Expressiveness of XSDs: from Practice to Theory, There and Back Again

Geert Jan Bex
Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek, Belgium
gjb@luc.ac.be

Wim Martens
Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek, Belgium
wim.martens@luc.ac.be

Frank Neven
Limburgs Universitair Centrum
Universitaire Campus
B-3590 Diepenbeek, Belgium
frank.neven@luc.ac.be

Thomas Schwentick
Philipps Universität Marburg
Fachbereich 12
Mathematik und Informatik
tick@informatik.uni-marburg.de

ABSTRACT

On an abstract level, XML Schema increases the limited expressive power of Document Type Definitions (DTDs) by extending them with a recursive typing mechanism. However, an investigation of the XML Schema Definitions (XSDs) occurring in practice reveals that the vast majority of them are structurally equivalent to DTDs. This might be due to the complexity of the XML Schema specification and the difficulty to understand the effect of constraints on typing and validation of schemas. To shed some light on the actual expressive power of XSDs this paper studies the impact of the Element Declarations Consistent (EDC) and the Unique Particle Attribution (UPA) rule. An equivalent formalism based on contextual patterns rather than on recursive types is proposed which might serve as a light-weight front end for XML Schema. Finally, the effect of EDC and UPA on the way XML documents can be typed is discussed. It is argued that a cleaner, more robust, stronger but equally efficient class is obtained by replacing EDC and UPA with the notion of 1-pass preorder typing: schemas that allow to determine the type of an element of a streaming document when its opening tag is met. This notion can be defined in terms of restrained competition regular expressions and there is again an equivalent syntactical formalism based on contextual patterns.

Categories and Subject Descriptors

I.7.2 [**Documents and text processing**]: Document preparation—*Markup languages, XML*

General Terms

Languages, Theory

Keywords

XML Schema, expressiveness, formal model

1. INTRODUCTION

One of the many criticisms on Document Type Definitions (DTDs) was their restricted expressiveness. On an abstract level, XML Schema obtains a higher expressive power by extending DTDs with a recursive typing mechanism which allows to define types in terms of types. However, an investigation of the XML Schema Definitions (XSDs) occurring in practice reveals that the vast majority of them are structurally equivalent to a DTD.

More precisely, we harvested a large corpus of XSDs from the Web and investigated the use of features not present in DTDs such as namespaces, import facilities, built in basic types, keys, and also the ability to use the same element name with different types.

Concerning expressive power we were surprised that only 15% of the syntactically correct XSDs used typing in a way that goes beyond the power of DTDs. A possible explanation is that the actual modeling power of XSDs remains unclear to most users: the XML Schema specification is very hard to read and the effect of constraints on typing and validation is not fully understood.

The present paper has four main contributions:

(1) We give a detailed account of the features of XSDs that are actually used in practice.

(2) We try to shed some light on the actual expressive power of XSDs by discussing the impact of the Element Declarations Consistent (EDC) rule. It turns out that there is a simple criterion to check whether a set of documents can be described by an XSD fulfilling EDC.

(3) We propose a framework for pattern based specification of sets of documents and show that one of its simplest instantiations leads to a language with exactly the same power as XSDs with EDC, but avoiding the complexity of recursive types. We believe that with this framework a light-weight front-end for XML Schema can be built suitable for less experienced users. Alternatively, we discuss how XML Schema itself can be conservatively extended with contextual patterns. Furthermore, we

exemplify how the use of contextual patterns reduces duplication of definitions.

(4) Finally, we discuss the effect of the EDC and the Unique Particle Attribution (UPA) constraints on the way XML documents can be typed. These constraints allow efficient validation and typing of documents by ensuring that the type of an element never depends on its content or the following elements. Hence, the type of an element of a streaming document can be assigned when its opening tag is met. We refer to the latter requirement as *1-pass preorder typability.* Although EDC and UPA ensure 1-pass preorder typability they are not necessary for it. More interestingly, it turns out that 1-pass preorder typability is a very robust notion with various clean semantical and syntactical characterizations. In particular, it can be defined in terms of restrained competition regular expressions (introduced by Murata, Lee, and Mani [14]) and by an equivalent syntactical formalism based on contextual patterns which can serve as a light-weight front-end. We therefore propose to replace the rather ad-hoc EDC and UPA constraints by the more robust 1-pass preorder typability requirement thereby obtaining the maximal expressiveness in terms of typing in a streaming fashion.

These contributions are based on theoretical results obtained in previous work conducted by three of us [13]. The main goal of this paper is to make these results accessible to a more practical oriented audience and discuss their implications on the design of XML Schema.

The paper is organized as follows. In Section 2, we recall basic properties of DTDs, the notion of (single-type) specialized DTDs which correspond to XSDs respecting the EDC constraint. In Section 3, we give a more detailed report on the structural properties of XSDs that we found on the Web. Section 4 investigates XSDs with the EDC constraint more closely, and introduces the pattern based paradigm mentioned in (2) and its instantiation matching the expressiveness of XSDs with EDC. Section 5 defines the notion of 1-pass preorder typability and explains its relationship to EDC and UPA. In Section 6, we characterize the expressive power of 1-pass preorder typability and describe a simple specification language based on the pattern paradigm invented before. The paper concludes with Section 7.

2. SCHEMAS AND TYPES

In the present section, we provide the necessary background on the mathematical formalisms we employ as abstractions of DTDs and XSDs.

Since ordered trees serve as the logical data model for XML [31], we employ a tree based abstraction of XML documents. We focus in this work on the structure of XML documents and disregard data values, attributes, namespaces, and linking information. We refrain from giving a formal definition, but instead, give an example of an XML document and its tree representation in Figure 1. For clarity we also displayed the data values though they will be disregarded in the rest of the paper (cf. Example 1).

In this respect, a schema defines a set of allowed trees (also referred to as a tree *language* in formal language theory). The DTD in Figure 2, for instance, defines the set of trees where the root is labeled with `store`; the children of `store` are `dvd` elements; every `dvd` element has a `title`, `price`, and an optional `discount` child.

```
<store>
  <dvd>
    <title>Amelie</title>
    <price>17</price>
  </dvd>
  <dvd>
    <title>Good bye, Lenin!</title>
    <price>20</price>
    <discount>20%</discount>
  </dvd>
</store>
```

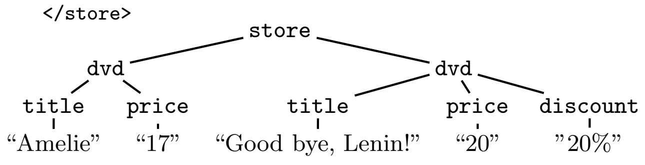


Figure 1: An XML document and its abstraction as a tree.

2.1 DTDs

It is customary to abstract DTDs by sets of rules of the form $a \to r$ where a is an element and r is a regular expression over the alphabet of elements. One element name is designated as the start symbol. For instance, the DTD of Figure 2 is represented by the set of rules:

$$\begin{array}{lcl} \texttt{store} & \to & \texttt{dvd}\,\texttt{dvd}^* \\ \texttt{dvd} & \to & \texttt{title}\,\texttt{price}\,(\texttt{discount} + \varepsilon) \end{array}$$

with start symbol `store`.

```
<!ELEMENT store       (dvd+)>
<!ELEMENT dvd         (title, price, discount?)>
<!ELEMENT title       (#PCDATA)>
<!ELEMENT price       (#PCDATA)>
<!ELEMENT discount    (#PCDATA)>
```

Figure 2: Example of a DTD describing the document in Figure 1.

Papakonstantinou and Vianu [16] provided a characterization of the structural expressive power of DTDs: a regular[1] set T of trees is definable by a DTD iff T has the following closure property: if two trees t_1 and t_2 are in T, and there are two nodes v_1 in t_1 and v_2 in t_2 with the same label, then the trees obtained by exchanging the subtrees rooted at v_1 and v_2 are also in the set T. We refer to this property as *label-guarded subtree exchange.* We illustrate this notion in Figure 3. Because of the latter characterization, the classes of XML documents defined by DTDs are also referred to as *local* classes. The characterization can be used to prove that certain languages can *not* be expressed by a DTD as exemplified in the following example.

EXAMPLE 1. *Suppose that we want to put the extra constraint on the DTD of Figure 2 requiring the presence of at least one DVD on discount. Then we get a language that is* not *definable by a DTD anymore. We can prove this by applying the above characterization. Indeed, the trees*

[1]The notion of a regular set of trees is defined in Section 2.2.

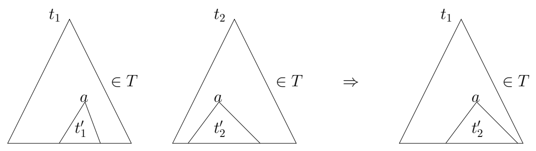
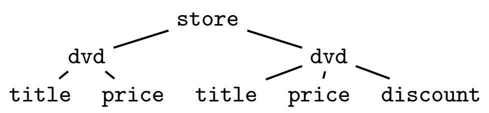


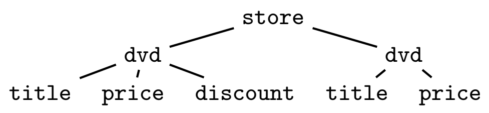
Figure 3: Label-guarded subtree exchange.

$t_1 :=$

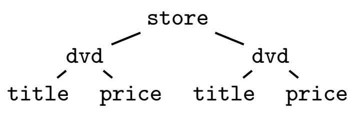


and

$t_2 :=$



are in the language, but the tree



which is obtained from t_1 by replacing its second subtree by the second subtree of t_2, is not *in the language.*

2.2 DTDs Plus Types

As discussed in the introduction, the expressive power of DTDs can be increased by adding types, as, e.g., in XML Schema and Relax NG [4]. There is a finite set of possible types that every element can assume. For notational simplicity, we denote types for element a by terms a^i with $i \in \mathbb{N}$. As can be seen in Example 2, rules are now of the form $a^i \to r$, where r is a regular expression over types (also referred to as specializations). The designated start symbol has only one type associated with it. These *specialized DTDs* (SDTDs) were introduced by Papakonstantinou and Vianu [16].

Formally, a tree satisfies an SDTD if there exists an assignment of types such that the typed tree is a derivation tree of the grammar. The following example displays an alternative SDTD for the tree in Figure 1. For clarity, types are not displayed for element names having a unique type.

Example 2. *Consider the following SDTD:*

$$\begin{array}{lll} \texttt{store} & \to & (\texttt{dvd}^1 + \texttt{dvd}^2)^*\texttt{dvd}^2(\texttt{dvd}^1 + \texttt{dvd}^2)^* \\ \texttt{dvd}^1 & \to & \texttt{title price} \\ \texttt{dvd}^2 & \to & \texttt{title price discount} \end{array}$$

Intuitively, $\texttt{dvd}^1$ *defines ordinary DVDs while* $\texttt{dvd}^2$ *defines DVDs on sale. The first rule specifies that there has to be at least one DVD on discount. The tree in Figure 1 satisfies this SDTD as assigning* $\texttt{dvd}^1$ *and* $\texttt{dvd}^2$ *to the left and right* dvd*-node respectively, gives a derivation tree of the grammar.*

In Figure 4, a fragment of an XSD corresponding to the rule for store is depicted. We note that the XSD is not syntactically correct because it violates EDC as explained in the next section.

```
<xs:element name="store">
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="dvd" type="dvd1"/>
        <xs:element name="dvd" type="dvd2"/>
      </xs:choice>
      <xs:element name="dvd" type="dvd2"/>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="dvd" type="dvd1"/>
        <xs:element name="dvd" type="dvd2"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 4: A fragment of an XSD (violating EDC) corresponding to the SDTD of Example 2.

From a structural perspective, SDTDs express exactly the *regular tree languages*; a similarly robust class as the regular string languages. In particular, SDTDs are as expressive as unranked tree automata [2, 15]. It should be noted that the formal underpinnings of the schema language Relax NG are also based upon regular tree languages.

2.3 XML Schema = SDTD + EDC

As noted by Murata et al. [14], XSDs can be abstracted by SDTDs extended with the Element Declarations Consistent rule (EDC). Roughly, the constraint forbids the occurrence of elements with the same name but different types in the same definition. For instance, the XSD of Figure 4 is not allowed as dvd occurs in the presence of the two types dvd1 and dvd2 in the same rule.

The EDC rule can be formalized by requiring an SDTD to be "single-type", defined as follows [13, 14]:[2]

Definition 1. *A* single-type SDTD *is an SDTD in which no regular expression two types b^i and b^j with $i \neq j$ occur.*

The SDTD of Example 2 is not single type as both $\texttt{dvd}^1$ and $\texttt{dvd}^2$ occur in the rule for store. An example of a single-type SDTD is given below.

Example 3.

$$\begin{array}{lll} \texttt{store} & \to & \texttt{regulars discounts} \\ \texttt{regulars} & \to & (\texttt{dvd}^1)^* \\ \texttt{discounts} & \to & \texttt{dvd}^2\ (\texttt{dvd}^2)^* \\ \texttt{dvd}^1 & \to & \texttt{title price} \\ \texttt{dvd}^2 & \to & \texttt{title price discount} \end{array}$$

Although there are two element definitions $\texttt{dvd}^1$ *and* $\texttt{dvd}^2$*, they occur in a different rule.*

Although the definition of single-type grammars is quite transparent, its effect on definable classes of XML documents is less so. In Section 4 and Section 5, we return to this issue and discuss the practical implications of the latter restriction on the expressiveness and typing algorithms for XSDs. First, we investigate in Section 3 to which extent the newly added features of XSDs (in comparison with the features of DTDs) are effectively used in practice.

[2]Actually, [14] used the equivalent model of tree grammars instead of SDTDs.

2.4 Attributes and Data Values

DTDs and XML Schema have different expressive power on attribute level, element level and data-value level. All these issues can be easily incorporated into the SDTD model by adapting the tree representation of XML documents. However, to keep the formalism simple, we focus in the present paper on the element structure of XML documents.

3. A PRACTICAL STUDY OF XSDS

A variety of sources [7, 9, 11] discuss the many drawbacks of DTDs: no modularity, no XML syntax, limited basic types, restricted referencing mechanism, verbose specification of unordered data, and limited expressiveness (definition of an element cannot depend on its context). Most of these concerns have been addressed by the XML Schema specification: namespaces and import facilities have been added; an extensive number of built in basic types as well as means to fine tune them by restriction are provided; XML Schema supports key and referential integrity; the `all` construct allows to specify unordered content; and finally, the same element name can be defined having different types. Of course, this raises the question to what extent the added features are actually used in practice. To this end, we studied a corpus of 819 XSDs harvested from the Web. Among the XSDs gathered, 93 were retrieved via the Cover Pages [6].[3] Hence, a substantial number of high-quality XSDs representing various standards such as the XML Schema Specification, XHTML, UDDI, RDF and others are represented in the corpus. Unfortunately this number is rather small, so the corpus was enlarged to its present size of 819 XSDs using Google's web services to retrieve an additional 726 XSDs.

The results concerning the use of syntactical features are summarized in Table 1. From this table one can conclude that XML Schema's simpleType library and the ability to place restrictions on those are heavily used. Derivation in the sense of the object orientation paradigm is only used in about 1/5 of all XSDs. Modularity by way of imports and (non-trivial) namespaces is fairly important as well. Uniqueness and key references are fairly uncommon.

As explained above, XSDs employ recursively defined types to increase its expressiveness beyond DTDs. The question remains whether XSDs occurring in practice actually use this feature. That is, what percentage of found XSDs are not structurally equivalent to a DTD? Unfortunately, out of the corpus of 819 harvested XSDs only 225 remain on which IBM's SQC [30] reports no errors.[4] Although syntactical correctness is less critical in testing for presence or absence of syntactical features, it is mandatory for the expressiveness analysis which is more semantical in nature. It is impossible to automatically guess for every syntactically incorrect XSD what its designer intended.

It turns out that out of the remaining 225 XSDs, 192 (85%) are in fact structurally equivalent to a DTD: at most one type is associated to every element name.[5] So only 33 XSDs (15%) used the typing mechanism to actually define non-local classes of XML documents. Surprisingly, in 30 XSDs, types only depend on the parent context. That is, they were of the kind as defined in Example 3, where the type of `dvd` only depends on the label of the parent (`regulars` or `discounts`). So, although non-trivial specialization is moderately used in practice, it is almost exclusively used in its most simplistic form: dependence on the label of the parent. Recall that for DTDs, the type of an element only depends on its own label. Basically, there are two possible explanations for the above observation. Either, advanced modeling power as allowed by SDTDs is not necessary in practice. Or, users are simply not aware of what kind of schemas can be expressed by XSDs. In Section 4.1, we attempt to address the latter concern as we explain what is theoretically possible when using XSDs. The former possibility is analyzed below in more detail.

feature	**% of XSDs**
derivation	
simpleType extension	18.9
simpleType restriction	45.5
complexType extension	20.7
complexType restriction	3.6
abstract attribute	9.8
final attribute	0.9
block attribute	0.0
fixed attribute	6.4
substitutionGroup	6.4
redefine	1.0
interleaving	
`xs:all`	5.5
modularity	
namespaces	12.1
import	27.7
linking	
key/keyref	4.1
unique	2.9

Table 1: XML Schema features use in the corpus

In the remaining 3 XSDs, types depend on the grand- or the great grand-parent context. We discuss an abstraction of one of them as an SDTD:

Example 4.

$$\begin{array}{rcl@{\qquad}rcl}
a & \to & b + c & h^1 & \to & j^1 \\
b & \to & e\ d^1\ f & h^2 & \to & j^2 \\
c & \to & e\ d^2\ f & j^1 & \to & k\ l \\
d^1 & \to & g\ h^1\ i & j^2 & \to & m\ n \\
d^2 & \to & g\ h^2\ i & & &
\end{array}$$

The interpretation of the example above is simple: an j^1 element can only occur as the great grandchild of a b element while an j^2 element can only occur as the great grandchild of a c element.

Two extreme approaches can be used to code the abstract example above in an XSD. On the one hand, one can use the "Russian doll" model, i.e., using anonymous type definitions within type definitions. In an abstract syntax the latter

[3]A previous study only focused on the Cover Pages and also investigated the structure of used regular expressions [1].

[4]Even worse, already 70% of the XSDs from the Cover pages do not pass the syntax checker SQC. In this respect it is interesting to note that Sahuguet reported similar findings on the sheer abundance of syntactically incorrect DTDs [18].

[5]Actually, we encountered one XSD using types to define a local language. The corresponding SDTD is of the form: $a^1 \to b$, $a^2 \to b$ where the types differ semantically.

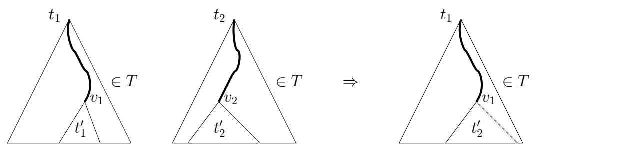


Figure 5: Ancestor-guarded subtree exchange.

reduces to the rules

$$b \rightarrow ed[gh[j[kl]]i]f \quad \text{and} \quad c \rightarrow ed[gh[j[mn]]i]f,$$

where the type definition of the element b encapsulates that of d^1 which in turn defines that of h^1 that finally contains j^1's definition. The alternative is to flatten the XSD as has been done in Example 4, but this leads to "artificial types" such as d^1 and h^1 that only exist to pass down the information that their parent and grandparent was a b-element. It is obvious that in practice both approaches are mixed to a certain extent. However, both lead to duplication of definitions, making maintenance and further development of an XSD much harder.

In view of the concerns mentioned in the previous paragraphs, we propose in Section 4.2 a general framework for XML schema languages that allows types to depend on the label of ancestors. It would no longer be required to define types in terms of types, which seems to be perceived as a challenge by the average XSD author, and would allow him to access the full power offered by XML Schema. Duplication of definitions would be reduced as well since dependencies on ancestor labels can straightforwardly be declared rather than being passed down via types.

4. EXPRESSIVENESS OF XML SCHEMA

As explained in Section 2.3, content models are restrained by the Element Declarations Consistent rule. Of course, immediately a question arises: what kind of classes of XML documents can be defined in the presence of this constraint? To this end, we discuss a subtree exchange property that characterizes the expressiveness of XSDs completely. Furthermore, we propose a simpler paradigm equivalent to XSDs which can serve as a light-weight front-end for XML Schema (for instance, for less experienced users). Alternatively, the paradigm can be combined with the actual XML Schema specification reducing artificial definition duplication.

4.1 Subtree Exchange Property

Recall the notion of label-guarded subtree exchange which characterized the class of DTD-definable languages. In [13, Theorem 11], it is shown that a set T of documents is definable by an SDTD satisfying EDC if and only if T is definable by an SDTD and T is closed under ancestor-guarded subtree exchange, i.e., if the following holds (cf. Figure 5): if two documents t_1 and t_2 are in T, and there are two ancestor equivalent elements v_1 in t_1 and v_2 in t_2, then the trees obtained by exchanging the subtrees rooted at v_1 and v_2 are also in the set T. Here, *ancestor equivalent* means that the sequence of labels on the path from the root of t_1 to v_1 is the same as on the path from the root of t_2 to v_2 (cf. Figure 6).

As an immediate consequence, the language we considered in Example 1 is not definable by a single-type SDTD. Note that the counterexample can be constructed in exactly the same manner. On the other hand, the language defined by the single-type SDTD in Example 3 is not definable by a DTD, so single-type SDTDs are strictly more expressive than DTDs. As a matter of fact, it can be decided in exponential time whether a given SDTD is equivalent to one that satisfies EDC. Unfortunately, this complexity bound is also tight [13, Theorem 14].

The importance of the above theoretical characterization stems from the fact that inexpressibility results can be formally proved rather than vaguely stated. For instance, a shortcoming recently attributed to XSDs is their inability to express certain co-constraints [17]. An example of such a co-constraint is: a `store`-element can only have a `dvd`-element with `discount` child if it also has a `dvd`-element without a `discount` child. Using the ancestor-guarded subtree exchange property, it is very easy to formally prove that this co-constraint cannot be expressed with XSDs. Indeed, the counterexample is constructed from t_1 in Example 1 by replacing its first subtree by the first subtree of t_2.

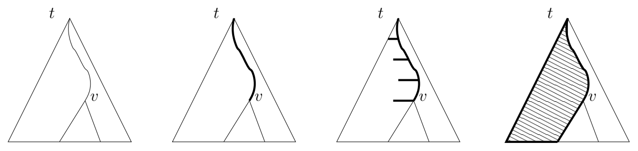


Figure 6: From left to right: a tree t, the ancestor-string of v, the ancestor-sibling-string of v and the preceding of v in t.

4.2 A General Framework: $\mathcal{P}$-Schemas

Although the subtree exchange property characterizes the classes of XML documents definable by XSDs, it is more a means to show that schemas are not definable. Therefore, the average user would benefit from a simple specification language allowing to express desired schemas in a transparent way. The conducted practical study reveals that the vast majority of non-local XSDs lets types depend only on the parent context. We therefore propose a framework in which such dependence can be made explicitly. The framework is related to the paradigm upon which languages like Schematron [19] and DSD [10] are based (contextual patterns). We combine the latter with non-recursive typing. The main difference, however, is that our framework is equivalent to the expressiveness of XML Schema: every such schema can be translated into an XSD. We discuss in Section 4.3 how to transform the framework into a workable form.

To ensure maximal flexibility, in a first stage, we only use patterns in an abstract way. Therefore, let $\mathcal{P}$ be a pattern language (e.g., XPath, regular expressions,...) defining unary patterns. That is, each pattern $\varphi \in \mathcal{P}$ associates with every tree a set of selected nodes. For instance, the XPath expression `//hobbit` defines the pattern that selects all `hobbit`-elements. We denote the set of nodes selected by φ on t by $\varphi(t)$. Let Σ and Types be finite alphabets of element and type names, respectively.

DEFINITION 2. *A $\mathcal{P}$-schema is a triple $S = (\Sigma, Types, R)$ where R is a finite set of rules of the form $(\alpha, \varphi) \Rightarrow r$. Here, $\alpha \in Types$, $\varphi \in \mathcal{P}$, and r is a regular expression over Σ.*

The semantics of a $\mathcal{P}$-schema is defined in two phases: (1) checking conformance w.r.t. the grammar; and (2) assignment of types (also referred to as schema-validity assessment in [26]).

Definition 3. *A tree t is* valid *w.r.t. a $\mathcal{P}$-schema $S := (\Sigma, \mathit{Types}, R)$ if the label of every node belongs to Σ and for every node v of t there is a rule $(\alpha, \varphi) \Rightarrow r$ such that $v \in \varphi(t)$ and the children of v match the regular expression r.*

If there is only one rule $(\alpha, \varphi) \Rightarrow r$ such that $v \in \varphi(t)$, v is assigned the type α.

In examples, we use the short hand $a \rightarrow r$ to define the rule $(a, //a) \Rightarrow r$ specifying that the children of every a-element should match regular expression r.

Example 5. *In the current framework, using XPath as a pattern language, the SDTD of Example 3 is equivalent to the following schema:*

$$\begin{array}{rcl}
\texttt{store} & \rightarrow & \texttt{regulars discounts} \\
\texttt{regulars} & \rightarrow & \texttt{dvd}^* \\
\texttt{discounts} & \rightarrow & \texttt{dvd dvd}^* \\
(\texttt{regular-dvd}, //\texttt{regulars/dvd}) & \Rightarrow & \texttt{title price} \\
(\texttt{discount-dvd}, //\texttt{discounts/dvd}) & \Rightarrow & \texttt{title price discount}
\end{array}$$

Here, Types $= \{\texttt{discount-dvd}, \texttt{regular-dvd}\}$. *Intuitively, a* `dvd` *element is a* `regular-dvd` *(*`discount-dvd`*) when its parent label is* `regulars` *(*`discounts`*); its content model is then determined by the regular expression* `title price` *(*`title price discount`*). For clarity we used the types* `regular-dvd` *and* `discount-dvd` *rather than the cryptic types* $\texttt{dvd}^1$ *and* $\texttt{dvd}^2$ *of Example 3. Clearly, the patterns* //`regulars`/`dvd` *and* //`discounts`/`dvd` *are disjoint. Hence, a unique type can be assigned to every* `dvd` *element*

Remark 1. *Definition 3 does not require that each node has a unique type. If unique types are desired one can add the requirement that $\varphi(t) \cap \varphi'(t) = \emptyset$ for all patterns φ, φ' related to the same tag and all trees t. For most pattern languages occurring in practice, it can be statically checked whether this requirements holds, e.g., for XPath cf. [20]. Hence, automatic tools can be developed to assist schema development. Alternatively, one can assign natural numbers to rules reflecting priorities in case of conflicts.*

Example 6. *The $\mathcal{P}$-schema equivalent to the SDTD of Example 4:*

$$\begin{array}{rclcrcl}
a & \rightarrow & b + c & \quad & h & \rightarrow & j \\
b & \rightarrow & e\,d\,f & & (j^1, //b//j) & \Rightarrow & k\,l \\
c & \rightarrow & e\,d\,f & & (j^2, //c//j) & \Rightarrow & m\,n \\
d & \rightarrow & g\,h\,i & & & &
\end{array}$$

Also in this case, types can be uniquely attributed.

Obviously, DTDs are $\mathcal{P}$-schemas where every rule is an $\rightarrow$-rule and the type of an element corresponds to its name. It remains to discuss how the above framework relates to single-type SDTDs, our abstraction of XSDs. Therefore, let $\mathcal{R}$ denote the class of regular expressions for Σ-strings. Regular expressions φ can be used as unary patterns in the following way: φ selects those nodes v on a tree t whose sequence of labels on the path from the root to v satisfies φ. We refer to the latter string as the ancestor-string of v (cf. Figure 6). The last two rules in the $\mathcal{R}$-schema equivalent to the $\mathcal{P}$-schema of Example 5 then become

$$\begin{array}{rcl}
(\texttt{regular-dvd}, \Sigma^* \cdot \texttt{regulars} \cdot \texttt{dvd}) & \Rightarrow & \texttt{title price} \\
(\texttt{discount-dvd}, \Sigma^* \cdot \texttt{discounts} \cdot \texttt{dvd}) & \Rightarrow & \texttt{title price discount}
\end{array}$$

Here, Σ^* denotes the set of all Σ-strings.

In [13, Theorem 11], it is shown that the class of $\mathcal{R}$-schemas corresponds *precisely* to the class of schemas represented by SDTDs satisfying EDC. In other words, the instantiation of the general framework with regular expressions over ancestor-strings can be used as an alternative syntax for XML Schema. XML experts already noticed that XSDs allow to express dependence of content models on ancestors (cf. [29]). However, the above characterization shows that this dependence can be extended to regular expressions over the ancestors and, moreover, that it cannot be extended any further. Actually, in [13], we did not introduce a general framework, but considered ancestor-guarded DTDs.

We illustrate the translation of $\mathcal{R}$-schemas into single-type SDTDs by means of an example. In general, the resulting SDTD can be exponentially larger. However, judging from the complexity of found XSDs, this will seldom be the case in practice. The schema of Example 5 translates into the following SDTD:[6]

$$\begin{array}{rcl}
\texttt{store}^{(\cdot,\cdot)} & \rightarrow & \texttt{regulars}^{(\text{reg},\cdot)}\ \texttt{discounts}^{(\cdot,\text{disc})} \\
\texttt{regulars}^{(\text{reg},\cdot)} & \rightarrow & (\texttt{dvd}^{(\text{reg-dvd},\cdot)})^* \\
\texttt{discounts}^{(\cdot,\text{disc})} & \rightarrow & \texttt{dvd}^{(\cdot,\text{disc-dvd})}\ (\texttt{dvd}^{(\cdot,\text{disc-dvd})})^* \\
\texttt{dvd}^{(\text{reg-dvd},\cdot)} & \rightarrow & \texttt{title}^{(\cdot,\cdot)}\ \texttt{price}^{(\cdot,\cdot)} \\
\texttt{dvd}^{(\cdot,\text{disc-dvd})} & \rightarrow & \texttt{title}^{(\cdot,\cdot)}\ \texttt{price}^{(\cdot,\cdot)}\ \texttt{discount}^{(\cdot,\cdot)}
\end{array}$$

The translation algorithm first converts every pattern to a deterministic finite automaton. Then it constructs the product automaton A of all obtained automata in the standard manner. We use states of A as types for the SDTD. The initial state of A is assigned to the start symbol of the SDTD and every symbol b occurring on the right-hand side of a rule with left-hand side a^s is replaced by $b^{s'}$ where s' is the state reachable from state s by reading a b.

In this example, we convert the pattern `//regulars/dvd` to the automaton A_1 depicted in Figure 7. Note that the latter automaton is optimized w.r.t. the strings that can occur as paths in derivation trees of the SDTD. The automaton A_2 corresponding to `//discounts/dvd` is similar. As an example of the type assignments, the initial state of $A_1 \times A_2$ is assigned to the start symbol `store`. Then, as A_1 enters state reg when reading `regulars` and A_2 remains in its initial state, we assign state $(\text{reg}, \cdot)$ of $A_1 \times A_2$ to `regulars`. Removing the types for elements having only one type then results in the SDTD equivalent to that of Example 3.

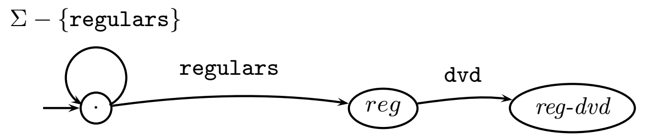


Figure 7: An automaton representing the pattern //regulars/dvd.

4.3 $\mathcal{R}$-Schemas in Practice

The usefulness of an XML schema language requires more than a thorough understanding of its expressiveness. Therefore, we discuss in the present section how we can migrate $\mathcal{R}$-schemas into a full fledged schema language. Rather than proposing yet another schema language we stipulate how existing languages and proposals can be adapted.

[6]As illustrated above, linear XPath expressions can be defined by regular expressions.

Several approaches guided by our practical study are conceivable. We suggest a two-pronged approach: on the one hand an extension to the DTD specification for those most comfortable with this formalism which probably includes inexperienced or new users, on the other hand an extension of XML Schema more suited for power users. However, these seemingly different approaches converge behind the scenes since schemas developed both according to the DTD extension as to the XML Schema extension can be translated into an XSD valid with respect to the current XML Schema specification. We discuss both proposals in more detail.

4.3.1 An extension of DTDs

The most direct approach is to extend DTDs to the formalism of $\mathcal{R}$-schemas as exemplified in Example 7. Instead of regular expressions, one could allow "linear XPath" expressions, incorporating only the axes child and descendant. The results mentioned in Section 3 suggest that the latter expressiveness would be sufficient to structurally capture the XSDs occurring in practice, though the power of full regular expressions is needed to capture all of XML Schema.

EXAMPLE 7. *The real world XSD of Example 4 can be rewritten as the following extended DTD:*

```
<!ELEMENT a (b | c)>
<!ELEMENT b (e, d, f)>
<!ELEMENT c (e, d, f)>
<!ELEMENT d (g, h, i)>
<!ELEMENT h (j)>
<!ELEMENT "//b//j" j1 (k, l)>
<!ELEMENT "//c//j" j2 (m, n)>
```

It is clear that the representation in Example 7 is much more compact than the corresponding XSD, and that duplicate definitions have been avoided altogether. Note that the evaluation of the linear XPath expression starts at the document root and that only the child and descendant axes are used. Hence one can limit oneself to the abbreviated syntax ('/' and '//') which substantially contributes to the transparency of the expressions.

To alleviate the problem of the very limited set of data types in DTDs, we propose the addition of the simple data types as defined in the XML Schema specification [27] for attributes and text. As we are not the first to propose an extension of DTDs for expressing schemas for XML, we do not further go into details here. Already two such proposals for extensions of DTDs exist [24, 28]. Both focus heavily on the addition of data types to DTDs. The former (DTD++ 1.0) also introduces namespaces and complex objects. To the best of our knowledge we are the first to justify such a proposal both by a practical study and a theoretical analysis. Indeed, in strong contrast to DTD++ 1.0, restriction to $\mathcal{R}$-schemas can structurally define all XSDs.

A superficially similar approach is taken in the specification of DTD++ 2.0 [8], however, the focus is entirely different. The emphasis in DTD++ 2.0 is on the expression of co-constraints so that the resulting specification exceeds the expressive power of XML Schema. Indeed, DTD++ 2.0 schemas are transformed into SchemaPath [17] which is strictly more expressive than XML Schema and that requires a transformation of the XML documents prior to validation. Our approach avoids the overhead of translating XML documents and leverages the use of existing XML Schema implementations and tools.

As illustrated in Section 4.2, the above proposed extended DTDs can automatically be translated into equivalent XSDs. This translation introduces many cryptic types (states of an automaton) so that it may be very difficult for a user to understand and also edit the resulting XSD. Such post processing might sometimes be needed, e.g., to add key/keyrefs, or to rename types. In that case, we can restrict to a limited vocabulary (`parent`, `grandparent`, `greatgrandparent` and `ancestor`) as opposed to linear XPath or regular expressions, that can be translated into more transparent types and is nevertheless sufficiently powerful to express all context dependencies found in the corpus we analyzed. This would facilitate the development of true round-trip editing software between extended DTDs and the generated XSD where the overall structure of the XML documents can be specified in the form of an extended DTD while nuts and bolts details can be fleshed out in the generated XSD.

4.3.2 A conservative extension of XML Schema

The second option is to extend the XML Schema specification in such a way that element type definitions are context dependent. A syntactic approach using conditional alternatives similar to SchemaPath [17] is suggested. However, rather than full XPath expressions, conditions would be limited to linear XPath (or general regular expressions) so that the expressive power of XML Schema is not exceeded as was discussed in the previous paragraph. Whereas the extended DTDs are more expressive than traditional DTDs, extended XML Schemas provide only syntactic sugar to ease the development and make XML Schema more legible and easier to maintain since a lot of definition duplications can be eliminated.

EXAMPLE 8. *The essential fragment of Example 4 rewritten as an extended XSD:*

```
<xs:element name="j">
  <xs:alt cond="//b/j" type="j1"/>
  <xs:alt cond="//c/j" type="j2"/>
</xs:element>
```

Example 8 shows a conditional element definition: element `j` is of type `j1` (`j2`) if it has a `b` (`c`) parent.

5. TOWARDS A ROBUST NOTION OF TYPING

As mentioned before, the expressive power of SDTDs (and Relax NG) corresponds to the well-understood and very robust class of regular tree languages. However, this expressive power comes at a price. Although it can be determined in linear time whether a tree satisfies a given SDTD, the way to do that is sometimes at odds with the way one would like to process XML documents. More concretely, it requires to process documents in a bottom-up fashion where the type(s) of an element is only determined after reading its content. In the context of streaming XML data or for SAX based processing, i.e., when processing an XML document as a SAX-stream of opening and closing tags, it is more desirable to determine the type of an element at the time its opening tag is met. If an SDTD fulfills this requirement we say it is *1-pass preorder typable* (1PPT). Note that not every SDTD admits 1PPT. Consider the example $a \to b^1 + b^2$, $b^1 \to c$, $b^2 \to d$ and the document `<a><b><d/></b></a>`. The type of b depends on the label of its child. It is hence impossible to assign a type to b when its opening tag <b> is met, i.e., without looking at its child. An alternative formulation of 1PPT is that the type of an element cannot depend on anything occurring in document order after that element. Hence, we require that a type is uniquely determined by the preceding of an element (cf. Figure 6). On top of one-pass typability, this notion therefore also enforces a unique type to every element. The latter is, for instance, not the case for Relax NG which allows ambiguous typing.

In the XML Schema specification as well as in research papers various kinds of constraints have been defined that enable efficient validation and typing of XML documents. Although it is hardly made precise what this should mean exactly, one might argue that the intention roughly matches our notion of 1-pass preorder typability. It should be noted here that 1PPT is a semantical notion, while the proposed notion of single-type SDTDs, for instance, is a syntactic one as its definition refers to syntactic restrictions of the schema rather than to the documents themselves. However, 1PPT is a robust notion precisely because it is semantic: it defines the largest class of SDTDs that can be typed when processed in a streaming fashion. In the following we argue that the single-type restriction (EDC) is too ad-hoc, formalize 1PPT by means of the restrained-competition SDTDs of Murata, Lee, and Mani [14], and propose to replace UPA and EDC by 1PPT to get a more robust notion of typing.

In Section 6, we generalize the analysis for the current proposal of XSDs to 1PPT schemas. In particular, we characterize the expressiveness of 1PPT schemas and present a purely *syntactical* framework with corresponding expressibility based on $\mathcal{P}$-schemas. First, we discuss validation and the constraint on deterministic content models in the context of DTDs.

5.1 Validation of DTDs

Validation of a document against a DTD simply boils down to testing local consistency: does the string formed by the labels of the children of every a-labeled element satisfy the associated regular expression r? No notion of typing is available. To ensure efficient validation, regular expressions in right-hand sides of rules are required to be *deterministic* [25] (also referred to as *one-unambiguous* [3]). Intuitively, a regular expression is deterministic if, when processing the input from left to right, it is always determined which symbol in the expression matches the next input symbol. We discuss the latter notion a bit more formally as it returns in the specification of XML Schema in the form of the Unique Particle Attribution rule. For a regular expression r over elements, we denote by $\overline{r}$ the regular expression obtained from r by replacing every ith a-element in r (counting from left to right) by a_i. For example, when $r = (a+b)^*ac(b+c)^*$, then $\overline{r}$ simply is $(a_1+b_1)^*a_2c_1(b_2+c_2)^*$.

Definition 4. *A regular expression r is* one-unambiguous *iff there are no strings wa_iv and wa_jv' in $L(\overline{r})$ so that $i \neq j$.*

The restriction to deterministic regular expressions is heavily criticized (cf., e.g., p. 98 of [23] and [12]) as it does not serve its purpose: even for unrestricted regular expressions efficient simple validation algorithms exist. Further, the constraint is semantical in nature, and therefore it is difficult for the average user to assess whether a given regular expression is deterministic or not. To date, no transparent syntactical equivalent counterpart is known.

5.2 Typing of XSDs

As noted by Murata et al. [14], the EDC rule induces a simple top-down validation algorithm which assigns to every element with a symbol a a unique type a^i. The algorithm proceeds as follows: The unique type is assigned to the root; for each interior node u with type a^i, it checks whether the children of u match the right-hand side of $a^i \to r$, ignoring all types; if not the tree is rejected, otherwise, as the SDTD is single-type, to each child a unique type can be assigned. The tree is accepted if this process terminates at the leaves without any rejection.

In [13, Theorem 11], we characterize single-type SDTDs precisely as the class of SDTDs where the type of every element v in a tree pppis uniquely determined by the sequence of labels on the path from the root to v (i.e., the ancestor-string of v, cf. Figure 6). Therefore, every single-type SDTD admits 1PPT. The converse, however, is not true. Consider for example the following SDTD which is not single-type: $a \to b^1\,b^2,\ b^1 \to c,\ b^2 \to d$. Nevertheless, the SDTD admits 1PPT. Indeed, it is easy to see that the SDTD only defines the singleton <a><b><c/></b><b><d/></b></a>. The rule for a says that the first b-child needs to be typed b^1 and the second b-child needs to be typed b^2. For each of the b's in the document, it can be easily determined whether it is the first or second child of a by investigating its preceding (cf. Figure 6). Hence, the notion of single-type SDTDs allows for efficient unique typing, but does not capture the robust class of 1PPT SDTDs.

Apart from EDC, the XML specification also mentions the Unique Particle Attribution (UPA) constraint which is a rephrasing of the debatable determinism constraint for DTDs. Although the constraint does not mention typing explicitly, it does influence typing in an unexpected but drastic way. Surprisingly, as explained in Section 5.4, every SDTD which satisfies UPA already admits 1PPT.

5.3 Restrained-Competition SDTDs

Murata, Lee, and Mani [14], propose the semantical restriction to restrained competition regular expressions.

Definition 5. *A regular expression r (over Types)* restrains competition *if there are no strings wa^iv and wa^jv' in $L(r)$ with $i \neq j$. An SDTD is* restrained competition *iff all regular expressions occurring in rules restrain competition.*

Intuitively, a restrained competition regular expression ensures that when visiting the children of a node from left to right it is always clear which type is associated to each node without seeing its right siblings. An example of a restrained competition SDTD that is not single-type is given next:

store	$\to$	$(\texttt{dvd}^1)^*$ discounts $(\texttt{dvd}^2)^*$
discounts	$\to$	ε
$\texttt{dvd}^1$	$\to$	title price
$\texttt{dvd}^2$	$\to$	title price discount

This restriction allows a strictly larger class of schemas than EDC while still permitting a unique top-down left-to-right assignment of types as discussed in the next paragraph. Note that both the single-type and the restrained competition constraint are local: they restrain the structure of admissible regular expressions. Unfortunately, EDC is syntactic while restrained-competition is a semantical notion.

Nevertheless, deciding whether an SDTD is restrained competition can be done in polynomial time [13, Theorem 13]. In Section 6.2 we generalize the framework of Section 4.2 to get an equivalent but *syntactical* language for restrained competition SDTDs.

We discuss typing of restrained competition SDTDs. The *ancestor-sibling string* of an element is the string of ancestors of the element, *and their left siblings* (cf. Figure 6).[7] We say that an SDTD has *ancestor-sibling-based types* if the type of an element in a tree only depends on its ancestor-sibling string. In [13, Theorem 12], we characterize restrained competition SDTDs *exactly* as the class of SDTDs with ancestor-sibling-based types.

One of the most remarkable results of [13, Corollary 10] is that the SDTDs satisfying 1PPT are *exactly* the restrained competition SDTDs. Hence, the latter can serve as a concrete instantiation of the semantical concept of 1-pass preorder typing. A remarkable consequence is that in a streaming context, for *any* 1PPT SDTD, documents can be validated and typed in a 1-pass preorder fashion only using memory of size $O(|d|)$, where $|d|$ is the maximum *depth* of the document that is being validated. The validation algorithm is then the preorder version of the algorithm sketched in Theorem 5.1 of [21].

5.4 Discussion

Although the XML Schema specification allows typing in multiple passes (Section 5.2 in [26], note on multiple assessment episodes), we now show that, even without the EDC constraint, 1PPT is always enforced by the UPA constraint.

An SDTD satisfies the UPA constraint when for every regular expression r over Types we have that $\mathrm{el}(r)$ is one-unambiguous (cf. Definition 4), where $\mathrm{el}(r)$ denotes the expression obtained from r by replacing every type a^i by the element a. We argue by means of a concrete example that any SDTD satisfying UPA is restrained competition, and therefore, admits 1PPT. Suppose that $r = a^1?b^1(b^1 + c^1)^*a^2c^1$. Then, $\mathrm{el}(r) = a?b(b+c)^*ac$ and $\overline{\mathrm{el}(r)} = a_1?b_1(b_2+c_1)^*a_2c_2$. Clearly, $\mathrm{el}(r)$ is one-unambiguous, which means that when we match, e.g., *bbcbac* against $\mathrm{el}(r)$, the symbol against which the a must be matched (a_2 in $\overline{\mathrm{el}(r)}$), is uniquely determined without looking ahead. But then, the symbol in r that corresponds to a_2 is also uniquely determined, and this symbol has only one type. So we also know what type must be assigned to a without looking ahead to c. It is easy to generalize this example to show that any SDTD satisfying UPA is also restrained competition.

The converse, however, is not true. A counterexample is the expression $r = (a^1 + b^1)^*a^1(a^1 + b^1)$. Indeed, when matching a string against this expression, we always know that we need to type a and b by a^1 and b^1, respectively. However, the expression $\mathrm{el}(r) = (a + b)^*a(a + b)$ is *not* one-unambiguous. Indeed, $a_1a_2a_3$ and a_2a_3 are both in $L((a_1 + b_1)^*a_2(a_3 + b_2))$. In [3] it is even shown that $\mathrm{el}(r)$ can not be defined by *any* one-unambiguous regular expression.

To summarize, we propose to replace the UPA and EDC constraint in the XML Schema specification by the robust notion of 1PPT. The latter guarantees a large and robust class of schemas that can be uniquely (unambiguously) and efficiently typed. A local semantical definition in terms of the restrained competition regular expressions of Murata, Lee, and Mani can be adopted. Next, in Section 6.2, we present a purely *syntactical* front-end for 1PPT which is a natural instantiation of the framework of $\mathcal{P}$-schemas in Section 4.2 capturing all one-pass preorder typable SDTDs.

[7]Here the sibling strings are suitably separated from each other by additional symbols.

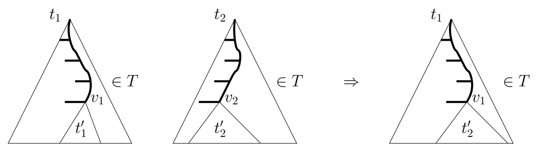


Figure 8: Ancestor-sibling-guarded subtree exchange.

6. EXPRESSIVENESS OF 1PPT

We provide the same kind of analysis for 1PPT SDTDs as we did for single-type ones. In particular, we provide a mechanism to show that certain schemas are not expressible by 1PPT SDTDs and instantiate the framework of $\mathcal{P}$-schemas to get an equivalent syntactical counterpart.

6.1 Subtree Exchange Property

Just like for DTDs and SDTDs satisfying EDC, there also exists a characterization of the sets of XML documents definable by SDTDs admitting 1PPT ([13, Theorem 12]). Indeed, a regular set T of documents is definable by a 1PPT SDTD if and only if it is closed under ancestor-sibling-guarded subtree exchange. The latter property is defined as follows (cf. Figure 8): if two documents t_1 and t_2 are in T, and there are two nodes v_1 in t_1 and v_2 in t_2 with the same ancestor-sibling string, then the trees obtained by exchanging the subtrees rooted at v_1 and v_2 are also in the set T.

The language we considered in Example 1 is not definable by an SDTD admitting 1PPT. Note that the counterexample can be constructed in exactly the same manner.

Just as for EDC, given an arbitrary SDTD it is decidable in exponential time whether there exists an equivalent SDTD admitting 1PPT. This upper bound is also tight [13, Theorem 14].

6.2 $\mathcal{R}^*$-Schemas

To raise the expressiveness of the framework proposed in Section 4.2 to the level of 1PPT SDTDs, we need an adequate pattern language. To this end, let $\mathcal{R}^*$ be the class of regular expressions over symbols $a[r]$ where r is a regular expression over element names and a is an element name. For instance, $(a[a+b^*]+b)^*a[b^*]$ belongs to $\mathcal{R}^*$. We simply write a for $a[\Sigma^*]$. We note that the given expression uses the three symbols $a[a+b^*]$, $b[\Sigma^*]$ and $a[b^*]$. The intuition is that $a[r]$ matches node v when v is labeled with a and the string formed by the labels of the left siblings of v match r. We explain how general regular expressions can be used as unary patterns. Let $a_1w_2a_2\cdots w_na_n$ be the ancestor-sibling string of node v in tree t, such that $a_1\cdots a_n$ are the labels on the path from the root to v. So, the root of t is labeled a_1 and v is labeled a_n, and w_i is the sequence of labels of left siblings of a_i (from left to right). Node v is selected by pattern φ iff there exists a string $a_1a_2[r_2]\cdots a_n[r_n] \in L(\varphi)$ such that for every $i = 2, \ldots, n$, $w_i \in L(r_i)$. In other words, for each symbol $a_i[r_i]$, r_i constrains the left siblings of the element a_i.

EXAMPLE 9. *Using $\mathcal{R}^*$ as pattern language, we can define the SDTD in Section 5.3 in the following way:*

$$\begin{array}{ll}
\texttt{store} & \rightarrow \texttt{dvd}^* \ \texttt{discounts}\ \texttt{dvd}^* \\
\texttt{discounts} & \rightarrow \varepsilon \\
(\texttt{regular-dvd},\ \texttt{store}\ \texttt{dvd}[\texttt{dvd}^*]) & \Rightarrow \texttt{title}\ \texttt{price} \\
(\texttt{discount-dvd},\ \texttt{store}\ \texttt{dvd}[\texttt{dvd}^* & \texttt{discount}\ \texttt{dvd}^*]) \\
 & \Rightarrow \texttt{title}\ \texttt{price}\ \texttt{discount}
\end{array}$$

In [13, Theorem 12] it is shown that the class of $\mathcal{R}^*$-schemas corresponds precisely to the class of schemas represented by SDTDs satisfying 1PPT. In other words, the instantiation of the general framework with regular expressions over ancestor-sibling-strings is an alternative syntax for *all* SDTDs admitting 1PPT. In [13], we considered ancestor-sibling-guarded DTDs which are equivalent to $\mathcal{R}^*$-schemas.

6.3 $\mathcal{R}^*$-Schemas in Practice

As in Section 4.3, we can adopt two approaches: extend DTDs or extend XML Schema. To capture 1PPT SDTDs it suffices to add $\mathcal{R}^*$-patterns. A more practical way is to add full XPath, but semantically restrict its evaluation to the preceding of each node (cf. Figure 6). For instance, the expression `//*[.//b]//c` selects only those c-elements having a b-element in their preceding as illustrated in Figure 9.

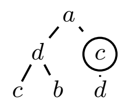


Figure 9: Only the circled c-element in the document has a b-element in its preceding.

7. DISCUSSION

The exchange property for single-type SDTDs and the framework of $\mathcal{R}$-schemas provides insight into what can and cannot be expressed on a structural level by the current definition of XML Schema. In future work, we plan to implement a concrete pattern based schema language as discussed in Section 4.3. Although already an abundance of schema languages for XML has been proposed, we believe that a light-weight front end for XML Schema is beneficiary for less experienced XML users (as, e.g., in the bioinformatics community [5]).

Although we think the restriction to unambiguous typing increases transparency and efficiency of validation, the recommendations in the present paper do not justify the former. For instance, Relax NG as well as the formal model for XML Schema of Siméon and Wadler [22] allow ambiguous typing to relieve users from opaque restrictions. However, if unambiguous typing is required, it should not be enforced by ad-hoc restrictions, but by the most liberal ones. We believe the restriction to 1-pass preorder typable schemas is adequate. Moreover, it can be reached by allowing restrained competition regular expressions. Further, we provided an equivalent syntactic framework in terms of $\mathcal{R}^*$-schemas.

Acknowledgements. We thank Dan Suciu for stimulating discussions and Jan Van den Bussche for comments on a previous draft of this paper.

8. REFERENCES

[1] G.J. Bex, F. Neven and J. Van den Bussche. DTDs versus XML Schema: A Practical Study. In *WebDB 2004*, pages 79–84, 2004.

[2] A. Brüggemann-Klein, M. Murata, and D. Wood. Regular tree and regular hedge languages over unranked alphabets. Hongkong Univ. of Sc. and Tech., 2001. Tech. Rep. HKUST-TCSC-2001-0.

[3] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[4] J. Clark and M. Murata. RELAX NG Specification, 2001. http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[5] E. Cerami. *XML for Bioinformatics.* Springer-Verlag, 2004.

[6] R. Cover. The Cover Pages. http://xml.coverpages.org/

[7] B. DuCharme. Filling in the DTD gaps with Schematron. O'Reilly xml.com, May 2002. http://www.xml.com/pub/a/2002/05/15/schematron.html

[8] D. Fiorello, N. Gessa, P. Marinelli and F. Vitali. DTD++ 2.0: adding support for co-constraints. In *Extreme Markup Languages 2004*, Montreal, Canada.

[9] R. Jelliffe. The current state of the art of schema languages for XML. Presentation at XML Asia Pacific, Sidney, Australia, 2001.

[10] N. Klarlund, A. Møller, and M. I. Schwartzbach. The DSD schema language. *Autom. Softw. Eng.*, 9(3):285–319, 2002.

[11] D. Lee and W.W. Chu. Comparative analysis of six XML schema languages. *ACM SIGMOD Record*, 29(3), 2000.

[12] M. Mani. Keeping chess alive - Do we need 1-unambiguous content models? In *Extreme Markup Languages*, Montreal, Canada, 2001.

[13] W. Martens, F. Neven, and T. Schwentick. Which XML schemas admit 1-pass preorder typing? In *ICDT 2005.* To Appear.

[14] M. Murata, D. Lee, and M. Mani. Taxonomy of XML schema languages using formal language theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.

[15] F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[16] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. In *PODS 2000*, pages 35–46, 2000.

[17] C. Sacerdoti Coen, P. Marinelli, and F. Vitali. Schemapath, a minimal extension to XML Schema for conditional constraints. In *WWW 2004*, pages 164–174, 2004.

[18] A. Sahuguet. Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask. In *WebDB 2000*, pages 69–74, 2000.

[19] Schematron. http://xml.ascc.net/schematron/

[20] T. Schwentick. XPath query containment. *SIGMOD Record*, 33(1):101–109, 2004.

[21] L. Segoufin and V. Vianu. Validating streaming XML documents. In *PODS 2002*, pages 53–64, 2002.

[22] J. Siméon and P. Wadler. The essence of XML. In *POPL 2003*, pages 1–13, 2003.

[23] E. van der Vlist. *XML Schema.* O'Reilly, 2002.

[24] F. Vitali, N. Amorosi and N. Gessa. Datatype- and namespace-aware DTDs: a minimal extension. In *Extreme Markup Languages 2003*, Montreal, Canada.

[25] World Wide Web Consortium. Extensible Markup Language (XML). http://www.w3.org/XML

[26] World Wide Web Consortium. XML Schema Part 1: Structures. http://www.w3.org/TR/xmlschema-1/

[27] World Wide Web Consortium. XML Schema Part 2: Datatypes. http://www.w3.org/TR/xmlschema-2/

[28] World Wide Web Consortium. Datatypes for DTDs (DT4DTD) 1.0 http://www.w3.org/TR/dt4dtd

[29] XML-dev, Monthly archives. http://lists.xml.org/archives/xml-dev/200102/msg00008.html

[30] XML Schema Quality Checker. http://www.alphaworks.ibm.com/tech/xmlsqc

[31] XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/xpath-datamodel/