Typechecking XQuery: A Prototype in ASF+SDF

Verificação de Tipos de XQuery: Um Protótipo em ASF+SDF

Sandra M. Venske
DECOMP–UNICENTRO, Guarapuava, PR
ssvenske@unicentro.br

Martin A. Musicante
DIMAP–UFRN, Natal, RN
mam@dimap.ufrn.br

Abstract: Semistructured data (particularly XML) are the standard data representation for information exchange in the world-wide web. A number of query languages for XML has been proposed. Most of them follow the style of SQL. One of these languages is XQuery. In this work, we propose the construction of a prototype for the static type analysis of XQuery programs. The prototype implements XQuery operational semantics, in a way that is close to that proposed by the W3C. The implementation was built using the ASF+SDF meta-environment. The prototype described here is a first step in the construction of a practical XML query language laboratory, in which different semantics for commands may be tested.

Key words: Operational Semantics; XQuery; Type Systems.

Resumo: Dados semi-estruturados, e em particular XML, têm se destacado como padrão de representação de dados na world-wide web. Para realizar consultas a documentos (ou bancos de dados) semi-estruturados, têm sido idealizadas linguagens de consulta. Estas linguagens seguem, em geral, o estilo da linguagem SQL. Uma destas linguagens é XQuery. Neste trabalho, propõe-se a construção de um protótipo para a verificação estática de XQuery, com base na sua semântica operacional, usando o meta-ambiente de programação ASF+SDF. Para isto, implementou-se um protótipo, que poderá ser usado como laboratório de teste de novas características a serem adicionadas à linguagem.

Palavras-Chave: Semântica Operacional; XQuery; Sistema de Tipos.

1 Introduction

Semistructured data [Buneman, 1997], particularly XML [Bray et al., 2000] is the standard data representation for information exchange in the world-wide web.

Trees or graph structures can be used to represent semistructured data. XML is a language for the textual description of these trees. The information contained by XML documents can be queried, in order to construct new documents.

There are many query languages for XML. These languages follow, in general, the style of SQL [Groff and Paul N, 1999]. XQuery [Boag et al., 2002] is one of such languages.

In this work, we describe a prototype for the static type analysis of XQuery programs. Our prototype is based on the operational semantics of the language, as it is being proposed by the W3C [Draper et al., 2002]. For this task, we have used the ASF+SDF meta-environment [van den Brand et al., 2001].

In section 2 we describe XML and XQuery. Section 3 briefly describes Operational Semantics and the ASF+SDF meta-environment. In section 4 we present our XQuery static typing prototype, constructed using the ASF+SDF meta-environment. Finally, section 5 is devoted to the conclusions and contributions of our work.

2 XML and XML Query Languages

In this section we present XML and XQuery.

The Extensible Markup Language (XML) [Abiteboul et al., 2000]is a language specified by the World Wide Web Consortium (W3C) group for data interchange between two or more data sources on the Web.

XML is a language for semistructured data representation and, consequently, it can represent different kinds of information, derived from different sources. XML query languages must encompass this flexibility.

Among the query languages projected for XML it can be mentioned LOREL [Abiteboul et al., 1997], XML-QL [Deutsch et al., 1999], XML-GL [Ceri et al., 1999], XSL [Schach et al., 1998], XQL [Robie et al., 1998] and XQuery [Boag et al., 2002].

XQuery [Boag et al., 2002] is a XML query language derived from an XML query language called Quilt [Chamberlin et al., 2001]. XQuery is a strongly typed language; processing a query involves a static typing phase and a dynamic evaluation phase. XQuery has the *expressions* as its basic block. XQuery is a functional language and it allows expressions to be nested.

The expressions have results. The information that can affect these results is called the *expression context*. The expression context has two components [Boag et al., 2002]: the *Static Context* is defined as all information that is available during static analysis of the expressions, prior to its evaluation (also called static environment). The *Evaluation Context* is defined by the information that is available at the time the expression is evaluated (also called dynamic environment).

XQuery uses a type system based on XML Schema [Fallside, 2001, Biron and Malhotra, 2001]. XQuery expressions can be: primary, path, sequence, arithmetic, comparison, logical, constructors, FLWR (for, let, where, return), sorting, conditional, quantified, on datatypes and validation expressions.

The description of each of these constructions is beyond the scope of this article. For more information about XQuery, the reader is referred to [Boag et al., 2002].

The description of XQuery includes two levels of the language: XQuery and XQuery Core. The second one is a restriction of the first, capturing all the basic features of XQuery. Each command in XQuery can be expressed by one or more commands in XQuery Core.

3 Operational Semantics and ASF+SDF

In this section we present some basic concepts of operational semantics as well as some notions about the ASF+SDF meta-environment. Our prototype for the static type checking of XQuery was implemented using the meta-environment.

3.1 Operational Semantics

The formal description of languages is an area of intensive research, that contributes for the improvement of software tools, by the mathematical manipulation of program and languages.

Among the different approaches for the semantics of programming languages, we can mention *Denotational Semantics* [Schmidt, 1986, Watt, 1991], *Action Semantics* [Mosses, 1992] and *Operational Semantics* [Winskel, 1993] formalism. In these formalisms program behavior can be compositionally defined by describing the behavior of its parts: commands, declarations, expressions, among others.

In this work we use an existing operational semantics of XQuery to specify and implement a prototype of part of the language. The operational semantics shows *how* the programming language programs behave. To construct an operational definition of a language is to construct a set of rules to establish the behavior of each phrase of the language. These rules are commonly given by the inductive definition of a transition relation. The formalism known as *transition semantics* is an operational formalism that defines a *step relation* between phrases of the language. The semantics of a program is given by the reflexive and transitive closure of this relation.

Our work is concerned with the *static semantics* of XQuery (those steps to be taken before the program is run, in order to check if it is well typed).

3.2 ASF+SDF Meta-environment

ASF+SDF [van den Brand et al., 2001, Brand and Klint, 2003] is an interactive development environment for the automatic generation of interactive systems for manipulating programs, specifications, or other texts written in a formal language. ASF+SDF combines two formalisms: *Algebraic Specification Formalism* (ASF) and *Syntax Definition Formalism* (SDF) [Heering et al., 1992]. The system receives the syntax and semantics of a language and produces an environment for it.

ASF is based on the notion of a module consisting in a signature defining the abstract syntax of functions and a set of conditional equations defining their semantics. SDF allows the definition of concrete (lexical and context-free [Aho et al., 1988]) and abstract syntax of prototype language and defines a translation from text strings to abstract syntax trees. The two formalisms allow ASF+SDF to integrate definitions of syntax and semantics in a modular specification.

The structure of a module is [Brand and Klint, 2003]:

```
module <ModuleName>
  <ImportSection>*
  <ExportOrHiddenSection>*
    <Grammar>
equations
  <ConditionalEquation>*
```

A module consists of a module header (with the module name), followed by a list of zero or more import sections, followed by zero or more hidden or export sections and an optional equations section that defines conditional equations. The *import section* imports one or more external modules that are needed according to the specification. The *export section* makes all entities in the section visible to other modules. The *hidden section* makes all entities in the section local to the module.

4 The Prototype

In this section we present the prototype constructed using ASF+SDF meta-environment, based on the operational semantics of XQuery language.

The version of the formal semantics for the XQuery Core language used in this work is given in [Draper et al., 2002]. The formal definition has two hundred and three rules of static type analysis and fifty four rules of dynamic evaluation. There are both rules specified in textual form and in the form of tables. In order to implement a prototype based on this definition, each one of the syntactic and semantics rules was adapted to be implemented in ASF+SDF. The implementation of the rules is faithful to the original semantic definition in [Draper et al., 2002].

Each of the original semantic rules for XQuery Core had to be adapted for the implementation, in order to respect the rigid syntactic and semantic restrictions of the meta-environment.

An XQuery Core program is the result of the application of normalization techniques to a XQuery program. These rules have been defined in [Draper et al., 2002]. After normalization, XQuery (core) programs become more compact than the original ones, as well as non redundant.

During our implementation, some small inconsistencies and problematic aspects of the operational semantics of XQuery were found, ranging from inconsistent syntactic definitions to incomplete semantic rules. We provide an analysis of these problems in section 5.

Our prototype has sixteen modules, as illustrated in Figure 1. Each of these modules is described as follows:

- *Layout*: defines the syntax for comments and blank space in XQuery programs.
- *TerminalsSyntax*: contains the syntax of XQuery core terminals.
- *XQueryCoreSyntax*: contains the syntax of the XQuery Core language.
- *TypeSystem*: contains the syntax of types.
- *SubTypes*: contains the syntax and semantic rules for subtyping.
- *PrimeTypes*: contains the syntax and semantics of prime types in XQuery.
- *TypeInteger*: contains the syntax and semantics of integer types, adapted from [Brand and Klint, 2003].
- *TypeBoolean*: contains the syntax and semantics of boolean types adapted from [Brand and Klint, 2003].
- *StatEnvironment*: contains the (static) inference rules needed to construct the static environment of the program being checked.
- *DynEnvironment*: contains the inference rules need to construct the dynamic environment of the program being checked.

- *FunctionOnQNames*: contains the implementation of the some extra functions used in the static type analysis. Its definition follows the specification in [Malhotra et al., 2002].

- *CastingTable*: contains the implementation of cast tables that are used during the static type analysis of cast expressions and expressions of sequence types. Its definition follows the specification in [Malhotra et al., 2002].

- *Auxiliary Judgments*: contains (auxiliary) judgments that are necessary during the static type analysis of path expressions. As some of these rules use the dynamic environment, this environment was added to our prototype.

- *Auxiliary Rules*: contains the implementation of the local type environment. This environment is used only in the static type analysis of path expressions.

- *SeqTypesNormalization*: contains the implementation of the normalization of sequence types, used during the static type analysis of FLWR expressions.

- *XQueryStaticTyping*: is the main module of our prototype. It contains the description of syntax and semantics necessary to type check XQuery core expressions.

Let us now present a part of original definition of syntax and semantics of XQuery Core, as appears in [Draper et al., 2002]. Our adaptation of them to be implemented in ASF+SDF is given as well.

The formal specification of *for* expressions, according [Draper et al., 2002], is given as:

$$\frac{\begin{array}{c}\mathbf{statEnvs} \vdash \mathit{Expr}_1 : \mathit{Type} \quad ; \\ \mathbf{statEnvs}[\mathbf{varType}(\mathit{Variable}_1 : \mathbf{prime}(\mathit{Type}_1))] \\ \vdash \mathit{Expr}_2 : \mathit{Type}_2\end{array}}{\begin{array}{c}\mathbf{statEnvs} \vdash \mathbf{for}\ \mathit{Variable}_1\ \mathbf{in}\ \mathit{Expr}_1\ \mathbf{return}\ \mathit{Expr}_2 : \\ \mathit{Type}_2.\mathbf{quantifier}(\mathit{Type}_1)\end{array}}$$

$$\frac{\begin{array}{c}\mathbf{statEnvs} \vdash \mathit{Expr}_1 : \mathit{Type} \quad ; \\ \mathit{Type}_0 = [\ \mathit{SequenceType}\ ]_{\mathit{sequencetype}} \quad ; \\ \mathit{Type}_1 <: \mathit{Type}_0 \quad ; \\ \mathbf{statEnvs}[\mathbf{varType}(\mathit{Variable}_1 : \mathbf{prime}(\mathit{Type}_1))] \\ \vdash \mathit{Expr}_2 : \mathit{Type}_2\end{array}}{\begin{array}{c}\mathbf{statEnvs} \vdash \mathbf{for}\ \mathit{SequenceType}\ \mathit{Variable}_1\ \mathbf{in}\ \mathit{Expr}_1 \\ \mathbf{return}\ \mathit{Expr}_2 : \mathit{Type}_2.\mathbf{quantifier}(\mathit{Type}_1)\end{array}}$$

The versions of these rules, when expressed in the syntax of the ASF+SDF meta-environment are:

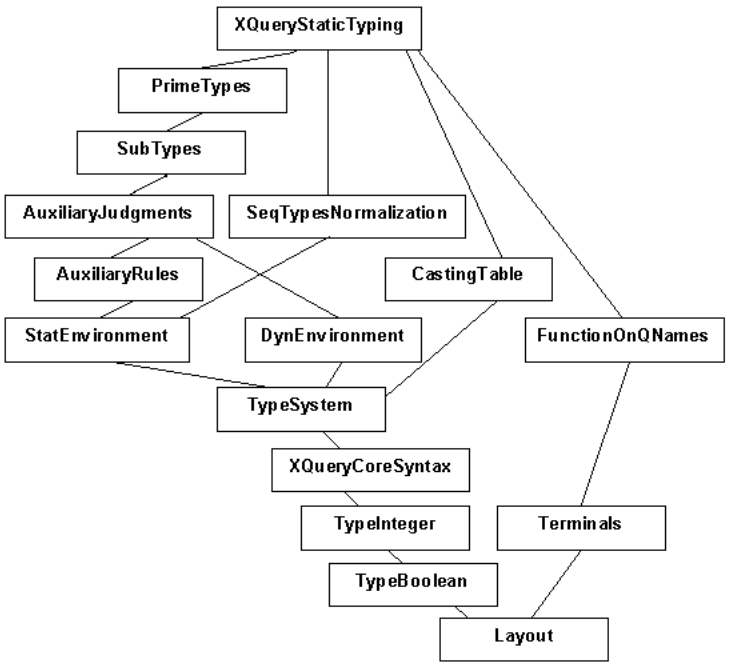

Figure 1: Architecture of the prototype.

```
[for-1] typecheck(E1, statEnvs) = T1,
        insert($ Var1 --> prime(T1)) in
                      VT of statEnvs = statEnvs1,
        typecheck(TExpr, statEnvs1) = T2
        ==================================================
        typecheck(for $ Var1 in E1 return TExpr,
                      statEnvs)          = T2.quantifier(T1)

[for-2] typecheck(E1, statEnvs) = T1,
        T0 = normSeqT(SeqT),
        subtype(T1, T0, statEnvs) = true,
        insert($ Var1 --> prime(T1)) in
                      VT of statEnvs = statEnvs1,
        typecheck(TExpr, statEnvs1) = T2
        ==================================================
        typecheck(for SeqT $ Var1 in E1 return TExpr,
                     statEnvs)          = T2.quantifier(T1)

[default-for] typecheck(For, statEnvs) = static error
```

We defined (meta) variables in ASF+SDF that correspond to objects in the XQuery specification. These (meta) variables are identified by capitalized words in the ASF+SDF specification and correspond to expressions, variables, types, static environment, among others. We have defined them by using the same patterns as in the original definition of the XQuery core language.

The `[for-1]` rule is used when a sequence type `SeqT` is not present. Initially, the first expression `E1` is verified. The function `insert`, that adds pairs to static environment, is used to insert the variable of the *for* expression into the environment's component `var-type` (`VT`). This variable is mapped to the *prime type* of expression `E1`. Finally, the expression result type is verified. The resulting type of the *for* expression is given as `T2.quantifier(T1)`.

The `[for-2]` rule is applied when a sequence type `SeqT` is present. Essentially, there are two differences between this rule and `[for-1]`. The first is the normalization function call to the sequence types variable (`normSeqT(SeqT)`). The second difference is that the subtype relation must be verified between the type of first expression `E1` and the result type `T0` of the normalized one. The resulting type of the *for* expression is given by `T2.quantifier(T1)`.

In the case in which no rule can be applied, the `[default-for]` rule is used, returning a static typing error.

The other static rules of XQuery Core were defined in the meta-environment in a similar way.

An example of use of our prototype including the results that it provides is now shown. Let us suppose that we have a program containing a *for* expression to be checked. In order to verify it, we use the function `typecheck`, which takes two arguments: a program phrase and a (static) environment containing the types of all the known variables and functions. This can be expressed in our prototype as follows:

```
typecheck(
  for $b in
    (if (eq($v1,$v2)) then $v1 else $root)
  return $dot,
  [],[],[],[],[],
  [($dot --> xs:decimal) ($b --> xs:integer)
   ($root --> xs:decimal) ($r --> xs:string)
   ($v1 --> xs:decimal) ($v2 --> xs:decimal)],
  [((eq) --> define function eq(xs:decimal, xs:decimal)
                                   returns xs:boolean)],
  []
)
```

As it can be seen, the static environment has eight components (lists). Each of these lists corresponds to each of the components for static environments specified in [Draper et al., 2002].

The result type for this phrase, returned by type checking function is:

```
[xs:decimal.quantifier(xs:decimal)]
```

The results presented by our prototype are incomplete. We have the intention of update our prototype, in accordance with the new versions of the XQuery formal specification.

5 Conclusions

Our experience in developing a prototype for the XQuery core language has shown that the static operational semantics defined for the language can be used as a base for the implementation of the typechecker. However, our implementation also pointed to some omissions and problematic issues in the working draft specification [Draper et al., 2002]. Amongst them, it can be mentioned:

- An *error* value needs to be included to the production of non-terminals *Value* and *Type* in the language's grammar. The error value is used but not defined in the specification.
- The non-terminal *Wildcard* should be added to the production rules of *ElementName* and *AttributeName*. Wildcards are used but not defined by these rules in the specification.
- Unification of occurrence indicators. The XQuery type system and the XQuery normalized grammar declare two different non-terminals for occurrence indicators. These definitions are very similar and produced parsing ambiguities.
- Inclusion of non-terminals *AttributeAll* and *ElementContent* to the XQuery type system. These non-terminals have production rules, but they do not belong to the right-hand side of any grammatical rule of the language We have included them to the productions of *ItemType*.
- The sort *LetExpr* should be included in the XQuery Core grammar. Currently there are no grammar rules associated to the Let expression.
- The rule defining node tests has a problem that needs to be fixed. The original rule is:

$$\frac{\begin{array}{c} QName_2 = Prefix_2 : LocalPart_2 \quad ; \\ \mathbf{xf:get-namepace-uri}(QName_1) = \\ \mathbf{statEnvs.varType}(Prefix_2) \quad ; \\ \mathbf{xf:get-local-name}(QName_1) = LocalPart_2 \end{array}}{\begin{array}{c} \mathbf{statEnvs} \vdash \mathbf{element}, \\ QName_2 \ \mathbf{on\ element}\ QName_1\ \{Type\} : \\ \mathbf{element}\ QName_1\ \{Type\} \end{array}}$$

This rule establishes a comparison between the value returned by `xf:get-na-mepace-uri` and the result of search in the `statEnvs.varType` environment using the `lookup` function. This equality never will be true since the `varType` environment maps variable names to their static types while `get-namespace-uri` returns a URI. The correct environment to be used; in this case is `statEnvs.namespace`. This alteration was implemented in our prototype.

The adaptation of the original operational semantics rules of XQuery Core to the ASF+SDF meta-environment was a demanding task, due to the fact that language specification does not have the immediate implementation as one of its main concerns. On the other hand, the study of the formal semantics specification proved to be very useful to enhance our understanding of the language. Our next activity in this project is to keep our prototype up-to-date with the new versions of the specification as well as to develop the prototype for the dynamic aspects of XQuery.

References

[Abiteboul et al., 2000] Abiteboul, S., Buneman, P., and Suciu, D. (2000). **Data on the Web - From Relations to Semistructured Data and XML**. Morgan Kaufmann Publishers.

[Abiteboul et al., 1997] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. L. (1997). **The Lorel query language for semistructured data**. *International Journal on Digital Libraries*, 1(1):68–88. citeseer.nj.nec.com/article/abiteboul97lorel.html.

[Aho et al., 1988] Aho, A., Sethi, R., and Ullman, J. D. (1988). **Compilers: principles, techniques, and tools**. Addison-Wesley.

[Biron and Malhotra, 2001] Biron, P. V. and Malhotra, A. (2001). **XML Schema Part 2: Datatypes**. Technical report, W3C World Wide Web Consortium. www.w3.org/TR/2001/REC-xmlschema-2-20010502.

[Boag et al., 2002] Boag, S., Chamberlin, D., Fernandez, M. F., Florescu, D., Robie, J., Simeon, J., and Stefanescu, M. (2002). **XQuery 1.0: An XML Query Language**. Working draft, W3C World Wide Web Consortium. www.w3.org/TR/2002/WD-xquery-20020430.

[Brand and Klint, 2003] Brand, M. G. J. V. D. and Klint, P. (2003). **Asf+sdf Meta-Environment User Manual - Revision 1.134**. Technical report, CWI Centrum voor Wiskunde en Informatica, Amsterdam. www.cwi.nl/projects/MetaEnv/meta.

[Bray et al., 2000] Bray, T., Paoli, J., Sperberg-McQueen, C. M., and Maler, E. (2000). **Extensible Markup Language (XML) 1.0**. Technical report, W3C World Wide Web Consortium. www.w3.org/TR/2000/REC-xml-20001006.

[Buneman, 1997] Buneman, P. (1997). **Semistructured data**. In *16th ACM Symposium on Principles of Database Systems*, pages 117–121. citeseer.nj.nec.com/buneman97semistructured.html.

[Ceri et al., 1999] Ceri, S., Comai, S., Damiani, E., Fraternali, P., Paraboschi, S., and Tanca, L. (1999). **XML-GL: A Graphical Language for Querying and Restructuring XML Documents**. In *Sistemi Evoluti per Basi di Dati*, pages 151–165. citeseer.nj.nec.com/ceri99xmlgl.html.

[Chamberlin et al., 2001] Chamberlin, D., Robie, J., and Florescu, D. (2001). **Quilt: An XML Query Language for Heterogeneous Data Sources**. *Lecture Notes in Computer Science*, 1997. citeseer.nj.nec.com/chamberlin00quilt.html.

[Deutsch et al., 1999] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., and Suciu, D. (1999). **A query language for XML**. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1155–1169. citeseer.nj.nec.com/deutsch98query.html.

[Draper et al., 2002] Draper, D., Fankhauser, P., Fernadez, M., and Malhotra, A. (2002). **XQuery 1.0 and XPath 2.0 Formal Semantics**. Working draft, W3C World Wide Web Consortium. www.w3.org/TR/2002/WD-query-semantics-20020816.

[Fallside, 2001] Fallside, D. C. (2001). **XML Schema Part 0: Primer**. Technical report, W3C World Wide Web Consortium. www.w3.org/TR/2001/REC-xmlschema-0-20010502.

[Groff and Paul N, 1999] Groff, J. R. and Paul N, W. (1999). **SQL: The Complete Reference**. Osborne McGraw-Hill.

[Heering et al., 1992] Heering, J., Hendriks, P. R. H., Klint, P., and Rekers, J. (1992). **The Syntax Definition Formalim SDF - reference manual**. ftp.cwi.nl/pub/gipe/reports/SDFmanual.ps.Z.

[Malhotra et al., 2002] Malhotra, A., Melton, J., Robie, J., and Walsh, N. (2002). **XQuery 1.0 and XPath 2.0 Functions and Operators**. Technical report, W3C World Wide Web Consortium. www.w3.org/TR/2002/WD-xquery-operators-20020816.

[Mosses, 1992] Mosses, P. D. (1992). **Action Semantics**. In *Action Semantics*. Cambridge University Press.

[Robie et al., 1998] Robie, J., Lapp, J., and Schach, D. (1998). **XML Query language(XQL)**. In *Query Languages Workshop (QL98)*. www.w3.org/TandS/QL/QL98/pp/xql.html.

[Schach et al., 1998] Schach, D., Lapp, J., and Robie, J. (1998). **Querying and Transforming XML**. In *Query Languages Workshop (QL98)*. www.w3.org/TandS/QL/QL98/pp/query-transform.html.

[Schmidt, 1986] Schmidt, D. A. (1986). **Denotational Semantics: A Methodology for Language Development**. Allyn Bacon.

[van den Brand et al., 2001] van den Brand, M., van Deursen, A., Heering, J., de Jong, H. A., de Jonge, M., Kuipers, T., Klint, P., Moonen, L., Olivier, P. A., Scheerder, J., Vinju, J. J., Visser, E., and Visser, J. (2001). **The ASF+SDF Meta-environment: A Component-Based Language Development Environment**. In *Computational Complexity*, pages 365–370.

[Watt, 1991] Watt, D. (1991). **Programming Language Syntax and Semantics**. Prentice Hall International (UK).

[Winskel, 1993] Winskel, G. (1993). **The Formal Semantics of Programming Languages: An Introduction**. Foundations of Computing Series. MIT Press.