XCentric: A logic-programming language for XML Processing

Jorge Coelho
Instituto Superior de Engenharia
do Porto & LIACC Porto, Portugal
jcoelho@ncc.up.pt

Mário Florido
University of Porto,
DCC-FC & LIACC
Porto, Portugal
amf@ncc.up.pt

1. INTRODUCTION

XML is a powerful format for tree-structured data. A need for programming language support for XML processing led to the definition of XML programming languages, such as XSLT [12], XDuce [6], CDuce [1], XMλ [9], SXSLT [7], Xtatic [13] and Xcerpt [2].

In this demonstration we present XCentric, a logic programming language based on the *unification of terms with flexible arity function symbols* extended with a new *type system* for dealing with sequences and new features for searching sequences inside trees at arbitrary depth.

The main features are:

Regular expression types and unification: regular expression types give us a compact representation of sequences of arguments of functors with flexible arity. They also add extra expressiveness to the unification process. Let us present an illustrating example.

The following declaration introduces regular expression types describing terms in a simple bibliographic database:

```
:-type bib ---> bib(book+).
:-type book ---> book(author+, name).
:-type author ---> author(string).
:-type name ---> name(string).
```

Type expressions of the form $f(...)$ classify tree nodes with the label f (XML structures of the form $< f > ... < /f >$). Type expressions of the form $t*$ denote a sequence of arbitrary many ts, and $t+$ denotes a sequence of arbitrary many ts with at least one t. Thus terms with type *bib* have *bib* as functor and their arguments are a sequence of one or more books. Terms with type *book* have *book* as functor and their arguments are a sequence consisting of one or more authors followed by the name of the book.

The next type describes arbitrary sequences of authors with at least two authors:

```
:-type type_a ---> (author(string),author(string),
                                        author(string)*).
```

To get the names of all the books with two or more authors we just need the following query ($= * =$ stands for unification of terms with flexible arity functors and $t :: \tau$ means that term t has type τ):

```
?-bib(_,book(X::type_a,name(N)),_)=*=BibDoc::bib.
```

This unifies two terms typed by *bib*. The type declaration in the first argument is not needed because it can be easily reconstructed from the term. In this case we bind the variable N to the name's content of the first book with at least two authors. Note how the type *type_a* in the first argument of the unification is used to jump over an arbitrary number of arguments and extract the name of the first book with at least two authors. All the results can then be obtained, one by one, by backtracking. This goes far beyond standard Prolog unification.

Incomplete terms in depth. The previous point gives XCentric the power of partially specifying terms in breadth (i.e. within the subterms of the same parent term). In XCentric one can also partially specify a term in depth using the *deep* predicate. A query term of the form $deep(t_1, t_2)$ matches all subterms of t_2 that match the term t_1. Consider the following example: in XCentric we can explicitly refer to sequences of terms $t_1, ..., t_n$ as $< t_1, ..., t_n >$. Suppose we want to find sequences of two authors in a document to which the variable *XML* is bound. We can use the query:

```
?-deep(<author(A1),author(A2)>,XML).
```

The names of the two authors will bind variables A_1 and A_2, and all solutions can be found by backtracking. Note that we can search not only single elements but also sequences of elements.

The goal of XCentric is to give programmers using the logic programming paradigm a tool that makes it easier to process XML. This is relevant for the logic programming community because subjects such as databases or datamining are quite relevant application areas of logic programming, and in these areas XML is becoming more and more a standard data format for information exchange.

Dealing directly with sequences of terms and using types as a compact representation of these sequences has been used intensively by functional mainstream XML processing languages based on pattern matching, such as XDuce and CDuce [6, 1]. In contrast logic programming languages usually have libraries [11, 10] to deal with XML that do not use the notions of sequence or regular types: they just translate XML documents to a list of terms. XCentric uses recent results of unification theory (*unification of terms with functors*

of flexible arity [8]) and type systems for logic programming (*regular types* [14, 5]) as the theoretical basis of a logic programming language for XML processing where sequences of terms are first class objects of the language. As the next examples show, this leads to a much more declarative way of processing XML when compared to the standard Prolog libraries for XML processing.

Theoretical foundations of the non-typed version of flexible arity term unification can be found in [8, 3, 4].

2. DEMONSTRATION OVERVIEW

In this demonstration we will introduce XCentric by showing examples of:

- Querying XML documents with unification of flexible arity function symbols which give a compact representation for XML data.
- Regular types as a representation for DTDs and XML Schemas.
- Use of types in the unification process as a mechanism to validate data and as a way to make queries more compact.

We will also introduce the XCentric syntax which is an extension to Prolog with regular types and a non-standard unification.

In the homepage of XCentric, http://www.ncc.up.pt/xcentric/, one can find other references, many XCentric programs, and the XCentric distribution.

3. REFERENCES

[1] Véronique Benzaken, Giuseppe Castagna, and Alain Frisch. CDuce: an XML-centric general-purpose language. In *Proceedings of the eighth ACM SIGPLAN Int. Conference on Functional Programming*, Uppsala, Sweden, 2003.

[2] F. Bry and S. Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *2nd Annual International Workshop Web and Databases*, volume 2593 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. Unification with flexible arity symbols: a typed approach. In *Informal proceedings of the 20th International Workshop on Unification (UNIF'06)*, Seattle, USA, 2006.

[5] Mário Florido and Luís Damas. Types as theories. In *Proc. of post-conference workshop on Proofs and Types, Joint International Conference and Symposium on Logic Programming*, 1992.

[6] Haruo Hosoya and Benjamin Pierce. XDuce: A typed XML processing language. In *Third Int. Workshop on the Web and Databases*, volume 1997 of *LNCS*, 2000.

[7] Oleg Kiselyov and Shriram Krishnamurthi. SXSLT: Manipulation Language for XML. In *Practical Aspects of Declarative Languages*, volume 2562 of *LNCS*, 2003.

[8] Temur Kutsia. Unification with sequence variables and flexible arity symbols and its extension with pattern-terms. In *Joint AISC'2002 - Calculemus'2002 conference*, LNAI, 2002.

[9] Erik Meijer and Mark Shields. XMλ: A functional language for constructing and manipulating XML documents. (Draft), 1999.

[10] Pillow: Programming in (Constraint) Logic Languages on the Web. http://clip.dia.fi.upm.es/Software/pillow/pillow.html, 2001.

[11] SWI Prolog. http://www.swi-prolog.org/.

[12] XSL Transformations (XSLT). http://www.w3.org/TR/xslt/, 1999.

[13] Xtatic. http://www.cis.upenn.edu/~bcpierce/xtatic/, 2004.

[14] Justin Zobel. Derivation of polymorphic types for prolog programs. In *Proc. of the 1987 International Conference on Logic Programming*. MIT Press, 1987.