

Checking XML-Specifications

Harald Hiss, University Freiburg
hiss@informatik.uni-freiburg.de

June 20, 2007

Abstract

New developments in databases build on XML-technologies. Concepts of the relational model can be transferred to XML. This approach is not unproblematic, transfer of integrity constraints causes a problem. Some XML-specifications are unsatisfiable.

The development of a deductive checker is presented. An extensive formalization developed with Isabelle integrates circular XML-specifications with an inductive method. Circular XML-specifications are unsatisfiable. A checker generates F-Logic facts that are checked with Florid. The theorems of the formalization prove the correctness of the checker.

1 Development of a Checker

New developments in databases build on XML [BPSM⁺06] technologies. Concepts of the relational model [AHV95] can be transferred to XML. This approach is not unproblematic, transfer of integrity constraints causes a problem. Some XML-specifications are unsatisfiable.

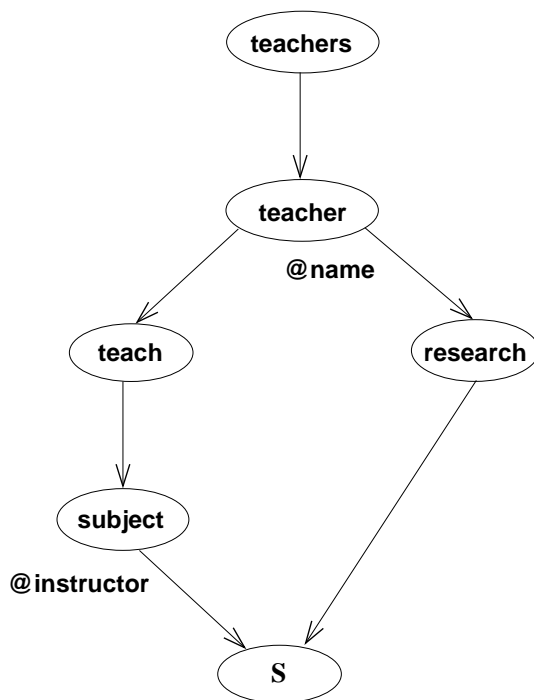
A deductive checker for XML-specifications is presented. In [FL02], results are presented for the complexity of checking XML-specifications. Implication of integrity of the relational model that is undecidable [CV85] is represented with XML-specifications. The complexity of restricted XML-specifications is proven. In [His07], a transformation for model checking XML-specifications is presented. The transformation generates constraints. A model checker proves the satisfiability of the constraints. An extensive formalization developed with Isabelle [Pau94b] proves the correctness of the constraints. An inductive method [Pau94a] formalizes circular XML-specifications that are unsatisfiable. A deductive checker for XML-specifications is presented based on circular XML-specifications. The checker generates F-Logic [KLW95] facts that are checked with Florid [HLS07]. The formalization proves the correctness of the checker.

The next section introduces unsatisfiable XML-specifications with a database of teachers. Section 3 presents a formalization of XML-specifications illustrated with the example. Then section 4 formalizes circular XML-specifications. Section 5 presents theorems to prove that circular XML-specifications are unsatisfiable in section 6. Then section 7 presents a deductive checker for XML-specifications and section 8 concludes the contribution.

2 A Database of Teachers

The section presents an unsatisfiable example. An XML-specification models a database of teachers. The structural schema in figure 1 defines elements (*teachers*, *research*, *subject*) and attributes (*name*, *instructor*). Content models form a structure for XML-trees. The root labeled *teachers* stores content model *teacher*⁺. XML-trees of the structural schema have a *teachers* root with *teacher* children. Figure 2 contains an example XML-tree, the next section presents details. Attribute *instructor* represents a teacher. Integrity considers attributes of XML-trees. Keys and inclusion constraints formalize integrity. Key *teacher.name* \rightarrow *teacher* represents *teacher*

Figure 1 A graph visualizes the structural schema of the example.



with *name*. Inclusion constraint $subject.instructor \subseteq teacher.name$ represents the dependency of *instructor* of *subject* on *name* of *teacher*.

$$\begin{aligned}
 &teacher.name \rightarrow teacher \\
 &subject.instructor \rightarrow subject \\
 &subject.instructor \subseteq teacher.name
 \end{aligned}$$

The XML-tree presented in figure 2 satisfies $teacher.name \rightarrow teacher$. The *teacher* nodes store the names *Dr. Brett* and *Prof. Crey*. The example satisfies $subject.instructor \subseteq teacher.name$, the names contain the instructors. The example doesn't satisfy $subject.instructor \rightarrow subject$, two subjects store *Dr. Brett*. The example is unsatisfiable. The structural schema contains a branch. The *teacher* nodes, formalized with $ext(teacher)$, have two *subject* descendants.

$$2|ext(teacher)| \leq |ext(subject)|$$

A document has at least one *teacher*.

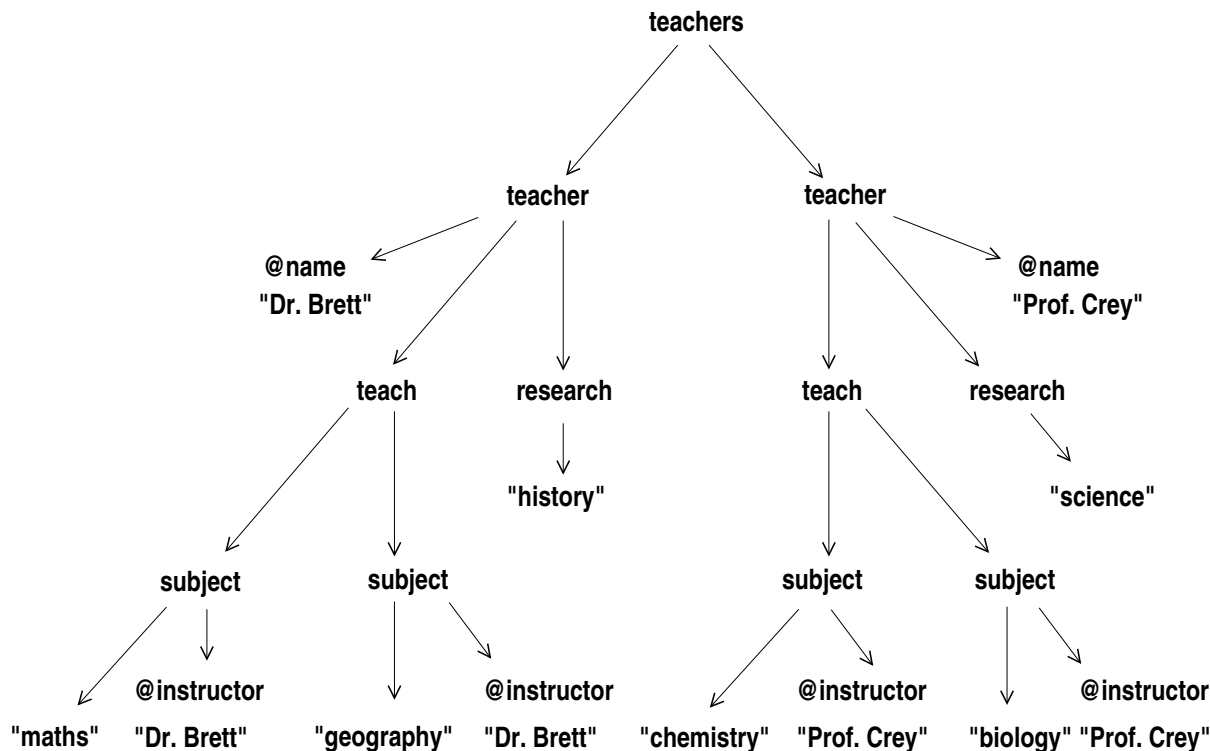
$$|ext(teacher)| < |ext(subject)|$$

Key $subject.instructor \rightarrow subject$ and inclusion constraint $subject.instructor \subseteq teacher.name$ prove a contradiction. The next section presents the formalization of XML-specifications that forms the fundament for the checker presented in section 7.

3 Formalization of XML-Specifications

The section presents a formalization of XML-specifications illustrated with the database of teachers. The formalization is the fundament for the following sections. The section formalizes the structural schema and integrity of XML-documents. A Document Type Definition

Figure 2 An example XML-tree stores two teachers.



(DTD [BPSM⁺06]) defines a structural schema with attributes A ($name$, $instructor$) and elements E ($teachers$, $subject$) that define a root r . The example in figure 3 defines root $teachers$. The attributes of an element are stored with function R , $teacher$ stores attribute $name$. Function P stores the content models. Element $teachers$ stores $teacher^+$. Regular expressions [HMU06] are formalized with inductive data types [BW]. Concatenation, union, star, plus and question mark form content models with labels $\tau \in E$, text \mathbf{S} and empty content ϵ .

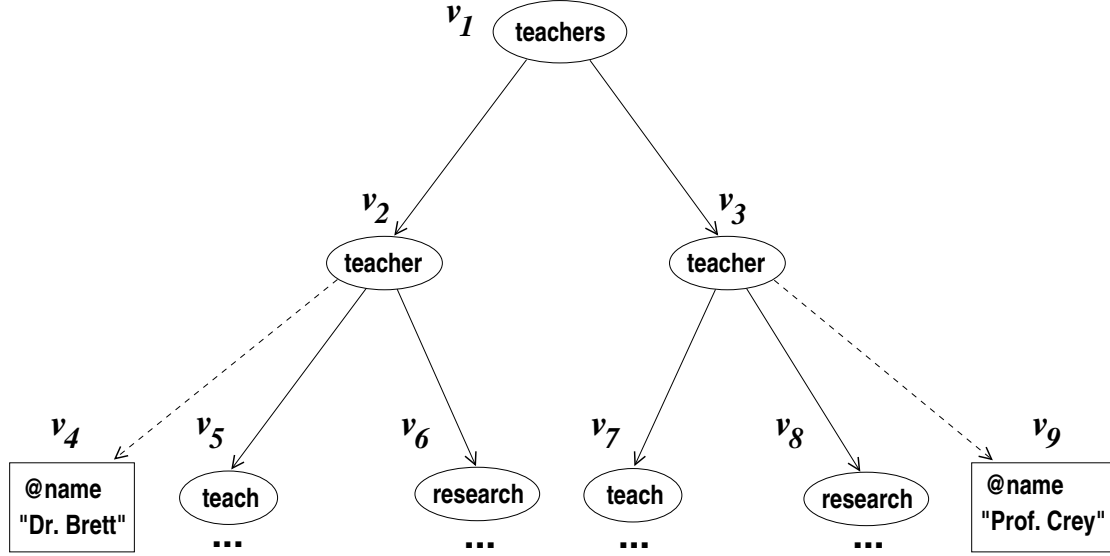
$$\alpha ::= \epsilon \mid \mathbf{S} \mid \tau \mid (\alpha, \alpha) \mid (\alpha|\alpha) \mid \alpha^* \mid \alpha^+ \mid \alpha?$$

Wellformed structural schemas are formalized. The sets A and E are disjoint and don't include \mathbf{S} . The functions P and R are defined for E . Labels in $(E \cup \{\mathbf{S}\}) \setminus \{r\}$ form content models that connect r with the elements. XML-trees are formalized with nodes V . The example XML-tree in figure 4 includes the nodes v_1, v_2, \dots, v_9 . Function lab stores labels in $A \cup E \cup \{\mathbf{S}\}$, $teacher$

Figure 3 A DTD defines the structural schema of figure 1.

```
<!DOCTYPE teachers [
  <!ELEMENT teachers (teacher+)>
  <!ELEMENT teacher (teach, research)>
  <!ELEMENT teach (subject, subject)>
  <!ELEMENT research (#PCDATA)>
  <!ELEMENT subject (#PCDATA)>
  <!ATTLIST teacher name CDATA #REQUIRED>
  <!ATTLIST subject instructor CDATA #REQUIRED> ]>
```

Figure 4 A detailed view of the XML-tree in figure 2.



is stored for v_2 and $name$ for v_4 . Children are stored with ele . Parents are unique. Nodes $[v_2, v_3]$ are the children of v_1 that represents $root$, the unique node labeled r . Attribute nodes are stored with att . The example defines $name$ for $teacher$. For v_2 and this attribute att stores v_4 . Function val stores text of nodes with a label in $A \cup \{\mathbf{S}\}$, v_4 stores $Dr. Brett$. Text nodes don't have children. Label $name$ proves v_4 doesn't have children.

$$ext(\tau) = \{v \mid v \in V \wedge lab\ v = \tau \wedge \tau \in E \cup \{\mathbf{S}\}\}$$

Nodes labeled $\tau \in E \cup \{\mathbf{S}\}$ are formalized with $ext(\tau)$. For example, $ext(teacher)$ stores $\{v_2, v_3\}$. Paths are formalized with an inductive method [Pau94a].

$$\frac{v_1 \in ext(\tau)}{path(v_1, v_1)} \quad \frac{path(v_1, v_3) \quad v_2 \in ele\ v_3}{path(v_1, v_2)}$$

Paths are reflexive and $path(v_1, v_3)$ can be extended with children of v_3 . The example satisfies $path(v_1, v_1)$, $path(v_1, v_2)$ and $path(v_1, v_5)$. XML-trees don't have cycles. Paths connect $root$ with the element nodes. The section formalizes the validation of XML-trees. Children are labeled in accordance with the content models.

$$parse\ B\ (grammar(lab\ v)) \wedge\ getWord(B) = (\text{map}\ lab\ (ele\ v))$$

An element node v has a parse tree B for the formal grammar [HMU06], formalized with $grammar(lab\ v)$. The labels of B computed with $getWord(B)$ are equal to the labels of the children. Children v_5, v_6 of v_2 have the following labels.

$$\text{map}\ lab\ (ele\ v_2) = [teach, research]$$

The grammar for $teacher$ accepts the labels. Details of the formalization are presented in [His07]. The section formalizes integrity. Attribute l of a τ node v is stored with $v.l = val(att(v, l))$. The example stores $Prof. Crey$ with $v_9.name$. Attributes $L = \langle l_1, \dots, l_n \rangle$ are stored with $v[L] = \langle v.l_1, \dots, v.l_n \rangle$. The L tuples of τ nodes are formalized with $ext(\tau[L])$. When there is

one attribute, $ext(\tau[L])$ is replaced with $ext(\tau.l)$. With $ext(teacher.name)$, the example stores $\{Dr. Brett, Prof. Crey\}$.

$$ext(\tau[L]) = \{v[L] \mid v \in V \wedge lab\ v = \tau\}$$

The section formalizes integrity. Key $\tau[L] \rightarrow \tau$ identifies τ nodes with attributes L . The formalization considers a function $f(v) = v[L]$.

$$\tau[L] \rightarrow \tau \Leftrightarrow inj_on\ f\ ext(\tau)$$

For $\tau \in E$ with attributes L , the key is satisfied provided f is injective restricted to τ nodes. The example satisfies the key $teacher.name \rightarrow teacher$. Inclusion constraints represent dependencies of attributes.

$$\tau_1[L_1] \subseteq \tau_2[L_2] \Leftrightarrow ext(\tau_1[L_1]) \subseteq ext(\tau_2[L_2])$$

An XML-tree satisfies inclusion constraint $\tau_1[L_1] \subseteq \tau_2[L_2]$ whenever L_1 tuples of τ_1 nodes depend on L_2 tuples of τ_2 . The formalization in this section has been implemented with Isabelle [NP07]. The formalization is the fundament for the contribution. The next section formalizes circular XML-specifications.

4 Circular XML-Specifications

The section presents the formalization of circular XML-specifications. They are formalized with an inductive method [Pau94a] that proves the correctness of cryptographic protocols in [Pau98]. The section formalizes ways. A branch has two ways to a descendant. XML-specifications are circular when there is a branch without cycle and the descendant depends on the ancestor. The next section proves that circular XML-specifications are unsatisfiable. The branch proves a constraint and the dependency proves the opposite. Section 7 presents a deductive checker for XML-specifications based on circular XML-specifications. The formalization considers normalized content models.

$$\forall \tau \in E. (P\ \tau = \epsilon) \vee (\exists \tau_1, \tau_2 \in E \cup \{\mathbf{S}\}. P\ \tau = \tau_1 \vee P\ \tau = (\tau_1, \tau_2) \vee (P\ \tau = (\tau_1 | \tau_2) \wedge \tau_1 \neq \tau_2))$$

They have less or equal two labels and don't contain plus, question mark and star. The appendix contains details. The section formalizes ways formed with content models.

$$\frac{\tau_1 \in E}{way(\tau_1, \tau_1)} \qquad \frac{P\ \tau_1 = \tau_3 \quad way(\tau_3, \tau_2)}{way(\tau_1, \tau_2)}$$

Ways are reflexive and content τ_3 of τ_1 with $way(\tau_3, \tau_2)$ satisfies $way(\tau_1, \tau_2)$.

$$\frac{P\ \tau_1 = (\tau_3, \tau_4) \quad way(\tau_3, \tau_2)}{way(\tau_1, \tau_2)} \qquad \frac{P\ \tau_1 = (\tau_4, \tau_3) \quad way(\tau_3, \tau_2)}{way(\tau_1, \tau_2)}$$

Ways can be extended with concatenated content provided a label τ_3 satisfies $way(\tau_3, \tau_2)$.

$$\frac{P\ \tau_1 = (\tau_3 | \tau_4) \quad way(\tau_3, \tau_2) \quad way(\tau_4, \tau_2)}{way(\tau_1, \tau_2)}$$

Content models $(\tau_3 | \tau_4)$ extend ways where the elements τ_3, τ_4 have a way. The τ_1 nodes have τ_2 descendants when there is a way from τ_1 to τ_2 . Section 6 proves this.

$$branch(\tau_1, \tau_2) \Leftrightarrow \exists \tau_3, \tau_4, \tau_5 \in E. way(\tau_1, \tau_3) \wedge P\ \tau_3 = (\tau_4, \tau_5) \wedge way(\tau_4, \tau_2) \wedge way(\tau_5, \tau_2)$$

A structural schema has a branch from τ_1 to τ_2 when there is a way from τ_1 to an element τ_3 such that the labels of the concatenated content model have a way to the descendant. The element τ_3 represents the branch. An XML-tree of a structural schema with $branch(\tau_1, \tau_2)$ includes τ_2 descendants for τ_1 nodes. The example has a branch, *teacher* and *teach* have a way to *subject*. Element *teach* represents the branch with $(subject, subject)$. The example satisfies $branch(teach, subject)$ and $branch(teacher, subject)$.

The section formalizes cycles. Circular XML-specifications have a branch without cycle. Elements that have a possible way are formalized.

$$\frac{\tau_2 \in P \tau_1}{possibleWay(\tau_1, \tau_2)} \quad \frac{possibleWay(\tau_1, \tau_3) \quad \tau_2 \in P \tau_3}{possibleWay(\tau_1, \tau_2)}$$

Possible ways are proven with content models. A possible way is obtained with an element $\tau_3 \in P \tau_1$ that satisfies $possibleWay(\tau_3, \tau_2)$. XML-trees of a structural schema with $possibleWay(\tau_1, \tau_2)$ possibly have a τ_2 descendant for a τ_1 node. The example has a possible way from *teacher* to *subject*. Possible ways formalize cycles.

$$cycle(\tau) \Leftrightarrow possibleWay(\tau, \tau)$$

Structural schemas that satisfy $possibleWay(\tau, \tau)$ have a cycle with τ . The example doesn't have a cycle. The section formalizes integrity that bounds elements.

$$anchor(\tau_1, \tau_2) \Leftrightarrow \exists L_1 \subseteq (R \tau_1). \exists L_2 \subseteq (R \tau_2). \tau_1[L_1] \rightarrow \tau_1 \wedge \tau_1[L_1] \subseteq \tau_2[L_2]$$

A key $\tau_1[L_1] \rightarrow \tau_1$ and an inclusion constraint form an anchor when $\tau_1[L_1]$ depends on $\tau_2[L_2]$. The example satisfies $anchor(subject, teacher)$.

$$onceOccurs(\tau_1, \tau_2) \Leftrightarrow P \tau_2 = \tau_1 \vee \exists \tau_3. \tau_3 \neq \tau_1 \wedge (P \tau_2 = (\tau_1, \tau_3) \vee P \tau_2 = (\tau_3, \tau_1))$$

Simple and concatenated content models that contain a label once are formalized with $onceOccurs$. The example satisfies $onceOccurs(teacher, teach)$.

$$moreOccurs(\tau_1) \Leftrightarrow \exists \tau_2, \tau_4 \in E. \tau_1 \in P \tau_2 \wedge \tau_1 \in P \tau_4 \wedge \tau_2 \neq \tau_4$$

A structural schema satisfies $moreOccurs(\tau_1)$ when τ_1 occurs in the content models of two elements. The example satisfies $moreOccurs(\mathbf{S})$ because *research* and *subject* store text. The section presents the formalization of bounds.

$$\frac{anchor(\tau_1, \tau_2)}{bounds(\tau_1, \tau_2)} \quad \frac{onceOccurs(\tau_1, \tau_2) \quad \neg moreOccurs(\tau_1)}{bounds(\tau_1, \tau_2)}$$

Element τ_2 bounds τ_1 when integrity forms an anchor. The example bounds *subject* with *teacher*. Element τ_2 that stores τ_1 once bounds τ_1 .

$$\frac{way(\tau_1, \tau_2) \quad \neg cycle(\tau_1)}{bounds(\tau_1, \tau_2)} \quad \frac{bounds(\tau_1, \tau_3) \quad bounds(\tau_3, \tau_2)}{bounds(\tau_1, \tau_2)}$$

A way without cycle satisfies $bounds(\tau_1, \tau_2)$. Element τ_2 that bounds τ_3 that bounds τ_1 satisfies $bounds(\tau_1, \tau_2)$. The section formalizes circular XML-specifications.

$$circular \Leftrightarrow way(r, \tau_1) \wedge branch(\tau_1, \tau_2) \wedge \neg cycle(\tau_1) \wedge bounds(\tau_2, \tau_1)$$

An XML-specification is circular, when the structural schema has a way from r to a branch that doesn't have a cycle at the ancestor of the branch and the descendant bounds the ancestor. The example is circular. There is a way from r to *teacher* and a branch from *teacher* to *subject*. The ancestor *teacher* doesn't have a cycle. The descendant *subject* is bounded with *teacher*. The section has presented the formalization of circular XML-specifications. The next section proves theorems of ways and paths for proving the correctness of the checker presented in section 7.

5 Paths and Ways

The previous section has formalized circular XML-specifications based on the formalization in section 3. This section formalizes descendants and proves theorems of paths and ways. The next section proves that circular XML-specifications are unsatisfiable. The theorems prove the correctness of the checker presented in section 7.

Descendants are formalized with an inductive method [Pau94a] that formalizes circular XML-specifications in section 4. Then the section presents theorems for proving that the descendants of a branch of an XML-tree are disjoint in the next section. The τ_2 descendants of τ_1 nodes are formalized with $descendant(\tau_1, \tau_2)$.

$$\frac{v_1 \in ext(\tau_1)}{v_1 \in descendant(\tau_1, \tau_1)} \quad \frac{v_1 \in ext(\tau_2) \quad v_1 \in ele v_2 \quad v_2 \in descendant(\tau_1, \tau_3)}{v_1 \in descendant(\tau_1, \tau_2)}$$

The descendants are reflexive for τ_1 nodes. Node v is descendant of a τ_1 node when the parent $v_3 \in ext(\tau_3)$ is a descendant. The *subject* nodes of the example in figure 2 are descendants of teachers. They are children of a *teach* node that is a child of a *teacher* node of $descendant(teacher, teacher)$. Next, paths without label τ are formalized with $differentLabel$.

$$\frac{v_1 \in ext(\tau_1) \quad \tau_1 \neq \tau}{differentLabel(v_1, v_1, \tau)} \quad \frac{lab v_2 \neq \tau \quad v_2 \in ele v_3 \quad differentLabel(v_1, v_3, \tau)}{differentLabel(v_1, v_2, \tau)}$$

Nodes v_1 with a label τ_1 not equal τ satisfy $differentLabel(v_1, v_1, \tau)$. Such paths are extended with nodes of a label not equal τ . The section formalizes next nodes of a specified label.

$$same(v_1, v_2, \tau) \Leftrightarrow v_1 = v_2 \wedge v_1 \in ext(\tau)$$

The τ nodes satisfy *same*.

$$nextDifferent(v_1, v_2, \tau) \Leftrightarrow \exists v_3. v_2 \in ext(\tau) \wedge v_2 \in ele v_3 \wedge differentLabel(v_1, v_3, \tau)$$

The section formalizes paths to τ nodes that don't contain τ .

$$next(v_1, v_2, \tau) \Leftrightarrow same(v_1, v_2, \tau) \vee nextDifferent(v_1, v_2, \tau)$$

Nodes that are next have a path. The section formalizes descendants of the branch.

$$v \in descendant1(\tau_1, \tau_2, \tau_3) \Leftrightarrow \exists v_1, v_2, v_3, v_4. v_1 \in ext(\tau_1) \wedge next(v_1, v_2, \tau_2) \wedge ele v_2 = [v_3, v_4] \wedge next(v_3, v, \tau_3)$$

A node $v \in descendant1(\tau_1, \tau_2, \tau_3)$ takes the first way at the next τ_2 descendant of a τ_1 node. Node v is the next τ_3 descendant of the first child of the τ_2 descendant. Maths and chemistry are stored with $descendant1(teacher, teach, subject)$.

$$v \in descendant2(\tau_1, \tau_2, \tau_3) \Leftrightarrow \exists v_1, v_2, v_3, v_4. v_1 \in ext(\tau_1) \wedge next(v_1, v_2, \tau_2) \wedge ele v_2 = [v_3, v_4] \wedge next(v_4, v, \tau_3)$$

Nodes in $descendant2(\tau_1, \tau_2, \tau_3)$ take the second way to τ_3 . The section presents theorems of paths and ways, the next section proves circular XML-specifications are unsatisfiable.

$$\frac{v_2 \in ele v_1 \quad v_3 \in ele v_1 \quad v_2 \neq v_3}{\neg path(v_2, v_3)} T_1$$

Theorem T_1 proves that children aren't connected. The proof assumes the opposite, an induction with $path(v_2, v_3)$ proves the following hypothesis.

$$P_1(v_2, v_3) \Leftrightarrow v_2 \in \text{ele } v_1 \wedge v_3 \in \text{ele } v_1 \wedge v_2 \neq v_3 \rightarrow \text{false}$$

The base case proves a contradiction with $v_2 \neq v_3$. The induction step considers a node v_4 with the child v_3 and $path(v_2, v_4)$, $P_1(v_2, v_3)$ is proven. Parents are unique, v_1 is equal to v_4 . XML-trees don't have cycles. A contradiction is proven with $v_2 \in \text{ele } v_1$ and $path(v_2, v_1)$.

$$\frac{path(v_1, v_3) \quad path(v_2, v_3)}{path(v_1, v_2) \vee path(v_2, v_1)} T_2$$

Nodes with a path to the same node are connected (T_2). The contrapositive is proven. An induction with $path(v_1, v_3)$ proves the XML-tree doesn't have a path from v_2 to v_3 .

$$P_2(v_1, v_3) \Leftrightarrow \neg path(v_1, v_2) \wedge \neg path(v_2, v_1) \rightarrow \neg path(v_2, v_3)$$

The hypothesis with one node is a tautology. The induction step considers a node v_4 with $path(v_1, v_4)$ that satisfies $P_2(v_1, v_4)$. Node v_4 has the child v_3 and $P_2(v_1, v_3)$ is proven. When v_2 is equal to v_3 , it is a child of v_4 and the path from v_1 to v_4 gives a contradiction with $\neg path(v_1, v_2)$. Otherwise, $path(v_2, v_3)$ proves a path from v_2 to v_4 with child v_3 . Then $P_2(v_1, v_4)$ proves a contradiction.

$$\frac{path(v_1, v_2) \quad \text{lab } v_1 = \tau_1 \quad \text{lab } v_2 = \tau_2 \quad v_1 \neq v_2}{\text{possibleWay}(\tau_1, \tau_2)} T_3$$

Paths prove a possible way (T_3). An induction with $path(v_1, v_2)$ proves the theorem.

$$P_3(v_1, v_2) \Leftrightarrow \forall \tau_1, \tau_2. \text{lab } v_1 = \tau_1 \wedge \text{lab } v_2 = \tau_2 \wedge v_1 \neq v_2 \rightarrow \text{possibleWay}(\tau_1, \tau_2)$$

The base case proves a contradiction. The induction step satisfies $P_3(v_1, v_3)$ and considers a child v_2 of v_3 . The section proves that τ_1 has a possible way to the label of v_2 . When v_1 equals v_3 the content model of τ_1 includes τ_2 , a possible way is proven. Otherwise, the induction hypothesis proves $\text{possibleWay}(\tau_1, (\text{lab } v_3))$. The content model of the label of v_3 includes τ_2 .

$$\frac{\text{next}(v_1, v_2, \tau) \quad \text{next}(v_1, v_3, \tau) \quad path(v_2, v_3)}{v_2 = v_3} T_4$$

Theorem T_4 proves that nodes are equal when they are connected next descendants of the same node. Nodes that are equal can satisfy *same*. Otherwise, v_2 and v_3 have a path without τ from v_1 . Then v_1 has a *differentLabel* path to the parent of v_3 . The path contains the τ node v_2 .

$$\frac{v_1 \in \text{ext}(\tau_1) \quad \text{way}(\tau_1, \tau_2)}{\exists v_2 \in \text{ext}(\tau_2). path(v_1, v_2)} T_5$$

With T_5 , a path to a τ_2 node is proven for a τ_1 node when there is a way from τ_1 to τ_2 . An induction with $\text{way}(\tau_1, \tau_2)$ uses hypothesis $P_4(\tau_1, \tau_2)$.

$$P_4(\tau_1, \tau_2) \Leftrightarrow \forall v_1 \in \text{ext}(\tau_1). \exists v_2 \in \text{ext}(\tau_2). path(v_1, v_2)$$

The base case is proven with one node. The induction step considers the content models of τ_1 . Concatenated content models prove a τ_3 child that extends a path to v_2 proven with hypothesis $P_4(\tau_3, \tau_2)$. Otherwise $P \tau_1 = (\tau_3 | \tau_4)$, a child in $\text{ext}(\tau_3) \cup \text{ext}(\tau_4)$ is proven. Then $P_4(\tau_3, \tau_2)$ and $P_4(\tau_4, \tau_2)$ prove a path from v_1 to v_2 .

The section has presented theorems for proving that circular XML-specifications are unsatisfiable. Section 7 presents a checker based on circular XML-specifications.

6 Unsatisfiable Circular XML-Specifications

The section proves that circular XML-specifications are unsatisfiable with the theorems of the previous section. A branch proves there are more nodes of the descendant. The branch is bounded, a contradiction is proven.

$$\frac{\text{branch}(\tau_1, \tau_2) \quad \text{ext}(\tau_1) \neq \emptyset \quad \neg \text{cycle}(\tau_1)}{|\text{ext}(\tau_1)| < |\text{ext}(\tau_2)|}$$

An XML-tree with τ_1 nodes contains more descendants of a branch when the structural schema doesn't have a cycle with τ_1 . The section proves the first and second descendants of the XML-tree are disjoint. Then a cycle with τ_1 is proven.

The proof considers τ_3 that represents the branch and proves $\text{descendant1}(\tau_1, \tau_3, \tau_2) \cap \text{descendant2}(\tau_1, \tau_3, \tau_2) = \emptyset$. Then a node v of the intersection is presumed. Function f_1 (f_2) chooses the ancestor of the first (second) descendant of a branch of the XML-tree. The function $f_1(v) = v_1$ is defined with nodes v_2, v_3 and v_4 that satisfy $\text{next}(v_1, v_2, \tau_3)$, $\text{ele } v_2 = [v_3, v_4]$ and $\text{next}(v_3, v, \tau_2)$. In this way, $f_2(v) = v_5$ is defined with nodes v_6, v_7 and v_8 that satisfy $\text{next}(v_5, v_6, \tau_3)$, $\text{ele } v_6 = [v_7, v_8]$ and $\text{next}(v_8, v, \tau_2)$. When $v_1 \neq v_5$, T_3 proves $\text{possibleWay}(\tau_1, \tau_1)$ with the path of v_1 and v_5 proven with T_2 . This is a contradiction with $\text{cycle}(\tau_1)$. Thus, v_1 and v_5 are equal. The proof considers node v_2 (v_6) that represents the first (second) branch. They have a path to v . Theorem T_2 proves a path connects them. The nodes are next τ_3 nodes of v_1 , T_4 proves they are equal. Moreover, the children are equal. They aren't connected (T_1) and have the descendant v , T_2 proves a contradiction. The first and second descendants are disjoint.

An XML-tree has more τ_2 descendants than descendants of the first branch.

$$|\text{descendant}(\tau_1, \tau_2)| \geq |\text{descendant1}(\tau_1, \tau_3, \tau_2) \cup \text{descendant2}(\tau_1, \tau_3, \tau_2)|$$

The τ_2 descendants contain the descendants of a branch.

$$|\text{descendant1}(\tau_1, \tau_3, \tau_2)| + |\text{descendant2}(\tau_1, \tau_3, \tau_2)| > |\text{descendant1}(\tau_1, \tau_3, \tau_2)|$$

They are disjoint, the sum is considered. The XML-tree contains τ_1 nodes, T_5 proves there are descendants of the branch.

The proof presumes there are less or equal τ_2 descendants than τ_1 nodes. The previous inequation proves there are more τ_1 nodes than τ_2 descendants of the first branch. A function f_3 chooses a first descendant of a branch. The function is defined with $f_3(v_1) = v_2$ and nodes v_3, v_4 and v_5 such that $\text{next}(v_1, v_3, \tau_3)$, $\text{ele } v_3 = [v_4, v_5]$ and $\text{next}(v_4, v_2, \tau_2)$ are satisfied. The range equals the first descendants, T_5 proves f_3 is defined for τ_1 nodes. The domain is greater than the range, there are nodes $v_1, v_2 \in \text{ext}(\tau_1)$ with the descendant $v_3 = f_3(v_1) = f_3(v_2)$. The nodes have a path to v_3 , T_2 proves v_1 and v_2 are connected. Then T_3 proves $\text{possibleWay}(\tau_1, \tau_1)$. This is a contradiction with $\neg \text{cycle}(\tau_1)$. The XML-tree satisfies $|\text{ext}(\tau_1)| < |\text{descendant}(\tau_1, \tau_2)|$. The descendants are contained in $\text{ext}(\tau_2)$, the theorem is proven. Theorem T_5 proves with $\text{way}(r, \tau_1)$ that there are τ_1 nodes. Circular XML-specifications satisfy $|\text{ext}(\tau_1)| < |\text{ext}(\tau_2)|$. The next theorem proves the opposite with $\text{bounds}(\tau_2, \tau_1)$. Circular XML-specifications are unsatisfiable.

$$\frac{\text{bounds}(\tau_1, \tau_2)}{|\text{ext}(\tau_1)| \leq |\text{ext}(\tau_2)|}$$

An induction with $\text{bounds}(\tau_1, \tau_2)$ proves there are less or equal τ_1 than τ_2 nodes. Anchors prove the inequation with integrity. A constraint of the transformation for model checking XML-specifications presented in [His07] proves an equation for elements that occur once. An injective function chooses a descendant for ways without cycle. Finally, the induction hypothesis proves

the theorem. The proof defines the induction hypothesis $|ext(\tau_1)| \leq |ext(\tau_2)|$. Integrity implies inequations proven in [His07]. An anchor is defined with a key $\tau_1[L_1] \rightarrow \tau_1$ and an inclusion constraint $\tau_1[L_1] \subseteq \tau_2[L_2]$.

$$|ext(\tau_1)| = |ext(\tau_1[L_1])| \leq |ext(\tau_2[L_2])| \leq |ext(\tau_2)|$$

The key proves there is an equal number of τ_1 nodes as L_1 tuples. The inclusion constraint proves that they are less or equal than the L_2 tuples of τ_2 . Then the proof considers elements that occur once. The transformation proves a structured representation of nodes.

$$|ext(\tau_1)| = \sum_{\substack{\tau_1 \in P \tau_2 \\ i \in \{0, 1\}}} |children(\tau_2, \tau_1, i)|$$

The constraint represents the τ_1 nodes with the τ_1 children of τ_2 nodes at position $i \in \{0, 1\}$.

$$|ext(\tau_1)| = |children(\tau_2, \tau_1, i)| \leq |ext(\tau_2)|$$

Then *onceOccurs*(τ_1, τ_2) and \neg *moreOccurs*(τ_1) prove that τ_1 has the parent τ_2 at i . An XML-tree has less or equal children than τ_2 nodes. The proof considers ways without cycle.

$$\frac{way(\tau_1, \tau_2) \quad \neg cycle(\tau_1)}{|ext(\tau_1)| \leq |ext(\tau_2)|}$$

An injective function proves the theorem choosing the next τ_2 node of a τ_1 node. The way proves with T_5 that a path exists. The function f_4 is proven injective. Otherwise there are nodes v_1 and v_2 with $v_3 = f_4(v_1) = f_4(v_2)$. With the paths to v_3 T_2 proves v_1 and v_2 are connected. Moreover, with T_3 a possible way from τ_1 to itself is proven. This is a contradiction with $\neg cycle(\tau_1)$, the inequation is proven.

Finally, the induction proves $|ext(\tau_1)| \leq |ext(\tau_2)|$ with labels τ_1, τ_2 and τ_3 that satisfy *bounds*(τ_1, τ_3) and *bounds*(τ_3, τ_2). The induction hypothesis proves $|ext(\tau_1)| \leq |ext(\tau_3)|$ and $|ext(\tau_3)| \leq |ext(\tau_2)|$. The next section presents a checker based on circular XML-specifications.

7 Deductive Checker

The previous section proves that circular XML-specifications are unsatisfiable. This section presents a checker based on circular XML-specifications. The checker generates F-Logic [KLW95] facts that represent XML-specifications. Objects represent elements and attributes, a class hierarchy represents the structural schema. The section presents a deductive checker based on circular XML-specifications. Section 6 proves the correctness of the checker that has been implemented with the DEAXS [His07] project. The checker generates F-Logic facts that are checked with Florid [HLS07].

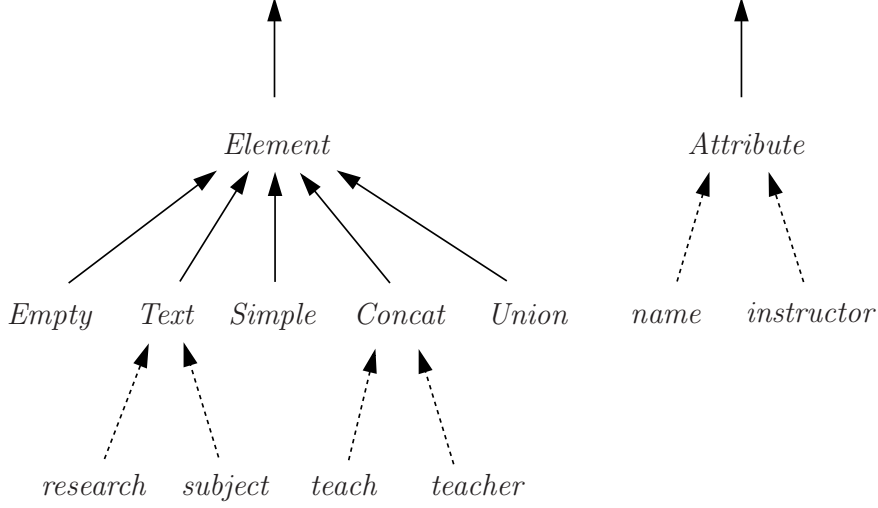
The section presents the formalization of XML-specifications with F-Logic. Objects of class *Element* represent elements, the subclasses represent the normalized structural schemas. The appendix contains details of the normalization. The checker is illustrated with the example defined in figure 3 and root *teacher*. The section formalizes the representation of the structural schema. Class *Element* has the subclasses *Empty*, *Simple*, *Text*, *Concat* and *Union* that represent the content models. A signature defines the class hierarchy and declares methods.

```

Element[attributes $\Rightarrow$ Attribute].
Empty :: Element.
Simple :: Element[contents $\Rightarrow$ Element].
Text :: Element[contents $\Rightarrow$ Empty].
Concat :: Element[contents@(integer) $\Rightarrow$ Element].
Union :: Element[contents@(integer) $\Rightarrow$ Element].

```

Figure 5 A class hierarchy represents the example in figure 3.



For example, *teacher* stores content model (*teach*, *research*). Object *teacher* is an instance of *Concat*. Attributes are stored with method *attributes*. Element *teacher* stores *name*, an instance of *Attribute*. Figure 5 visualizes the hierarchy. Fact $\tau : \text{Empty}$ represents ϵ . Contents $P \tau = \tau_1$ is represented with $\tau : \text{Simple}[\text{contents} \rightarrow \tau_1]$, fact $\tau : \text{Concat}[\text{contents}@ (0) \rightarrow \tau_1; \text{contents}@ (1) \rightarrow \tau_2]$ represents (τ_1, τ_2) . The section represents the structural schema of the example.

name : *Attribute*.
instructor : *Attribute*.
teacher : *Concat*[*contents*@(0) \rightarrow *teach*; *contents*@(1) \rightarrow *research*; *attributes* \rightarrow *name*].
teach : *Concat*[*contents*@(0) \rightarrow *subject*; *contents*@(1) \rightarrow *subject*; *attributes* \rightarrow {}].
research : *Text*[*contents* \rightarrow *S*; *attributes* \rightarrow {}].
subject : *Text*[*contents* \rightarrow *S*; *attributes* \rightarrow *instructor*].
S[*attributes* \rightarrow {}].

Rules define relation *way* to check circular XML-specifications formalized in section 4.

$$\begin{aligned} \text{way}(X_1, X_1) &\leftarrow X_1 : \text{Element}. \\ \text{way}(X_1, X_2) &\leftarrow X_1 : \text{Simple}[\text{contents} \rightarrow X_3] \wedge \text{way}(X_3, X_2). \\ \text{way}(X_1, X_2) &\leftarrow X_1 : \text{Concat}[\text{contents}@(-) \rightarrow X_3] \wedge \text{way}(X_3, X_2). \\ \text{way}(X_1, X_2) &\leftarrow X_1 : \text{Union}[\text{contents}@ (0) \rightarrow X_3; \text{contents}@ (1) \rightarrow X_4] \wedge \text{way}(X_3, X_2) \wedge \\ &\quad \text{way}(X_4, X_2). \end{aligned}$$

The example satisfies $\text{way}(\text{subject}, \text{subject})$, $\text{way}(\text{teach}, \text{subject})$ and $\text{way}(\text{teacher}, \text{subject})$. Next, a rule defines a branch.

$$\text{branch}(X_1, X_2) \leftarrow \text{way}(X_1, X_3) \wedge X_3 : \text{Concat}[\text{contents}@ (0) \rightarrow X_4; \text{contents}@ (1) \rightarrow X_5] \wedge \text{way}(X_4, X_2) \wedge \text{way}(X_5, X_2).$$

Relation $\text{branch}(X_1, X_2)$ is defined with a way from X_1 to X_3 that represents the branch with ways to the descendant. Elements *teacher* and *teach* have a branch to *subject*. The example satisfies $\text{branch}(\text{teach}, \text{subject})$ and $\text{branch}(\text{teacher}, \text{subject})$. The section formalizes cycles.

$$\begin{aligned} X_1[\text{occurs} \rightarrow X_2] &\leftarrow X_2 : \text{Simple}[\text{contents} \rightarrow X_1]. \\ X_1[\text{occurs} \rightarrow X_2] &\leftarrow X_2 : \text{Element}[\text{contents}@(-) \rightarrow X_1]. \end{aligned}$$

Attribute *occurs* ($Element[occurs \Rightarrow Element]$) is defined with the content models. For example, *subject* satisfies $subject[occurs \rightarrow teach]$. The section formalizes possible ways.

$$\begin{aligned} possibleWay(X_1, X_2) &\leftarrow X_2[occurs \rightarrow X_1]. \\ possibleWay(X_1, X_2) &\leftarrow X_3[occurs \rightarrow X_1] \wedge possibleWay(X_3, X_2). \end{aligned}$$

The example satisfies $possibleWay(teacher, X)$ for $X \in \{research, subject, teach\}$.

$$cycle(X_1) \leftarrow possibleWay(X_1, X_1).$$

A cycle with X_1 is proven when X_1 has a possible way to itself. The example doesn't have cycles. The section formalizes elements that occur with more content models.

$$moreOccurs(X_1) \leftarrow X_1[occurs \rightarrow X_2] \wedge X_1[occurs \rightarrow X_3] \wedge X_2 \neq X_3.$$

A label X_1 that occurs in two content models satisfies $moreOccurs(X_1)$.

$$\begin{aligned} onceOccurs(X_1, X_2) &\leftarrow X_2 : Simple[contents \rightarrow X_1]. \\ onceOccurs(X_1, X_2) &\leftarrow X_2 : Concat[contents@0 \rightarrow X_1; contents@1 \rightarrow X_3] \wedge X_1 \neq X_3. \\ onceOccurs(X_1, X_2) &\leftarrow X_2 : Concat[contents@0 \rightarrow X_3; contents@1 \rightarrow X_1] \wedge X_1 \neq X_3. \end{aligned}$$

Content models $P \tau_2$ of the form $\tau_1 | (\tau_1, \tau_3) | (\tau_3, \tau_1)$ with $\tau_3 \neq \tau_1$ satisfy $onceOccurs(\tau_1, \tau_2)$. The example satisfies $onceOccurs(research, teacher)$. Next, the checker defines *bounds*.

$$\begin{aligned} bounds(X_1, X_2) &\leftarrow anchor(X_1, X_2). \\ bounds(X_1, X_2) &\leftarrow bounds(X_1, X_3) \wedge bounds(X_3, X_2). \\ bounds(X_1, X_2) &\leftarrow onceOccurs(X_1, X_2) \wedge \neg moreOccurs(X_1). \\ bounds(X_1, X_2) &\leftarrow way(X_1, X_2) \wedge \neg cycle(X_1). \end{aligned}$$

An *anchor* bounds elements with integrity. XML-specifications with $\tau_1[L_1] \rightarrow \tau_1$ and $\tau_1[L_1] \subseteq \tau_2[L_2]$ satisfy $anchor(\tau_1, \tau_2)$. The example satisfies $anchor(subject, teacher)$ with the integrity constraints $subject.instructor \rightarrow subject$ and $subject.instructor \subseteq teacher.name$. Element X_2 that bounds X_3 that bounds X_1 satisfies $bounds(X_1, X_2)$. Elements that occur once satisfy *bounds*. Ways from X_1 to X_2 without cycle satisfy $bounds(X_1, X_2)$. The example bounds *subject* with *teacher*. Florid [HLS07] proves that the example is circular.

$$\begin{aligned} way(teacher, teacher). \\ branch(teacher, subject). \\ \neg cycle(teacher). \\ bounds(subject, teacher). \end{aligned}$$

The rules check a branch. The root element *teacher* has a way to *teach* with a branch to *subject*. The example doesn't have a cycle with *teacher* that is bounded with *subject*.

$$?- way(r, X_1) \wedge branch(X_1, X_2) \wedge \neg cycle(X_1) \wedge bounds(X_2, X_1).$$

The example is proven circular with *teacher* for X_1 , *subject* for X_2 and the root *teacher*. The section has presented a checker for XML-specifications. Section 6 has proven the correctness of the checker. The checker has been implemented with the DEAXS [His07] project. The last section concludes the contribution.

8 Conclusion

The previous section has presented a checker for XML-specifications. The contribution is concluded with an overview.

An extensive formalization has been developed with Isabelle [Pau94b]. Details of the formalization are presented in [His07]. Circular XML-specifications are formalized with an inductive method [Pau94a]. Section 6 proves that circular XML-specifications are unsatisfiable. Section 7 presents a checker based on circular XML-specifications. XML-specifications are represented with F-Logic [KLW95] facts that are checked with Florid [HLS07]. The correctness of the checker is proven. The checker has been implemented with the DEAXS [His07] project. The checker normalizes structural schemas (appendix A), generates graphs and the representation of XML-specifications with F-Logic [KLW95]. The facts are checked with Florid [HLS07].

The work has been supported by the DFG Graduiertenkolleg Mathematical Logic and Applications of the University Freiburg.

References

- [AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- [BPSM⁺06] Tim Bray, Jean Paoli, Michael C. Sperberg-McQueen, Eve Maler, and François Yergeau. XML. <http://www.w3.org/TR/REC-xml/>, 2006.
- [BW] Stefan Berghofer and Markus Wenzel. Inductive Datatypes in HOL—Lessons Learned in Formal- Logic Engineering. In *TPHOLs 99*, pages 19–36.
- [CV85] Ashok K. Chandra and Moshe Y. Vardi. The Implication Problem for Functional and Inclusion Dependencies is Undecidable. *SIAM Journal on Computing*, 14(3):671–677, 1985.
- [FL02] Wenfei Fan and Leonid Libkin. On XML Integrity Constraints in the Presence of DTDs. *Journal of the ACM*, 49(3):368–406, 2002.
- [His07] Harald Hiss. Werkzeuge zur Entwicklung von XML-Spezifikationen. Technical report, Institut für Informatik, Universität Freiburg, 2007.
- [HLS07] Thomas Hornung, Georg Lausen, and Florian Schmedding. Florid. <http://dbis.informatik.uni-freiburg.de/index.php?project=Florid>, 2007.
- [HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley, 2006.
- [KLW95] Michael Kifer, Georg Lausen, and James Wu. Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM*, 42(4):741–843, 1995.
- [NP07] Tobias Nipkow and Lawrence C. Paulson. Isabelle. <http://www.cl.cam.ac.uk/Research/HVG/Isabelle/>, 2007.
- [Pau94a] Lawrence C. Paulson. A Fixedpoint Approach to Implementing (Co)Inductive Definitions. In *CADE-12*, pages 148–161, London, UK, 1994. Springer-Verlag.
- [Pau94b] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer, 1994.
- [Pau98] Lawrence C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.

A Normalization of Structural Schemas

This section contains details of the normalization of the structural schema presented in [His07]. The formalization of circular XML-specifications in section 4 and the deductive checker presented in section 7 are based on the normalization that preserves satisfiability.

$$\forall \tau \in E. (P \tau = \epsilon) \vee (\exists \tau_1, \tau_2 \in E \cup \{\mathbf{S}\}. P \tau = \tau_1 \vee P \tau = (\tau_1, \tau_2) \vee (P \tau = (\tau_1 | \tau_2) \wedge \tau_1 \neq \tau_2))$$

Normalized content models have less or equal than two labels and don't contain plus, question mark and star. Contents $\alpha?$ is replaced with $(\epsilon | \alpha)$ and α^+ is replaced with (α, α^*) . Method *normalize* (figure 6) generates content models without star and less or equal than two labels. Method *setContentModel*(τ, α) writes $P \tau = \alpha$. Nested content models satisfy *nestedContentModel*(τ). Content models can be modified with *updateContentModel*. With (τ, i, α) the method writes α at position i of $P \tau$. Method *updateContentModel*($\tau, 1, \beta$) writes $P \tau = (\beta, \alpha_2)$ when τ stores content model (α_1, α_2) . Content models ($\tau | \tau$) are simplified.

Figure 6 Normalization of nested content models with star.

```

VOID normalize(Element  $\tau$ ){
  IF(starContentModel( $\tau$ )){
    \ \       $P \tau = \alpha^*$ 
     $\tau_1 := new Element();$ 
    setContentModel( $\tau, \tau_1$ );
    setContentModel( $\tau_1, (\epsilon | (\alpha, \tau_1))$ );
    normalize( $\tau_1$ );
  }
  IF(nestedContentModel( $\tau$ )){
    \ \       $P \tau = (\alpha_1, \alpha_2)$  or  $P \tau = (\alpha_1 | \alpha_2)$  and
    \ \       $\alpha_1$  or  $\alpha_2$  is nested
    FOR  $i \in \{1, 2\}$  {
      IF(nested( $\alpha_i$ )){
         $\tau_i := new Element();$ 
        setContentModel( $\tau_i, \alpha_i$ );
        updateContentModel( $\tau, i, \tau_i$ );
        \ \      if ( $i = 1$ ) then  $P \tau = (\tau_1, \alpha_2)$ ,
        \ \      otherwise  $P \tau = (\alpha_1, \tau_2)$ 
        normalize( $\tau_i$ );
      }
    }
  }
}

```
