PROGRAM VERIFICATION TECHNIQUES FOR XML SCHEMA-BASED TECHNOLOGIES

Suad Alagić, Mark Royer and David Briggs
University of Southern Maine
Department of Computer Science, Portland, ME 04104-9300

Keywords: Types, constraints, XML Schema, PVS, theories, program verification, transaction verification.

Abstract: Representation and verification techniques for XML Schema types, structures, and applications, in a program verification system PVS are presented. Type derivations by restriction and extension as defined in XML Schema are represented in the PVS type system using predicate subtyping. Availability of parametric polymorphism in PVS makes it possible to represent XML sequences and sets via PVS theories. Powerful PVS logic capabilities are used to express complex constraints of XML Schema and its applications. Transaction verification methodology developed in the paper is grounded on declarative, logic-based specification of the frame constraints and the actual transaction updates. A sample XML application given in the paper includes constraints typical for XML schemas such as keys and referential integrity, and in addition ordering and range constraints. The developed proof strategy is demonstrated by a sample transaction verification with respect to this schema. The overall approach has a model theory based on the view of XML types and structures as theories.

1 INTRODUCTION

Research on integration of programming languages and XML technologies has been based primarily on integrated type systems. These type systems have been the basis for the design of XML-oriented programming languages (Hosoya and Pierce, 2003; Benzanken et al., 2003) including those that represent extensions of major object-oriented languages with XML-oriented types and structures (Gapayev and Pierce, 2003; Bierman et al., 2004).

Our approach to this integration goes beyond an integrated type system. We take an advanced and fairly general type system as the basis, and we extend it with logic-based constraints. This approach is justified because of significant recent research efforts to extend major object-oriented languages with assertions (Leavens et al., 2005; Bierman et al., 2004), and XML models with constraint capabilities as in XML Schema (W3C, 2006a).

Constraints are critical for most database technologies, and XML Schema features some basic constraints typical for database systems, such as keys and referential integrity. Even these basic constraints cannot be expressed in a type system, let alone more complex database integrity constraints that are not expressible within the limited XML Schema constraint capabilities. Except in particular cases, typical XML Schema constraints on the range of occurrences of elements cannot be expressed in standard type systems either. This explains the importance of the research efforts to extend the paradigm of type systems with logic-based constraints.

The constraint-based view on the integration of programming languages and XML opens up the possibility of using program verification systems to verify properties related to XML Schema and its applications. A program verification system is an essential component of a programming environment that extends object-oriented languages with assertions, as in (Barnett et al., 2004). Application of these sophisticated tools to XML technologies that are based on a type system and are equipped with constraints has not been investigated. This is the main contribution of this paper.

The choice of the prover (verification system) in this research was determined by the fact that PVS has a sophisticated type system, including subtyping and bounded parametric polymorphism, and allows usage of sophisticated logics with higher-order features. A

Alagić S., Royer M. and Briggs D. (2006).
PROGRAM VERIFICATION TECHNIQUES FOR XML SCHEMA-BASED TECHNOLOGIES.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 86-93
DOI: 10.5220/0001309400860093

PVS specification consists of a collection of theories. A theory is a specification of the required type signatures (of functions in particular) along with a collection of constraints in a suitable logic applicable to instances of the theory.

The XML type hierarchy with the root type anyType, simple and complex types, and other types derived from those, is represented by a collection of PVS theories. Type derivations by extension and restriction in XML Schema (W3C, 2006a; W3C, 2006b) are defined using import of theories and the PVS notion of predicate subtyping. The XML Schema rules that govern type derivations are specified by logic based constraints. This approach is also applied to XML Schema structures (W3C, 2006b) such as element, group, and particle, so that each one of them has a corresponding theory specifying the typing rules and constraints related to those structures.

Specifying the rules of XML Schema and application-oriented schemas in terms of PVS theories has several advantages. The first one is that structural properties are expressed in a type system that conforms to well-established type systems of programming languages with subtyping and parametric polymorphism. The second advantage is that complex rules specified in the XML Schema documents in semi-formal English are now precisely specified in PVS theories in a suitable logic. Likewise, specification of a variety of constraints in an application schema is now both required and possible in a more general formal logic-based framework. The complexity of constraints, the XML Schema rules in particular, makes the usage of an automated theorem prover tool useful in detecting human errors that might otherwise go unnoticed.

Of course, the most important advantage is that PVS allows automated reasoning about a variety of properties expressed in its theories. This applies even to application properties and requirements that are not expressible in XML Schema. Thus, reasoning and verification are supported in situations when XML data is processed by a transaction or a general purpose programming language, as illustrated by a transaction verification proof strategy presented in the paper.

The most basic constraints in XML Schema are related to the ranges of occurrences of elements in a sequence. A core idea behind type derivations in XML Schema is that an instance of a derived type may be viewed as a valid instance of its base type. This includes the requirement that all constraints associated with the base type are still valid when applied to an instance of a derived type. The overall approach in constructing PVS theories for XML Schema types and structures is governed by this basic requirement. The formal model theory based on the view of types as theories has also been developed with this compatibility requirement as its cornerstone. This requirement corresponds to the notion of behavioral subtyping in object-oriented programming languages (Liskov and Wing, 1994).

One particularly important application of a prover technology is verification that a transaction respects the integrity of a schema equipped with the above type of constraints. In this paper we present a typical application and develop a suitable proof methodology and implement it in PVS. The pragmatic goal is to make static deductions that would avoid expensive run-time violations of database integrity or at least reduce the amount of dynamic checks by verifying some conditions statically.

The methodology developed in the paper requires explicit specification of the frame constraints of a transaction. The frame constraints specify the integrity constraints which the transaction does not affect. In addition, the active part (the actual update) that a transaction performs is specified in a declarative, logic-based style, and the verification is carried out using a proof strategy presented in the paper. This methodology is independent of a particular transaction language.

2 XML TYPES AS PVS THEORIES

In XML Schema, anyType is the root of the XML type hierarchy (W3C, 2006a; W3C, 2006b). All other XML types are directly or indirectly derived from anyType by restriction or extension. In our PVS representation two types that are derived directly from the type `XMLany` are `XMLsimple` and `XMLcomplex`. TYPE+ denotes a nonempty type.

```
XMLany: THEORY
BEGIN  XMLany: TYPE+
 % body of theory XMLany
END XMLany
```

```
XMLsimple: THEORY
BEGIN  IMPORTING XMLany
       XMLsimple: TYPE+ FROM XMLany
 % body of theory XMLsimple
END XMLsimple
```

A complex type is always derived from some other type, which may be either simple or complex. The predicates `restriction?` and `extension?` in the theory `XMLcomplex` below determine which one of the two derivation techniques is being used for the underlying complex type.

A complex XML type is equipped with a set of attributes and a content. The content of a complex type is potentially very complex and its structure is, in our approach, determined in a theory corresponding to the XML notion called particle specified in section 3. Three options for an XML content model are

specified below: a complex type may have empty content, indicated by the predicate `emptyContent?`, simple content indicated by `simpleContent?`, or complex content indicated by `complexContent?`. If the content is simple, the function `simpleValue` returns the complex type's simple value, and if the content is complex the function `complexValue` returns its complex value.

```
XMLcomplex: THEORY
BEGIN
     IMPORTING XMLsimple, XMLattribute,
               XMLparticle
     XMLcomplex: TYPE+ FROM XMLany
attributes:
 [XMLcomplex -> XMLset[XMLattribute]]

restriction?: [XMLcomplex -> bool]
extension?:   [XMLcomplex -> bool]

emptyContent?:  [XMLcomplex -> bool]
simpleContent?: [XMLcomplex -> bool]
simpleValue:
      [(simpleContent?) -> XMLsimple]
complexContent?: [XMLcomplex -> bool]
content:
     [(complexContent?) -> XMLparticle]
complexValue:
[(complexContent?) ->
               XMLsequence[XMLelement]]
% constraints for simpleRestrict
% and complexRestrict
% constraints for simpleExtend
% and complexExtend
END XMLcomplex
```

The above theory is a simplification of the XML Schema for presentation purposes. In PVS notation, `(p?)` denotes a type whose elements satisfy the predicate `p`. The omitted constraints are described below. The `simpleRestrict` constraint specifies the XML rule for deriving a complex type with simple content from another complex type with simple content. The core property is that any constraint that holds for all instances of the base type must also hold for the derived complex type. In other words, the derived type is only allowed to further restrict the constraints in the base type. Likewise, the `complexRestrict` constraint applies to types with complex content derived by restriction. The predicate `restricts` defined in the theory `XMLparticle` allows only strengthening the constraints from the base type with no changes to the underlying structure.

The `simpleExtend` constraint specifies the XML rule for deriving a complex type by extension from a simple type. In this case the underlying simple value remains the same. The only change is in possibly adding attributes. The constraint `complexExtend` specifies the XML rule for extending complex types with complex content. This constraint refers to the predicate `extends` defined in the theory `XMLparticle` which specifies valid structural extensions of the content of the base type into the content of the derived type.

3 XML STRUCTURES

XML structures are composed of attributes, elements, groups, etc. The general XML notion for specifying XML content is particle (W3C, 2006b). The subtyping relationships among XML types and the types specifying XML structures are represented in figure 1.

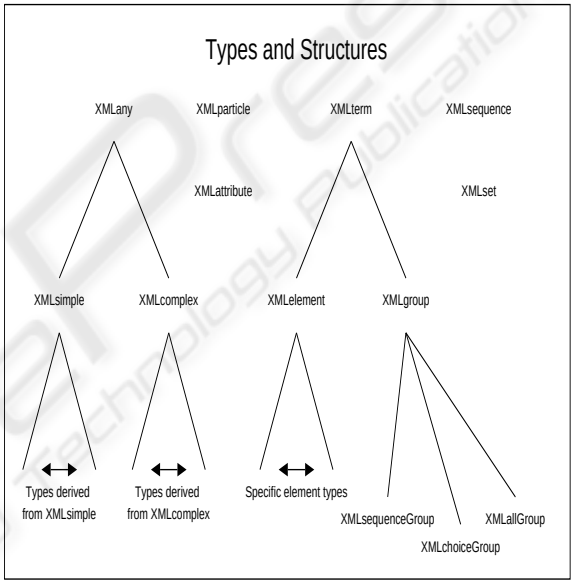

Figure 1: XML types and structures.

The PVS theory `XMLparticle` given below specifies the main features of XML structures. The structural part of a particle is described by its term that is returned by the function `particleTerm`. The number of allowed occurrences of the term in a particle is specified by two functions, `minOccurs` and `maxOccurs`. The associated range constraint specifies the constraint on the range of occurrences.

A term may be an element or a group, hence `XMLelement` and `XMLgroup` are defined as XML subtypes of `XMLterm`. There are three types of groups: a sequence group, a choice group, and an all group, hence the predicates `sequenceGroup?`, `choiceGroup?`, and `allGroup?`. A sequence group is a sequence of particles. A choice group specifies a selection of one out of a finite number of particles. An all group specifies a set of elements that may appear in any order.

```
XMLparticle: THEORY
BEGIN
     IMPORTING XMLany, XMLattribute,
               XMLsequence, XMLset
     XMLparticle: TYPE+
     XMLterm:     TYPE+

 particleTerm: [XMLparticle -> XMLterm]
 minOccurs: [XMLterm -> nat]
 maxOccurs: [XMLterm -> nat]
 unbounded: nat
 % range constraint

 XMLelement:   TYPE+ FROM XMLterm
 element?:     [XMLterm -> bool]
 elementName:  [XMLelement -> string]
 elementValue: [XMLelement -> XMLany]

 XMLgroup:        TYPE+ FROM XMLterm
 sequenceGroup?:  [XMLgroup -> bool]
 choiceGroup?:    [XMLgroup -> bool]
 allGroup?:       [XMLgroup -> bool]
 groupParticles:  [XMLgroup ->
            XMLsequence[XMLparticle]]
% specification of extends
% and restricts predicates
END XMLparticle
```

The predicates `extends` and `restricts` specify valid extensions and restrictions of XML particles. The `extends` constraint specifies that the extension is either equal to the original particle or simply extends the original particle by appending another particle. The `restricts` constraint specifies that a particle that extends another particle is equal to the original one and the only change that is allowed is restricting the range of the original particle. Otherwise, this condition is applied recursively where the term of the particle and the restriction base term are requested to be either the same element or the same type of a group and the corresponding elements in their underlying sequence of particles are requested to satisfy the restrict predicate. For presentation purposes, the above is a simplification of the very complicated conditions specified in XML Schema.

An abbreviated theory `XMLtags` given below shows how the tag language associated with an XML structure (element, term, or particle) is defined. The tag language of a particle constrains the sequence of elements of the particle. The type `XMLtags` is specified as a set of sequences of tags along with the standard operators for regular languages: union, concatenation and iteration. This theory contains constraints specifying the semantics of these operators. More importantly, this theory also contains constraints that specify recursively the tag language of a particle, which is expressed in terms of the tag language of a term and the `minOccurs` and `maxOccurs` constraints of the particle. The tag language of a term (also specified recursively) is based on the rules for constructing the tag language for the group operators which make use of the operators of the `XMLtags` theory (specifically, `conCat` for sequence and `union` for choice). These rules are omitted due to their complexity.

```
XMLtags: THEORY
BEGIN
     IMPORTING XMLparticle
     XMLtag:  TYPE+ FROM string
     XMLtags: TYPE+ =
            XMLset[XMLsequence[XMLtag]]
conCat:  [XMLtags, XMLtags -> XMLtags]
star:           [XMLtags -> XMLtags]
starPlus:       [XMLtags  -> XMLtags]
% constraints for union, concatenate
% and star
seq:       [string -> XMLsequence[XMLtag]]
elementTags(e): XMLtags =
            singleton(seq(elementName(e)))
particleTags:   [XMLparticle -> XMLtags]
termTags:       [XMLterm -> XMLtags]
% constraints for termTags
% and particleTags
END XMLtags
```

4 COMPATIBILITY OF TYPE DERIVATIONS

The approach developed in this paper has a formal basis in model theory that has been applied to a variety of programming and database paradigms (Goguen, 1991; Alagić and Kouznetsova, 2002; Alagić and Bernstein, 2002; Alagić and Briggs, 2004).

The first component of this particular model theory is a formal definition of a type equipped with logic based constraints that matches the PVS notion of a theory and captures XML Schema types and structures along with constraints. The relationships among types equipped with constraints such as those expressed by type derivations in XML Schema are represented as morphisms of theories. In fact, the core of the model theory captures precisely XML Schema type derivations ensuring that the compatibility conditions expressed by XML constraints are satisfied in those derivations.

The key notion of the model theory is the satisfaction of constraints. Specification of a type structure equipped with constraints requires an interpretation in terms of sets of instances along with operations on those instances. This interpretation is called a model.

A type theory $Th = (\Sigma, E)$ consists of a type signature Σ and a finite set E of Σ sentences. A sentence is a closed formula, i.e., a formula with all variables quantified.

Given a theory (Σ, E), $Mod(\Sigma)$ denotes a collection of Σ models and $\models$ denotes the satisfaction re-

lation between models and Σ sentences. A Σ model provides an interpretation of the signatures of functions given in Σ. The fact that a model M satisfies a sentence e is thus denoted as

$M \models e$.

If $Th_A = (\Sigma_A, E_A)$ and $Th_B = (\Sigma_B, E_B)$ are type theories, $\phi : Th_A \rightarrow Th_B$ is a theory morphism iff $\phi : \Sigma_A \rightarrow \Sigma_B$ is a type signature morphism such that for all Σ_B models M_B, $M_B \models E_B$ implies $M_B \models \phi(e)$ for all sentences $e \in E_A$.

A theory morphism maps a sentence e from the source theory Th_A to the target theory Th_B in such a way that the transformed sentence $\phi(e)$ fits into the target theory, i.e., it belongs to the set of all sentences of the target theory. This is a semantic requirement rather than a typing requirement.

Import of theories in PVS is a specific case of a theory morphism in which the signature Σ_A is a subsignature of Σ_B and E_A is a subset of E_B.

The relationship of a theory Th_B representing an XML Schema type derived from another type represented by a theory Th_A is expressed in PVS by combining import of theories and the PVS notion of predicate subtyping.

This representation technique has the following form in the PVS notation.

```
A: THEORY
BEGIN
% body of theory A
END A

B: THEORY
BEGIN IMPORTING A
      B: TYPE FROM A
  % body of theory B
END B
```

The statement `B: TYPE FROM A` that defines `B` as a PVS subtype of `A` is equivalent to a definition of a predicate `B_pred:[A -> bool]` and defining `B` as a type that satisfies `B_pred`.

PVS also supports bounded parametric polymorphism. Full details are elaborated in (Alagić et al., 2006).

Two parametric PVS theories `XMLset` and `XMLsequence` are not given in the paper but are used to specify a set of attributes and a sequence of elements. These theories are represented by adapting parametric theories for sets and sequences from the standard PVS prelude.

5 APPLICATION SCHEMAS

In this section we specify a sample schema equipped with a variety of constraints that include uniqueness, referential integrity, ordering, and ranges of values. The theory `XMLprojectManagement` contains specification of a sequence of contracts and a sequence of projects. The corresponding theories make subtle use of options available in the PVS type system, illustrated in the theory `XMLproject`.

The three components of this specification are: the type information associated with an element type, the type signatures of accessor functions, and the constraints. The type information for subelements and attributes is represented by record types. However, because of repetition of subelements, `XMLproject` type itself is not represented as a record, since that would not be an accurate representation with respect to XML. The repetition is expressed via `minOccurs` and `maxOccurs` constraints and also by specifying the tag language of `XMLproject`. In addition to the above two components (type structure and constraints), the third component consists of accessor functions that apply to an instance of an `XMLproject`. Note that the accessor function `contracts` returns a sequence of `XMLcontract` elements.

```
XMLproject: THEORY
BEGIN
     IMPORTING XMLcomplex, XMLcontract,
               XMLstring, XMLattribute
     XMLproject: TYPE+ FROM XMLelement

  XMLprojectElements: TYPE  =
        [# leader: string, funds: real,
           contract:  XMLcontract #]
  XMLprojectAttributes: TYPE =
              [# projectId: string #]

  projectElements: [XMLproject ->
                    XMLprojectElements]
  projectAttributes: [XMLproject ->
                   XMLprojectAttributes]
  leader:    [XMLproject -> string]
  contracts: [XMLproject ->
             XMLsequence[XMLcontract]]
  funds:     [XMLproject -> real]

  fundsConstraint(p: XMLproject): bool =
  (funds(projectElements(p))) >= 1000000

  contractElementsConstraint(p:
                      XMLproject): bool =
  minOccurs(contract(
            projectElements(p)))>= 1 AND
  maxOccurs(contract(projectElements(p)))
                              = unbounded

  elementTags(p: XMLproject): XMLtags =
         conCat(singleton(seq("leader")),
    conCat(singleton(seq("funds")),
 starPlus(singleton(seq("contract")))) )
END XMLproject
```

The theory `XMLprojectManagement` specifies two constraints typical for XML Schema. The

uniqueness constraint specifies that contract numbers uniquely determine contracts in the sequence of contracts. The referential constraint specifies that contracts of projects in the sequence of projects exist in the sequence of contracts. In addition to the above two, the ordering constraint specifies that contracts appear in the sequence of contracts in the increasing order of their contract numbers. There is also a self-explanatory fundsRange constraint.

```
XMLprojectManagement: THEORY
BEGIN
  IMPORTING XMLcomplex, XMLcontract,
   XMLproject, XMLsequence, XMLschema
  XMLprojectManagement:
                   TYPE+  FROM XMLschema

  projects:   [XMLprojectManagement ->
                XMLsequence[XMLproject]]
  contracts:  [XMLprojectManagement ->
               XMLsequence[XMLcontract]]

M: VAR XMLprojectManagement
p: VAR XMLproject
c: VAR XMLcontract

 uniquenessConstraint(M): bool =
           (FORALL (c1,c2: XMLcontract):
       member(contracts(M),c1)
          AND member(contracts(M),c2) AND
  contractNo(contractAttributes(c1)) =
      contractNo(contractAttributes(c2))
                        IMPLIES c1 = c2)

referentialConstraint(M): bool =
(FORALL (p,c): (member(projects(M),p) AND
    contract(projectElements(p)) = c)
                          IMPLIES
    (EXISTS (c1:XMLcontract):
             (member(contracts(M),c1) AND
   (contractNo(contractAttributes(c1)) =
    contractNo(contractAttributes(c))))))

 orderingConstraint(M): bool =
          (FORALL (c1,c2: XMLcontract,
     n1,n2: below(length(contracts(M)))):
              member(contracts(M),c1) AND
              member(contracts(M),c2) AND
 contractNo(contractAttributes(c1)) <=
   contractNo(contractAttributes(c2)) AND
 nth(contracts(M))(n1) = c1 AND
        nth(contracts(M))(n2) = c2
                    IMPLIES n1 <= n2)

 fundsRange(M): bool =
 (FORALL (n: below(length(projects(M)))):
    fundsConstraint(nth(projects(M))(n)))

 consistent(M): bool =
             uniquenessConstraint(M) AND
            referentialConstraint(M) AND
              orderingConstraint(M)  AND
                fundsRange(M)
END XMLprojectManagement
```

6 TRANSACTION VERIFICATION

In this section we show how transactions are specified in a declarative, logic-based style, and demonstrate a proof strategy applied to verify the transaction safety with respect to the schema constraints.

XMLprojectTransaction is a transaction theory that contains specification of both the frame constraint and the actual update that the transaction performs. The frame constraint specifies which of the integrity constraints are not affected by the transaction. This particular transaction only updates contract funds and hence it has no impact on the uniqueness, referential, and ordering constraints.

Explicit specification of the frame constraints is essential in our proof strategy and guides the prover appropriately. The actual update that the transaction performs is specified in a declarative fashion as a predicate over a pair of object states, the states before and after transaction execution. A transaction is then a binary predicate specified as a conjunction of its frame constraint and the actual update constraint.

```
XMLprojectTransaction: THEORY
BEGIN
    IMPORTING XMLprojectManagement
    M1,M2: VAR XMLprojectManagement

 frameAx(M1,M2): bool =
     (uniquenessConstraint(M1) IMPLIES
           uniquenessConstraint(M2)) AND
     (referentialConstraint(M1) IMPLIES
         referentialConstraint(M2)) AND
           (orderingConstraint(M1)
        IMPLIES  orderingConstraint(M2))

 update(M1,M2): bool =
            length(projects(M1)) =
            length(projects(M2)) AND
 FORALL (n:below(length(projects(M2)))):
  (funds(projectElements(
                nth(projects(M2))(n))) =
   funds(projectElements(
        nth(projects(M1))(n)))+ 100000)

 transaction(M1,M2): bool =
                  frameAx(M1,M2) AND
                   update(M1,M2)
END XMLprojectTransaction
```

In order to prove that a transaction conforming to the above theory maintains the integrity of the XMLprojectManagment database, the following theory is constructed. To simplify the proof, a simple

update lemma is proved first. The integrity theorem is then proved using the update lemma.

```
VerifyProjectTransaction: THEORY
BEGIN
     IMPORTING XMLprojectTransaction
     M1,M2: VAR XMLprojectManagement

updateLemma: LEMMA fundsRange(M1) AND
  update(M1,M2) IMPLIES fundsRange(M2)

Integrity: THEOREM FORALL (M1,M2):
  consistent(M1) AND transaction(M1,M2)
                 IMPLIES consistent(M2)
END VerifyProjectTransaction
```

Consider an example of a characterization of a transaction update that violates the referential integrity constraint and hence its `Integrity` theorem fails. Let us define `badUpdate` as

```
badUpdate(M1,M2): bool =
          length(projects(M1)) > 0 AND
       projects(M2) = projects(M1) AND
            length(contracts(M2)) = 0
```

This update does not affect the sequence of projects but it deletes all contracts which is an obvious violation of referential integrity. The PVS attempted proof of the `updateLemma` leads to a contradiction demonstrating violation of integrity.

7 RELATED RESEARCH

The types as theories view is based on (Goguen, 1991). This view is also the basis of previous results on behavioral compatibility problems for object-oriented languages (Alagić and Kouznetsova, 2002), generic data model management (Alagić and Bernstein, 2002), and semantics of objectified XML (Alagić and Briggs, 2004).

A classical result on the application of theorem prover technology based on computational logic to the verification of transaction safety is (Sheard and Stemple, 1989). Other results include (Benzanken and Schaefer, 1997), usage of Isabelle/HOL (Spelt and Even, 1999), and PVS (Alagić and Logan, 2004).

A variety of recent results address the problems of integration of a type system for XML with standard type systems (Gapayev and Pierce, 2003; Benzanken et al., 2003; Hosoya and Pierce, 2003; Hosoya et al., 2005; Bierman et al., 2004; Simeon and Wadler, 2003). However, these results are confined to the problems of an integrated type system, and do not address the issue of logic-based constraints, which is a distinctive feature of our work.

A variety of results are available on constraints for XML such as (Fan and Simeon, 2003; Buneman et al., 2002; Kuper and Simeon, 2001). We consider XML constraints associated with a type system (Alagić and Briggs, 2004), and provide a prover technology to reason about constraints. This is probably the most distinctive feature of our work with respect to other related results.

In a separate piece of research (Alagić et al., 2006) we make use of a temporal logic specified by a suitable PVS theory in order to prove properties of object-oriented programs. This makes it possible to apply the prover technology to constraint based object-oriented technologies such as JML (Leavens et al., 2005) and to technologies that integrate the object-oriented technology with XML (Bierman et al., 2004).

8 CONCLUSIONS

Our experience in using PVS shows that intuitive verification techniques, even if they are largely formal, are inadequate. We discovered over and over again that in human generated proofs there are many assumptions that are not carefully specified while correctness of proofs depends critically upon those assumptions. One advantage of a prover is that it requires careful specification of all the relevant constraints and assumptions, otherwise the most obvious simple properties are not provable.

Another issue is that PVS does not check consistency of a collection of axioms. This means that systems of axioms should be avoided when possible since they may be inconsistent and hence the proofs would not be valid. This is why a definitional style has been used in the paper, specifying a variety of constraints as formulas, and using PVS to prove that the desired properties follow from these definitions.

A further conclusion is that tools such as PVS are not easy to use and require expertise and experience. A valid research goal is to develop proof strategies for particular tasks following the guidelines in (Owre and Shankar, 2005; Archer et al., 2003). For a transaction verification proof strategy, a critical issue was separation of frame constraints from the logic-based specification of the actual updates. This strategy avoids expanding and rewriting the frame constraints and makes it possible to focus on the details of the proof of the active part of a transaction. In order to make these tools usable by typical programmers, a high-level user friendly interface based on suitable proof strategies is really required.

From the research viewpoint the availability of a fairly sophisticated type system in PVS which includes a particular form of subtyping and bounded parametric polymorphism was essential. This made it possible to use the PVS predicate subtyping to specify the semantic compatibility conditions of type derivations in XML Schema and represent sequences of ele-

ments and sets of attributes using parametric theories. Higher-order features of PVS also proved to be very important.

The compromise between static and dynamic checking amounts to treating constraints that are meant to be checked at run-time as axioms and proving the remaining ones as theorems. The proofs will be valid as long as the truth of the axioms is guaranteed at run-time. At the same time the constraints that are proved as theorems under the above assumptions will not be checked at run-time. This improves efficiency of dynamic checking of constraints and reliability of transactions.

REFERENCES

Alagić, S. and Bernstein, P. A. (2002). A model theory for generic schema management. In *Proceedings of DBPL 2001, Lecture Notes in Computer Science, 2397, pp. 228 - 246*. Springer.

Alagić, S. and Briggs, D. (2004). Semantics of objectified XML. In *Proceedings of DBPL 2003, Lecture Notes in Computer Science, 2921, pp. 147-165*. Springer.

Alagić, S. and Kouznetsova, S. (2002). Behavioral compatibility of self-typed theories. In *Proceedings of ECOOP 2002, Lecture Notes in Computer Science, 2374, pp. 585-608*. Springer.

Alagić, S. and Logan, J. (2004). Consistency of Java transactions. In *Proceedings of DBPL 2003, Lecture Notes in Computer Science, 2921, pp. 71-89*. Springer.

Alagić, S., Royer, M., and Crews, D. (2006). Temporal verification of Java-like classes. In *Proceedings of the ECOOP 2006 FTfJP Workshop (Formal Techniques for Java like Programs)*. http://www.disi.unige.it/person/AnconaD/FTfJP06/.

Archer, M., Vito, B. D., and Munoz, C. (2003). Developing user strategies in PVS: A tutorial. In *Proceedings of STRATA 2003*.

Barnett, M., Rustan, K., and Schulte, W. (2004). The Spec# programming system: an overview. In *Microsoft Research 2004, also in Proceedings of CASSIS 2004*.

Benzanken, V., Castagna, G., and Frisch, A. (2003). Cduce: An XML-centric general-purpose language. In *Proceedings of ICFP 2003, pp. 51-63*. ACM.

Benzanken, V. and Schaefer, X. (1997). Static integrity constraint management in object-oriented database programming languages via predicate transformers. In *Lecture Notes in Computer Science, 1241, pp. 60-84*. Springer.

Bierman, G., Meijer, E., and Schulte, W. (2004). The essence of data access in c_ω. In *Microsoft Research*.

Buneman, P., Davidson, S., Fan, W., Hara, C., and Tan, W.-C. (2002). Reasoning about keys for XML. In *Proceedings of DBPL 2001, Lecture Notes in Computer Science, 2397, pp.133-148*. Springer.

Fan, W. and Simeon, J. (2003). Integrity constraints for XML. In *Journal of Computer and System Sciences 66, pp. 254-291*.

Gapayev, V. and Pierce, B. (2003). Regular object types. In *Proceedings of ECOOP 2003, Lecture Notes in Computer Science, 2743, pp. 151-175*. Springer.

Goguen, J. (1991). Types as theories. In *G. M. Reed, A. W. Roscoe and R. F. Wachter, Topology and Category Theory in Computer Science, pp. 357-390*. Clarendon Press, Oxford.

Hosoya, H., Frisch, A., and Castagna, G. (2005). Parametric polymorphism for XML. In *Proceedings of POPL 2005, pp. 50-62*. ACM.

Hosoya, H. and Pierce, B. (2003). XDuce: A typed XML processing language. In *ACM Transactions on Internet Technology, 3(2), pp. 117-148*. ACM.

Kuper, G. M. and Simeon, J. (2001). Subsumption for XML types. In *Proceedings of ICDT, Lecture Notes in Computer Science, 1973, pp. 331-345*. Springer.

Leavens, G. T., Poll, E., Clifton, C., Cheon, Y., Ruby, C., Cook, D., Muller, P., and Kiniry, J. (2005). *JML Reference Manual*. http://www.cs.iastate.edu/ leavens/JML/, Iowa State, draft edition.

Liskov, B. and Wing, J. M. (1994). A behavioral notion of subtyping. In *ACM Transactions on Programming Languages and Systems, pp. 1811-1841*. ACM.

Owre, S. and Shankar, N. (2005). *Writing PVS proof strategies*. SRI International, http://www.csl.sri.com.

Sheard, T. and Stemple, D. (1989). Automatic verification of database transaction safety. In *ACM Transactions on Database Systems 14, pp. 322-368*. ACM.

Simeon, J. and Wadler, P. (2003). The essence of XML. In *Proceedings of POPL 2003, pp. 1-13*. ACM.

Spelt, D. and Even, S. (1999). A theorem prover-based analysis tool for object-oriented databases. In *Lecture Notes in Computer Science, 1579, pp 375 - 389*. Springer.

W3C (2006a). *W3C: XML Schema Part 0: Primer*. W3C, http://www.w3.org/TR/xmlschema-0/, second edition.

W3C (2006b). *W3C: XML Schema Part 1: Structures*. W3C, http://www.w3.org/TR/xmlschema-1/, second edition.