```

*We want to know if it includes information about the classes he teaches. We get that information from our database (in an XML file) and compare both:*

```
DBase =*= staff(_,teacher(_,name(N),_,courses(_,class(C),_),_),_),
write('Database: '),write(C),
XML =*= teacher(_,name(N),_,courses(_,name(C),_),_),
write('>Found in XML<'),nl.
```

*Here we query the database file stored in variable* DBase *and for each teacher we store his name in variable* N *and the classes he teaches one by one in variable* C*. Then, we query the teacher homepage, stored in variable* XML *trying to find the same classes which were found in the database. The output follows:*

```
Database: Compilers >Found in XML<
Database: Logic
Database: Programming Languages
```

*Meaning that, from the three courses found in the database only* Compilers *was found in the teacher homepage. This can be easily generalized to a set of teachers (using http2pro/2 to retrieve each ones data).*

## 4 XCentric as an intermediate language

Although a lot of the framework power is inherited from XCentric itself, a user less experienced with logic programming but used to XML can have some trouble figuring out how to do verification. However, note that our framework can be used as an intermediate language for another language based in XML itself. We can build an XML-like syntax for a new language and then translate it to our framework. For example, given the following query in a language similar to the one presented in [5]:

```
<pubs>
  _
  <pub>
       _
       <publisher> X </publisher>
       _
  </pub>
  _
</pubs>
=> X
```

This can be translated to our framework as (note that '_' ignores sequences of elements):

```
XML =*= pubs(_,pub(_,publisher(X),_),_).
```

Due to the power of constraint solving for terms with arbitrary arity, XCentric revealed to be a better language than Prolog as the target of a pre-processing step for XML based verification language such as [5].

## 5 Related work

Imposing criteria in website creation using logic programming approach was addressed in several previous works. In [5] the author proposed the use of a simple pattern-matching-based language and its translation to Prolog as a framework for website verification. Our work also uses Prolog but our syntax smoothly integrates with it, thus our framework inherits all the power of Prolog. We also provide a richer interface to semistructured data. In [12] logic was proposed as the rule language for semantic verification, there the authors provide a mean for introducing rules in a graphical format. In contrast, our work provides a powerful programming language and thus a richer and more flexible way to write rules. In [1] it was presented a rewriting-based framework that uses simulation [6] in order to query terms, this was a new rewriting-based language quite different from ours. In [9] the author proposed an algorithm for website verification similar to [2] in expressiveness but based in a different theoretical approach. The idea was to extend sequence and non-sequence variable pattern matching with context variables, allowing a more flexible way to process semistructured data but the author doesn't provide an implementation.

## 6 Conclusion

Our main contribution is a framework for website verification applying concepts from a language that revealed to be quite appropriate for this task (this is shown by the quite simple code presented in this paper to perform some standard verification tasks). We presented examples of application and proposed the use of this framework as an intermediate language where a more "XML-programmer friendly" language is used on top of the framework. As far as we know this is the most complete approach to website verification based in logic programming in the sense that it integrates an interface to semi structured data, syntactic validation and semantic verification. We are now working to improve our work namely by building an automatic website crawler and a graphical interface in order to increase the ease of use of this tool. Another novelty of this approach is that it smoothly integrates with Prolog and thus it inherits all the power of the language. Moreover we are convinced that XML-based websites will increase in number in the next years increasing the utility of verification tools such as VeriFLog.

## References


1. M. Alpuente, D. Ballis, and M. Falaschi. A Rewriting-based Framework for Web Sites Verification. In *Electronic Notes in Theoretical Computer Science*, pages 41–61. Elsevier Science, 2005.
2. F. Bry and S. Schaffert. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *International Conference on Logic Programming (ICLP)*, volume 2401 of *LNCS*, 2002.

3. Jorge Coelho and Mário Florido. CLP(Flex): Constraint Logic Programming Applied to XML Processing. In *Ontologies, Databases and Applications of SEmantics (ODBASE)*, volume 3291 of *LNCS*. Springer Verlag, 2004.
4. Jorge Coelho and Mário Florido. CLP(Flex): Constraint logic programming applied to XML processing. Technical Report 06, DCC-FC, LIACC. University of Porto, (available from www.ncc.up.pt/~jcoelho/clpflex.pdf), July 2004.
5. Thierry Despeyroux. Practical semantic analysis of web sites and documents. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *WWW*, pages 685–693. ACM, 2004.
6. Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *FOCS*, pages 453–462, 1995.
7. Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
8. T. Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In *Joint AICS'2002 - Calculemus'2002 conference*, LNAI, 2002.
9. Temur Kutsia. Context sequence matching for xml. In *Proceedings of the 1th International Workshop on Automated Specification and Verification of Web Sites*, pages 103–119, Valencia, Spain, 14–15 March 2005.
10. J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second edition, 1987.
11. SWI Prolog. http://www.swi-prolog.org/.
12. Frank van Harmelen and Jos van der Meer. Webmaster: Knowledge-based verification of web-pages. In Ibrahim F. Imam, Yves Kodratoff, Ayman El-Dessouki, and Moonis Ali, editors, *IEA/AIE*, volume 1611 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 1999.
13. Extensible Markup Language (XML). http://www.w3.org/XML/, 2003.