XML Types

Non Peer-reviewed author version

NEVEN, Frank (2009) XML Types. In: Liu, Ling & M. Tamer, Özsu (Ed.)
Encyclopedia of Database Systems, p. 3650-3654..

DOI: 10.1007/978-0-387-39940-9_1563
Handle: http://hdl.handle.net/1942/10422

XML TYPES

Frank Neven
Hasselt University and transnational University of Limburg,
http://alpha.uhasselt.be/~fneven/

SYNONYMS

XML schemas

DEFINITION

To constrain the structure of allowed XML documents, for instance with respect to a specific application, a target schema can be defined in some schema language. A schema consists of a sequence of type definitions specifying a (possibly infinite) class of XML documents. A type can be assigned to every element in a document valid w.r.t. a schema. As the same holds for the root element, also the document itself can be viewed to be of a specific type. The schema languages DTDs, XML Schema, and Relax NG, are, on an abstract level, different instantiations of the abstract model of unranked regular tree languages.

HISTORICAL BACKGROUND

Brüggemann-Klein, Murata, and Wood [3] where the first to revive the theory of regular unranked tree automata [14] for the modelling of XML schema languages. Murata et al. [10] provided the formal taxonomy as presented here. Martens et al. [8] characterized the expressiveness of the different models and provided type-free abstractions.

SCIENTIFIC FUNDAMENTALS

Intuition

Consider the XML document in Figure 1 that contains information about store orders and stock contents. Orders hold customer information and list the items ordered, with each item stating its id and price. The stock contents consists of the list of items in stock, with each item stating its id, the quantity in stock, and – depending on whether the item is atomic or composed from other items – some supplier information or the items of which they are composed, respectively. It is important to emphasize that order items do not include supplier information, nor do they mention other items. Moreover, stock items do not mention prices. DTDs are incapable of distinguishing between order items and stock items because the content model of an element can only depend on the element's name in a DTD, and not on the context in which it is used. For example, although the DTD in Figure 2 describes all intended XML documents, it also allows supplier information to occur in order items and price information to occur in stock items.

The W3C specification essentially defines an XSD as a collection of *type definitions*, which, when abstracted away from the concrete XML representation of XSDs, are rules like

$$store \rightarrow \texttt{order}[order]^*, \texttt{stock}[stock] \qquad (\star)$$

that map type names to regular expressions over pairs $a[t]$ of element names a and type names t. Intuitively, this particular type definition specifies an XML fragment to be of type *store* if it is of the form

<order> f_1 </order> ... <order> f_n</order> <stock> g </stock>

```
<store>
  <order>
    <customer>
       <name>John Mitchell</name>
       <email> j.mitchell@yahoo.com </email>
    </customer>
    <item> <id> I18F </id>
           <price> 100 </price>
    </item>
    <item> ... </item> ... <item> ... </item>
  </order>
  <order> ... </order> ... <order> ... </order>
  <stock>
    <item>
      <id> IG8 </id> <qty> 10 </qty>
      <supplier> <name> Al Jones </name>
                 <email> a.j@gmail.com </email>
                 <email> a.j@dot.com </email>
      </supplier>
    </item>
    <item>
       <id> J38H </id> <qty> 30 </qty>
       <item>
          <id> J38H1 </id> <qty> 10 </qty>
          <supplier> ... </supplier>
       </item>
       <item>
          <id> J38H2 </id> <qty> 1 </qty>
          <supplier> ... </supplier>
       </item>
       <item> ... </item> ... <item> ... </item>
    </item>
    ...
    <item> ... </item>
  </stock>
</store>
```

Figure 1: Example XML document.

```
<!ELEMENT store (order*, stock)>
<!ELEMENT order (customer, item+)>
<!ELEMENT customer (name, email*)>
<!ELEMENT item (id, price + (qty, (supplier
                    + item+)))>
<!ELEMENT stock (item+)>
<!ELEMENT supplier (name, email*)>
```

$$
\begin{array}{lcl}
\mathit{root} & \rightarrow & \texttt{store}[\mathit{store}] \\
\mathit{store} & \rightarrow & \texttt{order}[\mathit{order}]^*, \texttt{stock}[\mathit{stock}] \\
\mathit{order} & \rightarrow & \texttt{customer}[\mathit{person}], \texttt{item}[\mathit{item}_1]^+ \\
\mathit{person} & \rightarrow & \texttt{name}[\mathit{emp}], \texttt{email}[\mathit{emp}]^+ \\
\mathit{item}_1 & \rightarrow & \texttt{id}[\mathit{emp}], \texttt{qty}[\mathit{emp}], \texttt{price}[\mathit{emp}] \\
\mathit{stock} & \rightarrow & \texttt{item}[\mathit{item}_2]^+ \\
\mathit{item}_2 & \rightarrow & \texttt{id}[\mathit{emp}], \texttt{qty}[\mathit{emp}], \\
 & & \quad (\texttt{supplier}[\mathit{person}] + \texttt{item}[\mathit{item}_2]^+) \\
\mathit{emp} & \rightarrow & \varepsilon
\end{array}
$$

Figure 2: A DTD and an XSD describing the document in Figure 1.

where $n \geq 0$; $f_1, \ldots, f_n$ are XML fragments of type *order*; and g is an XML fragment of type *stock*. Each type name that occurs on the right hand side of a type definition in an XSD must also be defined in the XSD, and each type name may be defined only once. Using types, an XSD can specify that an item is an order item when it occurs under an order element and is a stock item otherwise. For example, Figure 2 shows an XSD describing the intended set of store document. Notice in particular the use of the types item_1 and item_2 to distinguish between order items and stock items.

It is important to remark that the 'Element Declaration Consistent' constraint of the W3C specification requires multiple occurrences of the same element name in a single type definition to occur with the same type. Hence, type definition ($\star$) is legal, but

$$\mathit{persons} \rightarrow (\texttt{person}[\mathit{male}] + \texttt{person}[\mathit{female}])^+$$

is not, as `person` occurs both with type *male* and type *female*. Of course, element names in *different* type definitions can occur with different types (which is exactly what yields the ability to let the content model of an element depend on its context). On a structural level, ignoring attributes and the concrete syntax, the structural expressiveness of Relax NG corresponds to XSDs without the EDC constraint.

A formalization of Relax NG

An XML fragment $f = f_1 \cdots f_n$ is a sequence of labeled trees where every tree consists of a finite number of nodes, and every node v is assigned an element name denoted by $\mathrm{lab}(v)$. There always is a virtual root root which acts as the common parent of the roots of the different f_i. For a set EName and Types of element and type names, respectively, the set of elements is defined as $\{a[t] \mid a \in \mathsf{EName}, t \in \mathsf{Types}\}$. The set of regular expressions is given by the following syntax:

$$r ::= \varepsilon \mid \alpha \mid r, r \mid r + r \mid r^* \mid r^+ \mid r?$$

where ε denotes the empty string and α is an element. Their semantics is the usual one and is therefore omitted.

An *XSchema* is a tuple $S = (\mathsf{EName}, \mathsf{Types}, \rho, t_0)$ where EName and Types are finite sets of elements and types, respectively, ρ is a mapping from Types to regular expressions, and, $t_0 \in \mathsf{Types}$ is the start type. A *typing* τ of f is a mapping assigning a type $\tau(v) \in \mathsf{Types}$ to every node v in f (including the virtual root). For a node v with children $v_1, \ldots, v_m$, define child-string(τ, v) as the string $\mathrm{lab}(v_1)[\tau(v_1)] \cdots \mathrm{lab}(v_1)[\tau(v_1)]$. An XML fragment f then *conforms to* or is *valid w.r.t.* S if there is a typing τ of f such that for every node v, child-string(τ, v) matches the regular expression $\rho(\tau(v))$, and $\tau(\mathsf{root}) = t_0$. The mapping τ is then called a *valid typing*.

Despite the clean formalization, the above definition does not entail a validation algorithm. One possibility is to compute for each node v in f a set of possible types $\Delta(v) \subseteq \mathsf{Types}$ such that for each type $t \in \Delta(v)$, the XML subfragment rooted at v is valid w.r.t. the schema with start type t. The XML fragment is then valid w.r.t. S itself when the start type t_0 belongs to $\Delta(\mathsf{root})$. The sets $\Delta(v)$ can be computed in a bottom-up fashion. Indeed, $t \in \Delta(v)$ iff (1) v is a leaf node and $\rho(t)$ contains the empty string; or, (2) v is a non-leaf node with children $v_1, \ldots, v_n$ and there are $t_1 \in \Delta(v_1)$, $\ldots$, $t_n \in \Delta(v_n)$ such that $\mathrm{lab}(v_1)[t_1] \cdots \mathrm{lab}(v_n)[t_n] \in \rho(t)$. A valid typing can then be computed from the sets Δ by an additional top-down pass through the tree. Although this kind of bottom-up validation is a bit at odds with the general concept of top-down or streaming XML processing, the algorithm can be adapted to this end (cf., for instance, [10, 13]). For general XSchema's, a valid typing is not necessarily unique and can not always be computed in a single pass [8].

XSchemas as defined above correspond precisely to the class of unranked regular hedge languages [3] and can be seen as an abstraction of Relax NG. Note that the present formalization is overly simplistic w.r.t. attributes as Relax NG treats them in a way uniform to elements using attribute-element constraints [6].

Relationship with tree and hedge automata

Although an XML fragment can consist of a sequence of labeled trees, in the literature it is accustomed to restrict this sequence to simply one tree. XSchemas as defined above then define precisely the unranked regular tree languages [3, 11]. Although several automata formalism capturing this class have been defined [3, 4, 5], each with their own advantages [9], XSchemas correspond most closely to the model of Brüggemann-Klein, Murata, and Wood [3] which is defined as follows. A tree automaton $A = (Q, \mathsf{EName}, \delta, q_0)$ where Q is the set of states (or, types), $q_0 \in Q$ is the start state (start type), and δ maps pairs $(q, a) \in Q \times \mathsf{EName}$ to regular expressions over Q. An input tree f is accepted by the automaton if there exists a mapping τ from the nodes of f to Q, called a run (or, a typing), such that the root is labeled with the start state, and for every non-root node v with children $v_1, \ldots, v_n$, the string $\mathrm{lab}(v_1) \cdots \mathrm{lab}(v_n)$ matches $\delta(\tau(v), \mathrm{lab}(v))$. The translation between XSchemas and tree automata is folklore and can for instance be found in [3].

Deterministic regular expressions

The unique particle attribution constraint (UPA) requires regular expressions to be deterministic in the following sense: the form of the regular expression should allow to match each symbol of the input string uniquely against a position in the expression when processing the input string in one pass from left to right. That is, without looking ahead in the string. For instance, the expression $r = (a + b)^*a$ is not deterministic as already the first symbol in the string aaa can be matched to two different a's in r. The equivalent expression $b^*a(b^*a)^*$, on the other hand, is deterministic. Unfortunately, not every regular expression can be rewritten into an equivalent deterministic one [2]. Moreover, it is not a very robust subclass, as it is not closed under union, concatenation, or Kleene-star, prohibiting an elegant constructive definition [2]. Deterministic regular expressions are characterized as one-unambiguous regular expressions by Brüggemann-Klein and Wood [2]. Deciding whether a regular expression is one-unambiguous can be done in quadratic time [1]. Furthermore, it can be decided in EXPTIME whether there is a deterministic regular expression equivalent to a given regular expressions [2]. If so, the algorithm can return an expression of a size which is double exponential. It is unclear whether this can be improved.

A formalization of DTDs and XSDs

Let $S = (\mathsf{EName}, \mathsf{Types}, \rho, t_0)$ be an XSchema. Then, S is *local* when $\mathsf{EName} = \mathsf{Types}$ and regular expressions in ρ are defined over the alphabet $\{a[a] \mid a \in \mathsf{EName}\}$. This simply means that the name of the element also functions as its type. Furthermore, S is *single-type* when there are no elements $a[t_1]$ and $a[t_2]$ in a $\rho(t)$ with $t_1 \neq t_2$. A DTD is then a local XSchema where regular expressions are restricted to be deterministic. Finally, an XSD is then a single-type XSchema where regular expressions are restricted to be deterministic.

Expressiveness and complexity

XSchemas form a very robust class, for instance, equivalent to the monadic second-order logic (MSO) definable classes of unranked trees [12], and are closed under the Boolean operations. XSDs on the other hand are not closed under union or complement [8, 10]. They define precisely the subclasses of XSchemas closed under ancestor-guarded subtree exchange and are much closer to DTDs than to XSchemas as becomes apparent from the following equivalent type-free alternative characterization. A *pattern-based XSD* P is a set of rules $\{r_1 \to s_1, \ldots, r_m \to s_m\}$ where all r_i are horizontal regular expressions and all s_i are deterministic vertical regular expressions. An XML fragment f is *valid* with respect to P if, for every node v of f, there is a rule $r \to s \in P$ such that the string formed by the labels of the nodes on the path from the root to v match r and the string formed by the children of v match s (c.f.. [7, 8] for more details). The single-type restriction further ensures that XSDs can be uniquely typed in a one-pass top-down fashion. To be precise, one-pass typing in a top-down fashion means that the first time a node is visited a type should be assigned (so only based on what has been seen up to now) and that a child can be visited only when its parent is already visited. Type inclusion and equivalence is EXPTIME-complete for XSchemas and is in PTIME for DTDs and XSDs. In fact, w.r.t. the latter, the problem reduces to the corresponding problem for the class of employed regular expressions [8].

CROSS REFERENCE

XML schema, XML Typechecking

RECOMMENDED READING

Between 3 and 15 citations to important literature, e.g., in journals, conference proceedings, and websites.

[1] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993.

[2] A. Brüggemann-Klein and Wood D. One unambiguous regular languages. *Information and Computation*, 140(2):229–253, 1998.

[3] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKUST-TCSC-2001-0, The Hongkong University of Science and Technology, 2001.

[4] J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *RTA*, pages 105–118, 2004.

[5] J. Cristau, C. Löding, and W. Thomas. Deterministic automata on unranked trees. In *FCT*, pages 68–79, 2005.

[6] H. Hosoya and M. Murata. Boolean operations and inclusion test for attribute-element constraints. *Theoretical Computer. Science.*, 360(1-3):327–351, 2006.

[7] W. Martens, F. Neven, and T. Schwentick. Simple off the shelf abstractions for XML Schema. *SIGMOD Record*, 36(4), 2007.

[8] W. Martens, F. Neven, T. Schwentick, and G.J. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[9] W. Martens and J. Niehren. Minimizing tree automata for unranked trees. In *Prodeedings of the 10th International Symposium on Database Programming Languages (DBPL 2005)*, pages 232–246, 2005.

[10] M. Murata, D. Lee, M. Mani, and K. Kawaguchi. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technology*, 5(4):660–704, 2005.

[11] F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3), 2002.

[12] F. Neven and T. Schwentick. Query automata on finite trees. *Theoretical Computer Science*, 275:633–674, 2002.

[13] L. Segoufin and V. Vianu. Validating streaming XML documents. In *Proceedings of the 21th Symposium on Principles of Database Systems (PODS 2002)*, pages 53–64. ACM Press, 2002.

[14] J. W. Thatcher. Characterizing derivation trees of context-free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, 1(4):317–322, 1967.